US 20080198935A1

(19) United States

(12) Patent Application Publication

Srinivasan et al.

(10) Pub. No.: US 2008/0198935 A1

(43) Pub. Date: Aug. 21, 2008

(54) **COMPUTATIONAL COMPLEXITY AND PRECISION CONTROL IN TRANSFORM-BASED DIGITAL MEDIA CODEC**

(75) Inventors: **Sridhar Srinivasan**, Redmond, WA (US); **Chengjie Tu**, Sammamish, WA (US); **Shankar Regunathan**, Bellevue, WA (US)

Correspondence Address:
KLARQUIST SPARKMAN LLP
121 S.W. SALMON STREET, SUITE 1600
PORTLAND, OR 97204

(73) Assignee: **Microsoft Corporation**, Redmond, WA (US)

(21) Appl. No.: **11/772,076**

(22) Filed: **Jun. 29, 2007**

Related U.S. Application Data

(60) Provisional application No. 60/891,031, filed on Feb. 21, 2007.

Publication Classification

(51) **Int. Cl.**
H04N 7/24 (2006.01)

(52) **U.S. Cl.** **375/240.23**; 375/E07.001

(57) **ABSTRACT**

A digital media encoder/decoder includes signaling of various modes relating to computation complexity and precision at decoding. The encoder may send a syntax element indicating arithmetic precision (e.g., using 16 or 32-bit operations) of the transform operations performed at decoding. The encoder also may signal whether to apply scaling at the decoder output, which permits a wider dynamic range of intermediate data at decoding, but adds to computational complexity due to the scaling operation.

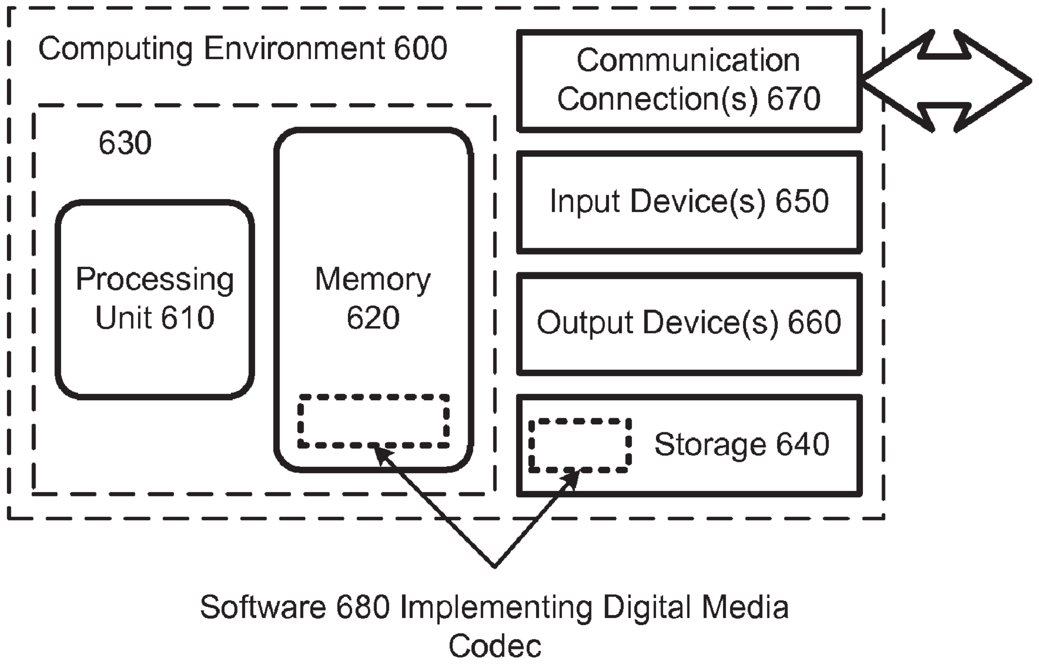

Computing Environment 600
630
Processing Unit 610
Memory 620
Communication Connection(s) 670
Input Device(s) 650
Output Device(s) 660
Storage 640
Software 680 Implementing Digital Media Codec

