Minimizing the total projection of a set of vectors, with applications to Layered Manufacturing

Man Chung Hon* Ravi Janardan*† Jörg Schwerdt‡§
Michiel Smid‡§

February 1, 2001

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 some small non-zero width), in a process called hatching. A critical step in hatching is choosing a direction which minimizes the number of segments. In this paper, this problem is approximated as the problem of finding a direction which minimizes the total projected length of a certain set of vectors. Efficient algorithms are proposed for the latter problem, using techniques from computational geometry. Experimental and theoretical analyses show that this approach yields results that approximate closely the optimal solution to the hatching problem. Extensions of these results to several related problems are also discussed.

1 Introduction

This paper addresses a geometric problem motivated by *Layered Manufacturing* (LM), which is an emerging technology that allows the construction of

*Department of Computer Science & Engineering, University of Minnesota, Minneapolis, MN 55455, U.S.A. {hon,janardan}@cs.umn.edu . Research supported, in part, by NSF grant CCR–9712226.

†Portions of this work were done when RJ visited the University of Magdeburg, Germany under a joint grant from NSF and DAAD for international research.

‡Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, D-39106 Magdeburg, Germany. {schwerdt,michiel}@isg.cs.uni-magdeburg.de .

§Portions of this work were done when MS and JS visited the University of Minnesota under a joint grant from DAAD and NSF for international research.

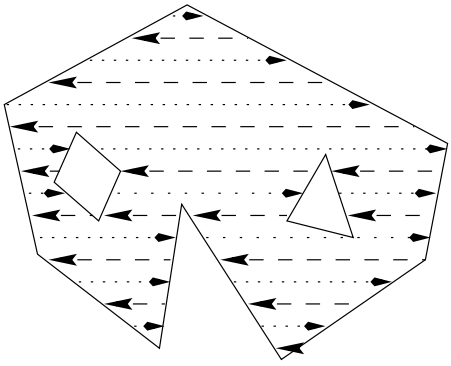


Figure 1: Hatching a polygonal slice

---

physical prototypes of three-dimensional parts directly from their computer representations, using a "3D printer" attached to a personal computer. LM provides an additional level of physical verification by allowing the designer to "feel and touch" the model, and makes possible the detection of design flaws that may have otherwise gone unnoticed. It is used extensively in the automotive, aerospace, and medical industries, among others [5].

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* [5]. 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); this process is called *hatching*. Another process called *Fused Deposition Modeling* hatches the slices by depositing fine strands of molten plastic via a nozzle [6].

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 path hits the slice boundary is proportional to the number of line segments. It is important to keep this quantity small since it determines how often the laser has to decelerate and stop, change direction, 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

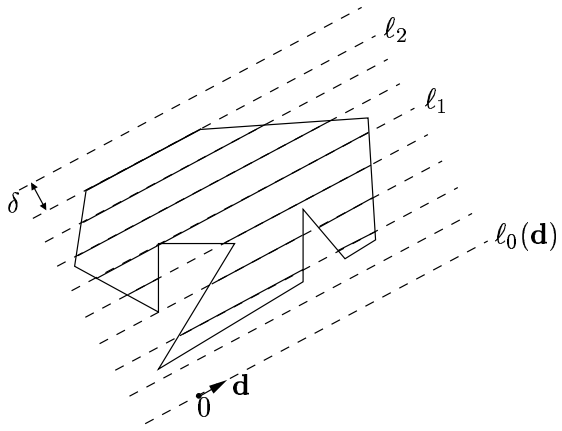


Figure 2: $H(\mathbf{d}) = 10$. Notice that both lines ℓ_1 and ℓ_2 contribute 1 segment.

---

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 δ be the width of the "tool-tip" (e.g., the laser in Stereolithography or the nozzle in Fused Deposition Modeling). Let $\mathbf{d}$ be a unit vector in the plane, and $\ell_0(\mathbf{d})$ be 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 intersection of any line ℓ in $\mathcal{L}(\mathbf{d})$ with polygon $\mathcal{P}$ is either empty, or consists of a collection of points and/or disjoint line segments of non-zero length. We denote by $\mathcal{S}_\ell$ the set consisting of the line segments in the intersection between ℓ and $\mathcal{P}$. Each segment in $\mathcal{S}_\ell$ is a *hatching segment* and ℓ is the *centerline* of the segment. We define $H(\mathbf{d})$ to be the sum of the cardinalities of the sets $\mathcal{S}_\ell$, taken over all the lines ℓ in $\mathcal{L}(\mathbf{d})$ (Fig. 2). That is,

$$H(\mathbf{d}) = \sum_{\ell \in \mathcal{L}(\mathbf{d})} |\mathcal{S}_\ell| \,.$$

The optimization problem can be stated as follows:

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

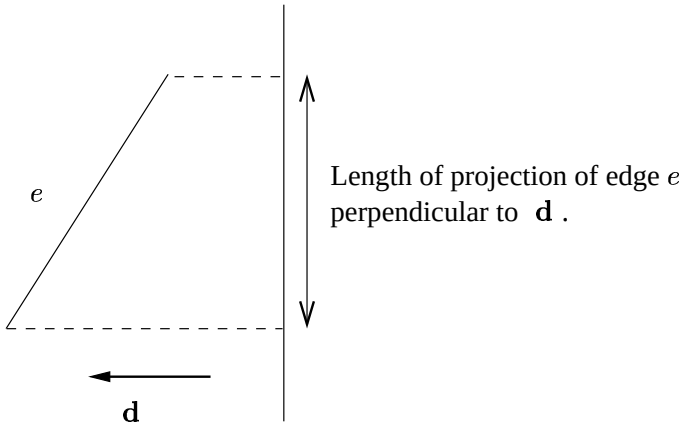


Figure 3: **d** is the hatching direction. The number of times a tool with an infinitesimally small tip runs into an edge e is proportional to the length of e's projection perpendicular to **d**.

Suppose that the width δ of the tool-tip is infinitesimally small. Then, given any hatching direction **d**, the number of times a centerline runs into an edge e of $\mathcal{P}$ is proportional to the length of e's projection perpendicular to **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. (Fig. 3.) Clearly the smaller δ is, the better is the approximation.

In this setting, the length of the projection of an edge is equal to the absolute value of the dot product of the outer normal of the edge with **d**, where the normal has the same length as the edge and is translated to the origin. (Fig. 4.) It will be convenient to work with these normals rather than the edges of $\mathcal{P}$. The use of normals also facilitates the generalization of our discussion later to higher dimensions. Therefore, as an approximation to the original hatching problem, we have:

Problem 2 (Minimum projection) *Let $\mathcal{P}$ be an n-vertex simple polygon (possibly with holes). Let $\mathbf{n}_e$ be the outer normal of each edge e of $\mathcal{P}$, where $\mathbf{n}_e$ has the same length as e and begins at the origin. Let $\mathcal{S}$ be the set of these outer normals. Find a unit vector* **d** *such that $\sum_{\mathbf{n}_e \in \mathcal{S}} |\mathbf{n}_e \cdot \mathbf{d}|$ is minimized.*

Note that Problem 2 depends only on the lengths and orientations of the edges of the original polygon, and not how they connect to each other in the polygon. This implies that, if we have an algorithm for Problem 2, then

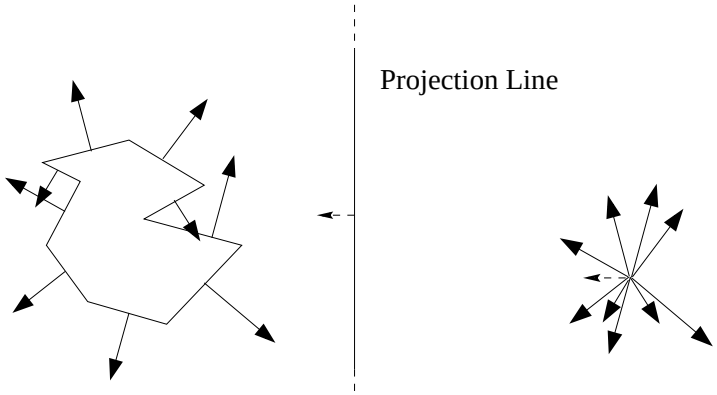


Figure 4: A simple polygon and the corresponding set of outer normals translated to the origin.

---

we can also use it to find a *globally* optimal hatching direction for *all* the layers: we project onto the plane the normals corresponding to the edges from all layers and run our algorithm on them.

