Chapter 8

Incremental convex hulls

To compute the convex hull of a finite set of points is a classical problem in computational geometry. In two dimensions, there are several algorithms that solve this problem in an optimal way. In three dimensions, the problem is considerably more difficult. As for the general case of any dimension, it was not until 1991 that a deterministic optimal algorithm was designed. In dimensions higher than 3, the method most commonly used is the incremental method. The algorithms described in this chapter are also incremental and work in any dimension. Methods specific to two or three dimensions will be given in the next chapter.

Before presenting the algorithms, section 8.1 details the representation of polytopes as data structures. Section 8.2 shows a lower bound of $\Omega(n \log n + n^{\lfloor d/2 \rfloor})$ for computing the convex hull of n points in d dimensions. The basic operation used by an incremental algorithm is: given a polytope $\mathcal{C}$ and a point P, derive the representation of the polytope $conv(\mathcal{C} \cup \{P\})$ assuming the representation of $\mathcal{C}$ has already been computed. Section 8.3 studies the geometric part of this problem. Section 8.4 shows a deterministic algorithm to compute the convex hull of n points in d dimensions. This algorithm requires preliminary knowledge of all the points: it is an *off-line* algorithm. Its complexity is $O(n \log n + n^{\lfloor (d+1)/2 \rfloor})$, which is optimal only in even dimensions. In section 8.5, the influence graph method explained in section 5.3 is used to obtain a semi-dynamic algorithm which allows the points to be inserted on-line. The randomized analysis of this algorithm shows that its average complexity is optimal in all dimensions. Finally, section 8.6 shows how to adapt the augmented influence graph method of chapter 6 to yield a fully dynamic algorithm for the convex hull problem, allowing points to be inserted or deleted on-line. The expected complexity of an insertion or deletion is $O(\log n + n^{\lfloor d/2 \rfloor - 1})$, which is optimal.

Throughout this chapter, we assume that the set of points whose convex hull is to be computed is in *general position*. This means that any subset of $k+1 \leq d+1$ points generates an affine subspace of dimension k. This hypothesis is not crucial

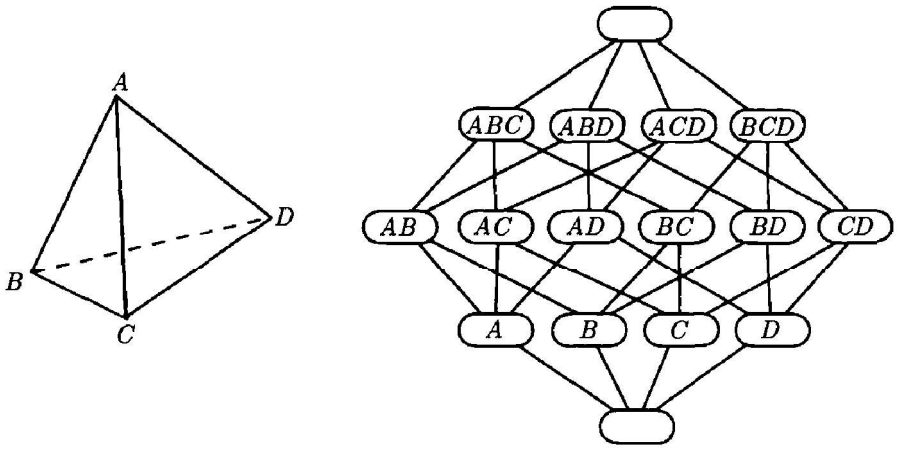

Figure 8.1. A tetrahedron and its incidence graph.

for the deterministic algorithm (see exercise 8.4), but it allows us to simplify the description of the algorithm and to focus on the central ideas. It becomes an essential assumption, however, for the randomized analyses of the on-line and dynamic algorithms.

8.1 Representation of polytopes

To compute the convex hull of a set of points amounts to setting up a data structure that represents the polytope which is the convex hull of the set. A polytope is generally represented by the *incidence graph* of its faces, which stores a node for each face and an arc for each pair of incident faces. Recall that two faces are incident if their dimensions differ by one and if one is contained in the other. Figure 8.1 shows the incidence graph of a tetrahedron.

Using the upper bound theorem 7.2.5, the incidence graph of a d-polytope can be stored using $O(n^{\lfloor d/2 \rfloor})$ space. This graph describes the entire combinatorial structure of the polytope. In order to describe its geometric structure, some additional information has to be stored: for instance, the node storing a vertex contains the coordinates of that vertex, and the node storing a facet contains the coefficients in an equation of the hyperplane that supports the polytope along that facet.

Sometimes, it may be enough to store subgraphs of the incidence graph. The *j-skeleton* of a polytope is the subgraph of the incidence faces of dimension at most j. The 1-skeleton of a polytope is simply made up of the vertices and edges of that polytope.

In a d-polytope, every $(d-2)$-face is incident to exactly two $(d-1)$-faces

(theorem 7.1.7); two $(d-1)$-faces of a polytope are said to be *adjacent* if they are incident to a common $(d-2)$-face. Thus, the incidence graph of a polytope also encodes the *adjacency graph*, which has a node for each facet and an arc for each pair of adjacent facets. The arcs of the adjacency graph are in one-to-one correspondence with the $(d-2)$-faces of the polytope. If the polytope is simplicial, the full incidence graph can be retrieved from the adjacency graph in time linear in the number of faces (see exercise 8.2).

8.2 Lower bounds

Theorem 8.2.1 *The complexity of computing the convex hull of n points in d dimensions is $\Omega(n \log n + n^{\lfloor d/2 \rfloor})$.*

Proof. Subsection 7.2.4 shows that the convex hull of n points in the Euclidean space $\mathbb{E}^d$ may have $\Omega(n^{\lfloor d/2 \rfloor})$ faces. In any dimension, $\Omega(n^{\lfloor d/2 \rfloor})$ is thus a trivial lower bound for the complexity of computing convex hulls. In two dimensions, the lower bound $\Omega(n \log n)$ is a consequence of theorem 8.2.2 proved below. Finally, any set of points in $\mathbb{E}^2$ can be embedded into $\mathbb{E}^3$, so the complexity of computing convex hulls in $\mathbb{E}^3$ cannot be smaller than in $\mathbb{E}^2$. □

Theorem 8.2.2 *The problem of sorting n real numbers can be transformed in linear time into the problem of computing the convex hull of n points in $\mathbb{E}^2$.*

Proof. Consider n real numbers $x_1, x_2, \ldots, x_n$, which we want to sort. One way to do this is to map the number x_i to the point A_i with coordinates (x_i, x_i^2) on the parabola with equation $y = x^2$ (see figure 8.2). The convex hull of the set of points $\{A_i : i = 1, \ldots, n\}$ is a cyclic 2-polytope, and the list of its vertices is exactly the list of the vertices $\{A_i : i = 1, \ldots, n\}$ ordered according to their increasing abscissae. □

8.3 Geometric preliminaries

The incremental method for computing the convex hull of a set $\mathcal{A}$ of n points in $\mathbb{E}^d$ consists in maintaining the succession of convex hulls of the consecutive sets obtained by adding the points in $\mathcal{A}$ one by one. Each convex hull is represented by its incidence graph. Let $\mathcal{C}$ be the convex hull of the current subset and P the point to be inserted next into the structure, at a given step in the algorithm. The problem is thus to obtain the incidence graph of $conv(\mathcal{C} \cup \{P\})$, once we know that of $\mathcal{C}$. The following lemmas clarify the relations existing between these two graphs.

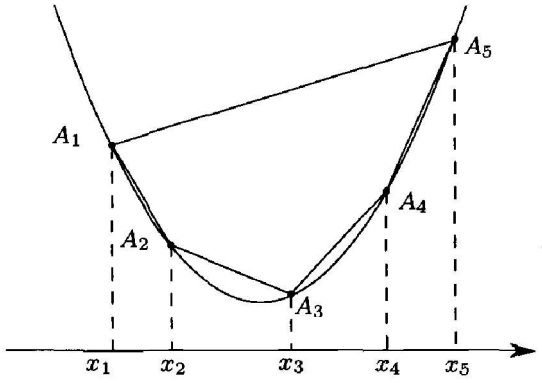

Figure 8.2. Transforming a sorting problem into a convex hull problem in two dimensions.

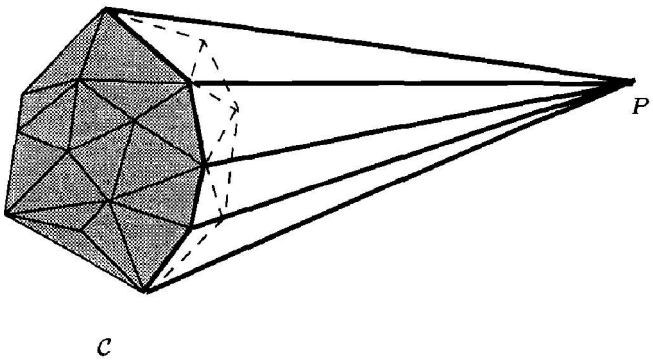

Figure 8.3. The incremental construction of a convex hull.

Suppose that point P and polytope C are in *general position*, meaning that P and the vertices of C form a set of points in general position. The facets of C can then be separated into two classes with respect to P. Let F be a facet of C, H_F the hyperplane that supports C along F, and H_F^+ (resp. H_F^-) the half-space bounded by H_F that contains (resp. does not contain) C. The facet F is *red* with respect to P if it is visible from point P, that is if P belongs to the half-space H_F^-. It is colored *blue* if P belongs to H_F^+. From the general position assumption, it follows that P never belongs to the supporting hyperplane H_F and therefore every facet of C is either red or blue with respect to P.

Using theorem 7.1.7, any face of C is the intersection of the facets of C which contain it. The faces of C of dimension strictly smaller than $d-1$ can be separated into three categories with respect to P: a face of C is *red* if it is the intersection

of red facets only, *blue* if it is the intersection of blue facets only, or *purple* if it is the intersection of red and blue facets.

