RC22295 (W0201-037) January 4, 2002
Mathematics

IBM Research Report

CONSTRUCTING HAMILTONIAN TRIANGLE STRIPS ON QUADRILATERAL MESHES

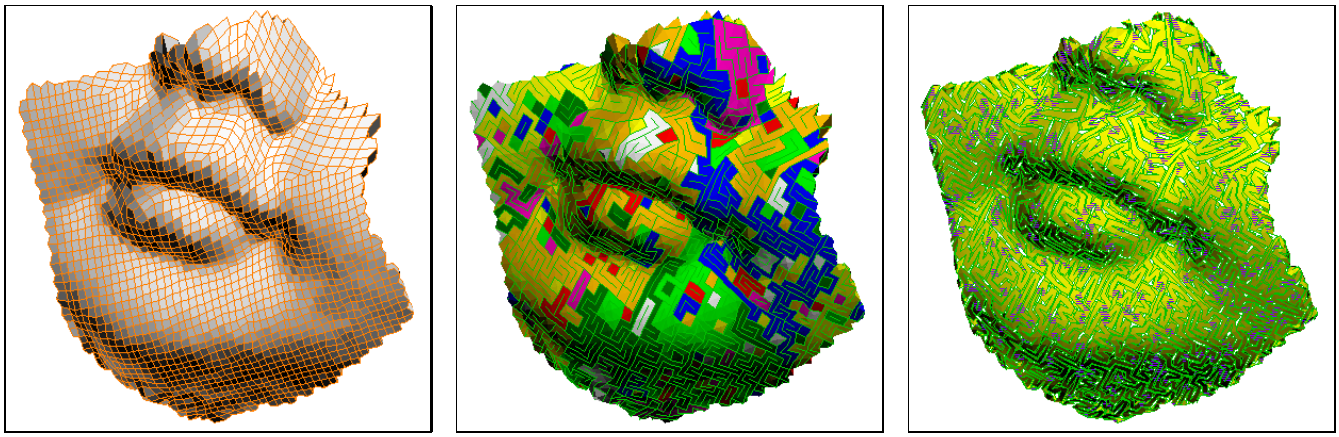
Gabriel Taubin
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

Constructing Hamiltonian Triangle Strips on Quadrilateral Meshes

Gabriel Taubin*

IBM T. J. Watson Research Center



A B C

Figure 1: Every connected manifold quadrilateral mesh without boundary (A) can be represented as a single Hamiltonian generalized triangle strip cycle by splitting each face along one of its diagonals, and connecting the resulting triangles along the original mesh edges. (B) An arbitrary choice of face diagonals produces several cycles. (C) Cycles are then joined to form a single cycle by flipping diagonals.

ABSTRACT

Because of their improved numerical properties, quadrilateral meshes have become a popular representation for finite elements computations and computer animation. In this paper we address the problem of optimally representing quadrilateral meshes as generalized triangle strips (with one swap bit per triangle). This is important because 3D rendering hardware is optimized for rendering triangle meshes transmitted from the CPU to the GPU in the form of triangle strips. We describe simple linear time and space constructive algorithms, where each quadrilateral face is split along one of its two diagonals and the resulting triangles are linked along the original mesh edges. We show that with these algorithms every connected manifold quadrilateral mesh without boundary can be optimally represented as a single Hamiltonian generalized triangle strip cycle in multiple ways, and we discuss simple strategies to tailor the construction for transparent vertex caching.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—display algorithms I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling—surface, solid, and object representations

Keywords: triangle strips, geometry compression, algorithms, graphics.

*IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, taubin@us.ibm.com

1 INTRODUCTION

Because of their simplicity, triangle meshes (T-meshes) are one of the most widely used representations for 3D models in Computer Graphics, and the basic geometric primitive in the 3D rendering pipeline. Since in general a large number of triangles is required to faithfully describe the geometry of a complex surface, a bottleneck problem exists in the transmission of 3D models from the CPU to the GPU (graphics processing unit)[8], and also across other communication channels such as networks. This problem has motivated the search for more efficient encoding schemes for triangle meshes [20]. Instead of specifying each triangle by the coordinates of its three vertices, triangles can be sequentially organized forming *triangle strips* so that every pair of consecutive triangles share a *marching edge*. In this way, only the first triangle of a triangle strip requires the transmission of the coordinates of its three vertices, and each subsequent triangle is specified by only one vector of vertex coordinates.

Triangle strips (T-strips) are widely supported in graphics hardware. In a *sequential* T-strip of length n, composed of n triangles and specified by $n+2$ vertices $v_0, \ldots, v_{n+1}$, the corners of the i-th. triangle are the vertices v_i, v_{i+1}, and v_{i+2}. In a *generalized* T-strips one additional *marching bit* is transmitted for each triangle of the strip other than the last one (i.e., for each marching edge). This bit specifies whether the next triangle is attached to the left or the right edge of the last triangle opposite to the last marching edge. In the sequential triangle strips the marching bits alternate between left and right, and are implicitly specified. From now on, when we refer to a T-strip we will mean a generalized T-strip.

Quadrilateral meshes (Q-meshes) have become a popular representation in modelling and animation [26], in part because of Catmull-Clark Subdivision surfaces [5], and in finite element computations because of their superior numerical properties. In this

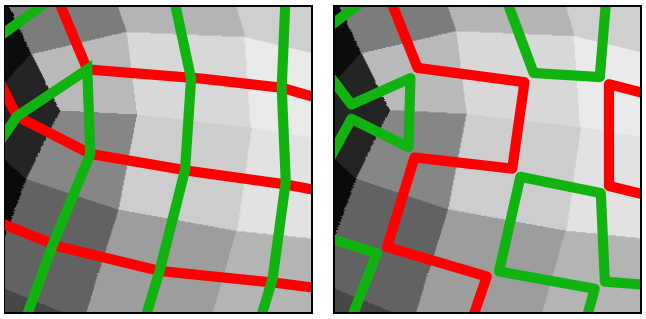


Figure 2: Not every Eulerian circuit corresponds to a Hamiltonian triangulation. Edges that are opposite to each other on a face cannot be contiguous in the Eulerian circuit.

