

Optimal Decomposition of Polygonal Models into Triangle Strips *

Regina Estkowski[†]
HRL Laboratories
3011 Malibu Canyon Road
Malibu, CA, 90265
regina@hrl.com

Joseph S. B. Mitchell
Applied Math & Statistics
University at Stony Brook
Stony Brook, NY, 11794
jsbm@ams.sunysb.edu

Xinyu Xiang[‡]
Bloomberg, Inc.
499 Park Avenue
New York, NY, 10022
xxiang@bloomberg.net

ABSTRACT

Motivated by applications in computer graphics, we study the problem of computing an optimal encoding in “triangle strips” of a triangulation of a polygonal surface model. The goal is to facilitate the transmission and rendering of a polygonal model by decomposing its triangulation into a minimum number of “tristrips,” each of which has its connectivity stored implicitly in the ordering of the data points. While this optimization problem has been conjectured to be hard, its complexity status has been open. We prove that the tristrip decomposition problem is, in fact, NP-complete. We also propose two methods for solving the problem to optimality, one based on an integer programming formulation, one based on a branch-and-bound scheme that relies on lower bounding techniques for its efficiency. We perform an extensive set of experiments to test the efficiencies of these methods and some of their variants. These methods have been integrated also with the practical system FTSG (Fast Triangle Strip Generator), in order to utilize optimization methods on small subproblems to improve the quality of the heuristic solutions obtained by FTSG. We use experimentation to judge the quality of the improvements.

Categories and Subject Descriptors

I.1.2 [Symbolic and Algebraic Manipulation]: Algorithms; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

*Research partially supported by the US-Israel Binational Science Foundation, HRL Laboratories, NASA Ames Research, National Science Foundation, Northrop-Grumman, Sandia National Labs, Sun Microsystems.

[†]This work was done while the author was at the University at Stony Brook.

[‡]This work was done while the author was at the University at Stony Brook.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SoCG'02, June 5-7, 2002, Barcelona, Spain.

Copyright 2002 ACM 1-58113-504-1/02/0006 ...\$5.00.

General Terms

Algorithms, Experimentation, Theory

Keywords

triangle strips, tristrips, rendering, NP-completeness, integer programming, branch and bound

1. INTRODUCTION

The problem of computing a succinct encoding of a triangulation of a polygonal model arises in graphics and visualization, where it is important to be able to transmit and render massive models at real-time frame rates. Current 3D graphics rendering hardware often faces a memory bus bandwidth bottleneck in the processor-to-graphics pipeline. Visibility culling and model simplification methods help to reduce the number of triangles that must be transmitted, but it is also important to minimize the time needed to transmit n triangles that are to be rendered, e.g., by compressing the geometric and topological information in a model, transmitting the compressed data, and decompressing at the rendering stage, hopefully using a very small on-chip cache.

An ideal encoding for the smallest possible cache (a 2-vertex cache) is based on “tristrips”: when vertex v_{i+2} is transmitted, it determines a triangle together with the cache vertices v_i, v_{i+1} . A *tristrip* is an ordered sequence (with repetitions) of vertices, (v_1, \dots, v_m) , which *encodes* a set of $m-2$ triangles. A *sequential* tristrip encodes the set of triangles $\{(v_i, v_{i+1}, v_{i+2})\}$, $1 \leq i \leq m-2$. While we concentrate on the familiar sequential tristrip in this paper, one can also use a *fan* tristrip, encoding the set $\{(v_1, v_{i+1}, v_{i+2})\}$, $1 \leq i \leq m-2$, of triangles all incident on v_1 . Figure 1 gives an example showing a sequential tristrip encoding of 12 triangles.

If the triangulation having n triangles consists of a single tristrip, then only $n+2$ (rather than $3n$) vertices would have to be transmitted for a triangulation with n triangles. In general, if we are able to decompose a surface triangulation into k tristrips, we will need only $n+2k$ vertices.

In some cases (e.g., Iris GL), a tristrip may have *swaps*, special marks (bits) within a vertex sequence to indicate an exchange of the two cache vertices; alternatively (e.g., OpenGL), a swap can be effected by sending a zero-area triangle (transmitting an extra vertex instead of a “swap” bit). For example, $(1, 2, 3, \text{swap}, 4, 5)$ yields triangles $((1, 2, 3),$

(3, 2, 4), (2, 4, 5)), which also result from the sequence (1, 2, 3, 2, 4, 5) (with zero-area triangle (2, 3, 2)). A sequential tristrip is *pure* if it contains no zero-area triangles.

In this paper we focus on the case of pure sequential trisrips and address the problem of computing an *optimal* partition of a triangulated model into the fewest trisrips.

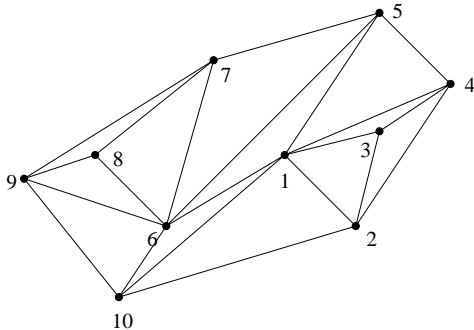


Figure 1: The triangulation is encoded using one trisrip: (1,2,3,4,1,5,6,7,8,9,6,10,1,2).

Summary of Results We give both theoretical and experimental results:

- (1) We establish the complexity of the k -STRIPABILITY problem, showing that it is NP-complete to decide if a given triangulation can be decomposed into k trisrips, even if the input triangulation is homeomorphic to a sphere. This settles an open question posed in 1993 by J. Rossignac.
- (2) We devise two classes of algorithms for computing an optimal decomposition of a triangulation into trisrips – one based on an efficient integer programming formulation, and one based on a branch-and-bound algorithm, utilizing some effective lower bounding techniques. These are the first algorithms that have been developed and implemented to solve the trisrip optimization problem.
- (3) We have implemented our new algorithms and integrated them into a leading practical software package (FTSG) that computes stripifications.
- (4) We have conducted an extensive experimental study of the relative merits of the new algorithms and compare their performance to that of a straightforward enumerative search for an optimal solution. Also, as our algorithms are the first to be implemented that produce a guaranteed optimal decomposition, they help give the first insight into how close to optimal the heuristic methods used in practice tend to be.
- (5) We also have devised a method to combine the fast heuristic-based methods of the FTSG system with our guaranteed optimization methods, allowing a trade-off between speed and quality of solution. Since our optimization algorithms are only capable of solving to optimality problems having about 100 triangles in a reasonable amount of time, this capability is important in being able to study the decomposition problem on much larger models as arise in practice.

Related Work. The first algorithmic study of problems related to trisrips, both sequential and Hamiltonian, appears in Arkin et al. [2], who consider sequential triangulations of point sets and polygons. In Evans et al. [9], it was shown that it is NP-complete to determine if a surface having holes and untriangulated faces can be triangulated in such a way that the resulting triangulated surface can be decomposed into k trisrips. While this result was intended to partially address the complexity of the optimization problem posed by Rossignac, it relied strongly on the flexibility of having untriangulated faces and on the use of many holes (very high genus) in the construction.

