(12) UK Patent (19) GB (11) 2619627 (13) B

(45) Date of B Publication 28.02.2024

(54) Title of the Invention: Low complexity enhancement video coding

(51) INT CL: ***H04N 19/132*** (2014.01) ***H04N 19/157*** (2014.01) ***H04N 19/187*** (2014.01) ***H04N 19/33*** (2014.01) ***H04N 19/59*** (2014.01) ***H04N 19/70*** (2014.01)

(21) Application No: **2312599.0**

(22) Date of Filing: **18.03.2020**

Date Lodged: **17.08.2023**

(30) Priority Data:

(31) **1903844**	(32) **20.03.2019**	(33) **GB**
(31) **1904014**	(32) **23.03.2019**	(33) **GB**
(31) **1904492**	(32) **29.03.2019**	(33) **GB**
(31) **1905325**	(32) **15.04.2019**	(33) **GB**
(31) **1909701**	(32) **05.07.2019**	(33) **GB**
(31) **1909724**	(32) **06.07.2019**	(33) **GB**
(31) **1909997**	(32) **11.07.2019**	(33) **GB**
(31) **1910674**	(32) **25.07.2019**	(33) **GB**
(31) **1911467**	(32) **09.08.2019**	(33) **GB**
(31) **1911546**	(32) **13.08.2019**	(33) **GB**
(31) **1914215**	(32) **02.10.2019**	(33) **GB**
(31) **1914414**	(32) **06.10.2019**	(33) **GB**
(31) **1914634**	(32) **10.10.2019**	(33) **GB**
(31) **1915553**	(32) **25.10.2019**	(33) **GB**
(31) **1916090**	(32) **05.11.2019**	(33) **GB**
(31) **1918099**	(32) **10.12.2019**	(33) **GB**
(31) **2000430**	(32) **12.01.2020**	(33) **GB**
(31) **2000483**	(32) **13.01.2020**	(33) **GB**
(31) **2000600**	(32) **15.01.2020**	(33) **GB**
(31) **2000668**	(32) **16.01.2020**	(33) **GB**
(31) **2001408**	(32) **31.01.2020**	(33) **GB**
(31) **62984261**	(32) **02.03.2020**	(33) **US**

(62) Divided from Application No
2303563.7 under section 15(9) of the Patents Act 1977

(43) Date of A Publication **13.12.2023**

(56) Documents Cited:
US 20170127085 A1 **US 20070160126 A1**
US 20050259729 A1

(58) Field of Search:
As for published application 2619627 A viz:
INT CL **H04N**
Other: **SEARCH-PATENT, SEARCH-NPL**
updated as appropriate

Additional Fields
Other: **None**

(72) Inventor(s):
Guido Meardi
Simone Ferrara
Lorenzo Ciccarelli
Ivan Damnjanovic
Richard Clucas
Sam Littlewood

(73) Proprietor(s):
V-Nova International Limited
(Incorporated in the United Kingdom)
20 Eastbourne Terrace, Paddington, London, W2 6LG, United Kingdom

(74) Agent and/or Address for Service:
Gill Jennings & Every LLP
The Broadgate Tower, 20 Primrose Street, LONDON, EC2A 2ES, United Kingdom

GB 2619627 B

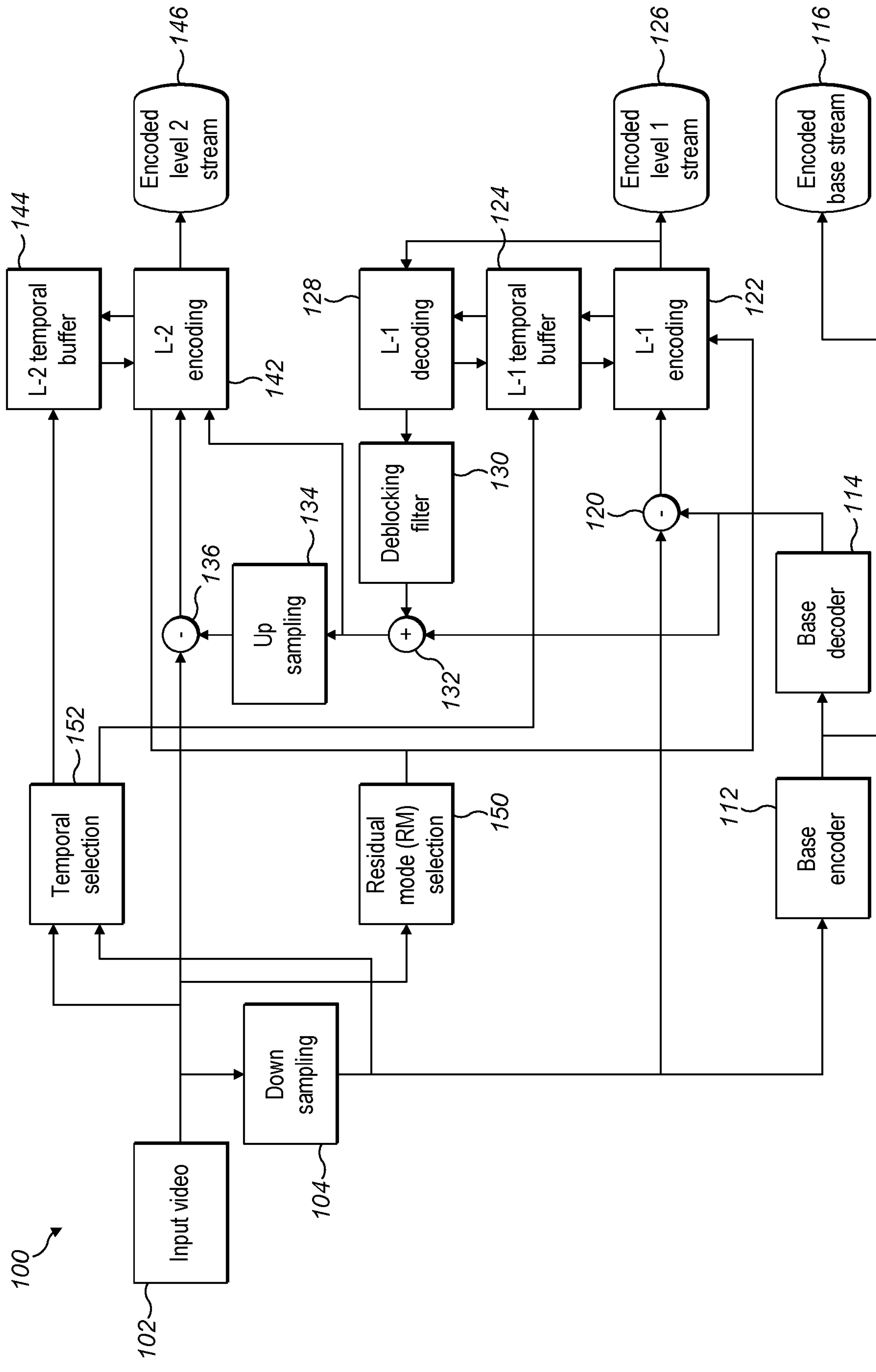

100
Input video
102
Down sampling
104
Temporal selection
152
Residual mode (RM) selection
150
Base encoder
112
Base decoder
114
Encoded base stream
116
-
120
L-1 encoding
122
L-1 temporal buffer
124
Encoded level 1 stream
126
L-1 decoding
128
Deblocking filter
130
+
132
Up sampling
134
-
136
L-2 encoding
142
L-2 temporal buffer
144
Encoded level 2 stream
146

FIG. 1

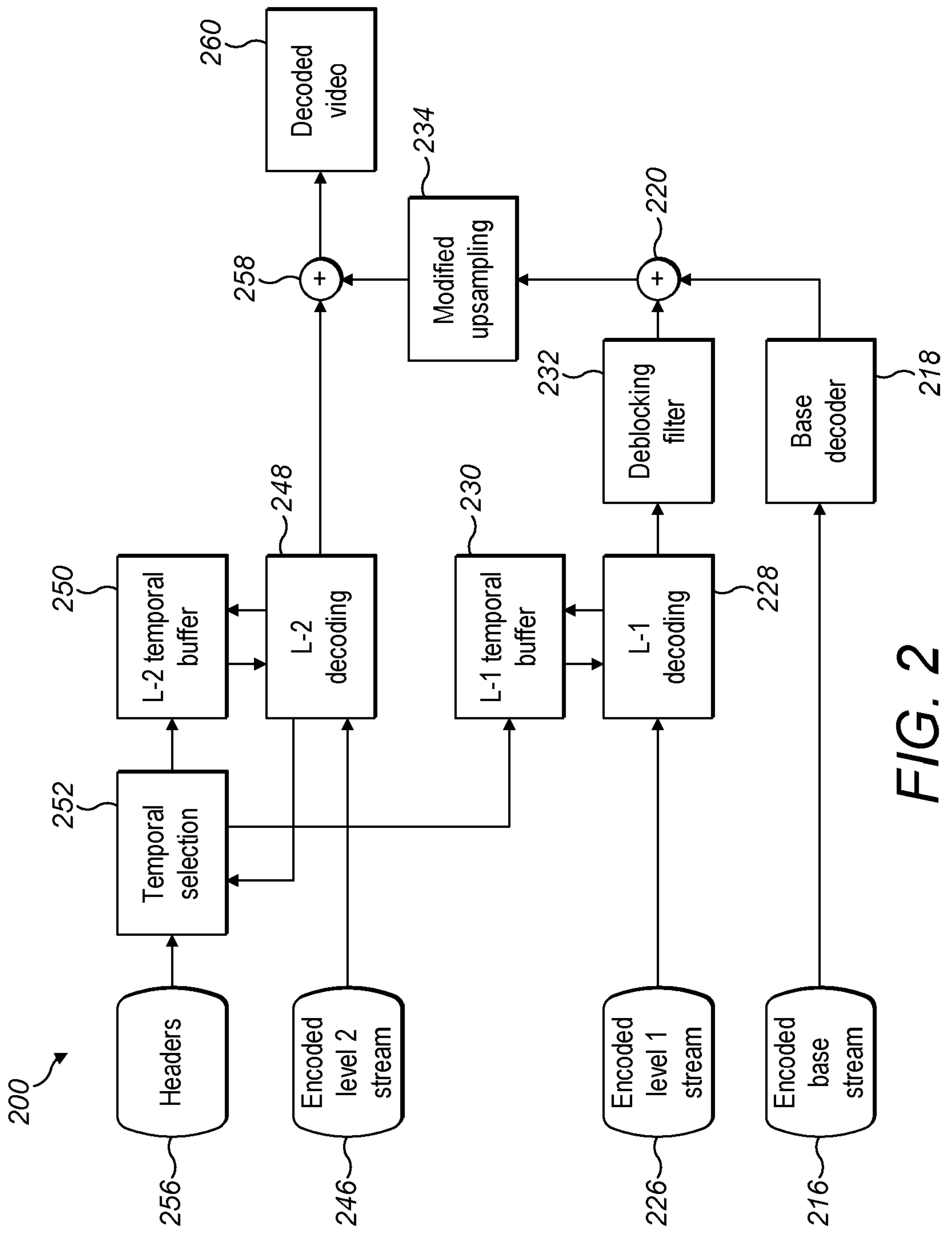

200
Headers
256
Encoded level 2 stream
246
Encoded level 1 stream
226
Encoded base stream
216
Temporal selection
252
L-2 temporal buffer
250
L-2 decoding
248
L-1 temporal buffer
230
L-1 decoding
228
Deblocking filter
232
Base decoder
218
+
220
Modified upsampling
234
+
258
Decoded video
260

FIG. 2

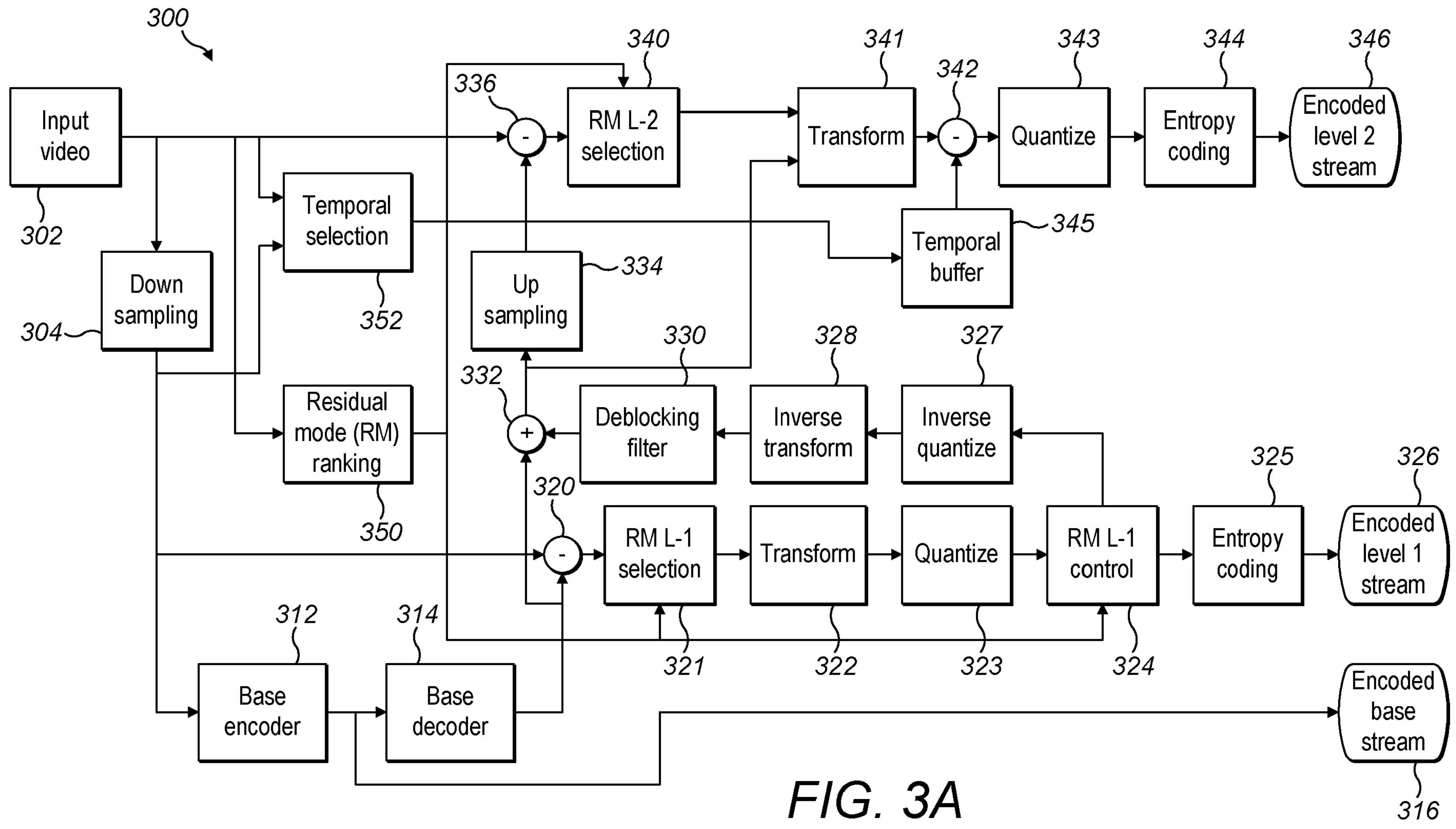

300
Input video
302
Down sampling
304
Temporal selection
352
Residual mode (RM) ranking
350
Base encoder
312
Base decoder
314
336
RM L-2 selection
340
Up sampling
334
332
330
Deblocking filter
328
Inverse transform
327
Inverse quantize
320
RM L-1 selection
321
Transform
322
Quantize
323
RM L-1 control
324
Entropy coding
325
Encoded level 1 stream
326
Transform
341
342
Quantize
343
Entropy coding
344
Encoded level 2 stream
346
Temporal buffer
345
Encoded base stream
316

FIG. 3A

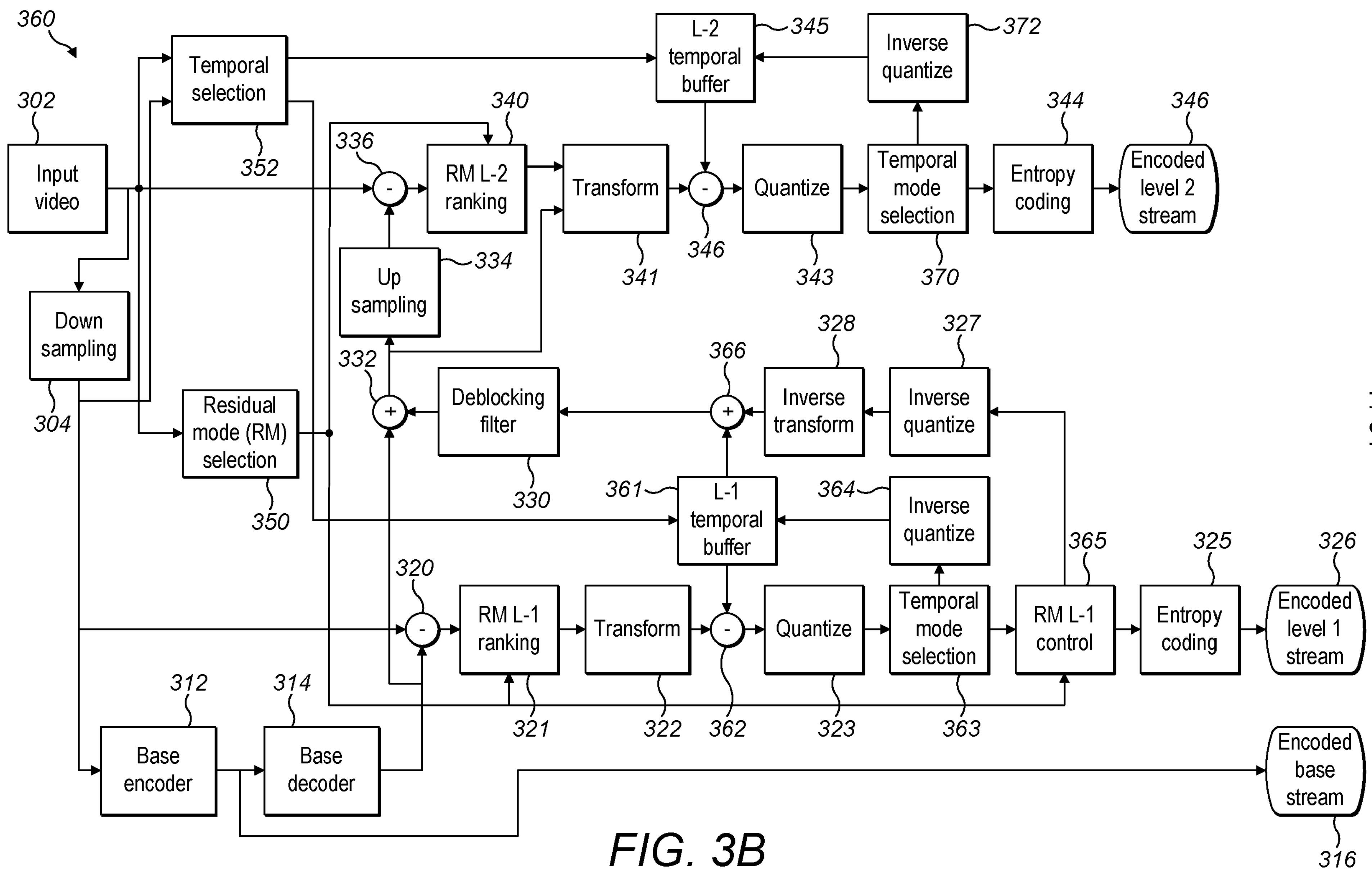

360
302
Input video
Down sampling
304
Temporal selection
352
Residual mode (RM) selection
350
L-2 temporal buffer
345
Inverse quantize
372
336
RM L-2 ranking
340
Transform
341
346
Quantize
343
Temporal mode selection
370
Entropy coding
344
Encoded level 2 stream
346
Up sampling
334
332
Deblocking filter
330
366
Inverse transform
328
Inverse quantize
327
L-1 temporal buffer
361
Inverse quantize
364
365
320
RM L-1 ranking
321
Transform
322
362
Quantize
323
Temporal mode selection
363
RM L-1 control
Entropy coding
325
Encoded level 1 stream
326
Base encoder
312
Base decoder
314
Encoded base stream
316

FIG. 3B

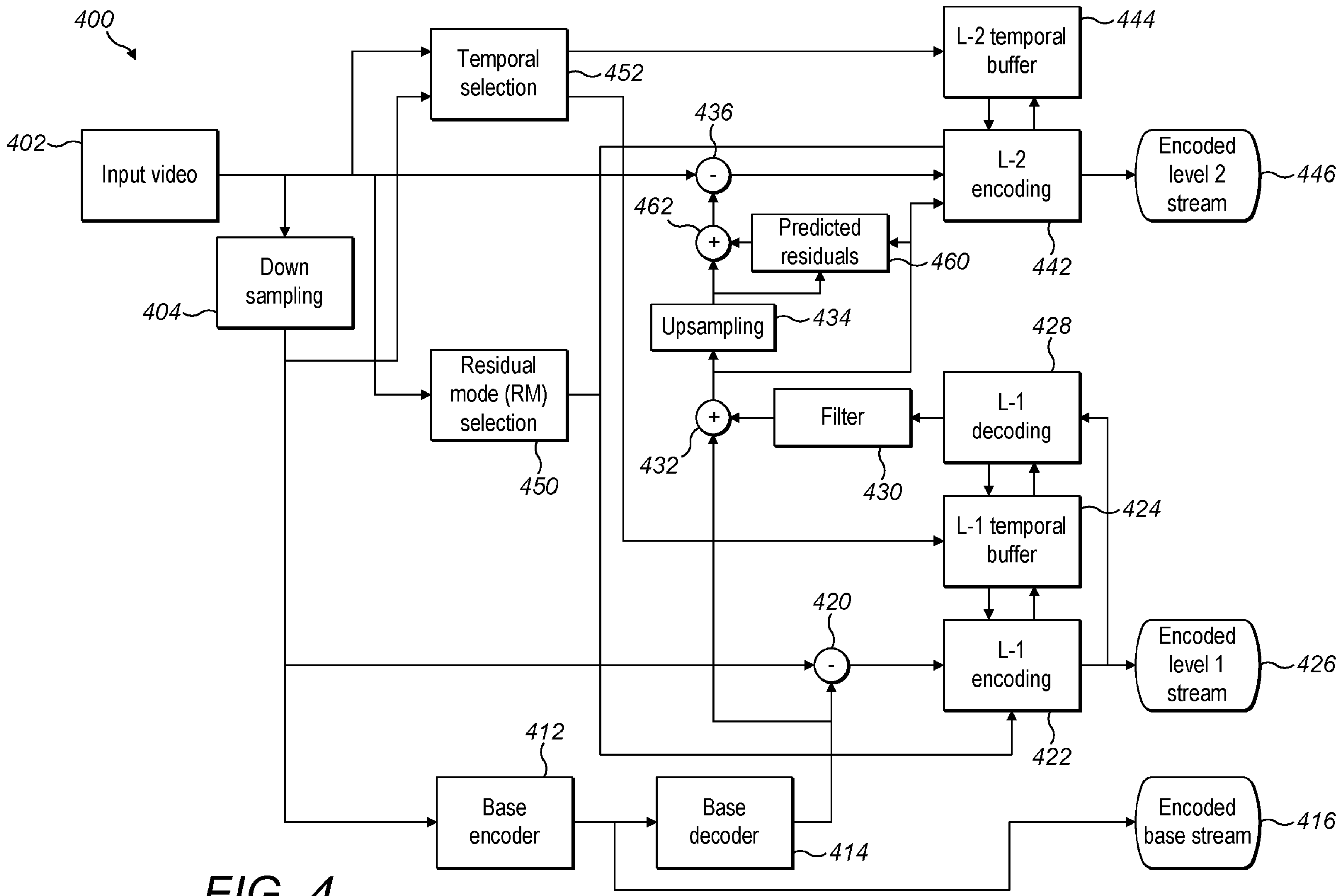

400
Input video
402
Down sampling
404
Temporal selection
452
Residual mode (RM) selection
450
Base encoder
412
Base decoder
414
436
462
Predicted residuals
460
Upsampling
434
432
Filter
430
420
L-2 temporal buffer
444
L-2 encoding
442
L-1 decoding
428
L-1 temporal buffer
424
L-1 encoding
422
Encoded level 2 stream
446
Encoded level 1 stream
426
Encoded base stream
416

FIG. 4

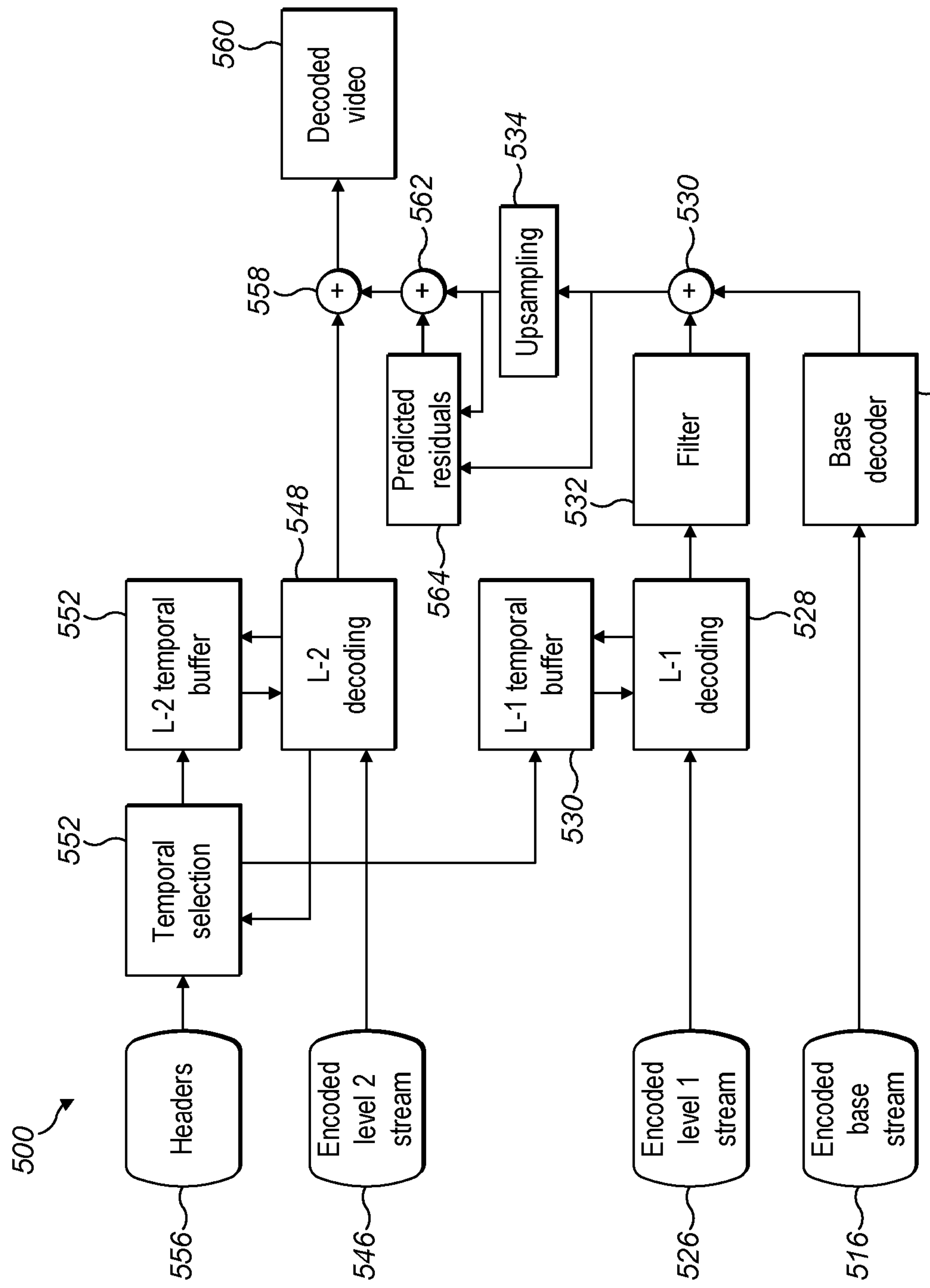

500
Headers
556
Encoded level 2 stream
546
Encoded level 1 stream
526
Encoded base stream
516
Temporal selection
552
L-2 temporal buffer
552
L-2 decoding
548
L-1 temporal buffer
530
L-1 decoding
528
Filter
532
Base decoder
518
Predicted residuals
564
Upsampling
534
+
530
+
562
+
558
Decoded video
560

FIG. 5A

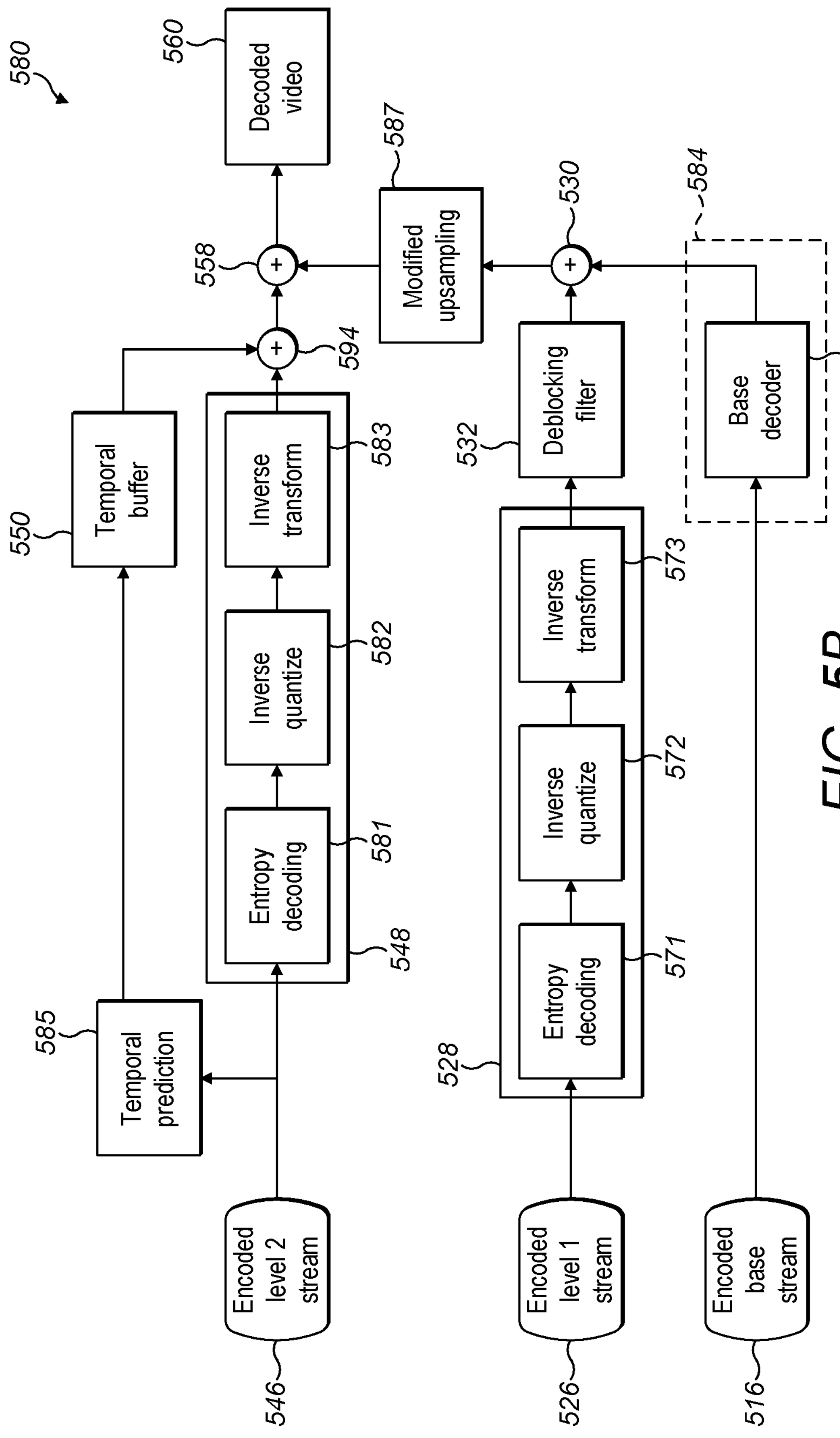

580
Encoded level 2 stream
546
Temporal prediction
585
Temporal buffer
550
Entropy decoding
581
Inverse quantize
582
Inverse transform
583
548
594
558
Decoded video
560
Modified upsampling
587
Encoded level 1 stream
526
528
Entropy decoding
571
Inverse quantize
572
Inverse transform
573
Deblocking filter
532
530
584
Encoded base stream
516
Base decoder
518

FIG. 5B

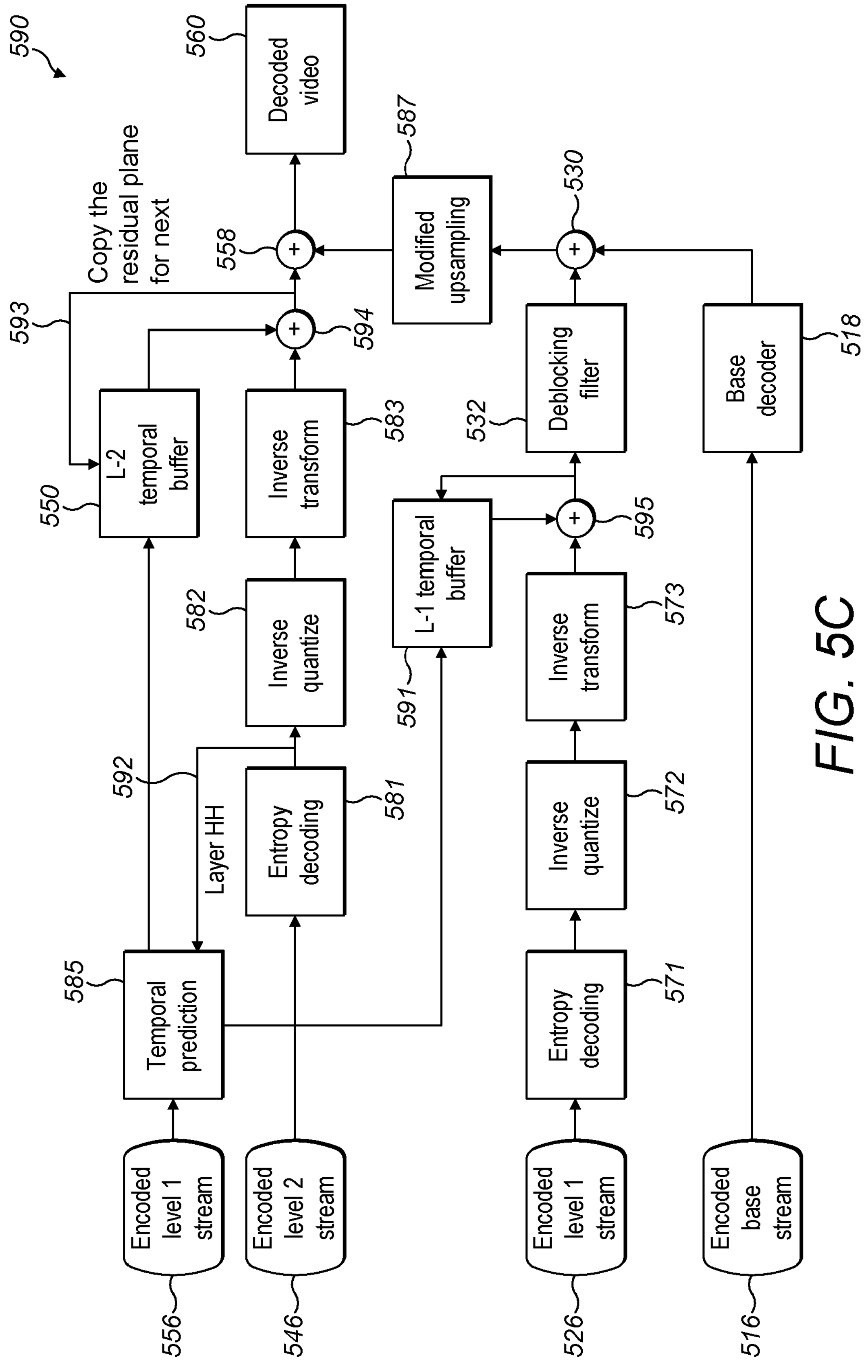

590
Encoded level 1 stream
556
Encoded level 2 stream
546
Encoded level 1 stream
526
Encoded base stream
516
Temporal prediction
585
Entropy decoding
581
Layer HH
592
Inverse quantize
582
Inverse transform
583
L-2 temporal buffer
550
593
Copy the residual plane for next
594
558
Decoded video
560
Modified upsampling
587
L-1 temporal buffer
591
Entropy decoding
571
Inverse quantize
572
Inverse transform
573
595
Deblocking filter
532
530
Base decoder
518

FIG. 5C

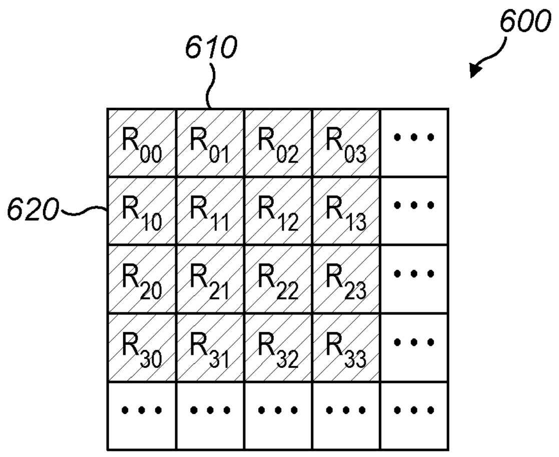

600
610
620
R_{00}
R_{01}
R_{02}
R_{03}
R_{10}
R_{11}
R_{12}
R_{13}
R_{20}
R_{21}
R_{22}
R_{23}
R_{30}
R_{31}
R_{32}
R_{33}

FIG. 6A

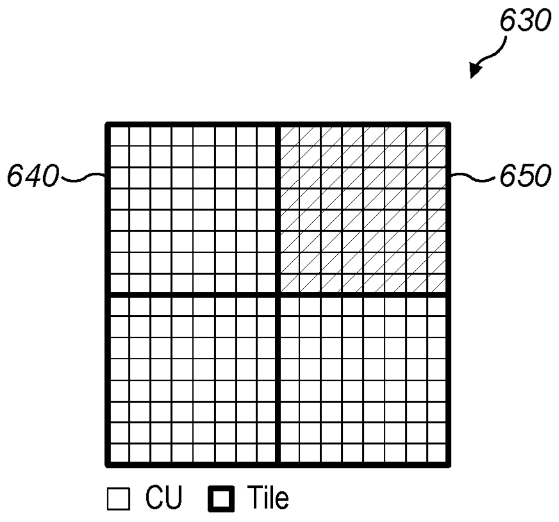

630
640
650
□ CU
▣ Tile

FIG. 6B

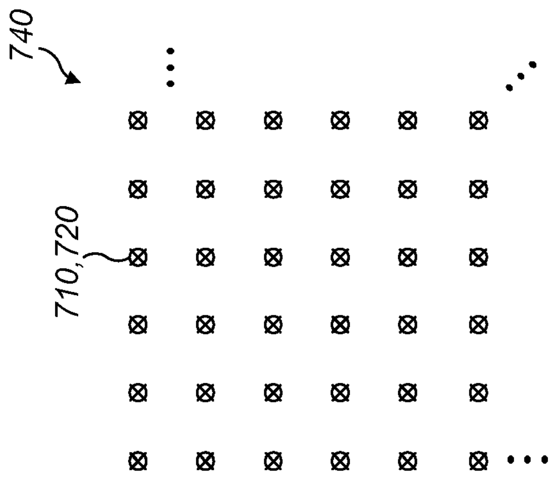


FIG. 7C

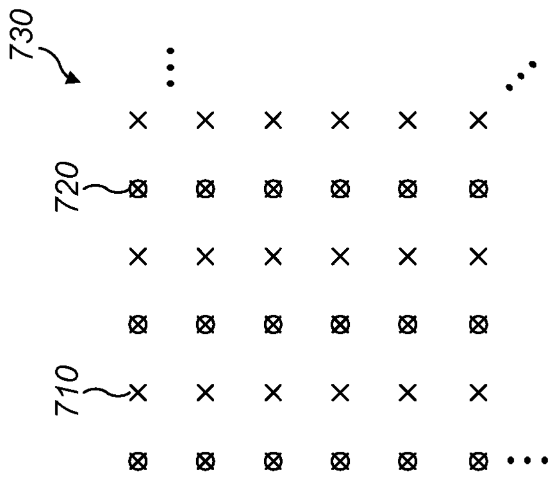


FIG. 7B

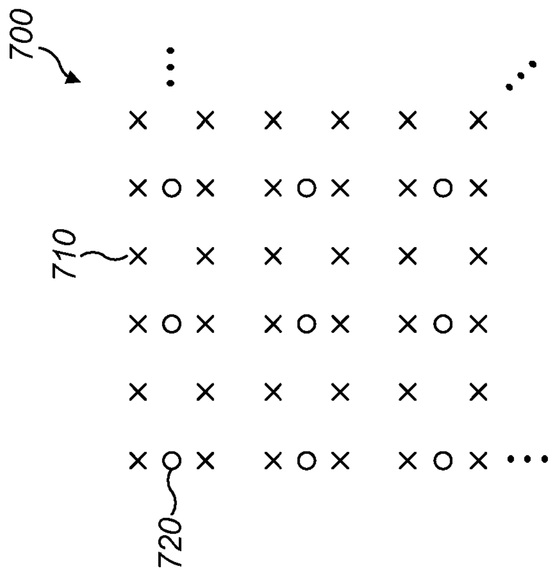


FIG. 7A

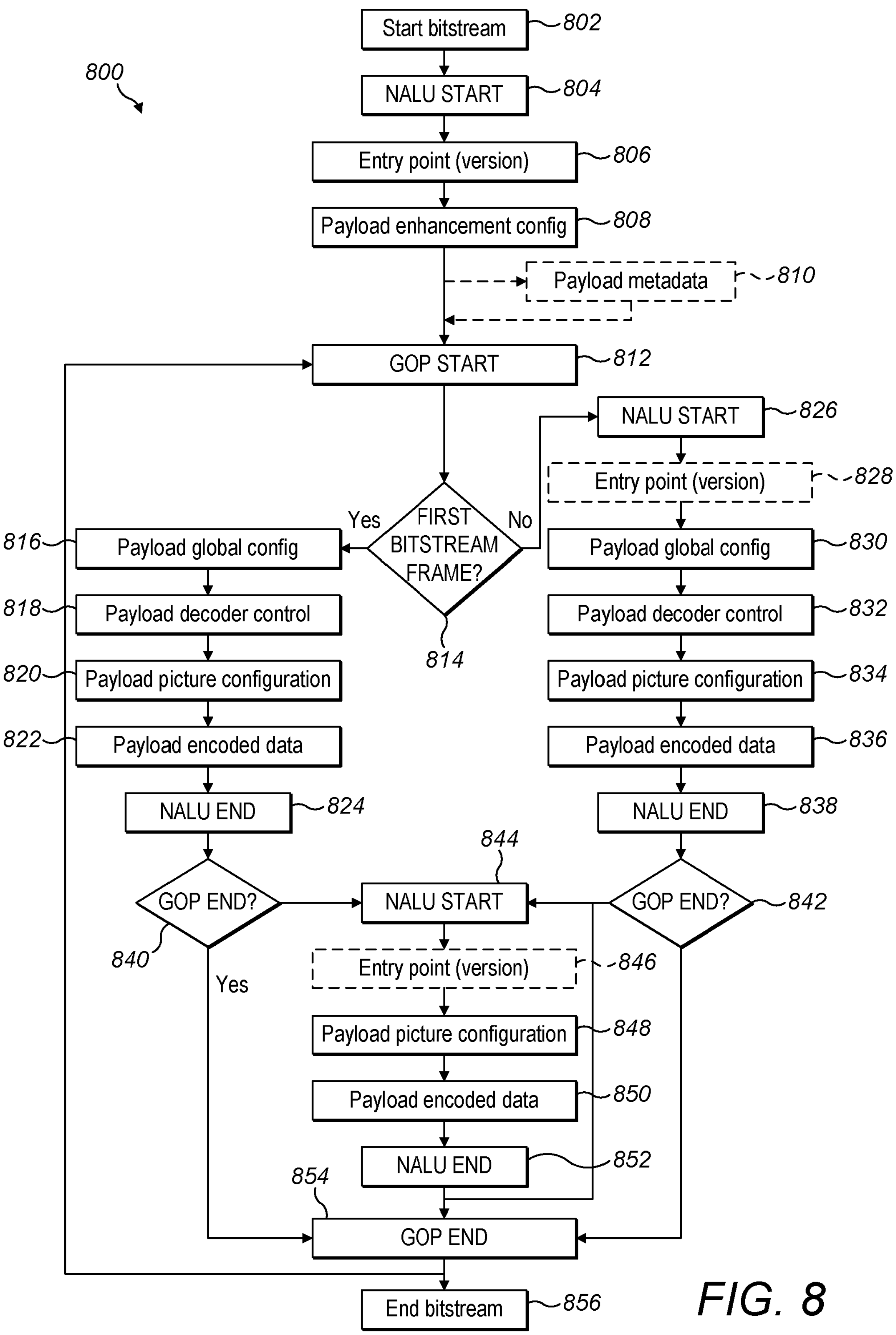

800
Start bitstream
802
NALU START
804
Entry point (version)
806
Payload enhancement config
808
Payload metadata
810
GOP START
812
NALU START
826
Entry point (version)
828
Yes
FIRST BITSTREAM FRAME?
No
816
Payload global config
Payload global config
830
818
Payload decoder control
Payload decoder control
832
814
820
Payload picture configuration
Payload picture configuration
834
822
Payload encoded data
Payload encoded data
836
NALU END
824
NALU END
838
844
GOP END?
NALU START
GOP END?
842
840
Entry point (version)
846
Yes
Payload picture configuration
848
Payload encoded data
850
NALU END
852
854
GOP END
End bitstream
856

FIG. 8

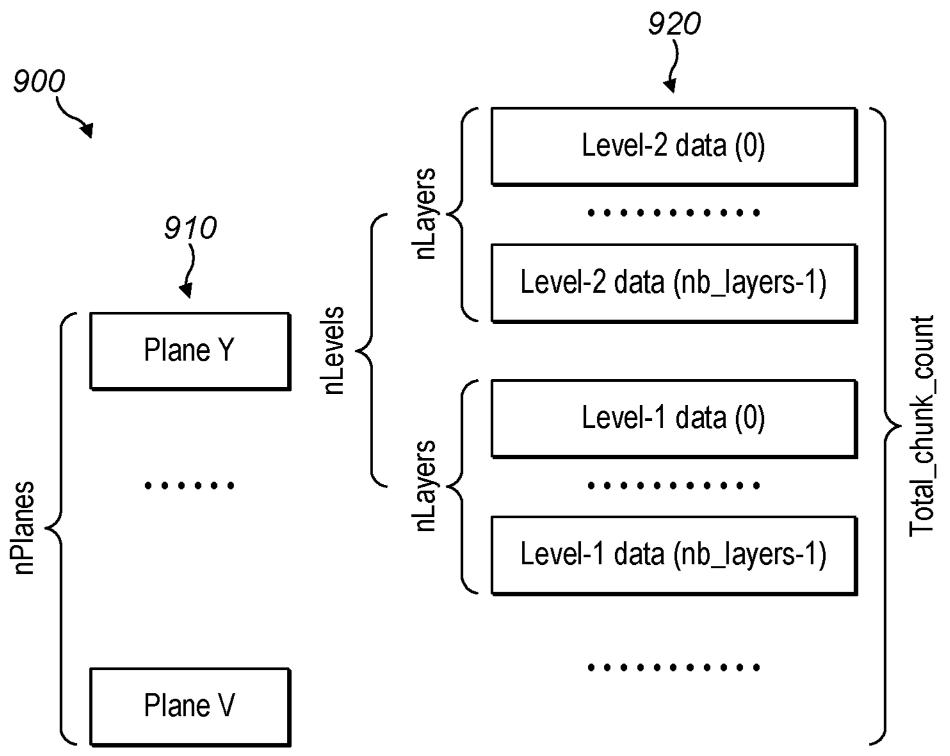

900
910
920
Plane Y
......
Plane V
nPlanes
nLevels
nLayers
Level-2 data (0)
Level-2 data (nb_layers-1)
nLayers
Level-1 data (0)
Level-1 data (nb_layers-1)
Total_chunk_count

FIG. 9A

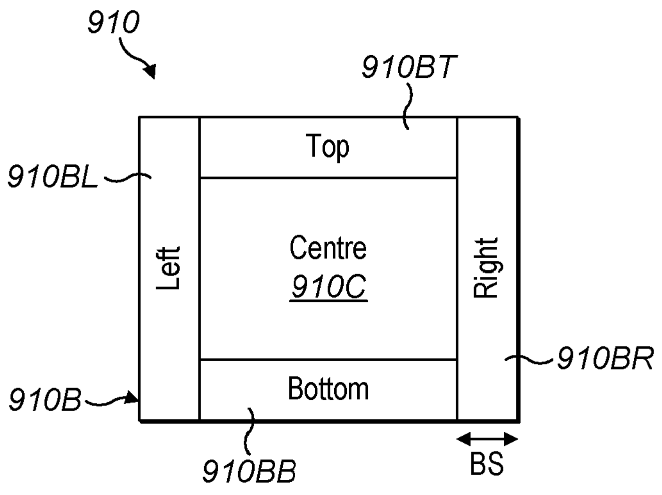

910
910BT
Top
910BL
Left
Centre
910C
Right
910BR
910B
Bottom
910BB
BS

FIG. 9B

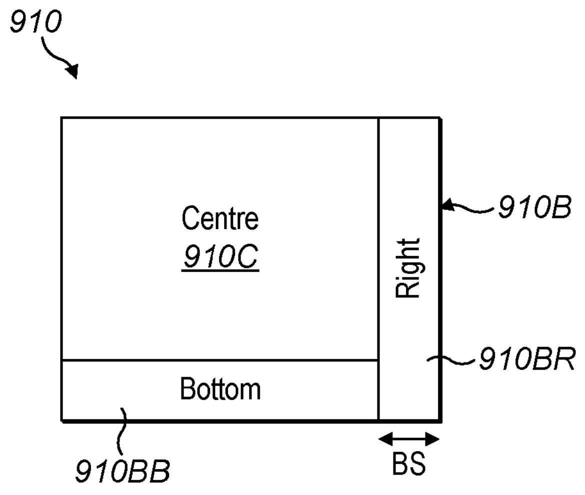

910
Centre
910C
Right
910B
910BR
Bottom
910BB
BS

FIG. 9C

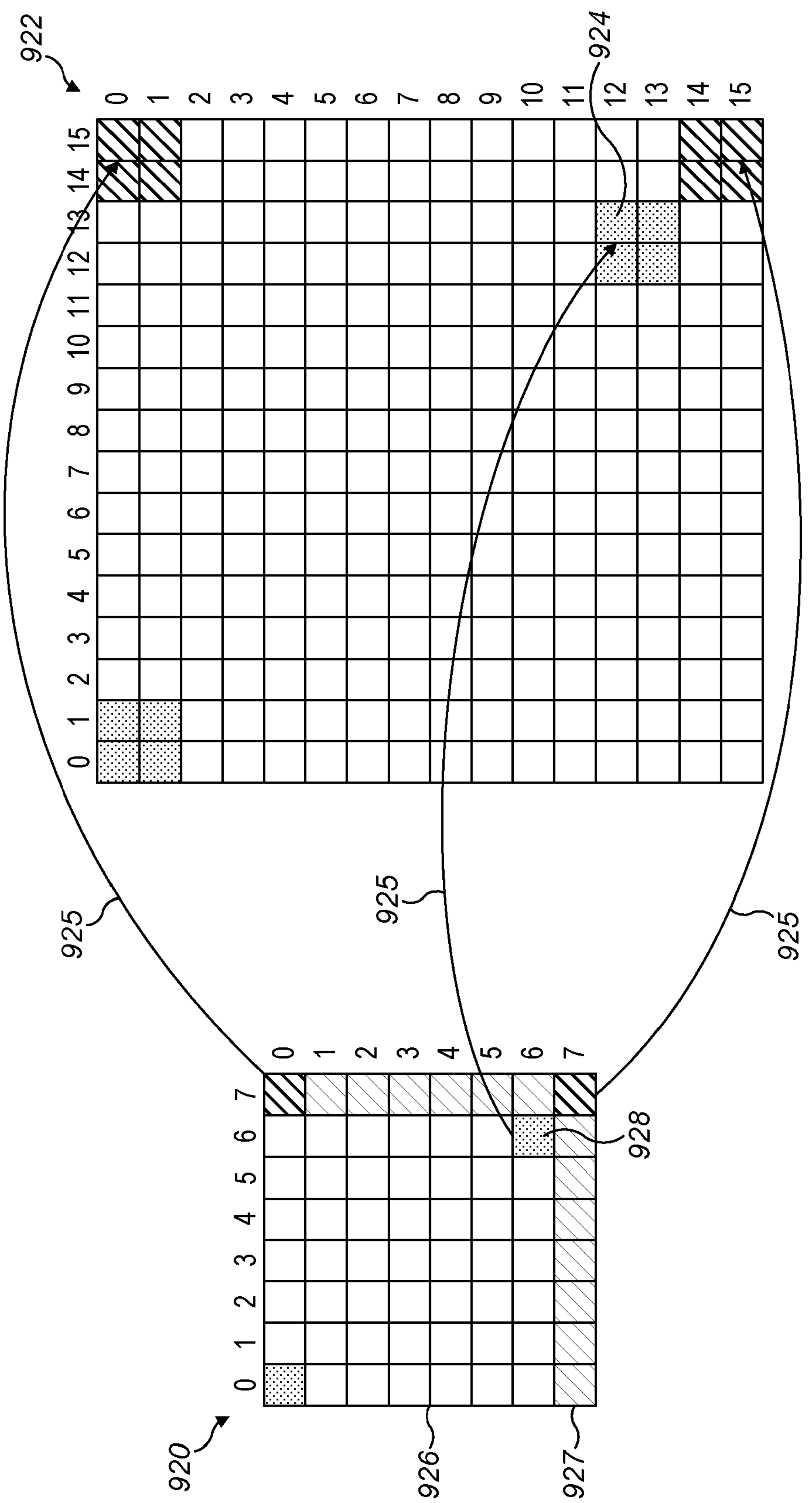

922
924
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
925
925
925
920
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
926
927
928

FIG. 9D

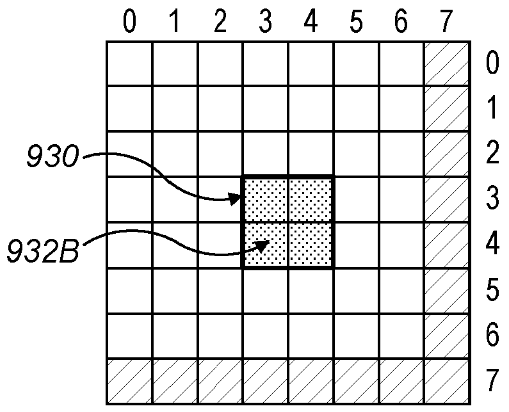

0
1
2
3
4
5
6
7
930
932B

FIG. 9E

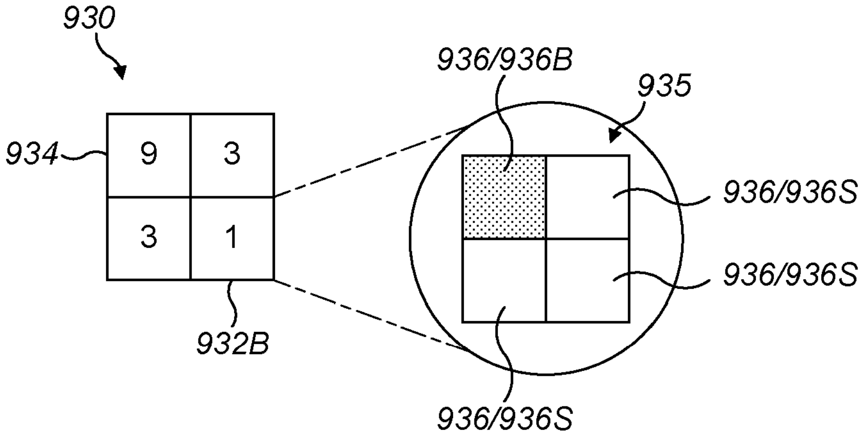
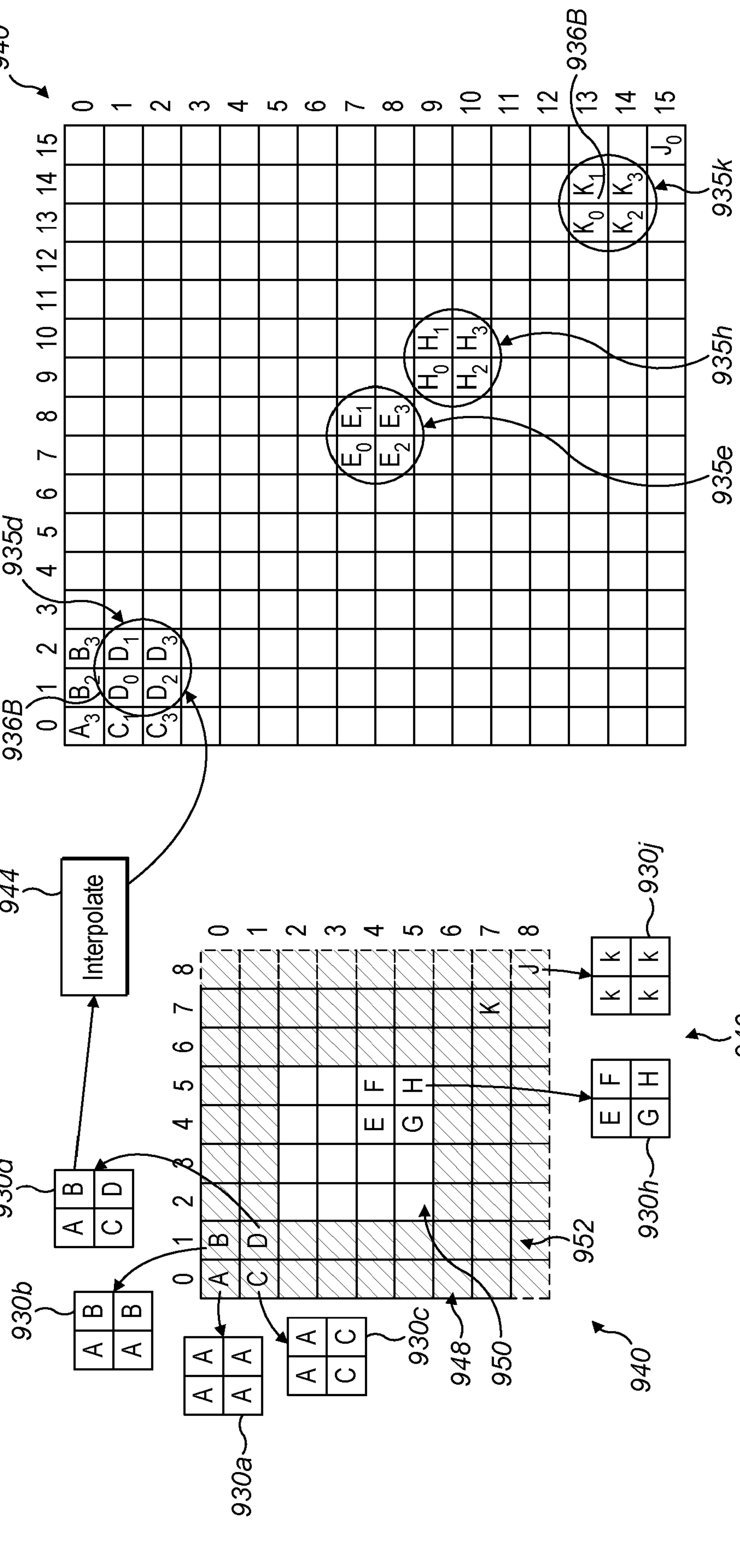

930
934
9
3
3
1
932B
936/936B
935
936/936S
936/936S
936/936S

FIG. 9F



940
944
Interpolate
930a
930b
930c
930d
930h
930j
935d
935e
935h
935k
936B
946
948
950
952

FIG. 9G

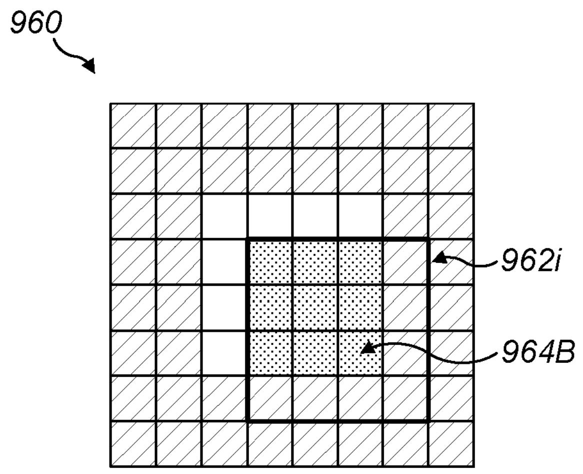


FIG. 9H

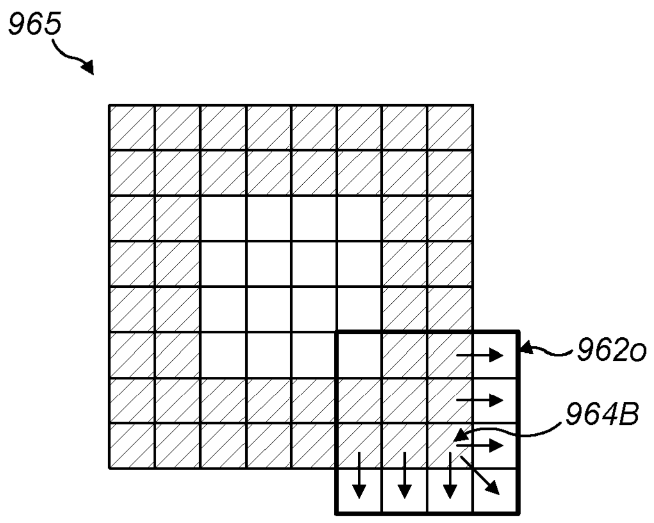


FIG. 9I

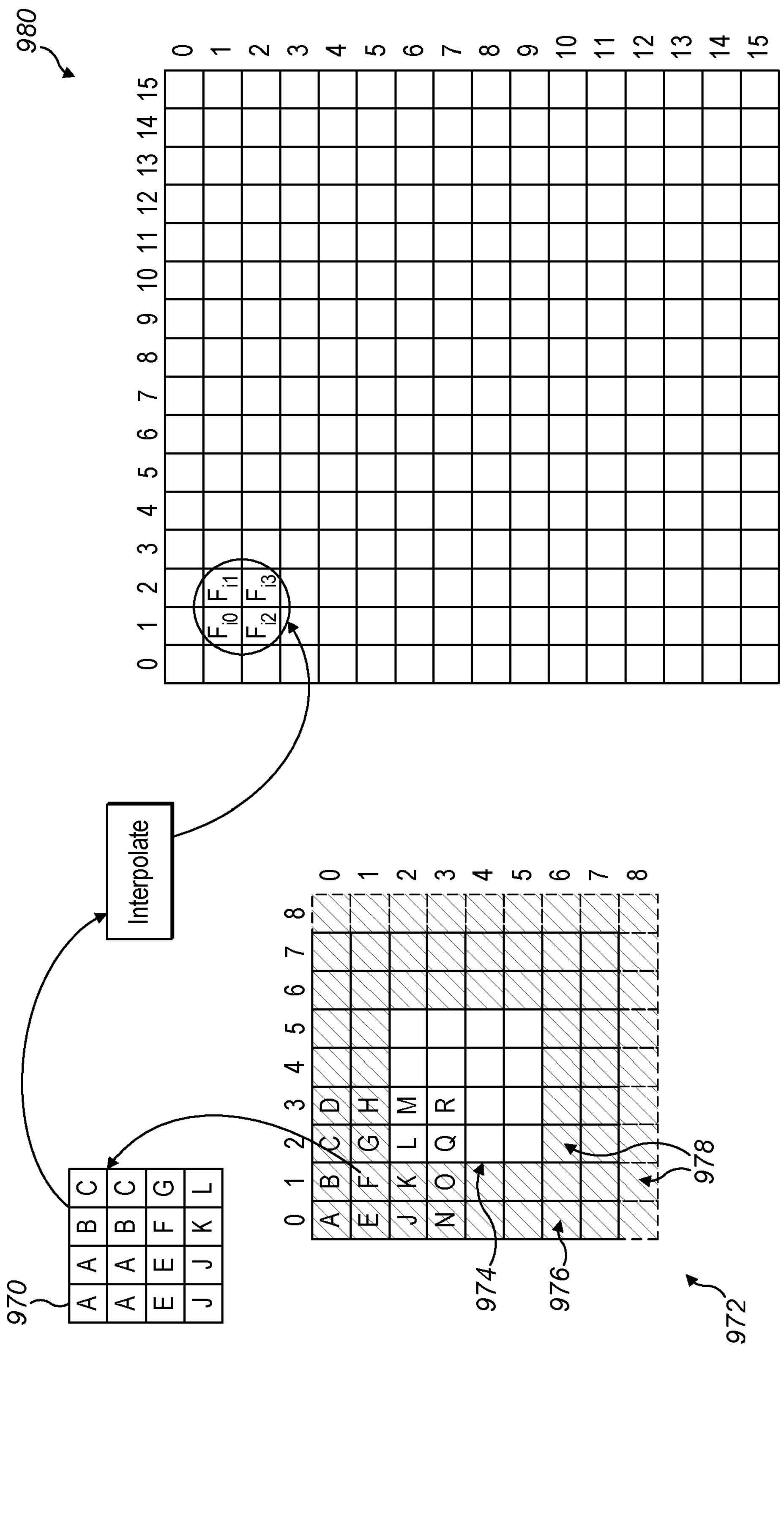

980
970
972
974
976
978
Interpolate
A B C
A B C
E F G
J K L
A B C D
E F G H
J K L M
N O Q R
F_i0 F_i1
F_i2 F_i3

