Is This A Quadrisected Mesh?

Gabriel Taubin

California Institute of Technology

Technical Report CSTR 2000.008

December 2000

ABSTRACT

In this paper we introduce a fast and efficient linear time and space algorithm to detect and reconstruct uniform Loop subdivision structure, or triangle quadrisection, in irregular triangular meshes. Instead of a naive sequential traversal algorithm, and motivated by the concept of covering surface in Algebraic Topology, we introduce a new algorithm based on global connectivity properties of the covering mesh. We consider two main applications for this algorithm. The first one is to enable interactive modeling systems that support Loop subdivision surfaces, to use popular interchange file formats which do not preserve the subdivision structure, such as VRML, without loss of information. The second application is to improve the compression efficiency of existing lossless connectivity compression schemes, by optimally compressing meshes with Loop subdivision connectivity. Extensions to other popular uniform primal subdivision schemes such as Catmul-Clark, and dual schemes such as Doo-Sabin, are relatively straightforward but will be studied elsewhere.

CR Categories and Subject Descriptors:
I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - surface, solid, and object representations.


1 INTRODUCTION

Subdivision surfaces are becoming a popular multi-resolution representation in modeling and animation [18, 17]. For example Figure 2-AB shows the result of applying Loop’s triangle quadrisection scheme [6] to a triangular mesh. Since the most popular interchange file formats, such as VRML [15], do not preserve the subdivision structure, a problem exists if the model is saved using one of these file formats and further editing is required at a later time. Alternatively, a proprietary file format with support for subdivision surfaces can be used, but limiting the distribution of the content. The method introduced in this paper to detect uniform quadrisection connectivity and to reconstruct the subdivision structure solves this problem.

Most 3D geometry compression techniques for polygonal meshes preserve the connectivity information without loss [13]. Lossless connectivity compression schemes are important for example, when a mesh is carefully constructed by an artist using a modeling or animation package. In this framework, changing the connectivity may destroy important features such as crease lines. In most interactive modeling systems polygonal meshes are constructed and refined by recursively applying a sequence of operators to a relatively simple base mesh. Some of these operators, such as uniform subdivision, change the connectivity, while others, such as smoothing, change the geometry. One example of this process is a mesh constructed by recursively applying a small number of uniform subdivision and smoothing steps to a coarse mesh [10].

When a 3D polygonal mesh is large and generated by oversampling a relatively smooth surface with simple topology, such as those produced by 3D scanning systems, lossy connectivity compression schemes can be used. Simplification algorithms [3] can be regarded as lossy connectivity compression techniques. Another very efficient scheme to compress this kind of data is based on remeshing, i.e., on approximating the geometry of the given polygonal mesh by a semi-regular subdivision surface within certain tolerance, and using wavelet-based coding techniques to compress the
geometry information [5]. This method does not produce good compression results when the topology is not simple, though, and replacing the connectivity of the mesh is not always acceptable.

Uniform subdivision schemes can be regarded as optimal progressive connectivity compression schemes, because the cost of encoding each subdivision step is constant [11]. Unfortunately, current geometry compression schemes [13] do not detect subdivision connectivity, and as a result, the cost of encoding a uniform subdivision step is normally function of the size of the coarse mesh.

For example, Table 1 shows the cost of encoding the connectivity of a tetrahedron and eight meshes constructed by recursive triangle quadrisection with the MPEG-4 3D Mesh coder [8] in single-resolution mode [12]. Note that, the total cost of encoding a quadrisection mesh is about twice the cost of encoding the original mesh \( B(4T) \approx 2B(T) \), while if optimally encoded, the incremental cost should be constant \( B(4T) = B(T) + O(1) \), corresponding to the number of bits used to represent the instruction specifying the subdivision operation in the compressed bitstream. Other single-resolution schemes [14] are more efficient at compressing these quasi-regular meshes, but still the incremental cost of encoding a quadrisection mesh is function of the size of the coarse mesh.

The method introduced in this paper, to detect uniform subdivision connectivity and to reconstruct the subdivision structure, can be used to minimize the cost of encoding the connectivity information of a fine mesh with uniform subdivision connectivity by representing the connectivity information as a coarse mesh followed by one or more uniform subdivision steps, rather than as a fine mesh compressed with a single-resolution or progressive scheme.

