Single Triangle Strip and Loop on Manifolds with Boundaries

Pablo Diaz-Gutierrez David Eppstein M. Gopi
Department of Computer Science
University of California, Irvine.

Abstract

The single triangle-strip loop generation algorithm on a triangulated two-manifold presented by Gopi and Eppstein [4] is based on the guaranteed existence of a perfect matching in its dual graph. However, such a perfect matching is not guaranteed in the dual graph of triangulated manifolds with boundaries. In this paper, we present algorithms that suitably modify the results of the dual graph matching to generate a single strip loop on manifolds with boundaries. Further, the algorithm presented in [4] can produce only strip loops, but not linear strips. We present an algorithm that does topological surgery to construct linear strips, with user-specified start and end triangles, on manifolds with or without boundaries. The main contributions of this paper include graph algorithms to handle unmatched triangles, reduction of the number of Steiner vertices introduced to create strip loops, and finally a novel method to generate single linear strips with arbitrary start and end positions.

1. Introduction

The advantages of constructing triangle strips are not restricted to a reduction in the number of vertex transformations during rendering, but they have been shown to be useful in efficient data structures for fast back-face culling, vertex caching[6, 1], mesh simplification, compression [5, 11, 12, 7, 8], and generation of space-filling curves[4]. Most stripification algorithms use a greedy technique to incrementally grow the strips [3, 11], possibly followed by local optimizations to increase the strip length[15, 13]. An exception is [14], which takes quadrilateral meshes as its input, and then appropriately splits them into triangles before constructing a hamiltonian triangle strip. On the other hand, the algorithm presented by [4] reduces the stripification problem to a graph matching problem in the dual graph of the triangle mesh. Thanks to the availability of public domain software for the graph matching problem, the implementation of the algorithm in [4] is straight-forward, eliminating as well most of the book-keeping required by greedy stripification methods. This algorithm has also been studied further to achieve global controllability in strips, in order to aid efficient back-face culling and transparent vertex caching [2].

The stripification algorithm based on graph matching relies on the existence of a perfect matching in the dual graph of a triangle mesh. The sequences of unmatched neighbors form strip loops. Dual graphs of triangulated manifolds with boundaries are not guaranteed to have a perfect matching. The existence of triangles without matched neighbors possibly leads to multiple linear triangle strips, as opposed to strip loops. Linear strips cannot be further processed to create a single triangle strip. Producing a single triangle strip yields advantages that go well beyond rendering. With a single strip, a continuous linear parametrization of the entire model can be induced. Such parametrization can be used for stratified sampling, compression of the topology and geometry of the model, total ordering of mesh elements, etc. A single strip can also be used for efficient data structure manipulation for view-dependent rendering [2].

In this paper, we make important and non-trivial contributions towards analyzing and extending the stripification algorithm based on graph matching to manifolds with boundaries, in order to create single triangle strip loops and linear strips that provide all the advantages mentioned above.

Main Contributions: The following are the main contributions of the paper.

- We study the effects of the graph matching algorithm on the dual graph of triangulated manifolds with boundaries, and its applicability to stripification.
- Based on the above study, we develop algorithms to produce single triangle strip loops on manifolds with boundaries.
- Variants of the above algorithm are presented in order to both improve the run-time efficiency, and to reduce the number of additional (Steiner) vertices introduced while merging different strip-loops.

XIX Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'06)
0-7695-2686-1/06 $20.00 © 2006 IEEE

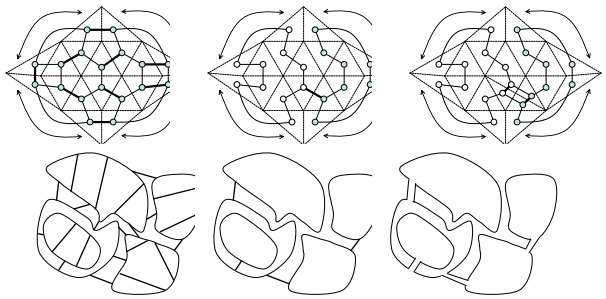

Figure 1. **Top (left to right): (a) The dual degree three graph of the triangulation of a genus 0 manifold and a perfect matching shown by dark edges. (b) The set of unmatched edges create disjoint cycles. Two such cycles are shown. These disjoint cycles are connected to each other by matched edges. The algorithm construct a spanning tree of these disjoint cycles and hence choose matched edges that connect these cycles. (c) The triangle pair corresponding to chosen matched edges in the tree are split creating two new triangles. Matching is toggled around the new (nodal) vertices resulting in a triangulation with a Hamiltonian cycle of unmatched edges. Bottom (left to right): (d-f) A generalized example of the same process shown just on the dual graph. [4]**

- Finally, we present a new algorithm based on local topological surgery, that uses the above stripification algorithm to generate a linear strip starting and ending at user-specified triangles.

