Polygons Cuttable by a Circular Saw

Erik D. Demaine* Martin L. Demaine* Craig S. Kaplan†

Abstract

We introduce and characterize a new class of polygons that models wood, stone, glass, and ceramic shapes that can be cut with a table saw, lapidary trim saw, or other circular saw. In this model, a *circular saw* is a line segment (in projection) that can move freely in empty space, but can only cut straight into a portion of material. Once a region of material is separated from the rest, it can be picked up and removed to allow the saw to move more freely. A polygon is called *cuttable by a circular saw* if it can be cut out of a convex shape of material by a sufficiently small circular saw. We prove that a polygon has this property precisely if it does not have two adjacent reflex vertices. As a consequence, every polygon can be modified slightly to make it cuttable by a circular saw. We give a linear-time algorithm to cut out such a polygon using a number of cuts and total length of cuts that are at most 2.5 times the optimal. We also study polygons cuttable by an arbitrarily large circular saw, which is equivalent to a ray, and give two linear-time recognition algorithms.

1 Introduction

There are many saw tools[1] for cutting a desired shape out of material ranging from soft wood to brittle ceramic, glass, and stone. Cutting out a complex nonconvex shape requires careful maneuvering of the saw, and for many devices is simply impossible because turns cannot be made. For soft materials, the primary example being wood, this problem is easily solved with such devices as the scroll saw and band saw, which permit cutting along a bounded-curvature path because the cut made is slightly wider than the blade. Another example is cutting (indeed melting) styrofoam with a hot wire, a problem studied by Jaromczyk and Kowaluk [7].

For more brittle material, turns are generally impractical but possible using wire saws, either with a diamond-impregnated wire, or a braided wire dripping with silicon-carbide grit. Such saws are slow and can be expensive even for straight cuts. For stained glass and ceramic tiles it is common to score the material, which leads to the well-studied notion of a *glass cut* or *guillotine cut* [2, 8, 10, 12, 13], but only allows for convex shapes. With lapidary saws, e.g., cutting precious and semiprecious stones for jewelry, traditionally the only nonconvex shape that has been of interest is a heart, but more recently there has been a growing interest in cutting more complex nonconvex shapes.

In this paper we discuss *circular saws* such as the popular table saw. Such a device makes precise cuts in essentially any material, using different blades, and is likely the cheapest and fastest motorized saw available. Its only problem is that turns are impossible, essentially because the projection of the blade is a positive-length line segment, and the width of the cut is equal to the width of the blade. As we will show, even without turns it is possible to cut a wide variety of nonconvex shapes.

We stress that a circular saw cannot make turns even if the blade is small. To illustrate this, let us contrast between two popular saws: the table saw and the scroll saw (see Figure 1). The table saw is a type of circular saw, and thus falls under the umbrella of this paper; the scroll saw does not. The difference between the two saws is illustrated by looking at the blades. The key is that the blades have teeth to cut

*Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, email: {eddemaine, mldemaine}@uwaterloo.ca

†Department of Computer Science and Engineering, University of Washington, Seattle, Washington 98195-2350, USA, email: csk@cs.washington.edu

[1] We do not consider other cutting tools that puncture the material from above rather than cutting through it, such as lasers and water jets.

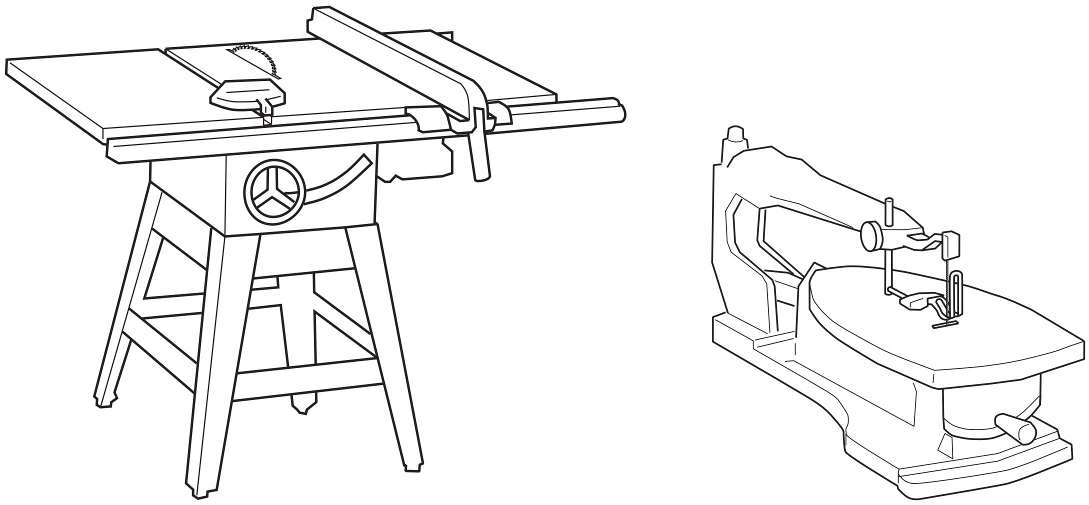


Figure 1: Left: A table saw, which is a type of circular saw, and can only cut straight. Right: A scroll saw, which can cut curves.

a path thicker than the blade itself. A circular saw has teeth all around the circle, and hence the cut-out of the blade is at any moment a rectangle, preventing turns because teeth only cut forward. On the other hand, a scroll saw has teeth only at the front edge, and hence the back of the blade is surrounded by empty space, allowing turning.

Model. More formally, we model a circular saw by concentrating on the projection in two dimensions. A circular saw is thus a line segment, of some length s called the *size* of the saw.[2] We start with some open set of material M_0, say the interior of a convex polygon. Our goal is to cut away from this material all but the interior of a contained polygon P.

For convenience, we consider the saw moving while the material remains stationary, although in reality the roles are reversed. The saw is permitted to translate and rotate freely in empty space. On the other hand, when the saw is in contact with the material, it may only move by translating in one of the two directions parallel to the saw itself, and thus the three degrees of freedom are reduced to one. In other words, in addition to moving in free space, the saw can either cut all the way through the material, or cut part way in and then retract. Such a motion is called a *cut*; we assume that only a finite number of cuts can be made in a finite amount of time.