FIG. 9J

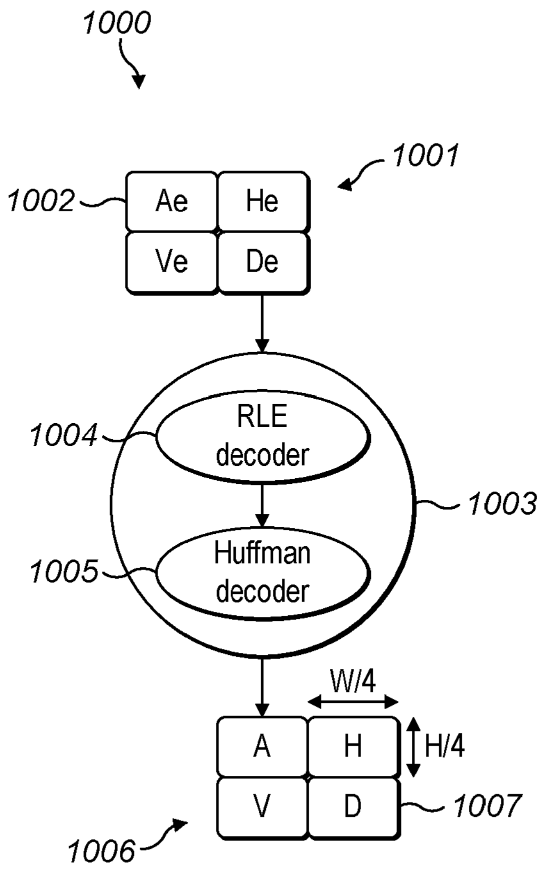


FIG. 10A

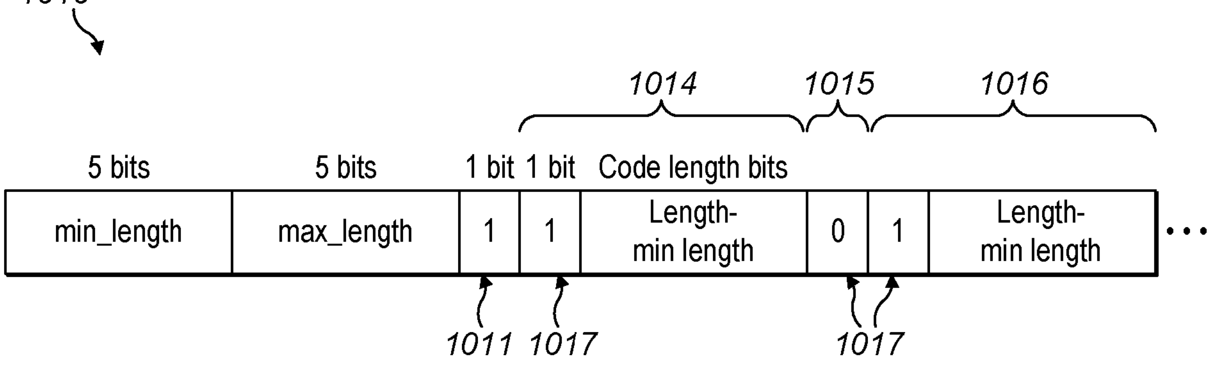


FIG. 10B

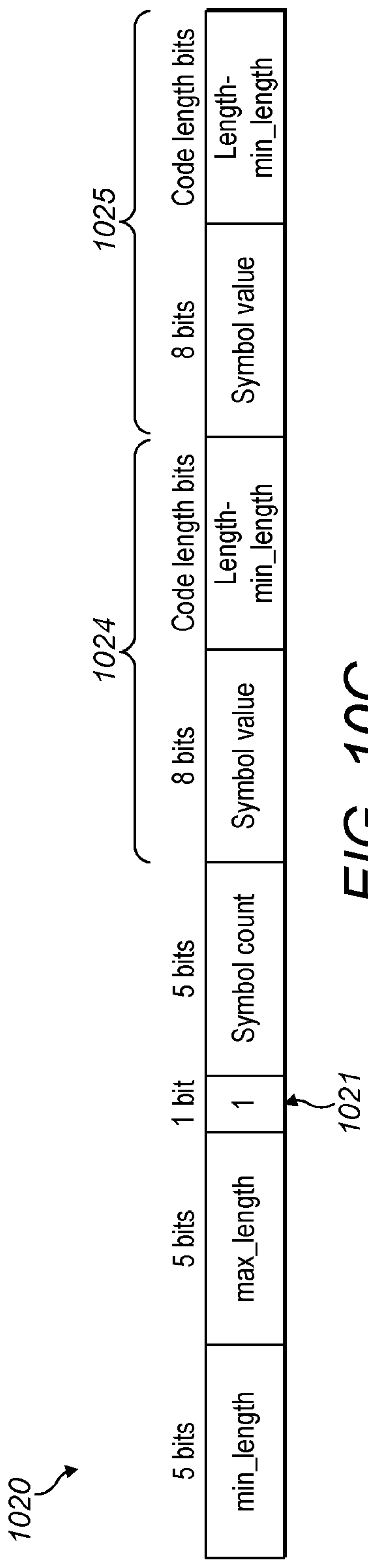


FIG. 10C

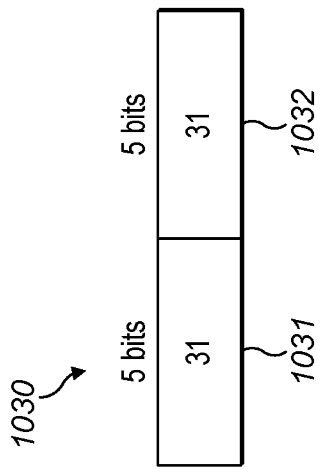


FIG. 10D

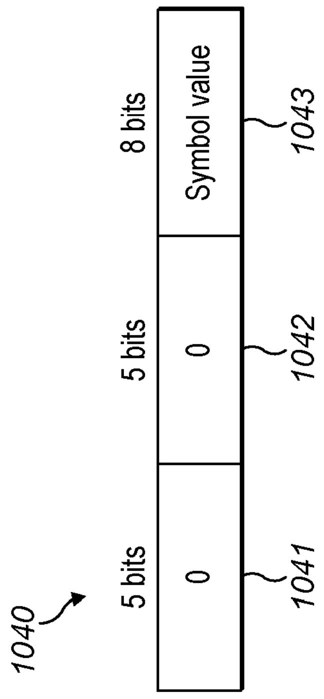


FIG. 10E

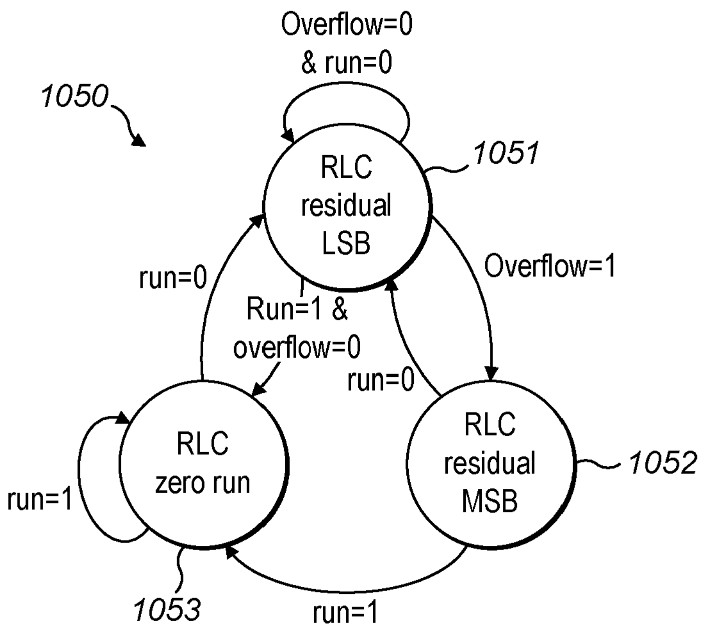


FIG. 10F

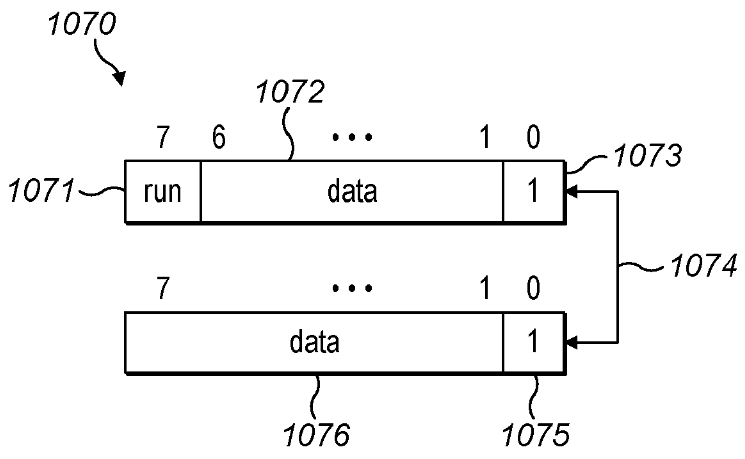


FIG. 10G

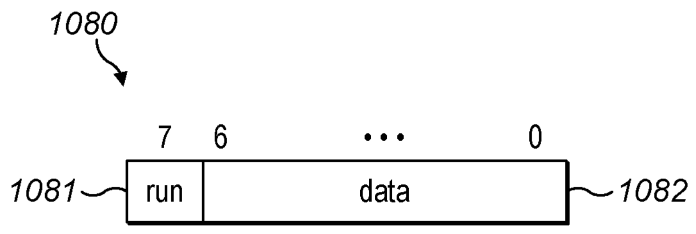


FIG. 10H

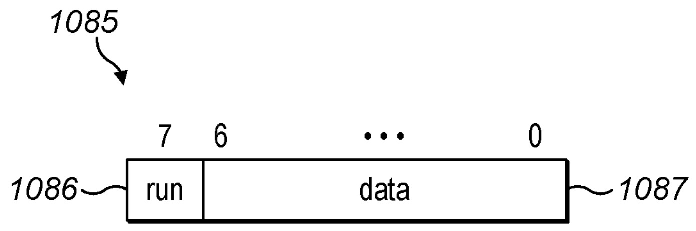


FIG. 10I

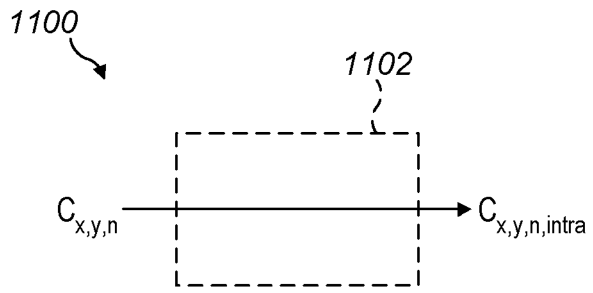


FIG. 11A

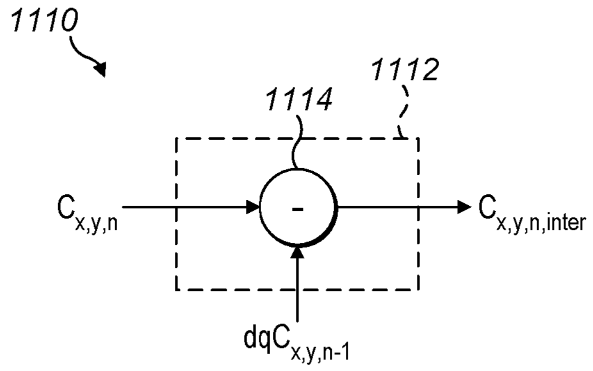


FIG. 11B

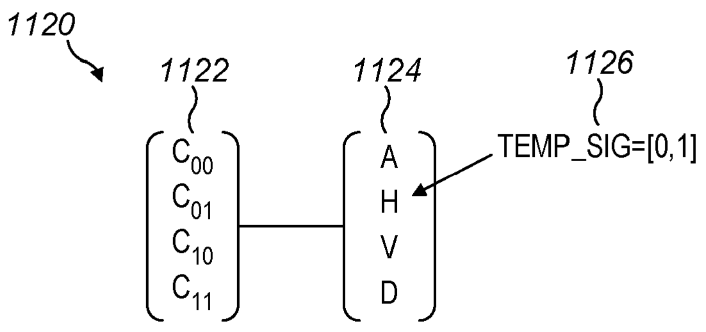


FIG. 11C

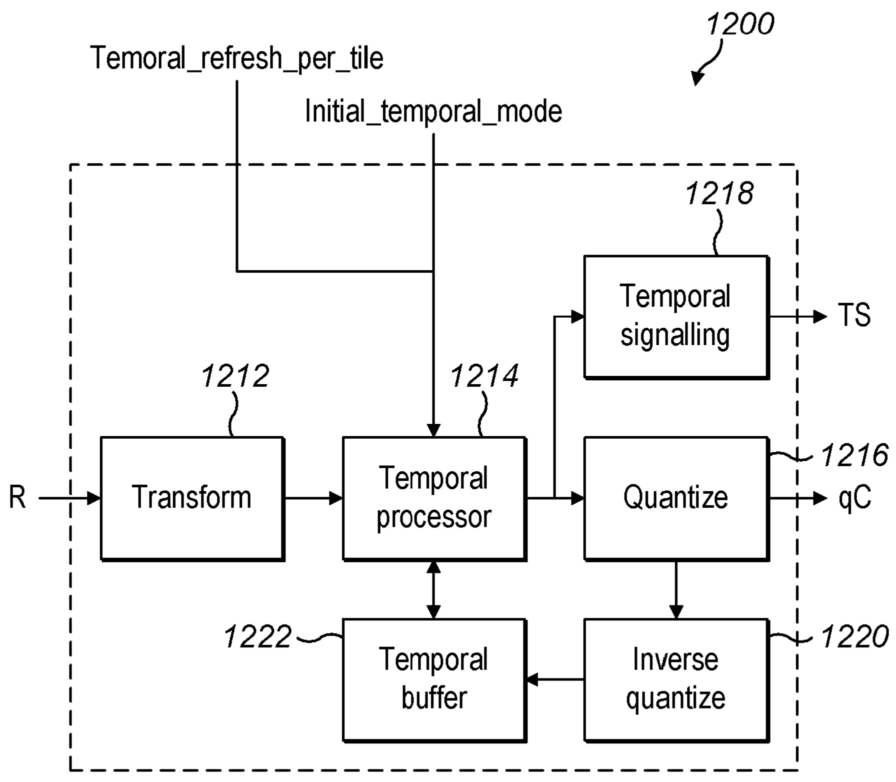


FIG. 12A

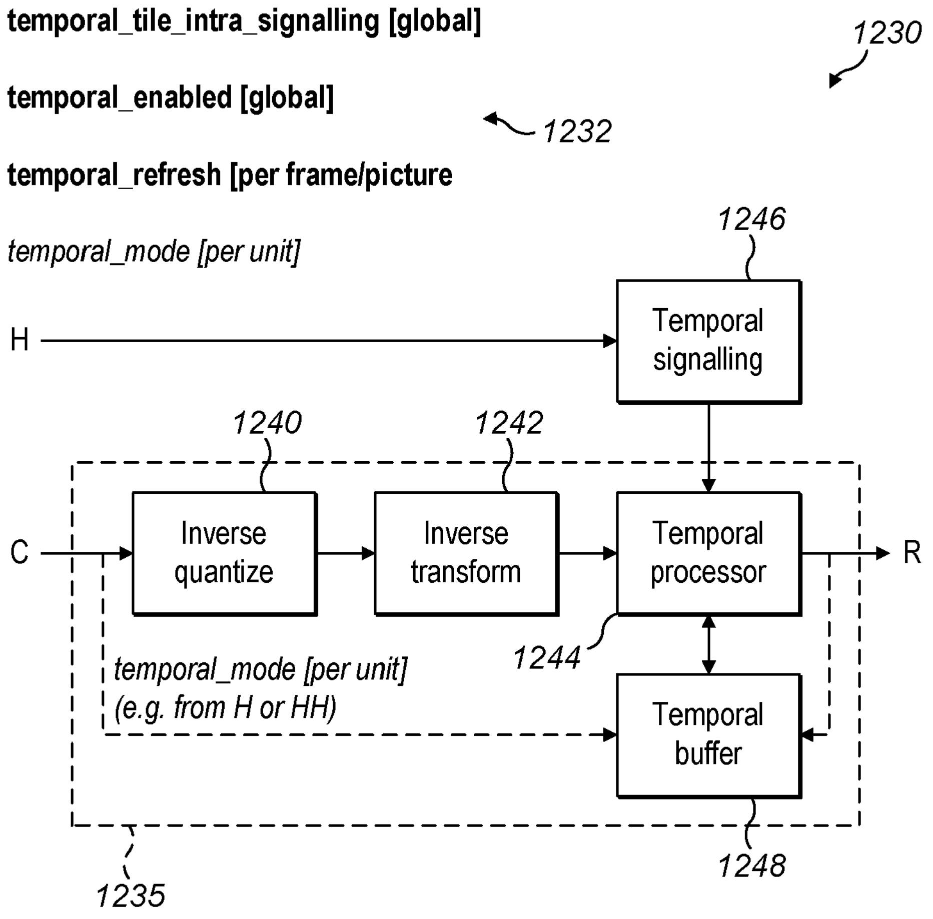


FIG. 12B

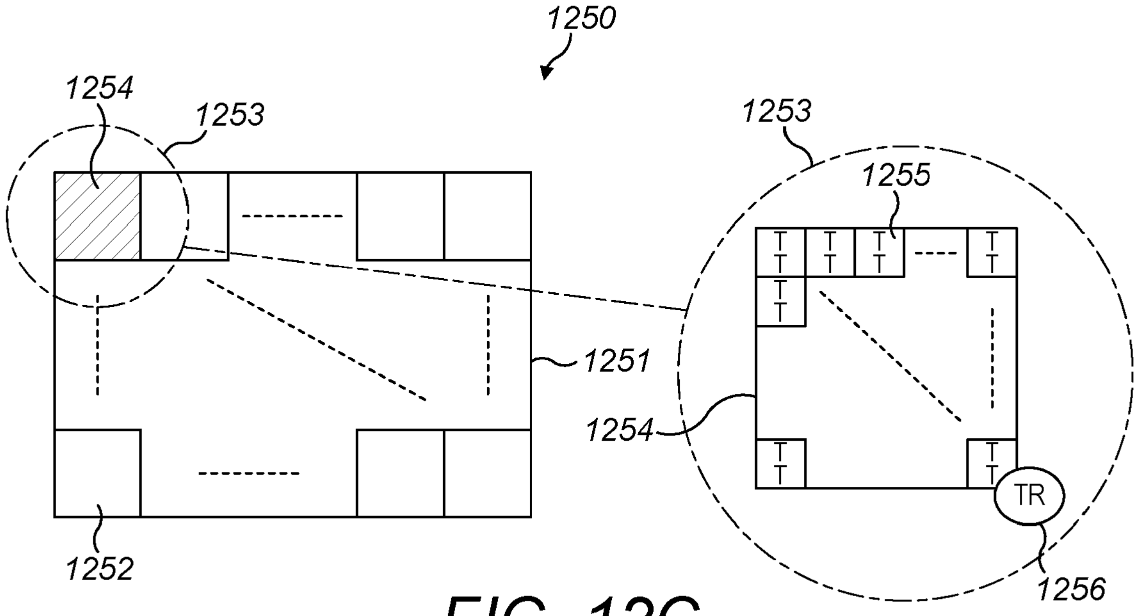


FIG. 12C

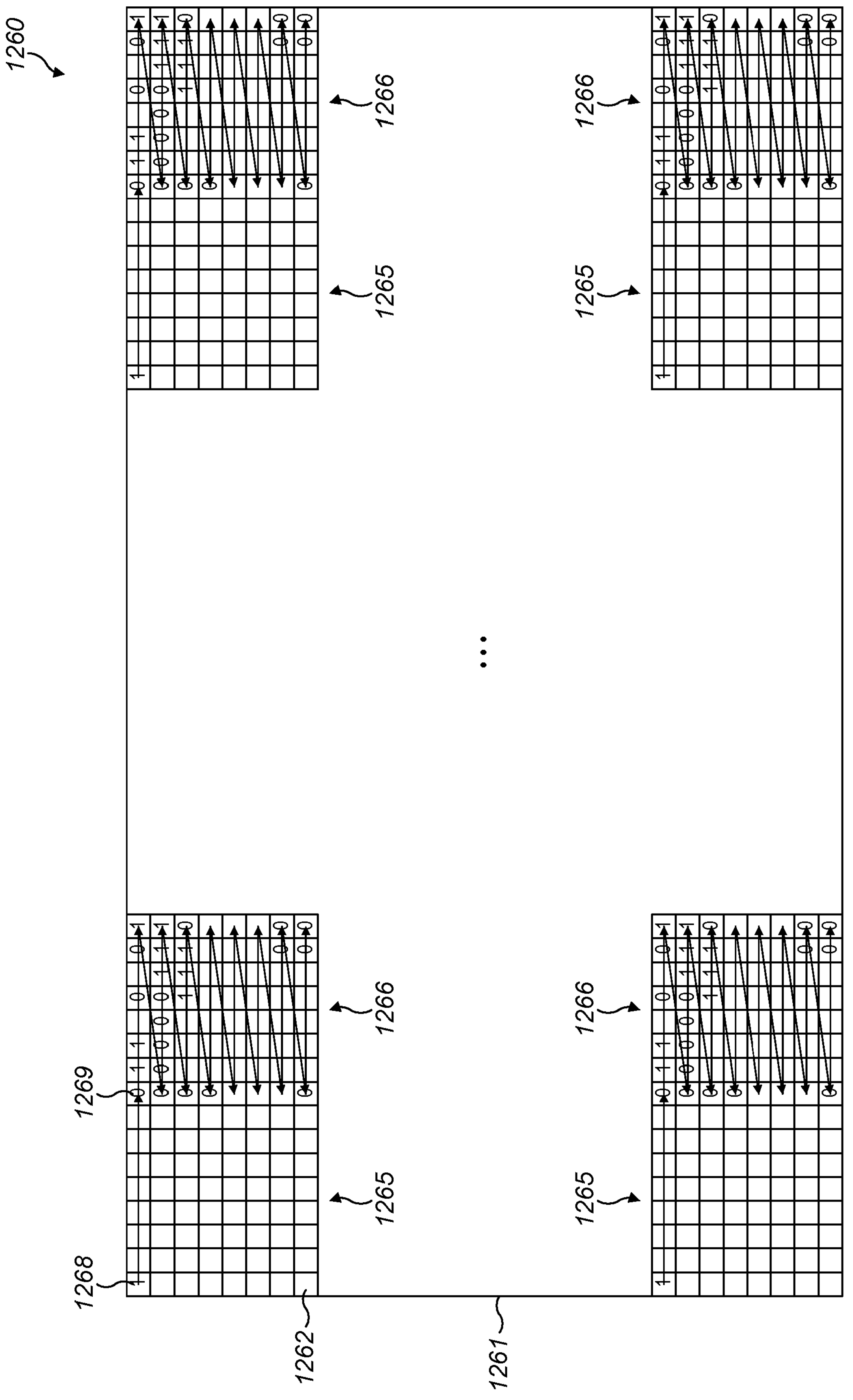


FIG. 12D

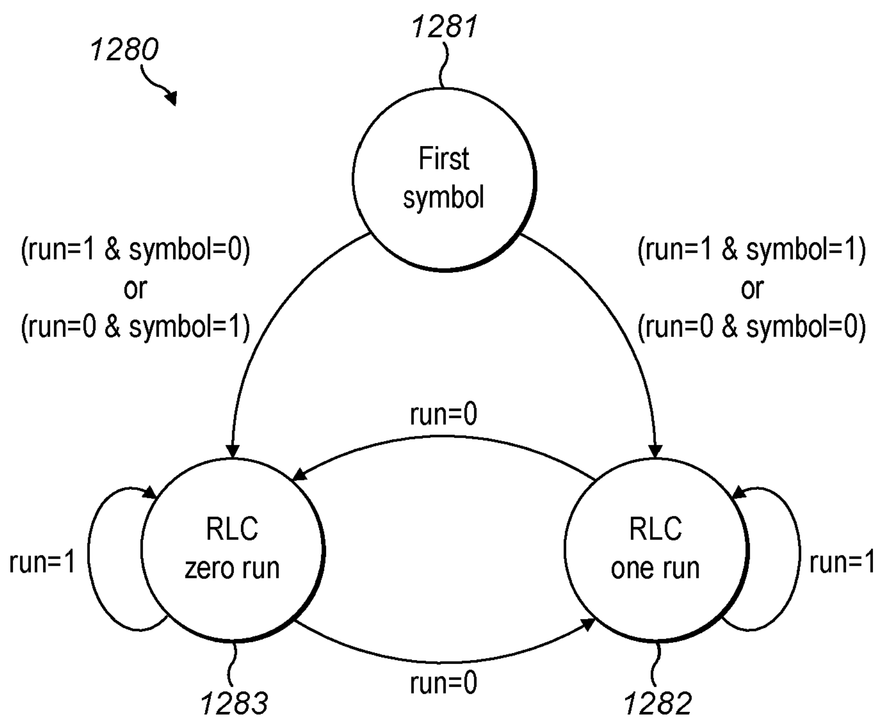

1280
1281
First symbol
(run=1 & symbol=0) or (run=0 & symbol=1)
(run=1 & symbol=1) or (run=0 & symbol=0)
run=0
run=1
RLC zero run
RLC one run
run=1
1283
run=0
1282

FIG. 12E

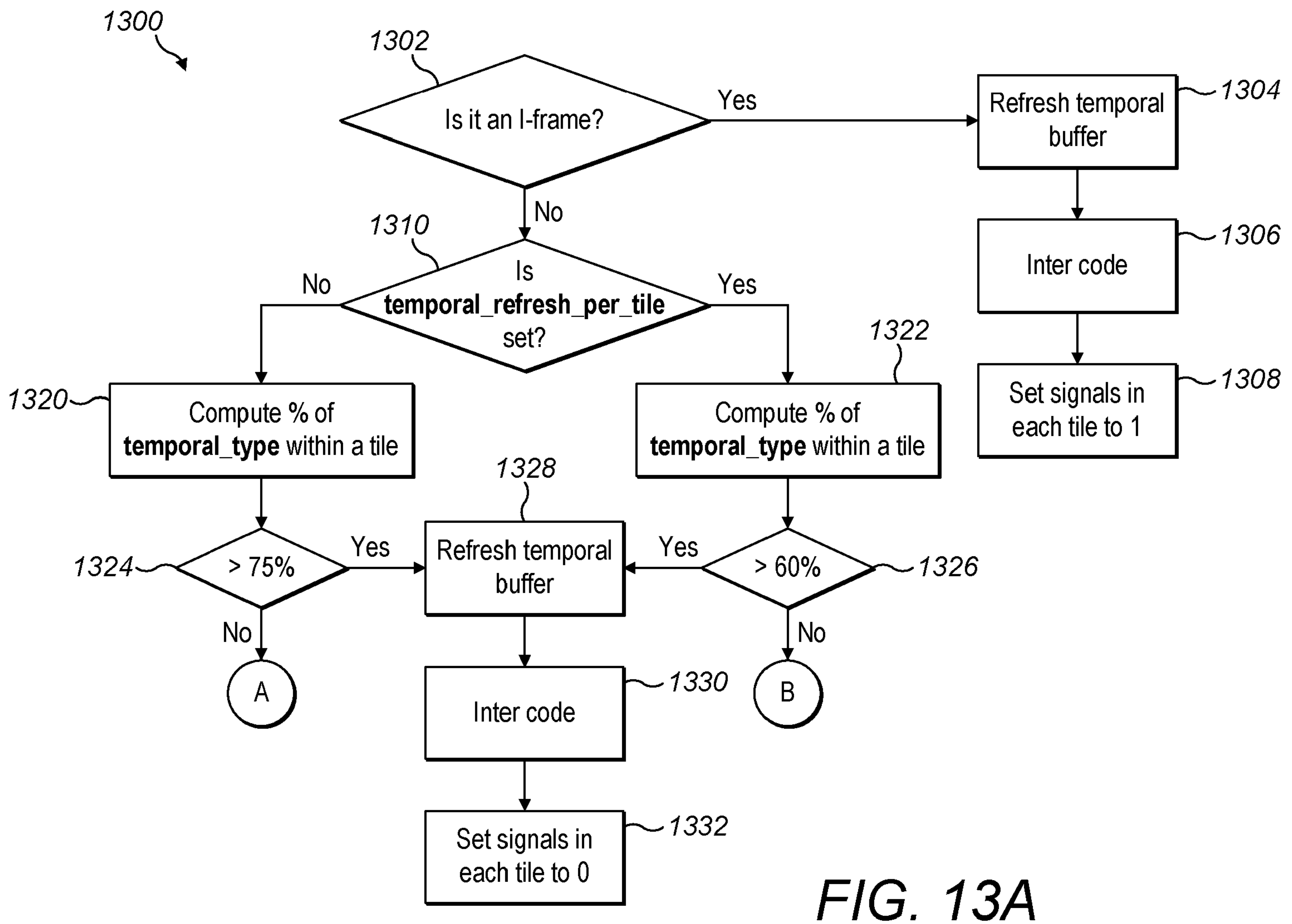

1300
1302
Is it an I-frame?
Yes
No
1304
Refresh temporal buffer
1306
Inter code
1308
Set signals in each tile to 1
1310
Is **temporal_refresh_per_tile** set?
No
Yes
1320
Compute % of **temporal_type** within a tile
1322
Compute % of **temporal_type** within a tile
1324
> 75%
Yes
No
A
1328
Refresh temporal buffer
1326
> 60%
Yes
No
B
1330
Inter code
1332
Set signals in each tile to 0

FIG. 13A

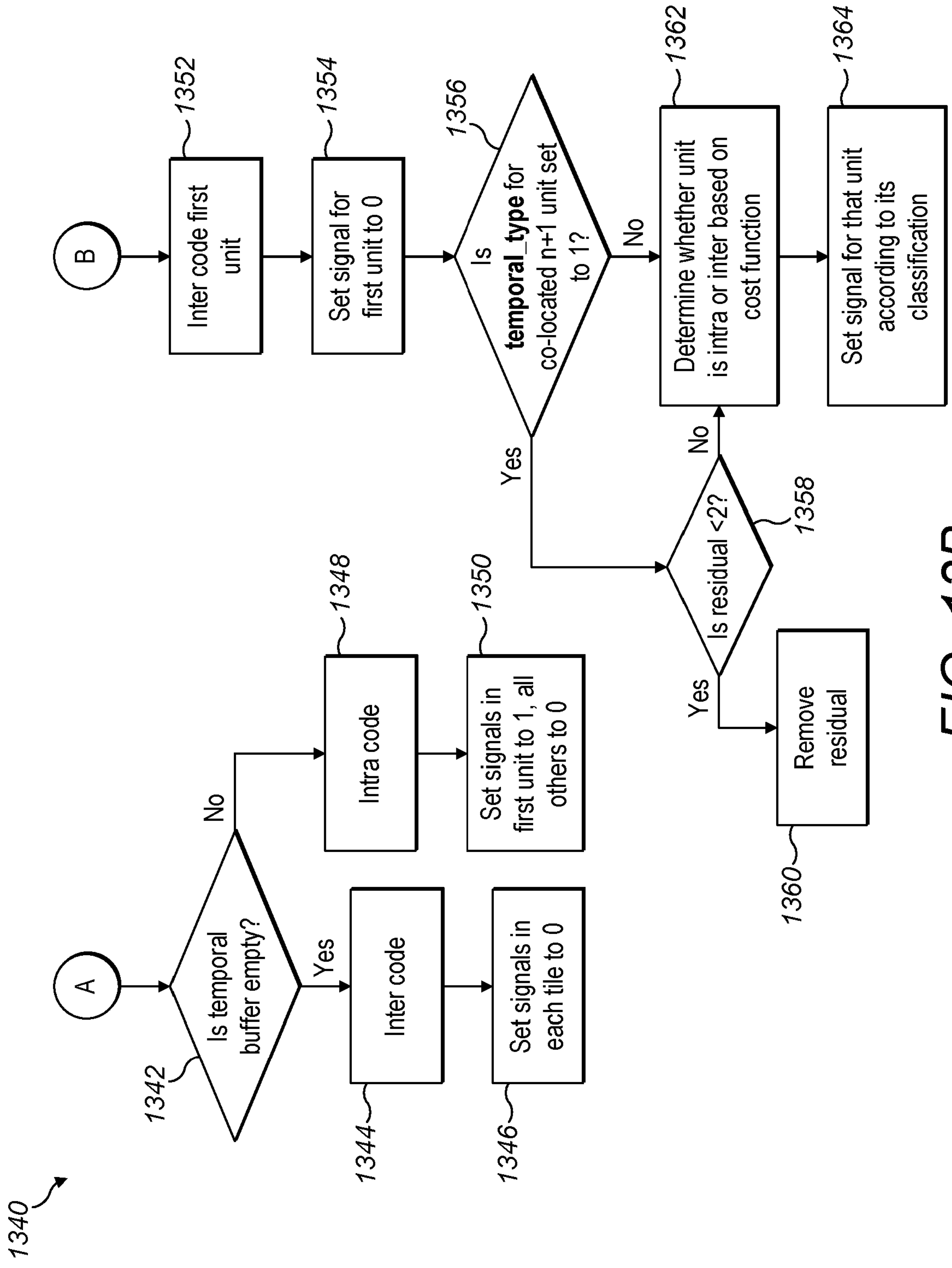

1340
A
1342
Is temporal buffer empty?
Yes
No
1344
Inter code
1346
Set signals in each tile to 0
1348
Intra code
1350
Set signals in first unit to 1, all others to 0
B
1352
Inter code first unit
1354
Set signal for first unit to 0
1356
Is **temporal_type** for co-located n+1 unit set to 1?
Yes
No
1358
Is residual <2?
Yes
No
1360
Remove residual
1362
Determine whether unit is intra or inter based on cost function
1364
Set signal for that unit according to its classification

FIG. 13B

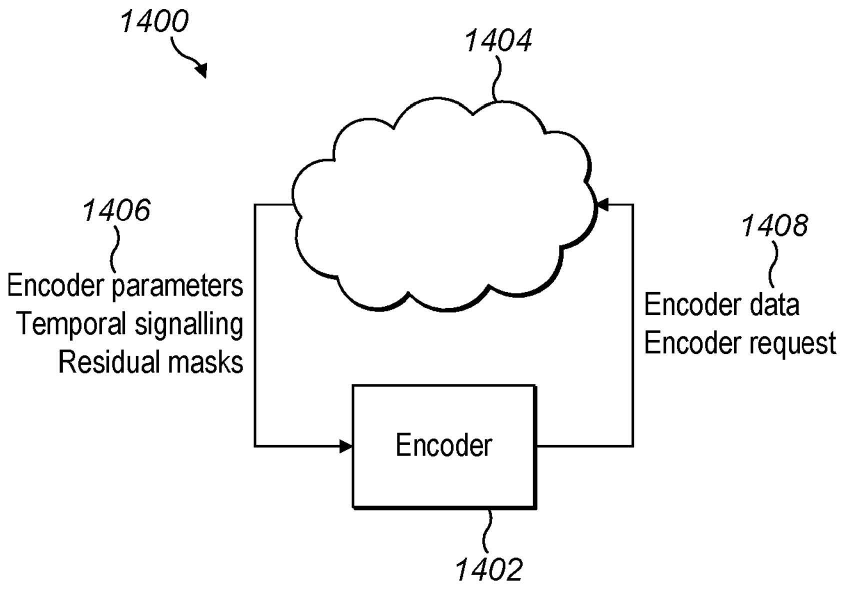


FIG. 14A

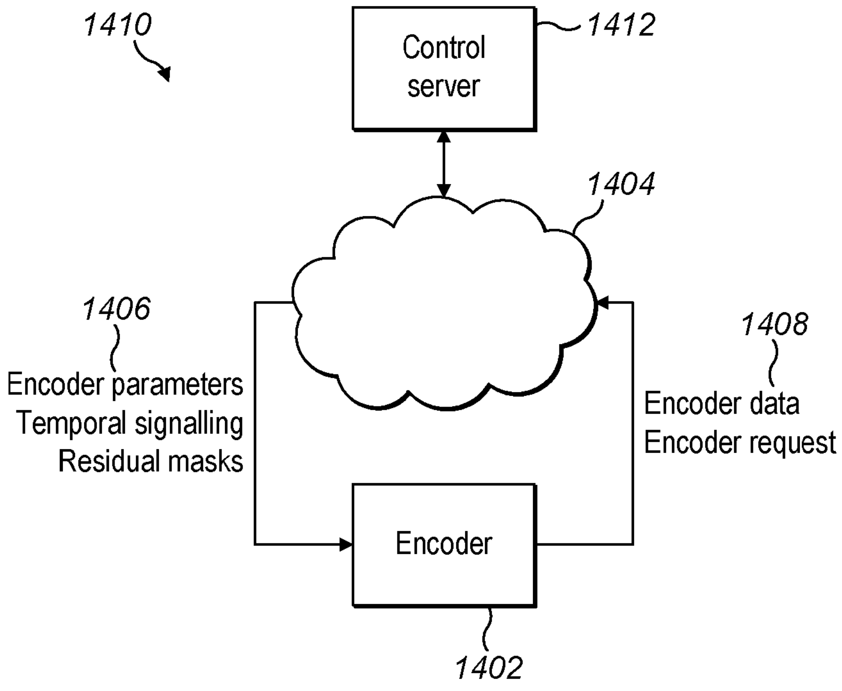


FIG. 14B

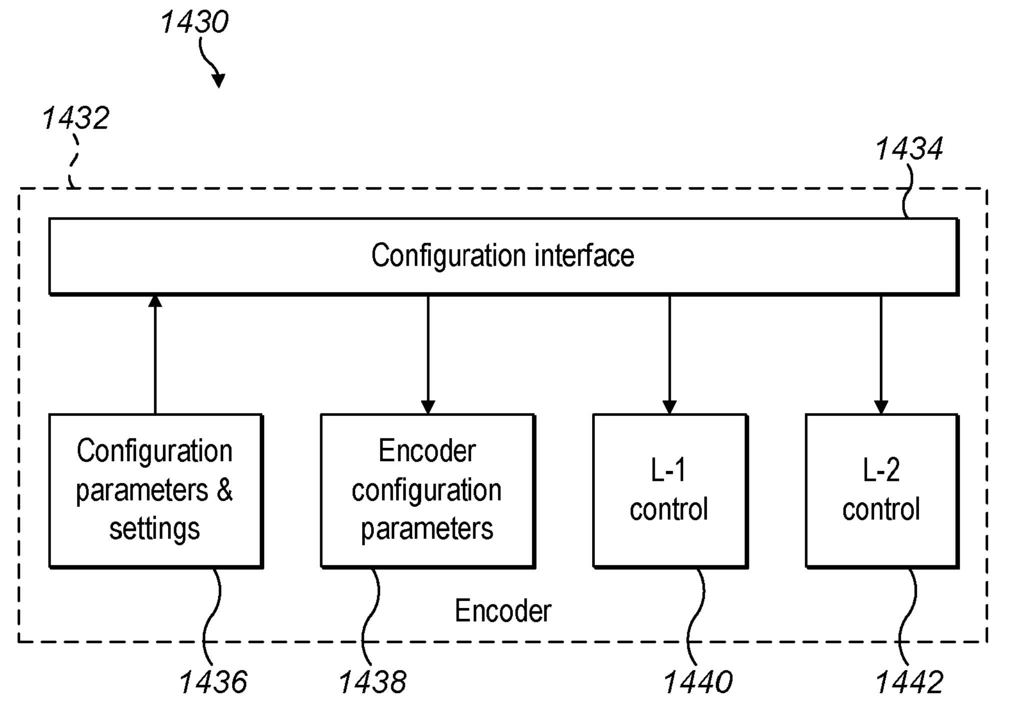


FIG. 14C

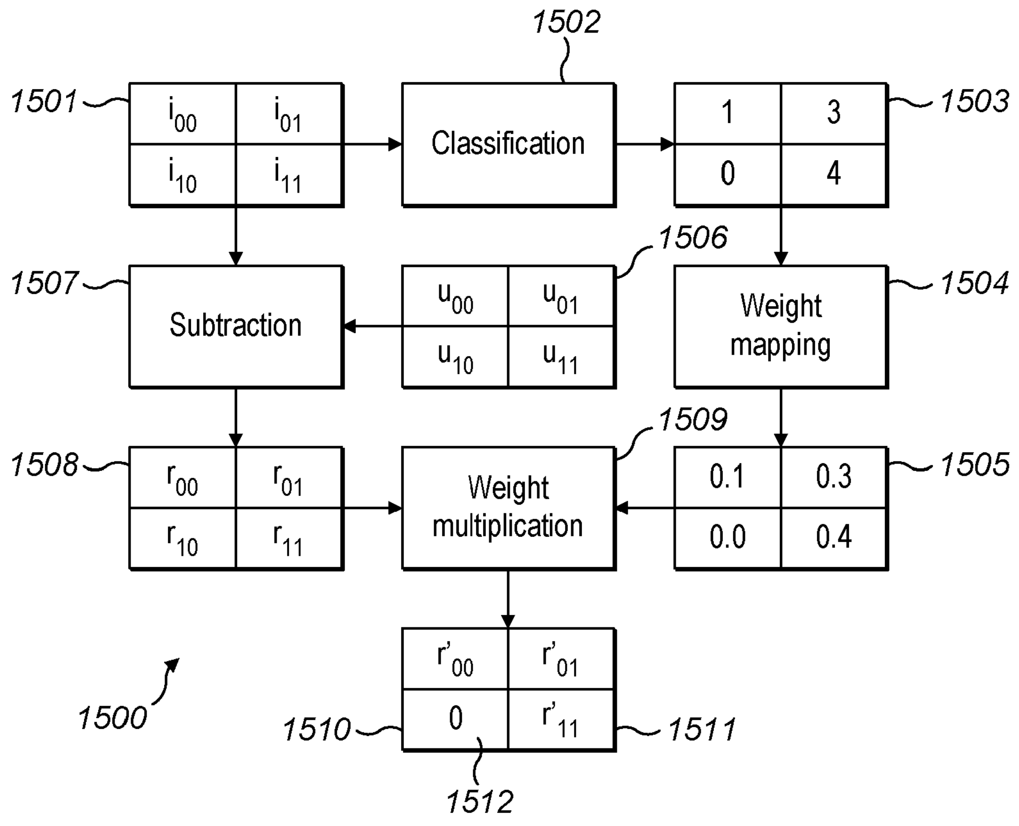


FIG. 15

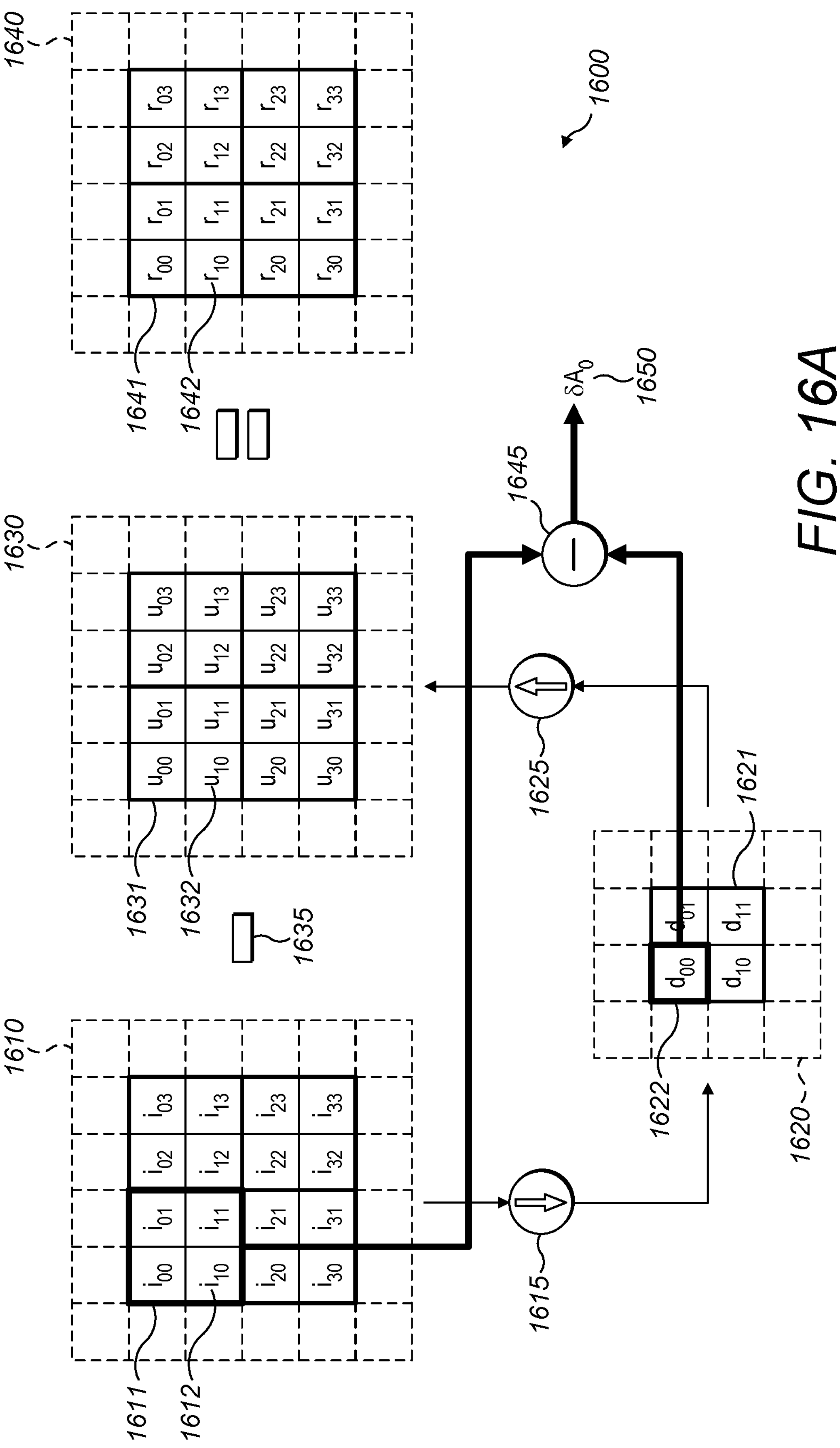

1600
1610
1611
1612
i_{00} i_{01} i_{02} i_{03}
i_{10} i_{11} i_{12} i_{13}
i_{20} i_{21} i_{22} i_{23}
i_{30} i_{31} i_{32} i_{33}
1615
1620
1621
1622
d_{00} d_{01}
d_{10} d_{11}
1625
1630
1631
1632
u_{00} u_{01} u_{02} u_{03}
u_{10} u_{11} u_{12} u_{13}
u_{20} u_{21} u_{22} u_{23}
u_{30} u_{31} u_{32} u_{33}
1635
1640
1641
1642
r_{00} r_{01} r_{02} r_{03}
r_{10} r_{11} r_{12} r_{13}
r_{20} r_{21} r_{22} r_{23}
r_{30} r_{31} r_{32} r_{33}
1645
δA_0
1650

FIG. 16A

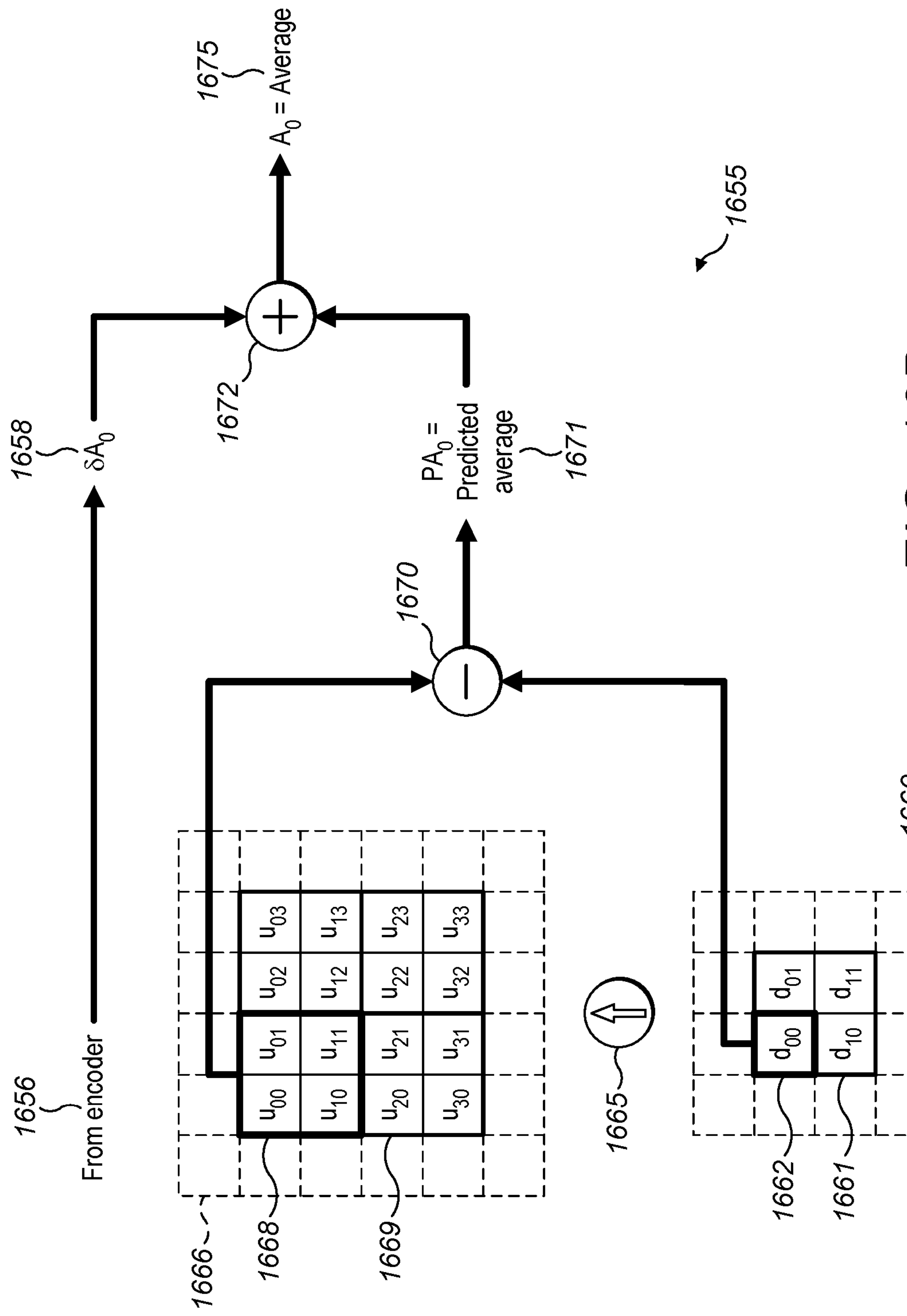

1656
From encoder
1658
δA_0
1672
1675
A_0 = Average
PA_0 = Predicted average
1671
1670
1655
1666
1668
1669
u_{00} u_{01} u_{02} u_{03}
u_{10} u_{11} u_{12} u_{13}
u_{20} u_{21} u_{22} u_{23}
u_{30} u_{31} u_{32} u_{33}
1665
1660
d_{00} d_{01}
d_{10} d_{11}
1662
1661

FIG. 16B

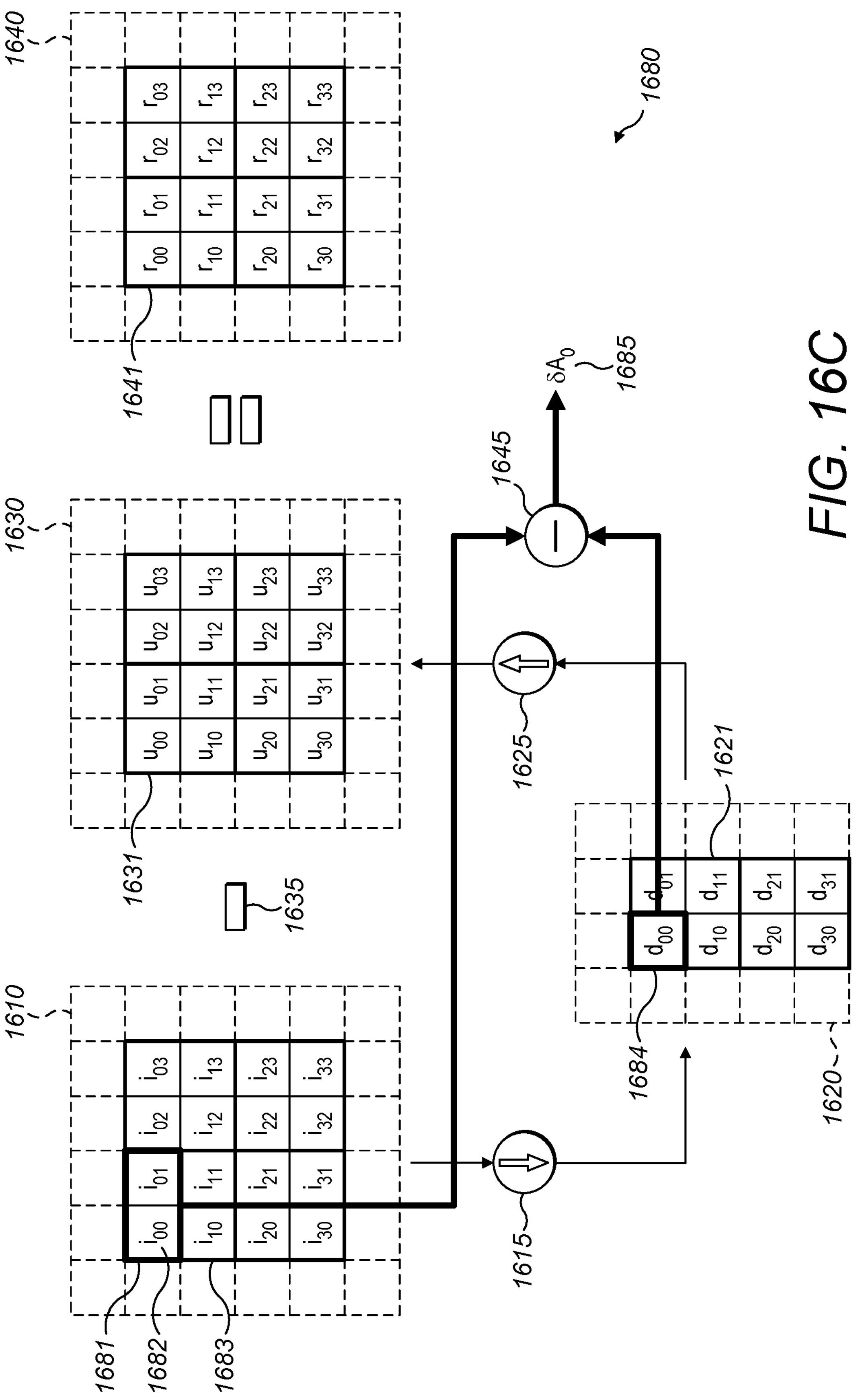

1610
1620
1621
1615
1625
1630
1631
1635
1640
1641
1645
1680
1681
1682
1683
1684
1685
δA_0
i_{00} i_{01} i_{02} i_{03}
i_{10} i_{11} i_{12} i_{13}
i_{20} i_{21} i_{22} i_{23}
i_{30} i_{31} i_{32} i_{33}
d_{00} d_{01}
d_{10} d_{11}
d_{20} d_{21}
d_{30} d_{31}
u_{00} u_{01} u_{02} u_{03}
u_{10} u_{11} u_{12} u_{13}
u_{20} u_{21} u_{22} u_{23}
u_{30} u_{31} u_{32} u_{33}
r_{00} r_{01} r_{02} r_{03}
r_{10} r_{11} r_{12} r_{13}
r_{20} r_{21} r_{22} r_{23}
r_{30} r_{31} r_{32} r_{33}

FIG. 16C

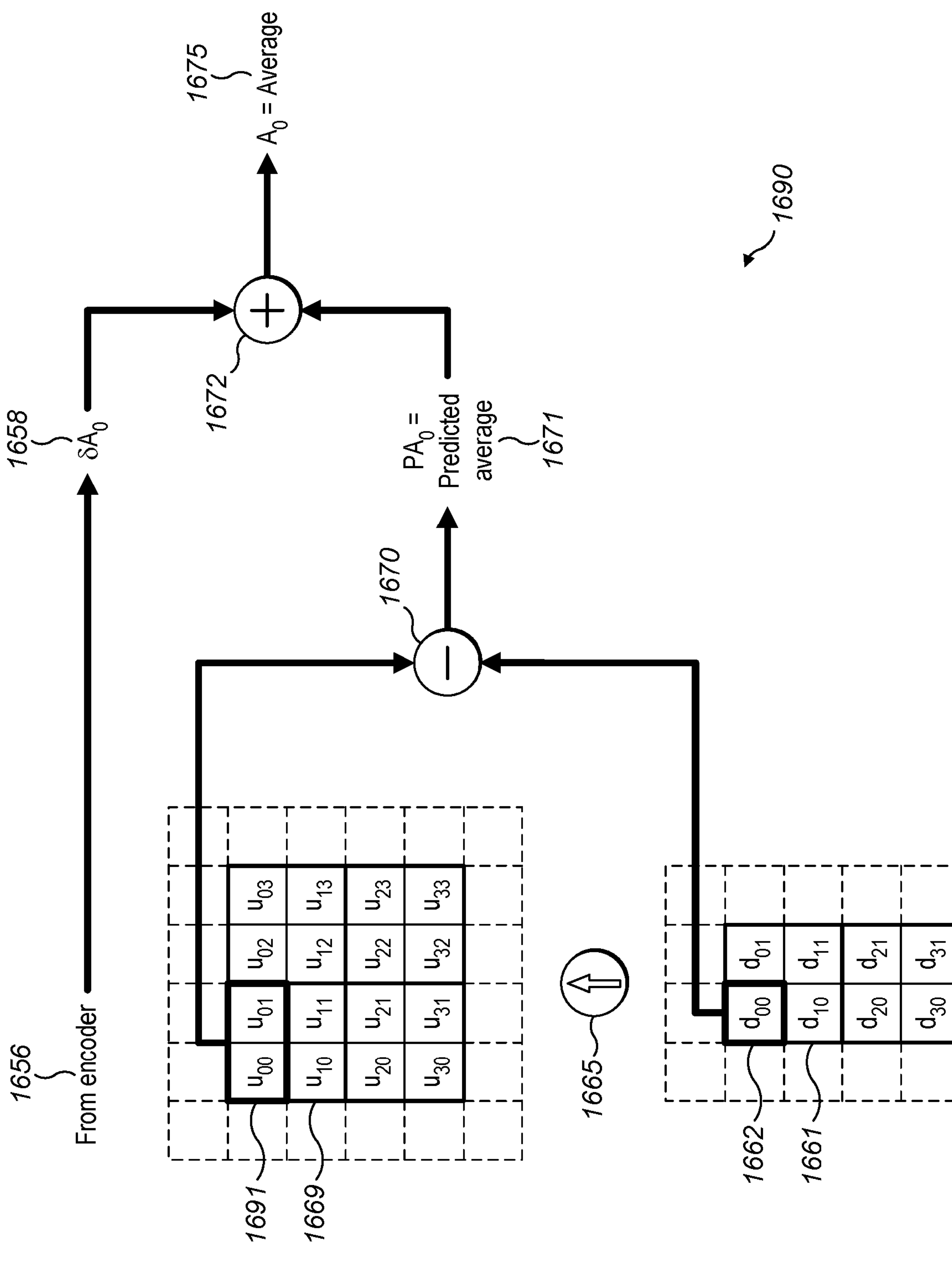

1656
From encoder
1658
δA_0
1672
1675
A_0 = Average
1690
PA_0 = Predicted average
1671
1670
1691
1669
u_{00} u_{01} u_{02} u_{03}
u_{10} u_{11} u_{12} u_{13}
u_{20} u_{21} u_{22} u_{23}
u_{30} u_{31} u_{32} u_{33}
1665
1662
1661
1660
d_{00} d_{01}
d_{10} d_{11}
d_{20} d_{21}
d_{30} d_{31}

