The Vertex-Adjacency Dual of a Triangulated Irregular Network has a Hamiltonian Cycle

John J. Bartholdi, III * Paul Goldsman

November 21, 2003

Abstract

Triangulated irregular networks (TINs) are common representations of surfaces in computational graphics. We define the dual of a TIN in a special way, based on vertex-adjacency, and show that its Hamiltonian cycle always exists and can be found efficiently. This result has applications in transmission of large graphics datasets.

Keywords: *Hamiltonian cycle, triangulated irregular network, triangle mesh, triangulation*

*Corresponding author. Address: School of Industrial and Systems Engineering, Georgia Institute of Technology, 765 Ferst Drive, Atlanta, Georgia 30332-0205 USA. E-mail: `john.bartholdi@isye.gatech.edu`. We appreciate the support of the Office of Naval Research through Grant #N00014-89-J-1571, the National Science Foundation through Grant #DMI-9908313, The Logistics Institute at Georgia Tech and its sister organization, The Logistics Institute Asia Pacific in Singapore. We are also grateful for the useful comments of an anonymous referee.

1 Introduction

Following Peter Alfeld [1]: A *triangulated irregular network* (*TIN*, or *triangle mesh*, or *triangulation*) T is a collection of n triangles satisfying the following:

1. The interiors of the triangles are pairwise disjoint;
2. Each edge of a triangle in T is either a common edge of two triangles in T or else it is on the boundary of the union of all the triangles;
3. The union of all the triangles is homeomorphic to a square.

Alfeld observes that

> The first requirement says that triangles don't overlap. The second requirement rules out combinations of triangles where one has a vertex in the interior of an edge of another triangle, and the strange sounding last requirement rules out holes, pinchpoints (where just 2 triangles meet in a single point) and disjoint sets of triangles.

TINs are commonly used to represent surfaces such as terrains and 3-D computer graphics models (see for example [6]).

A dual graph of a TIN can be constructed by defining a vertex (node) for each triangle, and an edge for each pair of triangles that share an edge. A TIN is said to be *Hamiltonian* if this dual has a Hamiltonian cycle (or path). Thus a Hamiltonian cycle orders the triangular cells of a TIN in a way that preserves edge-adjacency. The idea of finding sequences of edge-adjacent triangles has been used to compress large polyhedral surface models [3, 5, 7, 8, 9]. The problem arises because each vertex in a TIN may belong to multiple triangles and so a naive method of transmitting or processing a TIN, such as by sending the triangles individually, would contain much duplicated data. If the TIN can be disassembled into a Hamiltonian path of triangles — that is, if the dual graph is Hamiltonian — then the TIN can be transmitted very economically, by a strategy of sending the single remaining vertex of the next triangle along the path. In this way, each point would be processed once.

Unfortunately, it is NP-Complete to determine whether or not a TIN is Hamiltonian [2]. Consequently, this method of compression is generally implemented by trying to find long sequences of edge-adjacent triangles (segments of paths in the dual graph). Evans, Skiena, and Varshney (1996) discuss some typical greedy heuristics to find long paths in the dual in hopes of achieving greater compression [4]. However, increasing computation risks offsetting any gains in compression.

We show that a weaker version of the dual guarantees the existence of a Hamiltonian circuit. We suggest considering the *vertex-adjacency dual* of a TIN, which is constructed by defining a vertex for each triangle, and an edge for each pair of triangles that share a vertex or an edge. A triangle has more vertex-adjacent neighbors than it does edge-adjacent neighbors; we show that this extra connectivity is sufficient to guarantee the existence of a Hamiltonian

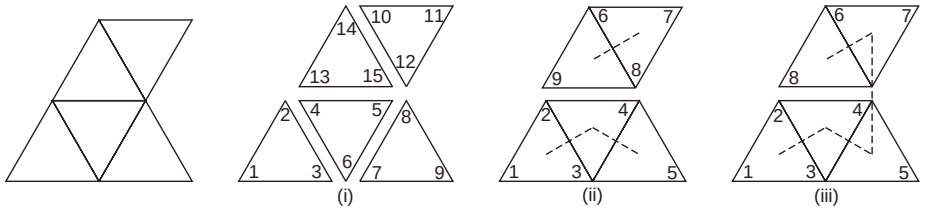

Figure 1: Three approaches to transmitting TIN data. (i): as individual triangles, (ii): by edge-adjacency, (iii): by vertex-adjacency.

circuit. Moreover, such a circuit can be constructed in linear time, so that it seems to offer a practical basis for data compression.