Intuitively, the red faces are those that would be lit if a point source of light was shining from P, the blue faces are those that would remain in the shadow, and the purple faces would be lit by rays tangent to $\mathcal{C}$. In figure 8.3, the blue faces of $\mathcal{C}$ are shaded, the red edges are outlined in dashed lines, and the purple edges are shown in bold.

Lemma 8.3.1 *Let $\mathcal{C}$ be a polytope and P a point in general position with respect to $\mathcal{C}$. Every face of $conv(\mathcal{C} \cup \{P\})$ is either a blue or purple face of $\mathcal{C}$, or the convex hull $conv(G \cup \{P\})$ of P and a purple face G of $\mathcal{C}$.*

Proof. Note that if P belongs to $\mathcal{C}$, all the facets of $\mathcal{C}$ are blue with respect to $\mathcal{C}$ (theorem 7.1.4) and the content of the lemma is trivial.

In the other case, we first show that a blue face of $\mathcal{C}$ is a face of $conv(\mathcal{C} \cup \{P\})$. Let F be a facet of $\mathcal{C}$ that is blue with respect to P. Since P belongs to the half-space H_F^+, the hyperplane H_F which supports $\mathcal{C}$ along F also supports $conv(\mathcal{C} \cup \{P\})$ and $conv(\mathcal{C} \cup \{P\}) \cap H_F = F$, which proves that F is indeed a facet of $conv(\mathcal{C} \cup \{P\})$. Any blue facet of $\mathcal{C}$ is thus a facet of $conv(\mathcal{C} \cup \{P\})$. Any blue face of $\mathcal{C}$, being the intersection of blue facets of $\mathcal{C}$, is also the intersection of facets of $conv(\mathcal{C} \cup \{P\})$: therefore a blue face of $\mathcal{C}$ is also a face of $conv(\mathcal{C} \cup \{P\})$ (theorem 7.1.7).

Next we show that, for any purple face G of $\mathcal{C}$, G and $conv(G \cup \{P\})$ are faces of $conv(\mathcal{C} \cup \{P\})$. If G is a purple face of $\mathcal{C}$, then there is at least one red facet of $\mathcal{C}$, say F_1, and one blue facet of $\mathcal{C}$, say F_2, that both contain G (see figure 8.4). Let H_1 (resp. H_2) be the hyperplane supporting $\mathcal{C}$ along F_1 (resp. F_2). Point P belongs to the half-space H_1^+ which contains $\mathcal{C}$, and since $H_1 \cap conv(\mathcal{C} \cup \{P\}) = G$ we have shown that G is a face of $conv(\mathcal{C} \cup \{P\})$. Point P also belongs to the half-space H_2^- that does not contain $\mathcal{C}$. Imagine a hyperplane that rotates around $H_1 \cap H_2$ while supporting $\mathcal{C}$ along G. There is a position H for which this hyperplane passes through point P. Hyperplane H supports $conv(\mathcal{C} \cup \{P\})$, and since $conv(\mathcal{C} \cup \{P\}) \cap H = conv(G \cup \{P\})$, we have proved that $conv(G \cup \{P\})$ is a face of $conv(\mathcal{C} \cup \{P\})$.

Finally, let us show that every face of $conv(\mathcal{C} \cup \{P\})$ is either a blue or a purple face of $\mathcal{C}$, or the convex hull $conv(G \cup \{P\})$ of P and of a purple face G of $\mathcal{C}$. Indeed, a hyperplane that supports $conv(\mathcal{C} \cup \{P\})$ is also a supporting hyperplane of $\mathcal{C}$, unless it intersects $conv(\mathcal{C} \cup \{P\})$ only at point P. As a consequence, any face of $conv(\mathcal{C} \cup \{P\})$ that does not contain P is a (blue or purple) face of $\mathcal{C}$, and any face $conv(\mathcal{C} \cup \{P\})$ that contains P is of the form $conv(G \cup \{P\})$ where G is a purple face of $\mathcal{C}$. Note that the vertex P of $conv(\mathcal{C} \cup \{P\})$ is also a face of the form $conv(G \cup \{P\})$ obtained when G is the empty face of $\mathcal{C}$. Indeed, when

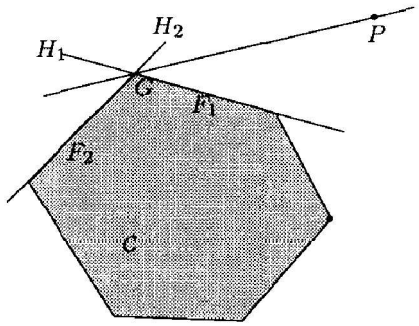

Figure 8.4. Faces of $conv(G \cup \{P\})$.

P does not belong to $\mathcal{C}$, $\mathcal{C}$ necessarily has some facets that are blue and some facets that are red with respect to P. The empty face, being the intersection of all faces of $\mathcal{C}$, is therefore purple. □

The following lemma, whose proof is straightforward, investigates the incidence relationships between the faces of $\mathcal{C}$ and those of $conv(\mathcal{C} \cup \{P\})$.

Lemma 8.3.2 *Let $\mathcal{C}$ be a polytope and P a point in general position with respect to $\mathcal{C}$.*

- *If F and G are two incident faces of polytope $\mathcal{C}$, either blue or purple with respect to P, then F and G are incident faces of $conv(\mathcal{C} \cup \{P\})$.*
- *If G is a purple face of $\mathcal{C}$, then G and $conv(G \cup \{P\})$ are incident faces of $conv(\mathcal{C} \cup \{P\})$.*
- *Finally, if F and G are incident purple faces of $\mathcal{F}$, then $conv(F \cup \{P\})$ and $conv(G \cup \{P\})$ are incident faces of $conv(\mathcal{C} \cup \{P\})$.*

Recall that two facets of a polytope $\mathcal{C}$ are adjacent if they are incident to the same $(d-2)$-face and that the adjacency graph of a polytope stores a node for each facet and an arc for each pair of adjacent facets.[1] We say that a subset of facets of a polytope $\mathcal{C}$ is *connected* if it induces a connected subgraph of the adjacency graph of $\mathcal{C}$.

Lemma 8.3.3 *Consider a polytope $\mathcal{C}$ and a point P in general position. The set of facets of $\mathcal{C}$ that are red with respect to P is connected, and the set of facets of $\mathcal{C}$ that are blue with respect to P is also connected.*

[1]Two facets sharing a common k-face, $k < d-2$, may be not adjacent, even though they are connected as a topological subset of the boundary of the polytope. Such a situation is only possible in dimension $d \geq 3$.

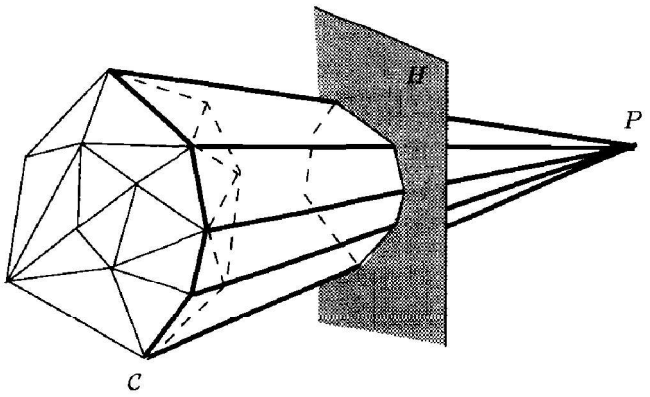

Figure 8.5. Isomorphism between the purple faces and the faces of a $(d-1)$-polytope.

Proof. If P belongs to C, the set of the red facets is empty, any facet is blue, and the lemma is trivial. We will therefore assume that P does not belong to C.

The connectedness of the set of red facets can be proved easily in two dimensions. Indeed, the polytope $conv(C \cup \{P\})$ has two edges incident to P. By lemma 8.3.1, there are exactly two purple vertices of C with respect to P. Hence, the adjacency graph of the 2-polytope C is a cycle that has exactly two arcs connecting a blue and a red facet.

Let us now discuss the case of dimension d, and suppose for a contradiction that the set of facets of C that are red with respect to P is not connected. Therefore, we may choose two points Q and R on two facets of C that belong to two distinct connected components of the set of red facets of C. Let H be the affine 2-space passing through points P, Q, and R. This plane intersects polytope C along a 2-polytope $\mathcal{H} \cap H$. The edges of $C \cap H$ that are red with respect to P are exactly the intersections of the red facets of C with H. The points Q and R belong to two separate connected components of the set of red edges of $C \cap H$. Connectedness of the set of red faces of a 2-polytope would then not hold, a contradiction.

Analogous arguments prove the connectedness of the set of facets of $conv(C \cup \{P\})$ that are blue with respect to P. □

Finally, the lemma below completely characterizes the subgraph of the incidence graph induced on the faces of C that are purple with respect to P.

Lemma 8.3.4 *Let C be a polytope and P a point in general position with respect to C. If C has n vertices and does not contain P, then the set of the proper faces of C that are purple with respect to P is isomorphic, for the incidence relationship, to the set of faces of a $(d-1)$-polytope whose number of vertices is at most n.*

Proof. From lemma 8.3.1, we know that the faces of polytope C that are purple with respect to P are in one-to-one correspondence with the faces of $conv(C \cup \{P\})$

that do not contain P. Since point P does not belong to $\mathcal{C}$, there must be a hyperplane H which separates P from $\mathcal{C}$ (see exercise 7.4). Hyperplane H intersects all the faces of $conv(\mathcal{C} \cup \{P\})$ that contain P except for the vertex P, and those faces only. Moreover, the traces in H of the faces of $conv(\mathcal{C} \cup \{P\})$ are the proper faces of the $(d-1)$-polytope $conv(\mathcal{C} \cup \{P\}) \cap H$, and the traces in H of incident faces of $conv(\mathcal{C} \cup \{P\})$ are incident faces of $conv(\mathcal{C} \cup \{P\}) \cap H$. Thus, the incidence graph of the $(d-1)$-polytope $conv(\mathcal{C} \cup \{P\}) \cap H$ is isomorphic to the subgraph of the incidence graph of $conv(\mathcal{C} \cup \{P\})$ induced by the faces that contain vertex P. Lemmas 8.3.1 and 8.3.2 show that this subgraph is isomorphic to the subgraph of the incidence graph of $\mathcal{P}$ induced by the faces of $\mathcal{C}$ that are purple with respect to P. Lastly, the vertices of polytope $conv(\mathcal{C} \cup \{P\}) \cap H$ are the traces in H of the edges of $conv(\mathcal{C} \cup \{P\})$ incident to vertex P, and their number is at most n. □

8.4 A deterministic algorithm

In this section we describe an incremental deterministic algorithm to build the convex hull of a set $\mathcal{A}$ of n points. The points in $\mathcal{A}$ are processed in increasing lexicographic order of their coordinates. To simplify the description of the algorithm, we assume below that the set is in general position. We denote by $\{A_1, A_2, \ldots, A_n\}$ the points of $\mathcal{A}$ indexed by lexicographic order. Let $\mathcal{A}_i$ be the set of the first i points of $\mathcal{A}$.

