The Visual Computer

Detecting and reconstructing subdivision connectivity

Gabriel Taubin

California Institute of Technology, Department of Electrical Engineering, MS-136-93, Pasadena, CA 91125, USA
E-mail: taubin@caltech.edu

Published online: 3 July 2002

In this paper we introduce fast and efficient *inverse subdivision algorithms*, with linear time and space complexity, to detect and reconstruct uniform Loop, Catmull–Clark, and Doo–Sabin subdivision structure in irregular triangular, quadrilateral, and polygonal meshes. We consider two main applications for these algorithms. The first one is to enable interactive modeling systems that support uniform 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. Our Loop inverse subdivision algorithm is based on global connectivity properties of the *covering mesh*, a concept motivated by the *covering surface* from Algebraic Topology. Although the same approach can be used for other subdivision schemes, such as Catmull–Clark, we present a Catmull–Clark inverse subdivision algorithm based on a much simpler graph-coloring algorithm and a Doo–Sabin inverse subdivision algorithm based on properties of the dual mesh. Straightforward extensions of these approaches to other popular uniform subdivision schemes are also discussed.

Key words: Inverse subdivision – 3D file formats – Geometry compression – Algorithms – Graphics

On sabbatical from IBM T.J. Watson Research Center, Yorktown Heights, N.Y. 10598, USA;
E-mail: taubin@us.ibm.com

1 Introduction

Subdivision surfaces are becoming popular multiresolution representations in modeling and animation (Zorin et al. 1997; Zorin and Schröder 2000). The most popular uniform recursive subdivision schemes are due to Loop (1987), Catmull and Clark (1978), and Doo and Sabin (1978). For example Fig. 1b shows the result of applying Loop's triangle quadrisection scheme (Loop 1987) to the triangular mesh shown in Fig. 1a.

Since the most popular interchange file formats, such as VRML (The Virtual Reality Modeling Language 1997), 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 it limits the distribution of the content. The methods introduced in this paper to detect uniform subdivision connectivity and to reconstruct the subdivision structure solve this problem.

Another application area for the algorithms introduced in this paper is to improve the efficiency of lossless connectivity compression schemes. Most 3D geometry compression techniques for polygonal meshes preserve the connectivity information without loss (Taubin and Rossignac 2000). 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.

Uniform subdivision schemes can be regarded as optimal *progressive* connectivity compression schemes, because the cost of encoding each subdivision step is constant (Taubin et al. 1998). Unfortunately, although many commercial modeling and animation packages use subdivision surfaces as one of their main surface representation primitives (Zorin and Schröder 2000), current connectivity compression schemes (Taubin and Rossignac 2000) do not detect subdivision connectivity. As a result, the cost of encoding each uniform subdivision step is normally a 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 (Information Technology 1999) in single-resolution mode (Taubin and Rossignac 1998). 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

