

An Algorithmic Approach to Plateau's Problem

Vladimir Stoyanov Tenev
Senior Honors Thesis
Department of Mathematics, Stanford University
Advisor: Lawrence Guth

June 5, 2008

Abstract

Plateau's problem, as first raised by Joseph Lagrange in 1760, is to show the existence of a minimal surface (a surface with everywhere zero mean curvature) with a given closed curve as its boundary. The problem is named after Joseph Plateau, whose experiments with soap films and wire frames motivated much theoretical interest in the topic.

The existence of a minimal surface having an arbitrary simple closed curve as its boundary was first proved by Jesse Douglas over 70 years ago by minimizing a quantity called the "Douglas integral." This work won Douglas the first-ever Fields medal in 1936 and led to much development in the field of geometric measure theory (GMT).

In this paper, we are dealing with least area surfaces, which are a type of minimal surface. Specifically, we introduce two algorithms that aim to produce a surface of least area given an arbitrary piecewise linear closed curve as the input. In the first part of the paper we present an algorithm that relies on picking favorable triangles to construct a triangulation of small total area, while in the second part we reduce the problem to two dimensions by considering the minimum-weight perfect matching of a set of nodes and suggest an algorithm based on this construction.

Contents

1	Introduction	2
2	Preliminary Definitions and Remarks	2
3	Algorithm 1	3
4	A New Approach	9
4.1	Introduction to Matchings	9
4.2	Alternating Trees and Perfect Matching	11
4.3	Perfect Matching in Bipartite Graphs	13
4.4	The Blossom Algorithm	13
4.5	Minimum-Weight Perfect Matching in Bipartite Graphs	16
5	Algorithm 2	19
5.1	A Few Basic Constructions	19
5.2	Simplicity and the Interface Plane	20
5.3	The Algorithm	25
5.4	Bipartite Matchings and Orientability	27
6	Conclusions and Future Research	28
7	Acknowledgements	28

1 Introduction

The motivating problem that we will try to tackle in this paper is the following:

Find a polynomial time algorithm that, when given any simple piecewise linear curve, generates a surface having that curve as a boundary and such that the ratio of the area of the surface to the area of the least area surface with that boundary is bounded by a constant.

This problem is related to Plateau's problem, which involves finding a minimal surface with a given simple closed curve as its boundary. Plateau's problem was raised by Joseph Lagrange in 1760 and was named after Joseph Plateau, whose experiments with soap films and wire frames motivated much theoretical interest in the topic.

The first algorithm we present is based on the idea that a reasonable way to create a surface with a piecewise linear closed curve as its boundary would be to create a set of non-overlapping triangles, such that each triangle shares one or two sides with the boundary and its remaining sides with other triangles. The area of the surface is then defined as the sum of the areas of the individual triangles.

2 Preliminary Definitions and Remarks

Here we present some convenient definitions which will be used throughout the paper.

Definition 2.1. *Given a planar polygon A , a **diagonal** is a line connecting two nonadjacent vertices that lies completely inside the polygon.*

Definition 2.2. *Given a planar polygon A , consider a vertex (and associated interior angle) α and an adjacent edge a with length $|a|$. The interior angle α is defined to be **concave** if for every $\epsilon > 0$, there exists a ray \hat{a} (colinear with a) directed toward the vertex such that $a \subset \hat{a}$, $|\hat{a} - a| < \epsilon$, and \hat{a} contains a point in the interior of A .*

Definition 2.3. *Given a planar polygon A with n vertices, a **triangulation** of A is a set of $n - 2$ nonoverlapping triangles that cover A and such that each vertex of each triangle in the set is a vertex of A .*

Definition 2.4. An *ear* of a polygon A is a set of three contiguous vertices α , β , and γ , with β and γ on opposite sides of α , such that the line $\beta\gamma$ is contained completely inside the polygon A .

Definition 2.5. Given a triangulation $T = \{E_1, \dots, E_n\}$, we define its boundary inductively as follows: $\partial T_1 = \partial E_1$, and $\partial T_{n+1} = \partial T_n \cup \partial E_{n+1} - \partial T_n \cap \partial E_{n+1}$.

3 Algorithm 1

Notice that a piecewise linear boundary is simply a polygon, call it A . Extending our notion of a triangulation to arbitrary non-planar polygons would immediately provide an example of such an algorithm. We do so as follows:

Algorithm 1.

Given a polygon A_n with n vertices, we choose the least area ear E_n and add it to our set of triangles, which we will call T . Then we return a polygon A_{n-1} with $n - 1$ vertices, given by

$$A_{n-1} = (A_n - (A_n \cap \partial E_n)) \cup (\partial E_n - (A_n \cap \partial E_n))$$

We then descend until $n = 3$, at which point we add the remaining triangle E_3 to T and the algorithm concludes.

It might not be immediately obvious that this works, so we will prove it.

Theorem 3.1. *Given a piecewise linear curve A_n , Algorithm 1 produces a surface of disk type with A_n as its boundary.*

Proof. We consider the algorithm in reverse. Let $T_m = \{E_3, \dots, E_m\}$. We now proceed by induction.

Base case: $n = 3$. Clearly $T_3 = \{E_3\}$, and E_3 is a triangle with boundary A_3 . Also, a triangle is clearly topologically equivalent to a disk.

Inductive step: Suppose T_m ($m < n$) is a surface of disk type with boundary A_m . Then we must show that T_{m+1} is a surface of disk type with boundary A_{m+1} .

Note $T_{m+1} = T_m \cup \{E_{m+1}\}$, and

$$\begin{aligned}\partial T_{m+1} &= \partial T_m \cup \partial E_{m+1} - \partial T_m \cap \partial E_{m+1} \\ &= A_m \cup \partial E_{m+1} - A_m \cap \partial E_{m+1},\end{aligned}$$

by the inductive hypothesis. Now note that $A_m \cup \partial E_{m+1} - A_m \cap \partial E_{m+1} = ((A_{m+1} - B) \cup (\partial E_{m+1} - B) \cup \partial E_{m+1}) - (((A_{m+1} - B) \cup (\partial E_{m+1} - B)) \cap \partial E_{m+1})$, where $B = A_{m+1} \cap \partial E_{m+1}$. Then

$$\begin{aligned}RHS &= ((A_{m+1} - B) \cup \partial E_{m+1}) - (\partial E_{m+1} - B) \\ &= (A_{m+1} \cup \partial E_{m+1}) - (\partial E_{m+1} \cap A_{m+1}^c) \\ &= A_{m+1} \cup (\partial E_{m+1} \cap A_{m+1}) = A_{m+1}\end{aligned}$$

Therefore, we have $\partial T_{m+1} = A_m \cup \partial E_{m+1} - A_m \cap \partial E_{m+1} = A_{m+1}$, as desired. Furthermore, it is easy to see that T_{m+1} is a disk, since it is formed by gluing a triangle to a disk along one side. □

The fact that Algorithm 1 only produces disks is an obvious flaw (consider the canonical “earmuffs” shape pictured in Figure 3.1), but how do the disks Algorithm 1 produces compare with the minimal disks? We continue our analysis by restricting our attention to planar boundary curves.

