Is This A Quadrisected Mesh ?

Gabriel Taubin

California Institute of Technology[1]

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 strightforward 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.

General Terms: Subdivision surfaces, 3D Geometry Compression, Algorithms, Graphics.

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 por 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 conectivity 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

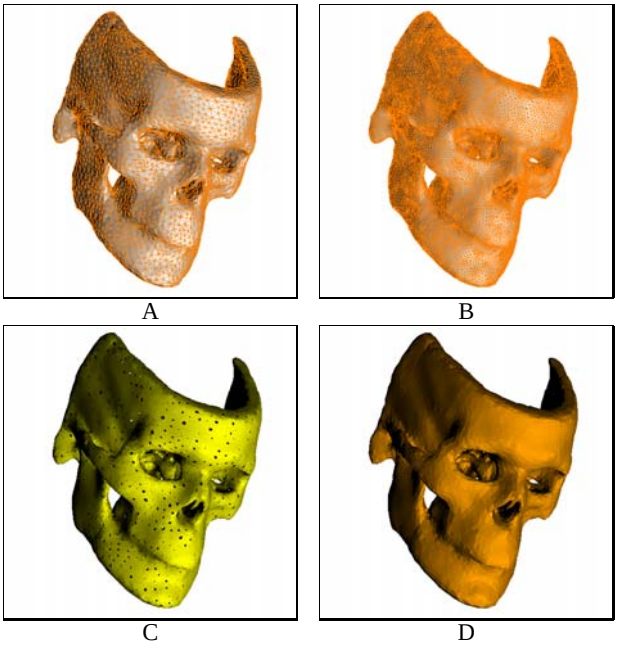


Figure 1: Example with complex topology. A: Coarse triangular mesh ($V = 10952, F = 22104, V - E + F = -100$) is not a quadrisected mesh because its covering mesh is connected. B: Quadrisection of coarse mesh ($V = 44108, F = 88416, V - E + F = -100$). The clustering mesh of this quadrisected mesh has two connected components. Rendering of edges has been turned off. C: First component of clustering mesh ($V = 33156, F = 66312 V - E + F = -24084$). D: Second component of clustering mesh equivalent to coarse mesh.

[1]California Institute of Technology, Department of Electrical Engineering, MS-136-93, Pasadena, CA 91125, taubin@caltech.edu. On sabbatical from IBM T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598