2. Single Triangle Strip Loop Creation on Manifolds

The problem of finding a single triangle strip in a triangle mesh is equivalent to finding a Hamiltonian path in its dual graph, which is a well known computationally intractable problem. However, if we allow the addition of a few Steiner vertices, without changing the geometric fidelity or the topology of the mesh, we can find a single triangle strip in polynomial time. The algorithm presented by [4] is one such method that uses a perfect graph matching algorithm on the dual graph of a triangulated 2-manifold to create a single loop representation. Here, we briefly explain this algorithm for the sake of completion.

A matching in a graph pairs each vertex with at most one of its adjacent vertices. A perfect matching is a matching in which every vertex of the graph is matched. It is known from [10] that such a perfect matching exists for a 3-regular (each node is adjacent to 3 others), 3-connected (it is possible to find a path from any vertex to every other one after removing any 2 vertices) graph, such as the dual graph of a triangulation of a manifold without boundary. In the dual

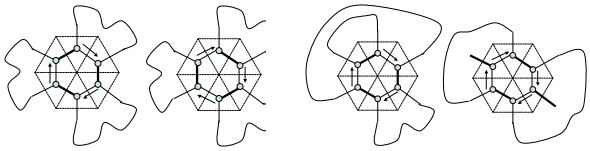

Figure 2. **(a) Left side pair of graphs: A nodal vertex with (six) even number of incident triangles and triangles belonging to three unique cycles. By switching the matched and unmatched edges, all these cycles can be merged to a single cycle. (b) Right side pair: Examples of non-nodal vertices. In both the examples, there are six incident triangles but only two unique cycles. [4]**

graph of a triangulation, a perfect matching is a pairing of every triangle with exactly one of its three edge-connected neighbor triangles. Triangle strip loops can be formed by connecting every triangle with its two unmatched neighbors. This yields not one, but many disjoint strip loops.

Next, all the disjoint loops can be iteratively joined into one by means of two simple operations, each of which takes two or more loops and merges them. The first operation splits two adjacent triangles that belong to different loops (refer to Figure 1) and merges the loops into one. The second operation is called *nodal vertex processing*. A *nodal vertex* with degree n is a vertex in the original mesh where n is even and the number of different loops incident on that vertex is $n/2$ (Figures 2(a), 2(b)). Around such a vertex, pairs of matched and unmatched triangles alternate around the nodal vertex. Swapping the matched and unmatched triangle relationship around a *nodal vertex* merges all the incident strip loops into one. Unlike triangle splitting, nodal-vertex processing merges loops without introducing additional faces to the mesh.

The perfect matching on the dual graph, on which the above algorithm relies, is guaranteed to exist only on 3-regular bridgeless graphs. But the dual graph of a triangulation of a manifold with boundary will have nodes (corresponding to the boundary triangles) with degree less than three, and hence a perfect matching is not guaranteed to exist. Therefore, applying the stripification algorithm [4] would produce both linear triangle strips and strip loops (Figure 3(a)). In the following sections, we study the matching algorithm on such graphs and extend the stripification algorithm to work on manifolds with boundaries.

3. Single Triangle Strips on Manifolds with Boundaries

As in the case of manifolds, the goal of our stripification algorithm for manifolds with boundaries is to first create a set