The effect of a cut is that it removes material, i.e., subtracts from the remaining material M any points traversed by the saw. Note that such an operation preserves the property that the remaining material M is an open set. Thus the *boundary* of M is simply its closure minus itself, i.e, $\overline{M} \setminus M$. In particular, we will refer to the edges of this boundary as *edges* of M.

Finally, once the material M has been disconnected into multiple components, any of the components may be picked up and discarded. (We do not impose that pieces be removed by motions restricted to the plane.) Discarded components become new free space, allowing the saw to move more freely.

A polygon P is *cuttable by a circular saw of size* s if it can be cut from its convex hull, or equivalently any polygon containing the convex hull of P. Specifically, a polygon is *cuttable by a circular saw* (or *segment-cuttable*) if it can be cut with a saw of some (sufficiently small) size s. On the other extreme, if a polygon is cuttable by all circular saws, i.e. by an arbitrarily large circular saw, then it can also be cut by a sequence of straight cuts originating at infinity; thus we call the polygon *ray-cuttable*.

Note that it is easy to cut any polygon down to a contained convex polygon: a convex polygon is an intersection of halfplanes, so it can be cut with lines and thus with a circular saw of any size. Thus, for

[2]This value depends both on the diameter of the circular saw, and how much of the saw is exposed to the material. We assume that the circular saw is oriented perpendicular to the material; although some saws allow it to be tilted, this is not relevant to our problem.

defining cuttability by a circular saw, it is irrelevant which polygon containing the convex hull of P we start with.

Main Results. In this paper we present linear-time algorithms to recognize polygons cuttable by a (small) circular saw, and to recognize polygons cuttable by all (large) circular saws. In particular, for small saws, we prove the following simple characterization:

Theorem 1 *A polygon is cuttable by a circular saw precisely if it does not have two adjacent reflex vertices (vertices with interior angle $> \pi$).*

We also design efficient approximation algorithms for cutting with a small circular saw:

Theorem 2 *There is a linear-time algorithm that, given a polygon with no two adjacent reflex vertices, cuts the polygon with a circular saw so that the total length of cuts and total motion of the saw are within a factor of 2.5 of optimal. If no two edges of the polygon are collinear, then the number of cuts is also within a factor of 2.5 of optimal.*

For ray-cuttability, the characterization is somewhat more complicated but nonetheless optimal in terms of running time:

Theorem 3 *Ray cuttability of a polygon can be decided in linear time.*

The rest of this paper is outlined as follows. Section 2 analyzes polygons cuttable by small circular saws, proving Theorems 1 and 2. In Section 2.4, we show some examples of cuttable polygons, and present methods for making polygons cuttable by a circular saw. Section 3 analyzes polygons cuttable by large saws, proving Theorem 3. We end with several open problems in Section 4.

2 Small Saws

This section begins by proving the simple characterization in Theorem 1 of polygons cuttable by a circular saw. There are two parts to the proof: that a polygon is uncuttable if it has two adjacent reflex vertices (Section 2.1), and that a polygon is cuttable otherwise (Section 2.2). In Section 2.3 we use the latter proof to develop approximation algorithms for cutting such polygons, thereby proving Theorem 2. Finally, Section 2.4 demonstrates a method for slightly modifying polygons to make them cuttable by a circular saw.

2.1 Necessity

Consider an edge e of P whose endpoints are reflex vertices; see Figure 2. In particular, e cannot be an edge of the initial material M_0 because M_0 contains the convex hull of P. We claim that e will never be entirely on the boundary of the remaining material M, no matter how we cut M_0 without cutting P. Indeed, we will show the following stronger claim: at all times, the set of points along e that are on the boundary of M is finite.

Initially this claim is true for all points p along e. Apply induction on the number of cuts; in other words, suppose that the claim is true up to some time, and then we make another cut. If this cut starts at a point along e, we claim that it cannot move along the edge starting at that point. This is because the induction hypothesis and the reflexivity of the endpoints of e imply that the saw cannot be oriented against the edge e

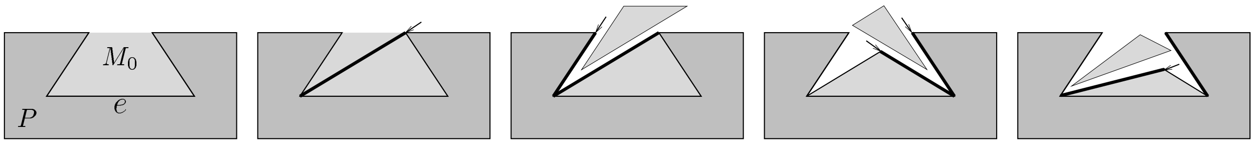


Figure 2: Attempts at cutting all material against an edge e with two reflex endpoints, using a circular saw.

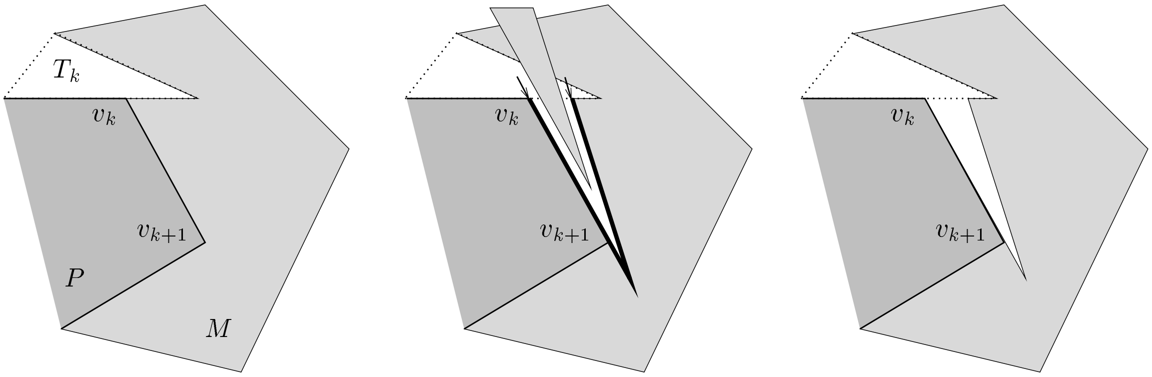


Figure 3: The case of a convex angle v_{k+1}.

