Computing an optimal hatching direction in Layered Manufacturing*

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

Abstract

In Layered Manufacturing (LM), a prototype of a virtual polyhedral object is built by slicing the object into polygonal layers, and then building the layers one after another. In StereoLithography, a specific LM-technology, a layer is built using a laser which follows paths along equally-spaced parallel lines and hatches all segments on these lines that are contained in the layer. We consider the problem of computing a direction of these lines for which the number of segments to be hatched is minimum, and present an algorithm that solves this problem exactly. The algorithm has been implemented and experimental results are reported for real-world polyhedral models obtained from industry.

1 Introduction

Layered Manufacturing (LM) makes it possible to rapidly build 3D prototypes from polyhedral CAD descriptions. This technology, which is heavily used in industry, includes techniques such as StereoLithography, fused deposition modeling, and laminated object manufacturing. (See e.g. the book by Jacobs [5].) The input to these processes is a three-dimensional polyhedron, whose boundary is triangulated. This polyhedron is first sliced into horizontal polygonal layers. Then these slices are successively "built", where

*This work was funded in part by a joint research grant by DAAD and by NSF.

†Department of Computer Science, University of Magdeburg, D-39106 Magdeburg, Germany. E-mail: {schwerdt,michiel}@isg.cs.uni-magdeburg.de.

‡Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, U.S.A. E-mail: {hon,janardan}@cs.umn.edu. Research also supported in part by NSF grant CCR-9712226.

each one is attached to the one immediately below it. The way the slices are built depends on the specific LM-technology. For example, in StereoLithography, the prototype is built in a vat of liquid resin. A laser first traces out the polygonal contour of the slice and then *hatches* (i.e., fills) the interior, which hardens to a depth equal to the layer thickness. When hatching the interior of a slice, the laser follows paths along equally-spaced parallel lines and hatches all segments on these lines that are contained in the polygon defined by the contour. (See Figure 1.) Each segment that is hatched by the laser has a fixed width, which we denote by δ. Hence, any two neighboring lines containing hatched segments have distance δ. If we assume that these lines are horizontal, then we can think of this hatching process as covering the polygonal slice with axes-parallel rectangles of height δ, whose horizontal central axes have distances to each other which are multiples of δ. The time needed to hatch the entire slice heavily depends on the number of rectangles (i.e., the number of segments that are hatched by the laser) that cover the slice. (See e.g. Sarma [12] for a discussion about this.) Clearly, by choosing another direction for the parallel lines, i.e., the *hatching direction*, the number of hatched segments may be substantially smaller.

In this paper, we consider the problem of computing an optimal hatching direction. That is, given a simple polygon, which represents the contour of a slice, we want to compute a hatching direction for which the number of segments drawn by the laser is minimum. In the next subsection, we give a formal definition of this problem.

1.1 Problem definition

Let $\mathcal{P}$ be a simple polygon, possibly with holes, and let n denote the number of vertices of $\mathcal{P}$. This polygon is given as a sequence containing one or more lists. Each list contains the vertices of a closed polygonal part of the boundary of $\mathcal{P}$. The order of the vertices in these lists is such that the interior of $\mathcal{P}$ is to the left of the (directed) edges.

Let $\mathbf{d}$ be any *direction*, i.e., a unit-vector in the two-dimensional plane. We will refer to $\mathbf{d}$ as the *hatching direction*. Let $\ell_0(\mathbf{d})$ be the line through the origin having direction $\mathbf{d}$. 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 (integer) multiples of δ. (See Figure 1.)

For any line ℓ in $\mathcal{L}(\mathbf{d})$, consider the intersection of ℓ with the polygon $\mathcal{P}$. This intersection consists of zero or more line segments, and zero or more single points. We denote by S_ℓ the set of line segments (of positive length)

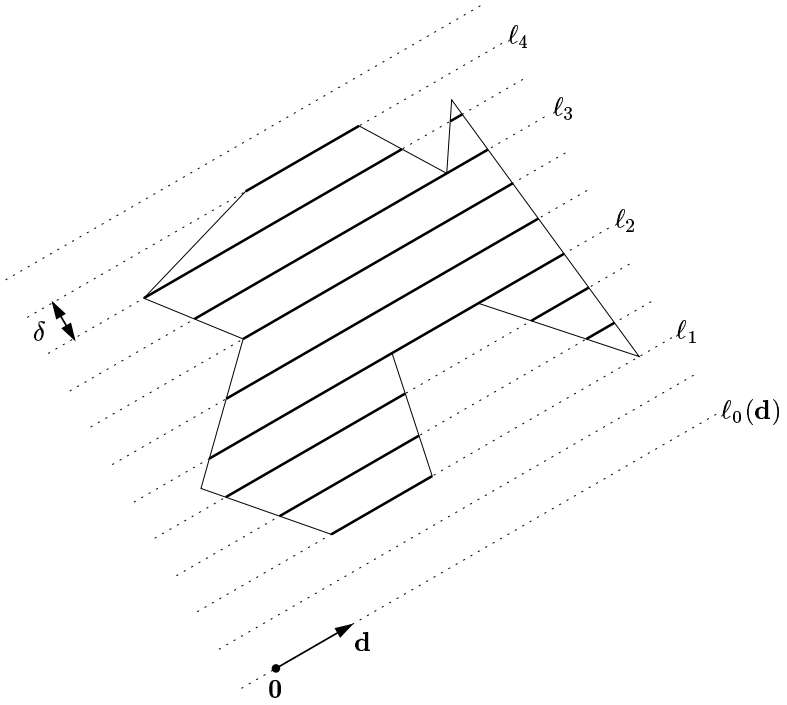

Figure 1: *Illustrating the hatching problem.* $\mathbf{0}$ *denotes the origin. The dashed lines are parallel, and two neighboring lines have distance* δ*. For the given hatching direction* $\mathbf{d}$*, the value of* $H(\mathbf{d})$*, i.e., the number of segments hatched by the laser, is equal to twelve. Note that for each* i*,* $1 \leq i \leq 4$*, the set* S_{ℓ_i} *consists of a single line segment.*

---

in this intersection, and define

$$H(\mathbf{d}) := \sum_{\ell \in \mathcal{L}(\mathbf{d})} |S_\ell|.$$

Hence, $H(\mathbf{d})$ is equal to the number of segments that the laser draws when hatching the polygon $\mathcal{P}$ in direction $\mathbf{d}$. Using this terminology, the *hatching problem* can be stated as follows.

