

Quadrilateral and Tetrahedral Mesh Stripification Using 2-Factor Partitioning of the Dual Graph

Pablo Diaz-Gutierrez*

M. Gopi†

Computer Graphics Lab
Department of Computer Science
University of California, Irvine

ABSTRACT

Finding a 2-factor of a generic graph is a difficult problem and there are randomized algorithms proposed to solve this problem in $O(n^3)$ complexity [12]. In this paper, we propose algorithms that find a 2-factor of a graph, if one exists, for a restricted class of graphs in which all vertices have degree four or less, in $O(n^2)$ complexity where n is the number of vertices of the graph. Such graphs are actually dual graphs of quadrilateral and tetrahedral meshes that are widely used in graphics and visualization applications. We use the 2-factor of these graphs to find linear ordering of the primitives in the form of strips. Applications like compression, access and rendering of such data benefits a lot from such linear ordering. We use the similarity between the dual graphs of the quadrilateral and tetrahedral meshes to introduce a novel, unified graph based algorithm to produce quad and tetrahedral strip representations respectively. Further, by introducing a few additional vertices, we can represent the entire quad-surface using a single quad-strip loop. We can use a similar technique to reduce the number of tetrahedral strips, to represent the entire tetrahedral mesh.

CR Categories: I.3.5 [Computer Graphics]: Geometric Algorithms—Quadrangulation, Tetrahedralization, Stripification; K.7.m [Graph Algorithms]: k-factor—Perfect Matching, 2-factor;

Keywords: Single-strip, weighted perfect matching, quadrilateral strips, tetrahedral strips, 2-factor, graph algorithms.

1 INTRODUCTION

Quadrilateral and tetrahedral meshes are fundamental geometric structures in many mechanical and scientific simulations, and visualization. Due to the importance of these primitives, a number of algorithms have been designed to create these primitive representations of the given mesh from other representations [1, 2, 3, 7]. Given a mesh with quadrilateral or tetrahedral representation, a linear ordering of these primitives can be used in geometric processing applications including rendering and compression of large data sets [8]. In fact, many compression techniques yield primitive strips as a byproduct [10, 9, 8, 17, 11, 14, 19]. Most of the compression based stripification algorithms are greedy algorithms that collect primitives while walking along the mesh elements. Further, there are specific algorithms for tetrahedral and quadrilateral stripification. For regular tetrahedral meshes, [13] suggests space-filling curve approach for stripification. Heuristics for strip generation from rectangular ‘patches’ of quadrilaterals was suggested by [5] and was used by [16]. Further, given a polygonal mesh, algorithm to efficiently decompose them to create a triangular strips was suggested

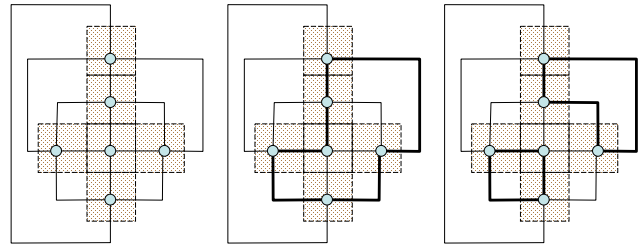


Figure 1: A cube and its dual graph. A 2-factor of the graph is a 2-regular spanning graph. Two possible 2-factors of the dual graph are shown. They define disjoint quadrilateral strip cycles on a manifold quadrangulation. Note that the edges that belong to the complement of 2-factor also define disjoint cycles in the dual of a quadrangulated manifold.

by [21]. In this context, there are also works on creating triangle strips specifically from a quadrilateral meshes [18, 20].

In this paper, given a quad or tetrahedral mesh, we propose a graph based algorithm that performs a global analysis of the mesh to find quad and tetra strips. This algorithm takes advantage of the similarity in the dual graph structures of quad and tetrahedral meshes and presents a unified solution for the problem in meshes with either of these primitives. Our work is closely related to the triangle strip generation algorithm by Gopi and Eppstein [6] that finds strips using the 1-factor in the dual graph of a manifold triangulated mesh. We use the 2-factor in the dual graph of tetrahedral meshes and manifold quadrilateral meshes.

1.1 Main Contributions

- The fundamental contribution of this paper are algorithms to find the 2-factor from a graph with degree four or less. This 2-factor finding algorithm has many more applications than just in quadrilateral and tetrahedral stripification.
- Further, we present a unified algorithm to find quadrilateral and tetrahedral strips from quad manifold surface meshes and tetrahedral volume meshes using the above 2-factor finding algorithm.
- Efficient management of strips can be done if the number of strips is less. We propose novel and generalized subdivision techniques for quad and tetrahedral elements to merge strip-cycles and minimize the number of strip loops. In the process, we can achieve a single Hamiltonian quadrilateral strip covering the entire manifold quadrilateral mesh.