Figure 1 illustrates the three approaches to transmitting a TIN that we have mentioned: (i) *naive* approach, in which each triangle is sent individually, resulting in the transmission of 15 vertices; (ii) *triangle strip* approach, in which the TIN is divided into two edge-adjacent triangle strips, and 9 vertices are transmitted; (iii) *vertex-adjacency* approach, in which 8 vertices are transmitted.

2 Constructing a Hamiltonian cycle

We show how to construct a Hamiltonian cycle through the vertex-adjacency dual of a TIN T. The cycle of the nodes in the dual corresponds to a cyclic ordering of the triangles in the TIN. For convenience, where there can be no confusion we will speak interchangeably of cycles of either *nodes* (in the dual) or *triangles* (in the TIN).

Each triangle t has two neighbors in a cycle: a predecessor $pred(t)$ and a successor $succ(t)$. Neighboring triangles of T sharing a common edge (two vertices) will be called *2-adj*; neighboring triangles sharing exactly one common vertex will be called **1-adj**.

The cycle that we create has the following special property that is useful in avoiding an awkward case in the construction of the cycle.

Property 1 *No triangle is* 1-adj*, at the same vertex, to both of its neighbors in the cycle.*

Theorem 1 *The vertex-adjacency dual of a TIN is Hamiltonian.*

Proof (constructive)

1. Initialization.

   Assume TIN T has n triangles. The theorem is trivially true for $n = 1$ or $n = 2$, so assume $n \geq 3$. Choose any triangle a, a second triangle b that is *2-adj* to a, and a third triangle c that is *2-adj* to either a or b. Assume without loss of generality that c is *2-adj* to b. Let C signify the

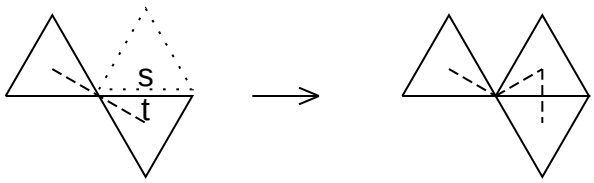

Figure 2: Case I: t is 1-adj (at a vertex on e) to at least one neighbor.

subset of T composed of triangles a, b, and c. C is itself a TIN, and since triangle a is clearly *1-adj* or *2-adj* to triangle c, a Hamiltonian cycle of the vertex-adjacency dual of C is simply: a, b, c, a. Property 1 is satisfied, since each triangle is *2-adj* to at least one other triangle.

The procedure continues by adding triangles to C, one at a time, and updating the cycle of its dual at each step, until C includes all of T. After each update, we must verify that the new cycle is still Hamiltonian (by checking adjacencies around the change). We also verify that Property 1 remains satisfied by showing, for each triangle involved in the update, that either: its neighbors are unchanged; it has a 2-adj neighbor; it is 1-adj to both neighbors at different vertices; or, it has exchanged one *1-adj* neighbor for a different *1-adj* neighbor at the *same* vertex. The straightforward details of all these checks are deferred to the Appendix.

2. Iteration.

   If C includes all the triangles in T then the procedure is done. Otherwise, find a triangle s, not in C, that shares edge e with some triangle t in C. Add s to C, updating the cycle according to the following cases, which are partitioned according to t's relationship with its neighbors:

   - *Case I: t is* 1-adj*, on e, to (at least) one of its neighbors (Figure 2).* Assume, without loss of generality, that t is *1-adj*, on e, to $pred(t)$.

     Solution: Add s to the cycle as follows: $pred(t), s, t, succ(t)$. (Note that $pred(t)$ now actually becomes $pred(s)$, but for clarity we continue to refer to vertices by their initial labels, here and elsewhere.)