Deering [7] considers *generalized triangle meshes*, having a cache for > 2 vertices, to facilitate geometry compression; particularly efficient algorithms for obtaining such compressions are given by Chow [5]. Bar-Yehuda and Gotsman [4] use planar separator theorems to show that a vertex cache of size $\Theta(\sqrt{n})$ is sufficient and sometimes necessary to attain optimal transmission (one vertex per triangle). Gumhold and Straßer [11] have developed a connectivity compression and decompression method, using a cache that permits one vertex per triangle transmitted.

Heuristic methods for decomposing triangulations into a small number of trisrips have been studied in several recent papers. The “STRIPES” [8] system utilizes the popular earlier SGI greedy heuristic *tomesh* [1] as its local algorithm, while applying a global approach (“patchification”) to be able to handle very effectively models containing many quadrangular faces. The FTSG system [16] addressed some of the deficiencies of STRIPES (e.g., its speed and robustness) and often gives better results (smaller encodings) in practice. The FTSG methods are most closely related to that of Speckmann and Snoeyink [15], who implemented an algorithm specially designed for TIN (Triangulated Irregular Network) models, employing a unique traversal algorithm to find the spanning trees of the dual graphs of TIN models. [16] also proved that one can compute in $O(n)$ time a pure sequential trisrip encoding using at most $2n + 1$ vertices for an input triangulation having n triangles.¹

2. PRELIMINARIES

We are given a *polygonal surface model*, S , whose set of *polygonal faces* represent the boundary of a polyhedral object in 3-space. Each *polygonal face* is represented by a circular list of *vertices* that describe the outer boundary of the face, followed by a possibly empty list of *holes*. A polygonal face with no holes is *simply connected*, while a face with one or more holes is *multiply connected*. Each polygonal face is usually expected to lie in a plane and each has an associated outward surface normal; however, we have observed that real-world data will often have non-coplanar vertices.

From S we can obtain a *triangulated model* (or simply *triangulation*), \mathcal{T} , all of whose faces are triangles. (In practice, this can be efficiently and robustly accomplished with the FIST system of Held [12].) In some cases, we allow \mathcal{T} to be an arbitrary cell complex (any two triangles are either disjoint or they intersect exactly in a common edge or a common vertex); however, unless stated otherwise, we assume that

¹Note that even if a triangulation on v vertices can be encoded with a single trisrip (one vertex per triangle), it will require roughly $2v$ vertices in the encoding, assuming that every triangle has 3 neighboring triangles, since there are $2v - 4$ triangles in a triangulated planar graph on v vertices.

\mathcal{T} is a manifold surface, with each edge incident to at most two triangles.

We assume that \mathcal{T} is connected; if it is not, then our methods can be applied to each component separately.

We let $\mathcal{G} = (\mathcal{T}, E)$ denote the dual graph of the (manifold) triangulation \mathcal{T} , with the set of nodes given by the set \mathcal{T} of triangles of \mathcal{T} and the edges E linking two triangles that share a common edge. (If \mathcal{T} is not manifold, we “split” each non-manifold edge, pairing up its incident triangles so that each (new) instance of the edge is bounding either one or two triangles.)

A sequential² *tristrip* is a triangulation \mathcal{T} for which the dual graph \mathcal{G} has a Hamiltonian path such that no three consecutive edges crossed by the path are incident on a common vertex. Equivalently, \mathcal{T} is a tristrip if there exists a vertex sequence, $\sigma = (v_1, v_2, \dots, v_{n+2})$, possibly with repetitions, such that the n triangles of \mathcal{T} are exactly given by the triples of consecutive vertices in the sequence: $(v_1, v_2, v_3), (v_2, v_3, v_4), \dots, (v_n, v_{n+1}, v_{n+2})$.

We are interested in partitioning \mathcal{T} into a minimum number, k^* , of tristrips, $\{\sigma_1, \dots, \sigma_{k^*}\}$, which *encode* \mathcal{T} . The k -STRIPABILITY problem is that of deciding if \mathcal{T} can be partitioned into k tristrips. It can be thought of as the problem of deciding if the dual graph (which is a planar 3-regular graph in the case of a triangulation homeomorphic to a sphere) can be partitioned into k “alternating” paths (that turn left, then right, then left, etc).

3. HARDNESS

We prove that k -STRIPABILITY is NP-complete, even for fully triangulated genus-zero models. We outline the proof of the following theorem; details are omitted from this abstract.

THEOREM 1. *Given a triangulated surface model \mathcal{T} homeomorphic to a sphere, it is NP-complete to decide if \mathcal{T} can be decomposed into k tristrips.*

It is clear that k -STRIPABILITY is in NP, since any set of k tristrips can be readily checked in polynomial time to verify that it is a valid decomposition of the input.

To prove NP-hardness, we use a reduction from planar MAX 2SAT (proved NP-complete in [10]):

Instance : *A set of variables \mathcal{V} and a set of disjunctive clauses \mathcal{C} , each containing at most two literals, and an integer $N \geq 0$. The bipartite graph $G = (\mathcal{C} \cup \mathcal{V}, E)$ is planar, with edges E linking a vertex $v \in \mathcal{V}$ to a vertex $c \in \mathcal{C}$ if and only if either v or its negation, \bar{v} , is in clause c .*

Question : *Is there a truth assignment for \mathcal{V} that satisfies at least N clauses in \mathcal{C} .*

Given an arbitrary instance I of planar MAX 2SAT we produce an instance I' of k -STRIPABILITY such that there is a stripification for I' having at most N' tristrips if and only if there is a truth assignment for I that satisfies at least N clauses. It is easy to argue that we may assume, with no loss of generality, that no clause in I contains both v and \bar{v} for any variable v .

We let \mathcal{V} denote the set of variables, and K the number of clauses in I . Suppose G is the graph associated with I . If $c \in \mathcal{C}$ is a binary clause node in G , we replace c and the

²We focus on pure sequential tristrips, not on “fan” tristrips and sequential tristrips with “swaps” (see, e.g., [16]).

two edges incident on c by one edge. We call this a *binary edge*. We call the edge incident to a unary clause a *unary edge*. Let the resulting graph be denoted G' and embed G' , with straight edges, on an integer grid, $O(|\mathcal{V}| + K)$ -by- $O(|\mathcal{V}| + K)$. It is known (e.g. [14]) that this can be done while guaranteeing a minimum separation distance of $\delta \geq 1/\sqrt{2}$ between any embedded node and any embedded edge.

At each variable node of G , we center a square of sufficiently small side length ℓ ($\ell < \delta/10$ suffices) and at each unary clause node we temporarily center a *terminating square* having side length $\Theta(\ell/K)$. The terminating squares are only used in the construction and are not part of I' .

