Edgebreaker: A Simple Compression for Surfaces with Handles

Jarek Rossignac♥, Hélio Lopes♠, Alla Safanova♦, Geovan Tavares♠, Andrzej Szymczak♥

♥ Georgia Institute of Technology, College of Computing and GVU Center, Atlanta, GA. USA

♠ Pontifical Catholic University (PUC-Rio), Department of Mathematics, Rio de Janeiro, RJ, Brazil

♦ Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, USA

{jarek@cc.gatech.edu, lopes@mat.puc-rio.br, asafanov@cs.cmu.edu, geovan@mat.puc-rio.br, andrzej@cc.gatech.edu}

ABSTRACT

The Edgebreaker is an efficient scheme for compressing triangulated surfaces. A surprisingly simple implementation of Edgebreaker has been proposed for surfaces homeomorphic to a sphere. It uses the Corner-Table data structure, which represents the connectivity of a triangulated surface by two tables of integers, and encodes them with less than 2 bits per triangle. We extend this simple formulation to deal with triangulated surfaces with handles and present the detailed pseudocode for the encoding and decoding algorithms (which take one page each). We justify the validity of the proposed approach using the mathematical formulation of the Handlebody theory for surfaces, which explains the topological changes that occur when two boundary edges of a portion of a surface are identified.

Keywords

Triangle meshes; Connectivity Graph; 3D Compression; Handlebody Theory.

1. INTRODUCTION

The Edgebreaker compression and decompression algorithms [13] may be used to encode the connectivity of any simply connected manifold triangle mesh with a guaranteed worst case code of 1.80 bits per triangle [4]. In practice, the Edgebreaker encoding may often be further compressed to less than one bit per triangle through the use of Entropy codes [14]. But the true value of Edgebreaker lies in the efficiency and surprising simplicity of the algorithms [15], which fit in a couple of pages and use only a few arrays of integers as sole data structure. Because of its simplicity, Edgebreaker is viewed as the emerging standard for 3D compression [17] and may provide an alternative for the current MPEG-4 standard that is based on Rossignac's previous work with Gabriel Taubin [19]. This simple implementation of Edgebreaker, as a state machine, is publicly available through the web [3] and has been recently enhanced to support meshes with an arbitrary number of handles (also called through-holes). We provide in this paper a detailed description of this extension, imbedding it in a theoretic setting of the Handlebody theory, and including formal proofs.

An important topological property of boundary representations (abbreviated B-Reps) [1] is the Euler-Poincaré formula, dated from the turn of the century [11], which states that an orientable connected triangulated surface S without boundary is uniquely identified by its Euler characteristic $\chi(S)=|V|-|E|+|F|$, where $|V|$,$|E|$ and $|F|$ indicate respectively the number of vertices, edges and faces of S. The Euler characteristic classifies S according to the Euler formula that is $\chi(S)= 2 - 2g$ (where g is the genus of the surface, or in other words, the number of through-holes). From this two equations, the genus of a connected triangulated surface without boundary may be expressed as $g = 1 - (|V|-|E|+|F|)/2$.

The Handlebody theory [10,16] refines the traditional Euler-Poincaré theory [11] by bringing several new topological invariant for n-dimensional smooth manifolds. Its fundamental problem is to study the topological changes generated by handle attachments to manifolds with boundary.

The paper is organized as follows. Section 2 describes the Handlebody Theory. Section 3 presents some definitions of combinatorial topology to precisely establish our notation. Section 4 presents the Handle Operators for triangulated surfaces, which consists in a set of topological operators to build and un-build surfaces with or without boundary. Section 5 describes the Corner-Table data structure. Section 6 and 7 introduce, respectively, the Edgebreaker simplified compression and decompression algorithm for surfaces with handles.

2. HANDLEBODY THEORY

In classical Handlebody theory, the object of study is taken to be a compact n-dimensional manifold with or without boundary. The main purpose of this section is to give a brief introduction to this theory, in particular for the 2-dimensional case.

A set $V \subset R^m$ is an *n-dimensional manifold with boundary* if each point in V has an open neighborhood homeomorphic to either R^n or R^n_+.

Let D^i be the i-dimensional disk. The boundary of a set P is denoted by ∂P. Notice that D^0, D^1, and D^2 correspond, respectively, to a point, to a line, and to a disk. Moreover, the set ∂D^0 is an empty set, the set ∂D^1 consists in two points, and the set ∂D^2 is a circle. Let R and S be two topological spaces, then $R \times S$ corresponds to the set obtained by the usual Cartesian product, which means that $R \times S$ is the set of all pairs elements (r,s) such that $r \in R$ and $s \in S$.

There are three types of handles for 2-manifold and they will be distinguished by an index λ that varies from *0* to *2*.

Definition 2.1: *A handle of index λ, denoted by $\boldsymbol{H}_\lambda$, is a pair of topological spaces (A_λ , B_λ) such that $B_\lambda \subseteq \partial A_\lambda$, $A_\lambda = D^\lambda \times D^{2-\lambda}$ and $B_\lambda = \partial D^\lambda \times D^{2-\lambda}$.*

According to this definition, one can observe that for $\lambda=0$, the set $A_0=D^0 \times D^2$ is a 2-disk and $B_0=\partial D^0 \times D^2$ is the empty space, since ∂D^0 is $\varnothing$. The set $A_1=D^1 \times D^1$ is a square (Cartesian product of two intervals) and $B_1=\partial D^1 \times D^1$ is defined to be two opposite sides on the boundary of A_1, since ∂D^1 is composed by two

points and D^1 is an interval. Finally, in the case where $\lambda=2$, the set $A_2=D^2\times D^0$ is a 2-disk and $B_2=\partial D^2\times D^0$ is the circle that is exactly the boundary of A_2.

To attach a handle $\boldsymbol{H}_\lambda=(A_\lambda, B_\lambda)$ to the boundary of a 2-manifold S means to identify by a homeomorphism the set $B_\lambda\subseteq \partial A_\lambda$ with a subset I contained in the boundary of S.

The next theorem is the main mathematical tool in which this work is based [16].

Theorem 2.2 (Handlebody Decomposition): *For every manifold S there is a finite sequence of surfaces $\{S_i\}_{i=1..N}$ such that $S_0=\varnothing$, $S_N=S$ and the manifold S_i is obtained by attaching a handle $\boldsymbol{H}_\lambda=(A_\lambda,B_\lambda)$ to the boundary of S_{i-1}. This sequence is called the Handlebody Decomposition of S.*

Figure 1 illustrates the handlebody decomposition of a torus.

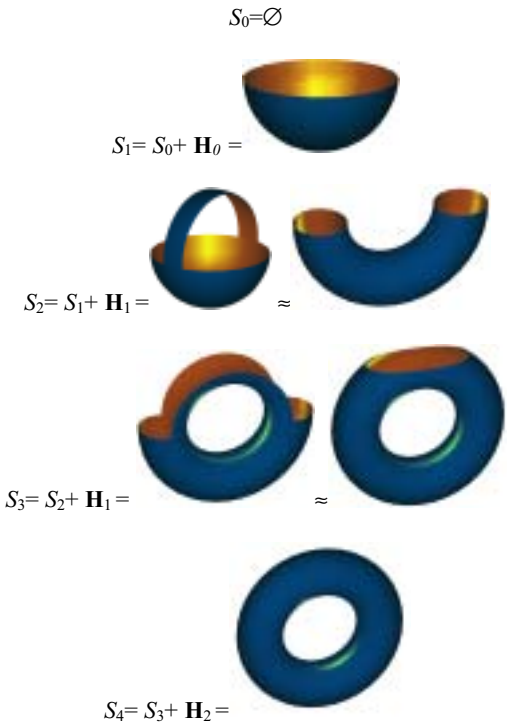


Figure 1: Handlebody decomposition of a torus.

