Finding Large Sticks and Potatoes in Polygons

Olaf Hall-Holt* Matthew J. Katz† Piyush Kumar‡ Joseph S. B. Mitchell§
Arik Sityon¶

Abstract

We study a class of optimization problems in polygons that seek to compute the "largest" subset of a prescribed type, e.g., a longest line segment ("stick") or a maximum-area triangle or convex body ("potato"). Exact polynomial-time algorithms are known for some of these problems, but their time bounds are high (e.g., $O(n^7)$ for the largest convex polygon in a *simple* n-gon). We devise efficient approximation algorithms for these problems. In particular, we give near-linear time algorithms for a $(1-\epsilon)$-approximation of the biggest stick, an $O(1)$-approximation of the maximum-area convex body, and a $(1-\epsilon)$-approximation of the maximum-area fat triangle or rectangle. In addition, we give efficient methods for computing large ellipses inside a polygon (whose vertices are a dense sampling of a closed smooth curve). Our algorithms include both deterministic and randomized methods, one of which has been implemented (for computing large area ellipses in a well sampled closed smooth curve).

1 Introduction

In many applications one wants to approximate a shape by some other shape having special structure. For a given input shape P, we are often interested in approximations by a convex body. The tightest fitting *outer* approximation of P is its convex hull, $CH(P)$. For an *inner* approximation of P, a natural choice is to use a largest (e.g., maximum-volume) convex body, Q^*, that lies within P. The problem of computing a largest convex body inside a given shape has been called the "potato peelers problem" [11, 22]: How does one optimally "peel" a nonconvex potato P, to obtain a convex potato, while wasting as little as possible of the original potato? A related optimization problem is to find the longest line segment (the biggest "stick") that lies within P. Both of these optimization problems arise naturally in computational geometry and have been studied previously from the point of view of exact algorithms. The problems are also motivated by computer graphics applications, in which one desires good inner approximations of P by simple shapes, for occlusion culling.

In this paper, we study the longest stick and biggest potato optimization problems from the point of view of approximation algorithms. We obtain substantially more efficient algorithms than what was known for exact solutions.

Overview of Results. Our results include:

(1) Approximation algorithms for the biggest stick in a simple polygon: We give a simple $O(n \log n)$ 1/2-approximation, as well as an $O(n \log^2 n)$ polynomial-time approximation scheme (PTAS), which computes an $(1-\epsilon)$-approximation for any fixed $\epsilon > 0$.

(2) Approximation algorithms for maximum-area potatoes in polygons: In particular, we give an $O(n \log n)$ $O(1)$-approximation for computing a largest area triangle or convex polygon inside a simple n-gon. We give an $O(n)$-time $(1-\epsilon)$-approximation algorithm for computing a large *fat* triangle or rectangle in a simple polygon; in a polygon with holes, the algorithm runs in time $O(n \log n)$. We also give randomized methods to compute approximations of maximum-area triangles. Further, we give an approximation algorithm to compute maximum-area ellipses in well-sampled smooth closed curves in time $O(n^2)$; a variant of this algorithm has been implemented. This approximation algorithm produces better approximations as the density of the sampling increases.

Related Work. Optimal inclusion problems have a rich history in computational geometry. One can compute a maximum area (or perimeter) convex k-gon inside a convex n-gon in time $O(kn + n \log n)$ [4]; see also [9, 19]. The special case of maximum-area inscribed equilateral triangles or squares within a simple polygon

*St. Olaf College, Northfield, MN 55057. olaf@stolaf.edu.
†Ben-Gurion University, Beer-Sheva 84105, Israel. matya@cs.bgu.ac.il.
‡Florida State University, Tallahassee, FL 32306-4530. piyush@cs.fsu.edu.
§Stony Brook University, Stony Brook, NY 11794-3600. jsbm@ams.sunysb.edu.
¶Ben-Gurion University, Beer-Sheva 84105, Israel. sityon@cs.bgu.ac.il.

was solved in time $O(n^3)$ [18] and was improved, for similar copies of a convex k-gon, using motion planning methods of Chew and Kedem [14]; see also [21]. Melissaratos and Souvaine [26] compute a maximum-area triangle (of any shape) inside a simple polygon in time $O(n^3)$. The best (and only) polynomial-time algorithm for computing a maximum-area convex polygon inside a simple polygon takes time $O(n^7)$ and space $O(n^5)$ [11].

The longest stick problem in a simple polygon was first solved in subquadratic time ($O(n^{1.99})$) by Chazelle and Sharir [13]. Improved randomized algorithms [1, 3, 2] eventually led to an expected running time of $O(n^{3/2+\epsilon})$ given by Agarwal and Sharir [2].

On the practical side, graphics researchers have implemented algorithms for finding good occluders in visibility preprocessing [25, 7]. Recent work by Daniel Cohen-Or et al. [16] finds the inner cover of non-convex polygons that function as occluders. Packing ellipsoids [8] in three-dimensional shapes has been studied by Bischoff and Kobbelt, who start with a sphere inside the model and then gradually inflate and translate it until it gets pinned. Non-convex occluders, called *hoops*, have been studied by Brunet et al. [10]; they compute polylines that have convex silhouettes when seen from certain view points.

2 The Biggest Stick Problem

In this section, we study the problem of computing an approximation of the "biggest stick" (longest line segment) within a simple polygon P that has n vertices. We consider P to be a closed region (i.e., it includes its boundary) in the plane.

2.1 A Simple 1/2-Approximation

We begin with a standard divide-and-conquer approach. Consider a hierarchical cutting of P using diagonals ("Chazelle cuts"). Specifically, we initially determine a cut ξ of P that gives a (roughly) balanced partition of P into two simple polygons, each with at most $2n/3$ vertices; then, recursively, we continue the partitioning for each of the resulting subpolygons, until all subpolygons are triangles. Such a hierarchical decomposition of a simple polygon can be computed in $O(n)$ time, as a result of the linear-time triangulation algorithm of Chazelle [12].

Consider an optimal solution, s^*, which is a segment of length l^* in P. Then, s^* must intersect at least one diagonal in the hierarchical decomposition; let ξ^* denote the diagonal having the smallest depth (i.e., the first one to intersect s^* in the recursive cutting), and let $P(\xi^*)$ denote the subpolygon that is partitioned by ξ^* into subpolygons $P_1(\xi^*)$ and $P_2(\xi^*)$. The segment s^* is cut into two subsegments by ξ^*, at least one of which is of length at least $l^*/2$; let $s_1^* \subset P_1(\xi^*)$ be a subsegment of length at least $l^*/2$. Then, s_1^* has one endpoint on an edge of $P_1(\xi^*)$ (the edge corresponding to cut ξ^*). Our goal, then, is to compute a longest *anchored* stick, incident on a specified edge, for each face of the hierarchical partitioning, at each of the $O(\log n)$ levels of the hierarchy.

