ELSEVIER

www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Computational Geometry 25 (2003) 67–95

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

Sets of lines and cutting out polyhedral objects

Jerzy W. Jaromczyk [a,*], Mirosław Kowaluk [b,1]

[a] *Department of Computer Science, University of Kentucky, Lexington, KY 40506, USA*
[b] *Institute of Informatics, Warsaw University, Warsaw, Poland*

Received 12 July 2001; received in revised form 14 March 2002; accepted 20 June 2002

Communicated by H. Alt

Abstract

We study algorithmic questions related to cutting polyhedral shapes with a hot wire cutter. Such cutters are popular manufacturing tools for cutting expanded polystyrene (styrofoam) with a thin, moving heated wire. In particular, we study the question of polyhedral-wise continuity: *Can a given object be cut out without disconnecting and then reattaching the wire*? In an abstract setting this question translates to properties of sets of lines and segments and therefore becomes suitable for computational geometry techniques. On the combinatorial and algorithmic levels the results and methods are related to two problems: (1) given a set $F = \{f_1, \ldots, f_k\}$ of polygons and a polygon f, decide if there is a subset of lines in the set of lines not stabbing F that cover f; (2) construct the connectivity graph for free movements of lines that maintain contact with the polyhedral shape. Problem (1) is solved with the dual projection and arrangements of convex and concave x-monotone curves. Problem (2) can be solved with a combination of the skewed projections [6] and hyperbola arrangements proposed by McKenna and O'Rourke [11]. We provide an $O(n^5)$ algorithm for constructing a cutting path, if it exists. The complexity of the algorithm is determined by the $O(n^4)$ size of the connectivity graph and the cost of solving (2).

Keywords: Polyhedral objects; Dual projection; Skewed projection; Manufacturing; Hot-wire cutting

1. Introduction

With the tools of computational geometry we study algorithmic problems related to manufacturing applications of hot wire cutters. Such a cutter is a thin, electrically heated wire (geometrically, a line) that slices through a block of styrofoam while cutting out a desired object (in this study, a polyhedral shape).

* Corresponding author.
E-mail address: jurek@cs.engr.uky.edu (J.W. Jaromczyk).

[1] Partially supported by grant KBN 8T11C03915. This research was partially supported by the Center for Computational Sciences of the University of Kentucky.

0925-7721/02/$ – see front matter
doi:10.1016/S0925-7721(02)00131-1

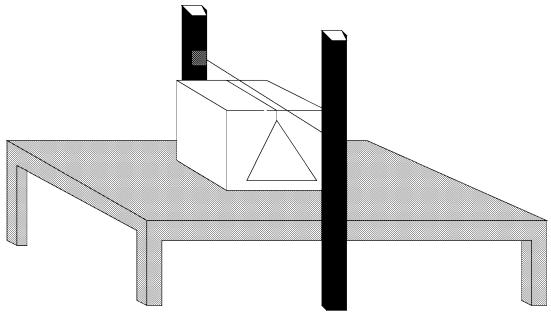

Fig. 1. A schematic view of a cutter. A cutter can be either operated manually or be computer-controlled. The picture shows the wire in park after cutting out a prism with a triangular base from a block of styrofoam.

The wire is mounted on a mobile frame (gantry) that moves above the stationary block of foam during the cutting process. The block sits on a table. A schematic view of such a cutter is shown in Fig. 1.

More advanced models are equipped with a turntable so the block of styrofoam can be rotated, which allows for cutting shapes with rotational symmetry. The two ends of the wire are attached to the opposite sides of the frame and they can move vertically on the frame. Foam hot wire cutters are used for a variety of applications ranging from serious manufacturing jobs to crafts and hobbies. Products that can be manufactured with hot wire cutters include reverse molds for the concrete industry, packaging projects, signs and retail graphics, props and exhibit stands, pre-production prototypes, packaging molds, architectural and building components, foam props for retail designs, movie sets and stage props, pipe insulation, hobby items (such as railroad models), and craft projects. These products can be either cut-out from a block of styrofoam or assembled from a number of separately cut-out elements.

In spite of a long list of capabilities of the hot-wire cutters, it is clear that there are shapes that cannot be manufactured with this method, at least not without assembling them from separately cut modules. Examples include shapes with cavities. The objective of this paper is to study both the shapes that can be produced with foam cutters and the algorithmic problems related to wire cutting. Specifically, we look at the following questions:

- 1–1 cutting—can the object be cut out in such a way that each point of its faces is touched exactly once (e.g., consider a square with a small cube attached to its center)?
- face-wise continuity—can each face be cut out without moving the wire away from the plane containing this face and then returning to this plane (e.g., consider a triangle with two small cubes attached to it)?
- polyhedron-wise continuity—can the polyhedral set be cut out without disconnecting and reattaching the wire (e.g., consider a cube with a hole through the middle)?
- feasibility—what shapes can be cut out?
- cutting-path description—can the ends of the wire move along any curve, or piece-wise linear curves only?

The technological rationale behind 1–1 cutting is to avoid overheating of the cut surface, behind face-wise continuous cutting to avoid inaccuracies in cutting the surface, and behind polyhedron-wise continuity to allow the wire to cut out the shape without changing the tool.

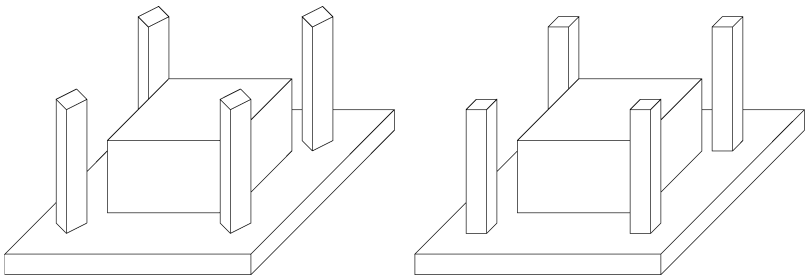

Fig. 2. These upside-down tables appear similar but only the one on the right can be cut out with a wire.

Preliminary results addressing the above questions were presented in two conference papers [7] and [8].

Clearly, from a manufacturing point of view we need not only to answer the feasibility problem but also to construct a description of the cutting path for the wire (via moving its endpoints) and develop a framework for constructing and describing cutting paths. It is possible to see that there are two major obstacles to this type of cutting. The first difficulty may be posed by the connectivity structure (for example a cube with a hole). The second arises from the possibly complex relations between facets of polyhedral sets. This issue is well-illustrated in Fig. 2. Although the shapes seem to be similar, only the table on the right side can be cut out.

Let us formalize our problem to be able to study it with the tools of computational geometry. Let P be an open polyhedral shape (later simply called a polyhedron) understood for the purpose of this paper as a 3-dimensional solid bounded by planar facets, and assume that its closure $\overline{P}$ has a total of n faces, edges and vertices. For technical reasons it is convenient to consider open polyhedra rather than closed ones (think that the wire is peeling off the boundary of the object). Edges of P are defined as the corresponding edges of $\overline{P}$. It is convenient to define faces of P to be the interiors of the corresponding faces in $\overline{P}$.

A path p of lines is a continuous mapping from an interval $[a, b]$ into the space of lines in R^3. Such a path can be described with a pair of functions representing the location (in some coordinate system) of two selected points of the line (wire) as a function of time.

The concepts of paths, subpaths and continuity can be formalized further by introducing a natural topology in the space of lines represented with Plücker coordinates following [1,15]. This formalization led to a number of strong results for lines in three dimensional spaces, including some relevant to our study of lines moving among obstacles. In particular, [15] mentions applications related to modeling radiation or light beams.

Paths of lines that do not intersect P (P is open) and intersect the boundary of $\overline{P}$ are called *admissible.* The rationale behind this definition is that in this paper we are planning for paths that always stay in contact with the closure $\overline{P}$ of P (compliant motions). Note that admissible paths can contain lines that are tangent to two skewed edges of P and are not necessarily in contact with a face.

We say that a path of lines cuts out P if (1) it is admissible and (2) the union of the intersections of the lines in the path with $\overline{P}$ is equal to the boundary of $\overline{P}$.

We further say that a line (in the path) *cuts* a facet f, defined as the interior of the corresponding facet in $\overline{P}$, if it is admissible, lies in the plane supporting f and has in this plane a non-empty intersection with f. A path p *cuts out* f if it is admissible and the intersection of lines in p with the closure of P contains f. Note that p that cuts out f can cut or cut out facets of P that are coplanar with f. Also f can

be cut out with a number of subpaths $p_1, \ldots, p_m$ of p, each cutting out only a portion of f. This leads to the next definition. We say that f is cut out with the face-wise continuity if there is a continuous subpath p' of p that cuts out f and all lines in p' cut f. A face-wise continuous cutting path p for f such that each point of f belongs to exactly one line of p is called a 1–1 cutting path.

Finally, we say that subpaths $p_1, \ldots, p_m$ cutting out facets of P can be *blended* into a cut-out path for P if there exists a sequence $q_1, \ldots, q_{m-1}$ of admissible paths for P such that $p = p_1 q_1 p_2 q_2 \ldots q_{m-1} p_m$ is a continuous path that cuts out P.

The problem of wire cutting with polyhedron-wise continuity can be now defined as: "Given an open polyhedral set P find a (continuous) cut-out path p for P, if it exists".

2. Overview of results and methods

A solution to the problem of planning a cutting path requires a schedule for moving the wire between the faces of P to cut them out. Individual faces may require more than one visit of the wire to avoid obstacles during cutting (consider as an example the task of cutting out the bottom of a table with four legs).

Our solution has, therefore, two components: (a) build a graph of admissible movements for the cutting wire and (b) verify if there is a path in this graph that visits all the faces in such a way that allows them to be cut out, in portions if necessary.

The first component is based on analyzing the structure of the space of free lines (admissible movements for the cutting wire). We build the connectivity graph (the lattice structure) of this space, from whose connected components and edges we can derive answers to our questions. This general approach of constructing free-spaces, introduced by Lozano-Pérez [9], is commonly used in robotics for robot motion planning, see [17]. The connectivity-graph technique was introduced and used in a series of papers by Schwartz and Sharir, see [16]. Questions related to collision-free translations in three dimensions for lines and polyhedra have been studied in the computational geometry literature. In particular, an $O(n)$ algorithm for translating convex polyhedra with n vertices is given in [12]. Simple polyhedra with n edges each can be translated in $O(n^4 \log n)$, as presented in [13]. Pellegrini [14] solves the problem of separating a set of n blue lines from a set of m red lines in $O(n^{3/4}m^{3/4+\varepsilon} + m^{1+\varepsilon} + n^{1+\varepsilon})$ where ε is an arbitrary small positive number and the asymptotic constant may depend on ε. Our problem is different and pertains to the feasibility of moving a line (wire) along faces of a given polyhedron—a compliant motion.

Before presenting the connectivity-graph for the wire motion, we will discuss cutting out individual faces, consecutively focusing on 1–1 cutting, face-wise continuous cutting and the feasibility of cutting. As we will see, the problems involve lines that avoid certain polygons (obstacles), and the dual transformation is used as the major technique. Specifically, we will analyze lines in $plane(f)$, the plane supporting face f, that intersect f without intersecting (the interiors of) $f_1, \ldots, f_k$. It is convenient to look at such lines using the dual projection that maps points into lines and vice versa using the following transformation: $(a, b) \to y = ax - b$, see [3,4]. The dual projection depends on the choice of the coordinate system and is not defined for lines parallel to the OY axis.

3. Cutting out single faces

This section addresses the problem of cutting out single faces: face-wise continuous cutting, 1–1 cutting, and arbitrary cutting.