1.2 Contributions

We present two algorithms to solve Problem 2 in $O(n \log n)$ time and $O(n)$ space. The first algorithm (Section 2) constructs a new convex polygon whose edges consist of the normals of $\mathcal{S}$. The direction sought in Problem 2 turns out to be the direction which minimizes the width of this convex polygon, which can be found efficiently. The second algorithm (Section 3) sidesteps explicit construction of the convex polygon. Instead it identifies n candidates for the optimal direction, examines them in turn, and computes the total projection for each using an efficient incremental update scheme. 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; we will use this fact in Section 6. Moreover, this algorithm can be generalized easily to higher dimensions.

Our algorithms for Problem 2 yield an approximate solution to Problem 1. We investigate the quality of this approximation both theoretically and experimentally. Our theoretical analysis, presented in Section 4, shows that, under reasonable assumptions, the number of hatching segments produced by the approximation algorithm is only a constant times more than the minimum number of hatching segments. We have also implemented the

algorithm of Section 3 and experimented with it on real-world LM models obtained from industry. These results, presented in Section 5, show that our algorithm is significantly faster than an exact algorithm [11] for Problem 1, and yet generates only a slightly larger number of hatching segments.

As discussed in Section 6, our algorithm for Problem 2 also yields efficient solutions to several related problems in LM. These include approximation algorithms for (i) a weighted version of Problem 1, (ii) optimal hatching along two directions, and (iii) hatching to minimize the so-called stair-step error in a slice (similar to antialiasing in computer graphics). We also discuss, in Section 7, the generalization and solution of Problem 2 in higher dimensions.

1.3 Related Work

To evaluate the performance of our approximation algorithm, we have designed and implemented an algorithm which solves Problem 1 exactly; this work is reported in a companion paper [11]. In essence, this algorithm works by performing a rotational sweep of the polygon and dynamically maintaining the value of $H(\mathbf{d})$ during the sweep. Although conceptually simple, the algorithm involves extensive case analysis. (To keep the present paper to a reasonable length, we omit a detailed discussion of the exact algorithm here and refer the reader to [11]. However, a brief comparison of the performance of the exact and approximation algorithms is given in Section 5.)

We note that an algorithm similar to our first algorithm in Section 2 was discovered independently by Sarma [10] in the context of planning an optimal path for milling machines.

2 Minimizing the total projected length of a simple polygon

Recall that our goal is to solve Problem 2. We first replace the vectors in $\mathcal{S}$ that point in the same direction by their sum. We then sort the vectors in circular order around the origin and walk through this list. During the walk we maintain a chain of vectors; we initialize this chain to empty at the beginning of the walk. When we encounter a vector in $\mathcal{S}$ during the walk, we put it onto the chain, with its tail at the head of the old chain (Fig. 5).

It is easy to see that $\sum_e \mathbf{n}_e$ is zero, since $\mathcal{P}$ is closed. Indeed, we can "vectorize" each edge of $\mathcal{P}$ by tracing its boundary so that the interior of $\mathcal{P}$ is always to the left of the boundary. Each edge e is assigned a vector, $\mathbf{v}_e$,

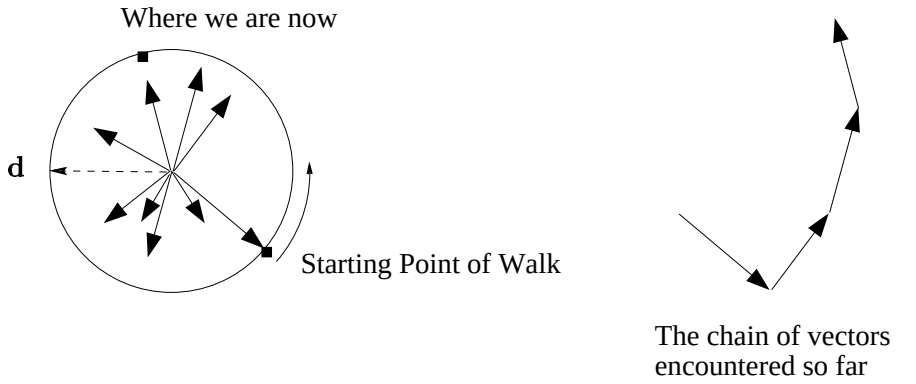


Figure 5: Doing a circular walk to construct a convex polygon.

---

whose direction is parallel to e, and whose length is the same as that of e. The outward normal $\mathbf{n}_e$ of e is equal to $T(\mathbf{v}_e)$, where T is the linear operator which turns vectors 90 degrees clockwise; i.e., $T = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$. Therefore, $\sum_e \mathbf{n}_e = \sum_e T(\mathbf{v}_e) = T(\sum_e \mathbf{v}_e)$ since T is linear. Since $\mathcal{P}$ is closed, we have $\sum_e \mathbf{v}_e = 0$. (That is, we always return to where we started when we trace out the boundary of a closed polygon.) This implies that $\sum_e \mathbf{n}_e = T(0) = 0$.

Since the vectors in the set $\mathcal{S}$ sum to zero, we will get a polygon, $\mathcal{Q}$, at the end of our walk. Moreover, $\mathcal{Q}$ is convex because we visited the vectors in the sorted order of the slopes of the corresponding edges of $\mathcal{P}$. (This approach was discovered independently in [10].)

Now, for any $\mathbf{n}_e \in \mathcal{S}$, $|\mathbf{n}_e \cdot \mathbf{d}|$ is the length of $\mathbf{n}_e$ projected in the direction perpendicular to $\mathbf{d}$. Let H and L be the two extreme vertices of $\mathcal{Q}$ in direction $\mathbf{d}$. These vertices partition the boundary of $\mathcal{Q}$ into two chains (Fig. 6). Note that when Chain 1 is projected in the direction perpendicular to $\mathbf{d}$, no two of its edges overlap, except at their endpoints; similarly for Chain 2. Consider the lines through H and L that are perpendicular to $\mathbf{d}$ and enclose $\mathcal{Q}$; the distance between these lines is called the *width* of $\mathcal{Q}$ perpendicular to $\mathbf{d}$.

From the preceding discussion, it follows that $\sum_e |\mathbf{n}_e \cdot \mathbf{d}|$ is just twice the width of $\mathcal{Q}$ in the direction perpendicular to $\mathbf{d}$, for any direction $\mathbf{d}$. Therefore, the minimizing direction in Problem 2 can be found by determining the direction that minimizes the width of $\mathcal{Q}$. We can compute the minimum width of $\mathcal{Q}$ by using the algorithm given in [4, 12], which takes $O(n \log n)$ time and uses $O(n)$ space.

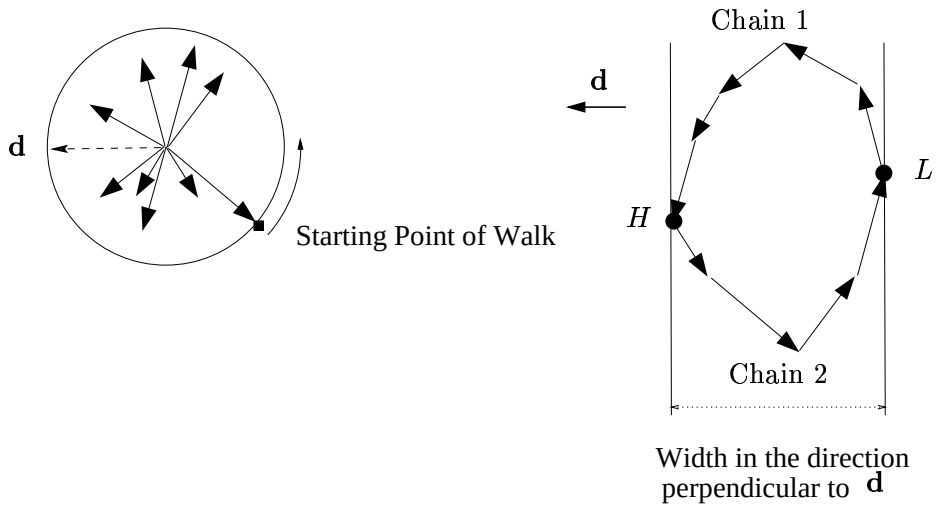


Figure 6: A set of vectors and the resulting convex polygon. The sum of the absolute values of the dot products of the vectors with respect to direction **d** is twice the width of the convex polygon in the direction perpendicular to **d**.