Figure 1
Prior Art

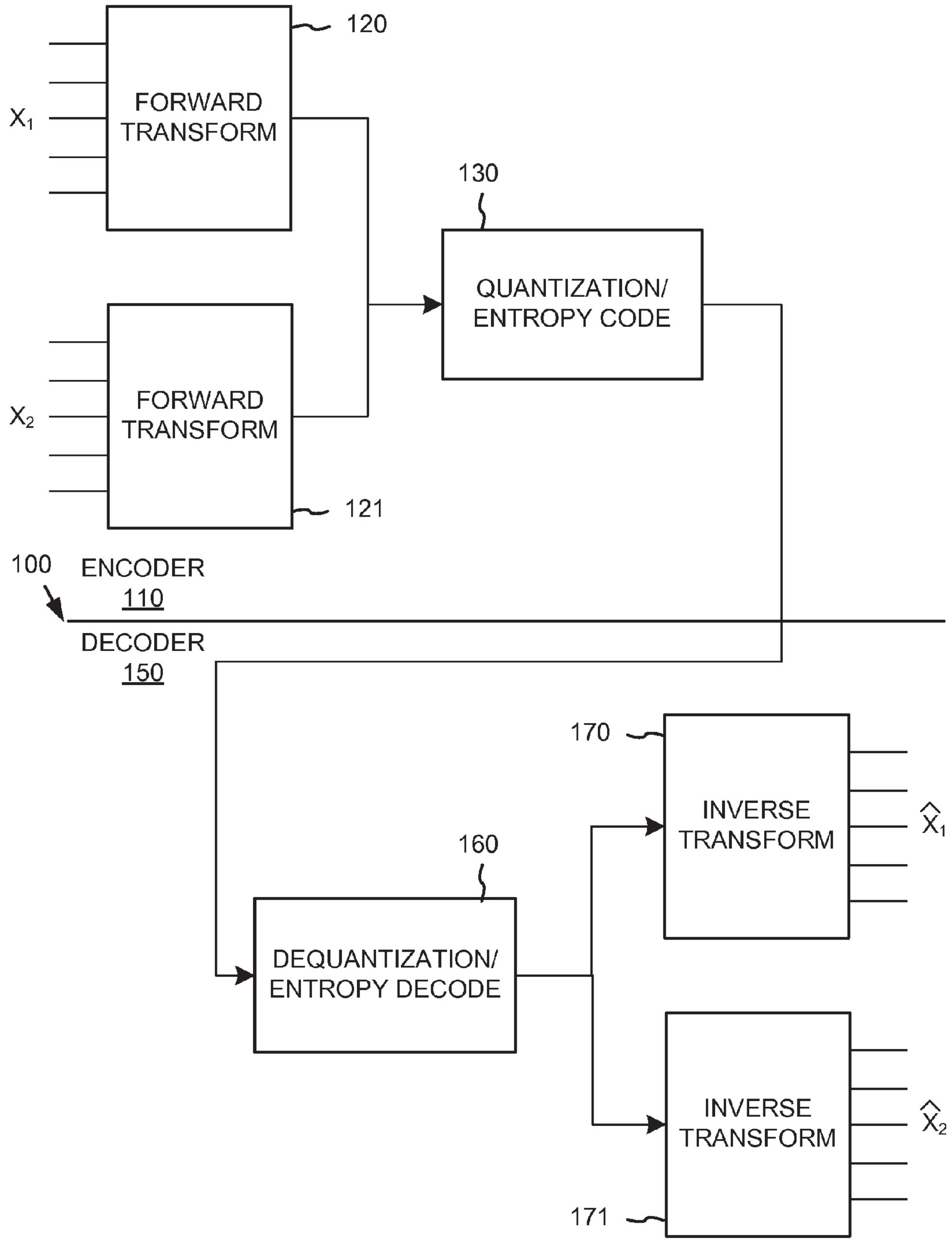


Figure 2

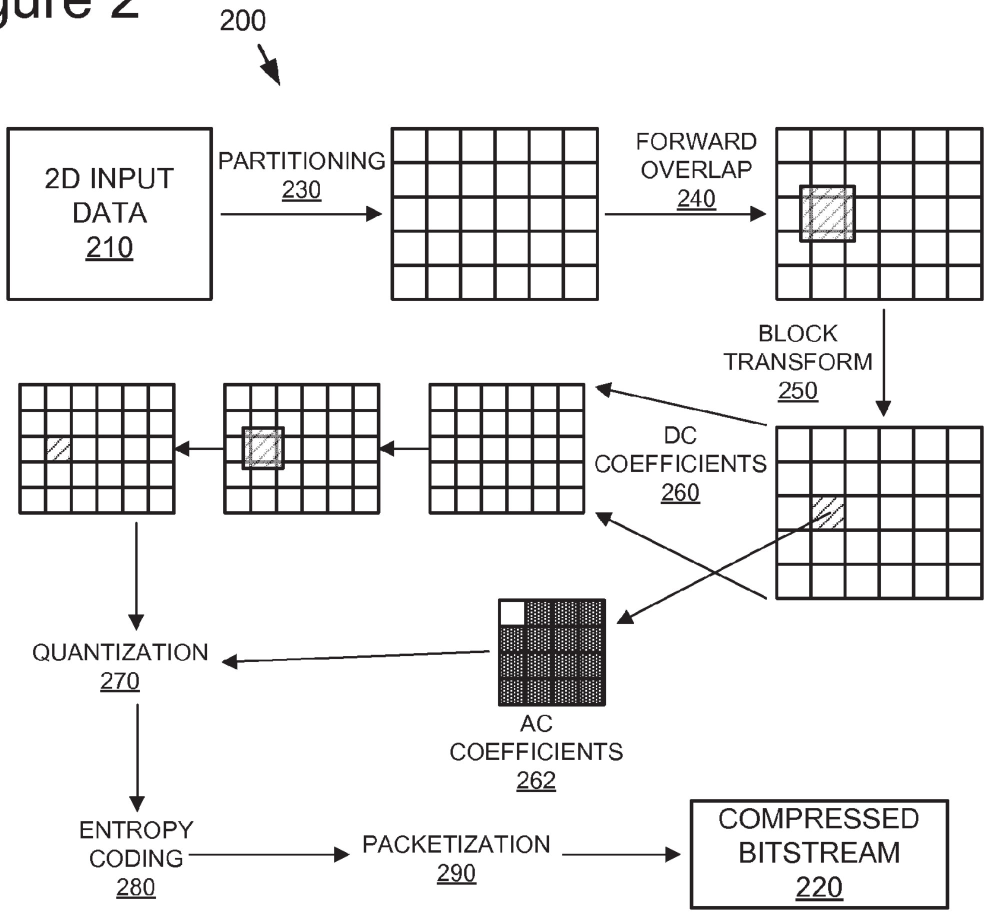

200
2D INPUT DATA 210
PARTITIONING 230
FORWARD OVERLAP 240
BLOCK TRANSFORM 250
DC COEFFICIENTS 260
AC COEFFICIENTS 262
QUANTIZATION 270
ENTROPY CODING 280
PACKETIZATION 290
COMPRESSED BITSTREAM 220

Figure 3

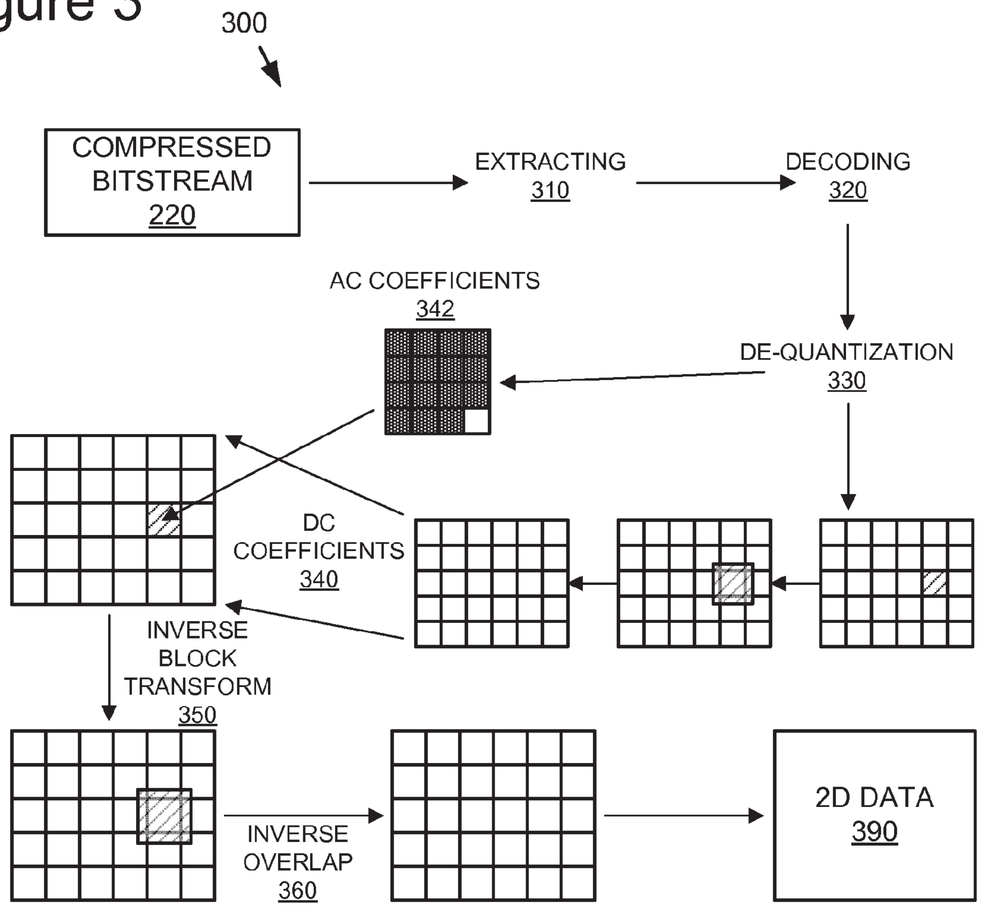

300
COMPRESSED BITSTREAM 220
EXTRACTING 310
DECODING 320
AC COEFFICIENTS 342
DE-QUANTIZATION 330
DC COEFFICIENTS 340
INVERSE BLOCK TRANSFORM 350
INVERSE OVERLAP 360
2D DATA 390

Figure 4

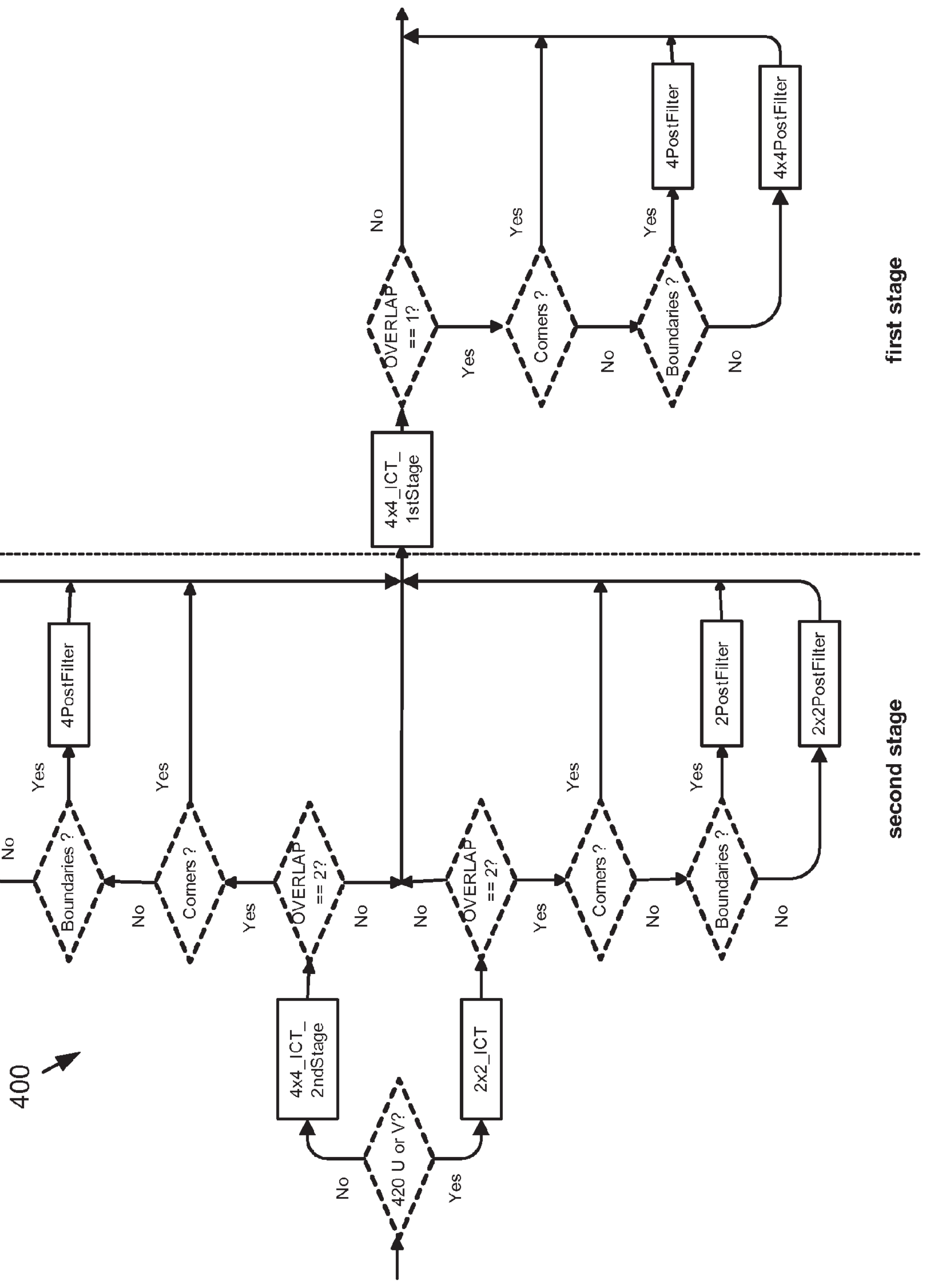

400
420 U or V?
No
Yes
4x4_ICT_ 2ndStage
2x2_ICT
OVERLAP == 2?
Yes
No
Corners ?
Boundaries ?
4PostFilter
4x4PostFilter
2PostFilter
2x2PostFilter
second stage
4x4_ICT_ 1stStage
OVERLAP == 1?
first stage

