Compression of Arbitrary Triangle Meshes With Attributes For Selective Refinement

Markus Grabner
Computer Graphics and Vision
Graz University of Technology
grabner@icg.tu-graz.ac.at

ABSTRACT

We present a method for compact encoding of attributes such as normal vectors defined on a triangle mesh of arbitrary topology suitable in the context of view-dependent simplification. It is based on the recently introduced CAME data structure (Compressed Adaptive Multiresolution Encoding). In accordance with the vertex hierarchy defined by mesh simplification, we define an *attribute hierarchy* to track attribute values within the multiresolution representation. It is shown that the *corner subgraph* describing the relations between attributes in a mesh simplification step can be efficiently encoded if attributes are properly ordered.

Keywords

Mesh compression, multiresolution, mesh attributes, non-manifold triangle meshes

1. INTRODUCTION

Due to significant improvements in data acquisition techniques and increasing requirements in modeling there are meshes available today that are far beyond the rendering capabilities even of high-end graphics workstations. However, since those models contain much more data than can fit on screen, we are often interested only in a small fraction of the whole data set at any time. Moreover, for any viewing conditions except a very close-up view the model can be replaced by a simpler approximation without decreasing image quality. This principle is called *view-dependent simplification*.

There are at least two other concepts besides mesh simplification that are useful in dealing with huge 3D objects. One is based on the observation that geometry information stored in a straightforward way contains a large amount of redundancy. By carefully exploiting coherence within the data set the size of the data can be reduced significantly (*mesh compression*). And second, if the simplified and compressed model is still too large to be displayed immediately after the user requested it, *progressive transmission* can be used to give the user a fast impression of how the model looks like.

A visualization method unifying view-dependent simplification, compression, and progressive transmission has recently been introduced (CAME [7]). Like most other view-dependent visualization algorithms, CAME doesn't take into account attributes defined at the object's surface. However, geometry alone is not sufficient to accurately represent an object. By tessellating a CAD model, information about regions building functional elements (*features*, see [12, 2]) is lost. Nevertheless, the appearance of the object can be preserved by explicitly providing surface attributes such as normal vectors. Under proper lighting conditions, discontinuities of the surface normals result in intensity discontinuities in the rendered image. This makes it easy for a human observer to distinguish individual features.

In this paper, we extend the CAME algorithm to support attribute data within its adaptive multiresolution framework. Our method cannot compete against optimized compression algorithms (such as [13, 1]) in terms of compression ratio. However, it is the first to take into account mesh attributes in the context of compression and view-dependent simplification of arbitrary triangle meshes. In Section 2, previous work in related fields of research is reviewed. Section 3 explains the new method in detail. The paper is finished by some conclusions in Section 4.

2. RELATED WORK

2.1 Multiresolution

The representation of geometric objects at different levels of detail has first been studied in [3] in the context of visible surface algorithms. The *Progressive Meshes* representation [10] allows a triangle mesh to be stored as a coarse *base mesh* together with a sequence of *detail records*. By iteratively applying *edge collapse* or *vertex split* operations (see Figure 1), the mesh can be retrieved at any desired level of detail. The *view-dependent refinement* extension introduced in [11] allows the user to "zoom in" into a particular region of the mesh. This avoids having to refine the whole mesh to the same level of detail as the region the user is interested in. The idea of a *simplification hierarchy* derived from the mesh simplification sequence has been independently presented in [15, 11].

Journal of WSCG, Vol.11, No.1., ISSN 1213-6972

WSCG'2003, February 3-7, 2003, Plzen, Czech Republic.

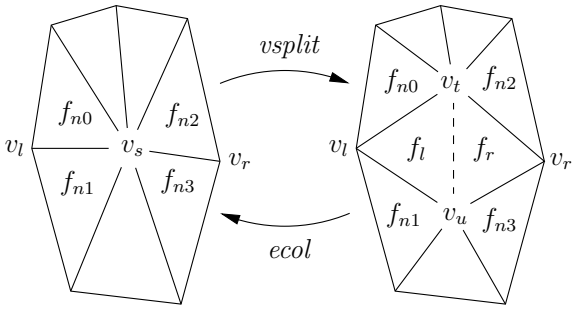


Figure 1: Basic operations for mesh simplification (edge collapse) and refinement (vertex split), the more general case of simplification doesn't require the vertices to be connected by an edge (pair contraction, indicated by dashed line)

For extremely large meshes it is not possible to load all mesh data into main memory for preprocessing and rendering. Therefore external memory techniques such as [5] have been developed. The considerable memory overhead introduced by this method has been addressed in the CAME framework [7]. Mesh vertices are referenced such that triangle adjacency relations are maintained implicitly. Each vertex is identified by the path to be taken in the simplification hierarchy from the root to the corresponding node.

