



# Technologies for 3D mesh compression: A survey

Jingliang Peng<sup>a</sup>, Chang-Su Kim<sup>b,\*</sup>, C.-C. Jay Kuo<sup>a</sup>

<sup>a</sup> *Integrated Media Systems Center, Department of Electrical Engineering,  
University of Southern California, Los Angeles, CA 90089-2564, USA*

<sup>b</sup> *Department of Information Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong*

Received 1 February 2004; accepted 5 March 2005

Available online 16 April 2005

---

## Abstract

Three-dimensional (3D) meshes have been widely used in graphic applications for the representation of 3D objects. They often require a huge amount of data for storage and/or transmission in the raw data format. Since most applications demand compact storage, fast transmission, and efficient processing of 3D meshes, many algorithms have been proposed to compress 3D meshes efficiently since early 1990s. In this survey paper, we examine 3D mesh compression technologies developed over the last decade, with the main focus on triangular mesh compression technologies. In this effort, we classify various algorithms into classes, describe main ideas behind each class, and compare the advantages and shortcomings of the algorithms in each class. Finally, we address some trends in the 3D mesh compression technology development.

© 2005 Elsevier Inc. All rights reserved.

*Keywords:* 3D mesh compression; Single-rate mesh coding; Progressive mesh coding; MPEG-4

---

## 1. Introduction

Graphics data are more and more widely used in various applications, including video gaming, engineering design, architectural walkthrough, virtual reality, e-com-

---

\* Corresponding author.

*E-mail addresses:* [jingliap@usc.edu](mailto:jingliap@usc.edu) (J. Peng), [cskim@ieee.org](mailto:cskim@ieee.org) (C.-S. Kim), [cckuo@siipi.usc.edu](mailto:cckuo@siipi.usc.edu) (C.-C. Jay Kuo).

merce, and scientific visualization. Among various representation tools, triangular meshes provide an effective means to represent 3D mesh models. Typically, connectivity, geometry, and property data are used to represent a 3D polygonal mesh. Connectivity data describe the adjacency relationship between vertices; geometry data specify vertex locations; and property data specify several attributes such as the normal vector, material reflectance, and texture coordinates. Geometry and property data are attached to vertices in many cases. Thus, they are often called vertex data, and most 3D triangular mesh compression algorithms handle geometry and property data in a similar way. Therefore, we concentrate on the compression of connectivity and geometry data in this survey.

To achieve a high level of realism, complex models are required, and they are obtained from various sources such as modelling software and 3D scanning. They usually demand a huge amount of storage space and/or transmission bandwidth in the raw data format. As the number and the complexity of existing 3D meshes increase explosively, higher resource demands are placed on storage space, computing power, and network bandwidth. Among these resources, the network bandwidth is the most severe bottleneck in network-based graphic applications that demand real-time interactivity. Thus, it is essential to compress graphics data efficiently. This research area has received a lot of attention since early 1990s, and there has been a significant amount of progress along this direction over the last decade.

Early research on 3D mesh compression focused on single-rate compression techniques to save the bandwidth between CPU and the graphics card. In a single-rate 3D mesh compression algorithm, all connectivity and geometry data are compressed and decompressed as a whole. The graphics card cannot render the original mesh until the entire bit stream has been wholly received. Later, with the popularity of the Internet, the progressive compression and transmission has been intensively researched. When progressively compressed and transmitted, a 3D mesh can be reconstructed continuously from coarse to fine levels of detail (LODs) by the decoder while the bit stream is being received. Moreover, progressive compression can enhance the interaction capability, since the transmission can be stopped whenever an user finds out that the mesh being downloaded is not what he/she wants or the resolution is already good enough for his/her purposes.

Three-dimensional mesh compression is so important that it has been incorporated into several international standards. VRML [1] established a standard for transmitting 3D models across the Internet. Originally, a 3D mesh was represented in ASCII format without compression in VRML. For efficient transmission, Taubin et al. developed a compressed binary format for VRML [2] based on the topological surgery algorithm [3], which easily achieved a compression ratio of 50:1 over the VRML ASCII format. MPEG-4 [4], which is an ISO/IEC multimedia standard developed by the Moving Picture Experts Group for digital television, interactive graphics, and interactive multimedia applications, also includes three-dimensional mesh coding (3DMC) algorithm to encode graphics data. The 3DMC algorithm is also based on the topological surgery algorithm, which is basically a single-rate coder for manifold triangular meshes. Furthermore, MPEG-4 3DMC incorporates

progressive 3D mesh compression, non-manifold 3D mesh encoding, error resiliency, and quality scalability as optional modes.

In this survey, we intend to review various 3D mesh compression technologies with the main focus on triangular mesh compression. It is worthwhile to point out that there were several survey papers on this subject. Rossignac [5] briefly summarized prior schemes on vertex data compression and connectivity data compression. Taubin [6] gave a survey on various mesh compression schemes. Although the two schemes in the MPEG-4 standard (i.e., topological surgery and progressive forest split) were described in detail in [6], the review of other schemes was relatively sketchy. Shikhare [7] classified and described mesh compression schemes. However, this work did not treat progressive schemes with enough depth. Gotsman et al. [8] gave a tutorial on techniques for mesh simplification, connectivity compression, and geometry compression. This tutorial gave a detailed treatment on mesh simplification and geometry compression. However, its review on connectivity coding focused mostly on single-rate region-growing schemes. Recently, Alliez and Gotsman [9] surveyed techniques for both single-rate and progressive compression of 3D meshes. This survey gave a high-level algorithm classification, but focused only on static polygonal 3D mesh compression. Compared with previous survey papers, this work has attempted to achieve the following three objectives.

- Comprehensive and up-to-date coverage. This survey covers both single-rate and progressive mesh compression schemes. It covers not only techniques for triangular mesh compression, but also those for polygonal mesh compression, volume mesh compression, isosurface compression, and animated-mesh compression.
- In-depth classification and explanation. This survey tries to make more detailed classification and explanation of different algorithms.
- Analysis and comparison of coding performance and complexity. Coding efficiency is compared between different schemes to help practical engineers in the selection of schemes based on application requirements.

The rest of this paper is organized as follows. Section 2 provides a review of the background and introduces some definitions necessary to understand 3D mesh compression techniques. Sections 3 and 4 survey single-rate and progressive 3D mesh compression algorithms, respectively. Section 5 discusses new trends in 3D mesh compression research. Finally, concluding remarks are given in Section 6.

## 2. Background and basic concepts

Several definitions and concepts needed to understand 3D mesh compression algorithms are presented in this section. More rigorous definitions can be found in [10–12].

We say that two objects  $A$  and  $B$  are homeomorphic, if  $A$  can be stretched or bent without tearing to  $B$ . A 3D mesh is called a manifold if its every point has a neighborhood homeomorphic to an open disk or a half disk. In a manifold, the boundary

consists of the points that have no neighborhoods homeomorphic to an open disk but have neighborhoods homeomorphic to a half disk. In 3D mesh compression, a manifold with boundary is often pre-converted into a manifold without boundary by adding a dummy vertex to each boundary loop and then connecting the dummy vertex to every vertex on the boundary loop. Fig. 1A is a manifold mesh, while Figs. 1B and C are non-manifold meshes. Fig. 1B is non-manifold since each point on the edge  $(v_1, v_2)$  has no neighborhood that is homeomorphic to an open disk or a half disk. Similarly, the vertex  $v_1$  in Fig. 1C has no neighborhood homeomorphic to an open disk or a half disk.

The orientation of a polygon can be specified by the ordering of its bounding vertices. The orientations of two adjacent polygons are called compatible if they impose opposite directions on their common edges. A 3D mesh is said to be orientable if there exists an arrangement of polygon orientations such that each pair of adjacent polygons are compatible. Figs. 1A and C are orientable with the compatible orientations marked by arrows. In contrast, Fig. 1B is not orientable, since three polygons share the same edge  $(v_1, v_2)$ . Note that, after we make polygon B and C compatible, it is impossible to find an orientation of polygon A such that A is compatible with both B and C.

The genus of a connected orientable manifold without boundary is defined as the number of handles. For example, there is no handle in a sphere, one handle in a torus, and two handles in an eight-shaped surface as shown in Fig. 2. Thus, their genera are 0, 1, and 2, respectively. A mesh homeomorphic to a sphere is called a simple mesh. For a connected orientable manifold without boundary, Euler’s formula is given by

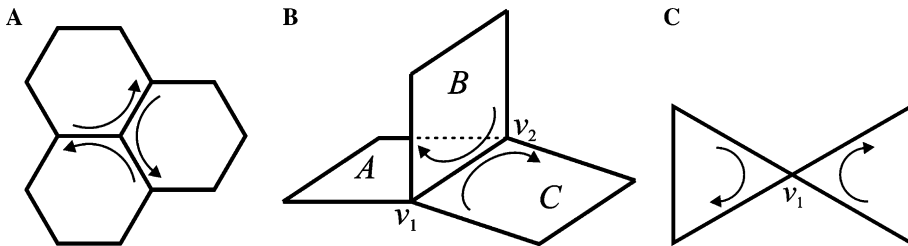


Fig. 1. Examples of (A) an orientable manifold mesh, (B) a non-orientable non-manifold mesh, and (C) an orientable non-manifold mesh.

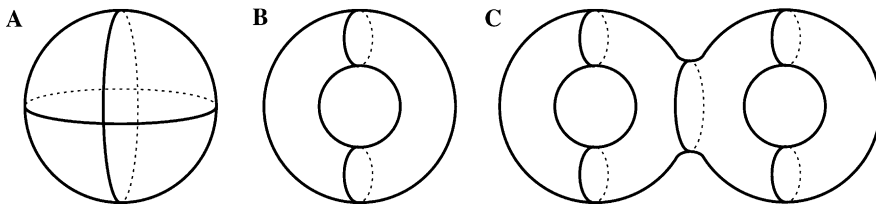


Fig. 2. (A) The sphere, (B) the torus, and (C) the eight-shaped mesh.

$$v - e + f = 2 - 2g, \quad (1)$$

where  $v$ ,  $e$ , and  $f$  are, respectively, the number of vertices, edges, and faces in the manifold, and  $g$  is the genus of the manifold.

Suppose that a triangular manifold mesh contains a sufficiently large number of edges and triangles, and that the ratio of the number of boundary edges to the number of non-boundary edges is negligible. Then, we can approximate the number of edges by

$$e \simeq 3f/2, \quad (2)$$

since an edge is shared by two triangles in general. Substituting (2) into (1), we have  $v \simeq f/2 + 2 - 2g$ . Since  $f/2$  is much larger than  $2 - 2g$ , we get

$$v \simeq f/2. \quad (3)$$

In other words, a typical triangular mesh has twice as many triangles as vertices.

Also, from (2) and (3), we have an approximate relation

$$e \simeq 3v. \quad (4)$$

The valence (or degree) of a vertex is the number of edges incident on that vertex. It can be shown that the sum of valences is twice the number of edges [12]. Thus, we have

$$\sum \text{valence} = 2e \simeq 6v. \quad (5)$$

Thus, in a typical triangular mesh, the average vertex valence is 6.

When reporting the compression performance, some papers employ the measure of bits per triangle (bpt) while others use bits per vertex (bpv). For consistency, we adopt the bpv measure exclusively, and convert the bpt metric to the bpv metric by assuming that a mesh has twice as many triangles as vertices.

### 3. Single-rate compression

A typical mesh compression algorithm encodes connectivity data and geometry data separately. Most early work focused on the connectivity coding. Then, the coding order of geometry data is determined by the underlying connectivity coding. However, since geometry data demand more bits than topology data, several methods have been proposed recently for efficient compression of geometry data without reference to topology data.

#### 3.1. Connectivity coding

We classify existing single-rate connectivity compression algorithms into six classes: the indexed face set, the triangle strip, the spanning tree, the layered decomposition, the valence-driven approach, and the triangle conquest. They are described in detail below.

### 3.1.1. Indexed face set

In the VRML ASCII format [1], a triangular mesh is represented with an indexed face set that consists of a coordinate array and a face array. The coordinate array lists the coordinates of all vertices, and the face array lists each face by indexing its three vertices in the coordinate array. For instance, Fig. 3 shows a mesh and its face array.

If there are  $v$  vertices in a mesh, the index of each vertex requires  $\log_2 v$  bits. Therefore, a triangular face needs  $3 \log_2 v$  bits for its connectivity information. Since there are about twice triangles as many as vertices in a typical triangular mesh, the connectivity information costs about  $6 \log_2 v$  bpv in the indexed face set method. This method provides a straightforward way for the triangular mesh representation. There is actually no compression involved in this method, but we still list it here to provide a basis of comparison for the following compression schemes.

In this method, each vertex is indexed several times by all its adjacent triangles. Repeated vertex references degrade the efficiency of connectivity coding. In other words, a good connectivity coding scheme should reduce the number of repeated vertex references. This observation leads to the triangle strip method.

### 3.1.2. Triangle strip

The triangle strip method attempts to divide a 3D mesh into long strips of triangles, and then encode these strips. The primary purpose of this method is to reduce the amount of data transmission between CPU and the graphic card, since triangle strips are well supported by most graphic cards. Although this scheme demands less storage space and transmission bandwidth than the indexed face set representation, it is still not very efficient for the compression purpose.

Fig. 4A shows a triangle strip, where each vertex is combined with the previous two vertices in a vertex sequence to form a new triangle. Fig. 4B shows a triangle fan, where each vertex after the first two forms a new triangle with the previous vertex and the first vertex. Fig. 4C shows a generalized triangle strip that is a mixture of triangle strips and triangle fans. Note that, in a generalized triangle strip, a new triangle is introduced by each vertex after the first two in the vertex sequence. However, in an indexed face set, a new triangle is introduced by three vertices. Therefore, the generalized triangle strip provides a more compact representation than the indexed face set, especially when the strip length is long. In a rather long generalized triangle strip, the ratio of the number of triangles to the number of vertices is very close to 1, meaning that a triangle can be represented by almost exactly 1 vertex index.

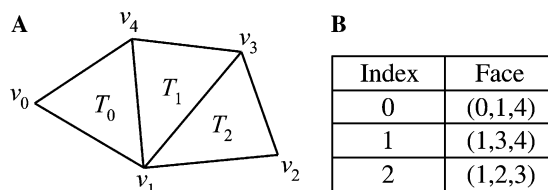


Fig. 3. The indexed face set representation of a mesh: (A) a mesh example and (B) its face array.

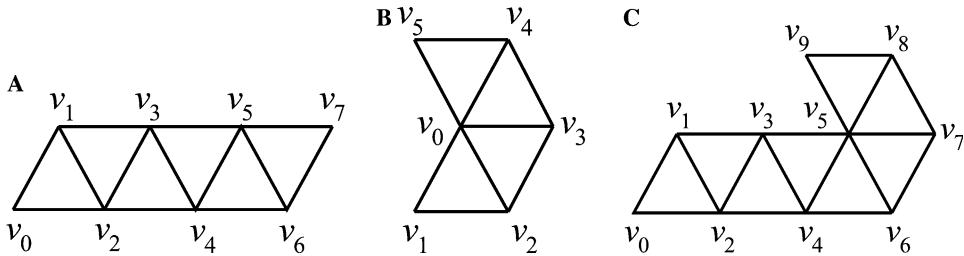


Fig. 4. (A) The triangle strip, (B) the triangle fan, and (C) the generalized triangle strip.

However, since there are about twice as many triangles as vertices in a typical mesh, some vertex indices should be repeated in the generalized triangle strip representation of the mesh, which indicates a waste of storage. To alleviate this problem, several schemes have been developed, where a vertex buffer is utilized to store the indices of recently traversed vertices.

Deering [13] first introduced the concept of the generalized triangular mesh. A generalized triangular mesh is formed by combining generalized triangle strips with a vertex buffer. He used a first-in-first-out (FIFO) vertex buffer to store the indices of up to 16 recently visited vertices. If a vertex is saved in the vertex buffer, it can be represented with the buffer index that requires a less number of bits than the global vertex index. Assuming that each vertex is reused by the buffer index only once, Taubin and Rossignac [3] showed that the generalized triangular mesh representation requires approximately 11 bpv to encode the connectivity data for large meshes. Deering, however, did not propose a method to decompose a mesh into triangle strips.