Figure 5

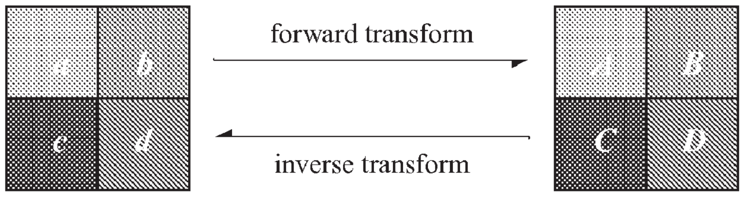
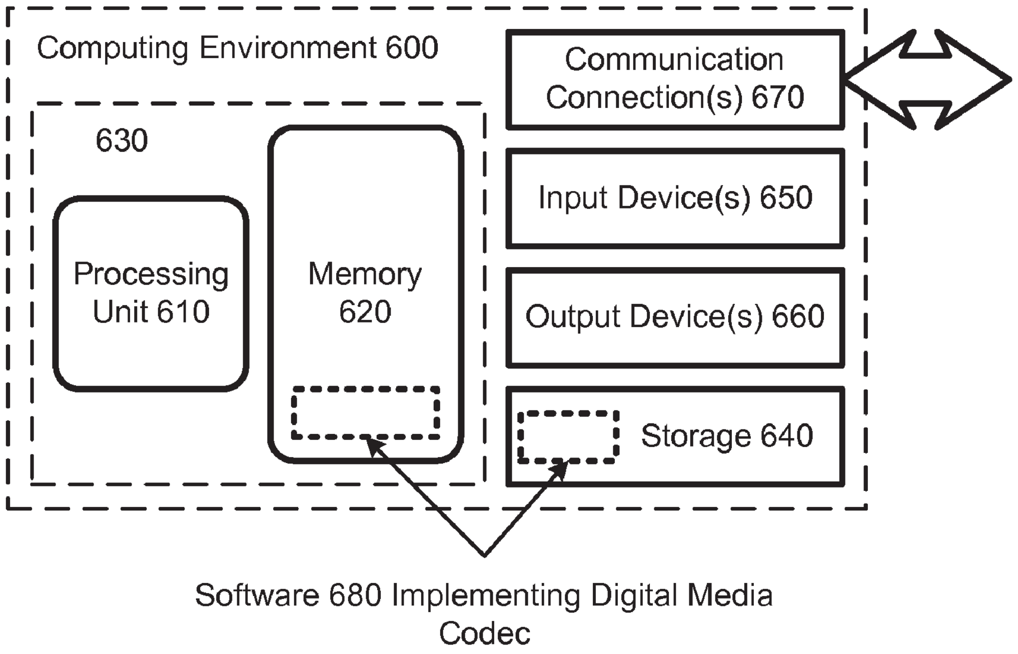


Figure 6


Computing Environment 600
630
Processing Unit 610
Memory 620
Communication Connection(s) 670
Input Device(s) 650
Output Device(s) 660
Storage 640
Software 680 Implementing Digital Media Codec

COMPUTATIONAL COMPLEXITY AND PRECISION CONTROL IN TRANSFORM-BASED DIGITAL MEDIA CODEC

CROSS REFERENCE TO RELATED APPLICATION

[0001] This application claims the benefit of U.S. Provisional 60/891,301, filed Feb. 21, 2007 and is incorporated herein by reference.

BACKGROUND

[0002] Block Transform-Based Coding

[0003] Transform coding is a compression technique used in many digital media (e.g., audio, image and video) compression systems. Uncompressed digital image and video is typically represented or captured as samples of picture elements or colors at locations in an image or video frame arranged in a two-dimensional (2D) grid. This is referred to as a spatial-domain representation of the image or video. For example, a typical format for images consists of a stream of 24-bit color picture element samples arranged as a grid. Each sample is a number representing color components at a pixel location in the grid within a color space, such as RGB, or YIQ, among others. Various image and video systems may use various different color, spatial and time resolutions of sampling. Similarly, digital audio is typically represented as time-sampled audio signal stream. For example, a typical audio format consists of a stream of 16-bit amplitude samples of an audio signal taken at regular time intervals.

[0004] Uncompressed digital audio, image and video signals can consume considerable storage and transmission capacity. Transform coding reduces the size of digital audio, images and video by transforming the spatial-domain representation of the signal into a frequency-domain (or other like transform domain) representation, and then reducing resolution of certain generally less perceptible frequency components of the transform-domain representation. This generally produces much less perceptible degradation of the digital signal compared to reducing color or spatial resolution of images or video in the spatial domain, or of audio in the time domain.

[0005] More specifically, a typical block transform-based encoder/decoder system **100** (also called a "codec") shown in FIG. **1** divides the uncompressed digital image's pixels into fixed-size two dimensional blocks ($X_1, \ldots X_n$), each block possibly overlapping with other blocks. A linear transform **120-121** that does spatial-frequency analysis is applied to each block, which converts the spaced samples within the block to a set of frequency (or transform) coefficients generally representing the strength of the digital signal in corresponding frequency bands over the block interval. For compression, the transform coefficients may be selectively quantized **130** (i.e., reduced in resolution, such as by dropping least significant bits of the coefficient values or otherwise mapping values in a higher resolution number set to a lower resolution), and also entropy or variable-length coded **130** into a compressed data stream. At decoding, the transform coefficients will inversely transform **170-171** to nearly reconstruct the original color/spatial sampled image/video signal (reconstructed blocks $\hat{X}_1, \ldots \hat{X}_n$).

[0006] The block transform **120-121** can be defined as a mathematical operation on a vector x of size N. Most often, the operation is a linear multiplication, producing the transform domain output y=Mx, M being the transform matrix. When the input data is arbitrarily long, it is segmented into N sized vectors and a block transform is applied to each segment. For the purpose of data compression, reversible block transforms are chosen. In other words, the matrix M is invertible. In multiple dimensions (e.g., for image and video), block transforms are typically implemented as separable operations. The matrix multiplication is applied separably along each dimension of the data (i.e., both rows and columns).

[0007] For compression, the transform coefficients (components of vector y) may be selectively quantized (i.e., reduced in resolution, such as by dropping least significant bits of the coefficient values or otherwise mapping values in a higher resolution number set to a lower resolution), and also entropy or variable-length coded into a compressed data stream.

[0008] At decoding in the decoder **150**, the inverse of these operations (dequantization/entropy decoding **160** and inverse block transform **170-171**) are applied on the decoder **150** side, as show in FIG. **1**. While reconstructing the data, the inverse matrix M^{-1} (inverse transform **170-171**) is applied as a multiplier to the transform domain data. When applied to the transform domain data, the inverse transform nearly reconstructs the original time-domain or spatial-domain digital media.

[0009] In many block transform-based coding applications, the transform is desirably reversible to support both lossy and lossless compression depending on the quantization factor. With no quantization (generally represented as a quantization factor of 1) for example, a codec utilizing a reversible transform can exactly reproduce the input data at decoding. However, the requirement of reversibility in these applications constrains the choice of transforms upon which the codec can be designed.

[0010] Many image and video compression systems, such as MPEG and Windows Media, among others, utilize transforms based on the Discrete Cosine Transform (DCT). The DCT is known to have favorable energy compaction properties that result in near-optimal data compression. In these compression systems, the inverse DCT (IDCT) is employed in the reconstruction loops in both the encoder and the decoder of the compression system for reconstructing individual image blocks.

[0011] Quantization

[0012] Quantization is the primary mechanism for most image and video codecs to control compressed image quality and compression ratio. According to one possible definition, quantization is a term used for an approximating non-reversible mapping function commonly used for lossy compression, in which there is a specified set of possible output values, and each member of the set of possible output values has an associated set of input values that result in the selection of that particular output value. A variety of quantization techniques have been developed, including scalar or vector, uniform or non-uniform, with or without dead zone, and adaptive or non-adaptive quantization.

[0013] The quantization operation is essentially a biased division by a quantization parameter QP which is performed at the encoder. The inverse quantization or multiplication operation is a multiplication by QP performed at the decoder. These processes together introduce a loss in the original transform coefficient data, which shows up as compression errors or artifacts in the decoded image.