We discuss the theory of k -factor of the graph in Section 2 and propose algorithms to compute a 2-factor of the graph. A 2-factor of the dual graph of a quad or tetrahedral mesh implicitly defines quad and tetra strips. In Section 3 we present subdivision techniques to

*e-mail: pablo@ics.uci.edu

†e-mail:gopi@ics.uci.edu

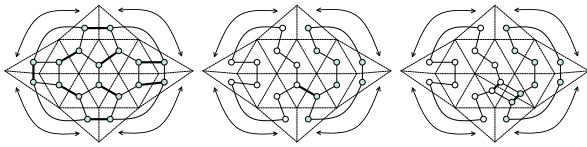


Figure 2: **Top:** (a) The dual degree three graph of the triangulation of a genus 0 manifold and a perfect matching shown by dark edges. (b) The set of unmatched edges create disjoint cycles. Two such cycles are shown. These disjoint cycles are connected to each other by matched edges. The algorithm constructs a spanning tree of these disjoint cycles and hence chooses 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. Reproduced from [6].

merge these disjoint strips. Finally, we discuss implementation and results of our stripification algorithms in Section 4.

2 QUADRILATERAL AND TETRAHEDRAL STRIPIFICATION

The dual of the quadrilateral mesh representation of a manifold, a *dual quad-graph*, is a 4-regular graph (every node has degree 4). For all practical purposes, tetrahedral volume mesh form 3-manifold surfaces *with boundaries*. Hence its dual graph, a *dual tetra-graph*, will have nodes with less than degree four on the boundaries. But for this difference, these dual graphs have very similar structures and hence most of the graph algorithms that are applicable to one dual graph can be applied to the other also.

In this paper, we propose algorithms that are applicable to both dual quad- and tetra-graphs to find disjoint cycles in the graph that translates to disjoint quad/tetra strip loops in the primal mesh. Post-processing of these strips are dependent on manifoldness of the mesh and hence independent techniques are developed for quad and tetra strips.

The fundamental concept we use to develop quad- and tetra-strips is the 2-factor of a graph. In the next section, we discuss the classical definition of a 2-factor of a graph. In Section 2.2 we present two algorithms to find a 2-factor in the dual graph which implicitly defines a partition of the quadrilateral and tetrahedral primal meshes into quad and tetrahedral strips.

2.1 The 2-factor of a Graph

Definition 1: A *k-factor* of a graph G is a spanning *k-regular* sub-graph of G .

For example, a 1-factor *matches* every node of the graph with one and exactly one of its neighbors. A 1-factor is also called a *perfect matching*. By Peterson’s theorem [15], a 3-regular, 3-connected graph always has a perfect matching. Using the fact that the dual graphs of triangulated two manifolds are 3-regular, 3-connected graphs and that a perfect matching exists for such graphs, Gopi and Eppstein [6] construct triangle strip loop partitions and eventually a single triangle strip loop of a triangulated two manifold of any arbitrary genus. Specifically, since every node has degree three and exactly one of the edges is chosen by the 1-factor graph, the rest of the two edges form disjoint loops that partition the vertex set of the dual graph and hence the input mesh. These disjoint loops were merged by subdividing two adjacent triangles belonging to two different cycles into four triangles by inserting a new vertex in the midpoint of the shared edge. This creates a single strip cycle covering all the triangles in the input mesh. This algorithm is illustrated in Figure 2.

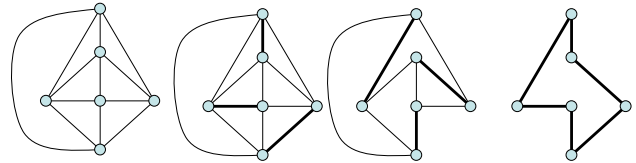


Figure 3: Illustration of a two-pass graph matching algorithm on the dual of cube. (a) Dual graph of a cube. (b) First pass perfect matching. (c) Remove the matched edges from the first pass and run the second pass of graph matching. (d) Union of matched edges from (b) and (c) gives a 2-factor.

We use a similar approach to construct quad and tetra strips using the following classical result attributed to Peterson [15].

Theorem 1: Every regular graph of even degree has a 2-factor.

The implication of the above theorem is that the dual quad-graph of a two manifold, which is a 4-regular graph, has a 2-factor; that is, there is a subgraph in which every vertex has degree two. Hence all the nodes in the dual graph along with the chosen edges in the 2-factor form disjoint loops and thus quad-strip loops in the primal quadrangulation.

Since the dual tetra-graph is not a regular graph, the above theorem does not apply to tetrahedral meshes. Hence the 2-factor finding algorithms that are explained in the subsections below, when applied to the dual tetra-graphs might not produce a 2-factor – not all nodes might have degree two. In other words, the stripification of the primal tetrahedral mesh may contain both linear strips and strip-loops. Since linear tetrahedral strips also are acceptable as solutions to our problem, we propose the following 2-factor finding algorithms as unified algorithms for the stripification of both quadrilateral and tetrahedral meshes.

2.2 Algorithms for Finding a 2-Factor

Most recent work [12] uses randomized algorithms to find a 2-factor in sparse graphs with n vertices in $O(n^3)$ expected time. Our graphs are special graphs that are dual to quad/tetra meshes which might lead to simpler solutions. Further, there are many implementations of graph matching (the 1-factor finding) algorithm available in public domain that we can use to find the 2-factor. Here we present two algorithms to find the 2-factor in the dual quad/tetra graph both in $O(n^2)$ expected time.

