Computing Optimal Hatching Directions in Layered Manufacturing*

Man Chung Hon[1], Ravi Janardan[1], Jörg Schwerdt[2], and Michiel Smid[2]

[1] Dept. of Computer Science & Engineering, University of Minnesota, Minneapolis, MN 55455, U.S.A.
{hon,janardan}@cs.umn.edu
[2] Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, D-39106 Magdeburg, Germany.
{schwerdt,michiel}@isg.cs.uni-magdeburg.de

Abstract. In Layered Manufacturing, a three-dimensional polyhedral solid is built as a stack of two-dimensional slices. Each slice (a polygon) is built by filling its interior with a sequence of parallel line segments, of small non-zero width, in a process called hatching. A critical step in hatching is choosing a direction which minimizes the number of segments. Exact and approximation algorithms are given here for this problem, and their performance is analyzed both experimentally and analytically. Extensions to several related problems are discussed briefly.

1 Introduction

This paper addresses a geometric problem motivated by *Layered Manufacturing* (LM), which is an emerging technology that allows the construction of physical prototypes of three-dimensional parts directly from their computer representations, using a "3D printer" attached to a personal computer.

The basic idea behind LM is very simple. A direction is first chosen to orient the computer model suitably. The model is then sliced with a set of equally spaced horizontal planes, resulting in a stack of 2-dimensional polygons. Starting from the bottom, each slice is sent to the LM machine and built on top of the layers below it. There are several different ways how this process is carried out physically. One particular implementation is through a process called *Stereolithography* [3]. Here the model is built in a vat of liquid which hardens when exposed to light. A laser is used to trace the boundary of each slice and then fill in its interior via a series of parallel line segments (Fig. 1(a)); this process is called *hatching*. Another process called Fused Deposition Modeling hatches the slices by depositing fine strands of molten plastic via a nozzle.

The hatching process in LM influences the process cost and build time quite significantly. For instance, in Stereolithography, the number of times the laser's

* Research of MCH and RJ supported, in part, by NSF grant CCR–9712226. Portions of this work were done when RJ visited the University of Magdeburg and JS and MS visited the University of Minnesota under a joint grant for international research from NSF and DAAD.

V.N. Alexandrov et al. (Eds.): ICCS 2001, LNCS 2073, pp. 683–692, 2001.

path hits the slice boundary is proportional to the number of line segments. It is important to keep this quantity small since it determines the number of times the laser has to decelerate and stop, change directions, and then accelerate; frequent starts and stops are time-consuming and reduce the life of the laser. The number of line segments can be kept small by picking a suitable hatching direction. We define this problem formally in the next section.

1.1 The Hatching Problem and Its Approximation

A slice is a simple polygon $\mathcal{P}$, possibly with holes, in the 2-dimensional plane. Let $\mathbf{d}$ be a unit vector in the plane, and $\ell_0(\mathbf{d})$ the line through the origin with direction $\mathbf{d}$; $\mathbf{d}$ is the *hatching direction*. Let $\mathcal{L}(\mathbf{d})$ be the set of all lines that are parallel to $\ell_0(\mathbf{d})$ and whose distances to $\ell_0(\mathbf{d})$ are multiples of δ, the width of the path. We denote by $\mathcal{S}_\ell$ the set containing the line segments in the intersection between ℓ and $\mathcal{P}$, and define $H(\mathbf{d}) := \sum_{\ell \in \mathcal{L}(\mathbf{d})} |\mathcal{S}_\ell|$. (Fig. 1(b).) The optimization problem can be stated formally as follows:

Problem 1 (Hatching Problem). Given a simple n-vertex polygon $\mathcal{P}$, possibly with holes, compute a hatching direction $\mathbf{d}$ such that $H(\mathbf{d})$ is minimized.

Suppose the width δ of the tool-tip is infinitesimally small. (By "tool" we mean, e.g., the laser in Stereolithography or the nozzle in Fused Deposition Modeling.) Then, given any hatching direction $\mathbf{d}$, the number of times the hatching path runs into an edge e of $\mathcal{P}$ is proportional to the length of e's projection *perpendicular* to $\mathbf{d}$. Thus the solution to the hatching problem can be approximated by finding a direction which minimizes the total length of the projections of the edges of $\mathcal{P}$ onto a line perpendicular to this direction. (Clearly the smaller δ is, the better is the approximation.) This yields the following problem, where, for simplicity, we consider not the edges themselves but their outward normals, each with the same length as its corresponding edge and translated to the origin.