Based on Deering's work, Chow [14] proposed a mesh compression scheme optimized for real-time rendering. He proposed a mesh decomposition method, illustrated in Fig. 5. First, it finds a set of boundary edges. Then, it finds a fan of triangles around each vertex incident on two consecutive boundary edges. These triangle fans are combined to form the first generalized triangle strip. The triangles in this strip are marked as discovered, and a new set of boundary edges is generated to separate discovered triangles from undiscovered triangles. The next generalized triangle strip is similarly formed from the new set of boundary edges. With the vertex

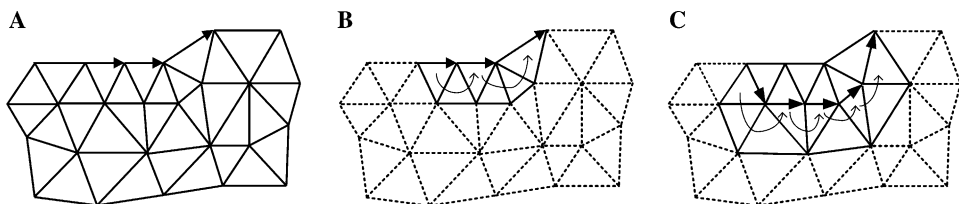


Fig. 5. (A) A set of boundary edges, (B) triangle fans for the first strip, and (C) triangle fans for the second strip, where thick arrows show selected boundary edges and thin arrows show the triangle fans associated with each inner boundary vertex.

buffer, the vertices in the previous generalized triangle strip can be reused in the next one. This process continues until all triangles in a mesh are traversed.

The triangle strip representation can be applied to a triangular mesh of arbitrary topology. However, it is effective only if the triangle mesh is decomposed into long triangle strips. It is a challenging computational geometry problem to obtain an optimal triangle strip decomposition [15,16]. Several heuristics have been proposed to obtain sub-optimal decompositions at a moderate computational cost [17–19].

### 3.1.3. Spanning tree

Turan [20] observed that the connectivity of a planar graph can be encoded with a constant number of bpv using two spanning trees: a vertex spanning tree and a triangle spanning tree. Based on this observation, Taubin and Rossignac [3] presented a topological surgery approach to encode mesh connectivity. The basic idea is to cut a given mesh along a selected set of cut edges to make a planar polygon. The mesh connectivity is then represented by the structures of cut edges and the polygon. In a simple mesh, any vertex spanning tree can be selected as the set of cut edges.

Fig. 6 illustrates the encoding process. Fig. 6A is an octahedron mesh. First, the encoder constructs a vertex spanning tree as shown in Fig. 6B, where each node corresponds to a vertex in the input mesh. Then, it cuts the mesh along the edges of the vertex spanning tree. Fig. 6C shows the resulting planar polygon and the triangle spanning tree. Each node in the triangle spanning tree corresponds to a triangle in the polygon, and two nodes are connected if and only if the corresponding triangles share an edge.

Then, the two spanning trees are run-length encoded. A run is defined as a tree segment between two nodes with degrees not equal to 2. For each run of the vertex spanning tree, the encoder records its length with two additional flags. The first flag is the branching bit indicating whether a run subsequent to the current run starts at the same branching node, and the second flag is the leaf bit indicating whether the

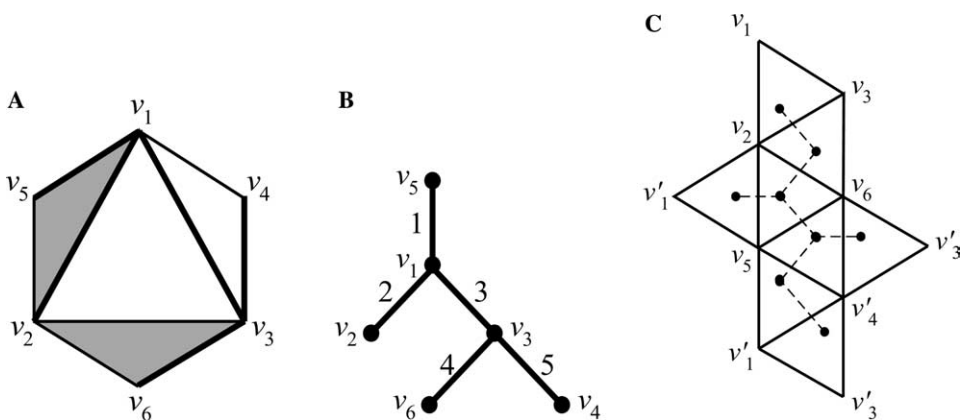


Fig. 6. (A) An octahedron mesh, (B) its vertex spanning tree, and (C) the cut and flattened mesh with its triangle spanning tree shown by dashed lines.



current run ends at a leaf node. For example, let us encode the vertex spanning tree in Fig. 6B, where the edges are labelled with their run indices. The first run is represented by  $(1,0,0)$ , since its length is 1, the next run does not start at the same node, and it does not end at a leaf node. In this way, the vertex spanning tree in Fig. 6B is represented by

$$(1,0,0), (1,1,1), (1,0,0), (1,1,1), (1,0,1).$$

Similarly, for each run of the triangle spanning tree, the encoder writes its length and the leaf bit. Note that the triangle spanning tree is always binary so that it does not need the branching bit. Furthermore, the encoder records the marching pattern with one bit per triangle to indicate how to triangulate the planar polygon internally. The decoder can reconstruct the original mesh connectivity from this set of information.

In both vertex and triangle spanning trees, a run is a basic coding unit. Thus, the coding cost is proportional to the number of runs, which in turn depends on how the vertex spanning tree is constructed. Taubin and Rossignac's algorithm builds the vertex spanning tree based on layered decomposition, which is similar to the way we peel an orange along a spiral path, to maximize the length of each run and minimize the number of runs generated.

Taubin and Rossignac also presented several modifications so that their algorithm can encode general manifold meshes: meshes with arbitrary genus, meshes with boundary, and non-orientable meshes. However, their algorithm cannot directly handle non-manifold meshes. As a preprocessing step, the encoder should split a non-manifold mesh into several manifold components, thereby duplicating non-manifold vertices, edges, and faces. Experimentally, Taubin and Rossignac's algorithm costs 2.48–7.0 bpv for mesh connectivity. It was also shown that the time as well as the space complexities of their algorithm are  $O(N)$ , where  $N$  is the maximum of the vertex number  $v$ , the edge number  $e$ , and the triangle number  $f$  in a mesh. This method demands a large memory buffer due to its global random vertex access at the decompression stage.

#### 3.1.4. Layered decomposition

Bajaj et al. [21] presented a connectivity coding method using a layered structure of vertices. It decomposes a triangular mesh into several concentric layers of vertices, and then constructs triangle layers within each pair of adjacent vertex layers. The mesh connectivity is represented by the total number of vertex layers, the layout of each vertex layer, and the layout of triangles in each triangle layer. Ideally, a vertex layer does not intersect itself and a triangle layer is a generalized triangle strip. In such a case, the connectivity compression is reduced to the coding of the number of vertex layers, the number of vertices in each vertex layer, and the generalized triangle strip in each triangle layer. However, in practice, overhead bits are introduced due to the existence of branching points, bubble triangles, and triangle fans.

Branching points are generated when a vertex layer intersects itself. In Fig. 7A, the middle layer intersects itself at the branching point depicted by a big dot. Branching points divide a vertex layer into several segments called contours. To encode the layout of a vertex layer, we need to encode the information of both contours and

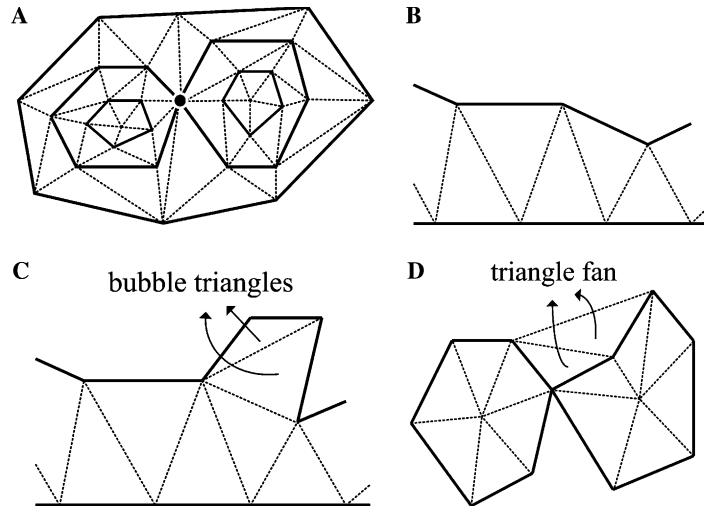


Fig. 7. Illustration of (A) the layered vertex structure and the branching point depicted by a black dot, (B) a triangle strip, (C) bubble triangles, and (D) a cross-contour triangle fan, where contours are depicted with solid lines and other edges with dashed lines.

branching points. Also, as shown in Figs. 7B–D, each triangle in a triangle layer can be classified into one of three cases.

- (1) Its vertices lie on two adjacent vertex layers. A generalized triangle strip is composed of a sequence of triangles of this kind.
- (2) All its vertices belong to one contour. It is called a bubble triangle.
- (3) Its vertices lie on two or three contours in one vertex layer. A cross-contour triangle fan consists of a sequence of triangles of this kind.

Therefore, in addition to encoding generalized triangle strips between two adjacent vertex layers, this algorithm requires extra bits to encode bubble triangles and cross-contour triangle fans.

Taubin and Rossignac [3] also employed layered decomposition in the vertex spanning tree construction. However, Bajaj et al.'s algorithm [21] is different from [3] in the following:

- It does not combine vertex layers into the vertex spanning tree.
- Its decoder does not require a large memory buffer, since it accesses only a small portion of vertices at each decompression step.
- It is applicable to any kind of mesh topology, while [3] cannot encode non-manifold meshes directly.

The layered decomposition method encodes the connectivity information using about 1.40–6.08 bpv. Moreover, it has a desirable property. That is, each triangle depends on at most two adjacent vertex layers and each vertex is referenced by at most

two triangle layers. This property enables the error-resilient transmission of mesh data, since the effects of transmission errors can be localized by encoding different vertex and triangle layers independently. Based on the layered decomposition method, Bajaj et al. [22] also proposed an algorithm to encode large CAD models. This algorithm extends the layered decomposition method to compress quadrilateral and general polygonal models as well as CAD models with smooth non-uniform rational B-splines (NURBS) patches.

### 3.1.5. Valence-driven approach

The valence-driven approach starts from a seed triangle whose three edges form the initial borderline. The borderline divides the whole mesh into two parts, i.e., the inner part that has been processed and the outer part that is to be processed. Then, the borderline gradually expands outwards until the whole mesh is processed. The output is a stream of vertex valences, from which the original connectivity can be reconstructed.

In [23], Touma and Gotsman proposed a pioneering algorithm known as the valence-driven approach. It starts from an arbitrary triangle, and pushes its three vertices into a list called the active list. Then, it pops up a vertex from the active list, traverses all untraversed edges connected to that vertex, and pushes the new vertices into the end of the list. For each processed vertex, it outputs the valence. Sometimes, it needs to split the current active list or merge it with another active list. These cases are encoded with special codes. Before encoding, for each boundary loop, a dummy vertex is added and connected to all the vertices in that boundary loop, making the topology closed. Fig. 8 shows an example of the encoding process, where the active list is depicted by thick lines, the focus vertex by the black dot, and the dummy vertex by the gray dot. Table 1 lists the output of each step in association with Fig. 8.

Since vertex valences are compactly distributed around 6 in a typical mesh, arithmetic coding can be adopted to encode the valence information of a vertex effectively [23]. The resulting algorithm uses less than 1.5 bpv on average to encode mesh connectivity. This is the state-of-the-art compression ratio which has not been seriously challenged till now. However, their algorithm is only applicable to orientable and manifold meshes.

Alliez and Desbrun [24] proposed a method to further improve the performance of Touma and Gotsman's algorithm. They observed that split codes, split offsets, and dummy vertices consume a non-trivial portion of coding bits in Touma and Gotsman's algorithm. To reduce the number of split codes, they used a heuristic method that chooses the vertex with the minimal number of free edges as the next focus vertex, instead of choosing the next vertex in the active list. To reduce the number of bits for split offsets, they excluded the two adjacent vertices of the focus vertex in the current active list that are not eligible for the split, and sort the remaining vertices according to their Euclidean distances to the focus vertex. Then, a split offset is represented with an index into this sorted list, which is further added by 6 and encoded in the same way as a normal valence. To reduce the number of dummy vertices, they used one common dummy vertex for all boundaries in the input mesh. In addition, they encoded the output symbols with the range encoder [25], an effective adaptive arithmetic encoder.

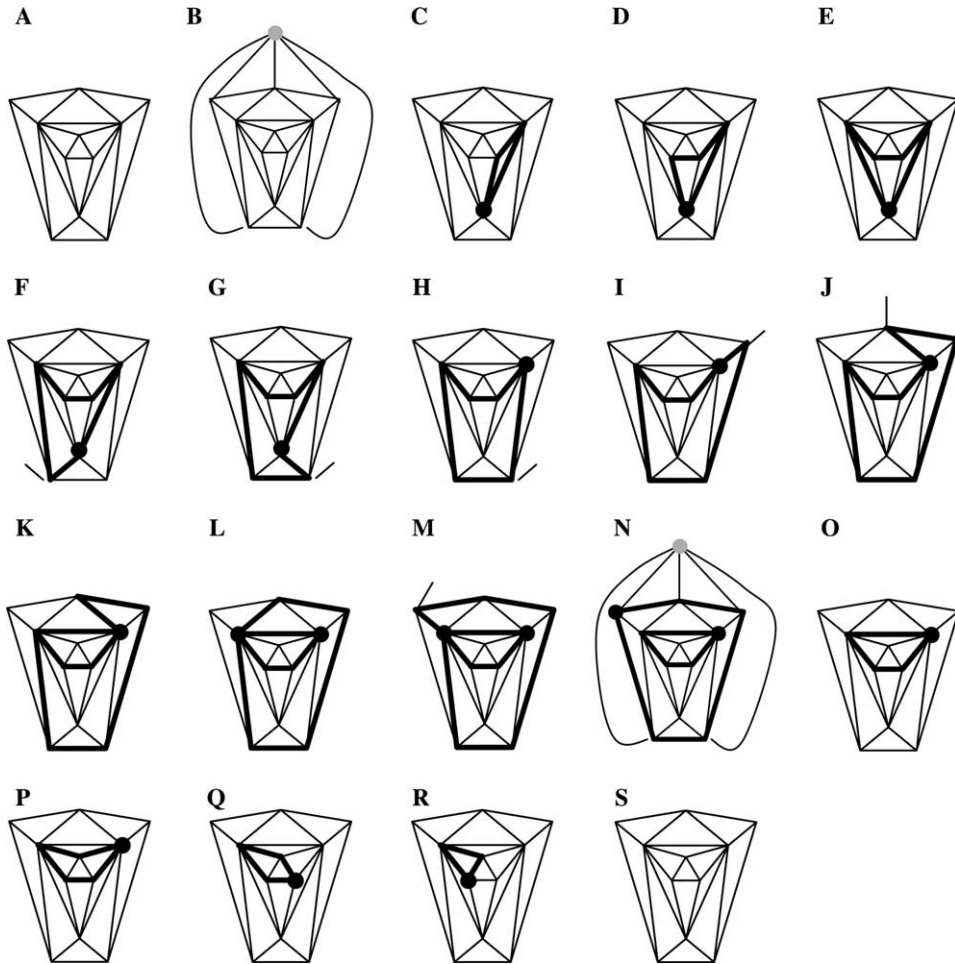


Fig. 8. A mesh connectivity encoding example by Touma and Gotsman [23], where the active list is shown with thick lines, the focus vertex with the black dot, and the dummy vertex with the gray dot.

Alliez and Desbrun’s algorithm is also applicable only to orientable manifold meshes. It performs better than Touma and Gotsman’s algorithm, especially for irregular meshes. Alliez and Desbrun proved that if the number of splits is negligible, the performance of their algorithm is upper-bounded by 3.24 bpv, which is exactly the same as the theoretical bpv value computed by enumerating all possible planar graphs [26].

### 3.1.6. Triangle conquest

Similar to the valence-driven approach, the triangle conquest approach starts from the initial borderline, which divides the whole mesh into conquered and unconquered parts, and inserts triangle by triangle into the conquered parts. The main

Table 1  
The output of each step in Fig. 8

Figure	Output	Comments
(A)		An input mesh is given
(B)		Add a dummy vertex
(C)	Add 6, add 7, add 4	Output the valences of starting vertices
(D)	Add 4	Expand the active list
(E)	Add 7	Expand the active list
(F)	Add 5	Expand the active list
(G)	Add 5	Expand the active list
(H)		Choose the next focus vertex
(I)	Add 4	Expand the active list
(J)	Add 5	Expand the active list
(K)	Split 5	Split the active list, and push the new active list into stack
(L)		Choose the next focus vertex
(M)	Add 4	Expand the active list
(N)	Add dummy 5	Choose the next focus vertex and conquer the dummy vertex
(O)		Pop the new active list from the stack
(P)	Add 4	Expand the active list
(Q)		Choose the next focus vertex
(R)		Choose the next focus vertex
(S)		The whole mesh is conquered

difference is that the triangle conquest approach outputs the building operations of new triangles, while the valence-driven approach outputs the valences of new vertices.