SUMMARY

[0014] The following Detailed Description presents tools and techniques to control computational complexity and precision of decoding with a digital media codec. In one aspect of the techniques, the encoder signals one of scaled or unscaled precision modes to use at the decoder. In the scaled precision mode, the input image is pre-multiplied (e.g., by 8) at the encoder. The output at the decoder also is scaled by rounded division. In the unscaled precision mode, no such scaling operations are applied. In the unsclaed precision mode, the encoder and decoder can deal with a smaller dynamic range for transform coefficient, and thus has lower computational complexity.

[0015] In another aspect of the techniques, the codec may also signal the precision required for performing transform operations to the decoder. In one implementation, an element of the bitstream syntax signals whether to employ a lower precision arithmetic operations for the transform at the decoder.

[0016] This Summary is provided to introduce a selection of concepts in a simplified form that is further described below in the Detailed Description. This summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used as an aid in determining the scope of the claimed subject matter. Additional features and advantages of the invention will be made apparent from the following detailed description of embodiments that proceeds with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] FIG. **1** is a block diagram of a conventional block transform-based codec in the prior art.

[0018] FIG. **2** is a flow diagram of a representative encoder incorporating the block pattern coding.

[0019] FIG. **3** is a flow diagram of a representative decoder incorporating the block pattern coding.

[0020] FIG. **4** is a diagram of the inverse lapped transform including a core transform and post-filter (overlap) operation in one implementation of the representative encoder/decoder of FIGS. **2** and **3**.

[0021] FIG. **5** is a diagram of identifying the input data points for the transform operations.

[0022] FIG. **6** is a block diagram of a suitable computing environment for implementing the media encoder/decoder of FIGS. **2** and **3**.

DETAILED DESCRIPTION

[0023] The following description relates to techniques to control precision and computational complexity of a transform-based digital media codec. The following description describes an example implementation of the technique in the context of a digital media compression system or codec. The digital media system codes digital media data in a compressed form for transmission or storage, and decodes the data for playback or other processing. For purposes of illustration, this exemplary compression system incorporating the computational complexity and precision control is an image or video compression system. Alternatively, the technique also can be incorporated into compression systems or codecs for other digital media data. The computational complexity and precision control technique does not require that the digital media compression system encodes the compressed digital media data in a particular coding format.

1. Encoder/Decoder

[0024] FIGS. **2** and **3** are a generalized diagram of the processes employed in a representative 2-dimensional (2D) data encoder **200** and decoder **300**. The diagrams present a generalized or simplified illustration of a compression system incorporating the 2D data encoder and decoder that implement compression using the computational complexity and precision control techniques. In alternative compression systems using the control techniques, additional or fewer processes than those illustrated in this representative encoder and decoder can be used for the 2D data compression. For example, some encoders/decoders may also include color conversion, color formats, scalable coding, lossless coding, macroblock modes, etc. The compression system (encoder and decoder) can provide lossless and/or lossy compression of the 2D data, depending on the quantization which may be based on a quantization parameter varying from lossless to lossy.

[0025] The 2D data encoder **200** produces a compressed bitstream **220** that is a more compact representation (for typical input) of 2D data **210** presented as input to the encoder. For example, the 2D data input can be an image, a frame of a video sequence, or other data having two dimensions. The 2D data encoder divides a frame of the input data into blocks (illustrated generally in FIG. **2** as partitioning **230**), which in the illustrated implementation are non-overlapping 4×4 pixel blocks that form a regular pattern across the plane of the frame. These blocks are grouped in clusters, called macroblocks, which are 16×16 pixels in size in this representative encoder. In turn, the macroblocks are grouped into regular structures called tiles. The tiles also form a regular pattern over the image, such that tiles in a horizontal row are of uniform height and aligned, and tiles in a vertical column are of uniform width and aligned. In the representative encoder, the tiles can be any arbitrary size that is a multiple of 16 in the horizontal and/or vertical direction. Alternative encoder implementations can divide the image into block, macroblock, tiles, or other units of other size and structure.

[0026] A “forward overlap” operator **240** is applied to each edge between blocks, after which each 4×4 block is transformed using a block transform **250**. This block transform **250** can be the reversible, scale-free 2D transform described by Srinivasan, U.S. patent application Ser. No. 11/015,707, entitled, “Reversible Transform For Lossy And Lossless 2-D Data Compression,” filed Dec. 17, 2004. The overlap operator **240** can be the reversible overlap operator described by Tu et al., U.S. patent application Ser. No. 11/015,148, entitled, “Reversible Overlap Operator for Efficient Lossless Data Compression,” filed Dec. 17, 2004; and by Tu et al., U.S. patent application Ser. No. 11/035,991, entitled, “Reversible 2-Dimensional Pre-/Post-Filtering For Lapped Biorthogonal Transform,” filed Jan. 14, 2005. Alternatively, the discrete cosine transform or other block transforms and overlap operators can be used. Subsequent to the transform, the DC coefficient **260** of each 4×4 transform block is subject to a similar processing chain (tiling, forward overlap, followed by 4×4 block transform). The resulting DC transform coefficients and the AC transform coefficients are quantized **270**, entropy coded **280** and packetized **290**.

[0027] The decoder performs the reverse process. On the decoder side, the transform coefficient bits are extracted **310** from their respective packets, from which the coefficients are themselves decoded **320** and dequantized **330**. The DC coefficients **340** are regenerated by applying an inverse transform, and the plane of DC coefficients is "inverse overlapped" using a suitable smoothing operator applied across the DC block edges. Subsequently, the entire data is regenerated by applying the 4×4 inverse transform **350** to the DC coefficients, and the AC coefficients **342** decoded from the bitstream. Finally, the block edges in the resulting image planes are inverse overlap filtered **360**. This produces a reconstructed 2D data output.

[0028] In an exemplary implementation, the encoder **200** (FIG. **2**) compresses an input image into the compressed bitstream **220** (e.g., a file), and the decoder **300** (FIG. **3**) reconstructs the original input or an approximation thereof, based on whether lossless or lossy coding is employed. The process of encoding involves the application of a forward lapped transform (LT) discussed below, which is implemented with reversible 2-dimensional pre-/post-filtering also described more fully below. The decoding process involves the application of the inverse lapped transform (ILT) using the reversible 2-dimensional pre-/post-filtering.

[0029] The illustrated LT and the ILT are inverses of each other, in an exact sense, and therefore can be collectively referred to as a reversible lapped transform. As a reversible transform, the LT/ILT pair can be used for lossless image compression.

[0030] The input data **210** compressed by the illustrated encoder **200**/decoder **300** can be images of various color formats (e.g., RGB/YUV4:4:4, YUV4:2:2 or YUV4:2:0 color image formats). Typically, the input image always has a luminance (Y) component. If it is a RGB/YUV4:4:4, YUV4:2:2 or YUV4:2:0 image, the image also has chrominance components, such as a U component and a V component. The separate color planes or components of the image can have different spatial resolutions. In case of an input image in the YUV 4:2:0 color format for example, the U and V components have half of the width and height of the Y component.

[0031] As discussed above, the encoder **200** tiles the input image or picture into macroblocks. In an exemplary implementation, the encoder **200** tiles the input image into 16×16 pixel areas (called "macroblocks") in the Y channel (which may be 16×16, 16×8 or 8×8 areas in the U and V channels depending on the color format). Each macroblock color plane is tiled into 4×4 pixel regions or blocks. Therefore, a macroblock is composed for the various color formats in the following manner for this exemplary encoder implementation:

[0032] 1. For a grayscale image, each macroblock contains 16 4×4 luminance (Y) blocks.

[0033] 2. For a YUV4:2:0 format color image, each macroblock contains 16 4×4 Y blocks, and 4 each 4×4 chrominance (U and V) blocks.

[0034] 3. For a YUV4:2:2 format color image, each macroblock contains 16 4×4 Y blocks, and 8 each 4×4 chrominance (U and V) blocks.

[0035] 4. For a RGB or YUV4:4:4 color image, each macroblock contains 16 blocks each of Y, U and V channels.

[0036] Accordingly, after transform, a macroblock in this representative encoder **200**/decoder **300** has three frequency sub bands: a DC sub band (DC macroblock), a low pass sub band (low pass macroblock), and a high pass sub band (high pass macroblock). In the representative system, the low pass and/or high pass sub bands are optional in the bitstream—these sub bands may be entirely dropped.

[0037] Further, the compressed data can be packed into the bitstream in one of two orderings: spatial order and frequency order. For the spatial order, different sub bands of the same macroblock within a tile are ordered together, and the resulting bitstream of each tile is written into one packet. For the frequency order, the same sub band from different macroblocks within a tile are grouped together, and thus the bitstream of a tile is written into three packets: a DC tile packet, a low pass tile packet, and a high pass tile packet. In addition, there may be other data layers.

[0038] Thus, for the representative system, an image is organized in the following "dimensions":

[0039] Spatial dimension: Frame→Tile→Macroblock;

[0040] Frequency dimension: DC|Low pass|High pass; and

[0041] Channel dimension: Luminance|Chrominance_0|Chrominance_1 . . . (e.g. as Y|U|V).

