

EFFICIENT EDGEBREAKER SPIRAL REVERSI IMPLEMENTATION

5 CROSS-REFERENCE

[0001] This application claims the priority of European Patent Application No. 24305005.1, filed 2 January 2024, entitled "Efficient Edgebreaker Spiral Reversi Implementation," which is incorporated herein by reference in its entirety.

BACKGROUND

10 [0002] Following the MPEG V-Mesh (now renamed V-DMC) CfP the solution proposed by Apple was selected to become the foundation of the MPEG V-Mesh Test Model (TM). This proposal is described at K. Mammou, J. Kim, A. Tourapis and D. Podborski, "m59281 - [V-CG] Apple's Dynamic Mesh Coding CfP Response," Apple Inc, 2022. An overview of mesh encoding and decoding are illustrated in FIGs. 1 and 2, respectively. FIG. 1 illustrates an
15 MPEG TM Intra frame encoding process. FIG. 2 illustrates an MPEG TM Intra frame decoding process. The test model, for some frames, uses a static mesh coder for the encoding of a base mesh that is then subdivided to obtain an approximation of the original mesh.

[0003] Until v3.0, the V-Mesh test model encoded the base mesh using the Google Draco implementation of the state-of-the-art mesh encoder named edgebreaker, and more
20 specifically its spiral reversi variant which they adapted to work on a Corner Table (CT). Edgebreaker is described in, for example, J. Rossignac, "Edgebreaker: Connectivity compression for triangle meshes," GVU center, Georgia Institute of Technology, 1999 ("Rossignac (1999)"); and J. Rossignac, "3D compression made simple: Edgebreaker with ZipandWrap on a corner-table," in Proceedings International Conference on Shape Modeling
25 and Applications, Genova, Italy, 2001 ("Rossignac (2001)"). The spiral reversi variant is described in M. Isenburg and J. Snoeyink, "Spirale Reversi: Reverse decoding of the Edgebreaker encoding," Computational Geometry, vol. 20, pp. 39-52, 2001 ("Isenburg & Snoeyink (2001)").

[0004] Note that Rossignac (1999) originally decodes in a forward manner and operates on a
30 half edge representation of the mesh. Later, Rossignac (1999) proposed a modified version, still decoding forward but running on a corner table representation of the mesh, which leads to a very compact algorithm. Spiral reversi, described by Isenburg & Snoeyink (2001), proposes an extension of Rossignac (1999), still running on a half edge representation, but performing the decoding in a reverse manner, which leads to a decoding on $O(n)$ in one pass
35 which is a better performance than Rossignac (1999) (worst case time complexity of $O(n^2)$) and Rossignac (2001) ($O(n)$ but multi pass).

[0005] For the standardization of V-DMC, it is desirable to provide an MPEG alternative implementation of the edgebreaker that could be used in the test model reference software and extended/specified as needed. InterDigital handled the task of crafting the first version of the software. The version that was adopted for the TM v3.0 is detailed in the following MPEG contributions: J.-E. Marvie and O. Mocquard, "m62603 - [V-DMC][EE4.4] Edgebreaker base mesh codec integration," MPEG meeting 142, Antalya, 2023; and J.-E. Marvie and O. Mocquard, "m63344 -[V-DMC][EE4.4]-an-efficient-Edgebreaker-implementation," MPEG meeting 142, Antalya. This software is further described in European Patent Application No. 23305505.2, filed 6 April 2023. In that application, specific adaptations of the forward edgebreaker are presented, working on a corner table that were made to support the V-Mesh TM requirements as well as the associated bitstream syntax. Further extensions are described in the following European Patent Applications:

- European Patent Application No. 23305941.9, filed 14 June 2023 (describing the coding of per-face attributes).
- European Patent Application No. 23306154.8, filed 7 July 2023 (describing alternative vertex traversals).
- European Patent Application No. 23306155.5, filed 7 July 2023 (describing prediction-degree based motion estimation).
- European Patent Application No. 23306292.6, filed 26 July 2023 (describing per-face attribute optimization).

[0006] For some meshes, especially the large ones with regular topology (vertex valence average of 6 neighbors), some specific extensions (known as valence driven encoding) can be used to better encode the CLERS table, such as those described in A. Szymczak, "Optimized Edgebreaker encoding for large and regular triangle meshes," *The Visual Computer*, vol. 19, pp. 271-278, 2003. However, this solution is configured for the reverse version of the edgebreaker. A first version of the reverse version is presented in Isenburg & Snoeyink (2001), which works on a half edge data structure. Later a version working on a corner table data structure, and supporting surfaces with holes and handles, was presented in T. Lewiner, H. Lopes, J. Rossignac and A. W. Vieira, "Efficient Edgebreaker for Surfaces of Arbitrary Topology," in *Proceedings, 17th Brazilian Symposium on Computer Graphics and Image Processing*, 2004 ("Lewiner et al. (2004)").

SUMMARY

[0007] A mesh encoding method according to an example embodiment comprises: encoding a connectivity of a triangular mesh in a bitstream using edgebreaker coding; signaling in the bitstream information indicating a direction of the connectivity encoding; and in response to a
5 determination that the information indicates reverse edgebreaker coding, signaling in the bitstream a parameter indicating a number of triangles in the triangular mesh.

[0008] In some embodiments, the method further includes signaling, separate from the parameter indicating a number of triangles in the triangular mesh, a parameter indicating a number of encoded CLERS symbols.

10 **[0009]** In some embodiments, the method further includes, in response to a determination that the information indicates reverse edgebreaker coding, signaling in the bitstream a parameter indicating a number of start positions in the mesh.

[0010] Some embodiments further include, in response to a determination that the information indicates reverse edgebreaker coding, signaling in the bitstream a parameter indicating a
15 number of connected components with boundaries in the triangular mesh

[0011] A mesh decoding method according to some embodiments comprises: obtaining a bitstream representing a connectivity of a triangular mesh encoded using edgebreaker coding; reading from the bitstream information indicating a direction of the connectivity encoding; in response to a determination that the information indicates reverse edgebreaker coding,
20 reading from the bitstream a parameter indicating a number of triangles in the triangular mesh; and using the parameter indicating a number of triangles in the triangular mesh, reconstructing the connectivity of the triangular mesh from the bitstream.

[0012] Some embodiments further include reading from the bitstream a parameter indicating a number of encoded CLERS symbols, which may be separate from the parameter indicating
25 a number of triangles in the triangular mesh.

[0013] Some embodiments further include, in response to a determination that the information indicates reverse edgebreaker coding, reading from the bitstream a parameter indicating a number of start positions in the mesh.

30 **[0014]** Some embodiments further include, in response to a determination that the information indicates reverse edgebreaker coding, reading from the bitstream a parameter indicating a number of connected components with boundaries in the triangular mesh.

[0015] An apparatus according to some embodiments comprises one or more processors, the apparatus being configured to perform any of the methods described herein.

[0016] An apparatus according to some embodiments comprises at least one processor and a computer-readable medium storing instructions for performing any of the methods described herein.

5 [0017] A computer-readable medium according to some embodiments store instructions for performing any of the methods described herein.

[0018] A computer-readable medium according to some embodiments stores a mesh encoded according to any of the methods described herein.

[0019] A signal according to some embodiments conveys a mesh encoded according to any of the methods described herein.

10

BRIEF DESCRIPTION OF THE DRAWINGS

[0020] FIG. 1 is a flow diagram illustrating an overview of an MPEG Test Model Intra frame encoding process.

15 [0021] FIG. 2 is a flow diagram illustrating an overview of an MPEG Test Model Intra frame decoding process.

[0022] FIG. 3 is a flow diagram illustrating an example of a complete edgebreaker mesh codec. The top row is the encoding line, and the bottom row is the decoding line.

[0023] FIG. 4 illustrates example source code for an encoding procedure according to an example embodiment.

20 [0024] FIG. 5 illustrates example source code for a decoding procedure according to an example embodiment.

[0025] FIG. 6 illustrates example source code for a topology encoding procedure according to an example embodiment.

25 [0026] FIG. 7 illustrates the procedure compressComponent according to an example embodiment.

[0027] FIG. 8 illustrates the procedure writeBoundary, which goes through the boundary of _b (a corner given as parameter) and writes each vertex once in order.

[0028] FIG. 9 illustrates the procedure checkHandle according to an example embodiment.

[0029] FIG. 10 illustrates the procedure writeVertex according to an example embodiment.

30 [0030] FIG. 11 illustrates the procedure topoPrepass according to an example embodiment.

[0031] FIG. 12 illustrates the procedure spiraleReversi according to an example embodiment.

[0032] FIG. 13 illustrates the procedure readBoundary according to an example embodiment.

[0033] FIG. 14 illustrates the procedure closeStar according to an example embodiment.

[0034] FIG. 15 illustrates the procedure readVertex according to an example embodiment.

[0035] FIG. 16 illustrates the procedure postReindex according to an example embodiment.

5 [0036] FIG. 17 illustrates the procedure encodeAttributesWithMainIndexTable according to an example embodiment.

[0037] FIG. 18 illustrates the procedure encodeAttributesWithPrediction according to an example embodiment.

10 [0038] FIG. 19 illustrates the procedure encodeAttributesWithAuxiliaryIndexTable according to an example embodiment.

[0039] FIG. 20 illustrates the AUX_UV_ENCODE statement to be copied in FIG. 19 according to an example embodiment.

[0040] FIG. 21 illustrates a first part of the procedure posEncodeWithPrediction according to an example embodiment.

15 [0041] FIG. 22 illustrates a second part of the procedure posEncodeWithPrediction according to an example embodiment.

[0042] FIG. 23 illustrates a first part of the procedure uvEncodeWithPrediction according to an example embodiment.

20 [0043] FIG. 24 illustrates a second part of the procedure uvEncodeWithPrediction according to an example embodiment.