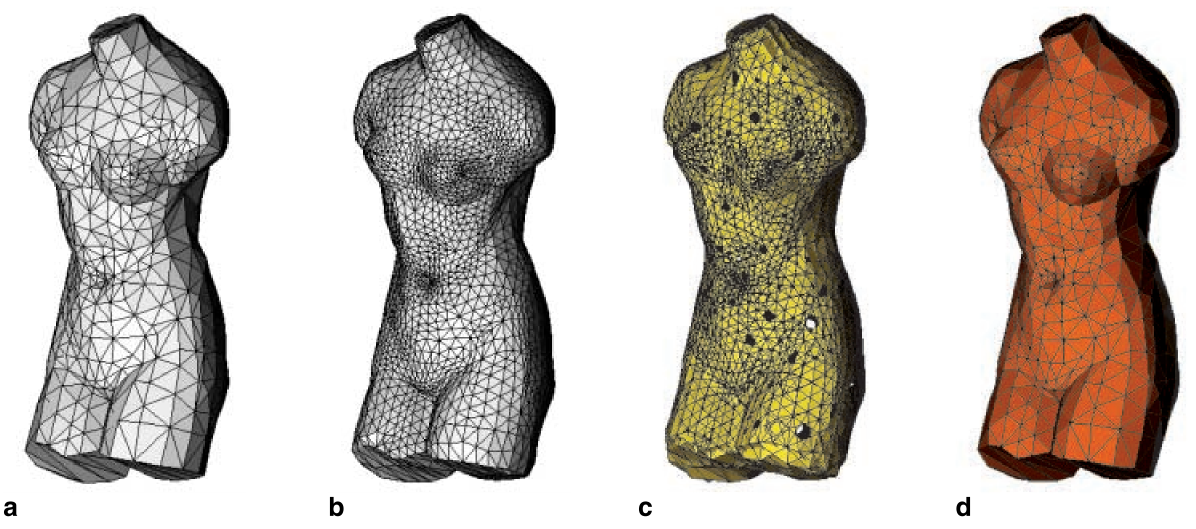


Fig. 1a–d. Loop inverse subdivision algorithm. **a** Coarse mesh ($V = 711$ $F = 1418$ $V - E + F = 2$). **b** Quadrisected mesh ($V = 2838$ $F = 5672$ $V - E + F = 2$). The covering mesh of this quadrisected mesh has two connected components. **c** First component of covering mesh ($V = 2127$ $F = 4254$ $V - E + F = -1368$). Note the large number of holes and face overlap. **d** Second component of covering mesh equivalent to coarse mesh

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 of encoding connectivity, in bits per triangle

T	B	B/T	T	B	B/T
4	64	16.00	4096	1704	0.42
16	96	6.00	16 384	3744	0.23
64	192	3.00	65 536	8192	0.13
256	384	1.50	262 144	18 248	0.07
1024	784	0.77			

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 (Touma and Gotsman 1998) are more efficient at compressing these quasi-regular meshes, but still the incremental cost of encoding a quadrisected mesh is a function of the size of the coarse mesh.

Lossy connectivity compression schemes can be used in some cases, such as when a 3D polygonal mesh is large and generated by over-sampling a relatively smooth surface with simple topology, such as those produced by 3D scanning systems. Simplification algorithms (Heckbert 1997) can be regarded as lossy connectivity compression techniques, but 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 (Khodakovsky et al. 2000; Guskov et al. 2000). Remeshing algorithms do not replace lossless connectivity compression schemes, because they do not produce good compression results when the topology is not simple and replacing the connectivity of the mesh is not always acceptable. The algorithms introduced in this paper can be used to compress the connectivity of these remeshed meshes in the case where they were saved without the subdivision structure.

The methods 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.

The paper is organized as follows: In Sect. 2 we introduce some definitions and nomenclature about polygonal meshes, which can be skipped on a first reading. In Sect. 3 we describe our Loop inverse subdivision algorithm, in Sect. 4 the Catmull–Clark algorithm, and in Sect. 5 the Doo–Sabin algorithm. Finally, we present our conclusions in Sect. 6.

2 Polygonal meshes

In this section we introduce some definitions, 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.

A polygonal mesh is defined by the position of the vertices (geometry), by the association between each face and its sustaining vertices (connectivity); and by optional colors, normals and texture coordinates (properties). The Loop inverse subdivision algorithm applies to the connectivity of triangular meshes (T-meshes). The Catmull–Clark inverse subdivision algorithm applies to the connectivity of quadrilateral meshes (Q-meshes), and the Doo–Sabin inverse subdivision algorithm to the connectivity of general polygonal meshes (P-meshes).

Connectivity. The connectivity of a polygonal mesh, M, is defined by the incidence relationships existing among its V vertices, E edges, and F faces. Since in this paper we only operate on the connectivity, when we refer to a *mesh*, we will mean the connectivity of a polygonal mesh. 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 $f = (v_1, \dots, v_n)$ is a face, all the cyclical permutations of its corners are considered identical, i.e., $f = (v_i, \dots, v_n, v_1, \dots, v_{i-1})$ for $i = 1, \dots, n$. Multiple connected faces (faces with holes) are not representable. Vertices not contained in any face are called *isolated*. An *edge*, e, is an un-ordered pair $e = \{v_1, v_2\}$ of different vertices that are consecutive in one or more faces of the mesh. The *graph* of a polygonal mesh is the graph defined by the mesh vertices as graph vertices and the mesh edges as graph edges. We will also denote the edge $e = \{v_1, v_2, f_1, \dots, f_n\}$, where $f_1, \dots, f_n$ are the incident mesh faces.