Handles can be attached to an orientable 2-manifold with boundary in such a way to preserve its orientability, i.e., the identification has to be coherent. If one starts with an orientable 2-manifold, then after attaching a handle it is again orientable, that is, it doesn't contain a Moebius strip [16]. The test to know if the handle is attached coherently is simple: check the number of boundary curves, if this number is changed, then it is coherent; if else, a Moebius strip has been added to the previous manifold.

When a handle $\mathbf{H}_\lambda$ is attached coherently to the boundary of S_{i-1} to obtain S_i, a topological change is generated and such change depends only on the index λ.

The topological change generated by a handle attachment of index 0 is a creation of a new 2-manifold component (see S_1 in Figure 1).

When the handle $\mathbf{H}_1$ is coherently attached to a 2-manifold, three situations can occur:

1. The set B_1 may be attached to disjoint intervals on the same boundary curve component. In this case, the topological change is an inclusion of a new boundary curve component (see S_2 in Figure 1).
2. The set B_1 may be attached to intervals on different boundary curve components of a 2-manifold component. The topological change is here characterized by increase of the genus. In addition, the number of boundary curve components decreases (see S_3 in Figure 1).
3. The set B_1 may be attached to intervals on different surface components. Here, one boundary curve component and one surface component are removed.

Handles of index 2 close a boundary curve component (see S_4 in Figure 1).

Thus, there are five different situations in which a handle can be attached coherently to the boundary of a 2-manifold.

The Edgebreaker encoding/decoding algorithms for triangulated surfaces with or without genus also defines a sequence of topological changes that the surface undergoes during the reconstruction process. In this work, the relation of these algorithm with the handlebody decomposition will be clarified. But first, some definitions of combinatorial topology must be invoked to precisely establish our notation.

3. BASIC CONCEPTS

A *simplex* σ^p of dimension p (*p-simplex*, for short) is the convex hull of $p+1$ linearly independent points in R^m, called its *vertices*. A simplex σ^k is a face of a simplex τ^p, $k \le p$, when each vertex of σ is a vertex of τ. This relation is denoted by $\sigma < \tau$. The empty simplex is a simplex.

An *n-dimensional simplicial complex* K is a finite collection of i-dimensional simplexes (i = 0,...,n) in R^m, under the following conditions:

1. If $\sigma \in$ K and $\tau < \sigma$ then $\tau \in$ K
2. If σ and $\tau \in$ K then $(\sigma \cap \tau) < \sigma$ and $(\sigma \cap \tau) < \tau$.

The *underlying polyhedron* $|K| \subset R^m$ corresponds to the union of all points contained in a simplex of K. If a collection of simplexes $L \subset$ K is a simplicial complex, then it is called a *subcomplex* of K.

Two k-simplexes σ and $\tau \in$ K are *adjacent* when $\sigma \cap \tau \neq \varnothing$. If ξ is a face of a simplex τ then they are said to be *incident* to each other.

A complex K is *connected* if it cannot be represented as a union of two non-empty disjoint subcomplexes L and M without common simplexes. A component of a simplicial complex K is a connected subcomplex that is not contained in a larger subcomplex of K.

The *star* of a vertex v is a subcomplex of K composed by the union of simplexes that are incident to v, and is denoted by *star*(v). The *link* of a vertex v, denoted by *link*(v), is the frontier of *star*(v). The open star of a vertex v is the set *star*(v)-*link*(v).

An *n*-dimensional simplicial complex M, $|M| \subset R^m$, is a *combinatorial n-manifold with boundary* if the following two conditions are satisfied:

1. A $(n-1)$-simplex in M is incident to one or two *n*-simplexes of M.

2. The open star of a vertex in M is homeomorphic to an open subset of either R^n or R^n_+.

The subcomplex formed by the $(n-1)$-simplexes in a combinatorial n-manifold M incident to only one n-simplex is called the boundary of M and is denoted by ∂M. The simplexes of M that belongs to ∂M are called boundary simplexes otherwise they are called interior cells. The boundary of a combinatorial n-manifold is a combinatorial $(n-1)$-manifold without boundary.

A combinatorial n-manifold is orientable when it is possible to choose a coherent orientation on its n-simplexes, i.e., two adjacent n-simplexes induce opposite orientations on their common $(n-1)$-simplexes.

From now on, a *surface* and a *curve* will always mean, respectively, a 2 and 1 dimensional connected oriented combinatorial manifold with or without boundary. The set of 2,1 and 0 dimensional simplexes of a surface S will be called, respectively, triangles, edges and vertices and they are denoted by $T(S)$, $E(S)$ and $V(S)$.

The *Triangle-Edge connectivity graph of a surface S*, denoted by $G_{TE}(S) = (N,L)$, is defined to be the graph whose nodes and lines are, respectively, their triangles and edges. Let $N(G_{TE})$ and $L(G_{TE})$ be the set of nodes and lines of G_{TE}. Then, two one-to-one functions n_{TE}: $T(S) \rightarrow N(G_{TE})$ and l_{TE}: $E(S) \rightarrow L(G_{TE})$ are defined in such a way that each edge $e \in E(S)$ that is incident to the triangles u and v corresponds to a line $l \in L(G_{TE})$ that connects the nodes n_{TE} (u) and n_{TE} (v).

The *Vertex-Edge connectivity graph of a surface S*, denoted by $G_{VE}(S)$, is defined to be the graph whose nodes and lines are, respectively, the surface vertices and edges. Two other one-to-one functions, named n_{VE}: $V(S) \rightarrow N(G_{VE})$ and l_{VE}: $E(S) \rightarrow L(G_{VE})$, are defined in such a way that each edge $e \in E(S)$ that is incident to the vertices u and v corresponds to a line $l \in L(G_{VE})$ that connects the nodes n_{VE} (u) and n_{VE} (v). Here, $N(G_{VE})$ and $L(G_{VE})$ denote the set of nodes and lines of G_{VE}.

4. HANDLE OPERATORS

Section 2 described the Handlebody theory for 2-manifolds. In this section, we introduce a set of operators that allows the implementation of the handlebody decomposition for triangulated surfaces (for more details, see [9]).

Given a surface S with or without boundary, we would like to construct a finite sequence of combinatorial surfaces $\{S_i\}_{i=0...n}$, in which $S_0=\varnothing$ and $S_n=S$. To build such sequence, we propose a set of operators, called Handle operators, and we study the topological changes caused by their actions.

In a combinatorial point of view, three types of operators are now be defined to represent the handle attachments.

4.1 Handle Operator of type 0

This operator creates a new surface component with only one triangle (see Figure 2).

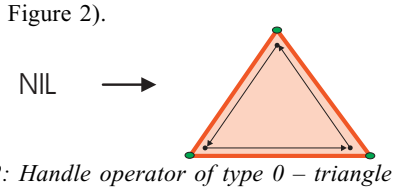


Figure 2: Handle operator of type 0 – triangle creator.

4.2 Handle Operators of type 1

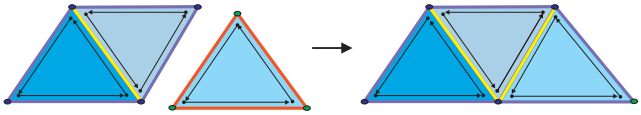
The purpose of this operator is to identify two given boundary edges with no vertices in common. There are three situations for this group. They are distinguished according to the answer of the following questions:

- Are those edges on the same surface ?

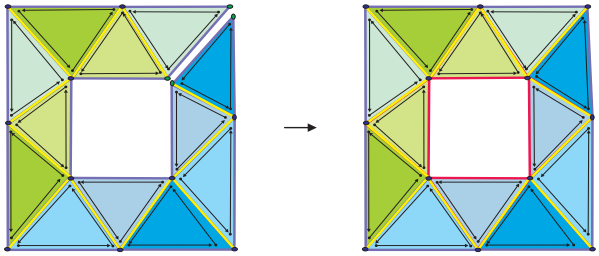
If not, the handle operator will attach the two different surfaces and remove one boundary curve component (see Figure 3(a)). Otherwise, the next question will identify the remaining two cases.

- Are those edges on the same boundary curve component ?