[0044] FIG. 25 illustrates a procedure predictUV for encoding according to an example embodiment.

[0045] FIG. 26 illustrates a procedure decodeAttributesWithMainIndexTable according to an example embodiment.

25 [0046] FIG. 27 illustrates a procedure decodeAttributesWithPrediction according to an example embodiment.

[0047] FIG. 28 illustrates a first part of a procedure decodeAttributesWithAuxiliaryIndexTable according to an example embodiment.

30 [0048] FIG. 29 illustrates a second part of the procedure decodeAttributesWithAuxiliaryIndexTable according to an example embodiment.

[0049] FIG. 30 illustrates a third part of the procedure `decodeAttributesWithAuxiliaryIndexTable` according to an example embodiment.

[0050] FIG. 31 illustrates an `AUX_UV_DECODE` statement to be copied in FIG. 30 according to an example embodiment.

5 [0051] FIG. 32 illustrates a first part of the procedure `posDecodeWithPredictions` according to an example embodiment.

[0052] FIG. 33 illustrates a second part of the procedure `posDecodeWithPredictions` according to an example embodiment.

10 [0053] FIG. 34 illustrates a first part of the procedure `uvDecodeWithPrediction` according to an example embodiment.

[0054] FIG. 35 illustrates a second part of the procedure `uvDecodeWithPrediction` according to an example embodiment.

[0055] FIG. 36 illustrates a procedure `predictUV` for decoding according to an example embodiment.

15 [0056] FIG. 37 illustrates a first part of the procedure `encodePerFaceAttributes` according to an example embodiment.

[0057] FIG. 38 illustrates a second part of the procedure `encodePerFaceAttributes` according to an example embodiment.

20 [0058] FIG. 39 illustrates a first part of the procedure `decodePerFaceAttributes` according to an example embodiment.

[0059] FIG. 40 illustrates a second part of the procedure `decodePerFaceAttributes` according to an example embodiment.

[0060] FIG. 41 illustrates a procedure `applyIndexDeduplicatePerFan` used for per fan deduplications according to some embodiments according to an example embodiment.

25 [0061] FIG. 42 illustrates code used in delta coding of a handles table in a `serialize` method according to an example embodiment.

[0062] FIG. 43 illustrates code used in delta coding of a handles table in an `unserialize` method according to an example embodiment.

30 [0063] FIG. 44 illustrates modification to a `serialize` method, particularly to portions related to duplicates, as implemented in some embodiments.

[0064] FIG. 45 illustrates modification to an `unserialize` method, particularly to portions related to duplicates, as implemented in some embodiments.

[0065] FIG. 46 illustrates modification to an unserialize method, particularly to portions related to arithmetic decoding of duplicates, as implemented in some embodiments.

[0066] FIG. 47 is a block diagram of an example of a system in which various aspects and embodiments are implemented.

5

DETAILED DESCRIPTION

Overview of mesh coding.

10 [0067] FIG. 1 is a schematic block diagram of a mesh encoding process in which some embodiments may be implemented. A source mesh model 302 is provided as an input mesh $M(i)$ to the mesh encoding process. The source mesh model 302 is associated with a source texture map 304 that is provided as an input texture map $A(i)$ to the encoding process. The input mesh is decimated at 306 to generate a base mesh $m(i)$ with a reduced number of vertices, and a UV atlas is generated for the base mesh at 308. The base mesh is quantized at 310 and encoded at 312, with the compressed base mesh data being multiplexed at 314 into a dynamic mesh bitstream. The compressed base mesh data is reconstructed at the encoder to generate reconstructed base mesh $m'(i)$ with a static mesh decoder 316. The reconstructed base mesh is subdivided at 318 by adding new vertices. A subdivision surface fitting process is performed at 320 by comparing the subdivided base mesh with the input mesh $M(i)$ to determine a set of displacements $d(i)$ that deform the vertices of the subdivided base mesh to correspond more closely to the surfaces defined by the input mesh $M(i)$. These displacements may be updated at 322 into updated displacements $d'(i)$ based on difference between the original base mesh $m(i)$ and the reconstructed base mesh $m'(i)$. These updated displacements are encoded using a wavelet transform 324 that generates wavelet coefficients $e'(i)$, which are quantized at 326 and packed at 328 into an image format. A time-varying series of images representing the wavelet coefficients may be encoded at 330 using conventional video encoding techniques, and the encoded video may be multiplexed at 314 with the data representing the compressed base mesh. At the encoder, the displacements are reconstructed from the encoded video through image unpacking 329, inverse quantization 331, and inverse wavelet transform 332 to generate a reconstructed set of displacements $d''(i)$. A reconstructed base mesh $M''(i)$ is obtained through inverse quantization at 334 of the reconstructed quantized base mesh $m'(i)$, and the reconstructed base mesh $M''(i)$ is subdivided at 336. A reconstructed deformed mesh $DM(i)$ is generated at 338 by applying the reconstructed set of displacements $d''(i)$ to the reconstructed base mesh $m'(i)$. The

20

25

30

35

reconstructed deformed mesh DM(i) is used as a destination mesh model 340 for the purpose of attribute transfer 341.

[0068] Using the reconstructed deformed mesh DM(i) (destination mesh model 340), the input mesh M(i) (source mesh model 302), and the input texture map A(i) (source texture map 304),
5 an attribute transfer process is performed to provide attribute values for a destination texture map A'(i) that is associated with the reconstructed deformed mesh DM(i). Pixels in the texture map A'(i) that are not associated with any triangle of the reconstructed deformed mesh DM(i) may be filled using a padding process 342. A color space conversion 344 may be performed, a time-varying series of texture maps A'(i) may be encoded using conventional video encoding
10 techniques 346, and the encoded video may be multiplexed at 314 into a bitstream 350 with the data representing the displacements and the compressed base mesh. Patch information 348 may also be multiplexed in the bitstream.

[0069] FIG. 2 illustrates an example of a mesh decoder. The mesh decoder demultiplexes different sub-bitstreams from, e.g. a V3C bitstream. The sub-bitstreams may include a sub-bitstream representing a static mesh, a sub-bitstream representing displacements, and one or
15 more sub-bitstreams representing attributes. In the illustration of FIG. 2, the displacements sub-bitstream is illustrated as being decoded with a video decoder, but as discussed in greater detail below, different decoder configurations may be used depending on parameters received in the bitstream. For example, the decoder may be configured to use an arithmetic decoder in place of a video decoder for the coding of the displacements. In addition, where video coding
20 is used, different decoder configurations may use different video codecs (such as AVC, HEVC, VVC, and the like).

Overview of example embodiments.

[0070] The present disclosure describes an implementation of the reverse version of edgebreaker coding, including adaptations included to make it work for all the sequences of
25 the V-Mesh CTC. In addition to encoding/decoding the topology of a mesh, the present disclosure also describes apparatus and methods for attribute prediction and arithmetic coding. This reverse version provides compression performance results on MPEC CTCs up to 6% better than the forward version with faster encoding/decoding runtimes.

[0071] Some embodiments use topology encoding and decoding techniques as described in Lewiner et al. (2004). In such embodiments, there is no need to handle virtual points and table restarts. This allows for simplification of several functional blocks of the system such as: attribute predictions, post-reindexing, alternative vertex traversals, adaptation of per face attributes extensions, and support for vertex deduplication.

[0072] The flow diagram of FIG. 3 provides an overview of an edgebreaker mesh codec according to some embodiments. The top row is the encoding line, and the bottom row is the decoding line. Example commented source code for the different functional blocks is provided in the present disclosure. Example modifications to the V3C syntax are further provided herein.

5 As illustrated in FIG. 3, an example edgebreaker encoder performs pre-processing 102 of a mesh, e.g. to accommodate non-manifold surfaces or to quantize attributes. At 104, the encoder performs connectivity coding using edgebreaker to generate a CLERS table representing the connectivity. At 106, the encoder performs prediction of attributes of the mesh. The prediction may proceed in a topology-based order determined by the edgebreaker
10 encoding. Alternatively, the prediction may proceed using a different order, such as a depth-first traversal or a vertex-degree-based traversal. At 108, the alternate traversal is performed. The encoder may signal in the bitstream information indicating which type of traversal is used for attribute prediction. The encoder may test different traversal types (e.g. a topology-based traversal type and one or more alternative traversal types) and encode the mesh using
15 whichever traversal type provides the best performance (e.g. the greatest compression). At 110, entropy coding is performed of the CLERS table, of other information (e.g. handle information) about the topology of the mesh, and/or of the attribute residuals. A bitstream including this entropy encoded information may be stored for later use or may be transmitted to an encoder without intermediate storage.

20 [0073] At 112, a decoder performs entropy decoding on a received bitstream, including entropy decoding as appropriate of a CLERS table, handle information, and/or attribute residual information. At 114, the decoder uses the CLERS table and related information to reconstruct the topology or connectivity of the mesh. At 116, the decoder performs attribute prediction and corrections using the appropriate traversal type corresponding to the traversal
25 type used by the encoder (which may be signaled in the bitstream). Where the traversal type is an alternative traversal type, the decoder performs the appropriate alternative traversal at 118. Where the traversal type is based on the topology of the mesh as reconstructed in 114, an alternative traversal is not needed. At 120, post-processing is performed, e.g. to recreate manifold structures, dequantize attributes, and perform post-indexing.

30 [0074] Example source code for an encoding high level procedure encode is depicted in FIG. 4. It performs the pre-processing, the connectivity coding, and the attributes prediction. The entropy coding and the encoding following the V3C syntax is performed afterward in method named `serialize`, relevant parts of which are described below.