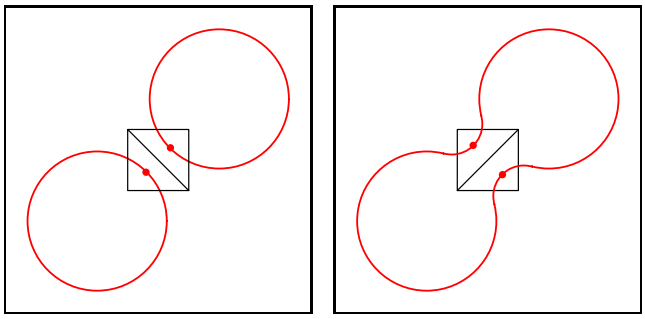


Figure 3: If the two triangles generated by splitting one quadrilateral face belong to different cycles, flipping the diagonal joins the two cycles into a single one.

paper we study the problem of optimally decomposing Q-meshes into T-strips by splitting each of the quadrilateral faces of the mesh along one of its two diagonals and connecting the resulting triangles. Since each T-strip of length n is specified by $n+2$ vertices, the absolute minimum representation cost of one vertex per triangle is achieved when the quadrilateral faces can be split so that all the triangles can be connected forming a single T-strip, with the last two vertices coinciding with the first two.

In this paper we show that this is always possible for connected manifold Q-meshes without boundary, and describe an efficient algorithm to do so. The resulting triangulation is called *Hamiltonian* because it corresponds to the existence of a Hamiltonian cycle in its dual graph.

Hardware support for T-strips requires a cache of size 2 in the graphics processing unit (GPU). Modern GPU's maintain a larger vertex FIFO cache to potentially achieve even higher performance. In the second part of the paper we discuss strategies to maximizing vertex locality in the construction of the Hamiltonian triangle strip to make better use of this cache.

1.1 Related Work

Although we are not aware of any previous solution to the problem that we address in this paper, a number of related problems have been studied. Whitney [25] proved that every planar triangulation without separating triangles has a Hamiltonian cycle. Tutte [23] extended this result to 4-connected planar graphs, and then other authors generalized the result in various ways [17, 21]. Arkin, et. al. [1] proved that every point set has a Hamiltonian triangulation, and showed that the problem of testing whether a triangulation is Hamiltonian or not is NP-complete. As a result, algorithms based on heuristics must be used to decompose triangle meshes into triangle strips. For example, Evans, et. al. [9] present algorithms for constructing triangle strips from partially triangulated models. Bose and Toussaint [4] studied the problem of constructing quadrangulations of point sets, and obtained an alternate method of computing Hamiltonian path triangulations. The problem of generating a quadrangulation of a set of points out of a triangulation has received considerable attention both in the mesh generation and the computational geometry literature. See for example, Ramaswami, et.al. [15] and related references. King et.al. proposes an algorithm to compress quadrilateral meshes [13] where the mesh faces are implicitly connected forming a tree. Velho, et.al. [24] present a refinement scheme that produces a hierarchy of triangle strip decompositions of a triangle mesh.

2 GRAPHS AND MESHES

In this section we introduce some definitions, notation, and facts to make the paper self-contained. We use this material in subsequent sections to formulate our main results more precisely. It can be skipped during a first reading. Just remember that in the rest of the paper, when we refer to a Q-mesh we always mean a connected manifold quadrilateral mesh without boundary (see definitions below).

2.1 Graphs

A *graph* $G = (V, E)$ is composed of a set V of *vertices* and a set E of *edges*. In addition, an *incidence* map $e \mapsto I(e) = \{v_1, v_2\}$ associates each edge with an *unordered pair* of vertices. The two vertices are the *ends* of the edge. Every edge *joins* or *connects* its ends and is *incident* to its ends. Two vertices connected by an edge are *adjacent* or *neighbors*. Two vertices not connected by an edge are *independent*. Two edges sharing exactly one end are *adjacent*. An edge is a *loop* if its two ends are the same. Two edges are *parallel* if they have the same ends. A graph is *simple* if it has no loops and no parallel edges. All the graphs in this paper will be simple. Since the incidence map is one-to-one, every edge of a simple graph is identified with the pair of its ends, and we write $e = \{v_1, v_2\}$.

A *walk* of length n in a graph G is an alternating sequence of vertices and edges $W = (v_1, e_1, \ldots, v_i, e_i, v_{i+1} \ldots, e_n, v_{n+1})$, possibly with repetitions, such that each edge connects the two vertices next to it in the sequence, i.e., $I(e_i) = \{v_i, v_{i+1}\}$. The first element of the sequence is the *beginning*, and the last one is the *end* of the walk. The beginning and end of the walk are the *ends* of the walk. A walk is *closed* if its two ends coincide, and *open* if not. A *trail* is a walk in which all the edges are different. A trail is *Eulerian* if it contains all the edges of the graph. A *path* is a walk with distinct vertices. In particular, every path is a trail. A *circuit* is a closed trail. A *cycle* is a circuit of length n with exactly n different vertices (a closed path). A *Hamiltonian* path (respectively cycle) contains all the vertices of the graph. A graph containing a Hamilton path or cycle is *Hamiltonian*. A graph is k-connected if between any pair of distinct vertices there are k edge-disjoint paths.

2.2 Meshes

A *polygon mesh* (P-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).

The connectivity of a P-mesh M is defined by the incidence relationships existing among its V vertices, E edges, and F faces.

```
BasicHamiltonianSplit (M = (V, E, F))
   # step 1
   split each face along one of its diagonals
   # step 2
   for each face f ∈ F
      if split triangles belong to different cycles
         flip diagonal of f
return
```

Figure 4: Algorithm to split the faces of a Q-mesh along face diagonals so that the resulting T-mesh is Hamiltonian. In the first step, the face diagonals can be chosen in an arbitrary fashion.

A *face* with n *corners* is a sequence of $n \geq 3$ different vertices. All the cyclical permutations of its corners are considered identical. Every face *joins* or *connects* its corners and is *incident* to its corners. 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 mesh is the graph defined by the V mesh vertices as graph vertices, and the E mesh edges as graph edges. The meshes considered in this paper have neither *isolated* vertices (not contained in any face) nor multiple connected faces (faces with holes).