P is bounded by planar facets. Let f be a facet and let $plane(f)$ be the supporting plane for f. Consider the intersection $plane(f) \cap \overline{P}$. This intersection is a set of polygons that include f and perhaps some other facets of P as well as cross-sections $f_1, \ldots, f_k$ of P with $plane(f)$. Polygons $f_1, \ldots, f_k$ are potential obstacles for the wire cutter. Since a line intersects f_i if and only if it intersects $conv(f_i)$, in the further discussion we can assume that all of $f_1, \ldots, f_k$ are convex polygons. Additionally, we will allow the cutting line to cut the boundaries of $f_1, \ldots, f_k$ so when needed $f_1, \ldots, f_k$ will be considered without their boundaries. For each face f its obstacles can be found in $O(n \log n)$ time assuming that a set of sequences of edges of the faces of the polyhedral shape is available. The process can be organized as follows: for each face f_p we find the intersections of the edges of this face with $plane(f)$. If there are intersection points, we connect them, based on the list of edges of f_p, to form segments that correspond to the intersection of f and f_p. The cost of finding such segments for all faces is $O(n)$. Knowing that these segments form disjoint cross-sections on f we can sweep the plane to identify segments with joint endpoints. The cost of this step is $O(n \log n)$. Based on the list of edges in the faces, we can group such connected segments into polygons in $O(n \log n)$ total time.

Since most of the results of this section are based on the dual projection, let us start with a number of its properties.

3.1. Dual projections for cutting lines, faces and obstacles

It is well known that the dual of a line segment e is a double wedge, see [3,4]. The double wedge is that part of R^2 that is bounded by the two dual lines of the endpoints of e and does not contain a vertical line parallel to OY.

The dual $S(f)$ of a convex polygon f with edges $e_1, \ldots, e_p$, which for the sake of this section is considered as a closed polygon in R^2, is the union of the double wedges of the edges of f, i.e., $S(f) = \bigcup_{i=1}^{p} S(e_i)$.

$S(f)$ does not contain any line parallel to OY, and each vertical line is intersected once by each boundary. Therefore it makes sense to talk about the upper boundary and the lower boundary of $S(f)$; they will be denoted by $U(S(f))$ and $L(S(f))$. The image of a convex polygon in the dual projection is illustrated in Fig. 3.

There are a number of elementary properties of $S(f)$, some of them well-known, that will be useful in the sequel.

Lemma 3.1. *The boundary of $S(f)$ consists of two polylines $U(S(f))$ and $L(S(f))$ whose angles are reflex.*

Proof. $S(f)$ is the union of double wedges whose boundaries are piecewise linear. Each vertex of the boundary is either the vertex of a double wedge, and thus reflex, or it is the intersection of halflines from the boundaries of some double wedges and their union forms a reflex angle at this intersection. □

Due to their shape dual projections of convex polygons will be called *butterflies*.

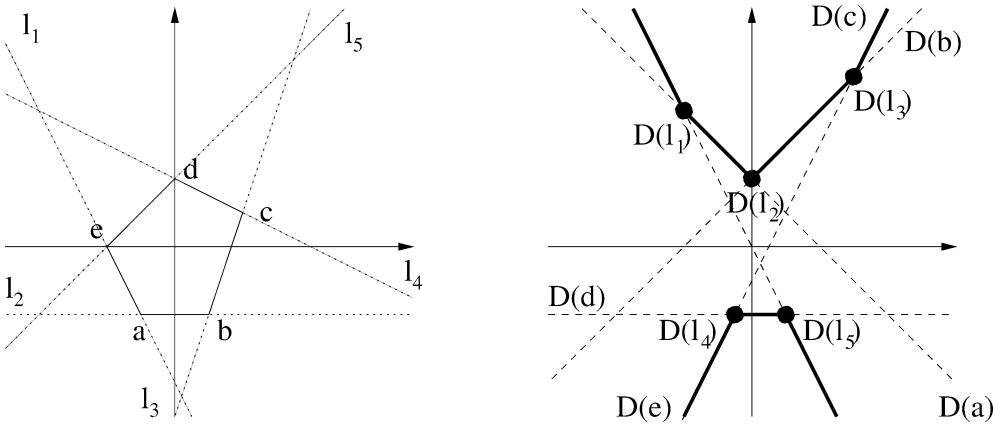

Fig. 3. A convex polygon and its dual image with edges on the upper and lower boundaries corresponding to the vertices of the polygon in primal space.

Lemma 3.2. (1) *Each line supporting a segment on the boundary of* $S(f)$ *is contained in* $S(f)$. (2) *Each line tangent to the boundary of* $S(f)$ *at vertex* v *is contained in* $S(f)$. (3) *Each segment connecting a point in* $U(S(f))$ *with a point in* $L(S(f))$ *is contained in* $S(f)$.

Proof. (1) The line supporting a segment of the boundary belongs to the double wedge of some edge e of f and as such is contained in the union of the double wedges; this proves the first statement.

(2) v is the dual image of the supporting line for some edges of f. Each line tangent to the boundary of $S(f)$ at v is in the double wedge defined by lines meeting at v and as such it is the dual image of a point in e. Since lines intersecting the interior of e intersect f, the dual lines of points in the interior of e are in $S(f)$.

(3) Statement (2) of this lemma implies that a line can intersect the boundary of $S(f)$ at most twice. Assume, for example, that l intersects $U(S(f))$ in two points (it cannot intersect in more because all the vertices on the boundary are reflex). Let l' be parallel to l and tangent to a vertex in $U(S(f))$. l' is contained in $S(f)$ and as such it separates l from $L(S(f))$. Consequently l cannot intersect $L(S(f))$. Therefore the segment connecting a pair of points in $U(S(f))$ and $L(S(f))$ does not intersect the boundary in any other point and thus is entirely included in $S(f)$. □

Let $f_1, \ldots, f_k$ be convex polygons with a total of $O(n)$ vertices. Denote by $S(f_1, \ldots, f_k)$ the union $\bigcup_{i=1}^{k} S(f_i)$. When $f_1, \ldots, f_k$ are obstacles then, as mentioned earlier, we take the duals of their interiors.

The following holds:

Lemma 3.3. (1) $R^2 \setminus S(f_1, \ldots, f_k)$ *is a union of convex polygons* (*possibly unbounded*). (2) *The combinatorial complexity of* $R^2 \setminus S(f_1, \ldots, f_k)$ *is* $O(n^2)$, *and the number of polygons in this difference is* $O(n^2)$.

Proof. (1) By virtue of Lemma 3.1 the boundary of each component of the difference is piecewise linear with the angle at each vertex being the complement of a reflex angle, and thus convex.

(2) The combinatorial complexity is not larger than that of an arrangement of $O(n)$ lines. In the primal space each polygon in the difference corresponds to a set of lines not cutting $f_1, \ldots, f_k$. Note that if

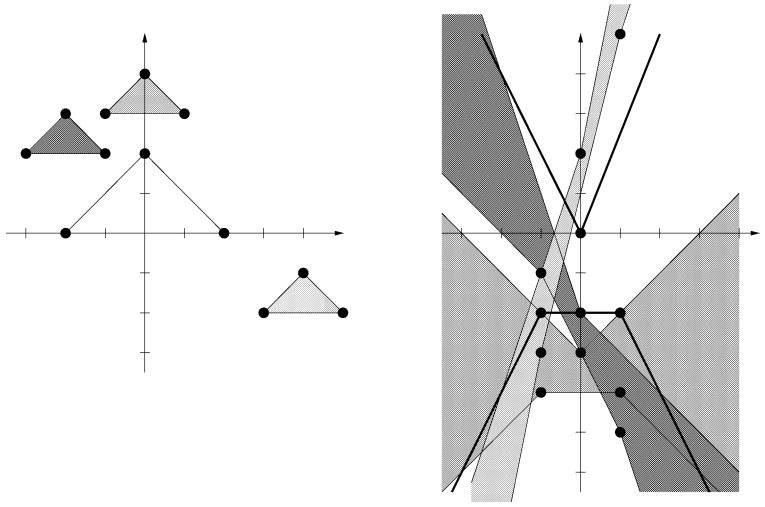

Fig. 4. A set of polygons and their corresponding butterflies. Gray polygons and butterflies correspond to obstacles.

obstacles $f_1, \ldots, f_k$ are arranged on a circle then each set of all of the lines passing through two gaps between the obstacles corresponds to a polygon in the difference. Clearly, for such an arrangement of the obstacles the number of polygons is proportional to n^2. □

Because of our interest in cutting out f the set $D(f, f_1, \ldots, f_k) = S(f) \setminus S(f_1, \ldots, f_k)$ is particularly important for our considerations. Fig. 4 illustrates the above concepts. The difference of the butterflies for the white and shaded triangles consists of four connected components (the white areas between the upper and lower solid lines).

As mentioned earlier, vertical lines do not have dual projections. To include them, we can use the projective plane (see [2]) and consider its subset consisting of all the lines that are contained in $D(f, f_1, \ldots, f_k)$. One of the ways to look at such a space is to identify the antipodal ends of the lines with the points at $-\infty$ and $+\infty$. This point at infinity corresponds to a vertical line. This affects the notion of continuity of curves contained in $D(f, f_1, \ldots, f_k)$. In particular, continuous are those curves that consist of two pieces with one piece having endpoints at $-\infty$ and a point on the boundary and the other piece having endpoints on the boundary and at $+\infty$ (the unbounded endpoints meet at infinity).

For our purposes, it is important to know that with a suitable choice of the coordinate system we can transform curves passing through infinity into curves joining the opposite boundaries of $S(f)$. For example, assume that in a coordinate system XOY both the endpoints of a curve passing through infinity are in $U(S(f))$ and their extremal x-coordinates are x_1 and x_2. Select a new coordinate system with OY' having slope $x_m = \frac{x_1+x_2}{2}$. In this new coordinate system lines with slope x_m become vertical and are mapped into a point at infinity. Also, the unbounded portions of the boundaries of $S(f)$ swap positions after such a change of the system so that curves passing through infinity that connect points both either in $U(S(f))$ or in $L(S(f))$ change to curves joining a pair of points from the opposite boundaries $U(S(f))$ and $L(S(f))$. This observation will be useful in selecting convenient coordinate systems and will allow us not to have to worry about vertical positions for the cutting line.

Fig. 5 illustrates the dependence of $D(f, f_1, \ldots, f_k)$ on the choice of the coordinate system. It also shows a schematic view of the fragment of the projective space for the butterflies and the cuts (the value

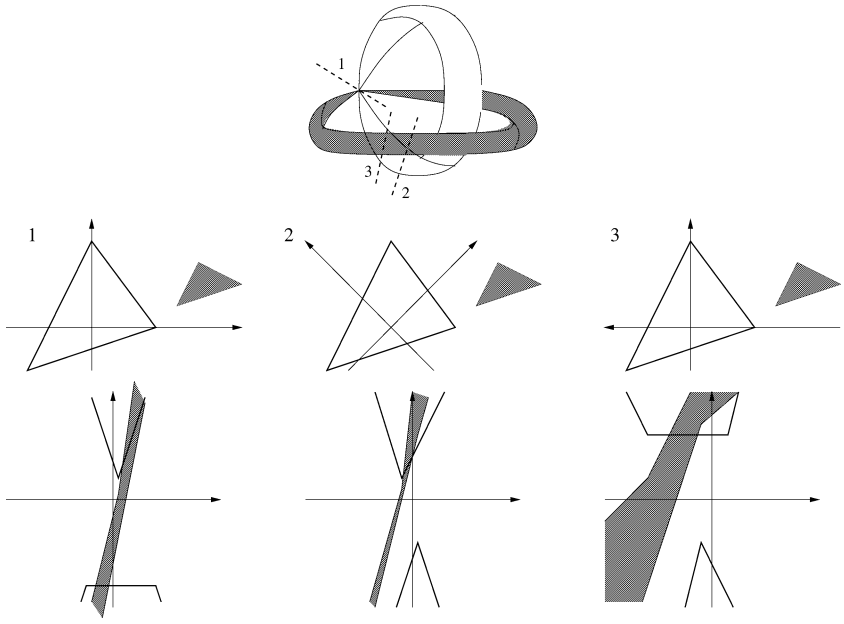

Fig. 5. A face (white), an obstacle (gray) and their dual images in three different coordinate systems. The upper picture is a schematic view of the dual space as a projective space with lines at infinity identified. Segments represent places where the space can be cut to represent standard coordinate systems as depicted in the second row of images.

of x) that allow the flattening of the fragment in order to present it in a standard R^2 coordinate system. Since x coordinates in the dual space correspond to slopes in the primal space, the choice of x determines the direction for OY. Note that Lemma 3.2 implies that the opposite unbounded segments of $U(S(f))$ and $L(S(f))$ are pairwise colinear.