2.2.1 Two Pass Graph Matching Algorithm

The first algorithm, which we call the *two-pass graph matching method*, applies the graph matching algorithm twice on the input dual graph to get a 2-factor. Specifically, we run the graph matching algorithm once, remove the matched edges from the input graph, and run the matching algorithm again on the new graph. The union of matched edges chosen from these two runs gives us a 2-factor of the original graph, and hence disjoint cycles in the mesh. This algorithm is illustrated in Figure 3. We use public implementation of a cardinality graph matching algorithm that gives a perfect matching if one exists and maximizes the matching, otherwise.

This simple algorithm produces correct results for most of the practical models but does not work on graphs with odd number of nodes even if they are 4-regular (Figure 4). But most of such odd-node graphs either do not have an orientable 2-manifold geometric realization or have geometry (e.g non-planar faces) and topology (e.g. two quadrilaterals sharing two edges) that are usually unacceptable in graphics and visualization applications. We show one

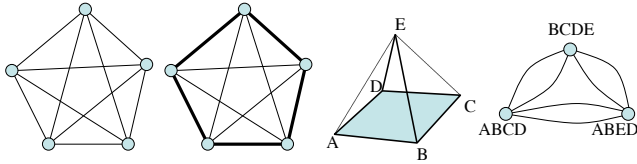


Figure 4: Left: A five node 4-regular graph with no 1-factor (perfect matching) but has a 2-factor as shown. Hence a two-pass graph matching algorithm to find a two-factor will not work in such graphs. Right: A geometric realization of a 3-node 4-regular graph. Every quadrilateral shares two edges with its neighbor – usually an unacceptable geometry for graphics and visualization applications. A simple cycle in the dual graph is its 2-factor.

geometric realization of an odd-vertex dual graph in Figure 4. Our graphs are derived from geometric meshes used in visualization applications and in fact, all the models we tried with the two-pass graph matching method yielded a 2-factor, if one existed.

The run-time complexity of the matching algorithm is $O(mn)$ where m is the number of nodes and n is the number of edges. Since the number of edges and nodes are of the same order in our dual graph, the complexity of the template-substitution method is $O(n^2)$.

2.2.2 Template Substitution Algorithm

The second algorithm, *the template substitution method*, is guaranteed to find a 2-factor, if one exists, on any graph in which every vertex is of degree four or less. Specifically, the graph need not be a 4-regular graph and hence this algorithm is suitable for the dual tetra-graphs also.

In this method, we first transform the input dual graph G into a new graph G' by substituting every node in G with the template shown in Figure 2.2.2. Let us call the graph G' , the *inflated graph*. We state and prove the following relationship between the 2-factor in the original graph G and the perfect matching (1-factor) in the inflated graph G' .

In the template that is substituted for each original node, in addition to the quadruplicates representing the original node, there are two more nodes. During graph matching on this template, these two additional nodes can get matched to two of the quadruplicates leaving exactly two other nodes to be matched outside the template. We call the process of adding these extra nodes to engage a subset of nodes in the template as *doping* for its resemblance to a similar process in semiconductor manufacturing. We use this concept of *doping* to prove the following theorem.

Theorem 2: There exists a 1-factor in the inflated graph if and only if there exists a 2-factor in the original graph.

Proof:

⇒ Let us assume that there exists a 1-factor in the inflated graph. This means that both the dopes are matched thus engaging exactly two of the quadruplicates. Since there exists a 1-factor, the other two quadruplicates are also matched, and have to be matched external to the template along the edges in the original graph. This yields a 2-factor in the original graph.

⇐ Let us assume that there is a 2-factor in the original graph. Hence exactly two of the quadruplicates are connected external to the template structure thus leaving two other nodes among the quadruplicates to be matched internally with the dopes. Note the important connectivity structure of dopes inside the template: *any* two nodes among the quadruplicates can be matched with the dopes. Hence, given any 2-factor, there exists a perfect matching in the inflated graph. □

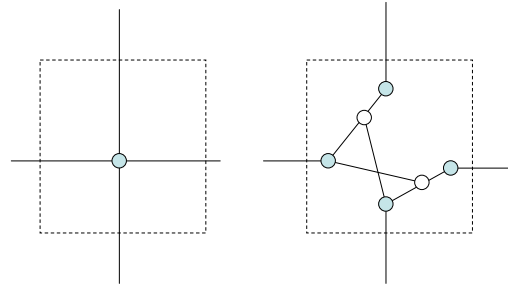


Figure 5: Left: Dual graph node of one quadrilateral or tetrahedron of a quad/tetra mesh. Right: The template that is substituted in the dual graph. Unshaded nodes are the new *dope* nodes added to the dual graph. If there exists a 2-factor outside this template, that is, if exactly two of the quadruplicates are matched externally, the dope nodes are matched with remaining two nodes thus producing a perfect matching for the entire graph. Note the internal connectivity with the dope nodes that enables perfect matching given *any* combination of two externally matched edges.