Problem 1 *Given a simple polygon* $\mathcal{P}$*, possibly with holes, having* n *vertices, compute a hatching direction* $\mathbf{d}$ *for which* $H(\mathbf{d})$ *is minimum. We will call such a direction* $\mathbf{d}$ *an* optimal hatching direction.

1.2 Our results

In Sections 2 and 3, we give an exact algorithm that solves Problem 1 in $O(Cn\log(Cn))$ time, where C is proportional to the ratio of the maximum distance from any vertex of $\mathcal{P}$ to the origin, and δ; specifically, $C = 1+\max_{v\in\mathcal{P}}\|v\|/\delta$, where $\|v\|$ is the distance of vertex v from the origin. This algorithm performs a rotational sweep, during which the value of $H(\mathbf{d})$ is dynamically maintained. Although this algorithm is conceptually simple, a large number of case distinctions is necessary to implement it. Nevertheless, we have implemented this algorithm in software and have tested it on polygonal slices generated by slicing real-world polyhedral models obtained from industry. We discuss this in Section 4. We conclude in Section 5 with directions for further work.

1.3 Related work

In a companion paper [4], we give a simple heuristic for approximating the minimum value of $H(\mathbf{d})$, and explore its applications to several other related problems in two and higher dimensions. In essence, this heuristic approximates the optimal hatching direction by finding a direction which minimizes the sum of the lengths of the edges when they are projected onto a line perpendicular to this direction. The quality of this heuristic depends on the value of δ; the smaller this value, the better the approximation. This heuristic was discovered independently by Sarma [12] in the context of planning an optimal zig-zag path for milling machines. To keep the present paper to a reasonable length, we omit further discussion of this heuristic and its implementation and applications; we refer the reader to [4] for further details.

As mentioned above, the polyhedral model to be built by Layered Manufacturing is first sliced into layers. McMains and Séquin [9] give an efficient space sweep algorithm for computing the slices. The prototype is built layer by layer, and the quality, costs, and time of this process depend on the orientation of the three-dimensional object. Recently, the problem of computing a good orientation has been considered in the computational geometry community. See [1, 2, 6, 7, 8, 13, 14, 15].

2 Solving the hatching problem

In this section, we give an outline of our algorithm that solves Problem 1. Details will be given in Section 3. We make the following assumptions about the polygon $\mathcal{P}$.

Assumption 1 *The following two conditions are satisfied.*

1. *No vertex of $\mathcal{P}$ is at the origin.*

2. *No three successive vertices of $\mathcal{P}$ are collinear.*

In Layered Manufacturing, the facets of a triangulated 3D-polyhedron are given in the so-called STL-format. In this format, each facet is specified by the coordinates of its three vertices, and the outer normal of the facet. The STL-format requires that all coordinates are positive. (See Jacobs [5, page 160].) Therefore, when building a model, all slices—which are polygons—are in the first quadrant. Hence, the first assumption is satisfied in Layered Manufacturing.

If the second assumption does not hold, then we delete the middle vertex of each triple of successive collinear vertices of the polygon $\mathcal{P}$. Note that this does not change the solution to Problem 1.

Let us see how we can solve Problem 1. First note that for any direction $\mathbf{d}$, we have $H(\mathbf{d}) = H(-\mathbf{d})$. Therefore, 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. Note that during the rotation, the collection $\mathcal{L}(\mathbf{d})$ of lines rotates, with the origin being the center of rotation.

We now give necessary conditions for a direction $\mathbf{d}$ to be critical. There are two types of directions $\mathbf{d}$, for which the value of $H(\mathbf{d})$ 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 the polygon $\mathcal{P}$. First 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 the convex hull $CH(\mathcal{P})$ such that the following two conditions hold.

- v is extreme in one of the directions $\mathbf{d}^\perp$ and $-\mathbf{d}^\perp$.

- 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 S_ℓ of line segments (of positive length) in the intersection $\ell \cap \mathcal{P}$ changes.

If this happens, then there is a vertex v of the polygon $\mathcal{P}$ such that the following two conditions hold.

- 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.
- Both vertices of $\mathcal{P}$ that are adjacent to vertex 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, e.g., when a vertex that is adjacent to v lies on the line $\ell_v(\mathbf{d})$. We will see later how such cases can be handled.)

Let $\mathbf{D}$ be the set of all directions $\mathbf{d}$ for which there is a vertex v of $\mathcal{P}$ such that the distance between v and the line $\ell_0(\mathbf{d})$ is a multiple of δ. Then the discussion above implies that this set $\mathbf{D}$ contains all critical directions.

2.1 Overview of the algorithm

We now give a high-level description of the algorithm that solves Problem 1.

Step 1: For each vertex v of the polygon $\mathcal{P}$, 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 set of directions obtained in this way.

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. Note that this is equivalent to sorting the directions of $\mathbf{D}$ in increasing order of their first coordinates. We denote this ordering relation by $\prec$.

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 index i and any two directions $\mathbf{d}$ and $\mathbf{d}'$ that are strictly between $\mathbf{d}^i$ and $\mathbf{d}^{i+1}$, we have $H(\mathbf{d}) = H(\mathbf{d}')$. (Here, indices are read modulo m.)

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

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 $\mathbf{d}^i$, check if it is a critical direction, and if so, 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 $\mathbf{d}^i \prec \mathbf{d} \prec \mathbf{d}^{i+1}$.