Variable Gadgets. For each variable, we construct a triangulation of the corresponding square, S , as shown in Figure 2. We first partition S into $m = 16K$ sectors so that each sector intersects ∂S in a segment of length $4\ell/m$. We place a regular convex polygon, P , at the center of S and $r = 32K^2$ polygonal “rings” radiating outward from P . The sectors partition each ring into m trapezoids. We triangulate each trapezoid in all rings, except the innermost ring, in a consistent manner, as shown in the figure. Also, within each sector, the area between the square boundary and the outermost ring forms a convex quadrilateral. We triangulate it in a similar manner. Each trapezoid in the innermost ring is triangulated via a *stopping gadget* as shown on the right in Figure 2. The polygon P is triangulated as shown; P , along with its triangulation, is called a *center gadget*.

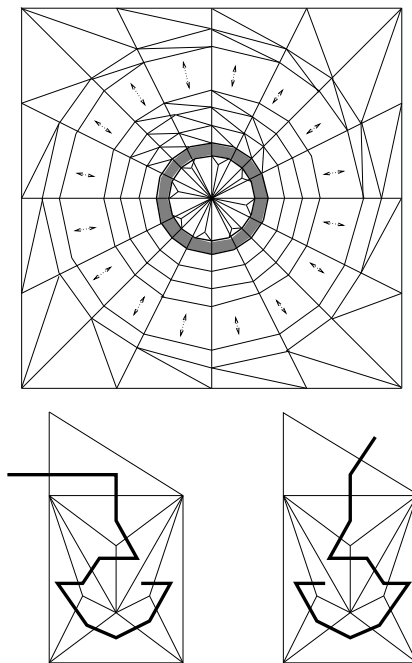


Figure 2: A variable gadget (top); each shaded trapezoid contains a stopping gadget (bottom), shown with each of its two coverings by a single tristrip.

Clause Gadgets. Let e be an edge in G' . If e is a binary edge then e connects two variable gadgets. We define a *tube* corresponding to e , which is a triangulated polygon sharing one or two edges with each of the corresponding variable squares; it will share one edge with a sector bounding edge of a variable square if that variable appears un-negated and will share two edges if the variable appears negated in the

clause. The left and right sides of a tube consist of either a single edge or a pair of edges, and the interior of the tube is triangulated according to the case. See Figure 3. When multiple tubes are incident on one side of a variable gadget, we space them out, leaving at least a certain number (2 suffices) of sector bounding edges between any two tubes. The tube corresponding to a unary clause is similar, except that it links one or two sectors of a variable gadget with a terminating square; see Figure 4.

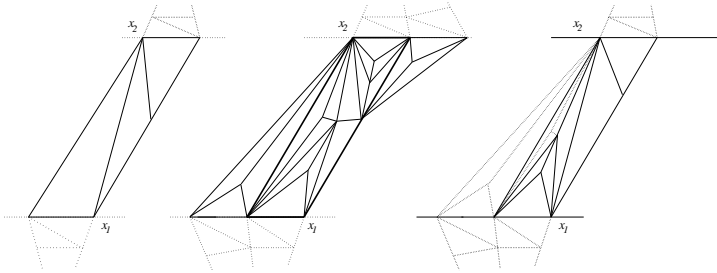


Figure 3: Clause gadgets: $x_1 \vee x_2$, $\bar{x}_1 \vee \bar{x}_2$, $\bar{x}_1 \vee x_2$.

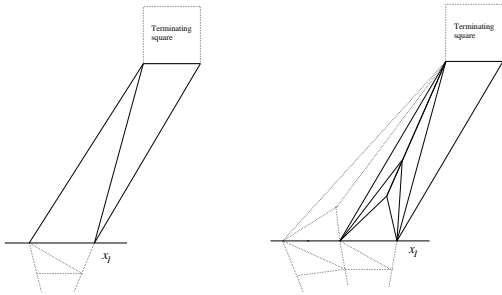


Figure 4: Unary clause gadgets: x_1 , \bar{x}_1 .

Fill In. We triangulate each face F in the embedding of G' by using a nested sequence of $64K|\mathcal{V}|^2$ *tristrip rings*, each beginning and ending with a stopping gadget (placed against a variable gadget sector bounding edge), and a center gadget; see Figure 5. The center gadget is made to have the same number of sides as the face F ; each of the $\leq 16K|\mathcal{V}|$ edges of F is mapped to an edge of the center gadget, with the endpoints specified at points along a set of disjoint polygonal curves (shown dashed in the figure), each of which has at most $4|\mathcal{V}|$ bends.

Optimal Stripification of the Construction. There are two optimal stripifications for a variable gadget, one corresponding to TRUE, one corresponding to FALSE. In each of these two stripifications, for each sector there is a *sector strip*, s , that starts in the stopping gadget contained in the sector as shown in Figure 2. This strip exits the top of the stopping gadget and turns either left or right depending on its orientation in the stopping gadget. If s turns right, it travels through the containing sector. If the sector meets a tube, s then enters the center triangle in the tube or the side triangle and turns right. In this case, we call s a *TRUE strip*. If s turns left upon exiting the stopping gadget, it travels around the variable gadget in a spiral until reaching the last triangle in the sector as shown in Figure 6(left). If the sector meets a tube or side triangle, s then enters the center triangle in the tube or the side triangle and turns left. In this case, s is a *FALSE strip*. In either case, if the segment does not meet a tube or switch, s ends in the last triangle

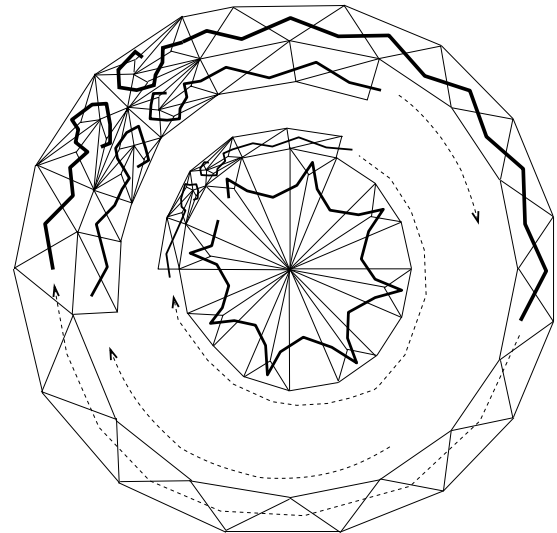
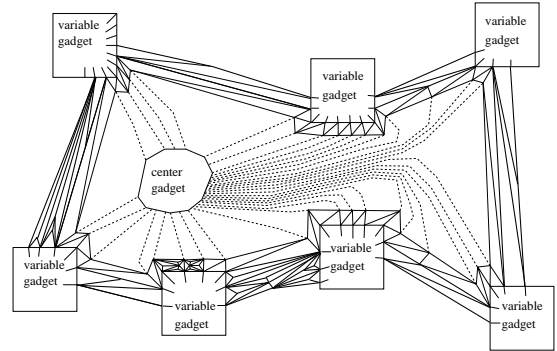
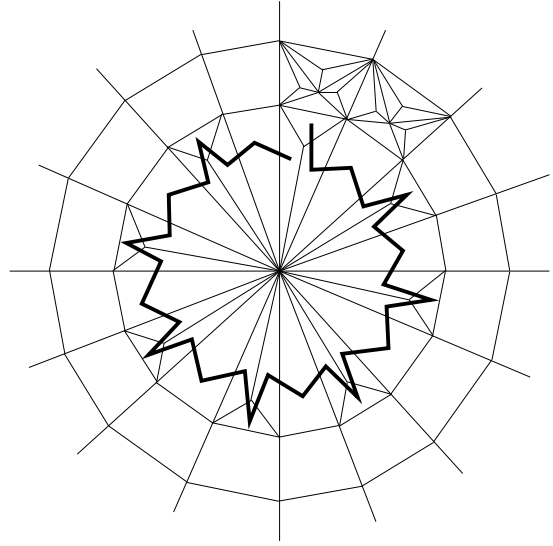