To find the biggest stick anchored on a given edge e of a simple m-gon, we walk around the boundary of the weakly visible polygon of e (which can be computed in time $O(m)$ [24]). This traversal divides the boundary into $O(m)$ segments, each of which has a fixed combinatorial description (i.e., the pair of view-delimiting vertices) of its visibility with respect to e. Let σ be such a segment along the boundary of the weakly visible polygon. The portion of e visible from each point along σ is determined by the pair of view-delimiting vertices, and this pair is the same for all points along σ. The variation of the length of the longest anchored stick from a point on σ to e can be described by an expression with either one or two square roots; thus, the maximum length for σ can be computed in constant time, making the overall time $O(m)$ for the anchored version of the longest stick problem.

Since the sum of the face complexities at any one level of the hierarchy is $O(n)$ (each level corresponds to a planar graph on n vertices), we get that the total time bound for the algorithm is $O(n \log n)$.

2.2 A PTAS for Biggest Stick

Our goal now is to compute, for a given $\epsilon > 0$, an approximation of the biggest stick in P, with length at least $(1-\epsilon)$ times the length, l^*, of the biggest stick, s^*.

We begin by applying the algorithm above for computing a 1/2-approximation: we determine a segment $s_0 \subset P$ with length $l_0 \geq (1/2)l^*$. Notice that we may assume that l_0 is at least the length of the longest edge of P, since otherwise we can set s_0 to be this edge.

A *mega-square* is an axis-parallel square of side length $4l_0$. We say that a mega-square σ is *good* if one can place a stick of length l^* in $\sigma \cap P$. We begin by arguing that P can be covered by a linear number of possibly overlapping mega-squares, such that at least one of them is good.

Consider an infinite regular grid of mega-squares, and make three additional copies of this grid by shifting the bottom left corner of the grid cells by $(2l_0, 0)$, $(0, 2l_0)$, and $(2l_0, 2l_0)$, respectively. Now, mark those grid cells (in the four copies) that are intersected by P's boundary. Let $\mathcal{C}$ denote the set of all marked cells. It is easy to verify that (i) $\mathcal{C}$ consists of $O(n)$ mega-squares that entirely cover P, (ii) each edge of P intersects only a constant number of mega-squares in $\mathcal{C}$, (iii) $\mathcal{C}$ contains

a mega-square that is good, and (iv) $\mathcal{C}$ can be computed in $O(n)$ time.

The combinatorial complexity of a mega-square $\sigma \in \mathcal{C}$ is the number n_c of edges bounding the region $\sigma \cap P$. Since an edge of P intersects only a constant number of mega-squares in $\mathcal{C}$, we have $\sum_{\sigma \in \mathcal{C}} n_\sigma = O(n)$. We now consider each of the mega-squares $\sigma \in \mathcal{C}$ separately.

Let σ be one of the mega-squares in $\mathcal{C}$. We tile σ with a constant number of small squares, *pixels*, whose side length is $\epsilon l_0/c_0$, where c_0 is an appropriate constant. Now for each pair of pixel sides s_1, s_2, we check whether there exists a line segment in $\sigma \cap P$ whose endpoints lie on s_1 and s_2, respectively. Before we describe how this is done, we observe that if σ is a good mega-square, then once the processing of σ is finished, we have a $(1-\epsilon)$-approximation of l^*, since "rounding" an optimal stick s^* by clipping its ends at the boundaries of the pixels containing them can result in shortening it by only $O(\epsilon l_0)$, and all visible segments between two pixels have approximately the same length (within an additive term $O(\epsilon l_0)$).

Let s_1 and s_2 be two pixel sides, and assume that both are horizontal. (This assumption is made only to simplify the exposition.) $s_1 \cap P$ is a sequence of intervals $a_1, \ldots, a_{k_1}$ on s_1, and $s_2 \cap P$ is a sequence of intervals $b_1, \ldots, b_{k_2}$ on s_2. We need to determine whether there exist intervals a_i on s_1 and b_j on s_2 such that a_i and b_j *see each other*, i.e., there exist points $p \in a_i$ and $q \in b_j$ such that the line segment $\overline{pq}$ is contained in $\sigma \cap P$; see Figure 1(a).

Consider the region that is formed by intersecting P with the parallelogram defined by the two sides s_1 and s_2. Each connected component of this intersection is a simple polygon, and we can compute these polygons in time linear in the combinatorial complexity of the mega-square. We treat each of these polygonal components separately. Consider a component, and let $a_1, \ldots, a_{k_1}$ and $b_1, \ldots, b_{k_2}$ now denote the two sequences of intervals, on s_1 and s_2, respectively, of this component. The portion of P's boundary between adjacent intervals, b_j and b_{j+1}, along s_2 defines a *finger* of the component; there are fingers associated with s_2 and also fingers associated with s_1.

For an interval a_i on s_1, let $R(a_i)$ denote the range on s_2 that is seen from a_i when ignoring the obstruction that is caused by the fingers associated with s_2. Similarly define the range $R(b_j)$ of points on s_1 seen from b_j, when ignoring the obstruction that is caused by the fingers associated with s_1. Refer to Figure 1(b). The ranges $R(a_i)$ and $R(b_j)$ can be computed in logarithmic time, after linear-time preprocessing of the component [23]. Also, given intervals a_i on s_1 and b_j on s_2, one can determine in logarithmic time whether a_i and b_j see each other, using standard visibility query methods in simple polygons.

Notice that if $R(a_i) \supseteq b_j$ and $R(b_j) \supseteq a_i$ then a_i and b_j must see each other. Moreover, if a_i and b_j see each other, then clearly $R(a_i) \cap b_j \neq \emptyset$ and $R(b_j) \cap a_i \neq \emptyset$. These simple observations allow us to determine whether there exists a pair of intervals (one from s_1 and one from s_2) that see each other in time $O((k_1+k_2)\log^2 k_2)$.

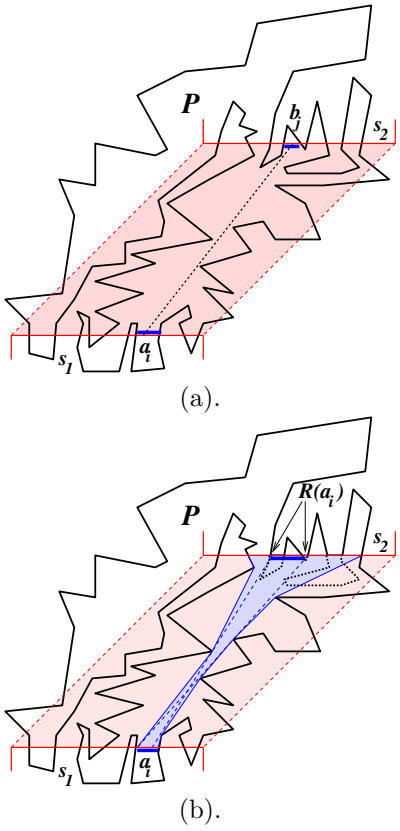


Figure 1: (a). Visibility between interval a_i on s_1 and interval b_j on s_2. (b). Using shortest path queries in the connected component (which define the shaded "hourglass"), we can compute the portion, $R(a_i)$, seen by a_i on the bottom of s_2, ignoring the fingers (dashed) associated with s_2 when computing visibility.