2.2 Mesh compression

The field of mesh compression has been pioneered by Michael Deering [4]. He introduced the *generalized triangle mesh*, which allows explicit storage of vertices in a so-called *mesh buffer* and retrieval of these vertices for later use. Moreover, vertex locations and normal vectors are quantized and delta encoded.

A more efficient compression algorithm is the *edgebreaker* algorithm by Rossignac [13]. Starting at an arbitrary edge, the mesh is traversed in a spiral-like way. The topological relation of each triangle to previously processed parts of the mesh is encoded by less than two bits on average. For geometry compression, Taubin and Rossignac use a *vertex predictor* that takes into account the K most recently encoded vertices [14]. Instead of storing absolute vertex coordinates, only the *correction vector* between the predicted and the actual position of a vertex is encoded. Predictor parameters are estimated to minimize the least square error of all correction vectors. A valence-driven approach to mesh compression has been presented in [1].

2.3 Attribute maintenance

Feature boundaries, playing an important role in recognizing an object's structure, need to be preserved during mesh simplification. See for example Figure 2. While the model with simplified geometry (Figure 2(b)) still looks very much like the original (Figure 2(a), this is not the case for improperly simplified attributes (Figure 2(c)).

The Progressive Meshes method [10] supports this feature preservation requirement. However, it is restricted to a linear sequence between the original mesh and its coarsest approximation, inhibiting the use of view-dependent simplification. It has been shown in [8] how the feature preservation methods can be applied to view-dependent simplification.

The following definitions used in this paper are taken from [10, Section 2]. A *corner* is a (vertex, face) tuple. *Discrete* and *scalar* attributes are properties associated with faces and corners of the mesh, respectively. Curves on the surface across which the scalar attribute field is discontinuous are called *surface creases*.

3. OUR METHOD

The sequence of mesh simplification steps (i.e., pair contraction) provides a natural way to create a binary tree representing the simplification hierarchy of the vertices [15, 11]. By explicitly storing the paths through this hierarchy, triangles can easily update their vertices according to view-dependent simplification [7]. If scalar attributes are defined for the mesh, a similar procedure to update the attributes is required. However, since there are generally more attribute values than vertices (see Figure 2(a) for an example), the *attribute hierarchy* is more complex and more difficult to traverse. In this section a method is developed that creates the attribute hierarchy which can be traversed in parallel to the vertex hierarchy.

Although our current implementation only supports normal vectors as attributes, the same considerations can also be applied to different attribute types (e.g., texture coordinates and color). We therefore refer to the more generic term "attribute" instead of "normal vector" in the remainder of this paper.

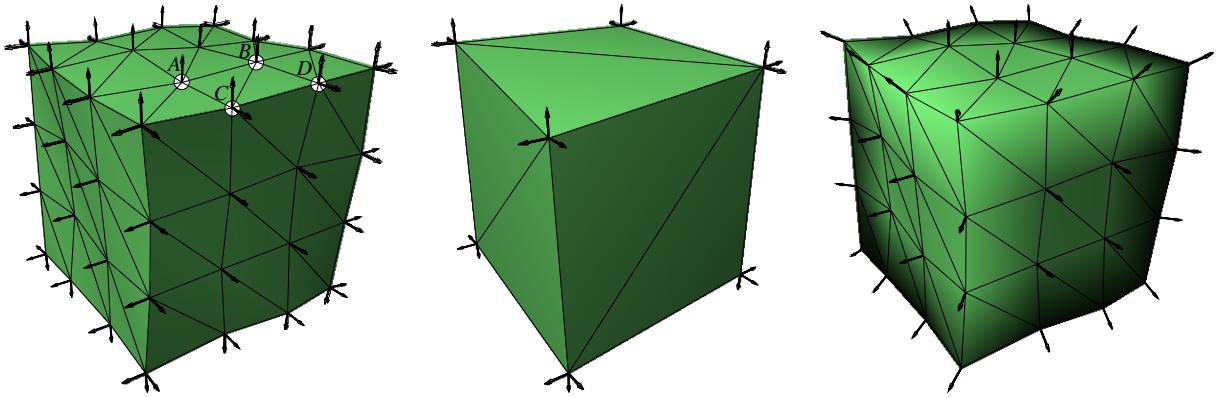
3.1 Edge types

In the following discussions we will frequently refer to attribute continuity properties of edges. We call an edge *smooth*, if the triangles adjacent to it share the same attributes at both vertices of the edge (such as the edge between A and B in Figure 2(a)). Similarly, we speak of a *sharp* edge if no attribute is shared across the edge (C and D in Figure 2(a)).

3.1.1 Semi-sharp edges

A situation deserving special considerations is illustrated in Figure 3, showing a cone approximated by ten triangles (not counting the base). If we want the surface to appear smooth, adjacent triangles should share normal vectors across the (smooth) edge by which they are connected. This can easily be done for vertices at the base[1] (see circle labeled "B" in Figure 3). However, if we try to apply the same procedure to the cone's apex, we would end up with a single normal vector shared by all triangles containing the apex. Therefore each triangle is assigned a different normal vector at the apex (circle "A" in Figure 3), resulting in edges that are sharp on one end (apex) and smooth on the other end (base). We call those edges *semi-sharp edges*.