### 2 ALGORITHM OVERVIEW

A polygonal mesh is defined by the position of the vertices (geometry), by the association between each face and its sustaining vertices (connectivity); and optional colors, normals and texture coordinates (properties). In this paper, we are primarily concerned with the connectivity of triangular meshes.

Figure 3 shows the notation we use to represent a triangle and the result of quadrisection it. We call tile set a group of four connected triangles with the same connectivity as the result of quadrisection one triangle, i.e., four triangles connected as in Figure 3-B. The center triangle of a tile set is connected to three corner triangles through regular edges. The corners of the tile set are the vertices of the corner triangles not shared with the center triangle.

In a naive traversal algorithm to solve our problem tile sets are sequentially constructed while the mesh is traversed, say in depth-first order, trying to cover it avoiding tile overlaps, i.e., every face is allowed to belong to at most one tile set. If when the mesh traver-
3 POLYGONAL MESSES

In this section we introduce some definition, notation, and facts about polygonal meshes that we will need in subsequent sections to formulate our main results more precisely. It can be skipped on a first reading.

Connectivity The connectivity of a polygonal mesh $M$ is defined by the incidence relationships existing among its $V$ vertices, $E$ edges, and $F$ faces. We will also use the symbols $V$, $E$, and $F$ to denote the sets of vertices, edges, and faces. A face with $n$ corners is a sequence of $n \geq 3$ different vertices. If $f = (v_1, \ldots, v_n)$ is a face, all the cyclic permutations of its corners are considered identical, i.e., $f = (v_1, \ldots, v_n, v_1, \ldots, v_{n-1})$ for $i = 1, \ldots, n$. Multiple connected faces (faces with holes) are not representable.

procedure ConnectedComponents ($M$)
# initialize partition to set of singletons
$P = \{ \{ f \} : f \in F \}$
# traversal
for each regular edge $e$ connecting $f_1$ and $f_2$
if $f_1$ and $f_2$ are in different partitions
if ($P\text{.find}(f_1) \neq P\text{.find}(f_2)$)
# join corresponding partitions
$P\text{.union}(f_1, f_2)$
# find sets of supporting vertices
$M_i = (V_i, F_i), i = 1, \ldots, \text{cc}$
# return submeshes
return $P = \{ M_1, \ldots, M_{\text{cc}} \}$

Vertices not contained in any face are called isolated. An edge $e$ is an unordered pair of different vertices that are consecutive in one or more faces of the mesh. We will denote by $e$ the set of faces incident to the edge. A boundary edge has exactly one incident face. A regular edge has exactly two incident faces. A singular edge has three or more incident faces. Because the edges are derived from the faces, we write a mesh as $M = (V, F)$. We assume the reader is familiar with the concepts of manifold, orientable, oriented and non-manifold mesh. The algorithms introduced in this paper do not make use of these concepts, though.

Connected Components We say that two faces $f_1$ and $f_2$ are connected if we can find faces $f_3, \ldots, f_n$ of the mesh that $f_1$ shares and edge with its successor $f_{i+1}$ in the sequence (note that $n = 1$ and $n = 2$ are valid choices). This is an equivalence relation on the set of faces $F$ that defines a partition into disjoint connected components $F_1, \ldots, F_{\text{cc}}$. Each one of these connected component is a maximal subset of connected faces, i.e., a subset of faces that satisfies the following property: given a face in the subset, a second face is connected to the first one if and only if it also belongs to the subset. Together with its subset of supporting vertices $V_i \subset V$, each connected component $F_i$ defines a submesh $M_i = (V_i, F_i)$. Note the subsets of vertices $V_1, \ldots, V_{\text{cc}}$ are not necessarily disjoint, i.e., different connected components may share vertices. We call a mesh connected if it is composed of only one connected component. It is sufficient to know how to solve our problem for connected meshes: if the mesh is not connected, first decompose it into connected components, and then solve the problem for each component.

An algorithm based on Tarjan’s fast union-find data structure [9] can be used to partition the set of faces of a mesh into its connected components. It is described in pseudocode in Figure 5. It first initializes the partition to one singleton per face, and then for each edge of the mesh, and each pair of different faces sharing the edge, replaces the subsets corresponding to the two faces by their union.