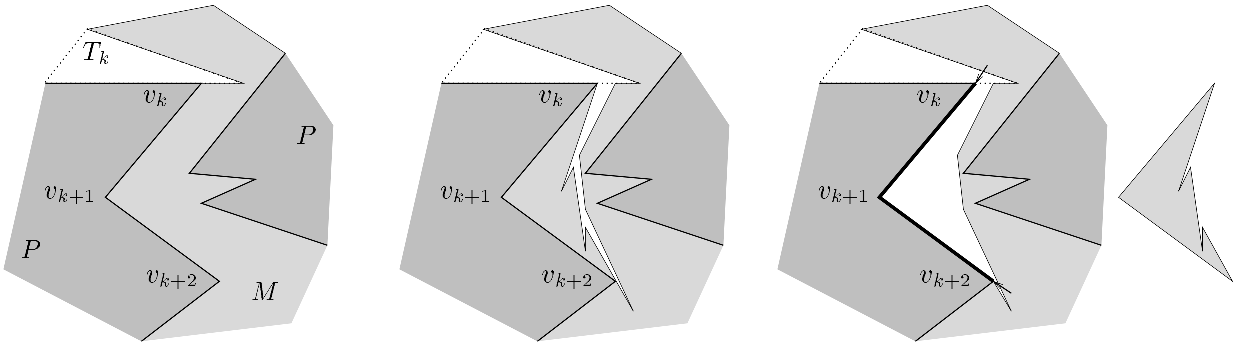


Figure 4: The case of a reflex angle v_{k+1}.

before coming in contact with the material M. Hence the best the saw could do is cut along a line segment with one endpoint p possibly on the edge e. This cut can only add p to the set of points along e that are on the boundary of M. By induction the result holds.

2.2 Sufficiency

Now suppose we have a polygon P that does not have two adjacent reflex vertices. Let v_0 be a vertex on the convex hull of P, and let $v_0, v_1, \ldots, v_{n-1}$ be the vertices of P in cyclic order.

We prove by induction that P is cuttable by a circular saw. The induction hypothesis is that there is a cutting such that the remaining material M, which diminishes from step to step, has the following properties:

1. the chain of polygon edges (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) are edges of M; and
2. there is a half disk centered at v_k that is disjoint from M (in particular, v_k must be convex).

A consequence of these two properties is that (provided $k > 0$) there is a triangle T_k, disjoint from M, that has one edge against the polygon edge (v_{k-1}, v_k). The idea is to use this triangle for "maneuvering space" for the saw in order to reach the next convex vertex of the polygon, either v_{k+1} or v_{k+2}.

The induction hypothesis is clearly true for $k = 0$ because v_0 is on the boundary of the convex hull of P. (If M_0 is not the convex hull of P, we can first cut M down to this convex polygon.) Assume by induction that the hypothesis is true for k. We distinguish cases depending on the angle at vertex v_{k+1}.

v_{k+1} is convex. This is the easy case; see Figure 3. By maneuvering the saw around T_k, we can make two cuts to remove a triangle against the edge (v_k, v_{k+1}) that overlooks v_{k+1}. This proves the induction hypothesis for $k + 1$.

v_{k+1} **is reflex.** This case is more difficult; see Figure 4. The idea is to cut a triangle overhanging v_{k+2}, which is guaranteed to be convex. But in this case v_k and v_{k+2} might not be visible from each other, so we may not be able to reach v_{k+2} in two cuts like the previous case. Instead, we choose a polygonal path starting at v_k, entering M, and ending so that the last edge touches and extends past v_{k+2}, while strictly avoiding the rest of the polygon P. Then we cut a sequence of triangles along this path, each time cutting slightly past the path so that the next triangle can be cut. At that point, we have cut slightly past v_{k+2}, so we can make a single cut from v_{k+2} to v_{k+1}, and another from v_k to v_{k+1}. This proves the induction hypothesis for $k+2$.

In the end, $k = n$ and the entire boundary of P is on the boundary of M; in other words, the polygon is cut out.

2.3 Approximation

The previous section in fact describes an algorithm to find a sequence of cuts to achieve a given polygon. Therese Biedl[3] observed that a particular choice of the polygonal path from v_k to v_{k+2} in the last step of the algorithm leads to good bounds. Namely, as shown in Figure 5, we can cut a triangle from v_k to v_{k+1}, then cut a triangle from near v_{k+1} to slightly past v_{k+2}, and then make a single cut back from v_{k+2} to v_{k+1}.

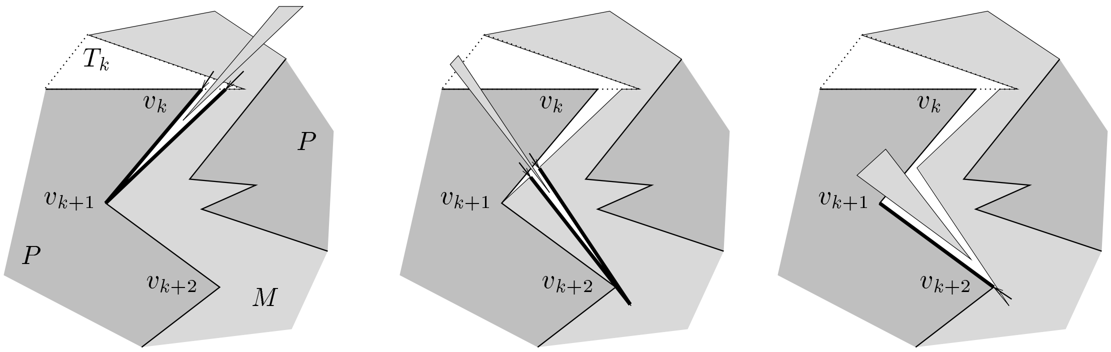


Figure 5: An efficient implementation of the step in Figure 4.

Thus, when v_{k+1} is reflex, five cuts suffice to start from just past v_k and reach just past v_{k+2}. Furthermore, the length of each cut is arbitrarily close to the length of a polygon edge between v_k and v_{k+2}. When v_{k+1} is convex (Figure 3), two cuts suffice to start from just past v_k and reach just past v_{k+1}, and each cut has length arbitrarily close to the polygon edge between v_k and v_{k+1}.