First, for each interval $t \in \{a_1, \ldots, a_{k_1}, b_1, \ldots, b_{k_2}\}$, we compute the range $R(t)$. If one or both of the endpoints of $R(t)$ lies in the interior of an interval, we determine in logarithmic time whether t sees one of these at most two intervals. At this point we shrink each of the ranges $R(t)$ so that only those intervals that are fully contained in it remain. Thus we can represent $R(t)$ by the pair of indices corresponding to the leftmost interval and rightmost interval that are contained in it.

In order to determine whether there is a pair of intervals a_i on s_1 and b_j on s_2 such that $R(a_i) \supseteq b_j$ and $R(b_j) \supseteq a_i$, we construct a two-level data structure. The first level of the data structure is a one-dimensional range tree T, of height $O(\log k_2)$, over the intervals on s_2 that are represented by their indices $1, \ldots k_2$. As usual, each node v of T is associated with a canonical subset of indices $\mathcal{I}_v$, and we attach to v an interval tree over the ranges corresponding to the indices in $\mathcal{I}_v$. The total size of the data structure is $O(k_2 \log k_2)$ ($O(k_2)$ per level of the range tree), and its construction time is $O(k_2 \log^2 k_2)$ ($O(k_2 \log k_2)$ per level of the range tree). The data structure supports queries of the form: Given a range $R(a_i)$ of an interval a_i on s_1, is there an interval b_j on s_2, such that a_i and b_j satisfy the condition above. The query time is $O(\log^2 k_2)$. We thus perform k_1 queries, one per interval on s_1. The total running time of this stage is $O((k_1+k_2)\log^2 k_2)$ per pixel pair. There are $O(1/\epsilon^4)$ pairs of pixels. Summing over all mega-squares, and recalling that the sum of the individual combinatorial complexities of the mega-squares is only linear ($\sum_{\sigma\in\mathcal{C}} n_\sigma = O(n)$), we get the complexity bound of the following theorem:

THEOREM 2.1. *One can compute a $(1-\epsilon)$-approximation of the largest stick inside a simple polygon in time $O((n/\epsilon^4)\log^2 n)$. A 1/2-approximation can be computed in time $O(n \log n)$.*

3 Approximating the Biggest Potato

In this section we give approximation algorithms for computing largest convex bodies ("potatoes") within a body P.

3.1 Largest Triangle

Let Δ^* be a maximum-area triangle in P with corners A, B, and C; we write $\Delta^* = ABC$. We again apply a recursive decomposition of the polygon using Chazelle cuts. Let a (resp., b, c) be the midpoint of side AB (resp., BC, CA). Let e be the first cut (diagonal) in the hierarchical cutting that intersects the (small) triangle abc. Note that e does not intersect any prior cut of the recursive decomposition, so e must intersect triangle ABC in a line segment, call it pq. Without loss of generality, we can assume that $p \in aA$ and $q \in BC$. We now consider two cases; refer to Figure 2.

Case 1. If $q \in Bb$, then area$(pqc) \geq (1/4)$area(ABC), since we can slide p along segment AB to a, reducing area(pqc) as we go, until area(pqc) = area$(abc) = (1/4)$area(ABC).

Case 2. If $q \in Cb$, let us assume that area$(pqa) \geq$ area(pqb); otherwise, we relabel. We wish to show that the larger of area(pqa) and area(pqc) is at least 3/16 of the area(ABC). Moving p toward A (or q toward C) reduces area(pqc) and increases area(pqa). Moving p toward a (or q toward b) has the opposite effect. By moving p and/or q we reduce the larger area and increase the smaller, until area(pqc) equals area(pqa). Let s be the midpoint of ac. Note that area(pqa) = area(pqc) if and only if the segment pq passes through the point s, since the altitudes from base pq to a or c will be equal. Now we continue to reduce area(pqa) by rotating pq around s until area(pqa) = area(pqb), namely when pq is parallel to AC. This triangle has area equal to 3/16 of the area of ABC.

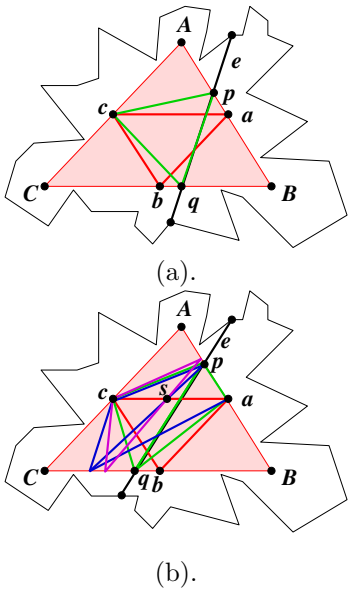


Figure 2: Illustration of (a). Case 1, and (b). Case 2.

To solve the subproblem, we proceed as before, traversing the boundary of the weakly visible polygon of e in order to compute a largest area triangle having one edge on e and having the opposite vertex on the boundary of P. We can argue that this triangle is at least a constant fraction (1/2) of the area of the largest triangle within P having one edge a subset of e.

In conclusion, we have

THEOREM 3.1. *In a simple polygon P with n vertices, one can compute an $O(1)$-approximation of the maximum-area triangle within P in time $O(n \log n)$.*

3.2 A PTAS for Largest Fat Triangles and Rectangles

In this section we give an efficient algorithm to compute

a $(1-\epsilon)$-approximation to the largest *fat* triangle or rectangle that can be placed in an n-vertex simple polygon P.

A triangle is *δ-fat* if all three of its angles are at least some specified constant, δ, the fatness parameter. Let $\Delta^* = ABC$ be a maximum-area δ-fat triangle contained in P, with angles α, β, γ at vertices A, B, C, respectively. Our goal is to compute a δ-fat triangle $\Delta \subset P$ such that $area(\Delta) \geq (1-\epsilon)area(\Delta^*)$.

We will need the following two claims.

