Peeling Meshed Potatoes

Boris Aronov

Marc van Kreveld

Maarten Löffler

Rodrigo I. Silveira

Technical Report UU-CS-2009-010
April 2009

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Peeling Meshed Potatoes*

Boris Aronov† Marc van Kreveld‡ Maarten Löffler‡ Rodrigo I. Silveira‡

Abstract

We study variants of the *potato peeling* problem on meshed (triangulated) polygons. Given a polygon with holes, and a triangular mesh that covers its interior (possibly using additional vertices), we want to find a largest-area connected set of triangles of the mesh that is convex, or has some other shape-related property. In particular, we consider (i) convexity, (ii) monotonicity, (iii) bounded backturn, and (iv) bounded total turning angle. The first three problems are solved in polynomial time, whereas the fourth problem is shown to be NP-hard.

1 Introduction

Suppose you live in a mountainous area, and you are interested in playing golf and drinking wine. Then you would like to find a large subregion of the area that is suitable to turn into a golf course or a vineyard. In the first case, this means your region should be reasonably flat. In the second case, it probably means your region should 'face south' (assuming that you live in the Northern Hemisphere). In addition, it is reasonable to want your region to have a compact shape and, at the same time, to be as large as possible.

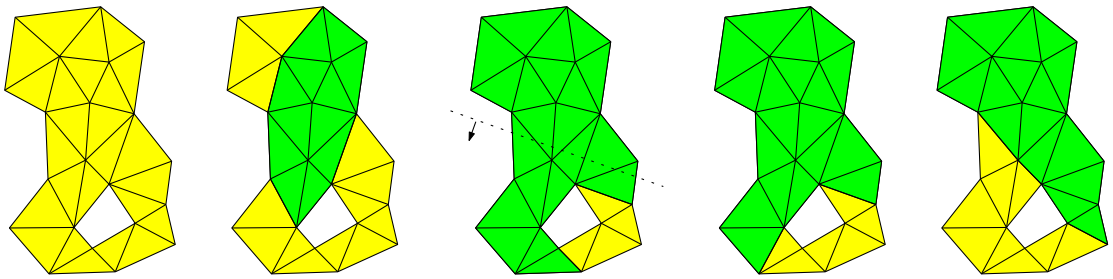
In Geographic Information Science, terrain data is often stored as a triangulated irregular network (TIN): a planar triangulation with additional height data on the vertices. In this context, it is easy to evaluate properties such as 'flatness' and 'facing south' on a triangle-by-triangle basis: each triangle is part of a plane and thus has a constant angle with respect to the horizontal or to the optimal south-facing orientation. We can identify for each triangle whether its angle is below a certain prespecified threshold, and discard it if it is not. After this, we are left with a collection of triangles in the plane, of which we want to find a largest-area subset that has a *good shape*—from here on, the height information can be ignored. Note that we can assume that the collection of triangles we get as input is connected, since otherwise we can just search for good regions within each connected component separately.

Now consider a seemingly unrelated problem. In computer graphics, before a scene is rendered an *occlusion culling* algorithm is often used, which serves to quickly remove a large portion of the scene that is hidden behind some object (occluder). Such occluders are often large and contain many triangles. However, culling algorithms work much faster with convex occluders, so it is interesting to find a large convex subset of a given occluder ('convex' should be interpreted in the 2-dimensional sense, as seen from the current point of view).

*Work by B.A. has been supported in part by a grant from the U.S.-Israeli Binational Science Foundation and by NSA MSP Grant H98230-06-1-0016. Partially funded by the Netherlands Organization for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.065.503, and under the project GOGO. A preliminary version of this paper was presented at the 19th Canadian Conference on Computational Geometry (CCCG 2007), under the title "Largest Subsets of Triangles in a Triangulation".

†Department of Computer Science and Engineering, Polytechnic Institute of NYU, Brooklyn, New York, USA. http://cis.poly.edu/~aronov

‡Department of Computer Science, Utrecht University, the Netherlands. {marc,loffler,rodrigo}@cs.uu.nl.



(a) (b) (c) (d) (e)

Figure 1: (a) A meshed potato with a hole. We will refer to the unbounded face and to the holes as the *exterior* of the polygon. (b) The largest convex subpotato. (c) The largest monotone subpotato (the direction of monotonicity is shown with an arrow) (d) The largest backturn-constrained subpotato with a maximum backturn of $\approx \frac{\pi}{2}$. (e) The largest subpotato with a total turning angle of at most 3π.

Both of these problems can be abstracted in the same way. We have a triangulated polygon (with extra interior vertices) with holes, of which we want to find a large subregion with a good shape. Finding well-shaped subregions of polygons is a problem that received quite some attention in computational geometry before, and one of its representatives is the well-known *potato peeling problem* [14], which asks for the largest convex subpolygon of a given polygon (potato). In our case, the problem is the same, except that we have an additional triangulation or *mesh* on the potato, and we require the solution to respect that mesh. Asking for convexity, in the context of our first example, can be a way to model the vague criterion of good shape for a region. However, because convexity is a rather restrictive condition in combination with respecting the triangular mesh, and because many applications may allow some flexibility on this requirement, we also study some relaxations of convexity that can be used to model the idea of a well-shaped region.

Problem description We are given a polygon with holes (or potato) $\mathcal{P}$, and a triangular mesh $\mathcal{M}$ that covers the interior of $\mathcal{P}$, and possibly has additional interior vertices. The total complexity of $\mathcal{P}$ and $\mathcal{M}$ is denoted by n, and is defined as their total number of vertices. We want to find the largest-area simply-connected region R comprised of triangles of $\mathcal{M}$ such that:

(i) it is convex (see Figure 1(b)),
(ii) it is monotone in some direction (see Figure 1(c)),
(iii) it has a maximum backturn of at most γ (see Figure 1(d)), or
(iv) it has total turning angle of at most β (see Figure 1(e)).

That is, we treat four different shape restrictions: convexity and three possible relaxations, explained below. Since the region boundary is required to follow the edges of $\mathcal{M}$, forcing R to be convex can be quite restrictive, while intuitively this should not be the case. For example, in Figure 2 a situation is depicted in which a very large area is available, but only rather small convex subregions exist. The problem is that because of the mesh, the boundary needs to make a small turn at every vertex, and in a convex polygon we are only allowed to turn in one direction. To counteract this problem, we study three more variants.

A polygon is said to be *monotone* if it is possible to rotate the polygon such that any vertical line intersects the polygon boundary at most twice. This allows for flexibility in the shape of R, but it comes at a price: There are monotone shapes that do not look nice and compact at all. Therefore we introduce two other measures with a parameter that specifies how 'close to convex' a region is.

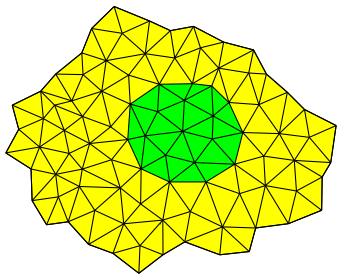


Figure 2: The largest convex subregion of a meshed polygon is restricted by the mesh, and not by the polygon boundary.

We define a *backturn* in a polygon as a turn of the polygon boundary that is made in the wrong direction. More formally, let R be defined by the edges $\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_m$ in counterclockwise order along its boundary, see Figure 3. We define the *turning angle* from edge $\vec{e}_i$ to edge $\vec{e}_j$ for $i \neq j$ as $\alpha(\vec{e}_i, \vec{e}_j) = \sum_{k=i}^{j-1} \measuredangle(\vec{e}_k, \vec{e}_{k+1})$ (indices wrap around, the sum is taken over at most $n-1$ angles), where $\measuredangle(\vec{e}_k, \vec{e}_{k+1}) \in (-\pi, \pi)$ is the counterclockwise turn from $\vec{e}_k$ to $\vec{e}_{k+1}$ (a clockwise turn yields a negative $\measuredangle(\vec{e}_k, \vec{e}_{k+1})$). When $i = j$, we set $\alpha(\vec{e}_i, \vec{e}_i) = 0$. The *maximum backturn* of a simple polygon R, Φ_R, is defined as $\min_{i,j} \alpha(\vec{e}_i, \vec{e}_j)$. It represents the maximum amount that part of the polygon boundary turns in the wrong direction. Note that $\Phi_R \leq 0$, and that a simple polygon R is convex if and only if $\Phi_R = 0$. When the polygon under consideration is clear, we will write Φ instead of Φ_R.