We classify the vertices of a P-mesh as *boundary*, *regular*, or *singular*. A *boundary* mesh edge has exactly one, a *regular* mesh edge has exactly two, and a *singular* mesh edge has three or more incident faces. The *dual graph* of a P-mesh is the graph defined by the mesh faces as graph vertices, and the regular mesh edges as graph edges. In this paper a mesh with no singular edges is *manifold*. It is *manifold without boundary* if in addition all its edges are regular.

Two faces are *connected* if they are the ends of a path in the dual graph. This equivalence relation defines a partition of the set of faces into *connected components*. A mesh with only one connected component is *connected*. An algorithm based on Tarjan's fast union-find data structure [18] can be used to generate the connected components. From now on, all the meshes are connected manifold without boundary.

3 HAMILTONIAN PATHS

With respect to the construction of Hamiltonian paths, the situation for Q-meshes is quite different than for triangle meshes. Hakimi et.al. [11] proved that every 4-connected planar triangulation on V vertices contains not only one, but at least $V/\log_2(V)$ distinct Hamiltonian cycles. The dual graph of a Q-mesh connected manifold without boundary is 4-connected but not necessarily planar, but we will see that nevertheless we have considerable freedom in the construction of Hamiltonian triangulations.

The existence of a Hamiltonian cycle in the dual graph of the mesh would certainly solve the problem we are addressing in this paper. In this case each quadrilateral face can be split into two triangles consecutive in the traversal order defined by the Hamiltonian cycle. After the quadrilateral faces are split, the resulting triangles are linked through an alternating sequence of new diagonal edges and original mesh edges. The remaining original edges become the boundary edges of the resulting Hamiltonian T-strip cycle.

Unfortunately, there exist non-Hamiltonian Q-meshes. This is true even though determining the existence a Hamiltonian cycle in the dual graph of a Q-mesh is linear-time solvable [6]. In this case the alternative is to structure the faces of the Q-mesh as a tree with long runs and few branching nodes. In the algorithm introduced by King et.al. to compress the connectivity of Q-meshes [13], the

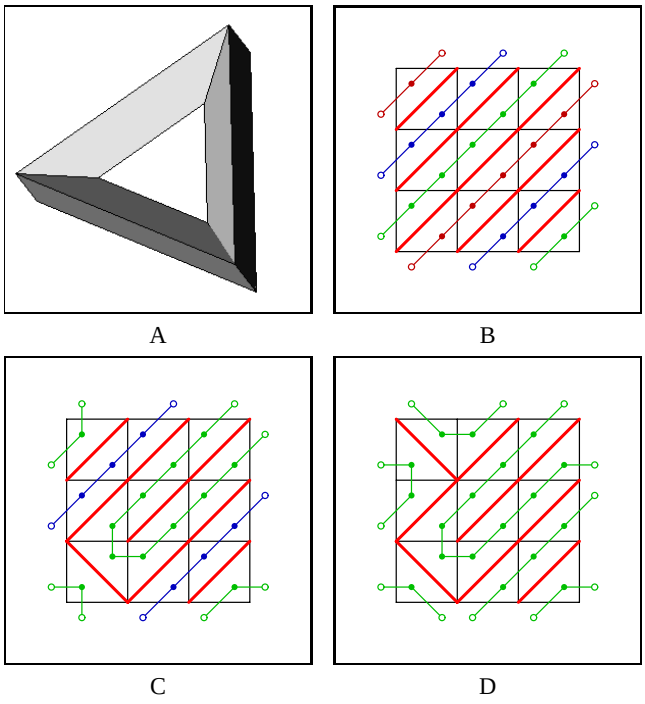


Figure 5: The algorithm of figure 4 applied to a 3×3 torus (A). An arbitrary choice of diagonals produces a triangulation with three cycles (B). After one diagonal flip the number of cycles is two (C). After an additional flip we obtain a triangulation with a single Hamiltonian cycle (D). Remember that opposite boundary edges and vertices are identified. In particular, the four corners of the square correspond to the same mesh vertex.

quadrilateral faces are split along one diagonal while the dual graph is tree-traversed in depth-first order. In this algorithm the two triangles resulting from splitting a face are also consecutive in the tree traversal order. The tree can be cut into generalized triangle strips, but unfortunately, the number of branching nodes in the spanning tree that links the faces in the dual graph, where the cuts have to be made, could be arbitrarily large. This problem is shared by a number of T-mesh connectivity encoding schemes [19, 22, 16]. As a result, we follow a different approach.

4 BASIC ALGORITHM

Instead of a Hamiltonian cycle, our approach is based on the existence of an Eulerian circuit in the dual graph of the Q-mesh, which is guaranteed for every connected graph in which all the vertices have even order [10]. After the quadrilateral faces are split, the resulting triangles are linked through the original mesh edges. The new edges associated with the diagonal splits become the boundary edges of the resulting Hamiltonian T-strip cycle.

Note however, that not every Eulerian circuit on the dual graph of a Q-mesh defines a Hamiltonian triangulation, because edges that are opposite to each other on a face cannot be contiguous in the Eulerian circuit. Figure 2 illustrates this problem. We need a special algorithm to construct the Eulerian circuit taking into account the constraints imposed by the cyclical ordering of the edges around each face: if two mesh edges are not adjacent in the graph of the mesh, they cannot be contiguous in the Eulerian circuit.

