Minimizing support structures and trapped area in two-dimensional layered manufacturing [⋆]

Jayanth Majhi [1]

Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, OR 97070, U.S.A.

Ravi Janardan [2]

Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, U.S.A.

Jörg Schwerdt and Michiel Smid [3]

Department of Computer Science, University of Magdeburg, D-39106 Magdeburg, Germany.

Prosenjit Gupta [4]

Delsoft (India) Pvt. Ltd.

Abstract

Algorithms are given for the two-dimensional versions of optimization problems arising in layered manufacturing, where a polygonal object is built by slicing its CAD model and manufacturing the slices successively. The problems considered are minimizing (i) the contact-length between the supports and the manufactured object, (ii) the area of the support structures used, and (iii) the area of the so-called trapped regions—factors that affect the cost and quality of the process.

Key words: Layered manufacturing, computational geometry, optimization.

1 Introduction

In *layered manufacturing*, a physical prototype of a 3D object is built from a (virtual) CAD model by orienting and slicing the model with parallel planes and then manufacturing the slices one by one, each on top of the previous one. Layered manufacturing is the basis of an emerging technology called *Rapid Prototyping and Manufacturing* (RP&M). This technology, which is used extensively in the automotive, aerospace, and medical industries, accelerates dramatically the time it takes to bring a product to the market because it allows the designer to create rapidly a physical version of the CAD model (literally on the desktop) and to "feel and touch" it, thereby detecting and correcting flaws in the model early on in the design cycle.

Although there are many types of layered manufacturing processes, the basic principle underlying them all is as outlined above. For concreteness, we will briefly describe one such method, called *StereoLithography*, which dominates the RP&M market, see [10]. (In fact, the recent report of the Computational Geometry Task Force explicitly identifies this process as one where geometric techniques could play a significant role [5, page 31].)

The input to the StereoLithography process is a surface triangulation of the CAD model in a format called STL. The triangulated model is oriented suitably, sliced by xy-parallel planes, and then built slice by slice in the positive z direction, as follows. In essence, the StereoLithography Apparatus (SLA) consists of a vat of photocurable liquid resin, a platform, and a laser. Initially, the platform is below the surface of the resin at a depth equal to the slice thickness. The laser traces out the contour of the first slice on the surface and then hatches the interior, which hardens to a depth equal to the slice thickness. In this way, the first slice is created and it rests on the platform.

⋆ Portions of this work appear, in preliminary form, in [11]. The present paper and a companion paper [12] extend upon and improve the results presented in [11].

[1] E-mail: `jayanth_majhi@mentorg.com`. This work was done while the author was at the Department of Computer Science & Engineering at the University of Minnesota. Research supported in part by a Grant–in–Aid of Research from the Graduate School of the University of Minnesota and by NSF grant CCR-9712226.

[2] E-mail: `janardan@cs.umn.edu`. Research supported in part by a Grant–in–Aid of Research from the Graduate School of the University of Minnesota and by NSF grant CCR-9712226.

[3] E-mail: {`schwerdt`,`michiel`}`@isg.cs.uni-magdeburg.de`. Part of this work was done while MS was at the Department of Computer Science, King's College London, UK.

[4] E-mail: `pgupta@delsoft.com`. Part of this work was done while the author was at the Max-Planck-Institut für Informatik, Saarbrücken, Germany, and at the Department of Computer Science & Engineering at the University of Minnesota.

Next, the platform is lowered by the slice thickness and the just-vacated region is re-coated with resin. The second slice is then built in the same way. Ideally, each slice after the first one should rest in its entirety on the previous one. In general, however, portions of a slice can overhang the previous slice and so additional structures, called *supports*, are needed to hold up the overhangs. Supports are generated automatically during the process itself. Once the solid has been made, it is postprocessed to remove the supports.

During the process, certain regions that are separate from the main body of liquid in the vat will hold resin. These regions are called *trapped regions* and they are undesirable because they slow down the process [10, page 161]. For example, if a coffee mug is built in the vertical direction, then the interior of the mug will hold resin, and the volume of this region is called the *trapped volume*.

A key step in this process is choosing an orientation for the model, i.e., the *build direction*. Among other things, the build direction affects the volume of supports, the surface area of the model that is in contact with the supports, and the trapped volume—factors which impact the cost and quality of the process.

In current systems, the build direction is often chosen by the human operator, based on experience, so that e.g. the amount of supports used and the trapped area is “small”. We seek to design computer algorithms that minimize these quantities automatically and lessen the need for human intervention.

1.1 Two-dimensional layered manufacturing

In this paper, we will consider the two-dimensional version of this problem. Throughout the paper, we denote by $\mathcal{P}$ the polygonal object that we wish to build and by n the number of vertices of $\mathcal{P}$. We assume that $\mathcal{P}$ is in general position, in the sense that no three vertices are collinear. All our results remain valid for arbitrary simple polygons. The algorithms and correctness proofs, however, need some minor modifications.

We let $\mathbf{d}$ denote the *build direction* and refer to notions such as “above” and “below” w.r.t. $\mathbf{d}$. The direction $\mathbf{d}$ can range a full 360° in the plane of $\mathcal{P}$. The criteria “volume” and “contact-area” mentioned earlier are now replaced by “area” and “contact-length”, respectively, as discussed below.

Our motivation for studying the polygon problem was to develop techniques that would be applicable to non-convex polyhedra, which is the actual problem of interest. In principle, our 2D techniques carry over to 3D, but it is not clear at this point how efficient or practical they would be. We are investigating this problem further.

We will design algorithms that compute a direction $\mathbf{d}$ which minimizes one of the following three parameters:

Contact-length of supports: The part of $\mathcal{P}$'s boundary that is in contact with the supports affects the postprocessing time, since the supports that "stick" to $\mathcal{P}$ must be removed. If $\mathcal{P}$ is convex, then this is the total length of the downward-facing edges. If $\mathcal{P}$ is non-convex, then this is the total edge length of the downward-facing edges and portions of certain upward-facing edges.

In Section 2, we show that for any simple polygon, a build direction which minimizes the contact-length of the supports can be computed in time $O(n \log n + np(n))$, where $p(n)$ is the time it takes to minimize a certain function $G(x)$ which is the sum of $\Theta(n)$ terms of the form $d/(1+cx)$, for some constants c and d. If the polygon's edges have only a constant number of different orientations, then this time bound improves to $O(n \log n)$.

Area of supports: The quantity of supports used affects both the building time and the cost. If $\mathcal{P}$ is convex, then the support area is the area of the region lying between $\mathcal{P}$ and the platform, i.e., the region which is bounded below by the platform and above by the downward-facing edges of $\mathcal{P}$. If $\mathcal{P}$ is non-convex, then the problem is more complex, since the supports for some edges may actually be attached to other edges instead of to the platform. (Figure 1 illustrates this.)

In Section 3, we give an algorithm that computes a build direction minimizing the area of the supports, in time $O(n \log n + nq(n))$, where $q(n)$ is the time it takes to minimize a certain function which is similar to $G(x)$ above. Again, if the edges have only a constant number of different orientations, then the running time improves to $O(n \log n)$. (In a preliminary version of this paper, see [11], the running times were $O(n^2 + nq(n))$ and $O(n^2)$, respectively.)

Trapped area: As mentioned before, trapped regions are areas that hold resin separate from the main body of liquid and are undesirable. (We defer a formal definition of trapped regions to Section 4.) In Section 4, we show that a variant of the algorithm of Section 3 can be used to compute a build direction for which the trapped area is minimal.

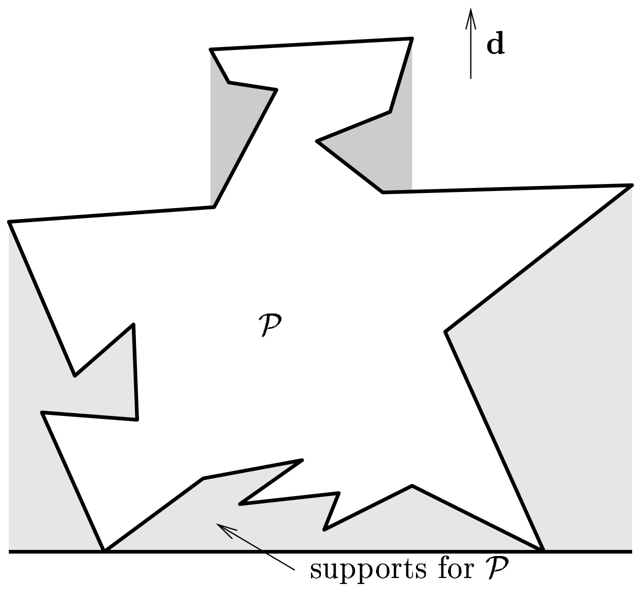

Fig. 1. *Supports for a simple polygon.*

1.2 Related work

Arkin *et al.* [2] solve the following related two-dimensional problem. Given a simple polygon $\mathcal{P}$, compute a monotone or star-shaped polygon that contains $\mathcal{P}$ and that has minimal area. Their algorithm is similar to ours and, in fact, they also have to minimize a function that is similar to our function $G(x)$. One of the differences between our problem and that of Arkin *et al.* is that we have to take care of support regions that are in contact with the platform.

Surprisingly little work has been done by way of efficient and provably optimal geometric optimization algorithms for layered manufacturing. In Asberg *et al.* [3], efficient algorithms are given for deciding if a two- or three-dimensional object can be made by StereoLithography without using supports. In a companion paper [12], we give algorithms for minimizing the contact-area and volume of supports for three-dimensional convex polyhedra.

The problem of generating optimal supports for arbitrary three-dimensional polyhedra is considered in Allen and Dutta [1]. Their algorithm essentially considers only directions parallel and orthogonal to edges of the convex hull. However, this is not optimal. Below, we give an example which shows that with this approach, the approximation factor w.r.t. the optimal solution can be made arbitrarily large.

Let $\mathcal{P}$ be the non-convex polygon in Figure 2. Vertices a and c are slightly above the horizontal line through b. Vertex i (resp. d) is slightly to the right (resp. left) of vertex a (resp. c). Vertex i is slightly above vertex d. Both edges

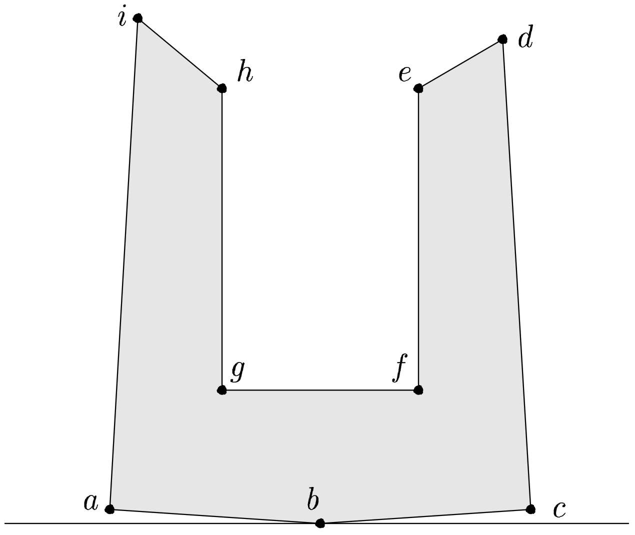

Fig. 2. *An example polygon for which the support area for any build direction that is parallel or orthogonal to an edge of the polygon's convex hull is much larger than the minimum support area.*

---

ef and gh are vertical; edge fg is horizontal. The edges ab and bc are very small compared to the edges ef and gh. Finally, the angle between ai (resp. cd) and the vertical is smaller than the angle between ab (resp. bc) and the horizontal.

For the vertically upwards build direction $\mathbf{d}_v$, the support area is very small (it is the area under edges ab and bc). It can be verified easily that for each build direction that is parallel or orthogonal to an edge of the convex hull of $\mathcal{P}$, the support area is much larger than for build direction $\mathbf{d}_v$. Hence, for any build direction that is parallel or orthogonal to a convex hull edge, the support area is much larger than for the optimal build direction (which need not be vertical).

In closing this section, we note that if $\mathcal{P}$ is a convex polygon with n vertices, then the optimal build direction $\mathbf{d}$ is either parallel or orthogonal to an edge of $\mathcal{P}$. (The proof follows from results in [7].) Hence, in this case, $\mathbf{d}$ can be found in $O(n)$ time.

2 Minimum-contact-length supports for simple polygons

In this section, we consider the problem of computing a direction that minimizes the total *length* of the boundary of a simple polygon to which the supports "stick".