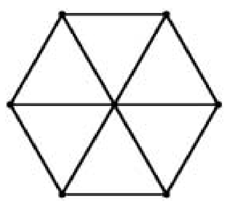
Duality and manifolds. We classify the edges and vertices of a polygonal mesh as *boundary*, *regular*, or *singular*. A boundary mesh edge has exactly one incident face, a regular mesh edge has exactly two incident faces, and a singular mesh edge has three or more incident faces. The *dual graph* of a polygonal mesh is the graph defined by the mesh faces as graph vertices, and the *regular mesh edges* as graph edges. The *edge star* of a vertex, v, is the set of edges, $E(v)$, incident to the vertex. The *vertex star* of a vertex, v, is the set of vertices, $V(v)$, connected to the vertex through an edge. The *face star* of a vertex, v, is the set of faces, $F(v)$, incident to the vertex. A mesh vertex is a *boundary* vertex if its edge star is composed of exactly two boundary edges, and no singular edge, that form an open path in the dual graph. A mesh vertex is *regular* if its edge star is composed of only regular edges that form a loop (of mesh faces) in the dual graph. A *singular* vertex is a vertex that is neither regular nor boundary. A mesh is *manifold* if none of its vertices and edges are singular. It is a *manifold without boundary* if all its vertices and edges are regular. The *dual mesh* of a polygonal mesh is defined by the mesh faces incident to regular vertices as dual vertices and the cycles of primal faces associated with primal regular vertices as dual faces. Note that, for a manifold without boundary, the dual mesh of its dual mesh has the same connectivity as the original mesh. And the converse of this statement is also true.

Orientation. The concepts of orientability and orientation, which only apply to manifold meshes (non-manifold meshes are non-orientable), play no role in this paper and will be ignored.

Connected components. We say that two faces, f_1 and f_n, are *connected* if we can find faces $f_2, \dots, f_{n-1}$ such that each face, f_i, shares an 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 in the set of faces, F, that defines a partition into disjoint *connected components*, $F_1, \dots, F_{\mathrm{cc}}$. Each 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 that the subsets of vertices, $V_1, \dots, V_{\mathrm{cc}}$, are not necessarily disjoint, i.e., dif-

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

2


3a

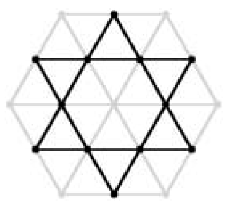

3b

Fig. 2. Procedure for constructing the connected components of a mesh

Fig. 3a,b. Example of triangle mesh quadrisection. **a** Coarse mesh; **b** quadrisected mesh with V-vertices (*gray*), corresponding to vertices of the coarse mesh, and E-vertices (*black*)

ferent connected components may share vertices, but they can be made disjoint by vertex duplication (Gueziec et al. 1998). 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 (1983) fast union-find data structure can be used to partition the set of faces of a mesh into its connected components. It is described in pseudocode in Fig. 2. 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.,

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

Equivalence. Two meshes M_1 and M_2 are called *equivalent* if a mapping $\phi : M_1 \to M_2$ 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 $F_1 = F_2$.

A simple linear time and space algorithm can be used to count the number of elements of the set

$$\{\psi(a) : a \in A\} \subset B ,$$

which can be used to determine if a function $\psi : A \to B$ between two finite sets is onto or not. Create a binary (0, 1) array with elements in correspondence with the elements of B, and initialize 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 3 shows an example of a *fine mesh* (Fig. 3b) with 24 triangles resulting from quadrisecting a *coarse mesh* (Fig. 3a) 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 Fig. 3b. 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 vertex function which assigns each vertex of M into a V-vertex of M^Q, and a face function that assigns each face of M into the center face of the corresponding quadrisected face. Both functions are 1–1 but not onto; they do not define the mapping $M \to M^Q$, though, because they do not satisfy the additional property required by the definition of mapping given above.
With respect to the number of vertices, edges, and faces, the following relations hold:

$$\begin{cases} V^Q = V + E\ , \\ E^Q = 2E + 3F\ , \\ F^Q = 4F\ . \end{cases} \tag{1}$$

The quadrisection operator is one of many subdivision schemes that introduces new vertices along the edges of the coarse mesh, and it replaces the coarse faces with 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) (Biermann et al. 2000). 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 the mesh M are always in 1–1 correspondence with the connected components of the quadrisected mesh M^Q.

3 Loop inverse subdivision

Figure 4 shows the result of quadrisecting a triangle. We call the *tile set* a group of four connected triangles with the same connectivity as the result of quadrisecting one triangle, i.e., four triangles connected as in Fig. 4b. 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

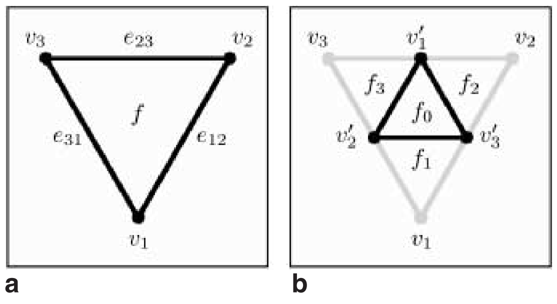
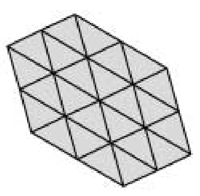