[0075] Example code for a decoding high-level procedure decode is depicted in FIG. 5.
35 Following the parsing of the bitstream and the arithmetic decoding (that are executed by a method `unserialize` that is partially presented), it performs the decoding of the connectivity

(procedures `topoPrepass` and `spiraleReversi`), of the attributes and the optional post-processing.

[0076] In these two high level procedures, the addition and removal of virtual points (a.k.a. dummy points), used in some implementations to handle holes in the surface, are not needed anymore and hence are removed.

[0077] Example embodiments implement topology coding and decoding techniques as described in Lewiner et al. (2004). Example implementations are provided below for encoding and decoding.

[0078] The procedure `encodeTopology` (see FIG. 6) is the entry point for the topology encoding part. As shown in bold in FIG. 6, example embodiments store the information `ccStartCorners` and `processedCorners`. Those are used later to perform the traversals for attributes predictions and post reindex. Example embodiments also encode some UV coordinates for which the complete pass on positions has been performed previously. Some example embodiments thus encode all the attributes in successive ordered passes after the topology encoding. (The same order may be used for decoding.) An example order is as follows:

1. Encoding of topology.
2. Encoding of position attributes (defines main index table).
3. Optionally encoding of UV attributes using main index table,
4. Or optionally encoding of UV attributes using auxiliary index table.
5. Optional encoding of per face attributes.

[0079] Passes that follow the topology order use the table `processedCorners`. Otherwise, the order may be reconstructed using the table `ccStartCorners` (which stores the first corner of each connected component – CC) and a predefined traversal method, such as one or more of the vertex traversal methods described in European Patent Application No. 23306154.8.

[0080] The procedure `encodeTopology` first encodes all the connected components (CCs) with boundaries and then all the connected components without boundaries. For each connected component it invokes the procedure `compressComponent` (see FIG. 7) with the start corner as parameter. This procedure goes through all the triangles of the connected component using the edge breaker spiraling method and generates the CLERS symbols that are written at the end of the `oCLers` table of symbols. In this embodiment, the component start corners are kept in memory but will not need to be written in the bitstream. The procedure

encodeTopology differs from that of Lewiner et al. (2004) by storing the processed corners while moving over the triangles of the connected component.

[0081] For each connected component with boundary, the procedure encodeTopology first encodes the first boundary of the connected component (by calling procedure
5 writeBoundary(ov.n(c))) before invoking the procedure compressComponent that will encode the other boundaries of the connected component, if any, by calling the procedure writeBoundary(c) when encountered during triangles traversal one some 'S' cases.

[0082] In the procedure writeBoundary, shown in FIG. 8, differs from a corresponding procedure disclosed in Lewiner et al. (2004) in that, for each new boundary, a unique Id is
10 generated between -3 and -infinite. Each opposite corner of the boundary corner is marked with this Id. Note that -1 and -2 are reserved values for other purposes in this embodiment, this is why we start at -3.

[0083] As for the topologic handles, the holes (boundaries) are marked using handles (pairs of integers to reconnect faces). Boundaries source corners are stored as negative integers
15 (starting from -3). Some additional precautions are taken when storing the corners Ids into the handle tables. The checkHandle procedure of the present disclosure differs from that of Lewiner et al. (2004) as depicted by FIG. 9. In the new following statement:

20 if (br != 0 && br != 1 && br != -1)
br == -1 means right corner is not valid, br == 1 means right triangle is visited but not on 'S' symbol and br == 0 means right triangle is not yet visited. In each of those case we do not want to store in the handle table. The corresponding test is done for left corners.

[0084] The procedure writeVertex stores vertex attributes for further predictions. In example
25 embodiments, if the predictions option is off for attributes, we directly store those in the table for AC coding. Otherwise, we append the corner Id given as a parameter at the end the table writtenCorners that is used later when using topology traversal for the predictions of the attributes. In the case of alternative traversals, such as prediction degree or depth traversals, there is no need to push the Id and the writeVertex procedure does not perform anything.

30 [0085] The procedures topoPrepass (see FIG. 11) and spiraleReversi (FIG. 12) are two steps of the topology decoding part. In some embodiments, the procedure topoPrepass may be skipped (at the cost of additional data in the bitstream). It is used to prepare a final CLERS table containing special P symbols that end a connected component with no boundaries. Those symbols are implicitly recovered from the rest of the chain. Then, the spiraleReversi
35 goes through the chain in a single pass to decode the topology.

[0086] In example embodiments, the topoPrepass procedure is configured to manage the handles properly. Example embodiments further fill the table ccStartCorners of corner indices to keep track of the first corner of each connected component. This table is used later on for predictions using traversals others than the spiral one (e.g. a vertex degree traversal).

- 5 [0087] In example embodiments, the spiraleReversi method is configured to store the order of processing of the corners (table processedCorners) and of reading of the corners (table readCorners). Those two tables are used later on when doing successive traversals for attributes predictions using the spiral order (i.e. not the vertex degree). The readBoundary (FIG. 13) is also configured to update the readCorners table. The method closeStar
 10 operates as described in Lewiner et al. (2004). The method readVertex is configured to consume attributes only if predictions are off. Otherwise, the predictions are read and computed during some other passes later.

[0088] In an example embodiment, once the topology is decoded, some or all of the following steps are executed (see FIG. 5):

- 15 1. Decode attributes using main index table.
 2. Decode attributes using auxiliary index table if any.
 3. Decode per face attributes.
 4. Finalize the deduplication if needed.
 5. Dequantize the attributes of the corner table (CT) mesh if the option is on (off for v-
 20 dmc).
 6. Reindex the topology if option is on.
 7. Convert the decoded corner table back to a triangle set.

- [0089] An example procedure postReindex that performs the topology reindexing is presented by FIG. 16. Compared to a forward implementation, it is simplified because no
 25 dummy points have to be handled and there is no need to manage multiple table starts. Instead, only the startCorners are used. The corner table does not need to be finalized with proper (shifted) indices since they are already valid. The decoding of the attributes is performed by steps 1 to 3.

- [0090] **Per vertex attributes using main index table.** The encoding and decoding of
 30 attributes using a main index table is initiated by the procedure encodeAttributesWithMainIndexTable (FIG. 17), or decodeAttributesWithMainIndexTable (FIG. 26), as appropriate. Those procedures use the traversal method selection from the topology order (EB) or using vertex degree traversal

(DEGREE), although any other traversal such as DEPTH could alternatively be used. Note that in the EB case the traversal order is the one contained by the `writtenCorners` table that was populated during the topology traversal. For the DEGREE case, the traversal may be performed using a vertex degree traverser such as that described in EP 23306154.8 for each start corner of each connected component (the table of `ccStartCorners` generated during the topology stages).

[0091] Using the proper traversal table, the `encodeAttributeWithPrediction(traversal)` or `decodeAttributeWithPrediction(traversal)` procedure is invoked as appropriate. The attribute prediction method invokes the position predictor `posEncodeWithPrediction(c)` or `posDecodeWithPrediction(c)` for each corner of the traversal given as parameter. Once all the positions are predicted, it then optionally (if UV uses main index) performs the same work for the uv coordinates by invoking the `uvEncodeWithPrediction` or `uvDecodeWithPrediction` for each corner of the traversal given as a parameter. This operation would be performed as additional passes for any other attributes using main index table such as normal vectors, colors, other.

[0092] At the decoding, some traversal tables are scanned in reverse since the topology traversal algorithm is performed in reverse. Hence the parameter “reverse” of the `decodeAttributesWithPrediction` procedure that is set to true when using the EB traversal and set to false when using the DEGREE traversal.

[0093] Example embodiments may perform the process of managing the deduplicates in a manner different from that of TMv6. In TMv6, positions and attributes are processed in the same pass, whereas here we use separate passes, which may lead to better predictions of the attributes. We thus reorganize the structure of the algorithm and we store in a table named `predictCorner` a Boolean for each corner that is set by first pass (the position pass) and used by any other subsequent passes (e.g. the uv prediction, if using main index in our implementation) to skip the prediction for the vertices that are duplicated due to the non-manifold issue during the preprocessing (see FIG. 3).

[0094] Per vertex attributes using auxiliary index table. The encoding or decoding of attributes using an auxiliary index table is initiated by the procedure `encodeAttributesWithAuxiliaryIndexTable` (FIG. 19) or `decodeAttributesWithAuxiliaryIndexTable` (FIGs. 28-30) as appropriate.

[0095] The `encodeAttributesWithAuxiliaryIndexTable` or `decodeAttributesWithAuxiliaryIndexTable` procedure follows the same general structure as the corresponding procedure `encodeSeparateIndexAttributes` or `decodeSeparateIndexAttributes` from the forward solution implemented in TMv6. One

difference is that TMv6 invokes the procedure per component whereas example embodiments described herein perform the traversal in one call. Also, example embodiments no longer test for virtual points during the uv seams definition, but a statement is added that detects boundaries using the opposite table to skip uv seams definition. There is no need to define a seam if we are on a boundary, so we make an economy by not pushing into the isSeam table.

[0096] Once the seams are properly defined, example embodiments invoke the uvEncodeWithPrediction for each corner of the selected traversal order (EB, DEGREE, DEPTH, ...) procedure.

[0097] **Per vertex attributes predictions.** The predictions of vertex positions in example embodiments are performed by the procedure posEncodeWithPrediction (FIGs. 21-22) and posDecodeWithPrediction (FIGs. 32-33).

[0098] In TMv6, the predictions of a given corner involve turning around the vertex of this corner to search for multiple parallelograms, to compute the predictions. When a surrounding corner is detected as a dummy point it is skipped so as not to disturb the prediction. As seen in the topology section, example embodiments mark the boundaries instead of using dummy points. The predictions make use of this boundary. For each corner, the associated vertex is either connected to a full triangle fan or to a partial one if on boundary. In an example embodiment, the search of parallelograms for a given corner thus rotates around its associated vertex, starting from the corner to the right until it reaches a boundary or the corner again. It then turns in the other direction until it reaches another boundary or c.