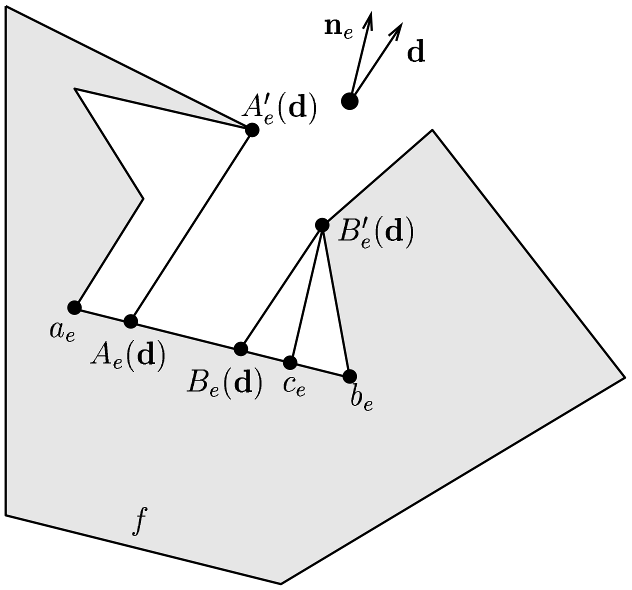

Fig. 3. *Edge e has endpoints a_e and b_e. This edge is attached to supports for direction $\mathbf{d}$. The set $S_e(\mathbf{d})$ consists of two segments $a_eA_e(\mathbf{d})$ and $B_e(\mathbf{d})b_e$. These segments are the parts of edge e that are attached to the supports. Edge f needs support for direction $\mathbf{d}$.*

First, we introduce some notation. Then, we give a precise definition of the problem. Finally, an algorithm for computing the optimal direction is given. Throughout this paper, we measure angles counterclockwise from 0 to 2π.

Henceforth, we let $\mathcal{P}$ be a simple n-vertex polygon. For each edge e of $\mathcal{P}$, let $\mathbf{n}_e$ denote its *outer normal*. Furthermore, let α_e denote the angle between the positive x-axis and the vector $\mathbf{n}_e$. For any direction $\mathbf{d}$, we denote by $\varphi_{\mathbf{d}}$ the angle between the positive x-axis and $\mathbf{d}$.

We say that edge e *needs support for direction* $\mathbf{d}$, if the dot product $\mathbf{n}_e \cdot \mathbf{d}$ is negative. In this case, the *entire* edge e needs support.

Even if an edge e does not need support for a direction $\mathbf{d}$, it may still be a part of some support for this direction, in the following sense. Let $\mathbf{d}$ be a direction, such that $\mathbf{n}_e \cdot \mathbf{d} \geq 0$. Let q be a point on e, such that the ray emanating from q in the direction $\mathbf{d}$ intersects the interior of the polygon $\mathcal{P}$. Then we say that q is *attached to a support for direction* $\mathbf{d}$. Let $S_e(\mathbf{d})$ be the set of all points on e that are attached to a support for direction $\mathbf{d}$. (See Figure 3.) Then $e - S_e(\mathbf{d})$, the set of points on e that are not attached to a support, is connected. To be more precise, the following lemma holds.

Lemma 1 *The set $S_e(\mathbf{d})$ is (i) empty, (ii) consists of one segment on e, one of whose endpoints is an endpoint of e, (iii) consists of two segments on e, and each segment has an endpoint which is an endpoint of e, or (iv) is equal* *to the entire edge e.*

Proof. The set $S_e(\mathbf{d})$ consists of line segments on edge e. It suffices to prove that no such segment has both its endpoints in the interior of e.

Assume there is such a segment, and let s and t be its endpoints. Let a_e and b_e be the endpoints of edge e. Assume w.l.o.g. that a_e, s, t, and b_e appear in this order from left to right along e, and that the direction $\mathbf{d}$ is vertically upwards. There is a point x (resp. y) on e, strictly between a_e and s (resp. t and b_e), such that the ray r_x (resp. r_y) emanating from x (resp. y) in the direction $\mathbf{d}$ does not intersect the polygon $\mathcal{P}$. On the other hand, the ray emanating from s in the direction $\mathbf{d}$ does intersect $\mathcal{P}$; let z be the first intersection. By walking along the boundary of $\mathcal{P}$, from a_e to z, we intersect one of the rays r_x and r_y. This is a contradiction. ∎

We define $l_e(\mathbf{d})$ as the length of edge e, in case e needs support for direction $\mathbf{d}$. Otherwise, $l_e(\mathbf{d})$ is defined as the length of the at most two segments on e that are determined by $S_e(\mathbf{d})$. Hence, $\sum_e l_e(\mathbf{d})$ is equal to the total length of the boundary of $\mathcal{P}$ to which the supports stick, when $\mathcal{P}$ is manufactured along direction $\mathbf{d}$. Therefore, we want to solve the following problem.

Problem 2 *Given a simple polygon $\mathcal{P}$, compute a direction $\mathbf{d}$ which minimizes*

$$L_{\mathcal{P}}(\mathbf{d}) := \sum_e l_e(\mathbf{d}).$$

We need some more notation. (Refer to Figure 3.) Let e be an edge of $\mathcal{P}$, and let $\mathbf{d}$ be a direction such that $S_e(\mathbf{d})$ consists of two segments. Let a_e and b_e denote the two endpoints of e, where a_e is to the left of b_e (w.r.t. direction $\mathbf{d}$), and let $A_e(\mathbf{d})$ and $B_e(\mathbf{d})$ denote the other two endpoints of the segments spanning $S_e(\mathbf{d})$. That is, $S_e(\mathbf{d})$ consists of the two segments $a_eA_e(\mathbf{d})$ and $b_eB_e(\mathbf{d})$. Let $A'_e(\mathbf{d})$ be the vertex of $\mathcal{P}$ that is hit first by shooting a ray from $A_e(\mathbf{d})$ in the direction $\mathbf{d}$. (Note that this ray indeed hits a vertex.) Define $B'_e(\mathbf{d})$ similarly w.r.t. $B_e(\mathbf{d})$. Note that the ray starting at $A_e(\mathbf{d})$ (resp. $B_e(\mathbf{d})$) and containing $A'_e(\mathbf{d})$ (resp. $B'_e(\mathbf{d})$) does not intersect the interior of $\mathcal{P}$. We have

$$l_e(\mathbf{d}) = |a_eA_e(\mathbf{d})| + |b_eB_e(\mathbf{d})|,$$

where $|xy|$ denotes the Euclidean distance between the points x and y.

If $S_e(\mathbf{d})$ consists of only one segment, then only one of $A_e(\mathbf{d})$ and $B_e(\mathbf{d})$ (and similarly, only one of $A'_e(\mathbf{d})$ and $B'_e(\mathbf{d})$) is defined and, hence, $l_e(\mathbf{d})$ contains only one term. If $S_e(\mathbf{d}) = \emptyset$ or $S_e(\mathbf{d}) = e$, then the points $A_e(\mathbf{d})$, $B_e(\mathbf{d})$, $A'_e(\mathbf{d})$

and $B'_e(\mathbf{d})$ are undefined. In the former case, we have $l_e(\mathbf{d}) = 0$, whereas in the latter case, $l_e(\mathbf{d})$ is equal to the length of e.

We fix an edge e, and consider the behavior of the function $l_e(\mathbf{d})$ as the angle $\varphi_{\mathbf{d}}$ varies from 0 to 2π. Consider an angle $\varphi_{\mathbf{d}}$, and assume that $B_e(\mathbf{d})$ and $B'_e(\mathbf{d})$ exist. We will express the distance $|b_e B_e(\mathbf{d})|$ in terms of the angle $\varphi_{\mathbf{d}}$.

Let c_e be the orthogonal projection of vertex $B'_e(\mathbf{d})$ on the line through edge e. (See Figure 3.) First assume that c_e lies on e, and $B_e(\mathbf{d})$ lies between a_e and c_e. Also, assume that $0 < \varphi_{\mathbf{d}} < \alpha_e < \pi/2$. The angle between the vectors $B'_e(\mathbf{d})B_e(\mathbf{d})$ and $B'_e(\mathbf{d})c_e$ is equal to $\alpha_e - \varphi_{\mathbf{d}}$. Therefore,

$$|b_e B_e(\mathbf{d})| = |b_e c_e| + |c_e B_e(\mathbf{d})| = |b_e c_e| + |B'_e(\mathbf{d}) c_e| \tan(\alpha_e - \varphi_{\mathbf{d}}).$$

If the angle $\varphi_{\mathbf{d}}$ does not satisfy $0 < \varphi_{\mathbf{d}} < \alpha_e < \pi/2$, or if (i) c_e lies on e, and $B_e(\mathbf{d})$ lies between b_e and c_e, or (ii) c_e does not lie on e, then we get a similar expression for $|b_e B_e(\mathbf{d})|$. Moreover, if $A_e(\mathbf{d})$ and $A'_e(\mathbf{d})$ also exist, then we can write $|a_e A_e(\mathbf{d})|$ in a similar fashion.

It follows that we can write $l_e(\mathbf{d})$ in the form

$$l_e(\mathbf{d}) = X_e + Y_e \tan(\alpha_e - \varphi_{\mathbf{d}}),$$

where X_e and Y_e are (possibly negative) real numbers that only depend on the edge e and the points $A'_e(\mathbf{d})$ and/or $B'_e(\mathbf{d})$. Clearly, if edge e needs support, or e is completely attached to supports, for direction $\mathbf{d}$, we get the same expression; in these cases, X_e is equal to the length of e, and $Y_e = 0$.

If we vary the angle $\varphi_{\mathbf{d}}$ by a small amount, then, in general, the values of X_e and Y_e do not change; hence the above expression for $l_e(\mathbf{d})$ remains the same. For some angles $\varphi_{\mathbf{d}}$, however, the values of X_e and Y_e will change. Therefore, we want to partition the interval $[0, 2\pi]$ of directions into subintervals, such that within each subinterval I, the function $l_e(\mathbf{d})$ can be written as

$$l_e(\mathbf{d}) = X_e^I + Y_e^I \tan(\alpha_e - \varphi_{\mathbf{d}}),$$

where X_e^I and Y_e^I are constant within I.

How do we find this partition? In order to answer this question, we define for each vertex v of $\mathcal{P}$, its *visibility cone*, *cone*(v), as the cone with apex v and maximum angular range in which v can see to infinity. (See [2].)

The following lemma identifies certain directions $\mathbf{d}_0$ where the combinatorial structure of the supports associated with an edge changes.

Lemma 3 *Let e be an edge of $\mathcal{P}$, $\mathbf{d}_0$ a direction, φ the angle corresponding to $\mathbf{d}_0$, and ϵ a positive real number.*

(1) Assume that for directions $\mathbf{d}$ such that $\varphi - \epsilon < \varphi_{\mathbf{d}} < \varphi$, the vertex $A'_e(\mathbf{d})$ is equal to A', and for directions $\mathbf{d}$ such that $\varphi < \varphi_{\mathbf{d}} < \varphi + \epsilon$, it is equal to A'', where $A' \neq A''$. Then, the line segment $A'A''$ is on one of the bounding rays of cone(A') and is in the direction $\mathbf{d}_0$.

(2) Assume that for directions $\mathbf{d}$ such that $\varphi - \epsilon < \varphi_{\mathbf{d}} < \varphi$, the vertex $A'_e(\mathbf{d})$ exists, and for directions $\mathbf{d}$ such that $\varphi < \varphi_{\mathbf{d}} < \varphi + \epsilon$, it does not exist, or vice versa. Then

  (a) the line segment $B'_e(\mathbf{d}_0)A'_e(\mathbf{d}_0)$ is on one of the bounding rays of cone$(B'_e(\mathbf{d}_0))$ and is in the direction $\mathbf{d}_0$, or

  (b) the line segment $A'_e(\mathbf{d}_0)B'_e(\mathbf{d}_0)$ is on one of the bounding rays of cone$(A'_e(\mathbf{d}_0))$ and is in the direction $\mathbf{d}_0$, or

  (c) the line segment $b_eA'_e(\mathbf{d}_0)$ is on one of the bounding rays of cone(b_e) and is in the direction $\mathbf{d}_0$, or

  (d) the line segment $a_eA'_e(\mathbf{d}_0)$ is on one of the bounding rays of cone(a_e) and is in the direction $\mathbf{d}_0$, or

  (e) the direction $\mathbf{d}_0$ is parallel to e.

A similar claim holds for the vertex $B'_e(\mathbf{d})$.

Proof. To prove 1, first observe that for all directions $\mathbf{d}$ such that $\varphi - \epsilon < \varphi_{\mathbf{d}} < \varphi + \epsilon$, the set $S_e(\mathbf{d})$ is not equal to the entire edge e, since $A'_e(\mathbf{d})$ exists. If we increase the direction angle $\varphi_{\mathbf{d}}$ from $\varphi - \epsilon$ to φ, then the ray starting at $A_e(\mathbf{d})$ and going into the direction $\mathbf{d}$ rotates around vertex A'. Moreover, the part of this ray that is beyond A' does not intersect the polygon $\mathcal{P}$. At direction $\mathbf{d}_0$, the ray contains both A' and A''. If we increase $\varphi_{\mathbf{d}}$ further from φ to $\varphi + \epsilon$, then the ray rotates around A'', and the part beyond A'' does not intersect $\mathcal{P}$. It follows that at direction $\mathbf{d}_0$, the vertex A'' must be beyond A' on this ray. Hence, the line segment $A'A''$ is on one of the bounding rays of *cone*(A').

