AN ALGORITHM FOR GENERATING SOLID ELEMENTS IN OBJECTS WITH HOLES†

TONY C. WOO and TIMOTHY THOMASMA
Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, Mi 48109, U.S.A.

(Received 21 April 1982; received for publication 17 January 1983)

Abstract—An algorithm for dividing an object with holes into solid elements for finite element preprocessing is presented. Since a tetrahedron can always be subdivided into prisms and cuboids, the approach of first dividing the given object into disjoint tetrahedra is taken.

Objects without holes are dealt with first. Two mesh operators, each generating a single tetrahedron, are presented. In addition to the construction procedure, it is shown that they handle all objects without holes. The algorithm for objects with holes requires a third operator. In addition to showing the necessary and sufficient condition for applying such an operator, it is shown that it effectively reduces the number of holes in an object by one while yielding three tetrahedra. The algorithm which sequences the three operators thus reduces a given polyhedron to a single tetrahedron iteratively. Data structure requirements and update procedures are also given in this paper.

NOTATION

\( \Pi \) a polyhedron
\( V \) a vertex in \( \Pi \)
\( E \) an edge in \( \Pi \)
\( F \) a face in \( \Pi \)
\( V \) number of vertices in \( \Pi \)
\( E \) number of edges in \( \Pi \)
\( F \) number of faces in \( \Pi \)
\( G \) number of holes in \( \Pi \)
\( t \) a tetrahedron
\( t_1, t_2, t_3 \) mesh operators
\( \Delta V \) change in \( V \) after each \( t_i \)
\( \Delta E \) change in \( E \) after each \( t_i \)
\( \Delta F \) change in \( F \) after each \( t_i \)
\( \Delta G \) change in \( G \) after each \( t_i \)

1. INTRODUCTION

In discrete mathematics and computing, the problem of dividing a complex geometric structure into simpler ones received some attention recently [2, 3, 7, 10, 13]. Since the primary emphasis had been on the analysis of algorithms for their computational complexities, a "complex geometric structure was often assumed to be a set of points [7] or a set of linear equations [3] while a "simple" structure only needed to be convex [2] without restriction on the number of vertices, say, in the structure. Two other attempts [10, 13] were made in dividing a polyhedron (defined by a set of vertices, edges and faces) into tetrahedra (defined as having four vertices, six edges and four faces). Success was largely limited to objects without holes.

This paper describes an algorithm for dividing an arbitrary polyhedral object (with or without holes) into solid elements. It is intended for the automatic processing of a geometric model [1, 9] into finite element models [8] in an integrated computer-aided design and analysis environment.

The approach taken in this paper for the automatic generation of solid elements is as follows.

†This work was supported in part by an SME Foundation research grant #481-186 and in part by the Center for Robotics and Integrated Manufacturing, The University of Michigan.

Step 1. Generate a rough mesh of tetrahedral elements without adding new vertices to the geometric model.

Step 2. Refine the elements by subdivision.

Step 1 is discussed in this paper. Step 2 is illustrated in Fig. 1 in which a tetrahedron is subdivided into tetrahedra, pentahedra, and hexahedra using new vertices such as the centroid of the tetrahedron, centers of the triangular facets and midpoints of the edges. It is assumed that subdivision can be carried out to any user-specified resolution.

In this paper two solid mesh operators are introduced in Section 2. Each operator generates one tetrahedron by either "slicing" or "digging" into the given polyhedron. After the tetrahedron passes geometric tests for non-interference tests, it is removed from the polyhedron.

This process iterates until the polyhedron is reduced to a single tetrahedron.

To ensure that the algorithm converges, the topological properties of the operators are examined in Section 3. Using Euler's formula for simple polyhedra, the operators are shown to maintain topological integrity at every step in the process.

Objects with holes are dealt with in Section 4. A third operator for "cutting" open a hole is introduced. As the operator for transforming a multiply-connected polyhedron into a simply-connected polyhedron is again examined for topological integrity, an algorithm combining all three operators is given.