This algorithm achieves constant-factor approximations because of two simple lower bounds. The perimeter of the polygon is a lower bound on the total length of cuts, as well as the total motion of the saw. Assuming no edges of the polygon are collinear, the number of edges of the polygon is a lower bound on the number of cuts.

Considering both cases in the algorithm, either we cut out the next two vertices using 5 cuts and 5 times the optimal length and motion, for an average of 2.5 per vertex, or we cut out the next vertex using 2 cuts and 2 times the optimal length and motion. In the worst case, the factor is 2.5, thus proving Theorem 2.

We observe that the bound of 2.5 on the approximation factor is in some sense tight for this algorithm. If P is a regular star with n vertices and unit edge lengths (as in the bottom of Figure 6), then the number of cuts and total length of cuts is roughly $2.5n$ in the algorithm, but only n in the optimal (ray) cutting. In this example, several cuts made by the algorithm are obviously unnecessary: the polygon would have been cut out without them. However, the suboptimality of the algorithm's cuts can be made more subtle by enclosing the star in a rectangle as in the top of Figure 6; this change to P adds only $O(1)$ additional cuts in both the algorithm's solution and the optimal solution.

[3]Personal communication, August 2000

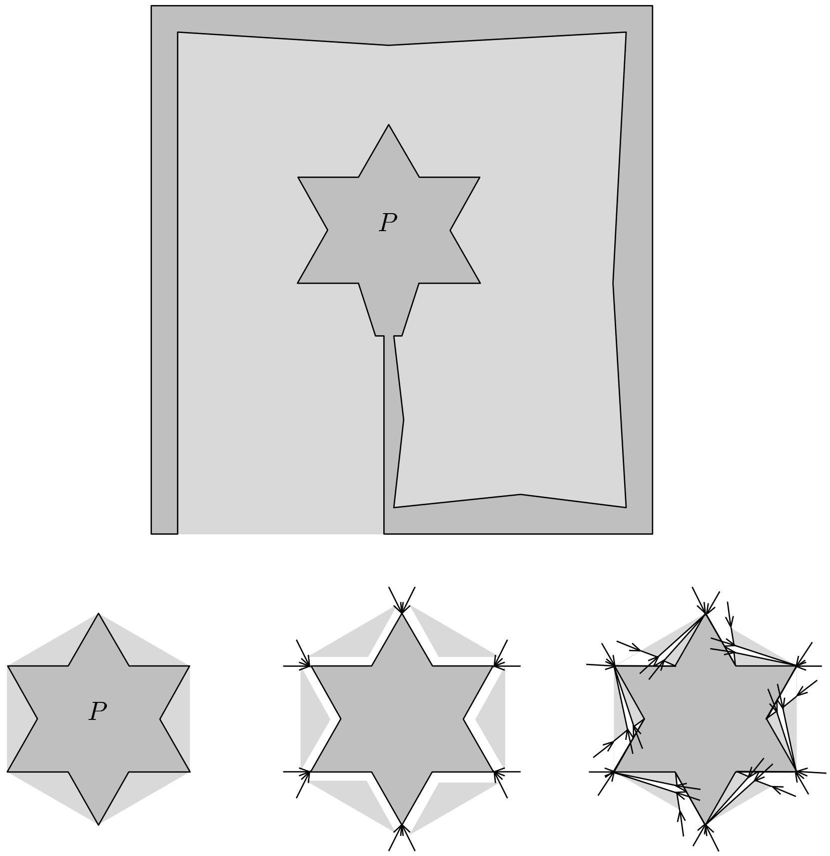


Figure 6: (Below) Cutting a six-pointed star requires only 12 cuts, but the approximation algorithm uses 30 cuts, for a ratio of 2.5. However, some of the algorithm's cuts are superfluous. (Above) If the star is enclosed in a rectangle, the algorithm has the same performance ratio but there are no trivial improvements to the sequence of cuts.

2.4 Examples

Some interesting examples of polygons cuttable and not cuttable by a circular saw are given in Figures 8, 9, 10, and 11. For the last three figures, we wrote a program to automatically modify a given polygon to be cuttable by a circular saw, and applied it to profiles of Trajan, Archimedes, and Hitchcock. Each figure shows the original polygon on the left and the resulting cuttable polygon on the right. Whenever the program finds an edge whose two endpoints are reflex vertices (drawn thick in the figures), it "dents" the entire edge slightly outward by introducing one additional vertex; see the left two pictures in Figure 7. Alternatively, to decrease the polygon area instead of increase it, we could have introduced a small inwards dent (say, an equilateral triangle) in the middle of each such edge, introducing three additional vertices; see the middle picture in Figure 7. More efficiently, we could align one edge of this dent with an edge of the polygon (as in the fourth picture in Figure 7), or use the first method but move the vertices slightly inwards (as in the rightmost picture in Figure 7).