[0099] If at least one parallelogram is found, it is used for the prediction (more than one can be used if available) and push the residual in the oVertices table, otherwise we use a delta coding with the next corner if already predicted, otherwise with the previous corner if already predicted. For arithmetic coding, the delta values would benefit to be stored in another table dedicated to deltas. In the current version those are also pushed into the oVertices table.

[0100] Example embodiments make use of another fallback (case 3) which is: if we are at a start, but we are near an external boundary where corners were already predicted we use it for delta prediction and push in oVertices. The same remark applies in that this provides a benefit for AC coding to be pushed into a table specific to delta residuals.

[0101] Example embodiments fall back to pushing a new start in global coordinates to the osVertices table. Normally this occurs only if c is the first corner of a new connected component.

[0102] The arithmetic coding of the tables oVertices (the predicted positions) and osVertices (the start positions) may be performed using the techniques as described in TMV6.0.

5 [0103] The posDecodeWithPrediction procedure may be used to reconstruct the position values from predictions.

[0104] The predictions of vertex uv coordinates are performed in some embodiments by the procedure uvEncodeWithPrediction (FIGs. 23-24), uvDecodeWithPrediction (FIGs. 34-35) which performs the search for valid predictors around the corner in the same manner as for positions and the two predictUV (FIG. 25, FIG. 36) procedures that actually compute the
10 min-stretch predictions.

[0105] The four methods take as a parameter a reference to the index table (tcIndices) to be used for the predictions. This permits use of the prediction mechanism for attributes that use the main index table and for attributes that use the auxiliary index table. The general algorithm is then simplified compared to the one of TMv6 that makes specific algorithmics for
15 each kind of index table.

[0106] In example embodiments, the prediction of UV coordinates includes checking if a min stretch is possible whatever the corner, before falling back to delta, then to boundary, then to a start in uv space global coordinates. This makes it possible to use UV predictions even for some corners that are a start regarding to the position prediction. This is not the case with the
20 forward version of TMv6 which systematically sets a start for first corner of a connected component, then delta for the two next corners, then min stretch predictions for the other corners.

[0107] **Per face attributes.** A technique for per face attributes coding of TMv6 is presented in European Patent Application No. 23305941.9. Example embodiments described herein are
25 simplified since they no longer need to handle virtual points. Example embodiments are also adapted to handle boundaries using the opposite table. The procedures encodePerFaceAttributes (FIGs. 37-38) and decodePerFaceAttributes (FIGs. 39-40) are presented with additions shown in boldface. As an example, when looking at the neighborhood for finding support for the predictions, example embodiments also test the
30 opposite table for negative values to check the existence of a boundary or not. Thanks to the coding of boundaries with negative integers in the corner table, the overall complexity is thus reduced.

[0108] **Arithmetic encoding and decoding.** The general schema for coding and decoding the different tables is the same as presented in TMv6.0. Example embodiments provide for a
35 modified coding of handles. In example embodiments, the handles table is used to code the

handles but also the boundaries using negative integer values for the first handle pair of values (see FIG. 11). The AC coding model from TMv6 is not as simple anymore since we would also need to code the sign. In some embodiments the handles are coded as delta values as seen in FIGs. 42 and 43.

- 5 **[0109]** In an example embodiment, a mesh coding header syntax includes a parameter indicating a mesh codec type, such as the parameter `mesh_codec_type`. The parameter may take on values with identifiers such as `CODEC_TYPE_FORWARD` for a forward codec or `CODEC_TYPE_REVERSE` for a reverse codec type.

- [0110]** In an example embodiment, a mesh position coding payload syntax (such as that of section I.8.3.7 of V-DMC) may be implemented as follows:

mesh_position_coding_payload() {		Descriptor
mesh_vertex_count		vu(v)
If (mesh_codec_type == CODEC_TYPE_REVERSE){	†	
mesh_triangle_count	†	vu(v)
}	†	
mesh_clers_count		vu(v)
mesh_cc_count		vu(v)
If (mesh_codec_type == CODEC_TYPE_FORWARD){	†	
mesh_virtual_vertex_count		vu(v)
}	†	
If (mesh_codec_type == CODEC_TYPE_REVERSE){	†	
mesh_position_start_count	†	
}	†	
If (mesh_codec_type == CODEC_TYPE_FORWARD){	†	
for(i=0; i<mesh_virtual_vertex_count; i++){		
mesh_virtual_index_delta[i]		vu(v)
}	†	
}		
mesh_cc_with_handles_count		vu(v)
NumHandles = 0		
If (mesh_codec_type == CODEC_TYPE_FORWARD){	†	
for(i=0; i<mesh_cc_with_handles_count; i++){		
mesh_handles_cc_offset[i]		vu(v)
mesh_handles_count[i]		vu(v)
NumHandles += mesh_handles_count[i]		
}		
}	†	
If (mesh_codec_type == CODEC_TYPE_REVERSE){	†	
NumHandles = mesh_cc_with_handles_count	†	
}	†	
for(i=0; i<NumHandles; i++){		
mesh_handle_index_first_delta[i]		vi(v)
mesh_handle_index_second_delta[i]		vi(v)
}		

If (mesh_codec_type == CODEC_TYPE_FORWARD){	†	
mesh_coded_handle_index_second_shift_size		vu(v)
for(i=0; i<NumHandles; i++){		
mesh_handle_index_second_shift[i]		ae(v)
}		
}	†	
length_alignment()		
mesh_coded_clers_symbols_size		vu(v)
prev[1] = 0		
prev[0] = 0		
Crun = 0		
for(i=0; i<mesh_clers_count; i++){		
mesh_clers_symbol[i]		ae(v)
prev[1] = prev[0]		
prev[0] = (mesh_clers_symbol[i] + 1) >> 1		
Crun = mesh_clers_symbol[i] ? 0 : Crun + 1		
}		
length_alignment()		
if (mesh_codec_type == CODEC_TYPE_FORWARD){	†	
NumPositionStart = mesh_cc_count	†	
}	†	
if (mesh_codec_type == CODEC_TYPE_REVERSE){	†	
NumPositionStart = mesh_position_start_count;	†	
}	†	
if(mesh_position_deduplicate_method == MESH_POSITION_DEDUP_DEFAULT) {	†	
mesh_position_deduplicate_information()	†	
}	†	
for(i=0; i< NumPositionStart; i++){		
for(j=0; j< 3; j++){		
mesh_position_start[i][j]		u(v)
}		
}		
length_alignment()		
mesh_coded_position_residuals_size		vu(v)
NumPredictedPositions = mesh_vertex_count - mesh_cc_count		
for(i=0; i<NumPredictedPositions; i++){		
for(j=0; j< 3; j++){		
mesh_position_residual[i][j]		ae(v)
}		
if (mesh_position_prediction_method == 0) {		
vi = vertex_index_in_cc(i)		
if (vi == 1) {		
k = position_start_index_in_cc_for_vertex(i)		
mesh_position_non_start[i] = mesh_position_start[k] +		
mesh_position_residual[i]		
} else if (vi == 2) {		
mesh_position_non_start[i] = mesh_position_non_start[i - 1] +		
mesh_position_residual[i]		

} else {	
predCount = 0	
predPos = (0, 0, 0)	
while (position_prediction_available(estG)) {	
if (preCount >	
mesh_position_prediction_max_parallelograms_minus1) {	
break;	
}	
predPos += estG	
predCount ++	
}	
if (predCount > 0) {	
predPos /= predCount	
} else {	
predPos = mesh_position_non_start[i - 1]	
}	
mesh_position_non_start[i] = predPos + mesh_position_residual[i]	
}	
}	
}	
length_alignment()	
}	

A mesh position coding payload may have semantics such as the following:

- **mesh_vertex_count** specifies the number of vertices in the decoded mesh.
5 (mesh_vertex_count may not include virtual points which are inserted during the topology reconstruction and removed in the subsequence mesh cleaning phase.
- **mesh_triangle_count** specifies the number of triangles in the decoded mesh.
- **mesh_clers_count** specifies the number of encoded CLERS symbols. (The first symbol of each per connected component sequence of symbols is not coded. After
10 topology reconstruction the number of triangles is (mesh_clers_count + mesh_cc_count). After mesh cleaning an unspecified number of triangles connected to virtual points are removed.)
- **mesh_cc_count** specifies the number of connected components comprised in the coded mesh. In case of mesh_codec_type=CODEC_TYPE_REVERSE,
15 **mesh_cc_count** specifies the number of connected components with boundaries comprised in the coded mesh (the number of components without boundaries is deduced from the CLERS parsing). (The number of absolute coded position can be less than mesh_cc_count if first positions of connected components are duplicates of previously encoded indices.)