2. MESH OPERATORS AND INTERFERENCE TESTS

A solid mesh of tetrahedral elements in a polyhedron consists of non-interfering tetrahedra. Each tetrahedron consists of four vertices, six edges and four (triangular) faces. There is no requirement for each tetrahedron to be regular, i.e. to have equilateral triangular faces with 60° dihedral angles. The only requirements are that the tetrahedra be non-overlapping and be in the interior of the polyhedron.

A polyhedron is represented by three kinds of entities—vertices, edges, and faces. Each kind of entity has two types of information associated with it—geometry and topology. The geometry of an entity records its...
The topology of an entity records its relation with another kind of entity.

The geometry of a vertex \( V_i \) is a triple \((X_i, Y_i, Z_i)\) which are the Cartesian coordinates of the vertex. Its topology is a list of pointers to all the incident edges. The geometry of an edge \( E_i \) is in the form of a parametric equation expressed in terms of its two endpoints \( V_i \) and \( V_j \):

\[
E_i = (1 - t)V_i + tV_j \quad t \in [0, 1]
\]

Its topology consists of two pointers to its two vertices and two pointers to its two faces. The geometry of a face \( F_i \) is a four-tuple \((A, B, C, D)\) where \( A, B, C \) are the direction cosines and \( D \) is the distance from the origin to a plane having the equation

\[
AX + BY + CZ + D = 0.
\]

It topology consists of a list of pointers to all the edges bounding the face. A summary of the geometry and the topology of the three entities is given below.

<table>
<thead>
<tr>
<th>entity</th>
<th>geometry</th>
<th>topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertex</td>
<td>( V_i = (X_i, Y_i, Z_i) )</td>
<td>points to ( n ) incident edges</td>
</tr>
<tr>
<td>edge</td>
<td>( E_i = (1 - t)V_i + tV_j )</td>
<td>points to two vertices and two faces</td>
</tr>
<tr>
<td>face</td>
<td>( F_i = AX + BY + CZ + D = 0 )</td>
<td>points to ( n ) bounding edges</td>
</tr>
</tbody>
</table>

It may be noted that the separation of topological information from geometric information permits greater flexibility for further development of algorithms. If, for instance, octahedral elements for polyhedra are desired, only the mesh operators need to be rewritten since the geometry does not change. If, on the other hand, curvilinear tetrahedral elements are desired for objects with curved surfaces, only the geometric tests need to be rewritten. In the following two mesh operators are presented which operate primarily on topological information. To ensure that a tetrahedron does not interfere with others, two geometric tests for vertex and edge interference are needed.

2.1 Mesh operators

The two mesh operators are called \( \tau_1 \) and \( \tau_2 \), each removing a tetrahedron \( T \) from the remaining polyhedron \( \pi \). In forming a tetrahedron, operator \( \tau_1 \) "slices" a corner off of \( \pi \) by making one "cut". The corner must necessarily be convex and trivalent (have three edges). If the remaining polyhedron does not have a convex trivalent vertex, operator \( \tau_2 \) "digs" out a tetrahedra from a convex edge by making two "cuts". All polyhedra have at least one convex edge, therefore \( \tau_2 \) can always be applied. These two operators are illustrated in Fig. 2.
An algorithm for generating solid elements in objects with holes

BEFORE DURING AFTER ΔV ΔE ΔF

Fig. 3. Cases of τ₁.

**Operator τ₁.**

τ₁ operates on a convex trivalent vertex Vᵢ. From the topology of Vᵢ, all four vertices, at least three of the six edges, and exactly three of the four faces for the tetrahedron to be constructed are immediately available. No vertex, zero to three edges, and one face need to be constructed. (See Fig. 3 for the various cases of τ₁.) The procedure for performing on vertex Vᵢ of polyhedron π can be stated as follows.