Figure 5: Top: A center gadget along with stopping gadgets and the tristrip covering the gadget in an optimal stripification. Middle: Filling in a face with tristrip rings; binary tubes, representing clause gadgets, are shown highlighted. Bottom: A schematic of the tristrip rings, each beginning and ending with stopping gadgets, surrounding a center gadget.

of the segment and does not exit the variable gadget (if not, problems are created as we will see later). All segment strips for a given variable must be consistent, since, if some strip is TRUE and some number of strips are FALSE, then the strips together are broken into at least $2K = (32K^2/16K)$ pieces. To see this, suppose some strip s_1 is TRUE and let S_F be the set of all FALSE strips in the variable. If Σ is a sector and f is a FALSE strip then f spirals around to intersect Σ $2K$ times. Each time Σ intersects f , it covers two adjacent triangles in Σ . This set of $4K$ triangles is uniquely determined and consists of $2K$ connected components for each FALSE strip. In particular, if s_1 is contained in the sector Σ , and T_F is the set of triangles covered by S_F in Σ , then there are at least $2K$ connected components of $\Sigma \setminus T_F$ that are not covered by S_F . If s_1 travels unbroken between two adjacent components, then all strips crossing Σ between the components must be broken; otherwise, s_1 must be broken between the components. Thus, the strips in $S_F \cup \{s_1\}$ are broken into at least $2K$ pieces. A variable gadget in which all strips are TRUE (resp., FALSE) corresponds to a TRUE (resp., FALSE) variable. In either case, there is one strip to cover the center gadget, as shown in Figure 5(top).

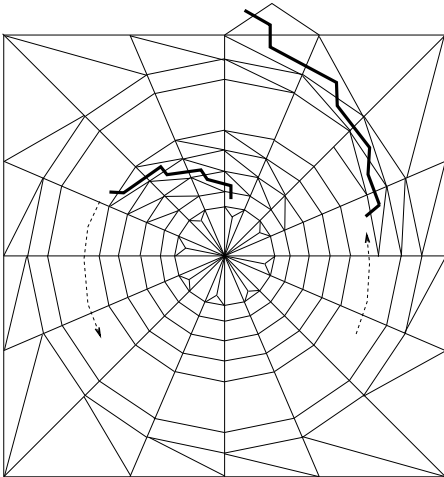


Figure 6: A portion of a FALSE segment strip.

The optimal stripification of clauses depends on a number of cases; we defer the enumeration to the full paper, where we present an enumeration of the possible optimal coverings of clause gadgets. The binary tube in a binary clause gadget is incident on two sector boundaries, one in each of the corresponding variable gadgets. For each of these sectors, there is a strip that exits the top of the sector and enters the incident triangle in the tube. This strip is TRUE or FALSE depending on the truth assignment of the variable. If the clause is satisfied, then the two sector strips are enough to cover the clause gadget. If the clause is not satisfied then one additional strip is needed. We have a similar case for unary clause gadgets except that there is one strip rather than two. If the clause is satisfied then the one strip is enough to cover the clause gadget and if the clause is not satisfied then one additional strip is needed. Thus, satisfied clauses are covered by the strips that cover the variable gadgets, while unsatisfied clauses require additional strips.

Finally, in an optimal stripification, each face of G' is covered by $64K|\mathcal{V}|^2 + 1$ tristrrips (one per ring, plus one for the center gadget).

Completing the Proof (Sketch). We claim that there exists a truth assignment for I such that at least N clauses are satisfied if and only if there is a stripification using at most $|\mathcal{V}| + 16K|\mathcal{V}| + \phi(64K|\mathcal{V}|^2 + 1) + (K - N)$ tristrrips, where ϕ is the number of faces of planar graph G' .

First, suppose there is a truth assignment such that at least N clauses are satisfied. Then, there is a stripification containing at most $|\mathcal{V}| + 16K|\mathcal{V}| + \phi(64K|\mathcal{V}|^2 + 1) + (K - N)$ strips. $|\mathcal{V}| + 16K|\mathcal{V}|$ of these strips cover the $|\mathcal{V}|$ variable gadgets, $\phi(64K|\mathcal{V}|^2 + 1)$ strips cover the faces of G' , and $K - N$ strips cover the portions of clause gadgets that are not covered by the $|\mathcal{V}| + 16K|\mathcal{V}|$ strips that cover variable gadgets.

Now suppose we have a stripification containing at most $|\mathcal{V}| + 16K|\mathcal{V}| + \phi(64K|\mathcal{V}|^2 + 1) + (K - N)$ strips. First, we claim that we need at least $|\mathcal{V}| + 16K|\mathcal{V}| + \phi(64K|\mathcal{V}|^2 + 1)$ strips to cover variable gadgets and faces. We now have $K - N$ additional strips to finish covering clause gadgets. Suppose one of these strips, s , enters more than one clause gadget. Then s must either cut across a filled in face or a variable gadget. Any strip entering a variable gadget travels towards the center to become trapped in a stopping gadget. A strip entering a filled in face zig-zags in to the center gadget, where it turns around and goes out, following next to the portion of the strip that went in to the center. In doing so, though, the strip cuts across the $64K|\mathcal{V}|^2$ ring tristrrips, resulting in an additional $64K|\mathcal{V}|^2$ strips required to cover. (The proof uses the fact that each stopping gadget has a vertex of degree 7, which, by Lemma 4 (below), implies the existence of the end triangle of a strip incident to it.) Thus, a strip should not enter a filled in face in a covering that uses $|\mathcal{V}| + 16K|\mathcal{V}| + \phi(64K|\mathcal{V}|^2 + 1) + (K - N)$ strips.

4. OPTIMAL STRIPIFICATION

We have shown that the optimal stripification problem is a hard combinatorial optimization problem. We now present algorithms to solve the problem exactly, based on integer programming techniques and branch-and-bound. Of course, these algorithms have a worst-case exponential running time; we perform an experimental study of the algorithms to determine their effectiveness in practice.