Our last relaxation of convexity also deals with the idea of not being too concave. We define the *total turning angle* of a polygon as the sum of the absolute values of the turning angles at the vertices. We will look for a polygon whose total turning angle does not exceed a predefined constant $\beta \geq 2\pi$. A convex polygon has a turning angle of exactly 2π, but we allow the polygon to turn 'the wrong way' a little bit, as long as the total turning angle is less than β.

Results We show that problem (i) can be solved in $O(n^2)$ time, problem (ii) in $O(n^2)$ time if the direction is given and in $O(n^3)$ time otherwise, problem (iii) in $O(n^6)$ time, and problem (iv) is NP-hard.

Related work As mentioned earlier, if the largest-area simple polygon we aim to compute were not constrained to be a union of triangles of $\mathcal{M}$, but just to be contained in $\mathcal{P}$, problem (i) would turn into the *potato peeling problem* [14], also known as the *convex skull problem* [19]. The problem consists in, given a simple polygon $\mathcal{P}$, determining the largest-area convex polygon inside $\mathcal{P}$. If $\mathcal{P}$ has n vertices, it can be solved in time $O(n^7)$ for a simple polygon, and in time $O(n^8)$ if

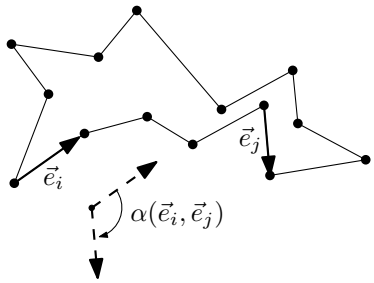


Figure 3: The maximum backturn (negative turning angle) of the polygon is $-\frac{2}{3}\pi$, and is achieved between $\vec{e}_i$ and $\vec{e}_j$.

the polygon has holes [6]. Approximation algorithms were studied by Hall-Holt *et al.* [15]. The orthogonal version of the problem, where both the given polygon and the subpolygon sought are bounded only by horizontal and vertical line segments, can be solved exactly in $O(n^2)$ time with an algorithm by Wood and Yap [20].

Also similar to the problems in this paper are many containment problems, in particular finding largest-area subpolygons. There is a large number of articles on different variants of this problem; we name just a few of them here. Barequet and Rogol [4] find a maximum area axially symmetric polygon contained in a given simple polygon. Daniels *et al.* [10] present an algorithm to find the largest-area axis-parallel rectangle in a general n-vertex polygon (allowing holes), which can be improved to run faster if the polygon is simple [5]. Another related problem is finding the largest similar copy of a polygonal pattern inside a 2D environment with obstacles [7]. In particular, several algorithms deal with the case when the pattern is a convex polygon [18]. Arkin *et al.* [3] study the problem of finding the *smallest superpolygon* of a given polygon, rather than the largest subpolygon. For convex polygons, this is simply the convex hull; they show that a smallest monotone or star-shaped superpolygon can be computed in $O(n \log n)$, respectively, $O(n^2)$ time.

Our problems (iii) and (iv) employ turning angles to capture a weaker notion of convexity. Turning angles have been studied before for optimization of geometric tours [1, 11] and also in contexts more similar to ours, for describing polygonal shapes. In particular, the closely-related *turning functions* have been used to compare the shape of polygons [2, 16]. Closer to our application, Reinbacher *et al.* [17] consider minimizing the total angular change as a shape criterion to delineate imprecise regions with compact shapes.

Occlusion culling is a well known technique in computer graphics, and occluders that are large and convex give faster results [9]. Cohen-Or *et al.* [8] use this fact by covering a simple polygon with a few large convex subpolygons. They present a heuristic that first subdivides the polygon into convex regions and then selects a subset of those regions to be combined into a larger subpolygon. Papaioannou *et al.* [12] construct large convex occluders by grouping partially overlapping smaller occluders together into convex shapes.

Outline of the paper The next section studies the first two problems: finding a largest-area polygon that is the union of triangles and is convex, and finding one that is monotone in some direction. Section 3 deals with problem (iii), where the restriction is now on the maximum backturn, and gives a polynomial-time algorithm to solve it. Section 4 studies problem (iv), finding a largest-area simple polygon with bounded total turning angle, and shows the problem is NP-hard. Finally, in Section 5 we make some concluding remarks and mention some problems left open.

Henceforth, we will refer to the holes in the polygon and to its unbounded face as the *exterior* of the polygon, or simply as *the exterior*. We assume that the triangles are closed, and therefore, that the exterior is open. Slightly abusing the notation, we will not distinguish between a subset of triangles and their union.

2 The largest convex polygon

In this section we study the problems of computing the largest-area convex polygon and the largest-area monotone polygon formed as a union of triangles of $\mathcal{M}$. Most of the section is devoted to the first problem; adaptations needed for the second problem are discussed in the last subsection. Without loss of generality, we assume that no two vertices of $\mathcal{M}$ have the same x-coordinate. Before presenting the algorithm, we will introduce some notation and definitions.

Let $e \in \mathcal{M}$ be an edge of the triangulation. We denote by $X(e)$ the x-span of the edge, that is, the projection of the edge onto the x-axis. For edges $e, e' \in \mathcal{M}$, we say that e is *above* e' if there

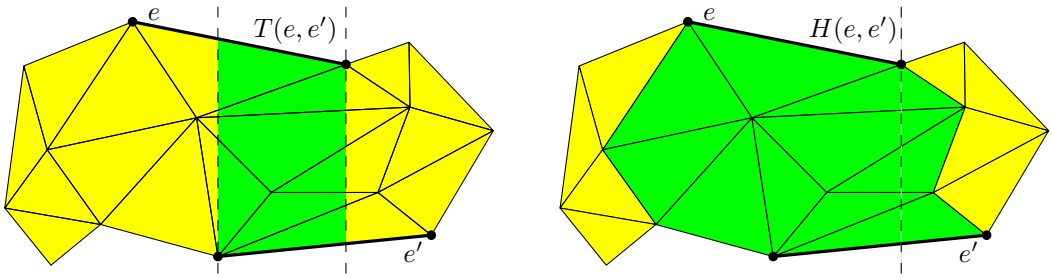


Figure 4: (a) A promising pair of edges and their trapezoid $T(e, e')$. (b) An admissible collection C of triangles (gray) for the edge pair (e, e').

is a vertical line that meets e at a point p and e' at a point p' with p above p'. Assume that e is above e'. Define $X(e, e') := X(e) \cap X(e')$. We now define $T(e, e')$ as the trapezoid enclosed by e from above, by e' from below, and by the vertical lines that bound the interval $X(e, e')$. If $T(e, e')$ avoids the exterior of $\mathcal{P}$, we say that the pair (e, e') is *promising* (see Figure 4(a)).

Let (e, e') be a promising pair of edges, and let $H(e, e')$ be the halfplane to the left of the line $x = \max X(e, e')$ (the same line that bounds $T(e, e')$ from the right). Let C be a collection of triangles of $\mathcal{M}$. We call C *admissible* for (e, e') if the following conditions hold (see Figure 4(b)):

- The pair (e, e') is promising.
- C contains the triangle below e.
- C contains the triangle above e'.
- Every triangle of C intersects $H(e, e')$.
- The polygon $U = U(C) := \bigcup C \cap H(e, e')$ is convex, where $\bigcup C$ is the union of the triangles in C.

Note that the last condition implies that U cannot contain holes or any other part of the exterior.

We will now define a value $Q(e, e')$ for any pair of edges. Set $Q(e, e') := -\infty$ if (e, e') has no admissible collection of triangles, otherwise $Q(e, e')$ is defined to be the largest area$(U(C))$ achieved by any admissible collection C. Clearly, any pair that is not promising has a Q-value of $-\infty$. The converse, however, is not true: a pair can be promising, but not have any admissible collection of triangles.