FIG. 16D

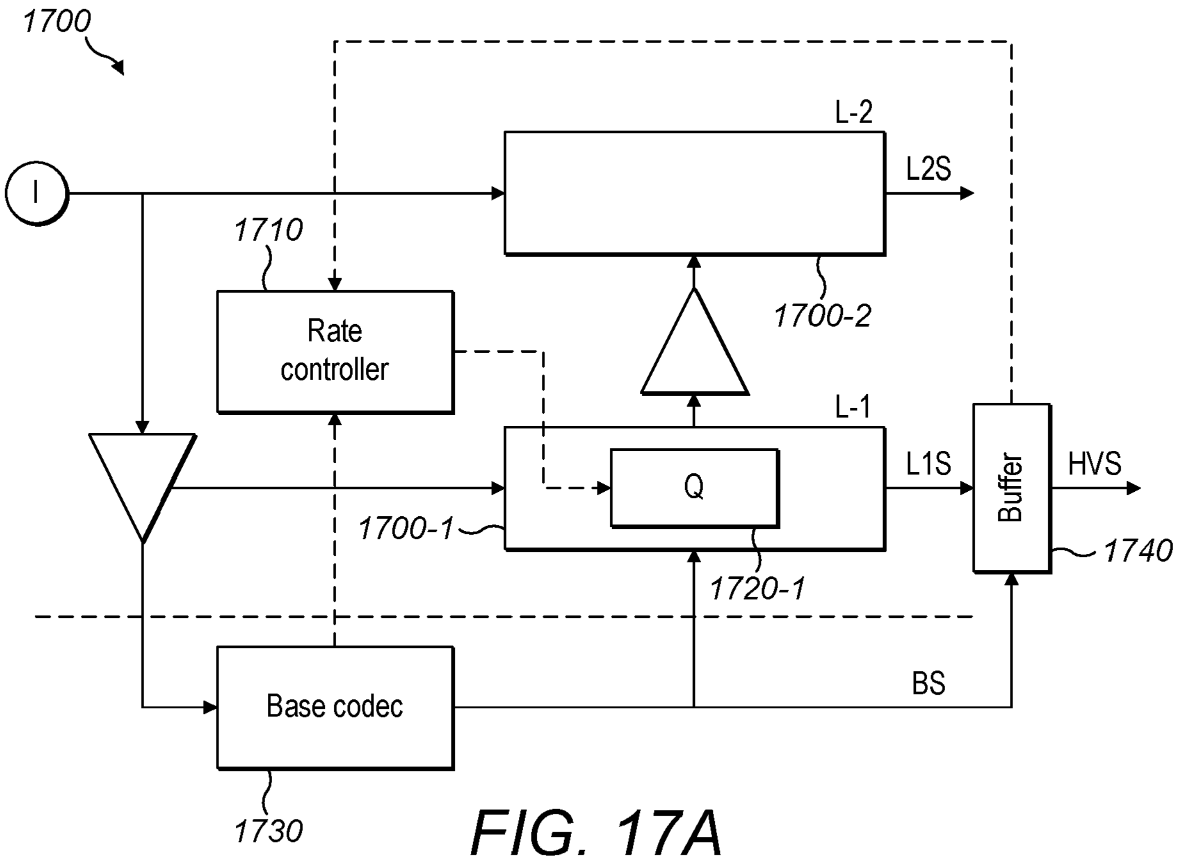


FIG. 17A

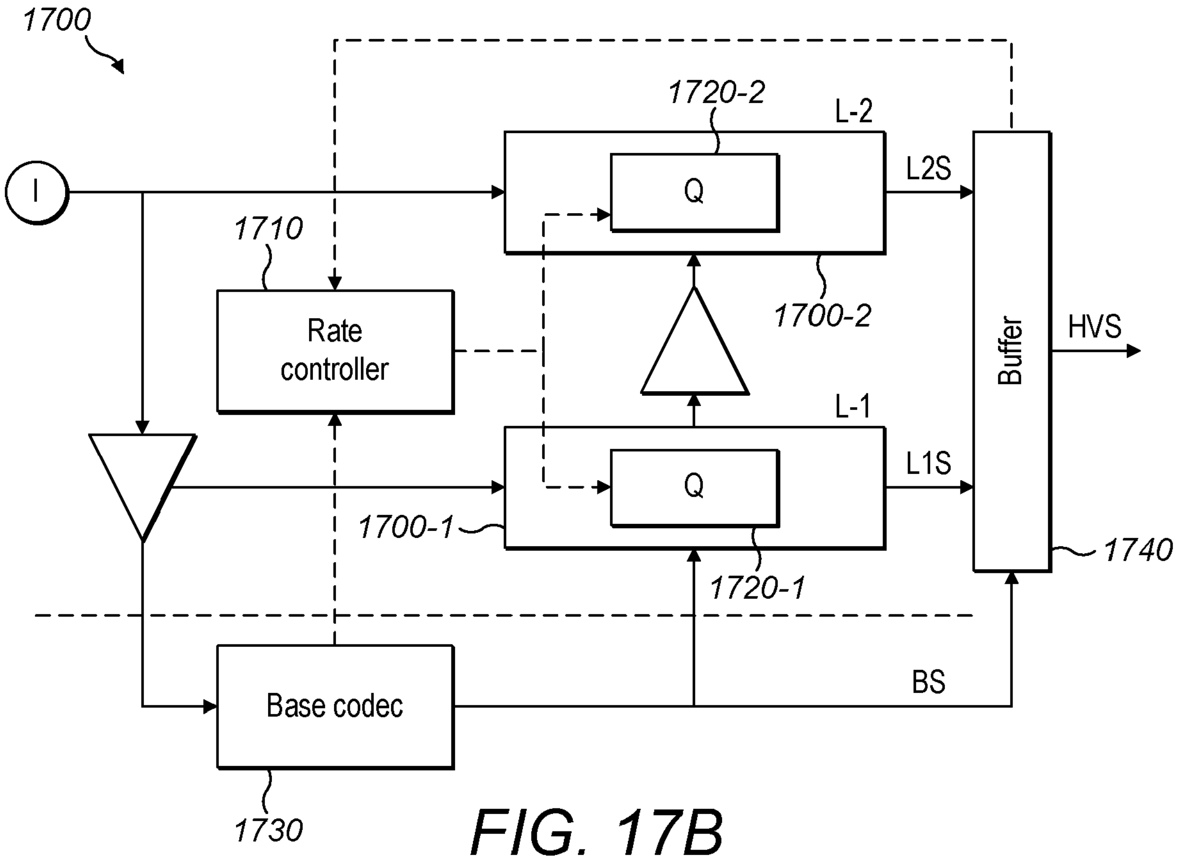


FIG. 17B

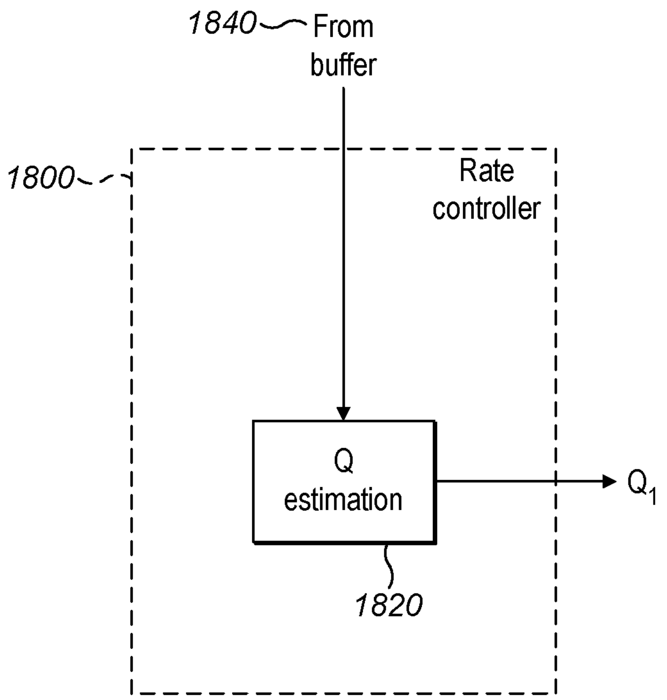


FIG. 18

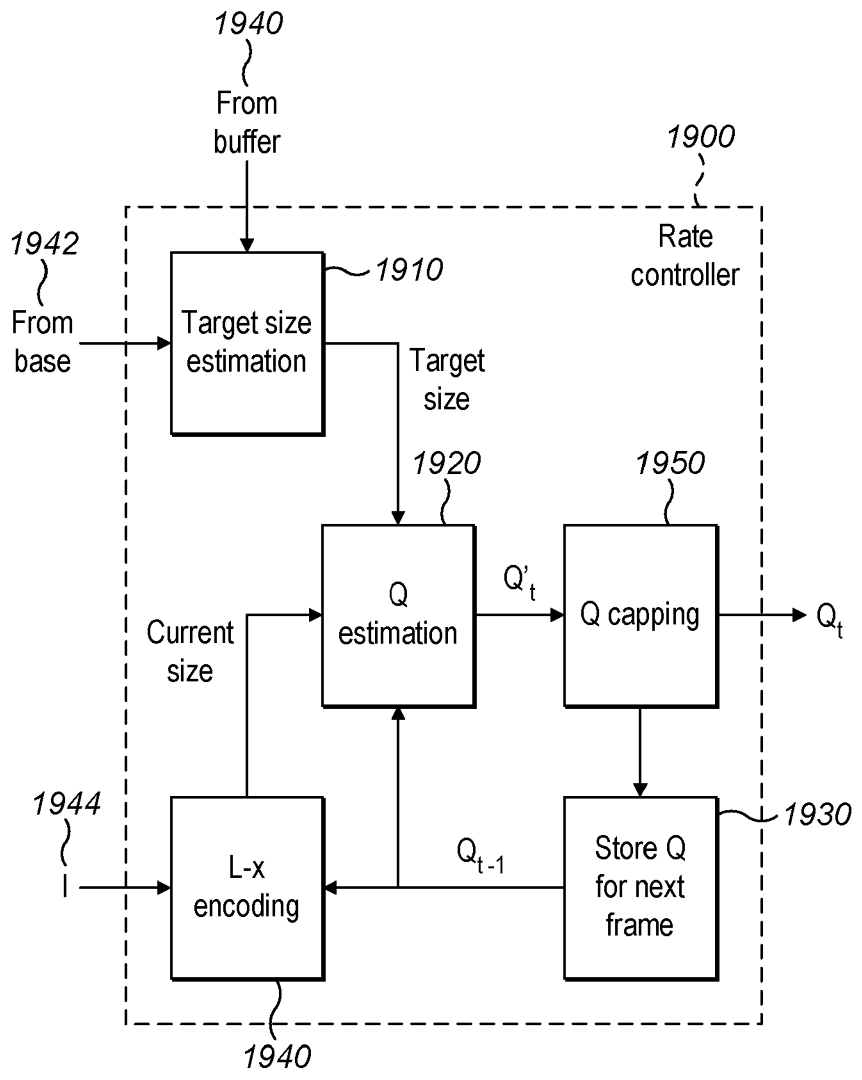

1940
From buffer
1900
Rate controller
1942
From base
Target size estimation
1910
Target size
1920
1950
Q estimation
Q'_t
Q capping
Q_t
Current size
1944
I
L-x encoding
Q_{t-1}
Store Q for next frame
1930
1940

FIG. 19

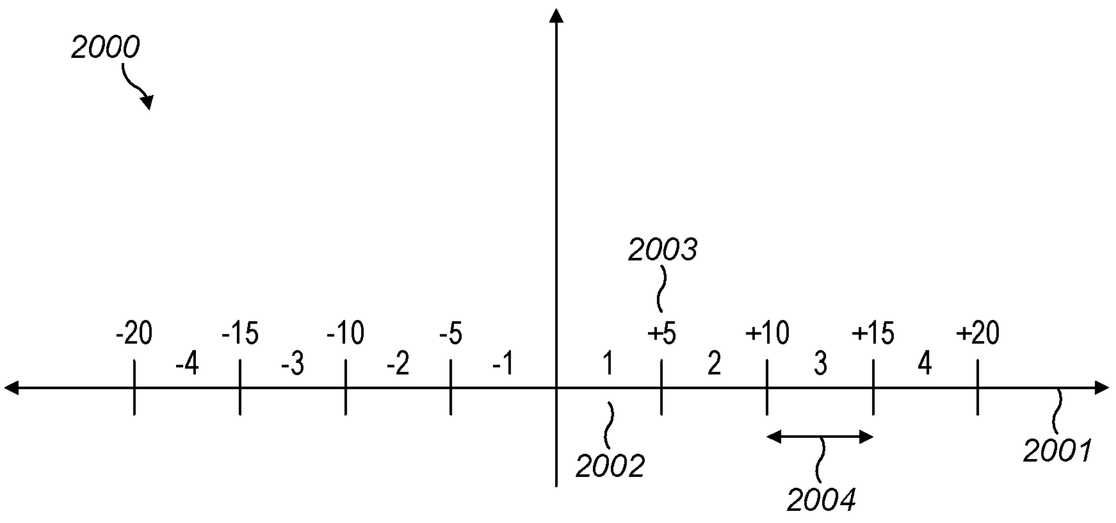


FIG. 20A

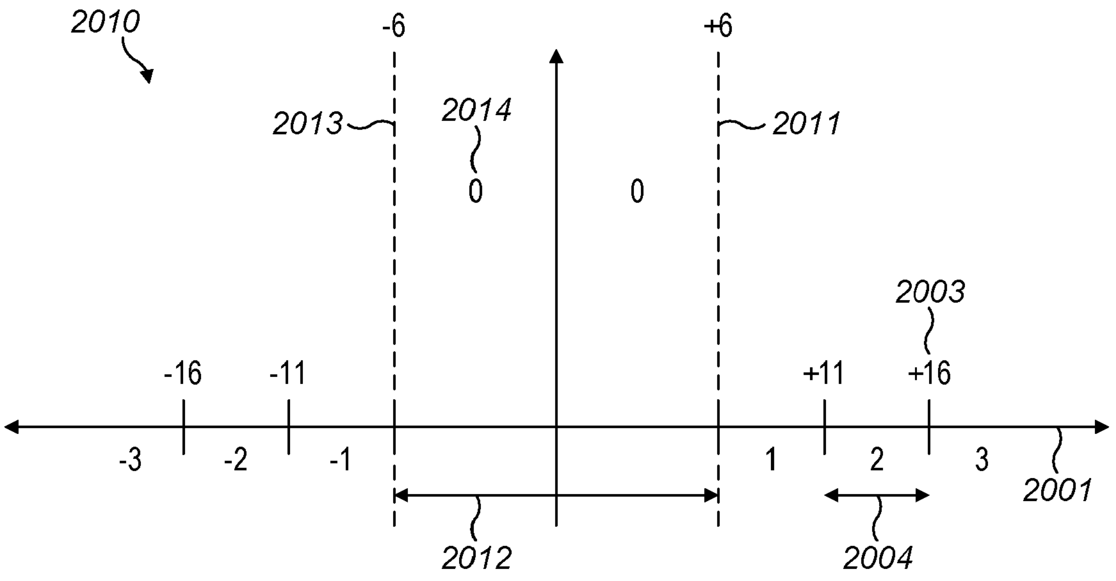


FIG. 20B

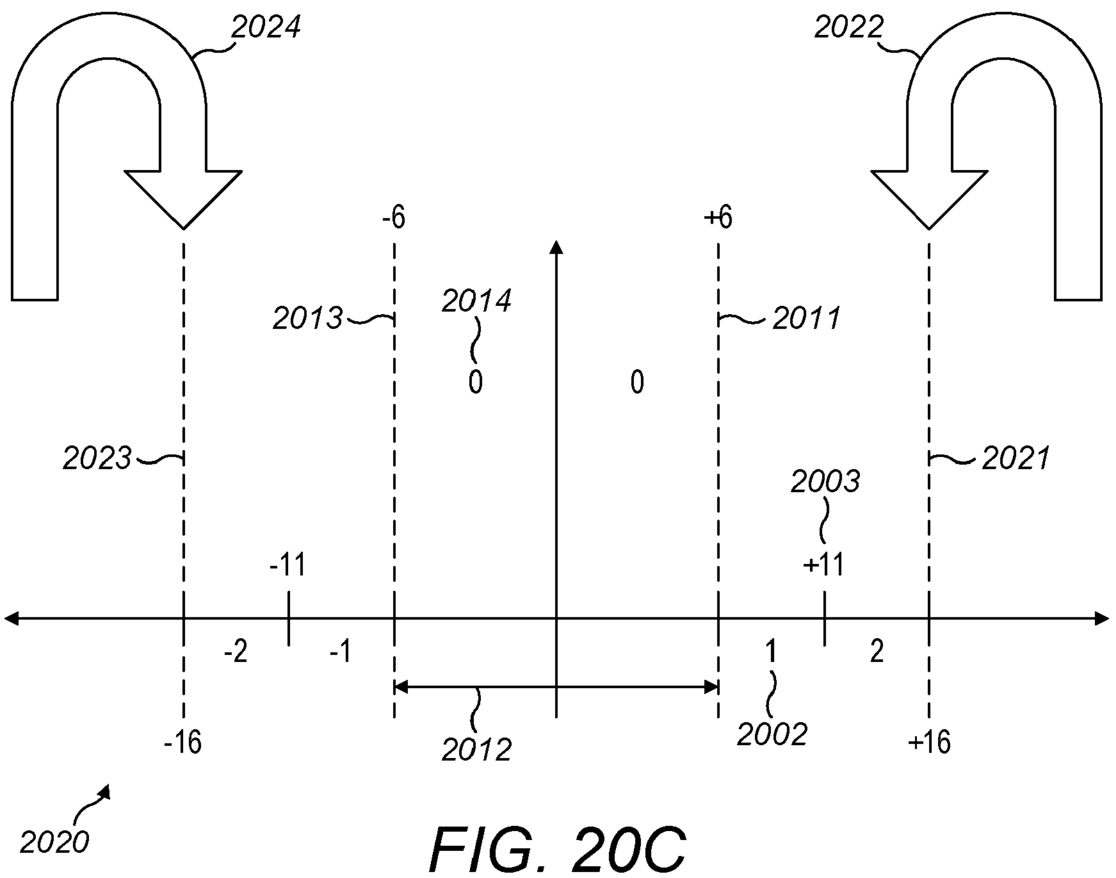


FIG. 20C

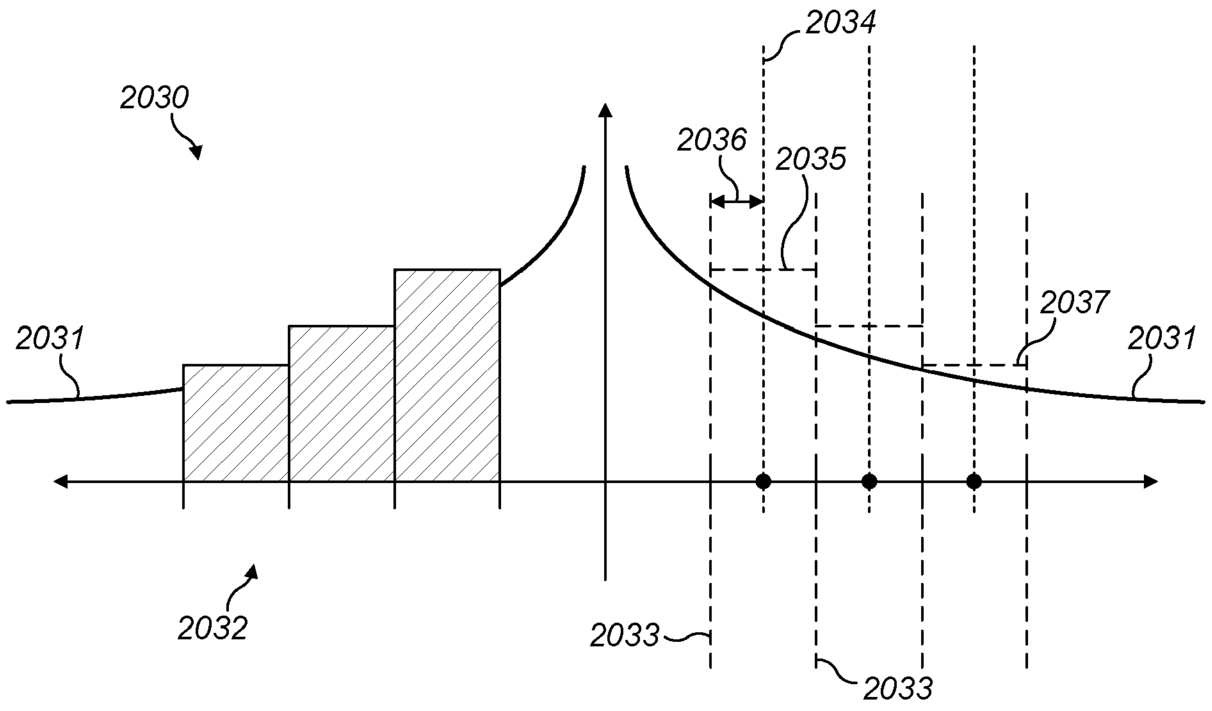


FIG. 20D

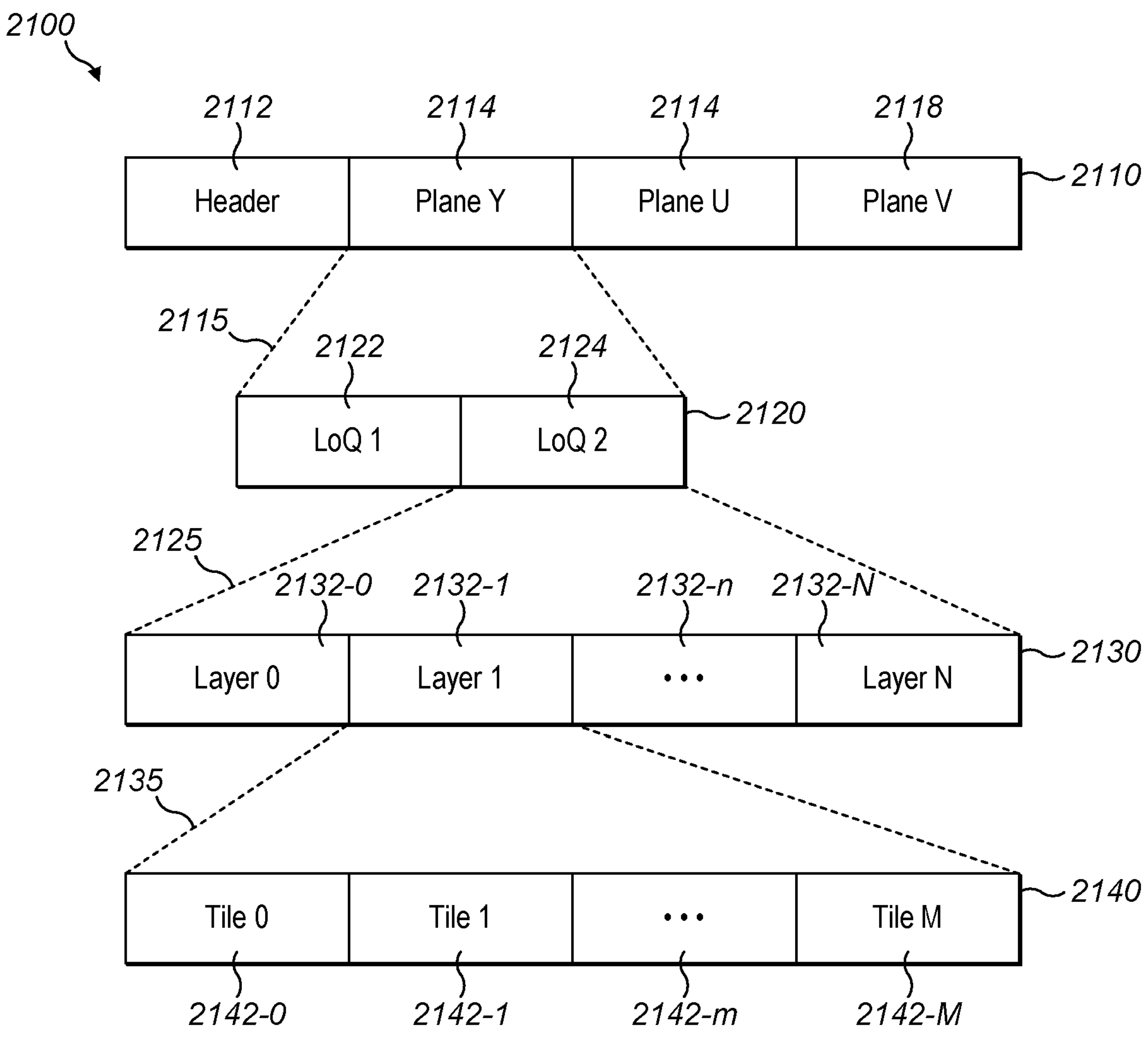

2100
2112
2114
2114
2118
Header
Plane Y
Plane U
Plane V
2110
2115
2122
2124
LoQ 1
LoQ 2
2120
2125
2132-0
2132-1
2132-n
2132-N
Layer 0
Layer 1
• • •
Layer N
2130
2135
Tile 0
Tile 1
• • •
Tile M
2140
2142-0
2142-1
2142-m
2142-M

FIG. 21A

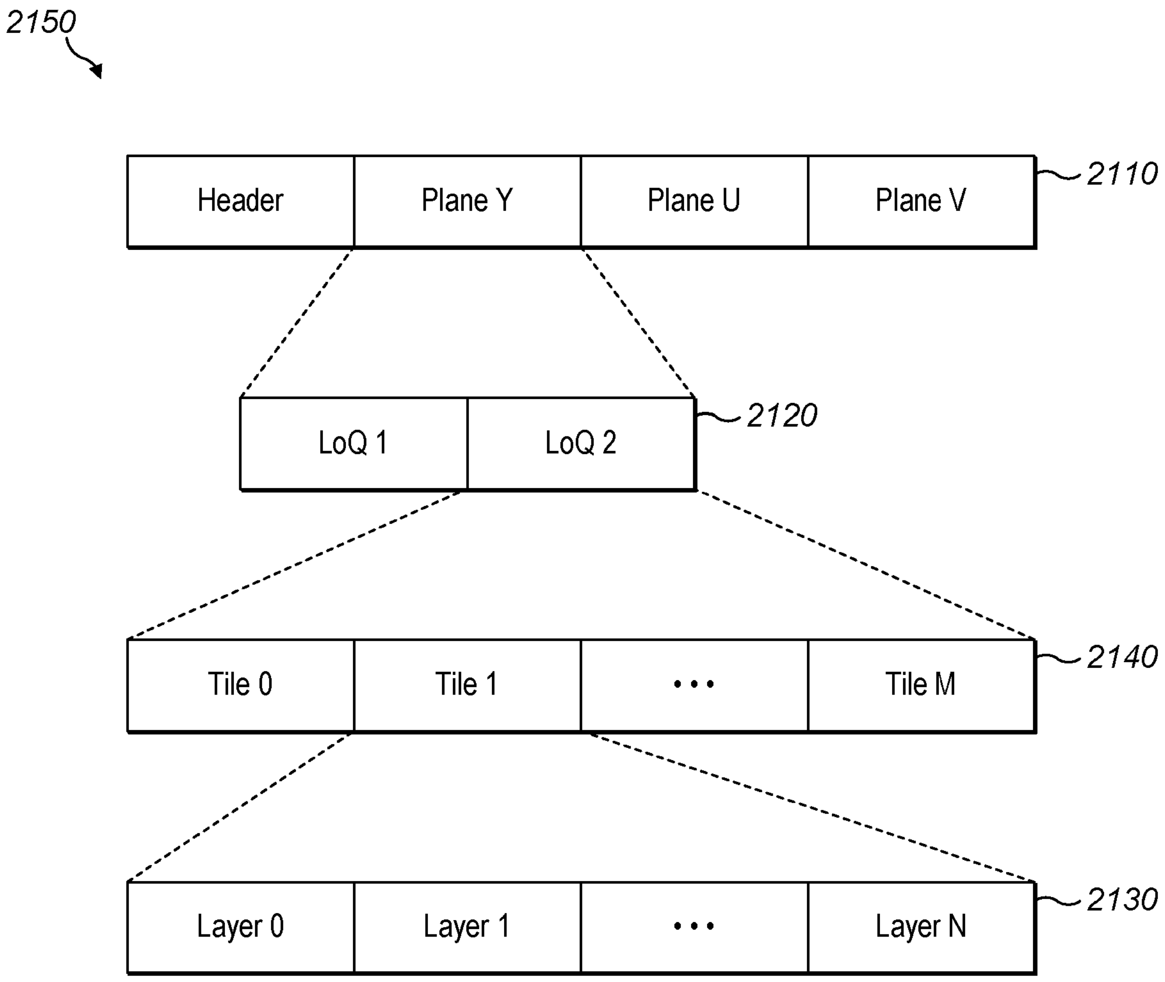

2150
Header
Plane Y
Plane U
Plane V
2110
LoQ 1
LoQ 2
2120
Tile 0
Tile 1
• • •
Tile M
2140
Layer 0
Layer 1
• • •
Layer N
2130

FIG. 21B

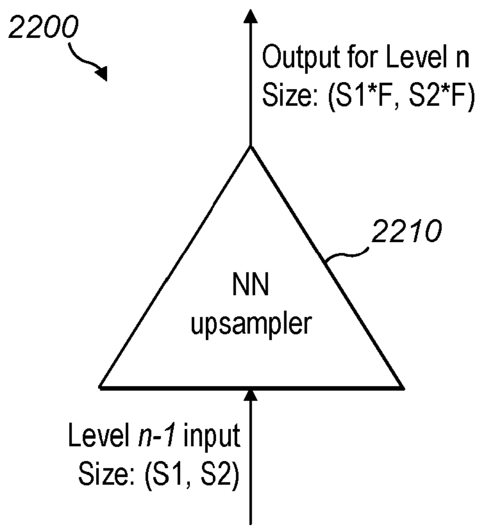


FIG. 22A

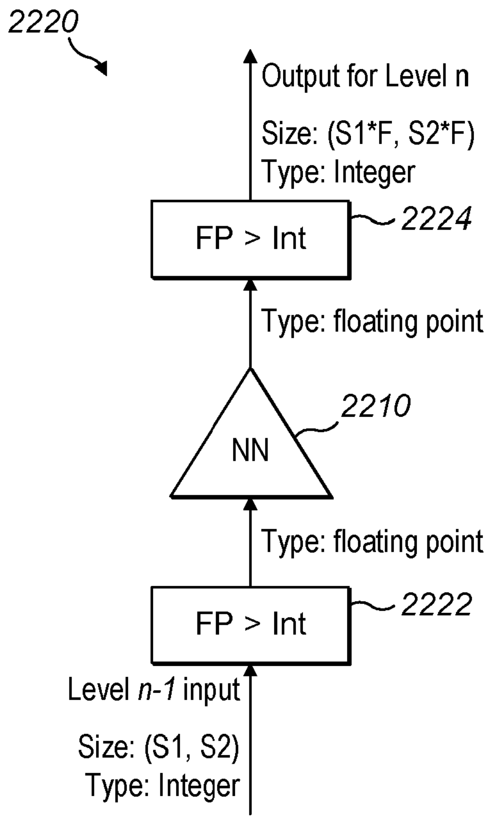


FIG. 22B

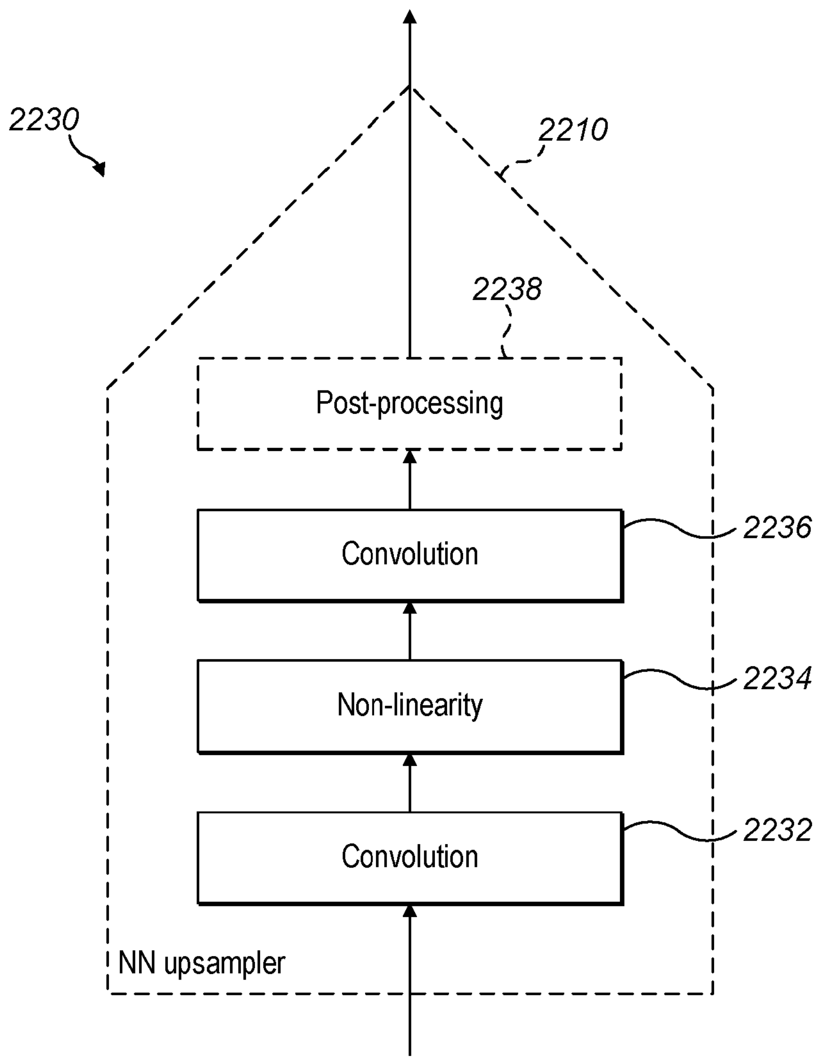

2230
2210
2238
Post-processing
Convolution
2236
Non-linearity
2234
Convolution
2232
NN upsampler

FIG. 22C

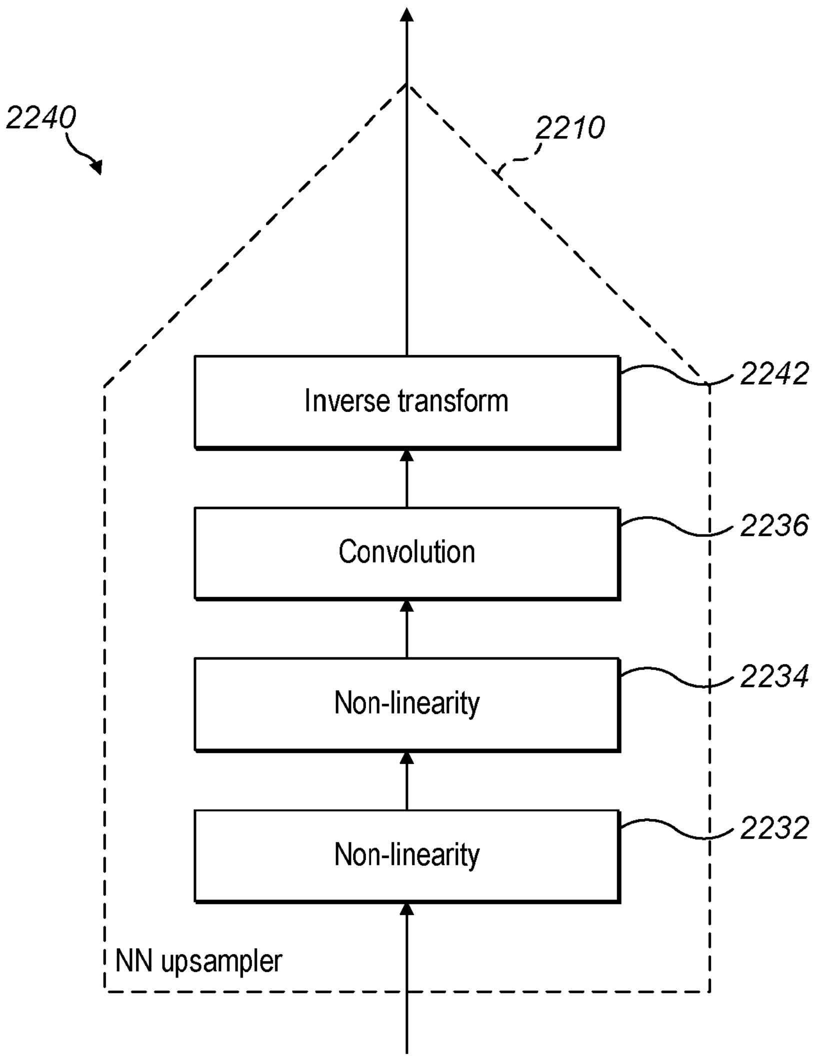

2240
2210
Inverse transform
2242
Convolution
2236
Non-linearity
2234
Non-linearity
2232
NN upsampler

FIG. 22D

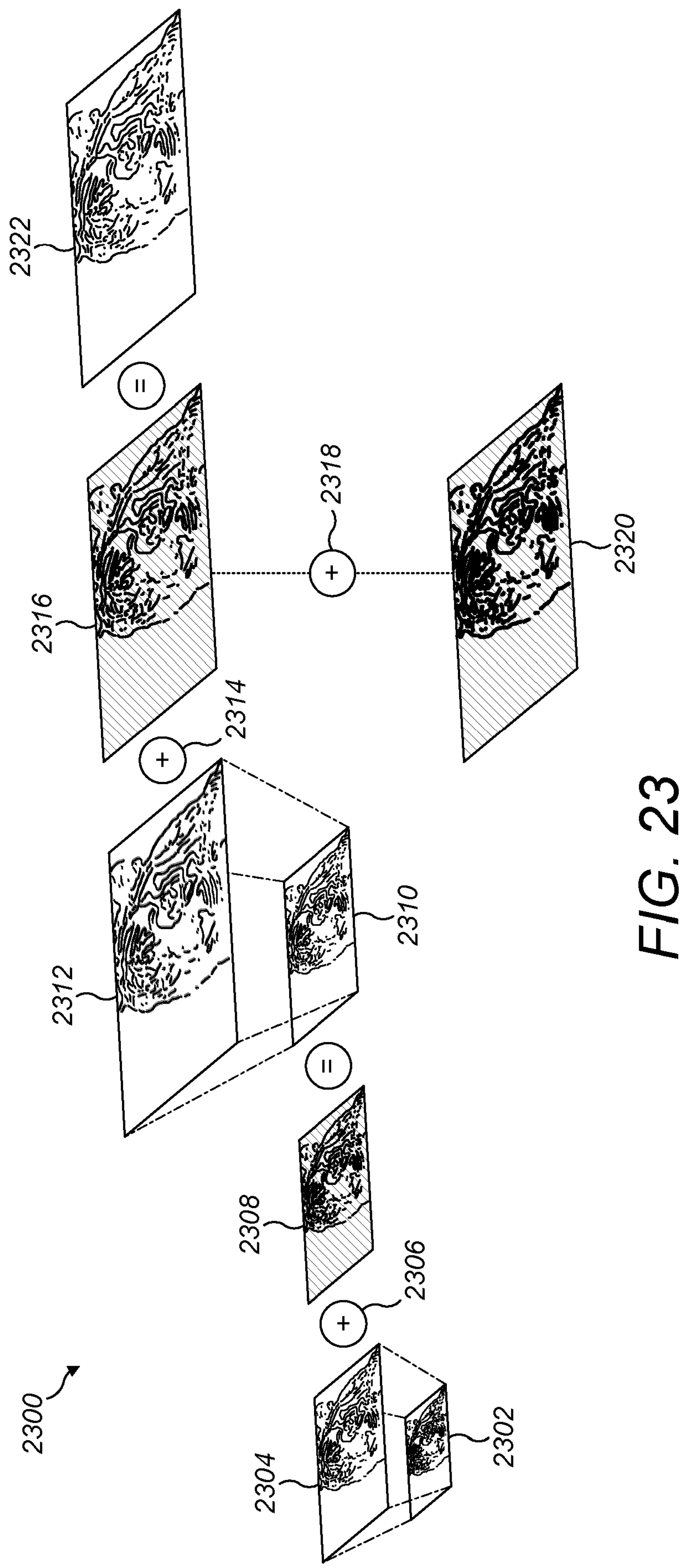


FIG. 23

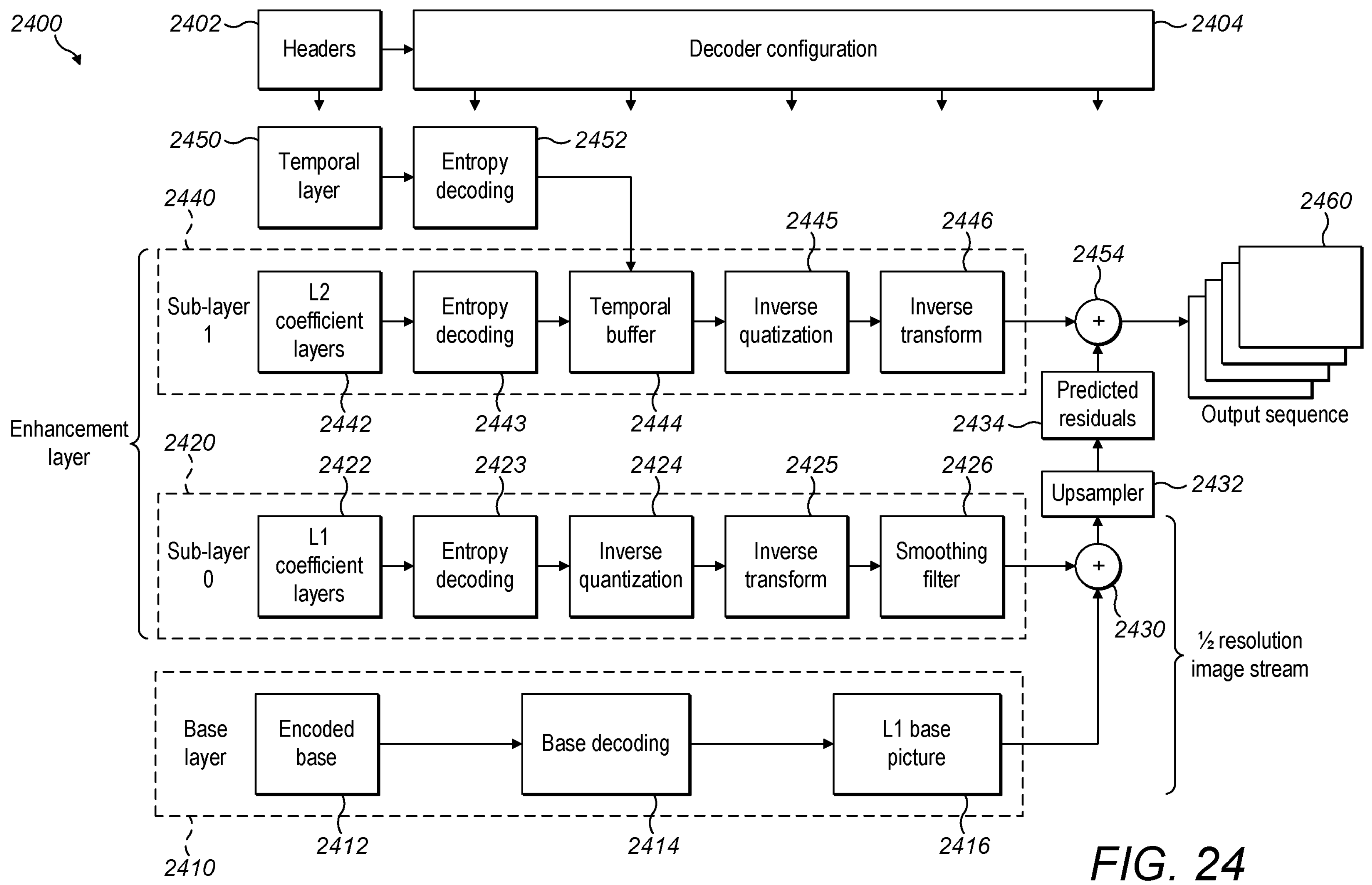

2400
2402
Headers
Decoder configuration
2404
2450
Temporal layer
Entropy decoding
2452
2440
Sub-layer 1
L2 coefficient layers
Entropy decoding
Temporal buffer
Inverse quatization
Inverse transform
2442
2443
2444
2445
2446
2454
2460
Output sequence
Predicted residuals
2434
Upsampler
2432
Enhancement layer
2420
Sub-layer 0
L1 coefficient layers
Entropy decoding
Inverse quantization
Inverse transform
Smoothing filter
2422
2423
2424
2425
2426
2430
½ resolution image stream
Base layer
Encoded base
Base decoding
L1 base picture
2410
2412
2414
2416

FIG. 24

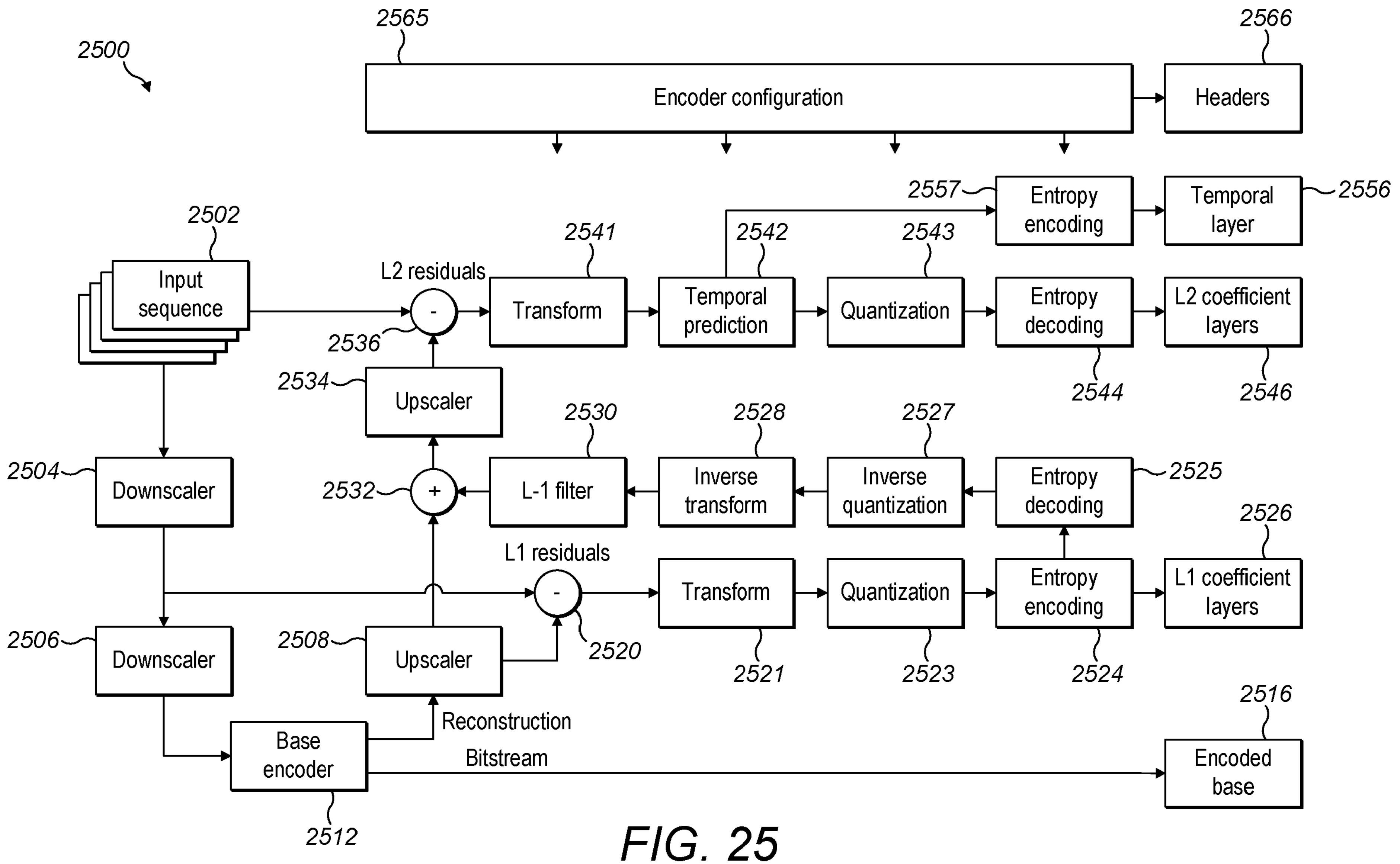

2500
2565
Encoder configuration
2566
Headers
2557
Entropy encoding
Temporal layer
2556
2502
Input sequence
L2 residuals
2541
Transform
2542
Temporal prediction
2543
Quantization
Entropy decoding
L2 coefficient layers
2536
2534
Upscaler
2544
2546
2530
2528
2527
2504
Downscaler
2532
+
L-1 filter
Inverse transform
Inverse quantization
Entropy decoding
2525
2526
L1 residuals
-
Transform
Quantization
Entropy encoding
L1 coefficient layers
2506
Downscaler
2508
Upscaler
2520
2521
2523
2524
2516
Base encoder
Reconstruction
Bitstream
Encoded base
2512

FIG. 25

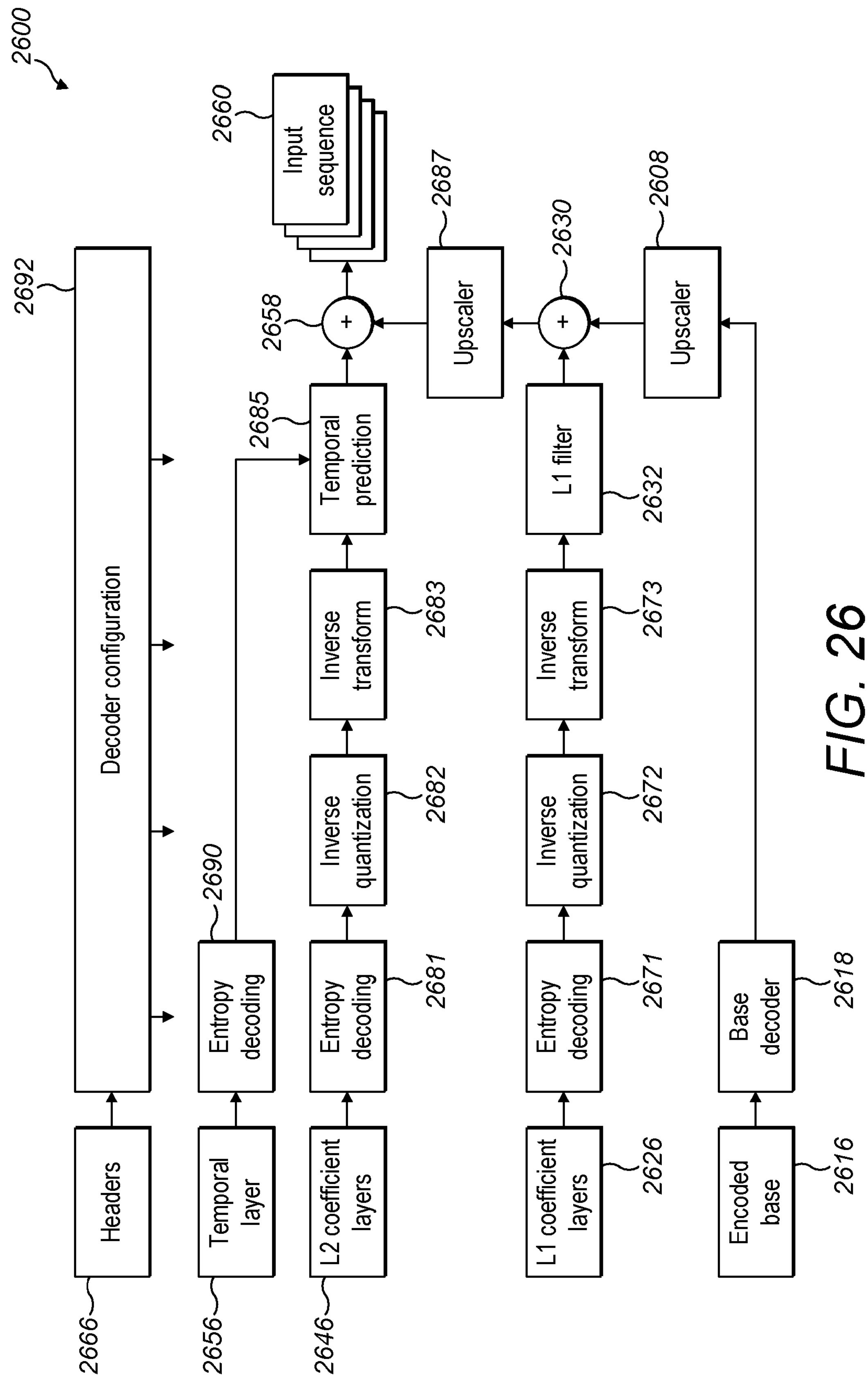
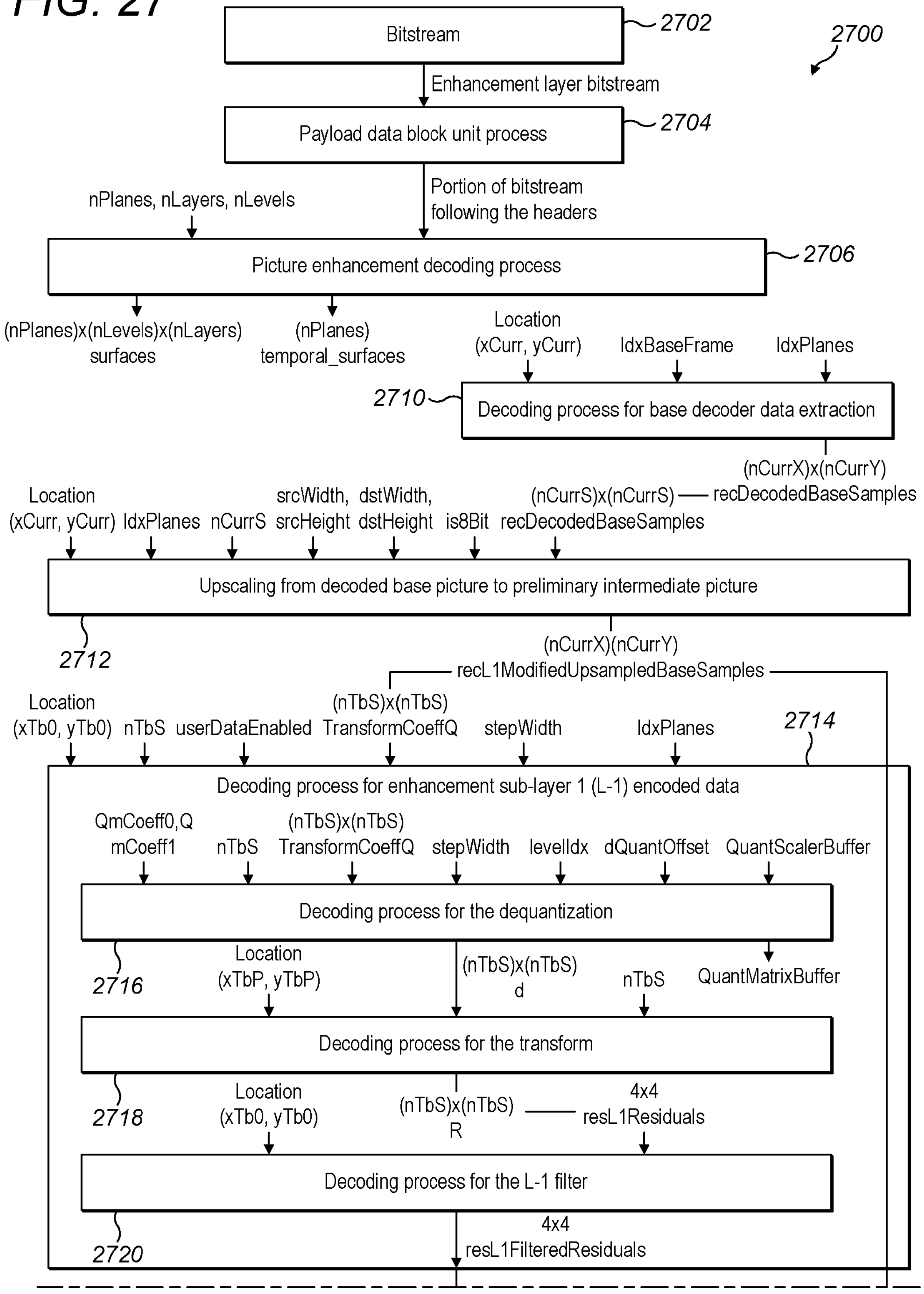

2600
Headers
2666
Temporal layer
2656
L2 coefficient layers
2646
L1 coefficient layers
2626
Encoded base
2616
Decoder configuration
2692
Entropy decoding
2690
Entropy decoding
2681
Inverse quantization
2682
Inverse transform
2683
Temporal prediction
2685
+
2658
Input sequence
2660
Upscaler
2687
Entropy decoding
2671
Inverse quantization
2672
Inverse transform
2673
L1 filter
2632
+
2630
Upscaler
2608
Base decoder
2618

FIG. 26

FIG. 27


2700
Bitstream
2702
Enhancement layer bitstream
Payload data block unit process
2704
Portion of bitstream following the headers
nPlanes, nLayers, nLevels
Picture enhancement decoding process
2706
(nPlanes)x(nLevels)x(nLayers) surfaces
(nPlanes) temporal_surfaces
Location (xCurr, yCurr)
IdxBaseFrame
IdxPlanes
2710
Decoding process for base decoder data extraction
(nCurrX)x(nCurrY) recDecodedBaseSamples
Location (xCurr, yCurr)
IdxPlanes
nCurrS
srcWidth, srcHeight
dstWidth, dstHeight
is8Bit
(nCurrS)x(nCurrS) recDecodedBaseSamples
Upscaling from decoded base picture to preliminary intermediate picture
2712
(nCurrX)(nCurrY) recL1ModifiedUpsampledBaseSamples
Location (xTb0, yTb0)
nTbS
userDataEnabled
(nTbS)x(nTbS) TransformCoeffQ
stepWidth
IdxPlanes
2714
Decoding process for enhancement sub-layer 1 (L-1) encoded data
QmCoeff0,QmCoeff1
nTbS
(nTbS)x(nTbS) TransformCoeffQ
stepWidth
levelIdx
dQuantOffset
QuantScalerBuffer
Decoding process for the dequantization
2716
Location (xTbP, yTbP)
(nTbS)x(nTbS) d
nTbS
QuantMatrixBuffer
Decoding process for the transform
2718
Location (xTb0, yTb0)
(nTbS)x(nTbS) R
4x4 resL1Residuals
Decoding process for the L-1 filter
2720
4x4 resL1FilteredResiduals

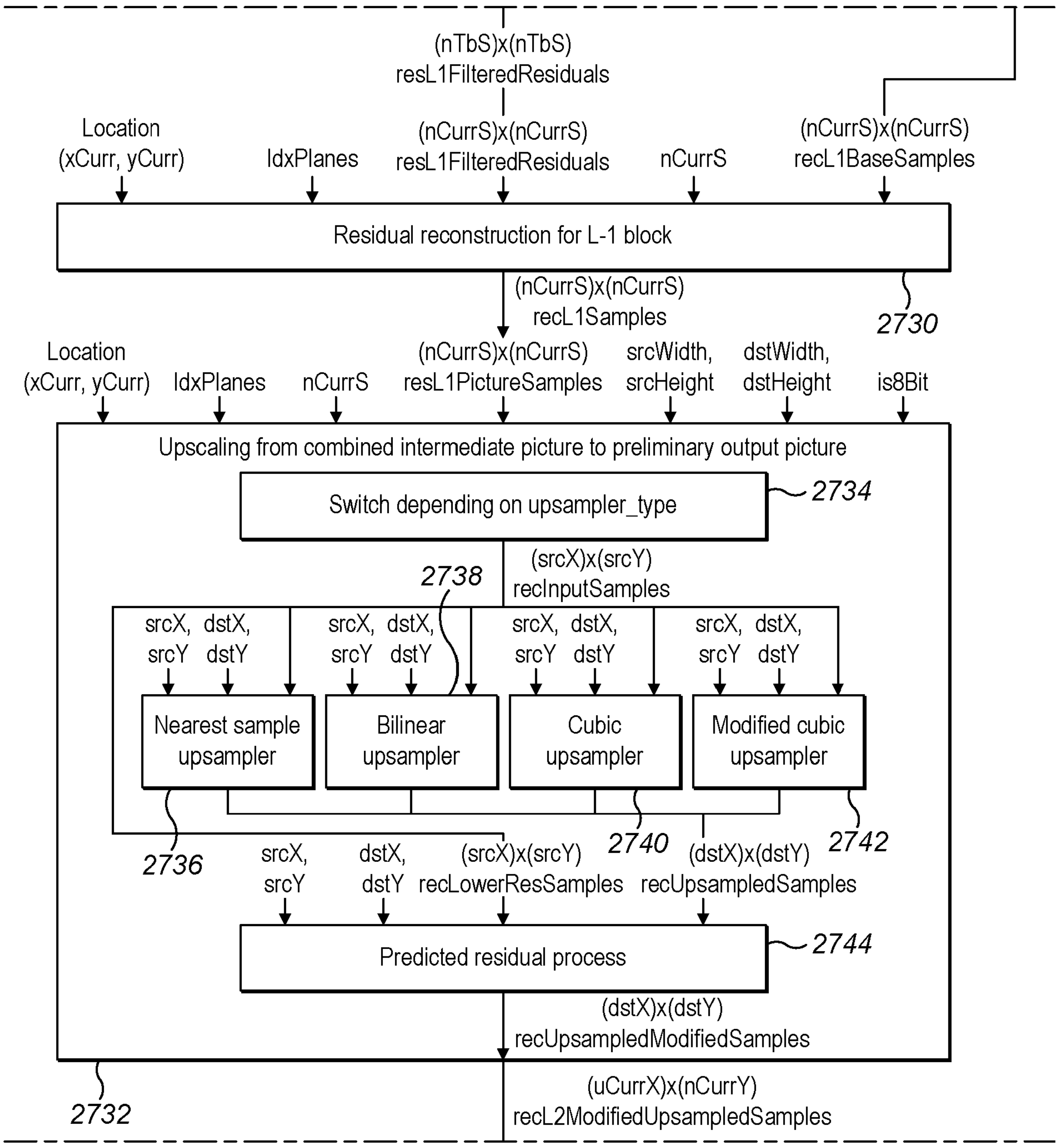

(nTbS)x(nTbS)
resL1FilteredResiduals
Location
(xCurr, yCurr)
IdxPlanes
(nCurrS)x(nCurrS)
resL1FilteredResiduals
nCurrS
(nCurrS)x(nCurrS)
recL1BaseSamples
Residual reconstruction for L-1 block
(nCurrS)x(nCurrS)
recL1Samples
2730
Location
(xCurr, yCurr)
IdxPlanes
nCurrS
(nCurrS)x(nCurrS)
resL1PictureSamples
srcWidth,
srcHeight
dstWidth,
dstHeight
is8Bit
Upscaling from combined intermediate picture to preliminary output picture
Switch depending on upsampler_type
2734
(srcX)x(srcY)
recInputSamples
2738
srcX, srcY
dstX, dstY
Nearest sample upsampler
Bilinear upsampler
Cubic upsampler
Modified cubic upsampler
2736
2740
2742
srcX,
srcY
dstX,
dstY
(srcX)x(srcY)
recLowerResSamples
(dstX)x(dstY)
recUpsampledSamples
Predicted residual process
2744
(dstX)x(dstY)
recUpsampledModifiedSamples
2732
(uCurrX)x(nCurrY)
recL2ModifiedUpsampledSamples

FIG. 27 Cont'd

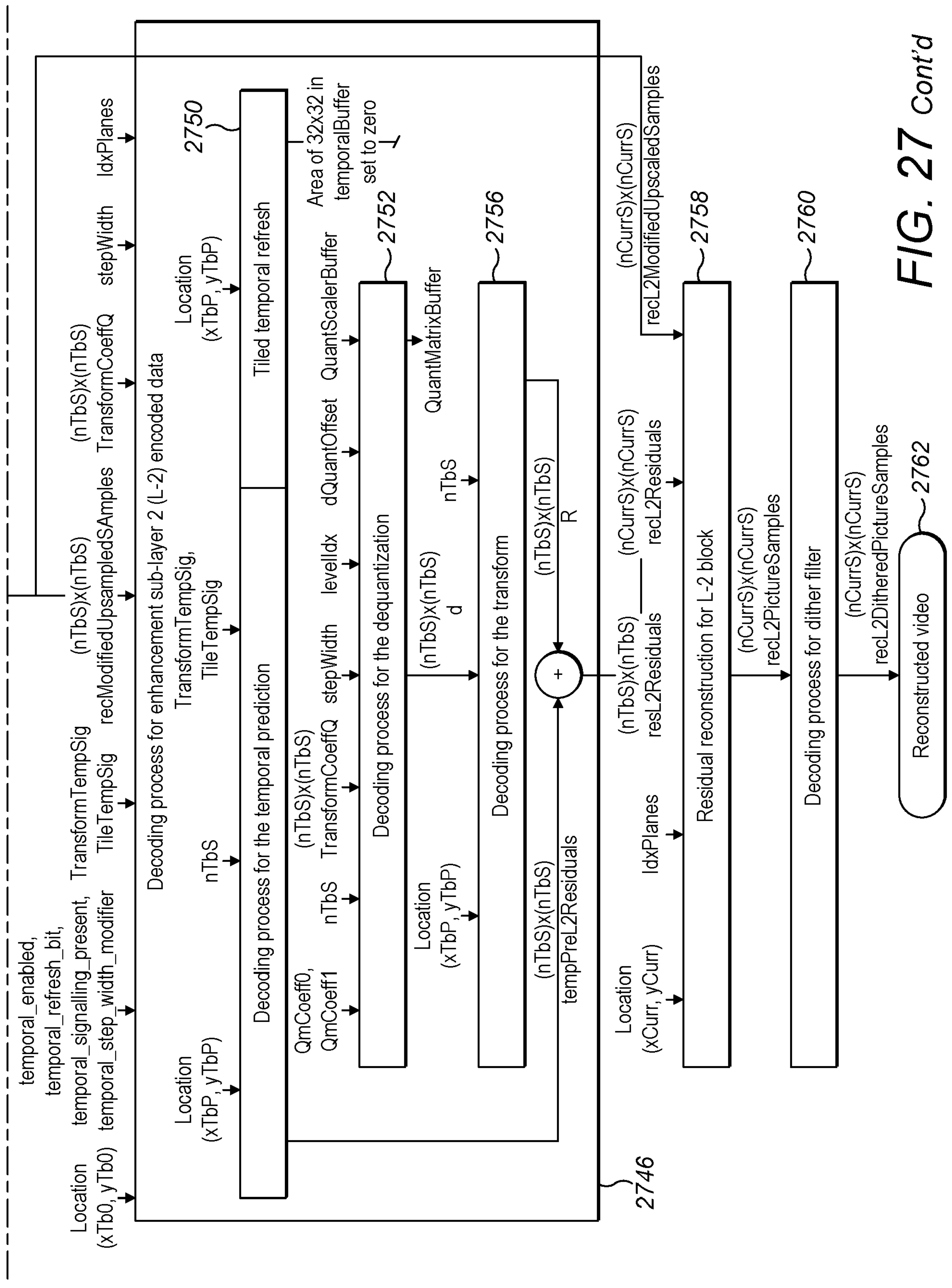

Location (xTb0, yTb0)
temporal_enabled, temporal_refresh_bit, temporal_signalling_present, temporal_step_width_modifier
TransformTempSig TileTempSig
(nTbS)x(nTbS) recModifiedUpsampledSAmples
(nTbS)x(nTbS) TransformCoeffQ
stepWidth
IdxPlanes
Decoding process for enhancement sub-layer 2 (L-2) encoded data
Location (xTbP, yTbP)
nTbS
TransformTempSig, TileTempSig
Location (xTbP, yTbP)
2750
Decoding process for the temporal prediction
Tiled temporal refresh
Area of 32x32 in temporalBuffer set to zero
QmCoeff0, QmCoeff1
nTbS
(nTbS)x(nTbS) TransformCoeffQ
stepWidth
levelIdx
dQuantOffset
QuantScalerBuffer
2752
Decoding process for the dequantization
QuantMatrixBuffer
Location (xTbP, yTbP)
(nTbS)x(nTbS) d
nTbS
2756
Decoding process for the transform
(nTbS)x(nTbS) tempPreL2Residuals
+
(nTbS)x(nTbS) R
2746
Location (xCurr, yCurr)
IdxPlanes
(nTbS)x(nTbS) resL2Residuals
(nCurrS)x(nCurrS) recL2Residuals
(nCurrS)x(nCurrS) recL2ModifiedUpscaledSamples
2758
Residual reconstruction for L-2 block
(nCurrS)x(nCurrS) recL2PictureSamples
2760
Decoding process for dither filter
(nCurrS)x(nCurrS) recL2DitheredPictureSamples
2762
Reconstructed video

FIG. 27 Cont'd

LOW COMPLEXITY ENHANCEMENT VIDEO CODING

TECHNICAL FIELD

The present invention relates to a video coding technology. In particular, the present invention relates to methods and systems for encoding and decoding video data. In certain examples, the methods and systems may be used to generate a compressed representation for streaming and/or storage.