CLAIM 3.1. *Assume that $\delta < 60$ and that $\gamma \geq \alpha, \beta$. Then, for any $\epsilon > 0$, there exists $\epsilon_0 > 0$ such that one can place a δ-fat triangle inside P with angles $\alpha+\epsilon_0, \beta+\epsilon_0, \gamma-2\epsilon_0$ and whose area is at least $(1-\epsilon)area(\Delta^*)$.*

Proof. Draw two line segments CD and CE from C to AB, such that CD creates an angle of ϵ_0 with CA, and CE creates an angle of ϵ_0 with CB, for some $0 < \epsilon_0 < \frac{\gamma}{2}$. Let a, b, and c denote the lengths of sides BC, CA, and AB, respectively. Let a' denote the length of CE and let b' denote the length of CD. From the law of sines we have $a' = \frac{a \sin\beta}{\sin(\beta+\epsilon_0)}$ and $b' = \frac{b \sin\alpha}{\sin(\alpha+\epsilon_0)}$. We also know that $area(ABC) = \frac{1}{2}ab\sin\gamma$ and $area(CDE) = \frac{1}{2}a'b'\sin(\gamma-2\epsilon_0)$. Replacing a' and b' in the latter equation and further developing the outcome we obtain that $area(CDE)$ is given by

$$\frac{1}{2}ab\sin\gamma\left(\frac{\sin\alpha\sin\beta\sin(\alpha+\beta+2\epsilon_0)}{\sin(\alpha+\epsilon_0)\sin(\beta+\epsilon_0)\sin(\alpha+\beta)}\right).$$

Thus, if one can pick ϵ_0, such that (i) $\gamma - 2\epsilon_0 \geq \delta$, and (ii) $\frac{\sin\alpha\sin\beta\sin(\alpha+\beta+2\epsilon_0)}{\sin(\alpha+\epsilon_0)\sin(\beta+\epsilon_0)\sin(\alpha+\beta)} \geq (1-\epsilon)$, then triangle CDE is δ-fat and its area is at least $(1-\epsilon)area(\Delta^*)$.

The assumptions $\delta < 60$ and $\gamma \geq \alpha, \beta$ imply that $\gamma > 60$ and that $\alpha + \beta < 120$. We can therefore strengthen the first condition above by requiring that $\epsilon_0 \leq \frac{60-\delta}{2}$. Concerning the second condition above, define

$$f(\epsilon_0) = \frac{\sin\alpha\sin\beta\sin(\alpha+\beta+2\epsilon_0)}{\sin(\alpha+\epsilon_0)\sin(\beta+\epsilon_0)\sin(\alpha+\beta)}.$$

Now, the two conditions above can be replaced by (i) $\epsilon_0 \leq \frac{60-\delta}{2}$, and (ii) $f(\epsilon_0) \geq (1-\epsilon)$.

Analyzing the function $f(\epsilon_0)$, we observe that for any α, β such that $\delta \leq \alpha, \beta < 90$ and $\alpha+\beta \leq 120$, $f(\epsilon_0)$ tends to 1 as ϵ_0 tends to 0. We conclude that there exists an ϵ_0 as required. □

CLAIM 3.2. *For any α', β' such that $\alpha \leq \alpha' \leq \alpha+\epsilon_0$ and $\beta \leq \beta' \leq \beta+\epsilon_0$, one can place a δ-fat triangle inside P such that two of its angles are α', β' and its area is at least $(1-\epsilon)area(\Delta^*)$.*

Proof. Draw two line segments CF and CG from C to AB, such that the angle CFB is α', and the angle CGA is β'. CFG is a δ-fat triangle. Since $\alpha \leq \alpha' \leq \alpha+\epsilon_0$ and $\beta \leq \beta' \leq \beta+\epsilon_0$, F must lie between A and D and G must lie between B and E. Now, CFG is contained in P, since it is contained in ABC, and $area(CFG) \geq (1-\epsilon)area(\Delta^*)$, since CFG contains CDE and the area of the latter triangle is at least $(1-\epsilon)area(\Delta^*)$ by Claim 3.1. □

We now describe a first, less efficient, approximation algorithm for placing in P a δ-fat triangle of area at least $(1-\epsilon)area(\Delta^*)$. Let *find_largest_copy(Q,P)* denote the algorithm of Chew and Kedem [14] for finding the largest copy (allowing translation and rotation) of a convex polygon Q within a simple polygon P. Given δ and ϵ, we first compute the value of ϵ_0 given by Claim 3.1. Let $T(\theta_1, \theta_2, \theta_3)$ denote a unit-diameter triangle with angles θ_1, θ_2, and θ_3. We first call *find_largest_copy(T(60,60,60),P)* to find a largest placement of an equilateral triangle within P. We record the area of the returned triangle. Now, for each $i, j \geq 0$, such that $\alpha' = \delta + i\epsilon_0$, $\beta' = \delta + j\epsilon_0$, and $\alpha', \beta' < 90$ and $\alpha'+\beta' < 120$, call *find_largest_copy(*$T(\alpha', \beta', 180-(\alpha'+\beta'))$, *P)* and record the area of the returned triangle. Finally, we return the triangle of maximum area among all triangles that were found by calls to *find_largest_copy*. We refer to this approximation algorithm as the *CK-algorithm*, since it is based on the method of Chew and Kedem.

The correctness of the CK-algorithm follows from the two claims preceding it. If δ=60, then we find the largest δ-fat triangle. If $\delta < 60$, then one of the triangles that is generated by the algorithm has angles α', β', such that $\alpha \leq \alpha' \leq \alpha+\epsilon_0$ and $\beta \leq \beta' \leq \beta+\epsilon_0$. According to the second claim above, the call to *find_largest_copy* with this triangle and P returns a δ-fat triangle of area at least $(1-\epsilon)area(\Delta^*)$. The running time of the CK-algorithm is determined by the running time of *find_largest_copy*, which is roughly $O(n^2 \log n)$.

We now devise a more efficient approximation algorithm, which uses the CK-algorithm as a subroutine. We take an approach similar to the approach we took in Section 2. Let a^* be the length of the shortest side of Δ^*. Since Δ^* is fat, it is easy to obtain a constant factor approximation a_0 of a^*, such that $a_0 \leq a^* \leq c_1 a_0$, for some small constant c_1. (This can be done by computing a largest inscribed disk in P, using the Voronoi diagram of P, which is computed in linear time for simple polygons [15].) A *mega-square* is an axis-parallel square of side length $O(a_0)$. We say that a mega-square m is *good* if one can place a δ-fat triangle in $m \cap P$ of area $area(\Delta^*)$.

CLAIM 3.3. *There exists a constant c_2, such that, for at least one of the vertices p_i of P, the mega-square of side length $c_2 a_0$ centered at p_i is good.*

Proof. The proof follows from the fatness of Δ^*. □

For each vertex $p \in P$, draw a mega-square of side length $c_2 a_0$ around p. Let $\mathcal{C}$ be the set of these n mega-squares. Consider each of the mega-squares in $\mathcal{C}$ separately. Let m be one of the mega-squares in $\mathcal{C}$. Tile m with a constant number of pixels of side length $\frac{\epsilon_0 a_0}{c_3}$, where ϵ_0 is the minimum between ϵ_0 as computed for the previous algorithm and a value that will be computed shortly, and $c_3 = (3 + \frac{3}{\tan\delta})$. Now, for each pixel, we check whether it is intersected by P's boundary or whether it lies completely outside P. The remaining pixels form a constant number of rectilinear polygons (each of constant complexity) that are contained in the interior of P. For each of these polygons, we apply the CK-algorithm in $O(1)$ time. Among all triangles that were found during the above process, we return the largest one.