- **mesh_position_start_count** specifies the number of start positions.
- **mesh_virtual_vertex_count** specifies the number of virtual vertices inserted to close holes in the reconstructed topology.
- 5 • **mesh_virtual_index_delta[i]** specifies the difference between the values of the i-th virtual point index and the (i – 1)-th virtual point index when i is greater than 0. When i equal to 0 mesh_virtual_index_delta[0] specifies the value of the index of the first virtual point.
- **mesh_cc_with_handles_count** specifies the number of connected components with non zero handle count.
- 10 • **mesh_handles_cc_offset[i]** specifies the difference between the values of the connected component index of the i-th connected component with non zero handle count and the (i – 1)-th connected component with non zero handle count when i is greater than 0. When i equal to 0, mesh_handles_cc_offset[0] specifies the connected component index of the first connected components with non zero handle
- 15 count.
- **mesh_handles_count[i]** specifies the number of handles comprised in the i-th connected component with non zero handle count.
- **mesh_handle_index_first_delta[i]** specifies the difference between the triangle index of the i-th handle first corner and the (i – 1)-th handle first corner when i is greater
- 20 than 0. When i equal to 0 mesh_handle_index_first_delta[0] specifies the triangle index of the first handle first corner.
- **mesh_handle_index_second_delta[i]** specifies the difference between the triangle index of the i-th handle second corner and the (i – 1)-th handle second corner when i is greater than 0. When i equal to 0 mesh_handle_index_second_delta[0] specifies
- 25 the triangle index of the first handle second corner.
- **mesh_coded_handle_index_second_shift_size** specifies the size in bytes of the arithmetically coded sequence of mesh_handle_index_second_shift values, including the final byte alignment.
- **mesh_handle_index_second_shift[i]** specifies the shift to apply when computing
- 30 the corner index associated with the i-th handle destination. (Handle indices are relative to a triangle/face index as related corner indices can be deduced implicitly. The corner index of the first handle is conforming to (3 * T + 0). The corner index of

the second handle f index is conforming to either (3 * T + 1) or (3 * T + 2).
mesh_handle_index_second_shift[i] is used to discriminate those two cases.)

- **mesh_coded_clers_symbols_size** specifies the size in bytes of the arithmetically coded sequence of CLERS symbols, including the final byte alignment. (Specifying mesh_coded_clers_symbols_size explicitly enables to pursue parsing and decode CLERS symbols in parallel.
- **mesh_clers_symbol[i]** specifies the value of the i-th CLERS symbol. The values are interpreted as specified by the following table:

mesh_clers_symbol[i]	0	1	3	5	7
i-th CLERS symbol	C	S	L	R	E

- **mesh_position_start[i][j]** specifies the value of the j-th component of the first position of the i-th connected component as an unsigned integer coded using mesh_position_bit_depth_minus1 + 1 bits.
- **mesh_coded_position_residuals_size** specifies the size in bytes of the arithmetically coded sequence of positions residuals, including the final byte alignment. (Specifying mesh_coded_position_residuals_size explicitly enables to pursue parsing and decode position prediction residuals in parallel.)
- **mesh_position_residual[i][j]** specifies the value of the prediction residual associated with the j-th component of the i-th predicted position.

[0111] Serializing/Un-serializing. In an example embodiment, the unserialize method is invoked after the syntax parsing and before the decode method is invoked. It finalizes the arithmetic decoding and set some other global variables that cannot be deduced during syntax parsing. Compared to the corresponding method from TMv6, the unserialize method according to an example embodiment has some or all of the following features:

- Code sections that are related to virtual points are removed as unnecessary.
- Parts related to deduplication in the serialize (FIG. 43) and unserialize (FIGs. 44-45) methods are modified.
- The nV and nUV global variables are computed as follows after the arithmetic decoding of iVertices, isVertices, iUVCoords and isUVCoords is performed as follows, with nV being the total number of decoded positions and nUV the total number of decoded uv coordinates:

```
nV = iVertices.size() + isVertices.size();  
const auto recUv = iUVCoords.size() + isUVCoords.size();
```

```
nUV = hasSeparateUvIndex ? recUv : (recUv ? nV : 0);
```

- The global variable NumPositionStart is set to **mesh_position_start_count** instead of **mesh_cc_count** since **mesh_cc_count** is not representing the full set of connected components, but only the ones with a boundary. The new **mesh_position_start_count** from the syntax is used to find the value of absolute positions.

[0112] Example embodiments may use a syntax largely in accordance with the Video-based dynamic mesh coding (V-DMC) working draft, although with some modifications. For example, in some embodiments, the syntax allows for signaling that the mesh is a reverse type. For example, the **mesh_codec_type** may be signaled as a named CODEC_TYPE_REVERSE. As shown in the syntax proposed above, in case of reverse **mesh_codec_type=**CODEC_TYPE_REVERSE, the parameters **mesh_triangle_count** and **mesh_position_start_count** are included in the syntax. In forward codecs, those parameters are deduced by other means, but they cannot be deduced in the same manner in the reverse version. The parameter **mesh_cc_count** may be signaled in the syntax for both forward and reverse codecs, but the meaning of that parameter may be different for those two cases as described above.

[0113] As is further shown above in the example syntax, in the case of reverse **mesh_codec_type=**CODEC_TYPE_REVERSE

- Information related to virtual points is removed.
- Information related to AC coding of handles is removed.
- The parameters **mesh_triangle_count** and **mesh_position_start_count** are signaled.

[0114] In reverse mode, mesh_attribute starts are stored in any case. This is because we cannot deduce those any more since uv predictor may have some predictions even for a start position. Hence, the uv start array does not have the same size as the position start array.

[0115] The presented software code is a (sometimes simplified) C++ code.

[0116] Mathematics:

- dvec2 are mathematical 2D vectors of c/c++ double values (64bits floating point)
- vec2 are mathematical 2D vectors of c/c++ float values (32bits floating point).
- std::sqrt returns mathematical square root of the parameter
- std::round returns mathematical rounding to nearest integer of the parameter

[0117] Data structures:

- `std::vector<float> name;` is an array of floats named `name`. `float` could be replaced another type such as `vec2`, then the following methods would all operate on `vec2` values instead of floats.
- 5 - `name[i]` dereferences the array `name` at index `i`.
- `name.push_back(v)` adds a new element of value `v` at the end of the array `name` by increasing its size of 1 element.
- `name.reserve(n)` reserves some memory for the vector `name` but its effective number of elements `name.size()` is not changed. Such a memory pre-allocation prevents
- 10 multiple memory reallocations on multiple calls to `name.push_back()`.
- `name.assign(n,v)` resizes the array `name` to contain `n` elements and set all the values to `v`. `v` is optional; default value is used if not provided.
- `name.resize(n, v)` resizes the array `name` to contain `n` elements and set the new values to `v`. Existing values are left unchanged. `v` is optional, default value is used if
- 15 not provided.

Corner table (named `ov` or `_ovTable`):

- `ov.t(c)` returns the triangle index of corner of index `c` if `c` is valid (in `[0,NB_CORNERS]`), -1 otherwise.
- `ov.n(c)` returns next corner index of corner of index `c` if `c` is valid (in
- 20 `[0,NB_CORNERS]`), -1 otherwise.
- `ov.p(c)` returns previous corner index of corner of index `c` if `c` is valid (in `[0,NB_CORNERS]`), -1 otherwise.
- `ov.o(c)` returns opposite corner index of corner of index `c` if `c` is valid (in `[0,NB_CORNERS]`), -1 otherwise.

25

[0118] FIG. 4 illustrates example source code for an encoding procedure according to an example embodiment.

[0119] FIG. 5 illustrates example source code for a decoding procedure according to an example embodiment.

- 30 **[0120]** FIG. 6 illustrates example source code for a topology encoding procedure according to an example embodiment. FIG. 6 illustrates a topology encoding procedure named `encodeTopology`. As compared to the procedure named `Compress` in Lewiner et al. (2004), the encoding procedure of FIG. 6 includes additional features such as the recording of the

start corner of the connected component and recording of the corner processing order. These features are illustrated in boldface in FIG. 6.

[0121] FIG. 7 illustrates the procedure `compressComponent`. As compared to the corresponding procedure in Lewiner et al. (2004), this procedure includes additional features
5 such as the recording of each corner in a processed corners table.

[0122] FIG. 8 illustrates the procedure `writeBoundary`, which goes through the boundary of `_b` (a corner given as parameter) and writes each vertex once in order. Features of particular interest are illustrated in boldface.

[0123] FIG. 9 illustrates the procedure `checkHandle` according to an example embodiment.
10 Features of particular interest are illustrated in boldface.

[0124] FIG. 10 illustrates the procedure `writeVertex` according to an example embodiment.

[0125] FIGs. 11-16 illustrate topology decoding procedures according to an example embodiment.

[0126] FIG. 11 illustrates the procedure `topoPrepass` according to an example embodiment.

15 [0127] FIG. 12 illustrates the procedure `spiraleReversi` according to an example embodiment.

[0128] FIG. 13 illustrates the procedure `readBoundary` according to an example embodiment.

[0129] FIG. 14 illustrates the procedure `closeStar` according to an example embodiment.

[0130] FIG. 15 illustrates the procedure `readVertex` according to an example embodiment.

20 [0131] FIG. 16 illustrates the procedure `postReindex` according to an example embodiment.

[0132] FIGs. 17-25 illustrate procedures related to attribute prediction for encoding according to an example embodiment.

[0133] FIG. 17 illustrates the procedure `encodeAttributesWithMainIndexTable` according to an example embodiment.

25 [0134] FIG. 18 illustrates the procedure `encodeAttributesWithPrediction` according to an example embodiment.

[0135] FIG. 19 illustrates the procedure `encodeAttributesWithAuxiliaryIndexTable` according to an example embodiment.

30 [0136] FIG. 20 illustrates the `AUX_UV_ENCODE` statement to be copied in FIG. 19 according to an example embodiment.

[0137] FIG. 21 illustrates a first part of the procedure `posEncodeWithPrediction` according to an example embodiment.

[0138] FIG. 22 illustrates a second part of the procedure `posEncodeWithPrediction` according to an example embodiment.

5 [0139] FIG. 23 illustrates a first part of the procedure `uvEncodeWithPrediction` according to an example embodiment.

[0140] FIG. 24 illustrates a second part of the procedure `uvEncodeWithPrediction` according to an example embodiment.

10 [0141] FIG. 25 illustrates a procedure `predictUV` for encoding according to an example embodiment.

[0142] FIGs. 26-36 illustrate attribute prediction procedures for decoding according to an example embodiment.

[0143] FIG. 26 illustrates a procedure `decodeAttributesWithMainIndexTable` according to an example embodiment.

15 [0144] FIG. 27 illustrates a procedure `decodeAttributesWithPredition` according to an example embodiment.