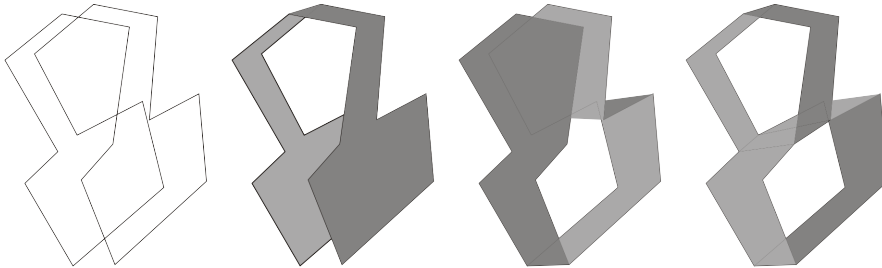


Figure 3.1. An example of a boundary curve (leftmost image) filled in three different ways. The middle two fillings are both minimal disks (minimal surfaces of disk type) of equal area, while the rightmost filling is a torus and has less area than the other two.

We will attempt to bound the area of the first triangle selected by Algorithm 1 (with a convex planar polygon as the input), but before we need some results about planar polygons.

We will begin with a key subsidiary lemma:

Lemma 3.1. *Every planar polygon has a diagonal.*

Proof. Take any planar polygon A . A contains at least one convex angle (for example we may take the rightmost point of A), call it α . Consider the adjacent vertices β and γ . There are two cases:

Case 1: The line connecting β and γ lies inside A .

Then this line $\beta\gamma$ is a diagonal, and we are done.

Case 2: $\beta\gamma$ crosses the boundary of A . Then there must be a vertex δ inside the triangle $\alpha\beta\gamma$ such that $\alpha\delta$ is a diagonal. To see this, consider a line parallel to $\beta\gamma$ crossing the vertex α . As this line moves in toward $\beta\gamma$, it must eventually intersect a point of the boundary inside the triangle $\alpha\beta\gamma$. The first point of intersection must be a vertex (call it δ), and consequently the line $\alpha\delta$ is our desired diagonal. \square

Using the result from Lemma 3.1, we can prove that every planar polygon admits a triangulation.

Lemma 3.2. *Every planar polygon admits a triangulation.*

Proof. We proceed by induction on the number of sides.

Base case: $n = 3$. Any polygon with three sides is defined to be a triangle. Thus, for our set of triangles we can just take $\{A\}$, which clearly satisfies the criteria.

Inductive step: Assume the theorem holds for a polygon with m sides. We will show that it holds for a polygon with $m + 1$ sides.

Consider any polygon A with $m + 1$ sides. Then by Lemma 3.1 it has a diagonal. The diagonal partitions A into two disjoint polygons, one of which has l sides, the other has $m - l + 3$ sides (the diagonal adds two to the total count). Note both l and $m - l + 3$ must be less than $m + 1$, since $(m - l + 3) \geq 3$ implies $l \leq m$, and $l \geq 3$ implies $(m - l + 3) \leq m$. By the inductive hypothesis, each can be triangulated, with the l -sided polygon having $l - 2$ triangles in its triangulation, and the $m - l + 3$ sided polygon having $m - l + 1$. The union of these two triangulations is disjoint, and hence a triangulation of A with a total of $m - 1 = (m + 1) - 2$ triangles, so the inductive step holds. \square

Lemma 3.3. *Every polygon contains at least two ears.*

Proof. Consider any triangulation of a polygon A . Proving the lemma amounts to proving that this triangulation contains at least one triangle with two boundary edges. Suppose the contrary.

If each triangle contains exactly one boundary edge, then the triangulation has n triangles, but we know that a triangulation has $n - 2$ triangles. This is a contradiction, and implies that a polygon must have at least two triangles with two boundary edges, and hence at least two ears. \square

Now we have enough machinery to prove our main result.

Theorem 3.2. *The area of the first triangle selected by Algorithm 1 (when given a convex polygon as its input) is bounded above by $\frac{3A}{n}$, where n is the number of vertices of the input polygon, and A is its area, computed by adding together the areas of each triangle in its triangulation.*

Proof. Our strategy will be to prove this upper bound for the average area of each selectable ear. Since the least area ear trivially has less area than this average, the theorem will follow. We again proceed by induction on the number of sides.

Base case: $n = 3$. Then clearly the area of the first triangle is the area of the entire figure, namely $A = \frac{3A}{3}$.

Inductive step: Assume the theorem holds for a polygon with m sides. We will show that it holds for a polygon with $m + 1$ sides.

Let A be a polygon with $m + 1$ sides. By Lemma 3.2 we know that A admits a triangulation T . By Lemma 3.3 we know that T contains an ear, call it E . We choose a direction for going around the boundary, and label five vertices a, b, c, d, e , where b, c, d are the vertices of E , and a, e are the immediately adjacent vertices on either side (of course for a quadrilateral we have the identification $a = e$). We define $T - E$ to be the m -sided polygon formed by removing E from T , and let S be the sum of the areas of all ears between vertex e and a (inclusive). Note that

$$A(abc) \leq A(bcd) + A(abd)$$

and

$$A(cde) \leq A(bcd) + A(bde).$$

Summing both expressions yields

$$A(abc) + A(cde) \leq 2A(bcd) + A(abd) + A(bde).$$

The inductive hypothesis gives us that

$$S + A(abd) + A(bde) \leq 3A_m,$$

so we have that

$$S + A(abd) + A(bde) + A(abc) + A(cde) \leq 2A(bcd) + A(abd) + A(bde) + 3A_m$$

Simplifying and adding $A(bcd)$ to both sides yields

$$S + A(abc) + A(bcd) + A(cde) \leq 3(A_m + A(bcd)) = 3A_{m+1},$$

so that

$$\frac{S + A(abc) + A(bcd) + A(cde)}{m + 1} \leq \frac{3A_{m+1}}{m + 1}$$

The LHS is just the average of the areas of each ear, which is greater than or equal to the area of the ear picked by Algorithm 1. The inductive step holds, and the theorem follows. \square

It is also easy to see that Algorithm 1 produces no error when given a planar convex polygon as its input. This is because since each line between any two vertices lies completely in the interior of the polygon, each triangle selected by Algorithm 1 is in the interior of the polygon.

We would like to be able to say more about Algorithm 1, but unfortunately its usefulness does not extend far past convex planar polygons. When the convexity condition is dropped, there exists no $O(\frac{1}{n})$ bound for the area of the first triangle picked, and the error can grow without bound. The following example (due to Juan Soto) illustrates these problems:

Bad Example 1. *See Figure 3.2*

We can let w get arbitrarily small and z (and hence n) get arbitrarily large. As $z \rightarrow \infty$, the area of the first triangle picked remains constant - it is always A , while $\frac{3A_n}{n}$ goes to zero. As $z \rightarrow \infty$, the error goes to infinity as $O(n)$.