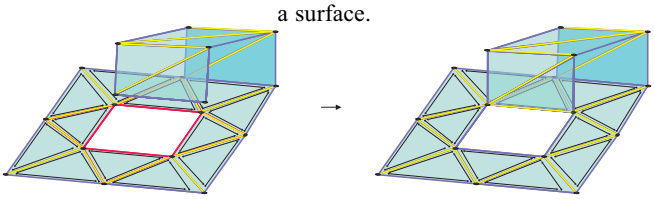
If so, then the operator will split the boundary curve into two different components (see Figure 3(b)). Otherwise, it will increase the number of genus on the surface and reduce in one the number of boundary curve components of the surface (see figure 3(c)). Interior edges are drawn in yellow, and different boundary curves have different colors.



(a) The boundary edges belongs to different surfaces.



(b) The boundary edges belong to the same boundary curve of a surface.

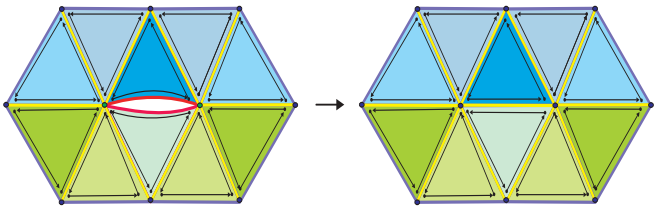


(c) The boundary edges belong to different boundary curves of a surface.

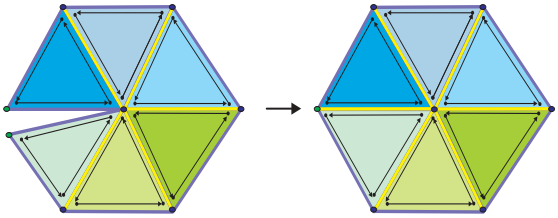
Figure 3: Handle operators of type 1.

4.3 Handle Operators of type 2

In this group there are two operators used to identify two given boundary edges with vertices in common, which could be one or two. Those operators will be used only in the Zipping part of the Edgebreaker decompression algorithm. When they have two vertices in common, the operator closes one boundary curve component and transform those boundary vertices into two interior vertices (see Figure 4(a)). Otherwise, it zips the two boundary edge and generates one interior vertex on the surface (see Figure 4(b)).



(a) The boundary edges have two vertices in common.



(b) The boundary edges have one vertex in common.

Figure 4: Handle operators of type 2.

4.4 Inverse Handle Operators

The inverse operators are naturally defined by exchanging the direction of the arrows in Figures 2, 3 and 4.

The inverse action of a Handle Operator of type 0 is the destruction of a triangle.

The direct Handle Operators of type 1 and 2 identify two boundary edges to create an interior edge. On the other way round, an inverse Handle Operator splits an interior edge into two boundary edges.

The inverse Handle operators of type 2 are distinguished according to the number of interior vertices incident to a given interior edge. If the edge to be operated has two interior vertex then it creates a new boundary curve component on the surface (invert the arrow in Figure 4(a)). If the edge has only one incident interior vertex, then the operator adds this vertex to the boundary (invert the arrow in Figure 4(b)).

Inverse Handle operators of type 1 are applied when its two incident vertices are on the boundary. If those vertices are on different boundary curve components then it joins the two boundary curves (invert arrow in Figure 3(b)). Otherwise, the two incident vertices are on the same boundary curve and in this case the boundary curve is separated in two components and after the edge split either the surface disconnects (invert arrow of Figure 3(c)), or a genus is removed (invert arrow in Figure 3(c)).

5. CORNER-TABLE DATA STRUCTURE

The purpose of this section is to describe the data used in the Edgebreaker compression and decompression algorithms.

The Corner-Table (CT) is a very compact data structure for triangular surfaces, introduced by Rossignac, Safonova and Szymczak in [15]. It uses the concept of a *corner* that represents the association of a triangle with one of its incident vertices. In the CT data structure, the corners, the vertices and the triangles are indexed by non-negative integers.

The frontier of a triangle is implicitly represented by an oriented cycle of three corners, whose identification is given by three consecutive indices. By definition, corners with indexes 0, 1 and 2 correspond to the first triangle frontier, the corners of indexes 3,4 and 5 correspond to the second triangle frontier and so on. As a consequence, a corner with index *c* is associated with the triangle of index c.t = (*c* DIV 3).

In a triangulated surface, every corner has a vertex and an opposite corner associated to it. This information is stored in two integer arrays, called the V and O tables. The dimension of both tables is three times the number of faces, which corresponds to the number of corners on the surface.

The notation c.v, which is a short for V[c], corresponds to the value stored in the V table whose entry is c and returns the id of the vertex associated with corner c.

Assuming that a counter-clockwise orientation has been opted for the surface, then for a given corner c, the next (c.n) and previous (c.p) corners on its triangle frontier cycle are obtained by the use of the following expressions: c.n = 3×c.t + (c+1) MOD 3, and c.p = 3×c.t + (c+2) MOD 3.

The notation c.o, stands for O[c] and returns the id of the corner opposite to c. To be precise, c.o is the only integer b for which: c.n.v = b.p.v and c.p.v = b.n.v.

In practice, the O table needs not be stored, because it may be easily derived in linear time from the V table using a hashing sort of triples (min(c.n.v, c.p.v), max(c.n.v, c.p.v), c) for all corners c. Entries that correspond to opposite corners are consecutive in the stored list. For convenience, the left and right corners c.l and c.r are, respectively, defined as c.p.o and c.n.o. All corner relations and notations are illustrated in Figure 5.

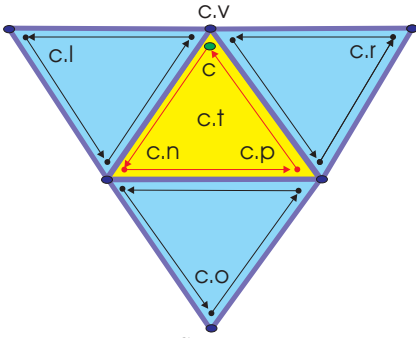
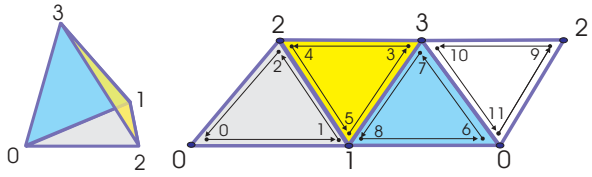


Figure 5: Corner notations.

The coordinates of the vertex with index *i* are stored in the row *i* of a matrix, called G (for geometry). The dimension of G is |V|×N, where |V| is the number of vertices and N is the dimension of the space where the surface is embedded. Usually N=3.The notation c.v.g is used to access the geometry of the vertex and stands for G[V[c]].

To illustrate the data structure tables consider the tetrahedron in Figure 6(a). Table 1 is an example of a CT data structure for this example considering the corners and vertices indices in Figure 6(b).



(a) tetrahedron (b) vertex and corner indices

Figure 6: Tetrahedron example.

Corner	O Table	V Table
0	3	0
1	10	1
2	7	2
3	0	3
4	6	2
5	11	1

6	4	0
7	2	3
8	9	1
9	8	2
10	1	3
11	5	0

Table 1: Example of a CT data structure for a tetrahedron.

6. EDGEBREAKER COMPRESSION

The Edgebreaker is a state machine that encodes a combinatorial surface without boundary *M*. At each state, a decision is made to move from a triangle Y to an adjacent triangle X. To perform this decision, all visited triangles and their incident vertices are marked.

Let Left and Right denote the other two triangles that are incident upon X. Let **v** be the vertex common to X, Left, and Right. Five situations are distinguished according to the Table 2. Those cases are denoted by the letters C,L,E,R and S (arranged for the mnemonic "Claire's"). The arrow in Figure 7 indicates the direction to the next triangle. Previously visited triangles are filled. The edge that crosses from Y to X will be called the gate.