The problem is to choose one out of the two diagonals of each quadrilateral face so that the result of splitting the faces along the

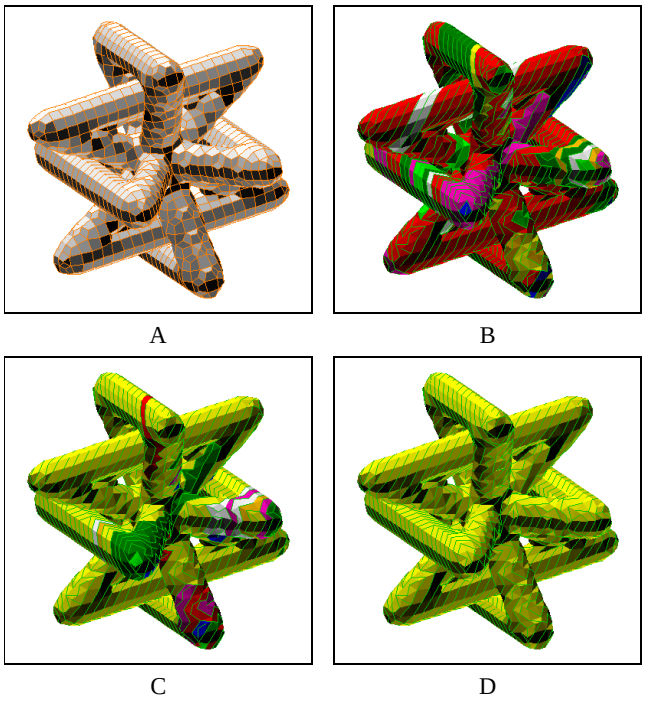


Figure 6: The algorithm of figure 4 applied to a more complex mesh (A). A random choice of diagonals produces a triangulation with 73 cycles (B). After 30 diagonal flips the resulting triangulation still has 43 cycles (C). After 42 additional flips we obtain a triangulation with a single Hamiltonian cycle (D).

chosen diagonals defines a single Hamiltonian T-strip cycle. If the diagonals are chosen at random, the graph defined by the triangles as graph vertices and the original mesh edges as graph edges is composed of a collection of disconnected cycles because each graph vertex is connected to exactly two other graph vertices. If the number of cycles in this graph is one, the problem is solved. If the number is larger than one, we only need to perform one additional step during which some diagonals are flipped. This step is based on the following observation illustrated in figure 3: if the two triangles generated by splitting one quadrilateral face belong to different cycles, splitting the same face along the other diagonal (flipping the diagonal) joins the two cycles into a single one. Since the number never increases, to construct a single cycle we just have to visit the faces of the mesh in an arbitrary order, and flip the diagonal of each face that joins different cycles. This simple algorithm is illustrated in figure 4. Figures 5 and 6 show examples of this algorithm applied to Q-meshes, with the state at an intermediate point where some diagonals have been flipped but not all. It is important to note that flipping a diagonal that joins two triangles that belong to the same cycle must be avoided, because doing so splits the cycle into two disconnected cycles.

5 EFFICIENT IMPLEMENTATION

Figure 7 illustrates an efficient and more detailed implementation of the algorithm of figure 4. This implementation uses two data structures. The first one is a boolean array B with one bit b_f per face that indicates along which of the two diagonals the face must be split. If $b_f = 0$ the face $f = (v_0, v_1, v_2, v_3)$ is split along the (v_0, v_2) diagonal, and along the (v_1, v_3) diagonal if $b_f = 1$. This array is filled by the calling function and modified here so that splitting

```
PartitionEdges (M = (V, E, F), B)
  p_E = new Partition (E)
  for each face f = (v_0, v_1, v_2, v_3) ∈ F
    if b_f = 0
      p_E.join(e_01, e_12)
      p_E.join(e_23, e_30)
    else if b_f = 1
      p_E.join(e_12, e_23)
      p_E.join(e_30, e_01)
return p_E

JoinCycles (M = (V, E, F), B, p_E, S)
  for each face f = (v_0, v_1, v_2, v_3) ∈ S
    if b_f = 0
      if p_E.find(e_01) ≠ p_E.find(e_23)
        b_f = 1
        p_E.join(e_01, e_23)
    else if b_f = 1
      if p_E.find(e_01) ≠ p_E.find(e_12)
        b_f = 0
        p_E.join(e_01, e_12)
return

EfficientHamiltonianSplit (M = (V, E, F), B, S)
  # initialize edge partition
  p_E = PartitionEdges (M, B)
  # join cycles by flipping diagonals
  JoinCycles (M, B, p_E, S)
return
```

Figure 7: Efficient implementation of the algorithm illustrated in figure 4. B is a boolean array with one face bit b_f per face initialized by the calling function. The function `PartitionEdges` initializes the partition of the set of edges into the cycles defined by the array B. The function `JoinCycles` joins the multiple cycles into a single one by flipping some face bits corresponding to faces in the subset S of F. $S = F$ ensures success, but other choices will be discussed later. The mesh edge $e_{ij} = M.\text{getEdge}(v_i, v_j)$ connects vertices v_i and v_j.

the faces along the corresponding diagonals generates a Hamiltonian triangulation. The second data structure is a set partition class based on Tarjan's fast Union-Find algorithm [18]. This set partition data structure efficiently implements the operations of membership (*find*) and union (*join*) of disjoint sets. It is used here to maintain a partition of the set of mesh edges into the cycles defined by the bits in the boolean array. It is neither necessary to maintain the cycles as linked lists, nor to explicitly split the faces into triangles. The Q-mesh is actually not modified by this algorithm, which returns just the boolean array. A subsequent traversal of the dual graph of the Q-mesh along the Eulerian circuit defined by the bits, can be used to create an explicit generalized T-strip representation. An additional hash table is used to efficiently implement the function $e_{ij} = M.\text{getEdge}(v_i, v_j)$ that locates an edge from its ends. But we consider this as part of the data structure used to represent the mesh connectivity.

Note that only the diagonals of faces belonging to the subset S of F, passed as an argument by the calling function as well, are considering for flipping. This subset must be chosen carefully to guarantee successful termination with a single cycle. A safe choice is $S = F$. Strategies to choose S containing a small number of faces will be discussed in the next section.

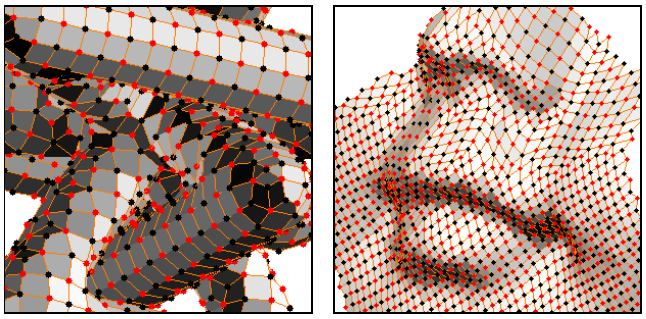


Figure 8: Examples of 2-colorable Q-meshes.