Gumhold and Straßer [27] first proposed a triangle conquest approach, called the cut-border machine. At each step, this algorithm inserts a new triangle into the conquered part, closed by the cut-border, using one of the five building operations: ‘new vertex,’ ‘forward,’ ‘backward,’ ‘split,’ and ‘close.’ The sequence of building operations is encoded using Huffman codes. This algorithm can encode manifold meshes that are either orientable or non-orientable. Experimentally, its compression performance lies within 3.22–8.94 bpv, mostly around 4 bpv. Its most important feature is that the decompression speed is very fast and the decompression method is easy to implement in hardware. Moreover, compression and decompression operations can be processed in parallel. These properties make the method very attractive in real-time coding applications. In [28], Gumhold further improved the compression performance using an adaptive arithmetic coder and optimizing the border encoding. The experimental compression ratio is within the range of 0.3–2.7 bpv, and on average 1.9 bpv.

Rossignac [5] proposed the edgebreaker algorithm, which is another example of the triangle conquest approach. It is nearly equivalent to the cut-border machine, except that it does not encode the offset data associated with the split operation. The triangle traversal is controlled by edge loops as shown in Fig. 9A. Each edge loop bounds a conquered region and contains a gate edge. At each step, this algorithm focuses on one edge loop and its gate edge is called the active gate, while the other edge loops are stored in a stack and will be processed later. Initially, for each connected component, one edge loop is defined. If the component has no physical boundary, two half edges corresponding to one edge are set as the edge loop. For

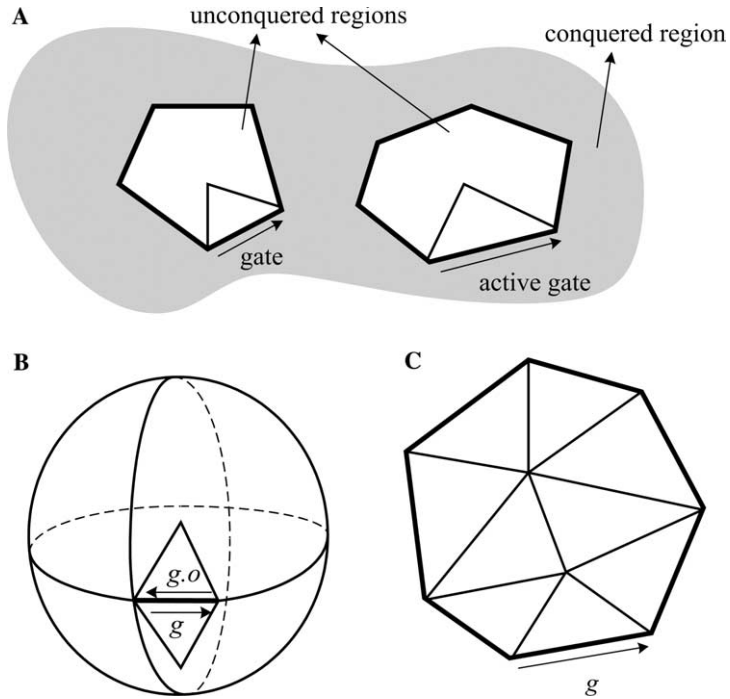


Fig. 9. Illustration of (A) edge loops and (B) gates and initial edge loops for a mesh without boundary, and (C) gates and initial edge loops for a mesh with boundary, where thick lines depict edge loops, and  $g$  denotes the gate.

example, in Fig. 9B, the mesh has no boundary and the initial edge loop is formed by  $g$  and  $g.o$ , where  $g.o$  is the opposite half edge of  $g$ . In Fig. 9C, the initial edge loop is the mesh boundary.

At each step, this algorithm conquers a triangle incident on the active gate, updates the current loop, and moves the active gate to the next edge in the updated loop. For each conquered triangle, this algorithm outputs an op-code. Assume that the triangle to be removed is enclosed by active gate  $g$  and vertex  $v$ , there are five kinds of possible op-codes as shown in Fig. 10A:

- (1)  $C$  (loop extension), if  $v$  is not on the edge loop;
- (2)  $L$  (left), if  $v$  immediately precedes  $g$  in the edge loop;
- (3)  $R$  (right), if  $v$  immediately follows  $g$ ;
- (4)  $E$  (end), if  $v$  precedes and follows  $g$ ;
- (5)  $S$  (split), otherwise.

Essentially, the compression process is a depth-first traversal of the dual graph of the mesh. When the split case is encountered, the current loop is split into two, and one of them is pushed into the stack while the other is further traced. Fig. 10B shows

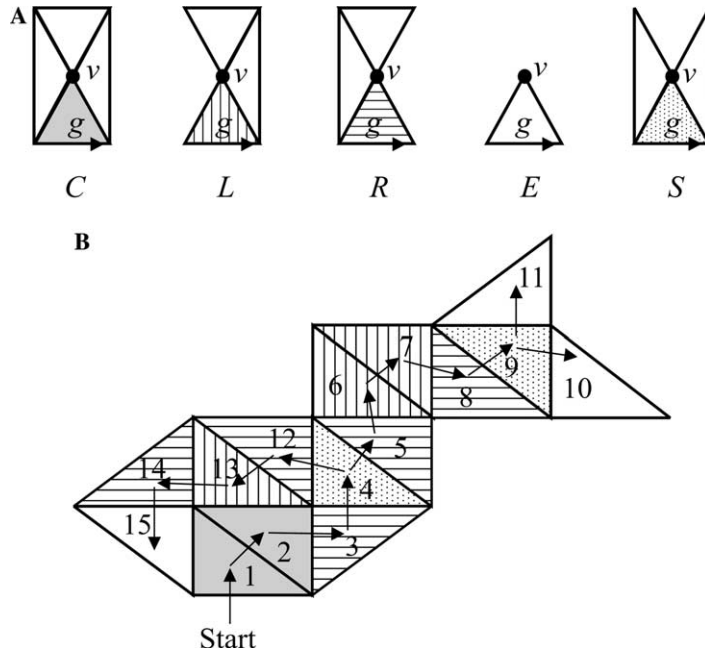


Fig. 10. (A) Five op-codes *C*, *L*, *R*, *E*, and *S*, where the gate *g* is marked with an arrow, and (B) an example of the encoding process in the edgebreaker algorithm where the arrows and the numbers show the traversal order and different filling patterns are used to represent different op-codes.

an example of the encoding process, where the arrows and the numbers give the order of the triangle conquest. The triangles are filled with different patterns to represent different op-codes, which are generated when they are conquered. For this case, the encoder outputs the series of op-codes as *CCRSRLLRSEERLRE*.

The edgebreaker method can encode the topology data of orientable manifold meshes with multiple boundary loops or with arbitrary genus, and guarantee a worst-case coding cost of 4 bpv for simple meshes. However, it is unsuitable for streaming applications, since it requires a two-pass process for decompression, and the decompression time is  $O(v^2)$ . Another disadvantage is that, even for regular meshes, it requires about the same bitrate as that for non-regular meshes.

King and Rossignac [29] modified the edgebreaker method to guarantee a worst-case coding cost of 3.67 bpv for simple meshes, and Gumhold [30] further improved this upper bound to 3.522 bpv. The decoding efficiency of the edgebreaker method was also improved to exhibit linear time and space complexities in [29,31,32]. Furthermore, Szymczak et al. [33] optimized the edgebreaker method for meshes with high regularity by exploiting dependencies of output symbols. It guarantees a worst-case performance of 1.622 bpv for sufficiently large meshes with high regularity.

As mentioned earlier, we can reduce the amount of data transmission between CPU and the graphic card by decomposing a mesh into long triangle strips, but find-

ing a good decomposition (or stripification) is often computationally intensive. Thus, it is often desirable to generate long strips from a given mesh only once and distribute the stripification information together with the mesh. Based on this observation, Isenburg [34] proposed an approach to encode the mesh connectivity together with its stripification information. It is basically a modification of the edgebreaker method, but its traversal order is guided by strips obtained with the STRIPE algorithm [17]. When a new triangle is included, its relation to the underlying triangle strip is encoded with a label. The label sequences are then entropy encoded. The experimental compression performance ranges from 3.0 to 5.0 bpv.

### 3.1.7. Summary

Table 2 summarizes the bitrates of various connectivity coding methods reviewed above. The bitrates marked by ‘\*’ are the theoretical upper bounds obtained by the worst-case analysis, while the others are experimental bitrates. Among these methods, Touma and Gotsman’s algorithm [23] is considered as the state-of-the-art technique for single-rate 3D mesh compression. With some minor improvements on Touma and Gotsman’s algorithm, Alliez and Desbrun’s algorithm [24] yields an improved compression ratio.

The indexed face set, triangle strip, and layered decomposition methods can encode meshes with arbitrary topology. In contrast, the other approaches can handle only manifold meshes with additional constraints. For instance, the valence-driven approach [23,24] require that the manifold is also orientable. Szymczak et al.’s algorithm [33] requires that the manifold has neither boundary nor handles. Note that using these algorithms, a non-manifold mesh can be handled only if it is pre-converted to a manifold mesh by replicating non-manifold vertices, edges, and faces as in [35].

Table 2  
The bitrates of single-rate mesh connectivity coding algorithms

Category	Algorithm	Bitrate (bpv)	Comment
Indexed face set	VRML ASCII Format [1]	$6 \log_2 v$	No compression
Triangle strip	Deering [13]	11	
Spanning tree	Taubin and Rossignac [3]	2.48–7.0	
Layered decomposition	Bajaj et al. [21]	1.40–6.08	
Valence-driven approach	Touma and Gotsman [23]	0.2–2.4, 1.5 on average	Especially good for regular meshes
	Alliez and Desbrun [24]	0.024–2.96, 3.24*	
Triangle conquest	Gumhold and Straßer [27]	3.22–8.94, 4 on average	Optimized for real-time applications
	Gumhold [28]	0.3–2.7, 1.9 on average	
	Rossignac [5]	4*	
	King and Rossignac [29]	3.67*	
	Gumhold [30]	3.522*	
	Szymczak et al. [33]	1.622* for sufficiently large meshes with high regularity	Optimized for regular meshes

The bitrates marked by ‘\*’ are theoretical upper bounds obtained by the worst-case analysis.



### 3.2. Geometry coding

The state-of-the-art connectivity coding schemes require only a few bits per vertex, and their performance is regarded as being very close to the optimal. In contrast, geometry coding received much less attention in the past. Since geometry data dominate the total compressed mesh data, more focus has been shifted to geometry coding recently.

All the single-rate mesh compression schemes encode connectivity data losslessly, since connectivity is a discrete mesh property. However, geometry data are generally encoded in a lossy manner. To exploit high correlation between the positions of adjacent vertices, most single-rate geometry compression schemes follow a three-step procedure: pre-quantization of vertex positions, prediction of quantized positions, and entropy coding of prediction residuals.

#### 3.2.1. Scalar quantization

Uncompressed geometry data typically specify each coordinate component with an IEEE 32-bit floating-point number. However, this precision is beyond human eyes' perceiving capability and is far more than needed for most applications. Thus, quantization can be performed to reduce the data amount without serious impairment on visual quality. Quantization is a lossy procedure since it represents a large or infinite set of values with a smaller set.

Quantization techniques can be classified into scalar/vector and uniform/non-uniform ones [36]. Each cell is of the same length in the uniform scalar quantizer while cells have different lengths in the non-uniform scalar quantizer. Compared with non-uniform vector quantization, uniform scalar quantization is simple and computationally efficient even though it is not optimal in the rate-distortion (R-D) performance.

Typical mesh geometry coding schemes uniformly quantize each coordinate at 8- to 16-bit quantization resolutions. In [13,3,23,21], the same quantization resolution is globally applied. In [14], a mesh is partitioned into several regions, and then different quantization resolutions are adaptively chosen for different regions according to local curvature and triangle sizes. Within each region, the vertex coordinates are still uniformly quantized.

#### 3.2.2. Prediction

After the quantization of vertex coordinates, vertex positions are predictively encoded. The prediction step exploits the correlation between adjacent vertex coordinates and is most crucial in reducing the amount of geometry data. A good prediction scheme generates prediction errors that have a highly skewed distribution, which are then encoded with entropy coders, such as the Huffman coder or the arithmetic coder. More detailed information on entropy coding can be found in [37].

Different geometry prediction schemes have been proposed in the literature, such as delta prediction [13,14], linear prediction [3], parallelogram prediction [23], and second-order prediction [21]. All these prediction schemes can be treated as a special case of the linear prediction scheme with carefully chosen coefficients.

*Delta prediction.* Adjacent vertices tend to have slightly different coordinates, and the differences (or deltas) between the coordinates are usually very small. In Deering’s work [13] and Chow’s work [14], the deltas of coordinates, instead of the original coordinates, were encoded with variable length codes according to the histogram of deltas. In Deering’s geometry coder, the quantization resolutions range between 10 and 16 bits per coordinate component and the performance is roughly between 36 and 17 bpv. Chow’s geometry coder achieves bitrates of 13–18 bpv at quantization resolutions of 9–12 bits per coordinate component.

*Linear prediction.* Taubin and Rossignac [3] employed a linear prediction scheme, where the position of a vertex is predicted from a linear combination of positions of  $K$  previous vertices in the vertex spanning tree. The  $K$  previous vertices are uniquely selected along the path from the root to the current vertex in the vertex spanning tree. More specifically, the position  $v_n$  of the  $n$ th vertex is given by

$$v_n = \sum_{i=1}^K \lambda_i \cdot v_{n-i} + \epsilon(v_n), \tag{6}$$

where  $\lambda_1, \lambda_2, \dots, \lambda_K$  are chosen to minimize the mean square error

$$E\{\|\epsilon(v_n)\|^2\} = E\{\|v_n - \sum_{i=1}^K (\lambda_i \cdot v_{n-i})\|^2\},$$

and transmitted to the decoder as the side information. The bitrate of geometry coding is not directly reported in [3]. However, as estimated by Touma and Gotsman [23], it is about 13 bpv at 8-bit quantization resolution. Note that the delta prediction is a special case of the linear prediction with  $K = 1$  and  $\lambda_1 = 1$ .

*Parallelogram prediction.* Touma and Gotsman [23] used a more sophisticated prediction scheme. To encode a new vertex  $r$ , it considers a triangle with two vertices  $u$  and  $v$  on the active list, where triangle  $(u, v, w)$  is already encoded as shown in Fig. 11. The parallelogram prediction assumes that the four vertices  $w, u, v$ , and  $r$  form a parallelogram. Therefore, the new vertex position can be predicted as  $r_1^p = v + u - w$ . This method performs well only if the four vertices are exactly or nearly co-planar.

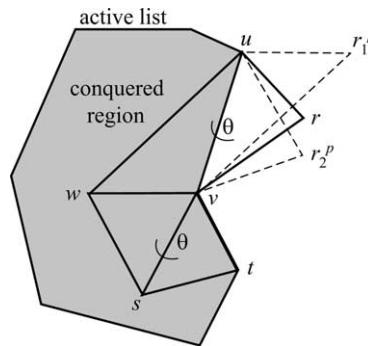


Fig. 11. Parallelogram prediction.

To further improve the prediction accuracy, the crease angle between the two triangles  $(u, v, w)$  and  $(u, v, r)$  can also be estimated using the crease angle,  $\theta$ , between the two triangles  $(s, v, w)$  and  $(s, v, t)$ . In Fig. 11,  $r_2^p$  is the predicted position of  $r$  using the crease angle estimation. This work achieves an average bitrate of 9 bpv at 8-bit quantization resolution. The parallelogram prediction is also a linear prediction in essence since the predicted vertex position is a linear combination of the three previously visited vertex positions.

*Second-order prediction.* In [21], the coordinates of branching points are encoded directly while those of vertices along contours are encoded with a second-order prediction. This is done in two steps. The first step computes and quantizes the differences between adjacent vertex positions. This first step alone is equivalent to the delta prediction. The second step calculates the difference of quantized difference codes. It was confirmed experimentally that the second-order prediction provides a better performance than the delta prediction, when incorporated with entropy coding techniques. The geometry coding bitrate is about 11 bpv at 8-bit quantization resolution and about 14 bpv at 15-bit quantization resolution. The second-order prediction predicts  $v_n - v_{n-1}$  from  $v_{n-1} - v_{n-2}$ . Therefore, it is still a linear predictor, which predicts  $v_n$  from  $2v_{n-1} - v_{n-2}$ .