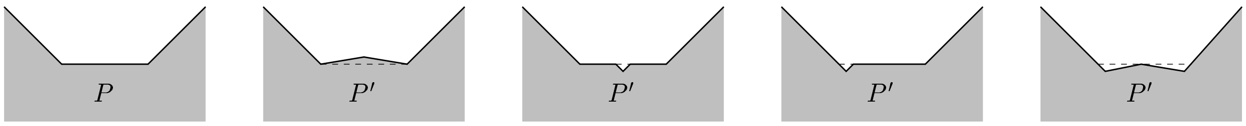


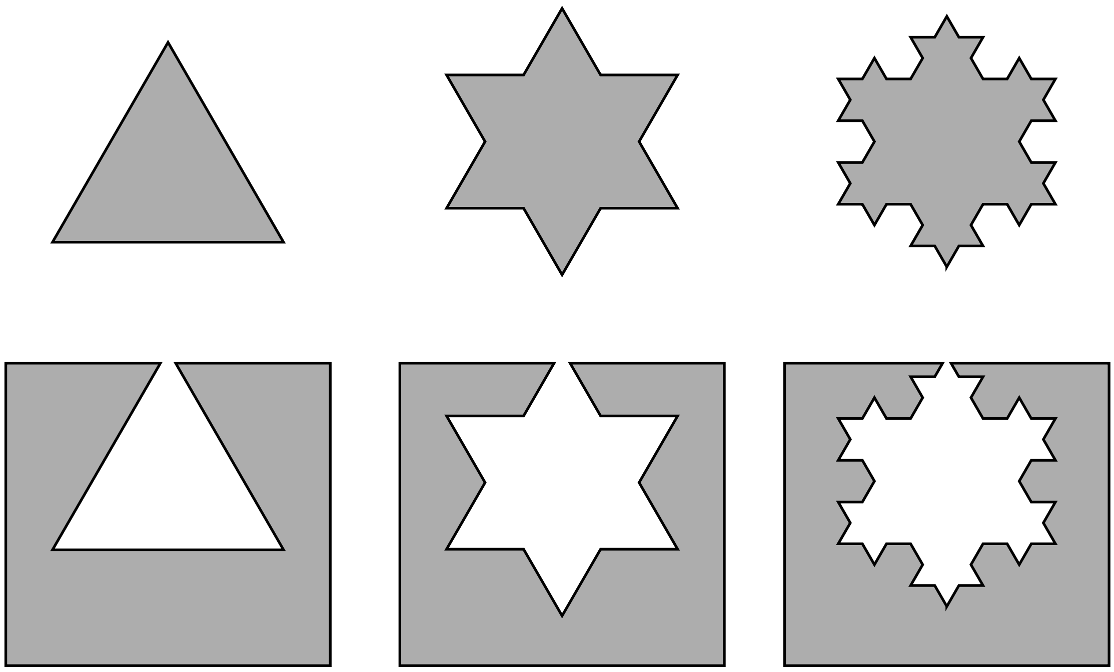
Figure 7: Four methods for making an edge incident to two reflex vertices cuttable by a small circular saw.

In any case, we obtain a polygon that is cuttable by a circular saw and arbitrary close to the given polygon in the following sense:

Corollary 1 *Given any polygon P with n vertices, there is a polygon Q with at most $2n-3$ vertices that is cuttable by a circular saw, and such that the area of the symmetric difference is as small as desired. In addition, the polygon Q can be chosen to contain P or to be contained in P.*

The $2n-3 = n+(n-3)$ bound follows because there are at least three convex vertices, and each convex vertex prevents at least one unique edge from having two reflex endpoints. A stronger bound of $2n-4$ holds except when $n=3$, i.e., P is a triangle, in which case $P=Q$.

Note that for the portraits in profile, few edges need to be modified. This is presumably because the head in profile is made up of a sequence of nearly convex chains: the forehead, top, and back of the head, the curve of the nose, and the chin. A common location for adjacent reflex vertices is in the space between the upper lip and the nose.



(a) (b) (c)

Figure 8: Saw cutting and the Koch snowflake: we attempt to cut the first three iterations of the fractal and their complements. In (a), the triangle can be cut but its negative cannot. In (b), both the positive and negative shapes are cuttable. In (c), the negative can be cut but the positive cannot. It is easy to see that all further iterations behave like (c).

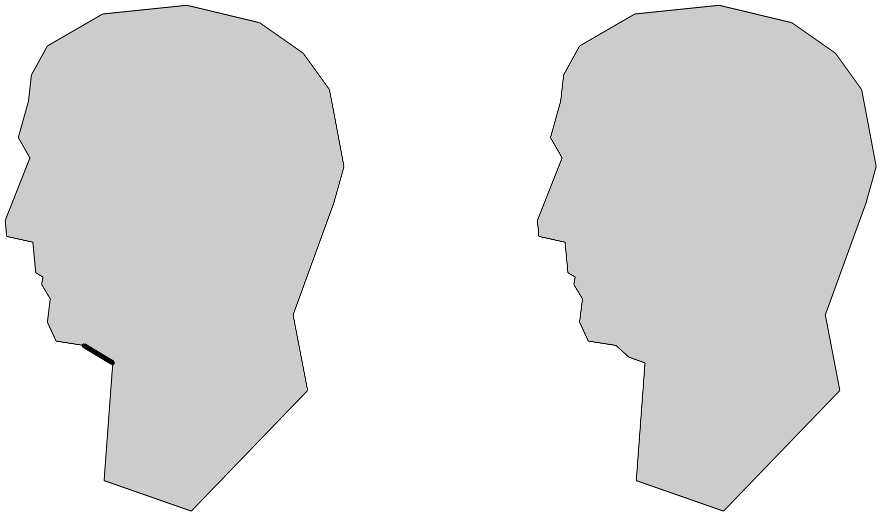


Figure 9: Profile of Trajan (Marcus Ulpius Traianus, 53–117 AD), Roman Emperor 98–117 AD.

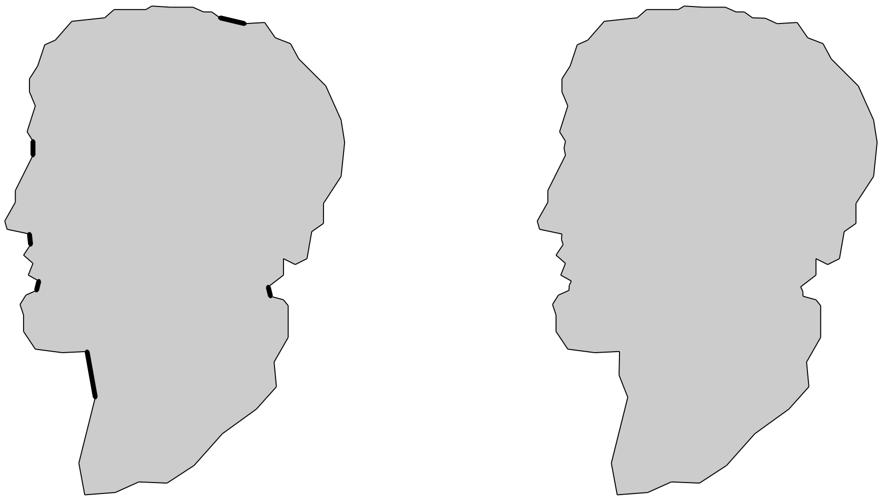


Figure 10: Profile of Archimedes (287–212 BC).