Problem 2 (Projection Problem). Given a finite set $\mathcal{S}$ of n vectors in the plane, each beginning at the origin, find a unit vector $\mathbf{d}$ such that $\sum_{\mathbf{v} \in \mathcal{S}} |\mathbf{v} \cdot \mathbf{d}|$ is minimized.

Note that Problem 2 depends only on the lengths and orientations of the edges of the original polygon, and not on how they connect to each other in the polygon. This suggests that we can find a *globally* optimal hatching direction for *all* the layers by projecting the edges from all layers onto the xy-plane and running our algorithm on the resulting set of vectors.

1.2 Contributions

In Sections 2 and 3 we present two simple and efficient algorithms for Problem 2; this yields an approximation to the optimal hatching direction. For comparison, we also designed an algorithm for Problem 1 which computes an optimal hatching direction; this algorithm is more complex and is described in Section 4. We

establish the performance of the approximation algorithms in two ways: First, we implemented the algorithms of Sections 3 and 4 and tested them on real-world polyhedral models obtained from industry (Section 5). We discovered that the approximation algorithm works very well in practice. Second, we show that, under reasonable assumptions, the number of hatching segments produced by the approximation algorithms is only a constant times more than the number produced by the optimal algorithm (Section 6). In Section 7, we discuss applications of the approximation algorithms to other related problems. For lack of space, we omit many details here; these can be found in [1,5].

2 Minimizing the Projected Length of a Simple Polygon

Recall what we want to accomplish. We are given a simple polygon, from which we get a set $\mathcal{S}$ of outward-pointing normal vectors $\mathbf{n}_e$ for each edge e, with $\mathbf{n}_e$ having the same length as e and beginning at the origin. We want to compute a direction $\mathbf{d}$ that minimizes the sum $\sum_e |\mathbf{n}_e \cdot \mathbf{d}|$. We replace all the vectors in $\mathcal{S}$ that point in the same direction by their sum. We then sort the vectors in circular order and do a circular walk around the origin. We keep an initially empty chain of vectors during our walk. Whenever we encounter a vector in $\mathcal{S}$, we put it onto the chain, with its tail at the head of the old chain.

It is easy to see that the sum of all these normals $\sum_e \mathbf{n}_e$ is *zero*, since our polygon is closed. It follows that we will get a polygon at the end of our circular walk. Moreover, this polygon is convex because the vectors are added in sorted order. Now it is clear that, for any direction $\mathbf{d}$, the sum of the absolute values of the dot products of the vectors in $\mathcal{S}$ w.r.t. $\mathbf{d}$ is twice the width of this convex polygon in the direction perpendicular to $\mathbf{d}$. (Fig. 2). Therefore, finding the minimizing direction in Problem 2 is equivalent to finding the direction that minimizes the width of the convex polygon. Using any of the standard algorithms that compute the smallest width of a convex polygon [2], we have:

Theorem 1. *Given a simple n-vertex polygon $\mathcal{P}$ in the plane, we can compute in $O(n \log n)$ time and using $O(n)$ space a unit vector $\mathbf{d}$ such that the sum $\sum_e |\mathbf{n}_e \cdot \mathbf{d}|$ is minimized.*

As noted in the discussion leading up to Problem 2, the direction $\mathbf{d}$ in Theorem 1 can be used as an approximation to the optimal hatching direction sought in Problem 1. A similar algorithm was discovered independently in [4].

3 An Alternative Algorithm

In this section, we present another approach to Problem 2. This algorithm has the advantage that it works on any set of vectors, not just those corresponding to the edge normals of a simple polygon; moreover, it generalizes easily to higher dimensions.

Consider the set $\mathcal{S}$ of normal vectors in the plane, each with its tail at the origin. We pick an arbitrary vector $\mathbf{d}$ as a candidate direction and draw a line perpendicular to $\mathbf{d}$ through the origin. This line cuts the plane into two half-planes. The normals that lie in the opposite half-plane as $\mathbf{d}$ will register a negative value in their inner products with $\mathbf{d}$. We correct the inner products of these vectors with a minus sign. This corresponds to "reflecting" these vectors through the origin. We replace the downward-pointing vectors (w.r.t. $\mathbf{d}$) with their reflected copies (Fig. 3). We call this new set of vectors $\tilde{\mathcal{S}}$.