BACKGROUND

Typical comparative video codecs operate using a single-layer, block-based approach, whereby an original signal is processed using a number of coding tools in order to produce an encoded signal which can then be reconstructed by a corresponding decoding process. For simplicity, coding and decoding algorithms or processes are often referred to as "codecs"; the term "codec" being used to cover one or more of encoding and decoding processes that are designed according to a common framework. Such typical codecs include, but are not limited, to MPEG-2, AVC/H.264, HEVC/H.265, VP8, VP9, AV1. There are also other codecs that are currently under development by international standards organizations, such as MPEG/ISO/ITU as well as industry consortia such as Alliance for Open Media (AoM).

In recent years, adaptations to the single-layer, block-based approach have been suggested. For example, there exists a class of codecs that operate using a multi-layer, block-based approach. These codecs are often known as "scalable" codecs within the video coding industry. They typically replicate operations performed by a single-layer, block-based approach over a number of layers, where a set of layers are obtained by down-sampling an original signal. In certain cases, efficiencies in the single-layer, block-based approach may be achieved by re-using information from a lower layer to encode (and decode) an upper layer. These scalable codecs are meant to provide scalability features to operators, in the sense that they need to guarantee that the quality of the scaled-down decoded signal (e.g., the lower resolution signal) satisfies the quality requirements for existing services, as well as ensuring that the quality of the non-scaled decoded signal (e.g., higher resolution signal) is comparable with that produced by a corresponding single-layer codec.

An example of a "scalable" codec is Scalable Video Coding - SVC (see for example "The Scalable Video Coding Extension of the H.264/AVC Standard", H. Schwarz and M. Wien, IEEE Signal Processing Magazine, March 2008, which is incorporated herein by reference). SVC is the scalable version of the Advanced Video Coding standard - AVC (AVC also being known as H.264). In SVC, each scalable layer is processed using the same AVC-based single-layer process, and upper layers receive information from lower layers (e.g., interlayer predictions including residual information and motion information) which is used in the encoding of the upper layer to reduce encoded information at the upper layer. Conversely, in order to decode, an SVC decoder needs to receive various overhead information as well as decode the lower layer in order to be able to decode the upper layer.

Another example of a scalable codec is the Scalable Extension of the High Efficiency Video Coding Standard (HEVC) - SHVC (see for example "Overview of SHVC: Scalable Extensions of the High Efficiency Video Coding Standard", J. Boyce, Y. Ye, J. Chen and A. Ramasubramonian, IEEE Trans. On Circuits and Systems for Video Technology, Vol. 26, No. 1, Jan 2016, which is incorporated by reference herein). Similar to SVC, SHVC also uses the same HEVC-based process for each scalable layer, but it allows for the lower layer to use either AVC or HEVC. In SHVC, the upper layer also receives information from the lower layer (e.g., inter layer processing including motion information and/or the up-sampled lower layer as an additional reference picture for the upper layer coding) in the encoding of the upper layer to reduce encoded information at the upper layer. Again, similarly to SVC, an SHVC decoder needs to receive various overhead information as well as decode the lower layer in order to be able to decode the upper layer.

Both SVC and SHVC may be used to encode data in multiple streams at different levels of quality. For example, SVC and SHVC may be used to encode e.g. a SD (standard definition) and an HD (high definition) stream or an HD and a UHD (ultra-high-definition) stream. The base stream (at the lowest level of quality) is typically encoded so that the quality of the base stream is the same as if the base stream were encoded as a single stream, separately from any higher-level streams. Both SVC and SHVC may be thought of primarily as a set of parallel copies of a common encoder and decoder structure, where the outputs of these parallel copies are respectively multiplexed and demultiplexed.

In more detail, within an example SVC encoding, a UHD stream (e.g. a series of images) may be down-sampled to generate an HD stream. The UHD stream and the HD stream are then each encoded separately using an AVC encoder. Although this example describes a two-layer encoder (for encoding two streams: a UHD stream and an HD stream), an SVC encoder may have *n* layers (where *n* > 2), where each layer operates as an independent AVC encoder.

As per standard AVC encoding, an AVC encoder of each SVC layer encodes each pixel block of image data using either inter-frame prediction (in which a different frame is used to estimate values for a current frame) or intra-frame prediction (in which other blocks within a frame are used to estimate values for a given block of that same frame). These blocks of pixels are typically referred to as "macroblocks". Inter-frame prediction involves performing motion compensation, which involves determining the motion between a pixel block of a previous frame and the corresponding pixel block for the current frame. Both inter- and intra-frame prediction within a layer involves calculating so-called "residuals". These "residuals" are the difference between a pixel block of the data stream of a given layer and a corresponding pixel block within the same layer determined using either inter-frame prediction or intra-frame prediction. As such, these "residuals" are the difference between a current pixel block in the layer and either: 1) a prediction of the current pixel block based on one or more pixel blocks that are not the current pixel block within the frame (e.g. typically neighbouring pixel blocks within the same layer); or 2) a prediction of the current pixel block within the layer based on information from other frames within the layer (e.g. using motion vectors).

In SVC, despite the implementation as a set of parallel AVC encoders, some efficiencies may be gained by re-using information obtained for a lower quality stream (such as an HD stream) for the encoding of a higher quality stream (such as an UHD stream). This re-using of information involves what is referred to as "inter-layer signalling". It should be noted that this is to be distinguished from the "inter-frame" and "intra-frame" prediction, the latter being "within layer" coding approaches. For example, without inter-layer signalling, the total bandwidth, BW_{Tot}, for an SVC stream may be expressed as $BW_{Tot} = BW_{HD} + BW_{UHD}$, where BW_{HD} is the bandwidth associated with sending the encoded HD stream separately and BW_{UHD} is the bandwidth associated sending the encoded UHD stream separately (assuming no sharing of information between the different streams). However, by using inter-layer signalling, the bandwidth for the UHD stream BW_{UHD} can be reduced compared to that if the UHD stream is sent separately from the HD stream. Typically, by using inter-layer signalling, the total bandwidth can be reduced so that $BW_{Tot} \cong 1.4\, BW_{UHD}$.

In SVC, inter-layer signalling may comprise one of three types of information: interlayer intra-prediction (in which an up-sampled pixel block from the HD stream is used in intra-prediction for the UHD stream), interlayer residual prediction (which involves calculating a residual between the residuals calculated for the HD stream after up-sampling and the residuals calculated for the UHD stream for a given pixel block), and interlayer motion compensation (which involves using motion compensation parameters determined for the HD stream to perform motion compensation for the UHD stream).

Similar to SVC being a scalable extension of AVC, SHVC is a scalable extension of HEVC. AVC involves dividing a frame into macroblocks (usually 16x16 pixels in size). A given macroblock can be predicted either from other macroblocks within the frame (intra-frame prediction) or from macroblock(s) of a previous frame (inter-frame prediction). The analogous structure to macroblocks for HEVC is a coding tree unit (CTU), which can be larger than macroblocks (e.g. up to 64x64 pixels in size), and which are further divided into coding units (CUs). HEVC offers some improvements over AVC, including improved motion vector determination, motion compensation and intra-frame prediction, that may allow for improved data compression when compared to AVC. However, the "scalable" aspect of HEVC is very similar to the "scalable" aspect of AVC; namely that both use the idea of parallel encoding streams, whereby some efficiencies may be gained via inter-layer information exchange. For example, SHVC also offers inter-layer signalling that includes interlayer intra-prediction, interlayer residual prediction, and interlayer motion compensation. Like SVC, different levels of quality, e.g. HD and UHD are encoded by parallel layers and then combined in a stream for decoding.

Despite the availability of SVC and SHVC, the take up of scalable codecs has been below expectations. One reason for this is the complexity of these schemes and the modest bandwidth savings. Within the field of video delivery, many leading industry experts believed that the current available solutions do not address the challenges of delivering video in the twenty-first century. These industry experts include a large range of entities from vendors to traditional broadcasters, and from satellite providers to over-the-top (OTT) service providers such as social media companies.

In general, video service providers need to work with complex ecosystems. The selection of video codecs are often based on many various factors, including maximum compatibility with their existing ecosystems and costs of deploying the technology (e.g. both resource and monetary costs). Once a selection is made, it is difficult to change codecs without further massive investments in the form of equipment and time. Currently, it is difficult to upgrade an ecosystem without needing to replace it completely. Further, the resource cost and complexity of delivering an increasing number of services, sometimes using decentralised infrastructures such as so-called "cloud" configurations, are becoming a key concern for service operators, small and big alike. This is compounded by the rise in low-resource battery-powered edge devices (e.g. nodes in the so-called Internet of Things). All these factors need to be balanced with a need to reduce resource usage, e.g. to become more environmentally friendly, and a need to scale, e.g. to increase the number of users and provided services.

There is also a problem that many comparative codecs were developed in a time where large-scale commodity hardware was unavailable. This is not the case today. Large-scale data centres provide cheap generic data processing hardware. This is at odds with traditional video coding solutions that require bespoke hardware to operate efficiently.

SUMMARY

Aspects of the present invention are set out in the appended independent claims. Certain variations of the invention are then set out in the appended dependent claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Examples of the invention will now be described, by way of example only, with reference to the accompanying drawings.

Figure 1 is a schematic illustration of an encoder according to a first example.

Figure 2 is a schematic illustration of a decoder according to a first example.

Figure 3A is a schematic illustration of an encoder according to a first variation of a second example.

Figure 3B is a schematic illustration of an encoder according to a second variation of the second example.

Figure 4 is a schematic illustration of an encoder according to a third example.

Figure 5A is a schematic illustration of a decoder according to a second example.

Figure 5B is a schematic illustration of a first variation of a decoder according to a third example.

Figure 5C is a schematic illustration of a second variation of the decoder according to the third example.

Figure 6A is a schematic illustration showing an example 4 by 4 coding unit of residuals.

Figure 6B is a schematic illustration showing how coding units may be arranged in tiles.

Figures 7A to 7C are schematic illustrations showing possible colour plane arrangements.

Figure 8 is a flow chart shows a method of configuring a bit stream.

Figure 9A is a schematic illustration showing how a colour plane may be decomposed into a plurality of layers.

Figures 9B to 9J are schematic illustrations showing various methods of up-sampling.

Figure 10A to 10I are schematic illustrations showing various methods of entropy encoding quantized data.

Figures 11A to 11C are schematic illustrations showing aspects of different temporal modes.

Figures 12A and 12B are schematic illustrations showing components for applying temporal prediction according to examples.

Figures 12C and 12D are schematic illustrations showing how temporal signalling relates to coding units and tiles.

Figure 12E is a schematic illustration showing an example state machine for run-length encoding.

Figure 13A and 13B are two halves of a flow chart that shows a method of applying temporal processing according to an example.

Figures 14A to 14C are schematic illustrations showing example aspects of cloud control.

Figure 15 is a schematic illustration showing residual weighting according to an example.

Figures 16A to 16D are schematic illustrations showing calculation of predicted average elements according to various examples.

Figures 17A and 17B are schematic illustrations showing a rate controller that may be applied to one or more of first and second level enhancement encoding.

Figure 18 is a schematic illustration showing a rate controller according to a first example.

Figure 19 is a schematic illustration showing a rate controller according to a second example.

Figures 20A to 20D are schematic illustrations showing various aspects of quantization that may be used in examples.

Figures 21A and 21B are schematic illustrations showing different bitstream configurations.

Figures 22A to 22D are schematic illustrations showing different aspects of an example neural network up-sampler.

Figure 23 is a schematic illustration showing an example of how a frame may be encoded.

Figure 24 is a schematic illustration of a decoder according to a fourth example.

Figure 25 is a schematic illustration of an encoder according to a fifth example.

Figure 26 is a schematic illustration of a decoder according to a fifth example.

Figure 27 is a flow chart indicated a decoding process according to an example.

DETAILED DESCRIPTION

Introduction

Certain examples described herein relate to a framework for a new video coding technology that is flexible, adaptable, highly efficient and computationally inexpensive coding. It combines a selectable a base codec (e.g. AVC, HEVC, or any other present or future codec) with at least two enhancement levels of coded data. The framework offers an approach that is low complexity yet provides for flexible enhancement of video data.

Certain examples described herein build on a new multi-layer approach that has been developed. Details of this approach are described, for example, in US Patent Nos. US8,977,065, US8,948,248, US8,711,943, US9,129,411, US8,531,321, US9,510,018, US9,300,980, and US9,626,772 and PCT applications Nos. PCT/EP2013/059833, PCT/EP2013/059847, PCT/EP2013/059880, PCT/EP2013/059853, PCT/EP2013/059885, PCT/EP2013/059886, and PCT/IB2014/060716, which are all included herein by reference. This new multi-layer approach uses a hierarchy of layers wherein each layer may relate to a different level of quality, such as a different video resolution.

Examples of a low complexity enhancement video coding are described. Encoding and decoding methods are described, as well as corresponding encoders and decoders. The enhancement coding may operate on top of a base layer, which may provide base encoding and decoding. Spatial scaling may be applied across different layers. Only the base layer encodes full video, which may be at a lower resolution. The enhancement coding instead operates on computed sets of residuals. The sets of residuals are computed for a plurality of layers, which may represent different levels of scaling in one or more dimensions. A number of encoding and decoding components or tools are described, which may involve the application of transformations, quantization, entropy encoding and temporal buffering. At an example decoder, an encoded base stream and one or more encoded enhancement streams may be independently decoded and combined to reconstruct an original video.

The general structure of an example encoding scheme presented herein uses a down-sampled source signal encoded with a base codec, adds a first level of correction data to the decoded output of the base codec to generate a corrected picture, and then adds a further level of enhancement data to an up-sampled version of the corrected picture.

An encoded stream as described herein may be considered to comprise a base stream and an enhancement stream. The enhancement stream may have multiple layers (e.g. two are described in examples). The base stream may be decodable by a hardware decoder while the enhancement stream may be suitable for software processing implementation with suitable power consumption.

Certain examples described herein have a structure that provides a plurality of degrees of freedom, which in turn allows great flexibility and adaptability to many situations. This means that the coding format is suitable for many use cases including OTT transmission, live streaming, live UHD broadcast, and so on.

Although the decoded output of the base codec is not intended for viewing, it is a fully decoded video at a lower resolution, making the output compatible with existing decoders and, where considered suitable, also usable as a lower resolution output.

In the following description, certain example architectures for video encoding and decoding are described. These architectures use a small number of simple coding tools to reduce complexity. When combined synergistically, they can provide visual quality improvements when compared with a full resolution picture encoded with the base codec whilst at the same time generating flexibility in the way they can be used.

The present described examples provide a solution to the recent desire to use less and less power and, contributes to reducing the computational cost of encoding and decoding whilst increasing performance. The present described examples may operate as a software layer on top of existing infrastructures and deliver desired performances. The present examples provide a solution that is compatible with existing (and future) video streaming and delivery ecosystems whilst delivering video coding at a lower computational cost than it would be otherwise possible with a tout-court upgrade. Combining the coding efficiency of the latest codecs with the processing power reductions of the described examples may improve a technical case for the adoption of next-generation codecs.

Certain examples described herein operate upon residuals. Residuals may be computed by comparing two images or video signals. In one case, residuals are computed by comparing frames from an input video stream with frames of a reconstructed video stream. In the case of the level 1 enhancement stream as described herein the residuals may be computed by comparing a down-sampled input video stream with a first video stream that has been encoded by a base encoder and then decoded by a base decoder (e.g. the first video stream simulates decoding and reconstruction of the down-sampled input video stream at a decoder). In the case of the level 2 enhancement stream as described herein the residuals may be computed by comparing the input video stream (e.g. at a level of quality or resolution higher than the down-sampled or base video stream) with a second video stream that is reconstructed from an up-sampled version of the first video stream plus a set of decoded level 1 residuals (e.g. the second video stream simulates decoding both a base stream and the level 1 enhancement stream, reconstructing a video stream at a lower or down-sampled level of quality, then up-sampling this reconstructed video stream). This is, for example, shown in Figures 1 to 5C.

In certain examples, residuals may thus be considered to be errors or differences at a particular level of quality or resolution. In described examples, there are two levels of quality or resolutions and thus two sets of residuals (levels 1 and 2). Each set of residuals described herein models a different form of error or difference. The level 1 residuals, for example, typically correct for the characteristics of the base encoder, e.g. correct artefacts that are introduced by the base encoder as part of the encoding process. In contrast, the level 2 residuals, for example, typically correct complex effects introduced by the shifting in the levels of quality and differences introduced by the level 1 correction (e.g. artefacts generated over a wider spatial scale, such as areas of 4 or 16 pixels, by the level 1 encoding pipeline). This means it is not obvious that operations performed on one set of residuals will necessarily provide the same effect for another set of residuals, e.g. each set of residuals may have different statistical patterns and sets of correlations.

In the examples described herein residuals are encoded by an encoding pipeline. This may include transformation, quantization and entropy encoding operations. It may also include residual ranking, weighting and filtering, and temporal processing. These pipelines are shown in Figures 1 and 3A and 3B. Residuals are then transmitted to a decoder, e.g. as level 1 and level 2 enhancement streams, which may be combined with a base stream as a hybrid stream (or transmitted separately). In one case, a bit rate is set for a hybrid data stream that comprises the base stream and both enhancements streams, and then different adaptive bit rates are applied to the individual streams based on the data being processed to meet the set bit rate (e.g. high-quality video that is perceived with low levels of artefacts may be constructed by adaptively assigning a bit rate to different individual streams, even at a frame by frame level, such that constrained data may be used by the most perceptually influential individual streams, which may change as the image data changes).

The sets of residuals as described herein may be seen as sparse data, e.g. in many cases there is no difference for a given pixel or area and the resultant residual value is zero. When looking at the distribution of residuals much of the probability mass is allocated to small residual values located near zero – e.g. for certain videos values of -2, -1, 0, 1, 2 etc occur the most frequently. In certain cases, the distribution of residual values is symmetric or near symmetric about 0. In certain test video cases, the distribution of residual values was found to take a shape similar to logarithmic or exponential distributions (e.g. symmetrically or near symmetrically) about 0. The exact distribution of residual values may depend on the content of the input video stream.

Residuals may be treated as a two-dimensional image in themselves, e.g. a delta image of differences. Seen in this manner the sparsity of the data may be seen to relate features like “dots”, small “lines”, “edges”, “corners”, etc. that are visible in the residual images. It has been found that these features are typically not fully correlated (e.g. in space and/or in time). They have characteristics that differ from the characteristics of the image data they are derived from (e.g. pixel characteristics of the original video signal).

As the characteristics of the present residuals, including transformed residuals in the form of coefficients, differ from the characteristics of the image data they are derived from it is generally not possible to apply standard encoding approaches, e.g. such as those found in traditional Moving Picture Experts Group (MPEG) encoding and decoding standards. For example, many comparative schemes use large transforms (e.g. transforms of large areas of pixels in a normal video frame). Due to the characteristics of residuals, e.g. as described herein, it would be very inefficient to use these comparative large transforms on residual images. For example, it would be very hard to encode a small dot in a residual image using a large block designed for an area of a normal image.

Certain examples described herein address these issues by instead using small and simple transform kernels (e.g. 2x2 or 4x4 kernels – the Directional Decomposition and the

Directional Decomposition Squared – as presented herein). This moves in a different direction from comparative video coding approaches. Applying these new approaches to blocks of residuals generates compression efficiency. For example, certain transforms generate uncorrelated coefficients (e.g. in space) that may be efficiently compressed. While correlations between coefficients may be exploited, e.g. for lines in residual images, these can lead to encoding complexity, which is difficult to implement on legacy and low-resource devices, and often generates other complex artefacts that need to be corrected. In the present examples, a different transform is used (Hadamard) to encode the correction data and the residuals than comparative approaches. For example, the transforms presented herein may be much more efficient than transforming larger blocks of data using a Discrete Cosine Transform (DCT), which is the transform used in SVC/SHVC.

Certain examples described herein also consider the temporal characteristics of residuals, e.g. as well as spatial characteristics. For example, in residual images details like "edges" and "dots" that may be observed in residual "images" show little temporal correlation. This is because "edges" in residual images often don't translate or rotate like edges as perceived in a normal video stream. For example, within residual images, "edges" may actually change shape over time, e.g. a head turning may be captured within multiple residual image "edges" but may not move in a standard manner (as the "edge" reflects complex differences that depend on factors such as lighting, scale factors, encoding factors etc.). These temporal aspects of residual images, e.g. residual "video" comprising sequential residual "frames" or "pictures" typically differ from the temporal aspects of conventional images, e.g. normal video frames (e.g. in the Y, U or V planes). Hence, it is not obvious how to apply conventional encoding approaches to residual images; indeed, it has been found that motion compensation approaches from comparative video encoding schemes and standards cannot encode residual data (e.g. in a useful manner).

An AVC layer within SVC may involve calculating data that are referred to in that comparative standard as "residuals". However, these comparative "residuals" are the difference between a pixel block of the data stream of that layer and a corresponding pixel block determined using either inter-frame prediction or intra-frame prediction. These comparative "residuals" are, however, very different from residuals encoded in the present examples. In SVC, the "residuals" are the difference between a pixel block of a frame and a predicted pixel block for the frame (predicted using either inter-frame prediction or intra-frame prediction). In contrast, the present examples involve calculating residuals as a difference between a coding block and a reconstructed coding block (e.g. which has undergone down-sampling and subsequent up-sampling, and has been corrected for encoding / decoding errors).

Furthermore, many comparative video encoding approaches attempt to provide temporal prediction and motion-compensation as default to conventional video data. These "built-in" approaches may not only fail when applied to sequential residual images, they may take up unnecessary processing resources (e.g. these resources may be used while actually corrupting the video encoding). It may also generate unnecessary bits that take up an assigned bit rate. It is not obvious from conventional approaches how to address these problems.

Certain examples described herein, e.g. as described in the "Temporal Aspects" section and elsewhere, provide an efficient way of predicting temporal features within residual images. Certain examples use zero-motion vector prediction to efficiently predict temporal aspects and movement within residuals. These may be seen to predict movement for relatively static features (e.g. apply the second temporal mode - inter prediction - to residual features that persist over time) and then use the first temporal mode (e.g. intra prediction) for everything else. Hence, certain examples described herein do not attempt to waste scare resources and bit rate predicting transient uncorrelated temporal features in residual "video".

Certain examples described herein allow for legacy, existing and future codecs to be enhanced. The examples may thus leverage the capabilities of these codes as part of a base layer and provide improvements in the form of an enhancement layer.

Certain examples described herein are low complexity. They enable a base codec to be enhanced with low computational complexity and/or in a manner that enables widespread parallelisation. If down-sampling is used prior to the base codec (e.g. an application of spatial scalability), then a video signal at the original input resolution may be provided with a reduced computational complexity as compared to using the base codec at the original input resolution. This allows wide adoption of ultra-high-resolution video. For example, by a combination of processing an input video at a lower resolution with a single-layer existing codec and using a simple and small set of highly specialised tools to add details to an up-sampled version of the processed video, many advantages may be realised.

Certain examples described herein implement a number of modular yet specialised video coding tools. The tools that make up the enhancement layer (including two levels of enhancement at two different points) are designed for a particular type of data: residual data. Residual data as described herein results from a comparison of an original data signal and a reconstructed data signal. The reconstructed data signal is generated in a manner that differs from comparative video coding schemes. For example, the reconstructed data signal relates to a particular small spatial portion of an input video frame – a coding unit. A set of coding units for a frame may be processed in parallel as the residual data is not generated using other coding units for the frame or other coding units for other frames, as opposed to inter- and intra- prediction in comparative video coding technologies. Although temporal processing may be applied, this is applied at the coding unit level, using previous data for a current coding unit. There is no interdependency between coding units.

Certain specialised video coding tools described herein are specifically adapted for sparse residual data processing. Due to the differing method of generation, residual data as used herein has different properties to that of comparative video coding technologies. As shown in the Figures, certain examples described herein provide an enhancement layer that processes one or two layers of residual data. The residual data is produced by taking differences between a reference video frame (e.g., a source video) and a base-decoded version of the video (e.g. with or without up-sampling depending on the layer). The resulting residual data is sparse information, typically edges, dots and details which are then processed using small transforms which are designed to deal with sparse information. These small transforms may be scale invariant, e.g. have integer values within the range of {-1, 1}.

Certain examples described herein allow efficient use of existing codecs. For example, a base encoder is typically applied at a lower resolution (e.g. than an original input signal). A base decoder is then used to decode the output of the base encoder at the lower resolution and the resultant decoded signal is used to generate the decoded data. Because of this, the base codec operates on a smaller number of pixels, thus allowing the codec to operate at a higher level of quality (e.g. a smaller quantization step size) and use its own internal coding tools in a more efficient manner. It may also consume less power.

Certain examples described herein provide a resilient and adaptive coding process. For example, the configuration of the enhancement layer allows the overall coding process to be resilient to the typical coding artefacts introduced by traditional Discrete Cosine Transform (DCT) block-based codecs that may be used in the base layer. The first enhancement layer (level 1 residuals) enables the correction of artefacts introduced by the base codec, whereas the second enhancement layer (level 2 residuals) enables the addition of details and sharpness to a corrected up-sampled version of the signal. The level of correction may be adjusted by controlling a bit-rate up to a version that provides maximum fidelity and lossless encoding. Typically, the worse the base reconstruction, the more the first enhancement layer may contribute to a correction (e.g. in the form of encoded residual data output by that layer). Conversely, the better the base reconstruction, the more bit-rate can be allocated to the second enhancement layer (level 2 residuals) to sharpen the video and add fine details.

Certain examples described herein provide for agnostic base layer enhancement. For example, the examples may be used to enhance any base codec, from existing codecs such as MPEG-2, VP8, AVC, HEVC, VP9, AV1, etc. to future codecs including those under development such as EVC and VVC. This is possible because the enhancement layer operates on a decoded version of the base codec, and therefore it can be used on any format as it does not require any information on how the base layer has been encoded and/or decoded.

As described below, certain examples described herein allow for parallelization of enhancement layer encoding. For example, the enhancement layer does not implement any form of inter (i.e. between) block prediction. The image is processed applying small (2x2 or 4x4) independent transform kernels over the layers of residual data. Since no prediction is made between blocks, each 2x2 or 4x4 block can be processed independently and in a parallel manner. Moreover, each layer is processed separately, thus allowing decoding of the blocks and decoding of the layers to be done in a massively parallel manner.

With the presently described examples, errors introduced by the encoding / decoding process and the down-sampling / up-sampling process may be corrected for separately, to regenerate the original video on the decoder side. The encoded residuals and the encoded correction data are thus smaller in size than the input video itself and can therefore be sent to the decoder more efficiently than the input video (and hence more efficiently than a comparative UHD stream of the SVC and SHVC approaches).

In further comparison with SVC and SHVC, certain described examples involve sending encoded residuals and correction data to a decoder, without sending an encoded UHD stream itself. In contrast, in SVC and SHVC, both the HD and UHD images are encoded as separate video streams and sent to the decoder. The presently described examples may allow for a significantly reduction in the overall bit rate for sending the encoded data to the decoder, e.g. so that $BW_{Tot} \cong 0.7\, BW_{UHD}$. In these cases, the total bandwidth for sending both an HD stream and a UHD stream may be less than the bandwidth required by comparative standards to send just the UHD stream.

The presently described examples further allow coding units or blocks to be processed in parallel rather than sequentially. This is because the presently described examples do not apply intra-prediction; there is very limited spatial correlation between the spatial coefficients of different blocks, whereas SVC/SHVC provides for intra-prediction. This is more efficient than the comparative approaches of SVC/SHVC, which involve processing blocks sequentially (e.g. as the UHD stream relies on the predictions from various pixels of the HD stream).

The enhancement coding described in examples herein may be considered an enhancement codec that encodes and decodes streams of residual data. This differs from comparative SVC and SHVC implementations where encoders receive video data as input at each spatial resolution level and decoders output video data at each spatial resolution level. As such, the comparative SVC and SHVC may be seen as the parallel implementation of a set of codecs, where each codec has a video-in / video-out coding structure. The enhancement codecs described herein on the other hand receive residual data and also output residual data at each spatial resolution level. For example, in SVC and SHVC the outputs of each spatial resolution level are not summed to generate an output video – this would not make sense.

It should be noted that in examples references to levels 1 and 2 are to be taken as an arbitrary labelling of enhancement sub-layers. These may alternatively be referred to be different names (e.g. with a reversed numbering system with levels 1 and 2 being respectively labelled as level 1 and level 0, with the "level 0" base layer below being level 2).

Definitions and Terms

In certain examples described herein the following terms are used.

"access unit" – this refers to a set of Network Abstraction Layer (NAL) units that are associated with each other according to a specified classification rule. They may be consecutive in decoding order and contain a coded picture (i.e. frame) of video (in certain cases exactly one).

"base layer" – this is a layer pertaining to a coded base picture, where the "base" refers to a codec that receives processed input video data. It may pertain to a portion of a bitstream that relates to the base.

"bitstream" – this is sequence of bits, which may be supplied in the form of a NAL unit stream or a byte stream. It may form a representation of coded pictures and associated data forming one or more coded video sequences (CVSs).

"block" – an MxN (M-column by N-row) array of samples, or an MxN array of transform coefficients. The term "coding unit" or "coding block" is also used to refer to an MxN array of samples. These terms may be used to refer to sets of picture elements (e.g. values for pixels of a particular colour channel), sets of residual elements, sets of values that represent processed residual elements and/or sets of encoded values. The term "coding unit" is sometimes used to refer to a coding block of luma samples or a coding block of chroma samples of a picture that has three sample arrays, or a coding block of samples of a monochrome picture or a picture that is coded using three separate colour planes and syntax structures used to code the samples.

"byte" – a sequence of 8 bits, within which, when written or read as a sequence of bit values, the left-most and right-most bits represent the most and least significant bits, respectively.

"byte-aligned" – a position in a bitstream is byte-aligned when the position is an integer multiple of 8 bits from the position of the first bit in the bitstream, and a bit or byte or syntax element is said to be byte-aligned when the position at which it appears in a bitstream is byte-aligned.

"byte stream" – this may be used to refer to an encapsulation of a NAL unit stream containing start code prefixes and NAL units.

"chroma" – this is used as an adjective to specify that a sample array or single sample is representing a colour signal. This may be one of the two colour difference signals related to the primary colours, e.g. as represented by the symbols Cb and Cr. It may also be used to refer to channels within a set of colour channels that provide information on the colouring of a picture. The term chroma is used rather than the term chrominance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term chrominance.

"chunk" – this is used to refer to an entropy encoded portion of data containing a quantized transform coefficient belonging to a coefficient group.

"coded picture" – this is used to refer to a set of coding units that represent a coded representation of a picture.

"coded base picture" – this may refer to a coded representation of a picture encoded using a base encoding process that is separate (and often differs from) an enhancement encoding process.

"coded representation" – a data element as represented in its coded form

"coefficient group (CG)" – is used to refer to a syntactical structure containing encoded data related to a specific set of transform coefficients (i.e. a set of transformed residual values).

"component" or "colour component" – this is used to refer to an array or single sample from one of a set of colour component arrays. The colour components may comprise one luma and two chroma components and/or red, green, blue (RGB) components. The colour components may not have a one-to-one sampling frequency, e.g. the components may compose a picture in 4:2:0, 4:2:2, or 4:4:4 colour format. Certain examples described herein may also refer to just a single monochrome (e.g. luma or grayscale) picture, where there is a single array or a single sample of the array that composes a picture in monochrome format.

"data block" – this is used to refer to a syntax structure containing bytes corresponding to a type of data.

"decoded base picture" – this is used to refer to a decoded picture derived by decoding a coded base picture.

"decoded picture" – a decoded picture may be derived by decoding a coded picture. A decoded picture may be either a decoded frame, or a decoded field. A decoded field may be either a decoded top field or a decoded bottom field.

"decoded picture buffer (DPB)" – this is used to refer to a buffer holding decoded pictures for reference or output reordering.

"decoder" – equipment or a device that embodies a decoding process.

"decoding order" – this may refer to an order in which syntax elements are processed by the decoding process.

"decoding process" – this is used to refer to a process that reads a bitstream and derives decoded pictures from it.

"emulation prevention byte" – this is used in certain examples to refer to a byte equal to 0x03 that may be present within a NAL unit. Emulation prevention bytes may be used to ensure that no sequence of consecutive byte-aligned bytes in the NAL unit contains a start code prefix.

"encoder" - equipment or a device that embodies a encoding process.

"encoding process" – this is used to refer to a process that produces a bitstream (i.e. an encoded bitstream).

"enhancement layer" – this is a layer pertaining to a coded enhancement data, where the enhancement data is used to enhance the "base layer" (sometimes referred to as the "base"). It may pertain to a portion of a bitstream that comprises planes of residual data. The singular term is used to refer to encoding and/or decoding processes that are distinguished from the "base" encoding and/or decoding processes.

"enhancement sub-layer" – in certain examples, the enhancement layer comprises multiple sub-layers. For example, the first and second levels described below are "enhancement sub-layers" that are seen as layers of the enhancement layer.

"field" – this term is used in certain examples to refer to an assembly of alternate rows of a frame. A frame is composed of two fields, a top field and a bottom field. The term field may be used in the context of interlaced video frames.

"video frame" – in certain examples a video frame may comprise a frame composed of an array of luma samples in monochrome format or an array of luma samples and two corresponding arrays of chroma samples. The luma and chroma samples may be supplied in 4:2:0, 4:2:2, and 4:4:4 colour formats (amongst others). A frame may consist of two fields, a top field and a bottom field (e.g. these terms may be used in the context of interlaced video).

"group of pictures (GOP)" – this term is used to refer to a collection of successive coded base pictures starting with an intra picture. The coded base pictures may provide the reference ordering for enhancement data for those pictures.

"instantaneous decoding refresh (IDR) picture" – this is used to refer to a picture for which an NAL unit contains a global configuration data block.

"inverse transform" – this is used to refer to part of the decoding process by which a set of transform coefficients are converted into residuals.

"layer" – this term is used in certain examples to refer to one of a set of syntactical structures in a non-branching hierarchical relationship, e.g. as used when referring to the "base" and "enhancement" layers, or the two (sub-) "layers" of the enhancement layer.

"luma" – this term is used as an adjective to specify a sample array or single sample that represents a lightness or monochrome signal, e.g. as related to the primary colours. Luma samples may be represented by the symbol or subscript Y or L. The term "luma" is used rather than the term luminance in order to avoid the implication of the use of linear light transfer characteristics that is often associated with the term luminance. The symbol

L is sometimes used instead of the symbol Y to avoid confusion with the symbol y as used for vertical location.

"network abstraction layer (NAL) unit (NALU)" – this is a syntax structure containing an indication of the type of data to follow and bytes containing that data in the form of a raw byte sequence payload (RBSP – see definition below).

“network abstraction layer (NAL) unit stream” – a sequence of NAL units.

“output order” – this is used in certain examples to refer to an order in which the decoded pictures are output from the decoded picture buffer (for the decoded pictures that are to be output from the decoded picture buffer).

“partitioning” – this term is used in certain examples to refer to the division of a set into subsets. It may be used to refer to cases where each element of the set is in exactly one of the subsets.

“plane” – this term is used to refer to a collection of data related to a colour component. For example, a plane may comprise a Y (luma) or Cx (chroma) plane. In certain cases, a monochrome video may have only one colour component and so a picture or frame may comprise one or more planes.

“picture” – this is used as a collective term for a field or a frame. In certain cases, the terms frame and picture are used interchangeably.

“random access” – this is used in certain examples to refer to an act of starting the decoding process for a bitstream at a point other than the beginning of the stream.

“raw byte sequence payload (RBSP)” - the RBSP is a syntax structure containing an integer number of bytes that is encapsulated in a NAL unit. An RBSP is either empty or has the form of a string of data bits containing syntax elements followed by an RBSP stop bit and followed by zero or more subsequent bits equal to 0. The RBSP may be interspersed as necessary with emulation prevention bytes.

“raw byte sequence payload (RBSP) stop bit” – this is a bit that may be set to 1 and included within a raw byte sequence payload (RBSP) after a string of data bits. The location of the end of the string of data bits within an RBSP may be identified by searching from the end of the RBSP for the RBSP stop bit, which is the last non-zero bit in the RBSP.

“reserved” – this term may refer to values of syntax elements that are not used in the bitstreams described herein but are reserved for future use or extensions. The term “reserved zeros” may refer to reserved bit values that are set to zero in examples.

"residual" – this term is defined in further examples below. It generally refers to a difference between a reconstructed version of a sample or data element and a reference of that same sample or data element.

"residual plane" – this term is used to refer to a collection of residuals, e.g. that are organised in a plane structure that is analogous to a colour component plane. A residual plane may comprise a plurality of residuals (i.e. residual picture elements) that may be array elements with a value (e.g. an integer value).

"run length encoding" – this is a method for encoding a sequence of values in which consecutive occurrences of the same value are represented as a single value together with its number of occurrences.

"source" – this term is used in certain examples to describe the video material or some of its attributes before encoding.

"start code prefix" – this is used to refer to a unique sequence of three bytes equal to 0x000001 embedded in the byte stream as a prefix to each NAL unit. The location of a start code prefix may be used by a decoder to identify the beginning of a new NAL unit and the end of a previous NAL unit. Emulation of start code prefixes may be prevented within NAL units by the inclusion of emulation prevention bytes.

"string of data bits (SODB)" – this term refers to a sequence of some number of bits representing syntax elements present within a raw byte sequence payload prior to the raw byte sequence payload stop bit. Within an SODB, the left-most bit is considered to be the first and most significant bit, and the right-most bit is considered to be the last and least significant bit.

"syntax element" – this term may be used to refer to an element of data represented in the bitstream.

"syntax structure" – this term may be used to refer to zero or more syntax elements present together in the bitstream in a specified order.

"tile" – this term is used in certain examples to refer to a rectangular region of blocks or coding units within a particular picture, e.g. it may refer to an area of a frame that contains a plurality of coding units where the size of the coding unit is set based on an applied transform.

"transform coefficient" (or just "coefficient") – this term is used to refer to a value that is produced when a transformation is applied to a residual or data derived from a residual (e.g. a processed residual). It may be a scalar quantity, that is considered to be in a transformed domain. In one case, an M by N coding unit may be flattened into an M*N one-dimensional array. In this case, a transformation may comprise a multiplication of the one-dimensional array with an M by N transformation matrix. In this case, an output may comprise another (flattened) M*N one-dimensional array. In this output, each element may relate to a different “coefficient”, e.g. for a 2x2 coding unit there may be 4 different types of coefficient. As such, the term “coefficient” may also be associated with a particular index in an inverse transform part of the decoding process, e.g. a particular index in the aforementioned one-dimensional array that represented transformed residuals.

“video coding layer (VCL) NAL unit” – this is a collective term for NAL units that have reserved values of NalUnitType and that are classified as VCL NAL units in certain examples.

As well as the terms above, the following abbreviations are sometimes used:

CG - Coefficient Group; CPB- Coded Picture Buffer; CPBB - Coded Picture Buffer of the Base; CPBL - Coded Picture Buffer of the Enhancement; CU - Coding Unit; CVS – Coded Video Sequence; DPB - Decoded Picture Buffer; DPBB - Decoded Picture Buffer of the Base; DUT - Decoder Under Test; HBD - Hypothetical Base Decoder; HD - Hypothetical Demuxer; HRD - Hypothetical Reference Decoder; HSS - Hypothetical Stream Scheduler; I – Intra; IDR - Instantaneous Decoding Refresh; LSB - Least Significant Bit; MSB - Most Significant Bit; NAL - Network Abstraction Layer; P – Predictive; RBSP - Raw Byte Sequence Payload; RGB – red, green blue (may also be used as GBR – green, blue, red – i.e. reordered RGB; RLE - Run length encoding; SEI - Supplemental Enhancement Information; SODB - String of data bits; SPS - Sequence Parameter Set; and VCL - Video Coding Layer.

Example Encoders and Decoders

First Example Encoder – General Architecture

Figure 1 shows a first example encoder 100. The illustrated components may also be implemented as steps of a corresponding encoding process.

In the encoder 100, an input full resolution video 102 is received and is processed to generate various encoded streams. At a down-sampling component 104, the input video 102 is down-sampled. An output of the down-sampling component 104 is received by a base codec that comprises a base encoder 102 and a base decoder 104. A first encoded stream (encoded base stream) 116 is produced by feeding the base codec (e.g., AVC, HEVC, or any other codec) with a down-sampled version of the input video 102. At a first subtraction component 120, a first set of residuals is obtained by taking the difference between a reconstructed base codec video as output by the base decoder 104 and the down-sampled version of the input video (i.e. as output by the down-sampling component 104). A level 1 encoding component 122 is applied to the first set of residuals that are output by the first subtraction component 120 to produce a second encoded stream (encoded level 1 stream) 126.

In the example of Figure 1, the level 1 encoding component 122 operates with an optional level 1 temporal buffer 124. This may be used to apply temporal processing as described later below. Following a first level of encoding by the level 1 encoding component 122, the first encoded stream 126 may be decoded by a level 1 decoding component 128. A deblocking filter 130 may be applied to the output of the level 1 decoding component 128. In Figure 1, an output of the deblocking filter 130 is added to the output of the base decoder 114 (i.e. is added to the reconstructed base codec video) by a summation component 132 to generate a corrected version of the reconstructed base coded video. The output of the summation component 132 is then up-sampled by an up-sampling component 134 to produce an up-sampled version of a corrected version of the reconstructed base coded video.

At a second subtraction component 136, a difference between the up-sampled version of a corrected version of the reconstructed base coded video (i.e. the output of the up-sampling component 134) and the input video 102 is taken. This produces a second set of residuals. The second set of residuals as output by the second subtraction component 136 is passed to a level 2 encoding component 142. The level 2 encoding component 142 produces a third encoded stream (encoded level 2 stream) 146 by encoding the second set of residuals. The level 2 encoding component 142 may operate together with a level 2 temporal buffer 144 to apply temporal processing. One or more of the level 1 encoding component 122 and the level 2 encoding component 142 may apply residual selection as described below. This is shown as being controlled by a residual mode selection component 150. The residual mode selection component 150 may receive the input video 102 and apply residual mode selection based on an analysis of the input video 102. Similarly, the level 1 temporal buffer 124 and the level 2 temporal buffer 144 may operate under the control of a temporal selection component 152. The temporal selection component 152 may receive one or more of the input video 102 and the output of the down-sampling component 104 to select a temporal mode. This is explained in more detail in later examples.

First Example Decoder – General Architecture

Figure 2 shows a first example decoder 200. The illustrated components may also be implemented as steps of a corresponding decoding process. The decoder 200 receives three encoded streams: encoded base stream 216, encoded level 1 stream 226 and encoded level 2 stream 246. These three encoded streams correspond to the three streams generated by the encoder 100 of Figure 1. In the example of Figure 2, the three encoded streams are received together with headers 256 containing further decoding information.

The encoded base stream 216 is decoded by a base decoder 218 corresponding to the base codec used in the encoder 100 (e.g. corresponding to base decoder 114 in Figure 1). At a first summation component 220, the output of the base decoder 218 is combined with a decoded first set of residuals that are obtained from the encoded level 1 stream 226. In particular, a level 1 decoding component 228 receives the encoded level 1 stream 226 and decodes the stream to produce the decoded first set of residuals. The level 1 decoding component 228 may use a level 1 temporal buffer 230 to decode the encoded level 1 stream 226. In the example of Figure 2, the output of the level 1 decoding component 228 is passed to a deblocking filter 232. The level 1 decoding component 228 may be similar to the level 1 decoding component 128 used by the encoder 100 in Figure 1. The deblocking filter 232 may also be similar to the deblocking filter 130 used by the encoder 100. In Figure 2, the output of the deblocking filter 232 forms the decoded first set of residuals that are combined with the output of the base decoder 218 by the first summation component 220. The output of the first summation component 220 may be seen as a corrected level 1 reconstruction, where the decoded first set of residuals correct an output of the base decoder 218 at a first resolution.

At an up-sampling component 234, the combined video is up-sampled. The up-sampling component 234 may implement a form of modified up-sampling as described with respect to later examples. The output of the up-sampling component 234 is further combined with a decoded second set of residuals that are obtained from the encoded level 2 stream 246. In particular, a level 2 decoding component 248 receives the encoded level 2 stream 246 and decodes the stream to produce the decoded second set of residuals. The decoded second set of residuals, as output by the level 2 decoding component 248 are combined with the output of the up-sampling component 234 by summation component 258 to produce a decoded video 260. The decoded video 260 comprises a decoded representation of the input video 102 in Figure 1. The level 2 decoding component 248 may also use a level 2 temporal buffer 250 to apply temporal processing. One or more of the level 1 temporal buffer 230 and the level 2 temporal buffer 250 may operate under the control of a temporal selection component 252. The temporal selection component 252 is shown receiving data from headers 256. This data may comprise data to implement temporal processing at one or more of the level 1 temporal buffer 230 and the level 2 temporal buffer 250. The data may indicate a temporal mode that is applied by the temporal selection component 252 as described with reference to later examples.

Second Example Encoder – Encoding Sub-Processing and Temporal Prediction

Figures 3A and 3B show different variations of a second example encoder 300, 360. The second example encoder 300, 360 may comprise an implementation of the first example encoder 100 of Figure 1. In the examples of Figures 3A and 3B, the encoding steps of the stream are expanded in more detail to provide an example of how the steps may be performed. Figure 3A illustrates a first variation with temporal prediction provided only in the second level of the enhancement process, i.e. with respect to the level 2 encoding. Figure 3B illustrates a second variation with temporal prediction performed in processes of both levels of enhancement (i.e. levels 1 and 2).

In Figure 3A, an encoded base stream 316 is substantially created by a process as explained with respect to Figure 1 above. That is, an input video 302 is down-sampled (i.e. a down-sampling operation is applied by a down-sampling component 304 to the input video 102 to generate a down-sampled input video. The down-sampled video is then encoded using a base codec, in particular by a base encoder 312 of the base codec. An encoding operation applied by the base encoder 312 to the down-sampled input video generates an encoded base stream 316. The base codec may also be referred to as a first codec, as it may differ from a second codec that is used to produce the enhancement streams (i.e. the encoded level 1 stream 326 and the encoded level 2 stream 346). Preferably the first or base codec is a codec suitable for hardware decoding. As per Figure 1, an output of the base encoder 312 (i.e. the encoded base stream 316) is received by a base decoder 314 (e.g. that forms part of, or provides a decoding operation for, the base codec) that outputs a decoded version of the encoded base stream. The operations performed by the base encoder 312 and the base decoder 314 may be referred to as the base layer or base level. The base layer or level may be implemented separately from an enhancement or second layer or level, and the enhancement layer or level instructs and/or controls the base layer or level (e.g. the base encoder 312 and the base decoder 314).

As noted with respect to Figure 1, the enhancement layer or level may comprise two levels that produce two corresponding streams. In this context, a first level of enhancement (described herein as "level 1") provides for a set of correction data which can be combined with a decoded version of the base stream to generate a corrected picture. This first enhancement stream is illustrated in Figures 1 and 3 as the encoded level 1 stream 326.

To generate the encoded level 1 stream, the encoded base stream is decoded, i.e. an output of the base decoder 314 provides a decoded base stream. As in Figure 1, at a first subtraction component, a difference between the decoded base stream and the down-sampled input video (i.e. the output of the down-sampling component 304) is then created (i.e. a subtraction operation is applied to the down-sampled input video and the decoded base stream to generate a first set of residuals). Here the term "residuals" is used in the same manner as that known in the art, that is, the error between a reference frame and a desired frame. Here the reference frame is the decoded base stream and the desired frame is the down-sampled input video. Thus, the residuals used in the first enhancement level can be considered as a corrected video as they 'correct' the decoded base stream to the down-sampled input video that was used in the base encoding operation.

In general, the term "residuals" as used herein refers to a difference between a value of a reference array or reference frame and an actual array or frame of data. The array may be a one or two-dimensional array that represents a coding unit. For example, a coding unit may be a 2x2 or 4x4 set of residual values that correspond to similar sized areas of an input video frame. It should be noted that this generalised example is agnostic as to the encoding operations performed and the nature of the input signal. Reference to "residual data" as used herein refers to data derived from a set of residuals, e.g. a set of residuals themselves or an output of a set of data processing operations that are performed on the set of residuals. Throughout the present description, generally a set of residuals includes a plurality of residuals or residual elements, each residual or residual element corresponding to a signal element, that is, an element of the signal or original data. The signal may be an image or video. In these examples, the set of residuals corresponds to an image or frame of the video, with each residual being associated with a pixel of the signal, the pixel being the signal element.

It should be noted that the "residuals" described herein are, however, very different from "residuals" that are generated in comparative technologies such as SVC and SHVC. In SVC, the term "residuals" is used to refer to a difference between a pixel block of a frame and a predicted pixel block for the frame, where the predicted pixel block is predicted using either inter-frame prediction or intra-frame prediction. In contrast, the present examples involve calculating residuals as a difference between a coding unit and a reconstructed coding unit, e.g. a coding unit of elements that has undergone down-sampling and subsequent up-sampling, and has been corrected for encoding / decoding errors. In the described examples, the base codec (i.e. the base encoder 312 and the base decoder 314) may comprise a different codec from the enhancement codec, e.g. the base and enhancement streams are generated by different sets of processing steps. In one case, the base encoder 312 may comprise an AVC or HEVC encoder and thus internally generates residual data that is used to generate the encoded base stream 316. However, the processes that are used by the AVC or HEVC encoder differ from those that are used to generate the encoded level 1 and level 2 streams 326, 346.

Returning to Figures 3A and 3B, an output of the subtraction component 320, i.e. a difference that corresponds to a first set of residuals, is then encoded to generate the encoded level 1 stream 326 (i.e. an encoding operation is applied to the first set of residuals to generate a first enhancement stream). In the example implementations of Figures 3A and 3B, the encoding operation comprises several sub-operations, each of which is optional and preferred and provides particular benefits. In Figures 3A and 3B, a series of components are shown that implement these sub-operations and these may be considered to implement the level 1 and level 2 encoding 122 and 142 as shown in Figure 1. In Figures 3A and 3B, the sub-operations, in general, include a residuals ranking mode step, a transform step, a quantization step and an entropy encoding step.

For the level 1 encoding, a level 1 residuals selection or ranking component 321 receives an output of the first subtraction component 320. The level 1 residuals selection or ranking component 321 is shown as being controlled by a residual mode ranking or selection component 350 (e.g. in a similar manner to the configuration of Figure 1). In Figure 3A, ranking is performed by the residual mode ranking component 350 and applied by the level 1 selection component 321, the latter selecting or filtering the first set of residuals based on a ranking performed by the residual mode ranking component 350 (e.g. based on an analysis of the input video 102 or other data). In Figure 3B this arrangement is reversed, such that a general residual mode selection control is performed by a residual mode selection component 350 but ranking is performed at each enhancement level (e.g. as opposed to ranking based on the input video 102). In the example of Figure 3B, the ranking may be performed by the level 1 residual mode ranking component 321 based on an analysis of the first set of residuals as output by the first subtraction component 320.

In general, the second example encoder 300, 360 identifies if the residuals ranking mode is selected. This may be performed by the residual mode ranking or selection component 350. If a residuals ranking mode is selected, then this may be indicated by the residual mode ranking or selection component 350 to the level 1 residuals selection or ranking component 321 to perform a residuals ranking step. The residuals ranking operation may be performed on the first step of residuals to generate a ranked set of residuals. The ranked set of residuals may be filtered so that not all residuals are encoded into the first enhancement stream 326 (or correction stream). Residual selection may comprise selecting a subset of received residuals to pass through for further encoding. Although the present examples describe a "ranking" operation, this may be seen as a general filtering operation that is performed on the first set of residuals (e.g. the output of the first subtraction component 320), i.e. the level 1 residuals selection or ranking component 321 is an implementation of a general filtering component that may modify the first set of residuals. Filtering may be seen as setting certain residual values to zero, i.e. such that an input residual value is filtered out and does not form part of the encoded level 1 stream 326.

In Figures 3A and 3B, an output of the level 1 residuals selection or ranking component 321 is then received by a level 1 transform component 322. The level 1 transform component 322 applies a transform to the first set of residuals, or the ranked or filtered first set of residuals, to generate a transformed set of residuals. The transform operation may be applied to the first set of residuals or the filtered first set of residuals depending on whether or not ranking mode is selected to generate a transformed set of residuals. A level 1 quantize component 323 is then applied to an output of the level 1 transform component 322 (i.e. the transformed set of residuals) to generate a set of quantized residuals. Entropy encoding is applied by a level 1 entropy encoding component 325 that applies an entropy encoding operation to the quantized set of residuals (or data derived from this set) to generate the first level of enhancement stream, i.e. the encoded level 1 stream 326. Hence, in the level 1 layer a first set of residuals are transformed, quantized and entropy encoded to produce the encoded level 1 stream 326. Further details of possible implementations of the transformation, quantization and entropy encoding are described with respect to later examples. Preferably, the entropy encoding operation may be a Huffman encoding operation or a run-length encoding operation or both. Optionally a control operation may be applied to the quantized set of residuals so as to correct for the effects of the ranking operation. This may be applied by the level 1 residual mode control component 324, which may operate under the control of the residual mode ranking or selection component 350.

As noted above, the enhancement stream may comprise a first level of enhancement and a second level of enhancement (i.e. levels 1 and 2). The first level of enhancement may be considered to be a corrected stream. The second level of enhancement may be considered to be a further level of enhancement that converts the corrected stream to the original input video. The further or second level of enhancement is created by encoding a further or second set of residuals which are the difference between an up-sampled version of a reconstructed level 1 video as output by the summation component 332 and the input video 302. Up-sampling is performed by an up-sampling component 334. The second set of residuals result from a subtraction applied by a second subtraction component 336, which takes the input video 302 and the output of the up-sampling component 334 as inputs.

In Figures 3A and 3B, the first set of residuals are encoded by a level 1 encoding process. This process, in the example of Figures 3A and 3B, comprises the level 1 transform component 322 and the level 1 quantize component 323. Before up-sampling, the encoded first set of residuals are decoded using an inverse quantize component 327 and an inverse transform component 328. These components act to simulate (level 1) decoding components that may be implemented at a decoder. As such, the quantized (or controlled) set of residuals that are derived from the application of the level 1 transform component 322 and the level 1 quantize component 323 are inversely quantized and inversely transformed before a de-blocking filter 330 is applied to generate a decoded first set of residuals (i.e. an inverse quantization operation is applied to the quantized first set of residuals to generate a de-quantized first set of residuals; an inverse transform operation is applied to the de-quantized first set of residuals to generate a de-transformed first set of residuals; and, a de-blocking filter operation is applied to the de-transformed first set of residuals to generate a decoded first set of residuals). The de-blocking filter 330 is optional depending on the transform applied and may comprise applying a weighted mask to each block of the de-transformed first set of residuals.

At the summation component 332, the decoded base stream as output by the base decoder 314 is combined with the decoded first set of residuals as received from the deblocking filter 330 (i.e. a summing operation is performed on the decoded base stream and the decoded first set of residuals to generate a re-created first stream). As illustrated in Figures 3A and B, that combination is then up-sampled by the up-sampling component

334 (i.e. an up-sampling operation is applied to the re-created first stream to generate an up-sampled re-created stream). The up-sampled stream is then compared to the input video at the second summation component 336, which creates the second set of residuals (i.e. a difference operation is applied to the up-sampled re-created stream to generate a further set of residuals). The second set of residuals are then encoded as the encoded level 2 enhancement stream 346 (i.e. an encoding operation is then applied to the further or second set of residuals to generate an encoded further or second enhancement stream).

As with the encoded level 1 stream, the encoding applied to the second set (level 2) residuals may comprise several operations. Figure 3A shows a level 2 residuals selection component 340, a level 2 transform component 341, a level 2 quantize component 343 and a level 2 entropy encoding component 345. Figure 3B shows a similar set of components but in this variation the level 2 residuals selection component 340 is implemented as a level 2 residuals ranking component 340, which is under control of the residual mode selection component 350. As discussed above, ranking and selection may be performed based on one or more of the input video 102 and the individual first and second sets of residuals. In Figure 3A, a level 2 temporal buffer 345 is also provided, the contents of which are subtracted from the output of the level 2 transform component 341 by third subtraction component 342. In other examples, the third subtraction component 342 may be located in other positions, including after the level 2 quantize component 343. As such the level 2 encoding shown in Figures 3A and 3B has steps of ranking, temporal prediction, transform, quantization and entropy encoding. In particular, the second example encoder 200 may identify if a residuals ranking mode is selected. This may be performed by one or more of the residual ranking or selection component 350 and the individual level 2 selection and ranking components 340. If a residuals ranking or filtering mode is selected the residuals ranking step may be performed by one or more of the residual ranking or selection component 350 and the individual level 2 selection and ranking components 340 (i.e. a residuals ranking operation may be performed on the second set of residuals to generate a second ranked set of residuals). The second ranked set of residuals may be filtered so that not all residuals are encoded into the second enhancement stream (i.e. the encoded level 2 stream 346). The second set of residuals or the second ranked set of residuals are subsequently transformed by the level 2 transform component 341 (i.e. a transform operation is performed on the second ranked set of residuals to generate a second transformed set of residuals). As illustrated by the coupling between the output of the summation component 332 and the level 2 transform component 341, the transform operation may utilise a predicted coefficient or predicted average derived from the re-created first stream, prior to up-sampling. Other examples of this predicted average computation are described with reference to other examples; further information may be found elsewhere in this document. In level 2, the transformed residuals (either temporally predicted or otherwise) are then quantized and entropy encoded in the manner described elsewhere (i.e. a quantization operation is applied to the transformed set of residuals to generate a second set of quantized residuals; and, an entropy encoding operation is applied to the quantized second set of residuals to generate the second level of enhancement stream).

Figure 3A shows a variation of the second example encoder 200 where temporal prediction is performed as part of the level 2 encoding process. Temporal prediction is performed using the temporal selection component 352 and the level 2 temporal buffer 345. The temporal selection component 352 may determine a temporal processing mode as described in more detail below and control the use of the level 2 temporal buffer 345 accordingly. For example, if no temporal processing is to be performed the temporal selection component 352 may indicate that the contents of the level 2 temporal buffer 345 are to be set to 0.

Figure 3B shows a variation of the second example encoder 200 where temporal prediction is performed as part of both the level 1 and the level 2 encoding process. In Figure 3B, a level 1 temporal buffer 361 is provided in addition to the level 2 temporal buffer 345. Although not shown, further variations where temporal processing is performed at level 1 but not level 2 are also possible.