[0145] FIG. 28 illustrates a first part of a procedure `decodeAttributesWithAuxiliaryIndexTable` according to an example embodiment.

20 [0146] FIG. 29 illustrates a second part of the procedure `decodeAttributesWithAuxiliaryIndexTable` according to an example embodiment.

[0147] FIG. 30 illustrates a third part of the procedure `decodeAttributesWithAuxiliaryIndexTable` according to an example embodiment.

[0148] FIG. 31 illustrates an `AUX_UV_DECODE` statement to be copied in FIG. 30 according to an example embodiment.

25 [0149] FIG. 32 illustrates a first part of the procedure `posDecodeWithPredictions` according to an example embodiment.

[0150] FIG. 33 illustrates a second part of the procedure `posDecodeWithPredictions` according to an example embodiment.

30 [0151] FIG. 34 illustrates a first part of the procedure `uvDecodeWithPrediction` according to an example embodiment.

[0152] FIG. 35 illustrates a second part of the procedure `uvDecodeWithPrediction` according to an example embodiment.

[0153] FIG. 36 illustrates a procedure `predictUV` for decoding according to an example embodiment.

[0154] FIG. 37 illustrates a first part of the procedure `encodePerFaceAttributes` according to an example embodiment.

5 [0155] FIG. 38 illustrates a second part of the procedure `encodePerFaceAttributes` according to an example embodiment.

[0156] FIG. 39 illustrates a first part of the procedure `decodePerFaceAttributes` according to an example embodiment.

10 [0157] FIG. 40 illustrates a second part of the procedure `decodePerFaceAttributes` according to an example embodiment.

[0158] FIG. 41 illustrates a procedure `applyIndexDeduplicatePerFan` used for per fan deduplications according to some embodiments.

[0159] FIG. 42 illustrates code used in delta coding of a handles table in a `serialize` method according to an example embodiment.

15 [0160] FIG. 43 illustrates code used in delta coding of a handles table in an `unserialize` method according to an example embodiment.

20 [0161] FIG. 44 illustrates modification to a `serialize` method, particularly to portions related to duplicates, as implemented in some embodiments. Compared to a forward version, example embodiments of the present disclosure remove code shown in strikethrough and incorporate code shown in boldface.

[0162] FIG. 45 illustrates modification to an `unserialize` method, particularly to portions related to duplicates, as implemented in some embodiments. Compared to a forward version, example embodiments of the present disclosure remove code shown in strikethrough and incorporate code shown in boldface.

25 [0163] FIG. 46 illustrates modification to an `unserialize` method, particularly to portions related to arithmetic decoding of duplicates, as implemented in some embodiments. Compared to a forward version, example embodiments of the present disclosure remove code shown in strikethrough and incorporate code shown in boldface.

30 [0164] Example embodiments make it possible to use edgebreaker spiral reversi within the MPEG EB and V3C syntax of V-DMC.

[0165] Example embodiments make it possible to use efficient per-vertex predictions with edgebreaker spiral reversi and boundary management.

[0166] Example embodiments make it possible to use efficient facelid predictions with edgebreaker spiral reversi and boundary management.

[0167] Example embodiments of reverse encoding and decoding systems and methods as described herein are believed to have lower complexity implementations than corresponding forward implementations. Such reductions in complexity allow for a greater likelihood of hardware-based codec implementation.

[0168] Example embodiments have been found to lead to more efficient encoding (obtained sizes are globally similar but we manage to reach from 1% to 6% gain on C0 conditions (the one with highest mesh density to code) of V-DMC CTCs. Example embodiments have been tested on 290 models from among a large dataset and have been found to perform encoding and decoding about 1% faster than a corresponding forward codec. Example embodiments allow for the possible implementation of valence-driven encoding.

Example system hardware.

[0169] Example embodiments of encoders and/or decoders (collectively coders) configured to implement embodiments described herein may be implemented using systems such as the system of FIG. 47. FIG. 47 is a block diagram of an example of a system in which various aspects and embodiments are implemented. System 1000 can be embodied as a device including the various components described below and is configured to perform one or more of the aspects described in this document. Examples of such devices, include, but are not limited to, various electronic devices such as personal computers, laptop computers, smartphones, tablet computers, digital multimedia set top boxes, digital television receivers, personal video recording systems, connected home appliances, and servers. Elements of system 1000, singly or in combination, can be embodied in a single integrated circuit (IC), multiple ICs, and/or discrete components. For example, in at least one embodiment, the processing and encoder/decoder elements of system 1000 are distributed across multiple ICs and/or discrete components. In various embodiments, the system 1000 is communicatively coupled to one or more other systems, or other electronic devices, via, for example, a communications bus or through dedicated input and/or output ports. In various embodiments, the system 1000 is configured to implement one or more of the aspects described in this document.

[0170] The system 1000 includes at least one processor 1010 configured to execute instructions loaded therein for implementing, for example, the various aspects described in this document. Processor 1010 can include embedded memory, input output interface, and various other circuitries as known in the art. The system 1000 includes at least one memory 1020 (e.g., a volatile memory device, and/or a non-volatile memory device). System 1000

includes a storage device 1040, which can include non-volatile memory and/or volatile memory, including, but not limited to, Electrically Erasable Programmable Read-Only Memory (EEPROM), Read-Only Memory (ROM), Programmable Read-Only Memory (PROM), Random Access Memory (RAM), Dynamic Random Access Memory (DRAM), Static Random Access Memory (SRAM), flash, magnetic disk drive, and/or optical disk drive. The storage device 1040 can include an internal storage device, an attached storage device (including detachable and non-detachable storage devices), and/or a network accessible storage device, as non-limiting examples.

[0171] System 1000 includes an encoder/decoder module 1030 configured, for example, to process data to provide an encoded video or decoded video, and the encoder/decoder module 1030 can include its own processor and memory. The encoder/decoder module 1030 represents module(s) that can be included in a device to perform the encoding and/or decoding functions. As is known, a device can include one or both of the encoding and decoding modules. Additionally, encoder/decoder module 1030 can be implemented as a separate element of system 1000 or can be incorporated within processor 1010 as a combination of hardware and software as known to those skilled in the art.

[0172] Program code to be loaded onto processor 1010 or encoder/decoder 1030 to perform the various aspects described in this document can be stored in storage device 1040 and subsequently loaded onto memory 1020 for execution by processor 1010. In accordance with various embodiments, one or more of processor 1010, memory 1020, storage device 1040, and encoder/decoder module 1030 can store one or more of various items during the performance of the processes described in this document. Such stored items can include, but are not limited to, the input video, the decoded video or portions of the decoded video, the bitstream, matrices, variables, and intermediate or final results from the processing of equations, formulas, operations, and operational logic.

[0173] In some embodiments, memory inside of the processor 1010 and/or the encoder/decoder module 1030 is used to store instructions and to provide working memory for processing that is needed during encoding or decoding. In other embodiments, however, a memory external to the processing device (for example, the processing device can be either the processor 1010 or the encoder/decoder module 1030) is used for one or more of these functions. The external memory can be the memory 1020 and/or the storage device 1040, for example, a dynamic volatile memory and/or a non-volatile flash memory. In several embodiments, an external non-volatile flash memory is used to store the operating system of, for example, a television. In at least one embodiment, a fast external dynamic volatile memory such as a RAM is used as working memory for video coding and decoding operations, such as for MPEG-2 (MPEG refers to the Moving Picture Experts Group, MPEG-2 is also referred

to as ISO/IEC 13818, and 13818-1 is also known as H.222, and 13818-2 is also known as H.262), HEVC (HEVC refers to High Efficiency Video Coding, also known as H.265 and MPEG-H Part 2), or VVC (Versatile Video Coding, a new standard being developed by JVET, the Joint Video Experts Team).

- 5 **[0174]** The input to the elements of system 1000 can be provided through various input devices as indicated in block 1130. Such input devices include, but are not limited to, (i) a radio frequency (RF) portion that receives an RF signal transmitted, for example, over the air by a broadcaster, (ii) a Component (COMP) input terminal (or a set of COMP input terminals), (iii) a Universal Serial Bus (USB) input terminal, and/or (iv) a High Definition Multimedia
10 Interface (HDMI) input terminal. Other examples include composite video.

- [0175]** In various embodiments, the input devices of block 1130 have associated respective input processing elements as known in the art. For example, the RF portion can be associated with elements suitable for (i) selecting a desired frequency (also referred to as selecting a signal, or band-limiting a signal to a band of frequencies), (ii) downconverting the selected
15 signal, (iii) band-limiting again to a narrower band of frequencies to select (for example) a signal frequency band which can be referred to as a channel in certain embodiments, (iv) demodulating the downconverted and band-limited signal, (v) performing error correction, and (vi) demultiplexing to select the desired stream of data packets. The RF portion of various embodiments includes one or more elements to perform these functions, for example,
20 frequency selectors, signal selectors, band-limiters, channel selectors, filters, downconverters, demodulators, error correctors, and demultiplexers. The RF portion can include a tuner that performs various of these functions, including, for example, downconverting the received signal to a lower frequency (for example, an intermediate frequency or a near-baseband frequency) or to baseband. In one set-top box embodiment, the RF portion and its associated
25 input processing element receives an RF signal transmitted over a wired (for example, cable) medium, and performs frequency selection by filtering, downconverting, and filtering again to a desired frequency band. Various embodiments rearrange the order of the above-described (and other) elements, remove some of these elements, and/or add other elements performing similar or different functions. Adding elements can include inserting elements in between
30 existing elements, such as, for example, inserting amplifiers and an analog-to-digital converter. In various embodiments, the RF portion includes an antenna.