All the vectors $\tilde{\mathbf{v}}$ in $\tilde{\mathcal{S}}$ lie in the same closed half-plane as $\mathbf{d}$. Therefore $\sum_{\mathbf{v}\in\mathcal{S}} |\mathbf{v}\cdot\mathbf{d}| = \sum_{\tilde{\mathbf{v}}\in\tilde{\mathcal{S}}} (\tilde{\mathbf{v}}\cdot\mathbf{d}) = (\sum_{\tilde{\mathcal{S}}} \tilde{\mathbf{v}})\cdot\mathbf{d}$. In other words, the sum of all the projection lengths is equal to the inner product of $\mathbf{d}$ with a single vector $\sum_{\tilde{\mathcal{S}}} \tilde{\mathbf{v}}$. If no element of $\tilde{\mathcal{S}}$ is on the cutting line, nothing prevents us from rotating $\mathbf{d}$ away from $\sum_{\tilde{\mathcal{S}}} \tilde{\mathbf{v}}$ and in the process decreasing the inner product it makes with $\sum_{\tilde{\mathcal{S}}} \tilde{\mathbf{v}}$. We can keep doing this until one of the vectors $\tilde{\mathbf{v}}$ is on the cutting line. Now any further movement of $\mathbf{d}$ will cause $\tilde{\mathbf{v}}$ to go to the other side of the cutting line and cause the total projection length to increase. Thus, the position of the cutting line that coincides with one of the input vectors must be a local minimum for the total projected length.

We can update $\sum_{\tilde{\mathcal{S}}} \tilde{\mathbf{v}}$ efficiently if we visit the vectors in a circular order. Specifically, each vector $\tilde{\mathbf{v}}$ has associated with it two regions, separated by the line perpendicular to $\tilde{\mathbf{v}}$. In our walk, whenever we pass this line, we know that the associated vector's contribution to the sum changes sign. If $\tilde{\mathbf{v}}_i$ is the associated vector, we subtract $2\tilde{\mathbf{v}}_i$ from $\sum_{\tilde{\mathcal{S}}} \tilde{\mathbf{v}}$, one copy to take it off from the sum, and another copy to insert it back in with a negative sign. We use the newly updated vector sum to calculate the projection at that event point. Since the update can be done in $O(1)$ time, we get the same result as in Theorem 1.

4 An Exact Algorithm for the Hatching Problem

In this section, we give an outline of our algorithm that solves Problem 1. W.l.o.g., we may assume that no vertex of the polygon $\mathcal{P}$ is at the origin and that no three successive vertices of $\mathcal{P}$ are collinear.

Since $H(\mathbf{d}) = H(-\mathbf{d})$ for any direction $\mathbf{d}$, it suffices to compute an optimal hatching direction $\mathbf{d} = (d_1, d_2)$ for which $d_2 \geq 0$. The idea of our algorithm is as follows. We start with an initial direction $\mathbf{d} = (-1, 0)$, and rotate it in clockwise order by an angle of π until $\mathbf{d} = (1, 0)$. At certain directions $\mathbf{d}$, the value of $H(\mathbf{d})$ changes. We will call such directions *critical*. During the rotation, we update the value of $H(\mathbf{d})$ at each such critical direction.

During the rotation, the collection $\mathcal{L}(\mathbf{d})$ rotates, with the origin being the center of rotation. We give necessary conditions for a direction $\mathbf{d}$ to be critical. There are two types of directions $\mathbf{d}$, for which $H(\mathbf{d})$ changes.

Type 1: The subset of lines in $\mathcal{L}(\mathbf{d})$ that intersect the polygon $\mathcal{P}$ changes.

We analyze when this can happen. Let $CH(\mathcal{P})$ be the convex hull of $\mathcal{P}$. Note that any line intersects $\mathcal{P}$ if and only if it intersects $CH(\mathcal{P})$. Let $\mathbf{d}$ be a direction at which the subset of $\mathcal{L}(\mathbf{d})$ that intersects $\mathcal{P}$ changes. Let $\mathbf{d}^{\perp}$ be a direction

that is orthogonal to $\mathbf{d}$. Then there must be a vertex v on $CH(\mathcal{P})$ such that: (i) v is extreme in one of the directions $\mathbf{d}^\perp$ and $-\mathbf{d}^\perp$, and (ii) v lies on a line of $\mathcal{L}(\mathbf{d})$, i.e., the distance between v and the line $\ell_0(\mathbf{d})$ through the origin having direction $\mathbf{d}$, is a multiple of δ.