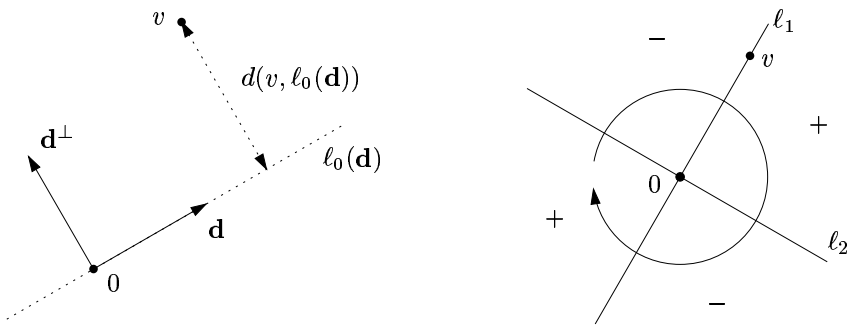

Figure 2: *The left part illustrates the two orthogonal directions* $\mathbf{d}$ *and* $\mathbf{d}^\perp$*, and the distance* $d(v, \ell_0(\mathbf{d}))$ *between vertex* v *and the line* $\ell_0(\mathbf{d})$ *through the origin having direction* $\mathbf{d}$*. The right part illustrates how the distance* $d(v, \ell_0(\mathbf{d}))$ *changes if we rotate the direction* $\mathbf{d}$ *as indicated by the circular arrow.* ℓ_1 *is the line through the origin and* v*;* ℓ_2 *contains the origin and is orthogonal to* ℓ_1*. If* $\mathbf{d}$ *rotates in clockwise order from* ℓ_1 *to* ℓ_2*, then the distance* $d(v, \ell_0(\mathbf{d}))$ *increases (indicated by the +). If* $\mathbf{d}$ *rotates in clockwise order from* ℓ_2 *to* ℓ_1*, then this distance decreases (indicated by the −).*

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

3 Details of the algorithm

In the following subsections, we describe Steps 1, 3, and 4 of our algorithm in more detail.

3.1 Step 1

Let v be any vertex of $\mathcal{P}$, and let $\mathbf{d} = (d_1, d_2)$ be any direction such that $d_2 \geq 0$. Note that $d_2 = \sqrt{1 - d_1^2}$. We denote by $\mathbf{d}^\perp$ the direction that is orthogonal to $\mathbf{d}$, and that is to the left of the vector $\mathbf{d}$. Hence, $\mathbf{d}^\perp$ has coordinates $(-d_2, d_1)$. (See the left part of Figure 2.)

We write the coordinates of vertex v as (v_1, v_2). Moreover, the vector from the origin to v is denoted by $\mathbf{v}$. The length of this vector is denoted by $\|\mathbf{v}\|$. Recall that, by Assumption 1, v is not equal to the origin.

The distance $d(v, \ell_0(\mathbf{d}))$ between v and the line $\ell_0(\mathbf{d})$ is equal to

$$\begin{aligned} d(v, \ell_0(\mathbf{d})) &= |\mathbf{v} \cdot \mathbf{d}^\perp| \\ &= |-v_1 d_2 + v_2 d_1| \end{aligned}$$

$$= \left| v_2 d_1 - v_1 \sqrt{1 - d_1^2} \right| .$$

Hence, $d(v, \ell_0(\mathbf{d}))$ is a multiple of δ if and only if $|\mathbf{v} \cdot \mathbf{d}^\perp|/\delta$ is an integer. If we denote this integer by j, then $j \geq 0$, and

$$j = \frac{|\mathbf{v} \cdot \mathbf{d}^\perp|}{\delta} \leq \frac{\|\mathbf{v}\|}{\delta}.$$

Therefore, we implement Step 1 as follows. For each vertex v of $\mathcal{P}$, and each integer j, $0 \leq j \leq \|\mathbf{v}\|/\delta$, we solve the equation

$$\left| v_2 d_1 - v_1 \sqrt{1 - d_1^2} \right| = j\delta \tag{1}$$

for d_1. Since we only consider directions $\mathbf{d} = (d_1, d_2)$ for which $d_2 \geq 0$, a simple geometric analysis shows that this equation has at most two solutions d_1. (See the right part of Figure 2.)

Equation (1) reduces to two quadratic equations, depending on the relative magnitude of $v_2 d_1$ and $v_1\sqrt{1 - d_1^2}$. A straightforward calculation shows that we get the following solutions for d_1.

1. Each of the two values

$$d_1 = \frac{v_2 j\delta \pm v_1 \sqrt{\|\mathbf{v}\|^2 - j^2\delta^2}}{\|\mathbf{v}\|^2}$$

   for which $v_2 d_1 \geq v_1\sqrt{1 - d_1^2}$, and for which (1) holds.

2. Each of the two values

$$d_1 = \frac{-v_2 j\delta \pm v_1 \sqrt{\|\mathbf{v}\|^2 - j^2\delta^2}}{\|\mathbf{v}\|^2}$$

   for which $v_2 d_1 < v_1\sqrt{1 - d_1^2}$, and for which (1) holds.

For each solution d_1 obtained, we compute the corresponding value $d_2 = \sqrt{1 - d_1^2}$, and add the direction $\mathbf{d} = (d_1, d_2)$ to the set $\mathbf{D}$.

We have seen already that for each vertex v, and each integer j, at most two directions $\mathbf{d}$ are added to $\mathbf{D}$. Therefore, we have

$$|\mathbf{D}| \leq \sum_v 2\left(1 + \frac{\|\mathbf{v}\|}{\delta}\right) \leq 2n\left(1 + \frac{\max_v \|\mathbf{v}\|}{\delta}\right).$$

The overall time for Step 1 is $O(n + |\mathbf{D}|)$, which is $O(|\mathbf{D}|)$, because $|\mathbf{D}| \geq n$.

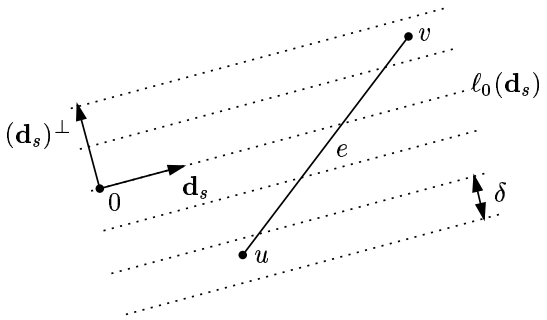

Figure 3: *The number I_e of lines in $\mathcal{L}(\mathbf{d}_s)$ that intersect edge e is equal to* $|\lfloor \mathbf{v} \cdot (\mathbf{d}_s)^\perp/\delta \rfloor - \lfloor \mathbf{u} \cdot (\mathbf{d}_s)^\perp/\delta \rfloor| = |1 - (-3)| = 4$.

3.2 Step 3

Let $\mathbf{d}_s$ be a direction that is not in the set $\mathbf{D}$. In this section, we show how to compute the value of $H(\mathbf{d}_s)$.