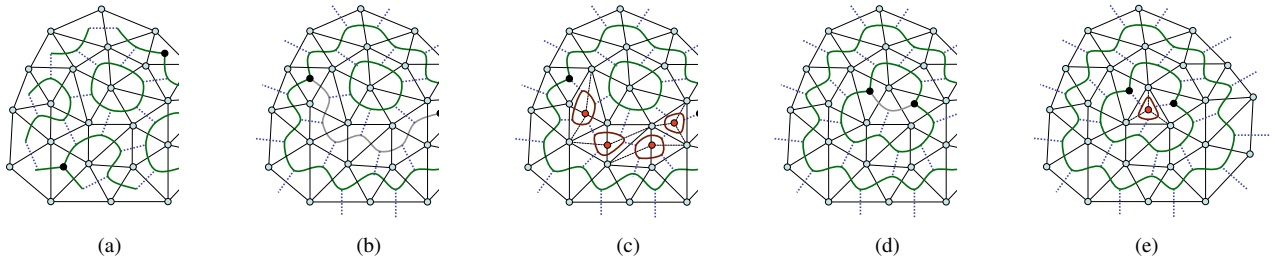

Figure 3. **(a) The stripification algorithm of [4] produces linear strips and strip loops in a mesh with boundary. (b) Forcing matching of the boundary triangles produces unmatched triangles (also called junction nodes, denoted by dark points) and paths of unmatched edges between them. The shortest path is shown using a lighter color. (c) The internal triangles along this shortest path are paired. Each pair of triangles is split into four triangles. Triangles without a pairing are split into three triangles. This yields perfect matching of the entire mesh. The number of new triangles introduced is dependent on the length of the shortest path. (d) An alternate matching yielding a shorter path between the junction nodes. Such shortest paths are produced by (optimal) forced-loop algorithm. (e) The only triangle in the shortest path, and hence without a pair, is split into three triangles as before.**

of disjoint triangle cycles, such that their union covers all the triangles of the model. This will be followed by the cycle merging operations, as explained in the previous section, to create one single triangle strip loop.

We conceptually close each hole in the manifold with boundaries by introducing one hypothetical vertex per boundary, and hypothetical triangles that connect the boundary edges with the hypothetical vertex associated to their hole. Given this new manifold without boundaries, we can use exactly the same algorithm as detailed in the previous section to create disjoint strip loops, since a perfect matching exists for its dual graph.

The strip loops produced this way might cross the original mesh boundary. This is undesirable because when the hypothetical triangles are later removed, such loops are cut and converted into linear (non cyclic) strips. In the rest of this section we describe two methods to prevent the strip loops from crossing the original boundary edges – forced-loop method and split-tie method.

Forced-loop method: In this method, we enforce a matching between each hypothetical triangle and the original triangles on the opposite side of the boundary. This way, hypothetical triangles around each boundary are forced to lie in the same strip. The undesired consequence of this approach is that if the forced matching is respected, a perfect matching need not exist in the rest of the graph. As we see later, the complexity of this method lies in achieving a perfect matching using triangle split operations.

Split-tie-loop method: In this method, we let the strips cross from original to hypothetical parts of the mesh. The complexity of this approach lies in applying minor 'surgery' to these cross-over points, splitting and tying the loops so that they remain either completely inside or completely outside the original mesh.

3.1. Forced-Loop Method

In order to prevent strips from crossing the boundary edges, the boundary edges have to be forced to be matched. Since at most one edge of each triangle can be matched, and hence become a boundary edge, an original triangle with two boundary edges is split along with its only neighbor triangle, so that the resulting triangles will have at most one boundary edge each. The graph matching algorithm is made to respect this pre-existing forced matching. If there exists a perfect matching in this mesh with forced matched edges, then the sequence of triangles along the unmatched edges of the original mesh form disjoint cycles and their union covers all the triangles of the mesh, thus solving our problem. But the fundamental problem is that if we respect the forced matching, the dual graph might not have a perfect matching. The following algorithm suitably modifies the original mesh in order to find a perfect matching. Further, it also minimizes the number of triangle-split operations necessary to achieve this perfect matching.

Let us first give an intuition behind our algorithm. Since the boundary triangles are matched with hypothetical triangles across the boundary edges, the boundaries can be considered to be closed by these hypothetical triangles, thus creating a triangulated manifold without boundaries. Further, since a triangulated manifold mesh has even number of triangles, there is an even number of unmatched triangles in the original mesh (Figures 3(b) and 3(d)). These unmatched triangles are connected by paths of unmatched edges. The goal is to match these unmatched triangles between themselves through these unmatched edge paths. To achieve a

perfect matching, every triangle along these paths has to be split (Figures 3(c) and 3(e)). The shorter these paths are, the fewer splits will be required. This process is shown for two different matchings in Figure 3, including the resulting triangle splits. One matching has longer unmatched paths than the other. The rest of the algorithm attempts to shorten the paths between pairs of unmatched triangles.

Consider the modified original mesh where every boundary triangle has exactly one boundary edge and construct hypothetical triangles across each of the boundary edges. Let G_1 be the dual graph of this modified mesh where all nodes have degree three and there is an extra degree-one node for each boundary edge.