We will compute $Q(e, e')$ by dynamic programming. Notice that if e and e' share their right endpoints, $Q(e, e')$ is precisely the area of the largest convex polygon consisting of triangles from $\mathcal{M}$ and having e and e' as its rightmost top and bottom edges, respectively. Hence, examining $Q(e, e')$ over all pairs of edges with the same right endpoint, one can find the area of the desired largest-area convex polygon. The polygon itself can be extracted by standard dynamic programming methods.

We will first describe an algorithm to identify all the promising pairs in $O(n^2)$ time. After this, we show how to compute the value Q for all those pairs, again in $O(n^2)$ time. The two algorithms rely on a plane sweep and can in principle be combined into a single pass, but we keep them separate for clarity.

2.1 Computing the promising pairs

We sweep $\mathcal{M}$ by a vertical sweep-line ℓ moving from left to right and stopping at every vertex. Using standard sweep techniques, we maintain the sorted order of the edges of $\mathcal{M}$ currently

intersected by ℓ. At a given point of the sweep, when we are at a sweep-line $\ell : x = a$, all the promising pairs with $X(e, e')$ lying to the left of a have already been discovered. All pairs with $X(e, e') \ni a$ for which the part of $T(e, e')$ to the left of ℓ does not meet the exterior of $\mathcal{P}$ are called *potentially promising* and are currently being maintained.

Along the sweep, we maintain the following information. For each edge e meeting the sweep-line, we keep two sorted lists $E_a(e)$ and $E_b(e)$. $E_b(e)$ contains all the edges e' meeting ℓ below e, which have their left endpoint to the left of that of e and such that the intersection of $T(e, e')$ with the halfplane $x < a$ lies completely inside $\mathcal{P}$. $E_b(e)$ is stored sorted in downward direction along ℓ. $E_a(e)$ is analogous but for the edges lying above e. In addition to these two lists, we also keep an inverted list $E_I(e)$ that contains all the edges e' such that $E_a(e')$ or $E_b(e')$ contains e. The need for these cross-references will become clear later.

We discern three different types of events that all occur when the sweep-line passes a vertex v. At this vertex, some edges end, which means potentially promising pairs using those edges are indeed promising and must be output and removed. Secondly, some new edges begin at v, which means that new potentially promising pairs must be generated and maintained. Thirdly, we may encounter parts of the exterior of $\mathcal{P}$ at v, which means that any pairs we are currently maintaining that have one edge above and one edge below v are no longer potentially promising. We process the events in the order they are described here: we first process the beginning edges, then the ones ending, and finally the new exterior parts.

A new beginning When a new edge e begins at v, we want to find all potentially promising pairs of the form (e, e') or (e', e) that cross ℓ. At this point, this just means that the exterior cannot intersect ℓ between them as promising. To process multiple edges starting at v correctly, we must treat them in the order in which these edges would cross ℓ if it were slightly to the right of v. We walk upwards along ℓ, collecting all the edges crossing ℓ above e, and adding them to $E_a(e)$ (and the corresponding cross-references to the E_I lists). This continues until a part of the boundary of $\mathcal{P}$ is reached: after that, no more promising pairs exist. In the same way, we walk downwards along ℓ to fill in $E_b(e)$. We spend constant time per added pair.

The end When an edge e ends at v, we want to find all potentially promising pairs that use e that we have stored, report them, and then remove them from the data structure. There are two types of such pairs: the ones for which the left endpoint of e lies to the right of the left endpoint of the other edge are stored in the lists $E_a(e)$ and $E_b(e)$ of e. The other pairs, which have the left endpoint of e to the left of the left endpoint of the other edge, are stored in the lists of the other edges. For these, we will use the cross-pointers in $E_I(e)$ to find them. In time proportional to the sizes of the lists stored with e, we find all such pairs. Since the lists are linked, we can remove the references in the lists of the other edges in constant time per entry, so we spend only constant time per removed pair.

Exterior parts The last type of event occurs when a part of the exterior begins at the vertex v. Intuitively we want to remove all pairs (e, e') for which $T(e, e')$ contains v. Each such no-longer-promising pair has a top edge e above v and a bottom edge e' below v. By the invariant we have been maintaining, we know that no other point from the exterior appears in $T(e, e') \cap \{x < a\}$. We walk up ℓ, from v, looking for candidates e, until we hit a part of the exterior. For each e found, we scan its list $E_b(e)$, from the tail backwards. If the last element e' is above v, the entire list is safe. If it is below v, the pair (e, e') is not promising, and we drop e' from $E_b(e)$ and e from $E_I(e')$, continuing upwards until the first e' above v. This is repeated until all no longer promising pairs have been discarded. Similarly, we walk down ℓ from v and scan $E_a(e)$.

For each list that does not need to be truncated we spend only $O(1)$ time, for $O(n)$ lists at each vertex, taking in total $O(n^2)$ time. In addition, constant time is spent on each pair that is

discarded, which can occur at most once over the execution of the algorithm for each pair, so we do at most $O(n^2)$ work overall. It follows that all the promising pairs can be computed in a total of $O(n^2)$ time.

2.2 Computing the Q-values

Having established how to identify promising pairs (e, e'), we explain how to compute the values $Q(e, e')$ for those pairs. Recall that this value corresponds to the area of a convex polygon bounded from the right by the line $x = \max X(e, e')$, and following edges of $\mathcal{M}$ for the rest. We view this polygon as a concatenation of vertical strips, the regions $T(\cdot, \cdot)$ of promising pairs, which guarantees that only triangles, and no parts of the exterior, are used. This allows us to incrementally compute the Q-values by dynamic programming.

Fix a promising pair (e, e'). To compute $Q(e, e')$, consider the left endpoints of both segments. If they are the same point, the value is simply the area of the triangle delimited by e, e' and the vertical line $x = \max X(e, e')$. Otherwise, assume that the x-coordinate of the leftmost endpoint of e lies to the left of that of e' (the other case is symmetric), and let v be the left endpoint of e'. Then $Q(e, e') = \mathit{area}(T(e, e')) + Q(e, e'')$, where e'' is an edge incident to $v = (v_x, v_y)$ with its other endpoint to the left of v_x such that $\measuredangle(e'', e') \geq 0$ and $Q(e, e'')$ is maximized over all possible edges e''.

However, computing the values in this way we may spend up to $O(n^3)$ time, since for each promising pair there may be up to a linear number of candidate edges e''. We can avoid this by making the following observation.

We consider three sets of edges E_A, E_R and E_L, which are defined for a fixed v. E_A contains those edges that intersect ℓ above v (with no intervening exterior points), E_R contains all edges incident to and to the right of v, and E_L contains all edges incident to and to the left of v, see Figure 5(a). For each combination of an edge $e \in E_A$ and an edge $e' \in E_R$ that form a promising pair, we must find the edge $e'' \in E_L$ that optimizes $Q(e, e'')$. We refer to this edge e'' as $\mathsf{best}(e, e')$.

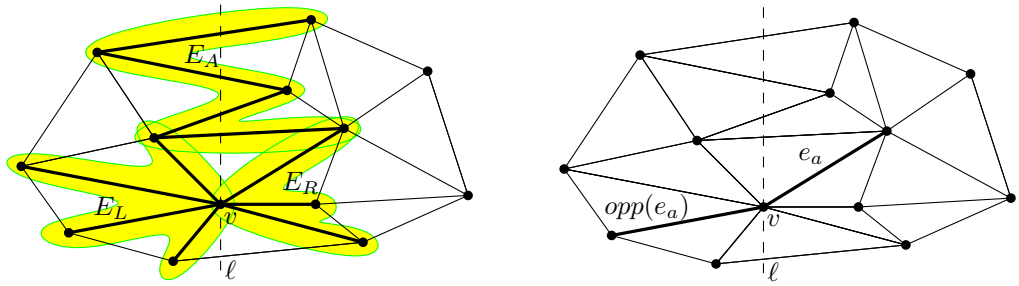
Given an edge $e_a \in E_R$, we define the *opposite* edge of e_a, $\mathsf{opp}(e_a)$, as the edge in E_L with smallest non-negative turning angle $\measuredangle(\mathsf{opp}(e_a), e_a) \geq 0$ (that is, it is convex), see Figure 5(b). Note that $\mathsf{best}(e, e')$ is always equal to or is above $\mathsf{opp}(e')$ (that is, it has a higher left endpoint). We make the following observation.