	C	L	E	R	S
Vertex **v**	Not Visited	Visited	Visited	Visited	Visited
Left Tr\iangle	Not Visited	Visited	Visited	Not Visited	Not Visited
Right Triangle	Not Visited	Not Visited	Visited	Visited	Not Visited

Table 2: Edgebreaker state machine.

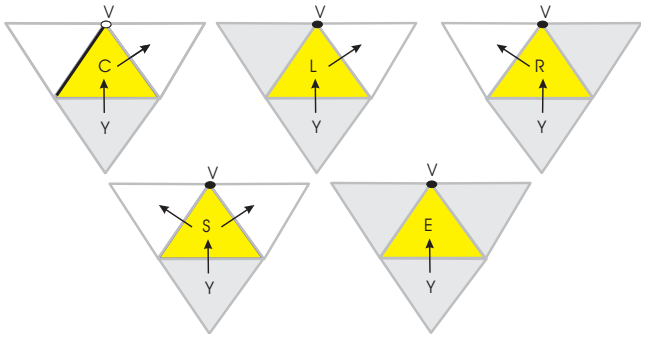


Figure 7: Edgebreaker clers cases.

6.1 The Algorithm

The compression algorithm (see Appendix 1) is composed of an initialization *InitCompression* followed by a call to *Compress*. The initial corner c may be chosen randomly. The initialization encodes and marks the three vertices of the first triangle, marks the triangle as visited, assigns an Id for each corner and calls *Compress*. The "visited" mark flags for triangles and vertices are, respectively, stored in the arrays U and M.

Compress is a recursive procedure that traverses the surface *M* in a spiraling spanning tree of triangles. The recursion starts only at triangles that are of type S and compresses the branch adjacent to the right edge of such a triangle. When the corresponding E triangle is reached, the branch traversal is complete and the routine returns from the recursion to pursue the left branch. At this point, two situations are distinguished. If the Left triangle has not been visited during the right branch traversal (case of normal S), we move to the left neighbor and continue our encoding of the left branch. Otherwise (case of a handle S) the pair of opposite corners separated by the left edge of the S triangle are stored in the stream or file called *handle* and the routine returns. These edges will have to be attached together during decompression to form handles. The encounter of an E that does not match an S terminates the compression process.

For C, L, E and R triangles, an U array entry stores the flag that indicates whether a triangle has been visited (1) or not (0). Since any S triangle is a potential handle S, its U entry stores an integer that corresponds to the corner id that will be assigned to the c.p corner in the decompression algorithm. The value of this integer may be later saved on the *handle* stream.

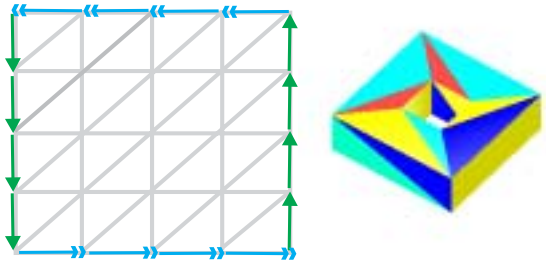
The *Compress* routine starts testing for handle attachment, i.e., whether the triangles to the left and to the right are adjacent to a previously visited S triangle. If so, it saves in the *handle* stream the pair of opposite corners.

Next, if the tip vertex of a new triangles has not yet been visited ("IF c.v.m != 1"), we are on a C triangle and we encode the information necessary for decoding the location of that vertex. Typically, this information is a residue vertex that the decoder will add to an estimation of c.v.g computed from neighboring vertices. Here, we are using the simplest Edgebreaker code, we encode a 0 in the *clers* string to indicate a C triangle and the vertex coordinates are directly stored in the *vertices* stream or file.

When the tip of the new triangle has been visited, we distinguish four other cases, based on the status of the neighboring (left and right) triangles. They correspond to the labels L, R, E and S as indicated in Table 2. To test whether the left and right adjacent triangles have been visited we use, respectively, the commands ("IF c.l.t.u > 0") and ("IF c.r.t.u > 0"). The L,R,E and S labels are each encoded using 3 bits. For so, the following convention are defined: 110 for L, 111 for E, 101 for R, and 100 for S (for more details, see [14]).

6.2 The torus example

To illustrate the algorithm, consider the surface given by the model for a triangulated torus illustrated in Figure 8(a). Identifying the edges indicated by the arrows on the opposite sides of the rectangle, one can build a simplicial complex in R^3 whose polyhedron is homeomorphic to the torus (see Figure 8(b)).



(a) triangulated rectangle (b) torus

Figure 8: Torus example and its triangulation.

Figure 9 shows some stages of the Edgebreaker compression algorithm. Starting with the gray triangle, one can easily obtain the sequence CCCCRCCCRCCRCCR for the 15 initial triangles. Notice that the opposite sides of the triangulated rectangle are identified. To visually percept the visited vertices, they are represented by black dots. The 16th triangle is of type S, since its tip vertex has already been marked (when the algorithm passes through the 7th triangle) and its left and right adjacent faces have not been visited yet.

Figure 10 illustrates the labels of all triangles of the torus defined by the Edgebreaker compression algorithm. At the end of the algorithm, the *clers* string obtained for this surface is CCCCRCCCRCCRCCRSCRSCRSRLSEERSEE.

As one can observe, in this example, that there are five triangles labeled with S. In the string sequence, the last three S are normal, since their right and left branches are traversed in the compression algorithm. On the other hand, the left branches of the first and of the second S triangles are not traversed since their left adjacent triangle have been visited during their right branch traversal. Therefore, the two red pairs of opposite corners are saved in the *handle* file.

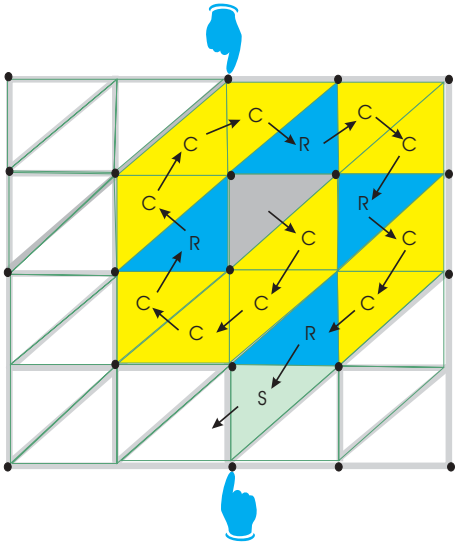


Figure 9: earlier stages for the torus example.

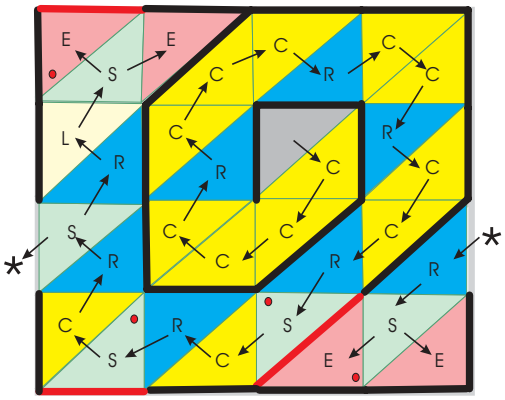


Figure 10: clers sequence for the torus example.

6.3 Why does it work?

The Edgebreaker compression algorithm moves from one triangle to an adjacent one that has not been previously visited until it passes through all triangles. As a consequence, it generates a spanning tree on the triangle-edge connectivity graph $G_{TE}(M)$, which is denoted by $T_{TE}(M)$.

A line l belongs to $T_{TE}(M)$ if and only if its corresponding edge $l_{TE}^{-1}(l)$ is a gate for a triangle. Notice that for a tree, the number of nodes minus the number of lines is equal to one. Thus, we can assert that the number of triangles minus the number of gates is equal to one.

Define that an edge $e \in E(M)$ belongs to a set $\Gamma(M)$ if $n_{TE}(e) \notin T_{TE}(M)$, i.e., e is not a gate.