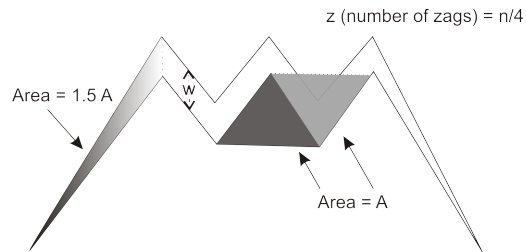


Figure 3.2. An example of a boundary curve for which Algorithm 1 produces an error that grows to infinity as $O(n)$. The algorithm will select a triangle of area A (depicted with solid shading above) until no more remain, at which point it will select the two triangles of area $1.5A$ (depicted with gradient shading above).

In this case it becomes clear that our strategy of only selecting polygon ears at each step is overly restrictive - sometimes we would like to select triangles that share only one edge with the boundary. The author has experimented with certain algorithms that remove this restriction, but their analysis is best left for another paper.

4 A New Approach

In this section we change our approach to the problem. This change is motivated by the idea that if we look at a plane intersecting a closed curve C (such that the curve is nowhere tangent to the plane), the intersection will consist of an even number (say $2n$) of points. Computational geometer Jack Edmonds discovered an algorithm that finds the n segments of minimal total length such that each segment ends in two of the intersection points - this is called the minimum weight perfect matching (MWPM). [?] This can be considered a lower dimensional analogue of our desired algorithm, and it seems to suggest an approach for our case.

Namely, if we consider some set of planes that intersect our surface, we can find the MWPM of the points in each plane, and by putting the planes together in some clever way we can create a surface with small area.

Before we present our algorithm, we explain the ideas behind Edmonds' matching algorithm, and for this we need some terminology and basic results from graph theory and combinatorial optimization. Much of the following discussion can be found in Chapter 5 of the book *Combinatorial Optimization* by Cook, et al. [?]

4.1 Introduction to Matchings

Let $G = (V, E)$ be a graph, where V is a set of nodes and E a set of edges. A *matching* is a set M of edges such that no node of G is incident with more than one edge in M . If some edge in M is incident with a node $v \in V$, we say that v is *M -covered*. Otherwise we say v is *M -exposed*. The number of nodes covered by M is exactly $2|M|$, and the number of M -exposed nodes is $|V| - 2|M|$. A *maximum* matching is one of maximum cardinality (one always exists). Let $\nu(G)$ denote the cardinality of a maximum matching of G , and *def*(G) (the deficiency) = $|V| - 2\nu(G)$. A *perfect* matching is one which covers all the nodes.

Our goals in this section are to examine under what conditions a graph has a perfect matching and if it does how to find one. But first we will need some additional terminology and a few more results.

A *path* is a sequence of nodes (also called vertices) such that from each of its nodes there is an edge in G to the next vertex in the sequence. The first vertex is called the *start vertex* and the last vertex is called the *end vertex* - both of them are *terminal vertices*. The other vertices are called *internal*. A *cycle* is a path such that the start and end vertex are the same.

Given a matching M of G , a path P is *M -alternating* if its edges are

alternately in and not in M . If in addition the end nodes of P are distinct and both M -exposed then we call P an M -augmenting path.

We begin with a theorem which allows us to check if a matching is maximum.

Theorem (Augmenting Path Theorem of Matchings) 4.1. *A matching M in a graph $G = (V, E)$ is maximum if and only if there is no M -augmenting path.*

Proof. Suppose there exists an M -augmenting path P joining nodes v and w . Then $N = M \triangle E(P)$ is a matching that covers all nodes covered by M , along with v and w . Thus M cannot be maximum.

Conversely, suppose M is not maximum and so there exists some other matching N with $|N| > |M|$. We argue that G contains an M -augmenting path. Let $J = N \triangle M$. Each node of G is incident with at most two edges of J , so J is the edge-set of some node-disjoint paths and circuits of G . For each such path or circuit, the edges alternately belong to M or N . Therefore, all circuits are even, and contain the same number of edges of M and N . Since $|N| > |M|$, there must therefore be at least one path with more edges of N than M . This path is M -augmenting. \square

Theorem 4.1 immediately suggests a possible way to construct a maximum matching. Namely, we search for an augmenting path and use the path to obtain a new matching, until we find a matching with no augmenting path. This is the approach we will follow, but first we need a way to tell when a matching is maximum, so that the search for an augmenting path may be abandoned.

We define a *cover* of G to be a set A of nodes such that each edge has at least one end in A . The cardinality of a cover is an obvious upper bound for the size of any matching. But we can get something stronger.

In a graph G , a *component* is a maximal connected subgraph. Two nodes are in the same component if there exists a path between them. If a component has an odd number of nodes, we call it an *odd component*. We define $oc(H)$ to be the number of odd components of a graph H .

Lemma 4.1. *For any $A \subset V$, $\nu(G) \leq \frac{1}{2}(|V| - oc(G - A) + |A|)$*

Proof. Let $G - A$ have k odd components H_1, \dots, H_k . Let M be a matching of G . For each i , either H_i has an M -exposed node, or M contains an edge having just one end in $V(H_i)$. All such edges must have their other ends in A , and since M is a matching, there can be at most $|A|$ such edges. So the number of M -exposed nodes must be at least $k - |A| = oc(G - A) - |A|$.

Note that this number is independent of the matching chosen, so it must hold for all matchings. Thus it must hold for the maximum matching. Thus for any $A \subset V$,

$$\nu(G) \leq \frac{1}{2}(|V| - (oc(G - A) - |A|)) \leq \frac{1}{2}(|V| - oc(G - A) + |A|).$$

□

There exists a strengthening of the above bound, known as the Tutte-Berge Formula.

Theorem (Tutte-Berge Formula) 4.2. $\nu(G) = \min\{\frac{1}{2}(|V| - oc(G - A) + |A|) : A \subset V\}$.

Corollary (Tutte's Matching Theorem). *A graph $G = (V, E)$ has a perfect matching if and only if for every subset A of nodes we have $oc(G - A) \leq |A|$.*

Proof. Suppose G has a perfect matching. Then $\nu(G) = \frac{1}{2}|V|$, so by the bound in Lemma 1, we must have $oc(G - A) \leq |A|$. Conversely, suppose for every subset A of nodes we have $oc(G - A) \leq |A|$. Now consider $A = \emptyset$. Then $|A| = 0$, so $oc(G - A) = 0$, and the result follows by the Tutte-Berge Formula. □

4.2 Alternating Trees and Perfect Matching

Beginning with a matching M of G and a fixed M -exposed node r of G , we can build what is known as an M -alternating tree. We do so by building up sets A, B of nodes, such that each node of A is the other end of an odd-length M -alternating path beginning at r (and each node of B is the other end of an even-length M -alternating path beginning at r) - the motivation of course is that if we find some edge vw such that $v \in B$ and $w \notin A \cup B$ is M -exposed, then the M -alternating path from r to w is an M -augmenting path, which could then be used to construct a higher cardinality matching.

