INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Complete and Efficient Algorithm for the Intersection of a General and a Convex Polyhedron

Katrin Dobrindt, Kurt Mehlhorn, Mariette Yvinec

N° 2023

Août 1993

PROGRAMME 4

Robotique,
image
et vision

Rapport de recherche

1993

ISSN 0249-6399

A Complete and Efficient Algorithm for the Intersection of a General and a Convex Polyhedron

Katrin Dobrindt, Kurt Mehlhorn*, Mariette Yvinec**

Programme 4 — Robotique, image et vision
Projet Prisme

Rapport de recherche n° 2023 — Août 1993 — 14 pages

Abstract: A polyhedron is any set that can be obtained from the open halfspaces by a finite number of set complement and set intersection operations. We give an efficient and complete algorithm for intersecting two three–dimensional polyhedra, one of which is convex. The algorithm is efficient in the sense that its running time is bounded by the size of the inputs plus the size of the output times a logarithmic factor. The algorithm is complete in the sense that it can handle all inputs and requires no general position assumption. We also describe a novel data structure that can represent all three–dimensional polyhedra (the set of polyhedra representable by all previous data structures is not closed under the basic boolean operations).

Key-words: Computational Geometry, Solid Modeling, Data Structures, Polyhedra Intersection.

(Résumé : tsvp)

The research of all three authors was partly supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II). The research of the second author was also partially supported by the BMFT (Förderungskennzeichen ITS 9103). The paper is based on the first author's master's thesis [Dob90]. A preliminary version was presented at the third Workshop on Algorithms and Data Structures (WADS'93).

*Max-Planck-Institut für Informatik, 6600 Saarbrücken, Germany.
**INRIA and Laboratoire I3S, CNRS-URA 1376, 06902 Sophia-Antipolis, France.

Un algorithme complet et efficace pour l'intersection d'un polyèdre général avec un polyèdre convexe

Résumé : Un polyèdre est tout ensemble qui peut être obtenu à partir de demi-espaces par un nombre fini d'opérations de complément et d'intersection. Nous proposons ici un algorithme complet et efficace pour construire l'intersection de deux polyèdres dans l'espace tridimensionel dont l'un est convexe. L'algorithme est efficace car son temps de calcul est, à un facteur logarithmique près, borné par la taille des entrées plus la taille de la sortie. L'algorithme est complet dans le sens qu'il peut traiter toutes les entrées sans aucune hypothèse de position générale. De plus, nous décrivons une nouvelle structure de données susceptible de représenter tout polyèdre dans l'espace (toutes les structures utilisées précédemment représentent seulement des ensembles de polyèdres qui ne sont pas stables pour les opérations booléennes de base).

Mots-clé : Geométrie algorithmique, modélisation de solides, structures de données, intersection de polyèdres.

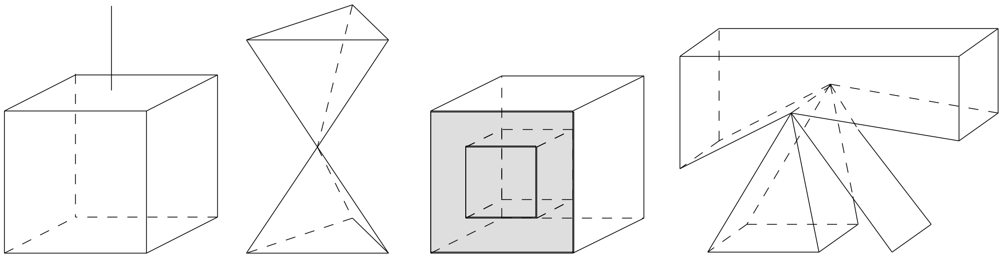

Figure 1: Examples of general polyhedra; the second polyhedron from the right is a cube with a hole whose frontside is closed by a plane.

1 Introduction

A polyhedron is any subset of three–dimensional Euclidean space that can be obtained from the open halfspaces by a finite number of set complement and set intersection operations. Figure 1 shows some polyhedra. We give an algorithm to compute the intersection of a polyhedron P with a convex polyhedron C. The algorithm runs in time $O((|P|+|C|+|P\cap C|)\log(|P|+|C|+|P\cap C|))$ where $|\ |$ denotes the size of a polyhedron. The algorithm works for all inputs and not only for inputs in general position. The only previous algorithm with similar efficiency of Mehlhorn and Simon [MS85] applied only to regular[1] polyhedra in general position, i.e., a face of P and a face of C may intersect only if the sum of their affine hulls is the entire space. The intersection of two regular polyhedra in general position is again regular.

