

Decoding of Scalable Video Streams

Technical Field

The present invention relates to the field of video decoding, and more specifically to methods and systems for decoding scalable video streams.

5

Background

One of the challenges of decoding a scalable video stream such as a scalable video stream in accordance with the MPEG-5 Part 2 LCEVC standard is that the ordering of the entropy encoded data can be presented in various formats, including independently decodable tiles and temporal blocks. This variability in data ordering poses significant challenges for decoders, particularly in terms of on-chip memory usage.

10

Therefore, there is a need to address at least one challenge posed by the variability in data ordering in a scalable video streams, and providing streamlined and efficient approach to handling scalable video streams.

Summary

15 According to a first aspect of the invention, there is provide a method for reordering entropy encoded data prior to a decoding operation. The method comprising: receiving the entropy encoded data in a bitstream; and reordering the entropy encoded data from a first order to a second order. In this way, the method addresses the challenge of variability in data ordering, which can be presented in various formats.

20 Optionally, the bitstream is an enhancement layer bitstream.

Optionally, the second order is a predetermined order.

Optionally, the method further comprises reading configuration information from the received video bitstream. The reordering of the entropy encoded data is based on the configuration information. In this way, the method allows for flexible reordering of the entropy encoded data based on the configuration information, ensuring that the data is processed in the most efficient and correct format for decoding.

25

Optionally, the configuration information indicates which encoding mode was enabled in the encoding process responsible for encoding the entropy encoded data from one or more of a tile mode and/or a temporal mode.

30 Optionally, the reordering of the entropy encoded data is based on the enabled encoding mode.

In this way, the method ensures that the entropy encoded data is reordered accurately based on the encoding mode, whether it is tile mode or temporal mode, leading to more efficient and precise decoding.

Optionally, the second order is a raster scan order.

Optionally, the reordering of the entropy encoded data functionally removes tiles and/or temporal blocks by ordering the entropy encoded data into a raster scan order.

- 5 Optionally, the second order matches an order of a base layer of the bitstream. In this way, the method ensures that the reordered entropy encoded data is in the same format as the base layer format.

Optionally, the reordering of the entropy encoded data comprises identifying distinct segments within the entropy encoded data.

Optionally, the distinct segments are adjacent segments in the entropy encoded data.

- 10 Optionally, identifying the distinct segments is based on analysing run length contexts within the entropy encoded data.

Optionally, the reordering of the entropy encoded data comprises combining the identified segments to form a data stream according to a raster scan order.

- 15 Optionally, the combining of identified segments to form a data stream according to a raster scan order comprises combining based on the run length context of the identified segments.

Optionally, the bitstream comprises multiple layers each comprising entropy encoded data.

Optionally, the method further comprises scheduling a group of modules to reorder the entropy encoded data, wherein at least one module of the group of modules reorders two or more layers of the multiple layers.

- 20 Optionally, the method comprises storing the output of the reordering in either an off-chip memory or an on-chip memory.

Optionally, the entropy encoded data is reordered before the decoding operation starts.

Optionally, the method further pauses the decoding operation if an error is determined in the reordering operation.

- 25 Optionally, the method comprises determining that the entropy encoded data is Huffman encoded and performing a Huffman decoding operation on the entropy encoded data before reordering the entropy encoded data.

Optionally, the method further comprises setting memory write offsets to the off-chip memory based on the configuration information.

- 30 Optionally, the raster scan order is a line-based raster scan order.

According to a second aspect of the invention, there is provided an apparatus configured to perform the method of any preceding statement.

According to a third aspect of the invention, there is provided a computer readable storage medium comprising instructions, when executed by a processor, causes the processor to perform the method of any preceding method statement.

Brief Description of the Drawings

- 5 The invention shall now be described, by way of example only, with reference to the accompanying drawings in which:

FIG. 1 shows a high-level schematic of a hierarchical encoding and decoding process;

FIG. 2 depicts a generalised encoding process;

FIG. 3 depicts a generalised decoding process;

- 10 FIG. 4 is an illustration of the order of the entropy encoded data in accordance with different modes of operation of the entropy encoder;

FIG. 5 illustrates a two-stage decoding architecture in accordance with an aspect of the invention;

FIG. 6 illustrates the operation of pre-decoder illustrated in FIG. 5 in more detail;

FIG. 7 is a block diagram illustrating scheduling of the prefix decoders;

- 15 FIG. 8A is a block diagram depicting storage of a DDS layer in DDR;

FIG. 8B is a block diagram depicting conceptually how entropy encoded data is split and stitched and buffered; and

FIG. 9 is a block depicting storage of context data in DDR after tile line pseudo-stitching.

20 Detailed Description

Introduction

- FIG. 1 shows a high-level schematic of a hierarchical encoding and decoding process. Data 101 to be encoded is retrieved by a hierarchical encoder 102 which outputs encoded data 103. Subsequently, the encoded data 103 is received by a hierarchical decoder 104 which decodes the data and outputs decoded data 105.
- 25

- Typically, the hierarchical coding schemes used in examples herein create a base or core level, which is a representation of the original data at a lower level of quality and one or more levels of residuals which can be used to recreate the original data at a higher level of quality using a decoded version of the base level data. 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
- 30

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 the generalised examples are agnostic as to 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.

In specific examples, the data 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.

The methods described herein may be applied to so-called planes of data that reflect different colour components of a video signal. For example, the methods may be applied to different planes of YUV or RGB data reflecting different colour channels. Different colour channels may be processed in parallel. The components of each stream may be collated in any logical order.

A further hierarchical coding technology with which the principles of the present invention may be utilised is illustrated in FIGS. 2 and 3. This technology is a flexible, adaptable, highly efficient and computationally inexpensive coding format which combines a different video coding format, a base codec, (e.g., AVC, HEVC, or any other present or future codec) with at least two enhancement levels of coded data.

The general structure of the encoding scheme 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. Thus, the streams are considered to be a base stream and two enhancement streams, which may be further multiplexed or otherwise combined to generate an encoded data stream. References to an encoded data as described herein may refer to each enhancement stream or a combination of the base stream and the enhancement stream. The base stream may be decoded by a hardware decoder while the enhancement streams may be suitable for software or hardware processing implementation with suitable power consumption. This general encoding structure creates a plurality of degrees of freedom that allow great flexibility and adaptability to many situations, thus making the coding format suitable for many use cases including OTT transmission, live streaming, live ultra-high-definition 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.