3.2. Face-wise continuous cutting out

Consider a convex polygon f and convex obstacles $f_1, \ldots, f_k$. In this section we are interested in a face-wise continuous cut-out for f. Considering convex f seems to be a restriction. However, we can work with a Boolean model where a facet is represented as the difference of a convex polygon and obstacles or "holes". To explain the idea let us use color coding: *white* for what we cut out, *black* for obstacles, and *gray* for portions that are not in f but are not obstacles. If we start with a *white* convex polygon f and place *black* obstacles and *gray* regions, then only the remaining *white* portion of f must be cut out. We cannot penetrate the *black* regions, and we can but do not need to cut the *gray* regions. In this model a convex f supports the facet of P that is viewed as the difference between a convex polygon and polygons corresponding to parts of P that protrude from f. Such obstacles play a different role than obstacles disjoint with f.

Although all the following results can be stated for *white, black* and *gray* regions, for the sake of simplicity we just consider convex faces (*white*) and obstacles (*black*).

We have the following straightforward result:

Lemma 3.4. *The dual image of a line that cuts f (without penetrating the obstacles) is a point in $D(f, f_1, \ldots, f_k)$. Each point in $D(f, f_1, \ldots, f_k)$ is the dual image of some line that cuts f.*

Proof. Follows directly from the definitions of cutting lines and $D(f, f_1, \ldots, f_k)$. Points at infinity correspond to vertical lines. □

Lemma 3.5. *If there exists a line l in $S(f)$ that is not contained in any $S(f_i)$, $i = 1, \ldots, k$, but is contained in $S(f_1, \ldots, f_k)$, then f cannot be cut out.*

Proof. l corresponds to a point z in f but not in any obstacle. Each point in l corresponds to a line l' passing through z and vice versa. Since each point in l is in some $S(f_i)$, the corresponding l' intersects f_i. Consequently point z cannot be cut out. □

Now the face-wise continuous cutting can be characterized as follows:

Theorem 3.1. *f can be face-wise continuously cut out if and only if there is a continuous curve ρ in $D(f, f_1, \ldots, f_k)$ that intersects all lines that are contained in $S(f)$ and are not contained in any $S(f_i)$, $i = 1, \ldots, k$. The curve ρ is contained in one connected component of $D(f, f_1, \ldots, f_k)$.*

Proof. Note that by Lemma 3.5 if ρ intersects all lines in $S(f)$ but not in $S(f_i)$ for $i = 1, \ldots, k$, then there are no lines in $S(f)$ that are contained in $S(f_1, \ldots, f_k)$.

Points in $D(f, f_1, \ldots, f_k)$ correspond to lines that cut f (Lemma 3.4). Any curve in $D(f, f_1, \ldots, f_k)$ induces in the primal space a continuous path of lines that cut (but do not necessarily cut out) f and do not intersect any of $f_1, \ldots, f_k$. Since ρ intersects each line in $S(f)$ that is not in $S(f_i)$, $i = 1, \ldots, k$, the corresponding path of lines in the primal space cuts each point in the facet represented by f, and the facet is cut out.

On the other hand, if such a path does not exist then every continuous path misses some point in f; consequently f cannot be cut out in a face-wise continuous fashion.

Since ρ is continuous it must be contained in one connected component of $D(f, f_1, \ldots, f_k)$. □

A special type of cutting can be obtained when ρ is limited to piecewise-linear curves.

We have a simple lemma that stems directly from the properties of the dual projection:

Lemma 3.6. (1) *A vertical segment in $D(f, f_1, \ldots, f_k)$ corresponds to a family of parallel lines that cut f.* (2) *A nonvertical segment in $D(f, f_1, \ldots, f_n)$ corresponds to a pencil (a double wedge) of lines cutting f. The center of this wedge is in f if and only if the line supporting this segment is contained in $S(f)$.*

Proof. Let us show part (1). Points (x, y) corresponding to a vertical (parallel to OY) segment can be parametrized with $x = a$, $y = -b + g(t)$, where $g(t)$ describes the position of the point on the segment. In real space it corresponds to a family of parallel lines $x = ax + b - g(t)$ that cut f without intersecting $f_1, \ldots, f_k$. Part (2) can be showed similarly. □

The lemma is illustrated in Fig. 5. Different segments in $D(f, f_1, \ldots, f_k)$, which is equal to $S(f)$ as there are no obstacles in this case, correspond to different cuts for f that are either translations or rotations

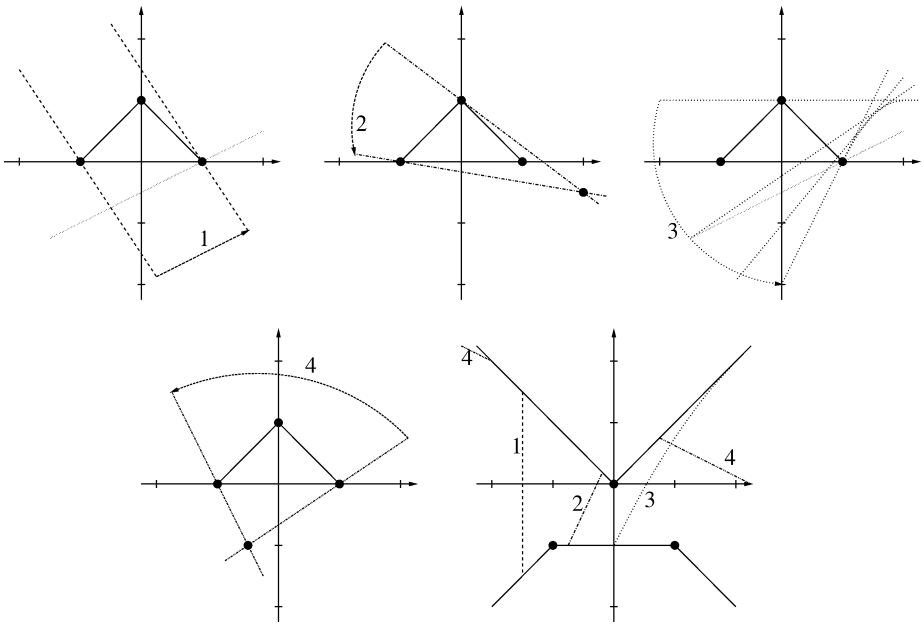

Fig. 6. An example ρ and the corresponding cutting paths. Depending on the shape of ρ (see the last image in the sequence), the cuts are (1) translations, (2) rotations or (3) more complicated motions (if ρ is not a line segment). Note that the line forming path (4) crosses infinity (unbounded components in the butterfly) and intersects $U(f(S))$ twice. The corresponding cut is a rotation that also includes a vertical position of the cutting line; see also Fig. 5.

of the cutting line. Since the segments join the boundaries of $S(f)$ and thus are intersected by all the lines in $S(f)$, the paths cut out f by virtue of Theorem 3.1. The segment labeled 4 consists of two halflines and exemplifies a continuous curve in the projective space as discussed above. The corresponding cutting path is a rotation of the cutting line that passes through a direction parallel to OY.

We will see in the next subsection that not all faces can be cut out in a face-wise continuous fashion. Then we will discuss 1–1 cutting out, which is a special type of face-wise continuous cutting. Finally, we will present an algorithm to decide if a face can be cut out with a face-wise continuous cut.

3.3. Faces and obstacles

In this section we analyze cuts for specific arrangements and shapes of faces and obstacles.

Lemma 3.7. *Let f be a polygon and let an obstacle f_1 be placed in such a way that it overlaps or is contained in f. Then f cannot be* 1–1 *cut out.*

Proof. Consider consecutive vertices A, B, C of f_1 such that the vertex B of the angle $\angle ABC$ is inside f.

We can assume that $\angle ABC$ is smaller than π (otherwise f clearly cannot be cut out and in particular cannot be 1–1 cut out). Assume now that f can be 1–1 cut out. Consider a sufficiently small $\triangle BB'C'$ with B' on the line l supporting AB (B between A and B'), and C' in BC. Line l is one of the cutting lines; otherwise points in f close to AB could not be cut. Now every line l' that cuts points inside

$\triangle BB'C'$ must intersect the interiors of two edges of this triangle. To avoid f_1 one of them must be BB', but points in this segment are cut by both l and l', a contradiction. □

If there is one convex obstacle f_1 and it is in the interior of f, then f can be face-wise continuously cut out (from Lemma 3.7 we know that it cannot be 1–1 cut out). A cutting line that rotates about f_1 maintaining contact with f_1 and touching the consecutive edges of f_1 provides a face-wise continuous cutting out for f.

In contrast, we have the following.

Lemma 3.8. *If there are two convex obstacles f_1, f_2 that are in the interior of f, then f cannot be cut out in a face-wise continuous fashion.*

Proof. This can be seen by analyzing the structure of $D(f, f_1, f_2)$ which has two connected components. There are lines in $S(f)$ and not in $S(f_1)$ or $S(f_2)$ that intersect one but not the other component. By Theorem 3.1 there is no continuous cutting path. □

We can consider other combinations of *white* and *black* polygons and use similar methods to decide about their properties regarding cutting out. We can also add *gray* polygons to mark regions that do not belong to f and thus do not need to be but can be cut. Such regions allow us to analyze non-convex f with the techniques developed in the previous section. Fig. 7 illustrates a face, an obstacle, a portion that does not belong to f but is not an obstacle and their corresponding dual projections. Note that face f can be cut out by rotating the wire around the obstacle.

Another interesting case that can also be analyzed with the techniques developed in the previous sections is cutting out coplanar faces. Fig. 8 illustrates the problem. In a 1–1 cut we cut out all the faces at once, or each of them individually, if possible. The dual projection for this case is more complicated than for a single face.

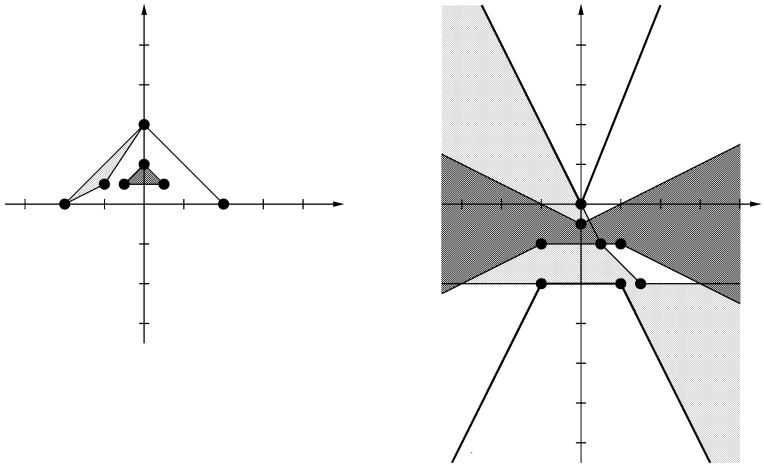

Fig. 7. A non-convex face and its butterfly. The light gray represents the cavity, the dark grey represents an obstacle inside the (white) face.