The standard data structures for three–dimensional polyhedra, e.g. the quad–edge–structure of [GS85, EM85], the doubly–connected–edge–list of [MP78, PS85], and the half–edge–structure of [Män88], cannot represent all polyhedra. This implies that the class of representable (in any one of these data structures) polyhedra is not closed under the basic boolean operations intersection, union, and complement. For instance, Figure 2 shows that the intersection of two regular polyhedra can actually be non-regular. Given these facts we are facing a crucial decision. We can either stick to the standard representations and redefine the basic boolean operations (by adding a regularization step which is the traditional remedy, cf. [Req80, Män88, Hof89]), or stick to the standard definitions of the basic operations and give up the standard representation. We believe that the second alternative is cleaner. Besides, the solids shown in Figure 1 look perfectly reasonable. We introduce a new data structure (called the *local–graphs–data–structure*) that can represent all three–dimensional polyhedra. Our data structure is based on the fundamental work of Nef [Nef78] (see also [BN88]) who studied the mathematical properties of polyhedra. The data structure stores a polyhedron as a collection of faces (vertices, edges, and facets); each face is described as the set of points comprising the face and its local graph. The local graph is a planar graph embedded into a sphere that captures the local properties

[1] All mathematical terms and notations are summarized in the appendix A.

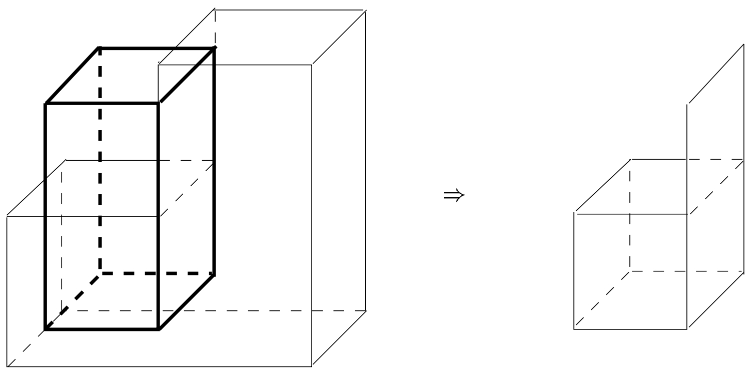

Figure 2: The intersection of two regular polyhedra is not regular

of the polyhedron in the neighborhood of the face. The details are given in section 2.

Apart from the algorithm given by Mehlhorn Simon [MS85] mentioned above, all efficient algorithm for intersecting two polyhedra in space apply only to convex polyhedra. The first efficient algorithm for solving this problem was given by Muller and Preparata [MP78]. This algorithm takes, for two convex polyhedra C_1 and C_2, time $O((|C_1|+|C_2|)\log(|C_1|+|C_2|))$. Alternative algorithms were proposed by Hertel *et al.* [HMMN84] and by Dobkin and Kirkpatrick [DK83]. The former is based on the space sweep technique and the latter uses the hierarchical representation of convex polyhedra. Recently, Chazelle [Cha92] presented an algorithm for constructing the intersection of two convex polyhedra in linear time $O(|C_1| + |C_2|)$. In section 3, we describe a complete algorithm for intersecting a general polyhedron P with a convex polyhedron C with running time $O((|P| + |C| + |P \cap C|)\log(|P| + |C| + |P \cap C|))$. This algorithm first computes the intersections of all faces of P and all faces of C and then builds the local–graphs–data–structure for $P \cap C$. We employ different strategies for intersecting faces depending on the dimension of the faces involved and use for the convex polyhedron also its hierarchical representation.

In [DMY93] we outlined an intersection algorithm based on the symbolic perturbation technique introduced by Edelsbrunner and Mücke [EM90]. We believe that the algorithm presented here is simpler. In the other algorithm, we first perturb the convex polyhedron C by moving its facets outwards by infinitesimal amounts. This brings the two polyhedra into general position. We then apply an extension of the intersection algorithm of Mehlhorn and Simon [MS85] to P and the perturbation $C(\varepsilon)$ of C. Finally, we let ε go to zero and obtain $P \cap C$ from $P \cap C(\varepsilon)$. The limit process is mathematically and algorithmically quite involved and so the overall algorithm is more complex than the algorithm presented here. However, if the exact output is not needed and $P \cap C(\varepsilon)$ suffices then the other algorithm is to be preferred.

The paper is organized as follows. In Section 2 we introduce a data structure for representing polyhedra. The intersection algorithm is described in Section 3.

2 The Local–Graphs–Data–Structure

We start with a brief review of Nef's theory of polyhedra [Nef78, BN88].

Definition 2.1 [Nef78] A *polyhedron* in $\mathbb{R}^3$ is a set $P \subseteq \mathbb{R}^3$ generated from a finite number of open halfspaces by set complement and set intersection operations.

Figure 1 shows some polyhedra. A face of a polyhedron is a maximal set of points which have the same local view of the polyhedron. The exact definition requires the concept of the local pyramid of a point.

Definition 2.2 A set $K \subseteq \mathbb{R}^3$ is called a *cone with apex 0*, if $K = \mathbb{R}^+K$ and a *cone with apex* x, $x \in \mathbb{R}^3$, if $K = x + \mathbb{R}^+(K - x)$. Thus, for a cone K with apex x, $(K - x)$ is a cone with apex 0. A cone is *polyhedral* if it is a polyhedron. A set $Q \subseteq \mathbb{R}^3$ is called a *pyramid with apex* x, if it is a a polyhedral cone with apex x.