4.1 Integer Programming Algorithm

We formulate the stripification problem as a 0-1 Integer Program (IP). We assume here that the input surface triangulation, \mathcal{T} , is 2-manifold, so that each edge in \mathcal{T} belongs to at most two triangles. (We omit in this abstract our extensions to the non-manifold case, which is part of the implementation.)

An edge is said to be *internal* if it is shared by exactly two triangles; otherwise, it is a *boundary* edge. We let $\mathcal{G} = (\mathcal{V}(\mathcal{G}), \mathcal{E}(\mathcal{G}))$ denote the dual graph of \mathcal{T} . A (sequential) stripification of \mathcal{T} corresponds to an exact covering of the nodes of \mathcal{G} by a set of sequential paths in \mathcal{G} . Thus, a stripification corresponds to a subgraph, \mathcal{G}' , of \mathcal{G} , with $\mathcal{V}(\mathcal{G}') = \mathcal{V}(\mathcal{G})$ and $\mathcal{E}(\mathcal{G}') \subset \mathcal{E}(\mathcal{G})$. We define a 0-1 integer variable x_e for each edge e in \mathcal{G} such that

$$x_e = \begin{cases} 0 & \text{if } e \in \mathcal{E}(\mathcal{G}') \\ 1 & \text{otherwise} \end{cases}$$

For any valid stripification, \mathcal{S} , we define the objective function $C(\mathcal{S}) = \sum_{e \in \mathcal{E}(\mathcal{G})} x_e$, which is simply the cardinality of $\mathcal{E}(\mathcal{G}) \setminus \mathcal{E}(\mathcal{G}')$.

LEMMA 1. An optimal stripification, \mathcal{S} , of \mathcal{T} minimizes the objective function $C(\mathcal{S})$.

PROOF. Let n be the number of triangles in \mathcal{T} , b be the number of boundary edges, and $m(\mathcal{S})$ be the number of strips in stripification \mathcal{S} . For a strip $s \in \mathcal{S}$, let $n(s)$ be the number of triangles in the strip. The number of edges (with possible repetitions) bounding the region defined by s is $n(s) + 2$. Summing over all strips, we have

$$\sum_{s \in \mathcal{S}} (n(s) + 2) = n + 2m(\mathcal{S}).$$

In this summation, each counted edge that is internal is counted twice and each that is boundary is counted once. Further, the counted internal edges have a one-to-one correspondence with the edges in $\mathcal{E}(\mathcal{G}) \setminus \mathcal{E}(\mathcal{G}')$. Thus, $\sum_{s \in \mathcal{S}} (n(s) + 2) = 2C(\mathcal{S}) + b$, so that $2C(\mathcal{S}) + b = n + 2m(\mathcal{S})$, and $C(\mathcal{S}) = m(\mathcal{S}) + (n - b)/2 = m(\mathcal{S}) + \text{constant}$.

By Lemma 1, our IP problem is to minimize $\sum_{e \in \mathcal{E}(\mathcal{G})} x_e$, subject to the x_e 's defining a valid stripification, \mathcal{S} , of \mathcal{T} . We now show how to define a small set of appropriate linear constraints on the x_e 's. The constraints can be categorized into local and global ones.

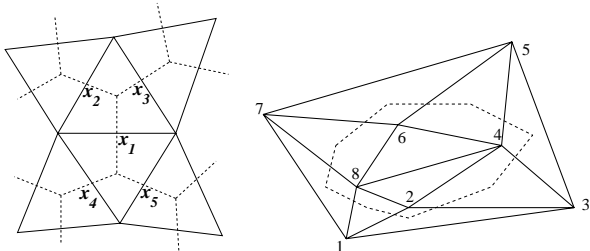


Figure 7: Left: A local configuration of a triangle mesh and its dual. Right: A sequential cycle, whose encoding is $(1, 2, 3, 4, 5, 6, 7, 8, 1, 2, \dots)$.

The local constraints enforce that the dual of a tristrip is a path. Since $x_e = 1$ indicates that edge e is *not* selected for a path, we require that the sum of the three variables corresponding to dual edges incident on a common node (triangle) sum to at least one. Referring to Figure 7 (left), these constraints say that $x_1 + x_2 + x_3 \geq 1$, and $x_1 + x_4 + x_5 \geq 1$. The local constraints must also enforce that the dual path “alternates” its turns left, then right, then left, etc. For the example in the figure, this leads to the constraints that $x_2 + x_1 + x_4 \geq 1$ and $x_3 + x_1 + x_5 \geq 1$.

The global constraints are necessary to ensure that any sequential strip has two “ends” and does not cycle; see Figure 7 (right). For each self-looping sequential strip (sequential cycle), with variables x_1, x_2, \dots, x_k corresponding to the dual edges, we impose the constraint $x_1 + x_2 + \dots + x_k \geq 1$ to ensure that at least one edge is *not* selected. (In the full paper, we give examples to show that without the global constraints, the IP solution leads to a suboptimal stripification.)

LEMMA 2. The IP formulation can be derived from \mathcal{T} in $O(n)$ time and consists of $\Theta(n)$ linear constraints, with a total of $\Theta(n)$ occurrences of variables in constraints, where n is the number of triangles in \mathcal{T} .

PROOF. (Sketch) The number of local constraints is at most 2 times the number of edges, plus the number of vertices in \mathcal{G} . They are readily found in linear time.

Because of the alternating nature of sequential cycles, each edge can appear in at most two sequential cycles, implying that there are $O(n)$ global constraints. (Examples exist with $\Omega(n)$ global constraints.) We can identify them in linear time by “stripping” them off one by one.

Example. Our software produces the IP formulation in MPS format for use in CPLEX, a dedicated commercial optimization code from ILOG [6]. For the example in Figure 8, it produces an IP formulation that includes 24 local constraints and one global constraint. CPLEX solves this IP in 0.13 seconds on our low-end workstation, resulting in the two strips shown in the figure.

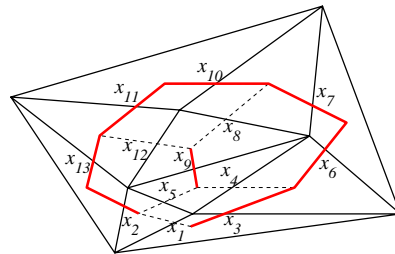


Figure 8: Example triangulation with its optimal decomposition into two tristrips (dual paths highlighted).

Comparison with Implicit Enumeration. Implicit enumeration is a commonly used branch-and-bound method that solves 0-1 IPs. (Details of the method appear in the full paper.) We implemented this method (using a depth-first-search order in the enumeration tree, in order to save on memory) to serve as a benchmark and to validate our codes. On a test suite of 10 randomly generated “grid” models, each with 32 triangles, the CPLEX-based solution based on our IP formulation is roughly 100 times faster (average of 0.77 seconds, versus 78.62 seconds).