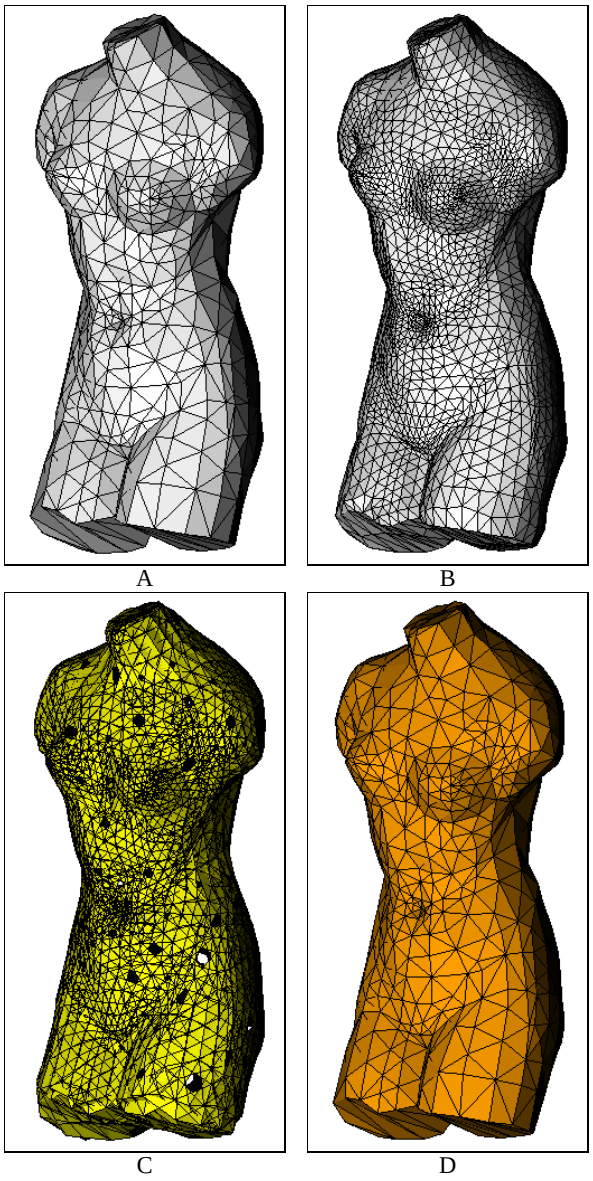


Figure 2: Algorithm overview. A: Coarse mesh ($V = 711, F = 1418, V - E + F = 2$). B: Quadrisected mesh ($V = 2838, , F = 5672, V - E + F = 2$). The clustering mesh of this quadrisected mesh has two connected components. C: First component of clustering mesh ($V = 2127, F = 4254, V - E + F = -1368$). Note the large number of holes and face overlap. D: Second component of clustering mesh equivalent to coarse mesh.

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 *progresive* 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 quadrisected 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 quadrisected mesh is function of the size of the coarse mesh.

T	B	B/T	T	B	B/T
4	64	16.00	4,096	1,704	0.42
16	96	6.00	16,384	3,744	0.23
64	192	3.00	65,536	8,192	0.13
256	384	1.50	262,144	18,248	0.07
1,024	784	0.77			

Table 1: Cost of encoding the connectivity of a tetrahedron and eight meshes constructed by recursive triangle quadrisection with the MPEG-4 3D Mesh coder. T: number of mesh triangles, B: cost to encode connectivity in bits, B/T average cost to encode connectivity, in bits per triangle.

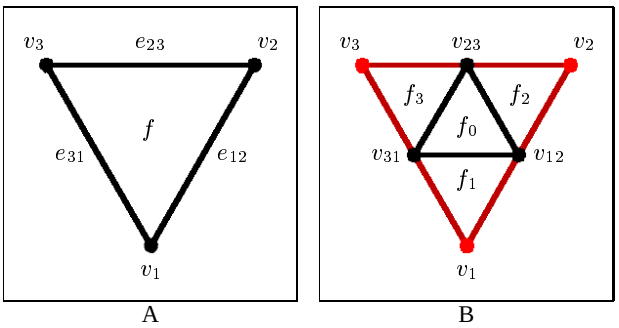


Figure 3: Notation used for triangle quadrisection. A: A triangle $f = (v_1, v_2, v_3)$ with edges edges $e_{12} = e(v_1, v_2)$, $e_{23} = e(v_2, v_3)$, and $e_{31} = e(v_3, v_1)$. B: The quadrisected triangle with V-vertices v_1, v_2, and v_3, E-vertices v_{12}, v_{23}, and v_{31}, and faces $f_0 = (v_{12}, v_{23}, v_{31})$, $f_1 = (v_1, v_{12}, v_{31})$, $f_2 = (v_2, v_{23}, v_{12})$, and $f_3 = (v_3, v_{31}, v_{23})$.

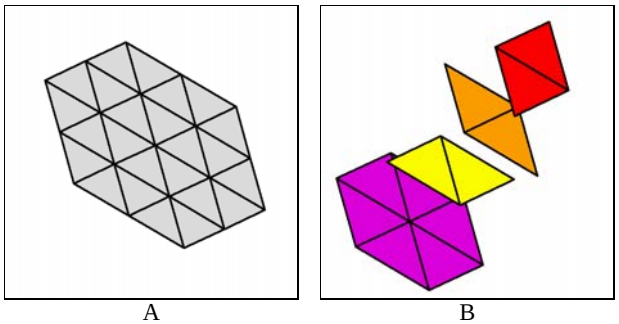
The method introduced in this paper, to detect uniform subdivision connectivity and to reconstruct the subdivision structure, can be used to minimize the cost of encoding the connectivity information of a fine mesh with uniform subdivision connectivity by representing the connectivity information as 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 quadrisecting it. We call *tile set* a group of four connected triangles with the same connectivity as the result of quadrisecting one triangle, i.e., four trianges 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-



A B

Figure 4: The covering mesh of a triangular mesh. A: quadrisected mesh. B: covering mesh with color-coded connected components. The connected components are artificially displaced in space. The quadrisection of the purple connected component is equivalent to the input mesh.

sal procedure stops, not all the faces are covered by tile sets, a new traversal must be started from a tile set not visited during the previous traversal. Since each triangle is covered by up to four tile sets, we may need to restart the traversal up to four times to decide if the fine mesh has subdivision structure or not. Non-manifold situations are difficult to handle, and may require backtracking.