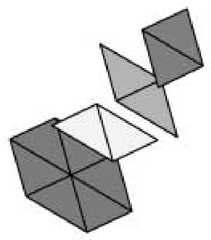
Fig. 4a,b. Quadrisecting a triangle. **a** Triangle $f = (v_1, v_2, v_3)$ with edges $e_{12} = e(v_1, v_2)$, $e_{23} = e(v_2, v_3)$, and $e_{31} = e(v_3, v_1)$. **b** The quadrisected triangle with V-vertices v_1, v_2, and v_3, E-vertices v'_3, v'_1, and v'_2, and faces $f_0 = (v'_3, v'_1, v'_2)$, $f_1 = (v_1, v'_3, v'_2)$, $f_2 = (v_2, v'_1, v'_3)$, and $f_3 = (v_3, v'_2, v'_1)$

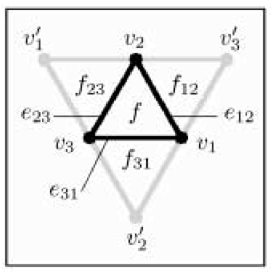
of the corner triangles not shared with the center triangle.
In a naive traversal algorithm, to solve our problem tile sets are sequentially constructed while the mesh is traversed, say in depth-first order, trying to cover it avoiding tile overlaps, i.e., every face is allowed to belong to at most one tile set. If, when the mesh traversal 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.

3.1 Algorithm overview

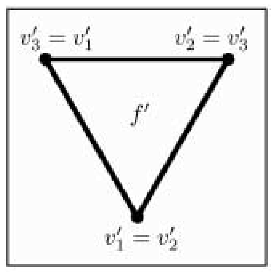
Instead of this sequential algorithm, we propose an alternative global approach, where all the traversal is avoided and replaced by a parallelizable algorithm to construct the *covering mesh* of a triangular mesh. Our algorithm has the same complexity as the sequential algorithm, but it is conceptually simpler, and all the book-keeping required to support backtracking is avoided. But our algorithm is potentially faster than the sequential algorithm, because the sequential algorithm requires four traversals to give a negative answer.
The covering mesh of a triangle mesh is composed of triangular faces called *tiles* supported on the same set


5a


5b



6a



6b

Fig. 5a,b. 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

Fig. 6a,b. Notation used for tile construction. **a** A tile set. **b** The corresponding tile

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 Loop inverse subdivision algorithm, motivated by the concept of the *covering surface* in Algebraic Topology (Massey 1991), 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 component of its covering mesh. Figure 5 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 canonical *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 canonical mapping restricted to the connected component is 1–1 and onto or not.

3.2 Constructing tiles on T-meshes

The *tiles* of a triangular mesh, $M = (V, F)$, are best defined by the algorithm used to construct them, which we will describe with the notation introduced in Fig. 6. 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}. If none of the three triangle vertices v_1, v_2, v_3 has valence 3, a tile corresponding to f can be constructed. It is defined by these three vertices $f' = (v_3', v_1', v_2')$, which are different to each other. 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 supporting vertices. Note that whenever one of the three vertices v_1, v_2, or v_3 has valency 3 (i.e., exactly three neighbours) two of the three *tile vertices* v_1', v_2', v_3' are identical and do not define a triangle tile.

3.3 The covering mesh of a T-mesh

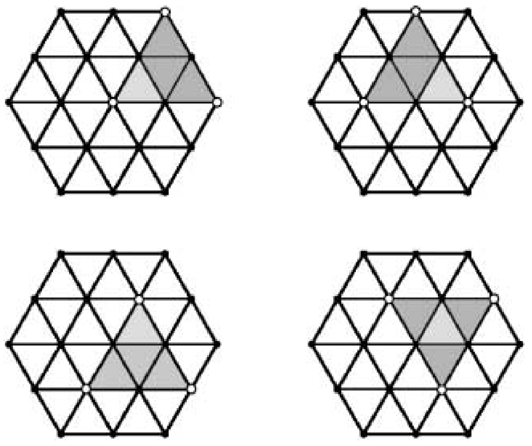
The *covering mesh* of the triangular mesh $M = (V, F)$ is the new triangular mesh $M^C = (V^C, F^C)$ defined by the vertices and tiles of M. Figure 7 illustrates the algorithm to construct the covering mesh of a mesh in pseudocode.