Returning to the initial process described above, where a base stream is provided along with two levels (or sub-levels) of enhancement within an enhancement stream or two enhancement streams, an example of a generalised encoding process is depicted in the block diagram of FIG. 2. An input

video 200 at an initial resolution is processed to generate various encoded streams 201, 202, 203 to form bitstream 230. A first encoded stream 201 (encoded base stream) is produced by feeding a base codec (e.g., AVC, HEVC, or any other codec) with a down-sampled version of the input video 200. The encoded base stream may be referred to as the base layer or base level. A second encoded stream 303 (encoded level 1 stream) is produced by processing the residuals obtained by taking the difference between a reconstructed base codec video and the down-sampled version of the input video. A third encoded stream 203 (encoded level 2 stream) 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. In certain cases, the components of FIG. 2 may provide a general low complexity encoder. In certain cases, the enhancement streams may be generated by encoding processes that form part of the low complexity encoder and the low complexity encoder may be configured to control an independent base encoder and decoder (e.g., as packaged as a base codec). In other cases, the base encoder and decoder may be supplied as part of the low complexity encoder. In one case, the low complexity encoder of FIG. 2 may be seen as a form of wrapper for the base codec, where the functionality of the base codec may be hidden from an entity implementing the low complexity encoder.

A down-sampling operation illustrated by down-sampling component 205 may be applied to the input video to produce a down-sampled video to be encoded by a base encoder 213 of a base codec. The down-sampling can be done either in both vertical and horizontal directions, or alternatively only in the horizontal direction. The base encoder 213 and a base decoder 214 may be implemented by a base codec (e.g., as different functions of a common codec). The base codec, and/or one or more of the base encoder 213 and the base decoder 214 may comprise suitably configured electronic circuitry (e.g., a hardware encoder/decoder) and/or computer program code that is executed by a processor.

Each enhancement stream encoding process may not necessarily include an upsampling step. In FIG. 2 for example, the first enhancement stream is conceptually a correction stream while the second enhancement stream is upsampled to provide a level of enhancement.

Looking at the process of generating the enhancement streams in more detail, to generate the encoded level 1 stream, the encoded base stream is decoded by the base decoder 214 (i.e. a decoding operation is applied to the encoded base stream to generate a decoded base stream). Decoding may be performed by a decoding function or mode of a base codec. The difference between the decoded base stream and the down-sampled input video is then created at a level 1 comparator 210 (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). The output of the comparator 210 may be referred to as a first set of residuals, e.g. a surface or frame of residual data, where a residual value is determined for each picture element at the resolution of the base encoder 213, the base decoder 214 and the output of the down-sampling block 205.

The difference is then encoded by a first encoder 215 (i.e. a level 1 encoder) to generate the encoded level 1 stream 202 (i.e. an encoding operation is applied to the first set of residuals to generate a first enhancement stream).

As noted above, the enhancement stream or streams may comprise a first level of enhancement
5 202 and a second level of enhancement 203. The first level of enhancement 202 may be considered to be a corrected stream, e.g. a stream that provides a level of correction to the base encoded/decoded video signal at a lower resolution than the input video 200. The second level of enhancement 203 may be considered to be a further level of enhancement that converts the corrected stream to the original input video 200, e.g. that applies a level of enhancement or
10 correction to a signal that is reconstructed from the corrected stream.

In the example of FIG. 2, the second level of enhancement 203 is created by encoding a further set of residuals. The further set of residuals are generated by a level 2 comparator 219. The level 2 comparator 219 determines a difference between an upsampled version of a decoded level 1 stream, e.g. the output of an upsampling component 217, and the input video 200. The input to the up-
15 sampling component 217 is generated by applying a first decoder (i.e. a level 1 decoder 218) to the output of the first encoder 215. This generates a decoded set of level 1 residuals. These are then combined with the output of the base decoder 214 at summation component 220. This effectively applies the level 1 residuals to the output of the base decoder 214. It allows for losses in the level 1 encoding and decoding process to be corrected by the level 2 residuals. The output of summation
20 component 220 may be seen as a simulated signal that represents an output of applying level 1 processing to the encoded base stream 201 and the encoded level 1 stream 202 at a decoder.

As noted, an upsampled stream is compared to the input video which creates a further set of residuals (i.e. a difference operation is applied to the upsampled re-created stream to generate a further set of residuals). The further set of residuals are then encoded by a second encoder 221 (i.e.
25 a level 2 encoder) as the encoded level 2 enhancement stream (i.e. an encoding operation is then applied to the further set of residuals to generate an encoded further enhancement stream).

Thus, as illustrated in FIG. 2 and described above, the output of the encoding process is a base stream 201 and one or more enhancement streams 202, 203 which preferably comprise a first level of enhancement and a further level of enhancement. It should be noted that the components shown
30 in FIG. 2 may operate on blocks or coding units of data, e.g. corresponding to 2x2 or 4x4 portions of a frame at a particular level of resolution. The components operate without any inter-block dependencies, hence they may be applied in parallel to multiple blocks or coding units within a frame. This differs from comparative video encoding schemes wherein there are dependencies between blocks (e.g., either spatial dependencies or temporal dependencies). The dependencies of
35 comparative video encoding schemes limit the level of parallelism and require a much higher complexity.

A corresponding generalised decoding process is depicted in the block diagram of FIG. 3. FIG. 3 may be said to show a low complexity decoder that corresponds to the low complexity encoder of FIG. 2. The low complexity decoder receives the three streams 201, 202, 203 generated by the low

complexity encoder together with headers 304 containing further decoding information as part of a bitstream 330. The encoded base stream 301 is decoded by a base decoder 310 corresponding to the base codec used in the low complexity encoder. The encoded level 1 stream 302 is received by a first decoder 311 (i.e. a level 1 decoder), which decodes a first set of residuals as encoded by the first encoder 215 of Figure 1. At a first summation component 312, the output of the base decoder 310 is combined with the decoded residuals obtained from the first decoder 311. The combined video, which may be said to be a level 1 reconstructed video signal, is upsampled by upsampling component 313. The encoded level 2 stream 303 is received by a second decoder 314 (i.e. a level 2 decoder). The second decoder 314 decodes a second set of residuals as encoded by the second encoder 221 of FIG. 2. Although the headers 304 are shown in FIG. 3 as being used by the second decoder 314, they may also be used by the first decoder 311 as well as the base decoder 310. The output of the second decoder 314 is a second set of decoded residuals. These may be at a higher resolution to the first set of residuals and the input to the upsampling component 313. At a second summation component 315, the second set of residuals from the second decoder 314 are combined with the output of the up-sampling component 313, i.e. an up-sampled reconstructed level 1 signal, to reconstruct decoded video 350.