### 3.2.3. Vector quantization

Recently, vector quantization (VQ) has been proposed for geometry compression in [38,39], which do not follow the conventional quantization-prediction-entropy coding approach. The conventional approach pre-quantizes each vertex coordinate using a scalar quantizer and then predictively encodes the quantized coordinates. In contrast, the VQ approach first predicts vertex positions and then jointly compresses the three components of each prediction residual. Thus, it can utilize the correlation between different coordinate components of the residual. Compared with scalar quantization, the main advantages of VQ include a superior R-D performance, more freedom in choosing quantization cell shapes, and better exploitation of dependence between vector components [36].

In Lee and Ko's work [38], the Cartesian coordinates of a vertex are transformed into a model space vector using the three previous vertex positions. The model space transformation is a kind of prediction and the model space vector can be regarded as a prediction residual. Then, the model space vector is quantized using the generalized Lloyd algorithm [36]. Since they use the original positions of previous vertices in the model space transform, the quantization errors can be accumulated in the decoder. To overcome this encoder–decoder mismatch problem, they periodically insert correction vectors into the bitstream. Experimentally, this scheme requires about 6.7 bpv on average to achieve the same visual quality as conventional methods at 8-bit quantization resolution. (Note that Touma and Gotsman's work requires about 9 bpv at 8-bit resolution [23].) This scheme is especially efficient for 3D meshes with high-geometry regularity.

Chou and Meng [39] also proposed a predictive VQ (PVQ) scheme for mesh geometry compression. To ensure a linear time complexity, a simple predictor is adopted to predict a new vertex from the midpoint of two previously traversed

vertices. Several VQ techniques, including the open loop VQ, the asymptotic closed-loop VQ, and the product code pyramid VQ are applied for residual vector quantization. All these VQ techniques yield a better R-D performance than Deering's work [13], which employs the uniform scalar quantizer and delta coding. A beneficial side effect of this PVQ scheme is that linear vertex transformation forms a rendering pipeline and can be greatly accelerated.

#### 4. Progressive compression

Progressive compression of 3D meshes is desirable for transmission of complex meshes over networks with limited bandwidth. In progressive compression and transmission, a coarse mesh is transmitted and rendered first. Then, refinement data are transmitted to enhance the mesh representation until the mesh is rendered in its full resolution or the transmission task is cancelled by the user. In other words, progressive compression allows transmission and rendering of different levels of details. However, there is a tradeoff between the compression ratio and the number of LODs. Generally speaking, a progressive coder is less effective than a single-rate coder in terms of the coding gain, since it cannot exploit the correlation in mesh data as freely as the single-rate coder.

Progressive mesh compression is closely related to the work on mesh simplification. A detailed review of mesh simplification techniques is beyond the scope of this paper. We refer interested readers to [40–42] for a broader survey on mesh simplification techniques. Typically, to progressively encode a 3D mesh, we first gradually simplify it to a base mesh, which has a much smaller number of vertices, edges, and faces than the original one. During the simplification process, each operation is recorded. By reversing the series of simplification operations, the base mesh can be restored to the original mesh. Progressive mesh coders attempt to compress the base mesh and the series of reversed simplification operations. Progressive mesh coders differ in their mesh simplification techniques, geometry coding methods, and interaction between connectivity coding and geometry coding.

As in single-rate compression, in many progressive coding schemes, the compact representation of connectivity data is given a priority and geometry coding is driven, but restrained at the same time, by connectivity coding. However, new approaches have emerged that compress geometry data with little reference to connectivity data, that drive connectivity coding with geometry coding, and that even change mesh connectivity in favor of a better compression of geometry data. Therefore, we classify the progressive coding schemes into two classes: connectivity-driven compression and geometry-driven compression.

##### 4.1. Connectivity-driven compression

###### 4.1.1. Progressive meshes

Hoppe [43] first introduced the concept of progressive mesh coding with a new mesh representation called progressive mesh (PM). This algorithm simplifies a given

orientable manifold mesh with successive edge collapse operations. As shown in Fig. 12, when an edge is collapsed, its two end points are merged into one, two triangles (or one triangle if the collapsed edge was on the boundary) incident on this edge are removed, and all vertices previously connected to the two end points are re-connected to the merged vertex. The inverse operation of edge collapse is vertex split which inserts a new vertex into the mesh together with corresponding edges and triangles.

An initial mesh  $M = M_k$  can be simplified into a coarser mesh  $M_0$  by applying  $k$  successive edge collapse operations. Each edge collapse  $\text{ecol}_i$  transforms mesh  $M_i$  to  $M_{i-1}$ , with  $i = k, k-1, \dots, 1$ . Since edge collapse operations are invertible, we can represent an arbitrary triangle mesh  $M$  with a base mesh  $M_0$  and a sequence of vertex split operations. Each vertex split  $\text{vsplit}_i$  refines mesh  $M_{i-1}$  to  $M_i$ , with  $i = 1, 2, \dots, k$ . Thus,  $(M_0, \text{vsplit}_1, \dots, \text{vsplit}_k)$  is referred to as the progressive mesh representation of  $M$ .

In the construction of a progressive mesh, it is essential to select a proper edge to be collapsed at each step. As done in Hoppe et al.'s mesh optimization method [44], one can employ an energy function  $E$  that takes into account distance accuracy, attribute accuracy, regularization, and discontinuity curves. Each edge is put into a priority queue, where the priority value is its estimated energy cost  $\Delta E$ . Initially, the algorithm calculates the priority value of each edge. Then, at each iteration, it collapses the edge  $e$  with the smallest priority value and then updates the priorities of edges in the neighborhood of  $e$ .

The connectivity of base mesh  $M_0$  can be encoded using any single-rate coder. The vertex split in Fig. 12 can be specified by the indices of the split vertex  $v_s$  and its left and right vertices,  $v_l$  and  $v_r$ . If there are  $v_i$  vertices in an intermediate mesh  $M_i$ , the index of  $v_s$  can be encoded with  $\log_2 v_i$  bits. Then, the two indices of  $v_l$  and  $v_r$  can be encoded with  $\log_2(\beta(\beta-1))$  bits, where  $\beta$  is the number of vertices connected to  $v_s$ . Since the average vertex valence is 6 in a typical mesh, the indices of  $v_l$  and  $v_r$  can be encoded with about 5 ( $\approx \log_2(6 \times 5)$ ) bits. Thus, about  $(\log_2 v_i + 5)$  bits are required to represent the vertex split operation. Overall, PM requires  $O(v \log_v)$  bits to represent the topology of a mesh with  $v$  vertices. Associated with the vertex split operation, positions of  $v_l$  and  $v_s$  are Huffman-coded after delta prediction.

Despite its innovative nature, the original PM is not a very efficient compression scheme. To improve coding efficiency, Hoppe proposed another PM implementation method in [45]. It reorders the vertex split operations to increase the compression

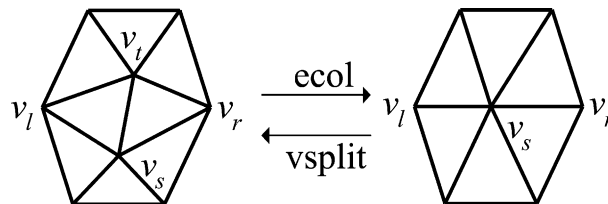


Fig. 12. Illustration of the edge collapse and the vertex split processes.

ratio at the cost of quality degradation of intermediate meshes. It requires about 10.4 bits to represent each vertex split operation.

Hoppe’s PM method has been extended by several researchers as discussed below.

*Progressive simplicial complex.* Popovic and Hoppe [46] observed that PM [43] has two restrictions. First, it is applicable only to orientable manifold meshes. Second, it does not have the freedom to change the topological type of a given mesh during the simplification and refinement, which limits coding efficiency. To overcome these problems, they proposed a method, called progressive simplicial complex (PSC). In this approach, a more general vertex split operation is introduced to encode the changes in both geometry and topology. A PSC representation consists of a single-vertex base model followed by a sequence of generalized vertex split operations. PSC can be employed to compress meshes of any topology type.

To construct a PSC representation, a sequence of vertex unification operations are performed to simplify a given mesh model. Each vertex unification merges an arbitrary pair of vertices, which are not necessarily connected by an edge, into a single vertex. The inverse of the vertex unification is the generalized vertex split operation that splits a vertex into two. Suppose that vertex  $a_i$  in mesh  $M_i$  is to be split to generate a new vertex which will be assigned index  $i + 1$  in mesh  $M_{i+1}$ . Each simplex adjacent to  $a_i$  in  $M_i$  is the vertex unification result of one of four configurations as shown in Fig. 13. For a rigorous definition of simplex, readers are referred to [46]. Intuitively, a zero-dimensional simplex is a point, a one-dimensional simplex is an edge, a two-dimensional simplex is a triangle face, and so on. For each simplex adjacent to  $a_i$ , PSC records a code to indicate one of the four configurations given in Fig. 13.











Simplex dimension	Before vertex split	After vertex split			
		Case 1	Case 2	Case 3	Case 4
0-dim	$\{a_i\}$ •	Undefined	Undefined	$\begin{matrix} \{i+1\} \bullet \\ \{a_i\} \bullet \end{matrix}$	$\begin{matrix}   \\   \\   \end{matrix}$
1-dim					
2-dim					

Fig. 13. Possible configurations after a generalized vertex split for one- and two-dimensional simplices.

Since the generalized vertex split operation is more flexible than the original vertex split operation in PM, PSC may demand more bits for connectivity coding than PM. Specifically, PSC requires about  $(\log_2 v_i + 8)$  bits to specify the connectivity change around the split vertex, while PM requires about  $(\log_2 v_i + 5)$  bits. However, the main benefit of PSC is its capability to deal with arbitrary triangular models without any topology constraint. Similar to PM, the geometry data in PSC are encoded using delta prediction.

*Progressive forest split.* Taubin et al. [47] proposed the progressive forest split (PFS) presentation that is applicable to manifold meshes. As done in PM [43], a triangular mesh is represented with a low resolution base model and a sequence of refinement operations in PFS. Instead of using the vertex split operation, the PFS representation adopts the forest split operation as illustrated in Fig. 14. The forest split operation cuts a mesh along the edges in the forest and fills in the resulting crevice with triangles. In Fig. 14, the forest contains only one tree for the sake of simplicity. However, a forest can be composed of many complex trees, and a single forest split operation can double the number of triangles in a mesh. Therefore, PFS can achieve a much higher compression ratio than PM at the expense of reduced granularity.

For each forest split operation, the structure of the forest, the triangulation information of the crevices, and the vertex displacements are encoded. To encode the structure of the forest, one bit is used for each mesh edge indicating whether it belongs to the forest. To encode the triangulation of the crevices, the triangle spanning tree and the marching patterns can be used as in Taubin and Rossignac's algorithm [3], or a constant length encoding scheme can be employed, which requires exactly 2 bits per new triangle. To encode the vertex displacements, a smoothing algorithm [48] is applied after connectivity refinement, and then the difference between the original vertex position and the smoothed vertex position is Huffman-coded.

To progressively encode a given mesh with four or five LODs, PFS requires about 7–10 bpv for the connectivity data and 20–40 bpv for the geometry data at 6-bit quantization resolution. Note that the bpv performance is measured with respect to the number of vertices in the original mesh in this section. PFS was adopted in MPEG-4 3DMC [4] as an optional mode for progressive mesh coding.

*Compressed progressive mesh.* Pajarola and Rossignac [49] proposed a modified PM, called the compressed progressive mesh (CPM), which is applicable to manifold

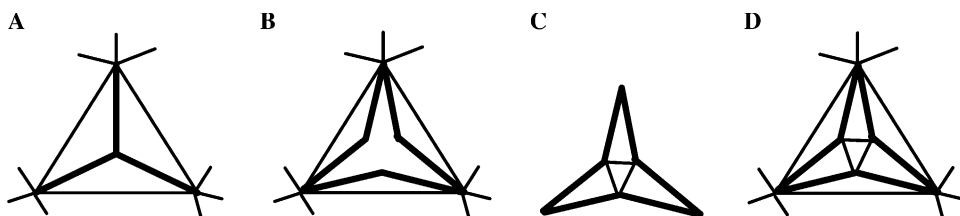


Fig. 14. A forest split operation: (A) an original mesh with a forest marked with thick lines, (B) the cut of the original mesh along the forest edges, (C) triangulation of the crevice, (D) the cut mesh in (B) is filled with the triangulation in (C).

meshes. CPM also improves the compression performance at the cost of reduced granularity. To use fewer bits for connectivity data, vertex splits are grouped into batches. CPM uses a sequence of marking bits to specify the vertices to be split in one batch, while PM uses  $\log_2 v_i$  bits for each vertex split in an intermediate mesh  $M_i$ . For geometry coding, an edge  $(v_1, v_2)$  is collapsed to its midpoint  $v = (v_1 + v_2)/2$ . Thus, if the vector  $D = v_2 - v_1$  is known, the positions of  $v_1$  and  $v_2$  can be reconstructed from  $v$  and  $D$ . CPM obtains the prediction  $\hat{D}$  of  $D$  using vertices which have a topological distance of 1 or 2 from vertex  $v$  in a similar way to the butterfly subdivision technique [50,51]. The prediction error  $E = D - \hat{D}$  is then Huffman-coded. CPM uses the Laplacian distribution to approximate the prediction error histogram. For each batch, it calculates and transmits the variance of the Laplacian distribution for the decoder to reconstruct the Huffman coding table, thus saving the need to transmit the table.

CPM can encode all connectivity information using about 7.0 bpv and all geometry information using about 12–15 bpv at 8- to 12-bit quantization resolutions. Overall, CPM requires about 22 bpv, which is approximately half the bitrate of PFS [47].

In [52], Pajarola and Rossignac optimized CPM for real-time applications. They adopted the half-edge collapse operation to collapse an edge into one of its ending points instead of its midpoint. The half-edge collapse is adopted, since the midpoint may not lie on the quantized coordinate grid which makes geometry coding more complex. Also, to reduce the computational overhead, a new vertex position is estimated from the average of only adjacent vertices within the topological distance of 1. Furthermore, a faster Huffman decoder [53] and a group of pre-computed Huffman coding tables are used. With these optimizations, this algorithm provides a faster decoding speed than Hoppe's efficient implementation of PM [45].

#### 4.1.2. Patch coloring

A triangular mesh can be simplified and hierarchically represented using vertex decimation [54,55]. Being different from the edge collapse approach, the vertex decimation approach removes a vertex and its adjacent edges, and then re-triangulates the resulting hole. The topology data record the way of re-triangulation after each vertex is decimated, or equivalently, the neighborhood of each new vertex before it is inserted.

Cohen-Or et al. [56] proposed the patch coloring algorithm for progressive mesh compression based on vertex decimation. First, an input mesh is simplified by iteratively decimating a set of vertices. At each iteration, decimated vertices are selected such that they are not adjacent to one another. Each vertex decimation leads to a hole, which is then re-triangulated. The set of new triangles filling in this hole is called a patch. By reversing the simplification process, a hierarchical progressive reconstruction process is obtained. In order for the decoder to identify the patches, two patch coloring techniques were proposed: 4-coloring and 2-coloring. The 4-coloring scheme colors adjacent patches with distinct colors, requiring 2 bits per triangle. It is applicable to patches of any degree. The 2-coloring scheme further saves topology bits by coloring the whole mesh with only two colors. It enforces the re-tri-



angulation of each patch in a zigzag way, and encodes the two outer triangles with bit ‘1,’ and the other triangles with bit ‘0.’ Thus, it requires only 1 bit per triangle but applies only to patches with a degree greater than 4. During the encoding process, at each level of detail, either the 2-coloring or the 4-coloring scheme is chosen based on the distribution of patch degrees. Then, the coloring bitstream is encoded with the Ziv–Lempel coder. For geometry coding, the position of a new vertex is simply predicted from the average of its direct neighboring vertices. Experimentally, this approach costs about 6 bpv for connectivity coding and about 16–22 bpv for geometry coding at the 12-bit quantization resolution.

#### 4.1.3. Valence-driven conquest

Alliez and Desbrun [57] proposed a progressive mesh coder for manifold 3D meshes. Observing that the entropy of mesh connectivity is dependent on the distribution of vertex valences, they iteratively applied the valence-driven decimating conquest and the cleaning conquest in pair to get multiresolution meshes. The vertex valences are output and entropy encoded during this process.