Theorem 1 *Problem 2 can be solved in $O(n \log n)$ time using $O(n)$ space.*

As noted in the discussion leading up to Problem 2 in Section 1.1, the direction **d** in Theorem 1 can be used as an approximation to the optimal hatching direction sought in Problem 1.

3 An alternative algorithm

The algorithm in Section 2 is efficient, but it is very difficult to generalize to higher dimensions. In the plane, we construct a convex polygon with edge orientations and lengths specified by the original polygon. In higher dimensions, a similar approach would require the construction of a convex polytope with prescribed facet orientations and areas. Such a polytope always exists by a beautiful theorem of H. Minkowski [2]. However, we are not aware of any combinatorial algorithm that explicitly constructs such a polytope efficiently.

In this section, we present another approach to Problem 2, which does not require the construction of the convex polygon. It has the advantage that it can be generalized easily to arbitrary dimensions, as we will see in Section 7. Moreover, it also works for more general sets of vectors than

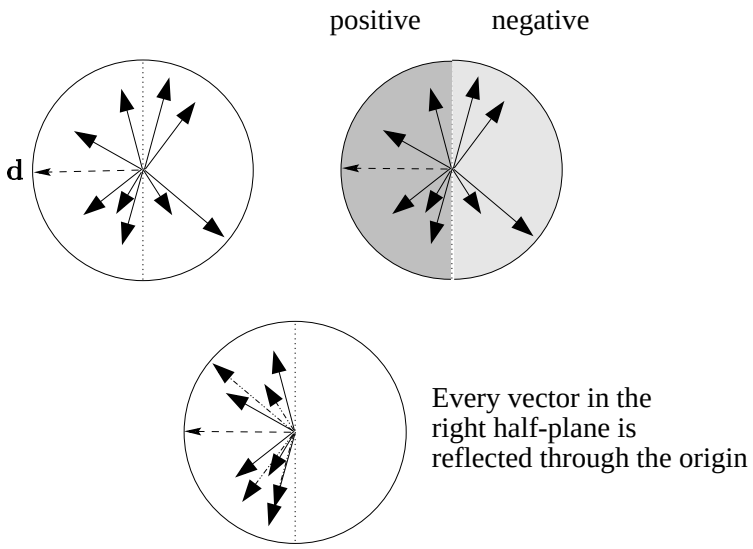


Figure 7: Picking a candidate direction, $\mathbf{d}$, and reflecting vectors.

---

those derived from the edge normals of a simple polygon—a fact we will use in Section 6.

Therefore, in this section, we will assume that $\mathcal{S}$ is a set of n arbitrary vectors in the plane, where each vector begins at the origin. We wish to compute a direction $\mathbf{d}$ such that $\sum_{\mathbf{v}\in\mathcal{S}} |\mathbf{v}\cdot\mathbf{d}|$ is minimized. We pick an arbitrary unit 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 vectors $\mathbf{v}\in\mathcal{S}$ that point "upward" (i.e., those that are in the same closed half-plane as $\mathbf{d}$) generate a non-negative dot product with $\mathbf{d}$. However, those pointing "downward" (i.e. those lying in the complement of the above half-plane), generate a negative dot product with $\mathbf{d}$. Since we are interested in the sum of the absolute values of the dot products, we must correct the dot products of the latter vectors with a minus sign. This corresponds to reflecting these vectors through the origin. Therefore, we replace the downward-pointing vectors with their reflected copies (Fig. 7). 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 $\sum_{\mathbf{v}\in\mathcal{S}} |\mathbf{v}\cdot\mathbf{d}|$ reduces to $\sum_{\tilde{\mathbf{v}}\in\tilde{\mathcal{S}}} (\tilde{\mathbf{v}}\cdot\mathbf{d})$. (Note that we no longer need to use absolute value signs in the sum.) Furthermore, $\sum_{\tilde{\mathbf{v}}\in\tilde{\mathcal{S}}}(\tilde{\mathbf{v}}\cdot\mathbf{d}) = \left(\sum_{\tilde{\mathbf{v}}\in\tilde{\mathcal{S}}} \tilde{\mathbf{v}}\right)\cdot\mathbf{d}$. In other words, we sum the vectors in $\tilde{\mathcal{S}}$ and take the dot product of the

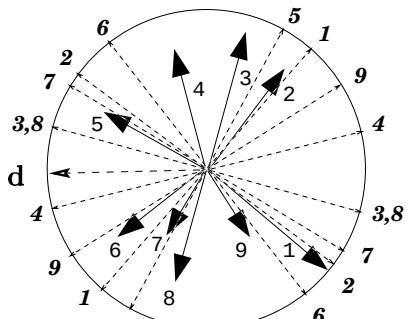


Figure 8: Vectors and their associated perpendicular cutting lines. Crossing the point 5, where the cutting line of vector 5 cuts the unit circle, requires that we flip vector 5 from its current direction to the opposite one.

---

resulting vector, $\sum_{\tilde{\mathbf{v}} \in \tilde{\mathcal{S}}} \tilde{\mathbf{v}}$, with $\mathbf{d}$. If no vector of $\tilde{\mathcal{S}}$ is on the cutting line, then nothing prevents us from rotating $\mathbf{d}$ away from $\sum_{\tilde{\mathbf{v}} \in \tilde{\mathcal{S}}} \tilde{\mathbf{v}}$, thereby decreasing the dot product $\sum_{\tilde{\mathbf{v}} \in \mathcal{S}} \tilde{\mathbf{v}}$ makes with $\mathbf{d}$. We can keep doing this until one of the vectors $\tilde{\mathbf{v}}$ is on the cutting line perpendicular to $\mathbf{d}$. Now any further movement of $\mathbf{d}$ will cause $\tilde{\mathbf{v}}$ to go to the other side of the cutting line. Thereafter, the contribution of (the reflection of) $\tilde{\mathbf{v}}$ will cause the total dot product 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 dot product.

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 $\tilde{\mathbf{v}}$ (Fig. 8). 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. (With this approach, it is sufficient to walk around half the circle, from $\mathbf{d}$ to $-\mathbf{d}$.) At each event, we use the newly updated vector sum to re-calculate the total dot product. Since the update can be done in $O(1)$ time, we can find the minimum in $O(n)$ time given the circular list of vectors. The total running time is dominated by the time to prepare the list itself, in this case the $O(n \log n)$ time for sorting. The space requirement is $O(n)$. This yields the following result, which generalizes Theorem 1.

Theorem 2 *Let $\mathcal{S}$ be a set of n arbitrary vectors in the plane, where each vector begins at the origin. A direction $\mathbf{d}$ which minimizes $\sum_{\mathbf{v} \in \mathcal{S}} |\mathbf{v} \cdot \mathbf{d}|$ can*

be computed in $O(n \log n)$ time using $O(n)$ space.

4 Analyzing the quality of the approximation

We investigate how well our solutions to Problem 2 approximate the solution to Problem 1.

As before, let n be the number of edges in $\mathcal{P}$, e any edge of $\mathcal{P}$, $\mathbf{d}$ any direction (unit vector) in the plane, and $\delta > 0$ the width of each hatching segment (i.e., the distance between adjacent centerlines). Additionally, we define the following:

$Proj_e(\mathbf{d}^\perp)$	:	the length of the projection of e onto the line perpendicular to $\mathbf{d}$,
$Proj(\mathbf{d}^\perp)$	:	$\sum_e Proj_e(\mathbf{d}^\perp)$,
$Cut_e(\mathbf{d})$	:	the number of times e is cut by centerlines in direction $\mathbf{d}$,
$Cut(\mathbf{d})$	:	the total number of cuts made by centerlines on the boundary of $\mathcal{P}$.

Note that $Cut(\mathbf{d})$ is not necessarily equal to $\sum_e Cut_e(\mathbf{d})$; if a centerline passes through a vertex, it is counted only once, in $Cut(\mathbf{d})$, for the two edges that share the vertex.

We have

$$(Cut_e(\mathbf{d}) - 1)\,\delta \leq Proj_e(\mathbf{d}^\perp) < (Cut_e(\mathbf{d}) + 1)\,\delta. \tag{1}$$

The lower bound in the above inequality occurs if there are centerlines that go through the two vertices of e. The upper bound occurs if the centerlines just miss the two vertices (Fig. 9).