6 DIAGONAL GRAPH STRUCTURE

The number of cycles produced by the algorithm described above, and as a consequence the number of diagonal flips necessary to link all the cycles into a single one, is in principle arbitrarily large. In this section we analyze the structure of the graph defined by the chosen diagonals in detail. This analysis will be used in the next section to develop strategies to tailor the construction of the array B and set S for applications to graphics cache optimization. In particular, we determine the minimum set of diagonals that, when considered for flipping during the second step of the algorithm, guarantee successful termination with a single cycle. We even show that in the most common cases there is actually no need to flip any diagonal.

The *diagonal graph* of a Q-mesh M is defined by the mesh vertices as graph vertices, and the diagonals of the quadrilateral faces as graph edges. If the Q-mesh has V vertices and F faces, then its diagonal graph has V vertices and $2F$ edges. The edges selected to split the faces by the algorithm described in the previous section define a maximal spanning subgraph D of the diagonal graph with V vertices and F edges.

6.1 Connected Components

We show here that D has at most two connected components. Later on we will see that we need different strategies to choose B and S for the two cases of one or two components. If we cut the Q-mesh through the chosen diagonals we obtain a single generalized triangle strip cycle. This T-mesh is topologically equivalent to a cylinder. Its boundary is a graph composed of two cycles. Let us call these cycles *left boundary* and *right boundary*. The Q-mesh vertices corresponding to left boundary T-mesh vertices and the diagonals that join pairs of these vertices form a subgraph of D that we denote D_L. This graph is clearly connected, because it can be obtained by clustering (identifying) vertices of a cycle, which is connected. A similar construction for the right boundary yields the connected subgraph D_R of D. Since D is the union of D_L and D_R, it follows that D has *at most* two connected components. But these two graphs may not be disjoint, in which case D has one connected component (is connected).

6.2 2-Coloring

A *2-coloring* of a graph is an assignment of one of two different colors (we will use red and black) to each vertex so that no edge has both ends of the same color. A graph is *2-colorable* if such assignment exists. A mesh is *2-colorable* if its graph is 2-colorable. Some examples of 2-colorable Q-meshes are shown in figure 8. Note that a mesh with a face with odd number of corners is not 2-colorable. In fact, a fundamental result in graph coloring is that a graph is 2-colorable if and only if it has no cycles of odd length [10].

```
Is2ColorableGraph (G = (V, E))
  T = SpanningTree (G)
  choose r ∈ V as the root of T
  # depth-first traversal
  for i ∈ V
    if depth_T(i) is even
      c_i = red
    else
      c_i = blue
  for e = (i, j) ∈ E
    if c_i = c_j
      return false
return true
```

Figure 9: Algorithm to determine if a connected graph is 2-colorable or not.

We show now that D has two components if and only if the Q-mesh is 2-colorable. It is a lot easier to determine 2-colorability than to count connected components. For the necessity, since D has two connected components, paint red all the vertices of D_L and black all the vertices of D_R. In this case the mesh is 2-colorable because every mesh edge, being a marching edges of the T-strip, joins a vertex of D_L (red) and a vertex of D_R (black). For the sufficiency, let's assume that the Q-mesh is 2-colorable and D is connected. Pick any edge of the mesh. Since D is connected, there is a path in D with the ends of the chosen edge as beginning and end. It follows that there is a path in the graph of the Q-mesh of even length with the same beginning and end (details left to the reader). If we add the original edge to this path we construct a cycle of odd length, which contradicts the 2-colorability.

Note that 2-colorability is independent of topological type. For example, verify that D has one connected component for the 3×3 torus of figure 5 and that the mesh is not 2-colorable (has a cycle of length 3). Do the same experiment with a 4×4 torus to verify that D has two connected components and that the mesh is 2-colorable.

It is important to point out that 2-colorable meshes are in widespread use in computer graphics, visualization, modelling and animation: it is not difficult to verify that Catmull-Clark meshes (even the non-manifold ones) and isosurfaces based on deformed cuberille Q-meshes (boundary mesh of set of voxels, regularized or not) are 2-colorable. It is also true that all planar Q-meshes (without handles) are 2-colorable (a proof is sketched in section 6.3). And a simple tree-traversal algorithm illustrated in figure 9, during which newly visited vertices are painted with alternating colors, can be used to determine if a connected graph is 2-colorable or not [10].

6.3 Spanning Forests

The *Euler characteristic* of a manifold mesh without border with V (non-singular) vertices, E edges, and F faces s the number $V - E + F$, which is a topological invariant. When a mesh is orientable, the Euler characteristic is also equal to $2 - 2G$, where G is the *genus*, or number of handles, of the mesh. A Q-mesh with V vertices, E edges, and F faces, has Euler characteristic $V - F$ because $E = 2F$ (each quadrilateral face is covered by four half-edges, and each edge is shared by two faces). A mesh of genus zero (with no handles) is *planar*.

Since D is composed of C (1 or 2) connected components, a spanning forest constructed in D with a maximal number of edges will contain C trees. Since the forest spans the diagonal graph, it contains $V - C$ edges, and so, the number of faces not split by a diagonal edge in the forest is $S = F-(V-C) = C+F-V$, which

```
DiagonalSpanningForest (M = (V, E, F))
  B = new BooleanArray (F)
  S = new Set ()
  # build spanning forest in diagonal graph
  p_V = new Partition (V)
  for f = (v0, v1, v2, v3) ∈ F
    if p_V.find(v0) = p_V.find(v2)
      b_f = 0
      p_V.join(v0, v2)
    else if p_V.find(v1) = p_V.find(v3)
      b_f = 1
      p_V.join(v1, v3)
    else
      # collect cycle-producing faces in stack
      S.include (f)
      # assign temporary random value
return (B, S)
```

Figure 10: Algorithm to construct a spanning forest in the diagonal graph of a Q-mesh M with maximal number of edges, but such that at most one diagonal of each face is included in the forest. The faces without a diagonal in the forest are collected in a subset $S \subset F$ for subsequent processing. Inserting a diagonal of one of these faces in the forest creates a cycle.

is typically a small number compared with the total number faces. In the orientable case this number is equal to $2G + (C - 2)$, i.e., $2G$ if the Q-mesh is 2-colorable, and $2G-1$ if not. For example, for any planar Q-mesh ($G = 0$) the graph D is a forest composed of exactly two trees, and there is no need to flip diagonals. In particular, this proves that any planar Q-mesh is 2-colorable, and equivalently, any non-2-colorable Q-mesh is non-planar.