The decimating conquest is a mesh simplification process based on vertex decimation. It only decimates vertices with valences less or equal to 6 to maintain a statistical concentration of valences around 6. In the decimating conquest, a 3D mesh is traversed from patch to patch. A degree- $n$  patch is a set of triangles incident to a common vertex of valence  $n$ , and a gate is an oriented boundary edge of a patch, storing the reference to its front vertex. The encoder enters a patch through one of its boundary edges, called the input gate. If the front vertex of the input gate has a valence less or equal to 6, the encoder decimates the front vertex, re-triangulates the remaining polygon, and outputs the front vertex valence. Then, it pushes the other boundary edges, called output gates, into a FIFO list, and replaces the current input gate with the next available gate in the FIFO list. This procedure repeats until the FIFO list becomes empty.

In fact, a breadth-first patch traversal is performed in the decimating conquest. Fig. 15A illustrates the decimating conquest on a 6-regular mesh. An initial input gate  $g_1$  is chosen, a degree-6 patch is conquered and the output gates,  $g_2$ – $g_6$ , are pushed into the FIFO list. Next,  $g_2$  is chosen as the new input gate and another patch is conquered, and so on. Each conquered patch is re-triangulated so that the valences of half of the vertices on the patch boundary become lower. Therefore, the mesh after the decimating conquest have many vertices with valence 3 as shown in Fig. 15B, and the vertex valences are no more concentrated around 6.

To maintain the statistical concentration of valences, a cleaning conquest is applied after each decimating conquest. The cleaning conquest is almost the same as the decimating conquest, except that the output gates are placed on the two edges of each face adjacent to the patch border, instead of on the patch border itself, and that only valence-three vertices are decimated. For example, in Fig. 15B, suppose that an initial input gate  $g_1$  is chosen. Then, its front vertex of valence 3 is decimated, and  $g_2$ – $g_5$  are chosen as the output gates. Fig. 15C shows the resulting mesh after a pair of decimating and cleaning conquests. We can see that the resulting mesh is also a 6-regular mesh as the original mesh in Fig. 15A. If an input mesh is irreg-

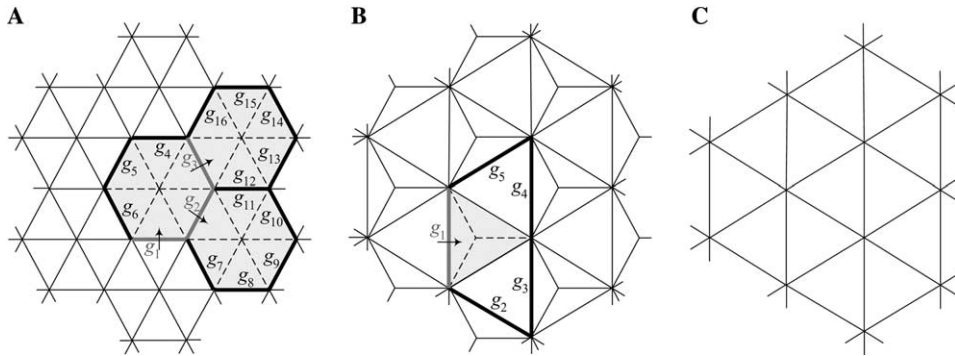


Fig. 15. An example of (A) the decimating conquest, (B) the cleaning conquest, and (C) the resulting mesh after the decimating conquest and the cleaning conquest, where the shaded areas represent the conquered patches and the thick lines represent the gates. The gates to be processed are depicted in the black color, while the gates already processed are in the gray color. Each arrow represents the direction of entrance into a patch.

ular, it may not be completely covered by patches in the decimating conquest. In such a case, null patches are generated.

For geometry coding, Alliez and Desbrun [57] used the barycentric prediction and the approximate Frenet coordinate frame. The normal and the barycenter of a patch approximates the tangent plane of the surface. Then, the position of the inserted vertex is encoded as an offset from the tangent plane.

For connectivity coding, this scheme requires about 2–5 bpv, on average 3.7 bpv, which is about 40% better than the results reported in [49,56]. For geometry coding, the cost typically ranges from 10 to 16 bpv for quantization resolutions between 10 and 12 bits. Especially, the geometry coding cost is much less than 10 bpv for meshes with high-connectivity regularity and geometry uniformity. Furthermore, this scheme has a comparable performance with that of the state-of-the-art single-rate coder. This scheme yields a compressed file size only about 1.1 times larger than Touma and Gotsman’s algorithm [23], even though it supports full progressiveness.

#### 4.1.4. Embedded coding

Li and Kuo [58] introduced the concept of embedded coding to encode connectivity and geometry data in an interwoven manner. The geometry data as well as the connectivity data are encoded progressively. Thus, as the coded data stream is received and decoded by the receiver, not only new vertices are added to the model, but also the precision of each old vertex position is progressively increased. This coding scheme is applicable to triangular meshes of any topology and it preserves the topology during mesh simplification.

For mesh simplification, Li and Kuo also adopted the vertex decimation method. To record the neighborhood of each new vertex before it is inserted, their algorithm utilizes a pattern table. It encodes the index to the pattern table and the indices of one marked triangle and one marked edge to locate the selected pattern within the

mesh. For each vertex insertion, the topology data requires about  $(\log_2 v_i + 6)$  bits experimentally, where  $v_i$  is the number of vertices in current mesh  $M_i$ .

The position of each vertex is predicted from the average position of its adjacent vertices, and the residue is obtained. Then, the encoder multiplexes topology data and geometry residual data into one data bitstream. Suppose that a residue is quantized as  $0a_0a_1\cdots$  in binary format. Fig. 16 shows the integration process, where each column represent the data associated with a vertex insertion. ‘\*’ denotes the topology data,  $a_0a_1\cdots$  denotes the residue data for that vertex, and the flags ‘0’ and ‘1’ determine the order of bits in the final bitstream, which is depicted by the zigzag lines in Fig. 16. As more bits are received and decoded, more vertices are inserted and the precision of each vertex positions is increased. The order of bits, determined by the flags, is selected by the encoder to achieve the R-D tradeoff.

This algorithm requires about 20 bpv to decode a mesh model at acceptable quality. However, at this bitrate, only one-third of the total number of vertices and triangles are reconstructed, since a significant portion of bits are used to increase the precisions of important vertices rather than to increase the number of reconstructed vertices.

#### 4.1.5. Layered decomposition

In [59], Bajaj et al. generalized their single-rate mesh coder [21] based on layered decomposition to a progressive mesh coder that is applicable to arbitrary meshes. An input mesh is decomposed into layers of vertices and triangles. Then, the mesh is simplified through three stages: intra-layer simplification, inter-layer simplification, and generalized triangle contraction. The former two are topology-preserving, whereas the last one may change the mesh topology.

The intra-layer simplification operation selects vertices to be removed from each contour. After those vertices are removed, re-triangulation is performed in the region between the simplified contour and its adjacent contours. A bit string is encoded to indicate which vertices are removed, and extra bits are encoded to reconstruct the

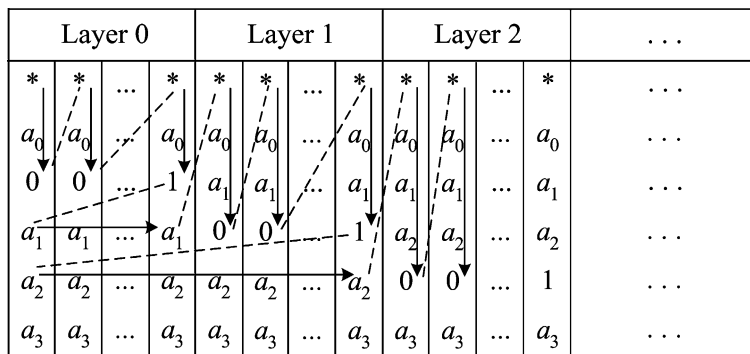


Fig. 16. The multiplexing of topology data and geometry data, where the zigzag lines illustrate the bit order.

original connectivity between the decimated vertex and its neighbors in the refinement process.

In the inter-layer simplification stage, a contour can be totally removed. Then, the two triangle strips sharing the removed contour are replaced by a single coarse strip [60]. Fig. 17 shows an example of contour removal and re-triangulation. A dashed line in Fig. 17B, called a constraining chord, is associated with each edge in the contour to be removed, which is illustrated with a thick line. The simplification process is encoded as (0, 6, 2, 3, 1, 3), where the first bit indicates whether the contour is open or closed, the second number is the number of vertices in the removed contour, and the remaining numbers indicate the number of triangles between every two consecutive constraining chords in the coarse strip.

After intra-layer and inter-layer simplifications, the mesh can be further simplified using the generalized triangle contraction [61], which contracts a triangle  $t$  into a single point  $p$ . To reduce the storage overhead, the single point  $p$  is chosen as the barycenter of the triangle  $t$ . By allowing the generalized triangle contraction, this scheme can simplify even a very complex model into a single triangle or vertex, achieving a guaranteed size of the coarsest level mesh.

The connectivity cost for the whole mesh is  $O(v)$  due to the locality of the layering structure, which is much better than PM that requires  $O(v \log_2 v)$  bits. Experimentally, it costs about 10–17 bpv for connectivity coding and 30 bpv for geometry coding at 10 or 12-bit quantization resolution. For geometry coding, as in the corresponding single-rate algorithm [21], the second-order prediction is used to exploit the correlation between consecutive correction vectors.

## 4.2. Geometry-driven compression

### 4.2.1. Kd-tree decomposition

In most mesh compression techniques, geometry coding is guided by the underlying connectivity coding. Gandoin and Devillers [62] proposed a fundamentally different strategy, where connectivity coding is guided by geometry coding. Their algorithm works in two passes: the first pass encodes geometry data progressively without considering connectivity data. The second pass encodes connectivity changes between two successive LODs. Their algorithm can encode arbitrary simplicial complexes without any topological constraint.

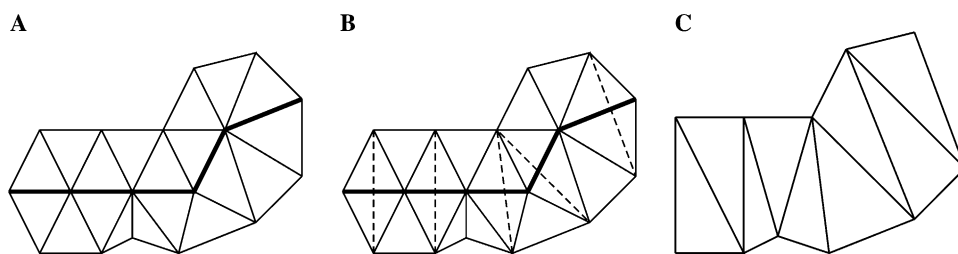


Fig. 17. Inter-layer simplification: (A) the fine level, (B) constraining chords, and (C) the coarse strip, where dashed lines depict constraining chords, and thick lines depict the contour to be removed.

For geometry coding, their algorithm employs a kd-tree decomposition based on cell subdivisions [63]. Each time it subdivides a cell into two child cells, it encodes the number of vertices in one of the two child cells. If the parent cell contains  $p$  vertices, the number of vertices in one of the child cells can be encoded using  $\log_2(p + 1)$  bits with an arithmetic coder [64]. This subdivision is recursively applied, until each non-empty cell is small enough to contain only one vertex and enable a sufficiently precise reconstruction of the vertex position. Fig. 18 illustrates the geometry coding process with a 2D example. First, the total number of vertices, 7, is encoded using a fixed number of bits (32 in this example). Then, the entire cell is divided vertically into two cells, and the number of vertices in the left cell, 4, is encoded using  $\log_2(7 + 1)$  bits. Note that the number of vertices in the right cell is not encoded, since it is deducible from the number of vertices in the entire cell and the number of vertices in the left cell. The left and right cells are then horizontally divided, respectively, and the numbers of vertices in the upper cells are encoded, and so on. To improve the coding gain, the number of vertices in a cell can be predicted from the point distribution in its neighborhood.

For connectivity coding, their algorithm encodes the topology change after each cell subdivision using one of two operations: vertex split [43] or generalized vertex split [46]. Specifically, after each cell subdivision, the connectivity coder records a symbol, indicating which operation is used, and parameters specific to that operation. Compared to [43,46], their algorithm has the advantage that split vertices are implicitly determined by the subdivision order given in geometry coding, reducing the topology coding cost. Moreover, to improve the coding gain further, they proposed several rules, which predict the parameters for vertex split operations efficiently using already encoded geometry data.

On average, this scheme requires 3.5 bpv for connectivity coding and 15.7 bpv for geometry coding at 10 or 12-bit quantization resolution, which is better than progressive mesh coders presented in [52,57]. This scheme is even comparable to the single-rate mesh coder given in [23], achieving a full progressiveness at the cost of only 5% overhead bitrate. It is also worthwhile to point out that this scheme is especially useful for terrain models and densely sampled objects, where topology data can be losslessly reconstructed from geometry data. Besides its good coding gain, it can be easily extended to compress tetrahedral meshes.

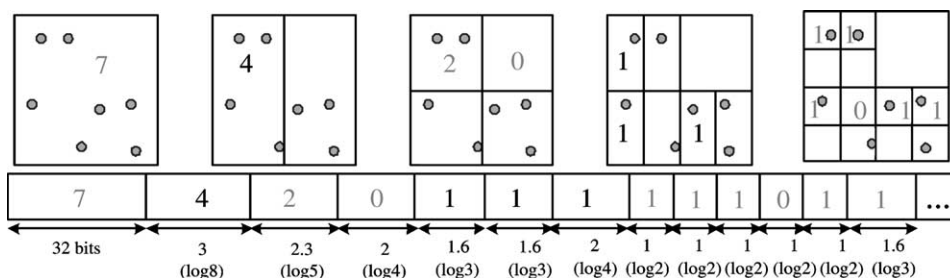


Fig. 18. Illustration of the kd-tree geometry coding in the 2D case.

#### 4.2.2. Octree decomposition

Peng and Kuo [65] proposed a progressive lossless mesh coder based on the octree decomposition, which can encode triangular meshes of arbitrary topology. Given a 3D mesh, an octree structure is first constructed through recursive partitioning of the bounding box. The mesh coder traverses the octree in a top-down fashion and encodes the local changes of geometry and connectivity associated with each octree cell subdivision. In [65], the geometry coder does not encode the vertex number in each cell, but encodes the information whether each cell is empty or not, which is usually more concise in the top levels of the octree. For connectivity coding, a uniform approach is adopted, which is efficient and easily extendable to arbitrary polygonal meshes.

For each octree cell subdivision, the geometry coder encodes the number,  $T$  ( $1 \leq T \leq 8$ ), of non-empty-child cells and the configuration of non-empty-child cells among  $K_T = C_8^T$  possible combinations. When the data are encoded straightforwardly,  $T$  takes 3 bits and the non-empty-child-cell configuration takes  $\log_2 K_T$  bits. To further improve coding efficiency,  $T$  is arithmetic coded using the context of the parent cell's octree level and valence, leading to 30–50% bit saving. Furthermore, all  $K_T$  possible configurations are sorted according to their estimated probability values, and the index of the configuration in the sorted array is arithmetic coded. The probability estimation is based on the observation that non-empty-child cells tend to gather around the centroid of the parent-cell's neighbors. This technique leads to more than 20% improvement.

For the connectivity coding, each octree cell subdivision is simulated by a sequence of kd-tree cell subdivisions. Each vertex split corresponds to a kd-tree cell subdivision, which generates two non-empty-child cells. Let the vertex to split be denoted by  $v$ , the neighbor vertices before the vertex split by  $N_i$ 's and the two new vertices from the vertex split by  $v_1$  and  $v_2$ . Then, the following information will be encoded: (1) vertices among  $N_i$ 's that are connected to both  $v_1$  and  $v_2$  (called the pivot vertices); (2) whether each non-pivot vertex in  $N_i$  is connected to  $v_1$  or  $v_2$ ; and (3) whether  $v_1$  and  $v_2$  are connected in the refined mesh. During the coding process, a triangle regularity metric is used to predict each neighbor vertex's probability of being a pivot vertex, and a spatial distance metric is used to predict the connectivity of non-pivot neighbor vertices to the new vertices. At the decoder side, the facets are constructed from the edge-based connectivity without an extra coding cost. To further improve the R-D performance, prioritized cell subdivision is applied. Higher priorities are given to cells of a bigger size, a bigger valence, and a larger distance from neighbors.

The octree-based mesh coder outperforms the kd-tree algorithm [62] in both geometry and connectivity coding efficiency. For the geometry coding, it provides about 10–20% improvement for typical meshes, but up to 50–60% improvement for meshes with highly regular geometry data and/or tightly clustered vertices. For the connectivity coding, the improvement ranges from 10 to 60%.