Recall that $\ell_0(\mathbf{d}_s)$ denotes the line through the origin having direction $\mathbf{d}_s$. Consider the set $\mathcal{L}(\mathbf{d}_s)$ of lines that are parallel to $\ell_0(\mathbf{d}_s)$ and whose distances to $\ell_0(\mathbf{d}_s)$ are multiples of δ. The value of $H(\mathbf{d}_s)$ is equal to the number of line segments of positive length in the intersection of the polygon $\mathcal{P}$ with the set $\mathcal{L}(\mathbf{d}_s)$. The endpoints of any such line segment are on the boundary of $\mathcal{P}$. Hence, if we count the total number of intersection points between $\mathcal{P}$ and the lines in $\mathcal{L}(\mathbf{d}_s)$, then we get twice the value of $H(\mathbf{d}_s)$. (Here we use the fact that no vertex of $\mathcal{P}$ is contained in any line of $\mathcal{L}(\mathbf{d}_s)$.)

Consider any edge $e = (u, v)$ of the polygon $\mathcal{P}$. Let I_e be the number of lines in $\mathcal{L}(\mathbf{d}_s)$ that intersect e. If e is parallel to $\mathbf{d}_s$, then no line of $\mathcal{L}(\mathbf{d}_s)$ intersects e, i.e., $I_e = 0$. (Again, we use the fact that $\mathbf{d}_s \notin \mathbf{D}$, i.e., edge e is not contained in any line of $\mathcal{L}(\mathbf{d}_s)$.) So assume that e is not parallel to $\mathbf{d}_s$. The signed distance between u and the line $\ell_0(\mathbf{d}_s)$ is equal to $\mathbf{u} \cdot (\mathbf{d}_s)^\perp$. Similarly, the signed distance between v and $\ell_0(\mathbf{d}_s)$ is equal to $\mathbf{v} \cdot (\mathbf{d}_s)^\perp$. From this it follows that

$$I_e = \left| \left\lfloor \frac{\mathbf{v} \cdot (\mathbf{d}_s)^\perp}{\delta} \right\rfloor - \left\lfloor \frac{\mathbf{u} \cdot (\mathbf{d}_s)^\perp}{\delta} \right\rfloor \right|.$$

See Figure 3 for an illustration.

Hence, in Step 3 of the algorithm, we compute the value of I_e for all edges e of the polygon $\mathcal{P}$. Then we compute the value of $H(\mathbf{d}_s)$ for direction $\mathbf{d}_s$ as

$$H(\mathbf{d}_s) = \frac{1}{2} \sum_e I_e.$$

The overall time for Step 3 is $O(n)$.

3.3 Step 4

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

Below, we will consider all possible cases that can occur. For each case, we will indicate how $H(\mathbf{d}_0)$ can be obtained from $H(\mathbf{d}_{-\epsilon})$, and how $H(\mathbf{d}_\epsilon)$ can be obtained from $H(\mathbf{d}_0)$.

Let v be any vertex of $\mathcal{P}$ that corresponds to direction $\mathbf{d}_0$. That is, the distance $d(v, \ell_0(\mathbf{d}_0))$ is a multiple of δ. (There may be more than one vertex that corresponds to $\mathbf{d}_0$. At the end of this subsection, we will indicate how this is handled.) Let v_p and v_s be the predecessor and successor vertices of v, respectively. Note that the interior of the polygon $\mathcal{P}$ is to the left of the directed edges (v_p, v) and (v, v_s).

We distinguish two cases, depending on whether the points v, $v + d_0$, and v_p are collinear or not, and/or the points v, $v + d_0$, and v_s are collinear or not, i.e., whether or not v_p and/or v_s is on the line through v with direction $\mathbf{d}_0$.

Case 1: The three points v, $v + d_0$, and v_p are not collinear, and the three points v, $v + d_0$, and v_s are not collinear.

In this case, neither of the two points v_p and v_s is on the line through vertex v having direction $\mathbf{d}_0$.

When $\mathbf{d}$ passes through $\mathbf{d}_0$, there are sixteen cases, depending on (i) the position of the vertex v with respect to the line through the origin that contains the vector $\mathbf{d}_0^\perp$, and (ii) the positions of the vertices v_p and v_s with respect to the line through vertex v having direction $\mathbf{d}_0$. All these cases are given in Table 1. The fourth column indicates how $H(\mathbf{d}_0)$ is obtained from $H(\mathbf{d}_{-\epsilon})$, whereas the fifth column indicates how $H(\mathbf{d}_\epsilon)$ is obtained from $H(\mathbf{d}_0)$.

We illustrate Table 1 by verifying the tenth line. In this case, we have $(0, d_0^\perp, v) = \mathrm{C}$, $(v, v + d_0, v_p) = \mathrm{L}$, and $(v, v + d_0, v_s) = \mathrm{L}$. This means that

1. the three points 0, $d_0^\perp$, and v are collinear, and the two vectors $\mathbf{d}_0^\perp$ and $\mathbf{v}$ point into the same direction,

$(0, d_0^\perp, v)$	$(v, v+d_0, v_p)$	$(v, v+d_0, v_s)$	$H(\mathbf{d}_0)$	$H(\mathbf{d}_\epsilon)$
R	R	R	$+0$	$+1$
R	L	L	-1	$+0$
R	R	L	$+0$	$+0$
R	L	R	$+0$	$+0$
L	R	R	-1	$+0$
L	L	L	$+0$	$+1$
L	R	L	$+0$	$+0$
L	L	R	$+0$	$+0$
C	R	R	$+0$	$+0$
C	L	L	-1	$+1$
C	R	L	$+0$	$+0$
C	L	R	$+0$	$+0$
-C	R	R	-1	$+1$
-C	L	L	$+0$	$+0$
-C	R	L	$+0$	$+0$
-C	L	R	$+0$	$+0$

Table 1: *The sixteen cases when* $d(v, \ell_0(\mathbf{d}_0)) \neq d(v_p, \ell_0(\mathbf{d}_0))$ *and* $d(v, \ell_0(\mathbf{d}_0)) \neq d(v_s, \ell_0(\mathbf{d}_0))$. *L and R indicate that the triple of points form a left-turn and right-turn, respectively; C indicates that the three points* 0, $d_0^\perp$, *and* v *are collinear, and the two vectors* $\mathbf{d}_0^\perp$ *and* $\mathbf{v}$ *point into the same direction; -C indicates that the three points* 0, $d_0^\perp$, *and* v *are collinear, and the two vectors* $\mathbf{d}_0^\perp$ *and* $\mathbf{v}$ *have opposite directions. The two rightmost columns indicate how* $H(\mathbf{d})$ *changes.* $\mathbf{d}_\epsilon$ *denotes the direction obtained by rotating* $\mathbf{d}_0$ *by an infinitesimally small angle in clockwise direction.*

---

2. the triple of points $(v, v+d_0, v_p)$ forms a left-turn, and

3. the triple of points $(v, v+d_0, v_s)$ forms a left-turn.