We start with $A = \emptyset$, $B = \{r\}$, and use the following rule: If $vw \in E$, $v \in B$, $w \notin A \cup B$, $wz \in M$, then add w to A , z to B . Notice that nodes are added to A and B in pairs, and so $|B| = |A| + 1$ at each stage of the tree's construction. The set $A \cup B$ and the edges used in its construction form a tree T with root r having the following two properties:

- (1) Every node of T other than r is covered by an edge of $M \cap E(T)$.

(2) For every node v of T , the path in T from r to v is M -alternating.

Given an M -alternating tree T , we denote by $A(T)$ and $B(T)$ the sets of nodes at odd and even distance from the root, respectively. We will refer to elements of $A(T)$ as *odd nodes* and elements of $B(T)$ as *even nodes*.

Now we introduce two useful subroutines - the first two extend the tree, and the second to augment the matching.

Subroutine 1 (Extending the Tree).

Input: A matching M of a graph G , an M -alternating tree T , and an edge vw of G such that $v \in B(T)$, $w \notin V(T)$ and w is M -covered.

Output: Let wz be the edge in M covering w . Replace T by the tree having edge set $E(T) \cup \{vw, wz\}$.

Note that in order for subroutine 1 to work, z cannot already be a vertex in the tree. But this obviously holds, because if z is already in the tree, then either $z = r$ or z is already matched to some other vertex in the tree. The latter cannot hold because z is already matched to w , and $w \notin V(T)$. Thus we have $z = r$, which cannot hold because r is M -exposed.

Subroutine 2 (Augmenting the Matching).

Input: A matching M of a graph G , an M -alternating tree T of G with root r , and an edge vw of G such that $v \in B(T)$, $w \notin V(T)$ and w is M -exposed.

Output: Let P be the path obtained by attaching vw to the path from r to v in T . Replace M by $M \Delta E(P)$.

In order for any perfect matching algorithm to be successful, we need to know when a graph has no perfect matching. For this we introduce the notion of a *frustrated* tree.

We call a tree T in a graph G *frustrated* if every edge of G having one end in $B(T)$ has the other end in $A(T)$.

Theorem (Frustration Theorem) 4.3. *Suppose that G has a matching M and a frustrated M -alternating tree T . Then G has no perfect matching.*

Proof. Every element of $B(T)$ (even node) is a single-node odd component of $G - A(T)$. Since $A(T) < B(T)$ the result follows by Tutte's Matching Theorem. \square

4.3 Perfect Matching in Bipartite Graphs

A bipartite graph G is a graph whose vertices can be partitioned into two sets such that no two vertices in the same set are connected by an edge.

Perfect Matching Algorithm for Bipartite Graphs.

```
Set  $M = \emptyset$ ;  
Choose an  $M$ -exposed node  $r$  and put  $T = (\{r\}, \emptyset)$ ;  
While there exists  $vw \in E$  with  $v \in B(T)$ ,  $w \notin V(T)$   
{  
  If  $w$  is  $M$ -exposed  
    Use  $vw$  to augment  $M$ ;  
    If there is no  $M$ -exposed node in  $G$   
      Return the perfect matching  $M$  and stop;  
    Else replace  $T$  by  $(\{r'\}, \emptyset)$ , where  $r'$  is  $M$ -exposed;  
  Else use  $vw$  to extend  $T$ ;  
}  
Stop;  $G$  has no perfect matching
```

The following theorem ensures us that the algorithm is correct.

Theorem (Correctness of Bipartite Matching Algorithm) 4.4. *Suppose that G is bipartite, M is a matching of G , and that T is an M -alternating tree such that no edge of G joins a node in $B(T)$ to a node not in $V(T)$. Then T is frustrated, and hence G has no perfect matching.*

Proof. We proceed by showing that every edge of the tree having one end in $B(T)$ has the other end in $A(T)$. By hypothesis, the only exception would be an edge joining two nodes in $B(T)$. But this edge, together with the paths joining each of its ends to the root, would form a closed path of odd-length, which is impossible in a bipartite graph. Thus the tree is frustrated, and by Theorem 4.3, G has no perfect matching. \square

4.4 The Blossom Algorithm

The algorithm from the previous section obviously cannot work in general because we have the issue of odd circuits. To deal with these seemingly problematic circuits, we need to introduce a new algorithmic idea - the *shrinking* of odd circuits.

Let C be an odd circuit in G . Define $G' = G \times C$, the subgraph obtained from G by shrinking C , as follows. $G \times C$ has node set $(V - V(C)) \cup \{C\}$

and edge-set $E - \gamma(V(C))$ (all edges except those between two nodes in C). Thus, if e is an edge of G' with end $v \in G$, v is an end in G' if $v \notin V(C)$, and otherwise C is an end of e in G' .

There are a few fundamental observations that we can make right away. First of all, a matching in $G \times C$ extends to a matching in G in an obvious way.

Theorem (Extension of a Matching) 4.5. *Let C be an odd circuit of G , let $G' = G \times C$, and let M' be a matching of G' . Then there is a matching M of G such that $M \subset M' \cup E(C)$ and the number of M -exposed nodes of G is the same as the number of M' -exposed nodes of G' .*

Proof. Choose a node $w \in V(C)$ as follows. If C is covered by $e \in M'$, then choose w to be the node in $V(C)$ that is an end of e , and otherwise, choose w arbitrarily. Deleting w from C results in a subgraph having a perfect matching M'' . Take $M = M' \cup M''$. \square

Note that Theorem 4.5 gives the following inequality:

$$def(G) \leq def(G \times C).$$

Suppose G' is a graph obtained from another graph G by a sequence of odd-circuit shrinkings. We will then call G' a *derived graph* of G . A node of G' is either a node of G (an *original node*), or not a node of G , in which case we call it a *pseudonode*. Given a node v of G' , there corresponds a set $S(v)$ of nodes of G . Note that if $v \in V$, that is if v is an original node, then $S(v) = \{v\}$. But if $v = C$ is a pseudonode, then $S(v)$ is recursively defined as $\bigcup_{w \in C} S(w)$. Note that the cardinality of $S(v)$ is odd and that the sets $S(v)$ form a partition of V .

Using this new language, we have the following extension of Theorem 4.3:

Theorem (New Frustration Theorem) 4.6. *Suppose G' is the derived graph of G and M' is a matching of G' . Let T be an M' -alternating tree of G' such that no element of $A(T)$ is a pseudonode. If T is frustrated, then G has no perfect matching.*

Proof. Removing $A(T)$ from G , we get a component with node-set $S(v)$ for each $v \in B(T)$. Therefore, $oc(G - A(T)) > |A(T)|$, and so by Tutte's Matching Theorem, there exists no perfect matching of G . \square

Now suppose there exists an edge vw , with $v, w \in B(T)$. Then this edge together with the path in T from v to w forms an odd circuit (called

a blossom), which we can then shrink to form a derived graph G' using the following subroutine.