As per the low complexity encoder, the low complexity decoder of FIG. 3 may operate in parallel on different blocks or coding units of a given frame of the video signal. Additionally, decoding by two or more of the base decoder 310, the first decoder 311 and the second decoder 314 may be performed in parallel. This is possible as there are no inter-block dependencies.

In the decoding process, the decoder may parse the headers 304 (which may contain global configuration information, picture or frame configuration information, and data block configuration information) and configure the low complexity decoder based on those headers. In order to re-create the input video, the low complexity decoder may decode each of the base stream, the first enhancement stream and the further or second enhancement stream. The frames of the stream may be synchronised and then combined to derive the decoded video 350. The decoded video 350 may be a lossy or lossless reconstruction of the original input video 100 depending on the configuration of the low complexity encoder and decoder. In many cases, the decoded video 350 may be a lossy reconstruction of the original input video 200 where the losses have a reduced or minimal effect on the perception of the decoded video 350.

In each of FIGS. 2 and 3, the level 2 and level 1 encoding operations may include the steps of transformation, quantization and entropy encoding (e.g., in that order). The encoding operations may also include residual ranking, weighting and filtering. Similarly, at the decoding stage, the residuals may be passed through an entropy decoder, a de-quantizer and an inverse transform module (e.g., in that order). Any suitable encoding and corresponding decoding operation may be used. Preferably however, the level 2 and level 1 encoding steps may be performed in software (e.g., as executed by one or more central or graphical processing units in an encoding device).

The transform as mentioned herein may use a directional decomposition transform such as a Hadamard-based transform. Both may comprise a small kernel or matrix that is applied to flattened

coding units of residuals (i.e. 2x2 or 4x4 blocks of residuals). More details on the transform can be found for example in patent WO 2013/171173 A1 or WO 2018/046941 A1, which are incorporated herein by reference. The encoder may select between different transforms to be used, for example between a size of kernel to be applied.

- 5 The transform may transform the residual information to four surfaces. For example, the transform may produce the following components or transformed coefficients: average, vertical, horizontal and diagonal. A particular surface may comprise all the values for a particular component, e.g. a first surface may comprise all the average values, a second all the vertical values and so on. As alluded to earlier in this disclosure, these components that are output by the transform may be taken in
10 such embodiments as the coefficients to be quantized in accordance with the described methods. A quantization scheme may be useful to create the residual signals into quanta, so that certain variables can assume only certain discrete magnitudes. Entropy encoding in this example may comprise run length encoding (RLE), then processing the encoded output is processed using a Huffman encoder. In certain cases, only one of these schemes may be used when entropy encoding
15 is desirable.

- In summary, the methods and apparatuses herein are based on an overall approach which is built over an existing encoding and/or decoding algorithm (such as MPEG standards such as AVC/H.264, HEVC/H.265, etc. as well as non-standard algorithm such as VP9, AV1, and others) which works as
20 a baseline for an enhancement layer which works accordingly to a different encoding and/or decoding approach. The idea behind the overall approach of the examples is to hierarchically encode/decode the video frame as opposed to the use of block-based approaches as used in the MPEG family of algorithms. Hierarchically encoding a frame includes generating residuals for the full frame, and then a decimated frame and so on.

- As indicated above, the processes may be applied in parallel to coding units or blocks of a colour
25 component of a frame as there are no inter-block dependencies. The encoding of each colour component within a set of colour components may also be performed in parallel (e.g., such that the operations are duplicated according to (number of frames) * (number of colour components) * (number of coding units per frame)). It should also be noted that different colour components may have a different number of coding units per frame, e.g. a luma (e.g., Y) component may be
30 processed at a higher resolution than a set of chroma (e.g., U or V) components as human vision may detect lightness changes more than colour changes.

- Thus, as illustrated and described above, the output of the decoding process is an (optional) base reconstruction, and an original signal reconstruction at a higher level. This example is particularly well-suited to creating encoded and decoded video at different frame resolutions. For example, the
35 input signal 30 may be an HD video signal comprising frames at 1920 x 1080 resolution. In certain cases, the base reconstruction and the level 2 reconstruction may both be used by a display device. For example, in cases of network traffic, the level 2 stream may be disrupted more than the level 1 and base streams (as it may contain up to 4x the amount of data where down-sampling reduces the dimensionality in each direction by 2). In this case, when traffic occurs the display device may revert

to displaying the base reconstruction while the level 2 stream is disrupted (e.g., while a level 2 reconstruction is unavailable), and then return to displaying the level 2 reconstruction when network conditions improve. A similar approach may be applied when a decoding device suffers from resource constraints, e.g. a set-top box performing a systems update may have an operation base decoder
5 220 to output the base reconstruction but may not have processing capacity to compute the level 2 reconstruction.

The encoding arrangement also enables video distributors to distribute video to a set of heterogeneous devices; those with just a base decoder 320 view the base reconstruction, whereas those with the enhancement level may view a higher-quality level 2 reconstruction. In comparative
10 cases, two full video streams at separate resolutions were required to service both sets of devices. As the level 2 and level 1 enhancement streams encode residual data, the level 2 and level 1 enhancement streams may be more efficiently encoded, e.g. distributions of residual data typically have much of their mass around 0 (i.e. where there is no difference) and typically take on a small range of values about 0. This may be particularly the case following quantization. In contrast, full
15 video streams at different resolutions will have different distributions with a non-zero mean or median that require a higher bit rate for transmission to the decoder.

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. Residuals are then transmitted to a decoder, e.g. as L-1 and L-2
20 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
25 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
30 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
35 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 residuals 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 above, 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). The transform described herein may be applied using a Hadamard matrix (e.g., a 4x4 matrix for a flattened 2x2 coding block or a 16x16 matrix for a flattened 4x4 coding block). This moves in a different direction from comparative video encoding approaches. Applying these new approaches to blocks of residuals generates compression efficiency. For example, certain transforms generate uncorrelated transformed coefficients (e.g., in space) that may be efficiently compressed. While correlations between transformed 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. Pre-processing residuals by setting certain residual values to 0 (i.e. not forwarding these for processing) may provide a controllable and flexible way to manage bitrates and stream bandwidths, as well as resource use.