Thus,

$$\frac{Proj_e(\mathbf{d}^\perp)}{\delta} - 1 < Cut_e(\mathbf{d}) \leq \frac{Proj_e(\mathbf{d}^\perp)}{\delta} + 1. \tag{2}$$

Summing inequality (2) over all n edges, we get

$$\sum_e \left(\frac{Proj_e(\mathbf{d}^\perp)}{\delta} - 1 \right) < \sum_e Cut_e(\mathbf{d}) \leq \sum_e \left(\frac{Proj_e(\mathbf{d}^\perp)}{\delta} + 1 \right),$$

i.e.,

$$\frac{Proj(\mathbf{d}^\perp)}{\delta} - n < \sum_e Cut_e(\mathbf{d}) \leq \frac{Proj(\mathbf{d}^\perp)}{\delta} + n. \tag{3}$$

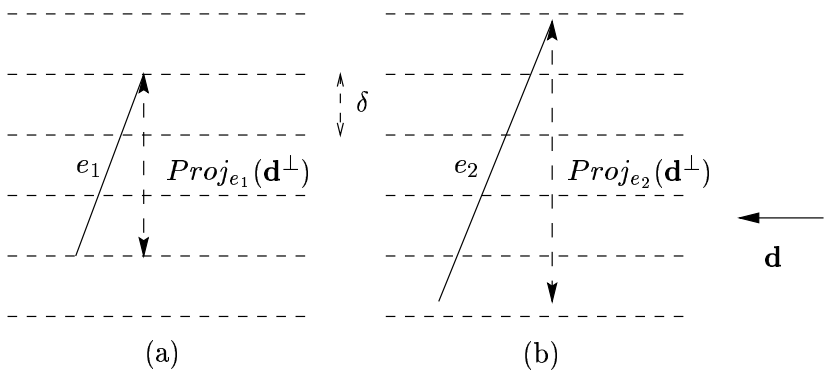


Figure 9: (a) Edge e_1 is cut by four centerlines and both of e's vertices are on the centerlines. The projected length of e_1, in direction $\mathbf{d}$, is exactly 3δ; this corresponds to the lower bound in inequality (1). (b) Edge e_2 is also cut by four centerlines. Its projected length is more than 3δ, but less than 5δ; this corresponds to the upper bound in inequality (1).

Clearly $Cut(\mathbf{d}) \leq \sum_e Cut_e(\mathbf{d})$, since every cut that contributes to $Cut(\mathbf{d})$ has to cut some edge, and thus gets counted in $\sum_e Cut_e(\mathbf{d})$. However, it may also happen that every vertex gets cut by a centerline, so this cut gets counted twice in $\sum_e Cut_e(\mathbf{d})$. Therefore, we have

$$Cut(\mathbf{d}) \leq \sum_e Cut_e(\mathbf{d}) \leq Cut(\mathbf{d}) + n. \tag{4}$$

Combining inequalities (3) and (4), we have, for any $\mathbf{d}$,

$$\frac{Proj(\mathbf{d}^\perp)}{\delta} - 2n < Cut(\mathbf{d}) \leq \frac{Proj(\mathbf{d}^\perp)}{\delta} + n,$$

or

$$-2n < Cut(\mathbf{d}) - \frac{Proj(\mathbf{d}^\perp)}{\delta} \leq n. \tag{5}$$

Let $\mathbf{d}_c$ be the direction $\mathbf{d}$ which minimizes $Cut(\mathbf{d})$, and let $\mathbf{d}_p$ be the direction $\mathbf{d}$ which minimizes $Proj(\mathbf{d}^\perp)$. Note that $Cut(\mathbf{d}_p) \geq Cut(\mathbf{d}_c)$, by definition of $\mathbf{d}_c$ and $\mathbf{d}_p$. Thus,

$$\begin{aligned} 0 \quad \leq \quad Cut(\mathbf{d}_p) - Cut(\mathbf{d}_c) \quad = \quad & \left(Cut(\mathbf{d}_p) - \frac{Proj(\mathbf{d}_p^\perp)}{\delta} \right) + \\ & \left(\frac{Proj(\mathbf{d}_p^\perp)}{\delta} - \frac{Proj(\mathbf{d}_c^\perp)}{\delta} \right) - \end{aligned}$$

$$\left(Cut(\mathbf{d}_c) - \frac{Proj(\mathbf{d}_c^{\perp})}{\delta} \right).$$

Since inequality (5) applies to any direction $\mathbf{d}$, the first and the third terms on the right-hand side above can be combined to give:

$$0 \leq Cut(\mathbf{d}_p) - Cut(\mathbf{d}_c) < 3n + \left(\frac{Proj(\mathbf{d}_p^{\perp})}{\delta} - \frac{Proj(\mathbf{d}_c^{\perp})}{\delta} \right),$$

i.e.,

$$0 \leq Cut(\mathbf{d}_p) - Cut(\mathbf{d}_c) < 3n, \tag{6}$$

since $Proj(\mathbf{d}_p^{\perp}) - Proj(\mathbf{d}_c^{\perp}) \leq 0$ by definition of $\mathbf{d}_c$ and $\mathbf{d}_p$.

From inequality (6), it follows that

$$1 \leq \frac{Cut(\mathbf{d}_p)}{Cut(\mathbf{d}_c)} < 1 + \frac{3n}{Cut(\mathbf{d}_c)}.$$

If the number of cuts is too small, features will be lost and the model will not be a faithful replica of the original. Realistically, it is reasonable to assume that $Cut(\mathbf{d}_c) \geq kn$, where $k \geq 1$. This is true if, for instance, many edges of the polygon are cut at least k times. In this case,

$$1 \leq \frac{Cut(\mathbf{d}_p)}{Cut(\mathbf{d}_c)} < 1 + \frac{3}{k}.$$

If we further assume that 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 our algorithm. (Thus, $\mathbf{d}_c$ and $\mathbf{d}_p$ are the directions sought in Problem 1 and Problem 2, respectively.) Therefore, the number of hatching segments generated by our algorithm is less than $1 + 3/k$ times the minimum number of hatching segments. That is,

$$1 \leq \frac{H(\mathbf{d}_p)}{H(\mathbf{d}_c)} < 1 + \frac{3}{k}.$$

Indeed, if the latter assumption holds, then the above inequality can be strengthened to $1 \leq H(\mathbf{d}_p)/H(\mathbf{d}_c) \leq 1 + 2/k$. This is because we now have $Cut(\mathbf{d}) = \sum_e Cut_e(\mathbf{d})$. Substituting this into inequality (3) and simplifying as above gives the stated bound.

model	z (inches)	#vertices	computed dir.
daikin_trt321	0.039	57	72.9°
	2.769	662	110.6°
	4.329	575	32.7°
impeller	0.579	208	59.1°
	1.489	412	178.3°
	2.799	405	150.0°
mj	0.029	32	12.2°
	1.509	52	78.8°
	2.029	64	93.1°

Table 1: *Single-layer* runs on some polyhedral models. The computed direction is measured in degrees, counterclockwise from the positive x-axis. The running time in all the models was less than 0.01 seconds on a Sun UltraSparcII*i* workstation with a 440 MHz CPU and 256 MB of RAM.

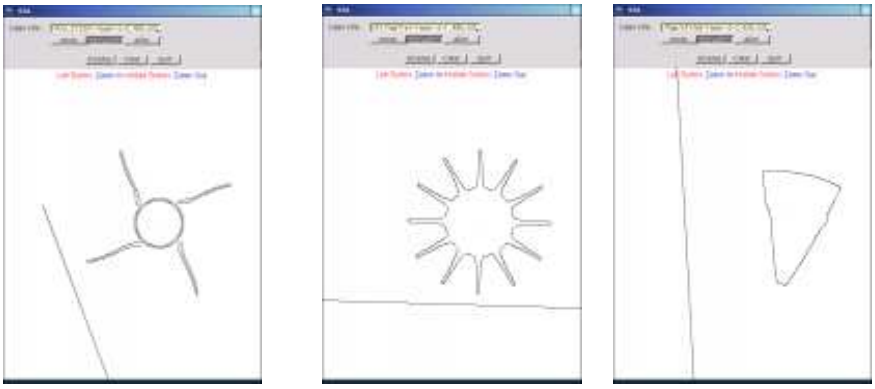
5 Experimental Results

We implemented our algorithm from Section 3 in C++, and tested it on slices generated from real-world polyhedral models we obtained from Stratasys, Inc., a Minnesota-based world-leader in LM. The models were given in the STL format, which is an unordered list of facets, each specified by its three vertices and outward-directed unit-normal. We used the `QuickSlice`[1] program provided by Stratasys to slice each model, with layer-to-layer distance chosen to be 0.01 inch. Each such layer was identified by its z coordinate and represented by its contour cycles, which were given as sequences of vertices. We ran our program on layers chosen at certain heights z. Table 1 shows our results and Figure 10 shows some sample layers.