Algorithm τ₁(Vᵢ)

Step 1. Determine if Vᵢ has exactly three edges.
Step 2. Determine if Vᵢ is convex.
Step 3. Construct a tetrahedron τ from Vᵢ. Return τ.

**Operator τ₂.**

τ₂ operates on a convex edge Eᵢ. From its topology, at least two of the four vertices, at least one of the six edges, and exactly two of the four faces for the tetrahedron to be constructed are immediately available. No vertex, zero to five edges, and exactly two faces need to be constructed. (See Fig. 4 for the various cases of τ₂.) The procedure for performing τ₂ on edge Eᵢ of polyhedron π can be stated as follows.

Algorithm τ₂(Eᵢ)

Step 1. Determine if Eᵢ is convex.
Step 2. Construct a tetrahedron τ from Eᵢ. Return τ.

### 2.2 Interference Tests

A tetrahedron τ constructed by operators τ₁ and τ₂ must not interfere with any part of the polyhedron π. Interference is defined by following two rules.

**Rule VT.** No vertex Vᵢ of the polyhedron π lies on any of the four faces of the tetrahedron τ.

**Rule ET.** No edge Eᵢ of the polyhedron π intersects any of the four faces of the tetrahedron τ. Figures 5(a, b) illustrate the violation of Rule VT and Rule ET, respectively. Because of the “local influence” of the operators τ₁ and τ₂, the faces of the polyhedron cannot be intersected by the edges of the tetrahedron.

The two interference rules can be formulated as procedures with candidate tetrahedron τ and polyhedron π as input.

Algorithm VT(τ, π)

Step 1. Evaluate all vertices Vᵢ of polyhedron π on all four faces Fᵢ of τ.
Step 2. If a Vᵢ is on Fᵢ, return False. Else, return True.

Algorithm ET(τ, π)

Step 1. Calculate points of intersection between all edges Eᵢ of polyhedron π and all four faces Fᵢ of tetrahedron τ.
Step 2. If a point of intersection lies within the boundary of Fᵢ, return False. Else, return True.

In the next section, an algorithm that applies τ₁ and τ₂ is developed. Tests VT and ET are performed on the tetrahedron τ produced by either a τ₁ or a τ₂. The procedure iterates until the polyhedron π is reduced to a single tetrahedron.
3. MESHING SIMPLE OBJECTS

In this section we first provide a more rigorous treatment of the relationship between the two operators and the polyhedron. Specifically, we show that they are applicable to any polyhedron satisfying the derivative form of Euler's formula:

\[ \Delta V - \Delta E + \Delta F = 0 \]  

(1)

where \( \Delta V \), \( \Delta E \), and \( \Delta F \) are the changes in the number of vertices, edges and faces of a polyhedron respectively at every step of the process. Next, we give the procedure for meshing simple polyhedra satisfying eqn (1).

3.1 \( \tau_1 \), \( \tau_2 \) and simple objects

A simple polyhedron \( \pi \) with \( V \) vertices, \( E \) edges and \( F \) faces satisfies Euler’s formula

\[ V \ - \ E \ + \ F \ = \ 2. \]  

(2)

Consider inverse operators \( \tau_1^{-1} \) and \( \tau_2^{-1} \) that “glue” tetrahedra on a polyhedron. We show by induction that at any step of the construction process the polyhedron satisfies eqn (1).

<table>
<thead>
<tr>
<th>Before</th>
<th>During</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="before" /></td>
<td><img src="image2" alt="during" /></td>
<td><img src="image3" alt="after" /></td>
</tr>
</tbody>
</table>

\[ \Delta V \ \Delta E \ \Delta F \]

\[ \begin{align*}
0 & & 0 & & 0 \\
0 & & 1 & & 1 \\
0 & & 2 & & 2 \\
0 & & 2 & & 2 \\
0 & & 3 & & 3 \\
0 & & 4 & & 4 \\
\end{align*} \]
The first tetrahedron trivially satisfies eqn (2) since $V = 4$, $E = 6$ and $F = 4$. Suppose eqn (2) is true after $n$ steps. Consider the $(n + 1)^{th}$ step. If the operation is a $\tau_1^{-1}$, then from Fig. 3 the following table can be constructed.