When temporal prediction is selected, the second example encoder 200 may further modify the coefficients (i.e. the transformed residuals output by a transform component) by subtracting a corresponding set of coefficients derived from an appropriate temporal buffer. The corresponding set of coefficients may comprise a set of coefficients for a same spatial area (e.g. a same coding unit as located within a frame) that are derived from a previous frame (e.g. coefficients for the same area for a previous frame). The subtraction may be applied by a subtraction component such as the third subtractions components 346 and 362 (for respective levels 2 and 1). This temporal prediction step will be further described with respect to later examples. In summary, when temporal prediction is applied, the encoded coefficients correspond to a difference between the frame and an other frame of the stream. The other frame may be an earlier or later frame (or block in the frame) in the stream. Thus, instead of encoding the residuals between the up-sampled re-created stream and the input video, the encoding process may encode the difference between a transformed frame in the stream and the transformed residuals of the frame. Thus, the entropy may be reduced. Temporal prediction may be applied selectively for groups of coding units (referred to herein as "tiles") based on control information and the application of temporal prediction at a decoder may be applied by sending additional control information along with the encoded streams (e.g. within headers or as a further surface as described with reference to later examples).

As shown in Figures 3A and 3B, when temporal prediction is active, each transformed coefficient may be:

$$\Delta = F_{current} - F_{buffer}$$

where the temporal buffer may store data associated with a previous frame. Temporal prediction may be performed for one colour plane or for multiple colour planes. In general, the subtraction may be applied as an element wise subtraction for a "frame" of video where the elements of the frame represent transformed coefficients, where the transform is applied with respect to a particular n by n coding unit size (e.g. 2x2 or 4x4). The difference that results from the temporal prediction (e.g. the delta above may be stored in the buffer for use for a subsequent frame. Hence, in effect, the residual that results to the temporal prediction is a coefficient residual with respect to the buffer. Although Figures 3A and 3B show temporal prediction being performed after the transform operation, it may also be performed after the quantize operation. This may avoid the need to apply the level 2 inverse quantization component 372 and/or the level 1 inverse quantize component 364.

Thus, as illustrated in Figures 3A and 3B and described above, the output of the second example encoder 200 after performing an encoding process is an encoded base stream 316 and one or more enhancement streams which preferably comprise an encoded level 1 stream 326 for a first level of enhancement and an encoded level 2 stream 346 for a further or second level of enhancement.

Third Example Encoder and Second Example Decoder – Predicted Residuals

Figure 4 shows a third example encoder 400 that is a variation of the first example encoder 100 of Figure 1. Corresponding reference numerals are used to refer to corresponding features from Figure 1 (i.e. where feature 1xx relates to feature 4xx in Figure 4). The example of Figure 4 shows in more detail how predicted residuals, e.g. a predicted average, may be applied as part of an up-sampling operation. Also, in Figure 4, the deblocking filter 130 is replaced by a more general configurable filter 430.

In Figure 4, a predicted residuals component 460 receives an input at a level 1 spatial resolution in the form of an output of a first summation component 432. This input comprises at least a portion of the reconstructed video at level 1 that is output by the first summation component 432. The predicted residuals component 460 also receives an input at a level 2 spatial resolution from the up-sampling component 434. The inputs may comprise a lower resolution element that is used to generate a plurality of higher resolution elements (e.g. a pixel that is then up-sampled to generate 4 pixels in a 2x2 block). The predicted residuals component 460 is configured to compute a modifier for the output of the up-sampling component 434 that is added to said output via a second summation component 462. The modifier may be computed to apply the predicted average processing that is described in detail in later examples. In particular, where an average delta is determined (e.g. a difference between a computed average coefficient and an average that is predicted from a lower level), the components of Figure 4 may be used to restore the average component outside of the level 2 encoding process 442. The output of the second summation component 462 is then used as the up-sampled input to the second subtraction component 436.

Figure 5A shows how a predicted residuals operation may be applied at a second example decoder 500. Like Figure 4, the second example decoder 500 may be considered is a variation of the first example decoder 200 of Figure 2. Corresponding reference numerals are used to refer to corresponding features from Figure 2 (i.e. where feature 2xx relates to feature 5xx in Figure 5). The example of Figure 5A shows in more detail how predicted residuals, e.g. a predicted average, may be applied at the decoder as part of an up-sampling operation. Also, in Figure 5A, the deblocking filter 232 is replaced by a more general configurable filter 532. It should be noted that the predicted residuals processing may be applied asymmetrically at the encoder and the decoder, e.g. the encoder need not be configured according to Figure 4 to allow decoding as set out in Figure 5A. For example, the encoder may applied a predicted average computation as described in US Patent 9,509,990, which is incorporated herein by reference.

The configuration of the second example decoder 500 is similar to the third example encoder 400 of Figure 4. A predicted residuals component 564 receives a first input from a first summation component 530, which represents a level 1 frame, and a second input from the up-sampling component 534, which represents an up-sampled version of the level 1 frame. The inputs may be received as a lower level element and a set of corresponding higher level elements. The predicted residuals component 564 uses the inputs to compute a modifier that is added to the output of the up-sampling component 534 by the second summation component 562. The modifier may correct for use of a predicted average, e.g. as described in US Patent 9,509,990 or computed by the third example encoder 400. The modified up-sampled output is then received by a third summation component 558 that performs the level 2 correction or enhancement as per previous examples.

The use of one or more of the predicted residuals components 460 and 564 may implement the "modified up-sampling" of other examples, where the modifier computed by the components and applied by respective summation components performs the "modification". These examples may provide for faster computation of predicted averages as the modifier is added in reconstructed video space as opposed to requiring conversion to coefficient space that represents transformed residuals (e.g. the modifier is applied to pixels of reconstructed video rather than applied in the A, H, V and D coefficient space of the transformed residuals).

Third Example Decoder – Sub-Operations and Temporal Prediction

Figures 5B and 5C illustrate respective variations of a third example decoder 580, 590. The variations of the third example decoder 580, 590 may be respective implemented to correspond to the variations of the third example encoder 300, 360 shown in Figures 3A and 3B. The third example decoder 580, 590 may be seen as an implementation of one or more of the first and second example encoders 200, 400 from Figures 2 and 4. As before, similar reference numerals are used where possible to refer to features that correspond to features in earlier examples.

Figures 5B and 5C show implementation examples of the decoding process described briefly above and illustrated in Figure 2. As is clearly identifiable, the decoding steps and components are expanded in more detail to provide an example of how decoding may be performed at each level. As with Figures 3A and 3B, Figure 5B illustrates a variation where temporal prediction is used only for the second level (i.e. level 2) and Figure 5C illustrates a variation where temporal prediction is used in both levels (i.e. levels 1 and 2). As before, further variations are envisaged (e.g. level 1 but not level 2), where the form of the configuration may be controlled using signalling information.

As shown in the examples of Figures 5A and 5C, in the decoding process, the decoder may parse headers 556 configure the decoder based on those headers. The headers may comprise one or more of global configuration data, picture (i.e. frame) configuration data, and assorted data blocks (e.g. relating to elements or groups of elements within a picture). In order to re-create the input video (e.g. the input video 102, 302 or 402 in previous examples), an example decoder such as the third example decoder may decode each of the encoded base stream 516, the first enhancement or encoded level 1 stream 526 and the second enhancement or encoded level 2 stream 546. The frames of the stream may be synchronised and then combined to derive the decoded video 560.

As shown in Figure 5B, the level 1 decoding component 528 may comprise a level 1 entropy decoding component 571, a level 1 inverse quantize component 572, and a level 1 inverse transform component 573. These may comprise decoding versions of the respective level 1 encoding components 325, 323 and 322 in Figures 3A and 3B. The level 2 decoding component 548 may comprise a level 2 entropy decoding component 581, a level 2 inverse quantize component 582, and a level 2 inverse transform component 583. These may comprise decoding versions of the respective level 2 encoding components 344, 343 and 341 in Figures 3A and 3B. In each decoding process, the enhancement streams may undergo the steps of entropy decoding, inverse quantization and inverse transform using the aforementioned components or operations to re-create a set of residuals.

In particular, in Figure 5B, an encoded base stream 516 is decoded by a base decoder 518 that is implemented as part of a base codec 584. It should be noted that the base and enhancement streams are typically encoded and decoded using different codecs, wherein the enhancement codec operates on residuals (i.e. may implement the level 1 and level 2 encoding and decoding components) and the base codec operates on video at a level 1 resolution. The video at the level 1 resolution may represent a lower resolution than the base codec normally operates at (e.g. a down-sampled signal in two dimensions may be a quarter of the size), which allows the base codec to operate at a high speed. This also marks a difference from SVC wherein each layer applies a common codec (AVC) and operates on video data rather than residual data. Even in SHVC, all spatial layers are configured to operate on a video in / video out manner where each video out represents a different playable video. In the present examples, the enhancement streams do not represent playable video in the conventional sense – the output of the level 1 and level 2 decoding components 528 and 548 (e.g. as received by the first summation component 530 and the second summation component 558) are "residual videos", i.e. consecutive frames of residuals for multiple colour planes rather than the colour planes themselves. This then allows a much greater bit rate saving as compared to SVC and SHVC, as the enhancement streams will often be 0 (as a quantized difference is often 0), where 0 values may be efficiently compressed using run-length coding. It is also to be noted that in the present examples, each coding unit of *n* by *n* elements (e.g. 2x2 or 4x4 blocks of pixels that may be flattened into one-dimensional arrays) does not depend on predictions that involve other coding units within the frame as per standard intra-processing in SVC and SHVC. As such, the encoding and decoding components in the enhancement streams may be applied in parallel to different coding units (e.g. different areas of a frame may be effectively processed in parallel), as unlike SVC and SHVC there is no need to wait for a decoded result of another coding unit to compute a subsequent coding unit. This means the enhancement codec may be implemented extremely efficiently on parallel processors such as common graphic processing units in computing devices (including mobile computing devices). This parallelism is not possible with the high complexity processing of SVC and SHVC.

Returning to Figure 5B, as in previous examples an optional filter such as deblocking filter 532 may be applied to the output of the level 1 decoding component 528 to remove blocking or other artefacts and the output of the filter is received by the first summation component 530 where it is added to the output of the base codec (i.e. the decoded base stream). Note that the output of the base codec may resemble a low resolution video as decoded by a conventional codec but the level 1 decoding output is a (filtered) first set of residuals. This is different from SVC and SHVC where this form of summation makes no sense, as each layer outputs a full video at a respective spatial resolution.

As in Figure 2, a modified up-sampling component 587 receives a corrected reconstruction of the video at level 1 that is output by the first summation component 530 and up-samples this to generate an up-sampled reconstruction. The modified up-sampling component 587 may apply the modified up-sampling illustrated in Figure 4. In other examples, the up-sampling may not be modified, e.g. if a predicted average is not being used or is being applied in the manner described in US Patent 9,509,990.

In Figure 5B, temporal prediction is applied during the level 2 decoding. In the example of Figure 5B, the temporal prediction is controlled by temporal prediction component 585. In this variation, control information for the temporal prediction is extracted from the encoded level 2 stream 546, as indicated by the arrow from the stream to the temporal prediction component 585. In other implementations, such as those shown in Figures 5A and 5C, control information for the temporal prediction may be sent separately from the encoded level 2 stream 546, e.g. in the headers 556. The temporal prediction component 585 controls the use of level 2 temporal buffer 550, e.g. may determine a temporal mode and control temporal refresh as described with reference to later examples. The contents of the temporal buffer 550 may be updated based on data for a previous frame of residuals. When the temporal buffer 550 is applied, the contents of the buffer are added to the second set of residuals. In Figure 5B, the contents of the temporal buffer 550 are added to the output of the level 2 decoding component 548 at a third summation component 594. In other examples, the contents of the temporal buffer may represent any set of intermediate decoding data and as such the third summation component 586 may be moved appropriated to apply the contents of the buffer at an appropriate stage (e.g. if the temporal buffer is applied at the dequantized coefficient stage, the third summation component 586 may be located before the inverse transform component 583). The temporal-corrected second set of residuals are then combined with the output of the up-sampling component 587 by the second summation component 558 to generate decoded video 560. The decoded video is at a level 2 spatial resolution, which may be higher than a level 1 spatial resolution. The second set of residuals apply a correction to the (viewable) up-sampled reconstructed video, where the correction adds back in fine detail and improves the sharpness of lines and features.

Figure 5C shows a variation 590 of the third example decoder. In this case, temporal prediction control data is received by a temporal prediction component 585 from headers 556. The temporal prediction component 585 controls both the level 1 and level 2 temporal prediction, but in other examples separate control components may be provided for both levels if desired. Figure 5C shows how the reconstructed second set of residuals that are input to the second summation component 558 may be fed back to be stored in the level 2 temporal buffer for a next frame (the feedback is omitted from Figure 5B for clarity). A level 1 temporal buffer 591 is also shown that operates in a similar manner to the level 2 temporal buffer 550 described above and the feedback loop for the buffer is shown in this Figure. The contents of the level 1 temporal buffer 591 are added into the level 1 residual processing pipeline via a fourth summation component 595. Again, the position of this fourth summation component 595 may vary along the level 1 residual processing pipeline depending on where the temporal prediction is applied (e.g. if it is applied in transformed coefficient space, it may be located before the level 1 inverse transform component 573.

Figure 5C shows two ways in which temporal control information may be signalled to the decoder. A first way is via headers 556 as described above. A second way, which may be used as an alternative or additional signalling pathway is via data encoded within the residuals themselves. Figure 5C shows a case whereby data 592 may be encoded into an HH transformed coefficient and so may be extracted following entropy decoding by the entropy decoding component 581. This data may be extracted from the level 2 residual processing pipeline and passed to the temporal prediction component 585.

In general, the enhancement encoding and/or decoding components described herein are low complexity (e.g. as compared to schemes such as SVC and SHVC) and may be implemented in a flexible modular manner. Additional filtering and other components may be inserted into the processing pipelines as determined by required implementations. The level 1 and level 2 components may be implemented as copies or different versions of common operations, which further reduces complexity. The base codec may be operated as a separate modular black-box, and so different codecs may be used depending on the implementation.

The data processing pipelines described herein may be implemented as a series of nested loops over the dimensions of the data. Subtractions and additions may be performed at a plane level (e.g. for each of a set of colour planes for a frame) or using multi-dimensional arrays (e.g. X by Y by C arrays where C is a number of colour channels such as YUV or RGB). In certain cases, the components may be configured to operate on *n* by *n* coding units (e.g. 2x2 or 4x4), and as such may be applied on parallel on the coding units for a frame. For example, a colour plane of a frame of input video may be decomposed into a plurality of coding units that cover the area of the frame. This may create multiple small one- or two-dimension arrays (e.g. 2x2 or 4x1 arrays or 4x4 or 16x1 arrays), where the components are applied to these arrays. As such, reference to a set of residuals may include a reference to a set of small one- or two-dimension arrays where each array comprises integer element values of a configured bit depth.

Each enhancement stream or both enhancement streams may be encapsulated into one or more enhancement bitstreams using a set of Network Abstraction Layer Units (NALUs). The NALUs are meant to encapsulate the enhancement bitstream in order to apply the enhancement to the correct base reconstructed frame. The NALU may for example contain a reference index to the NALU containing the base decoder reconstructed frame bitstream to which the enhancement has to be applied. In this way, the enhancement can be synchronised to the base stream and the frames of each bitstream combined to produce the decoded output video (i.e. the residuals of each frame of enhancement level are combined with the frame of the base decoded stream). A group of pictures may represent multiple NALUs.

Further Description of Processing Components

It was noted above how a set of processing components or tools may be applied to each of the enhancement streams (or the input video) throughout encoding and/or decoding. These processing components may be applied as modular components. They may be implemented in computer program code, i.e. as executed by one or more processors, and/or configured as dedicated hardware circuitry, e.g. as separate or combined Field Programmable Gate Arrays (FPGAs) or Application Specific Integrated Circuits (ASICs). The computer program code may comprise firmware for an embedded device or part of a codec that is used by an operating system to provide video rendering services. The following provides a brief summary each of the tools and their functionality within the overall process as illustrated in Figures 1 to 5C.

Down-sampling: The down-sampling process is applied by a down-sampling component in the examples (e.g. 104, 304 and 404). The down-sampling process is applied to the input video to produce a down-sampled video to be encoded by a base codec. The down-sampling can be done either in both vertical and horizontal directions, or alternatively only in the horizontal direction. A down-sampling component may further be described as a down-scaler.

Level-1 (L-1) encoding: The input to this component, which is shown as 122 in Figure 1 comprises the first set of residuals obtained by taking the difference between the decoded output of the base codec and the down-sampled video. The first set of residuals are then transformed, quantized and encoded as further described below.

Transform: In certain examples, there are two types of transforms that could be used in by the transform components (e.g. transform components 122, 322, and/or 341). The transform may be a directional decomposition. The transform may act to decorrelate the residual values in a coding unit (e.g. a small *n* by *n* block of elements). A transform may be applied as a matrix transformation, e.g. a matrix multiplication applied to a flattened array representing the coding unit.

In one case, the two types of transformation may correspond to two different sizes of transformation kernel. The size of the coding unit may thus be set based on the size of the transformation kernel. A first transform has a 2x2 kernel which is applied to a 2x2 block of residuals. The resulting coefficients are as follows:

$$\begin{pmatrix} C_{00} \\ C_{01} \\ C_{10} \\ C_{11} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} R_{00} \\ R_{01} \\ R_{10} \\ R_{11} \end{pmatrix}$$

A second transform has a 4x4 kernel which is applied to a 4x4 block of residuals. The resulting coefficients are as follows:

$$\begin{pmatrix} C_{00} \\ C_{01} \\ C_{02} \\ C_{03} \\ C_{10} \\ C_{11} \\ C_{12} \\ C_{13} \\ C_{20} \\ C_{21} \\ C_{22} \\ C_{23} \\ C_{30} \\ C_{31} \\ C_{32} \\ C_{33} \end{pmatrix} =$$

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \\
1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\
1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\
1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & 1 & 1 & -1 & -1 \\
1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & 1 & -1 & 1 & -1 \\
1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1
\end{pmatrix}
\begin{pmatrix} R_{00} \\ R_{01} \\ R_{02} \\ R_{03} \\ R_{10} \\ R_{11} \\ R_{12} \\ R_{13} \\ R_{20} \\ R_{21} \\ R_{22} \\ R_{23} \\ R_{30} \\ R_{31} \\ R_{32} \\ R_{33} \end{pmatrix}$$

These transformation matrices may comprise integer values in the range {-1, 1} or {-1, 0, 1}. This may simplify computations and allow for fast hardware implementations using addition and subtraction. The transformation matrices may comprise Hadamard matrices that have advantageous properties, such as orthogonal rows and being self-inverse (i.e. the inverse transformation is the same as the forward transformation). If a Hadamard matrix is used an inverse transformation may be referred to as a transformation as the same matrix may be used for both forward and inverse transformations.

In certain cases, the transformation may include the application of a predicted residuals computation (i.e. the use of a predicted average as described in more detail with reference to later examples).

Quantization: A set of transformed residuals (referred to herein as "coefficients") are quantized using a quantize component such as components 323 or 343). An inverse quantize component, such as components 327, 364, 372, 572 and 582 may reconstruct a version of a value pre-quantization by multiplying the quantized value by a defined quantization factor. The coefficients may be quantized using a linear quantizer. The linear quantizer may use a dead zone of variable size. The linear quantizer may use a dead zone of different size vs. the quantization step and a non-centred dequantization offset. These variations are described in more detail with reference to later examples.

Entropy coding: A set of quantized coefficients may be encoded using an entropy coder such as components 325 or 344. There are two schemes of entropy coding. In a first scheme, the quantized coefficients are encoded using a Run-Length-Encoder (RLE). In a second scheme, the quantized coefficients are first encoded using RLE, then the encoded output is processed using a Huffman Encoder.

Residual mode (RM) selection: If a residual (filtering) mode (RM) has been selected, the first set of residuals (i.e. level 1) may be further ranked and selected in order to determine which residuals should be transformed, quantized and encoded. Residual filtering may be performed by one or more of the components 321 or 340, e.g. under control of a control component such as 150 or 350. Filtering of residuals may be performed anywhere in the residual processing pipelines but preferably it is preformed prior to entropy encoding.

Temporal selection mode: If a temporal selection mode is selected, e.g. by a component such as 152 or 352, the encoder may modify the coefficients (i.e. the transformed residuals or data derived from these) by subtracting the corresponding coefficients derived from a temporal buffer, such as 345 or 361. This may implement temporal prediction as described below. The decoder may then modify the coefficients by adding the corresponding coefficients derived from a temporal buffer, such as one of components 230, 250, 530, 550 or 591.

Level-1 (L-1) decoding: This is shown as components 228 and 528. The input to this tool comprises the encoded level 1 stream 226 or 526 (i.e. L-1 encoded residuals), which are passed through an entropy decoder (such as 571), a de-quantizer (such as 572) and an inverse transform module (such as 573). The operations performed by these modules are the inverse operations performed by the modules described above. If the temporal selection mode has been selected, the residuals may be in part predicted from co-located residuals from a temporal buffer.

Deblocking and residual filters: In certain cases, if a 4x4 transform is used, the decoded residuals may be fed to a filter module or deblocking filter such as 130, 232, 330 or 535. The deblocking operates on each block of inversely transformed residuals by applying a mask whose weights can be specified. The general structure of the mask is as follows:

$$\begin{matrix} \alpha & \beta & \beta & \alpha \\ \beta & 1 & 1 & \beta \\ \beta & 1 & 1 & \beta \\ \alpha & \beta & \beta & \alpha \end{matrix}$$

where $0 \leq \alpha \leq 1$ and $0 \leq \beta \leq 1$. The weights may be specified within control signalling associated with the bitstream or may be retrieved from a local memory.

Up-sampling: The combination of the decoded (and filtered or deblocked, if applicable) first set of (L-1) residuals and base decoded video is up-sampled in order to generate an up-sampled reconstructed video. The up-sampling may be performed as described with respect to up-sampling components 134, 234, 334, 434, 534 or 587. Examples of possible up-sampling operations are described in more details below. The up-sampling method may be selectable and signalled in the bytestream. It should be noted that in examples herein, the term "bytestream" or an alternative term such as stream, bitstream or NALU stream may be used as appropriate.

Level-2 (L-2) encoding: This is represented as components 142, 442 and 548. The input to this encoding operation comprises the second set of (L-2) residuals obtained by taking the difference between the up-sampled reconstructed video and the input video. The second set of (L-2) residuals are then transformed, quantized and encoded as further described herein. The transform, quantization and encoding are performed in the same manner as described in relation to L-1 encoding. If a residual filtering mode has been selected, the second set of residuals are further ranked and selected in order to determine which residuals should be transformed and encoded.

Predicted coefficient (or predicted average) mode: If the predicted coefficient mode is selected, the encoder may modify the transformed coefficient C00, which is also referred to herein as A, i.e. an average value (which may be Ax for a 4x4 transform as described in more detail below). If the 2x2 transform is used, C00 may be modified by subtracting the value of the up-sampled residual which the transformed block of residuals is predicted from. If the 4x4 transform is used, C00 may be modified by subtracting the average value of the four up-sampled residuals which the transformed block of residuals is predicted from. The predicted coefficient mode may be implemented at the decoder using the modified up-sampling as described herein.

Level-2 (L-2) decoding: This is shown as components 248 and 548. The input to this decoding comprises the encoded second set of (L-2) residuals. The decoding process of the second set of residuals involves an entropy decoder (e.g. 581), a de-quantizer (e.g. 582) and an inverse transform module (e.g. 583). The operations performed by these components are the inverse operations performed by the encoding components as described above. If the temporal selection mode has been selected, the residuals may be in part predicted from co-located residuals from a temporal buffer.

Modified up-sampling: The modified up-sampling process comprises two steps, the second depending on a signalling received by the decoder. In a first step, the combination of the decoded (and deblocked, if applicable) first set of (L-1) residuals and base decoded video (L-1 reconstructed video) is up-sampled to generate an up-sampled reconstructed video. If the predicted coefficient mode has been selected, then a second step is implemented. In particular, the value of the element in the L-1 reconstructed value from which a 2x2 block in the up-sampled reconstructed video was derived is added to said 2x2 block in the up-sampled reconstructed video. In general, the modified up-sampling may be based on the up-sampled reconstructed video and on the pre-up-sampling reconstructed lower resolution video as described with reference to Figure 4.

Dithering: In certain examples, a last stage of dithering may be selectively applied to the decoded video 260 or 560 in Figures 2 and 5A to 5C. Dithering may comprise the application of small levels of noise to the decoded video. Dithering may be applied by adding random or pseudo-random numbers within a defined range to the decoded video. The defined range may be configured based on local and/or signalled parameters. The defined range may be based on a defined minimum and maximum value, and/or a defined scaling factor (e.g. for an output of a random number generator within a specific range). Dithering may reduce the visual appearance of quantization artefacts as is known in the art.

Example of 4x4 Residual Coding Unit and Tiles

Figure 6A shows an example 600 of a set of residuals 610 arranged in a 4x4 coding unit 620. There are thus 16 residual elements. The coding unit 620 may comprise an N by N array R of residuals with elements R[x][y]. For a 2x2 coding unit, there may be 4 residual elements. The transform may be applied to coding units as shown.

Figure 6B shows how a plurality of coding units 640 may be arranged into a set of tiles 650. The set of tiles may collectively cover the complete area of a picture or frame. In the example, of Figure 6B, a tile is made up of an 8x8 array of coding units. If the coding units are 4x4, this means that each tile has 32x32 elements; if the coding units are 2x2, this means that each tile has 16x16 elements.

Example Picture Formats

Figures 7A to 7C show a number of ways in which colour components may be organised to form a picture or frame within a video. In examples, frames of an input video 102, 302, 402 may be referred to as source pictures and a decoded output video 260, 560 may be referred to as decoded pictures. The encoding process as implemented by the encoder may general a bitstream as described in examples herein that is transmitted to, and received by, the decoding process as implemented by a decoder. The bitstream may comprise a combined bitstream that is generated from at least the encoded base stream, the encoded level 1 stream, the encoded level 2 stream and the headers (e.g. as described in examples herein). A video source that is represented by the bitstream may thus be seen as a sequence of pictures in decoding order.

In certain examples, the source and decoded pictures are each comprised of one or more sample arrays. These arrays may comprise: luma only (monochrome) components (e.g. Y); luma and two chroma components (e.g. YCbCr or YCgCo); Green, blue, and red components (e.g. GBR or RGB); or other arrays representing other unspecified monochrome or tri-stimulus colour samplings (for example, YZX, also known as XYZ). Certain examples described herein are presented with reference to luma and chroma arrays (e.g. Y, Cb and Cr arrays); however, those skilled in the art will understand that these examples may be suitably configured to operate with any known or future colour representation method.

In certain examples, a chroma format sampling structure may be specified through *chroma_sampling_type* (e.g. this may be signalled to the decoder). Different sampling formats may have different relations between the different colour components. For example: in 4:2:0 sampling, each of the two chroma arrays has half the height and half the width of the luma array; in 4:2:2 sampling, each of the two chroma arrays has the same height and half the width of the luma array; and in 4:4:4 sampling, each of the two chroma arrays has the same height and width as the luma array. In monochrome sampling there is only one sample array, which is nominally considered the luma array. The number of bits necessary for the representation of each of the samples in the luma and chroma arrays in a video sequence may be in the range of 8 to 16, inclusive, and the number of bits used in the luma array may differ from the number of bits used in the chroma arrays.

Figures 7A to 7C show different sampling types that may be represented by different values of the variable *chroma_sampling_type*. When the value of *chroma_sampling_type* is equal to 0, the nominal vertical and horizontal relative locations of luma samples 710 and chroma samples 720 in pictures are shown in Figure 7A. When the value of *chroma_sampling_type* is equal to 1, the chroma samples 720 are co-sited with the corresponding luma samples 710 and the nominal locations in a picture are as shown in Figure 7B. When the value of *chroma_sampling_type* is equal to 2, all array samples 710, 720 are co-sited for all cases of pictures and the nominal locations in a picture are as shown in Figure 7C. In these cases, the variables *SubWidthC* and *SubHeightC* may indicate how the chroma samples are shifted:

chroma_sampling_type	**Chroma format**	**SubWidthC**	**SubHeightC**
0	Monochrome	1	1
1	4:2:0	2	2
2	4:2:2	2	1
3	4:4:4	1	1

Example Bitstream Processing

Figure 8 shows an example method 800 that may be used to process a bitstream that has been encoded using the example encoders or encoding processes described herein. The method 800 may be implemented by an example decoder, such as 200 or 500 in Figures 2 and 5. The method 800 shows an example flow which facilitates separation of an enhancement bitstream.

At block 802, the method 800 comprises receiving an input bitstream 802. At block 804, a NALU start is identified within the received bitstream. This then allows identification of an entry point at block 806. The entry point may indicate which version of a decoding process should be used to decode the bitstream. Next, at block 808, a payload enhancement configuration is determined. The payload enhancement configuration may indicate certain parameters of the payload. The payload enhancement configuration may be signalled once per stream. Optionally, the payload enhancement configuration may be signalled multiple per group of pictures or for each NALU. The payload enhancement configuration may be used to extract payload metadata at block 810.

At block 812, a start of a group of pictures (GOP) is identified. Although the term group of pictures is used it will be understood that this term is used to refer to a corresponding structure to that of the base stream but not to define a particular structure on the enhancement stream. That is, enhancement streams may not have a GOP structure in the strict sense and strict compliance with GOP structures of the art is not required. If payload metadata is included, it may be included after the payload enhancement configuration and before the set of groups of pictures. Payload metadata may for example include HDR information. Following block 812, a GOP may be retrieved. At block 814, if the NALU relates to a first bitstream frame, the method may further comprise retrieving a payload global configuration at block 816. The payload global configuration may indicate parameters of the decoding process, for example, the payload global configuration may indicate if a predicted residual mode or temporal prediction mode was enabled in the encoder (and should be enabled at the decoder), thus the payload global configuration may indicate if a mode should be used in the decoding method. The payload global configuration may be retrieved once for each GOP. At block 818, the method 800 may further comprise retrieving a set of payload decoder control parameters which indicate to the decoder parameters to be enabled during decoding, such as dithering or up-sampling parameters. The payload decoder control parameters may be retrieved for each GOP. At block 820, the method 800 comprises retrieving a payload picture configuration from the bitstream. The payload picture configuration may comprise parameters relating to each picture or frame, for example, quantization parameters such as a step width. The payload picture configuration may be retrieved once for each NALU (that is, once for each picture or frame). At block 822, the method 800 may then further comprise retrieving a payload of encoded data which may comprise encoded data of each frame. The payload of encoded data may be signalled once for each NALU (that is, once for each picture or frame). The payload of encoded data may comprise a surface, plane or layer of data which may be separated into chunks as described with reference to Figures 9A, as well as the examples of Figures 21A and 21B. After the payload of encoded data is retrieved, the NALU may end at block 824.

If the GOP also ends, the method may continue to retrieve a new NALU for a new GOP. If the NALU is not the first bitstream frame (as the case here), then the NALU may then, optionally, retrieve an entry point (i.e. an indication of a software version to be used for decoding). The method may then retrieve a payload global configuration, payload decoder control parameters and payload picture configuration. The method may then retrieve a payload of encoded data. The NALU will then end.

If at block 814, the NALU does not relate to a first bitstream frame, then blocks 828 to 838 may be performed. Optional block 828 may be similar to block 806. Blocks 830 to 838 may be performed in a similar manner to blocks 816 to 824.

At blocks 840 and 842, after each NALU has ended, if the GOP has not ended, the method 800 may comprise retrieving a new NALU from the stream at block 844. For each second and subsequent NALU of each GOP, the method 800 may optionally retrieve an entry point indication at block 846, in a similar manner to blocks 806 and 828. The method 800 may then comprise retrieving payload picture configuration parameters at block 848 and a payload of encoded data for the NALU at block 850. Blocks 848 to 852 may thus be performed in a similar manner to blocks 820 to 824 and blocks 834 to 838. The payload encoded data may comprise tile data.

As above, if the NALU is not the last NALU for the GOP, the method may comprise retrieving a further NALU (e.g. looping around to block 844). If the NALU is the last NALU in the GOP, the method 800 may proceed to block 854. If there are further GOPs, the method may loop around to block 812 and comprise retrieving a further GOP and performing blocks 814 onwards as previously described. Once all GOPs have been retrieved the bitstream ends at block 856.

Example Form of Encoded Payload Data

Figure 9A shows how encoded data 900 within an encoded bitstream may be separated into chunks. More particularly, Figure 9A shows an example data structure for a bitstream generated by an enhancement encoder (e.g. level 1 and level 2 encoded data). A plurality of planes 910 are shown (of number *nPlanes*). Each plane relates to a particular colour component. In Figure 9A, an example with YUV colour planes is shown (e.g. where a frame of input video has three colour channels, i.e. three values for every pixel). In the examples, the planes are encoded separately.

The data for each plane is further organised into a number of levels (*nLevels*). In Figure 9A there are two levels, relating to each of enhancement levels 1 and 2. The data for each level is then further organised as a number of layers (*nLayers*). These layers are separate from the base and enhancement layers; in this case, they refer to data for each of the coefficient groups that result from the transform. For example, an 2x2 transform results in four different coefficients that are then quantized and entropy encoded and an 4x4 transform results in sixteen different coefficients that are then likewise quantized and entropy encoded. In these cases, there are thus respectively 4 and 16 layers, where each layer represents the data associated with each different coefficient. In cases, where the coefficients are referred to as A, H, V and D coefficients then the layers may be seen as A, H, V and D layers. In certain examples, these "layers" are also referred to as "surfaces", as they may be viewed as a "frame" of coefficients in a similar manner to a set of two-dimensional arrays for a set of colour components.

The data for the set of layers may be considered as "chunks". As such each payload may be seen as ordered hierarchically into chunks. That is, each payload is grouped into planes, then within each plane each level is grouped into layers and each layer comprises a set of chunks for that layer. A level represents each level of enhancement (first or further) and layer represents a set of transform coefficients. In any decoding process, the method may comprise retrieving chunks for two level of enhancement for each plane. The method may comprise retrieving 4 or 16 layers for each level, depending on size of transform that is used. Thus, each payload is ordered into a set of chunks for all layers in each level and then the set of chunks for all layers in the next level of the plane. Then the payload comprises the set of chunks for the layers of the first level of the next plane and so on.

As such, in the encoding and decoding methods described herein, the pictures of a video may be partitioned, e.g. into a hierarchical structure with a specified organisation.

Each picture may be composed of three different planes, organized in a hierarchical structure. A decoding process may seek to obtain a set of decoded base picture planes and a set of residuals planes. A decoded base picture corresponds to the decoded output of a base decoder. The base decoder may be a known or legacy decoder, and as such the bitstream syntax and decoding process for the base decoder may be determined based on the base decoder that is used. In contrast, the residuals planes are new to the enhancement layer and may be partitioned as described herein. A "residual plane" may comprise a set of residuals associated with a particular colour component. For example, although the planes 910 are shown as relating to YUV planes of an input video, it should be noted the data 920 does not comprise YUV values, e.g. as for a comparative coding technology. Rather, the data 920 comprises encoded residuals that were derived from data from each of the YUV planes.

In certain examples, a residuals plane may be divided into coding units whose size depends on the size of the transform used. For example, a coding unit may have a dimension of 2x2 if a 2x2 directional decomposition transform is used or a dimension of 4x4 if a 4x4 directional decomposition transform is used. The decoding process may comprise outputting one or more set of residuals surfaces, that is one or more sets of collections of residuals. For example, these may be output by the level 1 decoding component 228 and the level 2 decoding component 248 in Figure 2. A first set of residual surfaces may provide a first level of enhancement. A second set of residual surfaces may be a further level of enhancement. Each set of residual surfaces may combine, individually or collectively, with a reconstructed picture derived from a base decoder, e.g. as illustrated in the example decoder 200 of Figure 2.

Example Up-sampling Approaches

Figures 9B to 9J and the description below relate to possible up-sampling approaches that may be used when implementing the up-sampling components as described in examples herein, e.g. up-sampling components 134, 234, 334, 434, 534 or 587 in Figures 1 to 5C.

Figs. 9B and 9C show two examples of how a frame to be up-sampled may be divided. Reference to a frame may be taken as reference to one or more planes of data, e.g. in YUV format. Each frame to be up-sampled, called a source frame 910, is divided into two major parts, namely a centre area 910C, and a border area 910B. Fig. 9B shows an example arrangement for bilinear and bicubic up-sampling methods. In Fig. 9B, the border area 910B consists of four segments, namely top segment 910BT, left segment 910BL, right segment 910BR, and bottom segment 910BB. Fig. 9C shows an example arrangement for a nearest up-sampling method. In Fig. 9C, the border area 910B consists of 2 segments; right segment 910BR and bottom segment 910BB. In both examples, the segments may be defined by a border-size parameter (BS), e.g. which sets a width of the segment (i.e. a length that the segment extends into the source frame from an edge of the frame). The border-size may be set to be 2 pixels for bilinear and bicubic up-sampling methods or 1 pixel for the nearest method.

In use, determining whether a source frame pixel is located within a particular segment may be performed based on a set of defined pixel indices (e.g. in x and y directions). Performing differential up-sampling based on whether a source frame pixel is within a centre area 910C or a border area 910B may help avoid border effects that may be introduced due to the discontinuity at the source frame edges.

Nearest up-sampling

Figure 9D provides an overview of how a frame is up-sampled using a nearest up-sampling method. In Figure 9D, a source frame 920 is up-sampled to become destination frame 922. The nearest up-sampling method up-samples by copying a current source pixel 928 onto a 2x2 destination grid 924 of destination pixels, e.g. as indicated by arrows 925. Centre and edge pixels are respectively shown as 926 and 927. The destination pixel positions are calculated by doubling the index of the source pixel 928 on both axes and progressively adding +1 to each axis to extend the range to cover 4 pixels as shown on the right-hand side of Figure 9D. For example, the value of source pixel 928 with index location (x=6, y=6) is copied to destination grid 924 comprising pixels with index locations (12, 12) (13, 12) (12, 13) and (13, 13). Each pixel in the destination grid 924 takes the value of the source pixel 928.

The nearest method of up-sampling provides enables fast implementations that may be preferable for embedded devices with limited processor resources. However, the nearest method has a disadvantage that blocking, or “pixilation”, artefacts may need to be corrected by the level 2 residuals (e.g. that result in more non-zero residual values that require more bits for transmission following entropy encoding). In certain examples described below, bilinear and bicubic up-sampling may result in a set of level 2 residuals that can be more efficiently encoded, e.g. that require fewer bits following quantization and entropy encoding. For example, bilinear and bicubic up-sampling may generate an up-sampled output that more accurately matches the input signal, leading to smaller level 2 residual values.

Bilinear up-sampling

Figures 9E, 9F and 9G illustrate a bilinear up-sampling method. The bilinear up-sampling method can be divided into three main steps. The first step involves constructing a 2x2 source grid 930 of source pixels 932 in the source frame. The second step involves performing a bilinear interpolation. The third step involves writing the interpolation result to destination pixels 936 in the destination frame.

Bilinear up-sampling - Step 1: Source pixel grid

Fig. 9E illustrates a construction example of the 2x2 source grid 930 (which may also be called a bilinear grid). The 2x2 source grid 930 is used instead of a source pixel 932 because the bilinear up-sampling method performs up-sampling by considering the values of the nearest 3 pixels to a base pixel 932B, i.e. the nearest 3 pixels falling within the 2x2 source grid 930. In this example, the base pixel 932B is at the bottom right of the 2x2 source grid 930, but other positions are possible. During the bilinear up method the 2x2 source grid 930 may be determined for multiple source frame pixels, so as to iteratively determine destination frame pixel values for the whole destination frame. The base pixel 932B location is used to determine an address of a destination frame pixel.

Bilinear up-sampling - Step 2: Bilinear interpolation

Fig. 9F illustrates a bilinear coefficient derivation. In this example, the bilinear interpolation is a weighted summation of the values of the four pixels in the 2x2 source grid 930. The weighted summation is used as the pixel value of a destination pixel 936 being calculated. The particular weights employed are dependent on the position of the particular destination pixel 936 in a 2x2 destination grid 935. In this example, the bilinear interpolation applies weights to each source pixel 932 in the 2x2 source grid 930, using the position of the destination pixel 936 in the 2x2 destination grid 935. For example, if calculating the value for the top left destination pixel (shown as 936/936B in Fig. 9F), then the top left source pixel value will receive the largest weighting coefficient 934 (e.g. weighting factor 9) while the bottom right pixel value (diagonally opposite) will receive the smallest weighting coefficient (e.g. weighting factor 1), and the remaining two pixel values will receive an intermediate weighting coefficient (e.g. weighting factor 3). This is visualized in Fig. 9F with the weightings shown in the 2x2 source grid 930.

For the pixel on the right of 936/936B within the 2x2 destination grid 935, the weightings applied to the weighted summation would change as follows: the top right source pixel value will receive the largest weighting coefficient (e.g. weighting factor 9) while the bottom left pixel value (diagonally opposite) will receive the smallest weighting coefficient (e.g. weighting factor 1), and the remaining two pixel values will receive an intermediate weighting coefficient (e.g. weighting factor 3).

In Figure 9F, four destination pixels are computed for the base pixel 932B based on the 2x2 source grid 930 but each destination pixel is determined using a different set of weights. These weights may be thought of as an up-sampling kernel. In this way, there may be four different sets of four weighted values that are applied to the original pixel values within the 2x2 source grid 930 to generate the 2x2 destination grid 935 for the base pixel 932B. After the four destination pixel values are determined, another base pixel is selected with a different source grid and the process begins again to determine the next four destination pixel values. This may be iteratively repeated until pixel values for the whole destination (e.g. up-sampled) frame are determined. The next section describes in more detail the mapping of these interpolated pixel values from the source frame to the destination frame.

Bilinear up-sampling - Step 3: Destination pixels

Figure 9G shows an overview of the bilinear up-sampling method comprising a source frame 940, a destination frame 942, an interpolation module 944, a plurality of 2x2 source grids 930 (a,b,c,d,h,j), and a plurality of 2x2 destination grids 935 (d,e,h,k). The source frame 940 and destination frame 942 have indexes starting from 0 on each column and row for pixel addressing (although other indexing schemes may be used).

In general, each of the weighted averages generated from each 2x2 source grid 930 is mapped to a corresponding destination pixel 936 in the corresponding 2x2 destination grid 935. The mapping uses the source base pixel 932B of each 2x2 source grid 930 to map to a corresponding destination base pixel 936B of the corresponding 2x2 destination grid 942, unlike the nearest sampling method. The destination base pixel 936B address is calculated from the equation (applied for both axes):

$$Dst\_base\_addr = (Src\_base\_address \times 2)-1$$

Also, the destination pixels have three corresponding destination sub-pixels 721S calculated from the equation:

$Dst\_sub\_addr = Dst\_base\_addr{+}1$ (for both axes)

And so, each 2x2 destination grid 935 generally comprises a destination base pixel 936B together with three destination sub pixels 936S, one each to the right, below, and diagonally down to the right of the destination base pixel, respectively. This is shown in Figure 9F. However, other configurations of destination grid and base pixel are possible.

The calculated destination base and sub addresses for destination pixels 936B and 936S respectively can be out of range on the destination frame 942. For example, pixel A (0, 0) on source frame 940 generates a destination base pixel address (-1, -1) for a 2x2 destination grid 935. Destination address (-1, -1) does not exist on the destination frame 942. When this occurs, writes to the destination frame 942 are ignored for these out of range values. This is expected to occur when up-sampling the border source frames. However, it should be noted that in this particular example one of the destination sub-pixel addresses (0, 0) is in range on the destination frame 942. The weighted average value of the 2x2 source grid 930 (i.e. based on the lower left pixel value taking the highest weighting) will be written to address (0, 0) on the destination frame 942. Similarly, pixel B (1, 0) on source frame 940 generates a destination base pixel address (1, -1) which is out of range because there is no -1 row. However, the destination sub-pixel addresses (1, 0) and (2, 0) are in range and the corresponding weighted sums are each entered into the corresponding addresses. Similar happens for pixel C, but only the two values on the column 0 are entered (i.e. addresses (0, 1) and (0, 2)). Pixel D at address (1, 1) of the source frame contributes a full 2x2 destination grid 935d based on the weighted averages of source grid 930d, as do pixels E, H and K, with 2x2 destination grids 935e, 935h, and 935k and corresponding source grids 930e, 930h and 930k illustrated in Figure 9G.

As will be understood, these equations usefully deal with the border area 910B and its associated segments, and ensure that when the centre 910C segment is up-sampled it will remain in the centre of the destination frame 942. Any pixel values that are determined twice using this approach, e.g. due to the manner in which the destination sub-pixels are determined, may be ignored or overwritten.

Furthermore, the ranges for border segments 910BR and 910BB are extended by +1 in order to fill all pixels in the destination frame. In other words, the source frame 940 is extrapolated to provide a new column of pixels in border segment 910BR (shown as index column number 8 in Figure 9G), and a new row of pixels in border segment 910BB (shown as index row number 8 in Figure 9G).

Cubic up-sampling

Figures 9H, 9I and 9J together illustrate a cubic up-sampling method, in particular, a bicubic method. The cubic up-sampling method of the present example may be divided into three main steps. The first step involves constructing a 4x4 source grid 962 of source pixels with a base pixel 964B positioned at the local index (2, 2) within the 4x4 source grid 815. The second step involves performing a bicubic interpolation. The third step involves writing the interpolation result to the destination pixels.

Cubic up-sampling - Step 1: Source pixel grid

Figure 9H shows a 4x4 source grid 962 construction on source frame 960 for an in-bound grid 962i and separately an out-of-bound grid 962o. In this example, "in-bound" refers to the fact that the grid covers source pixels that are within the source frame, e.g. the centre region 910C and the border regions 910B; "out-of-bound" refers to the fact that the grid includes locations that are outside of the source frame. The cubic up-sampling method is performed by using the 4x4 source grid 962 which is subsequently multiplied by a 4x4 kernel. This kernel may be called an up-sampling kernel. During the generation of the 4x4 source grid 962, any pixels which fall outside the frame limits of the source frame 960 (e.g. those shown in out of bounds grid 962o) are replaced with the value of the source pixels 964 the at the boundary of the source frame 960.

Cubic up-sampling - Step 2: Bicubic interpolation

The kernels used for the bicubic up-sampling process typically have a 4x4 coefficient grid. However, the relative position of the destination pixel with reference to the source pixel will yield a different coefficient set, and since the up-sampling is a factor of two in this example, there will be 4 sets of 4x4 kernels used in the up-sampling process. These sets are represented by a 4-dimensional grid of coefficients (2 x 2 x 4 x 4). For example, there will be one 4x4 kernel for each destination pixel in a 2x2 destination grid, that represents a single up-sampled source pixel 964B.

In one case, the bicubic coefficients may be calculated from a fixed set of parameters. In one case, this comprises a core parameter (bicubic parameter) and a set of spline creation parameters. In an example, a core parameter of -0.6 and four spline creation parameters of [1.25, 0.25, -0.75 & -1.75] may be used. An implementation of the filter may use fixed point computations within hardware devices.

Cubic up-sampling - Step 3: Destination pixels

Figure 9J shows an overview of the cubic up-sampling method comprising a source frame 972, a destination frame 980, an interpolation module 982, a 4x4 source grid 970, and a 2x2 destination grid 984. The source frame 972 and destination frame 980 have indexes starting from 0 on each column and row for pixel addressing (although other addressing schemes may be used).

Similarly to the bilinear method, the bicubic destination pixels have a base address calculated from the equation for both axes:

$Dst\_base\_addr = (Src\_base\_address \times 2)-1$

Also, the destination addresses are calculated from:

$Dst\_sub\_addr = Dst\_base\_addr+1$ (for both axes)

And so, as for the bilinear method, each 2x2 destination grid 984 generally comprises a destination base pixel together with three destination sub pixels, one each to the right, below, and diagonally down to the right of the destination base pixel, respectively. However, other configurations of destination grid and base pixel are possible.

Again, these equations ensure that when the centre segment is up-sampled it will remain in the centre of the destination frame. Furthermore, the ranges for border segments 510BR and 510BB are extended by +1 in order to fill all pixels in the destination frame 980 in the same way as described for the bilinear method. Any pixel values that are determined twice using this approach, e.g. due to the manner in which the destination sub-pixels are determined, may be ignored or overwritten. The calculated destination base and sub addresses can be out of range. When this occurs, writes to the destination frame are ignored for these out of range values. This is expected to occur when up-sampling the border area.

Example Entropy Encoding

Figures 10A to 10I illustrate different aspects of entropy encoding. These aspects may relate to an entropy encoding performed, for example, by entropy encoding components 325, 344 in Figures 3A and 3B and/or an entropy decoding performed, for example, by entropy decoding components 571, 581 in Figures 5B and 5C.

Example Entropy Encoding – Header Formats

Figures 10B to 10E illustrate a specific implementation of the header formats and how the code lengths may be written to a stream header depending on the amount of non-zero codes.

Example Entropy Encoding – RLE State Machine

Figure 10F shows a state machine 1050 that may be used be a run length decoder, such as run length decoder 1005 in Figure 10A.

<u>*Temporal Prediction and Signalling*</u>

Certain variations and implementation details of the temporal prediction will now be described, including certain aspects of temporal signalling.

In certain examples described herein, information from two of more frames of video that relate to different time samples may be used. This may be described as a temporal mode, e.g. as it relates to information from different times. Not all embodiments may make use of temporal aspects. Components for temporal prediction are shown in the examples of Figures 1 to 5C. As described herein, a step of encoding one or more sets of residuals may utilise a temporal buffer that is arranged to store information relating to a previous frame of video. In one case, a step of encoding a set of residuals may comprise deriving a set of temporal coefficients from the temporal buffer and using the retrieved set of temporal coefficients to modify a current set of coefficients. “Coefficients”, in these examples, may comprise transformed residuals, e.g. as defined with reference to one or more coding units of a frame of a video stream – approaches may be applied to both residuals and coefficients. In certain cases, the modifying may comprise subtracting the set of temporal coefficients from the current set of coefficients. This approach may be applied to multiple sets of coefficients, e.g. those relating to a level 1 stream and those relating to a level 2 stream. The modification of a current set of coefficients may be performed selectively, e.g. with reference to a coding unit within a frame of video data.

Temporal aspects may be applied at both the encoding and decoding stages. Use of a temporal buffer is shown in the encoder 300 of Figures 3A and 3B and in the decoder 580, 590 of Figures 5B and 5C. As described herein, prior to modifying a current set of coefficients, the current set of coefficients may be one or more of ranked and transformed. In one case, dequantized transformed coefficients — $dqC_{x,y,n-1}$ — from a previous encoded (n-1) frame at a corresponding position (e.g. a same position or mapped position) are used to predict the coefficients $C_{x,y,n}$ in a frame to be encoded (n). If a 4x4 transform is used, x, y may be in the range [0,3]; if a 2x2 transform is used x,y may be in the range [0,1]. Dequantized coefficients may be generated by an inverse quantize block or operation. For example, in Figure 3B, dequantized coefficients are generated by inverse quantize component 372.

In certain examples, there may be at least two temporal modes.

- A first temporal mode that does not use the temporal buffer or that uses the temporal buffer with all zero values. The first temporal mode may be seen as an intra-frame mode as it only uses information from within a current frame. In the first temporal mode, following any applied ranking and transformation, coefficients may be quantized without modification based on information from one or more previous frames.
- A second temporal mode that makes use of the temporal buffer, e.g. that uses a temporal buffer with possible non-zero values. The second temporal mode may be seen as an inter-frame mode as it uses information from outside a current frame, e.g. from multiple frames. In the second temporal mode, following any applied ranking and transformation, previous frame dequantized coefficients may be subtracted from the coefficients to be quantized — $C_{x,y,n,inter.} = C_{x,y,n} - dqC_{x,y,n-1}$.

In one case, a first temporal mode may be applied by performing a subtraction with a set of zeroed temporal coefficients. In another case, the subtraction may be performed selectively based on temporal signalling data. Figures 11A and 11B show example operations in the encoder for two respective temporal modes. A first example 1100 in

Figure 12A shows an example 1200 of temporal processing that may be performed at the encoder.

Figure 12B shows a corresponding example 1230, e.g. as implemented at a decoder Figure 12C shows an example 1250 of how temporal signalling information may be provided for a frame of residuals 1251

Figure 12D shows a representation 1260 of temporal signals for 4x4 transform size (e.g. a DDS transform).Figures 13A and 13B are two halves 1300, 1340 of a flow chart showing a method of temporal processing according to an example. The method of temporal processing may be performed at the encoder. The method of temporal processing may implement certain processes described above. The method of processing may be applied to the frame of residuals shown in Figure 12C.

Cloud Configuration

In certain examples, an encoder (or encoding process) may communicate with one or more remote devices. The encoder may be an encoder as shown in any one of Figures 1, 3A and 3B or described in any other of the examples.

Figure 14A shows an example 1400 of an encoder 1402 communicating across a network 1404.

Figure 14B shows that the encoder 1402 may send and/or receive configuration data 1406, 1408 to and/or from a remote control server 1412.

Figure 14C shows how an encoder 1432 (which may implement any of the described encoders including encoder 1402 in Figures 14A and 14B) may comprise a configuration interface 1434 that is configured to communicate over the network, e.g. with the remote control server 1412.

Using a cloud configuration as described herein may provide implementation advantages. For example, an encoder may be controlled remotely, e.g. based on network control systems and measurements. An encoder may also be upgraded to provide new functionality by upgrading firmware that provides the enhancement processing, with additional data, e.g. based on measurements or pre-processing being supplied by one or more remote data sources or control servers. This provides a flexible way to upgrade and control legacy hardware devices.

Residual Mode Selection

As described above, e.g. in relation to Figures 3A and 3B, certain examples may implement different residual processing modes. For example, in Figure 3A, a residual mode ranking component 350 controls residual mode selection components 321, 340 in each of the level 1 and level 2 enhancement streams; in Figure 3B, a residual mode selection component 350 controls residual mode ranking components 321, 340 in each of the level 1 and level 2 enhancement streams. In general, an encoder may comprise a residual mode control component that selects and implements a residual mode and residual mode implementation components that implements processing for a selected residual mode upon one or more enhancement streams.

In one example, once the residuals have been computed, the residuals may be processed to decide how the residuals are to be encoded and transmitted. As described earlier, here residuals are computed by comparing an original form of an image signal with a reconstructed form of an image signal. For example, in one case, residuals for a level 2 enhancement stream are determined by subtracting an output of the up-sampling (e.g. in Figures 1, 3A and 3B) from an original form of an image signal (e.g. the input video 120, 302 as indicated in the Figures). The input to the up-sampling may be said to be a reconstruction of a signal following a simulated decoding. In another case, residuals for an level 1 enhancement stream are determined by subtracting an image stream output by the base decoder from a down-sampled form of the original image signal (e.g. the output of the down-sampling component 104, 304 in Figures 1, 3A and 3B).

To process residuals, e.g. in a selected residual mode, the residuals may be categorized. For example, residuals may be categorized in order to select a residual mode. A categorization process of the residuals may be performed based, for example, on certain spatial and/or temporal characteristic of the input image.

In one example, the input image is processed to determine, for each element (e.g., a pixel or an area including multiple pixels) and/or group of elements whether that element and/or group of elements has certain spatial and/or temporal characteristics. For example, the element is measured against one or more thresholds in order to determine how to classify it against respective spatial and/or temporal characteristics. Spatial characteristics may include the level of spatial activity between specific elements or groups of elements (e.g., how many changes exists between neighbouring elements), or a level of contrast between specific elements and/or between groups of elements (e.g., how much a group of element differs from one or more other groups of elements). The spatial characteristics may be a measure of a change in a set of spatial directions (e.g. horizontal and/or vertical directions for a 2D planar image). Temporal characteristics may include temporal activity for a specific element and/or group of elements (e.g., how much an element and/or a group of elements differ between collocated elements and/or group of elements on one or more previous frames). The temporal characteristics may be a measure of a change in a temporal direction (e.g. along a time series). The characteristics may be determined per element and/or element group; this may be per pixel and/or per 2x2 or 4x4 residual block.

The categorization may associate a respective weight to each element and/or group of elements based on the spatial and/or temporal characteristics of the element and/or group of elements. The weight may be a normalized value between 0 and 1.

In one residual mode, a decision may be made as to whether to encode and transmit a given set of residuals. For example, in one residual mode, certain residuals (and/or residual blocks – such as the 2x2 or 4x4 blocks described herein) may be selectively forwarded along the level 1 and/or level 2 enhancement processing pipelines by the RM

L-x ranking components and/or the RM L-x selection components as shown in Figures 3A and 3B. Put another way, different residual modes may have different residual processing in the level 1 and/or level 2 encoding components 122, 142 in Figure 1. For example, in one residual mode, certain residuals may not be forwarded for further level 1 and/or level 2 encoding, e.g. may not be transformed, quantized and entropy encoded. In one case, certain residuals may not be forwarded by setting the residual value to 0 and/or by setting a particular control flag relating to the residual or a group that includes the residual.

In one residual mode, a binary weight of 0 or 1 may be applied to residuals, e.g. by the components discussed above. This may correspond to a mode where selective residual processing is "on". In this mode, a weight of 0 may correspond to "ignoring" certain residuals, e.g. not forwarding them for further processing in an enhancement pipeline. In another residual mode, there may be no weighting (or the weight may be set to 1 for all residuals); this may correspond to a mode where selective residual processing is "off". In yet another residual mode, a normalised weight of 0 to 1 may be applied to a residual or group of residuals. This may indicate an importance or "usefulness" weight for reconstructing a video signal at the decoder, e.g. where 1 indicates that the residual has a normal use and values below 1 reduce the importance of the residual. In other cases, the normalised weight may be in another range, e.g. a range of 0 to 2 may give prominence to certain residuals that have a weight greater than 1.

In the residual modes described above, the residual and/or group of residuals may be multiplied by an assigned weight, where the weight may be assigned following a categorization process applied to a set of corresponding elements and/or groups of elements. For example, in one case, each element or group of elements may be assigned a class represented by an integer value selected from a predefined set or range of integers (e.g. 10 classes from 0 to 9). Each class may then have a corresponding weight value (e.g. 0 for class 0, 0.1 for class 1 or some other non-linear mapping). The relationship between class and weight value may be determined by analysis and/or experimentation, e.g. based on picture quality measurements at a decoder and/or within the encoder. The weight may then be used to multiply a corresponding residual and/or group of residuals, e.g. a residual and/or group of residuals that correspond to the element and/or group of elements. In one case, this correspondence may be spatial, e.g. a residual is computed based on a particular input element value and the categorisation is applied to the particular input element value to determine the weight for the residual. In other words, the categorization may be performed over the elements and/or group of elements of the input image, where the input image may be a frame of a video signal, but then the weights determined from this categorization are used to weight co-located residuals and/or group of residuals rather than the elements and/or group of elements. In this way, the characterization may be performed as a separate process from the encoding process, and therefore it can be computed in parallel to the encoding of the residuals process.

Example of Residual Mode Processing

Figure 15 shows an example of a residual mode. This example relates to a level 2 stream but a similar set of components may be provided for a level 1 stream. A set of input image elements i_{ij} 1501 are classified via a classification component 1502 to generate a set of class indications 1503 (e.g. in a range of 0 to 4). The class indications 1503 are then used by a weight mapping component 1504 to retrieve a set of weights 1505 associated with the class indications 1503. In parallel, a set of reconstructed up-sampled elements u_{ij} 1506 are subtracted from the input image elements i_{ij} 1501 to generate an initial set of residuals r_{ij} 1508. These residuals 1508 and the set of weights 1505 are then input to a weight multiplication component 1509 that multiplies the residuals 1508 by the set of weights 1505 to output a set of modified residuals r'_{ij} 1510. Figure 15 shows that the residual mode selection may involve filtering a subset of residual values 1512 (e.g. by multiplying them by a 0 weight) and passing through or modifying another subset of residual values 1511 (e.g. where there are non-zero weights).

In certain cases, the characterization may be performed at a location remote from the encoder and communicated to the encoder. For example, a pre-recorded movie or television show may be processed once to determine a set of weights for a set of residuals or group of residuals. These weights may be communicated over a network to the encoder, e.g. they may comprise the residual masks described with reference to Figures 14A to 14C.

In one case, instead of, or as well as weighting the residuals, the residuals may be compared against one or more thresholds derived from the categorization process. For example, the categorisation process may determine a set of classes that have an associated set of weights and thresholds, or just an associated set of thresholds. In this case, the residuals are compared with the determined thresholds and residuals that falls below a certain one or more thresholds are discarded and not encoded. For example, additional threshold processing may be applied to the modified residuals 1510 from Figure 15 and/or the weight mapping and weight multiplication components may be replaced with threshold mapping and threshold application stages. In general, in both cases, residuals are modified for further processing based on a categorisation process, where the categorisation process may be applied to corresponding image elements.

The above described methods of residual mode processing may be applied at the encoder but not applied at the decoder. This thus represents a form of asymmetrical encoding that may take into account increased resources at the encoder to improve communication. For example, residuals may be weighted to reduce a size of data transmitted between the encoder and decoder, allowing increases of quality for constrained bit rates (e.g. where the residuals that are discarded have a reduced detectability at the decoder).

Predicted Averages

As described herein, a residual element may be defined as a difference between an input frame element and a corresponding/co-located up-sampled element, as indicated below:

$$r_{ij} = i_{ij} - u_{ij}$$

At the encoder, the residuals are transformed before being quantized, entropy coded and transmitted to the decoder. In particular, the encoder uses two possible transforms, the first one called Directional Decomposition (DD), the other called Directional Decomposition Squared (DDS). More details on these transforms are also included in patent applications PCT/EP2013/059847 and PCT/GB2017/052632, which are included herein by reference.

Figure 16A shows a process 1600 involving a DD transform at the encoder. In the case of a DD, a transform is applied to each 2x2 block of a frame or plane of input data 1610. By reference to Figure 16A, four 2x2 blocks 1611 of input values 1612 are presented. These are down-sampled by a down-sampling process 1615 (e.g. similar to the down-sampling component 104, 304 of Figures 1 and 3A/B) to generate a down-sampled frame or plane 1620, with element values 1621. The down-sampled frame 1620 is then up-sampled by up-sampling process 1625 (e.g. as shown via components 134, 334 in Figures 1 and 3A/3B). This results in an up-sampled frame 1630, which also has blocks 1631 of up-sampled values 1632, where, in Figure 16A, one down-sampled value 1622 is up-sampled to generate four up-sampled values 1632 (i.e. one up-sampled block 1631). In Figure 16A, the up-sampled frame 1630 is subtracted 1635 from the input frame 1610 to generate a frame of residuals 1640, which comprise blocks 1641 of residual values 1642.

In a transformation, the following coefficients are calculated for each block of residuals 1641 (the expression below, for simplicity, refers to the left uppermost 2x2 block, but similar expressions can be easily derived for the other blocks):

$$A_0 = \frac{1}{4}\sum_{i,j=0}^{1} r_{ij} = \frac{1}{4}(r_{00} + r_{01} + r_{10} + r_{11})$$

$$H_0 = \frac{1}{4}(r_{00} - r_{01} + r_{10} - r_{11})$$

$$V_0 = \frac{1}{4}(r_{00} + r_{01} - r_{10} - r_{11})$$

$$D_0 = \frac{1}{4}(r_{00} - r_{01} - r_{10} + r_{11})$$

Looking now at an Average component (A0) this can be decomposed as follows:

$$A_0 = \frac{1}{4}\sum_{i,j=0}^{1} r_{ij} = \frac{1}{4}\sum_{i,j=0}^{1} i_{ij} - \frac{1}{4}\sum_{i,j=0}^{1} u_{ij}$$

It is noted that each up-sampled 2x2 block 1631 with up-sampled values 1632 as shown in Figure 16A is obtained from an up-sampling operation starting from the corresponding lower resolution element 1622. This lower resolution element 1622 may be referred to as a "controlling element". In the case of left uppermost block, that element would be d00.