Type 2: For some line $\ell \in \mathcal{L}(\mathbf{d})$, the set $\mathcal{S}_\ell$ of line segments (of positive length) in the intersection $\ell \cap \mathcal{P}$ changes.

If this happens, then there is a vertex v of $\mathcal{P}$ such that: (i) v lies on a line of $\mathcal{L}(\mathbf{d})$, i.e., the distance between v and the line $\ell_0(\mathbf{d})$ is a multiple of δ, and (ii) both vertices of $\mathcal{P}$ that are adjacent to v are on the same side of the line $\ell_v(\mathbf{d})$ through v that is parallel to $\ell_0(\mathbf{d})$. (We have to be careful with degenerate cases.)

Let $\mathbf{D}$ be the set of all directions $\mathbf{d}$ for which there is a vertex v of $\mathcal{P}$ whose distance to the line $\ell_0(\mathbf{d})$ is a multiple of δ. It follows from above that $\mathbf{D}$ contains all critical directions. We now give a brief overview of the algorithm.

Step 1: For each vertex v of $\mathcal{P}$, compute all directions $\mathbf{d} = (d_1, d_2)$ for which $d_2 \geq 0$, and for which the distance between v and the line $\ell_0(\mathbf{d})$ is a multiple of δ. Let $\mathbf{D}$ be the resulting set of directions.

A simple geometric analysis shows that this step can be reduced to solving $2(1 + \|v\|/\delta)$ quadratic equations for each vertex v of $\mathcal{P}$. Hence, the time for Step 1 is $O(|\mathbf{D}|)$, where $|\mathbf{D}| \leq 2n(1 + \max_v \|v\|/\delta)$.

Step 2: Sort the directions of $\mathbf{D}$ in the order in which they are visited when we rotate the unit-vector $(-1, 0)$ by an angle of π in clockwise order. We denote this ordering relation by $\prec$. The time for this step is $O(|\mathbf{D}| \log |\mathbf{D}|)$.

Let m be the number of distinct directions in the set $\mathbf{D}$. We denote the sorted elements of $\mathbf{D}$ by $\mathbf{d}^0 \prec \mathbf{d}^1 \prec \ldots \prec \mathbf{d}^{m-1}$. Note that for any i and any two directions $\mathbf{d}$ and $\mathbf{d}'$ strictly between $\mathbf{d}^i$ and $\mathbf{d}^{i+1}$, we have $H(\mathbf{d}) = H(\mathbf{d}')$.

Step 3: Let $\mathbf{d}_s$ be a direction that is not in $\mathbf{D}$. Compute $H(\mathbf{d}_s)$ for this direction.

Recall that $H(\mathbf{d}_s)$ is the number of line segments of positive length in the intersection of $\mathcal{P}$ with $\mathcal{L}(\mathbf{d}_s)$. The endpoints of any such line segment are on the boundary of $\mathcal{P}$. Hence, the total number of intersection points between $\mathcal{P}$ and the lines in $\mathcal{L}(\mathbf{d}_s)$ is twice $H(\mathbf{d}_s)$. For any edge $e = (u, v)$ of $\mathcal{P}$, let I_e be the number of lines in $\mathcal{L}(\mathbf{d}_s)$ that intersect e. Then $I_e = \left| \left\lfloor \frac{\mathbf{v} \cdot (\mathbf{d}_s)^\perp}{\delta} \right\rfloor - \left\lfloor \frac{\mathbf{u} \cdot (\mathbf{d}_s)^\perp}{\delta} \right\rfloor \right|$, where $(\mathbf{d}_s)^\perp$ is the direction orthogonal to $\mathbf{d}_s$ and to the left of $\mathbf{d}_s$.

Hence, we can implement this step, by computing $H(\mathbf{d}_s)$ as $(1/2) \sum_e I_e$. This takes $O(n)$ time.

Step 4: Let k be the index such that $\mathbf{d}^{k-1} \prec \mathbf{d}_s \prec \mathbf{d}^k$. Walk along the elements of $\mathbf{D}$ in the order $\mathbf{d}^k, \mathbf{d}^{k+1}, \ldots, \mathbf{d}^{m-1}, \mathbf{d}^0, \ldots, \mathbf{d}^{k-1}$. At each direction $\mathbf{d}^i$, we first compute $H(\mathbf{d}^i)$ from $H(\mathbf{d})$ for $\mathbf{d}^{i-1} \prec \mathbf{d} \prec \mathbf{d}^i$, and then compute $H(\mathbf{d})$ from $H(\mathbf{d}^i)$ for $\mathbf{d}^i \prec \mathbf{d} \prec \mathbf{d}^{i+1}$.