A 1-factor in the inflated graph gives a 2-factor in the original graph and hence quad/tetra strip loops in the primal mesh. Specifically, the inflated graph dual of a quadrangulated 2-manifold mesh will always have a 1-factor and can be found using a (cardinality) graph matching algorithm whose public implementations are available. Such a 1-factor is not guaranteed in the inflated dual graph of a tetrahedral mesh (with boundary) since a 2-factor is not guaranteed by Peterson’s theorem.

If there is no 1-factor, the graph matching algorithm will maximize the cardinality of matchings leaving minimum number of nodes unmatched. These unmatched nodes might be either the nodes of the quadruplicates or the dopes. If any of the dope nodes is unmatched, we break the external matching of quadruplicate nodes to match them internally with the free dope nodes. Thus in the original graph every node will have either two or fewer matched edges. All tetrahedra in the primal mesh corresponding to unmatched dual graph nodes (that is, with zero matched edges) are defined as singleton strips. From each of the tetrahedron corresponding to the dual node with exactly one matched edge, we follow the matched edges to form a linear strip of tetrahedra till it reaches another tetrahedron with exactly one matched edge. Rest of the tetrahedra have two matched edges each and hence they form disjoint tetrahedron strip loops.

Instead of using matched edges, an alternative approach is to use the unmatched edges to define strips. This approach is similar to that of [6] to use unmatched edges to define triangle strips. The advantage of this approach is that there will be no singleton tetrahedral strips since every node will either have two, three, or four unmatched edges (since they have two, one or zero matched edges). A suitable traversal along the unmatched edges will again produce tetrahedral strip loops or linear strips, and every tetrahedron in the mesh will belong to one of the strips. In fact, we follow this approach in our implementation.

The run-time complexity of the matching algorithm is also $O(n^2)$ as the two-pass graph matching algorithm. But this algorithm uses the inflated graph which has six times more nodes and edges than the original graph. Hence this algorithm is approximately 18 times slower than the two-pass graph matching method, but finds a two factor in any graph with vertex degrees four or less.

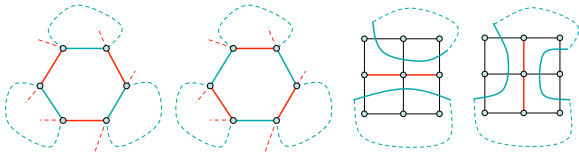


Figure 6: Left: The dual graph of a quad/tetra strip. Six primitives incident on an $(n-2)$ -dim simplex. The red edges are matched edges and the cyan edges show the unique disjoint cycles. Every node has degree four. Alternating edges are matched and unmatched edges. Toggling these assignments merges the incident cycles. Right: Geometric realization of the dual graph in quadrilateral mesh with four incident quadrilaterals on a vertex. Similar arrangement can be realized in tetrahedral mesh also with even number of tetrahedra incident on an edge.

3 MERGING CYCLES

Finding a single simple cycle that connects all nodes of the graph is NP-complete. But we can merge the strips yielded by the above algorithms to form a single strip by subdividing the mesh primitives. Even though multiple-strip partitions are valid representations for many graphics and visualization rendering applications, [4] showed that the overhead of maintaining multiple strips over a single strip is significant. Further, strips are used in many other applications like connectivity compression, and such applications would benefit from a single strip representation.

First, we describe an extension of the nodal vertex processing algorithm that was used to merge disjoint triangle strip cycles [6], to process quadrilateral strip cycles. Further, the same concept can be extended to tetrahedral mesh which we call *nodal edge processing*. Such nodal vertex/edge processing does not require subdivision of primitives but reduces the number of disjoint loops in the entire stripification of the model.

The remaining cycles after the nodal vertex/edge processing are merged through subdivision processes. Since the subdivisions are dependent on the geometry, even though the dual quad- and tetra-graphs are locally similar, we have to apply different subdivision techniques depending on the mesh primitive. We discuss independent algorithms for quadrilateral and tetrahedral subdivisions.

3.1 Nodal Fan-Simplex Processing

Since this processing is applicable to both quadrilaterals and tetrahedra, we refer to them in generic terms as primitives. Further, in a manifold (with boundaries) mesh one or two primitives are incident on an $(n-1)$ -dim simplex (two quadrilaterals on an edge, and two or less tetrahedra on a triangular face) and a number of primitives form a fan around an $(n-2)$ -dim simplex (quadrilateral fan around a vertex and tetrahedral fan around an edge). We call an $(n-2)$ -dim simplex, generically, as a *fan simplex* and hence the nodal vertex/edge processing as fan-simplex processing.

The goal of this optimization is to increase the length of the disjoint cycles by merging many cycles without any primitive splits. Assume that we have already constructed a 2-factor, and partitioned the primitives of the input mesh into disjoint cycles. We classify a mesh fan-simplex v as a nodal fan-simplex if it satisfies the following conditions: q_n , the number of primitives incident on v is even, the total number of unique disjoint cycles that these incident primitives belong to is $\frac{q_n}{2}$, and the incident primitives do not share more than one $(n-1)$ -dim simplex between the adjacent neighbors.