<table>
<thead>
<tr>
<th>$\tau_1^{-1}$</th>
<th>$\Delta V$</th>
<th>$\Delta E$</th>
<th>$\Delta F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

If the operation is a $\tau_2^{-1}$, then a similar table can be constructed from Fig. 4.

<table>
<thead>
<tr>
<th>$\tau_2^{-1}$</th>
<th>$\Delta V$</th>
<th>$\Delta E$</th>
<th>$\Delta F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-3</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-4</td>
<td>-4</td>
<td></td>
</tr>
</tbody>
</table>

Clearly, in both cases eqn (1) is satisfied. Hence, eqn (2) is satisfied.

Having shown the applicability of $\tau_1$ and $\tau_2$ to simple polyhedra, we proceed with the description of the algorithm.

3.2 Algorithm for simple objects

Intuitively, $\tau_1$ seems to be "easier" to apply than $\tau_2$ since fewer new entities must be constructed. It may also seem intuitive that $\tau_1$ and $\tau_2$ must work "in tandem". The latter intuition can be verified by the change in the number of vertices $\Delta V$ each operator makes. From Fig. 3 and 4, we see that for $\tau_1$, $\Delta V = 1$, and for $\tau_2$, $\Delta V = 0$. Thus, operator $\tau_1$ eventually reduces a polyhedron with $V$ vertices to a tetrahedron with four vertices. As we also see, $\tau_2$ is needed when none of the vertices are "$\tau_1$-able", i.e. convex, trivalent, and yielding a non-interfering tetrahedron. The algorithm presented in this section has two nested loops, with $\tau_1$ as the workhorse in the inner loop. The flow of control of the algorithm is as follows. The inner loop first executes all the applicable $\tau_1$'s. The outer loop then indices by one and executes one $\tau_2$. If the tetrahedron produced is non-interfering, the control drops down to the inner loop. Else, another $\tau_2$ is executed.

Algorithm $S(\pi)$

Step 1. If $\pi$ is a tetrahedron, return.

For all edges $E$, do

For all vertices $V_i$ do

Step 2. $\tau_{1i}$-call $\tau_1(V_i)$

Step 3. If $VT(\pi, \pi)$ and $ET(\pi, \pi)$, $\pi \leftarrow \pi - \tau_1$.

end

Step 4. $\tau_{2i}$-call $\tau_2(E_i)$

Step 5. If $VT(\pi, \pi)$ and $ET(\pi, \pi)$, $\pi \leftarrow \pi - \tau_2$.

end

Step 6. Go to step 1.

Algorithm $S$ works in the following fashion. If there exists a convex trivalent vertex in the polyhedron $\pi$, then apply a $\tau_1$ and construct a tetrahedron $\tau$. If $\tau$ passes interference tests $VT$ and $ET$ then remove it from $\pi$. Continue with $\tau_1$ on the remaining polyhedron. If all remaining the vertices fail the interference tests or if they are not convex trivalent, apply at $\tau_2$. If a tetrahedron is successful obtained from a $\tau_2$, go back to $\tau_1$. Continue this process until the polyhedron $\pi$ is reduced to a single tetrahedron.