     The curve in Figure 2 is shown to originate at the center of $pred(t)$, and terminate at the center of t. This is meant to indicate that we do not restrict how $pred(t)$ is adjacent to $pred(pred(t))$, or how t is adjacent to $succ(t)$ — because all (legal) possibilities are covered by the solution shown. For example, t and $succ(t)$ could be *2-adj* at either of t's open edges, or they could be *1-adj* at any of t's vertices (aside from the vertex at which t is *1-adj* to $pred(t)$).

   - *Case II: t is* 2-adj *to both neighbors (Figure 3).*

     Note that in the figure some curves begin (end) at a vertex, indicating that the first triangle is adjacent to its predecessor (successor) at the vertex indicated. When a curve begins (ends) at a triangle center,

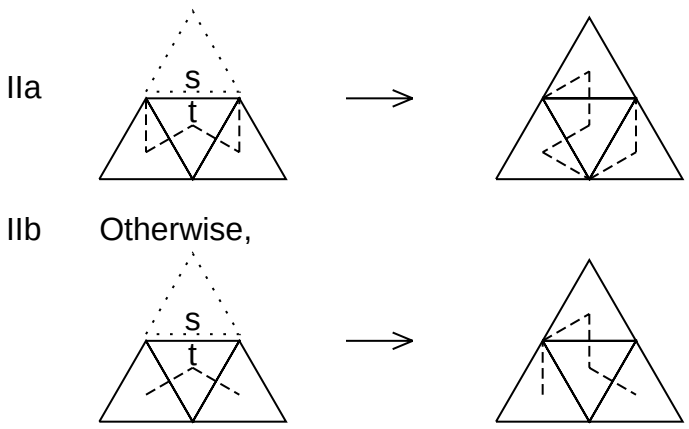

Figure 3: Case II: t is 2-adj to both neighbors. Subcase IIa: pred(pred(t)) is 1-adj to pred(t) on e, and succ(succ(t)) is 1-adj to succ(t) on e. Subcase IIb: Otherwise.

the preceding triangle may be adjacent by any of its edges or vertices, as long as the adjacency is *legal*.

- IIa: $pred(pred(t))$ is *1-adj* to $pred(t)$ on e, and $succ(succ(t))$ is *1-adj* to $succ(t)$ on e.
  Solution: $pred(pred(t)), s, t, pred(t), succ(t), succ(succ(t))$.
- IIb: Otherwise ($pred(pred(t))$ is not *1-adj* to $pred(t)$ on e, or $succ(succ(t))$ is not *1-adj* to $succ(t)$ on e, or both are not). Assume, without loss of generality, that $pred(pred(t))$ is not *1-adj* to $pred(t)$ on e.
  Solution: $pred(t), s, t, succ(t)$.

- Case III: t is *2-adj* to one neighbor, and *1-adj*, off e, to the other neighbor (Figure 4).

  Assume, without loss of generality, that t is *2-adj* to $pred(t)$, and *1-adj*, off e, to $succ(t)$.

  - IIIa: $pred(t)$ is *1-adj* to $pred(pred(t))$, on e.
    Solution: $pred(pred(t)), s, t, pred(t), succ(t)$.
  - IIIb: Otherwise ($pred(pred(t))$ is not *1-adj* to $pred(t)$ on e).
    Solution: $pred(t), s, t, succ(t)$.

The procedure consumes one triangle with each iteration, and so terminates after $n-3$ steps. To verify that the cases comprise the complete set of possibilities, observe that: Each triangle t in the current cycle (at each step) is either *1-adj* or *2-adj* to each of its two neighbors $pred(t)$ and $succ(t)$. Case I covers the case where t is *1-adj* to both neighbors (because at least one of the common vertices must be on e, by Property 1); Case II covers the case where t is *2-adj* to both neighbors; Case III covers the case where t is *2-adj* to one neighbor and