The proof of 2 is similar. In Case (a), while increasing $\varphi_{\mathbf{d}}$ from $\varphi - \epsilon$ to φ, the two endpoints $A_e(\mathbf{d})$ and $B_e(\mathbf{d})$ of the two segments defining the set $S_e(\mathbf{d})$ get closer together, and meet if $\varphi_{\mathbf{d}} = \varphi$. Note that $A'_e(\mathbf{d}_0)$ is beyond $B'_e(\mathbf{d}_0)$ on the ray from $A_e(\mathbf{d}_0)$ in direction $\mathbf{d}_0$. Case (b) is similar to Case (a). In Case (c), for $\varphi_{\mathbf{d}} \in (\varphi - \epsilon, \varphi)$, the set $S_e(\mathbf{d})$ only consists of the segment $a_eA_e(\mathbf{d})$. While increasing $\varphi_{\mathbf{d}}$ to φ, the point $A_e(\mathbf{d})$ moves to the vertex b_e. If $\varphi_{\mathbf{d}} = \varphi$, $A_e(\mathbf{d})$ and b_e are equal. Case (d) is the case where vertex $A'_e(\mathbf{d})$ changes from undefined to defined. Finally, in Case (e), the edge e starts or stops needing support at direction $\mathbf{d}_0$. In this case, the vertex $A'_e(\mathbf{d}_0)$ may change its status from undefined to defined, or vice versa. ∎

Let $\mathbf{D}_a$ be the set of directions $\mathbf{d}$ determined by the bounding rays of the non-empty visibility cones. We also define a set $\mathbf{D}_b$ containing the following directions. For each edge e of $\mathcal{P}$, $\mathbf{D}_b$ contains the two directions that are parallel to e. Let $\mathbf{D} := \mathbf{D}_a \cup \mathbf{D}_b$. Then the set $\mathbf{D}$ contains at most $4n$ directions. The following lemma follows from the discussion above.

Lemma 4 *The directions of the set* $\mathbf{D}$ *partition the interval* $[0, 2\pi]$ *into* $O(n)$ *subintervals, such that within each subinterval* I*, the function* $L_{\mathcal{P}}(\mathbf{d})$ *is the sum of* n *functions* $l_e(\mathbf{d})$*, each having the form*

$$l_e(\mathbf{d}) = X_e^I + Y_e^I \tan(\alpha_e - \varphi_{\mathbf{d}}), \tag{1}$$

where X_e^I *and* Y_e^I *are constant within* I.

We call *critical* the directions $\mathbf{d}$ at which expression (1) changes for some edge e (i.e., X_e^I or Y_e^I changes). Lemma 4 gives a set of $O(n)$ directions that includes all critical directions.

2.1 The algorithm

Now we are ready to give an outline of our algorithm for computing the direction $\mathbf{d}$ for which $L_{\mathcal{P}}(\mathbf{d}) = \sum_e l_e(\mathbf{d})$ is minimal. After this outline, we will give the details for each step.

Step 1: Compute the set $\mathbf{D}$ defined above, and sort its elements in counter-clockwise order. Preprocess $\mathcal{P}$ into a data structure, such that ray shooting queries can be answered.

At this point, we have a partition of $[0, 2\pi]$ into $O(n)$ subintervals. Within each subinterval I, we know that $L_{\mathcal{P}}(\mathbf{d})$ has the form

$$L_{\mathcal{P}}(\mathbf{d}) = \sum_e l_e(\mathbf{d}) = \sum_e (X_e^I + Y_e^I \tan(\alpha_e - \varphi_{\mathbf{d}})), \tag{2}$$

where X_e^I and Y_e^I are constant within I.

Step 2: Obtain expression (2) for the function $L_{\mathcal{P}}(\mathbf{d})$ in the first subinterval I, and compute the minimum of $L_{\mathcal{P}}(\mathbf{d})$ within I.

Step 3: Sweep over the elements of $\mathbf{D}$, thereby visiting the subintervals one by one. At each direction $\mathbf{d} \in \mathbf{D}$, obtain the new expression for $L_{\mathcal{P}}(\mathbf{d})$, by subtracting the functions $l_e(\mathbf{d})$ whose expressions change at this direction, and adding the corresponding new functions $l_e(\mathbf{d})$. Then compute the minimum of $L_{\mathcal{P}}(\mathbf{d})$ within the new subinterval.

Let us consider Step 1 first. Using the algorithm of [8,14], we compute in $O(n)$ time all vertices v for which $cone(v) \neq \emptyset$, together with their visibility cones. Then, we obtain the elements of $\mathbf{D}$ in sorted order, in $O(n \log n)$ time. We take the ray shooting data structure of [9]. There it is shown how to preprocess $\mathcal{P}$ in $O(n)$ time, such that ray shooting queries can be answered in $O(\log n)$ time. Hence, Step 1 can be completed in $O(n \log n)$ time.

Next, consider Step 2. For edges e that need support for directions in I, i.e., $\mathbf{n}_e \cdot \mathbf{d} < 0$ for all $\mathbf{d} \in I$, expression (1) is trivial.

In order to find expression (1) for edges e such that $\mathbf{n}_e \cdot \mathbf{d} \geq 0$ for all $\mathbf{d} \in I$, we have to determine the set $S_e(\mathbf{d})$. In particular, if $S_e(\mathbf{d}) \neq \emptyset$ and $S_e(\mathbf{d}) \neq e$, we have to find the vertices $A'_e(\mathbf{d})$ and/or $B'_e(\mathbf{d})$. We do the following.

Choose a direction $\mathbf{d}$ in I. For each vertex v of $\mathcal{P}$, perform a ray shooting query from v in direction $\mathbf{d}$. If this ray does not intersect $\mathcal{P}$, then we also perform a ray shooting query from v in direction $-\mathbf{d}$, provided this ray does not immediately go inside $\mathcal{P}$. If e is the edge that is hit first by this second ray and the ray intersects the interior of e, then $\mathbf{n}_e \cdot \mathbf{d} \geq 0$, $S_e(\mathbf{d}) \neq \emptyset$ and $S_e(\mathbf{d}) \neq e$. Moreover, v is equal to $A'_e(\mathbf{d})$ or $B'_e(\mathbf{d})$, and it is easy to decide whether $v = A'_e(\mathbf{d})$ or $v = B'_e(\mathbf{d})$: Take a point just to the right of the intersection point of the second ray with e, and shoot a third ray in direction $\mathbf{d}$. If this ray misses $\mathcal{P}$, then $v = A'_e(\mathbf{d})$; else $v = B'_e(\mathbf{d})$.

After these ray shooting queries, we have found all edges e such that $\mathbf{n}_e \cdot \mathbf{d} \geq 0$, $S_e(\mathbf{d}) \neq \emptyset$ and $S_e(\mathbf{d}) \neq e$, together with the corresponding vertices $A'_e(\mathbf{d})$ and/or $B'_e(\mathbf{d})$. For all remaining edges e for which $\mathbf{n}_e \cdot \mathbf{d} \geq 0$, we know that either $S_e(\mathbf{d}) = \emptyset$ or $S_e(\mathbf{d}) = e$. Therefore, for each such edge e, we take an arbitrary point x on e, and perform a ray shooting query from x in direction $\mathbf{d}$. If this ray intersects $\mathcal{P}$, then $S_e(\mathbf{d}) = e$; otherwise $S_e(\mathbf{d}) = \emptyset$.

It follows that expression (2) for $L_{\mathcal{P}}(\mathbf{d})$ in the first subinterval I can be computed in $O(n \log n)$ time. The problem that remains is that of computing the minimum of $L_{\mathcal{P}}(\mathbf{d})$ in I. We will consider this in Section 2.2.

We are left with Step 3. Assume, we move from subinterval I to I'. Let $\mathbf{d}$ be the critical direction corresponding to the right endpoint of I (which is the left endpoint of I').

First assume that $\mathbf{d} \in \mathbf{D}_b$, i.e., it is parallel to, say, edge e. Note that the sign of $\mathbf{n}_e \cdot \mathbf{d}'$ is the same for all $\mathbf{d}'$ in the interior of I'. There are two possible cases.

First, if $\mathbf{n}_e \cdot \mathbf{d}' < 0$ for all $\mathbf{d}'$ in the interior of I', then edge e needs support in the subinterval I'. Therefore, we subtract the (old) term $l_e(\mathbf{d})$ from $L_{\mathcal{P}}(\mathbf{d})$ and add the new value $l_e(\mathbf{d})$, which is equal to the length of e.

Otherwise, $\mathbf{n}_e \cdot \mathbf{d}' > 0$ for all $\mathbf{d}'$ in the interior of I'. In this case, we know that e needs support in the subinterval I, and the (old) term $l_e(\mathbf{d})$ is equal to the length of e. The new expression for $l_e(\mathbf{d})$ is obtained as follows. Perform a ray shooting query from some point on e in direction $\mathbf{d}$. (Note that this ray is "along" e.) If this ray does not intersect the interior of $\mathcal{P}$, then no point on e is attached to a support in the subinterval I', and we subtract $l_e(\mathbf{d})$ from $L_{\mathcal{P}}(\mathbf{d})$. Otherwise, the ray hits the interior of some edge of $\mathcal{P}$, and in the subinterval I', edge e is completely attached to supports. Hence, in I', the term $l_e(\mathbf{d})$ remains equal to the length of e; we do not have to update $L_{\mathcal{P}}(\mathbf{d})$.

It remains to consider the case when $\mathbf{d}$ is a "cone direction", i.e., it coincides with the direction of a bounding ray r of, say, $cone(v)$. In order to describe what has to be done now, assume w.l.o.g. that $\mathbf{d}$ goes vertically upwards. Let $w \neq v$ be the vertex that is on r. Note that both edges e_w and e'_w that are incident to w are on one side of the line containing r. Assume that they are to the left of this line, and that e_w is above e'_w. (The case when e_w and e'_w are to the right of this line can be handled similarly.) Note that by going down (resp. up) from any interior point on e_w (resp. e'_w), we go into the interior of $\mathcal{P}$. (Assume that by going up from any interior point on e_w, we go into the interior of $\mathcal{P}$. Then by walking along the boundary of $\mathcal{P}$, starting at w and following edge e_w first, we must intersect the ray r. This cannot happen, because r does not intersect the interior of $\mathcal{P}$.)

Let e_v and e'_v be the edges that are incident to v. There are three possible cases. First assume that e_v and e'_v are both to the right of the line containing r. Also, assume w.l.o.g. that e_v is above e'_v. (See Figure 4(a).) Then $cone(v)$ is also to the right of this line, and by going down (resp. up) from any interior point on e_v (resp. e'_v), we go into the interior of $\mathcal{P}$. Perform a ray shooting query from v in direction $-\mathbf{d}$, and let f be the edge that is hit first. Then, in the subinterval I, w is the vertex $A'_f(\mathbf{d})$, whereas $v = B'_f(\mathbf{d})$. In I', the edge f is completely attached to supports. (Recall that I' follows I in counterclockwise order.) This is because e_w and e'_w (resp. e_v and e'_v) are to the left (resp. right) of the line containing r. Therefore, we subtract the term $l_f(\mathbf{d})$ from $L_{\mathcal{P}}(\mathbf{d})$ and add the new term $l_f(\mathbf{d})$, which is equal to the length of f. Also, in I, the set $S_{e_v}(\mathbf{d})$ consists of at most one segment: the vertex $B'_{e_v}(\mathbf{d})$ may exist, but $A'_{e_v}(\mathbf{d})$ does not exist. In I', the vertex $B'_{e_v}(\mathbf{d})$ is still the same or still does not exist, but we have $w = A'_{e_v}(\mathbf{d})$. Therefore, we update the term $l_{e_v}(\mathbf{d})$.

Next, assume that e_v and e'_v are both to the left of the line containing r. (See Figure 4(b).) Again, assume w.l.o.g. that e_v is above e'_v. By going down (resp. up) from any interior point on e_v (resp. e'_v), we go into the interior of $\mathcal{P}$.