Instead of this sequential algorithm, we propose an alternative global approach, where all the traversal is avoided, and replaced by a parallelizable algorithm to construct the *covering mesh* of a triangular mesh. The covering mesh of a triangle mesh is composed of triangular faces called *tiles* supported on the same set of vertices. The tiles are in $1-1$ correspondence with all the tile sets that can be constructed in the original mesh, and when quadrisected, each one has the same connectivity as the corresponding tile set.

Our algorithm, motivated by the concept of *covering surface* in Algebraic Topology [7], is based on a theorem that states that a triangular mesh is a quadrisected mesh if and only if it is equivalent to the quadrisection of one connected components of its covering mesh. Figure 4 illustrates this construction for a simple quadrisected mesh. The connected components of the covering mesh are painted in different colors, and displaced in space (vertex positions are irrelevant). Note that the quadrisection of the purple connected component is equivalent to the input mesh.

There is a cannonical *mapping* between the covering mesh and the corresponding triangular mesh, that assigns vertices to vertices and faces to faces. Establishing whether or not the quadrisection of a given connected component of the covering mesh is equivalent to the original mesh reduces to simple and linear counting algorithms that determine if the cannonical mapping restricted to the connected component is $1-1$ and onto or not.

3 POLYGONAL MESHES

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 skiped 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 cyclical permutations of its corners are considered identical, i.e., $f = (v_i, \ldots, v_n, v_1, \ldots, v_{i-1})$ 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 ∈ F}
    # traversal
    for each regular edge e connecting f1 and f2
        # if f1 and f2 are in different partitions
        if (P.find(f1) ≠ P.find(f2))
            # join corresponding partitions
            P.union(f1, f2)
    # find sets of supporting vertices
    Mi = (Vi, Fi) i = 1, ..., cc
    # return submeshes
    return P = {M1, ..., Mcc}
```

Figure 5: Procedure to construct the connected components of a mesh.

Vertices not contained in any face are called *isolated*. An *edge* e is an un-ordered 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 asume 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_n are *connected* if we can find faces $f_2, \ldots, f_{n-1}$ such that each face f_i 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* $\phi : M_1 \to M_2$ from a first mesh M_1 into a second mesh M_2 is defined by a *vertex function* $\phi_V : V_1 \to V_2$ and a *face function* $\phi_F : F_1 \to F_2$ 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.,

$$\phi_F(f) = (\phi_V(v_1), \ldots, \phi_V(v_n)) \in F_2 \ .$$

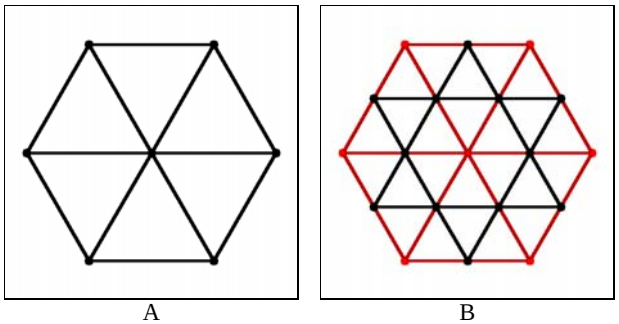


Figure 6: Example of triangle mesh quadrisection. A: coarse mesh; B: quadrisected mesh with V-vertices (red) corresponding to vertices of the coarse mesh, and E-vertices (black).

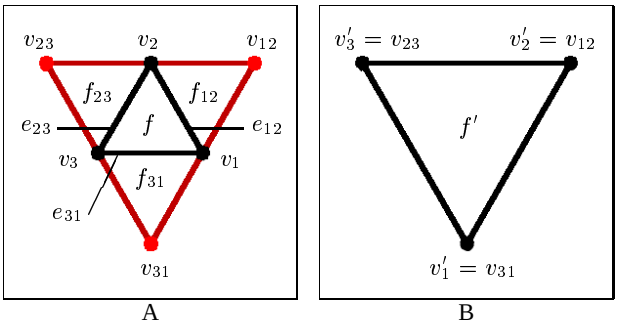


Figure 7: Notation used for tile construction. A: A tile set. B: The corresponding tile.