The arrows above denote a hierarchy, whereas the vertical bars denote a partitioning.

[0042] Although the representative system organizes the compressed digital media data in spatial, frequency and channel dimensions, the flexible quantization approach described here can be applied in alternative encoder/decoder systems that organize their data along fewer, additional or other dimensions. For example, the flexible quantization approach can be applied to coding using a larger number of frequency bands, other format of color channels (e.g., YIQ, RGB, etc.), additional image channels (e.g., for stereo vision or other multiple camera arrays).

2. Inverse Core and Lapped Transform

[0043] Overview

[0044] In one implementation of the encoder **200**/decoder **300**, the inverse transform on the decoder side takes the form of a two-level lapped transform. The steps are as follows:

[0045] An Inverse Core Transform (ICT) is applied to each 4×4 block corresponding to reconstructed DC and lowpass coefficients arranged in a planar array known as the DC plane.

[0046] A post filter operation is optionally applied to 4×4 areas evenly straddling blocks in the DC plane. Further, a post filter is applied to boundary 2×4 and 4×2 areas, and the four 2×2 corner areas are left untouched.

[0047] The resulting array contains DC coefficients of the 4×4 blocks corresponding to the first-level transform. The DC coefficients are (figuratively) copied into a larger array, and the reconstructed highpass coefficients populated into the remaining positions.

[0048] An ICT is applied to each 4×4 block.

[0049] A post filter operation is optionally applied to 4×4 areas evenly straddling blocks in the DC plane. Further, a post filter is applied to boundary 2×4 and 4×2 areas, and the four 2×2 corner areas are left untouched.

[0050] This process is shown in FIG. **4**.

[0051] The application of post filters is governed by an OVERLAP_INFO syntax element in the compressed bitstream **220**. OVERLAP_INFO may take on three values:

[0052] If OVERLAP_INFO=0, no post filtering is performed.

[0053] If OVERLAP_INFO=1, only the outer post filtering is performed.

[0054] If OVERLAP_INFO=2, both inner and outer post filtering is performed.

[0055] Inverse Core Transform

[0056] The Core Transform (CT) is inspired by the conventionally known 4×4 Discrete Cosine Transform (DCT), yet it is fundamentally different. The first key difference is that the DCT is linear whereas the CT is nonlinear. The second key difference is that due to the fact that it is defined on real numbers, the DCT is not a lossless operation in the integer to integer space. The CT is defined on integers and is lossless in this space. The third key difference is that the 2D DCT is a separable operation. The CT is non-separable by design.

[0057] The entire inverse transform process can be written as the cascade of three elementary 2×2 transform operations, which are:

[0058] 2×2 Hadamard transform: T_h

[0059] Inverse ID rotate: InvT_odd

[0060] Inverse 2D rotate: InvT_odd_odd

[0061] These transforms are implemented as non-separable operations and are described first, followed by the description of the entire ICT.

[0062] 2D 2×2 Hadamard Transform T_h

[0063] The encoder/decoder implements the 2D 2×2 Hadamard transform T_h as shown in the following pseudo-code table. R is a rounding factor which may take on the value 0 or 1 only. T_h is involutory (i.e. two applications of T_h on a data vector [a b c d] succeed in recovering the original values of [a b c d], provided R is unchanged between the applications). The inverse T_h is T_h itself.

```
T_h (a,b,c,d, R) {
    a += d;
    b -= c;
    int t1 = ((a - b + R) >> 1);
    int t2 = c;
    c = t1 - d;
    d = t1 - t2;
    a -= d;
    b += c;
}
```

[0064] Inverse 1D Rotate InvT_odd

[0065] The lossless inverse of T_odd is defined by the pseudocode in the following table.

```
InvT_odd (a,b,c,d) {
    b += d;
    a -= c;
    d -= (b >> 1);
    c += ((a + 1) >> 1);
    a -= ((3*b + 4) >> 3);
    b += ((3*a + 4) >> 3);
    c -= ((3*d + 4) >> 3);
    d += ((3*c + 4) >> 3);
    c -= ((b + 1) >> 1);
    d = ((a + 1) >> 1) - d;
    b += c;
    a -= d;
}
```

[0066] Inverse 2D Rotate InvT_odd_odd

[0067] Inverse 2D rotate InvT_odd_odd is defined by the pseudocode in the following table.

```
InvT_odd_odd (a,b,c,d) {
    int t1, t2;
    d += a;
    c -= b;
    a -= (t1 = d >> 1);
    b += (t2 = c >> 1);
    a -= ((b * 3 + 3) >> 3);
    b += ((a * 3 + 3) >> 2);
    a -= ((b * 3 + 4) >> 3);
    b -= t2;
    a += t1;
    c += b;
    d -= a;
    b = -b
    c = -c
}
```

[0068] ICT Operations

[0069] The correspondence between 2×2 data and the previously listed pseudocode is shown in FIG. **5**. Color coding using four gray levels to indicate the four data points is introduced here, to facilitate the transform description in the next section.

[0070] The 2D 4×4 point ICT is built using T_h, inverse T_odd and inverse T_odd_odd. Note the inverse T_h is T_h itself. The ICT is composed of two stages, which are shown in the following pseudo-code. Each stage consists of four 2×2 transforms which may be done in any arbitrary sequence, or concurrently, within the stage.

[0071] If the input data block is

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix},$$

4×4_IPCT_1stStage( ) and 4×4_IPCT_2ndStage( ) are defined as:

```
4x4_IPCT (a...p) {
    T_h(a, c, i, k);
    InvT_odd(b, d, j, l);
    InvT_odd(e, m, g, o);
    InvT_odd_odd(f, h, n, p);
    T_h(a, d, m, p);
    T_h(k, j, g, f);
    T_h(c, b, o, n);
    T_h(i, l, e, h);
}
```

[0072] The function 2×2_ICT is the same as T_h.

[0073] Post Filtering Overview

[0074] Four operators determine the post filters used in the inverse lapped transform. These are:

[0075] 4×4 post filter

[0076] 4 point post filter

[0077] 2×2 post filter

[0078] 2 point post filter

[0079] The post-filter uses T_h, InvT_odd_odd, invScale and invRotate. invRotate and invScale are defined in the below tables, respectively.

```
invRotate (a,b) {
    a −= ((b * 3 + 8) >> 4);
    b += ((a * 3 + 4) >> 3);
    a −= ((b * 3 + 8) >> 4);
}
```

```
invScale (a,b) {
    b += a;
    a −= ((b + 1) >> 1);
    b += ((a * 3 + 0) >> 3);
    a += ((b * 3 + 8) >> 4);
    b += ((a * 3 + 4) >> 3);
    a += ((b + 1) >> 1);
    b −= a;
}
```

[0080] 4×4 Post Filter

[0081] Primarily, the 4×4 post-filter is applied to all block junctions (areas straddling 4 blocks evenly) in all color planes when OVERLAP_INFO is 1 or 2. Also, the 4×4 filter is applied to all block junctions in the DC plane for all planes when OVERLAP_INFO is 2, and for only the luma plane when OVERLAP_INFO is 2 and color format is either YUV 4:2:0 or YUV 4:2:2.

[0082] If the input data block is

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix},$$

the 4×4 post-filter, 4×4PostFilter (a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p), is defined in the following table:

```
4x4PostFilter
        (a,b,...,p) {
    T_h (a, d, m, p, 0);
    T_h (b, c, n, o, 0);
    T_h (e, h, i, l, 0);
    T_h (f, g, j, k, 0);
    invRotate (n, m);
    invRotate (j, i);
    invRotate (h, d);
    invRotate (g, c);
    InvT_odd_odd (k, l, o, p);
    invScale (a, p);
    invScale (b, l);
    invScale (e, o);
    invScale (f, k);
    T_h (a, m, d, p, 0);
    T_h (b, n, c, o, 0);
    T_h (e, i, h, l, 0);
    T_h (f, j, g, k, 0);
}
```

[0083] 4-Point Post-Filter

[0084] Linear 4-point filters are applied to edge straddling 2×4 and 4×2 areas on the boundary of the image. If the input data is [a b c d], the 4-point post-filter, 4PostFilter(a, b, c, d), is defined in the following table.

```
4PostFilter (a,b,c,d) {
    a += d;
    b += c;
    d −= ((a + 1) >> 1);
    c −= ((b + 1) >> 1);
    invRotate(c, d);
    d += ((a + 1) >> 1);
    c += ((b + 1) >> 1);
    a −= d − ((d * 3 + 16) >> 5);
    b −= c − ((c * 3 + 16) >> 5);
    d += ((a * 3 + 8) >> 4);
    c += ((b * 3 + 8) >> 4);
    a += ((d * 3 + 16) >> 5);
    b += ((c * 3 + 16) >> 5);
}
```

[0085] 2×2 Post-Filter

[0086] The 2×2 post-filter is applied to areas straddling blocks in the DC plane for the chroma channels of YUV 4:2:0 and YUV 4:2:2 data. If the input data is

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix},$$

the 2×2 post-filter 2×2PostFilter (a, b, c, d), is defined in the following table:

```
2x2PostFilter
        (a,b,c,d) {
    a += d;
    b += c;
    d −= ((a + 1) >> 1);
    c −= ((b + 1) >> 1);
    b += ((a + 2) >> 2);
    a += ((b + 1) >> 1);
    b += ((a + 2) >> 2);
    d += ((a + 1) >> 1);
    c += ((b + 1) >> 1);
    a −= d;
    b −= c;
}
```

[0087] 2-Point Post-Filter

[0088] The 2-point post filter is applied to boundary 2×1 and 1×2 samples that straddle blocks. The 2-point post-filter, 2PostFilter (a, b) is defined in the following table:

```
2PostFilter (a,b) {
    b += ((a + 4) >> 3);
    a += ((b + 2) >> 2);
    b += ((a + 4) >> 3);
}
```

[0089] Signaling of the precision required for performing transform operations of the above described lapped transform can be performed in the header of a compressed image structure. In the example implementation, LONG_WORD_FLAG and NO_SCALED_FLAGS are syntax elements transmitted in the compressed bitstream (e.g., in the image header) to signal precision and computational complexity to be applied by the decoder.