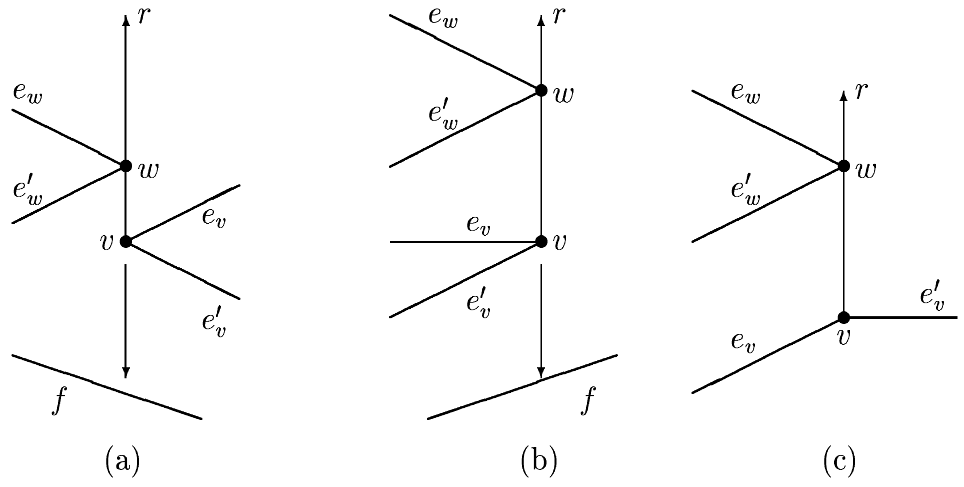

Fig. 4. *The three possible cases for a "cone direction"* $\mathbf{d}$. *We assume that* $\mathbf{d}$ *goes vertically upwards.*

Perform a ray shooting query from v in direction $-\mathbf{d}$, and let f be the edge that is hit first. (Assume for the moment that f exists.) Then, in I, we have $v = A'_f(\mathbf{d})$, whereas in I', we have $w = A'_f(\mathbf{d})$. Also, the vertex $B'_f(\mathbf{d})$ is the same in I and I', or is undefined in both these intervals. Therefore, we update the term $l_f(\mathbf{d})$. In I, we have $w = A'_{e_v}(\mathbf{d})$, whereas $B'_{e_v}(\mathbf{d})$ does not exist. In I', edge e_v is completely attached to supports. Therefore, we replace the term $l_{e_v}(\mathbf{d})$ by a new term, which is equal to the length of e_v. If edge f does not exist, then we only have to update the term $l_{e_v}(\mathbf{d})$, in the way just described.

Finally, assume that e_v and e'_v are on different sides of the line containing r. (See Figure 4(c).) Assume w.l.o.g. that e_v is to the left of this line. By going down from any interior point on e_v or e'_v, we go into the interior of $\mathcal{P}$. In subinterval I, the vertex w is equal to $A'_{e_v}(\mathbf{d})$, whereas $B'_{e_v}(\mathbf{d})$ is undefined in this interval. In I', edge e_v is completely attached to supports. So, we replace the term $l_{e_v}(\mathbf{d})$ by a new term, which is equal to the length of e_v. Also, in I, the vertex $A'_{e'_v}(\mathbf{d})$ is undefined, whereas in I', it is equal to w. In I and I', the vertex $B'_{e'_v}(\mathbf{d})$ may be defined or undefined. At direction $\mathbf{d}$, however, its "value" does not change. Therefore, we update the term $l_{e'_v}(\mathbf{d})$.

This concludes the description of Step 3. For each critical direction, we need $O(\log n)$ time to update the expression for $L_{\mathcal{P}}(\mathbf{d})$. As in Step 2, the problem is in computing the minimum of $L_{\mathcal{P}}(\mathbf{d})$ in the new subinterval I'. This problem will be addressed in Section 2.2.

Remark 5 The polygon $\mathcal{P}$ may have edges that need support, or are completely attached to supports, for any direction $\mathbf{d}$. These edges are invisible, in the sense that no point on them is visible from the "outside". Consider such an invisible edge e. In Step 2, we find out that in the first subinterval, $l_e(\mathbf{d})$ is equal to the length of e. During the sweep in Step 3, the term $l_e(\mathbf{d})$ never changes, i.e., it always stays equal to the length of e.

2.2 Minimizing the function $L_{\mathcal{P}}(\mathbf{d})$

The problem that remains is to compute the minimum of the function

$$L_{\mathcal{P}}(\mathbf{d}) = \sum_e (X_e^I + Y_e^I \tan(\alpha_e - \varphi_{\mathbf{d}})),$$

in a subinterval I of $[0, 2\pi]$. Recall that X_e^I and Y_e^I are real numbers that are constant for $\varphi_{\mathbf{d}} \in I$.

We write this optimization problem in a simpler form. Note that the term $\sum_e X_e^I$ is independent of $\mathbf{d}$. Introducing new variables (a_i for Y_e^I, b_i for α_e, and φ for $\varphi_{\mathbf{d}}$), leads to the problem of minimizing the function

$$F(\varphi) := \sum_{i=1}^{n} a_i \tan(b_i - \varphi),$$

in a subinterval I of $[0, 2\pi]$. Here, the a_i's and b_i's are real numbers. Using the formula $\tan(y-x) = (\tan y - \tan x)/(1 + \tan y \tan x)$, and defining $c_i := \tan b_i$, we get

$$F(\varphi) = \sum_{i=1}^{n} a_i \frac{c_i - \tan\varphi}{1 + c_i \tan\varphi} = \sum_{i=1}^{n} \left(\frac{a_i(c_i + 1/c_i)}{1 + c_i \tan\varphi} - \frac{a_i}{c_i} \right).$$

Let $d_i := a_i(c_i + 1/c_i)$. Then minimizing F is equivalent to minimizing the function

$$\sum_{i=1}^{n} \frac{d_i}{1 + c_i \tan\varphi}.$$

Let $x := \tan\varphi$. Then we get the following problem.

Problem *$PR(n)$: Given $2n+2$ real numbers $c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n, A$ and B, compute the global minimum of the function*

$$G(x) := \sum_{i=1}^{n} \frac{d_i}{1 + c_i x}, \tag{3}$$

in the interval $[A, B]$.

We can solve this problem using standard techniques from calculus. Let us consider the derivative of G:

$$G'(x) = \sum_{i=1}^{n} \frac{-d_i c_i}{(1 + c_i x)^2} = \frac{-1}{\prod_{k=1}^{n}(1 + c_k x)^2} \sum_{i=1}^{n} c_i d_i \prod_{j \neq i}(1 + c_j x)^2.$$

Hence, $G'(x) = 0$ if and only if

$$\sum_{i=1}^{n} c_i d_i \prod_{j \neq i}(1 + c_j x)^2 = 0. \qquad (4)$$

The expression in (4) is a polynomial in x of degree $2(n-1)$. Hence, the original function $L_{\mathcal{P}}(\mathbf{d})$ we are interested in can have a linear number of local minima. Using techniques from numerical analysis, we compute (i.e., approximate to any desired precision) the roots of (4), and for each of them that is contained in the interval $[A, B]$, we evaluate G. We also evaluate G for $x = A$ and $x = B$. In this way, we find the global minimum of G in $[A, B]$.

This approach is not efficient. Unfortunately, we are not aware of any efficient algorithm that solves Problem $PR(n)$. We leave the design and analysis of such an algorithm as an open problem. In the theorem below, we denote the time it takes to solve $PR(n)$ generically by $p(n)$.

Theorem 6 *Given a simple polygon with* n *vertices, a direction minimizing the total contact-length of supports can be found in* $O(n \log n + n\,p(n))$ *time, where* $p(n)$ *is the time for solving problem* $PR(n)$.

Proof. Consider our algorithm. Step 1 takes $O(n \log n)$ time. In Step 2, it takes $O(n \log n)$ time, to write down expression (2) for the function $L_{\mathcal{P}}(\mathbf{d})$ in the first subinterval I. Given this expression, we transform it into (3) in linear time. Then in $p(n)$ time, we compute the minimum of this function. Hence, Step 2 takes total time $O(n \log n + p(n))$. In Step 3, we visit the $O(n)$ critical directions one after another. Going from one direction to the next one, we update the functions $L_{\mathcal{P}}(\mathbf{d})$ and $G(x)$ in $O(\log n)$ time. The minimum of the updated function $L_{\mathcal{P}}(\mathbf{d})$ is then computed in $p(n)$ time. ∎

As we will show now, the running time can be improved considerably, if the edges of our polygon have only a small number of orientations. A polygon is called *C-oriented* if its edges have at most C different orientations. Suppose that our simple polygon $\mathcal{P}$ is C-oriented. In this case, the function $G(x)$ in problem $PR(n)$ can be rewritten such that it contains only C terms: There are at most C different normal angles α_e, hence at most C different values for the c_i's. If we group these together, we get an expression of the form

$$G(x) = \sum_{i=1}^{C} \frac{d_i'}{1 + c_i x},$$

for real numbers $d_1', d_2', \ldots, d_C'$. Therefore, solving $G'(x) = 0$ leads to a polynomial of degree at most $2(C-1)$. Assuming that C is a constant, and that roots of polynomials of constant degree can be computed in constant time, we can compute the minimum of $G(x)$ in constant time. This proves:

Theorem 7 *Given a simple C-oriented polygon with n vertices, where C is a constant, a direction minimizing the total contact-length of supports can be found in $O(n \log n)$ time.*

3 Minimum-area supports for simple polygons

We now consider the problem of computing a direction that minimizes the total *area* of supports for a simple polygon. Let $\mathcal{P}$ be a simple polygon having n vertices. As before, if $\mathbf{d}$ is a direction, then $\varphi_{\mathbf{d}}$ denotes the angle between the positive x-axis and $\mathbf{d}$.

Our basic approach is the same as in Section 2. We now split the interval $[0, 2\pi]$ into four subintervals $[0, \pi/2]$, $[\pi/2, \pi]$, $[\pi, 3\pi/2]$, and $[3\pi/2, 2\pi]$, and solve the problem within each subinterval separately. Since these four subproblems are similar, we only consider the interval $[\pi/2, \pi]$. That is, we show how to compute a direction $\mathbf{d}$ such that $\pi/2 \leq \varphi_{\mathbf{d}} \leq \pi$, and the area of the supports is minimal when $\mathcal{P}$ is built in direction $\mathbf{d}$. For this, we partition the interval $[\pi/2, \pi]$ into $O(n)$ subintervals, such that within each subinterval, there is a closed form for the total area of the supports. Then we sweep along these subintervals. In [11], we computed the area expression at the boundary of each subinterval from scratch. In this section, we give an improved solution, which is obtained by exploiting the combinatorial structure of the supports more carefully.

Let $\mathbf{d}$ be any direction such that $\pi/2 \leq \varphi_{\mathbf{d}} \leq \pi$. Let $s_b(\mathbf{d})$ be the vertex of $\mathcal{P}$ that is extreme in direction $-\mathbf{d}$. Hence, the platform is the line through $s_b(\mathbf{d})$ orthogonal to $\mathbf{d}$, and $\mathcal{P}$ is "above" (w.r.t. direction $\mathbf{d}$) this line.

The supports for build direction $\mathbf{d}$ consist of pairwise disjoint polygons, which we call *support polygons.* We divide these polygons into two classes. A support polygon that does not have an edge on the platform is colored *red,* otherwise, it is colored *blue.* The area expression for the supports can now be determined by considering the red and blue polygons separately. In Section 3.1, we analyze

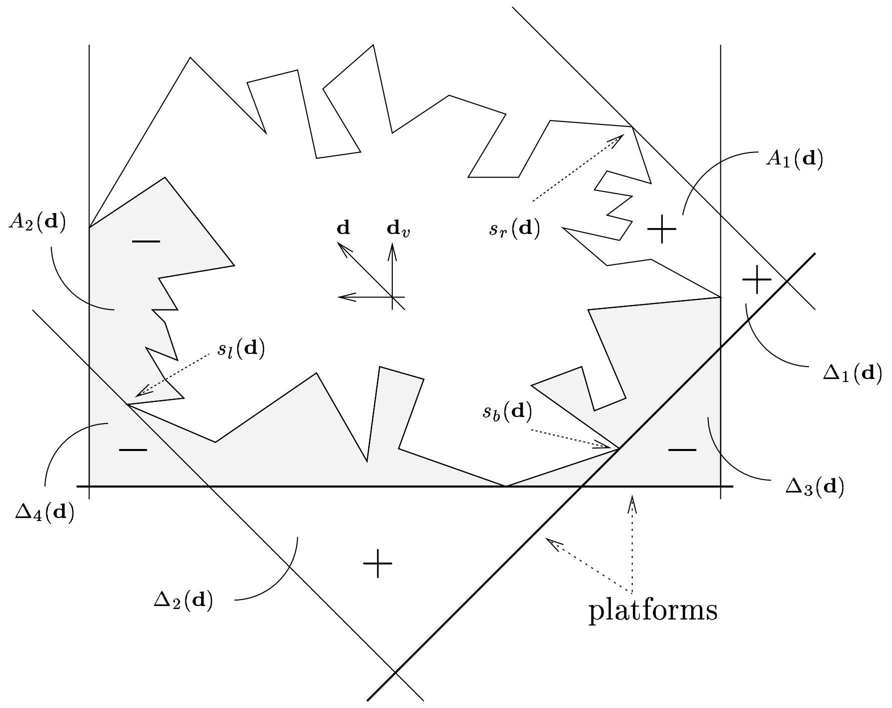

Fig. 5. *The blue support polygons for directions* $\mathbf{d}$ *and* $\mathbf{d}_v$.

the blue polygons; the red polygons are considered in Section 3.2. Then, in Section 3.3, we give the complete algorithm.

3.1 Blue support polygons

We will show that we can preprocess $\mathcal{P}$ in $O(n)$ time, such that for any direction $\mathbf{d}$ with $\pi/2 \leq \varphi_{\mathbf{d}} \leq \pi$, we can in $O(\log n)$ time, compute the expression—as a function of $\mathbf{d}$—for the area of the blue support polygons.