[1]Note that the triangles of the cone's base (not shown here due to occlusion) are assigned a single down-facing normal vector, defining the chain of edges surrounding the base as a surface crease.



(a) original model (108 triangles, 56 vertices, 96 normal vectors)

(b) simplified geometry (12 triangles, 8 vertices, 24 normal vectors)

(c) simplified attributes (108 triangles, 56 vertices, 56 normal vectors)

Figure 2: Simplification of a cube-like object (with normal vectors indicated as arrows)

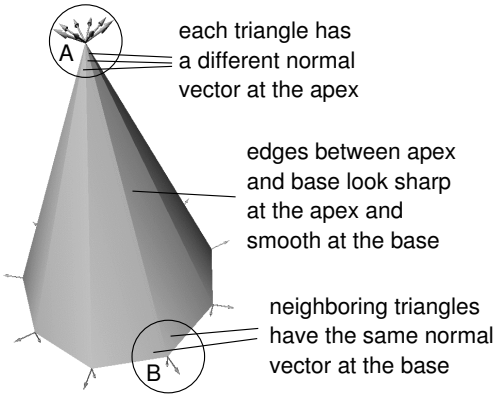


Figure 3: Semi-sharp edges appearing in an approximated cone

An important observation is that a non-boundary vertex in a two-manifold surface patch cannot belong to exactly one sharp edge while all other edges sharing this vertex are smooth. To prove this, we consider a vertex v belonging to n triangles $t_1 \dots t_n$ (in clockwise or counter-clockwise order) with attributes $a_1 \dots a_n$ at vertex v. Without loss of generality we assume the single sharp edge to be shared by the triangles t_1 and t_n, which implies $a_1 \neq a_n$ by the definition of sharp edges. Since all other edges are smooth, we find $a_i = a_{i+1}$ for all $i = 1 \dots n-1$. This immediately implies $a_1 = a_n$, in contradiction to $a_1 \neq a_n$ as required above. The proof for a non-manifold surface patch is similar (consider a "tree" of equality relations and a single inequality). Therefore a semi-sharp edge needs to be introduced if a discontinuity curve begins or ends within the surface.

3.2 Corner graph

Processing of normal vectors for view-dependent simplification has been studied in [8] for the special case of uncompressed two-manifold meshes. In this section, we present a method to encode attribute relations and discontinuity curves for meshes of arbitrary topology.

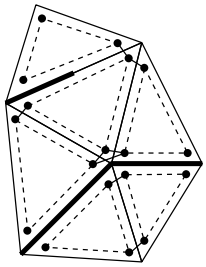


Figure 4: Corner graph

We introduce the *corner graph*, which is a representation of attribute discontinuities. The nodes of this graph are the triangle corners (solid circles in Figure 4). Two nodes are connected by an graph edge[2] if a "smooth path" of continuous attribute values (assuming proper interpolation, e.g. Phong's for normal vectors) on the surface exists between the corresponding corners visiting at most two triangles. More formal criteria for the existence of an edge are as follows:

- The corners represented by the two nodes belong to different triangles, but refer to the same vertex and the same attribute value (solid lines between circles in Figure 4).
- The corners belong to the same triangle, but refer to different vertices (dashed lines in Figure 4).

In a similar way as vertex references are stored with each pair contraction (see [7] for details), the portion of the corner graph affected by the operation has to be encoded. Although information-theoretically optimal encoding schemes

[2]Not to be confused with mesh edges!

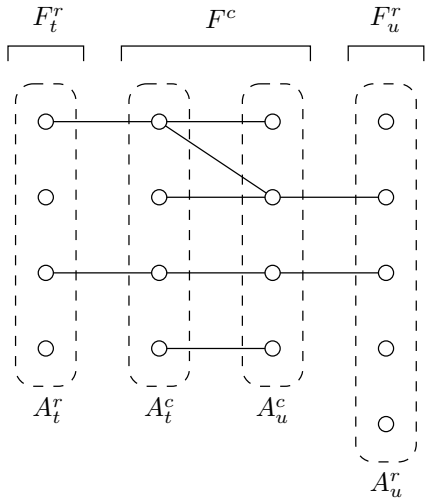


Figure 5: Corner subgraph

for graphs exist [9], we use prior knowledge about the graph to further reduce encoding size in most cases.

For clarity, we simplify the graph by replacing all nodes referring to the same vertex and attribute by a single node. Moreover, during pair contraction we only consider the actually affected subgraph. The resulting graph can be drawn with the nodes arranged in four columns (Figure 5) and will be referred to as the *corner subgraph* in the following. Details on the notation are given in Section 3.3.