Note that even if the mesh is manifold without boundary, the covering mesh may be non-manifold and with boundary. Also, as illustrated in Fig. 8, 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, such as when a fine triangle or some of its immediate neighboring triangles are incident to boundary or singular edges.

```
procedure CoveringMesh ($M$)
    # initialization
    $F^C = \emptyset$
    # construct tiles
    for each face $f = (v_1, v_2, v_3) \in F$
        # if incident edges are regular
        if ($e_{12}, e_{23}, e_{31}$ are regular)
            # construct tile and append to set of faces
            append $(v'_3, v'_1, v'_2)$ to $F^C$
    $M^C = (V, F^C)$
    return $M^C$
```

7

Fig. 7. Procedure to construct the covering mesh of a triangular mesh. Notation as in Fig. 6



8

Fig. 8. In general, four tiles (defined by the *white corners*) cover each triangle (*light gray*). Note that these four tiles have neither common vertices nor common edges in the covering mesh

The *covering mesh operator*, C, that assigns the triangular mesh $M = (V, F)$ to the new triangular mesh $M^C = (V^C, F^C)$ defines the canonical mapping $\pi : M^{CQ} \to M$ from the quadrisection of M^C into M. If we partition the covering mesh into connected components, $M_1^C, \dots M_{\mathrm{cc}}^C$, and apply the quadrisection operator to each of them, we obtain a partition of M^{CQ} into connected components $M_1^{CQ}, \dots M_{\mathrm{cc}}^{CQ}$. The canonical mapping π restricted to the connected components also defines mappings $\pi_i : M_i^{CQ} \to M$. Note that, in general, the corresponding 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 this is not necessarily so when restricted to the E vertices; in addition, some vertices of M may not correspond to any vertex of M_i^{CQ}.

3.4 Theorems

The following theorem constitutes the first main result of this paper:

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

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

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

Proof. 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^Q, 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 be extended to an equivalence between the corresponding quadrisected meshes. Since subdivision also preserves connected 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 Fig. 9.

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

9

```
procedure IsEquivalence ($\pi_i : M_i^{CQ} \to M$)
    # first check the easiest necessary conditions
    if ($F \neq 4F_i^C$) return false
    if ($V \neq V_i^C + E_i^C$) 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 ($F \neq \sum_f n_f$) return false
    if ($V \neq \sum_v n_v$) return false
    return true
```

10

Fig. 9. Pseudocode of the procedure to determine if a mesh has quadrisection connectivity and to recover the subdivision structure

Fig. 10. Procedure to determine whether the mapping $\pi_i : M_i^{CQ} \to M$ is an equivalence

To determine whether the mapping $\pi_i : M_i^{CQ} \to M$ 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 the tile sets covered by tiles in M_i^C. Figure 10 shows such an algorithm in pseudocode.

3.5 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 (The Virtual Reality Modeling Language 1997). 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 Sect. 3.2.

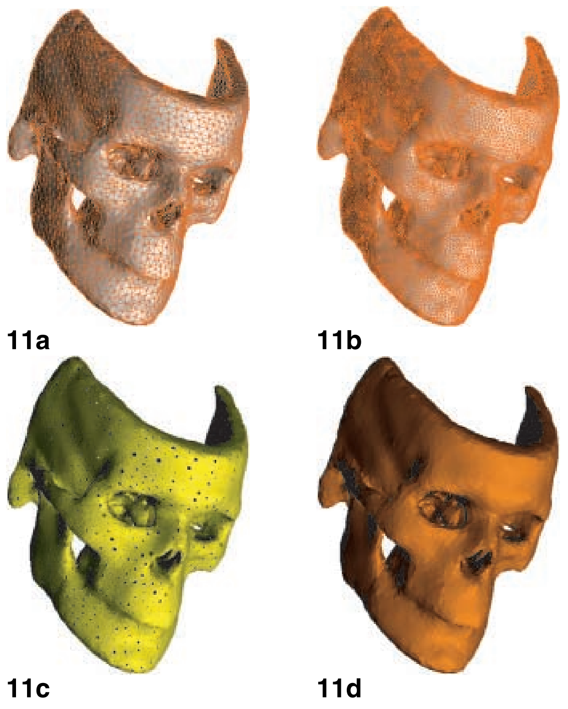
Efficient data structures to represent edges of oriented manifold meshes, such as the *half-edge* data structure (Weiler 1985) or the *quad-edge* data structure (Guibas and Stolfi 1985), are well known. For non-manifolds meshes, these data structures need extensions (Kettner 1998). We will assume that the data structure used to represent the set of edges efficiently implements the *edge access* function $e(v, w) = e(w, v)$, which when 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 Figs. 2, 7, 9, and 10 has been implemented in C++. Figures 1 and 11 show examples where the algorithm has been run on meshes of moderate size with simple and complex topology.