The symbols #C and #G denote, respectively, the number of C triangles and the number of edges that are gates. And the operator |A| means the cardinality of a set A.

One can observe that the number of vertices of M is equal to the #C plus 3 (the three vertices of the starting triangle) [13].

Let us consider the set $\Psi(M)$ of edges composed by the two non-gate edges of the starting triangle and all non-gate edges of a C triangle (black edge of the C triangle in Figure 7). As an immediate consequence $|\Psi(M)| = \#C+2$. We are now to study the several other import properties of this set.

Notice that $\Psi(M)$ is a subset of $\Gamma(M)$, since an edges in $\Psi(M)$ is not a gate by definition.

There is no vertex of M that is not incident to an edge in $\Psi(M)$, since the Edgebreaker visit all vertices through the C triangles.

There is an theorem [2] in graph theory that asserts an important property for subgraphs.

Theorem 6.1: *Let G be a simple graph with n nodes. If a subgraph G' with n nodes satisfies any two of the following three p, then it satisfies the third as well.*

(a) G' is connected.

(b) G' has (n-1) edges.

(c) G' is acyclic.

Notice that this theorem characterizes a spanning-tree in a graph and we can use it to proclaim the following Lemma.

Lemma 6.2: *The edges of $\Psi(M)$ form a spanning tree, denoted by T_{VE}, on the triangle-edge connectivity graph $G_{VE}(M)$.*

Proof: The hypothesis of theorem 6.1 are satisfied, since:

(a) G_{VE} is simple, otherwise M is not a combinatorial surface.

(b) T_{VE} contains all nodes of G_{VE}, since Edgebreaker visit all vertices of M.

Thus Theorem 6.1 guarantees that this assertion is true, based on the following facts:

(a) The number of lines on T_{VE} is equal to the number of nodes in T_{VE} minus one. Since in T_{VE} there are #C+2 lines and #C+3 nodes.

(b) T_{VE} is acyclic, because a C triangle is created only when the tip vertex has not been visited yet.

The next lemma calculates the cardinality of $\Gamma(M) - \Psi(M)$, which is denoted by the number h.

Lemma 6.3: *The number of non-gate edges that are not incident to a C triangle is twice the number of genus on the surface M, i.e., $|\Gamma(M)| - |\Psi(M)| = 2g$.*

Proof: For a resume of the facts observed above, we can write the following equations:

- $|T(M)| = \#G + 1$
- $|V(M)| = \#C + 3$
- $|E(M)| = \#G + |\Gamma(M)|$
- $|\Gamma(M)| = |\Psi(M)| + h$
- $|\Psi(M)| = \#C + 2$

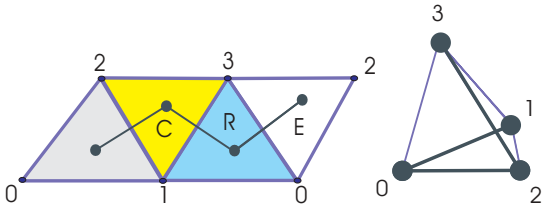
Thus, we can express the Euler characteristic of M in terms of h as follows:

$$\chi(M) = |V(M)| - |E(M)| + |T(M)|$$
$$= (\#C + 3) - (\#G + \#C + 2 + h) + (1 + \#G)$$
$$= 2 - h.$$

Substituting the equation above in the Euler formula $\chi(M) = 2 - 2g$, we can conclude that $h = 2g$.

When M is homeomorphic to a sphere ($g = 0$) we have $\Gamma(M) = \Psi(M)$, which means that the edges in the implicitly defined spanning-tree T_{VE} of G_{VE} together with the spanning-tree T_{TE} of G_{TE} contains all edges of M.

To exemplify, let us consider the tetrahedron example of Figure 6. The *clers* string obtained starting with the triangle <0,1,2> is CRE, where <a,...,z> means the the spanning space generated by the vertices in the brackets. In this example, C is the triangle <3,2,1>, R is the triangle <0,3,1>, and E is the triangle <2,3,0>. The spanning trees T_{TE} and T_{VE} are illustrated in Figure 11. The lines of T_{TE} correspond to the edges <1,2>,<1,3> and <3,0> and the lines of T_{VE} are associated to the edges <0,1>,<0.2> and <2,3>.



(a) T_{TE} (b) T_{VE}

Figure 11: spanning-trees of a tetrahedron.

Lemma 6.4: *If we cut the surface M without boundary along the edges of the T_{VE} spanning tree by the use of the inverse handle operators, then the resulted surface has an unique boundary curve component and has no interior vertex.*

Proof: When the first edge is cut, we have to apply the inverse handle operator of type 2 which creates a new boundary curve component by splitting an interior edge with no incident boundary vertices (see Figure 4(a)). Since T_{VE} is a tree, we can take the edge incident to one vertex on this boundary curve to apply the inverse handle operator. In this case, we have to apply the inverse handle operator of type 2 that split an interior vertex with one incident boundary vertex (see Figure 4(b)), i.e., it simply "unzips" the boundary. This edge couldn't have two incident boundary vertices otherwise T_{VE} is not a tree. We can continue this process until all edges are cut. Since the T_{VE} is a spanning tree, it contains all vertices of M, then after operating all edges with the inverse handle operators, there will be no interior vertices on the resulted surface.

Figure 12 illustrates the cut for the tetrahedron example along the T_{TE} tree. The resulting surface correspond exactly to the surface (path of faces) defined by the T_{TE} tree (see Figure 11(a)).

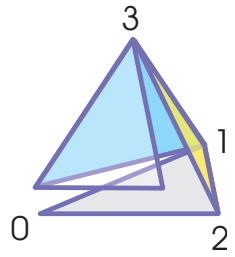


Figure 12: tetrahedron cut along T_{VE}.

Notice that T_{TE} can be obtained by decoding the *clers* sequence and the edges defined on graph T_{VE} can be completely recovered by the *Zip* procedure [14], which will be discussed in the next section. In conclusion, if the surface has no genus (g=0), the *clers* string is sufficient for the Edgebreaker algorithm presented in [15] to reconstruct the connectivity of the surface.

However, when the surface has g genus (g > 0), Lemma 6.3 affirms that if we consider all edges of T_{TE} and T_{VE} there will be 2g edges missing. In our algorithm we store these 2g edges in the *handle* stream, as one can observer in the next Lemma.

Lemma 6.5:. *Suppose that M has g genus. An edge belongs to $\Gamma(M) - \Psi(M)$ if and only if its corresponding pair of corners are stored in the handle file.*

Proof: ($\Leftarrow$) When the left adjacent triangle to an S has been visited during the right branch traversal, the pair of opposite corners to the left edge e are stored in the *handle* stream. In this case, the left adjacent triangle to the handle S, obviously, cannot be a C. In addition, e is not a gate in both incident faces. Thus, $e \in \Gamma(M) - \Psi(M)$.

($\Rightarrow$) By definition, if $e \in \Gamma(M) - \Psi(M)$, then e is not a gate and it couldn't be incident to a C triangle. Suppose, by contradiction, that the opposite corners of e are not stored in the *handle* file. Thus, e also couldn't be incident to an S triangle. So, one can observe that e has to be a non-gate edge of an E,R and L triangle, which must be incident to a C [14] that is an absurd.

The transmittance of the edges that belongs to the set $\Gamma(M) - \Psi(M)$ is the main innovation of the compression algorithm presented in this work. It will allow us to recover totally the surface connectivity just like we usually do for surfaces homeomorphic to the sphere, i.e., by the use of the *Zip* procedure.

As a consequence of lemmas 6.4 and 6.5, we can proclaim the following theorem.

Theorem 6.6: *If the surface has g genus, the Edgebreaker clers string together with the 2g pairs of corners in the handle file are sufficient to reconstruct the connectivity of the surface.*

Section 7 will talk about the decompression algorithm. There, an analysis based on the Handlebody theory will be done to explain the topological meaning of the edges in $\Gamma(M) - \Psi(M)$. Such analysis (section 7.3) can be used as a sketch of the proof for the above theorem.