LEMMA 3.1. *The area of the returned triangle is at least $(1-\epsilon_0)^3 area(\Delta^*)$.*

Proof. Consider a good mega-square m; then, $\Delta^* \subseteq m \cap P$. Let Q be the set of pixels of m that intersect Δ^*. A pixel in Q can be intersected by the boundary of P only if it is also intersected by the boundary of Δ^*. Let Q' denote the set Q of pixels, minus those pixels that are intersected by the boundary of Δ^*. We claim that the union of the set Q' of empty pixels forms a (single) rectilinear polygon that contains a triangle of area $(1-\epsilon_0)^2 area(\Delta^*)$. Indeed, it can easily be checked that the triangle that is obtained from Δ^* by shrinking Δ^* towards the center of its inscribed circle, with sides parallel to those of Δ^* and of lengths $(1-\epsilon_0)$ times the corresponding original lengths, is contained within the union of pixels Q'. Since there exists a triangle of area at least $(1-\epsilon_0)^2 area(\Delta^*)$, and since the CK-algorithm returns a fat triangle within factor $(1-\epsilon_0)$ of the largest possible, we have proved the claim. □

It remains to calculate ϵ_0. Let h be the length of the height of Δ^* that is perpendicular to the shortest side of Δ^* (whose length is a^*). Δ^*'s area is $\frac{a^* h}{2}$, and from the claim above the area of the triangle that is returned by the algorithm is $(1-\epsilon_0)^3 \Delta^*$. Therefore we choose ϵ_0, such that $(1-\epsilon_0)^3 = (1-\epsilon)$, or $\epsilon_0 = 1 - \sqrt[3]{(1-\epsilon)}$.

THEOREM 3.2. *One can compute a $(1-\epsilon)$-approximation of the largest fat triangle inside a simple polygon P in time $O(n)$.*

Remark 1: The algorithm above also applies to polygons with holes. However, the running time in this case becomes $O(n \log n)$, which is the time needed to compute the Voronoi diagram of P.
Remark 2: Instead of using the mega-squares/pixels approach, one can consider a constant number of orientations, and compute, for each of these orientations ρ, an approximation of the largest δ-fat triangle of orientation ρ that can be placed inside P.
Remark 3: Using similar ideas, one can obtain a PTAS for the largest fat rectangle problem (where a rectangle is c-fat if its aspect ratio is at most c, for some constant c). Details will appear in the full paper.

3.3 A Sampling Method for Maximum-Area Triangles

We give a randomized fully polynomial time approximation scheme (FPTAS) for finding a maximum-area empty triangle in a simple polygon, assuming that the area of the largest triangle is at least a constant fraction, δ, of the area of the polygon P. In the full paper we prove the following result:

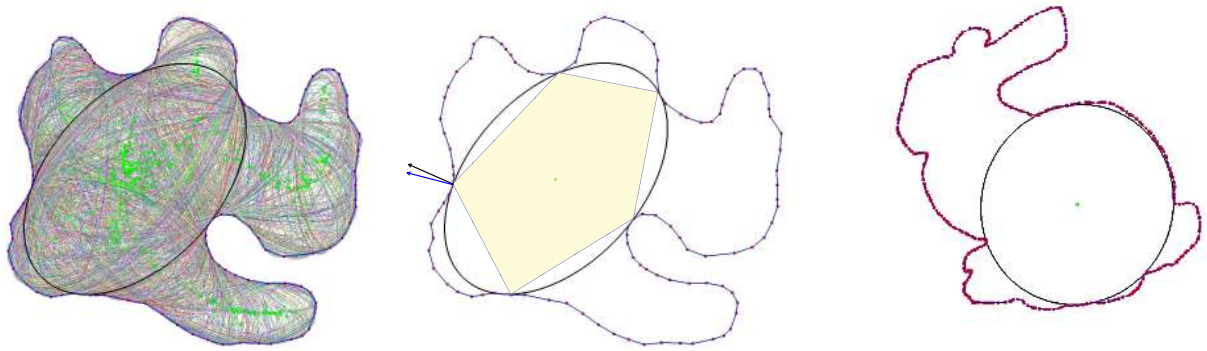
THEOREM 3.3. *Let P be a simple polygon with n vertices, and assume that a maximum-area triangle, τ^*, in P has area at least δ times the area of P, for fixed $\delta > 0$. Then, for any fixed $\epsilon > 0$, a $(1-\epsilon)$-approximation of τ^* can be computed, with high probability, in time $O(n + \min(k^{\frac{2\omega}{\omega+1}}, s^\omega) + s \log s \log n s)$, where k denotes the size of the visibility graph of $s = O\left(\frac{1}{\delta\epsilon^2} \log \frac{1}{\epsilon} \log \frac{1}{\delta}\right)$ uniform random samples of P, and $\omega \leq 2.36$ is the exponent for fast matrix multiplication [17].*

4 Maximum-Area Ellipses

In this section we study the problem of finding a maximum-area inscribed ellipse in a well-sampled smooth closed curve. We show that, as the sampling rate increases, our solution converges to an exact solution. We also show implementation results of a simplified version of the algorithm.

Let $\mathcal{E}^*$ denote a maximum-area empty ellipse contained in the smooth closed curve $\mathcal{S}$. Let a (resp., b) denote the length of the major (resp., minor) axis of $\mathcal{E}^*$. The *feature size* of a point on the curve is defined to be the distance from the point to the closest point of the medial axis [5]. Let ϕ be the minimum feature size of $\mathcal{S}$. Without loss of generality, assume that $\mathcal{S}$ lies inside a circle of diameter 1; hence, $\phi < 1$. Let $\epsilon_1, \epsilon_2 > 0$ be positive constants that are very small compared to 1.

A curve $\mathcal{S}$ is said to be *ϵ-sampled* if the length of the curve between any two consecutive sample points is no



(a). (b). (c).

Figure 3: (a) The set of all *feasible* ellipses for a given sampling of the boundary curve. The (dark) black ellipse is the maximum-area ellipse inside the sampled curve. (b) The maximum-area ellipse drawn inside the sampling. (c) A maximum-area ellipse inside a projection of the stanford bunny.

more than ϵ[1]. To make the sampling sufficiently dense, we assume that $\epsilon \leq \min\{\epsilon_1 b, \epsilon_2 \phi\}$.