Note that $\mathbb{R}^+$ does not include zero and thus a cone K with apex x may or may not include x. Furthermore, a cone can have more than one apex and the set of all apices of a cone is a flat.

Definition 2.3 [Nef78] Let $P \subseteq \mathbb{R}^3$ be a polyhedron and $x \in \mathbb{R}^3$. There is a neighborhood $U_0(x)$ of x such that the cone $Q := x + \mathbb{R}^+((P \cap U(x)) - x)$ is the same for all neighborhoods $U(x) \subseteq U_0(x)$. The cone Q is called the *local pyramid of the polyhedron* P *in the point* x and is denoted $Pyr_P(x)$.

Indeed, the cone Q is a polyhedron and thus a pyramid with apex x. It describes the local characteristics of the polyhedron in the neighborhood of the point x.

Definition 2.4 [Nef78] Let $P \subseteq \mathbb{R}^3$ be a polyhedron. A *face* s of P is a maximal (with respect to set inclusion) non-empty subset of $\mathbb{R}^3$ such that all of its points have the same local pyramid Q, i.e., $s = \{x \in \mathbb{R}^3 | Pyr_P(x) = Q\}$. Q is called the pyramid associated with the face and is denoted $Pyr_P(s)$ or simply $Pyr(s)$. The dimension of s is the dimension of the linear subspace of all apices of Q.

A face s_1 is *incident* to a face s_2 if $s_1 \subseteq clos(s_2)$. As usual we call a 0-dimensional face a *vertex*, a 1-dimensional face an *edge*, and a 2-dimensional face a *facet*. A face of dimension two or less is called *low–dimensional*. In Figure 3 an example in $\mathbb{R}^2$ is given: the polyhedron has one vertex v, two edges e_1 and e_2, and two 2-dimensional faces f and $int(cpl(f))$. We now list some basic properties of faces.

Fact 2.1 [Nef78]

a) *All faces of a polyhedron are polyhedra.*

b) *The linear subspace of all apices of the pyramid associated with a face is the affine hull of the face.*

c) *Faces are relatively open sets.*

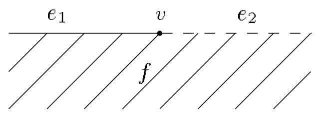

Figure 3: Example in $\mathbb{R}^2$: the edge e_1 belongs to the polyhedron and e_2 does not.

d) $x \in \mathrm{Pyr}_P(x)$ *iff* $x \in P$.

e) *A polyhedron* P *has at most two 3-dimensional faces, namely* $\mathrm{int}(P) = \{x \in \mathbb{R}^3 | \mathrm{Pyr}_P(x) = \mathbb{R}^3\}$ *and* $\mathrm{ext}(P) = \{x \in \mathbb{R}^3 | \mathrm{Pyr}_P(x) = \emptyset\}$. *The boundary* $\mathrm{bd}\ P = \{x;\ \emptyset \neq P \cap U \neq U$ *for every neighborhood* U *of* $x\}$ *is equal to the union of the low–dimensional faces.*

f) *Let* s *be a face of the polyhedron* P *and let* t *be a face of* s. *Then* t *is the union of some faces of* P *(cf. Figure 3 for an illustration: the face* f *has one edge* e *that is the union of faces* e_1, v, *and* e_2.*)*

g) *A face of* P *is either a subset of* P *or disjoint from* P.

A polyhedron may have an arbitrary number of edges with the same affine hull but can have at most six facets with the same affine hull (since the local pyramid of a facet is either an open or a closed halfspace or a plane or the complement of a plane). A face is not necessarily connected nor bounded and a polyhedron does not necessarily have faces of all dimensions. We are now ready to define the (abstract) representation of a polyhedron. We will later develop a concrete data structure for it.

Definition 2.5 For a polyhedron P let $rep(P)$ be the set $\{(s, Pyr_P(s));\ s$ is a low–dimensional face of $P\}$.

Every polyhedron different from the full space and the empty set has a low–dimensional face.

Lemma 2.2 *Let* P *and* R *be distinct polyhedra. Then* $rep(P) \neq rep(R)$ *or* $\{P, R\} = \{\emptyset, \mathbb{R}^3\}$.

Proof: Let P and R be distinct polyhedra. We may assume w.l.o.g. that $P \setminus R \neq \emptyset$. If $P = \mathbb{R}^3$ then there is nothing to show. Otherwise, let x and y be points with $x \in P \setminus R$ and $y \notin P$. Let z be the first point on the ray from x to y that belongs to the boundary of either P or R. We claim that $Pyr_P(z) \neq Pyr_R(z)$. If $x \neq z$ this follows from the observation that the open line segment with endpoints x and z is contained in P and is disjoint from R, and if $x = z$ this follows from $x \in P \setminus R$. Also, z belongs to a low–dimensional face of either P or R. Thus $rep(P) \neq rep(R)$. $\square$

We propose to store a polyhedron as the collection of its low–dimensional faces together with their local pyramids. So we need data structures for local pyramids and faces.