6.4 Compression analysis

Since in a zero-genus surface there are twice more triangles than vertices and since each vertex is associated with a different C triangle, half of the triangles are labeled with a C. Consequently, the Edgebreaker cost for encoding the

connectivity of a mesh with no handles is guaranteed not to exceed two bits per triangle, regardless of the simplicity or irregularity of the mesh. With slightly more complex codes, a tighter bounds of 1.80 bits per triangle can be guaranteed [3]. For large meshes, entropy codes yield less than 1.0 bits per triangle [13]. When the mesh is sufficiently regular, i.e., has a large number of vertices with exactly six incident triangles, we can guarantee an encoding of 0.811 bits per triangle [17].

When the mesh has *g* genus, we must encode, in addition to the *clers* string, 2g entries according to Theorem 6.6. Each entry identifies the pair of opposite corners that cannot be identified from the *clers* string alone. Each identifier requires log(3T) bits.

7. EDGEBREAKER DECOMPRESSION

The decompression algorithm (see Appendix 2) builds the V and O arrays of the corner Corner-Table and the G table of vertex locations, by reading the data stored in the *clers*, *vertices* and *handle* files.

7.1 The Algorithm

The *InitDecompression* routine initializes the first triangle and reads all pairs of corners in the handle file and store them in a queue (H array). Each pair of opposite corner are then assigned in the O table. The decompression starts when the recursive procedure *Decompress* is called with corner 1 as parameter. To completely build all topological connectivity, the *Zip* routine is called starting with the corner with index 0.

In the *Decompress* routine, Edgebreaker appends a new triangle to a previously visited triangle at each iteration of the loop, it reads the binary encoding of the next symbol from the *clers* string. The steps necessary to attach this new triangle based on its type are now to be described.

1. C triangle (binary code 0): Edgebreaker associates the label –1 with the corner opposite the left edge. This temporary marking is stored in the O table. It will be replaced with the correct reference to the opposite corner by a subsequent *Zip* operation that will be later discussed. Finally, the coordinates of the new vertex are read from the *vertices* file.
2. L triangle (binary code 110): Edgebreaker tests the value stored in O table entry for the corner opposite to the left edge. When this value is equal to –3, a different label –2 is than stored. Otherwise, it means that this corner has been previously assigned, i.e., it is one of the corners in the *handle* file.
3. R triangle (binary code 101): the O table entry for the corner opposite to the right edge is assigned –2 only when the opposite right edge is not part of a handle. Than, Edgebreaker goes to the left triangle.
4. E triangle (binary code 111): both of its boundary edges are labeled –2 if they are not part of a handle.
5. S triangle (binary code 100): this case forks a recursive call to *Decompress*, which will construct and zip a subset of the mesh that is incident to the right edge of the current triangle. If the corner associated to the left edge is not part of a handle, then the reconstruction proceeds to decode and build the branch attached to the left edge of the current triangle. Otherwise, the routine returns to continue the surface decoding.

The corners in the O table whose value –1, are the two opposite corners to the non-gate edges of the starting triangle plus all non-gate edges of a C triangle. Notice that all those corners have already the information about their incident vertices. Moreover, their opposite edges form the T_{VE} tree. On the other hand, the corners in the O table marked with –2 have no information about its incidence. These corners should be associated with an corner whose opposite is marked as –1.

The *Zip* routine starts by finding consecutive pairs of edges in the boundary curve whose labels are in order equal to –1 and -2. The initial edge in the zipping process is the opposite edge of the 2 corner and it will continue as long as exists a pair of free edges. To find the next pair, the *NextCWBoundaryEdge* is called. This procedure performs a walk over the surface until a CW oriented boundary edge (O entry equal to –1) is found.

To illustrate the *Zip* procedure action, let consider the tetrahedron decodification. Notice that its *clers* string is CRE. At the moment just after the *Decompress* call, Table 3 shows the information available in the O and V tables and Figure 13 illustrates this configuration. The first edge that is zipped is the one opposite to the corners 2 and 7. The *NextCWBoundaryEdge* is than called, and the next available boundary edge with O entry equal to –1 is the corner 1. Thus, the identification of the opposite boundary edges to the corners 1 and 10 form the second edge zipped. Finally, the third edge represented by the corners 5 and 11 is zipped.

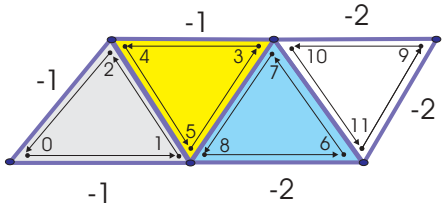


Figure 13: Tetrahedron corner assignment after Decompress.

Corner	O Table	V Table
0	3	0
1	-1	1
2	-1	2
3	0	3
4	6	2
5	-1	1
6	4	-1
7	-2	3
8	9	1
9	8	-1
10	-2	3
11	-2	-1

Table 3: V and O tables for a tetrahedron just afterDecompress.

7.2 The torus example

To illustrate the algorithm, consider the compressed triangulated torus illustrated in Figure 10. After updating the O table with two pairs of opposite corners saved on the *handle* file, the resulted surface obtained after decoding the *clers* string is shown Figure 14. In this figure the boundary edges are in green and the edges that have been identified by a handle are in red. The next step is to perform the *Zip* operation, to close the unique boundary curve component. Figure 15 shows the resulted surface after two zip operations, the zipped edges are in yellow. In this Figure, we indicate by the use of some symbols the orientation of the boundary curve.

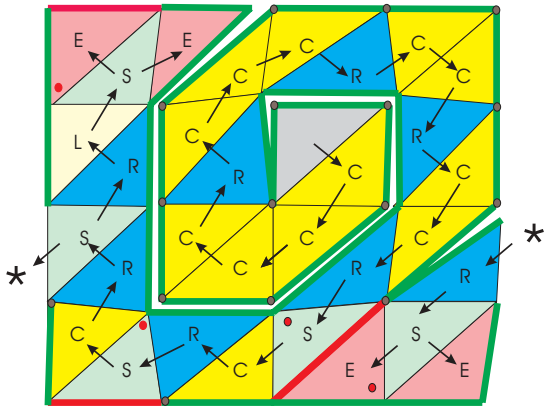


Figure 14: Torus surface decoded before Zipping.

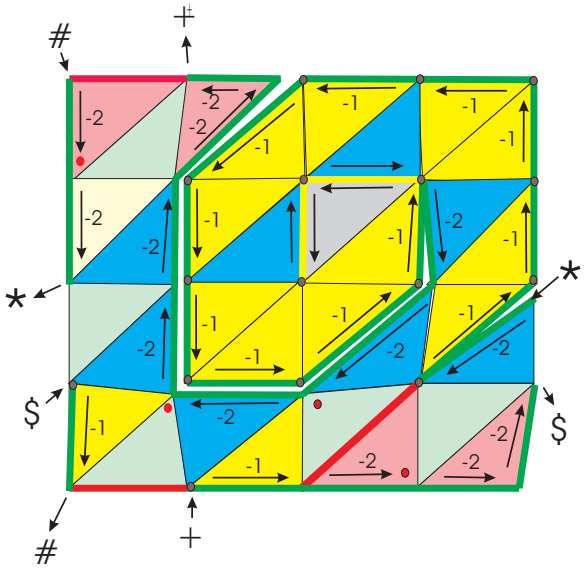


Figure 15: Earlier two steps of Zip.

7.3 Edgebreaker Handlebody Decomposition

In resume, the Edgebreaker decompression algorithm decodes the compressed surface adding a new triangle one by one until all symbols stored in the *clers* file are read. In addition, the *InitDecompression* updates the O table by assigning the opposite corners of the 2g edges saved in the *handle* file. The surface generated by the *Decompress* call is homeomorphic to the original surface M inverse handle operated over the edges in the spanning tree $T_{TE}(M)$. The next decompression step is the *Zip* process, which closes the unique boundary curve to finalize the reconstruction of the surface connectivity and geometry.