We give some details about this step in Section 4.1. For each direction $\mathbf{d}^i \in \mathbf{D}$, we spend $O(1)$ time to update $H(\mathbf{d})$, so the overall time for Step 4 is $O(|\mathbf{D}|)$.

Step 5: Report the minimum value of $H(\mathbf{d})$ found in Step 4, together with the corresponding optimal hatching direction(s) $\mathbf{d}$.

Theorem 2. *Given a simple polygon $\mathcal{P}$, possibly with holes, having n vertices, Problem 1 can be solved in $O(Cn \log(Cn))$ time, where $C = 1 + \max_v \|v\|/\delta$.*

4.1 Step 4

Let $\mathbf{d}_0$ be any direction of $\mathbf{D}$. We analyze how $H(\mathbf{d})$ changes, if $\mathbf{d}$ rotates in clockwise order, and "passes" through $\mathbf{d}_0$. We denote by $\mathbf{d}_{-\epsilon}$ (resp. $\mathbf{d}_{\epsilon}$) the direction obtained by rotating $\mathbf{d}_0$ by an infinitesimally small angle in counterclockwise (resp. clockwise) direction. Hence, $\mathbf{d}_{-\epsilon}$ (resp. $\mathbf{d}_{\epsilon}$) is the direction $\mathbf{d}$ immediately before it reaches (resp. immediately after it leaves) $\mathbf{d}_0$.

Let v be any vertex of $\mathcal{P}$ that corresponds to $\mathbf{d}_0$, i.e., $d(v, \ell_0(\mathbf{d}_0))$ is a multiple of δ. Let v_p and v_s be the predecessor and successor vertices of v, respectively. Note that the interior of $\mathcal{P}$ is to the left of the directed edges (v_p, v) and (v, v_s). There are two cases, one of which we describe here.

Assume that the points v, $v + d_0$, and v_p or the points v, $v + d_0$, and v_s are collinear. Hence, we have two adjacent vertices, whose (signed) distances to the line $\ell_0(\mathbf{d}_0)$ are equal to the same multiple of δ. We rename these vertices as u and v, and assume w.l.o.g. that the triple $(u, u + d_0^{\perp}, v)$ forms a right-turn. Let u' be the vertex of $\mathcal{P}$ that is adjacent to u and for which $u' \neq v$. Similarly, let v' be the vertex that is adjacent to v and for which $v' \neq u$.

When $\mathbf{d}$ passes through $\mathbf{d}_0$, there are fifty six cases. We consider one of these cases; for the other cases, we refer to [5]. As in Figure 4, assume that (1) $(0, d_0^{\perp}, u)$ forms a right-turn, (2) $(0, d_0^{\perp}, v)$ forms a right-turn, (3) $(u, u + d_0, u')$ forms a left-turn, (4) $(v, v + d_0, v')$ forms a left-turn, and (5) v is the successor of u. (Recall that we assume that $(u, u + d_0^{\perp}, v)$ forms a right-turn.)

We argue that $H(\mathbf{d}_0) = H(\mathbf{d}_{-\epsilon})$, and $H(\mathbf{d}_{\epsilon}) = H(\mathbf{d}_0) - 1$, as follows: Let j be the integer such that $d(u, \ell_0(\mathbf{d}_0)) = d(v, \ell_0(\mathbf{d}_0)) = j\delta$. For any direction $\mathbf{d}$, let $\ell_j(\mathbf{d})$ be the line having direction $\mathbf{d}$ and whose distance to $\ell_0(\mathbf{d})$ is equal to $j\delta$. (Figure 4.) Consider what happens if $\mathbf{d}$ rotates in clockwise order, and passes through $\mathbf{d}_0$. For direction $\mathbf{d}_{-\epsilon}$, the intersection of line $\ell_j(\mathbf{d}_{-\epsilon})$ with $\mathcal{P}$ contains a line segment L, whose endpoints are in the interiors of the edges (u', u) and (v, v'). For direction $\mathbf{d}_0$, the intersection of line $\ell_j(\mathbf{d}_0)$ with $\mathcal{P}$ contains the edge (u, v). If we rotate the direction from $\mathbf{d}_{-\epsilon}$ to $\mathbf{d}_0$, then L "moves" to the edge (u, v). Hence, we indeed have $H(\mathbf{d}_0) = H(\mathbf{d}_{-\epsilon})$. For direction $\mathbf{d}_{\epsilon}$, edge (u, v) does not contribute any line segment to the intersection of line $\ell_j(\mathbf{d}_{\epsilon})$ with $\mathcal{P}$. Therefore, we have $H(\mathbf{d}_{\epsilon}) = H(\mathbf{d}_0) - 1$.