Mappings A mapping $\phi : M_1 \rightarrow M_2$ from a first mesh $M_1$ into a second mesh $M_2$ is defined by a vertex function $\phi_V : V_1 \rightarrow V_2$ and a face function $\phi_F : F_1 \rightarrow F_2$ that satisfy the following additional property: for every face $f = (v_1, \ldots, v_n) \in F_1$ of the first mesh, the sequence of vertices of the second mesh defined by the vertex function applied to the corners of the face, is (modulo cyclical permutations) equal to the face of the second mesh that the face of the first mesh is mapped to by the face function. i.e.,

$$\phi_F(f) = (\phi_V(v_1), \ldots, \phi_V(v_n)) \in F_2.$$
E N U M E R A T I O N S  

Number of vertices, edges, and faces can be used to determine if a function \( \psi : A \rightarrow B \) between two finite sets is onto or not. Create a binary \((0, 1)\) array with elements in correspondence with the elements of \(B\) and initialized to 0. Then, for each element \(a \in A\) set the element corresponding to \(\psi(a)\) to 1. Finally, add all the values of the array. The function is onto if and only if the sum is equal to the number of elements of \(B\).

Equiv alence  
Two meshes \(M_1\) and \(M_2\) are called equivalent if a mapping \(\phi : M_1 \rightarrow M_2\) exists such that both \(\phi_V\) and \(\phi_E\) are \(1 \rightarrow 1\) and onto functions. In such case the mapping \(\phi\) is called a mesh equivalence.

Note that since the sets of vertices and faces are finite, the mapping \(\phi\) is an equivalence if and only if the vertex and face functions are onto, and the number of vertices and faces in both meshes are the same: \(V_1 = V_2\) and \(F_1 = F_2\).

And a simple linear time and space algorithm to count the number of elements of set

\[
\{\psi(a) : a \in A\} \subset B
\]
can be used to determine if a function \(\psi : A \rightarrow B\) between two finite sets is onto or not. Create a binary \((0, 1)\) array with elements in correspondence with the elements of \(B\) and initialized to 0. Then, for each element \(a \in A\) set the element corresponding to \(\psi(a)\) to 1. Finally, add all the values of the array. The function is onto if and only if the sum is equal to the number of elements of \(B\).

Quadrisection  
Figure 6 shows an example of a fine mesh (B) with 24 triangles resulting from quadrisecting a coarse mesh (A) with 6 triangles. The vertices of the coarse mesh are a subset of the vertices of the fine mesh. We call these vertices the \(V\)-vertices of the fine mesh. The remaining vertices of the fine mesh are in \(1 \rightarrow 1\) correspondence with the edges of the coarse mesh. We call these vertices the \(E\)-vertices of the fine mesh. Since we are only concerned with connectivity here, the position of the \(E\)-vertices in space irrelevant, but for illustration purposes, we draw them as the mid-edge points of the coarse mesh edges in Figure 6-B. Each triangular face of the coarse mesh is replaced by four triangles in the fine mesh. One triangle connects the three incident \(E\)-vertices, and each of the other three triangles connects one \(V\)-vertex and two \(E\)-vertices.

In general, the quadrisection operator \(Q\) transforms a triangular mesh \(M = (V, F)\) into a new triangular mesh \(M^Q = (V^Q, E^Q, F^Q)\), and defines a mapping \(\nu_M : M \rightarrow M^Q\), which assigns each vertex of \(M\) into a \(V\)-vertex of \(M^Q\), and each face of \(M\) into the center face of the corresponding quadrisected face. Both functions are \(1 \rightarrow 1\) but not onto. With respect to the number of vertices, edges, and faces, the following relations hold:

\[
\left\{
\begin{array}{l}
V^Q = V + E \\
E^Q = 2E + 3F \\
F^Q = 4F
\end{array}
\right.
\]

The quadrisection operator is one of many subdivision schemes that introduces new vertices along the edges of the coarse mesh, and replaces the coarse faces by fine faces supported on the new set of vertices. In general, because of limitations of the smoothing operators associated with these subdivision methods, meshes are required to be manifold without boundary, and special smoothing rules can be designed for manifold meshes with boundaries (holes) [1]. But since the connectivity refinement rules can be applied to non-manifold meshes, and our algorithm to detect and reconstruct subdivision connectivity also works on non-manifold meshes, we allow our meshes to be non-manifold.