- [0176]** Additionally, the USB and/or HDMI terminals can include respective interface processors for connecting system 1000 to other electronic devices across USB and/or HDMI connections. It is to be understood that various aspects of input processing, for example,
35 Reed-Solomon error correction, can be implemented, for example, within a separate input processing IC or within processor 1010 as necessary. Similarly, aspects of USB or HDMI

interface processing can be implemented within separate interface ICs or within processor 1010 as necessary. The demodulated, error corrected, and demultiplexed stream is provided to various processing elements, including, for example, processor 1010, and encoder/decoder 1030 operating in combination with the memory and storage elements to process the datastream as necessary for presentation on an output device.

[0177] Various elements of system 1000 can be provided within an integrated housing. Within the integrated housing, the various elements can be interconnected and transmit data therebetween using suitable connection arrangement 1140, for example, an internal bus as known in the art, including the Inter-IC (I2C) bus, wiring, and printed circuit boards.

[0178] The system 1000 includes communication interface 1050 that enables communication with other devices via communication channel 1060. The communication interface 1050 can include, but is not limited to, a transceiver configured to transmit and to receive data over communication channel 1060. The communication interface 1050 can include, but is not limited to, a modem or network card and the communication channel 1060 can be implemented, for example, within a wired and/or a wireless medium.

[0179] Data is streamed, or otherwise provided, to the system 1000, in various embodiments, using a wireless network such as a Wi-Fi network, for example IEEE 802.11 (IEEE refers to the Institute of Electrical and Electronics Engineers). The Wi-Fi signal of these embodiments is received over the communications channel 1060 and the communications interface 1050 which are adapted for Wi-Fi communications. The communications channel 1060 of these embodiments is typically connected to an access point or router that provides access to external networks including the Internet for allowing streaming applications and other over-the-top communications. Other embodiments provide streamed data to the system 1000 using a set-top box that delivers the data over the HDMI connection of the input block 1130. Still other embodiments provide streamed data to the system 1000 using the RF connection of the input block 1130. As indicated above, various embodiments provide data in a non-streaming manner. Additionally, various embodiments use wireless networks other than Wi-Fi, for example a cellular network or a Bluetooth network.

[0180] The system 1000 can provide an output signal to various output devices, including a display 1100, speakers 1110, and other peripheral devices 1120. The display 1100 of various embodiments includes one or more of, for example, a touchscreen display, an organic light-emitting diode (OLED) display, a curved display, and/or a foldable display. The display 1100 can be for a television, a tablet, a laptop, a cell phone (mobile phone), or other device. The display 1100 can also be integrated with other components (for example, as in a smart phone), or separate (for example, an external monitor for a laptop). The other peripheral devices 1120

include, in various examples of embodiments, one or more of a stand-alone digital video disc (or digital versatile disc) (DVR, for both terms), a disk player, a stereo system, and/or a lighting system. Various embodiments use one or more peripheral devices 1120 that provide a function based on the output of the system 1000. For example, a disk player performs the function of playing the output of the system 1000.

[0181] In various embodiments, control signals are communicated between the system 1000 and the display 1100, speakers 1110, or other peripheral devices 1120 using signaling such as AV.Link, Consumer Electronics Control (CEC), or other communications protocols that enable device-to-device control with or without user intervention. The output devices can be communicatively coupled to system 1000 via dedicated connections through respective interfaces 1070, 1080, and 1090. Alternatively, the output devices can be connected to system 1000 using the communications channel 1060 via the communications interface 1050. The display 1100 and speakers 1110 can be integrated in a single unit with the other components of system 1000 in an electronic device such as, for example, a television. In various embodiments, the display interface 1070 includes a display driver, such as, for example, a timing controller (T Con) chip.

[0182] The display 1100 and speaker 1110 can alternatively be separate from one or more of the other components, for example, if the RF portion of input 1130 is part of a separate set-top box. In various embodiments in which the display 1100 and speakers 1110 are external components, the output signal can be provided via dedicated output connections, including, for example, HDMI ports, USB ports, or COMP outputs.

[0183] The embodiments can be carried out by computer software implemented by the processor 1010 or by hardware, or by a combination of hardware and software. As a non-limiting example, the embodiments can be implemented by one or more integrated circuits. The memory 1020 can be of any type appropriate to the technical environment and can be implemented using any appropriate data storage technology, such as optical memory devices, magnetic memory devices, semiconductor-based memory devices, fixed memory, and removable memory, as non-limiting examples. The processor 1010 can be of any type appropriate to the technical environment, and can encompass one or more of microprocessors, general purpose computers, special purpose computers, and processors based on a multi-core architecture, as non-limiting examples.

Additional embodiments.

[0184] A mesh encoding method according to an example embodiment comprises: encoding a connectivity of a triangular mesh in a bitstream using edgebreaker coding; signaling in the

bitstream information indicating a direction of the connectivity encoding; and in response to a determination that the information indicates reverse edgebreaker coding, signaling in the bitstream a parameter indicating a number of triangles in the triangular mesh.

5 [0185] In some embodiments, the method further includes signaling, separate from the parameter indicating a number of triangles in the triangular mesh, a parameter indicating a number of encoded CLERS symbols.

[0186] In some embodiments, the method further includes, in response to a determination that the information indicates reverse edgebreaker coding, signaling in the bitstream a parameter indicating a number of start positions in the mesh.

10 [0187] Some embodiments further include, in response to a determination that the information indicates reverse edgebreaker coding, signaling in the bitstream a parameter indicating a number of connected components with boundaries in the triangular mesh

[0188] A mesh decoding method according to some embodiments comprises: obtaining a bitstream representing a connectivity of a triangular mesh encoded using edgebreaker coding; reading from the bitstream information indicating a direction of the connectivity encoding; in response to a determination that the information indicates reverse edgebreaker coding, reading from the bitstream a parameter indicating a number of triangles in the triangular mesh; and using the parameter indicating a number of triangles in the triangular mesh, reconstructing the connectivity of the triangular mesh from the bitstream.

20 [0189] Some embodiments further include reading from the bitstream a parameter indicating a number of encoded CLERS symbols, which may be separate from the parameter indicating a number of triangles in the triangular mesh.

[0190] Some embodiments further include, in response to a determination that the information indicates reverse edgebreaker coding, reading from the bitstream a parameter indicating a number of start positions in the mesh.

[0191] Some embodiments further include, in response to a determination that the information indicates reverse edgebreaker coding, reading from the bitstream a parameter indicating a number of connected components with boundaries in the triangular mesh.

30 [0192] An apparatus according to some embodiments comprises one or more processors, the apparatus being configured to perform any of the methods described herein.

[0193] An apparatus according to some embodiments comprises at least one processor and a computer-readable medium storing instructions for performing any of the methods described herein.

[0194] A computer-readable medium according to some embodiments store instructions for performing any of the methods described herein.

[0195] A computer-readable medium according to some embodiments stores a mesh encoded according to any of the methods described herein.

- 5 [0196] A signal according to some embodiments conveys a mesh encoded according to any of the methods described herein.

[0197] This disclosure describes a variety of aspects, including tools, features, embodiments, models, approaches, etc. Many of these aspects are described with specificity and, at least to show the individual characteristics, are often described in a manner that may sound limiting.

- 10 However, this is for purposes of clarity in description, and does not limit the disclosure or scope of those aspects. Indeed, all of the different aspects can be combined and interchanged to provide further aspects. Moreover, the aspects can be combined and interchanged with aspects described in earlier filings as well.

- [0198] The aspects described and contemplated in this disclosure can be implemented in many different forms. While some embodiments are illustrated specifically, other embodiments are contemplated, and the discussion of particular embodiments does not limit the breadth of the implementations. At least one of the aspects generally relates to mesh encoding and decoding, and at least one other aspect generally relates to transmitting a bitstream generated or encoded. These and other aspects can be implemented as a method, an apparatus, a computer readable storage medium having stored thereon instructions for encoding or decoding mesh data according to any of the methods described, and/or a computer readable storage medium having stored thereon a bitstream generated according to any of the methods described.
- 15
20

- [0199] Various methods are described herein, and each of the methods comprises one or more steps or actions for achieving the described method. Unless a specific order of steps or actions is required for proper operation of the method, the order and/or use of specific steps and/or actions may be modified or combined. Additionally, terms such as “first”, “second”, etc. may be used in various embodiments to modify an element, component, step, operation, etc., such as, for example, a “first decoding” and a “second decoding”. Use of such terms does not imply an ordering to the modified operations unless specifically required. So, in this example, the first decoding need not be performed before the second decoding, and may occur, for example, before, during, or in an overlapping time period with the second decoding.
- 25
30

- [0200] Various numeric values may be used in the present disclosure, for example. The specific values are for example purposes and the aspects described are not limited to these specific values.
- 35

[0201] Embodiments described herein may be carried out by computer software implemented by a processor or other hardware, or by a combination of hardware and software. As a non-limiting example, the embodiments can be implemented by one or more integrated circuits. The processor can be of any type appropriate to the technical environment and can encompass one or more of microprocessors, general purpose computers, special purpose computers, and processors based on a multi-core architecture, as non-limiting examples.

[0202] When a figure is presented as a flow diagram, it should be understood that it also provides a block diagram of a corresponding apparatus. Similarly, when a figure is presented as a block diagram, it should be understood that it also provides a flow diagram of a corresponding method/process.

[0203] The implementations and aspects described herein can be implemented in, for example, a method or a process, an apparatus, a software program, a data stream, or a signal. Even if only discussed in the context of a single form of implementation (for example, discussed only as a method), the implementation of features discussed can also be implemented in other forms (for example, an apparatus or program). An apparatus can be implemented in, for example, appropriate hardware, software, and firmware. The methods can be implemented in, for example, a processor, which refers to processing devices in general, including, for example, a computer, a microprocessor, an integrated circuit, or a programmable logic device. Processors also include communication devices, such as, for example, computers, cell phones, portable/personal digital assistants ("PDAs"), and other devices that facilitate communication of information between end-users.