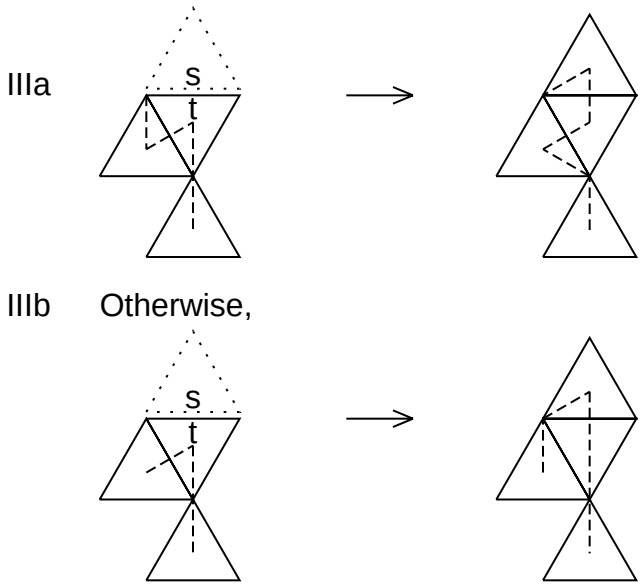

Figure 4: Case III: t is 2-adj to one neighbor, and 1-adj (at t's vertex opposite e) to the other neighbor. Subcase IIIa: pred(t) is 1-adj (on e) to pred(pred(t)). Subcase IIIb: Otherwise.

1-adj to the other (except where the *1-adjacency* is on e, which is covered in Case I). Cases II and III, are each partitioned by a pair of mutually-exclusive subcases. The remaining possibility, in which both of t's neighbors are *1-adj* at the same vertex, is forbidden by Property 1 and never occurs. ■

3 Conclusions

By accepting vertex-adjacency, rather than insisting on edge-adjacency, we have shown that, for any TIN, one can construct, in linear time, a single sequence of adjacent triangles that encompasses the entire TIN. This allows an alternative strategy for compression.

Recent efforts have focused on finding a small number of path segments that span all triangles of the TIN. Each path segment can then be sent efficiently — but there is extra work to find the path segments and extra work to shift from one path segment to the next. Our method quickly finds a single path, but it may not compress as well as the ideal one based on edge-adjacency (which may not exist and can be hard to find in any case).

Potential duplication arises in our method whenever two neighboring triangles (in the cycle) share just one vertex, so that we must send two new vertices rather than one to specify the second triangle. This never requires more than two points per triangle, which is worse than the (perhaps unachievable) lower

bound of one point per triangle, but better than sending each triangle separately (3 points per triangle). More to the point, we would not expect vertex-adjacent triangles to dominate; informal tests suggest that there will likely be runs of edge-adjacent triangles interspersed with occasional vertex-adjacent triangles.

Comparison of compression methods based on edge-adjacency and methods based on vertex-adjacency is an empirical question that will be explored in subsequent work.

References

[1] Peter Alfeld. Triangulations. `http://www.math.utah.edu/~alfeld/MDS/triangulation.html`, March 1999.

[2] Esther M. Arkin, Martin Held, Joseph S. B. Mitchell, and Steven S. Skiena. Hamiltonian triangulations for fast rendering. In *Visualization '95: Proceedings, Oct 29–Nov 3, 1995*, Atlanta, GA, 1995.

[3] John J. Bartholdi, III and Paul Goldsman. Multiresolution indexing of triangulated irregular networks. *IEEE Transactions on Visualization and Computer Graphics*. To appear.

[4] Francine Evans, Steven Skiena, and Amitabh Varshney. Optimizing triangle strips for fast rendering. In *Proceedings, IEEE Visualization 1996*, pages 319–326, 1996.

[5] Martin Isenburg. Triangle strip compression. In *Proceedings of Graphics Interface 2000*, pages 197–204, May 2000.

[6] Robert Laurini and Derek Thompson. *Fundamentals of Spatial Information Systems*. Academic Press, Ltd., San Diego, CA, 1992.

[7] Jarek Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1), 1999.

[8] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, Apr 1998.

[9] Costa Touma and Craig Gotsman. Triangle mesh compression. In *Proceedings of Graphics Interface, 1998*, pages 26–34, Vancouver, B. C., 1998.