This disclosure is concerned with a new architecture for a hierarchical decoder to be able to generate a signal reconstruction at different output levels such as the output levels of the hierarchical decoder of FIG. 3. The new architecture allows a decoder to process data that was encoded by an encoder in accordance with temporal mode and/or tile mode while significantly reducing on-chip memory usage versus known decoders.

Typically, a base layer of a hierarchical stream is received at a decoder in accordance with a line-based raster scan order format. Complications may occur when the enhancement layers are received in accordance with an order format that is different from the line-based raster scan order format of the base layer, e.g., in accordance with a block-based raster scan order. The complications occur because it becomes challenging for the decoder to map the enhancement layers data to the corresponding base layer data for example when reconstructing a full image or a frame of video or a component part thereof. Handling the different order formats of the base layer and enhancement layers can increase on-chip memory usage because additional steps to ensure correct alignment between the base layer data and enhancement layers data are required.

A decoder typically receives enhancement layers data as entropy encoded data. In some encoder configurations, the entropy encoded data is generated by an entropy encoder and then received at

the decoder in accordance with a block-based raster scan order. As mentioned in the paragraph above, the block-based raster scan order does not match a line-based raster scan order of the corresponding base layer data in that circumstance. The entropy encoded data is typically generated in accordance with a block-based raster scan order when temporal mode and/or tile mode are used

5 for encoding the data.

As such, the new architecture disclosed herein address a problem of how to decode efficiently a hierarchical stream when the order format of the entropy encoded enhancement layer data does not match the order format of the base layer data.

10 Although, this disclosure focuses particularly on the base layer data being in accordance with a line-based raster scan order and the enhancement layers data being in accordance with another order type, a skilled person would appreciate that the invention herein is also advantageous when the base layer data is in accordance with an order type that is different from a line-based raster scan order and that the enhancement layers data is also in accordance with a different order type to the base layer data.

15 FIG. 4 is an illustration of the order format of entropy encoded data in accordance with different modes of operation of the encoder.

1. Representation 410: This representation shows a first illustrative order format of the entropy encoded data when both tile mode and temporal mode are off during encoding. The entropy encoded data is ordered in a line-based raster scan order representing the underlying data, in this case the enhancement layer data, starting from the beginning of line 1 to the end of line 1 (as indicated by the arrow), then from the beginning of line 2 to the end of line 2, and so on until the last line, which in this example is line 9.
2. Representation 420: This representation illustrates a second illustrative order format of the entropy encoded data (which is in accordance with a block-based raster scan order) when temporal mode is on and tile mode is off during encoding. The entropy encoded data is ordered in blocks (e.g., blocks 421 and 428), and the entropy encoded data within each block is ordered in a line raster scan order (in the order of line 1, then line 2, then line 3). The blocks themselves are ordered in a raster scan order so that block 421 is ordered first, then the adjacent block to the right, and so on until the end of the row of blocks, and then starting again from the left hand most block on the next row of blocks. In this illustrative example, there are four blocks in each row of blocks, and three rows of blocks, but the number of blocks will vary depending on the block size and the underlying image data size that is being encoded. The block size will also vary depending on the configuration of the encoder.
3. Representation 430: This representation shows a third illustrative order format of the entropy encoded data (which is in accordance with a tile-based and block-based raster scan order) when tile mode is on and temporal mode is off during encoding. The entropy encoded data is ordered in tiles (e.g., tiles 433 and 437), and the tiles are further organized into blocks (e.g., block 431). The data within each block is ordered in a line raster scan order (in

the order of line 1, then line 2, then line 3), the blocks are ordered in a raster scan order, and the tiles themselves are also ordered for processing in a raster scan order. In this illustrative example, there are two tiles, with each tile comprising two blocks in each row of blocks and three rows of blocks, but the number of tiles and blocks per tile will vary depending on the block size and the underlying image data size that is being encoded. The block size will also vary depending on the configuration of the encoder.

4. Representation 440: This representation demonstrates a fourth illustrative order format of the entropy encoded data (which is in accordance with a tile-based and block-based raster scan order) when both tile mode and temporal mode are on during encoding. The fourth illustrative order format of the entropy encoded data is the same as the third illustrative order format in Block 430, with entropy encoded data being ordered in a line raster scan order within each block, blocks being ordered in a raster scan order, and tiles being ordered in a raster scan order.

This disclosure is concerned with reordering of entropy encoded data prior to a decoding operation.

In the examples given, the reordering of the entropy encoded data is from the format of representations 420, 430 or 440 to that of representation 410 to match the typical base layer format. However, the principles may be applied more broadly to other format reordering.

In the novel architecture disclosed herein, the received entropy encoded data is reordered from a first order to a second order. The first order corresponds to the format of the entropy encoded data as it arrives at the decoder for decoding. The second order, on the other hand, would typically align with a format of a base layer signal of the scalable video stream. The first order would typically be in a block-based order as shown in FIG. 4 (420, 430, 440). The second order would typically be in a line-based order as shown in FIG. 4 (410). The first order could be in the form of independently decodable tiles if tile mode was active during encoding, with or without blocks (blocks are referred to as temporal blocks when temporal mode is active). The first order could be in the form of independently decodable temporal blocks if temporal mode was active during encoding. The reordered entropy encoded data in the second order, once decoded from the entropy encoded format, can be combined with the base layer signal to enhance the overall quality of the video represented by the base layer. The entropy encoded data may represent residual/enhancement information as specified above and in the MPEG-5 Part 2 LCEVC standard, e.g., level 1 enhancements which correct for base decoding artefacts and/or level 2 enhancements which increases at least the resolution and/or bit depth of the base layer signal.

FIG. 5 illustrates a two-stage decoding architecture 500 in accordance with an aspect of the invention. The decoding architecture 500 comprises a pre-decoder 510 and a decoder 520. The pre-decoder 510 performs a pre-decode stage which is an initial pre-processing of received entropy encoded data before the decode stage at decoder 520 which decodes the pre-processed entropy encoded data. The pre-processing of the entropy encoded data includes reordering of the entropy encoded data.

The pre-decoder 510 signals meta-data to the decoder 520 via a shared on-chip memory 515. The meta-data is used to store state initialisation information and to minimise external memory accesses. Both the pre-decoder 510 and decoder 520 interact with an external memory 540 via an external memory interface 530. In this example implementation, the interface 530 comprises a DDR (Double Data Rate) controller and a DMA (Direct Memory Access) engine, which are used for managing data transfers between the pre-decoder 510, the decoder 520 and the external memory 540. In this implementation, the external memory 540 comprises DRAM (Dynamic Random Access Memory), in particular DDR SDRAM is useful. The external memory 540 is configured to store entropy encoded data for processing by the pre-decoder 510 and also serves as a buffer for the processed pre-decoded entropy encoded data before processing by the decoder 520.