[0204] Reference to "one embodiment" or "an embodiment" or "one implementation" or "an implementation", as well as other variations thereof, means that a particular feature, structure, characteristic, and so forth described in connection with the embodiment is included in at least one embodiment. Thus, the appearances of the phrase "in one embodiment" or "in an embodiment" or "in one implementation" or "in an implementation", as well any other variations, appearing in various places throughout this disclosure are not necessarily all referring to the same embodiment.

[0205] Additionally, this disclosure may refer to "determining" various pieces of information. Determining the information can include one or more of, for example, estimating the information, calculating the information, predicting the information, or retrieving the information from memory.

[0206] Further, this disclosure may refer to "accessing" various pieces of information. Accessing the information can include one or more of, for example, receiving the information, retrieving the information (for example, from memory), storing the information, moving the

information, copying the information, calculating the information, determining the information, predicting the information, or estimating the information.

[0207] Additionally, this disclosure may refer to “receiving” various pieces of information. Receiving is, as with “accessing”, intended to be a broad term. Receiving the information can include one or more of, for example, accessing the information, or retrieving the information (for example, from memory). Further, “receiving” is typically involved, in one way or another, during operations such as, for example, storing the information, processing the information, transmitting the information, moving the information, copying the information, erasing the information, calculating the information, determining the information, predicting the information, or estimating the information.

[0208] It is to be appreciated that the use of any of the following “/”, “and/or”, and “at least one of”, for example, in the cases of “A/B”, “A and/or B” and “at least one of A and B”, is intended to encompass the selection of the first listed option (A) only, or the selection of the second listed option (B) only, or the selection of both options (A and B). As a further example, in the cases of “A, B, and/or C” and “at least one of A, B, and C”, such phrasing is intended to encompass the selection of the first listed option (A) only, or the selection of the second listed option (B) only, or the selection of the third listed option (C) only, or the selection of the first and the second listed options (A and B) only, or the selection of the first and third listed options (A and C) only, or the selection of the second and third listed options (B and C) only, or the selection of all three options (A and B and C). This may be extended for as many items as are listed.

[0209] Also, as used herein, the word “signal” refers to, among other things, indicating something to a corresponding decoder. For example, in certain embodiments the encoder signals a particular one of a plurality of parameters for region-based filter parameter selection for de-artifact filtering. In this way, in an embodiment the same parameter is used at both the encoder side and the decoder side. Thus, for example, an encoder can transmit (explicit signaling) a particular parameter to the decoder so that the decoder can use the same particular parameter. Conversely, if the decoder already has the particular parameter as well as others, then signaling can be used without transmitting (implicit signaling) to simply allow the decoder to know and select the particular parameter. By avoiding transmission of any actual functions, a bit savings is realized in various embodiments. It is to be appreciated that signaling can be accomplished in a variety of ways. For example, one or more syntax elements, flags, and so forth are used to signal information to a corresponding decoder in various embodiments. While the preceding relates to the verb form of the word “signal”, the word “signal” can also be used herein as a noun.

[0210] Implementations can produce a variety of signals formatted to carry information that can be, for example, stored or transmitted. The information can include, for example, instructions for performing a method, or data produced by one of the described implementations. For example, a signal can be formatted to carry the bitstream of a described
5 embodiment. Such a signal can be formatted, for example, as an electromagnetic wave (for example, using a radio frequency portion of spectrum) or as a baseband signal. The formatting can include, for example, encoding a data stream and modulating a carrier with the encoded data stream. The information that the signal carries can be, for example, analog or digital information. The signal can be transmitted over a variety of different wired or wireless links,
10 as is known. The signal can be stored on a processor-readable medium.

[0211] We describe a number of embodiments. Features of these embodiments can be provided alone or in any combination, across various claim categories and types. Further, embodiments can include one or more of the following features, devices, or aspects, alone or in any combination, across various claim categories and types:

15 [0212] A bitstream or signal that includes one or more of the described syntax elements, or variations thereof.

[0213] A bitstream or signal that includes syntax conveying information generated according to any of the embodiments described.

20 [0214] Creating and/or transmitting and/or receiving and/or decoding a bitstream or signal that includes one or more of the described syntax elements, or variations thereof.

[0215] Creating and/or transmitting and/or receiving and/or decoding according to any of the embodiments described.

[0216] A method, process, apparatus, medium storing instructions, medium storing data, or signal according to any of the embodiments described.

25 [0217] Note that various hardware elements of one or more of the described embodiments are referred to as “modules” that carry out (i.e., perform, execute, and the like) various functions that are described herein in connection with the respective modules. As used herein, a module includes hardware (e.g., one or more processors, one or more microprocessors, one or more microcontrollers, one or more microchips, one or more application-specific integrated
30 circuits (ASICs), one or more field programmable gate arrays (FPGAs), one or more memory devices) deemed suitable for a given implementation. Each described module may also include instructions executable for carrying out the one or more functions described as being carried out by the respective module, and it is noted that those instructions could take the form of or include hardware (i.e., hardwired) instructions, firmware instructions, software

instructions, and/or the like, and may be stored in any suitable non-transitory computer-readable medium or media, such as commonly referred to as RAM, ROM, etc.

[0218] Although features and elements are described above in particular combinations, each feature or element can be used alone or in any combination with the other features and elements. In addition, the methods described herein may be implemented in a computer program, software, or firmware incorporated in a computer-readable medium for execution by a computer or processor. Examples of computer-readable storage media include, but are not limited to, a read only memory (ROM), a random access memory (RAM), a register, cache memory, semiconductor memory devices, magnetic media such as internal hard disks and removable disks, magneto-optical media, and optical media such as CD-ROM disks, and digital versatile disks (DVDs). A processor in association with software may be used to implement a radio frequency transceiver for use in a WTRU, UE, terminal, base station, RNC, or any host computer.

CLAIMS

1. A mesh encoding method comprising:
- 5 encoding a connectivity of a triangular mesh in a bitstream using edgebreaker coding;
 signaling in the bitstream information indicating a direction of the connectivity encoding;
and
 signaling in the bitstream a parameter indicating a number of triangles in the triangular
mesh.
- 10
2. A mesh encoding apparatus comprising one or more processors configured to perform at
least:
- encoding a connectivity of a triangular mesh in a bitstream using edgebreaker coding;
 signaling in the bitstream information indicating a direction of the connectivity encoding;
- 15 and
 signaling in the bitstream a parameter indicating a number of triangles in the triangular
mesh.
3. The method of claim 1 or the apparatus of claim 2, further comprising signaling, separate
20 from the parameter indicating a number of triangles in the triangular mesh, a parameter
indicating a number of encoded CLERS symbols.
4. The method of claim 1, or claim 3 as it depends from claim 1, or the apparatus of claim 2,
or claim 3 as it depends from claim 2, further comprising signaling in the bitstream a parameter
25 indicating a number of start positions in the mesh.
5. The method of claim 1, or claims 3-4 as they depend from claim 1, or the apparatus of claim
2, or claims 3-4 as they depend from claim 2, further comprising signaling in the bitstream a
parameter indicating a number of connected components with boundaries in the triangular
30 mesh.
6. The method of claim 1, or claims 3-5 as they depend from claim 1, or the apparatus of claim
2, or claims 3-5 as they depend from claim 2, wherein information indicating a number of virtual
vertices is not signaled in the bitstream.
- 35

7. The method of claim 1, or claims 3-6 as they depend from claim 1, or the apparatus of claim 2, or claims 3-6 as they depend from claim 2, wherein information identifying indices of virtual vertices is not signaled in the bitstream.
- 5 8. A mesh decoding method comprising:
- obtaining a bitstream representing a connectivity of a triangular mesh encoded using edgebreaker coding;
 - reading from the bitstream information indicating a direction of the connectivity encoding;
 - 10 reading from the bitstream a parameter indicating a number of triangles in the triangular mesh; and
 - using the parameter indicating the number of triangles in the triangular mesh, reconstructing the connectivity of the triangular mesh from the bitstream.
- 15 9. A mesh decoding apparatus comprising one or more processors configured to perform at least:
- obtaining a bitstream representing a connectivity of a triangular mesh encoded using edgebreaker coding;
 - reading from the bitstream information indicating a direction of the connectivity
 - 20 encoding;
 - reading from the bitstream a parameter indicating a number of triangles in the triangular mesh; and
 - using the parameter indicating the number of triangles in the triangular mesh, reconstructing the connectivity of the triangular mesh from the bitstream.
- 25 10. The method of claim 8 or the apparatus of claim 9, further comprising reading from the bitstream a parameter indicating a number of encoded CLERS symbols.
11. The method of claim 8, or claim 10 as it depends from claim 8, or the apparatus of claim 30 9, or claim 10 as it depends from claim 9, further comprising reading from the bitstream a parameter indicating a number of start positions in the mesh.
12. The method of claim 8, or claims 10-11 as they depend from claim 8, or the apparatus of claim 9, or claims 10-11 as they depend from claim 9, further comprising reading from the 35 bitstream a parameter indicating a number of connected components with boundaries in the triangular mesh.

13. The method of claim 8, or claims 10-12 as they depend from claim 8, or the apparatus of claim 9, or claims 10-12 as they depend from claim 9, wherein information indicating a number of virtual vertices is not read from the bitstream.

- 5 14. The method of claim 8, or claims 10-13 as they depend from claim 8, or the apparatus of claim 9, or claims 10-13 as they depend from claim 9, wherein information identifying indices of virtual vertices is not read from the bitstream.

15. A signal comprising:

- 10 information encoding a connectivity of a triangular mesh in a bitstream using edgebreaker coding;
information indicating a direction of the connectivity encoding; and
a parameter indicating a number of triangles in the triangular mesh.

15