An example of a nodal fan-simplex processing with four primitives and two unique incident cycles in each of them is shown in Figure 6. The neighborhood of every nodal fan-simplex is modified such that the matched and unmatched primitive pairs are tog-

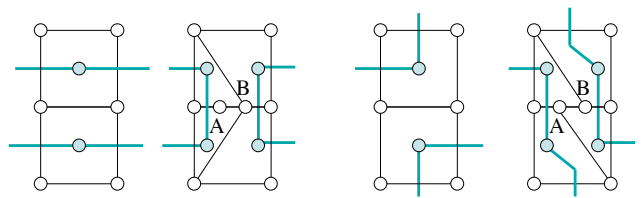


Figure 7: Subdividing a quadrilateral pair using two vertices (A and B). Note that in the case of strips crossing non-adjacent edges, after subdivision, a pair of adjacent subdivided quadrilaterals share two edges. Further, depending on the configuration of the strip-path the connectivity of edges from A and B changes.

gled. This merges all the incident cycles into one cycle. If we use a union-find data structure to keep track of which primitives belong to which cycles, we can test whether any mesh fan-simplex is nodal using a number of union-find queries proportional to the degree of the fan-simplex, so the total time for the optimization is $O(n\alpha(n))$ where α is the extremely slowly growing inverse Ackermann function.

Once this optimization is performed, we form the cycle graph of the remaining cycles, construct a spanning tree, and use the tree to guide primitive subdivisions as explained the following sections. This optimization step typically significantly reduces the number of subdivisions that must be performed, but we have no theoretical guarantees on its performance. The results on typical number of instances when such an optimization is done is given in Tables 1 and 2

3.2 Quadrilateral Subdivisions for Merging Strips

In a manifold quadrilateral mesh, there are exactly twice the number of edges than faces. Using this fact, we can prove that in order to maintain the Euler characteristic of the mesh during subdivision, the increase in number of vertices and faces have to be the same. In order not to affect the results of the matching algorithm, the new faces should potentially be matched to each other; in other words, there should be even number of additional faces, and hence we can insert only even number of new vertices during subdivision.

Given the above argument, the minimum number of vertices you can introduce for a quadrilateral mesh subdivision is two. Figure 7 shows two such subdivisions. First, we observe that the subdivision of the quadrilateral itself is dependent on the configuration of strip cycle path in both of the quadrilaterals. Further, routing of the strips inside the subdivision also depends on the original strip path. This requires case-by-case handling of the subdivision. Second, we see that under one of the strip path patterns two new quadrilaterals share two edges between them – an undesirable topological configuration. We can also show that the subdivision with four new vertices also suffers from the same two drawbacks as adding two vertices.

The minimum number of vertices that we can add to a quadrilateral pair subdivision without the above two drawbacks is six (Figure 8). Each quad-pair subdivision has six *external quadrilaterals* (three in each of the parent quadrilaterals), two *internal quadrilaterals* (one in each of the parent quadrilaterals), and two new *shared vertices* in the shared boundary of the quad-pair. We call the four of the six external quadrilaterals that are in the strip-path as *strip-quadrilaterals*. Within each parent quadrilateral, we first route the strip such that it traverses through all the new quadrilaterals as follows: one external strip-quad \rightarrow internal-quad \rightarrow external quad that is not a strip-quad \rightarrow the other external strip-quad. If such routing is done in both the adjacent subdivided quadrilaterals then at least one of the shared vertices will be a *nodal vertex* around which the two strips can be merged using *nodal vertex processing algorithm*

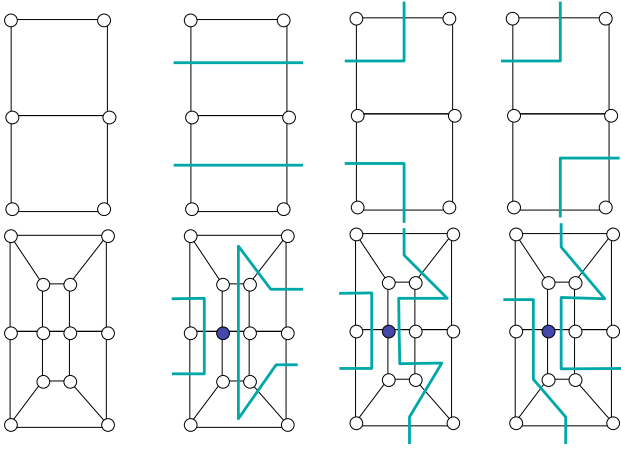


Figure 8: Subdividing a quadrilateral pair with six vertices. (a) The connectivity after subdivision is independent of the strip path. (b,c,d) The strip within each quadrilateral is routed in order to traverse through all the four quadrilaterals. Once such a routing is done, the appropriate vertex (shown shaded) on the shared edge is chosen for nodal vertex processing to merge the strips in both the quadrilaterals. All other configurations of strip paths are mirror reflections of the cases shown in the figure.

explained in the previous section. Such a processing removes the dependency of the geometric connectivity of the subdivided quadrilaterals from the strip path. Further, the routing can be done automatically without case-by-case analysis of the strip path configurations between the two quadrilaterals of the subdivided quad-pair.