Subroutine 3 (Shrinking the Blossom).

Input: A matching M of a graph G , an M -alternating tree T of G with root r , and an edge vw of G such that $v, w \in B(T)$.

Output: Let C be the odd circuit formed by vw together with the path in T from v to w . Replace G by $G \times C$, M by $M - E(C)$, and T by the tree (in G) having edge set $E(T) - E(C)$.

Theorem 4.7. *After an application of the shrinking subroutine, M is a matching of G , T is an M -alternating tree of G , and $C \in B(T)$.*

Proof. Trivial. □

Now we are ready to state the general algorithm for perfect matching - the Blossom Algorithm.

Blossom Algorithm.

```

Input graph  $G$ ;
Set  $M = \emptyset$ ;
Choose an  $M$ -exposed node  $r$  of  $G$  and put  $T = (\{r\}, \emptyset)$ ;
While there exists  $vw \in E$  with  $v \in B(T)$ ,  $w \notin A(T)$ 
{
  If  $w \notin V(T)$ ,  $w$  is  $M$ -exposed
    Use  $vw$  to augment  $M$ ;
    Extend  $M$  to a matching  $M'$  of  $G$ ;
    Replace  $M$  by  $M'$ ;
    If there is no  $M$ -exposed node in  $G$ 
      Return the perfect matching  $M$  and stop;
    Else replace  $T$  by  $(\{r'\}, \emptyset)$ , where  $r'$  is  $M$ -exposed;
  Else if  $w \notin V(T)$ ,  $w$  is  $M$ -covered
    Use  $vw$  to extend  $T$ ;
  Else if  $w \in B(T)$ 
    Use  $vw$  to shrink and update  $M$  and  $T$ ;
}
Return  $G$ ,  $M$ ,  $T$  and stop;  $G$  has no perfect matching.

```

Theorem 4.8. *The Blossom Algorithm terminates after $O(n)$ augmentation steps, $O(n^2)$ shrinking steps, and $O(n^2)$ tree-extension steps. It also determines correctly whether G has a perfect matching.*

Proof. Clearly M remains a matching throughout the algorithm. Since each augmentation decreases the number of exposed nodes, there will be $O(n)$ augmentations. Between augmentations, each shrinking step decreases the number of nodes in G while not changing the number of nodes not in T , and each tree-extension step decreases the number of nodes not in T while not changing the number of nodes of G . Thus there are $O(n)$ of each type of step per augmentation, which means $O(n^2)$ of each in total. Thus the blossom algorithm is polynomial, and at most $O(n^5)$.

Finally, if the algorithm halts with the conclusion that G has no perfect matching, then it has found a tree with the properties of Theorem 4.6, and so G has no perfect matching. \square

4.5 Minimum-Weight Perfect Matching in Bipartite Graphs

We begin this section with an integer-linear-programming formulation of the bipartite minimum weight perfect matching problem.

$$\begin{aligned} &\text{Minimize} && \sum_{(i,j) \in A \times B} c_{ij} x_{ij} \\ &&& \text{subject to} \\ &&& \sum_i x_{ij} = 1 \\ &&& \sum_j x_{ij} = 1 \\ &&& x_{ij} \in \{0, 1\} \end{aligned}$$

In this case the c_{ij} weights of each edge, and x_{ij} is either 0 or 1, depending on whether the edge connecting i and j is in the matching.

To get a lower bound on the minimum weight of a perfect matching, we consider the following linear-programming relaxation of the problem.

$$\begin{aligned} &\text{Minimize} && \sum_{(i,j) \in A \times B} c_{ij} x_{ij} \\ &&& \text{subject to} \\ &&& \sum_i x_{ij} = 1 \\ &&& \sum_j x_{ij} = 1 \end{aligned}$$

$$x_{ij} \geq 0$$

In general there is no guarantee that the optimal solution to the relaxation is an optimal solution to the integral problem. Fortunately, in the bipartite case the two indeed coincide. This fact follows from a classic theorem by Birkhoff, and will be evident in the following discussion.

Suppose we have values u_i for $i \in A$ and v_j for $j \in B$ such that $u_i + v_j \leq c_{ij}$ for all $i \in A$ and $j \in B$. Then for any perfect matching M , we have

$$\sum_{(i,j) \in M} c_{ij} \geq \sum_{i \in A} u_i + \sum_{j \in B} v_j. (1)$$

Thus the right hand side of the above inequality provides a *lower bound* for the total weight of the minimum weight perfect matching. To get the best lower bound, we would naturally like to maximize the above quantity, which leads us to the following linear program, called the *dual linear program*.

$$\text{Maximize } \sum_{i \in A} u_i + \sum_{j \in B} v_j$$

subject to

$$u_i + v_j \leq c_{ij}$$

$$i \in A, j \in B$$

Setting $w_{ij} = c_{ij} - u_i - v_j$ puts the dual problem in a more familiar form.

If we could always find a feasible solution u, v to the dual problem and a perfect matching M such that we have equality in (1) above (if and only if $w_{ij} = 0$ for all $(i, j) \in M$), then we know that the matching is optimum. This is known as the *complementary slackness* condition.

Now we summarize the algorithm. First, we start with any dual feasible solution - such as $u_i = 0$ for all i and $v_j = \min_i c_{ij}$ for all j . In a given iteration, the algorithm has a dual feasible solution (u, v, w) . We impose complementary slackness, that is we look at matchings that are subgraphs of $E_0 = \{(i, j) : w_{ij} = 0\}$, the set of *equality edges*. If E_0 has a perfect matching, then the incidence vector of that matching is a feasible solution of the original linear programming problem and satisfies complementary slackness with the current dual solution and, hence, must be optimal. We can check whether E_0 has a perfect matching using the algorithm from the previous section. Otherwise, the algorithm will deliver a matching M of $G_E = (V, E_0)$ and an M -alternating tree such that nodes of $B(T)$ are joined by equality edges only to nodes in $A(T)$.

In this latter case, there is a natural way to change u and v , keeping in mind that we would like edges of M and T to remain in E_0 , and that we would like w_{ij} to decrease for edges joining nodes in $B(T)$ to nodes not in $A(T)$. Namely, for a vertex i , we increase u_i (or v_i) by $\epsilon > 0$ if $i \in B(T)$ and decrease u_i (or v_i) by ϵ if $i \in A(T)$. We choose ϵ as large as possible so that feasibility in (1) is not lost, and as a result, provided that G has a perfect matching, some edge joining a node in $B(T)$ to a node not in $A(T)$ will enter E_0 , leading to a possible tree-extension or augmentation step. Note that all edges that were originally in E_0 are still in E_0 , since the graph is bipartite and hence vertices in $B(T)$ cannot be connected to others in $B(T)$ (and similarly for vertices in $A(T)$).