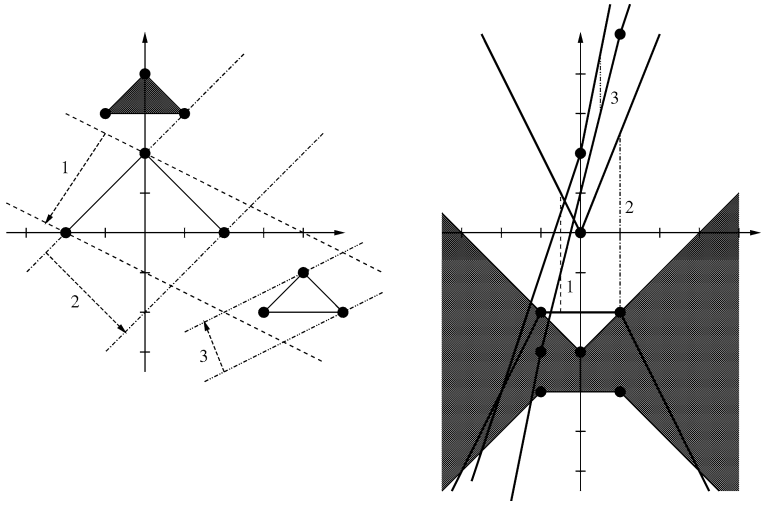

Fig. 8. Cutting two coplanar faces with one common cut (1) and with separate cuts (2) and (3).

3.4. 1–1 cutting out

We know from the previous section that there are configurations of faces and obstacles such that a face can be face-wise continuously cut out but not 1–1 cut out. This section presents an algorithm to decide if a 1–1 cutting-out path exists.

An observation, which we state without proof, is that the angle swept by a 1–1 cutting-out line is smaller than π. This implies that there is a coordinate system in which this 1–1 cutting path does not pass through the vertical line direction. Although we do not know such a coordinate system, because of the limit π on the swept angle we will be able to find two coordinate systems such that one of them has the desired property. In the sequel we always look at two such systems and focus on curves in $D(f, f_1, \ldots, f_k)$ that join opposite boundaries in $S(f)$. In each individual system we will look at connected components of $D(f, f_1, \ldots, f_k)$ in R^2, and not in the projective space discussed earlier.

Lemma 3.9. *A connected component D of $D(f, f_1, \ldots, f_k)$ that has a non-empty intersection with at most one boundary of $S(f)$ cannot contain a* 1–1 *cutting path.*

Proof. Note that 1–1 cut-outs for f start and end at positions tangent to f viewed as a polygon in R^2. This means that the corresponding cutting paths start and end in $L(S(f)) \cup U(S(f))$.

Assume that D shares a piece of boundary with $L(S(f))$ but not with $U(S(f))$. That is, cutting paths contained in D have both endpoints in $L(S(f))$. Assume that ρ is a 1–1 cutting path. Take a line l that supports a piece of $L(S(f))$ that is also in D. By Lemma 3.2, l is in $S(f)$, and a small rotation of it about a vertex of $L(S(f))$ on l yields a line $l(\varepsilon)$ that is also in $S(f)$ and intersects the interior of D. Let $l'(\varepsilon)$ and $l''(\varepsilon)$ be lines parallel to $l(\varepsilon)$, in $S(f)$, and intersecting D (assume that $l'(\varepsilon)$ is the line that separates $l''(\varepsilon)$ from $L(S(f))$. As a 1–1 cutting path, ρ must intersect both $l'(\varepsilon)$ and $l''(\varepsilon)$. Since both end-points of ρ are in $L(S(f))$ and $l'(\varepsilon)$ separates $l''(\varepsilon)$ from $L(S(f))$, ρ intersects $l'(\varepsilon)$ twice. This contradicts the assumption that ρ is a 1–1 cutting path. □

On the other hand we have the following:

Lemma 3.10. *A connected component D of $D(f, f_1, \ldots, f_k)$ that has a non-empty intersection with both $U(S(f))$ and $L(S(f))$, boundaries of $S(f)$, contains a* 1–1 *cutting path.*

Proof. By Lemma 3.2 and Lemma 3.3 a segment connecting points in D on the boundaries $U(S(f))$ and $L(S(f))$ is contained in D. This segment crosses $S(f)$ and therefore is intersected by all lines in $S(f)$. Thus, by Theorem 3.1, it induces a face-wise continuous cutting-out path. Since the segment intersects each line in $S(f)$ exactly once, the cutting path is 1–1. □

Lemma 3.10 implies a simple $O(n^2 \log n)$ algorithm to verify if a convex f can be cut out in a 1–1 fashion. Construct $D(f, f_1, \ldots, f_k)$. Then for each of the connected components in $D(f, f_1, \ldots, f_k)$, find if it has a nonempty intersection with both $U(S(f))$ and $L(S(f))$. Since the total number of components and the combinatorial complexity of $D(f, f_1, \ldots, f_k)$ is $O(n^2)$, the cost of this approach is $O(n^2 \log n)$.

However, there is a better algorithm that is suggested by the following observation.

Lemma 3.11. *There are only* $O(n)$ *components in $D(f, f_1, \ldots, f_k)$ that can share boundaries with both $U(S(f))$ and $L(S(f))$.*

Proof. Butterflies corresponding to the obstacles intersect each of the boundaries of $S(f)$ in $O(n)$ points. Two pieces of the boundary in $U(S(f))$ that belong to two disjoint components in $D(f, f_1, \ldots, f_k)$ are disjoint and have ends in these intersections (or at infinity). Thus there are only $O(n)$ components that can share the boundary with $U(S(f))$, which implies the lemma. □

By Lemma 3.10 deciding if there is 1–1 cut is equivalent to finding a connected component in the arrangement $D(f, f_1, \ldots, f_k)$ that shares boundaries with both $U(S(f))$ and $L(S(f))$.

A simplified but helpful view of this situation is to visualize a white horizontal strip $(S(f))$ shaped like $\asymp$ with gray strips (corresponding to $S(f_i)$) lying across it, which can be thought of as fences. The goal is to find a piece of the upper boundary and a piece of the lower boundary of $S(f)$ that are not fenced off from one another. To find if these exist, we will sweep the lower and upper boundaries of $S(f)$ with two points moving along them in a synchronized way. We want to find a location for the sweeping points such that they can see each other. Such positions must satisfy two basic conditions: (1) each of the sweeping points is located on a non-covered piece of the boundary, and (2) each of the sweeping points has crossed the same fences. The second condition simply means that the sweeping points are not on opposite sides of some obstacle. Technical complications to this general process may result from fences that do not completely cross the white strip.

The pattern of intersections between the butterfly of f and the butterfly of an obstacle f_i depends on the mutual position of f and f_i in the given coordinate system. Specifically, the intersections depend on the location of the external and internal tangents to f and f_i. To see this, recall that the dual projection maps parallel lines into points with the same x coordinates. Let $slab(\alpha, f)$ be a family of all parallel lines between a pair of tangents to f with slope α. Similarly, let $slab(\alpha, f_i)$ be a family of all parallel lines between a pair of tangents to f_i with slope α. Each such slab maps into a vertical line segment in the dual space, and this segment connects the upper and lower boundaries of $S(f)$. A similar situation holds for f_i and $S(f_i)$.

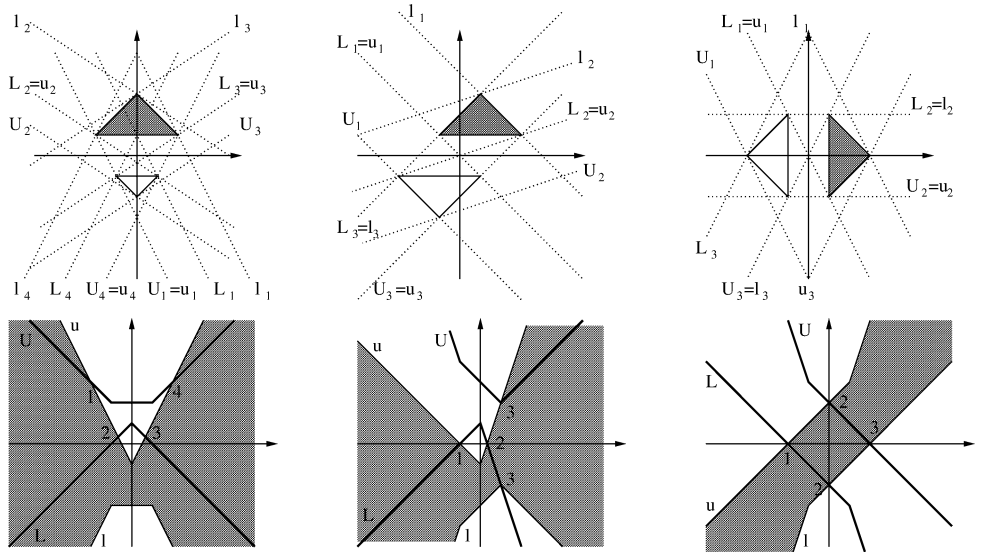

Fig. 9. f (white) and f_i (gray), their tangents, and the corresponding intersections between their boundaries in the dual space. Upper tangents, denoted with L and l, correspond to the lower boundaries.

Lemma 3.12. *Boundaries of $S(f)$ and $S(f_i)$ intersect at a point with coordinate $x = \alpha$ if there is a common tangent to f and f_i with slope α. If α is ∞, then this intersection is at infinity.*

Proof. Butterflies $S(f)$ and $S(f_i)$ contain a common vertical segment with $x = \alpha$ if $slab(\alpha, f)$ intersects $slab(\alpha, f_i)$; the intersection corresponds to the dual image of the common parallel lines in the intersecting slabs. If a line with the slope α is tangent to both f and f_i, then $slab(\alpha, f)$ and $slab(\alpha, f_i)$ share the boundary that maps into a common point in the dual space; this point is the intersection of the boundaries of the butterflies. If the slope of the common tangent is ∞ in the given coordinate system, then the dual image of such a line does not exist; this corresponds to an intersection of the boundaries at infinity. □

Depending on the position of the common tangents, the corresponding intersection point in the dual space is between upper or lower boundaries of $S(f)$ and $S(f_i)$. The total number of intersection points is four (including intersections at infinity), the same as the number of common tangent lines.

Fig. 9 illustrates a few cases of possible intersections. For example, the leftmost image in Fig. 10 shows four intersections of the upper boundary $u = U(S(f_i))$ with the boundaries of $S(f)$. Note that $l = L(S(f_i))$ does not intersect $S(f)$ at all. This is because the common tangents of f and f_i occur only among the tangents to f_i that are mapped into the upper boundary of $S(f_i)$. Such an intersection causes portions of one boundary of $S(f)$ to be separated from the other boundary, and in our algorithm we need to exclude those separated portions. Additional cases may be constructed similarly. For example, consider the leftmost image in Fig. 9 with the gray butterfly shifted upwards to totally contain $L(S(F))$; the gray butterfly separates $L(S(f))$ from $U(S(f))$.

The algorithm to find a component in $D(f, f_1, \ldots, f_k)$ that contains a path connecting $U(S(f))$ and $L(S(f))$ can be organized as follows:

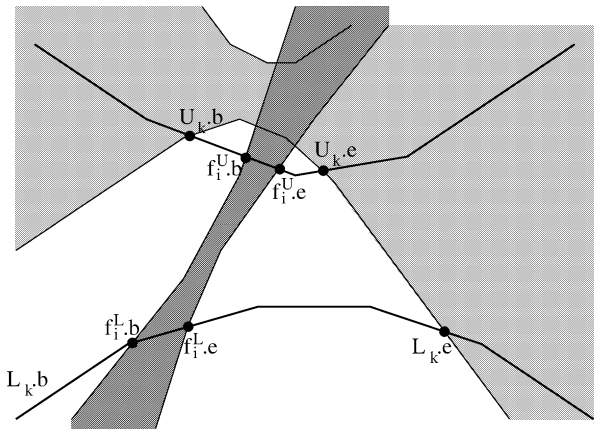

Fig. 10. Butterfly $S(f)$ (white) is intersected by a butterfly of group (1) (dark gray) and of group (2) (light gray).