In the non-planar 2-colorable case we can avoid flipping diagonals as well. We first construct a spanning forest with a maximal number of edges in the diagonal graph, and collect the faces whose diagonals create cycles if inserted in the forest. Figure 10 illustrates an algorithm to do so, where the non-split faces are collected in the set S. Then we paint red all the vertices of one of the trees, and black all the vertices of the other tree. Finally, we construct D by inserting all the diagonals of the faces in S that join black vertices in the forest. At this point there is no need to run the algorithm of figure 7 because the set of diagonals already define a Hamiltonian cycle. The result of splitting the faces through this set of diagonals is a single cycle because cutting through the red tree produces a boundary cycle with as many vertices and edges as edges in the tree, and every triangle has either one or two vertices on this boundary. This cycle is one of the boundaries of the resulting T-strip. This is closely related to the Topological Surgery scheme for P-mesh connectivity compression [19], particularly as implemented in the MPEG-4 standard [14]. Figure 11 illustrates the relation between a tree of edges and the boundary cycle it produces for a general P-mesh.

In the non-2-colorable case D is connected and the spanning forest is composed of a single tree. Again, cutting the Q-mesh through the diagonals belonging to the spanning tree produces a new mesh with a single boundary cycle. This mesh is composed of the remaining quadrilateral faces collected in the set S, and a number of generalized T-strips, each starting and ending at an edge shared with a quadrilateral. There are $2S$ T-strips counting T-strips of length zero corresponding to edges shared by two quadrilateral faces. Each of these T-strips may start and end in different quadrilaterals, or in the same quadrilateral. In the later case, the two edges that connect the T-strip to the quadrilateral may be opposite or adjacent. If opposite,

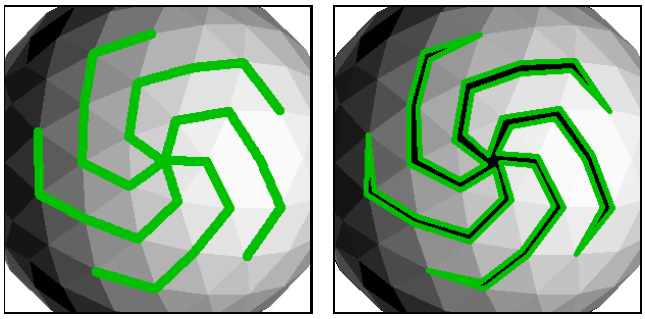


Figure 11: Cutting through a tree creates a boundary cycle.

neither one of the two diagonals splits the mesh into disconnected parts. If adjacent, the diagonal that leaves the two edges on opposites sides is the only choice that keeps the mesh connected. The simplest strategy here is to revert to running the algorithm of figure 7 as a post-processing step. That is, we choose the diagonals of the faces collected in S at random, partition the edges of the mesh into connected components corresponding to the generated cycles, and sequentially, flip the diagonals that join different cycles. But here we only need to consider flipping the diagonals of the faces that belong to the set S.

7 TRANSPARENT VERTEX CACHING

Support for T-strips requires a cache of size 2 in the graphics processing unit (GPU). Since processing a cached vertex is much faster than an uncached one, more modern GPU's maintain a larger vertex FIFO cache. To make good use of this cache these *indexed T-strips* must be constructed maximizing vertex locality. In this section we discuss strategies to optimize the use of this cache in our construction.

The triangle ordering is called *rendering sequence* by Bogomjakov and Gotsman [3], who describe methods to construct *universal rendering sequences* for T-meshes that preserve locality at all scales. Hoppe [12] introduced the *transparent vertex caching* problem, and presented algorithms to optimize the decomposition of T-meshes into T-strips for a particular cache size. Several years earlier, Deering [8] presented a hardware-oriented geometry compression scheme based on an actively managed (non-FIFO) cache of size 16. Chow [7] presented methods to decompose T-meshes into Deering's *generalized triangle meshes*, and so did Bar-Yehuda and Gotsman [2], who also showed that a cache of size $O(\sqrt{n})$ is necessary to minimize cache misses to zero.

A good rendering sequence minimizes the *average cache miss ratio* (acmr). This number, which measures the average number of cache misses per triangle, has a minimum value of about 0.5 (one cache miss per vertex) and a maximum of 3.0, which corresponds to the case when each T-strip is composed of a single triangle. In our case, since we can link all the triangles into a single T-strip, the actual maximum is slightly above 1.0.

7.1 Greedy Approach

As in the method proposed by Bogomjakov and Gotsman [3] for T-meshes, our scheme produces a rendering sequence independent of the cache size. Rather than an optimization-based procedure, we experimented with simple greedy schemes based on different face traversal algorithms and strategies to choose diagonals as the faces are visited. We implemented the traversal of faces in spiraling fashion, as used in most P-mesh connectivity encoding algorithms, and in random order, i.e., in the order the faces appear in the input

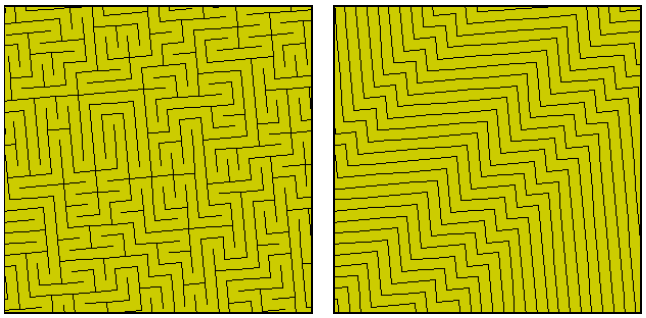


Figure 12: Maximizing vertex locality requires trees with large numbers of leafs and branching nodes.

file. When a face is visited during the traversal, one of its two diagonals is chosen. To maximize locality, we follow strategies that build spanning forests with lots of leaf nodes. This is illustrated in figure 12. A large number of leaf nodes implies a large number of branching nodes. The three most successful strategies were: 1) choosing the diagonal that locally maximizes the vertex valence of the spanning forest constructed so far, 2) always choosing the diagonal across the traversal (in the spiraling traversal), and 3) randomly choosing one of the two diagonals as a function of a random face bit produced by a pseudo-random number generator. In the later case, each run produces a different result, but the results of different runs are very consistent. On the other hand, we have verified experimentally that a strategy that minimizes the number of leafs and branching nodes in the spanning forest produces much worse rendering sequence, often with $acmr \approx 1$.