Let $\mathbf{d}_v$ be the vertical direction, i.e., $\varphi_{\mathbf{d}_v} = \pi/2$. Assume that we know the area A_v of the blue support polygons for this direction. Given a direction $\mathbf{d}$, our strategy will be to consider the "difference" of the blue polygons for $\mathbf{d}$ and $\mathbf{d}_v$.

For any direction $\mathbf{d}$, denote the total area of the blue polygons by $A(\mathbf{d})$. How can we use A_v to compute $A(\mathbf{d})$ efficiently? The blue support polygons for $\mathbf{d}_v$ and $\mathbf{d}$ determine (i) two simple polygons whose areas we denote by $A_1(\mathbf{d})$ and $A_2(\mathbf{d})$, and (ii) four triangles whose areas we denote by $\Delta_i(\mathbf{d})$, $1 \leq i \leq 4$, as indicated in Figure 5. From this figure we see that

$$A(\mathbf{d}) = A_v + A_1(\mathbf{d}) - A_2(\mathbf{d}) + \Delta_1(\mathbf{d}) + \Delta_2(\mathbf{d}) - \Delta_3(\mathbf{d}) - \Delta_4(\mathbf{d}). \tag{5}$$

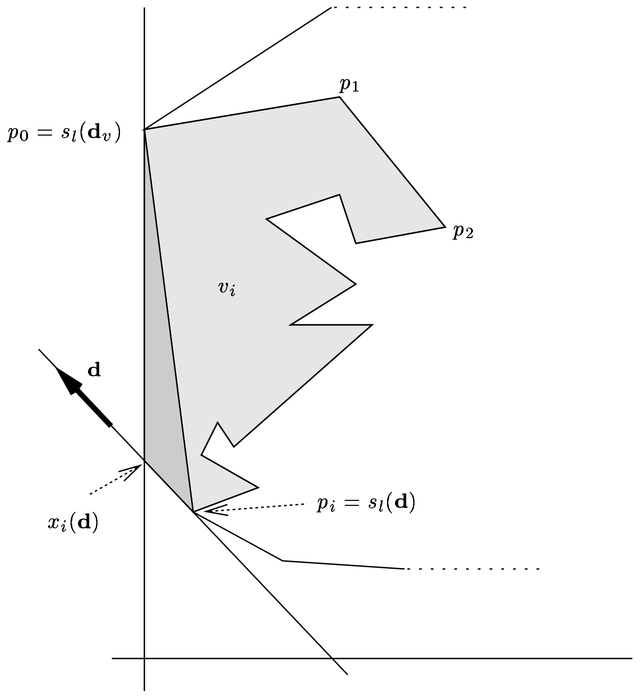

Fig. 6. *The blue area* $A_2(\mathbf{d})$.

Computing the areas $\Delta_i(\mathbf{d})$, $1 \leq i \leq 4$, as a function of $\mathbf{d}$ is easy; see Section 3.1.2. The problem is computing the areas $A_1(\mathbf{d})$ and $A_2(\mathbf{d})$. Note that the polygons determining $A_1(\mathbf{d})$ and $A_2(\mathbf{d})$ can have a linear number of vertices. We will show in Section 3.1.1 that after an appropriate preprocessing step, we can compute $A_1(\mathbf{d})$ and $A_2(\mathbf{d})$ as a function of $\mathbf{d}$, in $O(\log n)$ time.

3.1.1 Computing the expression $\boldsymbol{A_2}(\mathbf{d})$

Let $s_l(\mathbf{d}_v)$, $s_r(\mathbf{d}_v)$ and $s_b(\mathbf{d}_v)$ be the left, right and bottom extreme vertex for direction $\mathbf{d}_v$, respectively, and define $s_l(\mathbf{d})$, $s_r(\mathbf{d})$ and $s_b(\mathbf{d})$ analogously for direction $\mathbf{d}$.

We denote the vertices we encounter, when walking around $\mathcal{P}$ from $s_l(\mathbf{d}_v)$ to $s_b(\mathbf{d}_v)$ in counterclockwise order by $p_0 = s_l(\mathbf{d}_v), p_1, p_2, \ldots, p_k = s_b(\mathbf{d}_v)$. (Refer to Figure 6.) We will store with each vertex p_i, $0 \leq i \leq k$, a real number v_i. As we will see, these values can be used to determine the expression for the area $A_2(\mathbf{d})$.

We will use the following notation. If a, b and c are three points in the plane, then $area(a, b, c)$ denotes the *signed* area of the triangle with vertices a, b and c, where the sign is $+$ (resp. $-$) if c is to the left (resp. right) of the vector $\overline{ab}$.

The values of the variables v_i are defined incrementally, as follows. We define

$v_0 := 0$ and $v_1 := 0$. Let $2 \leq i \leq k$, and assume that the value of v_{i-1} has been defined already. Then

$$v_i := v_{i-1} + \mathit{area}(p_0, p_i, p_{i-1}).$$

We need the following lemma, whose proof can be found in [13, page 24].

Lemma 8 *Let Q be a simple polygon with vertices $q_0, q_1, \ldots, q_{n-1}$, labeled in clockwise order, let $q_n := q_0$, and let p be an arbitrary point in the plane. Then the area of Q is equal to*

$$\sum_{j=0}^{n-1} \mathit{area}(p, q_{j+1}, q_j).$$

Corollary 9 *Let $\mathbf{d}$ be a build direction such that $\pi/2 \leq \varphi_{\mathbf{d}} \leq \pi$. Let L be the vertical line through $p_0 = s_l(\mathbf{d}_v)$, and let $x_i(\mathbf{d})$, $1 \leq i \leq k$, be the intersection of L with the ray emanating from p_i and having direction $\mathbf{d}$. Assume that the line segment $p_i x_i(\mathbf{d})$ does not intersect the interior of $\mathcal{P}$. Then $(p_0, p_1, p_2, \ldots, p_i, x_i(\mathbf{d}))$ is a simple polygon, and its area is equal to*

$$v_i + \mathit{area}(p_0, x_i(\mathbf{d}), p_i). \tag{6}$$

Proof. It is clear that the polygon $Q = (p_0, p_1, \ldots, p_i, x_i(\mathbf{d}))$ is simple. Moreover, its vertices are in clockwise order. Applying Lemma 8 with $p = p_0$ shows that the area of Q is equal to

$$\sum_{j=0}^{i-1} \mathit{area}(p_0, p_{j+1}, p_j) + \mathit{area}(p_0, x_i(\mathbf{d}), p_i) + \mathit{area}(p_0, p_0, x_i(\mathbf{d})).$$

It is easy to see that the summation equals v_i, and $\mathit{area}(p_0, p_0, x_i(\mathbf{d})) = 0$. ■

Let $\mathbf{d}$ be a direction such that $\pi/2 \leq \varphi_{\mathbf{d}} \leq \pi$, and let i, $0 \leq i \leq k$, be the index such that $p_i = s_l(\mathbf{d})$. Then it is clear that the line segment $(p_i, x_i(\mathbf{d}))$ does not intersect the interior of $\mathcal{P}$. Hence, by Corollary 9, we have

$$A_2(\mathbf{d}) = v_i + \mathit{area}(p_0, x_i(\mathbf{d}), p_i).$$

Let a be the intersection between the vertical line through p_0 and the horizontal line through p_i. Then

$$\mathit{area}(p_0, x_i(\mathbf{d}), p_i) = \mathit{area}(p_0, a, p_i) - \mathit{area}(a, p_i, x_i(\mathbf{d}))$$

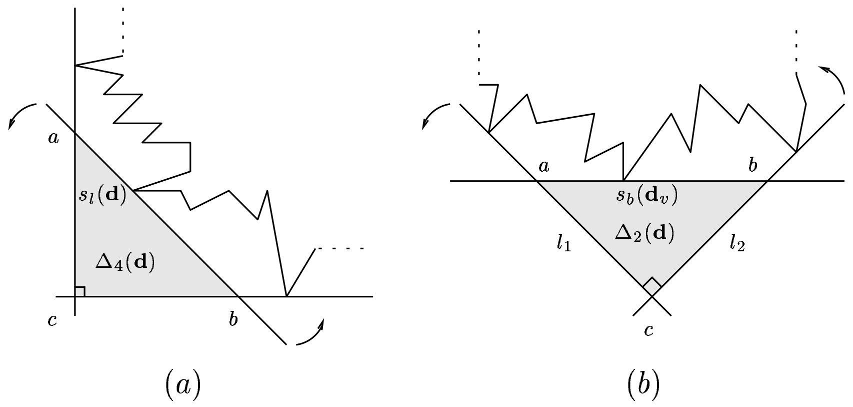

Fig. 7. *Triangle $\Delta_4(\mathbf{d})$ has one fixed point c; triangle $\Delta_2(\mathbf{d})$ has no fixed points.*

$$\begin{aligned}
&= area(p_0, a, p_i) - \tfrac{1}{2}\,|ap_i| \cdot |ax_i(\mathbf{d})| \\
&= area(p_0, a, p_i) - \tfrac{1}{2}\,|ap_i|^2 \tan(\pi - \varphi_{\mathbf{d}}) \\
&= area(p_0, a, p_i) + \tfrac{1}{2}\,|ap_i|^2 \tan\varphi_{\mathbf{d}}.
\end{aligned}$$

Hence, in a subinterval of $[\pi/2, \pi]$ in which the extreme vertex $s_l(\mathbf{d})$ does not change, we can write

$$A_2(\mathbf{d}) = A + B \cdot \tan\varphi_{\mathbf{d}}, \qquad (7)$$

where A and B are real numbers independent of $\mathbf{d}$ that change only when $s_l(\mathbf{d})$ changes.

Similarly, in a subinterval of $[\pi/2, \pi]$ in which the extreme vertex $s_r(\mathbf{d})$ does not change, we can write

$$A_1(\mathbf{d}) = A' + B' \cdot \tan\varphi_{\mathbf{d}}, \qquad (8)$$

where A' and B' are real numbers independent of $\mathbf{d}$ that change only when $s_r(\mathbf{d})$ changes.

3.1.2 Computing the expressions $\boldsymbol{\Delta_i}(\mathbf{d})$

Let $\mathbf{d}$ be a direction such that $\pi/2 \leq \varphi \leq \pi$. If we look at the four triangles $\Delta_i(\mathbf{d})$, $1 \leq i \leq 4$, in Figure 5, then we see that these are of two different types. (Refer to Figure 7.) The triangle $\Delta_4(\mathbf{d})$ has one fixed vertex c. If $\varphi_{\mathbf{d}}$ increases, then the line through the other two vertices a and b rotates around the extreme vertex $s_l(\mathbf{d})$. A similar remark holds for $\Delta_3(\mathbf{d})$. The triangle $\Delta_2(\mathbf{d})$ does not have a fixed point. If $\varphi_{\mathbf{d}}$ increases, then the line l_1 (resp. l_2) through a and c (resp. b and c) rotates around $s_l(\mathbf{d})$ (resp. $s_b(\mathbf{d})$). A similar remark holds for $\Delta_1(\mathbf{d})$.

Consider the triangle $\Delta_4(\mathbf{d})$ in Figure 7(a). Write the coordinates of $s_l(\mathbf{d})$ (which is the center of rotation of the line through a and b) as $s_l(\mathbf{d}) = (y_1, y_2)$, and write the coordinates of c as $c = (c_1, c_2)$. Then, the vertices a and b are equal to

$$a = (c_1, y_2 + (c_1 - y_1) \cdot \tan \varphi_{\mathbf{d}}),$$

and

$$b = (y_1 + (c_2 - y_2) \cdot \cot \varphi_{\mathbf{d}}, c_2).$$

Hence, for $\Delta_4(\mathbf{d})$, we get the following expression.

$$\begin{aligned}\Delta_4(\mathbf{d}) &= \tfrac{1}{2}\,|ac| \cdot |cb| \\ &= \tfrac{1}{2}\,[y_2 + (c_1 - y_1) \cdot \tan \varphi_{\mathbf{d}} - c_2] \cdot [y_1 + (c_2 - y_2) \cdot \cot \varphi_{\mathbf{d}} - c_1] \\ &= (y_1 - c_1)(y_2 - c_2) - \tfrac{1}{2}\,(y_1 - c_1)^2 \cdot \tan \varphi_{\mathbf{d}} - \tfrac{1}{2}\,(y_2 - c_2)^2 \cdot \cot \varphi_{\mathbf{d}}.\end{aligned}$$

The formula for $\Delta_3(\mathbf{d})$ is similar. Thus we can write

$$\Delta_3(\mathbf{d}) + \Delta_4(\mathbf{d}) = A + B \cdot \tan \varphi_{\mathbf{d}} + C \cdot \cot \varphi_{\mathbf{d}}, \tag{9}$$

where A, B and C are real numbers independent of $\mathbf{d}$, that change only when $s_l(\mathbf{d})$, $s_r(\mathbf{d})$ or $s_b(\mathbf{d})$ changes.