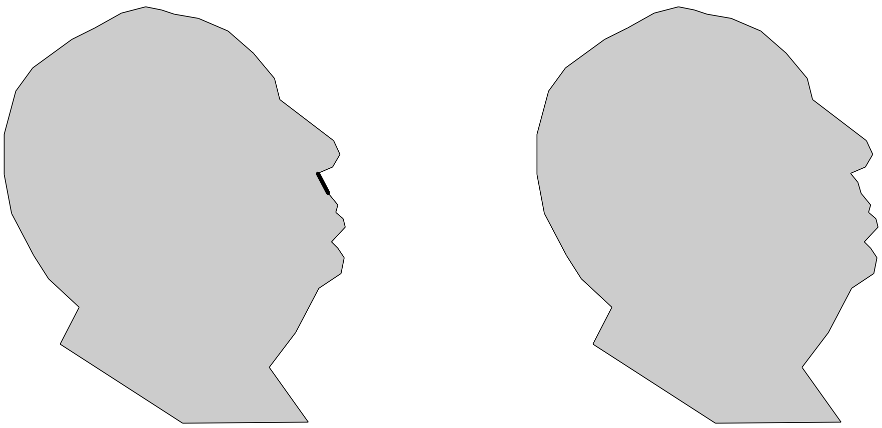


Figure 11: Profile of Alfred Hitchcock (1899–1980 AD).

3 Large Saws and Ray Cuttability

In this section we develop a linear-time algorithm to decide whether a given simple polygon can be cut out by circular saws of all sizes. This restriction prevents the saw from making fine maneuvering interior to the convex hull of the polygon, implying in particular that cuts cannot help to make other cuts: all possible cuts can be made from the initial configuration. To cut out a polygon is to cut along each of its edges, so for a polygon to be cuttable in this way, it is necessary and sufficient for each edge to be reachable along its line extension from outside the convex hull of the polygon. In other words, every edge of the polygon must be extendible to a ray (infinite in one direction) that does not intersect the polygon, and thus we call such polygons *ray-cuttable*.

Ray-shooting data structures [3, 4, 6] can test in $O(\log n)$ time whether one of the two ray extensions of a polygon edge does not intersect the polygon. Thus we immediately obtain an $O(n \log n)$-time algorithm for testing ray cuttability. The question addressed here is whether this can be improved to $O(n)$ time.

It is important to contrast ray cuttability with the well-studied notion of external visibility [1, 14], which can be tested in linear time. Any ray-cuttable polygon is *(weakly) externally visible* in the sense that every point on the polygon's boundary can see to infinity without obstruction by the polygon. But ray cuttability is a more stringent restriction, because it insists that the rays of visibility are aligned with corresponding edges of the polygon. For example, the polygon in Figure 2 is uncuttable by any circular saw, but it is externally visible.

We present two algorithms for testing ray cuttability in linear time. The first algorithm uses a connection between ray cuttability and external visibility via the shortest-path tree, which is conceptually very simple but relies heavily on known algorithms. The second algorithm is self-contained and simpler overall.

3.1 Pockets

At first glance, which way to extend a polygon edge to a ray seems to be an inherent binary choice. But in fact it is easy to derive in linear time which ray must be chosen for each edge.

The key tool is the convex hull of the polygon, which can be computed in linear time [9]. For example, each edge on the convex hull can be extended to a line, so either ray can be chosen. The remaining edges bound connected regions within the convex hull but exterior to the polygon, which are called *pockets*. A pocket is bounded by several polygon edges internal to the convex hull, and exactly one convex-hull edge called the *lid*. Each internal polygon edge must be extended to infinity without intersecting the polygon, so the extension must pass through the only possible exit from the pocket, the lid. By testing the slope of a polygon edge relative to its bounding lid, we can determine in constant time which ray extension of the edge cannot and which ray extension can escape the pocket through the lid.

Hence, in linear time we can compute the unique candidate for a collection of rays to cut out a polygon from its convex hull. It only remains to decide whether these cuts are valid, that is, whether they avoid cutting interior to the polygon.

For convenience of description, the algorithms view each pocket as if its lid were in *canonical orientation*: horizontal and above the polygon. Thus, each visibility ray points upward (the direction vector has a positive y component). We define the *angle* of a ray r, $\angle(r)$, to be the angle of the direction vector relative to the positive x axis. Hence, $\angle(r)$ is strictly between 0 (corresponding to pointing right) and π (corresponding to pointing left).

3.2 Relation to Shortest-Path Tree

Joseph Mitchell[4] observed the following connection between ray-cuttability and the shortest-path tree. The *shortest-path tree* [5, 11] of a vertex v in a simple polygon is the tree of all shortest paths from v to all other vertices of the polygon.

Suppose that the polygon is externally visible (which is necessary for ray-cuttability). Consider a pocket of the convex hull with lid (s, t), and re-orient the polygon to place the lid in canonical orientation. Consider

[4]Personal communication, November 2000 and February 2001.

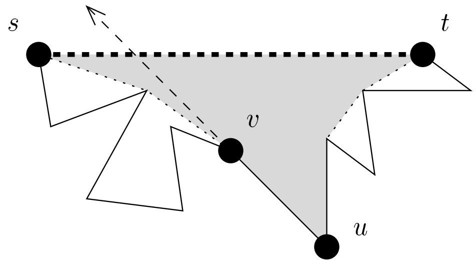


Figure 12: Connection between shortest paths and the ray extension of edge (u, v) reaching the pocket lid (u, v).

an edge (u, v), and suppose that v has higher y coordinate than u so that the candidate ray extension is from u to v.

Lemma 1 *If the interior of the polygon is to the left of edge (u, v), then the edge (u, v) is ray-cuttable precisely if the shortest path from s to u visits v just before u. Symmetrically, if the interior of the polygon is right of (u, v), then the edge is ray-cuttable precisely if the shortest path from t to u visits v just before u.*

The proof of this lemma is as follows; see Figure 12. By symmetry we concentrate on the first sentence of the lemma. Suppose first that edge (u, v) is ray-cuttable, so the ray extension from u to v hits the lid (s, t). Then the ray cuts the pocket into two pieces, one containing s and the other containing t. Once a path from s reaches the ray extension starting from v (which it must to reach u), it must follow exactly along the ray in order to be shortest, thereby passing by v just before reaching u.