3. Precision and Word Length

[0090] The example encoder/decoder performs integer operations. Further, the example encoder/decoder supports lossless encoding and decoding. Therefore, the primary machine precision required by the example encoder/decoder is integer.

[0091] However, integer operations defined in the example encoder/decoder lead to rounding errors for lossy coding. These errors are small by design, however they cause drops in the rate distortion curve. For the sake of improved coding performance by the reduction of rounding errors, the example encoder/decoder defines a secondary machine precision. In this mode, the input is pre multiplied by 8 (i.e. left shifted by 3 bits) and the final output is divided by 8 with rounding (i.e. right shifted by 3 bits). These operations are carried out at the front end of the encoder and the rear end of the decoder, and are largely invisible to the rest of the processes. Further, the quantization levels are scaled accordingly such that a stream created with the primary machine precision and decoded using the secondary machine precision (and vice versa) produces an acceptable image.

[0092] The secondary machine precision cannot be used when lossless compression is desired. The machine precision used in creating a compressed file is explicitly marked in the header.

[0093] The secondary machine precision is equivalent to using scaled arithmetic in the codec, and hence this mode is referred to as Scaled. The primary machine precision is referred to as Unscaled.

[0094] The example encoder/decoder is designed to provide good encoding and decoding speed. A design goal of the example encoder/decoder is that the data values on both encoder and decoder do not exceed 16 signed bits for an 8 bit input. (However, intermediate operation within a transform stage may exceed this figure.) This holds true for both modes of machine precision.

[0095] Conversely, when the secondary machine precision is chosen, the range expansion of the intermediate values is by 8 bits. Since the primary machine precision avoids a pre-multiplication by 8, its range expansion is 8−3=5 bits.

[0096] The first example encoder/decoder uses two different word lengths for intermediate values. These word lengths are 16 and 32 bits.

[0097] Second Example Bitstream Syntax and Semantics

[0098] The second example bitstream syntax and semantics is hierarchical and is comprised of the following layers: Image, Tile, Macroblock and Block.

	Num bits	Descriptor
Image (IMAGE)		
IMAGE ( ){		
IMAGE__HEADER	Variable	struct
bAlphaPlane = FALSE		
IMAGE__PLANE__HEADER	Variable	struct
if(ALPHACHANNEL__FLAG) {		
bAlphaPlane = TRUE		
IMAGE__PLANE__HEADER	Variable	Struct
}		
INDEX__TABLE	Variable	struct
TILE	Variable	struct
}		
Image Header (IMAGE__HEADER)		
IMAGE__HEADER ( ){		
GDISIGNATURE	64	uimsbf
RESERVED1	4	uimsbf
RESERVED2	4	uimsbf
TILING__FLAG	1	bool
	1	uimsbf
FREQUENCYMODE__BITSTREAM__FLAG		
IMAGE__ORIENTATION	3	uimsbf
	1	uimsbf
INDEXTABLE__PRESENT__FLAG		
OVERLAP__INFO	2	uimsbf
SHORT__HEADER__FLAG	1	bool
LONG__WORD__FLAG	1	bool
WINDOWING__FLAG	1	bool
TRIM__FLEXBITS__FLAG	1	bool
RESERVED3	3	uimsbf
ALPHACHANNEL__FLAG	1	bool
SOURCE__CLR__FMT	4	uimsbf
SOURCE__BITDEPTH	4	uimsbf
If(SHORT__HEADER__FLAG) {		
WIDTH__MINUS1	16	uimsbf
HEIGHT__MINUS1	16	uimsbf
}		
else {		
WIDTH__MINUS1	32	uimsbf
HEIGHT__MINUS1	32	uimsbf

-continued

	Num bits	Descriptor
}		
if(TILING_FLAG) {		
NUM_VERT_TILES_MINUS1	12	uimsbf
NUM_HORIZ_TILES_MINUS1	12	uimsbf
}		
for (n = 0; n < NUM_VERT_TILES_MINUS1; N++) {		
If (SHORT_HEADER_FLAG)		
WIDTH_IN_MB_OF_TILE_MINUS1[n]	8	uimsbf
else		
WIDTH_IN_MB_OF_TILE_MINUS1[n]	16	uimsbf
}		
for (n = 0; n < NUM_HORIZ_TILES_MINUS1; n++) {		
If (SHORT_HEADER_FLAG)		
HEIGHT_IN_MB_OF_TILE_MINUS1[n]	8	uimsbf
else		
HEIGHT_IN_MB_OF_TILE_MINUS1[n]	16	uimsbf
}		
if (WINDOWING_FLAG) {		
NUM_TOP_EXTRAPIXELS	6	uimsbf
NUM_LEFT_EXTRAPIXELS	6	uimsbf
NUM_BOTTOM_EXTRAPIXELS	6	uimsbf
NUM_RIGHT_EXTRAPIXELS	6	uimsbf
}		
}		
IMAGE_PLANE_HEADER ( ) {		
CLR_FMT	3	uimsbf
NO_SCALED_FLAG	1	bool
BANDS_PRESENT	4	uimsbf
if (CLR_FMT == YUV444) {		
CHROMA_CENTERING	4	uimsbf
COLOR_INTERPRETATION	4	uimsbf
}		
Else if (CLR_FMT == NCHANNEL) {		
NUM_CHANNELS_MINUS1	4	uimsbf
COLOR_INTERPRETATION	4	uimsbf
}		
if (SOURCE_CLR_FMT == BAYER) {		
BAYER_PATTERN	2	uimsbf
CHROMA_CENTERING_BAYER	2	uimsbf
COLOR_INTERPRETATION	4	uimsbf
}		
if (SOURCE_BITDEPTH ∈ {BD16,BD16S,BD32,BD32S}) {		
SHIFT_BITS	8	uimsbf
}		
if (SOURCE_BITDEPTH == BD32F) {		
LEN_MANTISSA	8	uimsbf
EXP_BIAS	8	uimsbf
}		
DC_FRAME_UNIFORM	1	bool
if (DC_FRAME_UNIFORM) {		
DC_QP( )	variable	struct
}		
if (BANDS_PRESENT != SB_DC_ONLY) {		
USE_DC_QP	1	bool
if (USE_DC_QP == FALSE) {		
LP_FRAME_UNIFORM	1	bool
if (LP_FRAME_UNIFORM) {		
NUM_LP_QPS = 1		
LP_QP( )	variable	struct
}		
}		
if (BANDS_PRESENT != SB_NO_HIGHPASS) {		
USE_LP_QP	1	bool
if (USE_LP_QP == FALSE) {		

-continued

	Num bits	Descriptor
HP__FRAME__UNIFORM	1	bool
if (HP__FRAME__UNIFORM) {		
NUM__HP__QPS = 1		
HP__QP( )	variable	struct
}		
}		
}		
}		
FLUSH__BYTE	variable	
}		

[0099] Some selected bitstream elements from the second example bitstream syntax and semantics are defined below.

[0100] Long Word Flag (LONG_WORD_FLAG) (1 bit)

[0101] LONG_WORD_FLAG is a 1-bit syntax element and specifies whether 16-bit integers may be used for transform computations. In this second example bitstream syntax, if LONG_WORD_FLAG==0 (FALSE), 16-bit integer numbers and arrays may be used for the outer stage of transform computations (intermediate operations within the transform (such as (3*a+1)>>1) are performed with higher accuracy). If LONG_WORD_FLAG==TRUE, 32-bit integer numbers and arrays shall be used for transform computations.

[0102] Note: 32-bit arithmetic may be used to decode an image regardless of the value of LONG_WORD_FLAG. This syntax element can be used by the decoder to choose the most efficient word length for implementation.