(See Figure 4.) The two rightmost columns indicate that $H(\mathbf{d}_0) = H(\mathbf{d}_{-\epsilon}) - 1$, and $H(\mathbf{d}_\epsilon) = H(\mathbf{d}_0) + 1$. Let us check if this is correct.

Let j be the integer such that $d(v, \ell_0(\mathbf{d}_0)) = j\delta$. For any direction $\mathbf{d}$, let $\ell_j(\mathbf{d})$ be the line that has direction $\mathbf{d}$ and whose distance to $\ell_0(\mathbf{d})$ is equal to $j\delta$. (See Figure 4. In this figure, the interior of $\mathcal{P}$ is above the edges (v_p, v) and (v, v_s). The discussion below also applies if v_p and v_s are interchanged, in which case the interior of $\mathcal{P}$ is below these two edges.)

Consider what happens if the direction $\mathbf{d}$ rotates in clockwise order, and passes through $\mathbf{d}_0$. For direction $\mathbf{d}_{-\epsilon}$, and in a sufficiently small neighborhood of vertex v, the intersection of line $\ell_j(\mathbf{d}_{-\epsilon})$ with the polygon $\mathcal{P}$ is a line

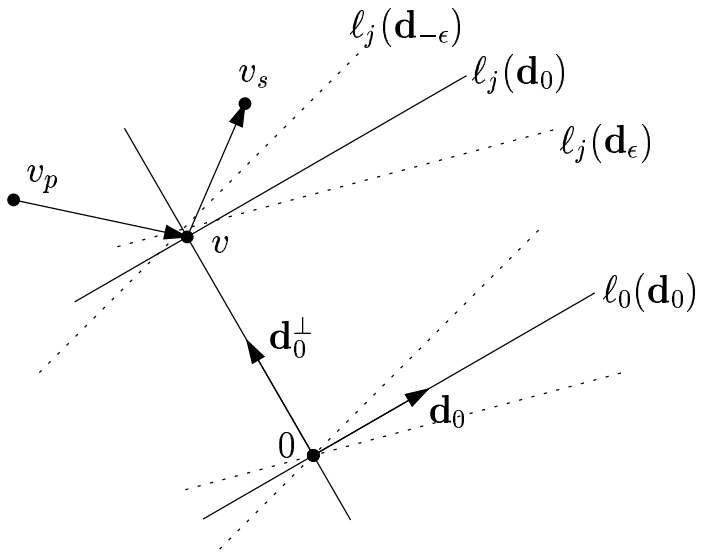

Figure 4: *Illustrating the tenth line of Table 1.*

---

segment of positive length. For direction $\mathbf{d}_0$, and in a sufficiently small neighborhood of v, the intersection of line $\ell_j(\mathbf{d}_0)$ with $\mathcal{P}$ is a single point, viz. the vertex v. Hence, we indeed have $H(\mathbf{d}_0) = H(\mathbf{d}_{-\epsilon}) - 1$. For direction $\mathbf{d}_\epsilon$, and in a sufficiently small neighborhood of v, the intersection of line $\ell_j(\mathbf{d}_\epsilon)$ with $\mathcal{P}$ is again a line segment of positive length. Therefore, we have $H(\mathbf{d}_\epsilon) = H(\mathbf{d}_0) + 1$.

Case 2: The three points v, $v + d_0$, and v_p are collinear, or the three points v, $v + d_0$, and v_s are collinear.

First note that, by Assumption 1 (part 2), exactly one of these two possibilities occurs. 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 without loss of generality that the triple of points $(u, u + d_0^\perp, v)$ forms a right-turn. (Otherwise, we swap u and v.) 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, depending on (i) the positions of the vertices u and v with respect to the line through the origin that contains the vector $\mathbf{d}_0^\perp$, (ii) the positions of the vertices u' and v' with respect to the line through vertex v having direction $\mathbf{d}_0$, and (iii) whether the interior of $\mathcal{P}$ is to the left or right of the directed edge (u, v). (Note that because of the possible renaming of u and v, the edge (u, v) may occur as (v, u) in the polygon $\mathcal{P}$.) All cases are given in Table 2. As in Table 1, the

two rightmost columns indicate how $H(\mathbf{d}_0)$ is obtained from $H(\mathbf{d}_{-\epsilon})$, and how $H(\mathbf{d}_\epsilon)$ is obtained from $H(\mathbf{d}_0)$.