Let $Pyr_P(x)$ be the local pyramid of point x and let $S(x)$ be a sphere with center x. The intersection of $S(x)$ with $Pyr_P(x)$ is a planar graph embedded into $S(x)$ that we denote $G_P(x)$. The nodes, arcs, and regions of this graph[2] correspond to the edges, facets, and three-dimensional faces of $Pyr_P(x)$ respectively. The graph $G_P(x)$ may consist of a single arc and no node. In this case we call this unique arc a *selfloop*. For each feature (=node, arc, or region) f of the graph we have a label $inP(f)$ indicating whether the feature is contained in $Pyr_P(x)$. We also have such a label for the point x. We call $G_P(x)$ together with these labels the *local graph* of x and also use $G_P(x)$ to denote it. The following lemmata characterize local graphs and give a criterion to determine the dimension of the face containing its center. We use the phrase *to classify* x to mean to determine the dimension of the face containing x.

Lemma 2.3 *The following properties hold for every local graph.*

a) *Every arc of G is part of a great circle.*

b) *For every arc a of G there is a region r incident to a with* $\mathrm{inP}(a) \neq \mathrm{inP}(r)$.

c) *For every node v of G there is an arc or region f incident to v with* $\mathrm{inP}(v) \neq \mathrm{inP}(f)$.

d) *For every node v of G of degree 2 with two cocircular arcs a_1 and a_2 incident to it the three labels* $\mathrm{inP}(a_1)$, $\mathrm{inP}(v)$, *and* $\mathrm{inP}(a_2)$ *are not identical.*

Moreover, any graph G (embedded into a sphere) satisfying properties a) *to* d) *above is the local graph $G_P(x)$ for some polyhedron P and some point x.*

Proof: a) Each arc correspond to a facet of $Pyr_P(x)$. The affine hull of this facet is a hyperplane passing through x. The intersection of this hyperplane with $S(x)$ is a great circle.

b) If for all regions r incident to an arc a the label $inP(a)$ equals $inP(r)$, then the pyramid of the points on a with respect to $Pyr_P(x)$ is either $\emptyset$ or $\mathbb{R}^3$. This is a contradiction since these points belong to a facet.

c) As in b).

d) If the label $inP(a_1)$, $inP(v)$, and $inP(a_2)$ are identical then v and all points on $a1$ and $a2$ have the same pyramid although they belong to different faces. This is a contradiction that faces are maximal subsets of points having the same pyramid.

Let x be the center of the sphere and P the cone with apex x intersecting the sphere along G. According to properties a) to d) P is a polyhedron, and the local graph for P and x is G. ◻

[2] We reserve the words vertex, edge, and facet for polyhedra.

Lemma 2.4 *Let G be the local graph of some point x with respect to some polyhedron P.*

a) *The point x belongs to a 3-dimensional face of P iff G has no nodes and no arcs and the unique region of G has the same label as x.*

b) *It belongs to a facet of P iff G consists of a single selfloop that, in addition, has the same label as x.*

c) *It belongs to an edge of P iff G has exactly two nodes and these nodes are antipodal and have the same label as x (there can also be an arbitrary number of arcs connecting the two nodes).*

d) *In all other cases, x is a vertex of P.*

Proof: a) According to the definition the point x belongs to a 3-dimensional face of P iff $Pyr_P(x) = \emptyset$ or $Pyr_P(x) = \mathbb{R}^3$. Thus G has no nodes and no arcs and the unique region of G has the same label as x.

b) The point x belongs to a facet of P iff the set of apices of $Pyr_P(x)$ is a hyperplane passing through x. Thus G is selfloop that has the same label as x.

c) The point x belongs to an edge of P iff the set of apices of $Pyr_P(x)$ is a line passing through x. Thus G has exactly two nodes and these nodes are antipodal.

d) Since the local graph is the intersection of a sphere with center x with $Pyr_P(x)$, the only possibility left is that x is a vertex of P.

□

We now complete the description of the local–graphs–data–structure. A vertex is represented by its coordinates and its local graph. An edge is represented by the equation of its affine hull, an ordered sequence of open line segments comprising the points belonging to the edge[3], and the local graph of the edge. The endpoints of these line segments are the 0–dimensional faces of the edge. They correspond to vertices of P.

A facet is represented by the equation of its affine hull, the set of points belonging to the facet, its local graph, and its set of vertices and edges[4]. The set of points of the facet is stored as a straight–line planar graph. For each region of this planar graph there is a label indicating whether the region belongs to the facet or not. The vertices (0–dimensional faces) of a facet are precisely the nodes of this planar graph. The edges (1–dimensional faces) of a facet partition the arcs of the planar graph. If two arcs belong to the same edge of the facet then the arcs are collinear (the converse is not true). We associate with each arc the edge containing it and with each edge (of the facet) the ordered sequence of arcs contained in the edge.

[3] Remember that faces are not necessarily connected.

[4] Keep in mind that according to Fact 2.1.d) an edge of a facet is the union of some edges and vertices of P.

We also store cross links between the different occurrences of the same object. For example, with every edge e of a facet f we associate the set of edges and vertices of P comprising e. If $\overline{e}$ is an edge or vertex of P contributing to e then there is a cross pointer between the item representing $\overline{e}$ in the set associated to e and the at most two arcs corresponding to f in the local graph $G_P(\overline{e})$. Similarly, there is a cross pointer between every node x of a local graph $G_P(v)$ for a vertex v of P and the edge segment corresponding to that node, ...