Let $\mathcal{E}_B$ be the ellipse obtained from $\mathcal{E}^*$ by doing the following transformation: Scale (enlarge) $\mathcal{E}^*$ until it hits a sample point p_1. Continue scaling the ellipse, but now impose the constraint that it passes through point p_1; let p_2 be the next sample point it hits. Now, keeping the ellipse in contact with both p_1 and p_2, continue enlarging the ellipse, while increasing both of its axes, until the ellipse hits a third point, p_3. If the triangle $\Delta p_1 p_2 p_3$ contains the center of the current ellipse, then we output $\mathcal{E}_B$. Otherwise, the current ellipse can be translated away from its contact with the three points p_1, p_2, and p_3. We then restart the enlarging process again, and continue until we find an ellipse $\mathcal{E}_B$ that is supported by three sample points, p_1, p_2, and p_3, with $\Delta p_1 p_2 p_3$ containing the center of the ellipse. The main idea of the algorithm is to determine an ellipse that is close in some sense to $\mathcal{E}_B$; then, by lemma 4.1, this implies that the ellipse is also close to $\mathcal{E}^*$.

LEMMA 4.1. *Let* $\mathcal{E}_{in} = (1 - \frac{\epsilon}{b})\mathcal{E}_B$. *Then* $area(\mathcal{E}_{in}) \leq area(\mathcal{E}^*) \leq area(\frac{b}{b-\epsilon}\mathcal{E}_{in})$.

Proof. The process of inflating $\mathcal{E}^*$ to $\mathcal{E}_B$ means that $area(\mathcal{E}_B) \geq area(\mathcal{E}^*)$ and hence $area(\mathcal{E}^*) \leq area(\frac{b}{b-\epsilon}\mathcal{E}_{in})$. Let the radii of $\mathcal{E}_B$ be $a' > a$ and $b' > b$. Now, using the definition of sampling, $\mathcal{S}$ cannot

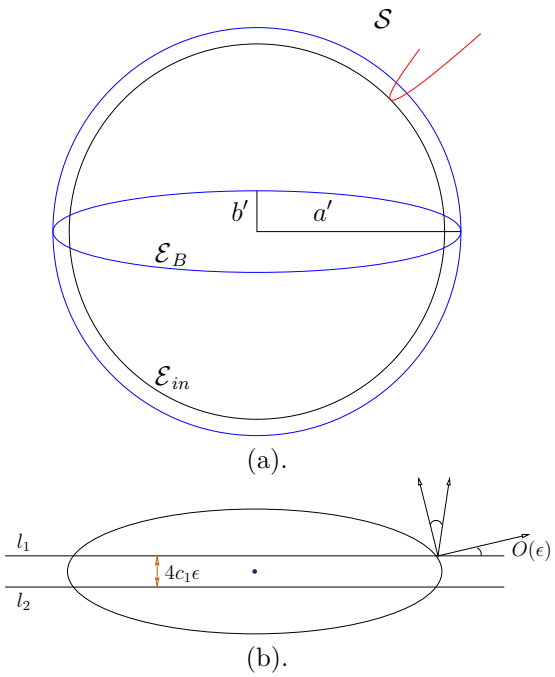


Figure 4: (a) The Kepler circle of $\mathcal{E}_B$. (b) An illustration for the proof of Lemma 4.3.

[1]In this section, ϵ is not an approximation factor.

penetrate $\mathcal{E}_B$ more than ϵ, since $\mathcal{E}_B$ is free of sample points. Scale the space such that $\mathcal{E}_B$ becomes a circle with radius a'. (Such a circle is also known as the *Kepler circle* of $\mathcal{E}_B$; see Figure 4(a).) Hence, in the rescaled space, $\mathcal{S}$ cannot penetrate the Kepler circle of $\mathcal{E}_B$ by more than $\frac{a'\epsilon}{b'}$. Let ${\mathcal{E}'}_{in} = \left(1 - \frac{\epsilon}{b'}\right)\mathcal{E}_B$. By the above argument, ${\mathcal{E}'}_{in} \subseteq \mathcal{S}$ and hence $area({\mathcal{E}'}_{in}) \leq area(\mathcal{E}^*)$. Also, $b' \geq b$ implies that $\left(1 - \frac{\epsilon}{b}\right) < \left(1 - \frac{\epsilon}{b'}\right)$, which means that $\mathcal{E}_{in} \subseteq {\mathcal{E}'}_{in}$, completing the proof. □

Next, we bound the angle between the normal of $\mathcal{E}_B$ and the normal to the curve $\mathcal{S}$ at each of the three points p_1, p_2, p_3. Towards this goal, we prove the following lemma about an arbitrary sample point p_i:

LEMMA 4.2. *If $\mathcal{E}_B$ and $\mathcal{S}$ both pass through the sample point p_i, then the angle between their normals at p_i is $O(\epsilon)$.*

Proof. Let p_{i-1} and p_{i+1} be the sample points before and after p_i. It can be shown using our sampling criterion that the angle spanned by $p_{i-1}p_ip_{i+1}$ is greater than $\pi - 4\arcsin(\epsilon/2)$ (see Lemma 10, [5]). A simple calculation shows that the normal to $\mathcal{S}$ at p_i and the normal to the edge $p_{i-1}p_i$ make an angle of at most $O(\epsilon)$. If $\mathcal{E}_B$ were arbitrarily large, then this would already prove that the angle between the normals to $\mathcal{S}$ and to $\mathcal{E}_B$ at p_i is $O(\epsilon)$; however, the curvature of $\mathcal{E}_B$ adds a small angle to the deviation, so we now show that this deviation is small.

We need to bound the angle between the normal to the ellipse $\mathcal{E}_B$ and the normal to the segment $p_{i-1}p_i$ given $|p_{i-1}p_i| \leq \epsilon$. Without loss of generality, assume that $\mathcal{E}_B$ is axis-aligned for this proof. Then the equation of $\mathcal{E}_B$ is $\frac{x^2}{a'^2} + \frac{y^2}{b'^2} = 1$. Note that as ϵ decreases, the angle between the normal to $\mathcal{E}_B$ at p_i and the normal to the segment $p_{i-1}p_i$ decreases. The maximum deviation of the normal occurs when the ellipse passes through either $p_{i-1}p_i$ or $p_{i+1}p_i$; thus, assume that $\mathcal{E}_B$ passes throuh $p_{i-1}p_i$. We now need to calculate the maximum angle between the normal of the ellipse at p_i and the normal to $p_{i-1}p_i$. Let $\tan(\theta')$ be the slope of the line passing through the center of the ellipse and p_i. Let 2δ be the angle subtended by $p_{i-1}p_i$ with respet to the center of the ellipse, so that p_{i-1} subtends an angle $\theta' + 2\delta$ on the x-axis. It is not hard to show that the maximum angle subtended by $p_{i-1}p_i$ is achieved at $\theta' = \pi/2$. A simple calculation shows that $\delta_{max} = O(\frac{\epsilon}{b})$. Let us sweep the ellipse $\mathcal{E}_B$ with a sector of angle δ_{max}. Let the chord subtended by δ_{max} be $p_{i-1}p_i$. Note that this only enlarges $p_{i-1}p_i$ and hence increases the deviation in the normal. Since we want to bound the deviation from above, we can afford to do this. It is not hard to show that the slope of such a chord is $-\frac{b'}{a'}\cot(\theta' + \delta_{max})$, and the slope of the tangent at p_i is $-\frac{b'}{a'}\cot(\theta')$. Using the fact that $b' \gg \epsilon$ and $\cos\delta \geq 0.5$, one can show that the angle between the slopes above is bounded by $O(\delta_{max})$. Hence, the total deviation added by the curvature of the ellipse is $O(\epsilon)$. This implies that the the angle between the normals of $\mathcal{E}_B$ and $\mathcal{S}$ at p_i is $O(\epsilon)$. See Figure 3(b) for an illustration. □