Minimum-Weight Perfect Matching Algorithm for Bipartite Graphs.

```

Let  $(u, v)$  be a feasible solution to (1),  $M$  a matching of  $G_E$ ;
If  $M$  is a perfect matching of  $G$ , return  $M$  and stop;
Set  $T = (\{r\}, \emptyset)$ , where  $r$  is an  $M$ -exposed node of  $G$ ;
Loop
While there exists  $vw \in E_0$  with  $v \in B(T)$ ,  $w \notin V(T)$ 
{
  If  $w$  is  $M$ -exposed
    Use  $vw$  to augment  $M$ ;
    If there is no  $M$ -exposed node in  $G$ 
      Return the perfect matching  $M$  and stop;
    Else replace  $T$  by  $(\{r'\}, \emptyset)$ , where  $r'$  is  $M$ -exposed;
  Else
    Use  $vw$  to extend  $T$ ;
If every  $vw \in E$  with  $v \in B(T)$  has  $w \in A(T)$ 
  Stop,  $G$  has no perfect matching;
Else
  Let  $\epsilon = \min\{c_{vw} : v \in B(T), w \notin V(T)\}$ ;
  For each vertex  $i$ , we increase  $u_i$  (or  $v_i$ ) by  $\epsilon > 0$  if  $i \in B(T)$  and decrease
   $u_i$  (or  $v_i$ ) by  $\epsilon$  if  $i \in A(T)$ ;
}

```

The running time for this algorithm is higher than for the unweighted case, since in the worst case we may have to do a dual change for each tree-extension step, giving $O(n^2)$ dual changes (and $O(n)$ steps in computing ϵ for each) and an overall running time of no more than $O(n^4)$.

5 Algorithm 2

5.1 A Few Basic Constructions

This section introduces a new algorithm for creating an orientable small area surface with an arbitrary piecewise linear curve as its boundary. Before stating the algorithm, we need a few more constructions.

A closed piecewise linear oriented curve $C = (V, E)$ in \mathbb{R}^3 with n vertices can be represented as a sequence of n ordered triplets, which give the coordinates of each vertex with respect to a chosen orthonormal basis $\{e_1, e_2, e_3\}$. The order of the triplets in the sequence represents the order in which they are connected - with the additional stipulation that the last triplet in the sequence is connected to the first. Additionally we can assign an orientation to the curve.

We will call a plane in \mathbb{R}^3 *admissible* if it has e_3 as a normal vector. Let \mathcal{P} be the set of admissible planes which intersect a vertex of C . Suppose the z -values of these planes range from z_0 to z_1 . Then the set of planes \mathcal{P} partitions the region $\{(x, y, z) \in \mathbb{R}^3 | z \in [z_0, z_1]\}$.

Let $\mathcal{S}_0 = \mathcal{P}$. For each integer n , let \mathcal{S}_n be the union of \mathcal{S}_{n-1} and the set of planes which lie at the midpoints of the z -coordinates of the planes in \mathcal{S}_{n-1} . Define $\mathcal{P}_n = \mathcal{S}_n - \mathcal{P}$.

For each plane P in \mathcal{S}_n , $P \cap C$ consists of an even number of points, which can be partitioned into two sets A and B , depending on the local orientation of the curve relative to P . Given these points, we can compute the distances between pairs and perform a minimum weight bipartite perfect matching (MWBPM) with the resulting graph using the algorithm from the previous section.

Define the following equivalence relation \sim for planes in \mathcal{P}_n : $P_1 \sim P_2$ if they, and all planes between them, have the same MWBPM. This equivalence relation partitions the planes into disjoint equivalence classes, which we will call *stacks*. Each stack consists of several chains of quadrilaterals. We define the area of a stack as the sum of the areas of the least area triangulations of each of the quadrilaterals. We also define the coordinate of a stack as the smallest z -coordinate of any of its planes. We will also call two stacks *adjacent* if there is no stack or element of \mathcal{P} between them.

Thus, we are left with a certain number of stacks, separated by certain elements of \mathcal{P} . To create a surface, we must then do the following three things:

- (1) Connect the planes within a stack
- (2) Connect distinct stacks together

(3) Connect stacks to planes in \mathcal{P}

The first of these is reasonably clear, and we now present an algorithm for it.

Subroutine 1 (Connect Planes within a Stack).

Input: A stack S ;

Action:

Set $T = \emptyset$;

For each quadrilateral Q in S , find the least area triangulation and add the two triangles to T ;

Return T ;

The second of the aforementioned algorithms is more tricky, and requires a bit more discussion.

5.2 Simplicity and the Interface Plane

In this section we motivate the idea of the *interface plane* to combine two stacks with different minimum weight perfect matchings. Given two such adjacent stacks A and B , a natural way to combine them is to create a new plane \mathcal{I} containing both matchings and to place the plane halfway between the two stacks. Thus, two points in \mathcal{I} are matched if and only if they are matched in A or B . Points in \mathcal{I} are grouped in pairs and polygons, each consisting of an even number of points. Note further that each polygon contains edges alternately in A and B (see Figure 5.1).

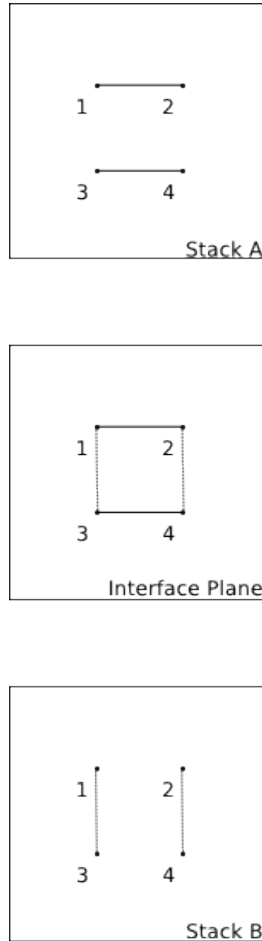


Figure 5.1. *Two stacks A and B together with their respective matchings. The interface plane consists of a polygon with edges alternating between the two matchings.*

Note that unless the polygons in these interface planes are filled in, the edges in \mathcal{I} will be part of the boundary. In order to fill in the polygons, we employ the standard criterion that a point is “inside” the polygon if a ray through that point intersects the boundary of the polygon in an odd number of places. This immediately suggests an algorithm for filling in the polygon.

Our strategy is to consider two sets of parallel lines \mathcal{L} and \mathcal{L}' (all having constant y -coordinate, for instance) in the plane of the polygon $P = (V, E)$. We include a line in \mathcal{L} if it intersects a vertex of the polygon. We call a vertex of the polygon “special” if it is connected (via edges in E) to two

points that are on the same line in \mathcal{L} . Finally we add the lines that lie halfway between adjacent lines in \mathcal{L} to \mathcal{L}' . For each line l' in \mathcal{L}' , we can join the points in $l' \cap P$ by first joining the endpoints to their closest neighbor and then working inward. Then we look at the special vertices, and finally we conclude by joining the remaining regular vertices on the lines in \mathcal{L} . The algorithm is as follows (see Figure 5.2 for example).