4 Catmull–Clark inverse subdivision

Figure 12a shows a portion of a polygonal mesh, and Fig. 12d shows the result of subdividing the mesh according to the Catmull–Clark (CC) scheme. The vertices of the original mesh correspond to a subset of the vertices of the refined mesh. We call these vertices V-vertices. The remaining vertices of the refined mesh correspond to faces (F-vertices), and to edges (E-vertices) of the original mesh. The faces of the refined mesh are all quadrilaterals and correspond to corners (incident vertex-face pairs) of the original mesh. They are constructed by connecting each F-vertex to all the E-vertices associated with edges of the corresponding face.

A tiling approach, similar to the one used in the Loop inverse subdivision algorithm, can be followed to detect and reconstruct CC subdivision.



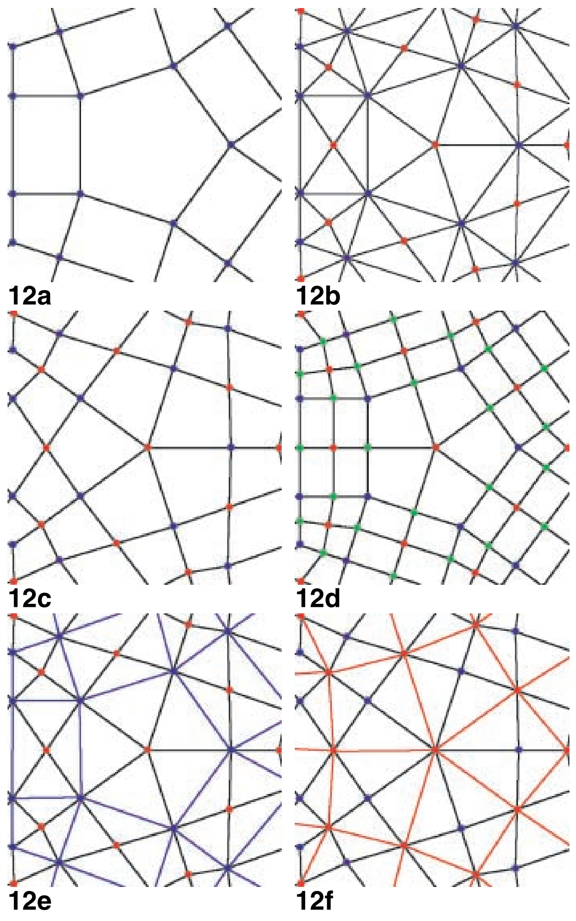


Fig. 11a–d. Example with complex topology. **a** Coarse triangular mesh ($V = 10\,952$, $F = 22\,104$, $V - E + F = -100$) is not a quadrisected mesh because its covering mesh is connected. **b** Quadrisection of a coarse mesh ($V = 44\,108$, $F = 88\,416$, $V - E + F = -100$). The covering mesh of this quadrisected mesh has two connected components. Rendering of edges has been turned off. **c** First component of covering mesh ($V = 33\,156$, $F = 66\,312 V - E + F = -24\,084$). **d** Second component of covering mesh equivalent to coarse mesh

Fig. 12a–f. $\sqrt{\text{CC}}$ and Catmull–Clark (CC) subdivision. **a** A coarse mesh. **b** The faces are triangulated by connecting the new face vertices (*red*) to the original vertices (*blue*). **c** The $\sqrt{\text{CC}}$ connectivity is obtained by removing the original edges from **(b)**. **d** The CC connectivity is obtained after a second $\sqrt{\text{CC}}$ refinement step. Once the vertices have been colored, **e** the primal connectivity can be recovered from the $\sqrt{\text{CC}}$ connectivity by inserting the primal diagonals and removing the dual vertices, and **f** the dual connectivity can be recovered by inserting the dual diagonals and removing the primal vertices

A *tile set* of a quadrilateral mesh is defined by a regular vertex and all its incident faces. The corresponding *tile* is constructed by removing the edges incident to the regular vertex and joining all the incident quadrilaterals into a single face. The *covering mesh* of a quadrilateral mesh is defined by mesh tiles and the mesh vertices that are corners of tiles. A similar theorem can be formulated and proved, but we leave this to the reader. The only algorithm that changes is the one to determine equivalence between a connected component of the covering mesh and the original mesh, but similar counting arguments can be used.