Suppose next that the shortest path from s to u visits v just before u. Then consider both shortest paths, from s to u and from t to u. These paths bound a "funnel" rooted at u with base (s, t), which contains the ray extension of (u, v) because the polygon is externally visible. Therefore the ray extension reaches the lid (s, t) without obstruction. This concludes the proof of Lemma 1.

As a consequence, to test ray-cuttability of a polygon, we can first test external visibility in linear time [1, 14], then, for each pocket of the convex hull, compute the shortest-path tree in linear time [5, 11] and check whether each edge satisfies the condition in Lemma 1. Thus by combining known results we obtain a linear-time algorithm for testing ray cuttability, thereby proving Theorem 3. In the remainder of this section we give a self-contained algorithm.

3.3 Algorithm

Our second linear-time algorithm for deciding ray cuttability proceeds along the boundary of each pocket of the convex hull. During this process, the algorithm maintains a convex chain called the *constraint chain*, consisting of a sequence of rays. This chain represents half-plane constraints that subsequent polygon edges must satisfy. More precisely, each edge of the constraint chain is part of a visibility ray from a polygon edge to the lid, so if a polygon edge crosses the constraint chain (from the right), the polygon is not ray-cuttable. Furthermore, the constraint chain intends to capture all known constraints of this type; in order words, if the constraint chain is never crossed by a polygon edge (from the right), then no visibility ray should be blocked, so the polygon should be ray-cuttable. In fact, the algorithm traverses each pocket twice, once in each direction, in each case checking for polygon edges that cross *previously visited* visibility rays. By scanning in both directions, we check for all intersections between polygon edges and visibility rays.

To decide whether a given simple polygon is ray-cuttable:

- Compute the convex hull of the polygon.
- For each pocket of the convex hull, and for each orientation (s, t) of the lid:
  - Conceptually rotate and reflect the polygon so that (s, t) is horizontal and above the polygon, and so that s is left of t.
  - Initialize the constraint chain to the empty chain.
  - For each edge e of the pocket, in order along the polygon, starting incident to s and ending incident to t:
    - Let r be the ray extending the edge e upward (toward the lid); in particular, r contains e as a subset.
    - For each ray, r', in order along the constraint chain:
      1. Let e' be the polygon edge that r' extends.
      2. If r' intersects e,[5] return "no": the visibility ray r' is blocked by the polygon edge e.
      3. If r' intersects r:
         (a) If $\angle(r') \leq \angle(r)$, i.e., r hits r' from the right side of r':
            i. If r' hits r before r' hits the next ray along the constraint chain, then exit the innermost loop.
            ii. Otherwise, remove r' from the constraint chain: the constraint r' is antiquated by the stronger constraint r.
         (b) Otherwise, return "no": it is impossible to connect s to e' and e to t without blocking a visibility ray.
      4. Otherwise, r' does not intersect r:
         (a) If $\angle(r') \geq \angle(r)$, i.e., r ends up to the right of r', then remove r' from the constraint chain: the constraint r' is antiquated by the stronger constraint r.
         (b) Otherwise, return "no": it is impossible to connect s to e' and e to t without blocking a visibility ray.
    - Add r as the new first ray in the constraint chain.
- Return "yes": the polygon is ray-cuttable.

3.4 Correctness

There are two aspects to correctness of the algorithm: that it detects all intersections between polygon edges and visibility rays, and that whenever it returns "no" there is indeed such an intersection. Let us begin with the latter aspect. The simple case is when Step 2 returns "no." In this case the algorithm exhibits a polygon edge blocking a visibility ray, so certainly the polygon is not ray-cuttable.

When Step 3b returns "no" (see Figure 13, left), the ray r extending polygon edge e hits the left of the ray r' extending a previously visited polygon edge e'. Now e must be connected to t by polygon edges without crossing r'. Similarly, e' must be connected to s by polygon edges without crossing r. But r and the connection from e to t separate e' from s, contradicting simplicity of the polygon.

When Step 4b returns "no" (see Figure 13, right), we have that r does not hit the ray r'. As before, e must be connected to t by polygon edges without crossing r', and e' must be connected to s by polygon edges without crossing r. But this is impossible for a simple polygon because r ends up to the left of r'.

This argument establishes that the algorithm is correct when returning "no." It remains to show that the algorithm is correct when returning "yes." First we need to establish that testing for intersections between polygon edges and visibility rays is equivalent to testing for polygon edges crossing the constraint chain. This equivalence follows because the antiquation process in Steps 3(a)ii and 4a is "valid" in the sense that,

[5] More precisely, if r' intersects the relative interior of e, or r' passes through any endpoints of e in such a way that it then enters the interior of the polygon

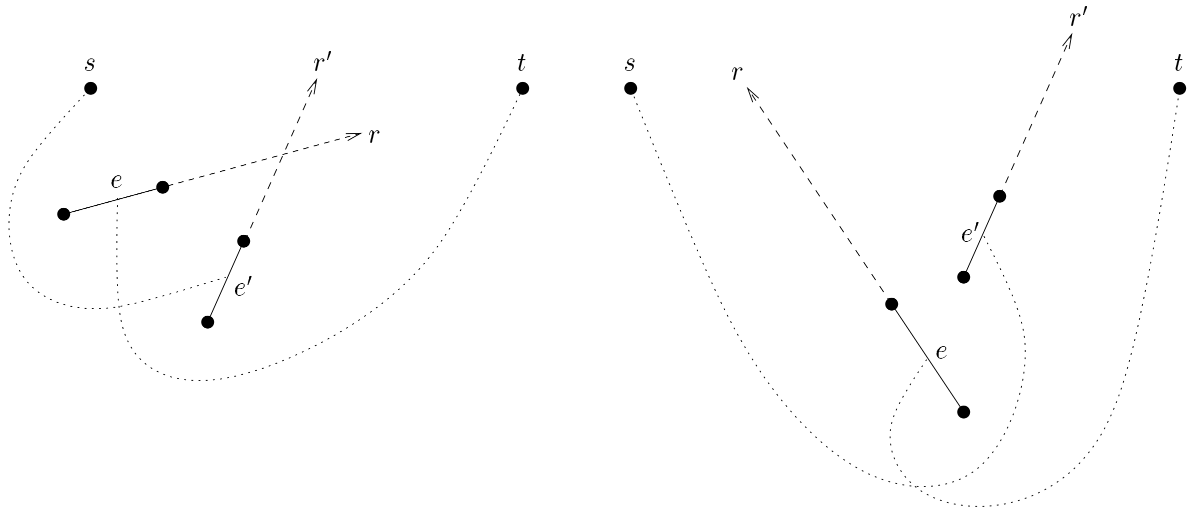


Figure 13: Justification behind Steps 3b and 4b: either a visibility ray is blocked, or the dotted connections cross as shown.

for the polygon boundary to cross the antiquated ray r' (from the right), it must first cross the ray r (from the right).