Algorithm (Filling in a Polygon).

Input: A polygon $G = (V, E)$ at $z = z_0$ (elements of V are ordered triplets of the form (x, y, z_0)), sets \mathcal{L} and \mathcal{L}' as described above;

Set $T = \emptyset$;

Action:

For each line $l' \in \mathcal{L}'$

{
 Arrange the points in $l' \cap P$ in a sequence (p'_n) in order of increasing x -coordinate;

 For each point at an odd position in the sequence, add the edge connecting it to the next point in the sequence to E ;

}

For each line $l \in \mathcal{L}$

{

 Let $S = l \cap P$;

 For each point $p \in S$

 {

 If p is a special vertex and is part of a triangle (with all edges in E), add this triangle to T and remove p from S ;

 Else if p is a special vertex, add to E the two segments connecting p to its two neighbors in S and remove all three points from S ;

 }

 Arrange the points in S in order of increasing x -coordinate; For each point at an odd position in the sequence, add the edge connecting it to the next point in the sequence to E ;

}

For each quadrilateral in E , pick a diagonal and add the two resulting triangles to T ;

Return T ;

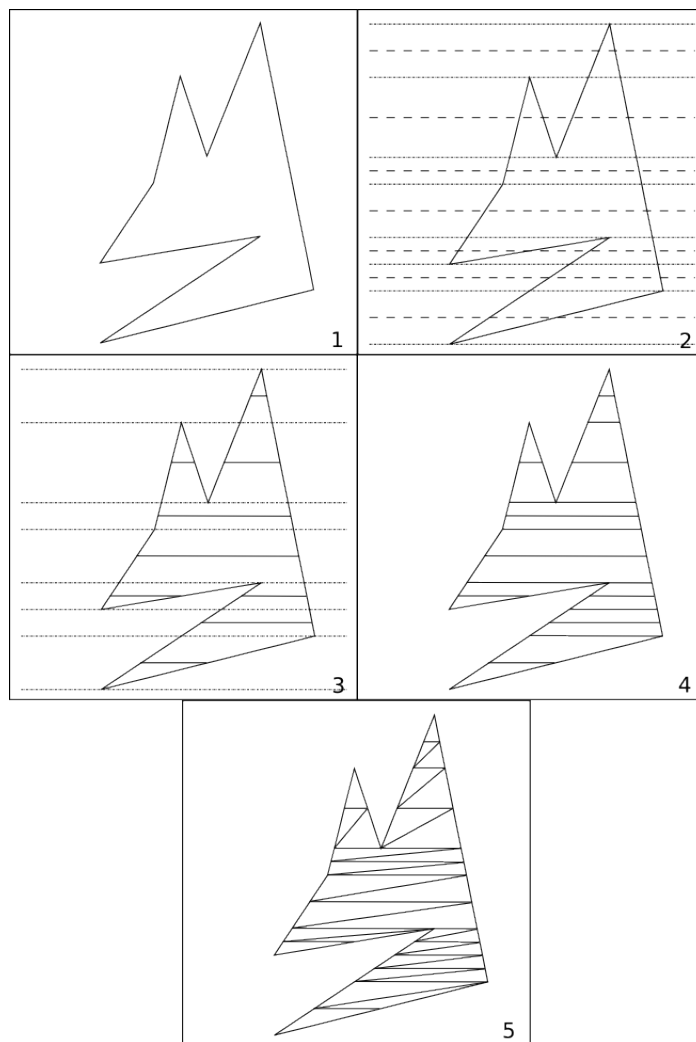


Figure 5.2. *Polygon-filling Algorithm in Action*

- 1) *Original Polygon*
- 2) *Polygon with \mathcal{L} (dotted lines) and \mathcal{L}' (dashed lines)*
- 3) *Adding edges to each line in \mathcal{L}'*
- 4) *Adding edges to each line in \mathcal{L}*
- 5) *Triangulating each quadrilateral*

We now conclude this section with the interesting result that far enough along in the algorithm, all polygons lying inside an interface plane will be simple. The reasoning is that, as the partitions become finer and finer, if two stacks differ in their matching, by continuity their interface plane will

approach a plane with two perfect matchings of equal minimum-weight. We next argue that these two matchings cannot have intersecting segments, but first we need the following result.

Lemma 5.1. *A minimum-weight perfect matching cannot have intersecting segments.*

Proof. Suppose for contradiction that the matching contains two intersecting segments ad and bc . Let e be the point of intersection. Then by the triangle inequality $|ab| < |ae| + |be|$ and $|cd| < |ce| + |ed|$. So $|ab| + |cd| < |ae| + |ed| + |be| + |ce| = |ad| + |bc|$ (See Figure 5.3). But then our matching could not be minimal because replacing the segments ad and bc with the segments ab and cd , while keeping all other segments the same, would result in a lesser weight matching. \square

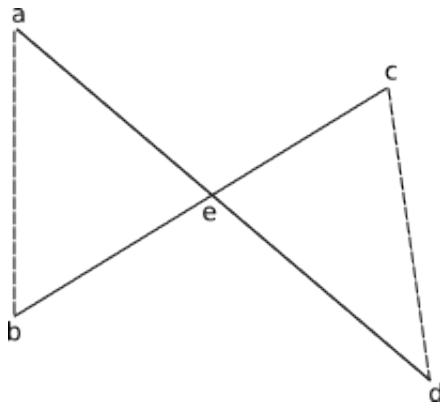


Figure 5.3. *A more optimal matching.*

Theorem 5.1. *Suppose a plane I contains the segments from two matchings of equal minimum-weight. Then the plane cannot have two intersecting segments.*

Proof. Suppose two segments intersect. Then we know that they cannot be from the same matching by our previous lemma. Again consider Figure 5.3, and by the previous statement we can assume ad is in Matching 1 and bc is in Matching 2. We replace these two segments by the segments ab and cd . If neither ad or bc are part of a polygon, then by the lemma we can assign ab to Matching 1 and cd to Matching 2, and thereby produce two new matchings with a combined weight less than the combined weight of the two original minimum-weight perfect matchings, an obvious contradiction.

Now suppose that both ad and bc are edges in larger polygons, with m and n edges, respectively. Then replacing these segments with ab and bc produces one large polygon with $m + n$ edges. Since $m + n$ is even, we can alternately place the edges in this polygon in Matching 1 and 2, and as before we get two new matchings with smaller combined weight - a contradiction. The case where only one of the edges is part of a larger polygon can be eliminated in a similar fashion.