5 Experimental Results

We implemented the 2-dimensional algorithm of Section 3 in C++, and tested it on slices generated from real-world polyhedral models obtained from Stratasys, Inc., a Minnesota-based LM company. We generated the slices using Stratasys' `QuickSlice` program. Figure 5 (top row) displays some of our results.

We also implemented the idea discussed at the end of Section 1.1 to compute a globally optimal direction for all slices. Figure 5 (bottom row) displays some of our results, as viewed in projection in the positive z-direction. (We used a layer thickness of 0.01 inches.) Additional results for both experiments are in [1].

We remark that the approximation algorithms work on polygons with holes in exactly the same way as they do on polygons without holes. In fact, the

algorithms only need the orientation and lengths of the edges; they do not use any information about the adjacency of the edges.

We also implemented the exact algorithm from Section 4. In a separate set of experiments, reported in detail in [5], we tested the exact and approximation algorithms on several additional test files, using now a Sun Ultra with a 400 MHz CPU and 512 MB of RAM. (We ran the algorithms only on single layers, not all layers.) The approximation algorithm generated at most fourteen percent more hatching segments than the exact algorithm. The running time of the exact algorithm ranged from 38 seconds (on a 32-vertex polygon) to 2485 seconds (890 vertices); the approximation algorithm never took more than 1 second.

6 Analysis of the Approximation Algorithm

Our experimental results suggest that the approximation algorithm does well in practice. To further understand its behavior, we also analysed it theoretically.

Let $\delta > 0$ be the width of the tool-tip and n the number of vertices in the polygon $\mathcal{P}$. For any direction $\mathbf{d}$, let $Proj(\mathbf{d}^{\perp})$ be the length of the projection of the edges of $\mathcal{P}$ perpendicular to $\mathbf{d}$, and let $Cut(\mathbf{d})$ be the number of times the boundary of $\mathcal{P}$ is cut when hatched in direction $\mathbf{d}$. Let $\mathbf{d}_p$ and $\mathbf{d}_c$ be the directions minimizing $Proj(\mathbf{d}^{\perp})$ and $Cut(\mathbf{d})$, respectively; $\mathbf{d}_p$ is the direction computed by the approximation algorithm.

In [1], we prove that $Cut(\mathbf{d}_p) - Cut(\mathbf{d}_c) < 3n + (Proj(\mathbf{d}_p^{\perp}) - Proj(\mathbf{d}_c^{\perp}))/\delta$. Since $Proj(\mathbf{d}_p^{\perp}) - Proj(\mathbf{d}_c^{\perp}) \leq 0$, we have that $Cut(\mathbf{d}_p) - Cut(\mathbf{d}_c) < 3n$, or $Cut(\mathbf{d}_p)/Cut(\mathbf{d}_c) < 1 + 3n/Cut(\mathbf{d}_c)$.

If the number of cuts is too small, features will be lost in the model. It is reasonable to assume that $Cut(\mathbf{d}_c) \geq kn$, where $k \geq 1$. This is true if, e.g., many edges of the polygon are cut at least k times. We then have $Cut(\mathbf{d}_p)/Cut(\mathbf{d}_c) < 1 + 3/k$. Furthermore, if in directions $\mathbf{d}_p$ and $\mathbf{d}_c$, each edge is cut in its interior only, then $Cut(\mathbf{d}_c)$ is twice the minimum number of hatching segments and $Cut(\mathbf{d}_p)$ is twice the number of the hatching segments generated by the approximation algorithm. This yields an approximation ratio of $1 + 3/k$.