[0103] No Scaled Arithmetic Flag (NO_SCALED_FLAG)(1 bit)

[0104] NO_SCALED_FLAG is a 1-bit syntax element that specifies whether the transform uses scaling. If NO_SCALED_FLAG==1, scaling shall not be performed. If NO_SCALED_FLAG==0, scaling shall be used. In this case, scaling shall be performed by appropriately rounding down the output of the final stage (color conversion) by 3 bits.

[0105] Note: NO_SCALED_FLAG shall be set to TRUE if lossless coding is desired, even if lossless coding is used for only a subregion of an image. Lossy coding may use either mode.

[0106] Note: The rate-distortion performance for lossy coding is superior when scaling is used (i.e. NO_SCALED_FLAG==FALSE), especially at low QPs.

4. Signaling and Use of Long Word Flag

[0107] One example image format for the representative encoder/decoder supports a wide range of pixel formats, including high dynamic range and wide gamut formats. Supported data types include signed integer, unsigned integer, fixed-point float and floating-point float. Supported bit depths include eight, 16, 24 and 32 bits per color channel. The example image format allows for lossless compression of images that use up to 24 bits per color channel, and lossy compression of images that use up to 32 bits per color channel.

[0108] At the same time, the example image format has been designed to provide high quality images and compression efficiency and allow low-complexity encoding and decoding implementations.

[0109] To support low-complexity implementation, the transform in an example image format has been designed to minimize expansion in dynamic range. The two-stage transform increases dynamic range by only five bits. Therefore, if the image bit depth is eight bits per color channel, 16 bit arithmetic may be sufficient for performing all transform operations at the decoder. For other bit depths, higher precision arithmetic may be needed for transform operations.

[0110] The computational complexity of decoding a particular bitstream can be reduced if the precision required for performing transform operations is known at the decoder. This information can be signaled to a decoder using a syntax element (e.g., a 1-bit flag in an image header). Described signaling techniques and syntax elements can reduce computational complexity in decoding bitstreams.

[0111] In one example implementation, the 1-bit syntax element LONG_WORD_FLAG is used. For example, if LONG_WORD_FLAG==FALSE, 16-bit integer numbers and arrays may be used for the outer stage of transform computations, and if LONG_WORD_FLAG==TRUE, 32 bit integer numbers and arrays shall be used for transform computations.

[0112] In the one implementation of the representative encoder/decoder, the in-place transform operations may be performed on 16-bit wide words, but intermediate operations within the transform (such as computation of the product **3***a for a "lifting" step given by b+=(3*a+1)>>1)) are performed with higher accuracy (e.g., 18 bits or higher precision). However, in this example, the intermediate transform values a and b themselves may be stored in 16-bit integers.

[0113] 32-bit arithmetic may be used to decode an image regardless of the value of the LONG_WORD_FLAG element. The LONG_WORD_FLAG element can be used by the encoder/decoder to choose the most efficient word length for implementation. For example, an encoder may choose to set the LONG_WORD_FLAG element to FALSE if it can verify that the 16-bit and 32-bit precision transform steps produce the same output value.

5. Signaling and Use of NO_SCALED_FLAG

[0114] One example image format for the representative encoder/decoder supports a wide range of pixel formats, including high dynamic range and wide gamut formats. At the same time, the design of the representative encoder/decoder optimizes image quality and compression efficiency and enables low-complexity encoding and decoding implementations.

[0115] As discussed above, the representative encoder/decoder uses two stage hierarchical block-based transform, where all the transform steps are integer operations. The small rounding errors present in these integer operations result in loss of compression efficiency during lossy compression. To combat this problem, one implementation of the representative encoder/decoder defines two different precision modes for decoder operations: the scaled mode and the unscaled mode.

[0116] In the scaled precision mode, the input image is pre multiplied by 8 (i.e. left shifted by 3 bits) at the encoder, and the final output at the decoder is divided by 8 with rounding (i.e. right shifted by 3 bits). Operation in the scaled precision mode minimizes the rounding errors, and results in improved rate-distortion performance.

[0117] In the unscaled precision mode, there is no such scaling. An encoder or decoder operating in unscaled precision mode has to deal with a smaller dynamic range for transform coefficients, and thus has lower computational complexity. However, there is a small penalty in compression efficiency for operating in this mode. Lossless coding (with no quantization, i.e. setting the Quantization Parameter or QP to 1) can only use the unscaled precision mode for guaranteed reversibility.

[0118] The precision mode used by the encoder in creating a compressed file is explicitly signaled in the image header of the compressed bitstream **220** (FIG. **2**) using the NO_SCALED_FLAG. It is recommended that the decoder **300** use the same precision mode for its operations.

[0119] NO_SCALED_FLAG is a 1-bit syntax element in the image header that specifies the precision mode as follows:

[0120] If NO_SCALED_FLAG==TRUE, unscaled mode should be used for decoder operation.

[0121] If NO_SCALED_FLAG==FALSE, scaling should be used. In this case, scaled mode should be used for operation by appropriately rounding down the output of the final stage (color conversion) by 3 bits.

[0122] The rate-distortion performance for lossy coding is superior when unscaled mode is used (i.e. NO_SCALED_FLAG==FALSE), especially at low QPs. However, the computation complexity is lower when unscaled mode is used due to two reasons:

[0123] Smaller dynamic range expansion in the unscaled mode means that shorter words may be used for transform computations, especially in conjunction with the "LONG_WORD_FLAG." In VLSI implementations, the reduced dynamic range expansion means that the gate logic implementing the more significant bits may be powered down.

[0124] The scaled mode requires an add and right bit shift by 3 bits (implementing a rounded divide by 8) on the decoder side. On the encoder side, it requires a left bit shift by 3 bits. This is slightly more computationally demanding than the unscaled mode overall.

[0125] Further, the unscaled mode allows for the compression of more significant bits than does the scaled mode. For instance, the unscaled mode permits the lossless compression (and decompression) of up to 27 significant bits per sample, using 32 bit arithmetic. In contrast, the scaled mode allows the same for only 24 bits. This is because of the three additional bits of dynamic range introduced by the scaling process.

[0126] The data values on decoder do not exceed 16 signed bits for an 8 bit input for both modes of precision. (However, intermediate operation within a transform stage may exceed this figure.)

[0127] Note: NO_SCALED_FLAG is set to TRUE by the encoder, if lossless coding (QP=1) is desired, even if lossless coding is required for only a subregion of an image.

[0128] The encoder may use either mode for lossy compression. It is recommended that the decoder use the precision mode signaled by NO_SCALED_MODE for its operations. However, the quantization levels are scaled such that a stream created with the scaled precision mode and decoded using the unscaled precision mode (and vice versa) produces an acceptable image in most cases.

6. Scaling Arithmetic for Increased Accuracy

[0129] In one implementation of the representative encoder/decoder, the transforms (including color conversion) are integer transforms and implemented through a series of lifting steps. In those lifting steps, truncation errors hurt transform performance. For lossy compression cases, to minimize the damage of truncation errors and thus maximize transform performance, input data to a transform needs to be left shifted several bits. However, another highly desired feature is if the input image is 8 bits, then the output of every transform should be within 16 bits. So the number of left shift bits cannot be large. The representative decoder implements a technique of scaling arithmetic to achieve both goals. The scaling arithmetic technique maximizes transform performance by minimizing damage of truncation errors, and still limits output of each transform step to be within 16 bits if input image is 8 bits. This makes simple 16-bit implementation possible.

[0130] The transforms utilized in the representative encoder/decoder are integer transforms and implemented by lifting steps. Most lifting steps involve a right shift, which introduces truncation errors. A transform generally involves many lifting steps, and accumulated truncation errors hurt transform performance visibly.

[0131] One way to reduce the damage of truncation errors is to left shift the input data before the transform in the encoder, and right shift the same number of bits after transform (combined with quantization) at the decoder. As described above, the representative encoder/decoder has a two-stage transform structure: optional first stage overlap+first stage CT+optional second stage overlap+second stage CT. Experiments show that left shift by 3 bits is necessary to minimize the truncation errors. So in lossy cases, before color conversion, input data may be left shifted by 3 bits, i.e. multiplied or scaled by a factor of 8 (e.g., for the scaled mode described above).

[0132] However, color conversion and transforms expand data. If input data is shifted by 3 bits, the output of second stage 4×4 DCT has a 17-bit dynamic range if input data is 8 bits (output of every other transform is still within 16 bits). This is hugely undesired since it prevents 16-bit implementation, a highly desired feature. To get around this, before the second stage 4×4 CT, the input data is right shifted by 1 bit and so the output is also within 16 bits. Since the second stage 4×4 CT applies to only $1/16$ of the data (the DC transform coefficients of the first stage DCT), and the data is already scaled up by first stage transform, so the damage of truncation errors here is minimal.

[0133] So in lossy cases for 8-bit images, on encoder side, input is left shifted by 3 bits before color conversion, and right shifted 1 bit before second stage 4×4 CT. On the decoder side, the input is left shifted by 1 bit before first stage 4×4 IDCT, and right shifted by 3 bits after color conversion.