The general idea of the algorithm is as follows:

1. Sort the points of $\mathcal{A}$ in increasing lexicographic order of their coordinates.
2. Initialize the convex hull to the simplex $conv(\mathcal{A}_{d+1})$, the convex hull of the first $d+1$ points of $\mathcal{A}$.
3. In the incremental step: the convex hull of $conv(\mathcal{A}_i)$ is built knowing the convex hull $conv(\mathcal{A}_{i-1})$ and the point A_i to be inserted.

Details of the incremental step

Because of the lexicographic order on the points of $\mathcal{A}$, point A_i never belongs to the convex hull $conv(\mathcal{A}_{i-1})$, and is therefore a vertex of $conv(\mathcal{A}_i)$. The preceding lemmas show that the subgraph of the incidence graph of $conv(\mathcal{A}_{i-1})$ restricted to the faces that are blue with respect to A_i is also a subgraph of the incidence graph of $conv(\mathcal{A}_i)$. All the efficiency of the incremental algorithm stems from the fact that the incidence graph of the current convex hull can be updated in an incremental step without looking at the blue faces or at their incidences.

To perform this incremental step, we proceed in four phases:

Phase 1. We first identify a facet of $conv(\mathcal{A}_{i-1})$ that is red with respect to A_i.

Phase 2. The red facets and the red or purple $(d-2)$-faces of $conv(\mathcal{A}_{i-1})$ are traversed. A separate list is set up for the red facets, the red $(d-2)$-faces, and the purple $(d-2)$-faces.

Phase 3. Using the information gathered in phase 2, we identify all the other red or purple faces of $conv(\mathcal{A}_{i-1})$. For each dimension k, $d-3 \geq k \geq 0$, a list $\mathcal{R}_k$ of the red k-faces is computed, as well as a list $\mathcal{P}_k$ of the purple k-faces.

Phase 4. The incidence graph is updated.

Before giving all the details for each phase, let us first describe precisely the data structure that stores the incidence graph. For each face F of dimension k $(0 \leq k \leq d-1)$ of the convex hull, this data structure stores:

- the list of the *sub-faces* of F, which are the faces of dimension $k-1$ incident to F,
- the list of the *super-faces* of F, which are the faces of dimension $k+1$ incident to F,
- the color of the face (red, blue, purple) in the current step, and
- a pointer $p(F)$ whose use will very soon be clarified.

If F is a super-face of G, then a bidirectional pointer links the record for F in the list of super-faces of G to the record for G in the list of sub-faces of F.

Phase 1. To find an initial red facet in $conv(\mathcal{A}_{i-1})$, we take advantage of the lexicographic order on the points in $\mathcal{A}$. Because of this order, A_{i-1} is always a vertex of $conv(\mathcal{A}_{i-1})$ and there is at least one facet of $conv(\mathcal{A}_{i-1})$ containing A_{i-1} which is red with respect to A_i. Indeed, let $\mathcal{F}_{i-1}$ be the set of facets of $conv(\mathcal{A}_{i-1})$ that contain A_{i-1} as a vertex. Let also H be the hyperplane whose equation is $x_1 = x_1(A_{i-1})$, and H^+ the half-space bounded by H that contains $conv(\mathcal{A}_{i-1})$, and H^- the other half-space bounded by H. Since A_{i-1} is a vertex of $conv(\mathcal{A}_{i-1})$ with maximal abscissa, $H^+ \cup A_{i-1}$ contains the intersection of all the half-spaces H_F^+ when $F \in \mathcal{F}_{i-1}$. Point A_i belongs to $\overline{H^-}$, and therefore cannot belong to this intersection of half-spaces (see figure 8.6). Thus, at least one facet F in $\mathcal{F}_{i-1}$ must be red with respect to A_i. All the facets of $\mathcal{F}_{i-1}$ were created at the previous incremental step, so it suffices to store the list of facets created during an incremental step and to traverse this list during the next incremental step in order to find an initial red facet.

Phase 2. In the second phase, we use the connectedness of the set of red facets (lemma 8.3.3). A depth-first traversal of the subgraph of red facets in

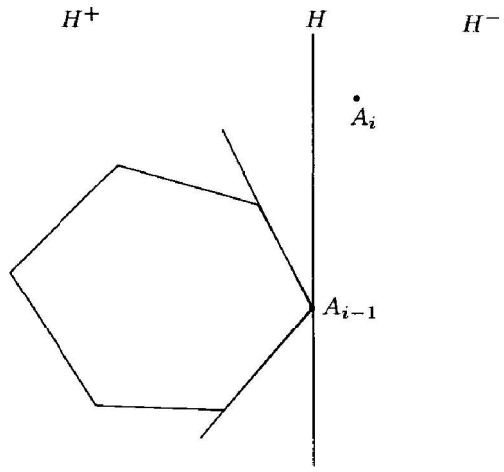

Figure 8.6. One of the facets of $conv(\mathcal{A}_{i-1})$ containing A_{i-1} must be red with respect to A_i.

the adjacency graph[2] of $conv(\mathcal{A}_{i-1})$, starting with the initial red facet that was found in phase 1, visits all the facets visible from A_i, which we color red, and their $(d-2)$-faces, which we color red if they are incident to two red facets, or purple if they are incident to a blue facet. The traversal backtracks whenever the facet encountered was already colored red, or if it is a blue facet.

Phase 3. We now know all the red and purple $(d-2)$-faces, and the red facets. In this phase, all the remaining red and purple faces are colored, and their lists are set up in order of decreasing dimensions. Assume inductively that all the red and purple faces of dimension $k' \geq k+1$ have already been identified and colored, and that the lists $\mathcal{R}_{k'}$ and $\mathcal{P}_{k'}$ have already been set up. We process the k-faces in the following way. Each sub-face of a face of $\mathcal{P}_{k+1}$ that has not yet been colored is colored purple and added to the list $\mathcal{P}_k$. Afterwards, each sub-face of $\mathcal{R}_{k+1}$ that has not yet been colored is added to the list $\mathcal{R}_k$.

Phase 4. To update the incidence graph, we proceed as follows. All the red faces are removed from the incidence graph, and so are all the arcs adjacent to these faces in the graph. The purple faces are processed in order of increasing dimension k. If F is a k-face purple with respect to P, a new node is created for the $(k+1)$-face $conv(F \cup \{A_i\})$ and linked by an arc to the node for F in the incidence graph. Also the pointer $p(F)$ is set to point to the new node created for $conv(F \cup \{A_i\})$. It remains to link this node to all the incident k-faces of the form $conv(G \cup \{A_i\})$, where G is a $(k-1)$-face incident to F. For each sub-face G of F, its pointer $p(G)$ gives a direct access to the node corresponding to $conv(G \cup \{A_i\})$, and the incidence arc can be created.

[2]The adjacency graph is already stored in the incidence graph, and need not be stored separately (see subsection 8.1).

Analysis of the algorithm

Phase 1 of each incremental step can be carried out in time proportional to the number of facets created at the previous step. The total cost of phase 1 over all the incremental steps is thus dominated by the total number of facets created.

At step i that sees the insertion of A_i, the cost of phase 2 is proportional to the number of nodes visited during the traversal of the adjacency graph. The nodes visited correspond to red facets of $conv(\mathcal{A}_{i-1})$, and to the blue facets adjacent to these red facets. The total cost of this phase is thus at most proportional to the number of red facets of $conv(\mathcal{A}_{i-1})$ and of their incidences.

The cost of phase 3 is bounded by (a constant factor times) the number of arcs in the incidence graph that are visited, and this number is the same as the number of incidences between red or purple faces of $conv(\mathcal{A}_{i-1})$.

Lastly, the cost of phase 4 is proportional to the total number of red faces and of their incidences, plus the number of purple faces and of their incidences to purple faces.

In short, when incrementally adding a point to the convex hull, the cost of phases 2, 3, and 4 is proportional to the number of red or purple faces, plus the number of faces incident to a red face, plus the number of incident purple faces. Red faces and their incidences correspond to the nodes and arcs of the incidence graph that are removed from the graph. The purple faces and the incidences between two purple faces correspond to nodes and arcs of the incidence graph that are added to the graph. The total cost of phases 2, 3, and 4 is thus proportional to the number of changes undergone by the incidence graph. Since a node or arc that is removed will not be inserted again (red faces will remain inside the convex hull for the rest of the algorithm), this total number of changes is proportional to the number of arcs and nodes of the incidence graph that are created throughout the execution of the algorithm, which also takes care of the cost of phase 1. The following lemma bounds this number.