7 Other Applications

Our methods can solve several related problems efficiently (see [1]):

To improve part strength it is desirable to hatch each slice along two non-parallel directions [3]. This yields the following problem: Given a simple n-vertex polygon $\mathcal{P}$, possibly with holes, and a fixed angle θ, $0 < \theta \leq 90°$, find a pair of directions $(\mathbf{d}, \mathbf{d}')$ that make an angle θ with each other such that the total number of hatching segments for $\mathcal{P}$ in these two directions is minimized. This problem can be converted to a form where the algorithm of Section 2 or Section 3 can be applied, and can be solved in $O(n \log n)$ time and $O(n)$ space.

Suppose that we wish to protect certain functionally critical edges of the slice from being hit too often during hatching. We can assign weights to edges in

proportion to their importance. This leads to a weighted version of Problem 2, which we can solve in $O(n \log n)$ time and $O(n)$ space.

When a polygonal slice is built via LM, certain edges will have a stair-step appearance due to the discretization introduced by the tool-tip width (similar to anti-aliasing in computer graphics). We quantify the error in terms of the total height of the stair-steps on all edges and show how our methods can be used to minimize the total error, again in $O(n \log n)$ time and $O(n)$ space.

We generalize Problem 2 to vectors in $k > 2$ dimensions and present two algorithms: one runs in $O(n^{k-1} \log n)$ time and $O(n)$ space, and the other in $O(n^{k-1})$ time and space. We also present experimental results for $k = 3$, using as input the facet normals of our models.

References

1. M. Hon, R. Janardan, J. Schwerdt, and M. Smid. Minimizing the total projection of a set of vectors, with applications to Layered Manufacturing. Manuscript, January 2001. `http://www.cs.umn.edu/~janardan/min-proj.ps`.
2. M. E. Houle and G. T. Toussaint. Computing the width of a set. *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-10(5):761 765, 1988.
3. P. Jacobs. *Rapid Prototyping & Manufacturing: Fundamentals of Stereolithography.* McGraw-Hill, 1992.
4. S. E. Sarma. The crossing function and its application to zig-zag tool paths. *Comput. Aided Design*, 31:881 890, 1999.
5. J. Schwerdt, M. Smid, M. Hon, and R. Janardan. Computing an optimal hatching direction in Layered Manufacturing. Manuscript, January 2001. `http://isgwww.cs.uni-magdeburg.de/~michiel/hatching.ps.gz`.

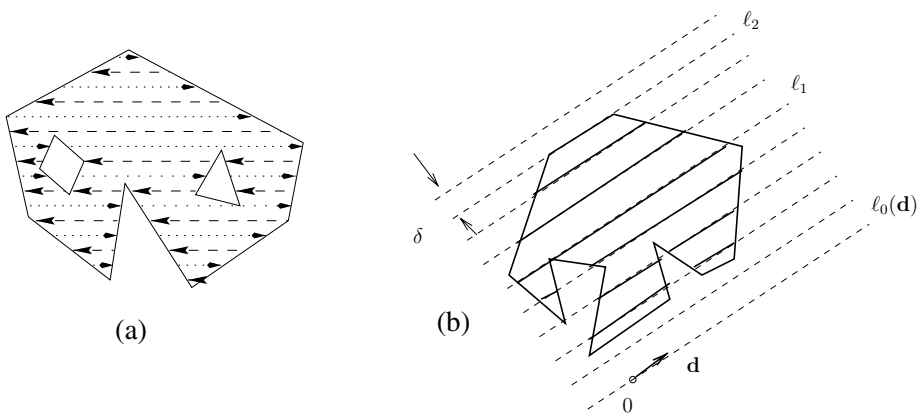

Fig. 1. (a) Hatching a polygonal slice. (b) Formal de nition for hatching problem. Here $H(\mathbf{d}) = 10$. Note that lines ℓ_1 and ℓ_2 each contribute one segment.

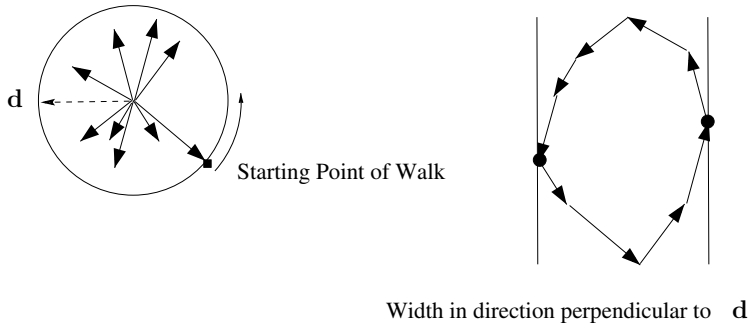

Fig. 2. A set of vectors and the resulting convex polygon. The sum of the absolute values of the dot products of the vectors w.r.t. direction **d** is twice the width of the convex polygon in the direction perpendicular to **d**.