7. Computing Environment

[0134] The above-described processing techniques for computational complexity and precision signaling in a digital media codec can be realized on any of a variety of digital media encoding and/or decoding systems, including among other examples, computers (of various form factors, including server, desktop, laptop, handheld, etc.); digital media recorders and players; image and video capture devices (such as cameras, scanners, etc.); communications equipment (such as telephones, mobile phones, conferencing equipment, etc.); display, printing or other presentation devices; and etc. The computational complexity and precision signaling techniques in a digital media codec can be implemented in hardware circuitry, in firmware controlling digital media processing hardware, as well as in communication software executing within a computer or other computing environment, such as shown in FIG. **6**.

[0135] FIG. **6** illustrates a generalized example of a suitable computing environment (**600**) in which described embodiments may be implemented. The computing environment (**600**) is not intended to suggest any limitation as to scope of use or functionality of the invention, as the present invention may be implemented in diverse general-purpose or special-purpose computing environments.

[0136] With reference to FIG. **6**, the computing environment (**600**) includes at least one processing unit (**610**) and memory (**620**). In FIG. **6**, this most basic configuration (**630**) is included within a dashed line. The processing unit (**610**) executes computer-executable instructions and may be a real or a virtual processor. In a multi-processing system, multiple processing units execute computer-executable instructions to increase processing power. The memory (**620**) may be volatile memory (e.g., registers, cache, RAM), non-volatile memory (e.g., ROM, EEPROM, flash memory, etc.), or some combination of the two. The memory (**620**) stores software (**680**) implementing the described digital media encoding/decoding with computational complexity and precision signaling techniques.

[0137] A computing environment may have additional features. For example, the computing environment (**600**) includes storage (**640**), one or more input devices (**650**), one or more output devices (**660**), and one or more communication connections (**670**). An interconnection mechanism (not shown) such as a bus, controller, or network interconnects the components of the computing environment (**600**). Typically, operating system software (not shown) provides an operating environment for other software executing in the computing environment (**600**), and coordinates activities of the components of the computing environment (**600**).

[0138] The storage (**640**) may be removable or non-removable, and includes magnetic disks, magnetic tapes or cassettes, CD-ROMs, CD-RWs, DVDs, or any other medium which can be used to store information and which can be accessed within the computing environment (**600**). The storage (**640**) stores instructions for the software (**680**) implementing the described digital media encoding/decoding with computational complexity and precision signaling techniques.

[0139] The input device(s) (**650**) may be a touch input device such as a keyboard, mouse, pen, or trackball, a voice input device, a scanning device, or another device that provides input to the computing environment (**600**). For audio, the input device(s) (**650**) may be a sound card or similar device that accepts audio input in analog or digital form from a microphone or microphone array, or a CD-ROM reader that provides audio samples to the computing environment. The output device(s) (**660**) may be a display, printer, speaker, CD-writer, or another device that provides output from the computing environment (**600**).

[0140] The communication connection(s) (**670**) enable communication over a communication medium to another computing entity. The communication medium conveys information such as computer-executable instructions, compressed audio or video information, or other data in a modulated data signal. A modulated data signal is a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media include wired or wireless techniques implemented with an electrical, optical, RF, infrared, acoustic, or other carrier.

[0141] The described digital media encoding/decoding with flexible quantization techniques herein can be described in the general context of computer-readable media. Computer-readable media are any available media that can be accessed within a computing environment. By way of example, and not limitation, with the computing environment (**600**), computer-readable media include memory (**620**), storage (**640**), communication media, and combinations of any of the above.

[0142] The described digital media encoding/decoding with computational complexity and precision signaling techniques herein can be described in the general context of computer-executable instructions, such as those included in program modules, being executed in a computing environment on a target real or virtual processor. Generally, program modules include routines, programs, libraries, objects, classes, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The functionality of the program modules may be combined or split between program modules as desired in various embodiments. Computer-executable instructions for program modules may be executed within a local or distributed computing environment.

[0143] For the sake of presentation, the detailed description uses terms like "determine," "generate," "adjust," and "apply" to describe computer operations in a computing environment. These terms are high-level abstractions for operations performed by a computer, and should not be confused with acts performed by a human being. The actual computer operations corresponding to these terms vary depending on implementation.

[0144] In view of the many possible embodiments to which the principles of our invention may be applied, we claim as our invention all such embodiments as may come within the scope and spirit of the following claims and equivalents thereto.

We claim:

1. A digital media decoding method comprising:

receiving a compressed digital media bitstream at a digital media decoder;

parsing a syntax element from the bitstream signaling a degree of arithmetic precision to use for transform computations during processing of the digital media data; and

outputting a reconstructed image.

2. The digital media decoding method of claim **1** wherein the syntax element signals to use one of a high arithmetic precision or a low arithmetic precision.

3. The digital media decoding method of claim **2** wherein the high arithmetic precision is 32-bit number processing, and the low arithmetic precision is 16-bit number processing.

4. The digital media decoding method of claim **2** further comprising:

decoding blocks of transform coefficients from the compressed digital media bitstream;

in the case that the syntax element signals use of the high arithmetic precision, applying an inverse transform to the transform coefficients using high arithmetic precision processing; and

in the case that the syntax element signals use of the low arithmetic precision, applying an inverse transform to the transform coefficients using low arithmetic precision processing.

5. The digital media decoding method of claim **4** wherein the high arithmetic precision is 32-bit number processing, and the low arithmetic precision is 16-bit number processing.

6. The digital media decoding method of claim **2** further comprising:

decoding blocks of transform coefficients from the compressed digital media bitstream;

applying an inverse transform to the transform coefficients using high arithmetic precision processing regardless of the degree of arithmetic precision signaled via the syntax element.

7. A digital media encoding method comprising:

receiving digital media data at a digital media encoder;

making a decision whether to use lower precision arithmetic for transform computations during processing of the digital media data;

representing the decision whether to use the lower precision arithmetic for transform computations with a syntax element in an encoded bitstream, wherein the syntax element is operable to communicate the decision to a digital media decoder; and

outputting the encoded bitstream.

8. The digital media encoding method of claim **7** wherein said making a decision comprises:

verifying whether the lower precision arithmetic for transform computations produces a same decoder output as using a higher precision arithmetic for transform computations; and

deciding whether to use the lower precision arithmetic based upon said verifying.

9. The digital media encoding method of claim **7** wherein said lower precision arithmetic is a 16-bit arithmetic precision.

10. The digital media encoding method of claim **7** further comprising:

making a decision whether to apply a scaling of the input digital media data prior to transform coding; and

representing the decision whether to apply the scaling with a syntax element in the encoded bitstream.

11. The digital media encoding method of claim **10** wherein said making a decision whether to apply a scaling comprises deciding not to apply scaling of the input digital media data when losslessly encoding the digital media data.

12. A digital media decoding method comprising:

receiving a compressed digital media bitstream at a digital media decoder;

parsing a syntax element from the bitstream signaling choice of precision modes for transform computations during processing of the digital media data;

in the case that a first precision mode using scaling is signaled, scaling output of the decoder;

in the case that a second precision mode without scaling is signaled, omitting to apply scaling of the output; and

outputting a reconstructed image.

13. The digital media decoding method of claim **12** wherein said scaling output of the decoder comprises rounded division of the output by a number.

14. The digital media decoding method of claim **12** wherein said rounded division of the output is a rounded division by the number 8.

15. The digital media decoding method of claim **12** further comprising:

parsing a second syntax element from the bitstream signaling whether to use a lower arithmetic precision for transform computations during processing of the digital media data;

decoding blocks of transform coefficients from the compressed digital media bitstream; and

in the case that the second precision mode without scaling and the use of lower arithmetic precision are signaled, performing inverse transform processing of the transform coefficients using the lower arithmetic precision.

16. The digital media decoding method of claim **15** wherein said lower arithmetic precision is a 16-bit arithmetic precision.

17. The digital media decoding method of claim **12** wherein the digital media data is encoding using a two stage transform structure having a first stage transform followed by a second stage transform on DC coefficients of the first stage transform, the digital media decoding method further comprising:

decoding digital media data from the compressed digital media bitstream;

applying an inverse second stage transform to the digital media data;

applying an inverse first stage transform to the digital media data;

performing color conversion of the digital media data; and

wherein said scaling output of the decoder in the case that the first precision mode using scaling is signaled, comprises:

left shifting the digital media data by a single bit before input to the inverse first stage transform; and

right shifting the digital media data by three bits after the color conversion.

18. The digital media decoding method of claim **12** wherein said compressed digital media bitstream is encoded according to a syntax scheme defining separate primary image plane and alpha image plane for an image, the syntax element signaling choice of precision mode being signaled per image plane, whereby the precision mode of the primary image plane and the alpha image plane are independently signaled, and the decoding method comprises performing said actions of parsing the syntax element signaling choice of precision mode for each image plane, and in the case that the first precision mode using scaling is signaled for a respective image plane, scaling output of the decoder for the respective image plane.

* * * * *