Finally, we must show that the algorithm detects when a polygon edge crosses the constraint chain. For a polygon edge e to intersect the constraint chain, its ray extension r must also intersect the constraint chain. The algorithm scans along the constraint chain for rays r' intersected by r, and checks these rays for intersection against e. The algorithm stops once it reaches a ray r' that hits r before r' hits the next ray along the constraint chain. Suppose that the algorithm does not return "no" for this edge e, that is, e does not intersect r' or any rays before r' on the constraint chain. Because the constraint chain is convex and all its rays have positive angle, no rays after r' on the constraint chain can intersect r, and hence they also cannot intersect e. Therefore the algorithm returns "yes" only if no polygon edges cross the constraint chain, concluding the proof of correctness.

3.5 Time Bound

A simple amortization shows that the algorithm runs in $O(n)$ time on a polygon with n vertices. The two outermost loops (over orientations of pockets and over edges of pockets, respectively) effectively loop over a subset of the edges of the polygon, so the time spent iterating them is linear. Whenever the innermost loop continues to another iteration, the algorithm first removes an element from the constraint chain. The ray extending a polygon edge is added to the constraint chain at most twice (once for each direction in which the pocket boundary is traversed), and so the ray can be removed at most twice. Hence the total time spent is linear.

4 Conclusion

Several interesting open problems remain:

1. What is the complexity of computing the largest circular saw that can cut a given polygon with no adjacent reflex vertices?

2. When the saw parameters are given, it is interesting to consider more general saws in which the cuts can be made with bounded curvature (such as scroll saws).[6] What is the complexity of deciding whether a given polygon (or, indeed, the interior of a simple closed curve) can be cut with a given saw?

3. What is the complexity of computing the optimal sequence of cuts to make a given polygon, where the metric is either the number of cuts, the total length of cuts, or the total motion of the saw? Can the approximation factors of 2.5 be improved?

4. What is the complexity of computing the optimal sequence of ray cuts to make a ray-cuttable polygon, where the metric is the total length of cuts? This problem was posed by Overmars and Welzl in 1985 [12], where they analyzed the case of cutting convex polygons with lines so that some metric is minimized, e.g., the total length of the intersections of the cuts with the current polygon. Many additional problems from [12] remain open.

5. What collections of nonoverlapping polygons in the plane can be simultaneously cut out by a circular saw? This problem is nontrivial when some of the polygons share edges.

6. How can a polygon be partitioned into the fewest subpolygons each of which is cuttable by a circular saw? Related to the previous question, how can a polygon be partitioned into the fewest subpolygons that can be simultaneously cut out by a circular saw?

7. Can a similar model be developed for cutting polyhedra in three dimensions? One natural extension is a *handsaw* (or a guillotine that can stop midway) which is modeled by an infinitely long rectangle that can only slice straight into the three-dimensional material.

Acknowledgments

We thank Therese Biedl for her observations described in Section 2.3, and Joseph Mitchell for his observations described in Section 3.2.

References

[1] David Avis and Godfried Toussaint. An optimal algorithm for determining the visibility of a polygon from an edge. *IEEE Transactions on Computers*, C-30(12):910–914, December 1981.

[2] Prosenjit Bose, Jurek Czyzowicz, Evangelos Kranakis, Danny Krizanc, and Dominic Lessard. Near-optimal partitioning of rectangles and prisms. In *Proceedings of the 11th Canadian Conference on Computational Geometry*, Vancouver, Canada, August 1999. `http://www.cs.ubc.ca/conferences/CCCG/elec_proc/fp45.ps.gz`.

[3] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

[4] Bernard Chazelle and Leonidas J. Guibas. Visibility and intersection problems in plane geometry. *Discrete and Computational Geometry*, 4:551–581, 1989.

[6] Note that it is important to fix the saw parameters unlike a circular saw: a scroll saw can cut out any polygon provided the projection of the saw blade is sufficiently short, i.e., the bound on curvature is sufficiently weak relative to the polygon.

[5] Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(2):209–233, 1987.

[6] John Hershberger and Subhash Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *Journal of Algorithms*, 18(3):403–431, May 1995.

[7] Jerzy W. Jaromczyk and Mirosław Kowaluk. The face-wise continuity in hot wire cutting of polyhedral sets. In *Proceedings of the 16th European Workshop on Computational Geometry*, pages 93–97, Eilat, Israel, March 2000.

[8] H. Martini and V. Soltan. Minimum convex partition of polygonal domains by guillotine cuts. *Discrete and Computational Geometry*, 19(2):291–304, 1998.

[9] Avraham A. Melkman. On-line construction of the convex hull of a simple polyline. *Information Processing Letters*, 25(1):11–12, April 1987.

[10] Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28(4):1298–1309, 1999.

[11] Joseph S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 15, pages 633–701. North-Holland, 2000.

[12] Mark H. Overmars and Emo Welzl. The complexity of cutting paper. In *Proceedings of the 1st Annual ACM Symposium on Computational Geometry*, pages 316–321, Baltimore, Maryland, June 1985.

[13] J. Pach and G. Tardos. Cutting glass. *Discrete and Computational Geometry*, 24:481–495, 2000.

[14] Godfried T. Toussaint and David Avis. On a convex hull algorithm for polygons and its application to triangulation problems. *Pattern Recognition*, 15(1):23–29, 1982.