#### 4.2.3. Spectral coding

Spectral (or transform) coding has been successfully used in the compression of 2D image and video data [66]. The discrete cosine transform (DCT) or the discrete

Fourier transform (DFT) is often used to represent a sequence of source samples to another sequence of transform coefficients, whose energy is concentrated in relatively few low frequency coefficients. Thus, graceful degradation can be obtained if we encode low frequency coefficients while discarding higher frequency ones.

Karni and Gotsman [67] used the spectral theory on meshes [68] to compress geometry data. Suppose that a mesh consists of  $n$  vertices. Then, the mesh Laplacian matrix  $L$  of size  $n \times n$  is derived from the mesh connectivity, given by

$$L_{i,j} = \begin{cases} 1 & \text{if } i = j, \\ -1/d_i & \text{if vertices } i \text{ and } j \text{ are adjacent,} \\ 0 & \text{otherwise,} \end{cases} \quad (7)$$

where  $d_i$  is the valence of vertex  $i$ . The eigenvectors of  $L$  form an orthogonal basis of  $R^n$  and the associated eigenvalues represent the frequencies of those basis functions. The encoder projects the  $x$ ,  $y$ , and  $z$  coordinate vectors of the mesh onto the basis functions to obtain the geometry spectra, respectively. Then, the encoder quantizes these spectra, truncates high-frequency coefficients, and entropy encodes the quantized coefficients. This approach can naturally support progressiveness by transmitting the coefficients in the increasing order of frequencies.

Experimentally, this approach requires only 1/2–1/3 of the bitrate of Touma and Gotsman's algorithm [23] to achieve a similar visual quality. This approach is especially suitable for smooth meshes, which can be faithfully represented with a fewer number of low frequency coefficients.

Finding the eigenvectors of an  $n \times n$  matrix requires  $O(n^3)$  computational complexity. To reduce the computations, an input mesh can be partitioned into several segments, and each segment can be independently encoded. However, the eigenvectors should be computed in the decoder as well. Thus, even though the partitioning is incorporated, the decoding complexity is too high for many real-time applications. To overcome this problem, Karni and Gotsman [69] proposed to use fixed basis functions, which are computed from a 6-regular connectivity. Those basis functions are actually the Fourier basis functions. Therefore, the encoding and the decoding can be performed with the fast Fourier transform (FFT) efficiently. Before encoding, the connectivity of an input mesh is mapped into a 6-regular connectivity. No geometry information is used during the mapping. Thus, the decoder can perform the same mapping with separately received connectivity data and determine the correct ordering of vertices. The use of fixed basis functions is obviously not optimal, but provides an acceptable performance at much lower complexity.

#### 4.2.4. Wavelet coding

Khodakovsky et al. [70] proposed a progressive compression algorithm based on the wavelet transform. It first remeshes an arbitrary manifold mesh  $M$  into a semi-regular mesh, where most vertices have degree 6, using the MAPS algorithm [71]. MAPS generates a semi-regular approximation of  $M$  by finding a coarse base mesh and successively subdividing each triangle into four triangles. Fig. 19 shows a reme-

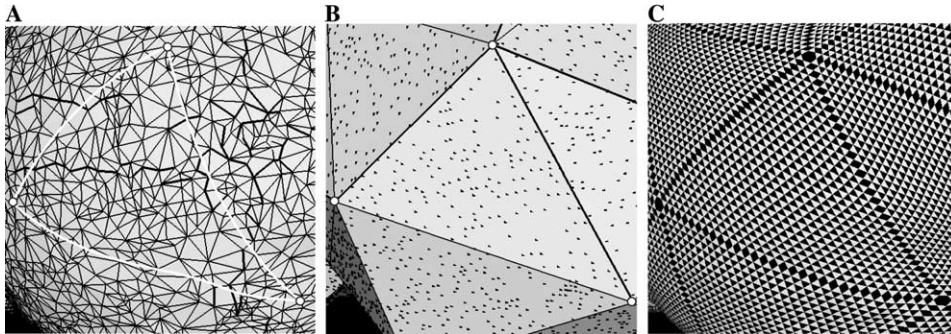


Fig. 19. A remeshing example: (A) an irregular mesh, (B) the corresponding base mesh, and (C) the corresponding semi-regular mesh, where triangles are illustrated with a normal flipping pattern to clarify the semi-regular connectivity.

shing example. In this figure, vertices within the region bounded by white curves in Fig. 19A are projected onto a base triangle. These projected vertices are depicted by black dots in Fig. 19B. Each projected vertex onto the base triangle contains the information of the original vertex position. By interpolating these original vertex positions, each subdivision point can be mapped approximately to a point (not necessarily a vertex) in the original mesh.

Note that the connectivity information of the semi-regular mesh can be efficiently encoded, since it can be reconstructed using only the connectivity of the base mesh and the number of subdivisions. However, this algorithm attempts to preserve only the geometry information. Thus, the original connectivity of  $M$  cannot be reconstructed at the decoder.

Then, using the Loop algorithm [72], the semi-regular mesh geometry is represented by the base mesh geometry and a sequence of wavelet coefficients. These coefficients represent the differences between successive LODs, and have a concentrated distribution around zero, which favors entropy coding. The wavelet coefficients are encoded using a zerotree approach, introducing progressiveness into the geometry data. More specifically, they modified the SPIHT algorithm [73], which is one of the successful 2D image coders, to compress the Loop wavelet coefficients. Their algorithm provides about 12 dB or four times better image quality than CPM [47], and even a better performance than Touma and Gotsman's single-rate coder [23]. This is mainly due to the fact that they employed semi-regular meshes, enabling the wavelet coding approach.

Khodakovskiy and Guskov [74] later proposed another wavelet coder based on the normal mesh representation [75]. In the subdivision, their algorithm restricts that the offset vector should be in the normal direction of the surface. Therefore, whereas 3D coefficients are used in [70], 1D coefficients are used in the normal mesh algorithm. Furthermore, their algorithm employs the unlifted version of butterfly wavelets [50,51] as the transform. As a result, it achieves about 2–5 dB image quality improvement over that in [70].



#### 4.2.5. Geometry image coding

Gu et al. [76] proposed to represent the geometry of a 3D manifold surface with a 2D regular array of resampled vertices, which constitute a geometry image. Each pixel value in the geometry image represents a 3D position vector  $(x, y, z)$ . Due to its regular structure, the geometry image representation can facilitate the compression and rendering of 3D data.

To generate the geometry image, an input manifold mesh is cut and opened to be homeomorphic to a disk. The cut mesh is then parameterized onto a 2D square, which is in turn regularly sampled. In the cut process, an initial cut is first selected and then iteratively refined. At each iteration, it selects a vertex of the triangle with the biggest geometric stretch and inserts the path, connecting the selected vertex to the previous cut, into the refined cut. After the final cut is determined, the boundary of the square domain is parameterized with special constraints to prevent cracks along the cut, and the interior is parameterized using the geometry-stretch parameterization in [77], which attempts to distribute vertex samples evenly over the 3D surface.

Geometry images can be compressed using standard 2D image compression techniques, such as wavelet-based coders. To seamlessly zip the cut in the reconstructed 3D surface, especially when the geometry image is compressed in a lossy manner, it encodes the sideband signal, which records the topological structure of the cut boundary and its alignment with the boundary of the square domain.

The geometry image compression provides about 3 dB worse R-D performance than the wavelet mesh coder [70]. Also, since it maps complex 3D shapes onto a simple square, it may yield large distortions for high-genus meshes and unwanted smoothing of 3D features. Praun and Hoppe [78] and Hoppe and Praun [79] proposed an approach to parameterize a manifold 3D mesh with genus 0 onto a spherical domain. Compared with the square domain approach [76], this approach leads to a simple cut topology and an easy-to-extend image boundary. It was shown by experiments that the spherical geometry image coder achieves better R-D performance than the square domain approach [76] and the wavelet mesh coder [70], but slightly worse performance than the normal mesh coder [74].

#### 4.3. Summary

In Table 3, we summarize the bitrates of progressive mesh coding algorithms, which are extracted from experimental results reported in the original papers. Those explicit bitrates stand for the final bitrates required to decode meshes at the most refined level.

The progressive mesh (PM) coder [43] was a pioneering algorithm that has a connectivity cost of  $O(v \log v)$ . PFS [47], CPM [49], the patch coloring technique [56], and the layered decomposition algorithm [59] reduced the cost to  $O(v)$ . The valance-driven conquest algorithm [57] and the kd-tree decomposition algorithm [62] require less than 4 bpv on the average for the connectivity coding. The embedded coding concept was introduced to multiplex the connectivity and the geometry data in [58].

Table 3  
 Bitrates of progressive mesh coding algorithms, which are of the form “geometry bitrate in bpv:connectivity bitrate in bpv (quantization resolutions in bits)”

Category		Algorithm	Bitrate $C:G$ (Q)	Comment
Connectivity-driven compression	Progressive meshes	Hoppe [43]	$O(v \log_2 v):N/A$	
		Popovic and Hoppe [46]	$O(v \log_2 v):N/A$	
		Taubin et al. [47]	7–10:20–40 (6)	
		Pajarola and Rossignac [49]	7:12–15 (8, 10, 12)	
Geometry-driven compression	Patch coloring	Cohen-Or et al. [56]	6:16–22 (12)	
	Valence-driven conquest	Alliez and Desbrun [57]	3.7:10–16 (10, 12)	
	Embedded Coding	Li and Kuo [58]	$O(v \log_2 v):N/A$	Embedded multiplexing
	Layered decomposition	Bajaj et al. [59]	10–17:30 (10, 12)	
	Kd-tree decomposition	Gandoin and Devillers [62]	3.5:15.7 (10, 12) for manifold meshes	Capable of encoding triangle soups
	Octree decomposition	Peng and Kuo [65]	40–90% bitrate of [62] for similar quality	
	Spectral coding	Karni and Gotsman [67]	30–50% bitrate of [23] for similar quality	
	Wavelet coding	Khodakovsky et al. [70]	12 dB better quality than [47] at the same bitrate	Loss of original connectivity
Geometry image coding		Khodakovsky and Guskov [74]	2–5 dB better quality than [70] at the same bitrate	
		Gu et al. [76]	3 dB worse quality than [70]	Loss of original connectivity
		Praun and Hoppe [78,79]	Better R-D than [76,70], slightly worse R-D than [74]	

For the geometry coding, a bitrate of 15 bpv at a quantization resolution of around 10 bits has been achieved by CPM [49], the valence-driven conquest [57], or the kd-tree decomposition [62]. These progressive coders [57,62] have excellent performance in the sense that they support the progressive coding property at a bitrate that is slightly higher than the state-of-the-art single-rate coder [23]. The octree decomposition algorithm [65] further reduces the overall bitrate of [62] by 10–60%.

The spectral coding [67], the wavelet coding [70,74], and the geometry image coding methods [76,78,79] improve the coding gain and provide even better compression performance than the single-rate coder in [23]. It is worthwhile to point out that these coding algorithms are generalizations of successful 2D image coding techniques, e.g., JPEG and JPEG-2000.

PSC [46] and the kd-tree decomposition algorithm [62] can compress arbitrary simplicial complexes. The patch coloring algorithm [56], the embedded coding algorithm [58] the layered decomposition algorithm [59], and the octree decomposition algorithm [65] can encode triangular meshes with arbitrary topology. All the remaining algorithms can deal with manifold triangular meshes only. In the wavelet coding [70,74] and the geometry image coding [76,78,79], the original connectivity is lost due to the remeshing procedure.

## 5. Trends

### 5.1. Polygonal mesh compression

Most 3D mesh compression algorithms focus on triangular meshes. To handle polygonal meshes, they triangulate polygons before the compression task. However, there are several disadvantages in this approach. First, the triangulation process imposes an extra cost in computation and efficiency. Second, the original connectivity information may be lost. Third, attributes associated with vertices or faces may require duplicated encoding. To address these problems, several algorithms have been proposed to encode polygonal meshes directly without pre-triangulation.

King et al. [80] first proposed a connectivity coding algorithm for quadrilateral or mixed triangle/quadrilateral meshes, by generalizing the edgebreaker algorithm [5,31] that is one of the triangle conquest methods. It guarantees the worst-case coding cost of 2.67 bpv for meshes without valence-two vertices. Isenburg and Snoeyink [81] proposed another extension of the triangle conquest methods, called the face fixer, to encode polygonal meshes. The connectivity coding cost is within 1.67–2.93 bpv for several test models.

Lee et al. [82] proposed the angle-analyzer for triangle/quadrilateral mesh coding. They also extended the triangle conquest approach, and introduced 12 kinds of op-codes to encode the connectivity of a mixed triangle/quadrilateral mesh. To minimize the entropy of the op-code sequence, their algorithm chooses each gate adaptively to suppress the splitting and the merging of edge loops. Also, they proposed the local coordinate-based scheme and the angle-based scheme to compress geometry data effectively. On average, their algorithm yields 40 and 20% better compression

ratios for connectivity data and geometry data than the state-of-the-art triangular mesh coder given in [23].

Isenburg [83] and Khodakovsky et al. [84] independently proposed similar algorithms to encode the connectivity of a manifold polygonal mesh. Their algorithms are extensions of the valence-driven approach [23,24], and represent the connectivity of a polygonal mesh by a sequence of vertex valences and a sequence of face degrees. Vertex valences and face degrees are highly correlated. Specifically, low valence vertices are likely to be surrounded by high-degree faces. This property is exploited for the mutual prediction of vertex valences and face degrees. Isenburg's algorithm provides slightly better compression performance than Khodakovsky et al.'s algorithm, and requires 0.76–2.54 bpv for connectivity coding experimentally.

Isenburg and Alliez [85] applied the parallelogram prediction rule to encode geometry data in polygonal meshes. This scheme attempts to form a parallelogram prediction within a polygon rather than across polygons. Since polygons tend to be planar and convex, the 'within' prediction leads to a better coding gain in general. Experimentally, this scheme requires about 23% less geometry bitrates than the conventional parallelogram prediction rule.

## 5.2. Volume mesh compression

The field of volume visualization has received much attention and made substantial progress recently. Its main applications include medical diagnostic data representation and physical phenomenon modelling. Tetrahedral meshes are popularly used to represent volume data, since they are suitable for irregularly sampled data and facilitate multiresolution analysis and visibility sorting. A tetrahedral mesh is typically represented by two tables: the vertex table that records the position and the attributes (such as the temperature or the pressure) of each vertex, and the tetrahedron table that stores a quadruple of four vertex indices for each tetrahedron. A tetrahedral mesh often requires an enormous storage space even at a moderate resolution. The huge storage requirement puts a great burden on the storage, communication, and rendering systems. Thus, efficient compression schemes are necessary.

Szymczak and Rossignac [86] proposed a connectivity coding algorithm for tetrahedral meshes. It can be seen as an extension of the topological surgery algorithm [3], and encodes the connectivity of a mesh using two strings: the tetrahedron spanning tree string and the folding string. The required bitrate is around 7 bits per tetrahedron. Gumhold et al. [87] proposed another approach by extending the cut-border machine [27]. In a tetrahedral mesh, the cut-border is the triangular surface dividing the conquered volume from the unconquered volume, and the gate is a triangle on the cut-border. Each time a face-adjacent tetrahedron is added at the gate, an opcode is generated. Experimentally, the bitrate is less than 2.4 bits per tetrahedron. Pajarola et al. [88] proposed a progressive connectivity coding algorithm for tetrahedral meshes. Similar to PM [43], it simplifies a tetrahedral mesh into a base mesh using edge collapse operations, and then represents the original mesh with the base mesh and a sequence of vertex split operations. This scheme requires about 5 bits per tetrahedron.

Recently, hexahedral meshes also have become popular due to their numerical advantages in finite element computations over tetrahedral meshes. Isenburg and Alliez [89] proposed a compression algorithm for hexahedral meshes, by extending Touma and Gotsman's algorithm [23]. Specifically, the topology of a hexahedral mesh is represented by a sequence of edge degrees. Note that the degree of an edge is defined as the number of faces adjacent to that edge. Also, the vertex positions are predictively encoded using the parallelogram prediction rule. On average, their algorithm requires 1.55 bits per hexahedron for topology coding, and 17.28 bpv for geometry coding at the 16-bit resolution.