Algorithm 1-1 Cutting out
Input: Butterflies $S(f)$, $S(f_i)$, $i = 1, \ldots, k$.
Question: Does f have a 1–1 cut-out?
Method: Finds a component of $D(f, f_1, \ldots, f_k)$ that shares a piece of boundary with $L(S(f))$ and $U(S(f))$. Uses two data structures, T_L and T_U, to store the sweeping point status. The sweeping is conducted with two points moving in a synchronized way along $L(S(f))$ and $U(S(f))$.

Step 1. Select a coordinate system. Compute $S(f)$ and $S(f_i)$ for $i = 1, \ldots, k$. Find the intersections of the boundaries of butterflies $S(f_i)$, $i = 1, \ldots, k$, with $U(S(f))$ and $L(S(f))$. Divide the butterflies $S(f_i)$, $i = 1, \ldots, k$, into two groups: (1) those properly intersecting $L(S(f))$ and $U(S(f))$ at four points in such a way that each boundary of $S(f_i)$ intersects each boundary of $S(f)$ twice and there are no intersections at infinity and (2) all other butterflies.

The first group corresponds to butterflies that cross $S(f)$ and split it into two components, each containing portions of $U(S(f))$ and $L(S(f))$. The second group corresponds to butterflies that do not split $S(f)$ into two components in the above sense. Fig. 10 shows a butterfly of each kind.

Step 2. For each butterfly $S(f_i)$ in group (2) find its intersection with the boundary of $S(f)$. Such an intersection forms an interval $(U_i.b, U_i.e)$ on the upper boundary and an interval $(L_i.b, L_i.e)$ on the lower boundary. These intervals are not separated from each other by $S(f_i)$. For butterflies of group (2) there is at most one such interval on each boundary. The endpoints of the intervals can be $-\infty$ or $+\infty$ if there are no intersections or there are no proper intersections (they correspond to vertical tangents of f and f_i.) If there are no non-empty intervals on the boundaries of $S(f)$, then the butterflies of the obstacles completely separate $L(S(f))$ from $U(S(f))$. In this case the algorithm returns with no 1–1 cutting-out path in this coordinate system and moves to Step 7.

Intervals $(U_i.b, U_i.e)$ and $(L_i.b, L_i.e)$ are computed based on the number of intersections and their types. Below we describe an interval computation procedure for a few representative cases. After finding the intersections between $S(f)$ and $S(f_i)$, we select the boundary of $S(f)$ with fewer intersections and compute intervals as follows: If there are 0 intersections (say, with

$U(S(f))$), then the corresponding interval $(U_i.b, U_i.e)$ is set to $(-\infty, +\infty)$. Then we check the other boundary.

1. If there are 0 intersections then $(L_i.b, L_i.e)$ is set to empty (boundaries are separated).
2. $L(S(f))$ is intersected once in a point p and the intersection is with $U(S(f_i))$. Then this is a double intersection point (the total number of intersections must be four, including intersections at the infinity) meaning that $U(S(f_i))$ is tangent to $L(S(f))$ at p. The interval is set to $[p, p]$. In the prime space this is exemplified by two equal squares, one stacked on the other. The point p corresponds to the common edge of the squares, and this edge is contained in a common tangent.
3. If there is one intersection p of $L(S(f))$ with $L(S(f_i))$, then $(L_i.b, L_i.e)$ is set to empty (boundaries are separated).
4. If there are two intersections, p_1 and p_2, of $L(S(f))$ with $U(S(f_i))$, then $(L_i.b, L_i.e)$ is set to (p_1, p_2).
5. If there are two intersections, p_1 and p_2, of $L(S(f))$ with $L(S(f_i))$, then $(L_i.b, L_i.e)$ is set to empty.

Other cases of different types of the intersections are classified similarly.

After finding the intervals, find the intersection $(L.b, L.e)$ (in the sense of set intersection) of all of the $(L_i.b, L_i.e)$, and find the intersection $(U.b, U.e)$ of all of the $(U_i.b, U_i.e)$. These intervals can be empty, in which case there is no 1–1 cutting path in the given coordinate system. If there are no butterflies in group (2), then these intervals can span the entire $U(S(f))$ or $L(S(f))$.

If at least one of $(L.b, L.e)$ and $(U.b, U.e)$ is empty, then the algorithm returns that there is no 1–1 cutting-out path in this coordinate system and moves to Step 7.

Step 3. For each $S(f_i)$ in group (1), find the four intersections of its boundary with the lower and upper boundaries of $S(f)$. Denote the intersection point on the upper boundary with the smaller x-coordinate as $f_i^U.b$ and the other one as $f_i^U.e$. Define $f_i^L.b$ and $f_i^L.e$ similarly.

Step 4. On each of $U(S(f))$ and $L(S(f))$, sort the intersection points in the above sets.

Step 5. Sweep $U(S(f))$ from $-\infty$ to $U.b$. For each encountered $f_i^U.b$, insert i into T_U and into T_L (if not already in). For each encountered $f_i^U.e$, delete i from T_U.

Similarly, sweep $L(S(f))$ from $-\infty$ to $L.b$. For each encountered $f_i^L.b$, insert i into T_L and into T_U (if not already in). For each encountered $f_i^L.e$, delete i from T_L.

If upon stopping at $U.b$ and $L.b$ the structures T_U and T_L are empty, then return that there is a 1–1 cutting out; a line segment connecting the sweeping points represents such a path (if any of $U.b$, $L.b$ is $-\infty$, use a point to the left of any proper intersections with butterflies of group (1) as the endpoint of this segment).

Otherwise continue with the next step.

Step 6. This step involves only intersections with butterflies of group (1).

While T_U is non-empty and we are not at $U.e$, continue sweeping $U(S(f))$ from $U.b$ and performing the following two actions: for each encountered $f_i^U.b$ insert i into T_U and into T_L (if not already in); for each encountered $f_i^U.e$ delete i from T_U.

While T_L is non-empty and we are not at $L.e$ continue sweeping $L(S(f))$ from $L.b$ and performing the following two actions: for each encountered $f_i^L.b$ insert i into T_U and into T_L (if not already in); for each encountered $f_i^L.e$ delete i from T_L.

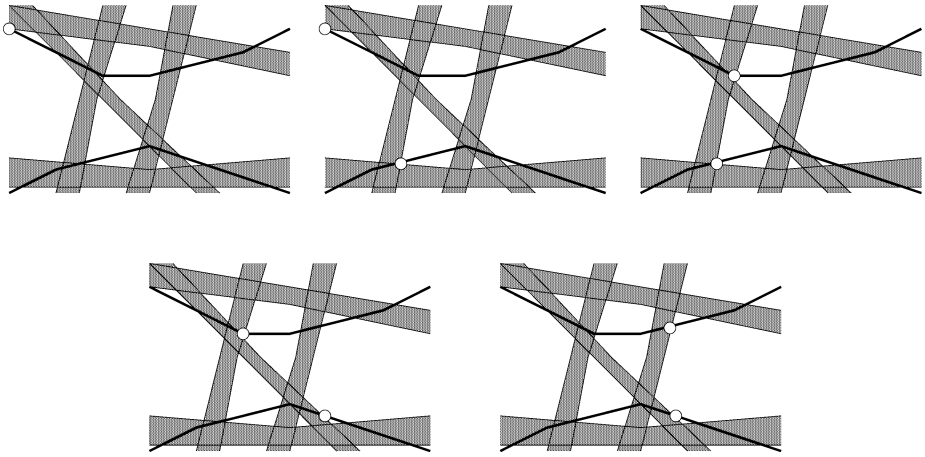

Fig. 11. The sweeping process. The two leftmost pictures of the sequence illustrate Step 5, and the rest of the pictures illustrate Step 6. The last picture in the sequence corresponds to empty structures T_U and T_L.

If both T_U and T_L are empty at this point, return that there is an 1–1 cutting out (a line connecting the sweeping points represents such a path). If T_U is non-empty and at $U.e$ or T_L is non-empty and at $L.e$, return that there is no 1–1 cutting-out path for f and move to Step 7. If T_U is non-empty but not at $U.e$ (note that the sweeping of $L(S(f))$ might have added objects to T_U), then repeat this step.

Step 7. If a 1–1 cutting path not found in this coordinate system, repeat the process for an additional coordinate system. The system is selected as follows. If there is not a path joining the opposite boundaries in a connected component of $D(f, f_1, \ldots, f_k)$ in the first coordinate system, select a new system whose OY axis is aligned with the slope α in the first coordinate system, where $\alpha = x_{\min} - \varepsilon$, where $x_{\min}$ is selected as the minimum x coordinate of the points in the right unbounded component of $D(f, f_1, \ldots, f_k)$ and ε is a small constant. Alternatively, select $\alpha = x_{\max} + \varepsilon$, where $x_{\max}$ is selected as the maximum x coordinate of the points in the left unbounded component of $D(f, f_1, \ldots, f_k)$; for explanation see the proof.

Fig. 11 illustrates the sweeping process described in Step 5 and Step 6 of the algorithm.

Before proceeding with the proof, let us visualize the algorithm for a simple boundary case of cutting-out a line segment with no obstacles. Formally, we consider cutting for open sets (such as interiors of polygons). Although a segment is not open in R^2, the algorithm will work and, moreover, makes a good illustration of the general idea. The butterfly for a line segment is a double wedge. If there are no obstacles at all, then both $U.b$ and $L.b$ are at $-\infty$, and any segment connecting the opposite boundaries of the double wedge (including a degenerate segment consisting of the vertex of the double wedge) provides a 1–1 cutting path.

Theorem 3.2. *Algorithm* 1–1 *cutting decides in* $O(n \log n)$ *time if there is a* 1–1 *cutting out of a convex face* f *in the presence of convex obstacles* $f_1, \ldots, f_k$ *that are disjoint with* f. *If such a cutting exists the algorithm returns a* 1–1 *cutting-out path.*

Proof. By Lemma 3.10 f can be 1–1 cut out if and only if there is a component in $D(f, f_1, \ldots, f_k)$ that shares its boundary with both $U(S(f))$ and $L(S(f))$. Let us demonstrate that the algorithm correctly detects such a component.

Firstly, note that only points in $(U.b, U.e)$ and $(L.b, L.e)$ of $U(S(f))$ and $L(S(f))$, respectively, can be in such a component; otherwise the boundaries are separated by a butterfly of group (2) (Step 2). In particular, if one of them is empty, there is no desired component in the given coordinate system.

Let a point p_U sweep $U(S(f))$ and a point p_L sweep $L(S(f))$. Assume that T_U or T_L is non-empty at some point during the sweeping process. Assume further that i is in T_U and was inserted while sweeping $U(S(f))$ with p_U. That means that p_U is inside butterfly $S(f_i)$, and, as such, it cannot belong to $D(f, f_1, \ldots, f_k)$. If i was inserted while sweeping $L(S(f))$, that means that p_U has not yet encountered $f_i^U.e$ and p_L passed $f_i^L.b$ (and possibly $f_i^U.e$). This means that the segment connecting p_U with p_L intersects $S(f_i)$, and p_U and p_L do not belong to a component of $D(f, f_1, \ldots, f_k)$ that has non-empty intersections with $U(S(f))$ and $L(S(f))$. The same holds for all segments connecting a point of $U(S(f))$ that is to the left of p_U with a point of $L(S(f))$ to the right of p_L.

On the other hand, if both T_U and T_L are empty at given positions of the sweeping points p_U and p_L, then both of them passed the same butterflies and have not entered any new ones. This means that the segment connecting p_U with p_L does not intersect any butterflies of group (1), and since p_U with p_L are in $(U.b, U.e)$ and $(L.b, L.e)$, respectively, this segment does not intersect any butterflies of group (2). Hence this segment connects $U(S(f))$ and $L(S(f))$ in a connected component of $D(f, f_1, \ldots, f_k)$. By Lemma 3.10, the segment connecting p_U and p_L determines a 1–1 cutting-out path.