To evaluate the performance of our algorithm, we also designed and implemented, in [11], an exact algorithm for Problem 1. The exact algorithm is conceptually simple but involves extensive case analysis (as many as seventy-two cases). We tested both algorithms extensively on slices generated from real-world polyhedral models. In our experiments, our approximation algorithm generated at most fourteen percent more hatching segments than the exact algorithm. This suggests that the analysis in Section 4 might be too conservative. Our approximation algorithm also ran significantly faster than the exact one. We refer the reader to [11] for more details.

We remark that our approach also works, without any changes, on poly-

[1] `QuickSlice` is a registered trademark of Stratasys, Inc.

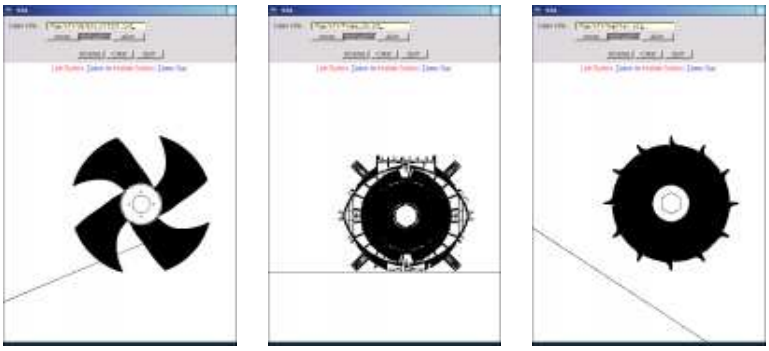


daikin_trt321 at z=2.769 impeller at z=1.489 mj at z=2.029

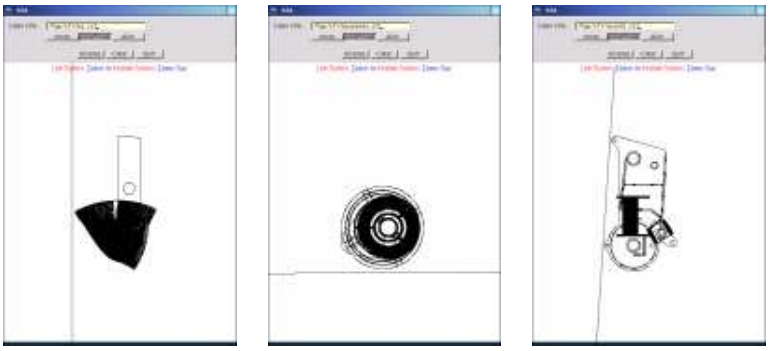
Figure 10: Screen shots of our algorithm running on a single layer, for different models. The long lines shown inside each window is the resulting hatching direction. The sum of the lengths of projections of the edges onto the line perpendicular to the hatching direction is minimal. The results are displayed using the LEDA C++ library [8].

model	#layers	computed dir.	time (sec.)
daikin_trt321	515	22.5°	2.28
frame_29	555	180.0°	1.77
impeller	374	146.9°	1.10
mj	322	90.0°	0.14
myspeedo	323	0.3°	0.55
nose02	457	86.3°	0.22
rd_yelo	338	135.0°	0.06
sa600280	529	90.0°	1.61
tod21	795	146.2°	1.74

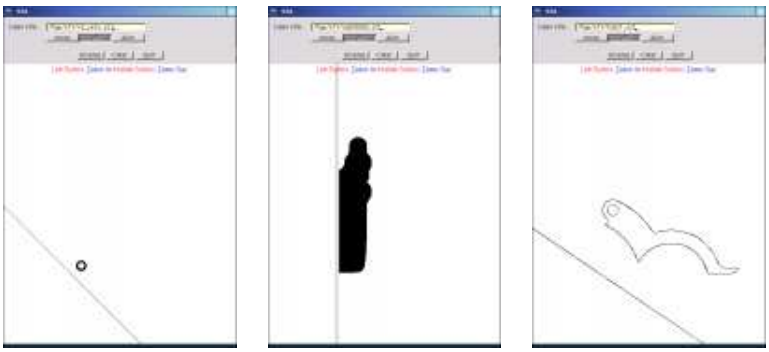
Table 2: *All-layers* runs on some polyhedral models. The computed direction is measured in degrees, counter-clockwise from the positive x-axis. Experiments were done on a Sun UltraSparcIIi, with a 440 MHz CPU and 256 MB of RAM.



daikin_trt321 frame_29 impeller



mj myspeedo nose02



rd_yelo sa600280 tod21

Figure 11: Screen shots of our algorithm running on *all* layers of different models. The viewpoint is from the positive z direction.

gons with holes. This is because we only need information about the orientation and lengths of the edges; we do not need information about how the edges connect to each other.

We also implemented the idea discussed at the end of Section 1.1 to compute a globally optimal direction for the entire model. Table 2 shows some of our results, and Figure 11 shows some of our models, as viewed along the positive z direction. The actual running time was very small; the computation seldom took more than 2 seconds.

6 Other applications in LM

We describe some related problems in LM that can be solved with our approach.

6.1 Weighted hatching

During hatching it may be desirable to protect certain edges of the polygon $\mathcal{P}$ from being hit too often by the tool, as these edges may be critical to the appearance or function of the part. We can accomplish this by assigning a non-negative weight, w_e, to each edge e, with higher weights on the more critical edges. Using the notation in Section 1.1, let

$$\tilde{H}(\mathbf{d}) = \sum_{\ell \in \mathcal{L}(\mathbf{d})} W_\ell \, |\mathcal{S}_\ell| \, ,$$

where W_ℓ is the sum of the weights, w_e, of the edges e that are hit by line ℓ. This yields a weighted version of Problem 1, where the goal is to find a direction $\mathbf{d}$ which minimizes $\tilde{H}(\mathbf{d})$. The corresponding weighted approximation problem is similar to Problem 2, except that the function to be minimized is

$$\sum_{\mathbf{n}_e \in \mathcal{S}} w_e \, |\mathbf{n}_e \cdot \mathbf{d}| \, .$$

Since this function is equal to $\sum_{\mathbf{n}_e \in \mathcal{S}} |(w_e \mathbf{n}_e) \cdot \mathbf{d}|$, we can solve the approximation problem using the algorithm of Section 3, after we scale each vector $\mathbf{n}_e$ by w_e. (Recall that this algorithm works for any set of vectors.) This yields a direction which approximates the optimal direction sought in the weighted version of the hatching problem.

Theorem 3 *The weighted version of Problem 2, with objective function* $\sum_{\mathbf{n}_e \in \mathcal{S}} w_e \, |\mathbf{n}_e \cdot \mathbf{d}|$, *can be solved in* $O(n \log n)$ *time using* $O(n)$ *space.*

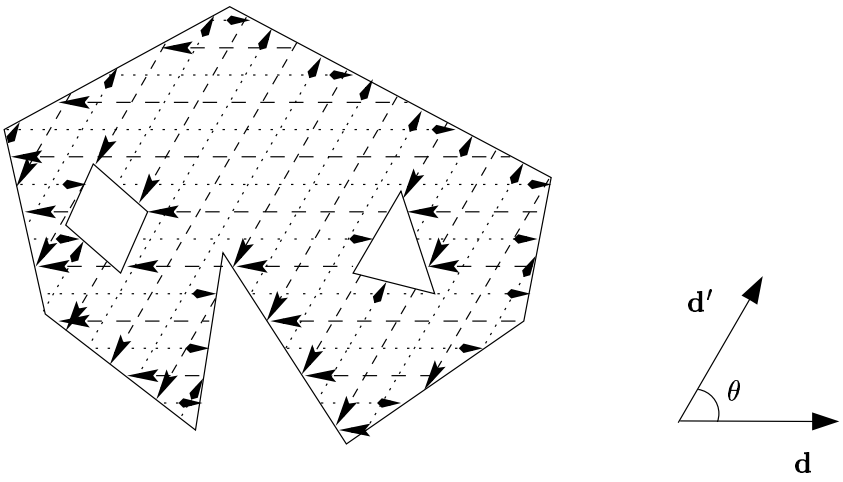


Figure 12: The slice is built by hatching along two directions that make an angle of θ, $0 < \theta \leq 90°$, with each other.