### 5.3. Isosurface compression

Numerous volume data have been and are being produced and/or utilized in fields including 3D photography, scientific visualization, magnetic resonance imaging (MRI), and computed tomography (CT). Typically, volume data are produced by sampling a scalar field (function)  $\omega(\mathbf{x})$ ,  $\mathbf{x} \in \mathbf{R}^3$  at points on a 3D grid, and are often visualized with isosurfaces. The isosurface for isovalue  $c$  consists of all 3D points satisfying  $\omega(\mathbf{x}) = c$ , and is often represented with a polygonal mesh. Due to the high resolution, an isosurface often demands a huge amount of storage/bandwidth. At the same time, due to the high-topology complexity (i.e., a great variety of the connected component number and the genus number), general polygonal mesh compression schemes often fail to yield satisfying performance. In the recent couple of years, special purpose algorithms have been proposed to compress isosurfaces exploiting their special structure properties: restricted vertex positions in cells of intersect and restorable mesh connectivity from geometry data. Those algorithms again can be classified into single-rate and progressive methods.

Saupe and Kuska [90] proposed an isosurface compression scheme with the Marching Cubes (MC) algorithm [91], where each vertex is restricted to one intersecting grid edge. Three coding phases are preformed: coding intersecting cells with an octree, coding the intersecting edge configuration in each intersecting cell with an integer between 0 and 6, and coding the interpolating value for each vertex along the corresponding intersecting edge with a quantized integer. The output symbols from all three phases are further compressed with **gzip**. Note that connectivity is not explicitly encoded, which can be reconstructed at the decoder from geometry data. As a result, it achieves about 2.2–2.8 better compression ratio than the general purpose polygonal mesh coder [3] for high-complex CT scanned volumes. Later on, Saupe and Kuska [92] improved their algorithm in, where a modified pruned octree is used, and eight neighboring grid edges of an intersecting grid edge form a context exploited by an adaptive arithmetic coder. The resultant compression ratio improvement is around 50%.

In Zhang et al.'s algorithm [93], grid points are organized into slices, and the cells between two adjacent slices form a layer. A grid point is called relevant if it is one end point of an intersecting grid edge. For a cell, once we know which points of its eight corners are relevant, we can determine which of its bounding edges are intersecting edges. One bit is associated with each grid point (cell) to indicate whether it is

a (an) relevant grid point (intersecting cell). The exact positions of vertices can be derived from the function values at the eight corners. This algorithm encodes all the slice bitmaps and layer bitmaps with arithmetic coding and encodes the function values associated with the relevant points in each slice with quantization, second-order differential, and arithmetic coding. Experimentally, the compression ratio improvement upon the general mesh coder [21] is 22–44%. Another advantage is that the coding (decoding) process is incremental with a small memory requirement.

Yang and Wu [94] proposed to compress MC-generated triangular isosurfaces through cube index encoding, edge index encoding, and vertex coordinate encoding. Using a 3D-checkbox structure, this algorithm encodes at most 1/4 of cube indices. The cubes to encode in a layer are organized into multiple chains, each of which is chain-coded. In edge index encoding, one sign bit is associated with each cube corner indicating whether its function value is greater or less than the isovalue. The sign bits of the eight corners form the sign pattern of an intersecting cube, which implies the configuration of the intersecting edges and the triangulation of the vertices in that cube. The sign patterns corresponding to the chain-coded cubes are encoded. Similar to [90], the vertex coordinates are also encoded with quantized interpolating values. Experimentally, this algorithm achieves 2.01–2.10 bits per triangle (bpt) with indistinguishable visual quality for 32-bit coordinate quantization, and about 1.5 bpt with some visual artifacts for 8-bit coordinate quantization.

Taubin [95] applied some ideas used in the JBIG image compression standard to isosurface compression, where the context-based arithmetic coding is used, exploiting the correlation between the volume data at adjacent grid points. Basically, several 3D binary images are encoded. One is occupancy image, formed by the sign bits at all grid points and compressed with an adaptive arithmetic coder. At each grid point, adjacent sign bits in the same and the previous layers form the context for the bit to encode. For vertex coordinate encoding, the quantized interpolating values are encoded bit layer by bit layer. For each bit layer and each axis, a 3D binary image is formed and encoded similarly to the occupancy image. As an option, this algorithm encodes the vertex normals with a  $2 + 2S$  scheme, where  $S$  is the times of Loop subdivision [72]. As a result, suppose that the interpolating values are quantized to  $B$  bits, the bitrate is about 0.60–0.95 bits per face (bpf) for  $B = 0$ , 1.20–1.80 bpf for  $B = 1$ , and 2.10–2.90 bpf for  $B = 3$ . The main limitation of this algorithm is its high-computational complexity at the decoder, which is proportional to the number of grid points.

Laney et al. [96] introduced progressiveness into the isosurface compression. Their algorithm can encode an isosurface or a triangular mesh through the wavelet compression of a signed-distance volume. The original mesh or isosurface can be reconstructed by extracting the isosurface with zero distance, though the original connectivity is lost. As a first step, an input isosurface or triangular mesh is distance transformed to produce an approximation of the actual distance function, using the modified closest-point propagation algorithm [97]. Secondly, a multiresolution representation is generated with the fast wavelet transform [98]. Then, a distance-based thresholding is performed to remove all wavelet coefficients not contributing to the reconstructed surface. Finally, the resultant wavelet coefficients are compressed with a zero-tree coder [73]. Compared with the wavelet mesh coding approach [70], no

base mesh construction is needed, which is often difficult for complex meshes. Thus, this algorithm is more suitable for surfaces with complex topology and many components. However, its R-D performance for smooth meshes is worse than that of [70].

Lee et al. [99] proposed another progressive isosurface compression scheme which performs three steps on input volume data: (1) an adaptive octree is built up to represent the 3D sign bitmap; (2) the octree structure is encoded; and (3) the geometry refinement data are encoded. The octree structure encoding is called connectivity encoding in [99], which encodes the sign bits for cell corners and the leaf bits for homogeneous cells, i.e., cells with the same sign bits at the eight corners. These two bit streams are encoded using arithmetic coders with different contexts. For the sign bitstream, the seven neighboring sign bits from the same cell and the eight sign bits from the parent cell form a 15-bit context. For the leaf bitstream, the context consists of the previous 1 leaf bit transmitted. Geometry refinement data are encoded at the end of connectivity encoding. Being different from MC-based methods, it localizes vertex positions not on grid edges but within cells. In each intersecting cell, the vertex position is predicted as the barycenter of all intersecting edge midpoints, the prediction residual is represented in a local frame formed by a fixed choice of orthogonal basis vectors with the help of a local interpolating plane. Then, the three components of the residuals are quantized and encoded with context-based arithmetic coders whose context is formed by the eight sign bits of the containing cell. Experimentally, the average compression ratio is 6.10 bpv, of which 0.65 bpv is used for the octree encoding, and the average ratio of improvement over the single-rate isosurface coder [95] is 76%.

Note that all the above algorithms only deal with isosurfaces from regular structured volume data. Compression of isosurfaces from unstructured volume data is still an unexploited field.

#### 5.4. Animated-mesh coding

Three-dimensional animation becomes more and more popular these days. A 3D animation sequence has not only spatial correlation within each frame, but also temporal correlation between adjacent frames. Most 3D animation compression algorithms utilize the spatio-temporal correlation to achieve a high-coding gain.

Lengyel [100] proposed an innovative algorithm for mesh sequence compression. The input mesh is first segmented and the motion of each segment is approximated by an affine transform. Then, the transform parameters and the approximation residuals are encoded. Ahn et al. [101] proposed another segmentation-based algorithm, where the approximation residuals are encoded using DCT. Zhang and Owen [102,103] proposed an octree-based coder, which represents the motion between adjacent frames with an octree. Each octree cell has eight motion vectors associated with its corners, and the motion of each vertex in that cell is approximated by the trilinear interpolation of the corner motion vectors.

Instead of using the segmentation-based approach, Yang et al. [104] proposed a sequence coder based on the vertex-wise motion vector prediction. The first frame

is intra-coded using the technique given in [23], and the following frames are traversed in the same breadth-first order. The motion vector of each vertex is spatio-temporally predicted using the information already available in the traversed region. Ibarria and Rossignac [105] proposed a similar algorithm that traverses each mesh surface in a depth-first order. Each vertex location is predicted from the spatial neighbor vertices and the corresponding vertices in the previous frame.

Alexa and Müller [106] proposed another scheme that represents 3D animation sequences based on the principal component analysis (PCA). Their algorithm forms a matrix that contains the geometry information of all frames and performs its singular value decomposition to measure the importance of each principal component. By omitting less important components, a coding gain can be achieved. However, the main disadvantage is that it demands a huge computational complexity for the large matrix manipulation. Karni and Gotsman [107] also proposed a PCA-based sequence coder by identifying spatial and temporal components and improving the coding gain with a predictive coding scheme.

Briceño et al. [108] proposed a scheme called the geometry video coding, which is a generalization of the geometry image coding [76]. They developed uniform cut and parametrization schemes that can be applied to all frames in a sequence. The first frame is intra-coded while subsequent frames are predictively coded with an affine motion compensation. Guskov and Khodakovskiy [109] proposed a wavelet coding method for mesh sequences. A progressive mesh hierarchy and an anisotropic wavelet are built for the first frame and maintained for subsequent frames. To exploit the temporal correlation, wavelet coefficients between adjacent frames are encoded differentially.

Most animation coding schemes consider the compression of isomorphic sequences only, where the topology and the number of vertices are invariant throughout all frames. An exception is Yang et al.'s scheme [110,111], which can encode non-isomorphic sequences as well. It first constructs a semi-regular mesh for the first frame and maps the regular structure to subsequent frames based on 3D motion estimation. The semi-regular wavelet coefficients are then encoded using an embedded coding scheme that supports SNR and temporal scalability modes.

Some standardization effort for the 3D animation coding is under way. To address the representation and coding of high-quality 2D and 3D animation data, the MPEG-4 animation framework extension (AFX) [112] is being developed by MPEG in cooperation with the Web3D Consortium. Features supported by AFX include a compact representation and low bitrate animation of static and dynamic data, and techniques such as subdivision, interpolator compression, dead-reckoning, and compression of animated paths and animated models are employed.

## 6. Conclusion

In this paper, we performed a survey on current 3D mesh compression techniques by classifying major algorithms, describing main ideas behind each category, and comparing their strength and weakness. Currently, the state-of-the-art single-rate mesh compression schemes are those based on the valence-driven approach. For pro-



gressive mesh compression, the valence-driven approach is still among the best ones. However, schemes driven by kd-tree/octree decomposition and those using transform and wavelet coding techniques have emerged recently.

In early mesh coding schemes, geometry coding was tightly coupled with and restrained by connectivity coding. However, this dependence has been weakened or even reversed. Geometry data tend to consume a dominant portion of the storage space, and their correlation can be exploited more effectively without the restraint of connectivity. Furthermore, remesh-based progressive mesh coders completely discard the irregular connectivity of an input mesh, and resample the surface with a regular pattern. Due to regular resampling, connectivity coding requires almost no information while geometry data can be efficiently compressed.

Research on single-rate coding seems to be mature except for further improvement on geometry coding. Progressive coding has been thought to be inferior to single-rate coding in terms of the coding gain. However, high-performance progressive codecs have emerged these days and they often outperform some of the state-of-the-art single-rate codecs. In other words, a progressive mesh representation seems to be a natural choice, which demands no extra burden in the coding process. There is still room to improve progressive coding to provide better R-D performance at a lower computational cost.

Future mesh coding schemes will be inspired by new 3D representations such as the normal mesh representation and the point-based geometry representation. Another promising research area is animated-mesh coding that was overlooked in the past but is getting more attention recently.

### **Acknowledgment**

The authors thank Prof. Mathieu Desbrun at the California Institute of Technology for his useful comments and suggestions.

### **References**

- [1] The Virtual Reality Modeling Language (VRML). ISO/IEC 14772-1, 1997.
- [2] G. Taubin, W. Horn, F. Lazarus, J. Rossignac, Geometry coding and VRML, *Proc. IEEE* 96 (6) (1998) 1228–1243.
- [3] G. Taubin, J. Rossignac, Geometric compression through topological surgery, *ACM Trans. Graph.* 17 (2) (1998) 84–115.
- [4] Coding of Audio-Visual Objects: Visual. ISO/IEC 14496-2. July 2001.
- [5] J. Rossignac, Edgebreaker: connectivity compression for triangle meshes, *IEEE Trans. Vis. Comput. Graph.* 5 (1) (1999) 47–61.
- [6] G. Taubin, 3D geometry compression and progressive transmission, in: *EUROGRAPHICS—State of the Art Report*, September 1999.
- [7] D. Shikhare, State of the art in geometry compression. Technical Report, National Centre for Software Technology, India, 2000.
- [8] C. Gotsman, S. Gumhold, L. Kobbelt, Simplification and compression of 3D meshes, in: *Tutorials on Multiresolution in Geometric Modelling*, 2002.

- [9] P. Alliez, C. Gotsman, Recent advances in compression of 3D meshes, in: Proceedings of the Symposium on Multiresolution in Geometric Modeling, September 2003.
- [10] H. Edelsbrunner, *Geometry and Topology for Mesh Generation*, Cambridge University Press, Cambridge, 2001.
- [11] D.W. Kahn, *Topology: an Introduction to the Point-set and Algebraic Areas*, Dover Publications, 1995.
- [12] J. Gross, J. Yellen, *Graph Theory and Its Applications*, CRC Press, Boca Raton, 1998.
- [13] M. Deering, Geometry compression, in: ACM SIGGRAPH, 1995, pp. 13–20.
- [14] M. Chow, Optimized geometry compression for real-time rendering, in: IEEE Visualization, 1997, pp. 347–354.
- [15] E.M. Arkin, M. Held, J.S.B. Mitchell, S. Skiena, Hamiltonian triangulations for fast rendering, *Vis. Comput.* 12 (9) (1996) 429–444.
- [16] F. Evans, S. Skiena, A. Varshney, Completing sequential triangulations is hard. Technical Report, Department of Computer Science, State University of New York at Stony Brook, 1996.
- [17] F. Evans, S.S. Skiena, A. Varshney, Optimizing triangle strips for fast rendering, in: IEEE Visualization, 1996, pp. 319–326.
- [18] B. Speckmann, J. Snoeyink, Easy triangle strips for tin terrain models. In Proceedings of 9th CCCG, pages 239–244, 1997..
- [19] X. Xiang, M. Held, J. Mitchell, Fast and efficient stripification of polygonal surface models, in: ACM 1999 Symposium on Interactive 3D Graphics, 1999, pp. 71–78.
- [20] G. Turan, On the succinct representations of graphs, *Discr. Appl. Math.* 8 (1984) 289–294.
- [21] C.L. Bajaj, V. Pascucci, G. Zhuang, Single resolution compression of arbitrary triangular meshes with properties, *Comput. Geom. Theor. Appl.* 14 (1999) 167–186.
- [22] C. Bajaj, V. Pascucci, G. Zhuang, Compression and coding of large cad models. Technical Report, University of Texas, 1998.
- [23] C. Touma, C. Gotsman, Triangle mesh compression, in: Proceedings of Graphics Interface, 1998, pp. 26–34.
- [24] P. Alliez, M. Desbrun, Valence-driven connectivity encoding for 3D meshes, in: EUROGRAPHICS, 2001, pp. 480–489.
- [25] M. Schindler, A fast renormalization for arithmetic coding, in: Proceedings of IEEE Data Compression Conference, 1998, p. 572.
- [26] W. Tutte, A census of planar triangulations, *Can. J. Math.* 14 (1962) 21–38.
- [27] S. Gumhold, W. Straßer, Real time compression of triangle mesh connectivity, in: ACM SIGGRAPH, 1998, pp. 133–140.
- [28] S. Gumhold, Improved cut-border machine for triangle mesh compression, in: Erlangen Workshop '99 on Vision, Modeling and Visualization, IEEE Signal Processing Society, November 1999.
- [29] D. King, J. Rossignac. Guaranteed 3.67v bit encoding of planar triangle graphs, in: 11th Canadian Conference on Computational Geometry, 1999, pp. 146–149.
- [30] S. Gumhold, New bounds on the encoding of planar triangulations. Technical Report WSI-2000-1, Wilhelm-Schickard-Institut für Informatik, University of Tübingen, Germany, January 2000.
- [31] J. Rossignac, A. Szymczak, Wrap and zip decompression of the connectivity of triangle meshes compressed with edgebreaker, *Comput. Geom.* 14 (1-3) (1999) 119–135.
- [32] M. Isenburg, J. Snoeyink, Spirale reversi: reverse decoding of the edgebreaker encoding, in: 12th Canadian Conference on Computational Geometry, 2000, pp. 247–256.
- [33] A. Szymczak, D. King, J. Rossignac, An edgebreaker-based efficient compression scheme for regular meshes, in: Proceedings of 12th Canadian Conference on Computational Geometry, 2000, pp. 257–264.
- [34] M. Isenburg, Triangle strip compression, in: Proceedings of the Graphics Interface, May 2000, pp. 197–204.
- [35] A. Guézic, G. Taubin, F. Lazarus, W. Horn, Converting sets of polygons to manifold surfaces by cutting and stitching, in: IEEE Visualization, 1998, pp. 383–390.
- [36] A. Gersho, R.M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Dordrecht, 1992.