Let G_2 be the spanning subgraph of G_1 with the fewest edges such that every node in G_2 has odd degree. Such a graph has interesting properties: if G_1 has a perfect matching, G_2 will consist of all nodes and only the matched edges so that every node has degree one. Even if G_1 has no perfect matching, every node in G_2 can be matched, not necessarily along a single edge, but along a sequence of edges such that no two such sequences share an edge. In the context of the example described earlier (Figure 3(d)), the nodes of G_2 consist of the dual of all triangles, all the matched edges, and the shortest sequence of unmatched edges between the unmatched triangles. In G_2, the nodes corresponding to the internal triangles of the path connecting two unmatched triangles will have degree three, and the rest of the triangles will have degree one. In order to find such a graph G_2 with minimum number of edges, we use the solution to the classic Chinese Postman Problem.

Complete node matching: Make a weighted complete graph which contains all nodes of G_1 with weight equal to the length of the shortest path between that pair of nodes in G_1. Compute a minimum weight perfect matching in this weighted complete graph. Such a perfect matching exists because it is a complete graph with even number of nodes. Note that most of these matched edges will have weight one. Let G_2 consist of the edges belonging to paths corresponding to the matched edges in the complete graph. Most of these paths have length one, and hence the corresponding nodes are directly matched to each other in G_2; others are matched along a sequence of edges with minimum number of internal nodes. (These paths that are longer than one edge are equivalent to the shortest path between two unmatched nodes in Figure 3(d).) The internal nodes have degree three. The next step is to split the triangles along these longer paths in such a way to find a perfect matching. Note that if the original graph has a perfect matching, G_2 will have no path longer than one and hence no internal nodes.

Path split operation: The goal of this step is to split the triangles along the longer paths to get a perfect matching of all triangles on the path. Find maximum matching among the internal nodes of G_2. Split each pair of matched internal triangles into four triangles using triangle-split operation, and each remaining unmatched internal triangle into three triangles. These triangle splits will induce the required perfect matching of all nodes in the longer paths, and hence the entire triangulation (Figures 3(c) and 3(e)).

The above algorithm, by virtue of being the result of a minimum weight perfect matching, minimizes the number of internal triangles, and hence the number of triangle splits to create a perfect matching. But on the other hand, since the weighted complete graph, on which minimum weight perfect matching algorithm is applied, has n^2 edges (where n is the number of triangles in the model), it is impractical to use this algorithm in normal graphics applications for which models with 1000 triangles (around one million edges in the extended graph) are considered tiny. Hence we are forced to design a sub-optimal but practical algorithm.

3.1.1. Modified Forced-Loop Method Here we present a modified forced loop method that is sub-optimal but practical. We consider the modified mesh with a hypothetical triangle across every boundary edge and every boundary triangle with only one boundary edge as input. We force the matching between the hypothetical triangle and the boundary triangle as before. In the dual graph of such a mesh, if a perfect matching exists the set of unmatched edges would form disjoint cycles. In the absence of a perfect matching, three paths of unmatched edges meet at unmatched nodes. These paths will start and end in unmatched nodes also called the *junction nodes* (Figure 3(b)). Hence that total number of such paths is at most $3j/2$ where j is the number of junction nodes. The first step is to match these junction nodes with each other along the unmatched edge-paths, such that the total path length of all the matchings is minimized. This can be solved using the same technique that we used to find the graph G_2, now with the weighted complete graph of only the junction nodes with the distance between each of them as the weights of the edges. Edges between junction nodes that are disconnected in the original graph are given an infinite weight. A minimum weight perfect matching on this complete graph will return paths that match these junction nodes with each other. Given these paths, the second step is to use the *path split operation* as described in the previous section to induce perfect matching.

The resulting strip loops lie completely inside the original mesh and do not cross the boundary edges. These loops are later merged into one single cycle using algorithms presented in [4].

The fundamental difference between the forced loop method and the modified forced loop method is that one

uses all the triangles in the entire mesh to form the weighted complete graph, while the modified algorithm uses only the junction nodes and the paths between them to form complete graph. Although this is the primary reason for the practicality of the method, this is also the reason for its sub-optimality: In the modified method, given the junction nodes and the paths between them, we choose pairs of junction nodes to minimize the total path length; in the optimal method, we try to find the junction nodes and paths between them so that the sum of the path lengths between each other is minimized. For example, while the examples shown in Figures 3(b) and 3(c) may be result of the modified (sub-optimal) forced loop algorithm, those shown in Figures 3(d) and 3(e) are result the of optimal forced loop algorithm.