The *size* of a polyhedron P is defined as the size of its local–graphs–data–structure and is denoted $|P|$. It is proportional to the number of incidences between the faces of P.

Definition 2.6 A *convex polyhedron* in $\mathbb{R}^3$ is a non-empty intersection of a finite number of open or closed halfspaces.

It follows that a convex polyhedron is not necessarily closed. For convex polyhedra we also use the *hierarchical representation* introduced by Dobkin and Kirkpatrick [DK83]. An hierarchical representation of a convex polyhedron C is a nested sequence $C_0 \subseteq C_1 \subseteq \ldots \subseteq C_k$ of convex polyhedra, with (i) C_0 is a tetrahedron and C_k is the polyhedron C and (ii) the set of vertices V_i of C_i is obtained from V_{i+1} by removing a subset I_{i+1} of pairwise non adjacent vertices of C_{i+1}. We can find a set $|I_{i+1}|$ of at least $|V_{i+1}|/7$ pairwise non adjacent vertices of degree at most 12, by considering the vertices of V_{i+1} in order of non-decreasing degree as shown in [Ede87]. The element C_i of the sequence is formed from C_{i+1} by removing the subset I_{i+1} and forming the convex hull of the remaining vertices. This computation requires linear time and gives a representation of C of linear space. The hierarchical representation supports intersection queries with lines and planes in logarithmic time. An intersection query with a line returns the intersection (a line segment) and an intersection query with a plane returns an arbitrary point of the boundary of the intersection (if any). If the plane intersects the polyhedron in a vertex, edge, or facet then this fact is also reported.

3 The Intersection Algorithm

We show how to compute $P \cap C$, where P is an arbitrary polyhedron and C is a convex polyhedron, in time $O((|P| + |C| + |P \cap C|) \log(|P| + |C| + |P \cap C|))$. We refer to this bound as our *target time* and use N to denote $|P| + |C| + |P \cap C|$, i.e., the combined input and output size. Our algorithm is based on the following fact.

Fact 3.1 [Nef78] *Let P and C be polyhedra. Then every face of $P \cap C$ is the union of intersections of faces of P and C.*

If one of the polyhedra is convex then a slightly stronger property holds (cf. Figure 4).

Lemma 3.2 *Let P be a polyhedron and let C be a convex polyhedron. Then every face of $P \cap C$ is the intersection of one face of C with the union of some faces of P.*

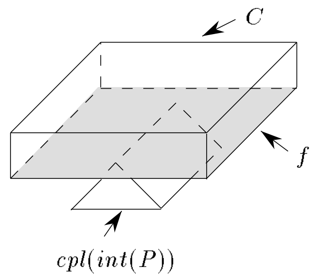

Figure 4: Illustration of Lemma 3.2: the facet f of $P \cap C$ is the union of the intersection of a facet of C with $int(P)$, an edge of P and two vertices of P

We use different strategies for intersecting faces depending on the dimension of the faces. In sections 3.2 to 3.4 we discuss how to intersect the vertices, edges, and facets of P with the faces (of all dimensions) of C. All three sections rely on a subroutine to intersect two local graphs that we introduce in section 3.1. In section 3.5 we show how to intersect the full–dimensional faces of P with the faces (of all dimensions) of C and finally in section 3.6 the information gained previously is put together to construct the local–graphs–data–structure for $P \cap C$.

3.1 Intersecting Two Local Pyramids

We show how to compute $G_{P\cap C}(x)$ from $G_P(x)$ and $G_C(x)$ for a point x and how to classify x with respect to $P \cap C$ in time $O((|G_P(x)|+|G_C(x)|)\cdot\log(|G_P(x)|+|G_C(x)|))$. First sweep a plane through the sphere centered at x and intersect the two graphs $G_P(x)$ and $G_C(x)$. Let $G(x)$ be the resulting graph. We also compute the label inP for each feature of $G(x)$ and for x. Since $G_C(x)$ is either trivial (if x is not a vertex of C) or corresponds to a convex cone, any arc of $G_P(x)$ can intersect at most two arcs of $G_C(x)$. Thus $|G(x)| = O(|G_P(x)| + |G_C(x)|)$ and hence $G(x)$ can be computed within the time bound stated above. An alternative strategy for computing $G(x)$ is to intersect any pair of features of the two local graphs. This takes time $O(|G_P(x)| \cdot |G_C(x)|)$ and is to be preferred if one of the two local graphs has constant size.

The graph $G(x)$ determines the local pyramid of x with respect to $P \cap C$ but it may contain spurious features, i.e., features violating one of the conditions (b) to (d) of Lemma 2.3. Simply remove all these features (ensure condition (b) first, then (c) and finally (d)) and obtain $G_{P\cap C}(x)$. Lemma 2.4 can then be used to classify the point x. Simplification and classification take time $O(|G(x)|)$.