Pre-Decode Stage

Payload Data Reordering

As mentioned above one of the challenges of decoding a scalable video stream such as a scalable video stream in accordance with the MPEG-5 Part 2 LCEVC standard is that the ordering of the entropy encoded data can vary. For example, in tile mode, the entropy encoded data is formatted in independently decodable tiles which can take on configurable rectangular shapes. When temporal mode is enabled, the entropy encoded data is arranged into temporal blocks (8x8 in DDS-mode and 16x16 in DD-mode). Tile mode also typically uses blocks within each tile, similar to the temporal blocks. DDS-Mode stands for Directional Decomposition Squared mode, and DD-Mode stands for Directional Decomposition mode, as defined in the MPEG-5 Part 2 LCEVC standard. The base layer data can also be presented in various formats, although is often presented in a streaming raster order or in compact tiles (e.g., 32x8). The LCEVC decoder must be able to support the decoding of these different combinations to avoid playback issues and degradation of quality. One solution is to utilise line buffers to perform line-to-block and block-to-line conversions, however these consume significant on-chip memory resources. Another solution is to reorder the compressed payload data (i.e., the enhancement layer's entropy encoded data) into one format, irrespective of the bitstream configuration.

A key concept in the new architecture is to reorder the entropy encoded data into a format, for example a raster format, to match the format of the base layer (typically a line raster scan format), irrespective of the runtime configuration of the encoded bitstream, comprising the entropy encoded data, received by a decoder. The two bitstream dependent options that dictate this process are whether tile mode is enabled and whether temporal mode is enabled (or both). The pre-decoder reads configuration information from the received bitstream, and reorders the entropy encoded data based on the configuration information. The configuration information indicates which encoding mode was enabled in the encoding process from one or both of a tile mode and a temporal mode.

FIG. 6 illustrates the operations of the pre-decoder 510 of FIG. 5 in more detail. The pre-decoder 510 consists of three main blocks: a prefix decoder 610 which is responsible for the initial processing of the entropy encoded data; split/stitch logic 620 which manages the division and combination of data streams as required; and pre-decode buffer control 630 which oversees the allocation and

management of buffers during the pre-decoding process. The pre-decoder 510 comprises a tile_mode/temporal_mode decision module 615 which checks if tile mode and/or temporal mode were used during the encoding process of the entropy encoded data, e.g., by reading the configuration information. If tile mode and/or temporal mode were used during the encoding process, the tile_mode/temporal_mode decision module 615 directs the Split/Stitch Logic 620 to operate on the entropy encoded data. However, if both tile mode and temporal mode were not used during the encoding process, the tile_mode/temporal_mode decision module 615 skips the Split/Stitch Logic 620 and directs the entropy encoded data to external memory 540.

The split/stitch logic 620 comprises a split module 623, a temporal mode stitch module 626, a tile_mode decision module 628 and tile pseudo-stitch module 629. The split module 623 is responsible for dividing the entropy encoded data in preparation for reordering of the entropy encoded data according to the encoding mode. The split module 623 directs the divided data to the temporal stitch module 626 where it is stitched into raster lines. After temporal stitching, if tile mode was on during the encoding process, the tile_mode decision module 628 directs tile pseudo-stitching to occur at tile pseudo-stitch module 629. If tile mode was not on during the encoding process, the tile_mode decision module 628 directs the entropy encoded data from temporal stitching module 626 to be passed to pre-decode buffer control 630 without tile pseudo-stitching. Detailed explanations of the splitting, temporal stitching, and tile pseudo-stitching processes follow below.

The pre-decoder 510 can operate in four different modes depending on the order format of the entropy encoded data received for processing (referring back to FIG. 4 is helpful for an understanding):

1. Tile mode and temporal mode off (410): If tile mode and temporal mode were both disabled in the encoding process of a received bitstream at the pre-decoder 510, the prefix decoder 610 prefix decodes the entropy encoded data and the tile_mode/temporal_mode decision module 615 skips the Split/Stitch Logic 620 and directs the entropy encoded data to external memory 540. In this mode of operation, the entropy encoded data exists in a line-based raster scan order and will be processed by the run length decoder and subsequent decoding stage in that order, where each line of data is processed sequentially from line 1 to line 9, see representation 410.
2. Tile mode off and temporal mode on (420): If temporal mode is enabled in the encoding process of a received bitstream at the pre-decoder 510, the pre-decoder 510 de-rasterizes (reorders) the entropy encoded data by splitting and stitching the entropy encoded data in split/stitch logic 620 (this is explained in more detail below). Essentially lines 1, 2, and 3 of the first block (and 4, 5, 6 of the second block etc) are created by the split of that part of the entropy encoded data, and then lines 1, 4, 7 and 10 of the first four blocks are stitched together to create a line that can be processed by the run length decoder and subsequent decoding stage. This eliminates the need for block to line conversions in the uncompressed data domain, saving a significant amount of on-chip memory. Each temporal block line is written to a known (calculated) offset in the external memory to achieve the result of block

lines 1, 4, 7 and 10 being later readable and output in order as a single line. The entropy encoded data is processed from a block-based raster scan order to a line-based raster scan order.

5 3. Tile mode on and temporal mode off (430): If tile mode is enabled in the encoding process of a received bitstream at the pre-decoder 510, the entropy encoded data is de-rasterized tile-by-tile by the split/stitch logic 620. This eliminates the need for block-to-line and tile-to-line conversions in the uncompressed data domain, saving a significant amount of on-chip memory. Each line of each block within each tile is written to the external memory 540
10 using offsets. Both the temporal stitching 626 (for each adjacent block within a tile) and tile pseudo-stitching 629 (for each adjacent tile) are performed. Each block line is written to a known (calculated) offset in the external memory to achieve the result of lines 1, 4, 19 and 22 (etc) being later readable and output in order as a single line. The entropy encoded data is processed from a tile-based block-based raster scan order to a line-based raster scan
15 order.