Note that the quadrisection operator preserves and reflects connected components, i.e., the connected components of a mesh \(M\) are always in \(1 \rightarrow 1\) correspondence with the connected components of the quadrisected mesh \(M^Q\).

4 CONSTRUCTING TILES

The tiles of a mesh \(M = (V, F)\) are best defined by the algorithm used to construct them, which we will describe with the notation introduced in Figure 7. Each face \(f = (v_1, v_2, v_3) \in F\), with three regular edges \(e_{12}, e_{23},\) and \(e_{31}\) has three neighboring triangular faces \(f_{12}, f_{23},\) and \(f_{31}\). Each one of these faces \(f_i\) has a vertex \(v_{ij}\) opposite to the corresponding edge \(e_{ij}\). The tile corresponding to \(f\) is defined by these three vertices \(f' = (v_{12}, v_{23}, v_{31})\). Note that as a mesh the quadrisected is equivalent to the submesh of \(M\) defined by the face \(f\), its three immediate neighbors \(f_{12}, f_{23},\) and \(f_{31}\), and their supporting vertices, which we call a tile set of \(M\).

5 THE COVERING MESH

The covering mesh of a triangular mesh \(M = (V, F)\) is a new triangular mesh \(M^C = (V^C, F^C)\) defined by the vertices and \(E\)-vertices of \(M\). Figure 8 illustrates the algorithm to construct the covering mesh of a mesh in pseudocode.
to vertices. As a result of this paper, holds:

**Theorem 1** A connected triangular mesh \( M = (V, F) \) has quadrisection connectivity, if and only if \( \pi_i : M_i^{Q} \rightarrow M \) is a mesh equivalence for some \( i \).

The proof of the sufficiency is trivial. We restate the necessity as follows:

**Theorem 2** For every connected triangular mesh \( M = (V, F) \), the canonical mapping \( \pi_i : M_i^{Q} \rightarrow M^{Q} \) is a mesh equivalence for some \( i \).

---

Figure 9: In general, four tiles (defined by the blue corners) cover each triangle (red). Note that these four tiles have neither common vertices nor common edges in the covering mesh.

Note that even if the mesh is manifold without boundary, the covering mesh may be non-manifold and with boundary. Also, as illustrated in Figure 9, each face may belong to up to four different tile sets, which is connected to each other by the center position, or one of the three corners in each one of the four tile sets. The number of tile sets covering a face may be less than four, and even zero, though, such as when a fine triangle or some of its immediate neighboring triangles are incident to boundary or singular edges.

The covering mesh operator \( C \), that assigns a triangular mesh \( M = (V, F) \) to a new triangular mesh \( M^C = (V^C, F^C) \) defines a canonical mapping \( \pi : M^C \rightarrow M \), from the quadrisection of \( M^C \) into \( M \). This mapping the covering mesh into connected components \( M_i^{CQ} \), and applies the quadrisection operator to each one of them, we obtain a partition of \( M^{Q} \) into connected components \( M_i^{Q} \). The canonical mapping \( \pi \) restricted to the connected components also define mappings \( \pi_i : M_i^{Q} \rightarrow M \).

Let \( \pi_i \) be a mesh equivalence for some \( i \), and \( \pi^{Q} \) is a mesh equivalence for some \( i \) on \( \pi \).

**Proof 2** Each face of \( M \) defines one tile set in \( M^{Q} \) and a corresponding tile in \( M^{Q} \). Let \( F^2 \) be the set of all these tiles in \( 1 \rightarrow 1 \) correspondence with \( F \). Since the vertices of these tiles are supported on \( V \) of \( M \), the set of vertices \( V^2 \) of these tiles is in \( 1 \rightarrow 1 \) correspondence with the set of vertices \( V \). We have constructed a mesh equivalence between \( M \) and the submesh \( M^2 = (V^2, F^2) \) of \( M^{Q} \), which can extended to an equivalence between the corresponding quadrisection meshes. Since subdivision also preserves connected components, we only need to show that \( M^2 \) is a connected component of \( M^{Q} \). \( M^2 \) is clearly connected, because it is equivalent to \( M \), the result of subdividing \( M \) is connected, and the quadrisection operator does not change the number of connected components. It only remains to be shown that no other tiles are connected to \( M^2 \). But the tiles in \( \pi \) that are not members of \( F^2 \) are supported on \( E \)-vertices, while tiles in \( F^2 \) are all supported on \( V \)-vertices, and so, disconnected.