The following lemma is proved in the full version of the paper:

LEMMA 4.3. *Let p, q, and r be three sample points on the boundary of $\mathcal{E}_B$. Any ellipse passing through p, q, and r whose center is contained in triangle Δpqr, whose normals at p, q, and r are close to $\mathcal{E}_B$, and whose area is greater than $area(\mathcal{E}_B)$ cannot have its smaller radius of $O(\epsilon)$.*

Consider the conic defined by the equation

$$\begin{aligned} f(\mathbf{x}) &= q_{11}x^2 + 2q_{12}xy + q_{22}y^2 + r_1x + r_2y + f \\ &= \mathbf{x}^tQ\mathbf{x} + R\mathbf{x} + 1 = 0. \end{aligned} \tag{4.1}$$

Without loss of generality, $f = 1$. Note that Q is symmetric and that $f(\mathbf{x})$ represents an ellipse if and only if Q is positive definite. The area of the ellipse is directly proportional to $\det Q^{-1}$. Now we are ready to present the approximation algorithm; refer to Algorithm 1. The first step in the algorithm is *linearization* [20]. In general, an ellipsoid in d dimensions admits a *linearization* of dimension $d + \frac{d(d+1)}{2}$; thus, for our purposes, we lift to $\mathbb{R}^5$. The complexity of the convex hull $\mathfrak{C}$ of the linearization L' is, in the worst case, $O(n^2)$ (Step 2). Note that any conic in $\mathbb{R}^2$ in the form of equation 4.1 becomes a hyperplane in $\mathbb{R}^5$ after linearization. Also note that a half-space in $\mathbb{R}^2$ maps to a half-space in $\mathbb{R}^5$ after linearization.

After the linearization step, the next step (Step 3) of the algorithm is to examine each 2-simplex (triangle) of $\mathfrak{C}$ to determine if there is an ellipse through its three vertices that is almost contained in $\mathcal{S}$; we want to maximize the area among such ellipses[2]. Note that each 2-simplex is adjacent to two 3-simplices and at most five 4-simplices; each 3-simplex is adjacent to two 4-simplices. Each of the neighbors of a 2-simplex can be obtained in constant time, assuming we have any standard adjacency representation of the simplices in $\mathfrak{C}$.

[2]Note that $\mathcal{E}_B$ linearizes to a plane in the linearization which passes through three points and supports $\mathfrak{C}$ since it is empty of sample points. By convexity arguments, the three points that $\mathcal{E}_B$ passes through in the linearization define a simplex subset of $\mathfrak{C}$. Algorithm 1 tries to find this 2-simplex in the linearization or a simplex that corresponds to an ellipse that is larger than $\mathcal{E}_B$ and follows all the constraints of $\mathcal{E}_B$.

Once we fix a τ to process, we know that the ellipse we seek passes through the vertices of τ (in $\mathbb{R}^2$). Let the vertices be p, q, and r. Then we have the relations in Step 4 of the algorithm. These three relations help us map the problem in 2-dimensional space, (r_1, r_2). We can now represent the equation of a conic in terms of $R = (r_1, r_2)$. Now we solve the optimization problem in Step 5, which computes an ellipse $\mathcal{E}_{\tau_{p,q,r}}$ passing through p, q and r, whose normals at p, q, r each make a small angle with the corresponding normals of $\mathcal{S}$, whose center is contained in the triangle Δpqr, and whose small radius is at least $c\epsilon$ for some constant much larger than 1. But since the ellipse has to match the normal and the triangle condition, its not hard to see that the ellipse cannot have any sample points inside it; otherwise, the normal condition is violated. Hence, we have an ellipse $\mathcal{E}_{\tau_{p,q,r}}$, which has all points of L outside or on it, passes through p, q, and r, and has *maximal* area (Step 5).

Algorithm 1 Algorithm for computing an approximate maximum-area inscribed ellipse.

Require: A point set $L = \{p[1], p[2], ..., p[n]\} \in \mathbb{R}^2$

1: $L' = \{(x^2, y^2, \frac{1}{2}xy, x, y) | (x, y) \in L\}$

2: $\mathfrak{C} = \text{ConvexHull}(L')$

3: **for all** $\tau_{p,q,r} \in \mathfrak{C}$ such that $\tau_{p,q,r}$ is a 2-simplex. **do**

4: Using the relations

$$p^tQp + Rp + f = 0 \quad (4.2)$$

$$q^tQq + Rq + f = 0 \quad (4.3)$$

$$r^tQr + Rr + f = 0 \quad (4.4)$$

represent $q_{11} = \psi_{11}(R)$, $q_{22} = \psi_{22}(R)$ and $q_{12} = \psi_{12}(R)$. Note that $\psi_{11}(R), \psi_{22}(R)$ and $\psi_{12}(R)$ are all linear in $R = \begin{pmatrix} r_1 & r_2 \end{pmatrix}$.

5: Solve the following optimization problem

$$\min A_{\tau_{p,q,r}} = \psi_{11}(R)\psi_{22}(R) - \psi_{12}^2(R)$$

subject to

$\psi_{11}(R)\psi_{22}(R) - \psi_{12}^2(R) > 0$

$\psi_{11}(R) > 0$

$\psi_{22}(R) > 0$

The ellipse hyperplane supports $\mathfrak{C}$

The center of the ellipse lies inside Δpqr.

The small radius of the ellipse $\geq c\epsilon$.

Ellipse satisfies the normal constraints at p, q, r

6: **if** The problem is infeasible **then**

7: $A_{\tau_{p,q,r}} = \infty$

8: **else**

9: Let $\mathcal{E}_{\tau_{p,q,r}}$ be the ellipse corresponding to the solution of the optimization problem.

10: **end if**

11: **end for**

12: Return $\mathcal{E}_{alg} =$ Max area $\mathcal{E}_{\tau_{p,q,r}}$ found in step 3.