4.2 Branch and Bound

We have devised a custom-tailored branch-and-bound algorithm for our optimization problem. The algorithm performs a limited enumeration of the possible stripifications, while utilizing efficient and effective *fathoming rules* in order to prune branches of the enumeration tree. At any given stage of the algorithm, the current candidate (partial) solution has covered some portion of \mathcal{T} with a set of (disjoint) tristrips, and we must consider which candidate tristrips should be used to cover the remaining n_0 triangles, \mathcal{T}_0 of \mathcal{T} . First, we note the following fact (proof omitted here):

LEMMA 3. There are $O(n_0^2)$ distinct tristrips passing through any given triangle in the remaining mesh, \mathcal{T}_0 .

There are many possible choices in designing how the algorithm enumerates the candidate tristrips. In our implementation, we adopt a method that tends to process the longer strips first. For each of the three ways in which a strip can be extended, we start one from the base triangle in one of the two directions, say, the “left” direction, and extend the strip until we encounter either the boundary of \mathcal{T} or the strip itself. This extremal “left hand” strip can

then be shortened one triangle at a time, as we consider alternatives. For each fixed “left hand” strip, we can extend it in the other direction from the base triangle until it can no longer be grown. Then this “right hand” strip can be shortened one triangle at a time. In this way, we enumerate all sequential strips that pass through a given triangle. For each such strip, we add it to the partial solution. If \mathcal{T} is covered, we have reached a solution; otherwise, we continue the enumeration. This enumeration procedure can be seen as traversing a branch-and-bound (B&B) tree of $O(n)$ height, each node of which has degree $O(n^2)$.

We now describe our fathoming rules, which are of critical importance to the efficiency of our B&B algorithm. Consider a vertex, v , of the triangle mesh. It has many triangles incident on it. Some have been covered already in the partial solution, while others are uncovered so far. The uncovered triangles, in general, form a set of “sectors” around v . Let us say they have cardinalities r_1, r_2, \dots .

LEMMA 4. *Let τ be a maximal subsequence of triangles in a trisrip such that each triangle in τ is incident on a vertex v . Then, τ consists of at most three triangles; if τ consists of one or two triangles, then at least one of these triangles must be an end triangle of the trisrip.*

PROOF. The proof is based on a simple local investigation of the cases in the neighborhood of v .

Thus, each “sector” on v will contribute to the “end triangles” of at least one sequential strip if its cardinality is not $0 \pmod 3$. Let $s_i = 0$ if $r_i = 0 \pmod 3$, and $s_i = 1$ otherwise.

LEMMA 5. *In any completion of the partial stripification, there must be at least $\sum_i s_i$ trisrips that have an end triangle incident on v , with v incident on at most 2 of the triangles in the strip.*

Consider a trisrip. It has two “end triangles” (only one if it is a trivial strip of only 1 triangle). Consider the vertices incident on these two end triangles. In general, there are six such vertices; however, we claim that at most 4 of them are vertices that are incident on *at most* two of the triangles of the strip. (In the case of a 1-triangle strip, the total number of such vertices is simply 3.)

LEMMA 6. *At most 4 vertices of a trisrip are incident on ≤ 2 triangles of the strip.*

We consider v to receive a “charge” from a strip s if v is incident on at most 2 of the triangles in s . (Necessarily these triangles will be end triangles of s .) Then, we have shown that v is charged at least an amount $C_v = \sum_i s_i$, so the total amount of charges over all vertices is at least $\sum_v C_v$. On the other hand, each strip can result in at most 4 charges, so the total charge is at most $4n_s$, where n_s is the number of strips in a completion of the partial stripification. Thus, we have the lower bound $n_s \geq (1/4) \sum_v C_v$, which we use as the basis of our fathoming rule. More specifically, if we keep track of the quantities C_v , for each vertex v , as we proceed, then whenever we get to a “node” in the B&B tree where $(1/4) \sum_v C_v$, plus the current number of strips (the ones that are already part of the partial solution), is no less than the best (complete) stripification found so far, we know that we do not have to explore the B&B tree below this

node – since the number of strips necessary to complete the covering is at least $(1/4) \sum_v C_v$, we are doomed to obtain a result that is no better than the best solution found so far.

Implementation Issues. In order to get an initial feasible (hopefully close to optimal) solution, we utilize the **FTSG** package (with the dynamic programming option on). Also, a key issue in implementing the algorithm is to update efficiently the strip lower bound when visiting new nodes. From the analysis above, one can update the lower bound “locally” by recalculating C_v for those vertices whose incident faces have changed. The computing time at each of those vertices is expected to be constant. Note that the new lower bound is established as if the remaining triangle mesh is still one connected component. It is certainly better to compute lower bounds componentwise and sum them; however, we have found in practice that the overhead for maintaining connected components offsets the benefit.

There are cases for which the above bounding scheme results in trivial lower bounds (0 or 1); e.g., see Figure 9, where the fathoming rule gives a lower bound of 1, while it actually requires 3 trisrips to cover the example. To improve the lower bound, we implemented a “one-strip” test (in linear time [2]) to determine whether a single strip can indeed cover a component. If so, no further fathoming is needed on that component; otherwise, we have established a better lower bound (of 2 strips).

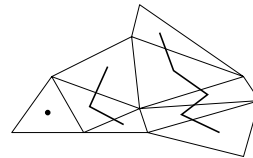


Figure 9: An example for which our fathoming rule (with lower bound of 1) is not very effective but can be improved with the “one-strip” test.

4.3 Experiments

We have conducted experiments using three main classes of data: randomly generated “grid” models, randomly generated “sphere” models, “real-world” models (extracting small patches, if the model is too large to run in a reasonable time). The “grid” models are generated by randomly triangulating each square (by randomly picking one of the two diagonals) in an $n \times n$ regular grid. To build “sphere” models, we first use the **sphere** program of O’Rourke and Xu [13] to generate random points on or near a sphere, and then apply the **qhull** program [3] of Barber and Huhdanpaa to obtain the triangulated convex hulls.

CPLEX vs. B&B. The following comparisons were done on a Sun4c with 37 MB memory running SunOS 4.1.3-U1.

In testing the branch-and-bound method on the same 10 “grid” models used in our base comparison with implicit enumeration, we obtained an average running time of 0.94 seconds, which is 22% slower than using **CPLEX** in the IP formulation, but still two orders of magnitude faster than using implicit enumeration.

We also ran tests on some larger models: 10 “grid” ones and 10 “sphere” ones, each containing 72 triangles, verifying that the **CPLEX** and B&B methods generate the same optimal number of strips. The average running times on the two sets of models are listed in Table 1. (In very recent tests,

No. of triangles	69	49	37	55	61	72	75	58	
No. of constraints	124	122	86	65	65	189	182	131	
Running Time (s)	CPLEX	0.83	0.32	0.82	0.25	0.12	3.42	1.45	0.35
	B&B	1.06	48.60	0.69	0.18	0.06	0.86	67.22	0.75

Table 2: The running times using both CPLEX and the B&B method on some small triangle meshes.