Observation 1 *The function* $\mathsf{best}(e, e')$*, when defined, is* monotone *in the sense that if* $e_a, e_b \in E_R$ *are such that if* e_a *is above* e_b*, then* $\mathsf{best}(e, e_b)$ *is equal to or above* $\mathsf{best}(e, e_a)$*.*

We now describe how to organize the computation to use this observation, and make the whole algorithm run in quadratic time.

Sweep algorithm We sweep the points from left to right, stopping at each vertex of $\mathcal{M}$. Let the current vertex be $v = (v_x, v_y)$. At this point, we will at once compute the values of $Q(e, e')$ for all promising pairs with $v_x = \min X(e, e')$, in time proportional to the degree of v. Recall that for each pair, e lies above e'. First we only consider pairs for which e intersects ℓ above v, and $e' \in E_R$. We separately handle the pairs where e is incident to v and e' intersects ℓ below v, in a symmetric way.

We compute for each $e \in E_A$ a list $L(e)$ of the edges in E_L sorted by angle (we can assume the edges are stored in sorted order around the vertices). The first entry of the list corresponds to the topmost edge (smallest slope). We go through this list and store at each entry the maximum of the largest value of $Q(e, e'')$ seen so far and the value for the current edge (storing also appropriate



(a) (b)

Figure 5: (a) The three sets of edges of interest when processing vertex v. (b) An edge $e_a \in E_R$ and its *opposite* edge, $\mathsf{opp}(e_a)$.

pointers to allow reconstructing the solution later). Since the values of $Q(e, e'')$ that are needed have been already computed, we can fill in $L(e)$, for all $e \in E_A$, in $O(|E_A| \cdot |E_L|)$ time.

The second step consists in using this list to compute $Q(e, e')$ for all promising pairs of edges $e \in E_A$ and $e' \in E_R$. In linear time, we can preprocess the mesh $\mathcal{M}$ to compute and store the opposite edge of each edge. Now, for each promising pair (e, e'), we look up $\mathsf{opp}(e')$ in the list $L(e)$. The edge $e'' = \mathsf{best}(e, e')$ we are looking for is, by Observation 1, the one associated with the Q-value stored with $\mathsf{opp}(e')$. (Note that $\mathsf{best}(e, e')$ may be different from $\mathsf{opp}(e')$.) Therefore we set $Q(e, e') = Q(e, e'') + area(T(e, e'))$.

All the Q-values can be computed in quadratic time in total: for a vertex v, of degree $deg(v)$, $O(deg(v) \cdot n)$ time is spent on filling the list of all edges crossing ℓ above v, which leads to quadratic time for all vertices. In addition, constant time is spent to fill in each entry of $Q(e, e')$. The total running time is hence quadratic. Together with the algorithm for computing the promising pairs, we conclude with the following theorem:

Theorem 1 *Given a polygon with holes $\mathcal{P}$, and a triangular mesh $\mathcal{M}$ with n vertices that covers the interior of $\mathcal{P}$, a maximum-area convex subpolygon that is the union of triangles of $\mathcal{M}$ can be computed in $O(n^2)$ time.*

2.3 Monotone polygons

Turning to problem (ii), we note that the algorithm given above can easily be adapted to compute the largest-area polygon that is monotone in a given direction. Assume without loss of generality that the given direction is horizontal. Then computing $Q(e, e')$ remains the same for edges with shared starting or ending endpoints. When "continuing through" from edge e' to e'', however, we need to remove the condition that the slope of e'' must be less than that of e'. When the direction of monotonicity is not given, we can check all $O(n)$ directions perpendicular to the edges of $\mathcal{M}$. To summarize, we obtain:

Theorem 2 *Given a polygon with holes $\mathcal{P}$, and a triangular mesh $\mathcal{M}$ with n vertices that covers the interior of $\mathcal{P}$, a maximum-area monotone subpolygon that is the union of triangles of $\mathcal{M}$ can be computed in $O(n^2)$ time if the direction is given, or in $O(n^3)$ time otherwise.*

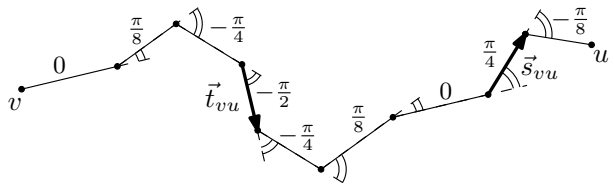


Figure 6: A path from v to u showing the turning angle of each edge, with respect to the first edge of the path. The extreme edges are the ones with the maximum and minimum turning angle.

3 Maximum negative turning angle

Next we study the problem of finding the collection of triangles of largest total area such that their union R is a simple polygon and the maximum backturn Φ of the boundary of R is bounded by some constant $0 \leq \gamma \leq \pi$. The backturn is defined as $\Phi = \min_{i,j} \alpha(\vec{e}_i, \vec{e}_j)$; see Section 1 for more details. The goal is to find a largest area subregion that does not include any part of the exterior and has $\Phi \geq -\gamma$.

For a given path from vertex v to vertex u, we define the *extreme edges* $\vec{s}_{vu}$ and $\vec{t}_{vu}$ as follows: $\vec{s}_{vu}$ is the edge of the path with the largest turning angle with respect to the first edge of the path, whereas $\vec{t}_{vu}$ is the edge with the smallest one, see Figure 6. Extreme edges will play an important role in the algorithm. We make the following simple observation, illustrated in Figure 7.

Observation 2 *Let $\{\vec{e}_1, \cdots, \vec{e}_m\}$ be a path from u to v, and let $\vec{s}_{vu}$ and $\vec{t}_{vu}$ be the extreme edges of the path. Then $\alpha(\vec{s}_{vu}, \vec{e}_m) \leq 0$ and $\alpha(\vec{e}_1, \vec{t}_{vu}) \leq 0$.*

We present a dynamic programming algorithm that computes a function Q which assigns to each subproblem the value of an optimal solution of the subproblem. A subproblem is defined on a pair of vertices, a pair of edges that are incident to those vertices, and two angle bounds for the extreme edges. Let $(v, u, \vec{e}_{v,\text{out}}, \vec{e}_{u,\text{in}}, \sigma, \tau)$ be such an instance; Figure 8 shows a visual representation.

The goal is to find a path from v to u that uses $\vec{e}_{v,\text{out}}$ as the first edge and $\vec{e}_{u,\text{in}}$ as the last edge, and such that the turning angle of the extreme edges $\vec{s}_{vu}$ and $\vec{t}_{vu}$ of the path are bounded by σ and τ, respectively. More precisely, we require $\alpha(\vec{e}_{v,\text{out}}, \vec{t}_{vu}) \geq -\tau$ and $\alpha(\vec{s}_{vu}, \vec{e}_{u,\text{in}}) \geq -\sigma$ (for $\sigma, \tau \geq 0$). In other words, we are limiting the turning angle from the first edge to the edge with smallest angle, and from the edge with largest angle to the last edge. Since $\vec{s}_{vu}$ and $\vec{t}_{vu}$ are *extreme* edges, this also limits the angles of all the other edges on the path from v to u. With some abuse of notation, we sometimes consider σ and τ as directions, and sometimes as angles. Note that σ and τ restrict the extreme directions of edges of our solution polygon R, so the only values they take are the linear number of directions of the edges of $\mathcal{M}$.