3.3 Set definitions

Before we explain our method in detail, some definitions are given. The sets of faces adjacent to the vertices v_t and v_u (Figure 1) are denoted by F_t and F_u, respectively. The set of faces F^c being collapsed during pair contraction is defined as $F^c = F_t \cap F_u$. The faces adjacent to v_t and v_u remaining after pair contraction are $F_t^r = F_t - F^c$ and $F_u^r = F_u - F^c$, respectively.

Similarly, we denote the set of attributes at corners referring to v_t and v_u by A_t and A_u, respectively. Again, A_t^r and A_u^r are subsets of A_t and A_u, respectively, containing only attributes at corners of triangles remaining after pair contraction. Finally, A_t^c and A_u^c are subsets of A_t and A_u, respectively, containing only attributes at corners of triangles being collapsed during pair contraction. We also define $A = A_t \cup A_u$, $A^c = A_t^c \cup A_u^c$, and $A^r = A_t^r \cup A_u^r$. Note that $A_t^r \cap A_t^c$ and $A_u^r \cap A_u^c$ are not necessarily empty sets since the same attribute can be referred to by different corners. Actually, calculating these set intersections is an important part of our method since it allows to identify smooth regions of the mesh that need to be preserved during mesh simplification.

3.4 Attribute propagation

In this section, we describe a set of rules to propagate (and eventually merge) attributes during pair contractions. In Figure 6, six examples of pair contractions are presented. A completely smooth region is shown in Figure 6(a). Each of the vertices v_t an v_u (refer to Figure 1 for labeling of the vertices) is associated with exactly one attribute value (a_1 and a_2, respectively). After collapsing the edge, the new vertex v_s is assigned a new attribute a_3, the value of which is interpolated between the values of a_1 and a_2. The situation in Figure 6(b) is similar, however, there are two pairs of attributes (a_1, a_3 and a_2, a_4) which result in two separate attributes a_5 and a_6 at vertex v_s. In Figure 6(c), a_2 and a_3 are merged into a_5 similar to the previous examples. However, a_1 has no corresponding attribute to be merged with and is therefore simply copied to a_4. Figure 6(d) is again similar, but the triangles being collapsed are now completely enclosed by sharp edges. Therefore a_1 and a_4 are copied to a_5 and a_6, respectively, while a_2 and a_3 are removed from the mesh.

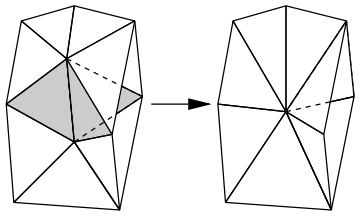


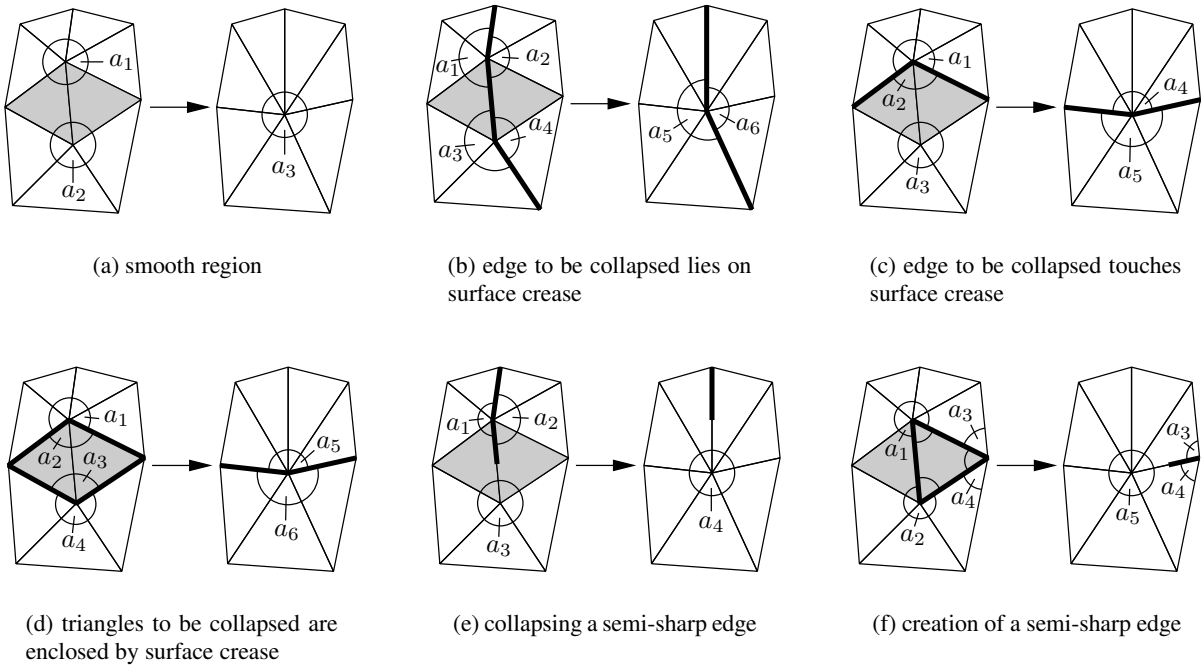
Figure 7: Collapsing a non-manifold edge