Lemma 8.4.1 *The number of faces and incidences created during the execution of an incremental algorithm building the convex hull of n points in d dimensions is $O(n^{\lfloor (d+1)/2 \rfloor})$.*

Proof. Lemma 8.3.1 shows that the subgraph of the incidence graph of $conv(\mathcal{A}_i)$ induced by the faces created upon the insertion of A_i is isomorphic to the set of faces of $conv(\mathcal{A}_{i-1})$ that are purple with respect to A_i. The number of incidences between a new face and a purple face of $conv(\mathcal{A}_{i-1})$ is also proportional to the number of purple faces of $conv(\mathcal{A}_{i-1})$. Finally, lemma 8.3.4 shows that the set of purple faces of $conv(\mathcal{A}_{i-1})$ is isomorphic to a $(d-1)$-polytope that has at most $i-1$ vertices. The upper bound theorem 7.2.5 shows that the number of these faces and incidences between these faces, is $O(i^{\lfloor (d-1)/2 \rfloor})$. This is thus a bound on

the number of faces and incidences created upon inserting A_i. Summing over all i, $i = 1, \ldots, n$, the total number of facets and incidences created by the algorithm is:

$$\sum_{i=1}^{n} O(i^{\lfloor (d-1)/2 \rfloor}) = O(n^{\lfloor (d+1)/2 \rfloor}).$$

□

The storage needed by this algorithm is proportional to the maximum size of the incidence graph stored at any step, which is $O(n^{\lfloor d/2 \rfloor})$. Taking into account the initial sorting of the vertices, we conclude with the following result:

Theorem 8.4.2 *The incremental algorithm builds the convex hull of n points in d dimensions in time $O(n \log n + n^{\lfloor (d+1)/2 \rfloor})$ and storage $O(n^{\lfloor d/2 \rfloor})$.*

This algorithm is optimal in the worst case when the dimension of the space is even.

8.5 On-line convex hulls

Computing the convex hull of a set of points is one of the geometric problems to which the randomization techniques developed in chapter 5 apply. Randomized algorithms compute the convex hull of n points in optimal expected time, in any dimension: $O(n \log n)$ in dimension 2 or 3, and $O(n^{\lfloor d/2 \rfloor})$ in dimension $d > 3$. Let us once again recall that the average value involved here is over all the possible random choices of the algorithm, not over some spatial distribution of the points. The only assumption we make on the points is that they are in general position.

The algorithm which we present here is an incremental on-line algorithm (or semi-dynamic) that uses the influence graph method described in section 5.3, to which we refer the reader if need be. The term "on-line" means that the algorithm is able to maintain the convex hull of a set of points as the points are added one by one without preliminary knowledge of the whole set. This algorithm is in fact deterministic. Only the analysis is randomized and assumes that the order in which the points are inserted is random.

Convex hulls in terms of objects, regions, and conflicts

This section applies the formalism described in chapter 4. In order to do so, we must first recast the convex hull problem in terms of objects, regions, and conflicts.

The *objects* are naturally the points of $\mathbb{E}^d$. A *region* is defined as the union of two open half-spaces. Such a region is determined by a set of $d+1$ points in general position. Let $\{P_0, P_1, \ldots, P_{d-1}, P_d\}$ stand for such a $(d+1)$-tuple. Let H_d be the hyperplane containing $\{P_0, P_1, \ldots, P_{d-1}\}$ and H_d^- be the half-space

bounded by H_d that does not contain P_d. Similarly let H_0 be the hyperplane containing $\{P_1, \ldots, P_{d-1}, P_d\}$ and let H_0^- be the half-space bounded by H_0 that does not contain P_d. The region determined by the $(d+1)$-tuple is the union of the two open half-spaces H_d^- and H_0^-. A point conflicts with a region if it belongs to at least one of the two open half-spaces that make up the region. In this case, the *influence domain* of a region is simply the region itself.

With this definition of regions and conflicts, the convex hull of a set $\mathcal{S}$ of n affinely independent points can be described as the set of regions defined and without conflict over $\mathcal{S}$. In fact, the regions defined and without conflict over $\mathcal{S}$ are in bijection with the $(d-2)$-faces of $conv(\mathcal{S})$. Indeed, let a region be determined by the $(d+1)$-tuple $\{P_0, P_1, \ldots, P_{d-1}, P_d\}$ of points in $\mathcal{S}$. Because the points in $\mathcal{S}$ are assumed to be in general position, if this region is without conflict over $\mathcal{S}$, the two $d-1$ simplices $F_d = conv(\{P_0, P_1, \ldots, P_{d-1}\})$ and $F_0 = conv(\{P_1, \ldots, P_{d-1}, P_d\})$ are facets of $conv(\mathcal{S})$, and the $(d-2)$-simplex $G = F_0 \cap F_d = conv(\{P_1, \ldots, P_{d-1}\})$ is the $(d-2)$-face of $conv(\mathcal{S})$ that is incident to both these facets. This region will be denoted below by (F_0, F_d) or sometimes by (F_d, F_0). The set of regions defined and without conflict over a set $\mathcal{S}$ therefore not only gives the facets of $conv(\mathcal{S})$, but also their adjacency graph. Using this information, it is an easy exercise to build the complete incidence graph of $conv(\mathcal{S})$ in time proportional to the number of faces of all dimensions of $conv(\mathcal{S})$ (see exercise 8.2).[3]

The algorithm

The algorithm is incremental, and in fact closely resembles that which is described in section 8.4. The convex hull $conv(\mathcal{S})$ of the current set $\mathcal{S}$ is represented by its incidence graph. At each step, a new point P is inserted. The faces of $conv(\mathcal{S})$ can be sorted into three categories according to their color with respect to P, as explained in section 8.3: red faces, blue faces, and purple faces. The on-line algorithm, like the incremental algorithm, identifies the faces that are red and purple with respect to P, then updates the incidence graph. The main difference resides in the order with which the points are inserted. The on-line algorithm processes the points in the order given by the input, and therefore cannot take advantage of the lexicographic order to detect the red facets. For this reason, the algorithm maintains an *influence graph*. As we may recall, the influence graph

[3]It would certainly be more natural to define a region as a open half-space determined by d affinely independent points. In this case the region is one of the half-spaces bounded by the hyperplane generated by these d affinely independent points, and a point conflicts with such a region if it lies in this half-space. With these definitions, the facets of the convex hull $conv(\mathcal{S})$ of a set $\mathcal{S}$ of n points in $\mathbb{E}^d$ are in bijection with the regions defined and without conflict over $\mathcal{S}$.

In fact, such a definition of regions is perfectly acceptable and so is an incremental algorithm based on these definitions (see exercise 8.5). Such an algorithm, however, does not satisfy the update conditions 5.2.1 and 5.3.3, and its analysis calls for the notion of biregion introduced in exercise 5.7.

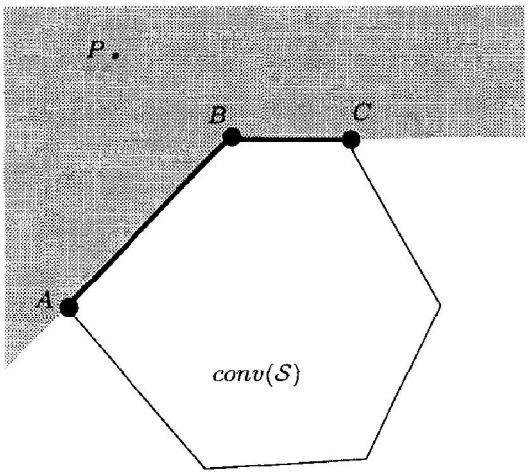

Figure 8.7. On-line convex hull: regions and conflicts.
The influence domain of region (AB, AC) is shaded, and the $(d-2)$-faces corresponding to regions conflicting with P are represented in bold.

is used mainly to detect the conflicts between the point to be inserted and the regions defined and without conflict over the points inserted so far. The influence graph is an oriented acyclic graph that has a node for each region that, at some previous step in the algorithm, appeared as a region defined and without conflict over the current subset of points. At each step of the algorithm, the regions defined and without conflict over the current subset correspond to the leaves of the influence graph. The arcs in this graph link these nodes such that the following *inclusion* property is always satisfied: the influence domain of a node is always contained in the union of the influence domains of its parents.[4] A depth-first traversal of the influence graph can detect all the conflicts between the new point P and the nodes in the graph. With a knowledge of the conflicts between points P and the regions defined and without conflict over $\mathcal{S}$, it is easy to find the facets of $conv(\mathcal{S})$ that are red with respect to P. Indeed:

- A region defined and without conflict over $\mathcal{S}$ that conflicts with P corresponds to a red or purple $(d-2)$-face of $conv(\mathcal{S})$, since it is incident to two $(d-1)$-faces of $conv(\mathcal{S})$, at least one of which is red (see figure 8.7).
- A region defined and without conflict over $\mathcal{S}$ that does not conflict with P corresponds to a $(d-2)$-face of $conv(\mathcal{S})$ that is blue with respect to P.

In an initial step, the algorithm processes the first $d+1$ points that are inserted into the convex hull. The incidence graph is set to that of the d-simplex formed

[4]Recall also that we frequently identify a node in the influence graph with the region that it corresponds to, which for instance lets us speak of conflicts with a node, of the influence domain of a node, or of the children of a region.

by these points, and the influence graph is initialized by creating a node for each of the regions that correspond to the $(d-2)$-faces of this simplex.

To describe the current step, we denote by $\mathcal{S}$ the current set of points already inserted, and by P the new point that is being inserted. The current step consists of a location phase and an update phase.

Locating. The location phase aims at detecting the regions *killed* by the new point P. These are the regions defined and without conflict over $\mathcal{S}$ that conflict with P. For this, the algorithm recursively visits all the nodes that conflict with P, starting from the root.

Updating. If none of the regions defined and without conflict over $\mathcal{S}$ is found to conflict with P, then P must lie inside the convex hull $conv(\mathcal{S})$, and there is nothing to update: the algorithm may proceed to the next insertion. If a region corresponding to a $(d-2)$-face of $conv(\mathcal{S})$ is found to conflict with P, however, then at least one of the two incident $(d-1)$-faces is red with respect to P. Starting from this red face, the incidence graph of $conv(\mathcal{S})$ can be updated into that of $conv(\mathcal{S} \cup \{P\})$ by executing phases 2, 3, and 4 of the incremental algorithm described above in section 8.4.

Its remains to show how to update the influence graph. Let us recall that the nodes of the influence graph are in bijection with the $(d-2)$-faces of the successive convex hulls, and that the corresponding regions are determined by a pair of adjacent facets, or also by the $d+1$ vertices that belong to these facets. To update the influence graph, the algorithm considers in turn each of the purple $(d-2)$-faces of $conv(\mathcal{S})$, and each of the $(d-3)$-faces incident to these faces.