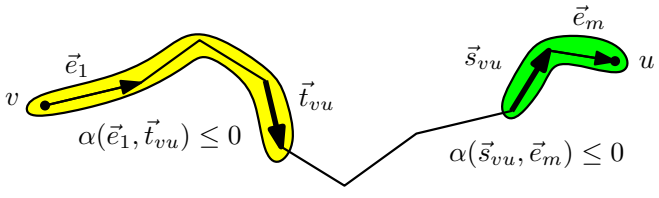


Figure 7: Example illustrating Observation 2. The total turning angle of each of the highlighted subpaths, $\alpha(\vec{e}_1, \vec{t}_{vu})$ and $\alpha(\vec{s}_{vu}, \vec{e}_m)$, is always less than or equal to zero. Note that the result also holds when $\vec{s}_{vu}$ appears before $\vec{t}_{vu}$.

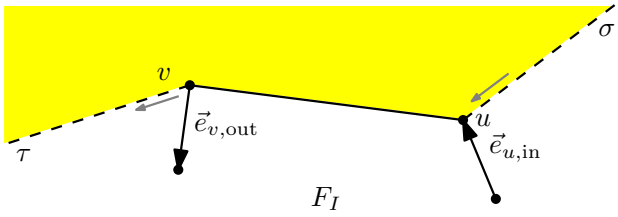


Figure 8: Schematic view of an instance $(v, u, \vec{e}_{v,\text{out}}, \vec{e}_{u,\text{in}}, \sigma, \tau)$ of the algorithm. Vertices u and v, together with directions τ and σ, separate the plane into two areas. The feasible region F_I, in white, contains the solution to the instance. The gray arrows indicate the orientation of σ and τ.

Moreover, we require the polygon consisting of this path and completed by $u\vec{v}$ to be simple, to contain no parts of the exterior, and to have optimal area. Note that $u\vec{v}$ is the only edge of this polygon that is not necessarily an edge of the triangulation.

3.1 Description of the algorithm

The algorithm is based on the fact that the optimal polygon R^* has a triangulation (not to be confused with the given mesh $\mathcal{M}$ of the input polygon), and we can use it to recursively define the solution to an instance in terms of the solutions to smaller instances. In particular, the path from v to u in the instance $I = (v, u, \vec{e}_{v,\text{out}}, \vec{e}_{u,\text{in}}, \sigma, \tau)$ is composed of a path from v to some vertex w and a path from w to the vertex u, and such that $v\vec{w}$ and $w\vec{u}$ are diagonals of a triangulation of the solution to the instance I (or, in some cases, edges of the solution itself), see Figure 9. We can maximize over all vertices w, the edges incident to w that are used on the paths, and the way the bounds on the angles of the extreme edges may occur in the subproblems, as long as the solutions do not contain part of the exterior and the angle bounds are respected.

Consider an optimal solution R^* and a triangulation of R^*. Let $u\vec{v}$ be a diagonal of this triangulation. We define Q as follows:

$$\begin{aligned} Q(v, u, \vec{e}_{v,\text{out}}, \vec{e}_{u,\text{in}}, \sigma, \tau) = \max_{w, \vec{e}_{w,\text{out}}, \vec{e}_{w,\text{in}}, \sigma', \tau'} (&Q(v, w, \vec{e}_{v,\text{out}}, \vec{e}_{w,\text{in}}, \sigma', \tau) \\ &+ Q(w, u, \vec{e}_{w,\text{out}}, \vec{e}_{u,\text{in}}, \sigma, \tau') \\ &+ \text{area}(\triangle uvw)) \end{aligned}$$

as long as the following conditions hold:

(1) $\alpha(\vec{e}_{v,\text{out}}, \vec{e}_{w,\text{in}}) \geq -\tau$ and $\alpha(\vec{e}_{w,\text{out}}, \vec{e}_{u,\text{in}}) \geq -\sigma$.

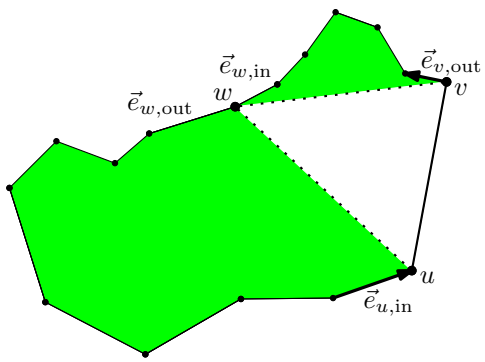


Figure 9: Example showing the decomposition of the problem into two smaller subproblems.

(2) $\sigma' + \tau' - \measuredangle(\vec{e}_{w,\text{in}}, \vec{e}_{w,\text{out}}) \leq \gamma$.

(3) $\tau' \leq \tau + \alpha(\vec{e}_{v,\text{out}}, \vec{e}_{w,\text{out}})$ and $\sigma' \leq \sigma + \alpha(\vec{e}_{w,\text{in}}, \vec{e}_{u,\text{in}})$.

(4) $\triangle uvw$ is contained in $\mathcal{P}$, and w lies to the left of $\vec{uv}$.

The first condition ensures that $\vec{e}_{w,\text{in}}$ and $\vec{e}_{w,\text{out}}$ are within the bounds imposed by τ and σ. The next two conditions guarantee that $\alpha(\vec{s}_{vu}, \vec{t}_{vu}) \geq -\gamma$: if both $\vec{s}_{vu}$ and $\vec{t}_{vu}$ lie in the same subproblem, then the recursion ensures their turning angle is within the right bound. Otherwise, if they lie in different subproblems, then the second condition relates the two angle bounds of the subproblems and the angle at w with γ, whereas the third condition enforces that the angle bounds of the subproblems do not exceed the existing ones. Together they guarantee that $\alpha(\vec{s}_{vu}, \vec{t}_{vu}) \geq -\gamma$. Last but not least, the fourth condition makes sure that the final solution will not contain parts of the exterior.

If no vertex w exists that satisfies these conditions, then $Q(\cdot)$ is defined to be $-\infty$. The base case is $Q(v, u, \vec{vu}, \vec{vu}, \sigma, \tau)$ and has value zero for all combinations of v, u, σ and τ (of course, $\vec{vu}$ must be an edge of $\mathcal{M}$).

Now we are ready to give a description of the algorithm. In Section 3.2 we argue that the algorithm indeed returns a solution to our problem, and in Section 3.3 we analyze the running time. We maintain a 6-dimensional table with one entry for each instance $I = (v, u, \vec{e}_{v,\text{out}}, \vec{e}_{u,\text{in}}, \sigma, \tau)$. In each cell of the table, we store the value $Q(I)$, as well as pointers to the entries of the two subproblems that give this value (unless I is a base case). For each entry, we compute the value by trying all possibilities for w, $\vec{e}_{w,\text{in}}$, and $\vec{e}_{w,\text{out}}$, looking up their solutions in the table and checking whether Conditions (1) to (4) hold. We start by filling in all base case entries of the table. If we fill the remaining entries in any order that complies with the partial order induced by the inclusion of feasible regions, then we are guaranteed that the subproblems we need to look up have already been solved before (this is proven in the next section).

3.2 Correctness of the algorithm

In this section, we show that the solution returned by the algorithm is an optimal solution to our problem, and that the algorithm always terminates. The following lemma shows that the solutions computed by the dynamic programming algorithm respect the angle constraint.

Lemma 1 *Let $I = (v, u, \vec{e}_{v,\text{out}}, \vec{e}_{u,\text{in}}, \sigma, \tau)$ be an instance of the dynamic programming algorithm above, such that $Q(I) > -\infty$ and $\sigma + \tau - \alpha(\vec{e}_{u,\text{in}}, \vec{e}_{v,\text{out}}) \leq \gamma$. Let R be the polygon defined by $\vec{uv}$ and the path from v to u associated with $Q(I)$. Then $\Phi_R \geq -\gamma$.*

Proof: Let R be the polygon whose boundary is the concatenation of $\vec{uv}$ and ρ_{vu}, where ρ_{vu} is the path associated with $Q(I)$. Assume I does not fall into the base case (otherwise the result holds immediately). Since it is the result of applying the dynamic programming algorithm, there must be a vertex w such that ρ_{vu} is the concatenation of ρ_{vw} and ρ_{wu}, where ρ_{vw} and ρ_{wu} are the paths which resulted from evaluating $Q(v, w, \vec{e}_{v,\text{out}}, \vec{e}_{w,\text{in}}, \sigma', \tau)$ and $Q(w, u, \vec{e}_{w,\text{out}}, \vec{e}_{u,\text{in}}, \sigma, \tau')$, for some combination of $\vec{e}_{w,\text{in}}$, $\vec{e}_{w,\text{out}}$, σ' and τ'.