3.3 Tetrahedral Subdivisions for Merging Strips

Tetrahedral subdivision for strip-merging is exactly similar to triangle subdivision for strip merging [6]. In triangulated models, two adjacent triangles belonging to different cycles are divided into two triangles each by introducing a vertex in the mid-point of the shared edge. Then a nodal vertex processing is performed at this new vertex to merge the cycles. In tetrahedral meshes, two adjacent tetrahedra belonging to different cycles are divided into three tetrahedra each by introducing a vertex in the centroid of the shared face. Then a nodal-edge processing is performed around one of the three newly introduced edges on the shared face. The process of finding this nodal edge is detailed below.

Unlike dual-quad/tri graph, the dual tetra-graph after subdivision is non-planar (Figure 9). First, as in quad/tri-strip merging method, strips in each of the tetrahedron is routed within the three subdivided tetrahedra. There is at least one adjacent tetrahedron-pair in the strip among the upper three tetrahedra, whose corresponding tetrahedra in the lower half also are adjacent in their strip path (tetrahedrons A, B, A', B' in Figure 9). In the dual graph, they form a four-cycle in which the matched and unmatched edges alternate. This corresponds to a nodal edge in the primal – the common edge between these four tetrahedra. Nodal fan-simplex processing is done around this edge to merge the upper and lower strips.

As discussed before, the stripification of tetrahedral meshes might produce a combination of linear strips and strip loops. We can merge two strips using subdivision if and only if they are disjoint and at least one of them is a strip loop. If both are linear strips then the subdivision and merging process would split and connect again to produce two linear strips.

4 IMPLEMENTATION AND RESULTS

Our system uses the LEDA implementation of graph matching algorithm. The two-pass matching method worked on almost all the quadrilateral meshes we tried. Since both the algorithms use the cardinality matching algorithm which is independent of its application (as in our case stripification), we do not have a metric to compare the quality of results of the strips. (If we had use weighted matching algorithm, then we can evaluate the quality of the strips based on its adherence to the weighting function.) On the other hand, the efficiency of these two algorithms can be compared. The template matching algorithm runs on an inflated graph of six times the number of edges and six times the number of nodes. On the other hand, the two-pass matching method runs the algorithm twice (but on a trimmed edge graph in the second pass). Hence there is an overall improvement of efficiency of around 20 times from template matching to two-pass matching method. Note the template matching algorithm works on all input graphs with degree four or less, even if it had odd number of vertices. Further, while the two-pass matching method may not work always work on tetrahedral meshes, template matching works on the dual of all tetrahedral meshes. Tables 1 and 2 gives more information about these algorithms on various inputs.

Model	Faces	Final faces	%inc.	Nodal verts	#edge #splits	time sec.
Sphere	640	646	0.9	26	1	0.12
Trico	2830	2902	2.5	119	12	0.99
Blob	8020	8146	1.6	334	21	6.89

Table 1: Quadrilateral Stripification: Note the significant number of nodal vertex processing that merges cycles without subdivision. The total number of cycles remaining after nodal vertex processing is one more than the number of edge splits. In spite of introducing four more quadrilaterals for every split, note that the percentage increase is very small. The result after the subdivisions and cycle mergings is one single Hamiltonian quadrilateral strip that traverses through all the quadrilaterals in the mesh.

Model	Tetra	Verts	Nodal Edges	Strip cycl.	Lin. str.	% inc. tetra/verts	Time secs.
Trico	15310	4633	139	376	12	9.8/8.1	132
Fandisk	22491	7144	290	471	2	8.3/6.6	414
Ball	430	162	3	9	0	7.4/4.9	<1
Spring	24359	7226	326	737	4	12.1/10.2	415
Blob	31526	8769	197	974	66	12.3/11.1	527

Table 2: Tetrahedral Stripification: The number of strip loops (cycles) and linear strips shown are after performing the nodal edge processing. The increase in number of tetrahedra and vertices are due to strip merging by subdivision. Note the number of new vertices (which equals the number of subdivisions) is exactly equal to $(\#Strip\ Loops - 1)$ – to merge all the loops, plus one to merge this loop with one linear strip, if one exists. Further, the number of new tetrahedra is four times the number of new vertices. The time taken is given for the template matching algorithm.

5 CONCLUSION

In this paper, we presented a new algorithm that finds the 2-factor of a (restricted class) of graph in $O(n^2)$ running time. We also presented novel algorithms that use this 2-factor in the dual graph to create quadrilateral and tetrahedral strips. The direction of the strips found by above methods depends on the results of the graph matching algorithm. As part of future work, we would like to use