4. Meshing objects with holes

The two operators $\tau_1$ and $\tau_2$ apply to simply-connected polyhedra. An object with holes is a multiply-connected polyhedron. In this section we introduce a special operator $\tau_3$ that cuts open holes hence reducing a multiply-connected polyhedron to a simply-connected one. A necessary condition for applying operator $\tau_3$ is that if the hole is cut by a plane there exists a triangular cross-section with vertices $V_i$, $V_j$ and $V_k$. (Intuitively, such a condition always exists in a multiply-connected polyhedron. When a hole is cut, it yields a cross-section and if the cross-section is not a triangle, more $\tau_3$s and $\tau_2$s could have been applied.) Figure 6(a) illustrates such a condition. A sufficient condition for applying operator $\tau_3$ is that there exists two other triangular cross-sections $V_i$, $V_j$ and $V_k$, $V_m$. Figure 6(b) illustrates such a condition. Operator $\tau_3$ transforms the polyhedron from the configuration in Fig. 6(a) to that in Fig. 6(b) in three distinct stages. The notion of a genus is needed for dealing with the reduction of holes at these stages.

Euler's formula for a simple polyhedra does not hold for objects with holes[6]. The concept of a "three-dimensional hole" can be described by a parameter called genus $G$, commonly referred to as a handle. A simple polyhedron is topologically equivalent to a sphere. The vertices, edges and faces become nodes, arcs and regions on the sphere. The genus of a sphere is zero. An object with one hole is topologically equivalent to a torus. The genus of a torus is one. The genus $G$ of an object is related to $V$, $E$ and $F$ by the Euler-Poincare formula[5]

$$V - E + F = 2 - 2G$$

Meshing an object with holes involves changing the genus. In transition, the operator $\tau_3$ produces a non-manifold object having $G = \frac{1}{2}$. A small sphere placed near the singularity of a non-manifold would be divided into four regions, alternating inside, outside, inside and outside of the object. A small sphere placed anywhere else would be divided into at most two regions, inside and outside. Figure 7 shows two non-manifold objects. The object in Fig. 7(a) has a singularity of a point. The object in Fig. 7(b) has a singularity of a line. These two non-manifold objects occur as intermediate stages of a multiply-connected polyhedron as it is cut open by a $\tau_3$.

4.1 $\tau_3$ and its three stages

Operator $\tau_3$ opens up a hole in three distinct stages. The polyhedron under the operation makes the following transitions:

Stage 1. From manifold, $G = 1$ to non-manifold, $G = \frac{1}{2}$.

Stage 2. From non-manifold, $G = \frac{1}{2}$ to non-manifold, $G = \frac{1}{2}$.

Stage 3. From non-manifold, $G = \frac{1}{2}$ to manifold $G = 0$.

Figure 6(a) corresponds to the configuration of the polyhedron with a hole before stage 1. Figure 6(b) cor-
Fig. 6. Conditions for $\tau_5$.

responds to the configuration of the same polyhedron without a hole after stage 3. The three stages are illustrated in Fig. 8.

At each stage, a tetrahedron is removed, thus changing the $V$, $E$, and $F$ of the polyhedron. In addition, genus $G$ changes by $-\frac{1}{2}$ at stages 1 and 3 satisfying the derivative form of eqn (3):

$$\Delta V - \Delta E + \Delta F = -2\Delta G. \quad (4)$$

Figures 9–11 give detailed cases of each of the three stages. We can construct the following tables that verify eqn (4) is obeyed.

<table>
<thead>
<tr>
<th>$\tau_5$ stage 1</th>
<th>$\Delta V$</th>
<th>$\Delta E$</th>
<th>$\Delta F$</th>
<th>$\Delta G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\tau_5$ stage 2</th>
<th>$\Delta V$</th>
<th>$\Delta E$</th>
<th>$\Delta F$</th>
<th>$\Delta G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\tau_5$ stage 3</th>
<th>$\Delta V$</th>
<th>$\Delta E$</th>
<th>$\Delta F$</th>
<th>$\Delta G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-2</td>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td></td>
</tr>
</tbody>
</table>

We can express operator $\tau_5$ procedurally in terms of
An algorithm for generating solid elements in objects with holes

Fig. 8. The three stages of τ.

the three triangular cross-sections \( F_k = (V_i V_j V_k) \), \( F_1 = (V_i V_j V_k) \) and \( F_2 = (V_i V_j V_k) \) shown in Fig. 6(c). An edge connecting vertices \( V_i \) and \( V_j \) will be denoted by \( (V_i V_j) \).