- [37] T.M. Cover, J.A. Thomas, *Elements of Information Theory*, Wiley, New York, 1991.
- [38] E.-S. Lee, H.-S. Ko, Vertex data compression for triangular meshes, in: *Pacific Graphics*, 2000, p. 225.
- [39] P.H. Chou, T.H. Meng, Vertex data compression through vector quantization, *IEEE Trans. Vis. Comput. Graph.* 8 (4) (2002) 373–382.
- [40] P. Heckbert, M. Garland, Survey of polygonal surface simplification algorithms, in: *Multiresolution Surface Modeling Course Notes of SIGGRAPH'97*, 1997.
- [41] P. Cignoni, C. Montani, R. Scopigno, A comparison of mesh simplification algorithms, *Comput. Graph.* 22 (1) (1998) 37–54.
- [42] D.P. Luebke, A developer's survey of polygonal simplification algorithms, *IEEE Comput. Graph. Appl.* 21 (3) (2001) 24–35.
- [43] H. Hoppe, Progressive meshes, in: *ACM SIGGRAPH*, 1996, pp. 99–108.
- [44] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, Mesh optimization, in: *ACM SIGGRAPH*, 1993, pp. 19–25.
- [45] H. Hoppe, Efficient implementation of progressive meshes, *Comput. Graph.* 22 (1) (1998) 27–36.
- [46] J. Popovic, H. Hoppe, Progressive simplicial complexes, in: *ACM SIGGRAPH*, 1997, pp. 217–224.
- [47] G. Taubin, A. Guezic, W. Horn, F. Lazarus, Progressive forest split compression, in: *ACM SIGGRAPH*, 1998, pp. 123–132.
- [48] G. Taubin, A signal processing approach to fair surface design, in: *ACM SIGGRAPH*, 1995, pp. 351–358.
- [49] R. Pajarola, J. Rossignac, Compressed progressive meshes, *IEEE Trans. Vis. Comput. Graph.* 6 (1) (2000) 79–93.
- [50] N. Dyn, D. Levin, J.A. Gregory, A butterfly subdivision scheme for surface interpolation with tension control, *ACM Trans. Graph.* 9 (2) (1990) 160–169.
- [51] D. Zorin, P. Schröder, W. Sweldens, Interpolating subdivision for meshes with arbitrary topology, in: *ACM SIGGRAPH*, 1996, pp. 189–192.
- [52] R. Pajarola, J. Rossignac, Squeeze: fast and progressive decompression of triangle meshes, in: *Proceedings of Computer Graphics International Conference*, 2000, pp. 173–182.
- [53] R. Pajarola, Fast Huffman code processing. Technical Report UCI-ICS-99-43, Information and Computer Science, UCI, 1999.
- [54] W.J. Schroeder, J.A. Zarge, W.E. Lorensen, Decimation of triangle meshes, in: *ACM SIGGRAPH*, 1992, pp. 65–70.
- [55] M. Soucy, D. Laurendeau, Multiresolution surface modeling based on hierarchical triangulation, *Comput. Vis. Image Understand.* 63 (1) (1996) 1–14.
- [56] D. Cohen-Or, D. Levin, O. Remez, Progressive compression of arbitrary triangular meshes, in: *IEEE Visualization*, 1999, pp. 67–72.
- [57] P. Alliez, M. Desbrun, Progressive encoding for lossless transmission of triangle meshes, in: *ACM SIGGRAPH*, 2001, pp. 198–205.
- [58] J. Li, C.-C.J. Kuo, Progressive coding of 3-D graphic models, *Proc. IEEE* 86 (6) (1998) 1052–1063.
- [59] C. Bajaj, V. Pascucci, G. Zhuang, Progressive compression and transmission of arbitrary triangular meshes, in: *IEEE Visualization*, 1999, pp. 307–316.
- [60] C.L. Bajaj, E.J. Coyle, K.-N. Lin, Arbitrary topology shape reconstruction from planar cross sections, *Graph. Models Image Proc.* 58 (6) (1996) 524–543.
- [61] T.S. Gieng, B. Hamann, K.I. Joy, G.L. Schussman, I.J. Trotts, Constructing hierarchies for triangle meshes, *IEEE Trans. Vis. Comput. Graph.* 4 (2) (1998) 145–161.
- [62] P.M. Gandoin, O. Devillers, Progressive lossless compression of arbitrary simplicial complexes, *ACM Trans. Graph.* 21 (3) (2002) 372–379.
- [63] O. Devillers, P. Gandoin, Geometric compression for interactive transmission, in: *IEEE Visualization*, 2000, pp. 319–326.
- [64] I.H. Witten, R.M. Neal, J.G. Cleary, Arithmetic coding for data compression, *Commun. ACM* 30 (6) (1987) 520–540.
- [65] J. Peng, C.-C.J. Kuo, Geometry-guided progressive lossless 3D mesh coding with octree (OT) decomposition, accepted by *SIGGRAPH*, 2005.

- [66] N.S. Jayant, P. Noll, *Digital Coding of Waveforms—principles and Applications to Speech and Video*, Prentice Hall, New Jersey, 1984.
- [67] Z. Karni, C. Gotsman, Spectral compression of mesh geometry, in: *ACM SIGGRAPH*, 2000, pp. 279–286.
- [68] G. Taubin, A signal processing approach to fair surface design, in: *ACM SIGGRAPH*, 1995, pp. 351–358.
- [69] Z. Karni, C. Gotsman, 3D mesh compression using fixed spectral bases, in: *Proceedings of the Graphics Interface*, 2001, pp. 1–8.
- [70] A. Khodakovsky, P. Schröder, W. Sweldens, Progressive geometry compression, in: *ACM SIGGRAPH*, 2000, pp. 271–278.
- [71] A.W.F. Lee, W. Sweldens, P. Schröder, L. Cowsar, D. Dobkin, MAPS: multiresolution adaptive parametrization of surfaces, in: *ACM SIGGRAPH*, 1998, pp. 95–104.
- [72] C. Loop, Smooth subdivision surfaces based on triangles. Master's thesis, Department of Mathematics, University of Utah, August 1987.
- [73] A. Said, W.A. Pearlman, A new, fast, and efficient image codec based on set partitioning in hierarchical trees, *IEEE Trans. Circuits Syst. Video Technol.* 6 (3) (1996) 243–250.
- [74] A. Khodakovsky, I. Guskov, Normal mesh compression, in: *Geometric Modeling for Scientific Visualization*, Springer-Verlag, Germany, 2002.
- [75] I. Guskov, K. Vidimce, W. Sweldens, P. Schröder, Normal meshes, in: *ACM SIGGRAPH*, July 2000, pp. 95–102.
- [76] X. Gu, S.J. Gortler, H. Hoppe, Geometry images, in: *ACM SIGGRAPH*, 2002, pp. 355–361.
- [77] P. Sander, S. Gortler, J. Snyder, H. Hoppe, Signal-specialized parametrization. Technical Report MSR-TR-2002-27, Microsoft Research, January 2002.
- [78] E. Praun, H. Hoppe, Spherical parametrization and remeshing, *ACM Trans. Graph.* 22 (3) (2003) 340–349.
- [79] H. Hoppe, E. Praun, Shape compression using spherical geometry images, in: N. Dodgson, M. Floater, M. Sabin (Eds.), *Advances in Multiresolution for Geometric Modelling*, Springer-Verlag, 2005, pp. 27–46.
- [80] D. King, J. Rossignac, A. Szymczak, Connectivity compression for irregular quadrilateral meshes. Technical Report TR-99-36, GVU, Georgia Tech, 1999.
- [81] M. Isenburg, J. Snoeyink, Face fixer: compressing polygon meshes with properties, in: *ACM SIGGRAPH*, 2000, pp. 263–270.
- [82] H. Lee, P. Alliez, M. Desbrun, Angle-analyzer: a triangle-quad mesh codec, in: *EUROGRAPHICS*, 2002, pp. 383–392.
- [83] M. Isenburg, Compressing polygon mesh connectivity with degree duality prediction, in: *Proceedings of the Graphics Interface*, May 2002, pp. 161–170.
- [84] A. Khodakovsky, P. Alliez, M. Desbrun, P. Schröder, Near-optimal connectivity encoding of 2-manifold polygon meshes, *Graphical Models* 64 (3) (2002) 147–168.
- [85] M. Isenburg, P. Alliez, Compressing polygon mesh geometry with parallelogram prediction, in: *IEEE Visualization*, 2002.
- [86] A. Szymczak, J. Rossignac, Grow & fold: compression of tetrahedral meshes, in: *Proceedings of the 5th Symposium on Solid Modeling and Applications*, ACM Press, 1999, pp. 54–64.
- [87] S. Gumhold, S. Guthe, W. Straßer, Tetrahedral mesh compression with the cut-border machine, in: *IEEE Visualization*, 1999, pp. 51–58.
- [88] R.B. Pajarola, J. Rossignac, A. Szymczak, Implant sprays: compression of progressive tetrahedral mesh connectivity, in: *IEEE Visualization*, San Francisco, 1999, pp. 299–306.
- [89] M. Isenburg, P. Alliez, Compressing hexahedral volume meshes, in: *Pacific Graphics*, 2002, pp. 284–293.
- [90] D. Saupe, J.-P. Kuska, Compression of isosurfaces for structured volumes, in: *Proceedings of Vision, Modeling and Visualization*, 2001, pp. 333–340.
- [91] W. Lorensen, H. Cline, Marching cubes: a high resolution 3d surface construction algorithm, *Comput. Graph.* 21 (4) (1987) 163–169.

- [92] D. Saupe, J.-P. Kuska, Compression of isosurfaces for structured volumes with context modelling, in: Proc. the First International Symposium on 3D Data Processing, Visualization, and Transmission, 2002, pp. 384–390.
- [93] X. Zhang, C. Bajaj, W. Blanke, D. Fussell, Scalable isosurface visualization of massive datasets on cots clusters, in: Proceedings of IEEE Symposium on Parallel and Large Data Visualization and Graphics, 2001, pp. 51–58.
- [94] S.-N. Yang, T.-S. Wu, Compressing isosurfaces generated with marching cubes, *Vis. Comput.* 18 (1) (2002) 54–67.
- [95] G. Taubin, Blic: bi-level isosurface compression, in: IEEE Visualization, 2002, pp. 451–458.
- [96] D. Laney, M. Bertram, M. Duchaineau, N. Max, Multiresolution distance volumes for progressive surface compression, in: Proc. the First International Symposium on 3D Data Processing, Visualization, and Transmission, 2002, pp. 470–479.
- [97] D.E. Breen, S. Mauch, R.T. Whitaker, 3d scan conversion of csg models into distance volumes, in: IEEE Symposium on Volume Visualization, 1998, pp. 7–14.
- [98] S. Mallat, A theory for multiresolution signal decomposition: the wavelet representation, *IEEE Trans. Pattern Anal. Mach. Intell.* 11 (7) (1989) 674–693.
- [99] H. Lee, M. Desbrun, P. Schröder, Progressive encoding of complex isosurfaces, in: ACM SIGGRAPH, 2003.
- [100] J. Lengyel, Compression of time dependent geometry, in: ACM 1999 Symposium on Interactive 3D Graphics, 1999.
- [101] J.-H. Ahn, C.-S. Kim, C.-C.J. Kuo, Y.-S. Ho, Motion compensated compression of 3D animation models, *IEE Electron. Lett.* 37 (24) (2001) 1445–1446.
- [102] J. Zhang, C.B. Owen, Octree-based animated geometry compression, in: Proceedings of IEEE Data Compression Conference, 2004, pp. 508–517.
- [103] J. Zhang, C.B. Owen, Hybrid coding for animated polygonal meshes: combining delta and octree, in: Proceedings of IEEE International Conference on Information Technology, 2005.
- [104] J.-H. Yang, C.-S. Kim, S.-U. Lee, Compression of 3D triangle mesh sequences based on vertex-wise motion vector prediction, *IEEE Trans. Circuits Syst. Video Technol.* 12 (12) (2002) 1178–1184.
- [105] L. Ibarria, J. Rossignac, Dynapack: space-time compression of the 3D animations of triangle meshes with fixed connectivity, in: SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2003, pp. 126–135.
- [106] M. Alexa, W. Müller, Representing animations by principal components, *Comput. Graph. Forum* 19 (3) (2000) 411–418.
- [107] Z. Karni, C. Gotsman, Compression of soft-body animation sequences, *Comput. Graph., Special Issue on Compression* 28 (1) (2004) 25–34.
- [108] H.M. Briceño, P.V. Sander, L. McMillan, S. Gortler, H. Hoppe, Geometry videos: a new representation for 3D animations, in: SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation, 2003, pp. 136–146.
- [109] I. Guskov, A. Khodakovsky, Wavelet compression of parametrically coherent mesh sequences, in: SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2004, pp. 183–192.
- [110] J.-H. Yang, C.-S. Kim, S.-U. Lee, Progressive compression of 3D dynamic sequences, in: Proc. ICIP, October 2004.
- [111] J.-H. Yang, C.-S. Kim, S.-U. Lee, Semi-regular representation and progressive compression of 3D dynamic mesh sequences, submitted to IEEE Trans. Image Processing, Nov. 2004 (in revision).
- [112] MPEG-4 animation framework extension (AFX). ISO/IEC JTC1/SC29/WG11, 2001.

**Jingliang Peng.** Jingliang Peng received the B.S. and M.S. degrees from Department of Computer Science and Technology, Peking University, China in 1997 and 2000, respectively. He is currently a Ph.D. candidate in Department of Electrical Engineering, University of Southern California. He is a Research Assistant of Integrated Media Systems Center and a member of the Interactive Media subgroup in Prof. C.-C. Jay Kuo's Multimedia Research Group. His research interests include 3D graphics data compression, point-based rendering, and image-based modelling.

**Chang-Su Kim.** Chang-Su Kim received the B.S. and M.S. degrees in Control and Instrumentation Engineering in 1994 and 1996, respectively, and the Ph.D. degree in Electrical Engineering in 2000, all from Seoul National University (SNU), Seoul, Korea. From 2000 to 2001, he was a Visiting Scholar with the Signal and Image Processing Institute, University of Southern California, Los Angeles, and a Consultant for InterVideo Inc., Los Angeles. From 2001 to 2003, he was a Postdoctoral Researcher with the School of Electrical Engineering, SNU. In August 2003, he joined the Department of Information Engineering, the Chinese University of Hong Kong as an Assistant Professor. His research topics include video and 3D graphics processing and multimedia communications. Dr. Kim has published more than 70 technical papers in international conferences and journals.

**C.-C. Jay Kuo.** Dr. C.-C. Jay Kuo received the B.S. degree from the National Taiwan University, Taipei, in 1980 and the M.S. and Ph.D. degrees from the Massachusetts Institute of Technology, Cambridge, in 1985 and 1987, respectively, all in Electrical Engineering. He is with the Department of Electrical Engineering, the Signal and Image Processing Institute (SIPI) and the Integrated Media Systems Center (IMSC) at the University of Southern California (USC) as Professor of Electrical Engineering and Mathematics. His research interests are in the areas of digital media processing, multimedia compression, communication and networking technologies, and embedded multimedia system design. Dr. Kuo is a fellow of IEEE and SPIE. He received the National Science Foundation Young Investigator Award (NYI) and Presidential Faculty Fellow (PFF) Award in 1992 and 1993, respectively. Dr. Kuo has guided about 60 students to their Ph.D. degrees and supervised 15 postdoctoral research fellows. Currently, his research group at USC consists around 40 Ph.D. students and 5 postdoctors (please visit website <http://viola.usc.edu>), which is one of the largest academic research groups in multimedia technologies. He is a co-author of about 100 journal papers, 600 conference papers, and 7 books. Dr. Kuo is Editor-in-Chief for the *Journal of Visual Communication and Image Representation*, and Editor for the *Journal of Information Science and Engineering* and the *EURASIP Journal of Applied Signal Processing*. He was on the Editorial Board of the *IEEE Signal Processing Magazine* in 2003–2004. He served as Associate Editor for *IEEE Transactions on Image Processing* in 1995–1998, *IEEE Transactions on Circuits and Systems for Video Technology* in 1995–1997, and *IEEE Transactions on Speech and Audio Processing* in 2001–2003.