The formulas for $\Delta_1(\mathbf{d})$ and $\Delta_2(\mathbf{d})$ are more complex, because all three vertices move if $\varphi_{\mathbf{d}}$ increases. Referring to Figure 7(b), we have

$$\Delta_2(\mathbf{d}) = \tfrac{1}{2}\,|ac| \cdot |cb|. \tag{10}$$

The expressions for $\Delta_2(\mathbf{d})$ and $\Delta_1(\mathbf{d})$ follow by straightforward calculations. The result is the following formula, whose verification is left to the reader.

$$\begin{aligned}\Delta_1(\mathbf{d}) + \Delta_2(\mathbf{d}) = {} & B' \cdot \cos \varphi_{\mathbf{d}} \cdot \sin \varphi_{\mathbf{d}} + C' \cdot \cos^2 \varphi_{\mathbf{d}} + D' \cdot \sin^2 \varphi_{\mathbf{d}} \\ & + E' \cdot \frac{\sin^3 \varphi_{\mathbf{d}}}{\cos \varphi_{\mathbf{d}}} + F' \cdot \frac{\cos^3 \varphi_{\mathbf{d}}}{\sin \varphi_{\mathbf{d}}},\end{aligned} \tag{11}$$

where $B', \ldots, F'$ are real numbers independent of $\mathbf{d}$, that change only when $s_l(\mathbf{d})$, $s_r(\mathbf{d})$ or $s_b(\mathbf{d})$ changes.

If we substitute (7), (8), (9) and (11) into (5), use the relation $\cos^2 \varphi_{\mathbf{d}} = 1/(1 + \tan^2 \varphi_{\mathbf{d}})$, and express everything in terms of $\tan \varphi_{\mathbf{d}}$, then we get

$$
\begin{aligned}
A(\mathbf{d}) = {} & A'' + B'' \cdot \tan\varphi_{\mathbf{d}} + C'' \cdot \frac{1}{\tan\varphi_{\mathbf{d}}} \\
& + \frac{1}{1+\tan^2\varphi_{\mathbf{d}}} \cdot \left[\frac{D''}{\tan\varphi_{\mathbf{d}}} + E'' + F'' \cdot \tan\varphi_{\mathbf{d}} \right],
\end{aligned} \tag{12}
$$

where $A'', B'', \ldots, F''$ are real numbers independent of $\mathbf{d}$, that change only when $s_l(\mathbf{d})$, $s_r(\mathbf{d})$ or $s_b(\mathbf{d})$ changes.

3.1.3 Putting everything together

First, let us see what the critical directions for the blue area $A(\mathbf{d})$ are. It follows from the discussion above that expression (12) changes, if one of the vertices $s_l(\mathbf{d})$, $s_r(\mathbf{d})$ or $s_b(\mathbf{d})$ changes. While increasing $\varphi_{\mathbf{d}}$ from $\pi/2$ to π, the vertex $s_l(\mathbf{d})$ (resp. $s_r(\mathbf{d})$) changes, if the left (resp. right) tangent to $\mathcal{P}$ in direction $\mathbf{d}$ touches two vertices. This happens, if $\mathbf{d}$ is parallel to an edge of the convex hull of $\mathcal{P}$. The vertex $s_b(\mathbf{d})$ changes if $\mathbf{d}$ has the same direction as the inner normal of a convex hull edge.

Next, we show how for a given direction $\mathbf{d}$, $\pi/2 \leq \varphi_{\mathbf{d}} \leq \pi$, expression (12) can be computed. In preprocessing, we compute (i) the extreme vertices $s_l(\mathbf{d}_v)$, $s_r(\mathbf{d}_v)$ and $s_b(\mathbf{d}_v)$ for the vertical direction, (ii) the blue area A_v for this direction, (iii) the values v_i, $0 \leq i \leq k$, that are needed to compute the area $A_2(\mathbf{d})$, and (iv) similar values that are needed to compute the area $A_1(\mathbf{d})$. All this can be done in $O(n)$ time. Then, we compute the convex hull of $\mathcal{P}$, and its hierarchical representation, see [6, page 194]. Since $\mathcal{P}$ is a simple polygon, this can also be done in $O(n)$ time.

Given a direction $\mathbf{d}$ with $\pi/2 \leq \varphi_{\mathbf{d}} \leq \pi$, we use the hierarchical representation to compute the extreme vertices $s_l(\mathbf{d})$, $s_r(\mathbf{d})$ and $s_b(\mathbf{d})$, in $O(\log n)$ time. Given these vertices, we can compute expression (12) for $A(\mathbf{d})$ in $O(1)$ time.

We summarize the result of Section 3.1 in the following lemma.

Lemma 10 *We can preprocess $\mathcal{P}$ in $O(n)$ time, such that for a given direction $\mathbf{d}$, $\pi/2 \leq \varphi_{\mathbf{d}} \leq \pi$, we can in $O(\log n)$ time compute expression (12) for the blue area $A(\mathbf{d})$.*

The critical directions for the blue area are (i) the inner normals of the edges of the convex hull of $\mathcal{P}$, and (ii) for each convex hull edge, the two directions that are parallel to it.

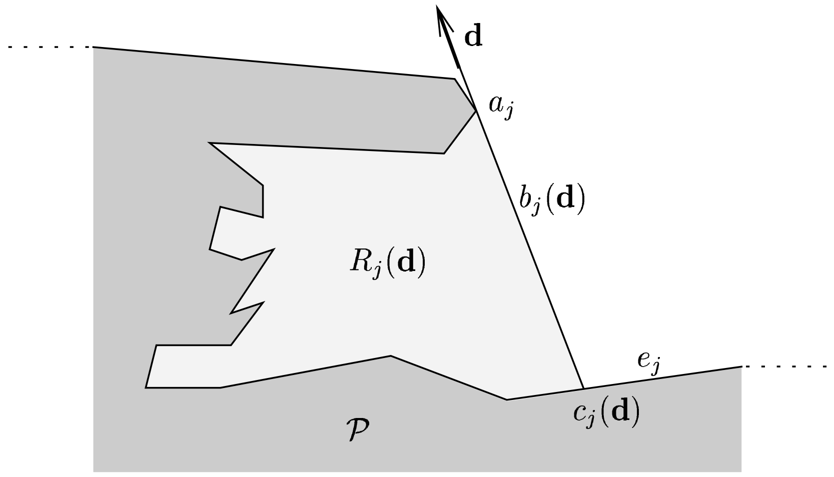

Fig. 8. *A red support polygon* $R_j(\mathbf{d})$.

3.2 Red support polygons

In this section, we consider the area of the red support polygons. Let $\mathbf{d}$ be a build direction such that $\pi/2 \leq \varphi_{\mathbf{d}} \leq \pi$. We denote the red polygons by $R_1(\mathbf{d}), \ldots, R_l(\mathbf{d})$. For simplicity, we use the same notation for their areas.

Each red polygon $R_j(\mathbf{d})$ consists of vertices and edges of $\mathcal{P}$, except for one edge, which we denote by $b_j(\mathbf{d})$. This edge lies on a line having direction $\mathbf{d}$. The endpoint of $b_j(\mathbf{d})$ that is extreme in direction $\mathbf{d}$, is a vertex of $\mathcal{P}$, which we denote by a_j. (Refer to Figure 8.) If $\varphi_{\mathbf{d}}$ increases, then edge $b_j(\mathbf{d})$ rotates around this vertex. Therefore we call this vertex a *center of rotation.* Let e_j be the edge of $\mathcal{P}$ that contains the other endpoint, $c_j(\mathbf{d})$, of $b_j(\mathbf{d})$. If $\varphi_{\mathbf{d}}$ increases, then $c_j(\mathbf{d})$ moves along e_j.

Suppose $I = [\alpha, \beta]$ is an interval, such that for each $\varphi_{\mathbf{d}} \in I$, (i) $b_j(\mathbf{d})$ has vertex a_j as one of its endpoints, and (ii) edge e_j does not change. Let $\mathbf{d}_0$ be the direction such that $\varphi_{\mathbf{d}_0} = \alpha$, and let $A_j := R_j(\mathbf{d}_0)$, and $c_j := c_j(\mathbf{d}_0)$. (Refer to Figure 9.) Then the area $R_j(\mathbf{d})$, for $\varphi_{\mathbf{d}} \in I$, is equal to

$$R_j(\mathbf{d}) = A_j + \Delta_j(\mathbf{d}),$$

where $\Delta_j(\mathbf{d})$ is the signed area of the triangle with vertices a_j, c_j, and $c_j(\mathbf{d})$.

In order to find an expression for $\Delta_j(\mathbf{d})$, let a' be the orthogonal projection of vertex a_j on the line through edge e_j. Let α_j be the angle between the positive x-axis and the outer normal of e_j. Assume that $0 < \varphi_{\mathbf{d}} < \alpha_j < \pi/2$. (The other cases can be handled similarly.) Note that the x-coordinate of $c_j(\mathbf{d})$ is between those of c_j and a'. The angle between the vectors $a_jc_j(\mathbf{d})$ and a_ja' is

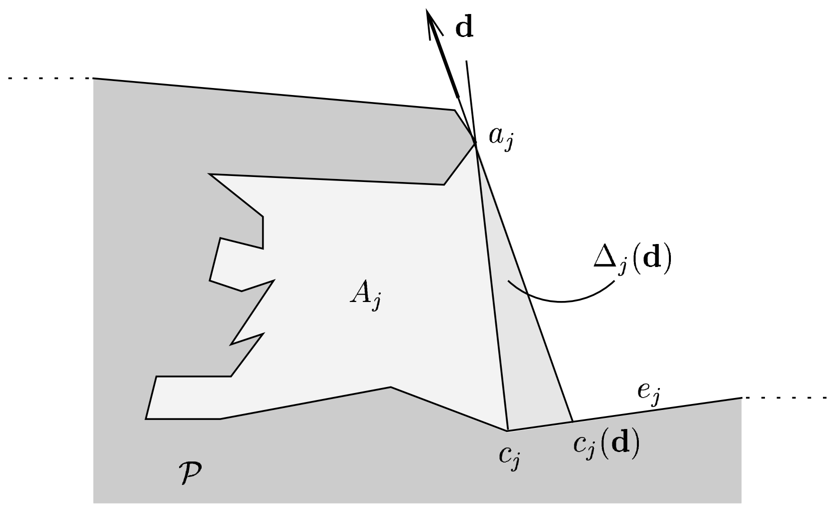

Fig. 9. *A red support polygon $R_j(\mathbf{d})$ consists of a constant part with area A_j and a triangle with area $\Delta_j(\mathbf{d})$.*

---

equal to $\alpha_j - \varphi_{\mathbf{d}}$. Therefore,

$$\begin{aligned}\Delta_j(\mathbf{d}) &= \frac{1}{2}\,|c_j, c_j(\mathbf{d})| \cdot |a_j, a'| \\ &= \frac{1}{2}\,(|c_j, a'| - |a', c_j(\mathbf{d})|) \cdot |a_j, a'| \\ &= \frac{1}{2}\,(|c_j, a'| - |a_j, a'| \tan(\alpha_j - \varphi_{\mathbf{d}})) \cdot |a_j, a'|,\end{aligned}$$

which can be written as

$$X_j + Y_j \tan(\alpha_j - \varphi_{\mathbf{d}}),$$

where X_j and Y_j are constants independent of $\varphi_{\mathbf{d}} \in I$. Hence,

$$R_j(\mathbf{d}) = A_j + X_j + Y_j \tan(\alpha_j - \varphi_{\mathbf{d}}). \tag{13}$$

Note that the sign of $\Delta_j(\mathbf{d})$ is hidden in the constants.

Suppose we increase the angle $\varphi_{\mathbf{d}}$ from $\pi/2$ to π. The formula for a red area $R_j(\mathbf{d})$ will change, if (i) the vertex a_j of $b_j(\mathbf{d})$, or (ii) the edge e_j changes.

Suppose the center of rotation a_j changes. Let a'_j be the new center of rotation. Then both vertices a_j and a'_j are on the ray from $c_j(\mathbf{d})$ in direction $\mathbf{d}$. If this ray first hits a_j, then the segment $a_j a'_j$ is on the boundary of the visibility cone of a_j, and is in direction $\mathbf{d}$. Similarly, if the ray first hits a'_j, then the segment $a'_j a_j$ is on the boundary of the visibility cone of a'_j, and is in direction $\mathbf{d}$.

Suppose the edge e_j changes. Then $c_j(\mathbf{d})$ must be equal to an endpoint, p, of e_j. In this case pa_j is on the boundary of the visibility cone of p, and is in direction $\mathbf{d}$. This proves the following lemma.

Lemma 11 *The critical directions for the red area are the directions of the bounding rays of the visibility cones of the vertices of $\mathcal{P}$.*

In Sections 3.2.1 and 3.2.2, we show how we can compute and update the expressions for the red areas at these critical directions.