6.2 Hatching along two prescribed directions

To improve the strength of the manufactured part, the polygon can be hatched in a weave-like pattern; i.e. each layer is hatched in not one but two independent directions $\mathbf{d}$ and $\mathbf{d}'$ [5]. We assume the two directions make some fixed angle θ, $0 < \theta \leq 90°$, with each other (Fig. 12). This yields a "2-directional" version of Problem 1, where the goal is to find directions $\mathbf{d}$ and $\mathbf{d}'$, as above, so that the function

$$\sum_{\ell \in \mathcal{L}(\mathbf{d})} |\mathcal{S}_\ell| + \sum_{\ell \in \mathcal{L}(\mathbf{d}')} |\mathcal{S}_\ell|$$

is minimized.

The corresponding approximation problem is similar to Problem 2, except that the objective function is

$$\sum_{\mathbf{n}_e \in \mathcal{S}} \left(|\mathbf{n}_e \cdot \mathbf{d}| + |\mathbf{n}_e \cdot \mathbf{d}'| \right).$$

As in Section 2, we can construct a convex polygon $\mathcal{Q}$ from the normal vectors. However, instead of finding the minimum width of this polygon, our problem now becomes:

Problem 3 (Minimum bounding parallelogram) *Given a convex polygon $\mathcal{Q}$, with n vertices, find a bounding parallelogram of minimum perimeter, whose adjacent sides make an angle θ with each other, $0 < \theta \leq 90°$.*

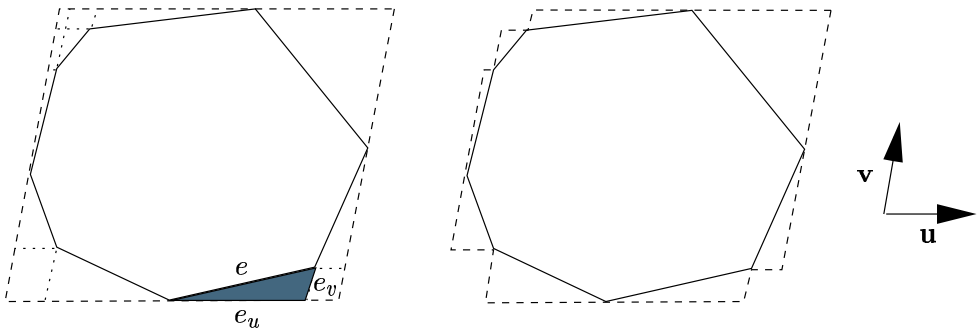


Figure 13: Every piece of the perimeter of the bounding parallelogram is associated with an edge of the convex polygon.

Consider any bounding parallelogram of $\mathcal{Q}$, where the adjacent edges of the parallelogram make an angle θ with each other. We associate pieces of the boundary of this parallelogram with the edges of $\mathcal{Q}$, as follows: Let $\mathbf{u}$ and $\mathbf{v}$ be unit vectors in the same directions as the two non-parallel edges of the bounding parallelogram. (Without loss of generality, assume that $\mathbf{v}$ is θ degrees counterclockwise from $\mathbf{u}$.) Each edge e of $\mathcal{Q}$ can be perceived as having two components: a projection, e_u, along direction $\mathbf{v}$ onto a line parallel to direction $\mathbf{u}$, and a projection, e_v, along direction $\mathbf{u}$ onto a line parallel to direction $\mathbf{v}$. (Fig. 13.) It is easy to see that the sum of the lengths of these projections over all the edges is the perimeter of the bounding parallelogram.

Let $\mathbf{e}$ be a vector along edge e, with the same length as e and oriented so that the polygon is to its left. We can write $\mathbf{e}$ in terms of the basis vectors $\mathbf{u}$ and $\mathbf{v}$ as:

$$\mathbf{e} = a\mathbf{u} + b\mathbf{v}. \tag{7}$$

Then e_u is given by $|a|$ and e_v by $|b|$. Let $\mathbf{u}^\perp$ (resp. $\mathbf{v}^\perp$) be $\mathbf{u}$ (resp. $\mathbf{v}$) rotated 90° counterclockwise. Taking the dot product with $\mathbf{u}^\perp$ on both sides of Equation (7) gives:

$$\mathbf{e} \cdot \mathbf{u}^\perp = 0 + b\mathbf{v} \cdot \mathbf{u}^\perp = b\left(\mathbf{v} \cdot \mathbf{u}^\perp\right).$$

Taking the dot product with $\mathbf{v}^\perp$ on both sides of Equation (7) gives:

$$\mathbf{e} \cdot \mathbf{v}^\perp = a\mathbf{u} \cdot \mathbf{v}^\perp + 0 = a\left(\mathbf{u} \cdot \mathbf{v}^\perp\right).$$

Therefore

$$e_u + e_v = |a| + |b| = \frac{\left|\mathbf{e} \cdot \mathbf{v}^\perp\right|}{\left|\mathbf{u} \cdot \mathbf{v}^\perp\right|} + \frac{\left|\mathbf{e} \cdot \mathbf{u}^\perp\right|}{\left|\mathbf{v} \cdot \mathbf{u}^\perp\right|}.$$

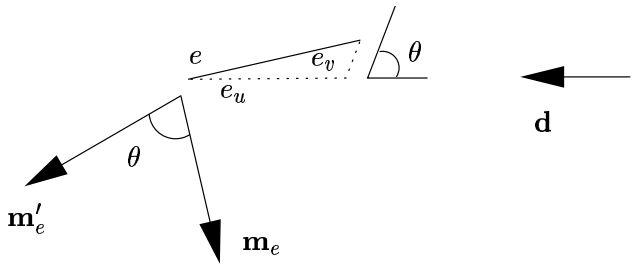


Figure 14: Associating pieces of the boundary of the bounding parallelogram with edges of the convex polygon. We rotate $\mathbf{m}_e$ clockwise by θ degrees to get $\mathbf{m}'_e$, so that $\|e_u\| = |\mathbf{m}'_e \cdot \mathbf{d}|$ and $\|e_v\| = |\mathbf{m}_e \cdot \mathbf{d}|$.

Note that $\left|\mathbf{u} \cdot \mathbf{v}^{\perp}\right| = \left|\mathbf{v} \cdot \mathbf{u}^{\perp}\right| = \sin\theta$, which is a constant over the entire class of parallelograms whose adjacent edges make an angle θ. We now have:

$$e_u + e_v = \frac{1}{\sin\theta}\left(\left|\mathbf{e} \cdot \mathbf{v}^{\perp}\right| + \left|\mathbf{e} \cdot \mathbf{u}^{\perp}\right|\right). \tag{8}$$

We represent the orientation of the parallelogram by the vector $\mathbf{u}$, which we rename as $\mathbf{d}$. Let us denote by $\mathbf{m}_e$ the outer normal of edge e of $\mathcal{Q}$; $\mathbf{m}_e$ has the same length as e and begins at the origin. Note that $\left|\mathbf{e} \cdot \mathbf{u}^{\perp}\right| = |\mathbf{m}_e \cdot \mathbf{d}|$. But what about the other dot product in Equation (8)? We can take care of this by inserting a new vector $\mathbf{m}'_e$, which is $\mathbf{m}_e$ rotated θ degrees clockwise. (Fig. 14.) It is not difficult to see that $\left|\mathbf{e} \cdot \mathbf{v}^{\perp}\right| = |\mathbf{m}'_e \cdot \mathbf{d}|$. The advantage of introducing $\mathbf{m}'_e$ is that we can now write $e_u + e_v$ in terms of a single direction $\mathbf{d}$. Specifically,

$$e_u + e_v = \frac{1}{\sin\theta}\left(|\mathbf{m}'_e \cdot \mathbf{d}| + |\mathbf{m}_e \cdot \mathbf{d}|\right). \tag{9}$$

We will henceforth ignore the constant factor $1/\sin\theta$ in Equation (9). We duplicate each normal $\mathbf{m}_e$ and rotate it θ degrees clockwise. Using our algorithm of Section 2 or Section 3 on this enlarged set of vectors we get a direction $\mathbf{d}$ which minimizes the sum of the absolute values of the dot products in Equation (9). The bounding parallelogram with one side parallel to the computed direction $\mathbf{d}$ and the other rotated θ degrees counterclockwise from $\mathbf{d}$ is the one which minimizes the perimeter among all bounding parallelograms of $\mathcal{Q}$ whose adjacent sides make the prescribed angle θ. This solves the approximate version of the 2-directional hatching problem.