Algorithm \( \tau \) \( (F_k, F_1, F_2, \pi) \)

Step 1. Let \( E \) be \( (V_i V_j) \). Apply step 1 of \( \tau \) to \( E \) and get tetrahedron \( \tau \). Remove \( \tau \) from \( \pi \).

Stage 2. Let \( E \) be \( (V_j V_k) \). Apply stage 2 of \( \tau \) to \( E \) and get \( \tau \). Remove \( \tau \) from \( \pi \).

Step 3. Let \( E \) be \( (V_i V_j V_k) \). Apply stage 3 of \( \tau \) to \( E \) and get \( \tau \). Remove \( \tau \) from \( \pi \).

The three stages of \( \tau \) are given in Fig. 9-11.

4.2 Algorithm for objects with holes

Compared to \( \tau_1 \) and \( \tau_2 \), \( \tau \) will be used less frequently. In fact, it will be used only as many times as there are holes in the object. For objects with holes, we shall develop an algorithm \( H \) with three nested loops. The inner loop will be for \( \tau_1 \). The middle loop will be for \( \tau_2 \). The outer loop will be for \( \tau \). The middle and the inner loops are Algorithm \( S \). As many \( \tau \)'s are executed as possible until all convex trivalent vertices are exhausted.

A \( \tau_2 \) is then executed and the control drops into the inner loop. The outer loop is indexed only when neither \( \tau_1 \) nor \( \tau_2 \) is applicable to the remaining polyhedron. At this point, a \( \tau \) is executed to transform the polyhedron into a \( \tau \)-able or a \( \tau \)-able polyhedron. The index of the outer loop \( F_k, F_1, F_2 \) corresponds to the \( \tau \) condition shown in Fig. 8.

Algorithm \( H(\pi) \)

Step 1. If \( \pi \) is a tetrahedron, return.

For all faces \( F_k, F_1, F_2 \) do

For all edges \( E \) do

For all vertices \( V \) do

Step 2. \( \tau \leftarrow \text{call } \tau_1(V_i) \)

Step 3. If \( VT(\tau, \pi) \) and \( ET(\tau, \pi) \), \( \pi \leftarrow \pi - \tau \).

end

Step 4. \( \tau \leftarrow \text{call } \tau_2(E_i) \)

Step 5. If \( VT(\tau, \pi) \) and \( ET(\tau, \pi) \), \( \pi \leftarrow \pi - \tau \).

end

Step 6. \( \text{Call } \tau_3(F_k, F_1, F_2, \pi) \)

end

Step 7. Go to Step 1.
<table>
<thead>
<tr>
<th>BEFORE</th>
<th>DURING</th>
<th>AFTER</th>
<th>$\Delta V$</th>
<th>$\Delta E$</th>
<th>$\Delta F$</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Before" /></td>
<td><img src="image2" alt="During" /></td>
<td><img src="image3" alt="After" /></td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td><img src="image4" alt="Before" /></td>
<td><img src="image5" alt="During" /></td>
<td><img src="image6" alt="After" /></td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><img src="image7" alt="Before" /></td>
<td><img src="image8" alt="During" /></td>
<td><img src="image9" alt="After" /></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 9. Stage 1 of r3.