 4. Tile mode and temporal mode on: When both tile mode and temporal mode are enabled in the encoding process of a received bitstream at the pre-decoder, the entropy encoded data is de-rasterized by the split/stitch logic 620 and each line of each tile is written to external
20 memory 640 using offsets. Both the temporal stitching 626 (for each adjacent block within a tile) and tile pseudo-stitching 629 (for each adjacent tile) are performed. Similar to mode 3, the entropy encoded data is processed from a tile-based block-based raster scan order to a line-based raster scan order.

25 By writing the de-rasterized data to known offsets in the external memory following splitting and stitching, the decoder can read the entropy encoded data in line raster scan order without storing meta-data for each offset.

 The entropy encoded data is in accordance with the MPEG-5 Part 2 LCEVC standard. In the exemplary
30 embodiments, the entropy encoded data is run length encoded and the run length encoding generates context data to describe the run length encoded data. In the examples described, the context data comprises the following structure: LSB (Least Significant Byte) which represents small magnitude non-zero data, MSB (Most Significant Byte) which is used when the magnitude of the data exceeds the bit width of the LSB, and RUN which represents a run of zeros. The RUN helps in
35 compressing sparse data by indicating the number of consecutive zeros efficiently. The LSB contains two flags to indicate what type of data follows. A first flag is a RUN bit (e.g. bit 7 of the LSB) and the first flag indicates that the next context data in the stream will be a RUN. A second flag is a MSB bit (e.g. bit 0) of the LSB which indicates that the next context is a MSB.

The decoder 520 uses the context data to perform run length decoding.

Prefix Decoder

The prefix decoder 610 is responsible for reading the entropy encoded data and performing prefix decoding. An advantageous implementation of the new architecture is the following:

- 5 • In an example streaming parallel decoder with one entropy decoder per layer there are 33 prefix decoders (temporal plus 2 LoQs of 16 layers) feeding the same number of run length decoders. When prefix decoding is performed in the pre-decode stage, a scheduling module could run a reduced number of prefix decoders iteratively, for example 5 (4 for the residual layers and one temporal layer). This would save a significant amount of logic and minimise the impact of the addition of split/stitch logic and the pre-decode buffer controllers.

10 FIG. 7 is a block diagram illustrating scheduling of the prefix decoders 610. The pre-decode scheduling starts with the pre-decode scheduler module 700 automatically initiating the prefix decoders 610 to cover all levels and layers (the four residual layers and one temporal layer) by setting the correct external memory offsets for the input bitstream. Note, the prefix decoders 610 operate on one layer at a time. If tile mode is enabled, all tiles associated with the current layer are
15 decoded by the prefix decoder before moving onto the next layer.

Split/Temporal Stitch Logic

Returning to FIG. 6, the following is a non-limiting example implementation for understanding the operation of the split/temporal stitching logic 620.

20 The split/stitch logic 620 is the conduit between the prefix decoder 610 and the pre-decode buffer control 630. In this example, the pre-decode buffer control 630 controls 16 buffers in total. Depending on the transform used at the encoder, either 8 (DDS transform) or 16 (DD transform) buffers are used within the pre-decode buffer control 630. The buffers are sized so that they can store two DDR burst lengths of data which allows enough space for accumulating a DDR burst and sufficient space to continue writing into the buffer while waiting for a flush to the external memory
25 540. External memory systems generally perform more efficiently with long, sequential bursts. In the following example, it is assumed that the minimum burst length to sustain a high bandwidth efficiency is a burst with a length of 8 on a 512-bit data bus. The buffers are used as a conceptual FIFO to accumulate a DDR burst length of data from the current line being pre-decoded.

30 When temporal mode and/or tile mode is/are enabled, the run length contexts are reordered by the split/stitch logic 620 which splits the entropy encoded data into smaller segments and then stitches the entropy encoded data segments together in a different order. To do this, the contexts are analysed as they are output from the prefix decoder 610 by counting how many transform coefficients they will produce. The current line in the current temporal block is tracked and a corresponding buffer (a buffer is used for each corresponding line in each block) is filled with the
35 contexts of the current line. When the end of the current line of the current temporal block is reached, either an LSB, an LSB/MSB pair or a RUN context will be encountered. If it is an LSB or LSB/MSB pair, the current line in the current temporal block is incremented as is the buffer and the

process continues until the end of the current block (the process then continues for other blocks). If a RUN is encountered, the RUN is 'split' into two RUNs, one that finishes the current line and the remainder is used at the start of the next line of the temporal block.

5 The most recent context is stored in a register rather than being written into the buffer. The type of the next context to arrive from the prefix decoder is compared with the registered context. If the previous context is a RUN and so is the next context, they will be 'stitched' together to form a single RUN context before overwriting the register. If the next context is an LSB, the previous context is simply written into the buffer and is replaced in the register by the next context.

10 Because the contexts within a block are being split to be suitable for creating lines, the split contexts will not always necessarily start with an LSB. Consequently, a single bit to indicate if the first context of the line is a RUN or an LSB is stored into the shared on-chip memory or a scratch area of the external memory. The decode stage can then read in the first context states for each line and initialize the RLD state machines accordingly.

15 To minimize the accesses to the external memory 540 in both the pre-decode and decode stages, a 'zero line' bit is also stored in the shared on-chip memory or in the scratch area of the external memory 540. This 'zero line' bit indicates that the entire layer line is all zeros which eliminates the need for the buffer control to write to the external memory for this line and for the data fetcher in the decode stage to read from the external memory for this line.

20 There are special cases where there is only one code in the prefix coding tree. These special cases consume 0-bits in the entropy encoded data. When the pre-decode stage prefix decodes these special cases, they are expanded to 8-bits of data. To minimise the impact on external memory bandwidth in such cases, a 'special-data' field is stored in the shared on-chip memory 515 or the external memory scratch area. This meta-data field contains a flag to indicate if an LSB or MSB context are special and if they are, what the symbol is for that context type. The split/stitch logic
25 620 does not write any data to the buffer control logic 630 if the data is of a context type that has been marked as special. The prefetcher reads the special-data field from the meta-data and if it encounters a context in the entropy data that has been marked as special, it inserts the symbol from the meta-data rather than reading it from external memory. If an LSB context is special and neither an MSB nor a RUN context exist in the entropy encoded data, then all of the contexts are the same
30 value and no reordering is necessary. RUN contexts are never marked as special because the split/stitch logic will need to generate new RUN contexts of different lengths during the reordering process.