We analyze two coordinate systems to map into a bounded component potential continuous curves connecting points on the $L(S(f))$ and $U(S(f))$ boundaries and passing through infinity. The choice of the coordinate systems ensures that the selected unbounded component becomes a connected one in the new system. Therefore any potential paths crossing infinity will also be considered and detected in some bounded component in this system. See also the discussion in the context of Fig. 5. Let us note here that an alternative, easier, but not always possible choice for a new coordinate system is to take in Step 7 α corresponding to an x coordinate of a position in the arrangement of butterflies where $S(f)$ is contained in a butterfly of some obstacle.

The entire process can be carried out in $O(n \log n)$ time. Finding $O(n)$ intersections between $S(f)$ and $S(f_i)$ of $O(n)$ line segments can be accomplished in an $O(n \log n)$ time; see, for example, [10] (we do not need intersection points between $S(f_i)$, $i = 1, \ldots, k$). Once the intersections are known, classifying them to determine type of the intersection (Step 1) and to determine intervals (Step 2) requires an $O(n)$ time. Sorting the intersection points (Step 3), and maintaining T_U, T_L implemented as balanced binary search trees (Step 5 and Step 6) for $O(n)$ points and $O(n)$ insert/delete operations requires $O(n \log n)$ time. □

3.5. Deciding if there is a cutting path

As we have seen, not all faces can be cut out in face-wise continuous fashion. However, they can still have a cutting out path, one that may require a discontinuous repositioning of the cutting line. The problem of finding if such a path exists will be addressed in the next section. In this section we ask how to decide if each point in a given face belongs to some cutting line.

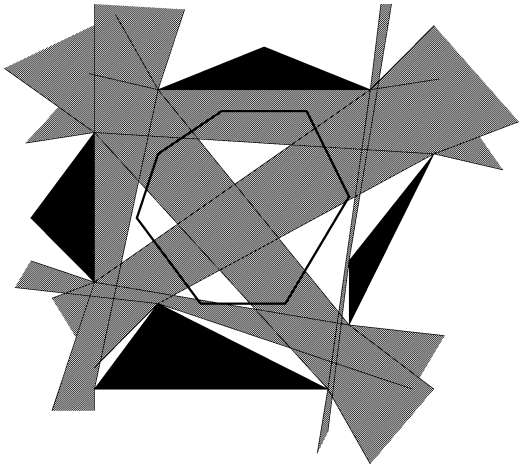

Fig. 12. White face, black obstacles and gray butterflies corresponding to cuts. The face can be cut out only if it is completely covered with the butterflies.

By Lemma 3.3 the connected components of $R^2 \setminus S(f_1, \ldots, f_k)$ are convex polygons. Therefore in the primal space each of these components is a butterfly, and there can be $O(n^2)$ of them. Lines in each of the butterflies avoid the obstacles.

We have the following:

Lemma 3.13. *f has a cutting out path if and only if f is contained in the union of the butterflies corresponding to the connected components of $R^2 \setminus S(f_1, \ldots, f_k)$ that have a nonempty intersection with $S(f)$.*

Proof. Clearly, the components that do not intersect $S(f)$ can be ignored as their primal space image is not connected with f. Each butterfly corresponds to lines that avoid obstacles. If the union of the butterflies covers f, then each point of f is cut by at least one line. To prove the other direction, take all the lines that cut f, and note that they cover f and belong to the butterflies. □

Lemma 3.13 is illustrated in Fig. 12, where the black polygons represent $f_1, \ldots, f_k$. In this case there are cells in the arrangement of the butterflies (gray areas) and the polygon that do not belong to any butterfly. Points in these cells cannot be cut and consequently the polygon cannot be cut out.

Lemma 3.13 implies an algorithm for checking if a face can be cut out.

`Algorithm: Feasibility of Cutting`
Input: f and $D(f, f_1, \ldots, f_k)$.
Output: Whether f can be cut out.

Step 1. Find $D(f, f_1, \ldots, f_k)$: with a line arrangement algorithm the cost of this step is an $O(n^2)$ time and space. Note that unlike in the case of Theorem 3.2, we need to construct the entire $D(f, f_1, \ldots, f_k)$.

Step 2. Find the pre-image for each of the connected components of $R^2 \setminus S(f_1, \ldots, f_k)$ that have nonempty intersection with $S(f)$; $D(f, f_1, \ldots, f_k)$ is useful here. There are $O(n^2)$ butterflies with the total of $O(n^2)$ vertices. The cost of this step is $O(n^2)$, and it uses $O(n^2)$ space.

Step 3. Incrementally construct the arrangement for all the butterflies. Since we have $O(n^2)$ butterflies, the cost of this step is $O(n^4)$, and it uses $O(n^4)$ space.

Step 4. Add the boundary of f to the above arrangement. Label *out* regions adjacent to the boundary of f and outside of f, and label *in* regions adjacent to the boundary of f and inside f. The cost of this step is $O(n^4)$, and it uses $O(n^4)$ space.

Step 5. Traverse the arrangement, marking cells that belong to at least one butterfly. This can be done by maintaining for each cell a count of butterflies covering the cell; this count is decreased or increased each time a boundary of the cell is crossed. The attributes *in* and *out* are used to determine if the current region is inside f. If there is a cell marked *in* that does not belong to any butterfly, report that the face cannot be cut out. The cost of this step is $O(n^4)$, and it uses $O(n^4)$ space.

The correctness and complexity of the algorithm is stated below:

Lemma 3.14. *The algorithm* `Feasibility of Cutting` *decides in* $O(n^4)$ *time and space if a collection of butterflies covers* f. *If* f *is not covered, then* f *cannot be cut out, and there is no path cutting out* f.

Proof. The complexity of the algorithm depends on the combinatorial complexity of the union of the butterflies. Since there are $O(n^2)$ lines, the complexity of the arrangement is $O(n^4)$, and all the steps, as indicated in the algorithm, can be carried out in $O(n^4)$ time. The algorithm reports that f cannot be cut out only if there is a point in f not covered by at least one butterfly. Since butterflies contain all the cutting lines for f, this point cannot be cut out. □

Note that the above algorithm can be modified to find out if a given subset of butterflies covers f. For a single butterfly this corresponds to deciding face-wise continuous cutting out for f. An optimization version of the problem is to find the smallest cover.

4. Connectivity graph

A cutting path that stays in contact with the closure of P consists of two types of moves: (1) cutting moves that cut a face, or (2) sliding moves that reposition the cutting line from one face f to another face or, if f does not have a face-wise continuous cut, to another part of f (or, in dual space terminology, to another component of $D(f, f_1, \ldots, f_k)$).

To cut out all the faces of P and, in particular, to cut out piece-by-piece those faces of P that cannot be cut out with face-wise continuity, we need to construct a cutting-out path that blends together cutting-out and sliding moves. A natural idea is to build a graph that describes all possible sliding moves. We will be able to identify paths between faces or between different portions of faces of P using standard graph traversal algorithms. We will call this graph a connectivity graph for P and denote it by $G(P)$.

To understand this more intuitively, it helps to visualize a wire-frame model of P and a long, magnetized, straight wire that is put on this frame. We can slide the wire along the edges of P trying to explore all possible paths. In this process the magnetic wire stays in contact with some edges and slides from one edge of P to another when it encounters a vertex or a face of P. We can further constrain this process by insisting that the wire always stays is touch with at least three edges of the frame. Indeed, if the wire initially touches only two edges we can "attach" the wire to one of the edges and move it until a third edge is encountered. Then we can continue sliding the wire on these three edges until a vertex or another edge is met, always keeping the wire in contact with at least three edges of P. Of course, it is possible to cut with a wire that moves freely in the space without necessarily touching three edges all the time. However, the compliant motion (sliding along three edges) allows us to discretize the motion of the wire and base it on the connectivity graph.

McKenna and O'Rourke [11], as well as the authors of this paper [6], showed that a free motion of a line in a set L of n lines in R^3 can be described by a sequence of combinations of four or fewer lines from L. In particular, McKenna and O'Rourke defined a graph G that describes classes of abstractions (connected components in the space of lines) for the moving lines. Each vertex in G corresponds to a line intersecting four lines in L or a line intersecting two lines and parallel to a third one (as we will see, we will not be concerned with this case). Each edge connecting vertices in G is labeled with three lines; sliding along these lines allows the sliding line to move between positions encoded by the vertices of G.

The connectivity graph $G(P)$ for cutting out P is a subgraph of G, where G is generated by the set L of lines supporting the edges of P. $G(P)$ does not include edges of G that are not feasible transitions for the cutting lines. Infeasible transitions are arcs in G that correspond to lines that move parallel to the planes containing the ends of the cutting wire and arcs corresponding to positions that would cause the moving line to penetrate the interior of P. Moreover, since each sliding move can maintain contact with three edges, we will only be interested in those arcs of G that are labeled with three lines of L.

Therefore, vertices of $G(P)$ correspond to some quadruples of edges of P and arcs between vertices of $G(P)$ correspond to some triples of edges of P. Intuitively, the arcs of $G(P)$ correspond to sliding the cutting line along three skewed edges to reposition the wire, and the vertices of $G(P)$ correspond to the positions where the wire touches four edges. In such a case the sliding must change its direction and continue on to another triple of edges (and their supporting lines), or the cutting line is positioned on a face. Arcs of $G(P)$ are computed based on pairs of edges of P. For edges that belong to the same face f of P, the connectivity is determined by looking at the structure of the f-lines and two edges from the obstacles.

For edges that lie on pairwise skewed lines, the construction is based on a generalization of the skewed projection introduced in [6]. Recall that two edges in R^3 are called skewed if they are not coplanar.

Lemma 4.1. *The number of edges in* $G(P)$ *is* $O(n^4)$.

Proof. $G(P)$ has $O(n^4)$ vertices, and the degree of each vertex is at most four. □

4.1. Constructing $G(P)$

Arcs of $G(P)$ will be constructed using generalized skewed projections [5,6]. (In [6] the skewed projection was defined for a pair of skewed (i.e., non-coplanar) lines in R^3 and screens parallel to the axes.)

The generalized skewed projection is defined R^3 and as follows:

Definition 4.1. Let l_1 and l_2 be two lines in the space R^3 or in the real projective space RP^3 and h be a hyperplane in this space that does not contain l_1 and l_2. The skewed projection $sp: R^3 \setminus (h_1 \cup h_2) \mapsto h$ ($sp: RP^3 \mapsto h$, respectively) of a point P on h with respect to l_1 and l_2 is the intersection of the line k intersecting l_1, l_2 and P with the hyperplane h (called the screen of the skewed projection).

Theorem 4.1. *Lines intersecting the axes l_1 and l_2 of the skewed projections and a given line l are contained in a plane, a one-sheeted hyperboloid or a parabolic hyperboloid.*

Proof. The proof is by straightforward calculations [5]. □

As an immediate corollary we have:

Corollary 4.1. *The skewed projection of a line in RP^3 is a line, an ellipse or a hyperbola. In R^3 the skewed projection of a line is contained in one of the above curves.*

Proof. Lines, ellipses and hyperbolas are the only possible intersections of a plane, one-sheeted hyperboloid or parabolic hyperboloid with a plane (the screen of the projection). □

To illustrate the above properties (in RP^3) consider the lines (axes of the projections) $(1, 0, 0, z)$ and $(1, 1, y, 0)$ and a line with the parametric equation $(1, 2, 0, 2) + t(0, 1, 1, 1)$. The surface generated by the lines intersecting all of the above three lines is a one-sheeted hyperboloid given by $2x^2 - 2xu - xz - yz = 0$ (see Fig. 13).