Let $\vec{e}_k, \vec{e}_l$ be a pair of edges with the maximum backturn in R. We will show that $\alpha(\vec{e}_k, \vec{e}_l) \geq -\gamma$. We distinguish between several cases depending on where $\vec{e}_k$ and $\vec{e}_l$ lie.

- Both $\vec{e}_k$ and $\vec{e}_l$ lie in ρ_{vw} or both lie in ρ_{wu}. Then both edges appear in the same subproblem, and it follows by induction that $\alpha(\vec{e}_k, \vec{e}_l) \geq -\gamma$.

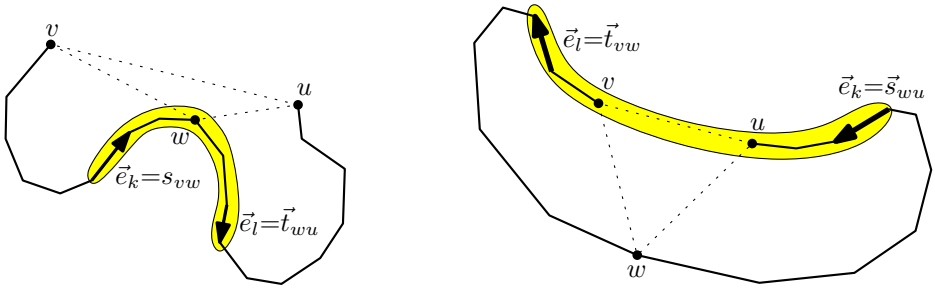


Figure 10: Illustrations for proof of Lemma 1. The path associated with $\alpha(\vec{e}_k, \vec{e}_l)$ is highlighted. (a) Case $\vec{e}_k \in \rho_{vw}$ and $\vec{e}_l \in \rho_{wu}$. (b) Case $\vec{e}_k \in \rho_{wu}$ and $\vec{e}_l \in \rho_{uw}$.

- $\vec{e}_k \in \rho_{vw}$ and $\vec{e}_l \in \rho_{wu}$. Then w lies between $\vec{e}_k$ and $\vec{e}_l$ (see Figure 10(a)). This implies that $\vec{s}_{vw} = \vec{e}_k$ and $\vec{t}_{wu} = \vec{e}_l$. To see why, assume that $\vec{t}_{wu} \neq \vec{e}_l$. By definition of $\vec{t}_{wu}$, $\vec{t}_{wu}$ achieves the minimum turning angle between $\vec{e}_{w,\text{out}}$ and any edge in ρ_{wu}. Therefore $e_l \neq \vec{t}_{wu}$ immediately implies $\alpha(\vec{e}_k, \vec{t}_{wu}) < \alpha(\vec{e}_k, \vec{e}_l)$, leading to a contradiction. A similar argument shows that $\vec{s}_{vw} = \vec{e}_k$.

  The turning angle $\alpha(\vec{e}_k, \vec{e}_l)$ can be decomposed into $\alpha(\vec{e}_k, \vec{e}_l) = \alpha(\vec{s}_{vw}, \vec{t}_{wu}) = \alpha(\vec{s}_{vw}, \vec{e}_{w,\text{in}}) + \measuredangle(\vec{e}_{w,\text{in}}, \vec{e}_{w,\text{out}}) + \alpha(\vec{e}_{w,\text{out}}, \vec{t}_{wu})$. From the definition of the angle bounds σ' and τ' we have $\alpha(\vec{s}_{vw}, \vec{e}_{w,\text{in}}) \geq -\sigma'$ and $\alpha(\vec{e}_{w,\text{out}}, \vec{t}_{wu}) \geq -\tau'$. In combination with condition (2) they yield $\alpha(\vec{s}_{vw}, \vec{t}_{wu}) \geq -\gamma$.

- $\vec{e}_k \in \rho_{wu}$ and $\vec{e}_l \in \rho_{vw}$. Similar to the previous case, it is easy to verify that $s_{wu} = \vec{e}_k$ and $t_{vw} = \vec{e}_l$ (see Figure 10(b)). The path from $\vec{s}_{vw}$ to $\vec{t}_{vw}$, with total angular change $\alpha(\vec{e}_k, \vec{e}_l)$, can be split into three parts: $\vec{s}_{wu}$ to $\vec{e}_{u,\text{in}}$, $\vec{e}_{u,\text{in}}$ to $\vec{e}_{v,\text{out}}$, and $\vec{e}_{v,\text{out}}$ to $\vec{t}_{vw}$.

  From the definition of the angle bounds σ and τ we have that $\alpha(\vec{s}_{wu}, \vec{e}_{u,\text{in}}) \geq -\sigma$ and $\alpha(\vec{e}_{v,\text{out}}, \vec{t}_{vw}) \geq -\tau$. Combining these with the lemma hypothesis $\sigma + \tau - \alpha(\vec{e}_{u,\text{in}}, \vec{e}_{v,\text{out}}) \leq \gamma$, we get $\alpha(\vec{s}_{wu}, \vec{t}_{vw}) = \alpha(\vec{e}_k, \vec{e}_l) \geq -\gamma$.

- $\vec{e}_k = u\vec{v}$ and $\vec{e}_l \in \rho_{vu}$ (or vice versa). The lemma hypothesis $\sigma + \tau - \alpha(\vec{e}_{u,\text{in}}, \vec{e}_{v,\text{out}}) \leq \gamma$ ensures that angles between $u\vec{v}$ and $\vec{e}_l$ that respect σ and τ will result in a total angular change $\geq -\gamma$, therefore $\alpha(\vec{e}_k, \vec{e}_l) \geq -\gamma$ follows.

Therefore in all cases $\alpha(\vec{e}_k, \vec{e}_l) = \Phi_R \geq -\gamma$, and the result follows. ⊠

The previous result implies that the polygon induced by the solution of a subproblem must be simple (that is, without self-intersections).

Corollary 1 *The polygon R, as defined in Lemma 1, is simple.*

Proof: Suppose that the solution to I uses vertex w and the solutions to both subproblems of I are simple. Since $\vec{e}_{w,\text{in}}$ comes before $\vec{e}_{w,\text{out}}$, and since the total backturn is not more than π, the solutions do not intersect except at w, and therefore the solution to I is also simple. ⊠

We further observe that the definition of the recursive algorithm, together with conditions (1) to (4), imply that the final solution does not contain any part of the exterior (that is, it is valid), and that its area is maximum.

It remains to show that the subproblems needed to solve a given instance I of the recursive algorithm are in some sense *smaller* than I, and therefore can be assumed to have been solved in

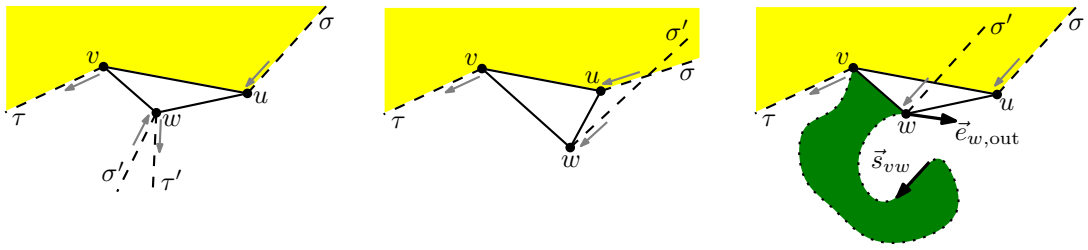


Figure 11: (a) The feasible regions of the two subproblems are both contained in the feasible region of the original problem. (b-c) The two situations in which the left subproblem has a feasible region that is not contained in the original problem. Case (b) violates Condition (3) (σ' is restricted by σ). Case (c) has a turning angle between $\vec{s}_{vw}$ and $\vec{e}_{w,\text{out}}$ that is smaller than $-\pi$.

earlier steps. This can be seen in the following way. For any subproblem $I = (v, u, \vec{e}_{v,\text{out}}, \vec{e}_{u,\text{in}}, \sigma, \tau)$, we can define a *feasible region* F_I that must contain any possible solution to that subproblem. Consider the oriented *Zorro* consisting of the halfline to u in the direction of σ, the line segment $\vec{uv}$, and the halfline from v in the direction of τ (see Figure 8). The feasible region F_I is the region to the left of this Zorro.