1. Consider a $(d-2)$-face G_1 of $conv(\mathcal{S})$ that is purple with respect to P, and let (F_1, F_1') be the corresponding region; F_1 and F_1' are two $(d-1)$-faces of $conv(\mathcal{S})$ that are incident to G_1. We may assume that F_1 is blue with respect to P and F_1' is red (see figure 8.8). The face G_1 is a $(d-2)$-face of $conv(\mathcal{S} \cup \{P\})$ that corresponds to the new region (F_1, F_1''), where F_1'' is the convex hull $conv(G_1 \cup \{P\})$. A new node of the influence graph is created for region (F_1, F_1'') and this node is hooked into the influence graph as the child of (F_1, F_1'). In this way, the inclusion property is satisfied. Indeed, let H_1 and H_1' be the hyperplanes supporting $conv(\mathcal{S})$ along F_1 and F_1', respectively. The hyperplane H_1'' supporting $conv(S \cup \{P\})$ along F_1'' is also a hyperplane supporting $conv(\mathcal{S})$ along G_1. As a consequence, the half-space $H_1''^-$ that does not contain $conv(\mathcal{S} \cup \{P\})$ is contained in the union of the half-spaces H_1^- and H'^-_1, which do not contain $conv(\mathcal{S})$. The influence domain of region (F_1, F_1'') is therefore contained within that of (F_1, F_1').

2. Let K be a $(d-3)$-face of $conv(\mathcal{S})$, purple with respect to P, and let G_1 and G_2 be the purple $(d-2)$-faces of $conv(\mathcal{S})$ that are incident to K.[5] Let (F_1, F_1') and (F_2, F_2') be the two regions corresponding to G_1 and G_2, the faces F_1 and F_2

[5]The set of purple faces of $conv(\mathcal{S})$ being isomorphic to a $(d-1)$-polytope (lemma 8.3.4), any purple $(d-3)$-face of $conv(\mathcal{S})$ is incident to exactly two purple $(d-2)$-faces (theorem 7.1.7).

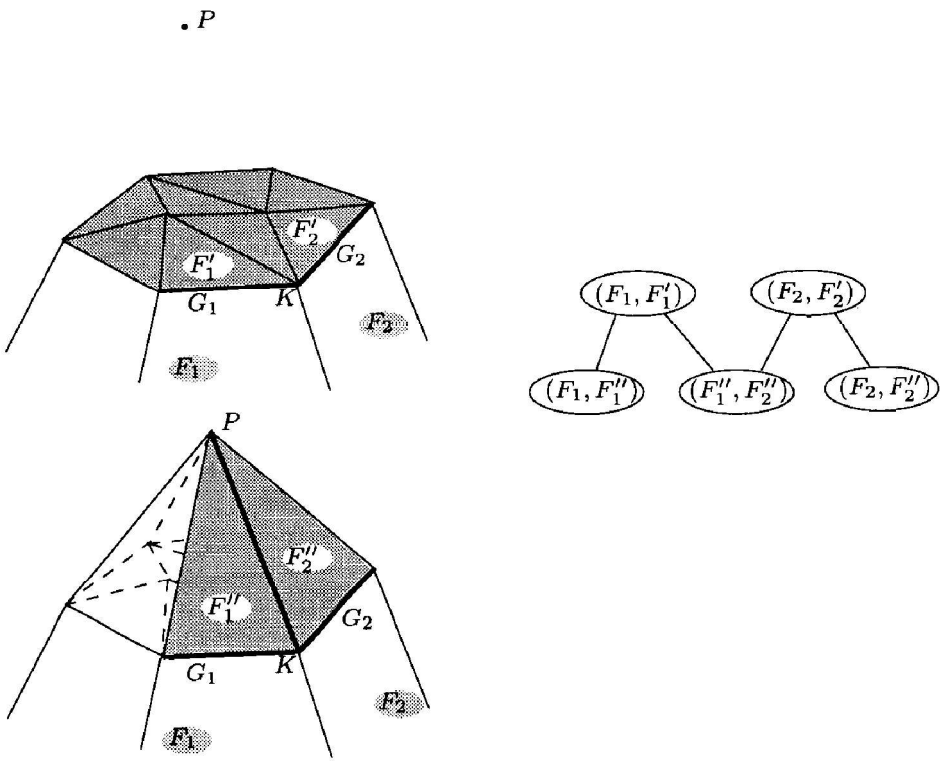

Figure 8.8. On-line convex hull: new regions when inserting a point P.

being blue with respect to P while faces F'_1 and F'_2 are red (see figure 8.8). The convex hull $conv(K \cup \{P\})$ is a $(d-2)$-face of $conv(\mathcal{S} \cup \{P\})$, and is incident to the $(d-1)$-faces $F''_1 = conv(G_1 \cup \{P\})$ and $F''_2 = conv(G_2 \cup \{P\})$. In the influence graph, a new node is created for the region (F''_1, F''_2), and hooked into the graph to two parents which are the nodes corresponding to regions (F_1, F'_1) and (F_2, F'_2). Let us verify that the inclusion property is satisfied. Indeed, the influence domain of (F''_1, F''_2) is the union $H''^{-}_1 \cup H''^{-}_2$, where H''^{-}_1 (resp. H''^{-}_2) is the half-space bounded by hyperplane H''_1 (resp. H''_2) that supports $conv(\mathcal{S} \cup \{P\})$ along F''_1 (resp. F''_2) and does not contain $conv(\mathcal{S} \cup \{P\})$. The half-space H''^{-}_1 is contained in the the influence domain of region (F_1, F'_1), and similarly H''^{-}_2 is contained in the influence domain of (F_2, F'_2). Consequently, the influence domain of (F''_1, F''_2) is contained in the union of the influence domains of (F_1, F'_1) and (F_2, F'_2).

This description can be carried over almost *verbatim* to the case of dimension 2. We need only remember that the polytope $conv(\mathcal{S})$ has an empty face of dimension -1, incident to all of its vertices. If P is not contained within $conv(\mathcal{S})$, the empty face is purple and incident to the two purple vertices of $conv(\mathcal{S})$ (see also figure 8.9).

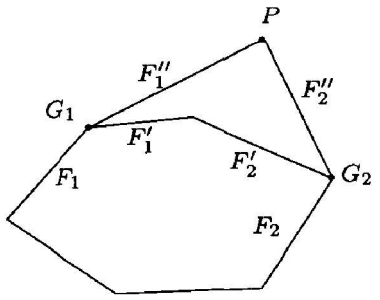

Figure 8.9. On-line convex hull in two dimensions.

Randomized analysis of the algorithm

In this randomized analysis, we assume that the points are inserted in a random order. The performances of the algorithm are then estimated on the average, assuming that all $n!$ permutations are equally likely.

To apply the results in chapter 5, we must verify that the algorithm satisfies the update condition 5.3.3 for algorithms that use an influence graph.

1. Testing conflict between a point and a region boils down to testing whether a point belongs to two half-spaces, and can be performed in constant time.

2. The number of children of each node in the influence graph is bounded. In fact, each node has d children, or none. Indeed, when inserting a point P, the node corresponding to a purple $(d-2)$-face G of $conv(\mathcal{S})$ receives d children: one for the $(d-2)$ face G of $conv(\mathcal{S} \cup \{P\})$, and $d-1$ corresponding to $conv(K \cup \{P\})$ for each $(d-3)$-subface K of G. The nodes corresponding to red or blue $(d-2)$-faces of $conv(\mathcal{S})$ do not receive children. The nodes corresponding to the red or purple $(d-2)$-faces of $conv(\mathcal{S})$ are killed by P: they no longer correspond to regions without conflict and will not receive children after the insertion of P.

3. The parents of a region created by a point P are recruited among the regions killed by P. From the analysis of phases 2, 3, and 4 of the incremental step in section 8.4, we can deduce that updating the incidence graph takes time proportional to the total number of red and purple faces of $conv(\mathcal{S})$ and of their incidences. If every $(d-2)$-face of the convex hull is linked by a bidirectional pointer with the corresponding node in the influence graph, it is easy to see that updating the influence graph takes about the same time as updating the incidence graph. The set of points being in general position, the facets of $conv(\mathcal{S})$ are simplices; thus the number of red or purple faces and of their incidences is proportional to the number of red facets of $conv(\mathcal{S})$. Each of these red facets is incident to $d-1$ red or purple

$(d-2)$-faces of $conv(\mathcal{S})$, each of which corresponds to a region that conflicts with P. Each region defined and without conflict over $\mathcal{S}$ that conflicts with $\mathcal{P}$ corresponds to a $(d-2)$-face of $conv(\mathcal{S})$ that is incident to one or two red facets. As a result, the number of red facets of $conv(\mathcal{S})$, and therefore the complexity of the update phase, is proportional to the number of regions killed by the new point P.

Since the update conditions are satisfied, the randomized analysis of the on-line convex hull computation can now be established readily by theorem 5.3.4 which analyzes algorithms that use an influence graph. The number of regions without conflict defined over a set $\mathcal{S}$ of n points in a d-dimensional space is exactly the number of $(d-2)$-faces of the convex hull $conv(\mathcal{S})$, which is $O(n^{\lfloor d/2 \rfloor})$ according to the upper bound theorem 7.2.5.

Theorem 8.5.1 *An on-line algorithm that uses the influence graph method to build the convex hull of n points in d dimensions requires expected time $O(n \log n + n^{\lfloor d/2 \rfloor})$, and storage $O(n^{\lfloor d/2 \rfloor})$. The expected time required to perform the n-th insertion is $O(\log n + n^{\lfloor d/2 \rfloor - 1})$.*

8.6 Dynamic convex hulls

The previous section shows that it is possible to build on-line the convex hull of a set of points in optimal expected time and storage, using an influence graph. Such an algorithm is called *semi-dynamic*, since it can handle insertions of new points. Fully dynamic algorithms, however, handle not only insertions but also deletions.

The possibility of deleting points makes the task of maintaining the convex hull much more complex. Indeed, during an insertion, the current convex hull and the new point entirely determine the new convex hull. After a deletion, however, points that were hidden inside the convex hull may appear as vertices of the new convex hull. A fully dynamic algorithm must keep, in one way or another, some information for all the points in the current set, be they vertices of the current convex hull or not.