Semi-sharp edges are considered in Figures 6(e) and 6(f). First, we show how a semi-sharp edge is collapsed in Figure 6(e). All attributes a_1, a_2, and a_3 have to be merged to a single new attribute a_4. The reason to do so is the fact that an interior vertex cannot belong to a single sharp edge as shown in Section 3.1.1. If we would demand the sharp edge at v_t to extend to v_s after the pair contraction, a semi-sharp edge had to be introduced at v_s, continuing the discontinuity curve to a neighbor of v_s. Since such a neighbor might not exist without changing the appearance of the discontinuity curve, we cannot use this method, therefore v_s needs to be a smooth vertex with a single attribute a_4. Finally, Figure 6(f) illustrates how a semi-sharp edge is created as a result of contracting triangles only containing sharp and smooth edges. Attributes a_1 and a_2 need to be merged to a_5 for similar reasons as in 6(e), while a_3 and a_4 remain unchanged (since they are not in A), creating a semi-sharp edge. Merging more than two attributes into a single one imposes some difficulties in our binary-tree based simplification hierarchy. See Section 3.6 for a proposed solution.

Contractions of non-manifold edges (Figure 7) are also classified according to Figure 6. Since only the relations between the attribute sets A_t^r, A_t^c, A_u^c, and A_u^r are considered for encoding as explained in Section 3.5, no additional case distinctions are required for non-manifold situations. Note that the cases discussed above are by far not all possible ones since they can be combined in almost arbitrary ways.

3.4.1 Lifetime of attributes

An attribute might need to be further traced in the hierarchy even if it is no longer in use during mesh simplification. The example in Figure 8 (taken from [8] and slightly modified)



(a) smooth region

(b) edge to be collapsed lies on surface crease

(c) edge to be collapsed touches surface crease

(d) triangles to be collapsed are enclosed by surface crease

(e) collapsing a semi-sharp edge

(f) creation of a semi-sharp edge

Figure 6: Different cases of pair contractions, gray triangles are removed during pair contraction, sharp edges are drawn as thick lines, semi-sharp edges as thick lines extending only over one half of the edge

illustrates a case where an attribute a_5 is required in M^S due to selective refinement (which is legal [6]) without having ever occurred during simplification. Since a_1 is no longer present in the mesh after the edge between v_1 and v_2 has been collapsed to v_5, it is not immediately evident that a_5 (a copy of a_1) will be needed in M^S. However, simply leaving a_1 in the set of attributes associated with vertex v_3 (although it doesn't belong to any triangle in the mesh) ensures that a_1 is considered in further simplification steps. Such *blind attributes* are treated in the same way as members of $A^r - A^c$.

While the above procedure guarantees the existence of all required attribute values during selective refinement, it tends to accumulate attributes upwards the simplification hierarchy. In the worst case, the number of attributes associated with the root vertex of the hierarchy could be $O(n)$, where n is the number of vertices in the mesh. It is therefore necessary to determine if an attribute a is no longer needed (i.e., cannot play the role of a_5 in M^S of Figure 8). This is the case for each attribute a satisfying one of the following conditions:

- $a \in A^c - A^r$ (belonging to collapsed triangles but not to remaining ones, such as a_2 and a_3 in Figure 6(d)) since such an attribute disappears from the mesh after contracting the associated vertex pair (v_t, v_u) and remains inactive until the reverse vertex split is performed. Unlike a_1 in Figure 8, a cannot reappear at a vertex different from the one it belonged to before it was removed (v_t or v_u). Note that a_1 in Figure 8 is not a member of the set A of attributes considered during contraction of the pair (v_1, v_2).
- All vertices of all triangles initially containing a have collapsed to a single vertex. Unless this vertex is split again, no triangle in the mesh can contain a (or require an ancestor of a such as a_5 in M^S of Figure 8). This condition is easily checked by maintaining the set V_a of vertices of all triangles containing a and replacing the vertices in V_a by their parents after each pair contraction. If $|V_a| = 1$ (without duplicates), a can safely be removed from the mesh and doesn't need to be further considered.

3.5 Encoding

The corner subgraph (Section 3.2) and the fixed set of rules (Section 3.4) exactly represent the topology of discontinuity curves in a surface patch before and after pair contraction. Encoding this graph is therefore sufficient to be able to restore the relations between triangle corners and attributes. Once the corner subgraph is known, it can immediately be used to create the corresponding nodes of the attribute hierarchy.