One interpretation of the skewed projection for a set of lines in R^3 is as follows: the arrangements of the curves on the projection screen correspond to the lines that limit the sliding of the sliding line. The

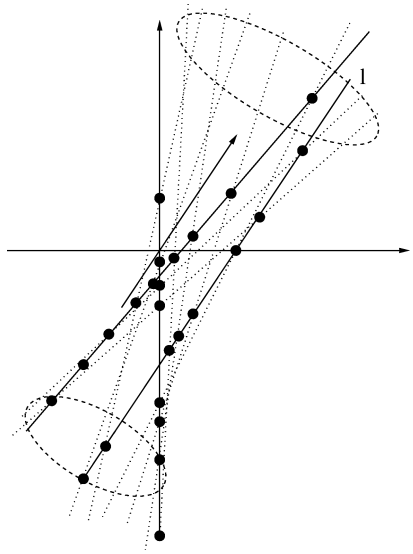

Fig. 13. Axes of a skewed projection and a line forming a one-sheeted hyperboloid.

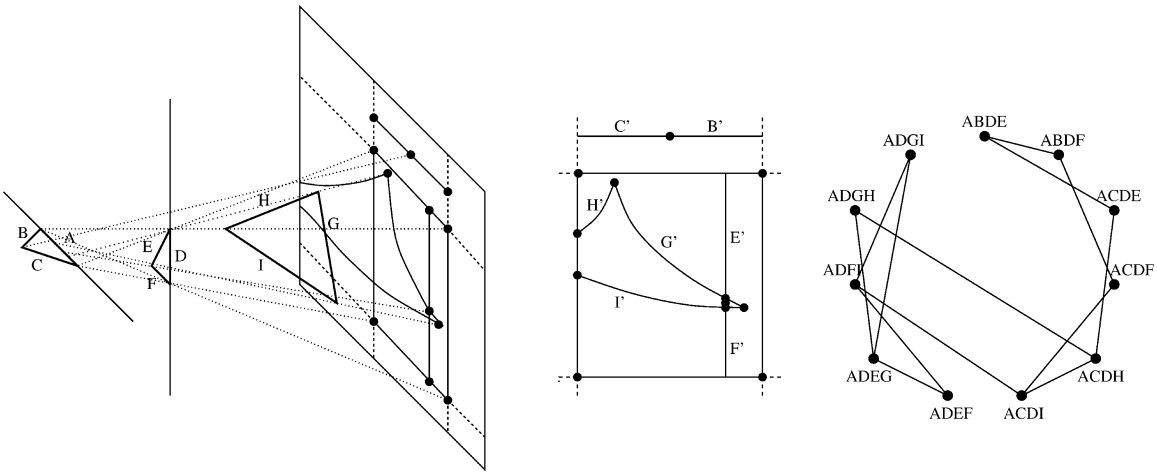

Fig. 14. The skewed projection of GHI with respect to edges A and D. The middle picture illustrates the result of the projection. Marked points correspond to projections of four lines. The rectangle is the projection of lines sliding along segments A and D. Projections of edges B and C are outside of the rectangle and will not be a part of the graph. Similarly, intersections of the projections of edges E and F with the boundary of the rectangle correspond to triples of lines ADE and ADF and therefore will not be vertices of the graph. The rightmost picture represents the connectivity graph for the skewed projection with respect to A and D.

curves correspond to triples of lines, and the intersections of curves correspond to at least quadruples of lines. The curves resulting from the skewed projection of edges of P, as well as their intersections, are used to construct $G(P)$.

As a remark let us note that the difference between projecting on the screen h_p parallel versus the screen h_{np} non-parallel to the axes is not significant. The conveyed information about the limitations on sliding is the same in either case. In fact, there is a simple relation between the two projections. The hyperbolic paraboloid of lines intersecting the axes and parallel to h_{np} intersects h_p along a line. When we cut h_p along this line and glue together the edges at infinity, the obtained arrangement of curves is identical with the one on h_{np}, up to a suitable transformation.

As another remark let us note that G can be built using the method from [11]; then $G(P)$ is obtained from G after removing unnecessary vertices and arcs. It is possible to show that this method reduces to the skewed projection with the projection screen parallel to the axes of the projection. However, because of restricting the screen to a particular position, constructing $G(P)$ from such an obtained G may be more complicated than with the above described process.

To move between faces of P let the cutting line l slide on edges e_1 and e_2 of P. The algorithm `Constructing vertices and arcs of` $G(P)$, presented below, identifies all edges e of P that l can touch and perhaps slide on while sliding on e_1 and e_2. The idea of the algorithm is to build the connectivity graph based on the skewed projections (with respect to lines including e_1 and e_2) of L, all the lines supporting edges of P. The obtained arrangement of the curves, the projections of L, will need to be pruned to remove fragments of curves that are beyond the edges of P or would cause l to penetrate P. The main difficulty in the algorithm is to recognize the latter. Each time the projecting line is coplanar with a face f of P, it intersects two edges of P. At this point, further sliding on one of these edges may cause l to penetrate P as the edge becomes invisible to l. To register this we mark this point on

the curve with the information that the adjacent face changes its orientation with respect to the projection line; later, the corresponding fragment of the curve will be removed from the arrangement.

Before providing pseudocode for the algorithm, let us take a look at the basic steps of computing the skewed projection; we will illustrate it with Fig. 14.

On this figure we have three selected triangular faces (the rest of P is not depicted), and we want to analyze the moves of the cutting line that slides on edges A and D and the edges of the face GHI; here G, H and I denote edges rather than vertices. We align the axes of the projection with the lines containing A and D; after a suitable transformation we can assume that the axes are given by $(x, 1, 0)$ and $(0, 2, z)$ and that the screen of the projection is $y = 3$, with all the triangles in front of it. If we are located at the point $(0, 0, 0)$, the axes appear as a cross, and the skewed projection of $\triangle GHI$ is obtained by tracing the intersection of a line that slides on the axes and the boundary of $\triangle GHI$. Note that while tracing the boundary of $\triangle GHI$ with a line sliding on the axes, it leaves e_1 and e_2. Since our intention is to stay in touch with edges of P, we crop the skewed projection of $\triangle GHI$ to a rectangle whose corners on the projection screen are determined by the projection line passing through the endpoints of edges e_1 and e_2. Anything that is outside of this rectangle is not interesting to us; this includes portions of the projections of edges G, H and I as well as the projection of C and B. The rightmost part of the figure shows a portion of the connectivity graph; its vertices correspond to positions of the sliding line when it touches four edges (A and D, which are the axes, and two other edges of the triangles). Arcs correspond to triples of edges; they connect two vertices of the graph if the quadruples share this triple of edges. The intersection of the projection of $\triangle GHI$ with the projections of edges E and F (both are coplanar with D, one of the axes, therefore their projections are colinear) exemplifies the situation where the projection line may look at two different sides of a face (and only one of them is visible). Invisible fragments of faces will be eliminated in Steps 5 and 6 of the algorithm presented below.

`Algorithm: Constructing vertices and arcs of` $G(P)$
Input: Polyhedral shape P and a pair of skewed edges e_1 and e_2 of P.
Output: Skewed projection of P with respect to the lines supporting e_1 and e_2.

Step 1. Find skewed projections (curves) for each line that supports an edge of P: time $O(n)$.
Step 2. As a side effect of projecting edges of P, the faces of P are also projected. For each edge e, mark next to the respective fragments of its projection the position of the projection of each face of P adjacent to e (whether the given face is to the left or to the right of the fragment in the projection): time $O(n)$, proportional to the number of faces and edges.
Step 3. Construct the arrangement $\mathcal{A}$ of the curves on the projection screen: time $O(n^2\alpha(n))$ based on [11].
Step 4. Crop the arrangement $\mathcal{A}$ to the curve fragments that can be reached by the sliding line limited to sliding on e_1 and e_2. This can be done with "windowing" the arrangement with four planes: two planes containing e_1 and the endpoints of e_2 and two for e_2 and e_1, respectively (without confusion we can call this arrangement $\mathcal{A}$): time $O(n^2)$.

The two following steps eliminate those fragments of curves in $\mathcal{A}$ that are invisible to the sliding line; l would pierce a face of P to slide on them:

Step 5. Select a position for the sliding line l (for example, let l pass through the endpoints of e_1 and e_2). Count the number of faces of P that l intersects. This is the count associated with the region C in $\mathcal{A}$ that corresponds to the intersection of l with the projection screen.

Step 6. Starting from C traverse $\mathcal{A}$, marking for each region in the arrangement the count of faces that are projected to this region using information collected in Steps 2 and 5. This count is changed by one each time the boundary of the region is crossed. Eliminate from $\mathcal{A}$ all curves that do not separate faces from regions with count equal to zero (that is, those regions which are not projections of any faces). The remaining arrangement (still called $\mathcal{A}$) represents the silhouette of P on which l can slide without penetrating P.

The algorithm constructs an arrangements that is a portion of $G(P)$ with vertices labeled with e_1, e_2 and two other edges of P, as well as arcs labeled with e_1, e_2 and another edge of P.

Theorem 4.2. *The arrangement of skewed projections of n lines can be constructed in* $O(n^2\alpha(n))$, *and its combinatorial complexity is* $O(n^2)$.

Proof. The number of intersections of n second-degree curves is $O(n^2)$. The arrangement for the lines and hyperbolas can be found in $O(n^2\alpha(n))$ time with the algorithm from [11]. To insert ellipses into the arrangement, we cut them into convex and concave pieces with respect to the direction of the intersection of h_1, h_2 with h and construct arrangements for each of these groups separately. Then we construct the arrangement for the parabolas in $O(n^2\alpha(n))$. All of the five arrangements (two for hyperbolas, two for ellipses, and one for parabolas) are merged together for the total of $O(n^2\alpha(n))$ time. □

Theorem 4.3. *For a given pair e_1, e_2 of skewed edges of P (edges supported by skewed lines), we can identify in* $O(n^2)$ *time, provided that the arrangement computed with the skewed projection is given, all edges e_3 in P that form triples describing a subset of edges of $G(P)$ and all quadruples of edges in P that describe a subset of vertices of $G(P)$. The triples describe possible slidings for the cutting line. The quadruples describe points where the direction of the cutting line changes.*

Proof. Each curve that is a skewed projection with respect to lines supporting e_1, e_2 determines the third edge in the triple. Intersections of the curves correspond to two edges that form quadruples together with e_1, e_2. The cost of the construction follows from the size of the arrangement as stated in Theorem 4.2. □

The entire graph $G(P)$ can be constructed as follows:

`Algorithm: Constructing` $G(P)$
Input: Polyhedral shape P.
Output: Connectivity graph $G(P)$.
For each pair of skewed edges e_1 and e_2 in P do

Step 1. Call algorithm `Constructing edges and arcs of` $G(P)$ for e_1 and e_2.

Step 2. Add to $G(P)$ quadruples of edges as vertices and triples of edges as arcs based on the arrangement constructed above.

Based on Theorem 4.3, $G(P)$ can be constructed in $O(n^4\alpha(n))$ time and $O(n^4)$ space.

Note that the skewed projection will also find arcs and vertices of $G(P)$ for edges in P that are coplanar.

Let us end this section with a comment related to the case when P is a convex polytope. One may wonder which are the skewed edges on which the cutting line l is sliding when cutting out P. To this end consider a face f of P. Let line l be in the plane of P and pass through a vertex v of f. Then l is tangent to the following edges: an edge not in the plane of f one of whose endpoints is v, an edge in the plane of f with one of its endpoints in v, another edge (across from v) in the plane of f. The first and third edges are skewed.

4.2. Finding cut-out paths

With the graph $G(P)$ we can decide if P can be polyhedron-wise cut out. We will associate a number of attributes with each vertex in the graph. First, each vertex whose name includes two edges that belong to the same face f of P will be labeled with this face (the vertex corresponds to the cutting line l placed in f). This can be done in constant time per vertex of $G(P)$ if each edge is labeled with its faces. Next, for each vertex in $G(P)$ that corresponds to a face that cannot be cut out in the face-wise continuous fashion, we find the connected component of $D(f, f_1, \ldots, f_k)$ that is adjacent to the edges of P meeting in this vertex.