We illustrate Table 2 by verifying the third line. In this case,

$(0, d_0^\perp, u)$	$(0, d_0^\perp, v)$	$(u, u+d_0, u')$	$(v, v+d_0, v')$	$u \leftrightarrow v$	$H(\mathbf{d}_0)$	$H(\mathbf{d}_\epsilon)$
R	R	R	R	$\rightarrow$	$+0$	$+1$
R	R	R	R	$\leftarrow$	$+1$	$+0$
R	R	L	L	$\rightarrow$	$+0$	-1
R	R	L	L	$\leftarrow$	-1	$+0$
R	R	R	L	$\rightarrow$	$+0$	$+0$
R	R	R	L	$\leftarrow$	$+0$	$+0$
R	R	L	R	$\rightarrow$	$+0$	$+0$
R	R	L	R	$\leftarrow$	$+0$	$+0$
C	R	R	R	$\rightarrow$	$+0$	$+1$
C	R	R	R	$\leftarrow$	$+1$	$+0$
C	R	L	L	$\rightarrow$	$+0$	$+0$
C	R	L	L	$\leftarrow$	-1	$+1$
C	R	R	L	$\rightarrow$	$+0$	$+0$
C	R	R	L	$\leftarrow$	$+0$	$+0$
C	R	L	R	$\rightarrow$	$+0$	$+1$
C	R	L	R	$\leftarrow$	$+0$	$+1$
-C	R	R	R	$\rightarrow$	-1	$+1$
-C	R	R	R	$\leftarrow$	$+0$	$+0$
-C	R	L	L	$\rightarrow$	$+0$	-1
-C	R	L	L	$\leftarrow$	-1	$+0$
-C	R	R	L	$\rightarrow$	-1	$+0$
-C	R	R	L	$\leftarrow$	-1	$+0$
-C	R	L	R	$\rightarrow$	$+0$	$+0$
-C	R	L	R	$\leftarrow$	$+0$	$+0$
L	L	R	R	$\rightarrow$	-1	$+0$
L	L	R	R	$\leftarrow$	$+0$	-1
L	L	L	L	$\rightarrow$	$+1$	$+0$
L	L	L	L	$\leftarrow$	$+0$	$+1$
L	L	R	L	$\rightarrow$	$+0$	$+0$
L	L	R	L	$\leftarrow$	$+0$	$+0$
L	L	L	R	$\rightarrow$	$+0$	$+0$
L	L	L	R	$\leftarrow$	$+0$	$+0$

Table 2; continued on next page

Table 2; continued from previous page						
$(0, d_0^\perp, u)$	$(0, d_0^\perp, v)$	$(u, u+d_0, u')$	$(v, v+d_0, v')$	$u \leftrightarrow v$	$H(\mathbf{d}_0)$	$H(\mathbf{d}_\epsilon)$
L	C	R	R	$\rightarrow$	-1	$+0$
L	C	R	R	$\leftarrow$	$+0$	-1
L	C	L	L	$\rightarrow$	$+0$	$+0$
L	C	L	L	$\leftarrow$	-1	$+1$
L	C	R	L	$\rightarrow$	-1	$+0$
L	C	R	L	$\leftarrow$	-1	$+0$
L	C	L	R	$\rightarrow$	$+0$	$+0$
L	C	L	R	$\leftarrow$	$+0$	$+0$
L	-C	R	R	$\rightarrow$	-1	$+1$
L	-C	R	R	$\leftarrow$	$+0$	$+0$
L	-C	L	L	$\rightarrow$	$+1$	$+0$
L	-C	L	L	$\leftarrow$	$+0$	$+1$
L	-C	R	L	$\rightarrow$	$+0$	$+0$
L	-C	R	L	$\leftarrow$	$+0$	$+0$
L	-C	L	R	$\rightarrow$	$+0$	$+1$
L	-C	L	R	$\leftarrow$	$+0$	$+1$
L	R	R	R	$\rightarrow$	-1	$+1$
L	R	R	R	$\leftarrow$	$+0$	$+0$
L	R	L	L	$\rightarrow$	$+0$	$+0$
L	R	L	L	$\leftarrow$	-1	$+1$
L	R	R	L	$\rightarrow$	-1	$+0$
L	R	R	L	$\leftarrow$	-1	$+0$
L	R	L	R	$\rightarrow$	$+0$	$+1$
L	R	L	R	$\leftarrow$	$+0$	$+1$

Table 2: *The fifty six cases when $d(u, \ell_0(\mathbf{d}_0)) = d(v, \ell_0(\mathbf{d}_0))$. We assume that the triple of points $(u, u + d_0^\perp, v)$ forms a right-turn. The meaning of L, R, C, and -C, and the two rightmost columns is as in Table 1. $\rightarrow$ means that v is the successor of u; $\leftarrow$ means that u is the successor of v.*

---

1. the triple of points $(0, d_0^\perp, u)$ forms a right-turn,
2. the triple of points $(0, d_0^\perp, v)$ forms a right-turn,
3. the triple of points $(u, u + d_0, u')$ forms a left-turn,
4. the triple of points $(v, v + d_0, v')$ forms a left-turn, and

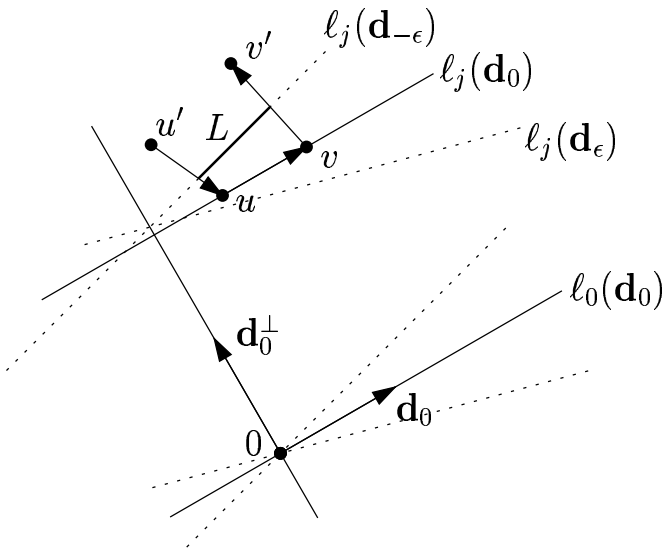

Figure 5: *Illustrating the third line of Table 2.*

---

5. v is the successor of u. (Recall that we assume that the triple of points $(u, u + d_0^\perp, v)$ forms a right-turn.)

(See Figure 5.) According to the two rightmost columns, we have $H(\mathbf{d}_0) = H(\mathbf{d}_{-\epsilon})$, and $H(\mathbf{d}_\epsilon) = H(\mathbf{d}_0) - 1$. Below, we argue why this is correct.

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 $j\delta$. (See Figure 5. Note that the interior of $\mathcal{P}$ is to the left of the directed edges (u', u), (u, v), and (v, v').)

Consider what happens if the direction $\mathbf{d}$ 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 the polygon $\mathcal{P}$ 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 the segment 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 $H(\mathbf{d}_\epsilon) = H(\mathbf{d}_0) - 1$.