3.5.1 Attribute permutation

Encoding the graph as it is introduces some redundancy since the permutation of the attributes would also get encoded. However, a permutation of attribute indices doesn't change the object's appearance as long as attribute values are permuted accordingly. We therefore try to reorder attributes such that no two edges cross each other in the corner subgraph (drawn with increasing attribute indices from top to bottom according to Figure 5). As we will see in Section 3.5.2, this allows encoding the graph with two bits per attribute instead of $O(\log |A|)$ if the permutation of the attributes in A is also

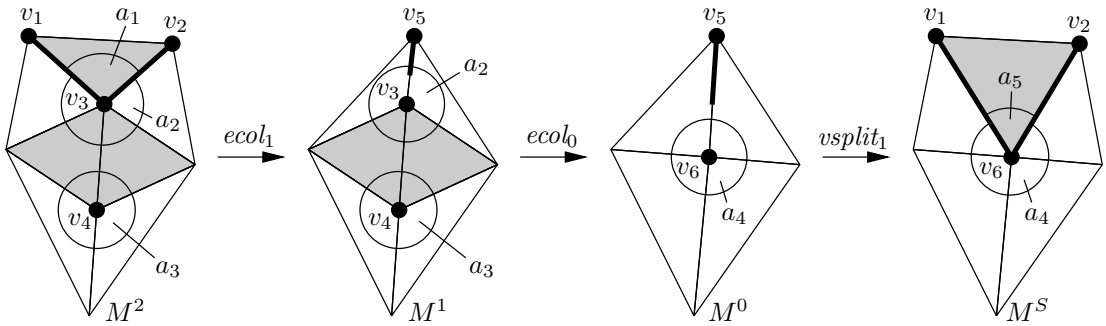


Figure 8: Propagation of attributes

encoded[3].

To get a non-crossing corner subgraph, attribute indices must be sorted in a consistent way at both vertices involved in a pair contraction. This is achieved by partitioning the input mesh into *smooth regions* and sorting attribute values according to their associated smooth region index. A smooth region of the mesh M is defined as a set R of triangles with the following properties:

- R is connected (or, more precisely, the graph containing a node for each triangle $t \in R$ and an edge for each pair of triangles $t_i, t_j \in R$ sharing a common mesh edge is connected).
- For any two triangles $t_i, t_j \in R$ that share a common edge e_k, t_1 and t_2 also share the same attribute values at both vertices of e_k.
- No triangle $t \in M - R$ exists such that $R \cup \{t\}$ satisfies the above criteria.

Consider the example in Figure 9, which illustrates a surface patch extending over two regions R_1 and R_2 separated by a chain of sharp edges (thick lines). If the attribute ordering is consistent with the region index ordering as in Figure 9(a), the corresponding corner graph in Figure 9(c) has no crossing edges. The inconsistent case is demonstrated in Figures 9(b) and 9(d). After pair contraction, the newly created attribute values are again sorted to satisfy the ordering constraint.

Is it always possible to find a consistent attribute ordering? Unfortunately the answer is no. A simple counter-example is the well-known Moebius strip. Figure 10 shows such an object with a slightly modified profile to introduce a surface crease, splitting the strip into two smooth regions. No matter how we try to fill the gap, we will always end up with an inconsistent setting as in Figure 9(b). However, the CAME algorithm [7] relies on simplifying the whole scene down to a single vertex, therefore we must also be able to handle this inconsistent situation. Since such a situation is very unlikely to occur, we employ a brute-force solution and explicitly store the uncompressed corner graph if no consistent ordering exists.

[3]This can easily be shown by using Sterling's formula $(n-1)! \approx \sqrt{2\pi/n} \cdot e^{-n} n^n$.

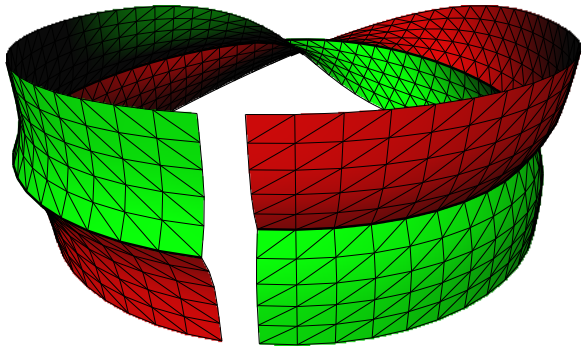


Figure 10: Moebius strip

3.5.2 *Attribute encoding/decoding*

During pair contraction, one of the following operations has to be performed on each attribute $a \in A$:

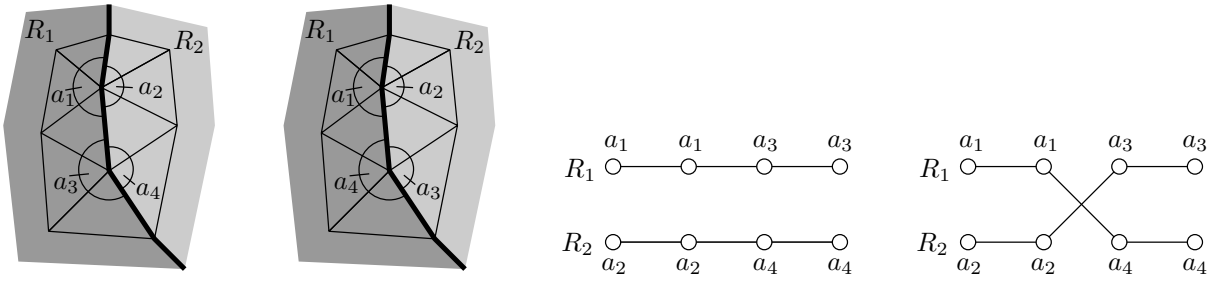
R: Remove the attribute from the mesh (such as a_2 and a_3 in Figure 6(d)) for one of the two reasons given in Section 3.4.1.

C: Copy the attribute to its parent node (such as a_1 in Figure 6(c), only for $a \in A^r - A^c$).

M: Merge the attribute with another one (such as a_1 and a_2 in Figure 6(a)).

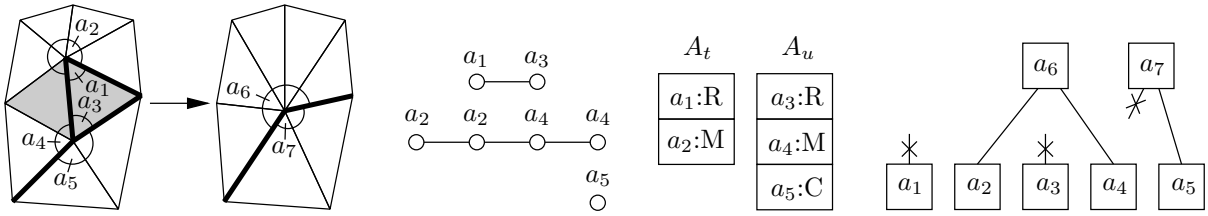
X: Merge more than two attributes (such as a_1, a_2, and a_3 in Figure 6(e), only for semi-sharp edges).

If a consistent attribute ordering as described in Section 3.5.1 can be found, the corner graph can be encoded simply by storing the type of operation to be performed on each attribute. This creates a stream of symbols, each two bits long to identify one of the four cases above. To restore the edge subgraph (and the attribute hierarchy), we simultaneously process the streams for A_t and A_u. Starting at either of the streams, we process case "R" by not assigning a parent node to the attribute and case "C" by assigning a parent node with the



(a) consistent (b) inconsistent (c) corner graph of 9(a) (d) corner graph of 9(b)

Figure 9: Consistent and inconsistent attribute ordering ($a_1 < a_2$ and $a_3 < a_4$ holds in both cases) and the corresponding corner graphs



(a) surface patch (containing creases) being simplified

(b) corner subgraph of Figure 11(a)

(c) encoding of Figure 11(b)

(d) attribute hierarchy reconstructed from the streams in Figure 11(c)

Figure 11: Corner subgraph encoding and decoding

current attribute as its only child. If an "M" is encountered, we also create a parent node, but then switch to the other stream and process it until we again receive and "M". Since the corner subgraph is free of edge crossings, both "M"-attributes correspond to each other (i.e., are connected by an edge in the subgraph). Both attributes are assigned the previously created node as a parent.

This procedure is demonstrated in Figure 11. The surface patch in Figure 11(a) contains both smooth and sharp edges and has a corresponding corner subgraph shown in Figure 11(b). Since the subgraph has consistent attribute ordering (no crossing edges), its attributes are classified as explained above, giving the streams "RM" and "RMC" for A_t and A_u, respectively (Figure 11(c)). The attribute hierarchy reconstructed from Figure 11(c) is given in 11(d). Note that Figure 11(d) is for illustration purposes only, in fact no absolute attribute indices are stored (see Section 3.6 for details).

The "X"-code is used to indicate that more than two attributes need to be merged. Due to the consistent attribute ordering we simply collect all subsequent "X"-attributes in A_t and A_u and assign a singe parent node to them. After an "X"-code appeared, the two-bit codes are replaced by a single bit per attribute to indicate if the sequence is to be continued (if not, the two-bit codes are used again).

The encoded corner subgraphs associated with each node of the simplification hierarchy are embedded into the CAME [7] data stream.

3.6 Attribute hierarchy traversal

Once the attribute hierarchy is reconstructed, it can be used to update attributes associated with triangle corners. Similar to updating vertices (see [7] for details), the attribute hierarchy is traversed upwards (i.e., towards the root) during pair contractions and downwards during vertex splits. In the latter case, the left branch is chosen if the new attribute belongs to v_t, and the right branch for v_u. However, in contrast to vertex coordinates, more than one attribute value can be associated with each vertex. Moreover, the number of attributes per vertex varies throughout the hierarchy since attributes are accumulated or become obsolete as explained in Section 3.4. Therefore the attribute hierarchy is used to guide the traversal of different attribute paths in parallel to the unique vertex paths.