3.2 Vertices

Let v be a vertex of P. Locate v with respect to C in time $O(\log|C|)$. This decides whether v belongs to C and determines the local graph $G_C(v)$. Next compute $G_{P\cap C}(v)$ as described in section 3.1 and classify v.

All of this takes time $O(|G_P(v)| + \log |C|)$ except if v is also a vertex of C in which case it takes time $O((|G_P(v)| + |G_C(v)|) \cdot \log N)$. Since a vertex of C can coincide with at most one vertex of P, summation over all vertices of P shows that this step stays within the target time.

3.3 Edges

Let e be an edge of P. First intersect *aff*(e) with C in time $O(\log |C|)$ and obtain either an empty intersection or a line segment $\overline{uv}$ ($u = v$ is possible) contained in *aff*(e). Intersect $\overline{uv}$ with e and obtain a number of subsegments of $\overline{uv}$. All endpoints of these subsegments except maybe u and v are vertices of P and hence were already treated in section 3.2. Next determine the local graphs $G_C(u), G_C(v)$, and $G_C(x)$ where x is an arbitrary point between u and v[5], and from this and the local graph of e compute $G_{P\cap C}(u), G_{P\cap C}(v)$, and $G_{P\cap C}(x)$.

The computation of the local graphs takes time $O(|G_P(e)|)$ except if either u or v is a vertex of either P or C. The vertices of P were already accounted for in section 3.2. For vertices u (or v) of C that are not also vertices of P the time required is $O((|G_P(e)| + |G_C(u)|) \cdot \log(|G_P(e)| + |G_C(u)|))$. Since any vertex of C can lie on at most one edge of P (recall that edges are relatively open sets) we conclude that the total cost of treating the edges of P is within our target.

3.4 Facets

Let f be a facet of P. It is given as a polygonal subset of *aff*(f) and by its local graph $G_P(f)$. Our goal is to intersect f with all faces of C.

There is a simple but inefficient way to do so. *Aff*(f) intersects C in a convex polygonal region whose relative interior we call R. We could explicitly compute R, then use plane sweep to intersect f and R, and finally compute the local graphs of all features of the intersection by the method of section 3.1. Unfortunately, this approach exceeds our target time (see Figure 5 for an example).

We now describe an alternative approach that only looks at those parts of R that actually contribute to the output. First test first *aff*(f) and C for intersection and determine an arbitrary point z of R (if any). This has cost $O(\log |C|)$ per facet of P and hence total cost $O(|P| \cdot \log N)$. If R is empty then we are done. Assume from now on that R is non-empty. Depending on the dimension of R we proceed differently.

Case 1: R is a vertex of C, say v. In this case, determine whether v belongs to f in time $O(|f|)$. If $v \in f$ then compute $G_{P\cap C}(v)$ and classify v in time $O(G_C(v))$: either $v \in ext(P \cap C)$ or v is a vertex of $P \cap C$. Since any vertex of C can lie in at most one facet of P the total cost of this case is $O(|C|)$.

Case 2: R is an edge of C, say e. We can compute $e \cap f$ as follows. Sort intersection points of e with *bd* f. This divides the edge into subsegments. The endpoints of these subsegments except maybe e's endpoints are vertices of P or the intersection with edges of P. The number of subsegments is $O(|f|)$. The total cost of computing all

[5] $G_C(x)$ does not depend on the choice of x.

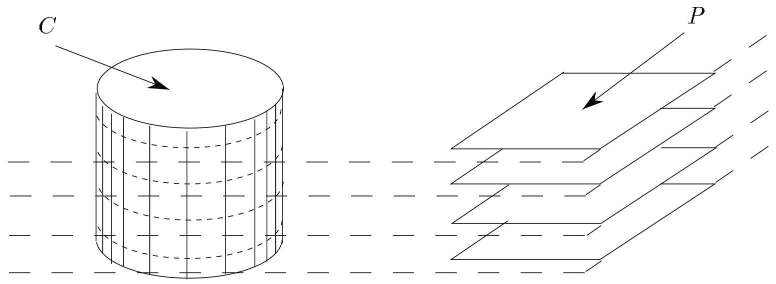

Figure 5: C is a polygonal cylinder with n facets and P consists of m disjoint rectangles contained in parallel planes orthogonal to the axis of C. Then the naive approach takes time $\Omega(m \cdot n)$ although P and C are disjoint.

the $e \cap f$'s is therefore $O(|P| \log N)$. We next compute the local graphs of all points in the point set $e \cap f$ and on its boundary. Consider first a point $x \in e \cap f$. The local graph $G_{P \cap C}(x)$ does not depend on x and can be computed from $G_P(f)$ and $G_C(e)$ in constant time (since $G_P(f)$ and $G_C(e)$ both have constant size). Thus, the local graphs of all the $e \cap f$'s can be computed in time $O(|P|)$. Consider next a point x in the boundary of $e \cap f$. If x is either a vertex of P or belongs to an edge of P then $G_{P \cap C}(x)$ is already known. Otherwise, x is also a vertex of C and $G_{P \cap C}(x)$ can be computed from $G_C(x)$ in time $O(|G_C(x)|)$. Since every vertex of C is contained in at most one facet of P, the total cost for computing the local graphs of boundary points is $O(|C|)$.