The goal of this section is to show that the augmented influence graph method described in chapter 6 allows the convex hull to be maintained dynamically.

The algorithm which we now present uses again the notions of objects, regions, and conflicts as defined in the preceding section. It conforms to the general scheme of dynamic algorithms described in chapter 6, to which the reader is referred should the need arise. Besides the current convex hull (described by the incidence graph of its faces), the algorithm maintains an augmented influence graph whose nodes correspond to regions defined over the current set. After

each deletion, the structure is rebuilt into the exact state it would have been in, had the deleted point never been inserted. Consequently, the augmented influence graph only depends on the sequence $\Sigma = \{P_1, P_2, \ldots, P_n\}$ of points in the current set, sorted by chronological order: P_i occurs before P_j if the last insertion of P_i occurred before the last insertion of P_j.

Let us denote by $\mathcal{I}a(\Sigma)$ the augmented influence graph obtained for the chronological sequence Σ. The nodes and arcs of $\mathcal{I}a(\Sigma)$ are exactly the same as those of the influence graph built by the incremental algorithm of the preceding section, when the objects are inserted in the order given by Σ. We denote by $\mathcal{S}_l$ the subset of $\mathcal{S}$ formed by the first l objects in Σ. The nodes of $\mathcal{I}a(\Sigma)$ correspond to the regions defined and without conflict over the subsets $\mathcal{S}_l$, for $l = 1, \ldots, n$. The arcs of $\mathcal{I}a(\Sigma)$ ensure both *inclusion properties*: that the domain of influence of a node is contained in the union of the domains of influence of its parents, and that a determinant of this node is either the creator of this node or is contained in the union of the sets of determinants of its parents. Moreover, the augmented influence graph contains a conflict graph between the regions that correspond to nodes in the influence graph, and the objects in $\mathcal{S}$. This conflict graph is implemented by a system of interconnected lists such as that described in section 6.2: each node of the conflict graph has a list (sorted in chronological order) of the objects that conflict with the corresponding region; also, for each object we maintain a list of pointers to the nodes in the influence graph that conflict with that object. The record corresponding to an object in the conflict list of a node is interconnected with the record corresponding to that node in the conflict list of the object.

Insertion

Inserting the n-th point into the convex hull is carried out exactly as in the on-line algorithm described in section 8.5, except that while we are locating the object in the influence graph, each detected conflict is added to the interconnected conflict lists.

Deletion

Let us now consider the deletion of point P_k. For $l = k, \ldots, n$, we denote by $\mathcal{S}'_l$ the subset $\mathcal{S}_l \setminus \{P_k\}$ of $\mathcal{S}$, and by Σ' the chronological sequence $\{P_1, \ldots, P_{k-1}, P_{k+1}, \ldots P_n\}$. When deleting P_k, the algorithm rebuilds the augmented influence graph, resulting in $\mathcal{I}a(\Sigma')$. For this, we must:

1. remove from the graph $\mathcal{I}a(\Sigma)$ the *destroyed* nodes, which correspond to regions having P_k as a determinant,[6]

[6]Recall that an object is a determinant of a region if it belongs to the set of objects that determine this region.

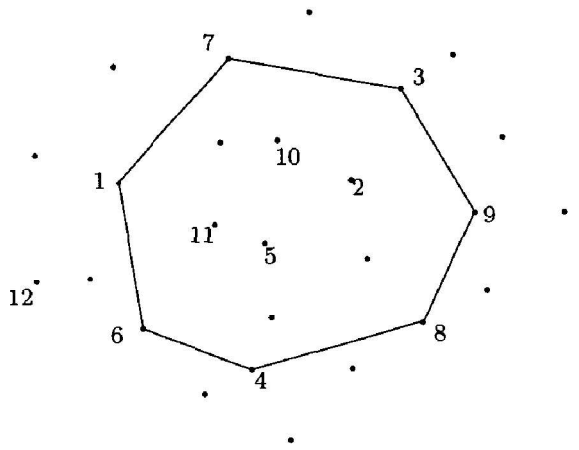

Figure 8.10. Convex hull: creator and killer of a region.
Points are numbered by chronological ranks. Unnumbered points have rank greater than 12. Region (6 1, 6 4) has point 6 as its creator and point 12 as its killer.

2. create a *new* node for each region defined and without conflict over one of the subsets $\mathcal{S}'_l$, $l = k+1, \ldots, n$ that conflicts with P_k,
3. set up the new arcs that are incident to the new nodes. The new nodes must be hooked to their parents which may or may not be new. The unhooked nodes, which are nodes of $\mathcal{I}a(\Sigma)$ that are not destroyed but have destroyed parents, must be rehooked.

Before we describe the deletion algorithm, it is useful to recall a few definitions. A region G of $\mathcal{I}a(\Sigma)$ is *created* by P_l or also P_l is the *creator* of G, if P_l is among all determinants of G the one with highest chronological rank. A region G of $\mathcal{I}a(\Sigma)$ is *killed* by P_l or also P_l is the *killer* of G if P_l has the lowest rank among the points that conflict with G (see figure 8.10).

The deletion algorithm proceeds in two substeps: the location phase and the rebuilding phase.

Locating. During this phase, the algorithm identifies the nodes in $\mathcal{I}a(\Sigma)$ that are killed by P_k, and the destroyed and unhooked nodes. For this, the algorithm recursively visits all the nodes that conflict with P_k or have P_k as a determinant, starting at the root. During the traversal, the algorithm removes P_k from the conflict lists, and builds a dictionary of the destroyed or unhooked nodes for use during the rebuilding phase.

Rebuilding. During this phase, the algorithm creates the new nodes, hooks them to the graph, builds their conflict lists and rehooks the unhooked nodes.

For this, the algorithm considers in turn all the objects P_l of rank $l > k$ that are the creators of some new or unhooked node. A point P_l of rank $l > k$ is the

creator of some new or unhooked node if and only if there exists a region defined and without conflict over $\mathcal{S}'_{l-1}$ which conflicts with both P_l and P_k (lemma 6.2.1). When processing P_l, we call a region *critical* if it is defined and without conflict over $\mathcal{S}'_{l-1}$ but conflicts with P_k. The *critical zone* is the set of all critical regions. The critical zone evolves as we consider the objects P_l in turn. At the beginning of the rebuilding phase, the critical regions are the regions of $\mathcal{I}a(\Sigma)$ that are killed by P_k. Subsequently, the critical regions are either regions in $\mathcal{I}a(\Sigma)$ that are killed by P_k, or new regions in $\mathcal{I}a(\Sigma')$. At each substep in the rebuilding phase, the next point to be processed is the point of smallest rank among all the points that conflict with one or more of the currently critical regions. To find this point, the algorithm maintains a priority queue $\mathcal{Q}$ of the points in Σ' that are the killers of critical regions. Each point P_l in $\mathcal{Q}$ also stores the list of the current critical regions that it kills. The priority queue $\mathcal{Q}$ is initialized with the killers in Σ' of the regions in $\mathcal{I}a(\Sigma)$ that were killed by P_k.

At each substep in the rebuilding phase, the algorithm extracts the point P_l of smallest rank in $\mathcal{Q}$, and this point is then *reinserted* into the data structure. To reinsert a point means to create new nodes for the new regions created by P_l, to hook them to the influence graph, and to rehook the unhooked nodes created by P_l. The $(d-2)$-faces of $conv(\mathcal{S}'_{l-1})$ that are red or purple with respect to the point P_k that is removed correspond to critical regions and are, below, called *critical faces*. Unless explicitly stated, the color *blue*, *red*, or *purple*, is now given with respect to the point P_l that is being reinserted. The regions that are unhooked or new and created by P_l can be derived from the critical purple $(d-2)$-faces and their $(d-3)$-subfaces, which will be considered in turn by the algorithm.

1. Processing the critical purple $(d-2)$-faces

Along with point P_l, we know the list of critical regions with which it conflicts. These regions correspond to the critical red or purple $(d-2)$-faces, and a linear traversal of this list allows the sublist of its critical purple $(d-2)$-faces to be extracted.

Let G be a critical purple $(d-2)$-face, and (F, F') be the corresponding region; F and F' are $(d-1)$-faces of $conv(\mathcal{S}'_{l-1})$, both incident to G, and we may assume that F is blue with respect to P_l while F' is red (see figure 8.11 in dimension 3 and figure 8.12 in dimension 2.)

In the convex hull $conv(\mathcal{S}'_l)$, G is a $(d-2)$-face that corresponds to (F, F''), a region defined and without conflict over $\mathcal{S}'_l$, where F'' is the convex hull $conv(G \cup \{P_l\})$ (see figure 8.11 in dimension 3 and figure 8.12 in dimension 2.)