3.2. Split-Tie-Loop Method

The (optimal) forced loop method is too expensive to compute. The (sub-optimal) modified forced loop method is unpredictable. The upper bound on the path length between the given junction nodes is the size of the triangulation, which is not very useful. For instance, this modified sub-optimal method on the bunny model produced 36.38% new faces due to triangle splits while in all other models the new faces were less than 2% of the original model size. Hence we present a method, Split-Tie-Loop method, that may not be optimal, but produces consistently good results on all the models, and its bound on the number of triangle splits is proportional to the number of boundary edges.

In the split-tie-loop method to find single strip loop on manifolds with boundaries, the algorithm design decision is to let the strips cross over from original mesh boundaries and post-process these cross-over points to split and tie the cross-over strips to lie within the original mesh.

We create a manifold out of a manifold with boundary by adding one hypothetical vertex for each closed boundary loop and connect the boundary vertices of the mesh to the corresponding hypothetical vertex. The dual of the above manifold mesh has a perfect matching and the sequence of unmatched edges forms disjoint strip loops in the primal. Note that there might be strips that cross over boundary edges between the original mesh and the hypothetical mesh. We observe that, because of the adjacency relationship between the hypothetical triangles, the cross-over boundary edges occur in pairs sharing a common boundary vertex.

The idea is to do 'surgery' around the triangle fan incident on the boundary vertex shared by cross-over boundary edges to create a trail for the outgoing and incoming strips in either end to be connected. The surgery is done as follows (refer to Figure 4). First tag the boundary edges as matched

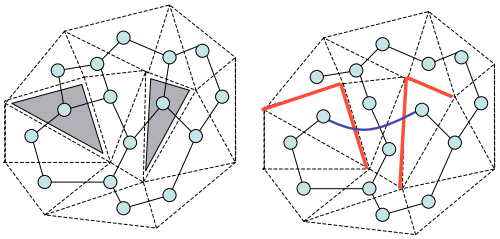

Figure 5. Linear strip algorithm: Given the start and end triangles of the strip, two edges of each of these triangles are cut and the triangles are pasted to each other along one of these cut edges. Single strip loop is found in the resulting manifold with boundary. Finally, the mesh is again cut along the pasted edge to create the required linear strip.

edges. Then processing the edges around the boundary vertex in the counter clockwise direction, if the current edge e is a matched edge, tag it as unmatched and move on to the next edge. If the current edge e is an unmatched edge then split the incident triangles. Let the two new edges be L and R. The edge L is tagged as a matched edge if e is not the first internal edge of the fan. The edge R is tagged as matched if the next edge $e+1$ in the fan is matched and is not the last internal edge of the fan. Strip loops can be formed by connecting triangles along the unmatched edges. These loops can be merged to form a single strip loop as explained in [4]. In summary, following are the possible cases:

IF e is unmatched split triangles
  IF e is the first internal edge
[Case 1] THEN left split edge L is unmatched.
[Case 2] ELSE the left split edge L is matched.
  IF $e+1$ is not the boundary edge
    IF $e+1$ is unmatched
[Case 3] THEN the right split edge R is unmatched.
[Case 4] ELSE the right split edge R is matched.
[Case 5] ELSE the right split edge is unmatched.
[Case 6] ELSE unmatch e.

An illustration of this method is given in Figure 4.

The above algorithm reduces the number of splits *given* a pair of boundary edge crossings. But the number of splits can be reduced further by reducing the number of boundary edge crossings. In order to achieve that, we assign high weights to the boundary edges and use a weighted graph matching algorithm that maximizes the sum of the weights of chosen matched edges. Boundary edges are preferred to be matched, thus reducing the number of strip crossovers.

'

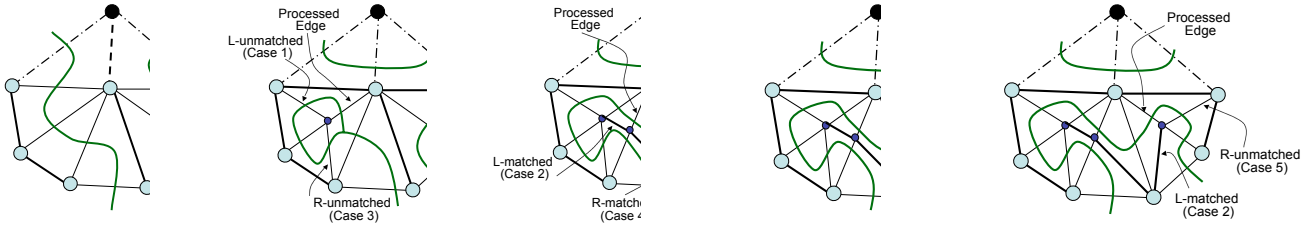

Figure 4. Split-tie algorithm: Once we define the adjacency of the hypothetical triangles (shown as dashed lines) added across the boundary edges of the mesh, we observe that the strip loops crosses the boundary edges in adjacent pairs. In order to split and tie the ends of the strip escaping out of the mesh, we process the edges of the fan of the boundary vertex in counter clockwise order. The edge under processing is shown using arrows. The dark edges are matched edges and the strips are shown as green curves. First, the boundary edges are made matched edges. Then subsequent edges are processed as described in Section 3.2. The final result shows the redirecting of the strip to be completely inside the mesh.