Constructing $\mathcal{Q}$ from $\mathcal{P}$ takes $O(n \log n)$ time, due to the sorting of the normals $\mathbf{n}_e$ of $\mathcal{P}$. Since $\mathcal{Q}$ is convex, its normals, $\mathbf{m}_e$, are already sorted in

circular order. The θ-degree rotations, $\mathbf{m}_e$, of these normals are in sorted order, too. Therefore, in $O(n)$ time, we can merge the two sets of vectors and obtain the $2n$ normals sorted in circular order. The rest of the algorithm takes $O(n)$ time and $O(n)$ space.

Theorem 4 *The 2-directional version of Problem 2, with objective function* $\sum_{\mathbf{n}_e \in \mathcal{S}} (|\mathbf{n}_e \cdot \mathbf{d}| + |\mathbf{n}_e \cdot \mathbf{d}'|)$, *can be solved in* $O(n \log n)$ *time using* $O(n)$ *space. (Here directions* $\mathbf{d}$ *and* $\mathbf{d}'$ *make a prescribed angle* θ *with each other,* $0 < \theta \leq 90°$.*)*

If we are given a convex polygon $\mathcal{Q}$ to begin with (as opposed to constructing it from a simple polygon $\mathcal{P}$) and wish to solve just Problem 3, then $O(n)$ time and $O(n)$ space suffice, since we can dispense with the initial sorting step. Indeed, we can solve Problem 3 in $O(n)$ time and $O(n)$ space even if $\mathcal{Q}$ is not convex, since we can replace it by its convex hull in $O(n)$ time [9].

6.3 Hatching to minimize stair-step error

Due to the non-zero width of the tool-tip, the hatching process cannot always produce an exact replica of the original polygon $\mathcal{P}$; rather, $\mathcal{P}$ gets approximated by a sequence of rectangular strips, i.e., the hatching segments. The resulting polygon has a stair-stepped appearance, where each edge is approximated as a sequence *error-triangles* (Fig. 15); this is similar to the phenomenon of antialiasing in computer graphics. We quantify the *stair-step error* for $\mathcal{P}$ as the sum of the heights of all the error-triangles. Clearly, the stair-step error is a function of the hatching direction. The problem we wish to solve is:

Problem 4 (Minimum stair-step error) *Given a simple n-vertex polygon* $\mathcal{P}$ *(possibly with holes), find a hatching direction which minimizes the stair-step error for* $\mathcal{P}$.

We note that the notion of stair-step error has been considered previously for LM in [1, 7]. However, the focus there was on finding a direction to build the three-dimensional model that minimized the stair-step error on the facets. Here we consider the problem in two dimensions and give a more efficient algorithm.

We convert this problem to a form where our algorithm from Section 3 can be applied. Let $\mathbf{d}$ be a candidate hatching direction; without loss of

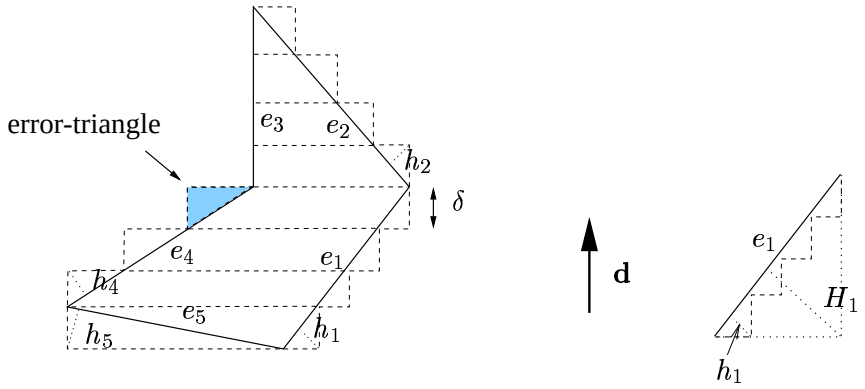


Figure 15: The polygon is approximated by a sequence of hatching segments, which results in a stair-stepped appearance. The shaded region is an error-triangle. The total height of all the error triangles for this polygon is $4h_1 + 4h_2 + 3h_4 + h_5$.

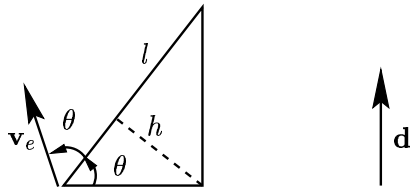


Figure 16: The height h of the triangle is $|\mathbf{v}_e \cdot \mathbf{d}|$, where $\|\mathbf{v}_e\| = l/2$.

---

generality, assume that $\mathbf{d}$ points upwards. Consider (say) edge e_1 in Figure 15. It is easy to see, by a simple similarity argument, that the sum of the heights of all the error-triangles for e_1 equals the height H_1 of the larger triangle shown. Let l be the length of e_1 and let θ be the angle e_1 makes with the horizontal. Then $H_1 = l \cos\theta \sin\theta = (l/2) \sin 2\theta$. This suggests the following approach: For every edge $e \in \mathcal{P}$, we create a vector $\mathbf{v}_e$ at the origin, whose length is half that of e and which makes an angle 2θ with the horizontal (Fig. 16). The sum of the heights of all the error-triangles of e equals $|\mathbf{v}_e \cdot \mathbf{d}|$. Therefore, our problem is to find a direction $\mathbf{d}$ which minimizes $\sum_{e \in \mathcal{P}} |\mathbf{v}_e \cdot \mathbf{d}|$. We can use the algorithm of Section 3 to solve this problem (recall that this algorithm works for any set of vectors).

Theorem 5 *Problem 4 can be solved in $O(n \log n)$ time using $O(n)$ space.*

7 Higher dimensions

We explore the generalization of our algorithm from Section 3 to higher dimensions. Specifically, we wish to solve the following problem:

Problem 5 (Minimum k-dimensional projection) *Given a finite set, $\mathcal{S}$, of n vectors in k-dimensional space, 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.*

We pick an initial direction $\mathbf{d}$ and reflect through the origin all vectors that generate a negative dot product with $\mathbf{d}$. We then consolidate those vectors that have the same direction into a single vector. Analogous to Section 3, we argue that the minimizing direction must be simultaneously orthogonal to at least $k-1$ vectors in $\mathcal{S}$. Specifically, let $\mathbf{d}$ be a candidate unit vector and let

$$\tilde{\mathcal{S}}_{\mathbf{d}} = \{\mathbf{v} \mid \mathbf{v}\in\mathcal{S} \text{ and } \mathbf{v}\cdot\mathbf{d}\geq 0\} \cup \{-\mathbf{v} \mid \mathbf{v}\in\mathcal{S} \text{ and } \mathbf{v}\cdot\mathbf{d} < 0\}.$$

The vector $\mathbf{d}$ defines a hyperplane $\mathcal{P}_{\mathbf{d}} = \{\mathbf{v}\in\mathbb{R}^k \mid \mathbf{v}\cdot\mathbf{d}=0\}$ through the origin. All the vectors $\tilde{\mathbf{v}}$ in $\tilde{\mathcal{S}}_{\mathbf{d}}$ are on the same side of $\mathcal{P}_{\mathbf{d}}$ as $\mathbf{d}$. We have

$$\sum_{\mathbf{v}\in\mathcal{S}} |\mathbf{v}\cdot\mathbf{d}| = \sum_{\tilde{\mathbf{v}}\in\tilde{\mathcal{S}}_{\mathbf{d}}} (\tilde{\mathbf{v}}\cdot\mathbf{d}) = \left(\sum_{\tilde{\mathbf{v}}\in\tilde{\mathcal{S}}_{\mathbf{d}}} \tilde{\mathbf{v}}\right)\cdot\mathbf{d}.$$