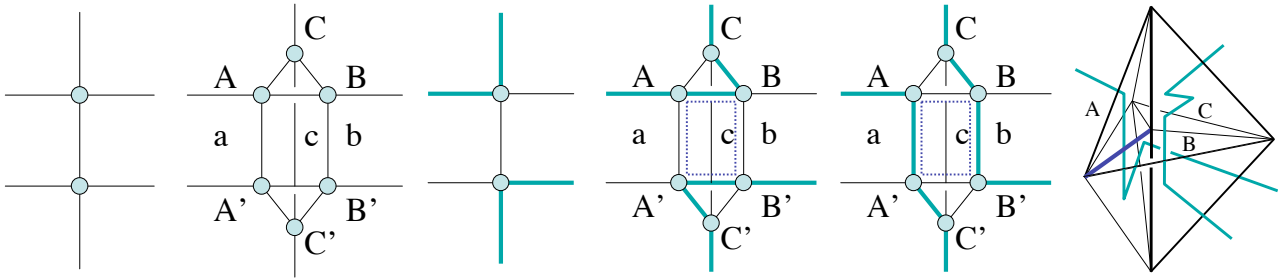


Figure 9: Subdividing a tetrahedron pair. (a-b) The dual graph before and after subdivision. (c) The strip path before subdivision. (d) Strips in each of the tetrahedron is routed within the three subdivided tetrahedra. There is at least one edge in the upper tetrahedron strip path, whose parallel path in the lower tetrahedron is traversed by the its strip path. In this example, edge AB in the upper and $A'B'$ in the lower tetrahedron are traversed. This forms a four-cycle (shown as dotted lines) along with the corresponding edges a and b in which the matched and unmatched edges in the dual alternate. This corresponds to a nodal edge in the primal, shown in dark blue in (f). (e) Merging of cycles by nodal-edge processing. (f) Result of subdivision and merging in the primal tetrahedral mesh.

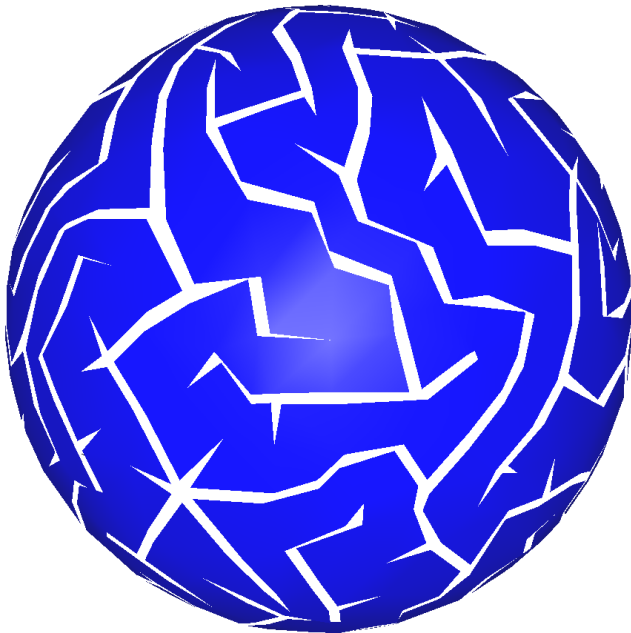


Figure 10: Representation of a single strip on an all-quads spheric model.

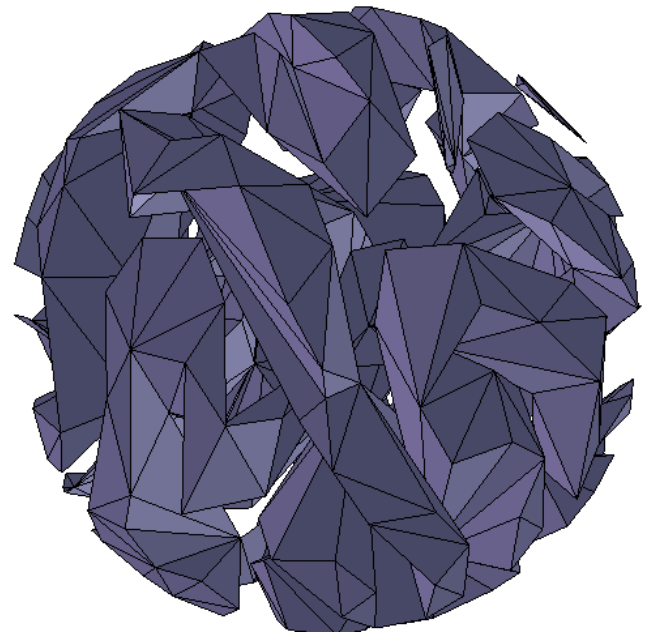


Figure 11: Rendering of the obtained tetrahedral strips on a spherical model.

the weighted graph matching to globally steer the strip in order to satisfy certain properties like normal based clustering. Similar techniques were used for back-face culling and transparent vertex caching in [4]. Further, we can use weighted perfect matching to choose the boundary triangles of the tetrahedral mesh to force external matching to reduce the number of internal tetrahedral strips. We would also like to develop out-of-core cardinality matching graph algorithms that can be applied on gigantic graphs to handle large models used in graphics and visualization applications.

Acknowledgments We would like to thank Hang Si, the author of TetGen that we used to tetrahedralize our models.