If region (F, F'') conflicts with P_k (see figures 8.11a and 8.12a), then it is a new region created by P_l. In the augmented influence graph, a new node is created for this region, with node (F, F') as parent. The conflict list of (F, F'') can be

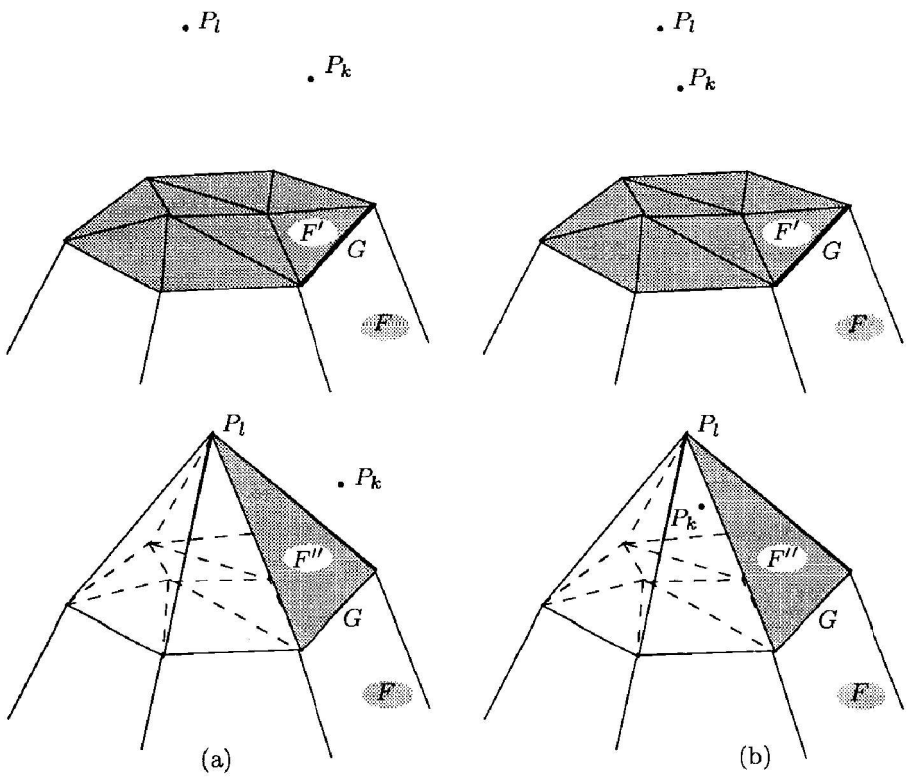

Figure 8.11. Deleting from a 3-dimensional convex hull: handling critical purple $(d-2)$-faces.
(a) (F, F'') is a new region.
(b) (F, F'') is an unhooked region.

set up by selecting the objects in conflict with (F, F'') from the conflict list of (F, F'). The killer of (F, F'') in Σ' is inserted in the priority queue $\mathcal{Q}$ if it was not found there. Finally, region (F, F'') is added to the list of critical regions killed by this point.

If region (F, F'') does not conflict with P_k (see figures 8.11b and 8.12b), then it corresponds to an unhooked node created by P_l. This node is found by using the dictionary $\mathcal{D}$ of destroyed and unhooked nodes, and hooked as a child of (F, F').

2. Handling the critical purple $(d-3)$-faces

Critical purple $(d-3)$-faces are subfaces of critical purple $(d-2)$-faces.[7] For each such $(d-3)$-face, we must know the at most two critical purple $(d-2)$-faces incident to it. To find them, we build an auxiliary dictionary $\mathcal{D}'$ of the $(d-3)$-subfaces of critical purple $(d-2)$-faces. Each entry in the dictionary $\mathcal{D}'$ for a

[7] According to lemma 8.3.4, each critical purple $(d-3)$-face is incident to two purple $(d-2)$-faces, at least one of which is critical.

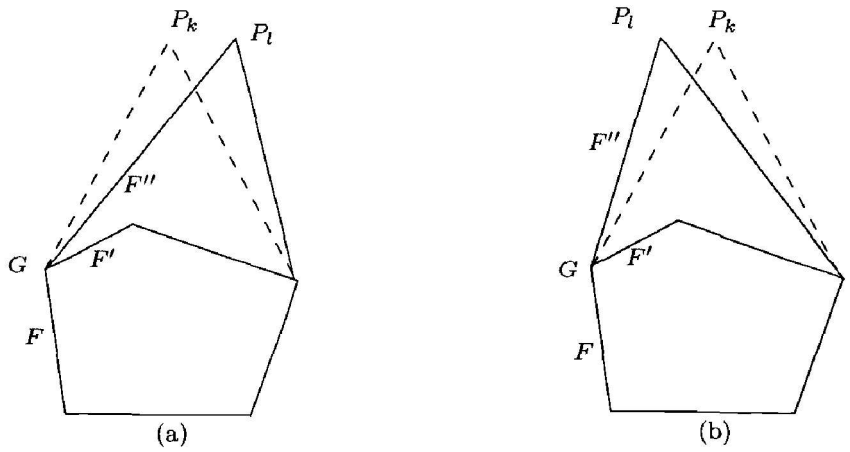

Figure 8.12. Deleting from a 2-dimensional convex hull: handling critical purple $(d-2)$-faces.
(a) (F, F'') is a new region.
(b) (F, F'') is an unhooked region.

$(d-3)$-face K has two pointers for keeping track of the critical purple $(d-2)$-faces incident to K.

Let K be such a $(d-3)$-face (see figure 8.13 in dimension 3 and figure 8.14 in dimension 2). We denote by G_1 and G_2 the two purple $(d-2)$-faces incident to K. At least one of them is a critical face, but not always both. We denote by (F_1, F_1') and (F_2, F_2') the regions corresponding to faces G_1 and G_2 of the convex hull $conv(\mathcal{S}'_{l-1})$. We may assume that facets F_1 and F_2 are blue, while F_1' and F_2' are red.

The $(d-2)$-face $conv(K \cup \{P_l\})$ of $conv(\mathcal{S}'_l)$ corresponds to some region (F_1'', F_2''), where $F_1'' = conv(G_1 \cup \{P_l\})$ and $F_2'' = conv(G_2 \cup \{P_l\})$ (see figure 8.13; see also figure 8.14, in dimension 2, in which K is the empty face of dimension -1, and G_1 and G_2 are the two vertices of $conv(\mathcal{S}'_{l-1})$, both purple with respect to P_l).

2.a If both G_1 and G_2 are critical faces, the corresponding nodes in $\mathcal{I}a(\Sigma')$ may be retrieved through dictionary $\mathcal{D}'$.

2.a.1 If region (F_1'', F_2'') conflicts with P_k (see figure 8.14a), it is a new region created by P_l; a node is created for this region, and inserted into the influence graph with both (F_1, F_1') and (F_2, F_2') as parents. The conflict list of (F_1'', F_2'') may be obtained by merging the conflict lists of (F_1, F_1') and (F_2, F_2'), and then selecting from the resulting list the objects that conflict with (F_1'', F_2''). Merging the conflict lists can be carried out in time proportional to the total length, because these lists are ordered chronologically.[8] The killer of (F_1'', F_2'') in the sequence Σ' is inserted into the priority queue $\mathcal{Q}$ if not found there, and region (F_1'', F_2'') is added to the list of critical regions killed by this point.

[8] An alternative to this solution is to forget about ordering the conflict lists and to resort to

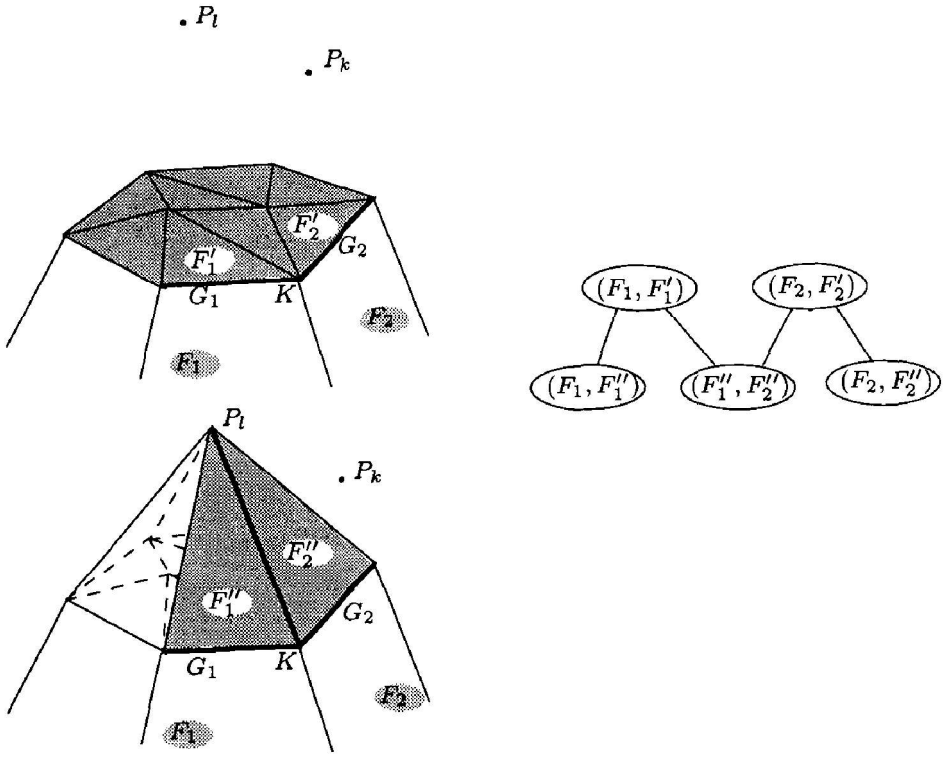

Figure 8.13. Deleting from a 3-dimensional convex hull: handling critical purple $(d-3)$-faces.

2.a.2 If region (F_1'', F_2'') does not conflict with P_k (see figure 8.14b), then this region is an unhooked region created by P_l. It suffices to find the corresponding node using dictionary $\mathcal{D}$ and hook it back to the nodes corresponding to (F_1, F_1') and (F_2, F_2').

2.b When only one of the purple $(d-2)$-faces G_1 and G_2 incident to K is critical, say G_1, the algorithm must find in the influence graph the node corresponding to G_2, the other purple $(d-2)$-face incident to K. Lemma 8.6.1 below proves that, in this case, $conv(K, P_l)$ is a $(d-2)$-face of $conv(\mathcal{S}_l)$ which corresponds to a destroyed or unhooked node of $\mathcal{I}a(\Sigma)$, whose parents include precisely the node corresponding to region (F_2, F_2'). To find (F_2, F_2'), we may therefore search in the dictionary $\mathcal{D}$ of destroyed or unhooked nodes, created by P_l, corresponding to the $(d-2)$-face $conv(K, P_l)$ of $conv(\mathcal{S}_l)$. This node is uniquely known from this criterion, because we know not only the $(d-2)$-face $conv(K, P_l)$ of its corresponding region, but also its creator P_l.

Lemma 8.6.1 *Let K be a $(d-3)$-face of $conv(\mathcal{S}'_{l-1})$ incident to two purple faces G_1 and G_2, only one of which is critical, say G_1. Then $conv(K, P_l)$ is a*

the method used in section 6.4 for merging the conflicts lists of trapezoids.