Thus we conclude that if a plane contains the segments from two matchings of equal minimum-weight, then none of the segments can intersect each other. \square

5.3 The Algorithm

We recall that to create a surface out of a collection of stacks, we must do the following three things:

- (1) Connect the planes within a stack
- (2) Connect distinct stacks together
- (3) Connect stacks to planes in \mathcal{P}

We have already given an algorithm for (1), and for (2) we have introduced the idea of an interface plane. Now, given these connected interface planes, we explain how to refine them to reduce the area, and then we present an algorithm for (3). Since the first algorithm is fairly straightforward and the second is quite similar to the one presented in the previous section, we will avoid the formalism and simply state them here.

The refinement algorithm takes a chain of connected stacks and, starting with the stack of smallest area, does one of the following three things, depending on which results in least area (area of stack + adjacent interface planes):

- (a) change matching of the stack to matching of adjacent stack in the positive z -direction (if it exists), remove interface plane above, alter interface plane below, restart
- (b) change matching of the stack to matching of adjacent stack in the negative z -direction (if it exists), remove interface plane below, alter interface plane above, restart
- (c) change nothing and move on to next smallest stack

The algorithm repeats as long as there are stacks to consider; the number of steps is polynomial in the number of stacks.

Now we turn to the algorithm for step (3). This algorithm, as expected, builds up a set of triangles T . First we will call a vertex in some plane $P \in \mathcal{P}$ “special” if the two nearest points it is connected to (via boundary edges) are in the same stack. We will call these two points its neighbors. We look at the planes in \mathcal{P} in order of increasing z -coordinate. For a given plane P , a special vertex v will either be part of a triangle or not. If it is part of a triangle, then we add that triangle to T . If it is not, then its neighbors must be connected to two other points, which must be connected via the boundary to two points p_1 and p_2 in P . We will then connect each of these points to v . The remainder of the points on P must then be ordinary vertices or non-vertex boundary points. There will be an even number of these points, say $2m$, and they will each be joined via boundary to two points in two different stacks. If these $2m$ points have the same matching in both of the adjacent stacks, then we give them that matching in P and add the triangles making up the resulting quadrilaterals to T . Otherwise we give both matchings to P and add the triangles from the resulting planar polygon, along with quadrilaterals connecting it to the adjacent stacks, to T . This concludes the algorithm.

Now all that remains is to put everything together.

Least Area Surface Algorithm.

Input: A closed piecewise linear oriented curve $C = (V, E)$ with n vertices and $n - 1$ edges - and a corresponding orthonormal coordinate system $\{e_1, e_2, e_3\}$ in \mathbb{R}^3 (the curve is provided as a sequence of n ordered triplets), sets of planes \mathcal{P} and \mathcal{P}_m .

Action: Set $m = 1$, $T = \emptyset$, $A_1 = 0$, $A_2 = 0$;

Loop

{

Make planes in \mathcal{P}_m into stacks;

Connect planes within stacks together, add the triangles to T and total area of triangles to A_2 ;

Connect each pair of adjacent stacks via interface plane, add triangles to T and total area of triangles to A_2 ;

Refine the stacks and update T and A accordingly;

Connect stacks to planes in \mathcal{P} , add triangles to T and total area of triangles to A_2 ;

If $0.99A_1 < A_2 < A_1$

```

    break;
Let  $A_1 = A_2$  and  $A_2 = 0$ ;
m++;
}
Return  $T$ ;

```

5.4 Bipartite Matchings and Orientability

The reader might wonder why we have taken so much care in distinguishing bipartite graphs from general graphs. Aside from making the related algorithms much simpler, bipartite matchings are inextricably connected to the question of orientability. The connection is summarized in the following theorem.

Theorem 5.2. *Algorithm 2 produces an orientable surface with C as its boundary.*

Proof. First we verify that the algorithm produces a connected surface with C as its boundary. It is easy to see that the algorithm produces at least one surface, since the output consists of a set T of triangles, each of which is a surface itself. All that remains is to show that the set of triangles is connected. For this we recognize that any triangle in T is either part of a stack or part of a filled polygon in an interface plane. If it is part of a stack, then it shares one edge with C and two edges with other triangles. If it is part of a filled planar polygon, then it shares three edges with other triangles. Conversely, each point in C is part of an edge of exactly one triangle in T , and each point in a planar polygon or MWBPM is part of an edge shared by two triangles. This shows that C is the boundary of T , and since C is connected, T must also be connected.

Now it remains to show that the surface must be orientable. We note that C is oriented, and that each segment in the MWBPM is chosen in a way that is consistent with this orientation. Thus each stack consists of a set of disjoint orientable surfaces. It is easy to check that connecting stacks together via interface plane and connecting stacks to planes in P both preserve orientability. Thus, the surface determined by T is orientable and has as its boundary the closed curve C . \square

Having determined that doing a MWBPM results in an orientable surface, it is natural to ask whether all orientable surfaces with a given boundary C can be obtained by a bipartite matching. More precisely, given an orientable surface with C as its boundary, and a set of parallel planes \mathcal{S} ,

none of which intersects a vertex of C , we note that each plane in \mathcal{S} intersects the surface in a certain number of “segments.” We ask whether there is a way to partition the endpoints of these segments (for all planes in \mathcal{S}) into two sets A and B such that an element in A is only connected to an element in B (and vice versa). The answer to this question is yes. To see this, we recall that an orientation on a surface induces a natural orientation on C , and since all endpoints are in C , this orientation allows us to place the endpoints in two sets $+$ and $-$. If two $+$ (or $-$) points are connected via an edge e , then this edge, together with the part of the boundary connecting the points, would form an orientation reversing path, which would contradict the fact that our surface is orientable. It is clear that the sets $+$ and $-$ can be renamed A and B without affecting the result in any way.

6 Conclusions and Future Research

Much about Algorithm 2 remains to be explored. As it stands now, the algorithm depends on the chosen coordinate system. The author believes that there must be some reasonably simple method of choosing coordinates such that the result of the algorithm is most optimal. An interesting approach that the author has considered would be to allow the planes in \mathcal{P} to not be parallel, and then to pick the orientation that results in the least number of planes in \mathcal{P} .

Indications are that Algorithm 2 works better than the simple algorithms introduced in the first section, but we have no way of quantifying this improvement. If there is a constant factor that bounds the ratio of the area of the surface produced to the area of the optimal surface, we haven’t found a way to determine it. At the very least the algorithm could benefit from some sort of statistical analysis.

Finally, we note that there is a fairly intuitive extension of Algorithm 2 that produces surfaces that are not necessarily orientable. The strategy behind the extension is to perform a general minimum-weight perfect matching (rather than a MWBPM) by combining linear programming with the ideas from Edmonds’ Blossom Algorithm. The general MWPM algorithm can be found in Chapter 5 of *Combinatorial Optimization* by W. Cook et al.

7 Acknowledgements

I would like to thank my advisor Larry Guth for his advice, encouragement, ideas, and most of all his time in working with me for this thesis. Every

part of the process was very enjoyable for me. I would also like to thank professors Jim Nolen and Brian White for meeting with me and discussing my project, and Juan Soto for providing me with interesting examples to think about. Finally, I am very indebted to Caroline Ghosn, my family, and my friends for being very supportive throughout this project.