Note that we can rotate $\mathcal{P}_{\mathbf{d}}$ about the origin such that the composition of the set $\tilde{\mathcal{S}}_{\mathbf{d}}$ does not change and still reduce the quantity $\left(\sum_{\tilde{\mathcal{S}}_{\mathbf{d}}} \tilde{\mathbf{v}}\right)\cdot\mathbf{d}$. We can do this until $\mathcal{P}_{\mathbf{d}}$ contains some vector of $\tilde{\mathcal{S}}_{\mathbf{d}}$. Next we can rotate $\mathcal{P}_{\mathbf{d}}$ about this vector without changing $\tilde{\mathcal{S}}_{\mathbf{d}}$ and continue to reduce $\left(\sum_{\tilde{\mathcal{S}}_{\mathbf{d}}} \tilde{\mathbf{v}}\right)\cdot\mathbf{d}$ until $\mathcal{P}_{\mathbf{d}}$ contains a second vector of $\tilde{\mathcal{S}}_{\mathbf{d}}$. And so on until $\mathcal{P}_{\mathbf{d}}$ contains some $k-1$ independent vectors of $\tilde{\mathcal{S}}_{\mathbf{d}}$. At this point $\mathcal{P}_{\mathbf{d}}$ is completely constrained, since any $k-1$ independent vectors of $\tilde{\mathcal{S}}_{\mathbf{d}}$ uniquely determine a hyperplane through the origin, and $\mathcal{P}_{\mathbf{d}}$ cannot be rotated further without changing $\tilde{\mathcal{S}}_{\mathbf{d}}$. It follows that a necessary condition for $\mathbf{d}$ to be a minimizing direction is that it should be perpendicular to at least $k-1$ vectors of $\tilde{\mathcal{S}}_{\mathbf{d}}$.

The algorithm in higher dimensions is best understood by first considering the problem in three dimensions. Let $\mathcal{S} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$. Each $\mathbf{v}_i \in \mathcal{S}$ determines a unique great circle $C_i = \{\mathbf{s}\in\mathbf{S}^2 \mid \mathbf{s}\cdot\mathbf{v}_i = 0\}$ on the unit sphere $\mathbf{S}^2$. Every vector on C_i is orthogonal to $\mathbf{v}_i$. From the discussion above, it suffices to examine only the intersections of any $3-1=2$ such C_i's. We fix a C_i, compute its intersection with each C_j, $j\neq i$, and sort these in

circular order on C_i. We pick an arbitrary intersection point as our first candidate direction $\mathbf{d}$. As in Section 3, we reflect through the origin any vector whose dot product with $\mathbf{d}$ is negative. Starting from $\mathbf{d}$ we walk on the half-circle of C_i (it does not matter which half) between $\mathbf{d}$ and $-\mathbf{d}$, visit the intersection points in circular order, and update our vector sum. If the current intersection point is the intersection of C_j and C_i, we update the vector sum by adding $-2\tilde{\mathbf{v}}_j$ to it. We compute the new dot product and update the minimum if we now have a smaller dot product. We do this for every $\mathbf{v}_i$ in $\mathcal{S}$.

Note that in the above computation for $\mathbf{v}_i$, we would have obtained the same result if we had first projected all the vectors in $\mathcal{S}$ onto the plane containing C_i, rotated the plane to make it coincide with the x-y plane, and then used our 2-dimensional algorithm from Section 3. Viewing the process this way allows us to design an algorithm that handles arbitrary dimensions by systematically reducing the problem's dimension: to solve the k-dimensional problem, we break it up into n subproblems in $(k-1)$ dimensions, one per vector $\mathbf{v}_i$. The running time for a k-dimensional problem is thus n times that of a $(k-1)$-dimensional one. Since the 2-dimensional problem takes $O(n \log n)$ time, this gives a total running time of $O(n^{k-1} \log n)$. However, the space requirement is still $O(n)$. The algorithm is also highly parallelizable.

We can devise a slightly faster algorithm at the expense of more space. Again, it is easiest to first discuss the approach in three dimensions. We first compute the *arrangement* of all the C_i's on the unit sphere, i.e., the subdivision of the unit sphere determined by the C_i's. This arrangement can be computed in $O(n^2)$ time and space [3]. We visit all the intersection points (on, say, the upper hemisphere) by following the edges (circular arcs) of the arrangement, update incrementally the quantity $\sum_{\tilde{\mathcal{S}}} \tilde{\mathbf{v}}$, and compute its dot product with the current direction. In more detail, suppose that we have just arrived at an intersection I following a circular arc. This arc belongs to the great circle C_i of some vector $\mathbf{v}_i$. Let I be the intersection between C_i and some other great circle C_j. We update $\sum_{\tilde{\mathcal{S}}} \tilde{\mathbf{v}}$ by adding $-2\mathbf{v}_j$ to it and then take the dot product of the resulting vector with the direction represented by I. The advantage of using the arrangement is that we avoid the n instances of circular sorting needed in the previous approach. However, the entire arrangement needs to be kept in memory while we search through it. This increases the space requirement to $O(n^2)$ in three dimensions, while the running time reduces to $O(n^2)$. In k-dimensions, the running time becomes $O(n^{k-1})$ at the expense of $O(n^{k-1})$ space.

model	#facets	computed dir.	time (sec.)
daikin_trt321	19402	(-0.47, 0.88, -0.00)	1393.96
frame_29	67056	(0, 0, -1)	3656.13
impeller	30896	(-0.75, 0.66, -0.00)	4206.27
mj	2832	(0.00, 1, -0.00)	17.34
myspeedo	16720	(0.00, -0.00, -1)	627.39
nose02	5090	(0.06, 1.00, -0.00)	17.92
rd_yelo	396	(0.00, -0.00, -1)	0.05
sa600280	74346	(-0, 0.71, -0.71)	34076.10
tod21	1128	(-0, 0, -1)	0.17

Table 3: Results from running the 3-dimensional algorithm on normals derived from polyhedral models. The computed direction is given as an (x, y, z) triple. Experiments were done on a Sun UltraSparcII*i*, with a 440 MHz CPU and 256 MB of RAM.

Theorem 6 *Problem 5 can be solved in k dimensions in time $O(n^{k-1} \log n)$ time using $O(n)$ space, or in $O(n^{k-1})$ time using $O(n^{k-1})$ space.*

We implemented the first algorithm above in C++ for $k = 3$. We ran it on the polyhedral models described in Table 2 of Section 5 (specifically, we used as input to our algorithm the outer normals of the facets of these polyhedra). Table 3 shows some of the results. As one might expect from Theorem 6, this algorithm was slower than the ones in Section 5. For instance, on the largest model, *sa600280*, with about 74,000 facets, the algorithm took about nine hours.

8 Conclusions and future work

In this paper, we have seen the role of the projection minimization problem (Problem 2) in unifying seemingly different geometric problems in LM under a single framework, leading to efficient solutions to all of them in one fell swoop. The algorithms are very fast to moderately fast in dimensions two and three, which is where the current applications of interest seem to lie. In higher dimensions, the algorithms are probably too slow to be of much practical use. One approach to improving upon the latter result is to design an efficient approximation algorithm for the projection minimization problem in higher dimensions. Another direction is to establish a lower bound for the problem. We leave these problems to future investigation.

Acknowledgements

We thank Stratasys, Inc. for providing us with test models and for access to their software front-end, `QuickSlice`, to slice these models.

References

[1] M. Bablani and A. Bagchi. Quantification of errors in rapid prototyping processes and determination of perferred orientation of parts. In *Transactions of the 23rd North American Manufacturing Research Conference*, 1995.

[2] M. Berger. *Geometry (vols. 1-2).* Springer-Verlag, 1987.

[3] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[4] 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.

[5] P. Jacobs. *Rapid Prototyping & Manufacturing: Fundamentals of Stereolithography.* McGraw-Hill, 1992.

[6] C.C. Kai and L.K. Fai. *Rapid Prototyping: Principles and applications in manufacturing.* John Wiley & Sons, New York, 1997.

[7] J. Majhi, R. Janardan, M. Smid, and P. Gupta. On some geometric optimization problems in layered manufacturing. *Comput. Geom. Theory Appl.*, 12:219–239, 1999.

[8] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing.* Cambridge University Press, Cambridge, UK, 1999.

[9] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, 3rd edition, October 1990.

[10] S. E. Sarma. The crossing function and its application to zig-zag tool paths. *Comput. Aided Design*, 31:881–890, 1999.

[11] 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.

[12] G. T. Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE MELECON '83*, pages A10.02/1–4, 1983.