All those steps are, in fact, defining a handlebody decomposition for the surface M. The finite sequence $\{M_i\}_{i=0...n}$ of combinatorial surfaces, such that $M_0=\emptyset$ and $M_n= M$, generated by the Edgebreaker decompression algorithm can now be analyzed by the use of handle operators. Where n corresponds to the number of handle operators we shall use to build the surface. In this section, we will show that this number is $2|T(M)|+|V(M)|+2g$.

Notice that the identification of two boundary edges is performed in the Corner-Table data structure by setting in the O table the corresponding two opposite corners.

The Edgebreaker decompression algorithm initializes with a triangle. In this case, a Handle operator of index 0 is applied to create such triangle that corresponds to M_1.

In the next step, the Edgebreaker begins to decode the *clers* string. The notation $clers_i$ will be used to access the i^{th} symbol of the clers string. Notice that the *clers* sequence has $|T(M)|-1$ symbols.

When the $clers_i$ symbol is read, a new triangle is created (Handle operator of type 0) and one of the edges of this new triangle is attached to the gate edge of the surface M_i (Handle operator of type 1 illustrated in Figure 3(a)). The surface M_{2i+2} is defined to be the surface resulted by the application of those two Handle operations.

The Euler characteristic of the surface M_{2i+2} is the same of M_{2i}, for i in $[0...|T(M)|]$. Since M_{2i+2} has one more triangle, two more edges and one more vertex than M_i. As a conclusion, the surface $M_{2|T(M)|+1}$ is a connected surface homeomorphic to a disk (with a unique boundary curve component) with no interior vertex, i.e., $\chi(M_{2|T(M)|+1})=1$.

If the *handle* file is empty (g=0), then we can go directly to the *Zip* part of the Handlebody decomposition. Otherwise, we have to identify the 2g pairs of boundary edges represented by the corners stored in the *handle* file.

For each genus, we identify the first pair of boundary edges to generate the surface $M_{2|T(M)|+2}$ by the use of a Handle operator of index 1 illustrated in Figure 3(b). Notice that those edges have no vertices in common and belong to the same boundary curve component. Notice that this operator splits the boundary curve into two components. Next, we identify the second pair of boundary edges, which belong to different boundary curves, by the use of the handle operator of index 1 of Figure 3(c) to generate the surface $M_{2|T(M)|+3}$. This operator adds a genus to the surface and concatenates two boundary curves. The Euler characteristic of $M_{2|T(M)|+3}$ is calculated according to the following expression:

$$\chi(M_{2|T(M)|+3}) = (|V(M_{2|T(M)|+1})|-2) - (|E(M_{2|T(M)|+1})|-1) + |T(M_{2|T(M)|+1}| = \chi(M_{2|T(M)|+1})-1 = 1-2 = -1.$$

This process is repeated until all pairs of boundary edges in the *handle* file are identified. As a consequence, the surface $M_{2|T(M))|+1+2g}$ is a connected surface with genus g that has an unique boundary curve component and no interior vertex. Its Euler characteristic is $\chi(M_{2|T(M))|+1+2g}) = \chi(M_{2|T(M))|+1})-2g = 1-2g$.

Next, the *Zip* routine is called to identify $|V(M)| - 1$ pairs of adjacent boundary edges with indexes –1 and –2. Those edges, after the identification, will correspond to the T_{VE} spanning tree. When the first pair is found, a Handle operator of index 2 is applied (Figure 4(b)). This operator identify those two boundary edges that have only one vertex in common to create one interior vertex on the surface $M_{(2|T(M)|+1+2g)+1}$. It doesn't change the Euler characteristic of

surface. Therefore, the resulted surface is homeomorphic to $M_{2|T(M)|+1+2g}$.

The *Zip* routine continues to search recursively and builds a new surface for each pair found. The recursion will stop when the last pair of boundary edges with indices –1 and –2 is reached (notice that all the $|V(M)|$-2 boundary edges pairs initially found by the *Zip* procedure have only one vertex in common). Thus, the surface $M_{(2|T(M)|+1+2g)+|V(M)|-2}$ is still homeomorphic to $M_{2|T(M)|+1+2g}$, and $\chi(M_{2|T(M)|+2g+|V(M)|-1}) = 1\text{-}2g$.

Finally, the last pair of boundary edges existent in the surface is identified by the Handle operator of type 2 shown in Figure 4(a), which removes the boundary curve component. We can conclude that $M_{2|T(M)|+2g+|V(M)|}$ is the original surface without boundary M whose Euler characteristic is $\chi(M)$=2-2g.

8. CONCLUSIONS

In conclusion, we have extended the simple Implementation of Edgebreaker to the connectivity graphs of triangle meshes that represent topological manifolds with handles. The compression and decompression algorithms use extremely simple data structures and each require only a couple of pages of simple code.

In addition to the *clers* string, the compression produces a list of pairs of opposite corners, which encode how to turn a simply connected triangulated polygon into a surface with a single bounding loop, which, when zipped, will produce a surface with genus.

We believe that this new implementation of Edgebreaker is well positioned to become the defacto standard in transmitting compressed triangle meshes, which may represent full resolution 3D models, lowest-resolution models to be followed by resolution refinements in a progressive transmission scheme, or the coarse polygons that control the shape of subdivision surfaces.

9. ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant 9721358.

10. REFERENCES

[1] Baumgart, B. G.. A polyhedron Representation for Computer Vision. AFIPS National Conference 44 (1975), 589-596.

[2] Chartand, G., Oellermann, O.R.. Applied and Algorithmic Graph Theory, McGraw-Hill International Series in Pure and Applied Mathematics, 1993.

[3] Edgebreaker Software URL: http://www.gvu.gatech.edu/~jarek/edgebreaker/eb.

[4] Gumhold, S., and Strasser W.. Real Time Compression of Triangle Mesh Connectivity, Proceedings of the ACM SIGGRAPH'98 (July, 1998), ACM Press, 133-240.

[5] Gumhold, S., New bounds on the encoding of planar graphs, WSI-2000-1, ISSN 0946-3852 (Course note on the ACM Siggraph lecture in July 2000), January 31, 2000.

[6] Isenburg, M. and Snoeyink. Spirale Reversi: Reverse decoding of the Edgebreaker encoding, Computational Geometry, Theory and Applications (October, 2001), 39-52.

[7] King, D., and Rossignac J.. Guaranteed 3.67V bit encoding of planar triangle graphs, Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG'99), (August, 1999), 146-149.

[8] King, D., and Rossignac, J.. Connectivity Compression for Irregular Quadrilateral Meshes, Research Report GIT-GVU-99-29 (December, 1999).

[9] Lopes, H., Algorithms to build and unbuild 2 and 3 dimensional manifolds, Ph.D. Thesis (in portuguese), Department of Mathematics, Pontifical Catholic University of Rio de Janeiro, Brazil, 1996.

[10] Milnor, J.. Morse Theory, Annals of Mathematics Study 51, Princeton University Press, 1963.

[11] Poincaré. Sur la Generalisation d'un Théoréme d'Euler relatif aux Poliédres, C. R. Acad. Sci. Paris 117 (1983), 437-464.

[12] Rossignac, J. and Cardoze, D.. Matchmaker: Manifold Breps for non-manifold r-sets, Proceedings of the ACM Symposium on Solid Modeling and its Applications'99, (June, 1999), 31-41.

[13] Rossignac, J.. Edgebreaker: Connectivity compression for triangle meshes, IEEE Transactions on Visualization and Computer Graphics 5(1), (March, 1999), 47-61.

[14] Rossignac, J., and Szymczak, A.. Wrap&Zip decompression of the connectivity of triangle meshes compressed with Edgebreaker, Computational Geometry, Theory and Applications 14(1/3), (November, 1999), 119-135.

[15] Rossignac, J., Safanova, A., and Szymczak, A.. 3D Compression Made Simple: Edgebreaker on a Corner-Table. Proceedings of the Shape Modeling International Conference, (May, 2001).