Model	#holes	#faces	(a) Final #faces	(b) %extra faces	(c) seconds	(d) Final #faces	(e) %extra faces	(f) seconds
Bunny	5	9580	13056	36.28	2.77	9816	2.46	1.91
Blob-8	8	16022	16318	1.85	10.68	16266	1.52	7.86
Blob-24	24	15922	16212	1.82	10.32	16200	1.75	8.17
Blob-40	40	15826	16122	1.87	10.69	16134	1.95	8.84
Skull	10	22046	22642	2.70	12.07	22578	2.42	6.97
Armadillo	172	344912	351040	1.78	2438.00	351144	1.81	1048.00

Table 1. Stripification results of 6 models: (a) Number of faces after applying (sub-optimal) force-loop method. (b) Percentage of added faces for same method. (c) Seconds to compute single-strip. (d,e,f) Same as (a,b,c) for split-tie method.

4. Creating Single Linear Strips

We use the above algorithms that produce single strip loops on manifolds with boundaries to produce a single linear strip. This strip starts and ends at any two arbitrary (but distinct) triangles on a manifold with or without boundaries. We achieve our goal by using topological surgery and the single strip loop finding technique.

Given the required start and end triangles T_1 and T_2, we have to construct a strip that starts from T_1, passes through all other triangles in the model exactly once and ends in T_2. We cut along two edges of T_1 and T_2 and make the triangles into boundary triangles (with two boundary edges each) and the mesh into a manifold with boundary. Then we paste T_1 and T_2 to each other along one of their cut edges. Now, each of these triangles has only one boundary edge. Next we construct a single strip loop that passes through the entire model with this changed topology, using one of the algorithms explained in previous sections. In this strip loop, triangles T_1 and T_2 have to be adjacent to each other, since the third edge in either of these triangles is a boundary edge, and the strip does not cross over boundary edges. Finally, we separate T_1 and T_2, to paste them back at their original positions in the mesh. After this restoration, the strip loop becomes a linear strip with T_1 and T_2 as its end points.

5. Implementation and Results

We tested the two stripification methods on a set of manifold models with varying number of boundaries, and the results are shown in Table 1. As we can see, the number of added triangles for most models is between 1.5% and 2.5% of the input size. As mentioned before, the increase in number of faces with the modified forced loop algorithm is not predictable, and hence in one case (the bunny model) it induces more than 35% of faces due to triangle split operations. It has been pointed out in [4], that in practice in manifold meshes, the number of additional faces is less than 2% of the original number of faces. The time spent in obtaining the results is dominated by the time to compute the weighted matching. For this we used LEDA's implementation [9] of this algorithm. Substantial improvements can be achieved with an alternate implementation that takes into account the properties of the dual graph of a triangulated manifold, specially in large models such as the one in Figure 9.

The preprocessing time for the (suboptimal) modified forced-loop algorithm is quadratic in the number of junction nodes. On the other hand, the split-tie method takes linear time in the size of the boundary. These differences can help decide which method to use for a particular input mesh.

In order to obtain the single strip, we allow the addition of some Steiner vertices that do not affect the geometric fidelity of the mesh. In the case of the Split-tie method, the number of additional vertices is bounded above by a factor linear on the size of the boundary. Specifically, the upper and lower bounds are $3n - 4\log_2 n$. For the modified forced-loop method, the upper bound is linear on the total size of the mesh.

Figure 6 shows the results of applying the modified forced loop algorithm to the blob and bunny models. In the blob model there are many unmatched triangles (junction nodes). Therefore, there exist shorter paths between them, which require fewer triangle splits. But in the bunny model there are only two junction nodes, and they are are very far apart, such that the shortest path between them is long enough to require a large number of triangle splits. This introduces around 35% additional faces. Figure 7 shows the single stripification of the same models using split-tie-loop method. A clearer illustration of the split-tie-loop method using an example of a sphere with a boundary is shown in Figure 8. Stripification of a large model and a linear stripification from and to given arbitrary triangles are shown in Figures 9 and 10.