3.2.1 Computing the red area for the vertical direction

Let $\mathbf{d}_v$ be the vertical direction. We compute the red polygons for this direction, as follows. Let BR be the bounding rectangle of $\mathcal{P}$, i.e., the smallest axes-parallel rectangle that contains $\mathcal{P}$. Let $p_l = s_l(\mathbf{d}_v)$ and $p_r = s_r(\mathbf{d}_v)$ be the leftmost and rightmost vertices of $\mathcal{P}$, respectively. We assume for simplicity that these vertices are unique. Note that p_l (resp. p_r) is on the left (resp. right) edge of BR.

The region $F := BR - \mathcal{P}$ consists of simple polygons. Let V be the set of all polygons of F that are not connected to the bottom edge of BR, i.e., the platform. Then V consists of exactly those polygons of F that are above the chain $p_r, p_{r+1}, \ldots, p_l$, which are the vertices of $\mathcal{P}$ encountered when walking from p_r to p_l in counterclockwise direction.

To compute the red areas for direction $\mathbf{d}_v$, we use the *trapezoidal decomposition* of V. This decomposition is defined as follows. For each vertex p of $\{p_r, p_{r+1}, \ldots, p_l\}$, shoot a ray from p in direction $\mathbf{d}_v$, provided that it does not immediately go inside $\mathcal{P}$. This ray intersects either the boundary of $\mathcal{P}$, or the top edge of BR, and it stops at the first intersection point. Similarly, shoot a ray from p in direction $-\mathbf{d}_v$, provided that it does not immediately go inside $\mathcal{P}$. This ray stops at the first intersection, which must lie on the boundary of $\mathcal{P}$. Using the algorithm of Chazelle [4], we can compute the trapezoidal decomposition of the polygons of V in linear time. In fact, it suffices to compute this decomposition using a simple plane sweep algorithm, which takes $O(n \log n)$ time.

The trapezoidal decomposition consists of trapezoids and triangles. We consider a triangle as a degenerate trapezoid. We remove all trapezoids that are connected to the upper edge of BR. It is clear that the remaining trapezoids cover exactly all red polygons. If we know which trapezoid covers which red polygon, then we can compute all areas $R_i(\mathbf{d}_v)$, and all edges $b_i(\mathbf{d}_v)$.

Consider the graph that has a vertex for each trapezoid. Two vertices are connected by an edge, if the corresponding trapezoids have a common vertical boundary edge. We compute the connected components of this graph. The trapezoids of each connected component cover exactly one red support polygon. To compute a red polygon $R_i(\mathbf{d}_v)$ and its area, we simply merge the corresponding trapezoids and add their areas.

We also need for each red polygon $R_i(\mathbf{d}_v)$ its bounding edge $b_i(\mathbf{d}_v)$. Note that we know all trapezoids covering $R_i(\mathbf{d}_v)$. Since $b_i(\mathbf{d}_v)$ is the only edge of $R_i(\mathbf{d}_v)$ that is not an edge of $\mathcal{P}$, we can easily find it by considering all edges of $R_i(\mathbf{d}_v)$. This proves the following lemma.

Lemma 12 *The above algorithm computes all red support polygons, and their total area, for the vertical build direction* $\mathbf{d}_v$*, in* $O(n)$ *time. This algorithm gives for each red polygon* $R_i(\mathbf{d}_v)$*, (i) its center* a_i *of rotation, (ii) its bounding edge* $b_i(\mathbf{d}_v)$*, together with information whether it is a left or right bounding edge, and (iii) the edge* e_i *of* $\mathcal{P}$ *that contains the endpoint* $c_i(\mathbf{d}_v)$ *of* $b_i(\mathbf{d}_v)$*.*

In Section 3.2.2, we will see that when $\varphi_{\mathbf{d}}$ increases from $\pi/2$ to π, a red polygon can be split into two red polygons. Hence, if we store with each red polygon its area expression $R_i(\mathbf{d}) = A_i + \Delta_i(\mathbf{d})$ explicitly, then we have to compute for each new red polygon its constant term A_i. Clearly, this takes too much time. Since we are only interested in the sum of all red areas, however, we maintain a global constant part A_G, whose value is $\sum_i A_i$.

3.2.2 Updating the red area at a critical direction

The formula for the area $R_i(\mathbf{d})$ of a red polygon is completely determined by (i) a constant term A_i, (ii) a center a_i of rotation, and (iii) an edge e_i of $\mathcal{P}$. Given this information, we can compute the intersection $c_i(\mathbf{d})$ of the ray from a_i in direction $-\mathbf{d}$ with edge e_i, and the area $\Delta_i(\mathbf{d})$ of the triangle in Figure 9.

Let $\mathbf{d}$ be a critical direction for the red support polygons, and assume that we have the expressions for the areas of the red support polygons for this direction.

Since $\mathbf{d}$ is critical, there are two vertices x and y of $\mathcal{P}$, such that the segment xy is on a bounding ray of the visibility cone of x, and this segment has direction $\mathbf{d}$. We will show how to update the expression for the red area at this direction. We distinguish several cases.

Case 1: y is the center of rotation of a red polygon $R_i(\mathbf{d})$, the ray from x in direction $-\mathbf{d}$ immediately enters $\mathcal{P}$, and xy is not an edge of $\mathcal{P}$. See Figure 10(a).

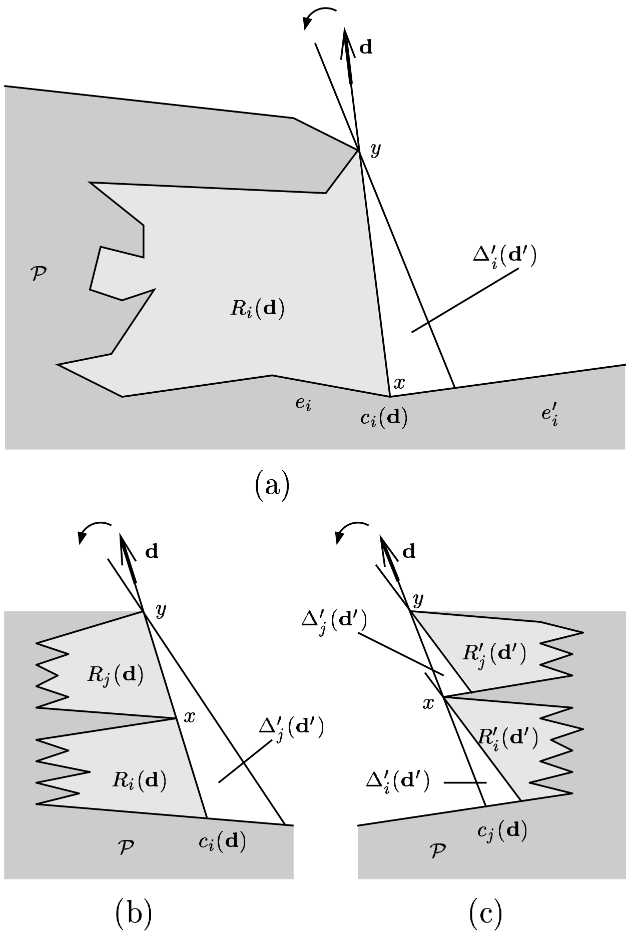

Fig. 10. *Cases 1, 2 and 3.1.* $\mathbf{d}$ *is a critical direction,* $\mathbf{d}'$ *is a direction such that* $\varphi_{\mathbf{d}'}$ *is slightly larger than* $\varphi_{\mathbf{d}}$.

In this case, the edge e_i of $R_i(\mathbf{d})$ changes to, say, e'_i. Assume that the area $R_i(\mathbf{d})$ is increasing. (The case when the area decreases can be handled similarly.) Since e_i is changing, the endpoint $c_i(\mathbf{d})$ of $b_i(\mathbf{d})$ is equal to x. Moreover, x is the common endpoint of e_i and e'_i. We add the area $\Delta_i(\mathbf{d})$ to the global variable A_G, and set up the formula for the area of the new triangle $\Delta'_i(\mathbf{d})$ that is determined by y and e'_i.

Case 2: x and y are centers of rotation of red polygons $R_i(\mathbf{d})$ and $R_j(\mathbf{d})$, respectively. The edges $b_i(\mathbf{d})$ and $b_j(\mathbf{d})$ are both right bounding edges. See Figure 10(b).

In this case, we have $x = a_i = c_j(\mathbf{d})$, and $y = a_j$. Also, a_i ceases being a center of rotation, and the polygons $R_i(\mathbf{d})$ and $R_j(\mathbf{d})$ are combined into a new red polygon $R'_j(\mathbf{d})$. We add the value $\Delta_i(\mathbf{d}) + \Delta_j(\mathbf{d})$ to the global variable A_G, and set up the formula for the area of the new triangle $\Delta'_j(\mathbf{d})$. This triangle is determined by a_j and edge e_i.

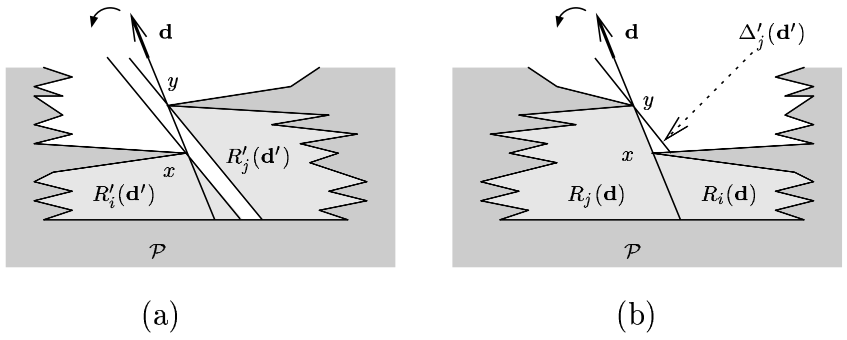

Fig. 11. *Cases 3.2 and 4.* $\mathbf{d}$ *is a critical direction,* $\mathbf{d}'$ *is a direction such that* $\varphi_{\mathbf{d}'}$ *is slightly larger than* $\varphi_{\mathbf{d}}$.

Case 3: y is the center of rotation of a red polygon $R_j(\mathbf{d})$, and x is not a center of rotation. Moreover, the ray from x in direction $-\mathbf{d}$ does not immediately enter $\mathcal{P}$. Finally, $b_j(\mathbf{d})$ is a left bounding edge.

In this case, we have $y = a_j$. Also, vertex x becomes a center of rotation.

Case 3.1: The two edges adjacent to x are to the right of $b_j(\mathbf{d})$. See Figure 10(c).

The red polygon $R_j(\mathbf{d})$ is split into two independent red polygons $R'_i(\mathbf{d})$ and $R'_j(\mathbf{d})$. We add the (signed) area of the old triangle $\Delta_j(\mathbf{d})$ to the global variable A_G. Furthermore, we set up formulas for the areas of the two new triangles $\Delta'_i(\mathbf{d})$ and $\Delta'_j(\mathbf{d})$. Triangle $\Delta'_i(\mathbf{d})$ is determined by x and the edge e_j of the old triangle $\Delta_j(\mathbf{d})$. Triangle $\Delta'_j(\mathbf{d})$ is determined by $y = a_j$, and that edge e'_j of $\mathcal{P}$ having x as endpoint, whose other endpoint is higher w.r.t. direction $\mathbf{d}$. Note that the signs of $\Delta'_i(\mathbf{d})$ and $\Delta'_j(\mathbf{d})$ are both negative.

Case 3.2: The two edges adjacent to x are to the left of $b_j(\mathbf{d})$. See Figure 11(a).

The red polygon $R_j(\mathbf{d})$ is split into two independent red polygons $R'_i(\mathbf{d})$ and $R'_j(\mathbf{d})$. We add the area of the old triangle $\Delta_j(\mathbf{d})$ to the global variable A_G. Furthermore, we set up formulas for the areas of the two new triangles $\Delta'_i(\mathbf{d})$ and $\Delta'_j(\mathbf{d})$, having x and $y = a_j$ as centers of rotation, respectively. Since xy has direction $\mathbf{d}$, the edges e'_i and e'_j of these two triangles are the same. We find them by shooting a ray from x in direction $-\mathbf{d}$; the first edge of $\mathcal{P}$ that is hit is e'_i.

Case 4: x and y are centers of rotation of red polygons $R_i(\mathbf{d})$ and $R_j(\mathbf{d})$, respectively. The edges $b_i(\mathbf{d})$ and $b_j(\mathbf{d})$ are left and right bounding edges, respectively. See Figure 11(b).