When the stitch logic of the split/stitch logic 620 stitches contexts together between lines of adjacent temporal blocks, the type of the adjacent contexts may change (due to the reordering process). For
35 example, an LSB context may expect a RUN context to follow but after reordering, the LSB context may be followed by another LSB context. In this case, bit 7 of the LSB context (that indicates that a RUN context follows), must be set to 0 from its original value of 1. For special-data cases, the value of the symbol cannot change as it is fixed for the entire layer and stored in the meta-data. To work around this, if an LSB to RUN sequence is reordered to LSB to LSB, a RUN context with a value

of 0 can be inserted. If an MSB to LSB sequence is reordered to MSB to RUN, an LSB with an effective value of 0 can be inserted with bit 7 set to 1. The RUN that follows then must have its value decremented by 1. If an MSB to RUN sequence is reordered to MSB to LSB, a RUN context with a value of 0 can be inserted.

- 5 A small on-chip memory is used to accumulate the first-context and zero-line bits prior to writing to the external memory which eliminates short bursts to and from the external memory. Some implementations may favour the use of on-chip memory over external memory (or vice versa) for meta-data. The first-context, zero-line and special-data meta-data fields can be stored in either the shared on-chip memory or a scratch space in the external memory. If the meta-data is stored in
- 10 external memory, a small on-chip memory buffer can be used to accumulate the meta-data prior to writing to the external memory. Such a buffer eliminates short bursts to and from the external memory.

- When tile mode is enabled, two or more tiles are used which causes tile boundaries which need to be handled. Each tile is also handled in blocks like the temporal mode, but the entropy encoded data
- 15 is ordered in a block-based raster scan order within each tile, creating a further complication for reordering the entropy encoded data into a line-based raster scan order. Tile pseudo-stitching 629 is used and is discussed later.

Buffer Control

- As previously discussed, 16 buffers are used for the temporal block de-rasterization (in DD mode, the temporal blocks are 16x16). The length of these buffers is set to 2 times the minimum external
- 20 memory burst length that maintains a high bandwidth efficiency (in this example the minimum burst length is 8 x 512-bits).

- There are 5 instances of the buffer sets (one for each prefix decoder 610 and split/stitch pair). The on-chip memory requirements are listed in Table 1 below for an example bus width of 512-bits and
- 25 a burst length of 8.

Table 1

Instances	Buffers per instance	Buffer length (bytes)	Total (bytes)
5	16	1024	81920

- The buffer control logic 630 is responsible for tracking the current line/tile/layer/level and setting the external memory write offset accordingly. When temporal mode is enabled, an external memory
- 30 offset is calculated for each line of each layer of each level.

The calculation for $Addr_i$, the address for line i $[0, LH]$ within a layer is:

$$Addr_i = i \times \text{ceil}(LW \times 2/A) \times A$$

Here:

- *LW* represents the layer width (tile layer width when tiling mode is enabled).
- *LH* represents the layer height (tile layer height when tiling mode is enabled).
- *A* represents the external memory alignment requirement in bytes.

- 5 This calculation ensures that the addresses are aligned to the specified A-byte boundaries within the external memory.

This approach simplifies both the writing process to the external memory during the pre-decode stage and the subsequent reading process from the external memory during the decode stage. When a buffer contains an external memory burst length of data, it is flushed. The buffers are also flushed
10 when the end of the current line is reached.

FIG. 8A is a block diagram depicting storage of a DDS layer 800 in DDR. The DDS layer 800 contains temporal blocks, with the numbers on the arrows indicating the sequence in which the temporal blocks have been encoded.

FIG. 8B is a block diagram depicting conceptually how the split/stitch logic 620 together with buffers
15 850 act to split and stitch and buffer the entropy encoded data (i.e. the context data as described above). In this example, there are 8 buffers. Each buffer is used to buffer entropy encoded data from a corresponding line within each of the temporal blocks in a row of blocks. The top lines (lines 0, 8, ..., and 128) of each the top row of blocks 800-0 comprising blocks 800-0-0, 800-0-1, ..., and 800-0-15 in DDS layer 800 are split and stored (and also stitched together) in a first buffer 850-0.
20 The second to top lines (lines 1, 9, ..., and 129) are split and stored (and stitched) in a second buffer 850-1. This process is repeated for all lines of the top row of blocks 800-0. The process performed on the first row of blocks 800-0 is then repeated for the second row of blocks 800-1 starting at block 800-1-0, and so on until the last row of blocks 800-n. Once the top blocks are processed, the buffers 850 are reused for the next line of blocks ensuring efficient memory usage. The data from buffers
25 850 is written to DDR after each row of blocks is processed. As can be seen, the output from buffers 850 to DDR is in accordance with a line-based raster scan order.

When tile mode is enabled, entropy encoded data is reordered first of all within a tile as above. Then, the pseudo-stitching process of tile pseudo-stitch 629 is used between horizontally adjacent tiles to create the final line raster scan order. In more detail, for each tile in a first column of tiles, the length
30 of the context data for each line in the reordered tile is stored in a register file. The length is stored as:

$$\text{ceil}(LL/A)$$

Where:

- *LL* is the total context data length for the line in bytes.
 - *A* represents the DDR alignment requirement in bytes.
- 35

When the horizontally adjacent tile from the next column is pre-decoded, tile line data is written to DDR at the offset indicated by the stored length of the previous line. The current line length is then written over the previous line length for use in the next tile pre-decoding step.

FIG. 9 is a block diagram depicting storage of context data in DDR after tile line pseudo-stitching.

- 5 One effect of writing the de-rasterized data to known offsets in the external memory 540 is that a fixed, worst-case footprint for the reordered payload data must be allocated in the external memory particularly for ASIC/FPGA implementations of the invention. This worst case must assume that an LSB (least significant bit)/MSB (most significant bit) context pair is required for each transform coefficient. Note that while this footprint needs to be allocated, under standard operating conditions,
- 10 the reordered data will be very sparse and does not need to be read/written in its entirety. Also, the footprint allocated must be aligned to DDR burst boundaries, so the allocated footprint will typically be larger than a calculated footprint based on the worst case as the footprint will be rounded up to the nearest DDR burst boundary.