Case 3: R is a facet of C, say g. The boundary of f consists of one or more polygons (not necessarily simple). The information gained in the previous sections permits to classify these polygons into three classes: those that intersect *bd* g, those that are contained in *bd* g, and those that either contain *bd* g or are disjoint from *bd* g and do not contain it. The last class can be split by locating an arbitrary point of g, e.g., the point z determined above, with respect to each polygon in the class. The classification process takes time $O(\log |C| + |f|)$ for the facet f and hence total time $O(|P| \log N)$.

We are now in a position to compute $s = f \cap g$. Note that all intersections between *bd* f and g are known at this point, and if *bd* g does not intersect *bd* f it is also known whether *bd* g contributes to *bd* s at all. Sort for each edge of g (that is intersected at least once) its intersection points with *bd* f (this divides the edge into segments) in time $O((|f| + |s|) \cdot \log(|f| + |s|))$. Then explore *bd* s starting at the intersection points between *bd* f and *bd* g in time linear to the size of s. Since the facet g is contained in *aff*(f) for at most six facets f of P[6] and $|s| = O(|f| + |g|)$[7], the total cost of computing all the s's is $O((|P| + |C|) \cdot \log N)$.

[6] Note that $g \subseteq \mathit{aff}(f) \cap \mathit{aff}(s')$ implies $\mathit{aff}(f) = \mathit{aff}(s')$.
[7] since g is convex

At this point we have computed the point set s. We next compute the local graphs of all points in this set and on its boundary. Consider first a point $x \in s$. The local graph $G_{P\cap C}(x)$ does not depend on x and can be computed from $G_P(f)$ and $G_C(g)$ (since $G_P(f)$ and $G_C(g)$ both have constant size). The computation of the local graphs of all the s's takes time $O(|P|)$. Consider next a point x in the boundary of s. If x is either a vertex of P or belongs to an edge of P then $G_{P\cap C}(x)$ is already known. Otherwise, $x \in f \cap \mathit{bd}\ g$ and $G_{P\cap C}(x)$ can be computed from $G_C(x)$ in time $O(|G_C(x)|)$. For the computation of the local graphs of the boundary points observe first that $x \in f \cap \mathit{bd}\ g$ implies that x is either a vertex of C or belongs to an edge of C. For the vertices we argue as in the previous paragraph. For a point x on an edge of C the local graph $G_{P\cap C}(x)$ can be computed in constant time and hence time $O(|s|)$ suffices for all edges of C contributing to the boundary of s. The total cost computing the local graphs of boundary points is therefore $O(|P| + |C|)$.

Case 4: R is a cross section of C. The classification of the polygons that contribute to the boundary of f is the same as in *Case 3.* To compute $s = f \cap R$ we use the procedure described in *Case 3.* The exploration of *bd* s is, however, different since we have no explicit representation of R. During the exploration we assume that *bd* C is triangulated which can be done in time $O(|C|)$. Now, explore *bd* s starting at the intersection points between *bd* f and *bd* R as follows. Let x be such an intersection point. We want to construct the at most two edges of $f \cap \mathit{bd}\ R$ incident to x. We use the information of the local pyramid $G_{P\cap C}(x)$ (which is already known) to get the directions of edges $f \cap \mathit{bd}\ R$ incident to x in constant time and from this the vertices adjacent to x. As we go we may encounter vertices that are not yet known, i.e. 0-dimensional intersections of $f \cap \mathit{bd}\ R$. Finding the incident edges for such a vertex y takes in total at most $O(|G_C(y)|)$ time. Note that only those parts of *bd* R are inspected that actually contribute to *bd* s. Furthermore, since R is a convex polygon, we visit a vertex of $f \cap \mathit{bd}\ R$ at most twice.

The local pyramids of s and *bd* s are computed as in Case 3. We still need to argue that we stay within our target time. If R is a cross section of C then all edges and vertices of *bd* s are also edges and vertices of $P \cap C$. The total cost of computing all the s's and their local graphs is therefore $O((|P| + |C| + |P \cap C|) \cdot \log N)$. For the local graphs of the boundary points $x \in f \cap \mathit{bd}\ R$ observe that $G_{P\cap C}(x)$ can be computed in constant time if x is not a vertex of C and in time $O(|G_C(x)|)$ if x is a vertex of C, and that any vertex of C can be contained in at most one facet of P. The total cost of computing local graphs of boundary points is thus $O(|P|+|C|+|P\cap C|)$.

3.5 Full–dimensional Faces

We show how to compute the intersections between $int(P)$ (the other three–dimensional face of P is not important) and the faces of C. We already know *bd* $P \cap$ *bd* C (and *bd* $P \cap$ *int* C) at this point and also the local graph for each point in *bd* $P \cap$ *bd* C.

Superimpose *bd* $P \cap$ *bd* C on *bd* C and obtain a planar graph embedded into the boundary of C. A traversal of the graph yields all points in *int* $P \cap$ *bd* C. The local graphs of the points in *int* $P \cap$ *bd* C are just their local graphs with respect to C.