Now we are ready to describe how Step 4 of the algorithm is implemented. Recall that

$$\mathbf{d}^0 \prec \mathbf{d}^1 \prec \ldots \prec \mathbf{d}^{m-1}$$

denotes the sorted sequence of distinct directions of the set $\mathbf{D}$. Consider the direction $\mathbf{d}_s \notin \mathbf{D}$ that was chosen in Step 3. Let k be the index such that $\mathbf{d}^{k-1} \prec \mathbf{d}_s \prec \mathbf{d}^k$. We visit the directions of $\mathbf{D}$ in the order

$$\mathbf{d}^k, \mathbf{d}^{k+1}, \ldots, \mathbf{d}^{m-1}, \mathbf{d}^0, \ldots, \mathbf{d}^{k-1}.$$

Let $\mathbf{d}_0$ be any such direction. We consider all vertices v of $\mathcal{P}$ for which the distance $d(v, \ell_0(\mathbf{d}_0))$ is a multiple of δ. For each such vertex v, we do the following.

1. If we are in Case 1, then we update $H(\mathbf{d})$ according to Table 1.
2. If we are in Case 2, then we update $H(\mathbf{d})$ according to Table 2. In this case, we do not update $H(\mathbf{d})$ for the neighbor u of v for which $d(u, \ell_0(\mathbf{d}_0)) = d(v, \ell_0(\mathbf{d}_0))$.

For each direction $\mathbf{d}_0 \in \mathbf{D}$, and each vertex v, for which $d(v, \ell_0(\mathbf{d}_0))$ is a multiple of δ, we spend $O(1)$ time to update $H(\mathbf{d})$. Hence, the overall time for Step 4 is $O(|\mathbf{D}|)$.

3.4 Complexity of the algorithm

We summarize the running times of the five steps of our algorithm, as given in Section 2.1. Step 1 takes $O(|\mathbf{D}|)$ time. In Step 2, we sort the elements of $\mathbf{D}$, which takes $O(|\mathbf{D}| \log |\mathbf{D}|)$ time. Step 3 takes $O(n)$ time. Finally, Steps 4 and 5 take $O(|\mathbf{D}|)$ time. Since $n \leq |\mathbf{D}| \leq 2n(1 + \max_v \|\mathbf{v}\|/\delta)$, we have proved the following result.

Theorem 1 *Given a simple polygon $\mathcal{P}$, possibly with holes, having n vertices, Problem 1 can be solved in $O(Cn \log(Cn))$ time, where*

$$C = 1 + \frac{\max_v \|\mathbf{v}\|}{\delta}.$$

4 Experimental results

We have implemented the algorithm of Section 2 in C++ using LEDA 4.1 [10, 11]. We now discuss the implementation and experiments in more detail.

It is well-known that implementations of geometric algorithms using double-precision are extremely prone to roundoff errors. Such errors may occur, e.g., in orientation tests, in which we have to decide whether three given points form a left-turn, a right-turn, or are collinear. These orientation tests are heavily used in the algorithm. Clearly, if they are handled incorrectly, the

output of the algorithm will be incorrect. Furthermore, recall that the algorithm works with unit-length vectors $\mathbf{d} = (d_1, d_2)$ for which $d_2 = \sqrt{1 - d_1^2}$. Again, in order to get an exact result, we need an implementation that can handle the square root operation exactly. Therefore, we implemented the algorithm using the LEDA number type `real`, which implements exact arithmetic with algebraic numbers. (See Burnikel *et al.* [3].) Hence, this program computes the exact minimum of the function $H(\mathbf{d})$.

We ran our program on seven real-world polyhedral models obtained from Stratasys, Inc. For each model, we chose five z-coordinates, and used their program Quickslice to slice the model using horizontal planes corresponding to these z-coordinates. This resulted in a set of 35 polygons, whose coordinates are in the range $[0, 12]$. We ran the program on a SUN Ultra (400 MHz, 512 MByte RAM), using $\delta = 0.1$.

The results of our experiments are given in Table 3. Each line gives the result for one polygon. For example, the polygon `4501005-layer-z-1.009.ssl` in the first line is obtained by slicing the model `4501005.stl` using a horizontal plane at height $z = 1.009$. The resulting polygon has $n = 257$ vertices. For each polygon, we measured the following information.

1. *num*: the minimum value of the function $H(\mathbf{d})$, as computed by the program.

2. $|\mathbf{D}|$: the number of elements of the set $\mathbf{D}$. Recall that this is the number of directions for which the program updates the value of $H(\mathbf{d})$.

3. *time*: the time (in seconds) taken by the program.

We also implemented the projection-based heuristic from [4], using the number type `double` of C++ and ran it on the above examples. We discovered that the number of hatching segments generated by the heuristic was always very close to the optimal number generated by the algorithm of Section 2 (in our experiments, the heuristic generated at most fourteen percent more hatching segments than the exact algorithm.) Thus, to the extent that we have tested it, the heuristic appears to work very well on practical inputs. Note that without our exact algorithm, it would not have been possible to support this claim.

Since the operations on LEDA `reals` are relatively slow, the exact program takes much longer than the heuristic. Another reason for this is the fact that the set $\mathbf{D}$ is quite large. The running time of the heuristic does not depend on δ or the size of $\mathbf{D}$; it only depends on n. On the above examples it ran in under one second.

5 Concluding remarks

We have given an efficient algorithm that solves an important problem arising in Layered Manufacturing and path planning of zig-zag paths for milling machines.

The running time of our algorithm heavily depends on the ratio of the maximum distance of any vertex to the origin, and the width δ of the laser path. We leave open the problem of deciding if Problem 1 can be solved exactly in a time that only depends on n.

Our algorithm, as presented in this paper, computes an optimal hatching direction for one single polygon. Recall from Section 1 that in Layered Manufacturing, a prototype of a three-dimensional polyhedron is built by successively hatching the different layers. In practice, one global hatching direction $\mathbf{d}$ is chosen, and all odd-numbered layers are hatched in direction $\mathbf{d}$, whereas all even-numbered layers are hatched in direction $\mathbf{d}^\perp$. Our algorithm can easily be extended so that it computes the globally optimal hatching direction: For each i, let $H_i(\mathbf{d})$ be defined as the number of segments that the laser draws when hatching the i-th layer in direction $\mathbf{d}$. We compute for each layer the set of critical directions and take the union of these sets for all layers to get a global set of critical directions. We then do the previously-described rotational sweep simultaneously over all layers, using this global set of critical directions and compute a direction $\mathbf{d}$ for which

$$\sum_i H_{2i+1}(\mathbf{d}) + \sum_i H_{2i}(\mathbf{d}^\perp)$$

is minimum.