If the mesh is a manifold without boundary the coloring of vertices in Fig. 12d suggest a much simpler graph coloring approach, described below. But

CC subdivision can be applied to any polygonal mesh, including manifolds with boundary and non-manifold edges, and the mesh resulting from the subdivision process has the same topology and singularities. In these cases we cannot base the CC inverse subdivision algorithm on the $\sqrt{\text{CC}}$ inverse subdivision algorithm, and we revert to the tiling approach.

It is sufficient to consider connected meshes, because otherwise the algorithm is applied to each connected component.

4.1 Algorithm for manifolds without boundary

As noted by Kobbelt (1996), the operator that transforms the connectivity of a manifold mesh without boundary into its CC connectivity (Catmull and Clark 1978) has a square root denoted $\sqrt{\text{CC}}$. The result of applying this to the connectivity of a manifold mesh without boundary has the vertices and faces of the original mesh as vertices (V-vertices and F-vertices), the edges of the original face as quadrilateral faces, and the vertex-face incident pairs as edges. The *quad-edge* data structure (Guibas and Stolfi 1985) can be used to operate on the $\sqrt{\text{CC}}$ mesh. Figure 12b and c illustrate the construction. Note that if we paint the V-vertices and F-vertices with different colors, all the edges of the $\sqrt{\text{CC}}$ mesh connect vertices of different colors. The converse of this fact is also true and allows us to detect and reconstruct the $\sqrt{\text{CC}}$ subdivision structure:

Theorem 3. *If the vertices of a manifold without boundary quadrilateral mesh can be painted with two colors so that every edge connects two vertices of different color, then the mesh has $\sqrt{\text{CC}}$ connectivity.*

The proof is constructive. We give a sketch here but leave the details to the reader. If red and blue are the two colors used to paint the vertices of the mesh, we construct a mesh on the red vertices (the red mesh) by inserting the red diagonals and removing all the blue vertices and all the original edges. We also construct a mesh on the blue vertices (the blue mesh) by inserting the blue diagonals and removing all the red vertices and all the original edges. Figure 12e and f illustrate this construction. It is easy to verify that the connectivities of both red and blue meshes are manifold without boundary, that the red and blue meshes are the dual of each other, and that the original mesh connectivity can be recovered by applying the $\sqrt{\text{CC}}$ operator to either the red or the blue mesh.

An algorithm to decide whether the vertices of a graph can be painted with two colors so that every edge connects vertices of a different color, and to produce such a painting if possible, can be based on a spanning tree traversal of the graph and a verification step. During the tree traversal we paint the root with one of the colors, and every time we visit a new vertex we paint it with the color different from its parent's. All the edges of the spanning tree satisfy the two-color condition. Finally, we visit all the other edges of the graph and verify whether or not the two-color condition is satisfied.

To detect and reconstruct CC subdivision structure in a manifold without boundary quadrilateral mesh, we first apply the $\sqrt{\text{CC}}$ inverse subdivision algorithm described above. Then we look at the red and blue meshes. If one of them is a manifold without boundary quadrilateral mesh, we apply the $\sqrt{\text{CC}}$ inverse subdivision algorithm to this mesh. If we obtain a positive answer, the result of applying the CC subdivision process to the reconstructed mesh after the second $\sqrt{\text{CC}}$ inverse subdivision step is equal to the original mesh. Note that both the red and blue meshes may be manifold without boundary quadrilateral meshes, in which case we may end up with two results. These two meshes are necessarily the dual of each other.

5 Doo–Sabin inverse subdivision

Since the Doo–Sabin (DS) connectivity of a manifold without boundary polygonal mesh can be obtained as the dual mesh connectivity of the Catmull–Clark connectivity of the original mesh, the DS inverse subdivision algorithm is very simple: if the dual mesh connectivity is composed of quadrilateral faces, apply the Catmull–Clark inverse subdivision algorithm for manifolds without boundary. Otherwise, the mesh does not have DS subdivision structure. Note that this algorithm does not require the explicit construction of the dual mesh of the original mesh. All the painting can be done in the dual graph that is obtained for free once mesh edges are constructed and classified, which must be done first to determine if the mesh is manifold without boundary or not.

6 Conclusions

In this paper we introduced very simple and efficient algorithms to detect Loop, Catmull–Clark, and Doo–Sabin subdivision structure in the connectivity of triangular, quadrilateral, and polygonal meshes, and we demonstrated them in a number of examples. As explained in the introduction, these algorithms have important applications in modeling systems and connectivity compression schemes. In a subsequent paper we plan to study similar algorithms for adaptive subdivision schemes.