```
procedure CoveringMesh (M)
    # initialization
    F^C = ∅
    # construct tiles
    for each face f = (v1, v2, v3) ∈ F
        # if incident edges are regular
        if (e12, e23, e31 are regular)
            # construct tile and append to set of faces
            append (v12, v23, v31) to F^C
    M^C = (V, F^C)
    return M^C
```

Figure 8: Procedure to construct the covering mesh of a triangular mesh. Notation as in Figure 7.

Equivalence Two meshes M_1 and M_2 are called *equivalent* if a mapping $\phi : M_1 \rightarrow M_2$ exists such that both ϕ_V and ϕ_F are $1-1$ and onto functions. In such case the mapping ϕ is called a *mesh equivalence*.

Note that since the sets of vertices and faces are finite, the mapping ϕ 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 $1-1$ 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 is 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 $M^Q = (V^Q, F^Q)$, and defines a mapping $\iota_M : M \rightarrow M^Q$, 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 $1-1$ but not onto. With respect to the number of vertices, edges, and faces, the following relations hold:

$$\begin{cases} V^Q &= V + E \\ E^Q &= 2E + 3F \\ F^Q &= 4F \end{cases} \quad (1)$$

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 $1-1$ 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_{ij} 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 tile 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 suporting 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 tiles of M. Figure 8 illustrates the algorithm to construct the covering mesh of a mesh in pseudocode.

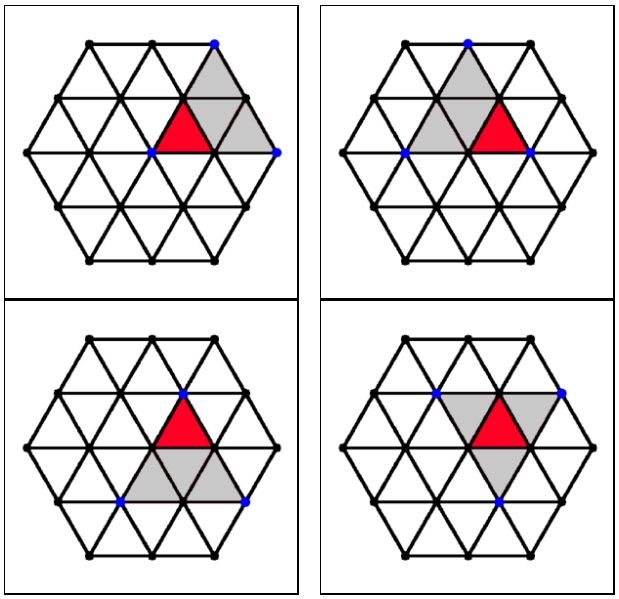


Figure 9: In general, four tiles (defined by the blue corners) cover each triangle (red). Note that these four tiles have neither common vertices nor common edges in the covering mesh.

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 of a triangular mesh, where the given fine triangle occupies either 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 cannonical mapping $\pi : M^{CQ} \rightarrow M$, from the quadrisection of M^C into M. If we partition the covering mesh into connected components $M_1^C, \ldots M_{\mathsf{CC}}^C$, and apply the quadrisection operator to each one of them, we obtain a partition of M^{CQ} into connected components $M_1^{CQ}, \ldots M_{\mathsf{CC}}^{CQ}$. The cannonical mapping π restricted to the connected components also define mappings $\pi_i : M_i^{CQ} \rightarrow M$. Note that in general, the coresponding vertex and face functions of these mappings are neither $1-1$ nor onto. For example, not all the faces of M may be covered by faces of M_i^{CQ}, and up to four faces of M_i^{CQ} may be covering the same face of M. With respect to vertices, restricted to the V-vertices of M_i^{CQ}, the mapping π_i is $1-1$, but not necesarily so when restricted to the E vertices; and some vertices of M may not correspond to any vertex of M_i^{CQ}.

However, the following theorem, which constitutes the main result of this paper, holds:

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

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

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