Lemma 2 *In the recursive step for subproblem $I = (v, u, \vec{e}_{v,\text{out}}, \vec{e}_{u,\text{in}}, \sigma, \tau)$, the feasible regions of the subproblems are contained in the feasible region of F_I. Furthermore, the number of vertices of the triangulated polygon inside the region strictly decreases.*

Proof: To prove the first part of the lemma, assume for contradiction that there is a subproblem J whose feasible region F_J is not contained in F_I. Further assume without loss of generality that this happens with subproblem $J = (v, w, \vec{e}_{v,\text{out}}, \vec{e}_{w,\text{in}}, \sigma', \tau)$. Clearly, the halflines from v in the Zorros of F_I and F_J are the same, and w must lie inside the feasible region F_I so the middle segment of the Zorro of F_J, $\vec{vw}$, is always inside F_I. If F_J is not contained in F_I, this means that the halfline of F_J from w intersects the Zorro of F_I.

There are essentially two ways in which F_J can intersect F_I, shown in Figure 11. Since the direction σ' is restricted by σ (by Condition (3)), the halfline from w in direction σ' cannot intersect the halfline from u in direction σ without also intersecting the middle segment of the Zorro of F_I, $\vec{vu}$. Thus the case shown in Figure 11(b) cannot arise.

To see that other possibility, shown in Figure 11(c), cannot happen as part of a valid problem, assume that there is an edge on the path from v to w that achieves a direction of σ'. If that is not the case, then we can discard this particular subproblem, since there will be another one with the same solution and smaller σ' (that is, more clockwise). Therefore we can assume that σ' is attained by some edge, in particular, by $\vec{s}_{vw}$. Then $\alpha(\vec{s}_{vw}, \vec{e}_{w,\text{out}}) \leq \alpha(\vec{s}_{vw}, \vec{wu}) < -\pi$, violating the angle constraint of $\Phi \geq -\gamma \geq -\pi$. It follows that such a solution does not fulfill Conditions (1) to (4) for I, thus it will never be considered by the algorithm.

To prove the second part of the lemma, we note that from the same arguments it follows that v cannot be in the interior of the feasible region of $(w, u, \vec{e}_{w,\text{out}}, \vec{e}_{u,\text{in}}, \sigma, \tau')$, and u cannot be in the feasible region of $(v, w, \vec{e}_{v,\text{out}}, \vec{e}_{w,\text{in}}, \sigma', \tau)$. Therefore the number of vertices in the subproblems decreases. ⊠

Because of Lemma 2, the feasible regions define a partial order on the subproblems in which we can evaluate them with the guarantee that all information to solve the current subproblem is available. Since the total number of subproblems is finite (we only need to check finitely many directions for τ and σ since they must be aligned with the point set), the algorithm terminates.

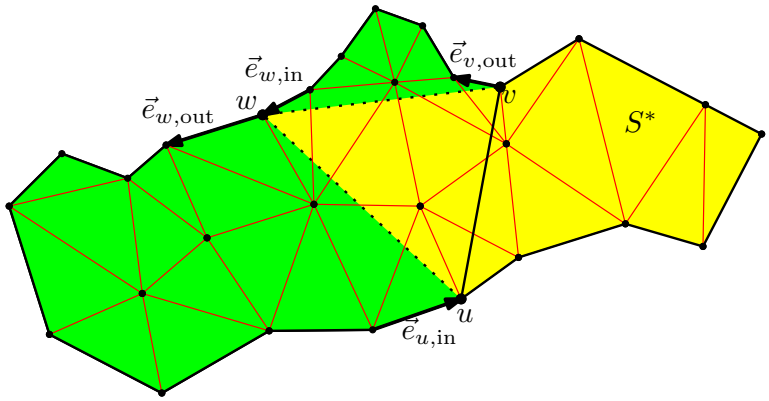


Figure 12: Example showing the decomposition of the problem into two smaller subproblems (darker gray).

3.3 Running time analysis

In order to analyze the running time of the algorithm, some more detail on the steps to be performed is needed. Recall that a subproblem is specified by a pair of vertices, a pair of edges that are incident to those vertices, and two angle bounds. Let $(v, u, \vec{e}_{v,\text{out}}, \vec{e}_{u,\text{in}}, \sigma, \tau)$ denote the current problem.

Since $\vec{uv}$ is a diagonal of a triangulation of R^*, there must be a triangle $\triangle uvw$ to the left of $\vec{uv}$. We will try all the possible candidates for w. Each triangle $\triangle uvw$ subdivides the problem into two new subproblems (see Figure 12).

Any pair of subproblems that needs to be combined must fulfill conditions (1) to (4). This can be easily checked in constant time for the first three conditions. In order to perform the check for the fourth condition in constant time, we need to do some preprocessing first. We can trivially check and store this for any triple of vertices in $O(n^4)$ time and $O(n^3)$ space. If one of the conditions does not hold, we assign the value $-\infty$ to this instance.

When maximizing over the values of σ' and τ', we only need to take the combinations into account where $\sigma' + \tau' - \measuredangle(\vec{e}_{w,\text{in}}, \vec{e}_{w,\text{out}})$ is the largest possible value that is still smaller or equal to γ. As observed earlier, σ' and τ' can take on a linear number of values. So, for each value of σ' there is only one value of τ' to consider, so there are only a linear number of combinations possible for choosing σ' and τ' together. Note that the two subproblems are independent as long as $\gamma \leq \pi$. So, together with the choice of w, each instance is solved by taking the maximum over $O(n^2)$ combinations of smaller instances.

As for the overall running time of the recursive algorithm, the base case can be resolved in constant time. For the general case we have a table with $O(n^4)$ entries: there are a linear number of possibilities for the pair $(v, \vec{e}_{v,\text{out}})$, a linear number for the pair $(u, \vec{e}_{u,\text{in}})$, a linear number for σ, and a linear number for τ. We can fill in an entry in quadratic time, so we need $O(n^6)$ time in total.

Theorem 3 *Given a constant $0 \leq \gamma \leq \pi$, a polygon with holes $\mathcal{P}$, and a triangular mesh $\mathcal{M}$ with n vertices that covers the interior of $\mathcal{P}$, a maximum-area polygon that is the union of triangles of $\mathcal{M}$ such that the maximum negative turning angle (or backturn) Φ is $\Phi \geq -\gamma$ can be computed in $O(n^6)$ time.*

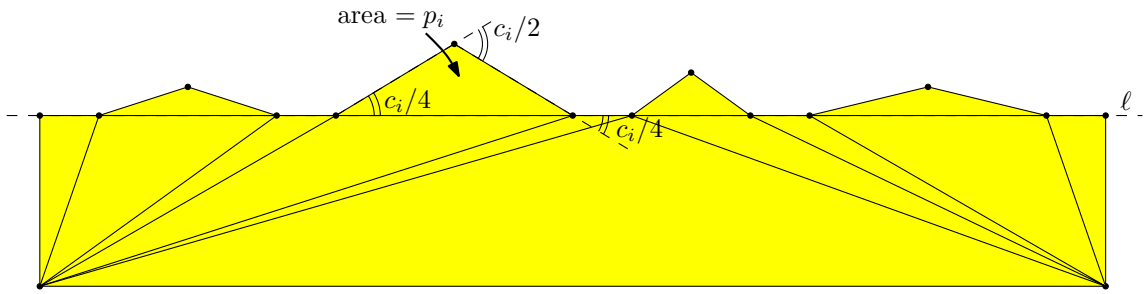


Figure 13: Reduction of knapsack to maximum area subtriangulation with bounded total angular change.