Accordingly, d00 may be can added and deleted as follows to obtain:

$$A_0 = \frac{1}{4}\sum_{i,j=0}^{1} i_{ij} - \frac{1}{4}\sum_{i,j=0}^{1} u_{ij} - d_{00} + d_{00}$$

Which could then be grouped as follows:

$$A_0 = \left(\frac{1}{4}\sum_{i,j=0}^{1} i_{ij} - d_{00}\right) - \left(\frac{1}{4}\sum_{i,j=0}^{1} u_{ij} - d_{00}\right) = \delta A_0 + PA_0$$

whereas δA0 (delta average) is shown as 1650 corresponds to the difference 1645 between the average of the elements in the input image (e.g. of block 1611) and the controlling element 1622. The predicted average PA0 corresponds to the difference between the average of the up-sampled elements and the controlling element. This may be computed at a decoder.

Figure 16B sets out a corresponding process 1655 at the decoder. Data from the encoder 1656 communicates the δA value 1658. In parallel, a level 1 resolution frame 1660 is reconstructed and up-sampled 1665 to form an up-sampled frame 1666. Figure 16B shows a block 1661 of four lower resolution elements 1662. These elements correspond to a reconstructed video signal. The up-sampled frame 1666 is shown with four blocks 1668 of four up-sampled elements 1669. The decoder is capable of calculating the PA using the up-sampled elements 1668 and the controlling element 1662 obtained from decoding the lower resolution frame (e.g., the frame obtained from decoding a base encoded with a separate codec such as AVC, HEVC, etc.). In Figure 16B, the predicted average 1671 is determined as the difference 1670 of the average of the block of up-sampled elements 1668 and the controlling element 1662. The original average 1675 may then be reconstructed by summing 1672 the δA value 1658 and the predicted average value PA 1671. This is also why this element is called "predicted average" in that it is the component of the Average that can be predicted at the decoder. The decoder would then need only the δA, which is provided by the encoder since there information about the input image frame is known at the encoder.

Accordingly, when using the DD transform type, the decoder is able to compute the predicted average using one or more up-sampled elements and a corresponding element from a lower resolution image ("controlling element"), said corresponding element being used to generate said one or more up-sampled elements. Then, it is able to decode a value received from an encoder, said value representing the difference between one or more elements in a reference (e.g., input) image and the controlled element. It is then able to combine said predicted average and decoded value to generate one of the transformed coefficients, namely the average coefficient.

When using the DD transform type, the encoder is able to compute a value to be transmitted to the decoder, said value representing the difference between one or more elements in a reference (e.g., input) image and a corresponding element from a lower resolution image ("controlling element"). The encoder is able to generate said controlling element by replicating the operations which an encoder would need to perform in order to reconstruct the image. In particular, the controlling elements correspond to the element which the decoder would use in order to generate said one or more up-sampled elements. The encoder is then able to further transmit the H, V and D coefficients to the decoder.

In the case of a DDS transform, the operations are slightly modified. The DDS operates over a 4x4 blocks of residuals and generate 16 transformed coefficients. A DDS could be implemented in at least two ways. Either directly, by summing and subtracting the 16 residuals in the 4x4 blocks – see below:

$$DDS = \begin{pmatrix} AA & AH & \cdots & VD & DD \end{pmatrix} = \begin{pmatrix} & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \end{pmatrix} \begin{pmatrix} r_{00} \\ r_{01} \\ \vdots \\ r_{32} \\ r_{33} \end{pmatrix}$$

Alternatively, and in a more efficient manner, it can be implemented as a "two-step" transform by first performing a DD transform over each 2x2 blocks of residuals to generate a 2x2 block of DD coefficients, and then applying a second DD transform over

First step:

$$DD_{00} = \begin{pmatrix} A_{00} & H_{00} & V_{00} & D_{00} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} r_{00} \\ r_{01} \\ r_{10} \\ r_{11} \end{pmatrix}$$

$$DD_{01} = \begin{pmatrix} A_{01} & H_{01} & V_{01} & D_{01} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} r_{02} \\ r_{03} \\ r_{12} \\ r_{13} \end{pmatrix}$$

$$DD_{10} = \begin{pmatrix} A_{10} & H_{10} & V_{10} & D_{10} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} r_{20} \\ r_{21} \\ r_{30} \\ r_{31} \end{pmatrix}$$

$$DD_{11} = \begin{pmatrix} A_{11} & H_{11} & V_{11} & D_{11} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} r_{22} \\ r_{23} \\ r_{32} \\ r_{33} \end{pmatrix}$$

Second step:

$$DDS = \begin{pmatrix} AA & AH & AV & AD \\ HA & HH & HV & HD \\ VA & VH & VV & VD \\ DA & DH & DV & DD \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} A_{00} & H_{00} & V_{00} & D_{00} \\ A_{01} & H_{01} & V_{01} & D_{01} \\ A_{10} & H_{10} & V_{10} & D_{10} \\ A_{11} & H_{11} & V_{11} & D_{11} \end{pmatrix}$$

As it can be seen, in the DDS case there are four "averages" coefficients, one for each directions: (1) AA, or average of the average coefficients; (2) AH, or average of the horizontal coefficients; (3) AV, or average of the vertical coefficients; and (4) AD, or average of the diagonal coefficients.

Similarly to the DD transform, each of these average coefficients can be decomposed into a delta average (to be computed by the encoder and decoded at the decoder) and a predicted average (to be computed by the decoder), as follows:

$$AA = \delta AA + PAA$$

$$AH = \delta AH + PAH$$

$$AV = \delta AV + PAV$$

$$AD = \delta AD + PAD$$

Accordingly, there are four delta averages to be computed by the encoder, namely δAA, δAH, δAV and δAD.

Using the two-step approach defined above, the four delta averages can be computed as follows:

$$\delta AA = \frac{1}{4}\sum_{i,j=0}^{1} \delta A_{ij}$$

$$\delta AH = \frac{1}{4}\sum_{i,j=0}^{1} \delta H_{ij}$$

$$\delta AV = \frac{1}{4}\sum_{i,j=0}^{1} \delta V_{ij}$$

$$\delta AD = \frac{1}{4}\sum_{i,j=0}^{1} \delta D_{ij}$$

On the other hand, the various predicted averages can be computed as follows:

$$PAA = \frac{1}{4}\sum_{i,j=0}^{1} PA_{ij}$$

$$PAH = \frac{1}{4}\sum_{i,j=0}^{1} PH_{ij}$$

$$PAV = \frac{1}{4}\sum_{i,j=0}^{1} PV_{ij}$$

$$PAD = \frac{1}{4}\sum_{i,j=0}^{1} PD_{ij}$$

where

$$PA_{ij} = d_{ij} - \frac{1}{4}\left(u_{(2i)(2j)} + u_{(2i+1)(2j)} + u_{(2i+1)(2j)} + u_{(2i+1)(2j+1)}\right)$$

$$PH_{ij} = d_{ij} - \frac{1}{4}\left(u_{(2i)(2j)} - u_{(2i+1)(2j)} + u_{(2i+1)(2j)} - u_{(2i+1)(2j+1)}\right)$$

$$PV_{ij} = d_{ij} - \frac{1}{4}\left(u_{(2i)(2j)} + u_{(2i+1)(2j)} - u_{(2i+1)(2j)} - u_{(2i+1)(2j+1)}\right)$$

$$PD_{ij} = d_{ij} - \frac{1}{4}\left(u_{(2i)(2j)} - u_{(2i+1)(2j)} - u_{(2i+1)(2j)} + u_{(2i+1)(2j+1)}\right)$$

An alternative way of computing the predicted averages is to first compute the predicted averages for each 2x2 block and then perform a Directional Decomposition on them.

In other words, the first step is to compute:

$$PA_{ij} = d_{ij} - \frac{1}{4}\left(u_{(2i)(2j)} + u_{(2i+1)(2j)} + u_{(2i+1)(2j)} + u_{(2i+1)(2j+1)}\right)$$

and then

$$PAA = \frac{1}{4}\sum_{i,j=0}^{1} PA_{ij}$$

$$PAH = \frac{1}{4}(PA_{00} - PA_{01} + PA_{10} - PA_{11})$$

$$PAV = \frac{1}{4}(PA_{00} + PA_{01} - PA_{10} - PA_{11})$$

$$PAD = \frac{1}{4}(PA_{00} - PA_{01} - PA_{10} + PA_{11})$$

Accordingly, when using a DDS transform, the encoder may generate the various delta averages δAA, δAH, δAV and δAD and send them to the decoder, along with the other DDS coefficients HA, HH, HV, HD, VA, VH, VV, VD, DA, DH, DV, DD.

At the decoder, the decoder may compute PAA, PAH, PAV and PAD as illustrated above. Further, in the present examples, it receives the delta averages, decode them and then may sum them to the predicted averages in order to obtain the averages AA, AH, AV and AD. The averages are then be combined with the other DDS coefficients, an inverse DDS is applied, and then residuals are obtained from the inverse transform.

Alternatively, as transform and inverse transform are linear operations, inverse DDS can be done on the delta averages δAA, δAH, δAV and δAD and the other DDS coefficients HA, HH, HV, HD, VA, VH, VV, VD, DA, DH, DV, DD to obtain residuals and PA$_{ij}$s could be added post-transform to the residuals in corresponding 2x2 blocks to obtain final residual values.

Figures 16C and 16D respectively show an encoding process 1680 and a decoding process 1690 that correspond to Figure 16A and 16B but where the transformation is one-dimensional, e.g. where down-sampling and up-sampling are performed in one direction rather than two directions. This, for example, may be the case for a horizontal-only scaling mode that may be used for interlaced signals. This may be seen by the indicated elements

1681 where two elements 1682 in a block 1683 are down-sampled to generate element 1684. The input data elements 1681 and the down-sampled ("control") element are then used to generate the delta average (δA) 1685. Correspondingly, at the decoding process 1690, a two element block 1691 of up-sampled elements is compared with the down-sampled element 1662 to determine the predicted average 1671.

Signalling within DDS

In certain implementations, bit or bytestream signalling may be used to indicate whether one or more of the coefficients from the DDS transform are used for internal signalling (e.g. as opposed to carrying transformed coefficient values).

For example, in one case, a signalling bit may be set to a value of 0 to indicate that no internal signalling is used (e.g. a predefined coefficient value carries the transformed residual value for the coding unit) and may be set to a value of 1 to indicate that internal signalling is used (e.g. any existing transformed residual value is replaced by a signalling value that carries information to the decoder). In the latter case, the value of the coefficient may be ignored when inverse transforming the transformed residuals, e.g. may be assumed to be 0 regardless of the value used for signalling therein.

In one case, the HH coefficient of the DDS transform may be adapted to carry signalling in the case that the signalling bit is set to 1. This coefficient may be selected as its value has been determined to least affect the decoded residual values for a coding block.

The value carried in the internal coefficient signalling may be used for a variety of purposes. The information may be used at the decoder if the decoder is configured to receive and act on the information (e.g. at the discretion of the decoder).

In one case, the within-coefficient signalling may indicate information associated with post-processing to perform on the wider coding unit (e.g. the coding unit associated with the signalling coefficient). In one case, the within-coefficient signalling may indicate information associated with a potential artefact or impairment that may be present when the decoded coding unit is applied in one or more of the level 1 and level 2 enhancement operations. For example, the within-coefficient signalling may indicate that decoded residual data (and/or a portion of reconstructed video frame) associated with the coding unit may be subject to banding, blockiness etc. One or more post-processing algorithms may then use this information embedded within the coefficient data to selective apply one or more post-processing operations to address the impairment and improve the reconstructed video.

Predicted Residuals

As described above, certain examples may use an approach that acts to predict a coefficient generated by the transform stage. In one case, an average component (A) may be predicted using a "predicted average" computation. The predicted average computation enables a delta average to be transmitted in place of a full average value. This can save a signification amount of data (e.g. reduce a required bitrate) as it reduces the entropy of the average component to be encoded (e.g. often this delta average may be small or zero).

For example, when decoding a level 2 enhancement stream, one picture element at a level 1 resolution may be input to an up-sampling operation, where it is used to create four picture elements at an up-sampled or level 2 resolution. As part of the reconstruction, the value of the predicted average for the up-sampled coding unit of four picture elements may be added to the up-sampled values for the four picture elements.

In one case, a variation to the above predicted average computation may be applied.

In this variation, the addition of the predicted average value after up-sampling may be modified. The addition may be modified by a linear or non-linear function that acts to add different proportions of the predicted average value to different locations within the up-sampled coding block.

For example, in one case, information from one or more neighbouring coding blocks may be used to weight the predicted average value differently for different picture elements. In this case, picture elements that neighbour lower-valued picture elements may receive less of the predicted average value and picture elements that neighbour higher-valued picture elements may receive more of the predicted average value. The weighting of the predicted average may thus be set for a picture element based on the relative values of its neighbouring picture elements.

This may provide improvements when an edge is present within the coding block. In cases, where an edge is present in the up-sampled coding block, it may be beneficial to weight the predicted average value in accordance with the edge location. For example, if the edge is vertical then picture elements within one column of the coding unit may be combined with a higher or lower value than the other column of the coding unit, wherein the exact weighting depends on the gradient of the edge. Edges at different angles may have more complex weightings of the predicted average value. This form of correction to the predicted average addition may be referred to as adding a form of "tilt". It may form part of a predicted residuals computation. In these cases, each picture element may receive a different value for combination, as opposed to a common single predicted average value.

Rate Control & Quantization

In certain implementations the quantization operation may be controlled to control a bit rate of one or more of the encoded streams. For example, quantization parameters for the quantize components 323 and/or 343 in Figures 3A and 3B may be set to provide a desired bitrate in one or more of the encoded video streams (whether that be a common bit rate for all streams so as to generate a common encoded stream or different bit rates for different encoded streams).

In certain cases, the quantization parameters may be set based on an analysis of one or more of the base encoding and the enhancement stream encoding. Quantization parameters may be chosen to provide a desired quality level, or to maximise a quality level, within a set of pre-defined bit-rate constraints. Multiple mechanisms may be used to control a variation in the original video.

Figure 17A shows a schematic diagram of an example encoder 1700.

Figure 17A shows the use of the buffer 1740 with respect to the encoded base stream and the encoded L-1 stream; Figure 17B shows another example, where the buffer 1740 receives the encoded base stream and both the encoded level 1 and level 2 enhancement streams.

Figures 18 and 19 show two possible implementations of the rate controller (e.g. rate controller 1710). These implementation uses a status of the buffer to generate a set of quantization parameters Q_t for a current frame t.

Quantization Features

Certain quantization variations will now be described with reference to Figures 20A to 20D.

Figure 20A provides an example 2000 of how quantization of residuals and/or coefficients (transformed residuals) may be performed based on bins having a defined step width.

Deadzone

Figure 20B shows an example 2010 how a so-called “deadzone” (DZ) may be implemented.

Bin Folding

Figure 20C shows an example 2020 of how an approach called bin folding may be applied.

Quantization Offsets

Figure 20D shows an example 2030 of how a quantization offset may be used in certain cases.

Tiling

As described above, for example with reference to Figure 12C, in certain configurations a frame of video data may be divided into two-dimensional portions referred to as "tiles". For example, a 640 by 480 frame of video data may contain 1200 tiles of 16 pixels by 16 pixels (e.g. 40 tiles by 30 tiles). Tiles may thus comprise non-overlapping successive areas within a frame, where each area is of a set size in each of two-dimensional. A common convention is for tiles to run successively in rows across the frame, e.g. a row of tiles may run across a horizontal extent of the frame before starting a row of tiles below (a so-called "raster" format, although other conventions, such as interlaced formats may also be used). A tile may be defined as a particular set of coding units, e.g. a 16 by 16 pixel tile may comprise an 8 by 8 set of 2x2 coding units or a 4 by 4 set of 4x4 coding units.

In certain cases, a decoder may selectively decode portions of one or more of a base stream, a level 1 enhancement stream and a level 2 enhancement stream. For example, it may be desired to only decode data relating to a region of interest in a reconstructed video frame. In this case, the decoder may receive a complete set of data for one or more of the base stream, the level 1 enhancement stream and the level 2 enhancement stream but may only decode data within the streams that is useable to render the region of interest in the reconstructed video frame. This may be seen as a form of partial decoding.

Partial decoding in this manner may provide advantages in a number of different areas.

When implementing a virtual or augmented reality application, only a portion of a wide field of view may be being viewed at any one time. In this case, only a small region of interest relating to the viewed area may be reconstructed at a high level of quality, with the remaining areas of the field of view being rendered at a low (i.e. lower) level of quality. Further details regarding this approach may be found in patent publication

WO2018/015764 A1, which is incorporated by reference herein. Similar, approaches may be useful when communicating video data relating to a computer game.

Partial decoding may also provide an advantage for mobile and/or embedded devices where resources are constrained. For example, a base stream may be decoded rapidly and presented to a user. The user may then select a portion of this base stream to render in more detail. Following selection of a region of interest, data within one or both of the level 1 and level 2 enhancement streams relating to the region of interest may be decoded and used to render a particular limited area in high detail. A similar approach may also be advantageous for object recognition, whereby an object may be located in a base stream, and this location may form a region of interest. Data within one or both of the level 1 and level 2 enhancement streams relating to the region of interest may then be decoded to further process video data relating to the object.

In the present examples, partial decoding may be based on tiles. For example, a region of interest may be defined as a set of one or more tiles within frames of the reconstructed video stream, e.g. the reconstructed video stream at a high level of quality or full resolution. Tiles in the reconstructed video stream may correspond to equivalent tiles in frames of the input video stream. Hence, a set of tiles that covers an area that is smaller that a complete frame of video may be decoded.

In certain configurations described herein, the encoded data that forms part of at least the level 1 enhancement stream and the level 2 enhancement stream may result from a Run-Length encoding then Huffman encoding. In this encoded data stream, it may not be possible to discern data relating to specific portions of the reconstructed frame of video without first decoding the data (e.g. until obtaining at least quantized transformed coefficients that are organised into coding units).

In the above configurations, certain variations of the examples described herein may include a set of signalling within the encoded data of one or more of the level 1 enhancement stream and the level 2 enhancement stream such that encoded data relating to particular tiles may be identifier prior to decoding. This can then allow for the partial decoding discussed above.

For example, in certain examples, the encoding scheme illustrated in one or more of Figures 10A to 10I may be adapted to include header data that identifies a particular tile within a frame. The identifier may comprise a 16-bit integer that identifies a particular tile number within a regular grid of tiles (such as shown in Figure 12C). For example, at the start of transmission of encoded data relating to a particular tile of the input video frame, an identifier for the tile may be added to a header field of the encoded data. At the decoder, all data following the identifier may be deemed to relate to the identified tile, up to a time where a new header field is detected within the encoded stream or a frame transition header field is detected. In this case, the encoder signals tile identification information within one or more of the level 1 enhancement stream and the level 2 enhancement stream and this information may be received within the streams and extracted without decoding the streams. Hence, in a case where a decoder is to decode one or more tiles relating to a defined region of interest, the decoder may only decode portions of one or more of the enhancement streams that relate to those tiles.

Use of a tile identifier within the encoded enhancement streams allows variable length data, such as that output by the combination of Huffman and Run-length encoding, while still enabling data that relates to particular areas of a reconstructed video frame to be determined prior to decoding. The tile identifier may thus be used to identify different portions of a received bitstream.

In the present examples, enhancement data (e.g. in the form of transformed coefficients and/or decoded residual data) relating to a tile may be independent of enhancement data relating to other tiles within the enhancement streams. For example, residual data may be obtained for a given tile without requiring data relating to other tiles. In this manner, the present examples may differ from comparative Scalable Video Coding schemes, such as in associated with the HEVC and AVC standards (e.g. SVC and SHVC), that require other intra or inter picture data to decode data relating to a particular area or macroblock of a reconstructed picture. This enables the present examples to be efficiently implemented using parallel processing – different tiles and/or coding units of the reconstructed frame may be reconstructed in parallel. This can greatly speed up decoding and reconstruction on modern computing hardware where multiple CPU or GPU cores are available.

Tiles Within the Bytestream

Figure 21A shows another example 2100 of a bit or bytestream structure for an enhancement stream. Figure 21A may be seen as another example similar to Figure 9A. The top of Figure 21A shows components 2112 to 2118 of an example bytestream 2110 for a single frame of video data. A video stream will then comprise multiple such structures for each frame of the video. The bytestream for a single frame comprises a header 2112, and data relating to each of three planes. In this example, these planes are colour components of the frame, namely Y, U and V components 2114, 2216 and 2218.

In the second level of Figure 21A, the general structure of a bytestream for a given colour plane 2115 is shown. In this case, the sub-portions of a Y plane are shown. The other planes may have a similar structure. In Figure 21A, each plane comprises data 2120 relating to each of the two levels of enhancement: a first level of quality (level or LoQ 1) 2122 and a second level of quality (level or LoQ 2) 2124. As discussed above, these may comprise data for the level 1 enhancement stream and the level 2 enhancement stream.

In the third level of Figure 21A, each enhancement level 2125 further data 2130 that comprises bytestream portions 2132 relating to a plurality of layers. In Figure 21A, N layers are shown. Each layer here may relate to a different "plane" of encoded coefficients, e.g. residual data following transformation, quantization and entropy encoding. If a 2x2 coding unit is used, there may be four such layers (e.g. each direction of the directional decomposition – DD). If a 4x4 coding unit is used, there may be sixteen such layers (e.g. each direction of the directional decomposition squared – DDS). In one case, each layer may be decoded independently of the other layers; as such each layer may form an Independently Decodable Unit – IDU. If a temporal mode is used, there may also be one or more layers relating to temporal information.

When a tiling configuration is used, e.g. for partial decoding, there may be an extra decomposition 2135 of the data 2140 for each layer into portions 2142 relating to multiple tiles. These tiles may correspond to a rectangular area of the original input video. Tile size may be fixed for each Group of Pictures (GOP). Tiles may be ordered in a raster order. Figures 6B and 12C show examples of a tile structure.

Figure 21A shows an example whereby each layer further comprises portions 2142 of the bytestream relating to M tiles. Each tile thus forms an IDU and may be decoded independently of other tiles. This independence then enables selectable or partial decoding. Figure 21B shows an alternative example 2150 where each level of quality 2120 or a bytestream 2110 is first decomposed into portions 2140 relating to the M tiles, whereby each tile portion is then decomposed into portions relating to each layer 2130. Either approach may be used.

In examples, each IDU may comprise header information such as one or more of an *isAlive* field (e.g. indicating use or non-zero data), a *StreamLength* (indicating a data size of the stream portion) and a payload carrying the encoded data for the IDU. Using an indication of whether a particular tile contains data (e.g. *isAlive*=1) may help reduce the data to be transmitted, as often particular tiles may be 0 due to the use of residual data, and so additional tile data to be transmitted may be minimised.

When tiling is used, a header, e.g. for a group of pictures (GOP), may be modified to include a tiling mode flag. In this case, a first flag value (e.g. 0) may represent a "null region" mode whereby partial decoding is not supported and a second flag value (e.g. 1) may represent a "tile" mode, whereby partial decoding is supported. The second flag value may indicate that a particular fixed-size tile mode is being used, whereby a plane (e.g. one of the YUV planes) is divided into fixed size rectangular regions (tiles), of size T_W x T_H, and that the tiles are indexed in raster-order. In other cases, different flag values may indicate different tiling modes, e.g. one mode may indicate a custom tile size that is transmitted together with the header information.

In one case, a tile size may be signalled in header information. The tile size may be signalled explicitly (e.g. by sending a tile width T_W in pixels and a tile height in pixels T_H). In one case, a tile size may be signalled by sending an index for a look-up table stored at the decoder. The tile size may thus be signalled using one byte that indicates one of up to 255 tile sizes. One index value may also indicate a custom size (e.g. to be additionally signalled in the header). The tile size, if signalled explicitly in the header information, may be communicated using 4 bytes (two bytes per width/height).

If a tiling mode is signalled, there may be one or more tile-specific configurations that are signalled in the header information. In one case, a data aggregation mode may be signalled (e.g. using a 1-bit flag). A value of one may indicate that tile data segments within the bytestream, such as the *isAlive* / *StreamLength* / *Payload* portions described above, are to be grouped or aggregated (e.g. the data stream first contains the *isAlive* header information for the set of tiles, then the StreamLength information for the set of tiles, followed by the payload information for the set of tiles). Organising the bytestream in this manner may facilitate selective decoding of tiles, e.g. as stream length information for each tile may be received prior to the payload data. In this case, the aggregated data may also be optionally compressed using Run-Length and Huffman encoding (e.g. as described herein) and this may also be flagged (e.g. using a 1-bit field). Different portions of the aggregated data stream may have different compression settings. If information such as the stream length fields are Huffman encoded, then these may be encoded as either absolute or relative values (e.g. as a relative difference from the last stream value). Relative value encoding may further reduce bytestream size.

In these examples, a method of encoding an enhancement stream is described whereby an enhancement bitstream may be split into portions or chunks that represent different spatial portions of a frame of video (i.e. tiles). The data relating to each tile may be received and decoded independently, allowing parallel processing and selective or partial decoding.

Neural Network Up-sampling

In certain examples, up-sampling may be enhanced by using an artificial neural network. For example, a convolutional neural network may be used as part of the up-sampling operation to predict up-sampled pixel or signal element values. Use of an artificial neural network to enhance an up-sampling operation is described in WO 2019/111011 A1, which is incorporated by reference herein. A neural network up-sampler may be used to implement any one of the up-sampling components described in the examples herein.

Figure 22A shows a first example 2200 of a neural network up-sampler 2210. The neural network up-sampler may be used to convert between signal data at a first level *(n-1)* and signal data at a second level *n*. In the context of the present examples, the neural network up-sampler may convert between data processed at enhancement level 1 (i.e. level of quality – LoQ – 1) and data processed at enhancement level 2 (i.e. level of quality – LoQ – 2). In one case, the first level *(n-1)* may have a first resolution (e.g. size_1 by size_2 elements) and the second level *n* may have a second resolution (e.g. size_3 by size_4 elements). The number of elements within each dimension at the second resolution may be a multiple of the number of elements within each dimension at the first resolution (e.g. size_3 = F1*size_1 and size_4 = F2*size_2). In described example, the multiples may be the same in both dimensions (e.g. F1=F2=F and in some examples, F=2).

In certain examples, use of an artificial neural network may include conversion of element data (e.g. picture elements such as values for a colour plane) from one data format to another. For example, element data (e.g. as input to the up-sampler in non-neural cases) may be in the form of 8- or 16-bit integers, whereas a neural network may operate upon float data values (e.g. 32- or 64-bit floating point values). Element data may thus be converted from an integer to a float format before up-sampling, and/or from a float format to an integer format after neural-enhanced up-sampling. This is illustrated in Figure 22B.

In Figure 22B, the input to the neural network up-sampler 2210 (e.g. the up-sampler from Figure 22A) is first processed by a first conversion component 2222. The first conversion component 2222 may convert input data from an integer format to a floating-point format. The floating-point data is then input to the neural network up-sampler 2210, which is free to perform floating-point operations. An output from the neural network up-sampler 2210 comprises data in a floating-point format. In Figure 22B, this is then processed by a second conversion component 2224, which converts the data from the floating-point format to an integer format. The integer format may be the same integer format as the original input data or a different integer format (e.g. input data may be provided as an 8-bit integer but output as a 10-, 12- or 16-bit integer). The output of the second conversion component 2224 may place the output data in a format suitable for upper enhancement level operations, such as the level 2 enhancement described herein.

In certain examples, instead of, or as well as data format conversion the first and/or second conversion components 2222 and 2224 may also provide data scaling. Data scaling may place the input data in a form better suited to the application of an artificial neural network architecture. For example, data scaling may comprise a normalisation operation. An example normalisation operation is set out below:

norm_value = (input_value - min_int_value) / (max_int_value – min_int_value)

where input_value is an input value, min_int_value is a minimum integer value and max_int_value is a maximum integer value. Additional scaling may be applied by multiplying by a scaling divisor (i.e. dividing by a scale factor) and/or subtracting a scaling offset. The first conversion component 2222 may provide for forward data scaling and the second conversion component 2224 may apply corresponding inverse operations (e.g. inverse normalisation). The second conversion component 2224 may also round values to generate an integer representation.

Figure 22C shows an example architecture 2230 for a simple neural network up-sampler 2210. The neural network up-sampler 2210 comprises two layers 2232, 2236 separated by a non-linearity 2234. There is also an optional post-processing operation 2238. By simplifying the neural network architecture, up-sampling may be enhanced while still allowing real-time video decoding.

The convolution layers 2232, 2236 may comprise a two-dimensional convolution. The convolution layers may apply one or more filter kernels with a predefined size. In one case, the filter kernels may be 3x3 or 4x4. The convolution layers may apply the filter kernels, which may be defined with a set of weight values, and may also apply a bias. The bias is of the same dimensionality as the output of the convolution layer. In the example of

Figure 22C both convolution layers 2232, 2236 may share a common structure or function but have different parameters (e.g. different filter kernel weight values and different bias values). Each convolution layer may operate at a different dimensionality. The parameters of each convolution layer may be defined as a four-dimensional tensor have size – (kernel_size1, kernel_size2, input_size, output_size). The input of each convolution layer may comprise a three-dimensional tensor of size – (input_size_1, input_size_2, input_size). The output of each convolution layer may comprise a three-dimensional tensor of size – (input_size_1, input_size_2, output_size). The first convolution layer 2232 may have an input_size of 1, i.e. such that it receives a two-dimensional input similar to a non-neural up-sampler as described herein. Example values for these sizes are as follows: kernel_size1 and kernel_size2 = 3; for the first convolutional layer 2232, input_size =1 and output_size = 16; and for the second convolutional layer 2236, input_size = 16 and output_size = 4. Other values may be used depending on the implementation and empirical performance. In the case that the output size is 4 (i.e. four channels are output for each input element), this may be refactored into a 2x2 block representing the up-sampled output for a given picture element.

The input to the first convolution layer 2232 may be a two-dimensional array similar to the other up-sampler implementations described herein. For example, the neural network up-sampler 2210 may receive portions of a reconstructed frame and/or a complete reconstructed frame (e.g. the base layer plus a decoded output of the level 1 enhancement). The output of the neural network up-sampler 2210 may comprise a portion of and/or a complete reconstructed frame at a higher resolution, e.g. as per the other up-sampler implementations described herein. The neural network up-sampler 2210 may thus be used as a modular component in common with the other available up-sampling approaches described herein. In one case, the selection of the neural network up-sampler, e.g. at the decoder, may be signalled within a transmitted bytestream, e.g. in global header information.

The non-linearity layer 2234 may comprise any known non-linearity, such as a sigmoid function, a tanh function, a Rectified Linear Unit (ReLU), or an Exponential Linear Unit (ELU). Variations of common functions may also be used, such as a so-called Leaky ReLU or a Scaled ELU. In one example, the non-linearity layer 2234 comprises a Leaky ReLU – in this case the output of the layer is equal to the input for values of input greater than 0 (or equal to 0) and is equal to a predefined proportion of the input, e.g. a*input, for values of the input less than 0. In one case, a may be set as 0.2.

Figure 22D shows an example 2240 with one implementation of the optionally post-processing operation 2238 from Figure 22C. In this case, the post-processing operation may comprise an inverse transform operation 2242. In this case, the second convolution layer 2236 may output a tensor of size (size1, size2, number_of_coefficients) – i.e. the same size as the input but with a channel representing each direction within a directional decomposition. The inverse transform operation 2242 may be similar to the inverse transform operation that is performed in the level 1 enhancement layer. In this case, the second convolution layer 2236 may be seen as outputting coefficient estimates for an up-sampled coding unit (e.g. for a 2x2 coding block, a 4-channel output represents A, H, V and D coefficients). The inverse transform step then converts the multi-channel output to a two-dimensional set of picture elements, e.g. an [A, H, V, D] vector for each input picture element is converted to a 2x2 picture element block in level *n*.

Similar adaptations may be provided for down-sampling. An up-sampling approach applied at the encoder may be repeated at the decoder. Different topologies may be provided based on available processing resources.

The parameters of the convolutional layers in the above examples may be trained based on pairs of level *(n-1)* and level *n* data. For example, the input during training may comprise reconstructed video data at a first resolution that results from applying one or more of the encoder and decoder pathways, whereas the ground truth output for training may comprise the actual corresponding content from the original signal (e.g. the higher or second resolution video data rather than up-sampled video data). Hence, the neural network up-sampler is trained to predict, as closely as possible, the input level *n* video data (e.g. the input video enhancement level 2) given the lower resolution representation. If the neural network up-sampler is able to generate an output that is closer to the input video that a comparative up-sampler, this will have a benefit of reducing the level 2 residuals, which will further reduce the number of bits that need to be transmitted for the encoded level 2 enhancement stream. Training may be performed off-line on a variety of test media content. The parameters that result from training may then be used in an on-line prediction mode. These parameters may be communicated to the decoder as part of an encoded bytestream (e.g. within header information) for a group of pictures and/or during an over-the-air or wire update. In one case, different video types may have different sets of parameters (e.g. movie vs live sport). In one case, different parameters may be used for different portions of a video (e.g. periods of action vs relatively static scenes).

Example Encoder and Decoder Variations

Graphical Example with Optional Level 0 Upscaling

Figure 23 shows a graphical representation 2300 of the decoding process described in certain examples herein. The various stages in the decoding process are shown from left to right in Figure 23. The example of Figure 23 shows how an additional up-sampling operation may be applied following the decoding of the base picture. An example encoder and an example decoder to perform this variation are shown respectively in Figures 25 and 26.

Fourth Example Decoder

Figure 24 shows a fourth example decoder 2400. The fourth example decoder 2400 may be seen as a variation of the other example decoders described herein. Figure 24 represents in a block diagram some of the processes described in more detail above and below. The scheme comprises an enhancement layer of residual data, which are then added, once processed and decoded, to a decoded base layer. The enhancement layer further comprises two sub-layers 1 and 2, each comprising different sets of residual data. There is also a temporal layer of data including signalling to predict some of the residuals at sub-layer 2, e.g. using a zero-motion vector algorithm.

Fifth Example Encoder and Decoder

Figures 25 and 26 respectively show variations of the encoder architecture of Figures 1, 3A and 3B and the decoder architecture of Figures 2, 5A and 5B.

The encoding process 2500 to create a bitstream is shown in Figure 25.

Figure 26 shows a variation of a decoder 2600 according to an example. The decoder may comprise a variation of the decoder shown in any one of Figures 2, 5A, 5B and 24. The decoder of Figure 26 may be used together with the encoder of Figure 25.

As described with reference to the above examples, unlike comparative scalable codecs, the new approaches described herein may be completely agnostic of the codec used to encode the lower layer. This is because the upper layer is decodable without any information about the lower layer, as it shown in Figures 2, 24 and 26, for example. As shown in Figure 26, a decoder receives multiple streams generated by the encoder. These may be five or so streams that include: a first encoded stream (encoded base) that is produced by feeding a base codec (e.g., AVC, HEVC, or any other codec) with a down-sampled version of the input video; a second encoded stream (level 1 coefficient layers) that is produced by processing the residuals obtained by taking the difference between the reconstructed base codec video and the down-sampled version of the input video (level 1 residuals); a third encoded stream (level 2 coefficient layers) that is produced by processing the residuals obtained by taking the difference between an up-sampled version of a corrected version of the reconstructed base coded video and the input video (level 2 residuals); a fourth encoded stream (e.g. in the form of a temporal layer) that is produced from the temporal processing to instruct the decoder; and a fifth stream (headers) that are produced for configuring the decoder. The encoded base stream is decoded by a base decoder implementing a decoding algorithm corresponding to the encoding algorithm implemented by the base codec used in the encoder, and the output of this is a decoded base. Separately, and independently, the level 1 coefficient groups are decoded in order to obtain level 1 residual data. Further, separately and independently, the level 2 coefficient groups are decoded in order to obtain level 2 residual data. The decoded base, the level 1 residual data and the level 2 residual data are then combined. In particular, the decoded base is combined with the level 1 residuals data to generate an intermediate picture. The intermediate picture may be then up-sampled and further combined with the level 2 residual data.

Moreover, the new approach uses an encoding and decoding process which processes the picture without using any inter-block prediction. Rather, it processes the picture by transforming an NxN block of picture elements (e.g., 2x2 or 4x4) and processing the blocks independently from each other. This results in efficient processing as well as in no-dependency from neighbouring blocks, thus allowing the processing of the picture to be parallelised.

In general summary, with reference to Figure 26, there is shown there is shown a non-limiting exemplary embodiment according to the present invention. In Figure 26, an exemplary decoding module 2600 is depicted. The decoding module 2600 receives a plurality of input bitstreams, comprising encoded base 2616, level 1 coefficient groups 2626, level 2 coefficient groups 2646, a temporal coefficient group 2656 and headers 2666.

In general, the decoding module 2600 processes two layers of data. A first layer, namely the base layer, comprises a received data stream 2616 which includes the encoded base. The encoded base 2616 is then sent to a base decoding module 2618, which decodes the encoded base 2616 to produce a decoded base picture. The base decoding may be a decoder implementing any existing base codec algorithm, such as AVC, HEVC, AV1, VVC, EVC, VC-6, VP9, etc. depending on the encoded format of the encoded base.

A second layer, namely the enhancement layer, is further composed of two enhancement sublayers. The decoding module receives a first group of coefficients, namely level 1 coefficient groups 2626, which are then passed to an entropy decoding module 2671 to generate decoded coefficient groups. These are then passed to an inverse quantization module 2672, which uses one or more dequantization parameters to generate dequantized coefficient groups. These are then passed to an inverse transform module 2673 which performs an inverse transform on the dequantized coefficient groups to generate residuals at enhancement sublayer 1 (level 1 residuals). The residuals may then be filtered by a smoothing filter 2632. The level 1 residuals (i.e., the decoded first enhancement sublayer) is applied to a processed output of the base picture.

The decoding module receives a second group of coefficients, namely level 2 coefficient groups 2646, which are then passed to an entropy decoding module 2681 to generate decoded coefficient groups. These are then passed to an inverse quantization module 2682, which uses one or more dequantization parameters to generate dequantized coefficient groups. The dequantization parameters used for the enhancement sublayer 2 may be different from the dequantization parameters used for the enhancement sublayer 1. The dequantized coefficient groups are then passed to an inverse transform module 2683 which performs an inverse transform on the dequantized coefficient groups to generate residuals at enhancement sublayer 2 (level 2 residuals).

Variations of Aspects of the Described Examples

A number of variations of certain aspects described above will now be described.

Partial Tiling

In certain examples, each group of coefficients may be encoded and decoded separately. However, each group contains the respective coefficients for the whole frame (e.g. one group may relate to all the "A" coefficients and another group may relate to all the "V" coefficients for a 2x2 transform). In the present description, the groups of coefficients are also referred to as coefficient layers.

In certain variations, smaller portions of the frame (e.g., tiles) may be decoded individually by the decoder, thus enabling features such as partial decoding.

In particular, the bitstream signals to the decoder whether the tiling of the coefficients has been enabled. If enabled, the decoder is then able to select which tiles to decode by identifying, within a group of coefficients, the portions of the group corresponding to the selected tiles.

For example, in one case, the layers of Figure 9A may be tiled as shown in Figure 21A. Each of the tiles 2140 may be alternatively referred to as sub-groups of coefficients (SGs). Each coefficient group may be split into *M* sub-groups, each sub-group corresponding to a tile.

In certain examples, the size of each sub-group may differ between sub-groups as the size may depend on the amount of data encoded in each group. The size of each sub-group as well as whether the sub-group is active or not (a subgroup is only active if it contains any encoded data) may be signalled as compressed metadata, which may, for example, be encoded and decoded using Huffman coding and/or RLE as described with respect to other examples.

Partial decoding, e.g. decoding certain tiles but not decoding other tiles, may be particularly useful for virtual and augmented reality applications and for telepresence applications (e.g. remote medicine or surgery). The solution described here enables a decoder to selectively choose the portion of the video to decode, for example based on a viewport area, and decode only that part. By way of non-limiting example, the decoder may receive an 8K picture (8,192 x 4,320 pixels) but decide only to display a portion of it due, for example, to the viewpoint of the user (e.g., a 4K area of 4,096 x 2,160 pixels).

In particular, in a hierarchical coding scheme like the one described in examples herein, a base layer may be a lower resolution layer (e.g., 4K) encoded with a legacy codec (e.g., HEVC, VVC, EVC, AV1, VP9, AVC, etc.) and the enhancement layer may be a higher resolution layer (e.g., 8K) encoded with an enhancement codec such as the low complexity enhancement video coding described herein. The decoder may select a portion of the 8K full resolution picture to decode, for example a 4K portion. The decoder would first decode the base layer using the legacy codec, and then would only select the portion of interest of the 8K enhancement layer, for example a 4K area or a slightly bigger one depending on the decision of the decoder. In this way, the decoder would significantly speed up the time to decode the region of interest of the picture without losing on the resolution.

An exemplary method of the above variation may comprise: receiving first and second sets of reconstruction data, said reconstruction data to be used to reconstruct a video sequence (e.g. comprising the encoded residual data described herein); selecting a region of interest in a video sequence; decoding a first portion of the first set of reconstruction data based on the selected region of interest; and decoding a second portion of the second set of reconstruction data based on the selected region of interest. The first portion may correspond to the entirety of the first set. The method may comprise a step of processing the first portion to produce a preliminary reconstruction of the video sequence. The method may further comprise combining the decoded second portion with the preliminary reconstruction to produce a final reconstruction of the video sequence. The final reconstruction may correspond to a region of interest of the reconstruction that would be produced if the whole first and second set were to be decoded and combined together.

Modular Signalling of Parameters

In an aspect of the present disclosure, there is provided a method for signalling certain decoding parameters in a modular manner. In particular, one or more bits may be used in a signalling portion of a bitstream (for example, in a header indicating parameters associated with a sequence, such as Sequence Parameter Sets (SPS), or with a picture, such as Picture Parameter Sets (PPS)) to indicate that certain parameters are indicated in the bitstream.

In particular, the bitstream may contain one or more bits which, when set to one or more certain values, indicate to the decoder the presence of additional information to be decoded. The decoder, once received the bitstream, decodes the one or more bits and, upon determining that the one or more bits corresponds to said one or more certain values, interpret one or more subsequent set of bits in the bitstream as one or more specific parameters to be used when decoding the bitstream (e.g., a payload included in the bitstream).

In a non-limiting example, said one or more specific parameters may be associated with the decoding of a portion of encoded data. For example, the one or more specific parameters may be associated with one or more quantization parameters to decode a portion of the encoded data. For example, if the encoded data comprises two or more portions of encoded data (for example, each portion may be a sublayer of an enhancement layer as described previously), the one or more specific parameters may be one or more quantization parameters associated with decoding some of the two or more portions of encoded data. In another example, the one or more specific parameters may be one or more parameters associated with some post-processing operations to be performed at the decoder, for example applying a dithering function.

In a specific example, the one or more bits may be a bit (e.g., *step_width_level1_enabled* bit) which enables explicit signalling of a quantization parameter (e.g., *step_width_level1*) only when required. For example, this may occur only when there are data encoded in sublayer 1 as described above. In particular, if the bit *step_width_level1_enabled* is set to "0", then the value of the step width for sublayer 1 would be set by default to a maximum value. On the other hand, when the bit *step_width_level1_enabled* is set to "1", then *step_width_level1* is explicitly signalled and the value of the step width for sublayer 1 is derived from it. A decoding module / decoder would decode the bit *step_width_level1_enabled* and, if it determines that it is set to "0", it is able to set the value of the step width for sublayer 1 to a maximum value. On the other hand, if it determines that it is set to "1", it is able to set the value of the step width for sublayer 1 to a value corresponding to the parameter *step_width_level1* (for example, a value between 0 and 2^N-1 where N is the number of bits associated with *step_width_level1*).

In a different example, the one or more bits may be a bit (e.g., *decoder_control* bit) to enable two parameters (e.g., dithering control variables *dithering_type* and *dithering_strength*) to be signalled on a per picture basis if *decoder_control* is set to "1". A decoding module / decoder would decode the bit *decoder_control* and, if it determines that it is set to "1", it would decode the dithering control variables *dithering_type* and *dithering_strength* and apply the dithering as described in the present application.

The above mechanism provides some important technical advantages, here described by reference to the specific examples but which can be easily generalised to general cases. First, there are some efficiency gains coming from the use of the bit *step_width_level1_enabled*, which brings an N bits per picture saving in the event no enhancement is used for sub-layer 1. This could result, for example, in a saving of 800 bps for a 50fps sequence. Second, the use of the bit *step_width_level1_enabled* may lead to a decoding module /decoder being able to "by-pass" completely any processing for enhancement sub-layer 1, thus further decreasing the decoding complexity.

Further examples of different signalling approaches are described with respect to the syntax and semantic sections below.

Hybrid Decoding Module

In an aspect of the present disclosure, there is provided a decoding module to enable decoding of a combined bitstream made of at least a first bitstream decodable with a first decoding algorithm (e.g., a base codec such as AVC, HEVC, VVC, etc.) and a second bitstream decodable with a second decoding algorithm (e.g., the enhancement codecs described herein). The two bitstreams may comprise the bitstreams referred to herein as the encoded base stream and the encoded enhancement stream, where the encoded enhancement stream may have two sub-streams corresponding to each of a plurality of layers, levels or sub-levels.

In a first non-limiting aspect, the combined bitstream is received by a receiving module which separates the first bitstream and the second bitstream, and sends the first bitstream to a first decoding module (capable of decoding with the first decoding algorithm) and a second bitstream to a second decoding module (capable of decoding with the second decoding algorithm). This may comprise a form of demultiplexer. Further, the module may receive from the first decoding module a stream corresponding to the decoded first bitstream and pass it to the second decoding module. The second decoding module may then use it in order to generate a final decoded stream as described in further detail in the present specification.

In a second non-limiting aspect, the combined bitstream is received by a first decoding module (capable of decoding with the first decoding algorithm) and at the same time by a second decoding module (capable of decoding with the second decoding algorithm). The first decoding module would decode only the first bitstream and discard the second bitstream. The second decoding module would decode only the second bitstream and discard the first bitstream. The second decoding module may then receive the decoded first bitstream and then use it in order to generate a final decoded stream as described in further detail in other examples.

The example of NALU processing set out below describes certain ones of these aspects in more detail.

NALU Processing

Examples are described herein where a base stream and an enhancement stream may be encapsulated within a set of Network Abstraction Layer Units or NALUs. The Network Abstraction Layer or NAL was introduced as part of the H.264/AVC and HEVC video coding standards. It provides a mechanism whereby a video coding layer, e.g. that may comprise one or more of the base stream and the enhancement stream, is mapped onto underlying network transport layers such as RTP/IP (for Internet traffic) and MPEG-2 (for broadcast signals).

Each NALU may be seen as a packet of information that contains an integer number of bytes. One set of bytes form a NAL header. The NAL header may indicate a type of data that is contained within NALU. This, for example, is illustrated in the later examples of syntax for the bitstream. The NAL header may be a number of bytes (e.g. 1 or 2 bytes). The remaining bytes of the NALU comprise payload data of the type indicated by the NAL header. The NAL header may comprise a *nal_unit_type* variable, which indicates the NALU type. This is shown in some of the later described examples.

The NAL unit may specify a generic format for use in both packet-oriented and bitstream-oriented transport systems, and a series of NALUs generated by an encoder may be referred to as a NALU stream. In the present case, both the base layer and the enhancement layer may be encapsulated as a NALU stream. In certain cases, each layer may comprise a different NALU stream. In those cases, the first and second enhancement layer streams (e.g. level 1 and level 2 as described herein) may be encapsulated in a single NALU stream (e.g. a general "enhancement stream") or supplied as separate NALU streams (e.g. enhancement stream 1 and enhancement stream 2).

In one embodiment, as indicated in the later section on syntax, at least one enhancement stream comprising the encoded enhancement data is indicated with a specific NAL header unit type value (e.g. 0 in the later section). This indicates to a decoder that the NAL stream relates to the video coding specifications described in examples herein.

In certain implementations, it may be desired that a legacy decoder is able to receive and decode the encoded base stream as described herein. However, certain decoders may not be able to parse NALUs for the enhancement layers, e.g. they may only be configured to process NALUs for legacy video coding standards such as AVC or HEVC. In this case, if the decoder receives NALUs that do not comply with the specified configurations of the legacy video coding standards, it may experience an error and/or refuse to decode the encoded base stream as well as the encoded enhancement streams. For example, a legacy decoder may receive both an encoded base stream and an encoded enhancement stream; however, as the encoded enhancement stream has a NALU type that is not expected by the legacy decoder, it may result in an exception that prevents the processing of the encoded base stream, despite the encoded base stream being configured according to the legacy standard. Or alternatively, the NALU type used by the enhancement stream may be parsed differently according to the legacy standard, resulting in unpredictable operation of the decoder.

One solution to this issue is to provide a front-end component at the decoder that parses received NALUs and that is configured with knowledge of the enhancement coding technology as well as the base coding technology and as such may filter the NALUs that are sent to a downstream legacy decoder. However, this may complicate decoding and requires an additional entity within the decoding pipeline.

Another solution is for the encoded enhancement stream to use a NALU structure that is supported by the base coding technology (e.g. the base codec) but where the NALU header indicates a unit type that is not used by the base coding technology. Reference will be made to a single enhancement stream in these examples, where this stream encapsulates the two layers of the described enhancement streams. However, in other examples, there may be two separate enhancement streams.

In the second solution discussed above, the enhancement stream may use an NALU structure supported by the base stream but may set the NALU type to a unit type that is not specified within the base coding technology or that is set as a reserved unit type. For example, a base coding technology may have a unit type that is set by a byte or two bytes, indicating, respectively, 256 or 65536 possible integer values representing the same number of possible unit types. Only a small number of these unit types may actually be used by the base coding technology (e.g. as specified in a decoding specification for the technology), with remaining unit types indicated as a range of “non-specified” unit types. In certain cases, certain ranges of integer values may be reserved as well as, or instead of, being indicated as “non-specified”.

In this case, the encoder of the enhancement stream may encapsulate the stream using NALUs that comply with the structure of the base coding technology but have an NALU type that is set to a non-specified or reserved value. A legacy decoder may then be able to receive and parse the header of the NALUs for the enhancement stream but the indication of the unit type as non-specified or reserved may cause the legacy decoder to simply ignore or discard these units (e.g. as instructed by the base coding technology). The legacy decoder may then also receive the NALUs for the encoded base stream, which will have the same NAL structure as the NALUs for the enhancement stream, but the NALU type will not be non-specified or reserved. As the same NAL structure is used, the header of the NALU may be processed as a conventional stream according to the legacy standard. In this case, an enhancement decoder that is configured to process the enhancement stream may receive the enhancement stream as a set of NALUs, and parse the NAL header to determine the unit type. In this case, although the unit type may be non-specified or reserved with respect to the base coding technology, it may be specified in a specification for the enhancement coding technology, meaning the enhancement decoder is able to parse and process the enhancement stream.

For example, a NALU header for an example base coding technology may be 1 byte. In this example base coding technology, a range of 0 to 128 may indicate different specified (i.e. supported) unit types, a range of 129 to 192 may indicate a range of non-specified unit types and a range of 193 to 255 may indicate reserved values. The encoded base stream as described herein may thus use a NALU structure that is supported by the base coding technology and have a unit type in the supported range (0 to 128). The enhancement coding technology may use the same NALU header and structure but use NALU types within the range 129 to 255 (or one of 129 to 192 or 193 to 255). A legacy decoder and an enhancement decoder may receive both the encoded base stream and the encoded enhancement stream. The enhancement coding technology may be configured to use a NALU type that is specified in the base coding technology to be ignored or discarded by a decoder. Hence, the legacy decoder receives both streams but only processes the base stream, discarding NALUs (i.e. packets) for the enhancement stream. The enhancement decoder, on the other hand, is able to process the packets for the enhancement stream but, if so configured, discard NALUs (i.e. packets) for the base stream. In this manner there is no requirement for a front-end parser to distribute packets. This is all performed based on the NALU type as specified in the NALU header.

Thus, in certain examples, there is a stream of packets (e.g. NALUs) where the packets relate to either an encoded base stream or an encoded enhancement stream. The packets of the encoded base stream and the encoded enhancement stream have a structure that is compatible with a base coding technology (e.g. a base codec). The packets comprise a header, where the header indicates a packet type (e.g. NALU type). Packets relating to the encoded base stream have a first range of packet type values that are supported by the base coding technology (e.g. that have a value that may be parsed and processed by a decoder configured according to the base coding technology). Packets relating to the encoded enhancement stream have a second range of packet type values that differ from the first range of packet type values and that do not have a function within the base coding technology (e.g. that are non-specified or reserved). The packet type thus allows for a mapping between the packets and a decoder adapted to process those packets.

A decoder configured according to the base coding technology may thus process the encoded base stream and output a decoded base stream using the packets relating to the encoded base stream. The same decoder may process the headers of the packets relating to the encoded enhancement stream (i.e. process the encoded enhancement stream packets within breaking) but may discard or ignore according to the specification of the base coding technology. The decoded base stream may be rendered on a display device or used together with a decoded enhancement stream as set out below.

A decoder configured according to the enhancement coding technology (e.g. as described with respect to "enhancement" coding herein, also referred to herein as a low complexity enhancement video coding or LCEVC codec) may thus process the encoded enhancement stream and output a decoded enhancement stream using the packets relating to the encoded enhancement stream. The same decoder may discard or ignore the packets relating to the encoded base stream according to the specification of the enhancement coding technology. The decoded enhancement stream may be combined with the decoded base stream as described herein, e.g. to generate an enhanced reconstructed video at a level of quality that is higher than the level of quality of the base stream.

In both cases, the packet type as set out in the packet header (e.g. the NALU type in the NALU header) enables a mapping between NALU and decoder. The same data stream may thus be received and processed by both a legacy and enhancement decoder but selectively processing applied to different components of that stream (e.g. base and enhancement portions) based on the unit type value. Legacy decoders may also operate with enhancement coding technology without error. Both decoders need only parse the header of the NALU, which allows for efficient processing of large quantities of data, e.g. neither decoder needs to parse payload data for a data stream it does not process.

Selection of NAL Structure by an Enhancement Encoder

In the case above, in the context of the previously described examples, different base coding technologies may be used. For example, a base coding technology (i.e. a base codec) may be selected by an enhancement encoder based on configuration data. The configuration data may represent a user selection and/or a selection according to one or more operating parameters. In this case, the enhancement encoder supports multiple base encodings.

In the case that the enhancement encoder supports multiple base encodings, the enhancement encoder may be configured to select a NAL structure, e.g. a format for the NALU and a NALU type, based on a selected base encoding. For example, a hybrid encoding may comprise a base encoding and an enhancement encoding as described herein. In the examples above, the NALUs for both the base encoding and the enhancement encoding have a structure where the NALU header may be parsed by a base decoder. In this case, the structure that is used for both the base encoding and the enhancement encoding may be selected based on the selected base encoding. While a base encoder may, by default, generate an encoded base stream with a compatible NALU structure, the enhancement encoder may need to be configured to generate one or more enhancement streams that have a NALU structure that is compatible with the base encoding. In other words, the enhancement encoder may support multiple NAL structures and select the structure that is needed based on the base encoder. The enhancement encoder may determine a base coding technology that is being used (e.g. AVC or HEVC) and then configured the NALUs and the NALU type in the header in accordance with that base coding technology. This may be useful where different base coding technologies have different non-specified and/or reserved unit types. For example, different base coding technologies may use a different number of bytes for the NALU header, and as such the integer values for the non-specified and/or reserved unit types may differ for the base coding technologies. The enhancement encoder in the above examples is adapted to select a NALU header value (e.g. a non-specified and/or reserved unit type) that is compatible with the base coding technology to facilitate success decoding of both the base and enhancement streams.

Similarly, when multiple base coding technologies are selectable, an enhancement decoder may be configured to determine a base coding technology that is being used in relation to a received stream (e.g. an enhancement stream that is associated with a corresponding base stream), and parse the NAL accordingly. For example, the enhancement decoder may determine a base codec that is being used and use this determination to configure the parsing of NALUs, including at least a parsing of the NALU header. In one case, the base coding technology may be signalled by the enhancement encoder. In another case, the enhancement decoder may be configured to match a received NALU against a set of possible NALUs, e.g. without explicit signalling from the enhancement encoder. For example, a byte size of the NALU header may indicate a particular base coding technology. The enhancement decoder may be configured to parse one or more NALU headers for one or more of the encoded base stream and the encoded enhancement stream to determine a base coding technology. In yet another case, the enhancement decoder may be configured to receive information from a base codec that indicates which base codec is being used. This information may then be used to select a NALU configuration for parsing one or more of the encoded base stream (e.g. to ignore) and the encoded enhancement stream (e.g. to process). In this case, the base codec and/or a configuration layer may comprise an application programming interface, where a method call is used to return the base codec type (i.e. to determine at least a base decoder that is used to decode the base stream).

Up-sampler Coefficient Signalling

An enhancement encoder and decoder as described herein may perform up-sampling ("up-scaling") to convert from one spatial layer to another (e.g. from a lower resolution to a higher resolution). The up-sampling may be performed in one or more dimensions, and in certain cases may be omitted.

Different types of up-sampling may be used. At least nearest neighbour, bilinear, bicubic, modified cubic and neural network up-samplers are described in the examples herein. These up-samplers may use an up-sampling kernel. An up-sampling kernel may comprise one or more coefficient values to implement the up-sampling. For example, the one or more coefficient values may be used in one or more up-sampling computations, such as additions or multiplications. In one case, an up-sampling kernel may comprise coefficient values for use in one or more matrix transformations. An up-sampling kernel may comprise a multi-dimensional array (e.g. a matrix or tensor). For example, a cubic up-sampler may use a two-dimensional matrix as an up-sampling kernel and neural network up-sampler may use a series of one or more convolutions (e.g. with or without non-linear activation functions) that use one or more multi-dimensional tensors (see the 4D and 3D examples described herein).

In the above cases, an up-sampler (or up-sampling component, process or operation) may be defined by way of an up-sampler type and a set of configurable coefficients (the "kernel" described above). The set of configurable coefficients may be signalled to an enhancement decoder. The signalling may be sent from an enhancement encoder and/or from a cloud configuration server. In one case, the up-sampler type may be determined by the enhancement decoder by parsing (e.g. processing or otherwise examining) a received set of configurable coefficients. This may avoid the need to explicitly signal the up-sampler type and thus free up bandwidth.

In one case, a plurality of different up-sampler types may have a set of configurable coefficients that are supplied in a common or shared format (e.g. as one or more matrices or a multi-dimensional array). For example, a set of cubic, modified cubic or neural network up-samplers may use a kernel that has coefficients stored as a multidimensional array. The values of these coefficients may then determine which type of up-sampler is applied. In this manner, an up-sampler may be changed by changing the kernel coefficient values that are signalled to the enhancement decoder. This again may avoid the need to explicitly signal the up-sampler type, and efficiencies in the up-sampler definitions may be shared by multiple up-sampler types (e.g. optimisations within compiled computer program code).

Bitstream

An example bitstream as generating by the video coding frameworks described herein may contain a base layer, which may be at a lower resolution, and an enhancement layer consisting of up to two sub-layers. The following subsection briefly explains the structure of this bitstream and how the information can be extracted.

The base layer can be created using any video encoder and is may be flexibly implemented using a wide variety of existing and future video encoding technologies. The bitstream from the base layer may resemble a bitstream as output by an existing codec. The enhancement layer has an additional different structure. Within this structure, syntax elements are encapsulated in a set of network abstraction layer (NAL) units. These also enable synchronisation of the enhancement layer information with the base layer decoded information (e.g. at a decoder so as to reconstruct a video). Depending on the position of a frame of video within a group of pictures (GOP), additional data specifying the global configuration and for controlling the decoder may be present.

As described in the examples herein, and as shown in Figures 9A, 21A and 21B, the data of one enhancement picture may be encoded as several chunks. These data chunks may be hierarchically organised as shown in the aforementioned Figures. In these examples, for each plane (e.g. corresponding to a colour component), up to two enhancement sub-layers are extracted. Each of them again unfolds into numerous coefficient groups of transform coefficients. The number of coefficients depends on the chosen type of transform (e.g. a 4x4 transform applied to 2x2 coding units may generate 4 coefficients and a 16x16 transform applied to 4x4 coding units may generate 16 coefficients). Additionally, if a temporal processing mode is used, an additional chunk with temporal data for one or more enhancement sub-layers may be present (e.g. one or more of the level 1 and level 2 sub-layers). Entropy-encoded transform coefficients within the enhancement bitstream may be processed at a decoder by the coding tools described herein.

As described herein the terms bitstream, bytestream and stream of NALUs may be used interchangeably. Implementations of examples may only comprise an implementation of the enhancement levels and base layer implementations, such as base encoders and decoders may be implemented by third-party components, wherein an output of a base layer implementation may be received and combined with decoded planes of the enhancement levels, with the enhancement decoding as described herein.

In certain examples, the bitstream can be in one of two formats: a NAL unit stream format or a byte stream format. A NAL unit stream format may be considered conceptually to be the more “basic” type. It consists of a sequence of syntax structures called NAL units. This sequence is ordered in decoding order. There may be constraints imposed on the decoding order (and contents) of the NAL units in the NAL unit stream. The byte stream format can be constructed from the NAL unit stream format by ordering the NAL units in decoding order and prefixing each NAL unit with a start code prefix and zero or more zero-valued bytes to form a stream of bytes. The NAL unit stream format can be extracted from the byte stream format by searching for the location of the unique start code prefix pattern within this stream of bytes.

For bit-oriented delivery, the bit order for the byte stream format may be specified to start with the most significant bit of the first byte, proceed to the least significant bit of the first byte, followed by the most significant bit of the second byte, etc. The byte stream format may consist of a sequence of byte stream NAL unit syntax structures. Each byte stream NAL unit syntax structure may contain one 4-byte length indication followed by one nal_unit(NumBytesInNalUnit) syntax structure. This syntax structure may be as follows:

Syntax	**Descriptor**
byte_stream_nal_unit( ) {	
nal_unit_length	u(32)
nal_unit(nal_unit_length)	
}	

The order of byte stream NAL units in the byte stream may follow a decoding order of the NAL units contained in the byte stream NAL units. The content of each byte stream NAL unit may be associated with the same access unit as the NAL unit contained in the byte stream NAL unit. In the above *nal_unit_length* is a 4-byte length field indicating the length of the NAL unit within the *nal_unit( )* syntax structure.

Relationship between base bitstream and enhancement bitstream

A relationship between the base bitstream and the enhancement bitstream may be realized using one of the two following mechanisms. In a first case, if the base bitstream and the enhancement bitstream are not interleaved, a relationship between the Access Units of the base decoder and the Access Units of the enhancement decoder (i.e. the enhancement layers) may be specified. In a second case, if the base decoder bitstream and the enhancement bitstream are interleaved in a single elementary stream, a relationship may be realized by interleaving the Access Units of the base bitstream and the Access Units of the enhancement bitstream.

For example, in the first case, the relationship may be specified using the interleaving and synchronization mechanisms specified by International Standard (IS) 13818-1 Program Stream or the interleaving and synchronization mechanisms specified by IS 14496-14 File Format. In the second case, the interleaving of base Access Units and corresponding enhancement Access Units may be implemented with a number of constraints. These constraints may comprise one or more of: the order of Access Units in the input base bitstream is preserved in the interleaved base and enhancement bitstream; the enhancement Access Unit associated to the corresponding base Access Unit is inserted immediately after the base Access Unit and immediately before the following base Access Unit in bitstream order; the discrimination between Access Units belonging to the base bitstream and Access Units belonging to the enhancement bitstream is realized by means of the NAL unit types, as described with respect to later examples; and the enhancement decoder infers that the residuals obtained from decoding the enhancement Access Unit are to be processed in combination with the samples of the base picture obtained from decoding the immediately preceding base Access Unit.