- FIG. 9 shows the de-rasterized data as stored in the external memory using the allocated footprint as an illustrative example. A first line of entropy encoded data 900-a (i.e. the stitched and pseudo-stitched context data) is shown stored in external memory 540 in accordance with the allocated footprint. In this example, the first line includes stitched context data from four tiles from Tile0 to Tile3, namely context data 910a, 920a, 930a, and 940a. The context data 910a corresponds to the output of buffer 850-0 from FIG. 8.
- 15

- 20 Between the stitched context data 910a of Tile0 and the stitched context data 920a of Tile1, there is unused memory space 915a because the allocated footprint is larger than the actual context data 910a. Between context data 920a for Tile1 and context data 930a for Tile2 there is unused memory space 925-a. As would be understood, the unused memory space could occur at several tile boundaries for each of lines 900-a, 900-b, 900-c, ..., 900-i. The unused memory spaces may be filled
- 25 with padding.

Three vertical dotted lines show the footprint allocated for each context data of each tile on each line. As will be realised, on each line, subsequent tile context data will be stored using an offset that corresponds to the footprint. Aligning the footprint with DDR burst boundaries avoids an inefficient number of read and write operations.

- 30 The same is repeated for lines 900b, 900c, ..., 900i. The context data 910b corresponds to the output of buffer 850-1 from FIG. 8, and 910c to 850-2, and 910i to 850-7.

Moving the prefix decoding to the pre-decoder stage from the decode stage and de-rasterizing the context data before writing them to the external memory offers several advantages:

- 35 1. Synchronisation between prefix decoders can be eliminated in streaming parallel decoder architectures where there is a prefix decoder per layer. This is achieved by bouncing the entropy encoded data through external memory, rather than having flow control

dependencies between prefix decoders. Consequently, stalling is prevented, and prefix decoder throughput potential is maximised.

2. Due to the parallelism and increased throughput advantage, the total number of prefix decoders used in the pre-decode stage can be reduced thereby allowing for a more area efficient implementation
3. On Chip Memory and Logic Savings: Eliminates the need for block to line conversions in the uncompressed data domain leading to a substantial reduction in on-chip memory usage and logic resources.
4. Error Resilience: Enhances error resilience by enabling the pre-decode stage to signal the decode stage if errors are detected in the LCEVC bitstream, thus facilitating error handling and recovery.

In one implementation, the pre-decode stage finishes completely before the decode stage begins which facilitates enhanced error resilience (the pre-decode stage checks that the payload data produces the correct number of transform coefficients). In another implementation, the decode stage begins before the pre-decode stage finishes completely so that both operations may overlap, in as far as the decode stage has enough input information from the pre-decode stage.

Decoding Stage

A data fetcher of the decoding stage is responsible for retrieving the pre-decoded data from external memory and supplying the RLD with the relevant input. The data fetcher also generates RUN contexts when it detects that an entire line consists of zeros (from the zero-line bits in the shared on-chip memory or the external memory). This optimization eliminates the need to unnecessarily fetch context data from external memory. Additionally, this module directs the RLD to initialize its state machine based on the first context type of each line (from the first-context bits in the shared on-chip memory or the external memory). The data fetcher keeps track of the current position in the output that is being decoded and selects the correct external memory offset to read from. When tile mode and temporal mode are both disabled, this is just a single offset for the entire layer. When either tile mode or temporal mode are enabled, one offset for each line is selected. It is also the responsibility of the data fetcher to remove, where necessary, the padding that is inserted between pseudo-stitched tile lines before passing the context data onto the run length decoder.

The above embodiments are to be understood as illustrative examples. Further embodiments are envisaged. It is to be understood that any feature described in relation to any one embodiment may be used alone or in combination with other features described, and may also be used in combination with one or more features of any other of the embodiments, or any combination of any other of the embodiments. Furthermore, equivalents and modifications not described above may also be employed without departing from the scope of the invention, which is defined in the accompanying claims.

Claims

1. A method for reordering entropy encoded data prior to a decoding operation, the method comprising:
receiving the entropy encoded data in a bitstream; and
5 reordering the entropy encoded data from a first order to a second order.
2. The method of claim 1, wherein the bitstream is an enhancement layer bitstream.
3. The method of either of claim 1 or 2, wherein the second order is a predetermined order.
- 10 4. The method of any preceding claim, wherein the method further comprises reading configuration information from the received video bitstream, and wherein the reordering of the entropy encoded data is based on the configuration information.
- 15 5. The method of claim 4, wherein the configuration information indicates which encoding mode was enabled in the encoding process responsible for encoding the entropy encoded data from one or more of a tile mode and/or a temporal mode.
- 20 6. The method of claim 5, wherein the reordering of the entropy encoded data is based on the enabled encoding mode.
7. The method of any preceding claim, wherein the second order is a raster scan order.
8. The method of any preceding claim, wherein the reordering of the entropy encoded data
25 functionally removes tiles and/or temporal blocks by ordering the entropy encoded data into a raster scan order.
9. The method of any preceding claim, wherein the second order matches an order of a base layer of the bitstream.
- 30 10. The method of any preceding claim, wherein the reordering of the entropy encoded data comprises identifying distinct segments within the entropy encoded data.
- 35 11. The method of claim 10, wherein the distinct segments are adjacent segments in the entropy encoded data.
12. The method of either of claims 10 or 11, wherein identifying the distinct segments is based on analysing run length contexts within the entropy encoded data.
- 40 13. The method of claim 12, wherein the reordering of the entropy encoded data comprises combining the identified segments to form a data stream according to a raster scan order.

14. The method of claim 13, wherein the combining of identified segments to form a data stream according to a raster scan order comprises combining based on the run length context of the identified segments.
- 5 15. The method of any preceding claim, wherein the bitstream comprises multiple layers each comprising entropy encoded data.
- 10 16. The method of claim 15, wherein the method further comprises scheduling a group of modules to reorder the entropy encoded data, wherein at least one module of the group of modules reorders two or more layers of the multiple layers.
17. The method of any preceding claim, wherein the method comprises storing the output of the reordering in either an off-chip memory or an on-chip memory.
- 15 18. The method of any preceding claim, wherein the entropy encoded data is reordered before the decoding operation starts.
19. The method of claim 18, wherein the method further pauses the decoding operation if an error is determined in the reordering operation.
- 20 20. The method of any preceding claim, wherein the method comprises determining that the entropy encoded data is Huffman encoded and performing a Huffman decoding operation on the entropy encoded data before reordering the entropy encoded data.
- 25 21. The method of any of claims 4-20 when dependent on claim 4, wherein the method further comprises setting memory write offsets to the off-chip memory based on the configuration information.
22. An apparatus configured to perform the method of any preceding claim.
- 30 23. A computer readable storage medium comprising instructions, when executed by a processor, causes the processor to perform the method of any of claims 1-21.

35