4 The largest-area polygon with bounded turning angle

We now consider the problem of finding the largest-area simple polygon formed as a union of triangles of $\mathcal{M}$, so that the total angular change of the perimeter is at most some constant β. We show that this problem is NP-hard for a total angular change of, say, 3π. The reduction is from the NP-hard problem KNAPSACK [13]: Given a set of n integer pairs $(c_1, p_1), \ldots, (c_n, p_n)$, where c_i is the size of the i^{th} item and p_i is its profit, and a maximum capacity C, select the subset of maximum total profit of which the total size is at most C. Without loss of generality, we scale the sizes $c_1, \ldots, c_n$ and C so that $C = \pi$.

We construct a triangulated polygon as follows, see Figure 13. Take a pair (c_i, p_i). If $c_i > C = \pi$ then we discard it. Otherwise we construct an isosceles triangle T_i with a horizontal base of width $2/\tan(c_i/4)$, and its top vertex m_i at height 1 above the middle of the base. Scale T_i so that its area is $2p_i$. The total angular change of a path that arrives horizontally from the left at the lower left corner of T_i, goes diagonally up to m_i, and then diagonally down to the lower right corner of T_i, and then continues horizontally to the right, is c_i, and the area in T_i and below the path is p_i. This corresponds to choosing the i^{th} item in the knapsack: the profit (area) is p_i and the cost (extra angular change) is c_i. Now we combine all these triangles so that all lower sides lie on a horizontal line ℓ, spaced sufficiently far apart; their order is irrelevant. Below the line ℓ we make large triangles whose union is a large rectangle R with the following properties:

- The top side of R is on ℓ.
- The upper left corner of R is strictly to the left of all the triangles T_i.
- The upper right corner of R is strictly to the right of all the triangles T_i.
- Every triangle in the polygon triangulation of R has area greater than $\sum_{i=1}^{n} p_i$ (note that R has two vertices of every triangle T_i on its top side).

The first three conditions can be trivially satisfied, whereas the last one can be easily fulfilled by making R very high. By construction, a maximum area solution will contain at least all triangles inside R, because leaving one out cannot be compensated even by using all the T_i's. The total angular change of R is exactly 2π, and we can choose any remaining triangle T_i, at the cost of c_i, which gives an extra area p_i. Hence, the subset of items with maximum profit in the knapsack with total size at most π corresponds precisely to the subset of extra triangles.

Although we assumed that the weights of the knapsack items can be scaled to add up to π and we compute coordinates of the form $2/\tan(c_i/4)$, it is not essential that these are computed exactly. Knapsack is already NP-hard for items with integer values, so if we use these we can afford to make small errors in the placements of the points, as long as they do not add up to something more than 1. Instead of scaling the weights to add up to π, we make them add up to a rational number slightly smaller than π. Still we cannot place the points exactly, but if we make sure the

errors are not more than $O(1/n)$ we will not influence the solutions of the knapsack problem. This can increase the description complexity of the points by at most a factor of $O(n)$.

Theorem 4 *Given a polygon with holes $\mathcal{P}$, and a triangular mesh $\mathcal{M}$ that covers the interior of $\mathcal{P}$, computing a maximum area subpolygon, comprised of triangles of $\mathcal{M}$, with total angular change at most 3π is NP-hard.*

5 Conclusions

We studied various area optimization problems on triangulated polygons with holes, where the boundary of the optimal simple polygon is restricted to the triangle boundaries. We focused on convexity as a way to form well-shaped areas, and given that convexity is a strong restriction in combination with forcing the polygon to be composed only of whole triangles, we also studied three different relaxations of convexity. For three versions we gave polynomial algorithms and for one, an NP-hardness proof. It is worth to note that our *meshed* version of the potato peeling problem, by restricting the convex subpolygon to be composed of whole triangles, allows for a much faster algorithm than the unrestricted version: $O(n^2)$ time against the $O(n^8)$ time needed for the original potato peeling problem.

This paper opens up several directions for further research. On the one hand we would like to improve the algorithm of Section 3. The current algorithm is rather involved and has a high running time. It would be interesting to know if there is a different way to approach it that results in a simpler or faster algorithm. On the other hand, there is the possibility of studying other relaxations of convexity or different properties that result in well-shaped subregions. Moreover, several other extensions are possible, such as studying the problem in three dimensions. That is, given a tetrahedralization of a solid, possibly with holes, find a subset of the tetrahedra that forms a convex (or nearly convex) solid. This would have applications, for example, in finding large occluders for hidden surface removal algorithms [8]. The algorithms presented here clearly serve as building blocks for the 3D version, although a deeper study of the problem is needed.

References

[1] A. Aggarwal, D. Coppersmith, S. Khanna, R. Motwani, and B. Schieber. The angular-metric traveling salesman problem. *SIAM J. Comput.*, 29:697–711, 1999.

[2] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(3):209–216, 1991.

[3] E. M. Arkin, Y.-J. Chiang, M. Held, J. S. B. Mitchell, V. Sacristán, S. S. Skiena, and T.-C. Yang. On minimum-area hulls. *Algorithmica*, 21:119–136, 1998.

[4] G. Barequet and V. Rogol. Maximizing the area of an axially symmetric polygon inscribed in a simple polygon. *Computers & Graphics*, 31(1):127–136, 2007.

[5] R. Boland and J. Urrutia. Finding the largest axis aligned rectangle in a polygon in $O(n \log n)$ time. In *Proc. 13th Canadian Conference on Computational Geometry (CCCG'01)*, pages 41–44, 2001.

[6] J. S. Chang and C. K. Yap. A polynomial solution for the potato-peeling problem. *Discrete Comput. Geom.*, 1:155–182, 1986.

[7] B. Chazelle. The polygon containment problem. In F. P. Preparata, editor, *Computational Geometry*, volume 1 of *Adv. Comput. Res.*, pages 1–33. JAI Press, Greenwich, Conn., 1983.

[8] D. Cohen-Or, S. Lev-Yehudi, A. Karol, and A. Tal. Inner-cover of non-convex shapes. *International Journal of Shape Modeling*, 9:223–238, 2003.

[9] S. R. Coorg and S. J. Teller. Real-time occlusion culling for models with large occluders. In *Symposium on Interactive 3D Graphics*, pages 83–90, 189, 1997.

[10] K. Daniels, V. Milenkovic, and D. Roth. Finding the maximum area axis-parallel rectangle in a polygon. *Comput. Geom. Theory Appl.*, 7:125–148, 1997.

[11] S. P. Fekete and G. J. Woeginger. Angle-restricted tours in the plane. *Comput. Geom. Theory Appl.*, 8(4):195–218, 1997.

[12] D. C. G. Papaioannou, A. Gaitatzes. Efficient occlusion culling using solid occluders. In *Proc. 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2006.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

[14] J. E. Goodman. On the largest convex polygon contained in a non-convex n-gon or how to peel a potato. *Geom. Dedicata*, 11:99–106, 1981.

[15] O. Hall-Holt, M. J. Katz, P. Kumar, J. S. B. Mitchell, and A. Sityon. Finding large sticks and potatoes in polygons. In *SODA '06: Proc. 17th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 474–483, 2006.

[16] W. Niblack and J. Yin. A pseudo-distance measure for 2D shapes based on turning angle. In *Proc. International Conference on Image Processing (ICIP)*, pages 352–355, 1995.

[17] I. Reinbacher, M. Benkert, M. van Kreveld, J. S. B. Mitchell, J. Snoeyink, and A. Wolff. Delineating boundaries for imprecise regions. *Algorithmica*, 50(3):386–414, 2008.

[18] M. Sharir and S. Toledo. Extremal polygon containment problems. *Comput. Geom. Theory Appl.*, 4:99–118, 1994.

[19] T. C. Woo. The convex skull problem. Technical Report TR 86-31, Department of Industrial and Operations Engineering, University of Michigan, 1986.

[20] D. Wood and C. K. Yap. The orthogonal convex skull problem. *Discrete Comput. Geom.*, 3(4):349–365, 1988.