Payload Processing

A payload data block unit process may be applied to the input bitstream. The payload data block unit process may comprise separating the input bitstream into data blocks, where each data block is encapsulated into a NALU. The NALU may be used as described above to synchronise the enhancement levels with the base level. Each data block may comprise a header and a payload. The payload data block unit may comprise parsing each data block to derive a header and a payload where the header comprises configuration metadata to facilitate decoding and the payload comprises encoded data. A process for decoding the payload of encoded data may comprise retrieving a set of encoded data and this may be performed following the decoding process for a set of headers. Payloads may be processed based on the structure shown in one or more of Figures 9A, 21A and 21B, e.g. a set of entropy encoded coefficients grouped be plane, levels of enhancement or layers. As mentioned, each picture of each NALU may be preceded by picture configuration payload parameters.

It is noted for example that each layer is a syntactical structure containing encoded data related to a specific set of transform coefficients. Thus, each layer may comprise, e.g. where a 2x2 transform is used, a set of 'average' values for each block (or coding unit), a set of 'horizontal' values for each block, a set of 'vertical' for each block and a set of 'diagonal' values for each block. Of course, it will be understood that the specific set of transform coefficients that are comprised in each layer will relate to the specific transform used for that particular level of enhancement (e.g. first or further, level 1 or 2, defined above).

Bitstream Syntax

In certain examples, the bitstreams described herein (e.g. in particular, the enhancement bitstream) may be configured according to a defined. This section presents an example syntax that may be used. The example syntax may be used for interpreting data and may indicate possible processing implementations to aid understanding of the examples described herein. It should be noted that the syntax described below is not limiting, and that different syntax to that presented below may be used in examples to provide the described functionality.

In general, a syntax may provide example methods by which it can be identified what is contained within a header and what is contained within data accompanying the header. The headers may comprise headers as illustrated in previous examples, such as headers 256, 556, 2402, 2566 or 2666. The syntax may indicate what is represented but not necessarily how to encode or decode that data. For example, with relation to a specific example of an up-sample operation, the syntax may describe that a header comprises an indicator of an up-sample operation selected for use in the broader encoding operation, i.e. the encoder side of the process. It may also be indicated where that indication is comprised in the header or how that indicator can be determined. As well as the syntax examples described below, a decoder may also implement components for identifying entry points into the bitstream, components for identifying and handling non-conforming bitstreams, and components for identifying and handling errors.

The table below provides a general guide to how the example syntax is presented. When a syntax element appears, it is indicated via a variable such as *syntax_element*; this specifies that a syntax element is parsed from the bitstream and the bitstream pointer is advanced to the next position beyond the syntax element in the bitstream parsing process. The letter "D" indicates a descriptor, which is explained below. Examples of syntax are presented in a most significant bit to least significant bit order.

General Guide - Syntax Specification	**D**
/* A statement can be a syntax element with an associated descriptor or can be an expression used to specify conditions for the existence, type and quantity of syntax elements, as in the following two examples */	
syntax_element	u(n)
conditioning statement	
/* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. */	
{	
Statement	
Statement	
…	
}	

General Guide - Syntax Specification	**D**
/* A “while” structure specifies a test of whether a condition is true, and if true, specifies evaluation of a statement (or compound statement) repeatedly until the condition is no longer true */	
while (condition)	
Statement	
/* A “do ... while” structure specifies evaluation of a statement once, followed by a test of whether a condition is true, and if true, specifies repeated evaluation of the statement until the condition is no longer true */	
do	
Statement	
while (condition)	
/* An “if ... else” structure specifies a test of whether a condition is true and, if the condition is true, specifies evaluation of a primary statement, otherwise, specifies evaluation of an alternative statement. The “else” part of the structure and the associated alternative statement is omitted if no alternative statement evaluation is needed */	
if (condition)	
primary statement	
else	
alternative statement	
/* A “for” structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. */	
for (initial statement; condition; subsequent statement)	
primary statement	

In the examples of syntax, functions are defined as set out in the table below. Functions are expressed in terms of the value of a bitstream pointer that indicates the position of the next bit to be read by the decoding process from the bitstream.

Syntax function	Use
byte_stream_has_data( )	If the byte-stream has more data, then returns TRUE; otherwise returns FALSE.
process_payload_function(payload_type, payload_byte_size)	Behaves like a function lookup table, by selecting and invoking the process payload function relating to the payload_type.
read_bits(n)	Reads the next n bits from the bitstream. Following the read operation, the bitstream pointer is advanced by n bit positions. When n is equal to 0, read_bits(n) returns a value equal to 0 and the bitstream pointer is not advanced.
read_byte(bitstream)	Reads a byte in the bitstream returning its value. Following the return of the value, the bitstream pointer is advanced by a byte.
read_multibyte(bitstream)	Executes a read_byte(bitstream) until the MSB of the read byte is equal to zero.
bytestream_current(bitstream)	Returns the current bitstream pointer.
bytestream_seek(bitstream, n)	Returns the current bitstream pointer at the position in the bitstream corresponding to n bytes.

The following descriptors, which may be used in the “D” column of the example tables, specify the parsing process of each syntax element:

b(8): byte having any pattern of bit string (8 bits). The parsing process for this descriptor is specified by the return value of the function read_bits(8).

f(n): fixed-pattern bit string using n bits written (from left to right) with the left bit first. The parsing process for this descriptor is specified by the return value of the function read_bits(n).

u(n): unsigned integer using n bits. When n is “v” in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read_bits(n) interpreted as a binary representation of an unsigned integer with most significant bit written first.

ue(v): unsigned integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified later examples.

mb: read multiple bytes. The parsing process for this descriptor is specified by the return value of the function read_multibyte(bitstream) interpreted as a binary representation of multiple unsigned char with most significant bit written first, and most significant byte of the sequence of unsigned char written first.

NAL unit and NAL unit header syntax

NAL unit and NAL unit header syntax may be configured as set out in the respective two tables below:

Syntax	**D**
nal_unit(NumBytesInNALunit) {	
nal_unit_header( )	
NumBytesInRBSP = 0	
for (i = 2; i < NumBytesInNALunit; i++) {	
if (i + 2 < NumBytesInNALunit && next_bits(24) == 0x000003) {	
rbsp_byte[NumBytesInRBSP++]	u(8)
rbsp_byte[NumBytesInRBSP++]	u(8)
i += 2	
emulation_prevention_three_byte /* equal to 0x03 */	u(8)
} else	
rbsp_byte[NumBytesInRBSP++]	u(8)
}	
}	

Syntax	**D**
nal_unit_header( ) {	
forbidden_zero_bit	u(1)

Syntax	D
forbidden_one_bit	u(1)
nal_unit_type	u(5)
reserved_flag	u(9)
}	

Process Block Syntax

An example process block syntax is set out in the table below:

Syntax	D
process_block( ) {	
payload_size_type	u(3)
payload_type	u(5)
payload_size = 0	
if (payload_size_type == 7) {	
custom_byte_size	mb
payload_size = custom_byte_size	
} else {	
if (payload_size_type == 0) payload_size = 0	
if (payload_size_type == 1) payload_size = 1	
if (payload_size_type == 2) payload_size = 2	
if (payload_size_type == 3) payload_size = 3	
if (payload_size_type == 4) payload_size = 4	
if (payload_size_type == 5) payload_size = 5	
}	
if (payload_type == 0)	
process_payload_sequence_config(payload_size)	
else if (payload_type == 1)	
process_payload_global_config(payload_size)	
else if (payload_type == 2)	
process_payload_picture_config(payload_size)	
else if (payload_type == 3)	
process_payload_encoded_data(payload_size)	
else if (payload_type == 4)	

Syntax	D
process_payload_encoded_data_tiled(payload_size)	
else if (payload_type == 5)	
process_payload_additional_info(payload_size)	
else if (payload_type == 6)	
process_payload_filler(payload_size)	
}	

Process Payload – Sequence Configuration

A process payload sequence configuration syntax may be as set out in the table below:

Syntax	D
process_payload_sequence_config(payload_size) {	
profile_idc	u(4)
level_idc	u(4)
sublevel_idc	u(2)
conformance_window_flag	u(1)
reserved_zeros_5bit	u(5)
if (profile_idc == 16 || level_idc == 16) {	
extended_profile_idc	u(3)
extended_level_idc	u(4)
reserved_zeros_1bit	u(1)
}	
if (conformance_window_flag == 1) {	
conf_win_left_offset	mb
conf_win_right_offset	mb
conf_win_top_offset	mb
conf_win_bottom_offset	mb
}	
}	

Process Payload – Global Configuration

A process payload global configuration syntax may be as set out in the table below:

Syntax	**D**
process_payload_global_config(payload_size) {	
processed_planes_type_flag	u(1)
resolution_type	u(6)
transform_type	u(1)
chroma_sampling_type	u(2)
base_depth_type	u(2)
enhancement_depth_type	u(2)
temporal_step_width_modifier_signalled_flag	u(1)
predicted_residual_mode_flag	u(1)
temporal_tile_intra_signalling_enabled_flag	u(1)
temporal_enabled_flag	u(1)
upsample_type	u(3)
level_1_filtering_signalled_flag	u(1)
scaling_mode_level1	u(2)
scaling_mode_level2	u(2)
tile_dimensions_type	u(2)
user_data_enabled	u(2)
level1_depth_flags	u(1)
reserved_zeros_1bit	u(1)
if (temporal_step_width_modifier_signalled_flag == 1) {	
temporal_step_width_modifier	u(8)
} else {	
temporal_step_width_modifier = 48	
}	
if (level_1_filtering_signalled_flag) {	
level_1_filtering_first_coefficient	u(4)
level_1_filtering_second_coefficient	u(4)
}	
if (tile_dimensions_type > 0) {	
if (tile_dimensions_type == 3) {	
custom_tile_width	u(16)

Syntax	D
custom_tile_height	u(16)
}	
reserved_zeros_5bit	u(5)
compression_type_entropy_enabled_per_tile_flag	u(1)
compression_type_size_per_tile	u(2)
}	
if (resolution_type == 63) {	
custom_resolution_width	u(16)
custom_resolution_height	u(16)
}	
}	

Process Payload – Picture Configuration

A process payload picture configuration syntax, e.g. for a frame of video, may be as set out in the table below:

Syntax	D
process_payload_picture_config(payload_size) {	
no_enhancement_bit_flag	u(1)
if (no_enhancement_bit_flag == 0) {	
quant_matrix_mode	u(3)
dequant_offset_signalled_flag	u(1)
picture_type_bit_flag	u(1)
temporal_refresh_bit_flag	u(1)
step_width_level1_enabled_flag	u(1)
step_width_level2	u(15)
dithering_control_flag	u(1)
} else {	
reserved_zeros_4bit	u(4)
picture_type_bit_flag	u(1)
temporal_refresh_bit_flag	u(1)
temporal_signalling_present_flag	u(1)
}	

Syntax	D
if (picture_type_bit_flag == 1) {	
field_type_bit_flag	u(1)
reserved_zeros_7bit	u(7)
}	
if (step_width_level1_enabled_flag == 1) {	
step_width_level1	u(15)
level_1_filtering_enabled_flag	u(1)
}	
if (quant_matrix_mode == 2 || quant_matrix_mode == 3 || quant_matrix_mode == 5) {	
for(layerIdx = 0; layerIdx < nLayers; layerIdx++) {	
qm_coefficient_0[layerIdx]	u(8)
}	
}	
if (quant_matrix_mode == 4 || quant_matrix_mode == 5) {	
for(layerIdx = 0; layerIdx < nLayers; layerIdx++) {	
qm_coefficient_1[layerIdx]	u(8)
}	
}	
if (dequant_offset_signalled_flag) {	
dequant_offset_mode_flag	u(1)
dequant_offset	u(7)
}	
if (dithering_control_flag == 1) {	
dithering_type	u(2)
reserverd_zero	u(1)
if (dithering_type != 0) {	
dithering_strength	u(5)
} else {	
reserved_zeros_5bit	u(5)
}	
}	

Syntax	D
}	

Process Payload – Encoded Data

A process payload encoded data syntax may be as set out in the table below:

Syntax	D
process_payload_encoded_data(payload_size) {	
if (tile_dimensions_type == 0) {	
for (planeIdx = 0; planeIdx < nPlanes; planeIdx++) {	
if (no_enhancement_bit_flag == 0) {	
for (levelIdx = 1; levelIdx <= 2; levelIdx++) {	
for (layerIdx = 0; layerIdx < nLayers;layerIdx++) {	
surfaces[planeIdx][levelIdx][layerIdx].entropy_enabled_flag	u(1)
surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag	u(1)
}	
}	
}	
if (temporal_signalling_present_flag == 1){	
temporal_surfaces[planeIdx].entropy_enabled_flag	u(1)
temporal_surfaces[planeIdx].rle_only_flag	u(1)
}	
}	
byte_alignment( )	
for (planeIdx = 0; planeIdx < nPlanes; planeIdx++) {	
for (levelIdx = 1; levelIdx <= 2; levelIdx++) {	
for (layerIdx = 0; layerIdx < nLayers; layerIdx++)	
process_surface(surfaces[planeIdx][levelIdx][layerIdx])	
}	
if (temporal_signalling_present_flag == 1)	
process_surface(temporal_surfaces[planeIdx])	
}	
} else {	
process_payload_encoded_data_tiled(payload_size)	

Syntax	D
}	
}	

Process Payload – Encoded Tiled Data

A process payload encoded tiled data syntax may be as set out in the table below:

Syntax	D
process_payload_encoded_data_tiled(payload_size) {	
for (planeIdx = 0; planeIdx < nPlanes; planeIdx++) {	
for (levelIdx = 1; levelIdx <= 2; levelIdx++) {	
if (no_enhancement_bit_flag == 0) {	
for (layerIdx = 0; layerIdx < nLayers;layerIdx++)	
surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag	u(1)
}	
}	
if (temporal_signalling_present_flag == 1)	
temporal_surfaces[planeIdx].rle_only_flag	u(1)
}	
byte_alignment( )	
if (compression_type_entropy_enabled_per_tile_flag == 0) {	
for (planeIdx = 0; planeIdx < nPlanes; planeIdx++) {	
if (no_enhancement_bit_flag == 0) {	
for (levelIdx = 1; levelIdx <= 2; levelIdx++) {	
if (levelIdx == 1)	
nTiles = nTilesL1	
else	
nTiles = nTilesL2	
for (layerIdx = 0; layerIdx < nLayers; layerIdx++) {	
for (tileIdx = 0; tileIdx < nTiles; tileIdx++)	
surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx]. *entropy_enabled_flag*	u(1)
}	
}	

Syntax	D
}	
if (temporal_signalling_present_flag == 1) {	
for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++)	
temporal_surfaces[planeIdx].tiles[tileIdx].entropy_enabled_flag	u(1)
}	
}	
} else {	
entropy_enabled_per_tile_compressed_data_rle	mb
}	
byte_alignment( )	
if (compression_type_size_per_tile == 0) {	
for (planeIdx = 0; planeIdx < nPlanes; planeIdx++) {	
for (levelIdx = 1; levelIdx <= 2; levelIdx++) {	
if (levelIdx == 1)	
nTiles = nTilesL1	
else	
nTiles = nTilesL2	
for (layerIdx = 0; layerIdx < nLayers; layerIdx++) {	
for (tileIdx = 0; tileIdx < nTiles; tileIdx++)	
process_surface(surfaces[planeIdx][levelIdx][layerIdx]. tiles[tileIdx])	
}	
}	
if (temporal_signalling_present_flag == 1) {	
for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++)	
process_surface(temporal_surfaces[planeIdx].tiles[tileIdx])	
}	
}	
} else {	
for (planeIdx = 0; planeIdx < nPlanes; planeIdx++) {	
for (levelIdx = 1; levelIdx <= 2; levelIdx++) {	
if (levelIdx == 1)	

Syntax	**D**
nTiles = nTilesL1	
else	
nTiles = nTilesL2	
for (layerIdx = 0; layerIdx < nLayers; layerIdx++) {	
if(surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag) {	
compressed_size_per_tile_prefix	mb
} else {	
compressed_prefix_last_symbol_bit_offset_per_tile_prefix	mb
compressed_size_per_tile_prefix	mb
}	
for (tileIdx=0; tileIdx < nTiles; tileIdx++)	
process_surface(surfaces[planeIdx][levelIdx][layerIdx]. tiles[tileIdx])	
}	
}	
if (temporal_signalling_present_flag == 1) {	
if(temporal_surfaces[planeIdx].rle_only_flag) {	
compressed_size_per_tile_prefix	mb
} else {	
compressed_prefix_last_symbol_bit_offset_per_tile_prefix	mb
compressed_size_per_tile_prefix	mb
}	
for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++)	
process_surface(temporal_surfaces[planeIdx].tiles[tileIdx])	
}	
}	
}	
}	

Process Payload – Surface

A process payload surface syntax (e.g. a syntax for a set of data that may comprise encoded coefficients and/or temporal signalling) may be as set out in the table below:

Syntax	**D**
process_surface(surface) {	
if (compression_type_size_per_tile == 0) {	
if (surface.entropy_enabled_flag) {	
if (surface.rle_only_flag) {	
surface.size	mb
surface.data	surface.size
} else {	
reserved_zeros_3bit	u(3)
surface.prefix_last_symbol_bit_offset	u(5)
surface.size	mb
surface.data	surface.size
}	
}	
} else {	
if (surface.entropy_enabled_flag) {	
surface.data	surface.size
}	
}	
}	

Process Payload – Additional Information

A process payload additional information syntax may be as set out in the table below:

Syntax	**D**
additional_info(payload_size) {	
additional_info_type	u(8)
if (additional_info_type == 0) {	
payload_type	u(8)
sei_payload(payload_type, payload_size − 2)	
} else if (additional_info_type == 1)	
vui_parameters (payload_size − 1)	
else // (additional_info_type >= 2)	

Syntax	D
// reserved for future use	
}	

Process Payload – Filler

A process payload filler syntax may be as set out in the table below:

Syntax	D
process_payload_filler(payload_size) {	
for(x = 0; x < payload_size; x++) {	
filler_byte // equal to 0xAA	u(8)
}	
}	

Byte Alignment

A byte alignment syntax may be as set out in the table below:

Syntax	D
byte_alignment( ) {	
alignment_bit_equal_to_one /* equal to 1 */	f(1)
while(!byte_aligned( ))	
alignment_bit_equal_to_zero /* equal to 0 */	f(1)
}	

Bitstream Semantics

The section below provides further detail on the meaning of certain variables set out in the tables above. This detail may be referred to as the "semantics" of the bitstream. Example semantics associated with the syntax structures and with the syntax elements within these structures are described in this section. In certain cases, syntax elements may have a closed set of possible values and examples of these cases are presented in certain tables below.

NAL Unit Semantics

A number of examples of variables or parameters that relate generally to a NAL unit will now be described. These should not be seen as limiting.

The variable *NumBytesInNalUnit* may be used to specify the size of the NAL unit in bytes. This value may be used for the decoding of the NAL unit. Some form of demarcation of NAL unit boundaries may be used to enable inference of *NumBytesInNalUnit*. One such demarcation method is described with reference to other examples of the NALU for the byte stream format. A variety of methods of demarcation may be used.

The variable *rbsp_byte[i]* is the *i*-th byte of a raw byte sequence payload (RBSP). An RBSP may be specified as an ordered sequence of bytes and contain a string of data bits (SODB) as follows:

If the SODB is empty (i.e., zero bits in length), the RBSP is also empty.

Otherwise, the RBSP contains the SODB as follows:

1. The first byte of the RBSP contains the (most significant, left-most) eight bits of the SODB; the next byte of the RBSP contains the next eight bits of the SODB, etc., until fewer than eight bits of the SODB remain.
2. *rbsp_trailing_bits( )* are present after the SODB as follows:
    i. The first (most significant, left-most) bits of the final RBSP byte contains the remaining bits of the SODB (if any).
    ii. The next bit consists of a single *rbsp_stop_one_bit* equal to 1.
    iii. When the *rbsp_stop_one_bit* is not the last bit of a byte-aligned byte, one or more *rbsp_alignment_zero_bit* is present to result in byte alignment.

Syntax structures having the above RBSP properties are denoted in the above syntax tables using an "*_rbsp*" suffix. These structures may be carried within NAL units as the content of the *rbsp_byte[i]* data bytes. The association of the RBSP syntax structures to the NAL units may be as set out in the table below. When the boundaries of the RBSP are known, the decoder can extract the SODB from the RBSP by concatenating the bits of the bytes of the RBSP and discarding the *rbsp_stop_one_bit*, which is the last (least significant, right-most) bit equal to 1, and discarding any following (less significant, farther to the right) bits that follow it, which are equal to 0. The data for the decoding process may be contained in the SODB part of the RBSP.

The variable *emulation_prevention_three_byte* is a byte equal to 0x03. When an *emulation_prevention_three_byte* is present in the NAL unit, it may be discarded by the decoding process. In certain cases, the last byte of the NAL unit is prevented from being equal to 0x00 and within the NAL unit, the following three-byte sequences are excluded at any byte-aligned position: 0x000000, 0x000001 and 0x000002. It may also be configured that, within the NAL unit, any four-byte sequence that starts with 0x000003 other than the following sequences may not occur at any byte-aligned position (e.g. the following four-byte sequences 0x00000300, 0x00000301, 0x00000302, and 0x00000303).

NAL Unit Header Semantics

A number of examples of variables or parameters that may be used to carry information relating a NAL unit header will now be described. These should not be seen as limiting.

In certain examples, the variable *forbidden_zero_bit* is set as being equal to 0 and the variable *forbidden_one_bit* is set as being equal to 1. The variable *nal_unit_type* may be used to specify the type of RBSP data structure contained in the NAL unit as specified in the table below:

nal_unit_type	**Name of nal_unit_type**	**Content of NAL unit and RBSP syntax structure**	**NAL unit type class**
0...27	UNSPEC0...UNSPEC27	unspecified	Non-VCL
28	LCEVC_LEVEL	segment of a Low Complexity Enhancement Level	VCL/Non-VCL
29-30	RSV_LEVEL	reserved	Non-VCL
31	UNSPEC31	unspecified	Non-VCL

In this example, NAL units that have *nal_unit_type* in the range of UNSPEC0...UNSPEC27, inclusive, and UNSPEC31 for which semantics are not specified, may be configured to not affect the enhancement decoding process. The *reserved_flag* may be equal to the bit sequence 111111111. NAL unit types in the range of UNSPEC0...UNSPEC27 and UNSPEC31 may be used as determined by a particular application or implementation. These may to relate to "enhancement" decoding processes as described herein, which may be associated with the LCEVC_LEVEL *nal_unit_type*.

Different applications may use NAL unit types in the range of UNSPEC0…UNSPEC27 and UNSPEC31NAL for different purposes, and encoders and decoders may be adapted accordingly. For purposes other than determining the amount of data in the decoding units of the bitstream (e.g. as used in certain text configurations), decoders may be configured to ignore (remove from the bitstream and discard) the contents of all NAL units that use reserved values of *nal_unit_type*. Future compatible extensions to the aspects described herein may use reserved and/or unspecified NAL unit types.

Data Block Unit General Semantics

A number of examples of variables or parameters that may be used to carry information regarding a data block of a NAL unit will now be described. These should not be seen as limiting.

The variable *payload_size_type* may be used to specify the size of the payload. It may take a value between 0 and 7, as specified by the table below.

payload_size_type	**Size (bytes)**
0	0
1	1
2	2
3	3
4	4
5	5
6	Reserved
7	Custom

The variable *payload_type* may specify the type of the payload used (e.g. the content of the payload). It may take a value between 0 and 31, as specified by the table below. The table also indicates a suggested minimum frequency of appearance of each content within an example bitstream.

payload_type	Content of payload	Minimum frequency
0	process_payload_sequence_config( )	at least once
1	process_payload_global_config( )	at least every IDR
2	process_payload_picture_config( )	picture

payload_type	Content of payload	Minimum frequency
3	process_payload_encoded_data( )	picture
4	process_payload_encoded_data_tiled( )	picture
5	process_payload_additional_info( )	picture
6	process_payload_filler( )	picture
7-30	Reserved	
31	Custom	

Data Block Semantics

The following describes the semantics for each of the data block units, e.g. the data that is carried by the NAL units. Certain variables discussed below relate to profiles, levels and toolsets. Profiles, levels and toolsets may be used to specify restrictions on the bitstreams and hence apply limits to the capabilities needed to decode the bitstreams. Profiles, levels and toolsets may also be used to indicate interoperability points between individual decoder implementations. It may be desired to avoid individually selectable "options" at the decoder, as this may increase interoperability difficulties.

A "profile" may specify a subset of algorithmic features and limits that are supported by all decoders conforming to that profile. In certain case, encoders may not be required to make use of any particular subset of features supported in a profile.

A "level" may specify a set of limits on the values that may be taken by the syntax elements (e.g. the elements described above). The same set of level definitions may be used with all profiles, but individual implementations may support a different level for each supported profile. For any given profile, a level may generally correspond to a particular decoder processing load and memory capability. Implementations of video decoders conforming to the examples described herein may be specified in terms of the ability to decode video streams conforming to the constraints of profiles and levels, e.g. the profiles and/or levels may indicate a certain specification for a video decoder, such as a certain set of features that are supported and/or used. As such, the capabilities of a particular implementation of a decoder may be specified using a profile, and a given level for that profile. The variable *profile_idc* may be used to indicate a profile for the bitstream and the variable *level_idc* may be used to indicate a level. The values for these variables may be restricted to a set of defined specifications. A reserved value of *profile_idc* between a set of specified values may not indicate intermediate capabilities between the specified profiles; however, a reserved value of *level_idc* between a set of specified values may be used to indicated intermediate capabilities between the specified levels. The variable *sublevel_idc* may also be used to indicate a sublevel for a set of capabilities. These levels and sublevels are not to be confused with the levels and sublevels of the enhancement encoders and decoders, which are a different concept.

As an example, there may be a "main" profile. Conformance of a bitstream to this example "main" profile may be indicated by *profile_idc* equal to 0. Bitstreams conforming to this example "main" profile may have the constraint that active global configuration data blocks have *chroma_sampling_type* equal to 0 or 1 only. All constraints for global configuration parameter sets that are specified may be constraints for global configuration parameter sets that are activated when the bitstream is decoded. Decoders conforming to the present example "main" profile at a specific level (e.g. as identified by a specific value of *level_idc*) may be capable of decoding all bitstreams and sublayer representations for which all of the following conditions apply: the bitstream is indicated to conform to the "main" profile and the bitstream or sublayer representation is indicated to conform to a level that is lower than or equal to the specified level. Variations of this example "main" profile may also be defined and given differing values of *profile_idc*. For example, there may be a "main 4:4:4" profile. Conformance of a bitstream to the example "main 4:4:4" profile may be indicated by *profile_idc* equal to 1. Bitstreams conforming to the example "main 4:4:4" profile may have the constraint that active global configuration data blocks shall have *chroma_sampling_type* in the range of 0 to 3, inclusive. Again, decoders conforming to the example "main 4:4:4" profile at a specific level (e.g. as identified by a specific value of *level_idc*) may be capable of decoding all bitstreams and sublayer representations for which all of the following conditions apply: the bitstream is indicated to conform to the "main" profile and the bitstream or sublayer representation is indicated to conform to a level that is lower than or equal to the specified level. The variables *extended_profile_idc* and *extended_level_idc* may be respectively used to indicate that an extended profile and an extended level are used.

In certain implementation, the "levels" associated with a profile may be defined based on two parameters: a count of luma samples of output picture in time (i.e. the Output Sample Rate) and maximum input bit rate for the Coded Picture Buffer for the enhancement coding (CPBL). Both sample rate and bitrate may be considered on observation periods of one second (e.g. the maximum CPBL bit rate may be measured in terms of bits per second per thousand Output Samples). The table below indicates some example levels and sublevels.

Level	Sublevel	Maximum Output Sample Rate	Maximum CPBL bit rate	Example Resolution and Frame Rate
1	0	29,410,000	4	1280x720 (30fps)
1	1	29,410,000	40	1280x720 (30fps)
2	0	124,560,000	4	1920x1080 (60fps)
2	1	124,560,000	40	1920x1080 (60fps)
3	0	527,650,000	4	3840x2160 (60 fps)
3	1	527,650,000	40	3840x2160 (60 fps)
4	0	2,235,160,000	4	7640x4320 (60fps)
4	1	2,235,160,000	40	7640x4320 (60fps)

Returning to further variables of the NAL unit data block, if the variable *conformance_window_flag* is equal to 1 this may be used to indicate that conformance cropping window offset parameters are present in the sequence configuration data block. If the variable *conformance_window_flag* is equal to 0 this may indicate that the conformance cropping window offset parameters are not present. The variables *conf_win_left_offset*, *conf_win_right_offset*, *conf_win_top_offset* and *conf_win_bottom_offset* specify the samples of the pictures in the coded video sequence that are output from the decoding process (i.e. the resulting output video), in terms of a rectangular region specified in picture coordinates for output. When *conformance_window_flag* is equal to 0, the values of *conf_win_left_offset*, *conf_win_right_offset*, *conf_win_top_offset* and *conf_win_bottom_offset* may be inferred to be equal to 0. The conformance cropping window may be defined to contain the luma samples with horizontal picture coordinates from (*SubWidthC* * *conf_win_left_offset*) to (*width* − (*SubWidthC* * *conf_win_right_offset* + 1)) and vertical picture coordinates from (*SubHeightC* * *conf_win_top_offset* to *height* − (*SubHeightC* * conf_win_bottom_offset + 1)), inclusive. The value of *SubWidthC* * (*conf_win_left_offset* + *conf_win_right_offset*) may be constrained to be less than *width*, and the value of *SubHeightC* * (*conf_win_top_offset* + *conf_win_bottom_offset*) may be constrained to be less than *height*. The corresponding specified samples of the two chroma arrays (e.g. in a YUV example) may be similarly defined as the samples having picture coordinates (*x* / *SubWidthC*, *y* /

SubHeightC), where (*x*, *y*) are the picture coordinates of the specified luma samples. Example value of *SubWidthC* and *SubHeightC* are indicated in the "Example Picture Formats" section above. Note that the conformance cropping window offset parameters may only be applied at the output; all internal decoding processes may be applied to the uncropped picture size.

Data Block Unit Global Configuration Semantics

A short description of certain global configuration variables as indicated in the above syntax will now be described. A number of examples of variables or parameters that may be used to carry information regarding the global configuration will be described. These should not be seen as limiting.

The variable *processed_planes_type_flag* may be used to specify the plane to be processed by the decoder. It may be equal to 0 or 1. For a YUV examples, if it is equal to 0, only the Luma (Y) plane may be processed; if it is equal to 1, all planes (e.g. one luma and two chroma) may be processed. In this case, if the *processed_planes_type_flag* is equal to 0, *nPlanes* shall be equal to 1 and if *processed_planes_type_flag* is equal to 1, *nPlanes* shall be equal to 3. An illustration of the variable *nPlanes* is shown in Figure 9A.

The variable *resolution_type* may be used to specify the resolution of a Luma (Y) plane of the enhanced decoded picture. It may be defined as a value between 0 and 63, as specified in the table below. The value of the type is expressed as *N*x*M*, where *N* is the width of the Luma (Y) plane of the enhanced decoded picture and *M* is height of the Luma (Y) plane of the enhanced decoded picture. For example, the following values (amongst others) may be available:

resolution_type	**Value of type**
0	unused /* Escape code prevention */
1	360x200
2	400x240
3	480x320
4	640x360
5	640x480
6	768x480
7	800x600
…	…

resolution_type	Value of type
45	5120x3200
46	5120x4096
47	6400x4096
48	6400x4800
49	7680x4320
50	7680x4800
51-62	Reserved
63	Custom

The variable *chroma_sampling_type* defines the colour format for the enhanced decoded picture as set out in the table in the "Example Picture Formats" section.

The variable *transform_type* may be used to define the type of transform to be used. For example, the following values (amongst others) may be available:

transform_type	Value of type
0	2x2 directional decomposition transform
1	4x4 directional decomposition transform

In the example above, if *transform_type* is equal to 0, *nLayers* (e.g. as shown in Figure 9A) may be equal to 4 and if *transform_type* is equal to 1, *nLayers* may be equal to 16.

The variable *base_depth_type* may be used to define the bit depth of the decoded base picture. For example, the following values (amongst others) may be available:

base_depth_type	Value of type
0	8
1	10
2	12
3	14

Similarly, the variable *enhancement_depth_type* may be used to define the bit depth of the enhanced decoded picture. For example, the following values (amongst others) may be available:

enhancement_depth_type	**Value of type**
0	8
1	10
2	12
3	14

The variable *temporal_step_width_modifier_signalled_flag* may be used to specify if the value of the *temporal_step_width_modifier* parameter is signalled. It may be equal to 0 or 1. If equal to 0, the *temporal_step_width_modifier* parameter may not be signalled.

The variable *predicted_residual_mode_flag* may be used to specifie whether the decoder should activate the predicted residual process during the decoding process. If the value is 0, the predicted residual process shall be disabled.

The variable *temporal_tile_intra_signalling_enabled_flag* may be used to specify whether temporal tile prediction should be used when decoding a tile (e.g. a 32x32 tile). If the value is 1, the temporal tile prediction process shall be enabled.

The variable *upsample_type* may be used to specify the type of up-sampler to be used in the decoding process. For example, the following values may be available:

upsample_type	**Value of type**
0	Nearest
1	Linear
2	Cubic
3	Modified Cubic
4-6	Reserved
7	Custom

The variable *level_1_filtering_signalled* may be used to specify whether a deblocking filter should use a set of signalled parameters, e.g. instead of default parameters. If the value is equal to 1, the values of the deblocking coefficients may be signalled.

The variable *temporal_step_width_modifier* may be used to specify a value to be used to calculate a variable step width modifier for transforms that use temporal prediction. If *temporal_step_width_modifier_signalled_flag* is equal to 0, this variable may be set to a predefined value (e.g. 48).

The variable *level_1_filtering_first_coefficient* may be used to specify the value of the first coefficient in the deblocking mask (e.g. α or the 4x4 block corner residual weight in the example from the earlier sections above). The value of the first coefficient may be between 0 and 15.

The variable *level_1_filtering_second_coefficient* may be used to specify the value of the second coefficient in the deblocking mask (e.g. β or the 4x4 block side residual weight in the example from the earlier sections above). The value of the second coefficient may be between 0 and 15.

The variable *scaling_mode_level1* may be provided to specify whether and how the up-sampling process should be performed between decoded base picture and preliminary intermediate picture (e.g. up-scaler 2608 in Figure 26). The scaling mode parameter for level 1 (e.g. to convert from level 0 to level 1) may have a number of possible values including:

scaling_mode_level1	**Value of type**
0	no scaling
1	one-dimensional 2:1 scaling only across the horizontal dimension
2	two-dimensional 2:1 scaling across both dimensions
3	Reserved

A similar variable *scaling_mode_level2* may be used to specify whether and how the up-sampling process is be performed between combined intermediate picture and preliminary output picture (e.g. as per up-scaler 2687 in Figure 26). The combined intermediate picture corresponds to the output of process 8.9.1. The scaling mode parameter for level 2 (e.g. to convert from level 1 to level 2) may have a number of possible values including:

scaling_mode_level2	**Value of type**
0	no scaling
1	one-dimensional 2:1 scaling only across the horizontal dimension
2	two-dimensional 2:1 scaling across both dimensions
3	Reserved

As described in the section title "User Data Signalling" above, the variable *user_data_enabled* may be used to specify whether user data are included in the bitstream and the size of the user data. For example, this variable may have the following values:

user_data_enabled	**Value of type**
0	disabled
1	enabled 2-bits
2	enabled 6-bits
3	reserved

Variables may also be defined to indicate the bit depth of one or more of the base layer and the two enhancement sub-layers. For example, the variable *level1_depth_flag* may be used to specify whether the encoding and/or decoding components at level 1 process data using the base depth type or the enhancement depth type (i.e. according to a base bit depth or a bit depth defined for one or more enhancement levels). In certain cases, the base and enhancement layers may use different bit depths. It may also be possible for level 1 and level 2 processing to be performed at different bit depths (e.g. level 1 may use a lower bit depth than level 2 as level 1 may accommodate a lower level of bit quantization or level 2 may use a lower bit depth to reduce a number of bytes used to encode the level 2 residuals). In a case where a variable such as *level1_depth_flag* is provided, then a value of 0 may indicate that the level 1 sub-layer is to be processed using the base depth type. If a value of 1 is used, this may indicate that the level 1 sub-layer shall be processed using the enhancement depth type.

A variable *tile_dimensions_type* may be specified to indicate the resolution of the picture tiles. Example values for this variable are shown in the table below. The value of the type may be mapped to an NxM resolution, where N is the width of the picture tile and M is height of the picture tile.

tile_dimensions_type	**Value of type**
0	no tiling
1	512x256
2	1024x512
3	Custom

As indicated by type "3" above, in certain cases a custom tile size may be defined. If a custom tile size is indicated (e.g. via a value of 3 in the table above), the variables *custom_tile_width* and *custom_tile_height* may be used to specify a custom width and height for the tile.

One or variables may be defined to indicate a compression method for data associated with a picture tile. The compression method may be applied to signalling for the file. For example, the *compression_type_entropy_enabled_per_tile_flag* may be used to specify the compression method used to encode the *entropy_enabled_flag* field of each picture tile. It may take values as shown in the table below.

compression_type_entropy_enabled_per_tile_flag	**Value of type**
0	No compression used
1	Run length encoding

Similarly, a variable *compression_type_size_per_tile* may be defined to indicate a compression method used to encode the size field of each picture tile. In this case, the *compression_type_size_per_tile* may take the values indicated in the table below (where the terms Huffman Coding and Prefix Coding are used interchangeably).

compression_type_size_per_tile	**Value of type**
0	No compression used
1	Prefix Coding encoding
2	Prefix Coding encoding on differences
3	Reserved

Lastly, the variables *custom_resolution_width* and *custom_resolution_height* may be used to respectively specify the width and height of a custom resolution.

Data Block Unit Picture Configuration Semantics

A number of examples of variables or parameters that may be used to carry information regarding a picture configuration will now be described. These should not be seen as limiting.

In certain examples, a variable may be defined to indicate that certain layers are not to feature enhancement. This may indicate that the enhancement layer is effectively turned off or disabled for certain pictures. For example, if there is network congestion it may be desirable to turn off the enhancement layer for a number of frames and so not receive and add any enhancement data (e.g. not add one or more of the first set and the second set of the decoded residuals). In certain examples, a *no_enhancement_bit_flag* variable may be specified to indicate that there are no enhancement data for all *layerIdx* < *nLayers* in the picture (e.g. as shown with respect to Figure 9A). A *no_enhancement_bit_flag* value of 0 may indicate that enhancement is being used and that there is enhancement data.

As described in other examples herein, a quantization matrix may be used to instruct quantization and/or dequantization. For dequantization at the decoder, signalling may be provided that indicates a quantization matrix mode, e.g. a particular mode of operation for generating and using one or more quantization matrices. For example, a variable such as *quant_matrix_mode* may be used to specify how a quantization matrix is to be used in the decoding process in accordance with the table below. In certain cases, when *quant_matrix_mode* is not present, i.e. when a mode is not explicitly signalled, the mode may be assumed to take a default value, e.g. be inferred to be equal to 0 as indicated below. By allowing the quantization matrix mode value to be absent, signalling bandwidth for each picture may be saved (e.g. the quantization components of the decoder may use a default setting). Use of modes such as indicated in the examples below may allow for efficient implementation of quantization control, whereby quantization parameters may be varied dynamically in certain cases (e.g. when encoding has to adapt to changing conditions) and retrieved based on default values in other cases. The examples in the table below are not intended to be limiting, and other modes may be provided for as indicated with respect to other examples described herein.

quant_matrix_mode	**Value of type**
0	each enhancement sub-layer uses the matrices used for the previous frame, unless the current picture is an instantaneous decoding refresh - IDR - picture, in which case both enhancement sub-layers use default matrices
1	both enhancement sub-layers use default matrices
2	one matrix of modifiers is signalled and should be used on both residual plane
3	one matrix of modifiers is signalled and should be used on enhancement sub-layer 2 residual plane

quant_matrix_mode	**Value of type**
4	one matrix of modifiers is signalled and should be used on enhancement sub-layer 1 residual plane
5	two matrices of modifiers are signalled – the first one for enhancement sub-layer 2 residual plane, the second for enhancement sub-layer 1 residual plane
6-7	Reserved

As described above, in certain examples a quantization offset may be used. For dequantization at the decoder, a quantization offset (also referred to as a dequantization offset for symmetrical quantization and dequantization) may be signalled by the encoder or another control device or may be retrieved from local decoder memory. For example, a variable *dequant_offset_signalled_flag* may be used to specify if the offset method and the value of the offset parameter to be applied when dequantizing is signalled. In this case, if the value is equal to 1, the method for dequantization offset and/or the value of the dequantization offset parameter may be signalled. When *dequant_offset_signalled_flag* is not present, it may be inferred to be equal to 0. Again, having an inferred value for its absence may help reduce a number of bits that need to be sent to encode a particular picture or frame.

Following from the above, the variable *dequant_offset_mode_flag* may be used to specify the method for applying dequantization offset. For example, different modes may be used to indicate different methods of applying the offset. One mode, which may be a default mode, may involve using a signalled *dequant_offset* variable that specifies the value of the dequantization offset parameter to be applied. This may vary dynamically. In one case, if the *dequant_offset_mode_flag* is equal to 0, the aforementioned default mode is applied; if the value of *dequant_offset_mode_flag* is equal to 1, a constant-offset method applies, which may also use the signalled *dequant_offset* parameter. The value of the dequantization offset parameter *dequant_offset* may be, in certain implementations, between 0 and 127, inclusive.

Further quantization variables may also be used. In one case, a set of variables may be used to signal one or more quantization step-widths to use for a picture or frame within the enhancement layer. The step-width values may be used to apply quantization and/or dequantization as explained with respect to the quantization and/or dequantization components of the above examples. For example, *step_width_level1* may be used to specify the value of the step-width to be used when decoding the encoded residuals in enhancement sub-layer 1 (i.e. level 1) and *step_width_level2* may be used to specify the value of the step-width value to be used when decoding the encoded residuals in enhancement sub-layer 2 (i.e. level 2).

In certain examples, a step-width may be defined for one or more of the enhancement sub-layers (i.e. levels 1 and 2). In certain cases, a step-width may be signalled for certain sub-layers but not others. For example, a *step_width_level1_enabled_flag* variable may be used to specify whether the value of the step-width to be used when decoding the encoded residuals in the enhancement sub-layer 1 (i.e. level 1 as described herein) is a default value or is signalled (e.g. from the encoder). It may be either 0 (default value) or 1 (to indicate that the value is signalled by *step_width_level1*). An example default value may be 32,767. When *step_width_level1_enabled_flag* is not present, it is inferred to be equal to 0.

In certain examples, a set of arrays may be defined to specify a set of quantization scaling parameters. The quantization scaling parameters may indicate how to scale each coefficient within a coding unit or block (e.g. for a 2x2 transform how to scale each of the four layers representing A, H, V and D components). In one example, an array *qm_coefficient_0*[*layerIdx*] may be defined to specify the values of the quantization matrix scaling parameter when *quant_matrix_mode* is equal to 2, 3 or 5 in the table above and an array *qm_coefficient_1*[*layerIdx*] may be used to specify the values of the quantization matrix scaling parameter when *quant_matrix_mode* is equal to 4 or 5 in the table above. The index layerIdx represents a particular layer (e.g. as shown in Figure 9A), which in turn relates to a particular set of coefficients (e.g. one layer may comprise A coefficients etc.).

In examples, a *picture_type_bit_flag* variable may be used to specify whether the encoded data are sent on a frame basis (e.g., progressive mode or interlaced mode) or on a field basis (e.g., interlaced mode). An example of possible values is shown in the table below.

picture_type_bit_flag	**Value of type**
0	Frame
1	Field

If a field picture type is specified (e.g. via a value of 1 from the table above), a further variable may be provided to indicate a particular field. For example, a variable *field_type_bit_flag* may be used to specify, if the *picture_type_bit_flag* is equal to 1, whether the data sent are for top or bottom field. Example values for the *field_type_bit_flag* are shown below.

field_type_bit_flag	**Value of type**
0	Top
1	Bottom

As discussed in the "Temporal Prediction and Signalling" section set out above, a number of variables may be defined to signal temporal prediction configurations and settings to the decoder. Certain variables may be defined at a picture or frame level (e.g. to apply to a particular picture or frame). Some examples are further discussed in this section.

In one case, a *temporal_refresh_bit_flag* variable may be signalled to specify whether the temporal buffer should be refreshed for the picture. If equal to 1, this may instruct the refreshing of the temporal buffer (e.g. the setting of values within the buffer to zero as described above).

In one case, a *temporal_signalling_present_flag* variable may be signalled to specify whether the temporal signalling coefficient group is present in the bitstream. If the *temporal_signalling_present_flag* is not present, it may be inferred to be equal to 1 if *temporal_enabled_flag* is equal to 1 and the *temporal_refresh_bit_flag* is equal to 0; otherwise it may be inferred to be equal to 0.

Lastly, a set of variables may be used to indicate and control filtering within the enhancement layer, e.g. as described with respect to the examples of the Figures. In one case, the filtering that is applied at level 1 (e.g. by filtering component 232, 532, 2426 or 2632 in Figures 2, 5A to 5C, 24 or 26) may be selectively controlled using signalling from the encoder. In one case, signalling may be provided to turn the filtering on and off. For example, a *level1_filtering_enabled_flag* may be used to specify whether the level 1 deblocking filter should be used. A value of 0 may indicate that filtering is disabled and a value of 1 may indicate that filtering is enabled. When *level1_filtering_enabled_flag* is not present, it may be inferred to be equal to 0 (i.e. that filtering is disabled as a default if the flag is not present). Although an example is presented with respect to the filtering of residuals that are decoded in the level 1 enhancement sub-layer, in other examples, (e.g. in addition or instead of), filtering may also be selectively applied to residuals that are decoded in the level 2 enhancement sub-layer. Filtering may be turned off and on, and/or configured according to defined variables, in one or more of the levels using signalling similar to the examples described here.

As described in examples above, in certain examples, dithering may be applied to the output decoded picture. This may involve the application of random values generated by a random number generator to reduce visual artefacts that result from quantization. Dithering may be controlled using signalling information.

In one example, a *dithering_control_flag* may be used to specify whether dithering should be applied. In may be applied in a similar way to the residual filtering control flags. For example, a value of 0 may indicate that dithering is disabled and a value of 1 may indicate that dithering is enabled. When *dithering_control_flag* is not present, it may be inferred to be equal to 0 (e.g. disabled as per the level filtering above). One or more variables may also be defined to specify a range of values the additional random numbers are to have. For example, a variable *dithering_strength* may be defined to specify a scaling factor for random numbers. It may be used to set a range between [-*dithering_strength*, + *dithering_strength*]. In certain examples, it may have a value between 0 and 31.

In certain examples, different types of dithering may be defined and applied. In this case, the dithering type and/or parameters for each dithering type may be signalled from the encoder. For example, a variable *dithering_type* may be used to specify what type of dithering is applied to the final reconstructed picture. Example values of the variable *dithering_type* are set out in the table below.

dithering_type	**Value of type**
0	None
1	Uniform
2-3	Reserved

Data Block Unit Encoded Data Semantics

The following section sets out some examples of how the encoded data may be configured. In certain examples, a portion of encoded data, e.g. that relates to a given coefficient, is referred to as a chunk (e.g. with respect to Figure 9A). The data structures 920 in Figure 9A or 2130 indicated in Figure 21 may be referred to as “surfaces”. Surfaces may be stored as a multi-dimensional array. A first dimension in the multi-dimensional array may indicate different planes and use a plane index – *planeIdx*; a second dimension in the multi-dimensional array may indicate different levels, i.e. relating to the enhancement sub-layers, and use a level index – *levelIdx*; and a third dimension in the multi-dimensional array may indicate different layers, i.e. relating to different coefficients (e.g. different locations within a block of coefficients, which may be referred to as A, H, V and D coefficients for a 2x2 transform), and use a layer index – *layerIdx*. This is illustrated in Figure 9A, where there are *nPlanes*, *nLevels*, and *nLayers* (where these indicate how many elements are in each dimension).

As described with respect to the examples of Figures 25 and 26, in certain cases additional custom layers may be added. For example, temporal signalling may be encoded as a non-coefficient layer in addition to the coefficient layers. Other user signalling may also be added as custom layers. In other cases, separate "surface" arrays may be provided for these uses, e.g. in addition to a main "surfaces" array structured as indicated in Figure 9A.

In certain cases, the "surfaces" array may have a further dimension that indicates a grouping such as the tiles shown in Figure 21A. The arrangement of the "surfaces" array is also flexible, e.g. tiles may be arranged below layers as shown in Figure 21B.

Returning to the examples of the above syntax section, a number of control flags that relate to the surfaces may be defined. One control flag may be used to indicate whether there is encoded data within the surfaces array. For example, a *surfaces[planeIdx][levelIdx][layerIdx].entropy_enabled_flag* may be used to indicate whether there are encoded data in *surfaces[planeIdx][levelIdx][layerIdx]*. Similarly, a control flag may be used to indicate how a particular surface is encoded. For example, a *surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag* may indicate whether the data in *surfaces[planeIdx][levelIdx][layerIdx].*are encoded using only run length encoding or using run length encoding and Prefix (i.e. Huffman) Coding.

If temporal data is configured as an additional set of surfaces, a *temporal_surfaces* array may be provided with a dimensionality that reflects whether temporal processing is performed on one or two enhancement levels. With regard to the example shown in Figure 3A and Figures 24 to 26, a one-dimensional *temporal_surfaces[planeIdx]* array may be provided, where each plane has a different temporal surface (e.g. providing signalling for level 2 temporal processing, where all coefficients use the same signalling). In other examples, with more selective temporal processing the temporal surfaces array may be extended into further dimensions to reflect one or more of different levels, different layers (i.e. coefficient groups) and different tiles.

With regard to the temporal surface signalling of the above syntax examples, similar flag to the other surfaces may be provided. For example, a *temporal_surfaces[planeIdx].entropy_enabled_flag* may be used to indicate whether there are encoded data in *temporal_surfaces[planeIdx]* and a *temporal_surfaces[planeIdx].rle_only_flag* may be used to indicate whether the data in *temporal_surfaces[planeIdx]* are encoded using only run length encoding or using run length encoding and Prefix (i.e. Huffman) Coding.

Data Block Unit Encoded Tiled Data Semantics

Similar variables to those set out above for the surfaces may be used for encoded data that uses tiles. In one case, the encoded tiled data block unit, e.g. tiled data, may have a similar *surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag*. However, it may have an additional dimension (or set of variables) reflecting the partition into tiles. This may be indicated using the data structure *surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx]*. As set out in the examples above, the tiled data may also have a *surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].entropy_enabled_flag* that indicates, for each tile, whether there are encoded data in the respective tiles (e.g. in *surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx]*).

The tiled data structures may also have associated temporal processing signalling that is similar to that described for the surfaces above. For example, *temporal_surfaces[planeIdx].rle_only_flag* may again be used to indicate whether the data in *temporal_surfaces[planeIdx]* are encoded using only run length encoding or using run length encoding and Prefix (i.e. Huffman) Coding. Each tile may have a *temporal_surfaces[planeIdx].tiles[tileIdx].entropy_enabled_flag* that indicates whether there are encoded data in *temporal_surfaces[planeIdx].tiles[tileIdx]*.

Tiled data may have some additional data that relates to the use of tiles. For example, the variable *entropy_enabled_per_tile_compressed_data_rle* may contain the RLE-encoded signalling for each picture tile. A variable *compressed_size_per_tile_prefix* may also be used to specify the compressed size of the encoded data for each picture tile. The variable *compressed_prefix_last_symbol_bit_offset_per_tile_prefix* may be used to specify the last symbol bit offset of Prefix (i.e. Huffman) Coding encoded data. Decoding examples that use this signalling are set out later below.

Data Block Unit Surface Semantics

The higher level "surfaces" array described above may additionally have some associated data structures. For example, the variable *surface.size* may specify the size of the entropy encoded data and *surface.data* may contain the entropy encoded data itself. The variable *surface.prefix_last_symbol_bit_offset* may be used to specify the last symbol bit offset of the Prefix (i.e. Huffman) Coding encoded data.

Data Block Unit Additional Info Semantics

The additional information data structures may be used to communicate additional information, e.g. that may be used alongside the encoded video. Additional information may be defined according to one or more additional information types. These may be indicated via an *additional_info_type* variable. As an example, additional information may be provided in the form of Supplementary Enhancement Information (SEI) messages or Video Usability Information (VUI) messages. Further examples of these forms of additional information are provided with respect to later examples. When SEI messages are used a *payload_type* variable may specify the payload type of an SEI message.

Data Block Unit Filler Semantics

In certain cases, it may be required to fill NAL units with filler. For example, this may be required to maintain a defined constant bit rate when the enhancement layer contains a large number of 0 values (i.e. when the size of the enhancement layer is small, which may be possible depending on the pictures being encoded). A filler unit may be constructed using a constant filler byte value for the payload. The filler byte may be a byte equal to 0xAA.

It should be noted that the example syntax and semantics that are set out above are provided for example only. They may allow a suitable implementation to be constructed. However, it should be noted that variable names and data formats may be varied from those described while maintaining similar functionality. Further, not all features are required and certain features may be omitted or varied depending on the implementation requirements.

Detailed Example Implementation of the Decoding Process

A detailed example of one implementation of the decoding process is set out below. The detailed example is described with reference to the method 2700 of Figure 27. The description below makes reference to some of the variables defined in the syntax and semantics section above. The detailed example may be taken as one possible implementation of the schematic decoder arrangements shown in Figures 2, 5A to 5C, 24, and 26. In particular, the example below concentrates on the decoding aspects for the enhancement layer, which may be seen as an implementation of an enhancement codec. The enhancement codec encodes and decodes streams of residual data. This differs from comparative SVC and SHVC implementations where encoders receive video data as input at each spatial resolution level and decoders output video data at each spatial resolution level. As such, the comparative SVC and SHVC may be seen as the parallel implementation of a set of codecs, where each codec has a video-in / video-out coding structure. The enhancement codecs described herein on the other hand receive residual data and also output residual data at each spatial resolution level. For example, in SVC and SHVC the outputs of each spatial resolution level are not summed to generate an output video – this would not make sense.

As set out in the "Syntax" section above, a syntax maybe defined to process a received bitstream. The "Syntax" section sets out example methods such as retrieving an indicator from a header accompanying data, where the indicator may be retrieved from a predetermined location of the header and may indicate one or more actions according to the syntax of the following sections. As an example, the indicator may indicate whether to perform the step of adding residuals and/or predicting residuals. The indicator may indicate whether the decoder should perform certain operations, or be configured to perform certain operations, in order to decode the bitstream. The indicator may indicate if such steps have been performed at the encoder stage.

General Overview

Turning to the method 2700 of Figure 27A, the input to the presently described decoding process is an enhancement bitstream 2702 (also called a low complexity enhancement video coding bitstream) that contains an enhancement layer consisting of up to two sub-layers. The outputs of the decoding process are: 1) an enhancement residuals planes (sub-layer 1 residual planes) to be added to a set of preliminary pictures that are obtained from the base decoder reconstructed pictures; and 2) an enhancement residuals planes (sub-layer 2 residual planes) to be added to the preliminary output pictures resulting from upscaling, and modifying via predicted residuals, the combination of the preliminary pictures 1 and the sub-layer 1 residual planes.

As described above, and with reference to Figures 9A, 21A and 21B, data may be arranged in chunks or surfaces. Each chunk or surface may be decoded according to an example process substantially similar to described below and shown in the Figures. As such the decoding process operates on data blocks as described in the sections above.

An overview of the blocks of method 2700 will now be set out. Each block is described in more detail in the subsequent sub-sections.

In block 2704 of the method 2700, a set of payload data block units are decoded. This allows portions of the bitstream following the NAL unit headers to be identified and extracted (i.e. the payload data block units).

In block 2706 of the method 2700, a decoding process for the picture receives the payload data block units and starts decoding of a picture using the syntax elements set out above. Pictures may be decoded sequentially to output a video sequence following decoding. Block 2706 extracts a set of (data) surfaces and a set of temporal surfaces as described above. In certain cases, entropy decoding may be applied at this block.

In block 2710 of the method 2700, a decoding process for base encoding data extraction is applied to obtain a set of reconstructed decoded base samples (recDecodedBaseSamples). This may comprise applying the base decoder of previous examples. If the base codec or decoder is implemented separately, then the enhancement codec may instruct the base decoding of a particular frame (including sub-portions of a frame and/or particular planes for a frame). The set of reconstructed decoded base samples (e.g. 2302 in Figure 23) are then passed to block 2712 where an optional first set of upscaling may be applied to generate a preliminary intermediate picture (e.g. 2304 in Figure 23). For example, block 2712 may correspond to up-scaler 2608 of Figure 26. The output of block 2712 is a set of reconstructed level 1 base samples (where level 0 may comprise to the base level resolution).

At block 2714, a decoding process for the enhancement sub-layer 1 (i.e. level 1) encoded data is performed. This may receive variables that indicate a transform size (nTbs), a user data enabled flag (userDataEnabled) and a step-width (i.e. for dequantization), as well as blocks of level 1 entropy-decoded quantized transform coefficients (TransformCoeffQ) and the reconstructed level 1 base samples (recL1BaseSamples). A plane index (IdxPlanes) may also be passed to indicate which plane is being decoded (in monochrome decoding there may be no index). The variables and data may be extracted from the payload data units of the bitstream using the above syntax.

Block 2714 is shown as comprising a number of sub-blocks that correspond to the inverse quantization, inverse transform and level 1 filtering (e.g. deblocking) components of previous examples. At a first sub-block 2716, a decoding process for the dequantization is performed. This may receive a number of control variables from the above syntax that are described in more detail below. A set of dequantized coefficient coding units or blocks may be output. At a second sub-block 2718, a decoding process for the transform is performed. A set of reconstructed residuals (e.g. a first set of level 1 residuals) may be output. At a third sub-block 2720, a decoding process for a level 1 filter may be applied. The output of this process may be a first set of reconstructed and filtered (i.e. decoded) residuals (e.g. 2308 in Figure 23). In certain cases, the residual data may be arranged in *N*x*M* blocks so as to apply an *N*x*M* filter at sub-block 2720.

At block 2730, the reconstructed level 1 base samples and the filtered residuals that are output from block 2714 are combined. This is referred to in the Figure as residual reconstruction for a level 1 block. At output of this block is a set of reconstructed level 1 samples (e.g. 2310 in Figure 23). These may be viewed as a video stream (if multiple planes are combined for colour signals).

At block 2732, a second up-scaling process is applied. This up-scaling process takes a combined intermediate picture (e.g. 2310 in Figure 23) that is output from block 2730 and generates a preliminary output picture (e.g. 2312 in Figure 23). It may comprise an application of the up-scaler 2687 in Figure 26 or any of the previously described up-sampling components.

In Figure 27, block 2732 comprises a number of sub-blocks. At block 2734, switching is implemented depending on a signalled up-sampler type. Sub-blocks 2736, 2738, 2740 and 2742 represent respective implementations of a nearest sample up-sampling process, a bilinear up-sampling process, a cubic up-sampling process and a modified cubic up-sampling process. Sub-blocks may be extended to accommodate new up-sampling approaches as required (e.g. such as the neural network up-sampling described herein). The output from sub-blocks 2736, 2738, 2740 and 2742 is provided in a common format, e.g. a set of reconstructed up-sampled samples (e.g. 2312 in Figure 23), and is passed, together with a set of lower resolution reconstructed samples (e.g. as output from block 2730) to a predicted residuals process 2744. This may implement the modified up-sampling described herein to apply predicted average portions. The output of block 2744 and of block 2732 is a set of reconstructed level 2 modified up-sampled samples (recL2ModifiedUpsampledSamples).

Block 2746 shows a decoding process for the enhancement sub-layer 2 (i.e. level 2) encoded data. In a similar manner to block 2714, it receives variables that indicate a step-width (i.e. for dequantization), as well as blocks of level 2 entropy-decoded quantized transform coefficients (TransformCoeffQ) and the set of reconstructed level 2 modified up-sampled samples (recL2ModifiedUpsampledSamples). A plane index (IdxPlanes) is also passed to indicate which plane is being decoded (in monochrome decoding there may be no index). The variables and data may again be extracted from the payload data units of the bitstream using the above syntax.

Block 2746 comprises a number of temporal prediction sub-blocks. In the present example, temporal prediction is applied for enhancement sub-layer 2 (i.e. level 2). Block 2746 may thus receive further variables as indicated above that relate to temporal processing including the variables *temporal_enabled*, *temporal_refresh_bit*, *temporal_signalling_present*, and *temporal_step_width_modifier* as well as the data structures *TransformTempSig* and *TileTempSig* that provide the temporal signalling data.

Two temporal processing sub-blocks are shown: a first sub-block 2748 where a decoding process for temporal prediction is applied using the *TransformTempSig* and *TileTempSig* data structures and a second sub-block 2750 that applies a tiled temporal refresh (e.g. as explained with reference to the examples of Figures 11A to 13B). Sub-block 2750 is configured to set the contents of a temporal buffer to zero depending on the refresh signalling.

At sub-blocks 2752 and 2756, decoding processes for the dequantization and transform are applied to the level 2 data in a similar manner to sub-blocks 2718 and 2720 (the latter being applied to the level 1 data). A second set of reconstructed residuals that are output from the inverse transform processing at sub-block 2756 are then added at sub-block 2756 to a set of temporally predicted level 2 residuals that are output from sub-block 2748; this implements part of the temporal prediction. The output of block 2746 is a set of reconstructed level 2 residuals (resL2Residuals).

At block 2758, the reconstructed level 2 residuals (resL2Residuals) and the reconstructed level 2 modified up-sampled samples (recL2ModifiedUpsampledSamples) are combined in a residual reconstruction process for the enhancement sub-layer 2. The output of this block is a set of reconstructed picture samples at level 2 (recL2PictureSamples). At block 2760, these reconstructed picture samples at level 2 may be subject to a dithering process that applies a dither filter. The output to this process is a set of reconstructed dithered picture samples at level 2 (recL2DitheredPictureSamples). These may be viewed at block 2762 as an output video sequence (e.g. for multiple consecutive pictures making up the frames of a video, where planes may be combined into a multi-dimensional array for viewing on display devices).

Payload Data Block Unit Process

The operations performed at block 2704 will now be described in more detail. The input to this process is the enhancement layer bitstream. The enhancement layer bitstream is encapsulated in NAL units, e.g. as indicated above. A NAL unit may be used to synchronize the enhancement layer information with the base layer decoded information.

The bitstream is organized in NAL units, with each NAL unit including one or more data blocks. For each data block, the *process_block()* syntax structure (as shown in the "Syntax" section above) is used to parse a block header (in certain cases, only the block header). It may invoke a relevant *process_block_()* syntax element based upon the information in the block header. A NAL unit which includes encoded data may comprise at least two data blocks: a picture configuration data block and an encoded (tiled) data block. A set of possible different data blocks are indicated in the table above that shows possible payload types.

A sequence configuration data block may occur at least once at the beginning of the bitstream. A global configuration data block may occur at least for every instantaneous decoding refresh picture. An encoded (tiled) data block may be preceded by a picture configuration data block. When present in a NAL unit, a global configuration data block may be the first data block in the NAL unit.

Picture Enhancement Decoding Process

The present section describes in more detail the picture enhancement decoding process performed at block 2706.