[16] Rourke, C. and Sanderson, B.. Introduction to Piecewise-Linear Topology, Springer Verlag, 1972.

[17] Salomon, D.. Data Compression: The Complete Reference, 2nd Edition, Springer Verlag, 2000.

[18] Szymczak, A., King, D., and Rossignac, J.. "An Edgebreaker-based efficient compression scheme for regular meshes", Computational Geometry, Theory and Applications (October, 2001),53-68.

[19] Taubin, G., Rossignac, J.. Geometric Compression through topological surgery. ACM Transaction on Graphics, 17(2), (1998), 84-115.

Appendix 1: Edgebreaker Compression Algorithm

```
1.  PROCEDURE initCompression (c) {                    # c is a starting corner
2.   GLOBAL M[] = {0...}, U[] = {0...};                # init tables for marking visited vertices and triangles
3.   GLOBAL T = 0;                                     # id of the last triangle compressed so far
4.   WRITE (vertices, c.v.g); M[c.v] = 1;              # store first vertex in vertices file and mark it as visited
5.   WRITE (vertices, c.n.v.g); M[c.n.v] = 1;          # store second vertex in vertices file and and mark it as visited
6.   WRITE (vertices, c.p.v.g); M[c.p.v] = 1;          # store third  vertex in vertices file and mark it as visited
7.   U[T] = 1;                                         # mark the first triangle as visited
8.   Compress(c.o);                                    # start the compression process
9.  }                                                  # end of initCompression

10. PROCEDURE Compress(c) {                            # compress simple triangulated surfaces with genus
11.  REPEAT {                                          # start traversal for T_TE spanning tree
12.   U[c.t] = 1;       T++;                           # mark the triangle as visited and increments triangle counts
13.   IF (c.n.o.t.u > 1) THEN {                        # check for handle from right
14.      WRITE (handles,c.n.o.t.u); WRITE (handles,T*3+1);  # encodes pair  of opposite corners to be glued for a handle
15.   }                                                # end of IF
16.   IF (c.p.o.t.u > 1) THEN {                        # check for handle from right
17.      WRITE (handles,c.p.o.t.u); WRITE (handles,T*3+2);  # encodes pair  of opposite corners to be glued for a handle
18.   }                                                # end of IF
19.   IF (c.v.m != 1) THEN {                           # test whether the tip vertex was visited
20.      WRITE (vertices, c.v.g);  M[c.v] =1;          # IF WAS, store vertex in vertices file and and mark it as visited
21.      WRITE (clers, 0);  c = c.r;                   # append encoding of C to clers and moves to the next triangle
22.   } ELSE {                                         # IF WAS NOT,
23.      IF (c.r.t.u > 0) THEN {                       # test whether right triangle was visited
24.         IF (c.l.t.u > 0) THEN { WRITE (clers, 111);  RETURN; }# if left  triangle was visited, append encoding of E to clers and pop
25.         ELSE { WRITE (clers,101);  c = c.l; }      # otherwise, append encoding of R and moves to the left neighbor; end  of  IF

26.      } ELSE {
27.         IF (c.l.t.u > 0) THEN { WRITE (clers, 110);  c = c.r; }   # if left  triangle was visited, append encoding of L and moves to the
    right neighbor
28.         ELSE { U[c.t] = T*3+2;                     # otherwise, store corner number in decompression (potential handle)
29.            WRITE (clers,101);                      # append encoding of S to clers
30.            Compress(c.r);                          # recursive call to first visit right branch of split
31.            c = c.l;                                # upon return, move to the left triangle
32.            IF (c.t.u > 0) THEN RETURN;             # if the triangle to the left was visited during the right branch traversal, then return
33. }}}}}                                              # end  of  IF; end  of  IF; end  of  IF; end  of  REPEAT; end  of  Compress
```

Appendix 2: Edgebreaker Decompression Algorithm

```
1.  PROCEDURE initDecompression  {
2.   GLOBAL V[] = {0,,1,2,-1,-1,-1,...};          # table of vertex Id for each corner
3.   GLOBAL O[] = {-3,-1,-1,-3,-3,-3,...};        # table of opposite corner Ids for each corner
4.   GLOBAL T = 0;                                # id of the last triangle compressed so far
5.   GLOBAL N = 2;                                # id of the last triangle visited so far
6.   GLOBAL A = 0;                                # id of the last handle attached
7.   GLOBAL H = READ(handles);                    # read handle pairs from handles file into an array H
8.   FOR (i = 0;  i < sizeof (H) ; i += 2) {      # for all pair of  cornes in the handle array
9.      O[H[i]] = H[i+1]; O[H[i+1]] = H[i];       # fill opposite cornes in the O table
10.  }                                            # end of for
11.  G[0] = READ (vertices);                      # read first vertex in vertices file
12.  G[1] = READ (vertices);                      # read second vertex in vertices file
13.  G[2] = READ (vertices);                      # read third vertex in vertices file
14.  Decompress(0);                               # start the decompression process
15.  Zip(2);                                      # zip the boundary curve, starting with the corner 2
16.  }                                            # end of initDecompression

17.  PROCEDURE Decompress(c) {                    # decompress simple triangulated surfaces with genus
18.   REPEAT {                                    # loop that builds T_TE spanning tree and zips it up
19.     T++;                                      # new triangle
20.    O[c] = 3*T; O[3T] = c;                     # attach new triangle, link opposite corners
21.    V[3*T+1] = c.p.v; V[3*T+2] = c.n.p;        # enter vertex Id for shared vertices
22.    c = c.o.n;                                 # move corner to new triangle
23.    SWITCH READ(clers) {                       # select operation based on next symbol
24.       CASE 'C': {O[c.n] = -1; V[3*T] = ++N; G[N] = READ(vertices); }   # C triangle: left edge is free, store ref and geometry of new vertex
25.       CASE 'L': {IF (O[c.n] < 0) THEN O[c.n] = -2; }                   # L triangle: if it is not a handle, orient left edge
26.       CASE 'R': {IF (O[c] < 0) THEN {O[c] = -2};  c = c.n;}            # R triangle: if it is not a handle, orient right edge; and go left
27.       CASE 'S': {Decompress (c); c = c.n;                              # S triangle: recursion going right, then go left
28.                  IF  (O[c]>=0) THEN RETURN;}                           #             if it is a handle, then return
29.       CASE 'E': { IF (O[c] < 0) THEN O[c] = -2;                        # E triangle: orient free edges checking handles and pop
30.                  IF (O[c.n] < 0) THEN O[c.n] = -2;RETURN;}
31.  } }  }                                      # end  of  SWITCH ; end of REPEAT; end of Decompress

32.  PROCEDURE Zip(c) {                           # tries to zip free edges opposite to c
33.   b = c.n; WHILE ((b.o>=0) AND (b.o!=c)) DO b=b.o.n;                  # search clockwise for free edges
34.   IF (b.o != -2) THEN {c = NextCWBoundaryEdge(c); Zip(c); RETURN;}# if no zip possible, find next C boundary edge ; try to zip again and pop
35.   O[c] = b; O[b] = c;                         # link opposite corners
36.   a = b.n; V[a.n] = c.n.v;                    # assign coincident vertices
37.    WHILE ((a.o>=0) AND (a!=c)) DO {a = a.o.n; V[a.n] = c.n.v;}        # update all incident corners to zipped vertex
38.   N-=1; IF (N > 0) THEN { c = NextCWBoundaryEdge(c); Zip(c);}        # if there is still C boundary edges, find next free and try to zip it.
39.  }

40.  FUNCTION INTEGER NextCWBoundaryEdge (c) {    # find of next boundary edge on the right
41.   REPEAT {                                    # loop walks along the boundary,
42.      c = c.n; WHILE (O[c] >= 0) DO c = c.o.n; # traversing the star of vertices
43.    } UNTIL (O[c] == -1); RETURN (c);          # until it finds an CW oriented edge (-1)
44.  }                                            # end of NextCWBoundaryEdge
```