3.6 Putting It All Together

At this point we have computed all intersection between faces of P and C involving at least one low–dimensional face and hence know $bd(P \cap C)$. We also know the local graphs of all points on the boundary. We now have to build the low–dimensional faces of $P \cap C$. According to Lemma 3.2 each face of $P \cap C$ is the union of intersections of faces of P and faces of C. We still have to perform the unions.

Let us consider the vertices first. For some vertices of P it may be the case that the new local graph indicates that the vertex now belongs to an edge or facet. If the vertex now belongs to an edge we join the two edge segments incident to the vertex[8]. If the vertex now belongs to a facet then we remove it from the description of the facet.

Consider the edges next. Edges that become part of a facet are removed from the description of the facet. We also have to determine whether several edges have to be merged into one[9]. Describe each edge by its affine hull and by a suitable encoding of its local graph (e.g., a list consisting of the arcs and the regions) and determine all edges with the same description (e.g., by building a trie [Meh84, section III.1.1] for the descriptions). The nodes of this trie are dictionaries and therefore it takes time $O(l \log N)$ to insert a description of length l into the trie). Then unite all edges with the same description.

Finally, consider the facets. Determine all facets with the same affine hull and the same local graph (use a dictionary) and unite these facets (e.g., by sweeping them).

The local–graphs–data–structure of $P \cap C$ is now available. The assembly phase stays within our target time since it essentially boils down to a constant number of dictionary operations for each feature of P, C, and $P \cap C$.

[8]These segments may belong to the same edge or to different edges.
[9]These edges do not necessarily share an endpoint.

References

[BN88] H. Bieri and W. Nef. Elementary set operations with d-dimensional polyhedra. In *Computational Geometry and its Applications*, volume 333 of *Lecture Notes in Computer Science*, pages 97–112. Springer-Verlag, 1988.

[Cha92] B. Chazelle. An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM J. Comput.*, 21(4):671–696, 1992.

[DK83] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoret. Comput. Sci.*, 27:241–253, 1983.

[DMY93] K. Dobrindt, K. Mehlhorn, and M. Yvinec. Manuscript, 1993.

[Dob90] K. Dobrindt. *Algorithmen für Polyeder.* Master's thesis, Fachbereich Informatik, Universität Saarbrücken, June 1990.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Heidelberg, West Germany, 1987.

[EM85] H. Edelsbrunner and H. A. Maurer. Finding extreme points in three dimensions and solving the post-office problem in the plane. *Inform. Process. Lett.*, 21:39–47, 1985.

[EM90] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9:66–104, 1990.

[GS85] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4:74–123, 1985.

[HMMN84] S. Hertel, M. Mäntylä, K. Mehlhorn, and J. Nievergelt. Space sweep solves intersection of convex polyhedra. *Acta Inform.*, 21:501–519, 1984.

[Hof89] C. Hoffmann. *Geometric and Solid Modeling.* Morgan Kaufmann, San Mateo, California, 1989.

[Män88] M. Mäntylä. *An introduction to solid modeling.* Computer Science Press, Rockville, Md., 1988.

[Meh84] K. Mehlhorn. *Sorting and Searching*, volume 1 of *Data Structures and Algorithms.* Springer-Verlag, Heidelberg, West Germany, 1984.

[MP78] D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoret. Comput. Sci.*, 7:217–236, 1978.

[MS85] K. Mehlhorn and K. Simon. Intersecting two polyhedra one of which is convex. In L. Budach, editor, *Proc. Found. Comput. Theory*, volume 199 of *Lecture Notes in Computer Science*, pages 534–542. Springer-Verlag, 1985.

[Nef78] W. Nef. *Beiträge zur Theorie der Polyeder.* Herbert Lang, Bern, 1978.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

[Req80] A. A. G. Requicha. Representations of rigid solids: theory, methods, and systems. *ACM Comput. Surv.*, 12:437–464, 1980.

A Appendix

Let M a subset of $\mathbb{R}^d$. We will denote the complement of M by $cpl(M)$.

Definition A.1 The *affine hull* $\mathit{aff}(M)$ of M is the intersection of all flats $N \subseteq \mathbb{R}^d$, which contain M. The *closure* $clos(M)$ of M is the intersection of all closed supersets of M. The *interior* $int(M)$ of M is the union of all open subsets of M. The *exterior* $ext(M)$ of M is defined as $int(cpl(M))$. The *relative interior* $rel\ int(M)$ of a point set M is the union of all relatively open subsets of M, i.e., open with respect to the affine hull of M.

Definition A.2 M is *regular*, if $M = clos(int(M))$ and if for each element $x \in M$, there is a neighborhood U of x in $\mathbb{R}^d$ such that $int(U \cap M)$ and $ext(U \cap M)$ are connected.

In other words, a set M in $\mathbb{R}^d$ is regular, if M is closed, does not contain dangling or isolated parts of dimension $< d$, and the interior and the exterior of M are connected in a neighborhood of each element of M.

Unité de recherche INRIA Lorraine, Technôpole de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399