Method	Time(s)	
	Grid	Sphere
B&B	4182	901
CPLEX	57	2625

Table 1: The average running times using both CPLEX and the branch-and-bound method on 10 “grid” models and 10 “sphere” models, each containing 72 triangles.

on another platform (Sun Ultra-30 with 512MB memory, running SunOS 5.6) with the latest (7.0) version of CPLEX, we obtained similar results, with the B&B method just under twice as fast as CPLEX in solving the sphere models, and about 100 times slower at solving the grid models.)

It is interesting to notice that our B&B method runs faster on “sphere” models than on “grid” models, and CPLEX behaves contrarily. Indeed, the B&B method is roughly 2-3 times faster than CPLEX on “sphere” models. However, it is 70-100 times slower on “grid” models. This may be partly because the IP formulations of these “sphere” models contain 39% more constraints than that of “grid” models on average (286 vs. 205), and partly because fewer strips are required to cover a “sphere” model than a “grid” model (Table 3), which probably implies a smaller B&B tree for a “sphere” model than for a “grid” model.

We also ran both methods on some small triangle meshes obtained from “real-world” models, by grouping short strips (obtained using FTSG) into connected components (we say more about this later). Table 2 shows the running times. Both methods seem to run faster on these small meshes. However, the B&B method exhibits more variation in running time. We will investigate this issue later.

Grid			Sphere		
OPT	FTSG	%	OPT	FTSG	%
12	14	16.7	7	11	57.1
11	15	36.4	9	11	22.2
11	13	18.2	9	12	33.3
13	14	7.7	7	10	42.9
12	13	8.3	8	10	25
12	15	25	7	11	57.1
12	14	16.7	8	12	50
11	14	27.3	8	9	12.5
12	15	25	7	12	71.4
13	14	7.7	7	12	71.4

Table 3: The optimal number of tristrrips vs. the number obtained by FTSG on 10 “grid” models and 10 “sphere” models, each containing 72 triangles.

Comparisons with FTSG. How well are the heuristic methods used in practice doing in comparison with an optimal stripification? We compare the number of tristrrips in the optimal solution with the number obtained using a

leading package (FTSG, run with the options “-dfs -alt -DP”, which is generally the best choice for minimizing the number of tristrrips). The following experiments were run on a Sun sparc Ultra-30, running SunOS 5.6 with 512MB. Table 3 lists the number of sequential strips in an optimal solution and in the FTSG solution on the 10 “grid” models and 10 “sphere” models. Also listed in the table are the percentages by which FTSG increases the number of strips over the optimal.

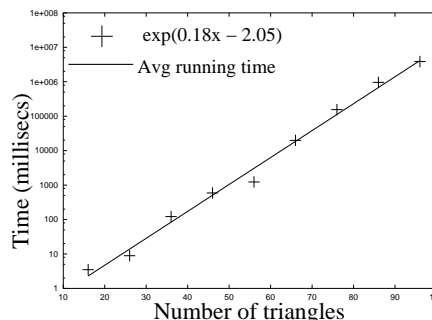


Figure 10: The running time of B&B grows exponentially with input size.

More Tests of the B&B. To see how the performance of our B&B algorithm scales with input size, we generated 9 groups of random “sphere” models, each with 10 models of the same size, ranging from 16 to 96. The average running time for each group is plotted (log-scale) in Figure 10. A least-squares fit of the data shows the exponential growth of $y = \exp(0.18x - 2.05)$. The predicted running time on a sphere model of 100 triangles is about 2.5 hours; for comparison, FTSG takes only a few milliseconds on such a model.

We also show in Figure 11 how the B&B method generates progressively fewer strips as the algorithm runs (on a “sphere” model of 96 triangles). The B&B method starts with a stripification obtained by FTSG, containing 15 strips, and eventually finds an optimal solution with 11 strips.

For practical problems, one may wish to spend a reasonable amount of time running the B&B method and then stop the algorithm within a specified time frame, reporting the best solution found so far. We investigate this strategy by running the following experiments on a full suite of 173 real-world models (the same set used in [16] for FTSG tests). For each model, we run the B&B method for a time equal to k times the FTSG running time, where $k = 2, 4, 8, \dots, 1024$, then stop and record the number of tristrrips. The results show that for $k = 1024$, we obtain improvements over FTSG on 10% (19) of the models. Table 4 shows how the number of strips changes with k for a few models. The column for $k = 0$ shows the number of strips obtained by FTSG.

Efficacy of Fathoming Rule. We have seen that the running time of the B&B algorithm is exponential. We conducted tests that show, though, that it is dramatically better

Model	Verts	Tris	k										
			0	2	4	8	16	32	64	128	256	512	1024
compdesk	136	204	51	50	49	48	46	43	40	35	34 (opt)		
doors	280	488	54	54	54	54	54	53	53	53	52	51	51
extrude	64	124	8	8	7	7	7	7	7	7	7	7	7
pilot	586	1012	65	55	55	55	55	55	55	55	54	54	54
roadarm	72	132	8	8	8	7	7	7	6 (opt)				
tool-1.2	1052	2096	5	5	5	5	5	5	5	4	3 (opt)		

Table 4: Number of strips using truncated B&B, stopping after k times the FSTG runtime.

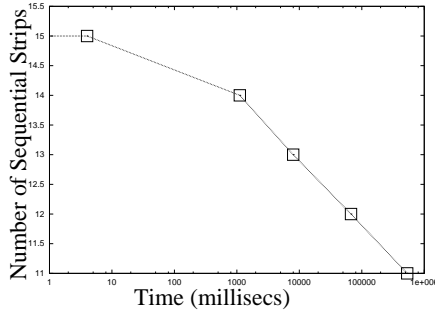


Figure 11: The number of sequential strips obtained by the branch-and-bound method decreases as running time increases.

than a “brute force” enumeration, showing that our fathoming rule is highly effective in pruning candidates; see Table 5.

Model	Running Time(ms)		Nodes visited	
	B&B	Brute Force	B&B	Brute Force
1	17	3916	1126	2006216
2	87	7093	6246	3401612
3	33	595	2486	307933
4	7	754	388	402350
5	38	2143	2838	1081780
6	37	1361	2716	698584
7	36	1738	2587	884116
8	7	440	462	227251
9	8	967	475	562687
10	46	1506	3358	716184

Table 5: Running time and number of nodes visited by B&B and brute-force method on 10 “sphere” models, each with 30 triangles.

In the experiments reported so far, we only applied the “local” fathoming rule, and did not perform it componentwise. In the full paper, we report on experiments that show the effect of applying the fathoming rule componentwise and also the effect of using the one-strip test to improve the lower bound. While in some cases there was a noticeable improvement, the results were not dramatic, so the extra time spent in performing the more advanced fathoming rules was not determined to be worth it in most cases.

5. PATCH OPTIMIZATION IN FTSG

Since the running times of our optimization algorithms grow exponentially with the model size, they can only be applied to relatively small models (fewer than 100 triangles). We have developed a method of *patch optimization* to allow

to trade off speed for optimality within the practical FTSG system.