In this paper we have assumed that the paths of the laser are on parallel lines whose distances are multiples of δ to a line containing the origin. Hence, the output of our algorithm depends on the coordinate system. For any point $x \in \mathbb{R}^2$, define

$$H_x(\mathbf{d}) := \sum_{\ell \in \mathcal{L}_x(\mathbf{d})} |S_\ell|,$$

where $\mathcal{L}_x(\mathbf{d})$ is defined as in Section 1.1, but with the coordinate axes translated such that the origin is at point x. We leave open the problem of designing an algorithm that computes a point x and a direction $\mathbf{d}$ for which $H_x(\mathbf{d})$ is minimum.

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] P. K. Agarwal and P. K. Desikan. Approximation algorithms for layered manufacturing. In *Proc. 11th ACM-SIAM Sympos. Discrete Algorithms*, pages 528–537, 2000.

[2] B. Asberg, G. Blanco, P. Bose, J. Garcia-Lopez, M. Overmars, G. Toussaint, G. Wilfong, and B. Zhu. Feasibility of design in stereolithography. *Algorithmica*, 19:61–83, 1997.

[3] C. Burnikel, R. Fleischer, K. Mehlhorn, and S. Schirra. Efficient exact geometric computation made easy. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 341–350, 1997.

[4] M. C. Hon, R. Janardan, J. Schwerdt, and M. Smid. Minimizing the total projection of a set of vectors, with applications to layered manufacturing. `http://www.cs.umn.edu/~janardan/min-proj.ps`.

[5] P. F. Jacobs. *Rapid Prototyping & Manufacturing: Fundamentals of StereoLithography.* McGraw-Hill, New York, 1992.

[6] J. Majhi, R. Janardan, J. Schwerdt, M. Smid, and P. Gupta. Minimizing support structures and trapped area in two-dimensional layered manufacturing. *Comput. Geom. Theory Appl.*, 12:241–267, 1999.

[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] J. Majhi, R. Janardan, M. Smid, and J. Schwerdt. Multi-criteria geometric optimization problems in layered manufacturing. In *Proc. 14th Annu. ACM Sympos. Comput. Geom.*, pages 19–28, 1998.

[9] S. McMains and C. Séquin. A coherent sweep plane slicer for layered manufacturing. In *Proc. 5th ACM Symposium on Solid Modeling and Applications*, pages 285–295, 1999.

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

[11] K. Mehlhorn, S. Näher, M. Seel, and C. Uhrig. *The LEDA User Manual*. Max-Planck-Institute for Computer Science, Saarbrücken, Germany, `http://www.mpi-sb.mpg.de/LEDA/MANUAL/MANUAL.html`.

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

[13] J. Schwerdt, M. Smid, R. Janardan, and E. Johnson. Protecting critical facets in layered manufacturing: implementation and experimental results. In *Proc. 2nd Workshop on Algorithm Engineering and Experiments*, pages 43–57, 2000.

[14] J. Schwerdt, M. Smid, R. Janardan, E. Johnson, and J. Majhi. Protecting critical facets in layered manufacturing. *Comput. Geom. Theory Appl.*, 16:187–210, 2000.

[15] J. Schwerdt, M. Smid, J. Majhi, and R. Janardan. Computing the width of a three-dimensional point set: an experimental study. *ACM Journal of Experimental Algorithmics*, 4, Article 8, 1999.

polygon	n	*num*	$\lvert\mathbf{D}\rvert$	*time*
4501005-layer-z-1.009.ssl	257	84	34715	903
4501005-layer-z-3.369.ssl	384	71	41054	1093
4501005-layer-z-5.019.ssl	322	52	32652	875
4501005-layer-z-6.099.ssl	290	44	29568	790
4501005-layer-z-7.329.ssl	187	33	18031	479
cover-5-layer-z-0.049.ssl	22	13	2838	76
cover-5-layer-z-3.199.ssl	47	32	3737	98
cover-5-layer-z-4.529.ssl	47	39	3153	83
cover-5-layer-z-5.419.ssl	22	39	1746	46
cover-5-layer-z-5.969.ssl	27	105	3119	83
eaton_sp-layer-z-0.229.ssl	165	18	11353	308
eaton_sp-layer-z-1.239.ssl	156	50	9822	265
eaton_sp-layer-z-1.949.ssl	209	52	11837	321
eaton_sp-layer-z-2.859.ssl	106	27	5064	133
eaton_sp-layer-z-3.199.ssl	90	16	4736	123
frame_29-layer-z-0.009.ssl	156	18	15070	406
frame_29-layer-z-1.509.ssl	214	30	20714	557
frame_29-layer-z-2.509.ssl	225	22	21777	586
frame_29-layer-z-3.919.ssl	355	57	36161	971
frame_29-layer-z-4.419.ssl	890	201	93028	2485
impller-layer-z-0.009.ssl	52	15	4132	110
impller-layer-z-0.669.ssl	405	96	33893	904
impller-layer-z-1.489.ssl	412	110	33746	904
impller-layer-z-2.469.ssl	559	120	45803	1223
impller-layer-z-2.999.ssl	133	15	10575	283
mj-layer-z-0.029.ssl	32	7	1450	38
mj-layer-z-0.529.ssl	49	14	2113	56
mj-layer-z-1.509.ssl	52	14	2256	62
mj-layer-z-2.029.ssl	64	14	3416	93
mj-layer-z-2.489.ssl	88	18	5054	136
sa600280-layer-z-0.779.ssl	238	35	11916	310
sa600280-layer-z-1.629.ssl	300	50	22142	566
sa600280-layer-z-2.209.ssl	445	56	31063	799
sa600280-layer-z-3.069.ssl	346	46	22824	589
sa600280-layer-z-4.539.ssl	187	15	11693	300

Table 3: *Results of our experiments.* n *gives the number of vertices of the polygon;* *num* *gives the minimum value of* $H(\mathbf{d})$*, as computed by the program;* $|\mathbf{D}|$ *gives the number of directions for which the program updates the value of* $H(\mathbf{d})$*; and* *time* *gives the time of the program, in seconds.*