Theorem 1 is the basis of our algorithm to detect uniform quadrisection connectivity and to reconstruct the subdivision structure, described in pseudocode in Figure 10.

To determine if the mapping \( \pi_i : M_i^{Q} \rightarrow M \) is an equivalence or not it is necessary to construct the quadrisection connected component \( M_i^{Q} \). It is sufficient to count all the vertices and faces of tile sets covered by tiles in \( M_i^{Q} \). Figure 11 shows such an algorithm in pseudocode.

---

Figure 10: Pseudocode of procedure to determine if a mesh has quadrisection connectivity, and to recover the subdivision structure.

```plaintext
procedure IsQuadrisection (M)
    # construct covering mesh
    M^C = CoveringMesh(M)
    # partition M^C into connected mponents
    {M_1^C, ..., M_\text{cc}^C} = ConnectedComponents(M^C)
    # for each connected component
    for i = 1, ..., \text{cc}
        # determine if equivalent to M
        if (IsEquivalence(\pi_i : M_i^{Q} \rightarrow M))
            return M_i^C
    return \emptyset
```

Figure 11: Procedure to determine if the mapping \( \pi_i : M_i^{Q} \rightarrow M \) is an equivalence.

```plaintext
procedure IsEquivalence (\pi_i : M_i^{Q} \rightarrow M)
    # first check the easiest necessary conditions
    if (\( F \neq A F_i^{Q} \)) return false
    if (\( V \neq V_i^{Q} + E_i^{Q} \)) return false
    # initialize binary arrays
    n_v = 0 : v \in V
    n_f = 0 : f \in F
    # traverse faces, subdivide, and count
    for each tile \( f^C \in F_i^C \)
        n_v = 1 for each vertex \( v \) of the tile set covered by \( f^C \)
        n_f = 1 for each face \( f \) of the tile set covered by \( f^C \)
    if (\( F \neq \sum n_f \)) return false
    if (\( V \neq \sum n_v \)) return false
    return true
```

---

Figure 9: In general, four tiles (defined by the blue corners) cover each triangle (red). Note that these four tiles have neither common vertices nor common edges in the covering mesh.

Figure 11 shows such a procedure to determine if the covering mesh operator \( C \) that assigns a triangular mesh \( M = (V, F) \) to a new triangular mesh \( M^C = (V^C, F^C) \) defines a canonical mapping \( \pi : M^C \rightarrow M \).
6 IMPLEMENTATION AND RESULTS

A polygonal mesh is normally specified only by its vertices and faces, such as in the IndexedFaceSet node of the VRML standard [15]. Neither the edges, which contain the incidence relationships among faces, nor the connected components of the mesh are explicitly represented.

An explicit representation of edges is needed both to partition the set of faces into its connected components, and by our tile construction algorithms described in section 4.

Efficient data structures to represent edges of oriented manifold meshes, such as the half-edge data structure [16] or the quad-edge [2] data structure are well known. For non-manifolds meshes, these data structures need extensions [4]. We will assume that the data structure used to represent the set of edges efficiently implements the edge access function \( e(v, w) = e(w, v) \), which given two vertices \( v \) and \( w \), returns the set of incident faces (which may be empty if the two vertices do not correspond to an edge of the mesh). In our implementation, we use a hash table to implement the edge access function. This data structure can be populated (constructed) in linear time by visiting the faces in sequential order.

The full algorithm described by the pseudocode methods shown in figures 5, 8, 10, and 11 has been implemented in C++. Figures 2, and 1 show examples where the algorithm has been run on meshes of moderate size with simple and complex topology.

7 CONCLUSIONS

In this paper we have introduced a conceptually very simple and efficient algorithm to detect quadrisection connectivity in triangular meshes, and we have demonstrated it in a number of examples. As explained in the introduction, this algorithm has important application in modeling systems, and connectivity compression schemes.

In a subsequent paper we plan to extend this algorithm to other uniform subdivision schemes, such as Catmull-Clark, and Doo-Sabin; and to some adaptive subdivision schemes.

REFERENCES