References

1. Biermann H, Levin A, Zorin D (2000) Piecewise smooth subdivision surfaces with normal control. In: SIGGRAPH '00 Conference Proceedings. ACM, New York, pp 113–120
2. Catmull E, Clark J (1978) Recursively generated B-spline surfaces on arbitrary topological meshes. Comput Aided Des 10:350–355
3. Doo D, Sabin M (1978) Behaviour of recursive division surfaces near extraordinary points. Comput Aided Des 10:356–360
4. Gueziec A, Taubin G, Lazarus F, Horn W (1998) Converting sets of polygons to manifold surfaces by cutting and stitching. In: IEEE Visualization '98. IEEE, Washington, DC, pp 383–390
5. Guibas LJ, Stolfi J (1985) Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. ACM Trans Graphics 4(2):74–123
6. Guskov I, Vidimče K, Sweldens W, Schröder P (2000) Normal meshes. In: SIGGRAPH '00 Conference Proceedings. ACM, New York, pp 95–102
7. Heckbert P (1997) Course 25: Multiresolution surface modeling. In: SIGGRAPH '97 Course Notes. ACM, New York
8. Information Technology – Generic Coding of Audio-Visual Objects (MPEG-4), Part 2: Visual Objects (1999) ISO/IEC 14496-2; available at http://www.cselt.it/mpeg
9. Kettner L (1998) Designing a data structure for polyhedral surfaces. In: 14th Annual ACM Symposium on Computational Geometry. ACM, New York, pp 146–154
10. Khodakovsky A, Schröder P, Sweldens W (2000) Progressive geometry compression. In: SIGGRAPH '00 Conference Proceedings. ACM, New York, pp 271–278
11. Kobbelt L (1996) Interpolatory subdivision on open quadrilateral nets with arbitrary topology. In: Eurographics '96 Conference Proceedings. Comput Graphics Forum 15:409–420
12. Loop C (1987) Smooth subdivision surfaces based on triangles. Master's thesis, Dept of Mathematics, University of Utah
13. Massey WS (1991) A basic course in algebraic topology. Springer, New York
14. Tarjan RE (1983) Data structures and network algorithms. In: CBMS-NSF Regional Conference Series in Applied Mathematics, Number 44. SIAM, Philadelphia
15. Taubin G, Rossignac J (1998) Geometry Compression through Topological Surgery. ACM Trans Graphics 17(2):84–115
16. Taubin G, Rossignac J (2000) Course 38: 3d geometry compression. In: SIGGRAPH '00 Course Notes. ACM, New York
17. Taubin G, Guéziec A, Horn W, Lazarus F (1998) Progressive forest split compression. In: SIGGRAPH '98 Conference Proceedings. ACM, New York, pp 123–132
18. The Virtual Reality Modeling Language (1997) ISO/IEC 14772-1; available at http://www.web3d.org
19. Touma C, Gotsman C (1998) Triangle mesh compression. In: Graphics Interface Conference Proceedings, Vancouver, pp 26–34
20. Weiler K (1985) Edge-based data structures for solid modeling in curved-surface environments. IEEE Comput Graphics Appl 5(1):21–40
21. Zorin D, Schröder P (2000) Course 23: Subdivision for modeling and animation. In: SIGGRAPH '00 Course Notes. ACM, New York
22. Zorin D, Schröder P, Sweldens W (1997) Interactive multiresolution mesh editing. In: SIGGRAPH '97 Conference Proceedings. ACM, New York, pp 259–268

GABRIEL TAUBIN is a Research Staff Member at the IBM T.J. Watson Research Center. He joined IBM in 1990 as member of the Exploratory Computer Vision group, from 1996 to 2000 he was Manager of the Visual and Geometric Computing Group, and during the 2000-2001 academic year he was on sabbatical at CalTech as Visiting Professor of Electrical Engineering. He earned a Ph.D. degree in Electrical Engineering from Brown University, and a Licenciado en Ciencias Matematicas degree from the University of Buenos Aires, Argentina. His main research interests fall into the following disciplines: Applied Computational Geometry, Computer Graphics, Geometric Modeling, 3D Photography, and Computer Vision. For the last few years his main line of research has been the development of efficient, simple, and mathematically sound algorithms to operate on 3D objects represented as polygonal meshes, with an emphasis on technologies to enable the use of 3D models for Web-based applications. He made significant contributions to 3D capturing and surface reconstruction, modeling, compression, progressive transmission, and display of polygonal meshes. The 3D geometry compression technology that he developed with his group is now part of the MPEG-4 standard, and part of the IBM HotMedia product. He was named IEEE Fellow for his contributions to the development of three-dimensional geometry compression technology and multimedia standards. He is also known for the development of geometric signal processing algorithms for polygonal meshes.