The input of this process may be the portion of the bitstream following the headers decoding process described in the "Process Block Syntax" section set out above. Outputs are the entropy encoded transform coefficients belonging to the picture enhancement being decoded. An encoded picture maybe preceded by the picture configuration payload described in the "Process Payload – Picture Configuration" and "Data Block Unit Picture Configuration Semantics" sections above.

The picture enhancement encoded data may be received as *payload_encoded_data* with the syntax for the processing of this data being described in the "Process Payload – Encoded Data" section. Inputs for the processing of the picture enhancement encoded data may comprise: a variable *nPlanes* containing the number of plane (which may depend on the value of the variable *processed_planes_type_flag*), a variable *nLayers* (which may depend on the value of *transform_type*), and a variable *nLevels* (which indicates the number of levels to be processed). These are shown in Figure 9A. The variable nLevels may be a constant, e.g. equal to 2, if two enhancement sub-layers are used and processed.

The output of block 2706 process may comprise a set of (*nPlanes*)x(*nLevels*)x(*nLayers*) surfaces (e.g. arranged as an array – preferably multi-dimensional) with elements *surfaces[nPlanes][nLevels][nLayers]*. If the *temporal_signalling_present_flag* is equal to 1, an additional temporal surface of a size *nPlanes* with elements *temporal_surface[nPlanes]* may also be retrieved. The variable *nPlanes* may be derived using the following processing:

if (processed_planes_type_flag == 0)
nPlanes = 1
else
nPlanes = 3

and the variable nLayers may be derived using the following processing:

if (transform_type == 0)
nLayers = 4
else
nLayers = 16

The encoded data may be organized in chunks as shown in Figure 9A (amongst others). The total number of chunks *total_chunk_count* may be computed as: *nPlanes* * *nLevels* * *nLayers* * (*no_enhancement_bit_flag* == 0) + *nPlanes* * (*temporal_signalling_present_flag* == 1). For each plane, a number (e.g. up to 2) enhancement sub-layers are extracted. For each enhancement sub-layer, a number (e.g. up to 16 for a 4x4 transform) coefficient groups of transform coefficients can be extracted. Additionally, if *temporal_signalling_present_flag* is equal to 1, an additional chunk with temporal data for enhancement sub-layer 2 may be extracted. Within this processing, a value of the variable *levelIdx* equal to 1 may be used to refer to enhancement sub-layer 1 and a value of the variable *levelIdx* equal to 2 may be used to refer to enhancement sub-layer 2. During the decoding process chunks may be read 2 bits at a time. Values for *surfaces[planeIdx][levelIdx][layerIdx].entropy_enabled_flag*, *surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag*, *temporal_surfaces[planeIdx].entropy_enabled_flag* and *temporal_surfaces[planeIdx].rle_only_flag* may be derived as follows:

```
shift_size = −1
for (planeIdx = 0; planeIdx < nPlanes; ++planeIdx) {
    if (no_enhancement_bit_flag == 0) {
        for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
            for (layerIdx = 0; layer < nLayers; ++layerIdx) {
                if (shift_size < 0) {
                    data = read_byte(bitstream)
                    shift_size = 8 – 1
                }
                surfaces [planeIdx][levelIdx][layerIdx].entropy_enabled_flag = ((data >> shift_size) & 0x1)
                surfaces [planeIdx][levelIdx][layerIdx].rle_only_flag = ((data >> (shift_size − 1)) & 0x1)
                shift_size −= 2
            }
        }
    } else {
        for (layerIdx = 0; layer < nLayers; ++layerIdx)
            surfaces [planeIdx][levelIdx][layerIdx].entropy_enabled_flag = 0
    }
    if (temporal_signalling_present_flag == 1) {
        if (shift_size < 0) {
            data = read_byte(bitstream)
            shift_size = 8 – 1
        }
        temporal_surfaces[planeIdx].entropy_enabled_flag = ((data >> shift_size) & 0x1)
        temporal_surfaces[planeIdx].rle_only_flag = ((data >> (shift_size − 1)) & 0x1)
        shift_size −= 2
    }
}
```

Data associated with the entropy-encoded transform coefficients and the entropy-encoded temporal signal coefficient group may be derived according to respective values of the *entropy_enabled_flag* and *rle_only_flag* fields. Here entropy encoding may comprise run-length encoding only or Prefix/Huffman Coding and run-length encoding. The content for the *surfaces[planeIdx][levelIdx][layerIdx].data* provides a starting address for the entropy encoded transform coefficients related to the specific chunk of data and *temporal_surfaces[planeIdx].data* provides the starting address for the entropy-encoded temporal signal coefficient group related to the specific chunk of data. These portions of data may be derived as set out below:

```
for (planeIdx = 0; planeIdx < nPlanes; ++planeIdx) {
    for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
        for (layerIdx = 0; layer < nLayers; ++layerIdx) {
            if (surfaces [planeIdx][levelIdx][layerIdx].entropy_enabled_flag) {
                if (surfaces [planeIdx][levelIdx][layerIdx].rle_only_flag) {
                    multibyte = read_multibyte(bitstream)
                    surfaces[planeIdx][levelIdx][layerIdx].size = multibyte
                    surfaces[planeIdx][levelIdx][layerIdx].data = bytestream_current(bitstream)
                } else {
                    data = read_byte(bitstream)
                    surfaces[planeIdx][levelIdx][layerIdx].prefix_last_symbol_bit_offset = (data&0x1F)
                    multibyte = read_multibyte(bitstream)
                    surfaces[planeIdx][levelIdx][layerIdx].size = multibyte
                    surfaces[planeIdx][levelIdx][layerIdx].data = bytestream_current(bitstream)
                    bytestream_seek(bitstream, surfaces[planeIdx][levelIdx][layerIdx].size)
                }
            }
        }
    }
    if (temporal_signalling_present_flag == 1) {
        if (temporal_surfaces[planeIdx].entropy_enabled_flag) {
            if (temporal_surfaces[planeIdx].rle_only_flag) {
                multibyte = read_multibyte(bitstream)
                temporal_surfaces[planeIdx].size = multibyte
                temporal_surfaces[planeIdx].data = bytestream_current(bitstream)
            } else {
                data = read_byte(bitstream)
                temporal_surfaces[planeIdx].prefix_last_symbol_bit_offset = (data&0x1F)
                multibyte = read_multibyte(bitstream)
                temporal_surfaces[planeIdx].size = multibyte
                temporal_surfaces[planeIdx].data = bytestream_current(bitstream)
                bytestream_seek(bitstream, temporal_surfaces[planeIdx].size)
            }
        }
    }
}
```

The transform coefficients contained in the block of bytes of length *surfaces[planeIdx][levelIdx][layerIdx].size* and starting from *surfaces[planeIdx][levelIdx][layerIdx].data* address may then be extracted and passed to an entropy decoding process, which may apply the methods described above with respect to Figures 10A to 10I, and/or the methods described in more detail in the description below.

If *temporal_signalling_present_flag* is set to 1, the temporal signal coefficient group contained in the block of bytes of length *temporal_surfaces[planeIdx].size* and starting from *temporal_surfaces[planeIdx].data* address may also be passed to similar entropy decoding process, .

Picture Enhancement Decoding Process – Tiled Data

The decoding process for picture enhancement encoded tiled data (*payload_encoded_tiled_data*) may be seen as a variation of the process described above. Syntax for this process is described in the above section entitled “Process Payload – Encoded Tiled Data”.

Inputs to this process may be: variables *nPlanes*, *nLayers* and *nLevels* as above; a variable *nTilesL2*, which equals to Ceil(*Picture_Width* / *Tile_Width*)xCeil(*Picture_Height*

/ *Tile_Height*) and refers to the number of tiles in the level 2 sub-layer; a variable *nTilesL1*, which refers to the number of tiles in level 1 sub-layer and equals: (a) *nTilesL2* if the variable *scaling_mode_level2* is equal to 0, (b) Ceil(Ceil(*Picture_Width* / 2) / *Tile_Width*)xCeil(Ceil(*Picture_Height*) / *Tile_Height*) if the variable *scaling_mode_level2* is equal to 1, and (c) Ceil(Ceil(*Picture_Width* / 2) / *Tile_Width*)xCeil(Ceil(*Picture_Height* / 2) / *Tile_Height*) if the variable *scaling_mode_level2* is equal to 2; *Picture_Width* and *Picture_Height*, which refer to the picture width and height as derived from the value of the variable *resolution_type*; and *Tile_Width* and *Tile_Height*, which refer to the tile width and height as derived from the value of the variable *tile_dimensions_type*. Further details of the variables referred to here is set out in the Data Block Semantics sections above.

An output of this process is the (*nPlanes*)x(*nLevels*)x(*nLayer*) array "surfaces", with elements *surfaces[nPlanes][nLevels][nLayer]*. If *temporal_signalling_present_flag* is set to 1, the output may also comprise an additional temporal surface of a size *nPlanes* with elements *temporal_surface[nPlanes]*. Values for the variables *nPlanes* and *nLayers* may be derived as set out in the above section.

As above, the encoded data is organized in chunks. In this case, each chunk may correspond to a tile, e.g. each of the portions 2140 shown in Figure 21A. The total number of chunks, *total_chunk_count*, is calculated as: *nPlanes* * *nLevels* * *nLayers* * (*nTilesL1* + *nTilesL2*) * (*no_enhancement_bit_flag* == 0) + *nPlanes* * *nTilesL2* * (*temporal_signalling_present_flag* == 1). The enhancement picture data chunks may be hierarchically organized as shown in one of Figures 21A and 21B. In accordance with the examples described herein, for each plane, up to 2 layers of enhancement sub-layers may be extracted and for each sub-layer of enhancement, up to 16 coefficient groups of transform coefficients may be extracted. Other implementations with different numbers of sub-layers or different transforms may have different numbers of extracted levels and layers. As before, as the present example applies temporal prediction in level 2, if *temporal_signalling_present_flag* is set to 1, an additional chunk with temporal data for enhancement sub-layer 2 is extracted. The variable *levelIdx* may be used as set out above.

In this tiled case, each chunk may be read 1 bit at a time. The *surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag* and, if *temporal_signalling_present_flag* is set to 1, *temporal_surfaces[planeIdx].rle_only_flag* may be derived as follows:

```
shift_size = −1
```

```
for (planeIdx = 0; planeIdx < nPlanes; ++ planeIdx) {
    if (no_enhancement_bit_flag == 0) {
        for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
            for (layerIdx = 0; layer < nLayers; ++layerIdx) {
                if (shift_size < 0) {
                    data = read_byte(bitstream)
                    shift_size = 8 – 1
                }
                surfaces [planeIdx][levelIdx][layerIdx].rle_only_flag = ((data >> (shift_size − 1)) & 0x1)
                shift_size −= 1
            }
        }
    }
    if (temporal_signalling_present_flag == 1) {
        if (shift_size < 0) {
            data = read_byte(bitstream)
            shift_size = 8 – 1
        }
        temporal_surfaces[planeIdx].rle_only_flag = ((data >> (shift_size − 1)) & 0x1)
        shift_size −= 1
    }
}
```

The *surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].entropy_enabled_flag* and, if *temporal_signalling_present_flag* is set to 1, *temporal_surfaces[planeIdx].tiles[tileIdx].entropy_enabled_flag* may be derived as follows:

```
if (compression_type_entropy_enabled_per_tile_flag == 0) {
    shift_size = −1
    for (planeIdx = 0; planeIdx < nPlanes; ++planeIdx) {
        if (no_enhancement_bit_flag == 0) {
            for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
```

```
        if (levelIdx == 1)
            nTiles = nTilesL1
        else
            nTiles = nTilesL2
        for (layerIdx = 0; layer < nLayers; ++layerIdx) {
            for (tileIdx=0; tileIdx < nTiles; tileIdx ++) {
                if (shift_size < 0) {
                    data = read_byte(bitstream)
                    shift_size = 8 – 1
                }
                surfaces[planeIdx][levelIdx][layerIdx]
                    .tiles[tileIdx].entropy_enabled_flag =
                        ((data >> (shift_size − 1)) & 0x1)
                shift_size −= 1
            }
        }
    }
} else {
    for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
        if (levelIdx == 1)
            nTiles = nTilesL1
        else
            nTiles = nTilesL2
        for (layerIdx = 0; layer < nLayers; ++layerIdx) {
            for (tileIdx = 0; tileIdx < nTiles; tileIdx++)
                surfaces[planeIdx][levelIdx][layerIdx]
                    .tiles[tileIdx].entropy_enabled_flag = 0
        }
    }
}
if (temporal_signalling_present_flag == 1) {
    for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++) {
        if (shift_size < 0) {
                data = read_byte(bitstream)
                shift_size = 8 – 1
            }
            temporal_surfaces[planeIdx].tiles[tileIdx].entropy_enabled_flag =
                    ((data >> (shift_size − 1)) & 0x1)
            shift_size = 1
        }
    }
  }
} else {
  RLE decoding process as defined herein.
}
```

According to the value of the *entropy_enabled_flag* and rle_only_flag fields, the content for the *surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].data* (i.e. indicating the beginning of the RLE only or Prefix Coding and RLE encoded coefficients related to the specific chunk of data) and, if *temporal_signalling_present_flag* is set to 1, according to the value of the *entropy_enabled_flag* and *rle_only_flag* fields, the content for the *temporal_surfaces[planeIdx].tiles[tileIdx].data* indicating the beginning of the RLE only or Prefix Coding and RLE encoded temporal signal coefficient group related to the specific chunk of data may be derived as follows:

```
if (compression_type_size_per_tile == 0) {
  for (planeIdx = 0; planeIdx < nPlanes; ++planeIdx) {
    for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
      if (levelIdx == 1)
        nTiles = nTilesL1
      else
        nTiles = nTilesL2
      for (layerIdx = 0; layer < nLayers; ++layerIdx) {
        for (tileIdx = 0; tileIdx < nTiles; tileIdx++) {
          if (surfaces
[planeIdx][levelIdx][layerIdx].tiles[tileIdx].entropy_enabled_flag) {
            if (surfaces [planeIdx][levelIdx][layerIdx].rle_only_flag) {
```

```
                        multibyte = read_multibyte(bitstream)
                        surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size = multibyte
                        surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx]data =
                                bytestream_current(bitstream)
                    } else {
                        data = read_byte(bitstream)
                        surfaces[planeIdx][levelIdx][layerIdx].
                                tiles[tileIdx].prefix_last_symbol_bit_offset = (data&0x1F)
                        multibyte = read_multibyte(bitstream)
                        surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size = multibyte
                        surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].data =
                                bytestream_current(bitstream)
                        bytestream_seek(bitstream, surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size)
                    }
                }
            }
        }
    }
    if (temporal_signalling_present_flag == 1) {
        for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++) {
            if (temporal_surfaces[planeIdx].tiles[tileIdx].entropy_enabled_flag) {
                if (temporal_surfaces[planeIdx].rle_only_flag) {
                    multibyte = read_multibyte(bitstream)
                    temporal_surfaces[planeIdx].tiles[tileIdx].size = multibyte
                    temporal_surfaces[planeIdx].tiles[tileIdx].data = bytestream_current(bitstream)
                } else {
                    data = read_byte(bitstream)
```

```
                        temporal_surfaces[planeIdx].tiles[tileIdx].
                            prefix_last_symbol_bit_offset = (data&0x1F)
                        multibyte = read_multibyte(bitstream)
                        temporal_surfaces[planeIdx].tiles[tileIdx].size = multibyte
                        temporal_surfaces[planeIdx].tiles[tileIdx].data                    =
bytestream_current(bitstream)
                        bytestream_seek(bitstream,
temporal_surfaces[planeIdx].tiles[tileIdx].size)
                    }
                }
            }
        }
    }
} else {
    for (planeIdx = 0; planeIdx < nPlanes; ++planeIdx) {
        for (levelIdx = 1; levelIdx <= nLevels; ++levelIdx) {
            if (levelIdx == 1)
                nTiles = nTilesL1
            else
                nTiles = nTilesL2
            for (layerIdx = 0; layer < nLayers; ++layerIdx) {
                for (tileIdx = 0; tileIdx < nTiles; tileIdx ++) {
                    if                                                   (surfaces
[planeIdx][levelIdx][layerIdx].tiles[tileIdx].entropy_enabled_flag) {
                        if (surfaces[planeIdx][levelIdx][layerIdx].rle_only_flag) {
                            Prefix Coding decoding process as defined in other sections to
fill surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size
                            surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].data =
                                bytestream_current(bitstream)
                        } else {
```

```
                        Prefix Coding decoding process as defined in other sections to
fill surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].
prefix_last_symbol_bit_offset
                        Prefix Coding decoding process as defined in other sections to
fill surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size
                        surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].data =
                                bytestream_current(bitstream)
                        bytestream_seek(
                            bitstream,surfaces[planeIdx][levelIdx][layerIdx].
                                tiles[tileIdx].size)
                    }
                }
            }
        }
    }
    if (temporal_signalling_present_flag == 1) {
        for (tileIdx = 0; tileIdx < nTilesL2; tileIdx++) {
            if (temporal_surfaces [planeIdx].tiles[tileIdx].entropy_enabled_flag) {
                if (temporal_surfaces [planeIdx].rle_only_flag) {
                    Prefix Coding decoding process as defined in other sections to fill
                        temporal_surfaces[planeIdx].tiles[tileIdx].size
                    temporal_surfaces[planeIdx].tiles[tileIdx].data =
bytestream_current(bitstream)
                } else {
                    Prefix Coding decoding process as defined in other sections to fill
temporal_surfaces[planeIdx].tiles[tileIdx].prefix_last_symbol_bit_offset
                    Prefix Coding decoding process as defined in other sections to fill
temporal_surfaces[planeIdx].tiles[tileIdx].size
                    temporal_surfaces[planeIdx].tiles[tileIdx].data =
bytestream_current(bitstream)
                    bytestream_seek(bitstream,
temporal_surfaces[planeIdx].tiles[tileIdx].size)
```

}
}
}
}
}
}

The coefficients contained in the block of bytes of length *surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].size* and starting from *surfaces[planeIdx][levelIdx][layerIdx].tiles[tileIdx].data* address may then be passed to for entropy decoding process as described elsewhere. If *temporal_signalling_enabled_flag* is set to 1, the temporal signal coefficient group contained in the block of bytes of length *temporal_surfaces[planeIdx].tiles[tileIdx].size* and starting from *temporal_surfaces[planeIdx].tiles[tileIdx].data* address are also passed for entropy decoding.

General Upscaling Process Description

Upscaling processes may be applied, at the decoder, to the decoded base picture at block 2712 in Figure 27 and to the combined intermediate picture at block 2732. In the present examples, upscaling may be configured based on a signalled scaling mode. In the processes described below the upscaling is configured based on the indications of *scaling_mode_level1* for the up-scaling to level 1 and based on *scaling_mode_level2* for the up-scaling to level 2.

Upscaling from Decoded Base Picture to Preliminary Intermediate Picture

The up-scaling from a decoded base picture to a preliminary intermediate picture, e.g. as performed in block 2712, may take the following inputs: a location (*xCurr*, *yCurr*) specifying the top-left sample of the current block relative to the top-left sample of the current picture component; a variable *IdxPlanes* specifying the colour component of the current block; a variable *nCurrS* specifying the size of the residual blocks used in the general decoding process; an (*nCurrS*)x(*nCurrS*) array *recDecodedBaseSamples* specifying decoded base samples for the current block; variables *srcWidth* and *srcHeight* specifying the width and the height of the decoded base picture; variables *dstWidth* and *dstHeight* specifying the width and the height of the resulting upscaled picture; and a variable *is8Bit* used to select the kernel coefficients for the scaling to be applied, e.g. if the samples are 8-bit, then variable *is8Bit* shall be equal to 0, if the samples are 16-bit, then variable *is8Bit* shall be equal to 1. An output of block 2712 may comprise a (*nCurrX*)x(*nCurrY*) array *recL1ModifiedUpsampledBaseSamples* of picture elements.

In the array of elements *recL1ModifiedUpsampledBaseSamples*[x][y] the variables *nCurrX* and *nCurrY* may be derived based on the scaling mode. For example, if *scaling_mode_level1* is equal to 0, no upscaling is performed, and *recL1ModifiedUpsampledBaseSamples*[x][y] are set to be equal to *recDecodedBaseSamples*[x][y]. If *scaling_mode_level1* is equal to 1, then *nCurrX* = *nCurrS* << 1, and *nCurrY* = *nCurrS*. If *scaling_mode_level1* is equal to 2, then *nCurrX* = *nCurrS* << 1, and *nCurrY* = *nCurrS* << 1.

The up-scaling applied at block 2712 may involve the use of a switchable up-scaling filter. The decoded base samples may be processed by an upscaling filter of a type signalled in the bitstream. The type of up-scaler maybe derived from the process described in the section “Data Block Unit Global Configuration Semantics”. Depending on the value of the variable *upsample_type*, a number of different kernel types may be applied. For example, each kernel types may be configured to receive a set of picture samples *recDecodedBaseSamples* as input and to produce a set of up-sampled picture samples *recL1UpsampledBaseSamples* as output. There may be four possible up-scaler kernels (although these may vary in number and type depending on implementation). These are also described in the section titled “Example Up-sampling Approaches”. In the present example, if *upsample_type* is equal to 0, the Nearest sample up-scaler described in the “Nearest up-sampling” section above may be selected. If *upsample_type* is equal to 1, the Bilinear up-scaler described in the “Bilinear up-sampling” section above may be selected. If *upsample_type* is equal to 2, a Bicubic up-scaler described in the “Cubic Up-sampling” section above may be selected. If *upsample_type* is equal to 3, a Modified Cubic up-scaler described in the “Cubic Up-sampling” section above may be selected.

A predicted residuals (e.g. predicted average) decoding computation may also be applied in certain cases as described below with respect to the level 1 to level 2 up-scaling.

A general up-scaler may divide the picture to upscale in 2 areas: center area and border areas as shown in Figures 9B and 9C. For the Bilinear and Bicubic kernel, the border area consists of four segments: Top, Left, Right and Bottom segments as shown in Figure

9B, while for the Nearest kernel consists of 2 segments: Right and Bottom as shown in Figure 9C. These segments are defined by the border-size parameter which may be set to 2 samples (1 sample for nearest method).

Level 1 Bit Depth Conversion

In certain examples, an up-scaling process may also involve a bit depth conversion, e.g. different levels (including levels 0, 1 and 2 described herein) may process data having different bit depths. The bit depths of each level may be configurable, e.g. based on configuration data that may be signalled from the encoder to the decoder. For example, the bit depths for each level, and any required conversion, may depending on the values of the bitstream fields in the global configuration as processed in the examples above. In one case, bit depth conversion is performed as part of the up-scaling process. Bit depth conversion may be applied using bit shifts and the difference between the bit depths of the lower and upper levels in the up-scaling.

When applying block 2712, the sample bit depth for level 1 may be based on *level1_depth_flag*. If *level1_depth_flag* is equal to 0, the preliminary intermediate picture samples are processed at the same bit depth as they are represented for the decoded base picture. If *level1_depth_flag* is equal to 1, the preliminary intermediate picture samples may be converted depending on the value of a variable *base_depth* and a variable *enhancement_depth*. The variable *base_depth* indicates a bit depth for the base layer. In certain examples, if *base_depth* is assigned a value between 8 and 14, e.g. depending on the value of field *base_depth_type* as specified in the "Data Block Unit Global Configuration Semantics" section above, then *enhancement_depth* is assigned a value between 8 and 14, depending on the value of field *enhancement_depth_type* as specified in the aforementioned semantics section.

If *base_depth* is equal to *enhancement_depth*, no further processing is required.

If *enhancement_depth* is greater than *base_depth*, the array of level 1 up-sampled base samples *recL1ModifiedUpsampledBaseSamples* may be modified as follows:

recL1ModifiedUpsampledBaseSamples[x][y] =
recL1ModifiedUpsampledBaseSamples[x][y] << (*enhancement_depth* - *base_depth*)

If *base_depth* is greater than *enhancement_depth*, the array *recL1ModifiedUpsampledBaseSamples* may be modified as follows:

recL1ModifiedUpsampledBaseSamples[x][y] =

recL1ModifiedUpsampledBaseSamples[x][y] >> (*base_depth* - *enhancement_depth*)

Upscaling from Combined Intermediate Picture to Preliminary Output Picture

A similar set of processes may be performed at block 2732. Inputs to this process may comprise: a location (*xCurr*, *yCurr*) specifying the top-left sample of the current block relative to the top-left sample of the current picture component; a variable *IdxPlanes* specifying the colour component of the current block; a variable *nCurrS* specifying the size of the residual block; an (*nCurrS*)x(*nCurrS*) array *recL1PictureSamples* specifying the combined intermediate picture samples of the current block; variables *srcWidth* and *srcHeight* specifying the width and the height of the reconstructed base picture; variables *dstWidth* and *dstHeight* specifying the with and the height of the resulting upscaled picture; and a variable *is8Bit* used to select the kernel coefficients for the scaling to be applied. If the samples are 8-bit, then variable *is8Bit* may be equal to 0, if the samples are 16-bit, then variable *is8Bit* may be equal to 1. An output of process 2732 is the (*nCurrX*)x(nCurrY) array *recL2ModifiedUpsampledSamples* of preliminary output picture samples with elements *recL2ModifiedUpsampledSamples*[x][y].

The variables *nCurrX* and *nCurrY* may be derived based on a scaling mode in a similar manner to the process described above. If *scaling_mode_level2* is equal to 0, no upscaling is performed, and *recL2ModifiedUpsampledSamples*[x][y] are set to be equal to *recL1PictureSamples*[x][y]. If *scaling_mode_level2* is equal to 1, then *nCurrX* = *nCurrS* << 1, and *nCurrY* = *nCurrS*. If *scaling_mode_level2* is equal to 2, then *nCurrX* = *nCurrS* << 1, and *nCurrY* = *nCurrS* << 1.

As described in the section above, the up-scaling performed at block 2732 may also involve the selective application of an upscaling filter, where an up-scaling type is signalled in the bitstream. Depending on the value of the variable *upsample_type*, each kernel type may be configured to *recL1PictureSamples* as input and producing *recL2UpsampledSamples* as output. There may be four possible up-scaler kernels (although these may vary in number and type depending on implementation). These are also described in the section titled “Example Up-sampling Approaches”. In the present example, if *upsample_type* is equal to 0, the Nearest sample up-scaler described in the “Nearest up-sampling” section above may be selected. If *upsample_type* is equal to 1, the Bilinear up-scaler described in the “Bilinear up-sampling” section above may be selected. If *upsample_type* is equal to 2, a Bicubic up-scaler described in the “Cubic Up-sampling” section above may be selected. If *upsample_type* is equal to 3, a Modified Cubic up-scaler described in the "Cubic Up-sampling" section above may be selected.

The division of the picture into multiple areas may be performed as described in the section above with reference to Figures 9B and 9C.

Following the upscaling, if *predicted_residual_mode_flag* as described above is equal to 1 process, a predicted residual (i.e. modified up-sampling) mode as described above and below (see sub-block 2744 in Figure 27) may be invoked with inputs as the (*nCurrX*)x(*nCurrY*) array *recL2UpsampledSamples* and the (*nCurrS*)x(*nCurrS*) array *recL1PictureSamples* specifying the combined intermediate picture samples of the current block. The output of this process may comprise a (*nCurrX*)x(*nCurrY*) array r*ecL2ModifiedUpsampledSamples* of picture elements. Otherwise, if *predicted_residual_mode_flag* is equal to 0, a predicted residual mode is not applied, and *recL2ModifiedUpsampledSamples*[x][y] are set to be equal to *recL2UpsampledSamples* [x][y] (i.e. set as the up-sampled values without modification).

Level 2 Bit Depth Conversion

Bit depth conversion as described above for level 1 may also (or alternatively) be applied when up-scaling from level 1 to level 2. Again, bit depth conversion may be performed depending on the values of the bitstream fields in the global configuration.

With respect to level 2, the sample bit depth may be derived from the *level1_depth_flag*. If *level1_depth_flag* is equal to 1, the preliminary output picture samples are processed at the same bit depth as they are represented for the preliminary intermediate picture. If *level1_depth_flag* is equal to 0, the output intermediate picture samples are converted depending on the value of the variables *base_depth* and *enhancement_depth*. These may be derived as discussed in the level 1 bit depth conversion section above. Again, if *base_depth* is equal to *enhancement_depth*, no further processing is required. If *enhancement_depth* is greater than *base_depth*, the array *recL2ModifiedUpsampledSamples* is modified as follows:

recL2ModifiedUpsampledSamples[x][y] =
recL2ModifiedUpsampledSamples[x][y] << (*enhancement_depth* - *base_depth*)

If *base_depth* is greater than *enhancement_depth*, the array *recL2ModifiedUpsampledSamples* is modified as follows:

recL2ModifiedUpsampledSamples[x][y] =

recL2ModifiedUpsampledSamples[x][y] >> (*base_depth* - *enhancement_depth*)

Nearest Sample Upsampler Kernel Description

The sections below set out additional details on the example up-samplers described above.

A first up-sampler is a nearest sample up-sampler as shown in sub-block 2736 and discussed in the "Nearest Up-sampling" section above. The example of sub-block 2736 takes as inputs: variables *srcX* and *srcY* specifying the width and the height of the input array; variables *dstX* and *dstY* specifying the width and the height of the output array; and a (*srcX*)x(*srcY*) array *recInputSamples*[x][y] of input samples. Outputs to this process are a (*dstX*)x(*dstY*) array *recUpsampledSamples*[x][y] of output samples.

The Nearest kernel performs upscaling by copying the current source sample onto the destination 2x2 grid. This is shown in Figure 9D and described in the accompanying description above. The destination sample positions are calculated by doubling the index of the source sample on both axes and adding +1 to extend the range to cover 4 samples as shown in Figure 9D.

The nearest sample kernel up-scaler may be applied as specified by the following ordered steps whenever (*xCurr*, *yCurr*) block belongs to the picture or to the border area as specified in Figure 9D.

If *scaling_mode_levelX* is equal to 1, the computation may be as follows:

```
for (ySrc = 0; ySrc < nCurrS; ++ySrc)
    yDst = ySrc
    for (xSrc = 0; xSrc < nCurrS; ++xSrc)
        xDst = xSrc << 1
        recUpsampledSamples[xDst][yDst] = recInputSamples[xSrc][ySrc]
        recUpsampledSamples[xDst + 1][yDst] = recInputSamples[xSrc][ySrc]
```

If *scaling_mode_levelX* is equal to 2, the computation may be as follows:

```
for (ySrc = 0; ySrc < nCurrS; ++ySrc)
    yDst = ySrc << 1
    for (xSrc = 0; xSrc < nCurrS; ++xSrc)
        xDst = xSrc << 1
        recUpsampledSamples[xDst][yDst] = recInputSamples[xSrc][ySrc]
```

recUpsampledSamples[xDst][yDst + 1] = recInputSamples[xSrc][ySrc]
recUpsampledSamples[xDst + 1][yDst] = recInputSamples[xSrc][ySrc]
recUpsampledSamples[xDst + 1][yDst + 1] = recInputSamples[xSrc][ySrc]

Bilinear Upsampler Kernel Description

A bilinear upsampler kernel process is described in the section titles "Bilinear up-sampling above". Further examples are now described with reference to sub-block 2738 in Figure 27. The inputs and outputs to sub-block 2738 may be the same as for sub-block 2736. The Bilinear up-sampling kernel consists of three main steps. The first step involves constructing a 2x2 grid of source samples with the base sample positioned at the bottom right corner. The second step involves performing the bilinear interpolation. The third step involves writing the interpolation result to the destination samples. The bilinear method performs the up-sampling by considering the values of the nearest 3 samples to the base sample. The base sample is the source sample from which the address of the destination sample is derived. Figure 9E shows an example source grid used in the kernel.

The bilinear interpolation is a weighted summation of all the samples in the source grid. The weights employed are dependent on the destination sample being derived. The algorithm applies weights which are relative to the position of the source samples with respect to the position of the destination samples. If calculating the value for the top left destination sample, then the top left source sample will receive the largest weighting coefficient while the bottom right sample (diagonally opposite) will receive the smallest weighting coefficient, and the remaining two samples will receive an intermediate weighting coefficient. This is visualized in Figure 9F and described in detail above.

An example bilinear kernel up-scaler is illustrated in Figure 9G. It may be applied as specified by the following ordered steps below when (*xCurr*, *yCurr*) block does not belong to the border area as specified in Figures 9B and 9C.

If *scaling_mode_levelX* is equal to 1, the following up-scaling computation may be performed:

for (ySrc = 0; ySrc < nCurrS + 1; ++ySrc)
for (xSrc = 0; xSrc < nCurrS + 1; ++xSrc)
xDst = (xSrc << 1) − 1

```
        bilinear1D(recInputSamples[xSrc − 1][ySrc], recInputSamples[xSrc1][ySrc], recUpsampledSamples [xDst][ySrc], recUpsampledSamples [xDst + 1][ySrc])
```

If *scaling_mode_levelX* is equal to 2, the following up-scaling computation may be performed:

```
for (ySrc = 0; ySrc < nCurrS + 1; ++ySrc)
    yDst = (ySrc << 1) − 1
    for (xSrc = 0; xSrc < nCurrS + 1; ++xSrc)
        xDst = (xSrc << 1) − 1
        bilinear2D(recInputSamples[xSrc − 1][ySrc − 1], recInputSamples[xSrc][ySrc − 1], recInputSamples[xSrc − 1][ySrc], recInputSamples[xSrc][ySrc], recUpsampledSamples[xDst][ySrc], recUpsampledSamples [xDst + 1][ySrc], recUpsampledSamples[xDst][ySrc + 1], recUpsampledSamples[xDst + 1][ySrc + 1])
```

The bilinear kernel up-scaler is applied as specified by the following ordered steps below when (*xCurr*, *yCurr*) block belongs to the border area as specified in Figures 9B and 9C:

If *scaling_mode_levelX* is equal to 1:

```
for (ySrc = 0; ySrc < nCurrS + 1; ++ySrc)
    for (xSrc = 0; xSrc < nCurrS + 1; ++xSrc)
        xDst = (xSrc << 1) − 1
        xSrc0 = Max(xSrc − 1, 0);
        xSrc1 = Min(xSrc, srcWidth − 1)
        bilinear1D(recInputSamples[xSrc0][ySrc], recInputSamples[xSrc1][ySrc], dst00, dst10)
        if (xDst >= 0)
            recUpsampledSamples[xDst][ySrc] = dst00
        if (xDst < (dstWidth–1))
            recUpsampledSamples[xDst + 1][ySrc] = dst10
```

If *scaling_mode_levelX* is equal to 2:

```
for (ySrc = 0; ySrc < nCurrS + 1; ++ySrc)
```

yDst = (ySrc << 1) − 1
ySrc0 = Max(ySrc − 1, 0)));
ySrc1 = Min (ySrc, srcHeight − 1)))
for (xSrc = 0; xSrc < nCurrS + 1; ++xSrc)
xDst = (xSrc << 1) −1
xSrc0 = Max(xSrc − 1, 0);
xSrc1 = Min(xSrc, srcWidth − 1)
bilinear2D(recInputSamples[xSrc0][ySrc0], recInputSamples[xSrc1][xSrc0], recInputSamples[xSrc0][ySrc1], recInputSamples[xSrc1][ySrc1], dst00, dst10, dst01, dst11)

The function bilinear1D (in00, in10, out00, out10) as set out above may be applied as set out below:

in00x3 = in00 * 3
in10x3 = in10 * 3
out00 = ((in00x3 + in10 + 2) >> 2)
out10 = ((in00 + in10x3 + 2) >> 2)

The function bilinear2D (in00, in10, in01, in11, out00, out10, out01, out11) as set out above may be applied as set out below:

in00x3 = in00 * 3
in10x3 = in10 * 3
in01x3 = in01 * 3
in11x3 = in11 * 3
in00x9 = in00x3 * 3
in10x9 = in10x3 * 3
in01x9 = in01x3 * 3
in11x9 = in11x3 * 3
out00 = ((in00x9 + in10x3 + in01x3 + in11 + 8) >> 4))
out10 = ((in00x3 + in10x9 + in01+ in11x3 + 8) >> 4))
out01 = ((in00x3 + in10 + in01x9 + in11x3 + 8) >> 4))
out11 = ((in00 + in10x3 + in01x3 + in11x9 + 8) >> 4))

Cubic Upsampler Kernel Description

The cubic up-sampler kernel process that is shown in sub-block 2740 may be applied as set out in this section. The inputs and outputs are the same as those described in the sections above. Further reference is made to Figures 9H, 9I and 9J and the section titled "Cubic Up-sampling".

The cubic up-sampling kernel of sub-block 2740 may be divided into three main steps. The first step involves constructing a 4x4 grid of source samples with the base sample positioned at the local index (2, 2). The second step involves performing a bicubic interpolation. The third step involves writing the interpolation result to the destination samples.

The cubic up-sampling kernel may be performed by using a 4x4 source grid which is subsequently multiplied by a 4x4 kernel. During the generation of the source grid, any samples which fall outside the frame limits of the source frame are replaced with the value of the source samples at the boundary of the frame. This is visualized in Figures 9H and 9I.

The kernels used for the cubic up-sampling process typically have a 4x4 coefficient grid. However, the relative position of the destination sample with regards to the source sample will yield a different coefficient set, and since the up-sampling is a factor of two, there will be 4 sets of 4x4 kernels used in the up-sampling process. These sets are represented by a 4-dimensional grid of coefficients (2 x 2 x 4 x 4). The bicubic coefficients are calculated from a fixed set of parameters; a core parameter (or bicubic parameter) of and four spline creation parameters. These may have values of, for example, −0.6 and [1.25, 0.25, −0.75, −1.75] respectively. The implementation of the filter uses fixed point computations.

The cubic kernel up-scaler is shown in Figure 9J and described in more detail in the "Cubic up-sampling" section above. The up-scaler is applied on one direction (vertical and horizontal) at time and follows different steps if (xCurr, yCurr) block belongs to the border as specified in Figures 9B and 9C.

Given a set of example coefficients as follows:

```
kernel[y][x] =
  {
  { −1382, 14285,  3942,  −461 }
  {  −461,  3942, 14285, −1382 }
  { −1280, 14208,  3840,  −384 }
```

{ −384, 3840, 14208, −1280 }

}

where y = 0...1 are coefficients to be used with 10-bit samples and y = 2...3 to be used with 8-bits samples. The up-scaler may thus be applied according to the following pseudo-code.

```
kernelOffset is equal to 4.
kernelSize is equal to 4.
if (Horizontal) {
   for (y = 0; y < nCurrS; y++)
      for (xSrc = 0; xSrc < nCurrS + 1; xSrc++)
         ConvolveHorizontal(recInputSamples, recUpsampledSamples, xSrc, y, kernel[is8Bit * 2])
} else if (Vertical) {
   dstHeightM1 = dstHeight – 1
   for (ySrc = 0; ySrc < nCurrS + 1; ySrc++)
      yDst = (ySrc << 1) − 1
      if (border) {
         yDst0 = ((yDst > 0)&&(yDst < dstHeight)) ? yDst : −1
         yDst1 = ((yDst + 1) < dstHeightM1) ? yDst + 1 : −1
      } else {
         yDst0 = yDst
         yDst1 = (yDst + 1)
      }
      for (x = 0; x < nCurrS; x++)
         ConvolveVertical(recInputSamples, recUpsampledSamples, yDst0, yDst1, x, ySrc, kernel[is8Bit * 2])
}
```

The function *ConvolveHorizontal(input, output, x, y, kernel, border)* as referenced above may be applied as set out below:

```
xDst = (x << 1) − 1;
if (border)
   dstWidthM1 = dstWidth − 1
if (xDst >= 0 && xDst < dstWidth)
```

```
    output[xDst][y] = ConvolveHorizontal (kernel[0], input[x + kernelOffset][y] , 14);
if (xDst < dstWidthM1)
    output [xDst + 1][y] = ConvolveHorizontal(kernel[1], input[x + kernelOffset][y])
, 14)
else
    output [xDst][y] = ConvolveHorizontal (kernel[0], input[x + kernelOffset][y] , 14)
    output [xDst + 1][y] = ConvolveHorizontal (kernel[1], input[x + kernelOffset][y]
, 14)
```

The function *ConvolveVertical (input, output, yDst0, yDst1, x, ySrc, kernel)* as referenced above may be applied as set out below:

```
if (border)
    dstWidthM1 = dstWidth − 1
if (yDst0 >= 0)
    output[x][yDst0] = ConvolveHorizontal (kernel[0], input[x][y + kernelOffset] , 14)
if (yDst0 >= 0)
    output [x][yDst] = ConvolveHorizontal(kernel[1], input[x][y + kernelOffset]) , 14)
else
    output [x][yDst0] = ConvolveHorizontal (kernel[0], input[x + kernelOffset][y] , 14)
    output [x][yDst1] = ConvolveHorizontal (kernel[1], input[x + kernelOffset][y], 14)
```

The function *ConvolveHorizontal (kernel, input, shift)* as referenced above may be applied as set out below:

```
accumulator = 0
for (int32_t x = 0; x < kernelSize; x++)
accumulator += input[x] * kernel[x]
offset = 1 << (shift − 1)
output = ((accumulator + offset) >> shift)
```

Modified Cubic Upsampler Kernel Description

Lastly in this section, a short description of an example implementation of sub-block 2742 is presented. The inputs and outputs may be defined as for the other up-sampling processes above. The implementation of the modified cubic filter again uses fixed point computations. It may be seen as a variation of the cubic up-sampler kernel described above, but with the following kernel coefficients:

```
kernel[y][x] =
 {
 { −2360, 15855,  4165, −1276 }
 { −1276,  4165, 15855, −2360 }
 { −2360, 15855,  4165, −1276 }
 { −1276,  4165, 15855, −2360 }
 }
```

where y = 0...1 are coefficients to be used with 10-bit samples and y = 2...3 to be used with 8 bits samples, the *kernelOffset* is equal to 4, and the *kernelSize* is equal to 4.

It should be noted the kernels provided herein are for example only and other implementations may use different kernels.

Predicted Residual Process Description

The following section will briefly provide an example implementation for the predicted residual process shown in sub-block 2744 of Figure 27. It may also be applied as part of the up-scaling of block 2712 in other examples. Inputs to this process are shown as: variables *srcX* and *srcY* specifying the width and the height of the lower resolution array; variables *dstX* and *dstY* specifying the width and the height of the upsampled arrays; a (*srcX*)x(*srcY*) array *recLowerResSamples*[x][y] of samples that were provided as input to the upscaling process; and a (dstX)x(dstY) array *recUpsampledSamples* [x][y] of samples that were output of the up-scaling process. The outputs to this process are a (*dstX*)x(*dstY*) array *recUpsampledModifiedSamples*[x][y] of output samples.

In the present example, the predicted residual process modifies *recUpsampledSamples* using a 2x2 grid if *scaling_mode_levelX* is equal to 2 (i.e. is two-dimensional) and using a 2x1 grid if *scaling_mode_levelX* is equal to 1 (i.e. is one-dimensional). The predicted residual process is not applied if *scaling_mode_levelX* is equal to 0 (e.g. as no up-scaling is performed).

The predicted residual process may be applied as specified by the following ordered steps whenever (*xCurr*, *yCurr*) block belongs to the picture or to the border area as specified in Figures 9B and 9C:

If scaling_mode_levelX is equal to 1 (i.e. scaling is one-dimensional), the following computation may be performed:

```
for (ySrc = 0; ySrc < srcY; ySrc++)
    yDst = ySrc
    for (xSrc = 0; xSrc < srcX; xSrc++)
        xDst = xSrc << 1
        modifier = recLowerResSamples[xSrc][ySrc] – (recUpsampledSamples[xDst][yDst] + recUpsampledSamples[xDst + 1][yDst]) >> 1
        recModifiedUpsampledSamples[xDst][yDst] = recUpsampledSamples[xDst][yDst] + modifier
        recModifiedUpsampledSamples[xDst + 1][yDst] = recUpsampledSamples[xDst + 1][yDst] + modifier
```

If scaling_mode_levelX is equal to 2 (i.e. scaling is two-dimensional), the following computation may be performed:

```
for (ySrc = 0; ySrc < srcY; ySrc++)
    yDst = ySrc << 1
    for (xSrc = 0; xSrc < srcX; xSrc++)
        xDst = xSrc << 1
        modifier = recLowerResSamples[xSrc][ySrc] –
            (recUpsampledSamples[xDst][yDst] +
            recUpsampledSamples[xDst + 1][yDst] +
            recUpsampledSamples[xDst][yDst + 1] +
                recUpsampledSamples[xDst + 1][yDst + 1]) >> 2
        recModifiedUpsampledSamples [xDst][yDst] = recUpsampledSamples[xDst][yDst] + modifier
        recModifiedUpsampledSamples [xDst][yDst + 1] = recUpsampledSamples[xDst + 1][yDst] + modifier
        recModifiedUpsampledSamples [xDst + 1][yDst] = recUpsampledSamples[xDst][yDst + 1] + modifier
        recModifiedUpsampledSamples [xDst + 1][yDst + 1] = recUpsampledSamples[xDst + 1][yDst + 1] + modifier
```

Conventions Used in Certain Examples

Certain examples described herein use conventional notation for video coding technologies. For example, notation is used herein to refer to one or more of programming functions and mathematical operations. Certain mathematical operators used herein are presented in a manner that is similar to the conventions used in the C programming language. In certain examples, the results of integer division and arithmetic shift operations are defined as set out below, and additional operations are defined, such as exponentiation and real-valued division. Numbering and counting conventions generally begin from 0, e.g., "the first" is equivalent to the 0-th, "the second" is equivalent to the 1-th, etc.

In examples, arithmetic operators use conventional notation:

+	Addition
−	Subtraction (as a two-argument operator) or negation (as a unary prefix operator)
*	Multiplication, including matrix multiplication
x^y	Exponentiation. Specifies x to the power of y.
/	Integer division with truncation of the result toward zero. For example, 7 / 4 and −7 / −4 are truncated to 1 and −7 / 4 and 7 / −4 are truncated to −1.
÷	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\frac{x}{y}$	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\sum_{i=x}^{y} f(i$	The summation of f(i) with i taking all integer values from x up to and including y.
x % y	Modulus. Remainder of x divided by y, defined only for integers x and y with x >= 0 and y > 0.

Conventional logical operators are also used. The following logical operators are defined as follows:

x && y - Boolean logical "and" of x and y x || y - Boolean logical "or" of x and y

! - Boolean logical "not"

x ? y : z - If x is TRUE or not equal to 0, evaluates to the value of y; otherwise, evaluates to the value of z.

Relational operators also have conventional meanings, e.g.: > - Greater than; >= - Greater than or equal to; < - Less than; <= - Less than or equal to; = = - Equal to; != - Not equal to.

When a relational operator is applied to a syntax element or variable that has been assigned the value "na" (not applicable), the value "na" may be treated as a distinct value for the syntax element or variable. The value "na" is considered not to be equal to any other value.

The following bit-wise operators are also used in examples:

& Bit-wise "and". When operating on integer arguments, this operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.

| Bit-wise "or". When operating on integer arguments, this operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.

^Bit-wise "exclusive or". When operating on integer arguments, this operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0.

x >> y Arithmetic right shift of a two's complement integer representation of x by y binary digits. This function is defined only for non-negative integer values of y. Bits shifted into the most significant bits (MSBs) as a result of the right shift have a value equal to the MSB of x prior to the shift operation.

x << y Arithmetic left shift of a two's complement integer representation of x by y binary digits. This function is defined only for non-negative integer values of y. Bits shifted into the least significant bits (LSBs) as a result of the left shift have a value equal to 0.

The following arithmetic operators are also used: = - Assignment operator; + + - Increment, i.e., $x++$ is equivalent to $x = x + 1$ (when used in an array index, this may evaluate to the value of the variable prior to the increment operation); − − - Decrement, i.e., $x--$ is equivalent to $x = x - 1$ (when used in an array index, this may evaluate to the value of the variable prior to the decrement operation); += - Increment by amount specified, i.e., x += 3 is equivalent to x = x + 3, and x += (−3) is equivalent to x = x + (−3); −= - Decrement by amount specified, i.e., x −= 3 is equivalent to x = x − 3, and x −= (−3) is equivalent to x = x − (−3).

A range of values may be specified using the notation: x = y..z or x = y to z, where x takes on integer values starting from y to z, inclusive, with x, y, and z being integer numbers and z being greater than y.

The following mathematical functions are also used in certain example computations:

$$\text{Abs}(\ x\ ) = \begin{cases} x & ; \quad x >= 0 \\ -x & ; \quad x < 0 \end{cases}$$

Ceil(x) the smallest integer greater than or equal to x.

$$\text{Clip3}(\ x, y, z\ ) = \begin{cases} x & ; \quad z < x \\ y & ; \quad z > y \\ z & ; \quad \text{otherwise} \end{cases}$$

Floor(x) the largest integer less than or equal to x.

Ln(x) the natural logarithm of x (the base-e logarithm, where e is the natural logarithm base constant 2.718 281 828...).

Log10(x) the base-10 logarithm of x.

$$\text{Min}(\ x, y\ ) = \begin{cases} x & ; \quad x <= y \\ y & ; \quad x > y \end{cases}$$

$$\text{Max}(\ x, y\ ) = \begin{cases} x & ; \quad x >= y \\ y & ; \quad x < y \end{cases}$$

Round(x) = Sign(x) * Floor(Abs(x) + 0.5)

$$\text{Sign}(\ x\ ) = \begin{cases} 1 & ; \quad x > 0 \\ 0 & ; \quad x == 0 \\ -1 & ; \quad x < 0 \end{cases}$$

$\text{Sqrt}(\ x\ ) = >\sqrt{x}$

When an order of precedence in an expression is not indicated explicitly by use of parentheses, the following rules may apply: operations of a higher precedence are evaluated before any operation of a lower precedence; and operations of the same precedence are evaluated sequentially from left to right. The table below indicates a preferred precedence of certain example operations (e.g. from highest to lowest where a higher position in the table indicates a higher precedence – this may be the same as the order of precedence as used in the C programming language).

operations (with operands x, y, and z)
"x++", "x– –"
"!x", "–x" (as a unary prefix operator)
x^y
"x * y", "x / y", "x ÷ y", "$\frac{x}{y}$", "x % y"
"x + y", "x – y" (as a two-argument operator), "$\sum_{i=x}^{y} f(i)$"
"x << y", "x >> y"
"x < y", "x <= y", "x > y", "x >= y"
"x = = y", "x != y"
"x & y"
"x | y"
"x && y"
"x || y"
"x ? y : z"
"x..y"

In descriptions of bitstreams in examples herein, syntax elements in the bitstream may be represented in **bold** type. A syntax element may be described by its name (e.g. in all lower-case letters with underscore characters), and one descriptor for its method of coded representation. The decoding processes described herein may be configured to behave according to the value of the syntax element and to the values of previously decoded syntax elements. When a value of a syntax element is used in the syntax tables or the text, it may appear in regular (i.e., not bold) type.

In some cases, the syntax tables may use the values of other variables derived from syntax elements values. Such variables appear in syntax tables, or text, named by a mixture of lower case and uppercase letter and without any underscore characters. Variables starting with an upper-case letter are derived for the decoding of the current syntax structure and all depending syntax structures. Variables starting with an upper-case letter may be used in the decoding process for later syntax structures without mentioning the originating syntax structure of the variable. Variables starting with a lower-case letter may only used within the clause in which they are derived.

In some cases, “mnemonic” names for syntax element values or variable values are used interchangeably with their numerical values. Sometimes “mnemonic” names are used without any associated numerical values. The association of values and names is specified in the text. The names are constructed from one or more groups of letters separated by an underscore character. Each group starts with an upper-case letter and may contain more upper-case letters. It should be noted that names are provided as examples only and implementations may use different names.

Functions that specify properties of the current position in the bitstream may be referred to as syntax functions. These functions are specified in examples and may assume the existence of a bitstream pointer with an indication of the position of the next bit to be read by the decoding process from the bitstream. Syntax functions may be described by their names, which may be constructed as syntax element names and end with left and right round parentheses including zero or more variable names (for definition) or values (for usage), separated by commas (if more than one variable).

Functions that are not syntax functions (e.g. mathematical functions) may be described by their names, which start with an upper case letter, contain a mixture of lower and upper case letters without any underscore character, and end with left and right parentheses including zero or more variable names (for definition) or values (for usage) separated by commas (if more than one variable).

A one-dimensional array may be referred to as a list. A two-dimensional array may be referred to as a matrix. Arrays can either be syntax elements or variables. Subscripts or square parentheses are used in examples for the indexing of arrays. In reference to a visual depiction of a matrix, the first subscript is used as a row (vertical) index and the second subscript is used as a column (horizontal) index. The indexing order may be reversed when using square parentheses rather than subscripts for indexing. Thus, an element of a matrix S at horizontal position x and vertical position y may be denoted either as S[x][y] or as S_{yx}. A single column of a matrix may be referred to as a list and denoted by omission of the row index. Thus, the column of a matrix s at horizontal position x may be referred to as the list S[x].

A specification of values of the entries in rows and columns of an array may be denoted by { {...} {...} }, where each inner pair of brackets specifies the values of the elements within a row in increasing column order and the rows are ordered in increasing row order. Thus, setting a matrix S equal to { { 1 6 } { 4 9 }} specifies that $S[0][0]$ is set equal to 1, $S[1][0]$ is set equal to 6, $S[0][1]$ is set equal to 4, and $S[1][1]$ is set equal to 9.

Binary notation is indicated in examples by enclosing the string of bit values by single quote marks. For example, '01000001' represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1. Hexadecimal notation is indicated by prefixing the hexadecimal number by "0x", it may be used instead of binary notation when the number of bits is an integer multiple of 4. For example, 0x41 represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1. Numerical values not enclosed in single quotes and not prefixed by "0x" may be considered as decimal values. A value equal to 0 may represent a FALSE condition in a test statement. The value TRUE may be represented by any value different from zero.

In pseudocode examples presented herein, a statement of logical operations as would be described mathematically in the following form:

```
if( condition 0 )
   statement 0
else if( condition 1 )
   statement 1
...
else /* informative remark on remaining condition */
statement n
```

may be described in the following manner:

```
.. as follows / ... the following applies:
If condition 0, statement 0
Otherwise, if condition 1, statement 1
...
Otherwise (informative remark on remaining condition), statement n
```

Statements such "If ... Otherwise, if ... Otherwise, ..." in the text may be introduced with "... as follows" or "... the following applies" immediately followed by "If ... ". The last condition of the "If ... Otherwise, if ... Otherwise, ..." is always an "Otherwise, ...".

Interleaved "If ... Otherwise, if ... Otherwise, ..." statements can be identified by matching "... as follows" or "... the following applies" with the ending "Otherwise, ...".

In certain pseudo-code examples, a statement of logical operations as would be described mathematically in the following form:

```
if( condition 0a && condition 0b )
   statement 0
else if( condition 1a || condition 1b )
   statement 1
...
else
   statement n
```

may be described in the following manner:

```
... as follows / ... the following applies:
If all of the following conditions are true, statement 0:
   condition 0a
   condition 0b
Otherwise, if one or more of the following conditions are true, statement 1:
   condition 1a
   condition 1b
   ...
Otherwise, statement n
```

In certain pseudo-code examples, a statement of logical operations as would be described mathematically in the following form:

```
if( condition 0 )
   statement 0
if( condition 1 )
   statement 1
```

may be described in the following manner:

```
When condition 0, statement 0
When condition 1, statement 1
```

In examples, processes are used to describe the decoding of syntax elements. A process may have a separately described specification and invoking. Syntax elements and upper-case variables that pertain to a current syntax structure and depending syntax structures may be available in the process specification and invoking. A process specification may also have a lower-case variable explicitly specified as input. Each process specification may have an explicitly specified output. The output is a variable that may either be an upper-case variable or a lower-case variable. When invoking a process, the assignment of variables is specified as follows: if the variables at the invoking and the process specification do not have the same name, the variables are explicitly assigned to lower case input or output variables of the process specification; otherwise (the variables at the invoking and the process specification have the same name), assignment is implied. In the specification of a process, a specific coding block may be referred to by the variable name having a value equal to the address of the specific coding block.

At both the encoder and decoder, for example implemented in a streaming server or client device or client device decoding from a data store, methods, "components" and processes described herein can be embodied as code (e.g., software code) and/or data. The encoder and decoder may be implemented in hardware or software as is well-known in the art of data compression. For example, hardware acceleration using a specifically programmed Graphical Processing Unit (GPU) or a specifically designed Field Programmable Gate Array (FPGA) may provide certain efficiencies. For completeness, such code and data can be stored on one or more computer-readable media, which may include any device or medium that can store code and/or data for use by a computer system. When a computer system reads and executes the code and/or data stored on a computer-readable medium, the computer system performs the methods and processes embodied as data structures and code stored within the computer-readable storage medium. In certain embodiments, one or more of the steps of the methods and processes described herein can be performed by a processor (e.g., a processor of a computer system or data storage system).

Generally, any of the functionality described in this text or illustrated in the figures can be implemented using software, firmware (e.g., fixed logic circuitry), programmable or nonprogrammable hardware, or a combination of these implementations. The terms "component" or "function" as used herein generally represents software, firmware, hardware or a combination of these. For instance, in the case of a software implementation, the terms "component" or "function" may refer to program code that performs specified tasks when executed on a processing device or devices. The illustrated separation of components and functions into distinct units may reflect any actual or conceptual physical grouping and allocation of such software and/or hardware and tasks.

Claims

1. A bitstream for transmitting one or more enhancement residuals planes suitable to be added to a set of preliminary pictures obtained from a decoder reconstructed video, comprising:

a decoder configuration for controlling a decoding process of the bitstream; and,

encoded enhancement data comprising encoded residual data representing differences between a reference video frame and a decoded version of the video frame,

wherein the decoder configuration comprises an *upsample_type* parameter indicating a type of upscaling that should be used in the decoding process;

wherein each of one or more values of *upsample_type* indicates that the type of upscaling is cubic upscaling;

wherein a plurality of values of *upsample_type* each indicate that the type of upscaling is cubic upscaling according to a respective set of coefficients;

wherein a third value of *upsample_type* indicates that the cubic upscaling should be performed according to a first set of kernel coefficients $\mathbf{k_1}$, wherein:

$$\boldsymbol{k_1} = \begin{pmatrix} -1382 & 14285 & 3942 & -461 \\ -461 & 3942 & 14285 & -1382 \end{pmatrix}$$

and/or a fourth value of *upsample_type* indicates that the cubic upscaling should be performed according to a second set of kernel coefficients $\mathbf{k_2}$, wherein:

$$\boldsymbol{k_2} = \begin{pmatrix} -2360 & 15855 & 4165 & -1276 \\ -1276 & 4165 & 15855 & -2360 \end{pmatrix}$$

2. A bitstream according to claim 1, wherein a first value of *upsample_type* indicates that the type of upscaling is nearest sample upscaling.

3. A bitstream according to claim 1 or claim 2, wherein a second value of *upsample_type* indicates that the type of upscaling is linear upscaling.

4. A bitstream according to any preceding claim, wherein the *upsample_type* parameter is included in the bitstream for a plurality of pictures, preferably wherein the video includes instantaneous decoding refresh, IDR, pictures and non-IDR pictures, the

10 11 23

bitstream includes a global configuration at least once per IDR picture, and the *upsample_type* parameter is included in each global configuration.

5. A bitstream according to any preceding claim, wherein the decoder configuration further comprises one or more scaling mode parameters indicating whether or not the decoding process should include the upscaling.

6. A bitstream according to claim 5, wherein the one or more scaling mode parameters are included in the bitstream for a plurality of pictures, preferably wherein the video includes instantaneous decoding refresh, IDR, pictures and non-IDR pictures, the bitstream includes a global configuration at least once per IDR picture, and the one or more scaling mode parameters are included in each global configuration.

7. A bitstream according to claim 5 or 6, wherein each of the one or more scaling mode parameters indicates whether or not the decoding process should include the upscaling for a respective enhancement sub-layer.

8. A bitstream according to any of claims 5 to 7, wherein:

a first value of one of the scaling mode parameters indicates that the decoding process should include the upscaling in one dimension; and

a second value of one of the scaling mode parameters indicates that the decoding process should include the upscaling in two dimensions.

9. A bitstream according to any preceding claim, wherein the decoder configuration comprises:

an indication of a scaling factor to be applied to the decoded version of the video frame; and/or

a type of transform to be applied to coding units of the encoded residual data.

10. A method of decoding an encoded bitstream into one or more enhancement residuals planes to be added to a set of preliminary pictures that are obtained from a decoder reconstructed video, the method comprising:

retrieving a plurality of decoding parameters from a decoder configuration associated with the encoded bitstream, wherein the decoding parameters are used to configure the decoding operations;

retrieving encoded enhancement data from the encoded bitstream; and,

decoding the enhancement data to generate a set of residuals representing differences between a reference video frame and a decoded version of the video frame,

wherein the decoder configuration comprises an *upsample_type* parameter indicating a type of upscaling to be used in the decoding process;

wherein each of one or more values of *upsample_type* indicates that the type of upscaling is cubic upscaling;

wherein plurality of values of *upsample_type* each indicate that the type of upscaling is cubic upscaling according to a respective set of coefficients;

wherein a third value of *upsample_type* indicates that the cubic upscaling should be performed according to a first set of kernel coefficients $\mathbf{k_1}$, wherein:

$$\boldsymbol{k_1} = \begin{pmatrix} -1382 & 14285 & 3942 & -461 \\ -461 & 3942 & 14285 & -1382 \end{pmatrix}$$

and/or a fourth value of *upsample_type* indicates that the cubic upscaling should be performed according to a second set of kernel coefficients $\mathbf{k_2}$, wherein:

$$\boldsymbol{k_2} = \begin{pmatrix} -2360 & 15855 & 4165 & -1276 \\ -1276 & 4165 & 15855 & -2360 \end{pmatrix}$$

11. A method according to claim 10, wherein a first value of *upsample_type* indicates that the type of upscaling is nearest sample upscaling.

12. A method according to claim 10 or claim 11, wherein a second value of *upsample_type* indicates that the type of upscaling is linear upscaling.

13. A method according to any of claims 10 to 12, wherein the *upsample_type* parameter is included in the bitstream for a plurality of pictures, preferably wherein the

10 11 23

video includes instantaneous decoding refresh, IDR, pictures and non-IDR pictures, the bitstream includes a global configuration at least once per IDR picture, and the *upsample_type* parameter is included in each global configuration.

14. A method according to any of claims 10 to 13, wherein the decoder configuration further comprises one or more scaling mode parameters indicating whether or not the decoding process should include the upscaling.

15. A method according to claim 14, wherein the one or more scaling mode parameters are included in the bitstream for a plurality of pictures, preferably wherein the video includes instantaneous decoding refresh, IDR, pictures and non-IDR pictures, the bitstream includes a global configuration at least once per IDR picture, and the one or more scaling mode parameters are included in each global configuration.

16. A method according to claim 14 or 15, wherein each of the one or more scaling mode parameters indicates whether or not the decoding process should include the upscaling for a respective enhancement sub-layer.

17. A method according to any of claims 14 to 16, wherein:

a first value of one of the scaling mode parameters indicates that the decoding process should include the upscaling in one dimension; and

a second value of one of the scaling mode parameters indicates that the decoding process should include the upscaling in two dimensions.

18. A processing apparatus configured to decode a bitstream according to any of claims 1 to 9 and/or perform a method according to any of claims 10 to 17.

19. A computer readable storage medium storing instructions which, when executed by a processing apparatus, cause the processing apparatus to perform a method according to any of claims 10 to 17.

20. An encoder configured to generate the bitstream of any one of claims 1 to 9.

21. A method of generating the bitstream of any one of claims 1 to 9.