<table>
<thead>
<tr>
<th>BEFORE</th>
<th>DURING</th>
<th>AFTER</th>
<th>$\Delta V$</th>
<th>$\Delta E$</th>
<th>$\Delta F$</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image10" alt="Before" /></td>
<td><img src="image11" alt="During" /></td>
<td><img src="image12" alt="After" /></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><img src="image13" alt="Before" /></td>
<td><img src="image14" alt="During" /></td>
<td><img src="image15" alt="After" /></td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td><img src="image16" alt="Before" /></td>
<td><img src="image17" alt="During" /></td>
<td><img src="image18" alt="After" /></td>
<td>0</td>
<td>-2</td>
<td>-2</td>
</tr>
</tbody>
</table>

Fig. 10. Stage 2 of r3.
An algorithm for generating solid elements in objects with holes

<table>
<thead>
<tr>
<th>BEFORE</th>
<th>DURING</th>
<th>AFTER</th>
<th>$\Delta V$</th>
<th>$\Delta E$</th>
<th>$\Delta F$</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
<td><img src="image3.png" alt="Diagram" /></td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td><img src="image4.png" alt="Diagram" /></td>
<td><img src="image5.png" alt="Diagram" /></td>
<td><img src="image6.png" alt="Diagram" /></td>
<td>0</td>
<td>-2</td>
<td>-1</td>
</tr>
<tr>
<td><img src="image7.png" alt="Diagram" /></td>
<td><img src="image8.png" alt="Diagram" /></td>
<td><img src="image9.png" alt="Diagram" /></td>
<td>0</td>
<td>-3</td>
<td>-2</td>
</tr>
</tbody>
</table>

Fig. 11. Stage 3 of $\tau_3$.

![Diagram](image10.png) ![Diagram](image11.png)

Fig. 12. Example of applying $\tau_1$, $\tau_2$, and $\tau_3$. 

![Diagram](image12.png) ![Diagram](image13.png)

![Diagram](image14.png) ![Diagram](image15.png)

![Diagram](image16.png) ![Diagram](image17.png)
Figure 12 shows the execution of algorithm H on an object with a hole. The object is transformed from Fig. 12(a, b) by a sequence of operators $t_2$, $t_1$, $r_2$, $r_3$ ($r_1$ could not be used as the first operation because the hole blocks the formation of a tetrahedron.) Eventually, the object is reduced to that shown in Fig. 12(c) where no $r_2$ is applicable. The three stages of $r_2$ are illustrated in Fig. 12(d-f). Since the object in Fig. 12(f) is simple, $r_1$ and $t_1$ can reduce it to a single tetrahedron.

5. CONCLUSIONS

We have shown that simple objects meshed by operators $t_1$ and $r_2$ obey $\Delta V - \Delta E - \Delta F = 0$. Algorithm S converges because $r_2$ reduces the degree of a vertex and $t_1$ reduces the number of vertices. For objects with holes, an operator $r_3$ is needed in conjunction with $t_1$ and $r_2$. $r_3$ changes the genus of a polyhedron from $G$ to $(G - 1)$. The three stages of $r_3$ yields a non-manifold object as tetrahedra are removed.

The implementation of Algorithm H can be simplified if we do not insist on mathematical rigor. For the reason of clarity, we have made a distinction between $t_1$ and $R$ by their topological differences in the updating of the polyhedron. For the reason of expediency, we allow $r_3$ to remove three tetrahedra in succession rather than letting $r_1$ take over after one tetrahedron is removed. In practice, if the updating rules for $r_2$ and $r_3$ are observed, there is no need to make a distinction between them. Hence, Algorithm S suffices.

As a side issue on storage allocation for the elements it would be of interest to know how many elements there are in a polyhedron without decomposing it. It is known that there can be $T$ tetrahedra in the interior of a polyhedron with $F$ faces on the boundary and $f$ faces in the interior satisfying the rule:

$$ T = \frac{1}{3}(F + 2f). $$

Recently, it has been shown that, a priori $T$ is related to the number of vertices $V$ by the relation:

$$ (V - 3) \leq T \leq \frac{(V - 3)(V - 2)}{2} $$

The bounds for $T$ can be quickly appreciated since a cube can be divided into five or six tetrahedra. If $v$ interior vertices are allowed in the polyhedron[17], there can be $T = V - v + d - 3$ tetrahedra, where $d$ is the number of interior diagonals.

REFERENCES