```
procedure IsQuadrisection (M)
    # construct covering mesh
    M^C = CoveringMesh(M)
    # partition M^C into connected components
    {M_1^C, ..., M_CC^C}=ConnectedComponents(M^C)
    # for each connected component
    for i = 1, ..., cc
        # determine if equivalent to M
        if (IsEquivalence(π_i : M_i^CQ → M))
            return M_i^C
    return ∅
```

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

```
procedure IsEquivalence (π_i : M_i^CQ → M)
    # first check the easiest necessary conditions
    if (F ≠ 4F_i^C) return false
    if (V ≠ V_i^C + E_i^C) return false
    # initialize binary arrays
    n_v = 0 : v ∈ V
    n_f = 0 : f ∈ F
    # traverse faces, subdivide, and count
    for each tile f^C ∈ 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 ≠ Σ_f n_f) return false
    if (V ≠ Σ_v n_v) return false
    return true
```

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

Proof 2 Each face of M defines one tile set in M^Q and a corresponding tile in M^{QC}. Let F^2 be the set of all these tiles in $1-1$ correspondence with F. Since the vertices of these tiles are supported on V-vertices of M, the set of vertices V^2 of these tiles is in $1-1$ 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^{QC}, which can extended to an equivalence between the corresponding quadrisected meshes. Since subdivision also preserves conected components, we only need to show that M^2 is a connected component of M^{QC}. M^2 is clearly connected, because it is equivalent to M, the result of subdividing M is M^Q, which 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 F^{QC} 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 $(\pi_i : M_i^{CQ} \rightarrow M)$ is an equivalence or not it is not necessary to construct the quadrisected connected component M_i^{CQ}. It is sufficient to count all the vertices and faces of tile sets covered by tiles in M_i^C. Figure 11 shows such an algorithm in pseudocode.

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 asume 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 sequencial 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

[1] H Bierman, A. Levin, and D. Zorin. Piecewise smooth subdivision surfaces with normal control. In *Siggraph'2000 Conference Proceedings*, pages 113–120, July 2000.

[2] L.J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, 1985.

[3] P. Heckbert. Course 25: Multiresolution surface modeling. Siggraph'97 Course Notes, August 1997.

[4] L. Kettner. Designing a data structure for polyhedral surfaces. In *14th. Annual ACM Symposium on Computational Geometry*, pages 146–154, 1998.

[5] A. Khodakovsky, P. Schröder, and W. Sweldens. Progressive geometry compression. In *Siggraph'2000 Conference Proceedings*, pages 271–278, July 2000.

[6] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Dept. of Mathematics, University of Utah, August 1987.

[7] W.S. Massey. *A basic course in algebraic topology*. Springer-Verlag, New York-Berlin, 1991. ISBN 0-387-97430-X.

[8] Mpeg4-3dmc-coder.

[9] R.E. Tarjan. *Data Structures and Network Algorithms*. Number 44 in CBMS-NSF Regional Conference Series in Applied Mathematics. SIAM, 1983.

[10] G. Taubin. A signal processing approach to fair surface design. In *Siggraph'95 Conference Proceedings*, pages 351–358, August 1995.

[11] G. Taubin, A. Guéziec, W. Horn, and F. Lazarus. Progressive forest split compression. In *Siggraph'98 Conference Proceedings*, pages 123–132, July 1998.

[12] G. Taubin and J. Rossignac. Geometry Compression through Topological Surgery. *ACM Transactions on Graphics*, 17(2):84–115, April 1998.

[13] G. Taubin and J. Rossignac. Course 38: 3d geometry compression. Siggraph'2000 Course Notes, July 2000.

[14] C. Touma and C. Gotsman. Triangle mesh compression. In *Graphics Interface Conference Proceedings*, Vancouver, June 1998.

[15] The Virtual Reality Modeling Language. ISO/IEC 14772-1, September 1997. `http://www.web3d.org`.

[16] K. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Application*, 5(1):21–40, January 1985.

[17] D. Zorin and P. Schröder. Course 23: Subdivision for modeling and animation. Siggraph'2000 Course Notes, July 2000.

[18] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *Siggraph'97 Conference Proceedings*, pages 259–268, August 1997.