In this case, we have $x = a_i$, and $y = a_j$. The two polygons $R_i(\mathbf{d})$ and $R_j(\mathbf{d})$ are combined into one polygon $R'_j(\mathbf{d})$. We add the areas of the old triangles

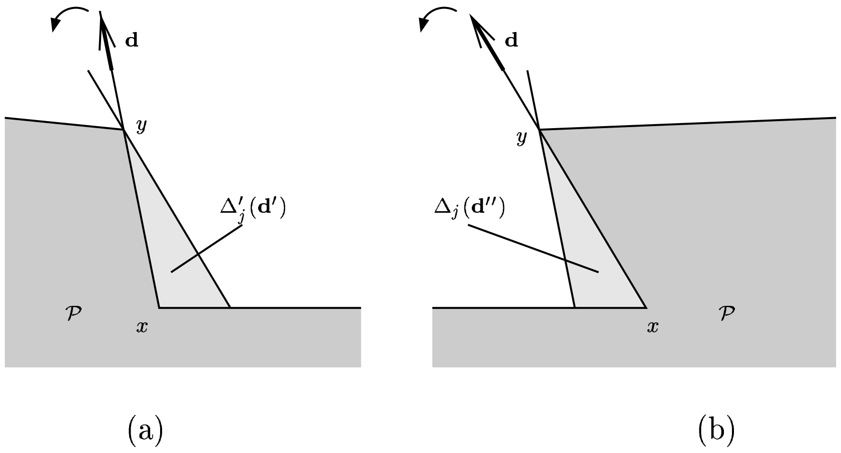

Fig. 12. *Cases 5 and 6.* $\mathbf{d}$ *is a critical direction,* $\mathbf{d}'$ *and* $\mathbf{d}''$ *are directions such that* $\varphi_{\mathbf{d}'}$ *and* $\varphi_{\mathbf{d}''}$ *are slightly larger and smaller than* $\varphi_{\mathbf{d}}$*, respectively.*

$\Delta_i(\mathbf{d})$ and $\Delta_j(\mathbf{d})$ to A_G. Note that the sign of $\Delta_i(\mathbf{d})$ is negative, and that of $\Delta_j(\mathbf{d})$ is positive.

We set up the formula for the area of the new triangle $\Delta'_j(\mathbf{d})$. This triangle is determined by $y = a_j$, and the edge having $x = a_i$ as endpoint, whose other endpoint is higher w.r.t. direction $\mathbf{d}$.

Case 5: y is not a center of rotation. See Figure 12(a).

In this case, the vertices x and y are connected by an edge, say e, of $\mathcal{P}$. Also, the outer normal of e is to the right of e, w.r.t. direction $\mathbf{d}$. A new red polygon $R'_j(\mathbf{d})$ with center of rotation y arises. At this direction, this polygon is a triangle, $\Delta'_j(\mathbf{d})$, which is determined by y and the edge e'_j of $\mathcal{P}$ with endpoint x that is not incident to y.

Case 6: y is the center of rotation of a red polygon $R_j(\mathbf{d})$, the ray from x in direction $-\mathbf{d}$ immediately enters $\mathcal{P}$, and xy is an edge of $\mathcal{P}$. See Figure 12(b).

In this case, $R_j(\mathbf{d})$ is a triangle, and the outer normal of e is to the left of e. Since this red polygon vanishes, we subtract the constant part of its area from A_G, and delete the non-constant part of the formula.

Since we assume that our polygon $\mathcal{P}$ is in general position, in the sense that no three vertices lie on a line, we have covered all cases. The following lemma follows from our discussion.

Lemma 13 *At a critical direction, we can update the expression for the red area in* $O(\log n)$ *time.*

We are ready now to give our algorithm.

Step 1: Preprocess $\mathcal{P}$ as indicated in Lemmas 10 and 12. This gives the expressions for the support area for the vertical direction.

Step 2: Compute the visibility cones of all vertices of $\mathcal{P}$. Let $\mathbf{D}_a$ be the set of all directions determined by the bounding rays of the non-empty cones. Compute the convex hull of $\mathcal{P}$. Compute the set $\mathbf{D}_b$ containing the following directions. For each edge e of the convex hull, $\mathbf{D}_b$ contains the the two directions parallel to e, and the inner normal of e. Sort the directions of the set $\mathbf{D} := \mathbf{D}_a \cup \mathbf{D}_b$. Finally, preprocess $\mathcal{P}$ such that ray shooting queries can be solved in $O(\log n)$ time.

Step 3: Sweep over the elements of $\mathbf{D}$, thereby visiting the corresponding subintervals one after another. For each subinterval I, update expressions (12) and (13), and compute its minimum within I. The global minimum of the area over all these subintervals is the desired result.

Consider the minimization problem within one subinterval I. As in Section 2.2, we can reduce this to the following problem.

Problem $PR'(n)$: *Given $2n+7$ real numbers $c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n$, B, C, D, E, F, and x_0 and x_1, compute the global minimum of the function*

$$H(x) := B \cdot x + C \cdot \frac{1}{x} + \frac{1}{1+x^2} \cdot \left[\frac{D}{x} + E + F \cdot x\right] + \sum_{j=1}^{n} \left(\frac{d_j}{1+c_j x}\right),$$

for $x \in [x_0, x_1]$.

If we solve this problem using standard calculus techniques, then we have to compute the roots of the derivative of H, which leads to a polynomial whose degree is linear in n. As in Section 2.2, we are not aware of any efficient algorithm that solves Problem $PR'(n)$, and we leave its design and analysis as an open problem. In the theorem below, we denote the time it takes to solve $PR'(n)$ generically by $q(n)$.

Theorem 14 *Let $\mathcal{P}$ be a simple polygon with n vertices, and let $q(n)$ be the time needed for solving problem $PR'(n)$. In $O(n \log n + n\, q(n))$ time, we can compute a direction* $\mathbf{d}$ *for which the total area of the supports is minimal.*

If the polygon $\mathcal{P}$ is C-oriented for some constant C, then this optimal direction *can be computed in* $O(n \log n)$ *time.*

Proof. The first claim follows from the above discussion. If $\mathcal{P}$ is C-oriented, then the summation in the function $H(x)$ contains at most C terms. Therefore, if we assume that the roots of a polynomial of constant degree can be computed in constant time, then we can compute the minimum of $H(x)$ for one subinterval in constant time. ∎

4 Minimizing the trapped area for simple polygons

In this section, we want to minimize the trapped area for a simple polygon $\mathcal{P}$. We show that we can use basically the same algorithm as in Section 3.

Recall that the trapped area is the total area of those regions that are separate from the main body of liquid in the vat. We define this notion formally. Let $\mathbf{d}$ be a build direction, and let $\mathbf{d}'$ be the direction that makes a clockwise angle of $\pi/2$ with $\mathbf{d}$. Hence, if $\mathbf{d}$ is directed vertically upwards, then $\mathbf{d}'$ is directed horizontally to the right.

Let x be a point in the exterior of $\mathcal{P}$. We say that x belongs to a *trapped region for direction* $\mathbf{d}$, if the ray from x in direction $\mathbf{d}$ does not intersect the interior of $\mathcal{P}$, but the two rays from x in directions $\mathbf{d}'$ and $-\mathbf{d}'$ both intersect $\mathcal{P}$'s interior. (Refer to Figure 13.) These trapped regions consist of a collection of simple polygons, which we call *trapped polygons.* The *trapped area for build direction* $\mathbf{d}$ is defined as the total area of all trapped polygons for this direction. Note that the trapped area does not include the area of the supports.

Let $V(\mathbf{d})$ be a trapped polygon for direction $\mathbf{d}$. The boundary of $V(\mathbf{d})$ is in contact with edges of P and certain red support polygons $R_i(\mathbf{d})$, $1 \leq i \leq l$, for direction $\mathbf{d}$. Also, $V(\mathbf{d})$ is bounded from above, w.r.t. $\mathbf{d}$, by a line segment $l(\mathbf{d})$ that is parallel to direction $\mathbf{d}'$. If the angle $\varphi_{\mathbf{d}}$ is varied slightly, then $l(\mathbf{d})$ rotates around one of its endpoints, say $a(\mathbf{d})$, which is a vertex of $\mathcal{P}$, whereas the other endpoint moves along an edge, say $e(\mathbf{d})$, which is an edge of $\mathcal{P}$ or a red polygon. Note that in the latter case, $e(\mathbf{d})$ also rotates around a vertex of $\mathcal{P}$.

Let $V'(\mathbf{d})$ denote the polygon which is the union of $V(\mathbf{d})$ and the red polygons $R_i(\mathbf{d})$ that are in contact with $V(\mathbf{d})$. Then, the area of $V(\mathbf{d})$ is equal to the area of $V'(\mathbf{d})$ minus the total area of these red polygons.

This suggests the following approach for minimizing the trapped area. For a given interval $I = [\alpha, \beta]$, let $\mathbf{d}_0$ be the direction such that $\varphi_{\mathbf{d}_0} = \alpha$. Then as in Section 3.2, the area of $V'(\mathbf{d})$ is equal to the area of $V'(\mathbf{d}_0)$ plus the signed

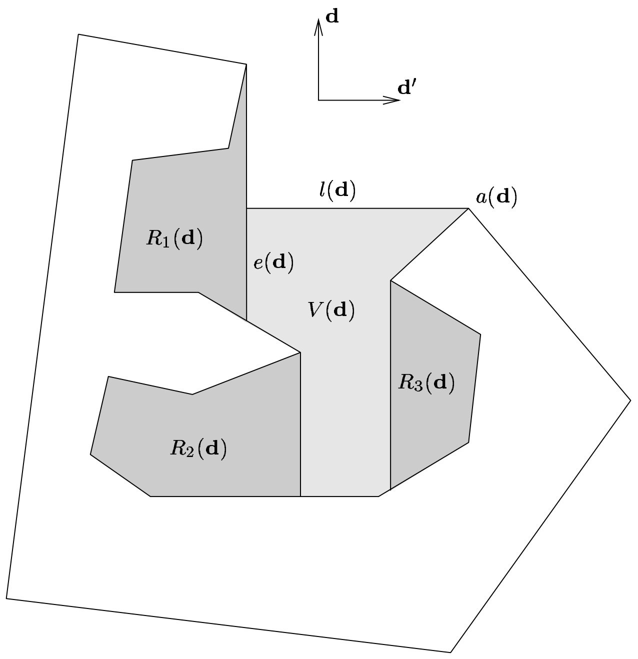

Fig. 13. *$R_1(\mathbf{d})$, $R_2(\mathbf{d})$ and $R_3(\mathbf{d})$ are red support polygons, $V(\mathbf{d})$ is a trapped polygon.*

---

areas of at most two triangles that are determined by $l(\mathbf{d})$, $l(\mathbf{d}_0)$, $e(\mathbf{d})$ and $e(\mathbf{d}_0)$. In this way, the total area of all the trapped polygons $V(\mathbf{d})$ can be written in the form of problem $PR'(n)$.

The area formula changes if (i) one of the formulas for $R_i(\mathbf{d})$ changes, (ii) the "description" of $e(\mathbf{d})$ changes, or (iii) the vertex $a(\mathbf{d})$ changes. It is not difficult to see that (ii) (resp. (iii)) can only happen if $\mathbf{d}$ is parallel (resp. orthogonal) to a bounding ray of a visibility cone.

Hence, we can basically apply the same algorithm as in Section 3, except that the number of critical directions increases by at most $2n$. This leads to the following result.

Theorem 15 *Let $\mathcal{P}$ be a simple polygon with n vertices, and let $q(n)$ be the time needed for solving problem $PR'(n)$. In $O(n \log n + n\, q(n))$ time, we can compute a direction $\mathbf{d}$ for which the trapped area is minimal.*

If the polygon $\mathcal{P}$ is C-oriented for some constant C, then this optimal direction can be computed in $O(n \log n)$ time.

5 Concluding remarks

We have given geometric algorithms for some optimization problems arising in layered manufacturing.

An interesting open problem that we are pursuing is the design of efficient algorithms for optimizing supports for a non-convex three-dimensional polyhedron. We believe that the ideas developed in this paper will be very helpful in this effort.

6 Acknowledgements

The fourth author thanks Yi-Jen Chiang and Jack Snoeyink for helpful discussions. The authors thank the referees for helpful feedback.

References

[1] S. Allen and D. Dutta. Determination and evaluation of support structures in layered manufacturing. *Journal of Design and Manufacturing*, 5:153–162, 1995.

[2] E. M. Arkin, Y.-J. Chiang, M. Held, J. S. B. Mitchell, V. Sacristan, S. S. Skiena, and T.-C. Yang. On minimum-area hulls. In *Proc. 4th Annu. European Sympos. Algorithms*, volume 1136 of *Lecture Notes Comput. Sci.*, pages 334–348. Springer-Verlag, 1996.

[3] 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.

[4] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6:485–524, 1991.

[5] B. Chazelle et al. Application challenges to computational geometry: CG impact task force report. Technical Report TR-521-96, Princeton Univ., 1996.

[6] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin, 1987.

[7] H. Freeman and R. Shapira. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Commun. ACM*, 18:409–413, 1975.

[8] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[9] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.

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

[11] J. Majhi, R. Janardan, M. Smid, and P. Gupta. On some geometric optimization problems in layered manufacturing. In *Proc. 5th Workshop Algorithms Data Struct.*, volume 1272 of *Lecture Notes Comput. Sci.*, pages 136–149. Springer-Verlag, 1997.

[12] J. Majhi, R. Janardan, M. Smid, and P. Gupta. On some geometric optimization problems in layered manufacturing. *Computational Geometry: Theory and Applications*, to appear.

[13] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 1994.

[14] G. T. Toussaint. A linear-time algorithm for solving the strong hidden-line problem in a simple polygon. *Pattern Recogn. Lett.*, 4:449–451, 1986.