7.2 Results

Figure 13 shows some of the models used in our experiments. The table in figure 14 shows the sizes and genus of these meshes, as well as the one shown in figure 1, and the *acmr* we obtain for different cache sizes. The cube was obtained by recursive Catmull-Clark connectivity subdividision from a regular cube with six faces, and subsequent low-pass filtering. The bunny is not the original Stanford bunny, but a resampled version with quadrilateral faces and filled boundaries. The shape and spine meshes are isosurfaces. The skull and the head originally came from 3D scanned data and both had lots of boundaries. To remove the boundaries we generated the boundary surface of the solid resulting from extruding the mesh along the normal direction by a fixed amount such as average edge length. The resulting mesh is composed of two parallel copies of the original mesh connected along the corresponding boundaries by cycles of quadrilateral faces.

Roughly speaking, we obtain average *acmr* of about 0.70 for cache size 16, 0.64 for cache size 32, and 0.60 for cache size 64, with very small deviation from these values, and very much independently of mesh size and regularity of the mesh. This performance values are comparable to those reported by Bogomjakov and Gotsman [3], whose algorithm for T-meshes is based on a complex combinatorial optimization procedure. Our results are also comparable with those produced by the greedy algorithm for T-meshes presented by Hoppe [12], which is tuned for a specific cache size.

It would be interesting to establish a theoretical *acmr* lower bound, and to see how much room is there for improvements using an additional combinatorial optimization step. However, given the good and consistent performance of the very simple strategies presented here, and the low computational cost, we do not see much practical need for further optimization.

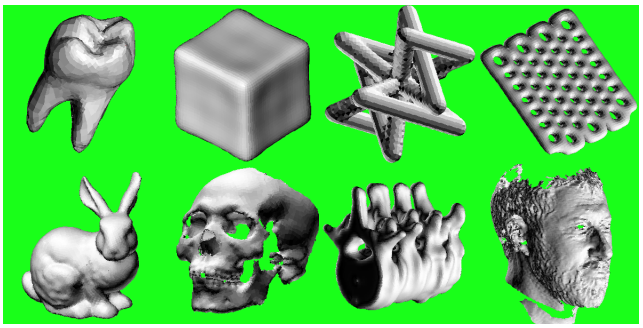


Figure 13: Some Q-meshes used in our experiments.

mesh				acmr		
name	connectivity			cache size		
	V	F	G	16	32	64
toothQ	2,633	2,631	0	0.698	0.638	0.592
angelMouthQ	4,030	4,028	0	0.671	0.623	0.587
cube6146	6,146	6,144	0	0.685	0.619	0.583
shape51Q	6,912	6,938	14	0.714	0.641	0.602
g49plateQ	10,016	10,112	14	0.708	0.645	0.608
bunnyQ	20,758	20,756	0	0.673	0.614	0.579
skullQ	26,436	26,456	11	0.706	0.642	0.601
spine4Q	46,254	46,286	17	0.714	0.651	0.607
headscanQ	191,890	191,934	23	0.722	0.660	0.616
average				0.699	0.637	0.598

Figure 14: Results corresponding to the meshes shown in figure 13.

8 IMPLEMENTATION AND COMPLEXITY

We implemented all the algorithms described in previous sections in Java and integrated them into our interactive mesh processing tool. The different steps of the algorithm can be run independently of each other, or all together by pressing a single button. The user can interactively set all the parameters and options. A screen shot of this application is illustrated in figure 15. Because of this tight integration, and the additional operations performed for visualization purposes, it is difficult to measure running times with precision. Note that Hoppe [12] reports running times of up to 4 hours on meshes of about $100,000$ vertices for his optimization-based algorithm. Our greedy algorithms run at interactive rates for meshes of this size and larger. In fact, running times for our algorithms are comparable with rendering times.

In terms of complexity, Tarjan's union-find algorithm [18] (used to maintain set partitions), and hash table access (used to represent mesh edges), correspond to steps with super-linear complexity. However, its is well known that the union-find algorithm requires linear storage and has linear complexity for practical purposes, and hash table access has expected linear complexity. The rest of the steps have linear space and time complexity.

9 SUBDIVISION

Catmull-Clark Subdivision [5] is the method of choice to refine Q-meshes. Here each quadrilateral face is subdivided into four smaller quadrilateral faces. Suppose that we have chosen diagonals in the coarse mesh so that the resulting linked triangles form a Hamiltonian T-strip. The diagonal chosen to split each coarse face splits two of the corresponding fine faces as well. If the two parallel diagonals are chosen for the other two fine faces, when we link the resulting fine triangles trough the mesh edges we obtain two paral-

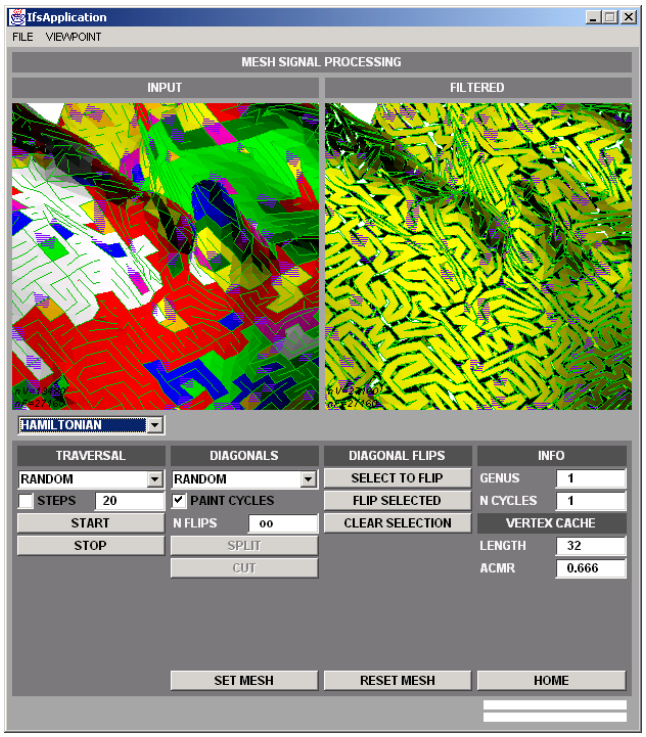


Figure 15: Java implementation of algorithms as part of interactive mesh processing tool.