The algorithm returns the area that is maximum over all areas computed for feasible ellipses. We will need the following lemma to bound the running time of our algorithm (whose proof is deferred to the full paper):

LEMMA 4.4. *Given $\tau_{p,q,r}$, the optimization problem in Step 5, Alg 1 can be solved in $O(1)$ time.*

We are ready to prove the following theorem:

THEOREM 4.1. *In $O(n^2)$ time, Algorithm 1 returns an ellipse $\mathcal{E}_{alg}$ such that*

$$(1-\frac{1}{2c})area(\mathcal{E}_{alg}) \leq area(\mathcal{E}^*) \leq area(\mathcal{E}_B) \leq area(\mathcal{E}_{alg}).$$

Proof. Note that $\mathcal{E}_{alg}$ passes though three points of L and has no sample points inside it. Its small radius is at least $c\epsilon$, where c is a constant much larger than 1. (The existence of such a c is guaranteed by Lemma 4.3.) We also know that $\mathcal{E}_B$ is a feasible solution of the optimization problem that we solve using the algorithm, and the algorithm maximizes the area; hence, $area(\mathcal{E}_{alg}) \geq area(\mathcal{E}_B) \geq area(\mathcal{E}^*)$. Since $\mathcal{E}_{alg}$ is free of sample points, $\mathcal{S}$ can penetrate it by a distance of at most $\frac{\epsilon}{2}$, and hence if we shrink $\mathcal{E}_{alg}$ by a factor of $(1 - \frac{1}{2c})$, it will be totally inside $\mathcal{S}$ and hence $(1 - \frac{1}{2c})area(\mathcal{E}_{alg}) \leq area(\mathcal{E}^*)$, proving the inequality.

The main time taken by the algorithm is to examine each 2-simplex of $\mathfrak{C}$ and solve the optimization problem. Lemma 4.4 readily implies that the running time of the algorithm is $O(n^2)$. □

Implementation. We have recently implemented a variant of Algorithm 1[3]. The algorithm we implemented can be made to fail on contrived examples but usually works well in practice for most input datasets. It relies on the observation that for a dense enough sampling, *usually* there exists an ellipse comparable in area to $\mathcal{E}_B$ passing through 5 sample points. In this case we can just linearize the problem, and then for each ellipse passing through the 4-simplices, check feasibility by testing (1) the normal condition, (b) the condition that the ellipse center lies inside the convex hull of the supporting points, and (3) the fatness of the ellipse. The implementation outputs the ellipse that has maximum area over all feasible ellipses. Two example runs of the implementation are shown in Figure 3. The implementation is written in C++ and uses qhull for the convex hull computation in $\mathbb{R}^5$ [6].

We have seen how to use a linearization of 2-dimensional ellipses to find the largest area ellipse inside

[3]See http://www.compgeom.com/~piyush/potato/

a well-sampled smooth curve. Although one could try to generalize this algorithm to higher dimensions, its running time would be exponential in the dimension d. Even for $d = 3$, it seems that the linearization step is too slow to be practical.

Acknowledgements.

M. Katz and J. Mitchell are partially supported by grant No. 2000160 from the U.S.-Israel Binational Science Foundation. J. Mitchell is partially supported by grants from Metron Aviation, NASA Ames (NAG2-1620), and the National Science Foundation (CCR-0098172, ACI-0328930, CCF-0431030, CCF-0528209). A. Sityon is partially supported by the Lynn and William Frankel Center for Computer Sciences. P. Kumar was supported by FSU FYAP Grant 080003-550-014868.

References

[1] P. K. Agarwal, B. Aronov, and M. Sharir. Computing envelopes in four dimensions with applications. *SIAM J. Comput.*, 26:1714–1732, 1997.

[2] P. K. Agarwal and M. Sharir. Efficient randomized algorithms for some geometric optimization problems. *Discrete Comput. Geom.*, 16:317–337, 1996.

[3] P. K. Agarwal, M. Sharir, and S. Toledo. Applications of parametric searching in geometric optimization. *J. Algorithms*, 17:292–318, 1994.

[4] A. Aggarwal, M. M. Klawe, S. Moran, P. W. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

[5] N. Amenta, M. Bern, and D. Eppstein. The Crust and the β-Skeleton: Combinatorial Curve Reconstruction. *Graphical Models and Image Processing*, 60:125–135, 1998.

[6] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, Dec. 1996.

[7] F. Bernardini, J. T. Klosowski, and J. El-Sana. Directional discretized occluders for accelerated occlusion culling. *Computer Graphics Forum*, 19(3), Aug. 2000.

[8] S. Bischoff and L. Kobbelt. Ellipsoid decomposition of 3d-models. In *Proceedings of 1st International Symposium on 3D Data Processing Visualization Transmission*, pages 480–488, 2002.

[9] J. E. Boyce, D. P. Dobkin, R. L. Drysdale, III, and L. J. Guibas. Finding extremal polygons. *SIAM J. Comput.*, 14:134–147, 1985.

[10] P. Brunet, I. Navazo, J. Rossignac, and C. Saona-Vzquez. Hoops: 3d curves as conservative occluders for cell visibility. *Computer Graphics Forum*, 20(3), 2001.

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

[12] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, 1991.

[13] B. Chazelle and M. Sharir. An algorithm for generalized point location and its application. *J. Symbolic Comput.*, 10:281–309, 1990.

[14] L. P. Chew and K. Kedem. A convex polygon among polygonal obstacles: Placement and high-clearance motion. *Comput. Geom. Theory Appl.*, 3:59–89, 1993.

[15] F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete Comput. Geom.*, 21(3):405–420, 1999.

[16] D. Cohen-Or, S. Lev-Yehudi, A. Karol, and A. Tal. Inner-cover of non-convex shapes, 2000.

[17] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proc. 9th Annu. ACM Sympos. Theory Comput.*, pages 1–6, 1987.

[18] A. DePano, Y. Ke, and J. O'Rourke. Finding largest inscribed equilateral triangles and squares. In *Proc. 25th Allerton Conf. Commun. Control Comput.*, pages 869–878, Oct. 1987.

[19] D. P. Dobkin and L. Snyder. On a general method for maximizing and minimizing among certain geometric problems. In *Proc. 20th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 9–17, 1979.

[20] J. Erickson and P. K. Agarwal. Geometric Range Searching and Its Relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry, Contemporary Mathematics 223*, chapter 1, pages 1–56. American Mathematical Society, 1999.

[21] S. J. Fortune. A fast algorithm for polygon containment by translation. In *Proc. 12th Internat. Colloq. Automata Lang. Program.*, volume 194 of *Lecture Notes Comput. Sci.*, pages 189–198. Springer-Verlag, 1985.

[22] 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.

[23] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, Oct. 1989.

[24] 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.

[25] V. Koltun, Y. Chrysanthou, and D. Cohen-Or. Virtual occluders: An efficient intermediate pvs representation. In *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 59–70, Jun 2000.

[26] E. A. Melissaratos and D. L. Souvaine. Shortest paths help solve geometric optimization problems in planar regions. *SIAM J. Comput.*, 21(4):601–638, 1992.