REFERENCES

- [1] Takao Asano, Tetsuo Asano, and Hiroshi Imai. Partitioning a polygonal region into trapezoids. *J. ACM*, 33(2):290–312, 1986.
- [2] Prosenjit Bose and Godfried T. Toussaint. No quadrangulation is extremely odd. In *Int. Symp. Algorithms and Computation*, pages 372–381, 1995.
- [3] H. E. Conn and J. O'Rourke. Minimum weight quadrilateralization in $O(n^3 \log n)$ time. In *Proc. of the 28th Allerton Conference on Comm. Control and Computing*, pages 788–797, 1990.
- [4] Pablo Diaz-Gutierrez, Anusheel Bhushan, M. Gopi, and Renato Pajarola. Constrained Strip Generation and Management for Efficient Interactive 3D Rendering. In *Proc. of Computer Graphics International Conference*, 2005.
- [5] F. Evans, S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *Proceedings IEEE Visualization 96*, pages 319–326. Computer Society Press, 1996.
- [6] M Gopi and David Eppstein. Single strip triangulation of manifolds with arbitrary topology. *Computer Graphics Forum (EUROGRAPHICS)*, 23(3):371–379, 2004.
- [7] E. Heighway. A mesh generator for automatically subdividing irregular polygons into quadrilaterals. *IEEE Trans. Magnetics*, 19(6):2535–2538, 1983.

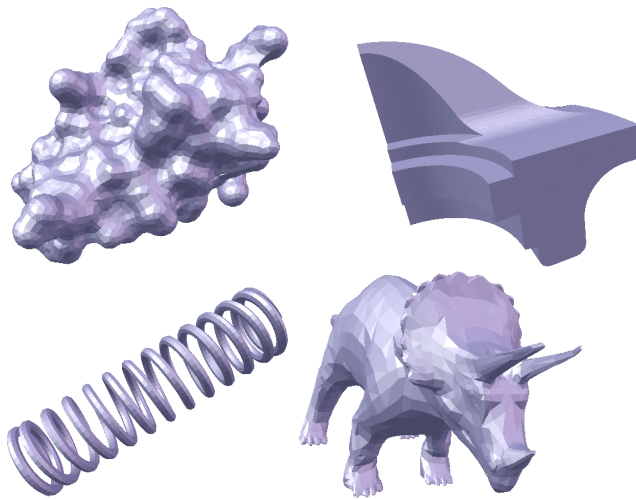


Figure 12: Rendering of used models: Blob, Fandisk, Spring and Trico. These original triangulated models were quadrangulated by combining triangles. Further, they were tetrahedralized by inserting a few extra vertices inside them to get high quality tetrahedra.

- [8] Davis King, Craig M. Wittenbrink, and Hans J. Wolters. An architecture for interactive tetrahedral volume rendering. Technical Report HPL-2000-121 (R.3), HP Laboratories Palo Alto, 2001.
- [9] P. N. Mallon, M. Boo, M. Amor, and J.D. Bruguera. Compression and onthefly rendering using tetrahedral concentric strips. Technical report, University of Santiago de Compostela, Spain.
- [10] Asish Mukhopadhyay and Quanbin Jing. Encoding Quadrilateral Meshes. In *15th Canadian Conference on Computational Geometry*, 2003.
- [11] Renato Pajarola, Jarek Rossignac, and Andrzej Szymczak. Implant sprays: Compression of progressive tetrahedral mesh connectivity. In *Proceedings IEEE Visualization 99*, pages 299–305. Computer Society Press, 1999.
- [12] G. Pandurangan. On a Simple Randomized Algorithm for Finding a 2-Factor in Sparse Graphs. *Information Processing Letters*, page accepted for publication, 2005.
- [13] V. Pascucci. Isosurface computation made simple: Hardware acceleration, adaptive refinement and tetrahedral stripping. In *Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization*, 2004.
- [14] Jingliang Peng, Chang-Su Kim, and C.-C. Jay Kuo. Technologies for 3d mesh compression: A survey. Technical report, Preprint, 2005.
- [15] Julius Peter Christian Peterson. Die theorie der regularen graphs (The Theory of Regular Graphs). *Acta Mathematica*, 15:193–220, 1891.
- [16] O. Sommer and T. Ertl. Geometry and Rendering Optimization for the Interactive Visualization of Crash-Worthiness Simulations. In *Proceedings of the Visual Data Exploration and Analysis Conference in IT&T/SPIE Electronic Imaging*, pages 124–134, January 2000.
- [17] Andrzej Szymczak and Jarek Rossignac. Grow & Fold: Compression of Tetrahedral Meshes. In *Fifth Symp. on Solid Modeling*, pages 54–64, 1999.
- [18] Gabriel Taubin. Constructing hamiltonian triangle strips on quadrilateral meshes. In *Int. Workshop on Visualization and Mathematics and IBM Research Tech. Rep. RC-22295.*, 2002.
- [19] Shyh-Kuang Ueng. Out-of-Core Encoding Of Large Tetrahedral Meshes. In *Volume Graphics*, pages 95–102, 2003.
- [20] Petr Vaneczek, Radek Svitak, Ivana Kolingerova, and Vaclav Skala. Quadrilateral meshes stripification. Technical report, University of West Bohemia, Czech Republic, 2004.
- [21] Xinyu Xiang, Martin Held, and Joseph S. B. Mitchell. Fast and effective stripification of polygonal surface models. In *Proc. Symp. on Interactive 3D Graphics*, pages 71–78. ACM Press, 1999.