lel cycles, and only one diagonal flip is sufficient to link them. For example, any of the fine faces not split by the coarse face diagonal is an acceptable choice here. Note that, although very simple, this procedure reduces the locality of the rendering sequence quite significantly. Consider two triangles that share a diagonal, and belong to the two original cycles. The distance along the joined Hamiltonian T-strip from one to the other may be up to one half the number of triangles in the strip. A better strategy is to first choose the diagonals of the two fine faces not split by the original coarse diagonal orthogonal to the other diagonals, and then run the algorithm of figure 7 to join all these cycles into a single one. Figure 16 illustrates the concepts discussed in this section.

10 CONCLUSIONS AND FUTURE WORK

In this paper we presented very simple algorithms to represent any connected manifold quadrilateral mesh without boundary as a single Hamiltonian generalized triangle strip cycle in many different ways, by splitting each face along one of its two diagonals. We analyzed the structure of the graph of diagonals so produced, and developed practical strategies to choose a rendering sequence that makes good use of a graphics processing unit's FIFO vertex cache. There is potential for further optimization for cache utilization, and it will be interesting to investigate how much more can be gained by such a procedure. The algorithms presented in this paper do not extend in a straightforward manner to meshes with boundary and non-manifold meshes. It would be interesting to find ways to extend these ideas to this larger family of meshes. Converting T-meshes to Q-meshes is important for numerical simulations, but difficult to do. Very often the result of these procedures is a TQ-mesh, i.e., a mesh composed of a majority of quadrilateral faces, and a few triangular faces. We plan to extend our ideas to these meshes as well.

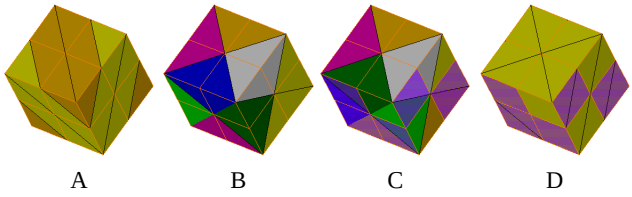


Figure 16: Construction of Hamiltonian T-strip cycles on subdivision meshes. (A) Choosing all the diagonals of the coarse quadrilateral faces parallel to the diagonal chosen for the corresponding coarse face produces two parallel cycles with little vertex locality. (B) Flipping all the marching edges of these two cycles produces a large number of small cycles. (C) Some of these diagonals must be flipped back to link all these cycles into a single one. (D) The resulting Hamiltonian T-strip.

References

[1] E.M. Arkin, M. Held, J.S.B. Mitchell, and S.S. Skiena. Hamiltonian triangulations for fast rendering. In J. Van Leeuwen, editor, *Algorithms-ESA'94*, volume 855 of *LNCS*, pages 36–47, Utrecht, NL, September 1994.

[2] R. Bar-Yehuda and C. Gotsman. Time/space tradeoffs for polygon mesh rendering. *ACM Transactions on Graphics*, 15(2):141–152, April 1996.

[3] A. Bogomjakov and C. Gotsman. Universal rendering sequences for transparent vertex caching of progressive meshes. In *Proceedings, Graphics Interface, GI'2001*, pages 81–90, June 2001.

[4] P. Bose and G.T. Toussaint. No quadrangulation is extremely odd. In *Proceedings, International Symposium on Algorithms and Computation*, Cairns, Australia, 1995.

[5] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, 10:350–355, 1978.

[6] N. Chiba and T. Nishizeki. The hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. *Journal of Algorithms*, 10(2):187–211, 1989.

[7] M.M. Chow. Optimized geometry compression for real-time rendering. In *IEEE Visualization'97 Conference Proceedings*, pages 347–354, 1997.

[8] M. Deering. Geometric compression. In *Siggraph'95 Conference Proceedings*, pages 13–20, August 1995.

[9] F. Evans, S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *Proceedings, IEEE Visualization'96*, pages 319–326, 1996.

[10] J. L. Gross and T. W. Tucker. *Topological Graph Theory*. Dover Publications, Inc., 2001.

[11] S.L. Hakimi, E.F. Schmeichel, and C. Thomassen. On the number of hamiltonian cycles in a maximal planar graph. *Journal of Graph Theory*, pages 365–370, 1979.

[12] H. Hoppe. Piecewise smooth subdivision surfaces with normal control. In *Siggraph'1999 Conference Proceedings*, pages 269–276, 1999.

[13] A. King, D. Szymczak and J. Rossignac. Connectivity compression for irregular quadrilateral meshes. Technical Report GIT-GVU-99-36, Georgia Tech GVU, 1999.

[14] ISO/IEC 14496-1 Information technology - Coding of audio-visual objects, Part 2: Visual / PDAM1 (MPEG-4 v.2), mar 1999.

[15] S. Ramaswami, P. Ramos, and G. Toussaint. Converting triangulations to quadrangulations. In *Proceedings, Seventh Canadian Conference on Computational Geometry, CCCG'95*, 1995.

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

[17] D.P. Sanders. On paths in planar graphs. *Journal of Graph Theory*, pages 341–345, 1997.

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

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

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

[21] R. Thomas and X. Yu. 4-connected projective-planar graphs are hamiltonian. *Journal of Combin. Theory Ser. B*, pages 114–132, 1994.

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

[23] W.T. Tutte. A theorem on planar graphs. *Trans. Amer. Math. Soc.*, pages 99–116, 1956.

[24] L. Velho, L.H. de Figueiredo, and J. Gomes. Hierarchical generalized triangle strips. *The Visual Computer*, 15(1):21–35, 1999.

[25] H. Whitney. A theorem on graphs. *Ann. Math.*, pages 378–390, 1931.

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