6. Conclusion

We have introduced two methods for obtaining a single strip in a triangulated manifold with boundaries. This constitutes a generalization of [4], which only handles manifolds without boundaries.

Furthermore, we used the techniques above to obtain a non-cyclic single strip on the input mesh, where the start and the end points are chosen arbitrarily.

7. Acknowledgments

We would like to thank Anusheel Bhushan for his work in the early stages of this and previous versions of the project. We also want to thank the reviewers for their useful suggestions.

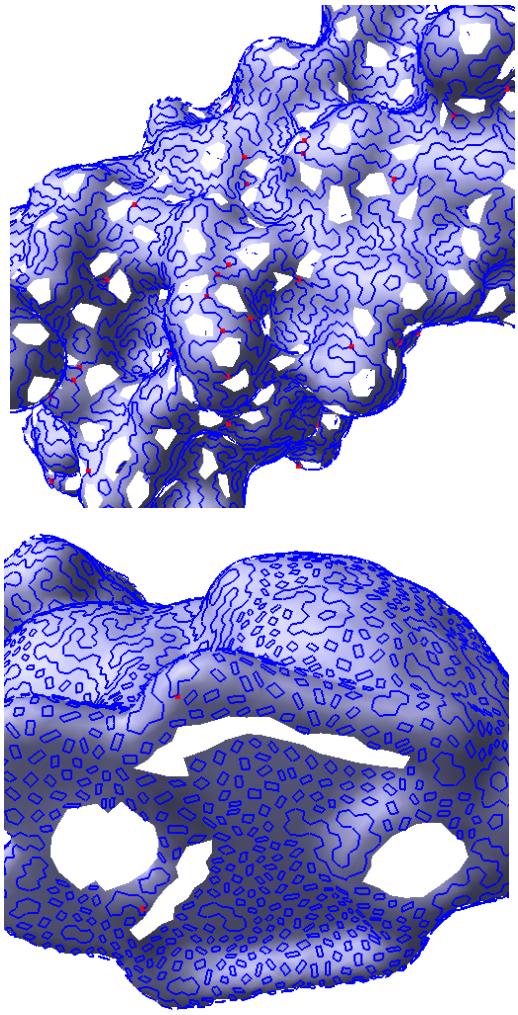

Figure 6. Modified forced-loop method on two models. Junction nodes shown in red. The quality of the result depends on the distance between junction nodes. Top: Many junction nodes produce few triangle splits. Bottom: Two far away junction nodes produce a large number of triangle splits.

References

[1] R. Bar-Yehuda and C. Gotsman. Time/space tradeoffs for polygon mesh rendering. *SIGGRAPH 96*, 15(2):141–152, 1996.

[2] P. Diaz-Gutierrez, A. Bhushan, M. Gopi, and R. Pajarola. Constrained strip generation and management for efficient interactive 3d rendering. In *Computer Graphics International Conference*, 2005.

[3] M. Gopi. Controllable single-strip generation for triangulated surfaces. In *Pacific Graphics 2004*, pages 61–69. IEEE, Computer Society Press, 2004.

[4] M. Gopi and D. Eppstein. Single strip triangulation of manifolds with arbitrary topology. *Computer Graphics Forum (EUROGRAPHICS)*, 23(3):371–379, 2004.

[5] S. Gumhold and W. Strasser. Real time compression of triangle mesh connectivity. In *Proceedings SIGGRAPH*, pages 133–140, 1998.

[6] H. Hoppe. Optimization of mesh locality for transparent vertex caching. In *SIGGRAPH 1999*, pages 269–276. ACM Press/Addison-Wesley Publishing Co., 1999.

[7] M. Isenburg. Triangle strip compression. In *Graphics Interface 2000*, pages 197–204, 2000.

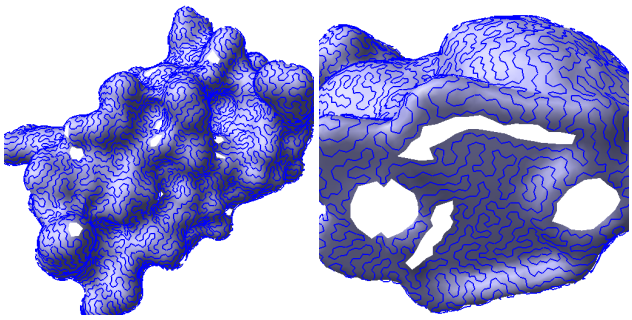

Figure 7. Single-strips obtained with the Split-tie method on two models.