With such an attributed $G(P)$ the algorithm `Cutting_Path(`$G(P)$) is defined as follows:

`Algorithm: Cutting_Path(`$G(P)$)
Input: $G(P)$.
Output: a polyhedron-wise continuous path for P, if it exists

Step 1. Find connected components of $G(P)$: time proportional to the size of $G(P)$ with Depth First Search.

Step 2. For each connected component of $G(P)$, find if all faces can be accessed, that is, if for each face f of P there is a vertex in this component labeled with f. If not all faces can be reached, then the component does not have the desired path.

Step 3. For each component returned from the previous step, for each face f that cannot be face-wise continuously cut out, find the collection of the connected components of $D(f, f_1, \ldots, f_k)$ labelling the respective vertices of $G(P)$.

Step 4. Call the algorithm 1–1 Cutting Out from Section 3.4 to verify if the connected components of $D(f, f_1, \ldots, f_k)$ contain paths that can be blended into a cutting out path for f.

Step 5. If there is a connected component in $G(P)$ that allows the accessing and cutting out of all of the faces of P, return the path in this connected component; otherwise, report that P cannot be polyhedron-wise continuously cut out.

The cost of the algorithm is dominated by calls to the algorithm 1–1 `Cutting Out`, whose cost is $O(n^4)$ per face.

Theorem 4.4. *It can be decided in* $O(n^5)$ *time and* $O(n^4)$ *space if a polyhedral shape with n edges can be cut out in a polyhedron-wise continuous fashion.*

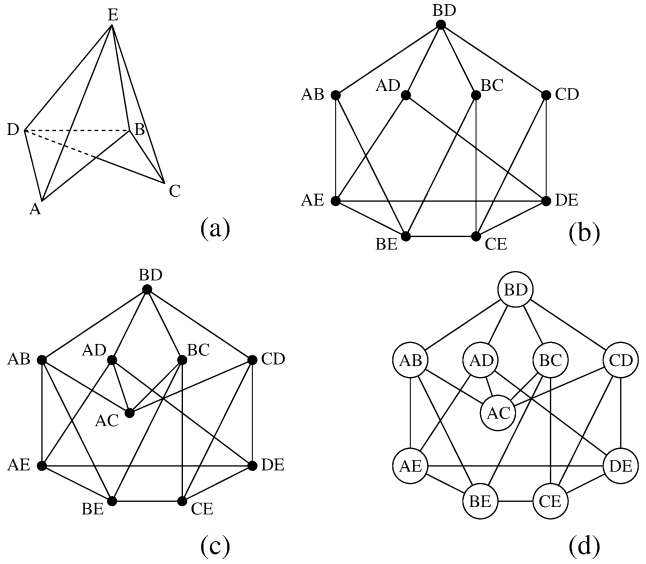

Fig. 15. Image (a) shows a polyhedron $ABCDE$ with six faces and pictures (b–d) show three different connectivity graphs for different types of cutting for this polyhedron. Graph (d) is a general graph discussed in this section. The nodes of the graph are not labeled with quadruples of edges. Instead, they are labeled with pairs of vertices corresponding to edges of $ABCDE$. The cutting line is moving from one edge to another by rotating about a vertex and sliding along a another edge.

Proof. $G(P)$ can be constructed and attributed in $O(n^4\alpha(n))$ time. The connected components can be found in $O(n^4)$ time. Verification of whether all the faces of P can be accessed requires time proportional to $O(n^4)$. The cost of verifying if a face can be cut out requires $O(n^4)$ time with the `Feasibility of Cutting` algorithm, for a total of $O(n^5)$ for $O(n)$ faces. All of the steps of the algorithm require no more than $O(n^4)$ memory. □

If a polyhedral shape P has special properties (e.g., it is convex), or if we know that all of its faces can be 1–1 cut out or face-wise continuously cut out, it is possible to build a simpler connectivity graph. For example, for 1–1 or face-wise continuous cutting such a graph can be constructed in $O(n^2)$ time.

For the general case of a polyhedral shape, we study a compliant motion that requires the wire to stay in contact with three edges. For special cases of polyhedral shapes, such as convex polyhedra, or for specific cutting requirements, such as cutting a face starting at an edge and completing the cutting with the wire aligned with another edge of the face, we can construct different connectivity graphs that correspond to more restricted ways of maintaining contact with three edges. Fig. 15 illustrates a simple polyhedron $ABCDE$ with six faces and three connectivity graphs that demonstrate that the graph $G(P)$ can be defined in a variety ways depending on the properties of P and on the mode of cutting. For example, in the first graph nodes of the graph correspond to the edges in $ABCDE$, and arcs correspond to a 1–1 cutting out of the faces where the cutting line moves from an edge to an edge. This requirement reduces the number of feasible movements and thus results in a simpler graph.

Vertices of the second graph are labeled with descriptions of the positions of lines. For example, AB means that the cutting line is defined by vertices A and B. After moving along the arcs, the line will assume a different position. This graph describes a face-wise continuous cutting out with a line moving from an edge or vertex in f to another edge or vertex of f.

The above graphs can be efficiently constructed. We do not present their constructions here; rather, we include them to show that the general approach of connectivity graphs can be applied to the wire cutting problem in a variety of ways.

The third graph is our general graph constructed with the skewed projection. However, it is presented in a compact form that is smaller, yet provides enough information about the motion. Because of the shape of $ABCDEF$, there is no situation that the cutting line slides on three edges. This allows us simplification. Instead of presenting nodes for quadruples of edges and presenting arcs for triples of edges, each node is labeled with a pair of vertices. AC describes all possible quadruples of edges that allow to align the cutting line with AC. For example, such an alignment can be achieved with the cutting line rotating about A and sliding along BC or EC. Similarly, the line can be rotated about C and sliding along BA or DA. An alternative way of thinking about this compact form is to see the node AC in the graph as a representation of a set of quadruples of edges that the cutting line touches while it is aligned with the edge AC; this idea is presented in [11].

The graphs illustrate many possible approaches in representing specialized motions. Such motions may lead to simpler graphs and to more efficient algorithms. One example is polyhedral shapes where no two different faces are coplanar.

5. Conclusions

We have presented algorithms for a number of tasks related to the wire-cutting of polyhedral shapes, including an algorithm for deciding and finding continuous cutting paths.

The paper uses two major groups of techniques. The first group of techniques, such as duality and skewed projections, is related to lines in two and three dimensional spaces. The second group borrows from robotics and robot motion planning methods.

The complexity of our algorithms depends on two subproblems: construction of the connectivity graph for admissible lines and deciding if a single face has a cutting-out path. Because of the high complexity of our algorithm, a natural task is to improve the running time and memory requirements of our solution in the general case. Detailed studies of special cases related to the geometry of the object to cut-out, a few of which were briefly mentioned in the paper, also provide a number of interesting questions. For example, how can advantage be taken of the multiple coplanar faces that can be cut simultaneously in planning the wire motion of a given object?

Other interesting, but possibly hard, open questions involve optimal planning, such as the minimal-time trajectory planning with the objective of minimizing the time the hot wire stays in contact with the material.

Among problems similar to cutting with a wire (line), one may consider cutting with a laser beam (a half-line), with a rod (a line segment), or even with task-specific tools such as a meat-cutter (Joe Mitchell, personal communication). Some of the methods presented in the paper are immediately relevant to the above problems. For example, one of the technological advantages of the laser beam comes into play when the connectivity graph is not connected. By simply switching the beam on and off it is possible to move from one connected component to another. This becomes relevant in manufacturing objects of genus higher than 0. For cutting with a thin rod (similar to a wire but attached to the moving arm only at one of the ends, and the tip of the wire cannot cut) note that more objects can be cut-out with it than with a wire. As an example we can consider a cube with one corner in the shape of a cube removed. To

detect if a particular face can be cut-out with a rod we can use an approach similar to Section 3.5. Instead of covering the face with regions outside tangents to the obstacles, we can analyze the intersection of the region between the tangents to a pair of obstacles. The face can be cut-out if the intersection with the face is empty. However, although the main approach and techniques used and developed in the paper are helpful for these problems, technological and geometric differences most likely will require analysis of many problem-specific details.

Acknowledgements

The authors would like to thank the referees for exceptionally helpful and detailed comments with specific suggestions that were critical in correcting and improving this paper. Our thanks also go to Ryan Gabbard for proofreading numerous iterations of the manuscript.

References

[1] B. Chazelle, H. Edelsbrunner, L.J. Guibas, M. Sharir, Algorithms for bichromatic line segment problems and polyhedral terrains, Algorithmica 11 (1994) 116–132.
[2] H.S.M. Coxeter, Projective Geometry, Second Edition, University of Toronto, Toronto, 1974.
[3] H. Edelsbrunner, Algorithms in Combinatorial Geometry, Springer-Verlag, Berlin, 1987.
[4] H. Edelsbrunner, H.A. Maurer, F.P. Preparata, A.L. Rosenberg, E. Welzl, D. Wood, Stabbing line segments, BIT 22 (1982) 274–281.
[5] J.W. Jaromczyk, M. Kowaluk, Generalized skewed projections in R^3, Manuscript.
[6] J.W. Jaromczyk, M. Kowaluk, Skewed projections with an application to line stabbing in R^3, in: Proc. 4th Annu. ACM Sympos. Comput. Geom., New York, ACM Press, 1988, pp. 362–370.
[7] J.W. Jaromczyk, M. Kowaluk, The face-wise continuity in hot wire cutting of polyhedral objects, in: Proc. 16th European Workshop on Computational Geometry, Eilat, 2000, pp. 93–97.
[8] J.W. Jaromczyk, M. Kowaluk, Set of lines and cutting out polyhedral objects, in: Proc. 17th European Workshop on Computational Geometry, Berlin, 2001, pp. 183–186.
[9] T. Lozano-Pérez, Spatial planning: A configuration space approach, IEEE Trans. Comput. C-32 (1983) 108–120.
[10] M. Overmars, M. de Berg, M. van Kreveld, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, Springer-Verlag, Berlin, 1999.
[11] M. McKenna, J. O'Rourke, Arrangements of lines in 3-space: A data structure with applications, in: Proc. 4th Annu. ACM Sympos. Comput. Geom., New York, ACM Press, 1988, pp. 371–380.
[12] O. Nurmi, J.-R. Sack, Separating a polyhedron by one translation from a set of obstacles, in: Proc. 14th Internat. Workshop Graph-Theoret. Concepts Comput. Sci., in: Lecture Notes Comput. Sci., Vol. 344, Springer-Verlag, Berlin, 1989, pp. 202–212.
[13] D. Nussbaum, J.-R. Sack, Translation separability of polyhedra, in: Abstracts 1st Canad. Conf. Comput. Geom., 1989, p. 34.
[14] M. Pellegrini, Lower bounds on stabbing lines in 3-space, Computational Geometry 3 (1993) 53–58.
[15] M. Pellegrini, Ray shooting and lines in space, in: Jacob E. Goodman, Joseph O'Rourke (Eds.), Handbook of Discrete and Computational Geometry, CRC Press LLC, Boca Raton, FL, 1997, pp. 599–614, Chapter 32.
[16] J.T. Schwartz, M. Sharir, On the piano mover's problem: V. the case of a rod moving in three-dimensional space amidst polyhedral obstacles, in: M. Sharir, J.T. Schwartz, J. Hopcroft (Eds.), Planning, Geometry, and Complexity of Robot Motion, Ablex Publishing Corporation, Norwood, NJ, 1986, pp. 154–186.
[17] J.T. Schwartz, Micha Sharir, A survey of motion planning and related geometric algorithms, in: D. Kapur, J. Mundy (Eds.), Geometric Reasoning, MIT Press, Cambridge, MA, 1989, pp. 157–169.