Unlike indicated in Figure 11(d), there are no absolute indices of vertices or attributes in the CAME decoder. Instead, the set A_i of attribute values belonging to a vertex v_i is stored as a continuous array, being accessed by indices $0 \dots |A_i| - 1$. These indices relative to the attribute storage of v_i are generally not valid within any other vertex than v_i. The correct mapping of indices between nodes up and down the

hierarchy is given by the attribute hierarchy stored with each simplification node. To keep track of the correct attributes, this relative attribute index is stored with each triangle corner in addition to the current position on the vertex path [7].

Some additional encoding is required for semi-sharp edges since they require merging of more than two attributes (see Section 3.4), which cannot be represented in a binary tree. Pair contractions are handled as before since many attributes can refer to the same node as a parent. However, to unambiguously identify the branch to descend during a vertex split, an additional code of $\lceil \log_2 m \rceil$ bits must be provided for m attributes sharing the same parent (e.g., $m = 2$ for a_1 and a_2 in Figure 6(e)).

Up to now, only the topological relations between attributes have been considered. The actual attribute values are also stored in the CAME data stream, in our case using the quantization method of [4] to encode normal vectors.

4. CONCLUSIONS AND FUTURE WORK

We presented a method to carry information about mesh attributes (e.g., normal vectors) and their relations to each other through the simplification and adaptive refinement of triangle meshes of arbitrary topology. It has been shown how the graph describing smooth paths on the surface patch under consideration can be transformed into nodes of the attribute hierarchy which is used during traversal of the multiresolution data to update the triangles' attributes. We also presented an ordering scheme for attributes that prevents their permutations from being encoded, thus reducing code size.

Currently normal vectors are encoded with 17 bits according to the method of [4]. However, spatial and hierarchical coherence could be used to improve compression of attribute values. Even the roughly known surface orientation (eigenvectors of each meta-node's edge vector covariance matrix [7]) might be useful for normal vector encoding.

5. ACKNOWLEDGMENTS

This work has in part been funded by the European Union under contract no. IST-1999-20273. Thanks to Konrad Schindler for fruitful discussions.

6. REFERENCES

[1] P. Alliez and M. Desbrun. Valence-driven connectivity encoding for 3D meshes. In A. Chalmers and T.-M. Rhyne, editors, *Proceedings Eurographics*, volume 20 of *Computer Graphics Forum*. Blackwell Publishers, Sept. 2001. ISSN 1067-7055.

[2] W. F. Bronsvoort and F. W. Jansen. Feature modelling and conversion – key concepts to concurrent engineering. *Computers in Industry*, 21(1):61–86, 1993.

[3] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, Oct. 1976. ISSN 0001-0782.

[4] M. F. Deering. Geometry compression. In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 13–20. ACM SIGGRAPH, Addison Wesley, Aug. 1995.

[5] J. El-Sana and Y.-J. Chiang. External memory view-dependent simplification. In *Proceedings Eurographics*, volume 19 of *Computer Graphics Forum*, pages 139–150. Blackwell Publishers, Aug. 2000. ISSN 1067-7055.

[6] M. Grabner. Consistency of the VDPM framework. In B. Falcidieno, editor, *Proceedings of SCCG 2000*, pages 147–155. Comenius University, Bratislava, May 2000. ISBN 80-223-1486-2.

[7] M. Grabner. Compressed adaptive multiresolution encoding. *Journal of WSCG*, 10(1):195–202, Feb. 2002. ISSN 1213-6972.

[8] M. Grabner. Feature preservation in view-dependent multiresolution meshes. In A. Chalmers, editor, *Proceedings of SCCG 2002*, pages 153–162. Comenius University, Bratislava, Apr. 2002. ISBN 80-223-1730-6.

[9] X. He, M.-Y. Kao, and H.-I. Lu. A fast general methodology for information-theoretically optimal encodings of graphs. *SIAM Journal on Computing*, 30(3):838–846, June 2001.

[10] H. Hoppe. Progressive meshes. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 99–108. ACM SIGGRAPH, Addison Wesley, Aug. 1996. ISSN 0097-8930.

[11] H. Hoppe. View-dependent refinement of progressive meshes. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 189–198. ACM SIGGRAPH, Addison Wesley, Aug. 1997. ISBN 0-89791-896-7.

[12] C. McMahon and J. Browne. *CAD/CAM: principles, practice and manufacturing management*. Addison Wesley, second edition, 1998. ISBN 0-201-17819-2.

[13] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1), 1999.

[14] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, Apr. 1998.

[15] J. C. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. In *IEEE Visualization '96*. IEEE, Oct. 1996. ISBN 0-89791-864-9.