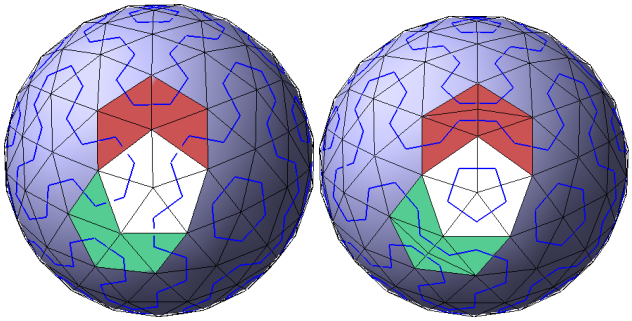

Figure 8. Split-tie method: Forcing the strip not to cross a boundary. Left: Hole shown in white; The 2 processed triangle fans in red and green Right: After processing the triangle fans, the strip avoids the boundary.

[8] T. Lewiner, H. Lopes, J. Rossignac, and A. Vieira. Efficient edgebreaker for surfaces of arbitrary topology. In *Proc. 17th Brazilian Symposium on Computer Graphics and Image Processing*, pages 218–225, 2004.

[9] S. Naher. LEDA — a library of efficient data types and algorithms. *LNCS*, 665:710–711, 1993.

[10] J. P. C. Peterson. Die theorie der regularen graphs (The Theory of Regular Graphs). *Acta Mathematica*, 15:193–220, 1891.

[11] J. Rossignac. Edgebreaker: Compressing the incidence graph of triangle meshes. *IEEE Trans. on Visualization and Computer Graphics*, 5(1):47–61, January-March 1999.

[12] J. Rossignac and A. Szymczak. Wrap&zip decompression of the connectivity of triangle meshes compressed with edgebreaker. *Journal of Computational Geometry, Theory and Applications*, 14(1-3):119–135, November 1999.

[13] A. J. Stewart. Tunneling for triangle strips in continuous level-of-detail meshes. In *Proceedings Graphics Interface*, pages 91–100, 2001.

[14] G. Taubin. Constructing hamiltonian triangle strips on quadrilateral meshes. In *Int. Workshop on Visualization and Mathematics and IBM Research Tech. Rep. RC-22295.*, 2002.

[15] X. Xiang, M. Held, and J. S. B. Mitchell. Fast and effective stripification of polygonal surface models. In *Proceedings Sym. Interactive 3D Graphics*, pages 71–78, 1999.

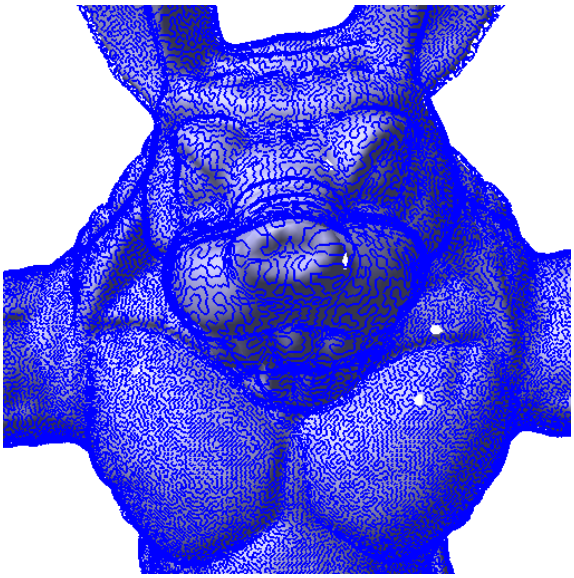

Figure 9. Stripification of a large (344k faces) model.

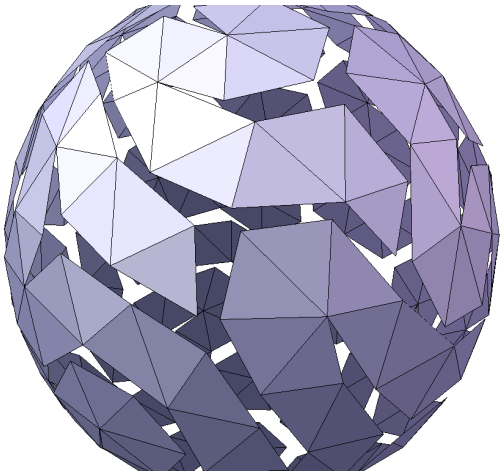

Figure 10. Non-cyclic single strip representation on a sphere. The first and last faces appear in the front.