We outline the method here. First, we use the “path-peeling” algorithm of FTSG to obtain an initial set of tristrrips. Then, according to a pre-specified cut-off size, k , we categorize the strips into “short” and “long” strips. The long strips are kept as is, but the short strips are regrouped into connected components (“patches”). In the grouping of short strips, we do not allow patches to become larger (in number of triangles) than a user-specified threshold, K . Then, each patch is fed into the optimization algorithm (using CPLEX or B&B) to compute its optimal stripification. As K increases, the method allows one to get closer and closer to an optimal solution.

A crucial part in this algorithm is the grouping of short strips into patches of bounded size. We have experimented with two main methods for doing this, one based on a simple BFS algorithm (adding adjacent small strips until the threshold K is reached) and one based on a prioritized approach, in which we give preference to adding shorter strips during the patch growing.

Experiments. We ran experiments on the suite of 173 real-world polygonal models used in [16]. In using FTSG, the strip concatenation option was disabled so as to focus exclusively on the effectiveness of the *patch optimization* algorithm.

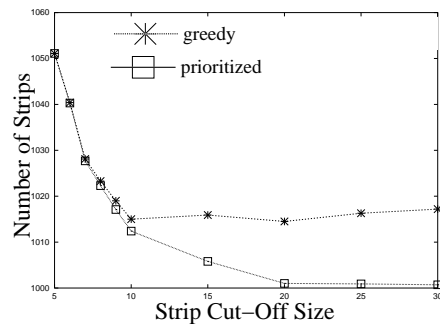


Figure 12: The average number of tristrrips vs. the strip cut-off size k .

We first used a patch size bound of $K = 30$ and measured the number of tristrrips obtained using various strip cut-off sizes, k . Both the greedy patching method and the prioritized method are tested and compared. Figure 12 shows how the number of strips varies with the strip cut-off size, k , for the two different patch growing methods. It clearly indicates that the prioritized method has the advantage over the greedy method. For the former method, the average number of strips decreases monotonically as the strip cut-off size increases. When the cut-off size reaches 30, we obtain the

best average number of 1000.7, which is 6.7% fewer than that obtained by FTSG alone. Note that there is virtually no discernible improvement for $k > 20$.

However, for the prioritized method, the average running time increases from 264 to 420 seconds per million triangles when the strip cut-off size increases from 5 to 30. This can be expected, since more strips are processed. Nevertheless, the best and worst average running times are still of the same order. This makes it reasonable to set $k = K$ in the future experiments.

Patch size	VPT	Strips	Time(s)
FTSG	1.2487	1072.7	14.3
20	1.2339	1019.2	255.7
25	1.2316	1009.3	274.7
30	1.2299	1000.7	419.6
35	1.2282	994.3	1115.8
40	1.2273	988.9	8402.5
45	1.2262	984.0	69777.1

Table 6: The performance of the patch optimization using different patch size bounds.

Next, we study how the performance of our algorithm varies with the patch size bound, K , for values ranging uniformly from 20 to 45. The first two columns in Table 6 list the average vertex per triangle ratio (VPT) and the average number of sequential strips, both of which decrease as the patch size increases. The second row in the table lists the results for the original FTSG. The third column in Table 6 lists the running time of each run in seconds per million triangles. We can see that, as the patch size bound increases, the running time can grow exponentially.

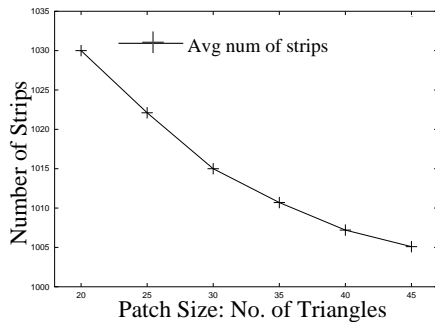


Figure 13: The number of strips decreases as the patch size bound increases.

6. CONCLUSION

We have developed the first exact optimization methods for optimizing the number of trisrips in a partitioning of a triangulated model. Our experiments compared the relative merits of two main approaches – using a commercial integer programming package and using a custom-tailored branch-and-bound algorithm. We have also integrated the optimization methods into the practical FTSG system in order to provide users a trade-off between running time and optimality.

On the theoretical front, we have established the complexity of the k -STRIPABILITY problem, showing that it is NP-complete and answering a question posed by J. Rossignac

in 1993. The most intriguing open problem is to provide any nontrivial approximation algorithm and/or to establish hardness of approximability results.

Acknowledgements. We thank the three anonymous referees for pointing out several corrections and making suggestions that improved the presentation of the paper. We thank Martin Held for research discussions, especially in the development of FTSG.

7. REFERENCES

- [1] K. Akeley, P. Haeberli, and D. Burns. **tomesh.c**: C Program on SGI Developer’s Toolbox CD, 1990.
- [2] E. M. Arkin, M. Held, J. S. B. Mitchell, and S. S. Skiena. Hamiltonian triangulations for fast rendering. *Visual Comput.*, 12(9):429–444, 1996.
- [3] C. B. Barber, H. Huhdanpaa. **qhull**: <http://www.geom.umn.edu/locate/qhull>.
- [4] R. Bar-Yehuda and C. Gotsman. Time/Space tradeoffs for polygon mesh rendering. *ACM Trans. Graph.*, 15(2):141–152, 1996.
- [5] M. M. Chow. Optimized Geometry Compression for Real-time Rendering. In *Proc. IEEE Visualization ’97*, pp. 347–354, 1997.
- [6] **CPLX**: <http://www.ilog.com/products/cplex/>.
- [7] M. Deering. Geometry compression. In *Proc. SIGGRAPH ’95*, ACM Computer Graphics Annual Conference Series, pp. 13–20, 1995.
- [8] F. Evans et al. **STRIFE**: <http://www.cs.sunysb.edu/~stripe/>.
- [9] F. Evans, S. S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *IEEE Visualization ’96 Proceedings*, pages 319–326, Oct. 1996.
- [10] L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *Internat. J. Comput. Geom. Appl* 3(4): 383–415, 1993.
- [11] S. Gumhold and W. Straßer. Real time compression of triangle mesh connectivity. In *Proc. SIGGRAPH ’98*, ACM Computer Graphics Annual Conference Series, pp. 133–140, 1998.
- [12] M. Held. Efficient and reliable triangulation of polygons. In *Proc. Comput. Graphics Internat. 1998*, pages 633–643, June 1998.
- [13] J. O’Rourke, M. Xu. **sphere.c**: <ftp://grendel.csc.smith.edu/pub/compgeom>.
- [14] W. Schnyder. Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Sympos. Discrete Algorithms*, pages 138–148, 1990.
- [15] B. Speckmann and J. Snoeyink. Easy triangle strips for TIN terrain models. In *Proc. 9th Canad. Conf. Comput. Geom.*, pages 239–244, 1997.
- [16] X. Xiang, M. Held and J. S. B. Mitchell. Fast and effective stripification of polygonal surface models. In *ACM Sympos. Interactive 3D Graphics*, 1999, pages 71–78, 224.