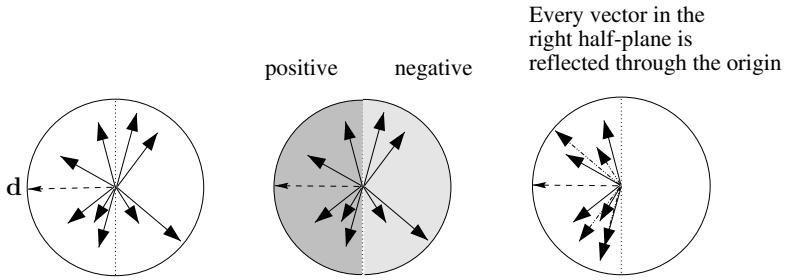

Fig. 3. As an initial step, we pick an arbitrary candidate direction **d** and make sure every vector falls in its positive half-plane. In this figure, the candidate direction is the negative x direction.

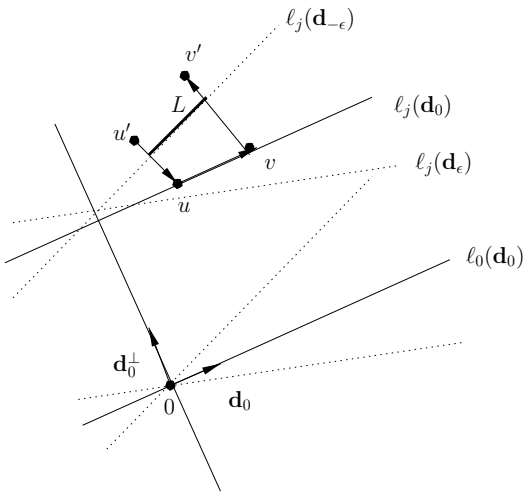

Fig. 4. Illustrating Step 4 in Section 4.1.

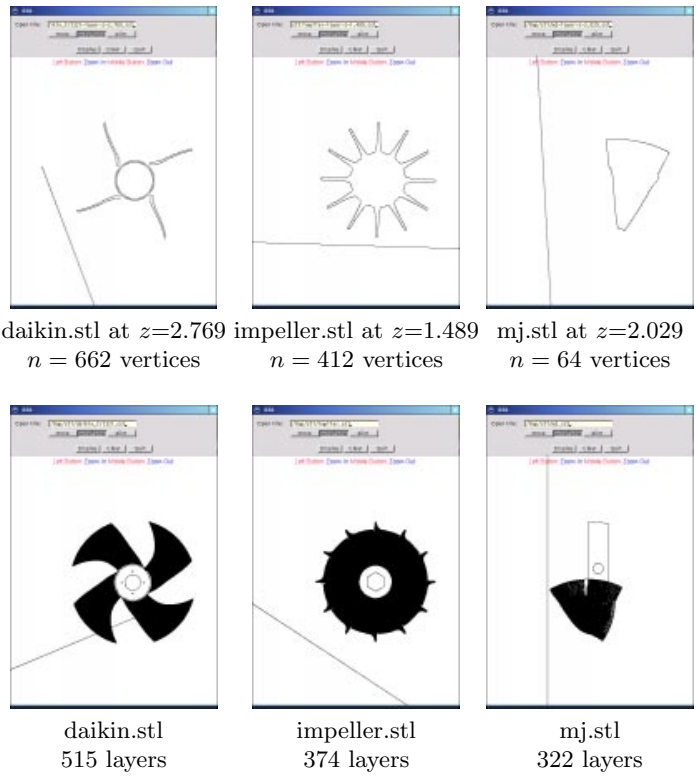

Fig. 5. Screen shots of the program running on a single layer (top row) and all layers (bottom row) of different models. (The z value in the top row shows the height of the layer above the platform.) The long lines inside each window is the resulting hatching direction, which minimizes the sum of the lengths of the projections of the edges onto a perpendicular line. For each model, the running time for a single layer was less than 0.01 seconds and for all layers was less than 2 seconds, on a Sun UltraSparcIIi workstation with a 440 MHz CPU and 256 MB of RAM.