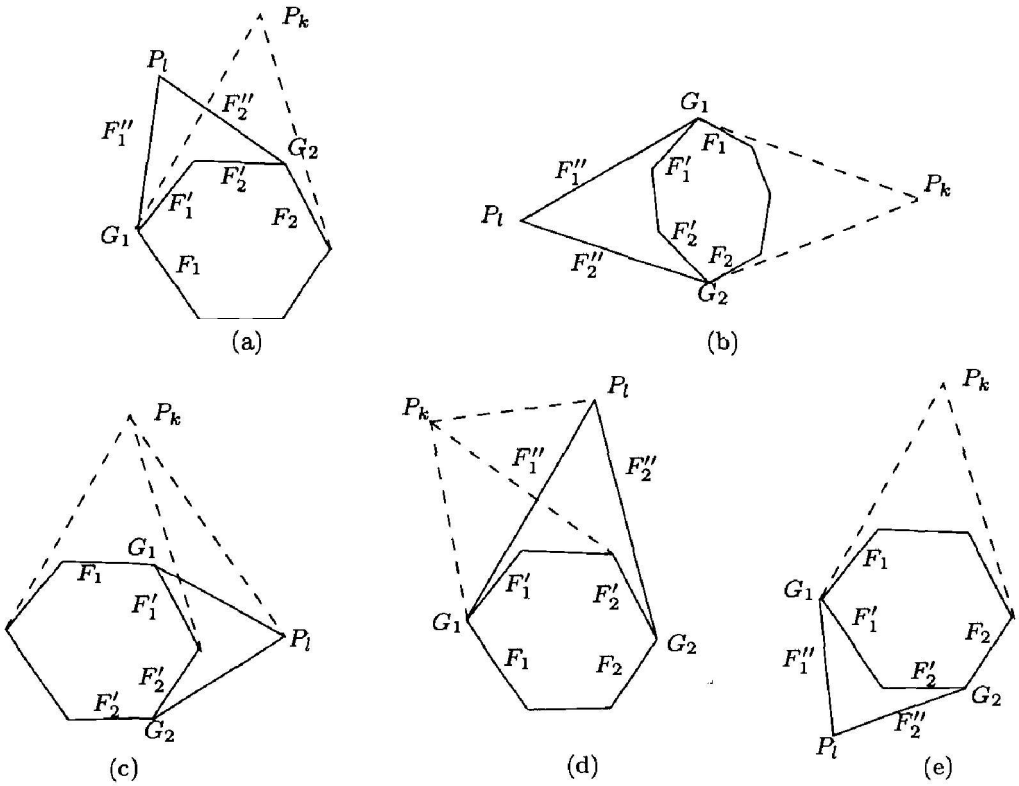

Figure 8.14. Deleting from a 2-dimensional convex hull: handling the critical purple $(d-3)$-faces. Critical purple $(d-3)$-face K here is the empty face of dimension -1. G_1 and G_2 are its two purple vertices.
(a) G_1 and G_2 are critical, (F_1'', F_2'') is new.
(b) G_1 and G_2 are critical, (F_1'', F_2'') is unhooked.
(c) G_1 is critical, G_2 is not, and G_1 is not a face of $conv(\mathcal{S}_{l-1})$.
(d) G_1 is critical, G_2 is not, and G_1 is a face of $conv(\mathcal{S}_{l-1})$, but not purple with respect to P_l.
(e) G_1 is critical, G_2 is not, and G_1 is a face of $conv(\mathcal{S}_{l-1})$, this time purple with respect to P_l.

$(d-2)$-face of $conv(\mathcal{S}_l)$, its corresponding node in $\mathcal{I}a(\Sigma)$ is destroyed or unhooked, and one of its parents is the region (F_2, F_2') that corresponds to the face G_2 of $conv(\mathcal{S}'_{l-1})$.

Proof. For the proof, imagine that P_k then P_l are inserted into $\mathcal{S}'_{l-1}$: then we obtain successively $\mathcal{S}_{l-1}$ and $\mathcal{S}_l$.

The $(d-2)$-face K of $conv(\mathcal{S}'_{l-1})$ is purple with respect to P_k since it belongs to a critical $(d-2)$-face as well as to a non-critical $(d-2)$-face. As a result, both K and $conv(K, P_k)$ are faces of $conv(\mathcal{S}_{l-1})$.

Since it is not critical, the $(d-2)$-face G_2 is also a $(d-2)$-face of $conv(\mathcal{S}_{l-1})$, and its corresponding region is still (F_2, F_2'), hence face G_2 of $conv(\mathcal{S}_{l-1})$ is purple with respect to P_l.

The $(d-3)$-face K of $conv(\mathcal{S}_{l-1})$ is purple with respect to P_l since it is incident to G_2. As a result, $conv(K, P_l)$ is a $(d-2)$-face of $conv(\mathcal{S}_l)$, incident to the $(d-1)$-face $conv(G_2, P_l)$. In the graph $\mathcal{I}a(\Sigma)$, one of the parents of the node corresponding to the $(d-2)$-face $conv(K, P_l)$ of $conv(\mathcal{S}_l)$, is region (F_2, F_2').

We now have to show that the $(d-2)$-face $conv(K, P_l)$ of $conv(\mathcal{S}_l)$ corresponds to a region that is either destroyed or unhooked when P_k is deleted. For this, we consider the face G_1 of $conv(\mathcal{S}'_{l-1})$. The situation is one of three (see figure 8.14c, d, e). If this face is red with respect to P_k (see figure 8.14c), then it is not a face of $conv(\mathcal{S}_{l-1})$ any more. If this face is purple with respect to P_k, it remains a face of $conv(\mathcal{S}_{l-1})$, but it may be blue (see figure 8.14d) or remain purple (see figure 8.14e) with respect to P_l.

In the first two cases, $conv(K, P_k)$ is necessarily a $(d-2)$-face of $conv(\mathcal{S}_{l-1})$, purple with respect to P_l. Indeed, the set of those purple faces is isomorphic to a $(d-1)$-polytope, and since G_1 is not purple with respect to P_l, it must be replaced by another $(d-2)$-face incident to K which can only be $conv(K, P_k)$. Consequently, the region corresponding to the $(d-2)$-face $conv(K, P_l)$ of $conv(\mathcal{S}_l)$ is region $(conv(K, P_l, P_k), F_2'')$ which is destroyed during the deletion of P_k.

In the third case, $F_1'' = conv(G_1, P_l)$ must be a facet of $conv(\mathcal{S}_l)$, and the region that corresponds to the $(d-2)$-face $conv(K, P_l)$ of $conv(\mathcal{S}_l)$ is region (F_1'', F_2''), which is an unhooked region created by P_l. □

Once the node corresponding to the $(d-2)$-face G_2 of $conv(\mathcal{S}'_l)$ has been found, operations can resume as before, apart from a simple detail. If the region (F_1'', F_2'') that corresponds to the $(d-2)$-face $conv(K, P_l)$ of $conv(\mathcal{S}'_l)$ is new, then its conflict list may be obtained by merging that of the critical region (F_1, F_1') and that of the destroyed region $(conv(K, P_l, P_k), F_2'')$. (We do this in order to avoid traversing the conflict list of region (F_2, F_2') corresponding to face G_2, which is neither new nor destroyed.)

Randomized analysis of the algorithm

The algorithm is deterministic. Yet the analysis given here is randomized and assumes the following probabilistic model:

- the chronological sequence Σ is a random sequence, each of the $n!$ permutations being equally likely;
- each insertion concerns, with equal probability, any of the objects present in the current set immediately after the insertion;
- each deletion concerns, with equal probability, any of the objects present in the current set immediately before the deletion.

Theorem 8.6.2 *Using an augmented influence graph allows the fully dynamic maintenance of the convex hull of points in $\mathbb{E}^d$, under insertion or deletion of points. If the current set has n points:*

- *the structure requires expected storage $O(n \log n + n^{\lfloor d/2 \rfloor})$,*
- *inserting a point takes expected time $O(\log n + n^{\lfloor d/2 \rfloor - 1})$,*
- *deleting a point takes expected time $O(\log n)$ in dimension 2 or 3 and time $O(tn^{\lfloor d/2 \rfloor - 1})$ in dimension $d > 3$. The parameter t represents the complexity of an operation on the dictionaries used by the algorithm ($t = O(\log n)$ if balanced binary trees are used, $t = O(1)$ if perfect dynamic hashing is used.)*

Proof. During the rebuilding phase in a deletion, the number of queries into the dictionary of destroyed or unhooked nodes is at most proportional to the number of destroyed or unhooked nodes. For each point P_l that is reinserted, the number of updates or queries on the dictionary of $(d-3)$-faces incident to critical purple $(d-2)$-faces is proportional to the number of these critical purple $(d-3)$-faces. Thus, the total number of accesses to the dictionaries is proportional to the total number of critical faces encountered that correspond to new or killed nodes. The conflict lists of new nodes can be set up in time at most proportional to the total sizes of the conflict lists of new or killed nodes. All the other operations performed during a deletion, except handling the priority queue, take constant time, and their number is proportional to the number of destroyed, new, or unhooked nodes.

As a result, the algorithm indeed satisfies the update condition 6.3.5 for algorithms that use an augmented conflict graph. Its randomized analysis is therefore the same as in section 6.3, and is given in theorem 6.3.6 in terms of $f_0(l, \mathcal{S})$, the expected number of regions defined and without conflict over a random l-sample of $\mathcal{S}$. For the case of convex hulls, since the number of such regions for any sample is bounded in the worst case by $O(l^{\lfloor d/2 \rfloor})$ (upper bound theorem 7.2.5), so is their expectation $f_0(l, \mathcal{S})$. This results in the performance given in the statement of theorem 6.3.6. In dimension 2 or 3, the number of operations to be performed on the dictionaries and on the priority queue is $O(1)$ whereas handling the conflict lists always takes $O(\log n)$ time. Therefore, it suffices to implement dictionaries and priority queues with balanced binary trees. In dimensions higher than 3, deletions have supra-linear complexity, and the priority queue may be implemented using a simple array. □

8.7 Exercises

Exercise 8.1 (Extreme points) *Extreme points* in a set of points are those which are vertices of the convex hull. Show that to determine the extreme points of n points in $\mathbb{E}^2$ is a problem of complexity $\Theta(n \log n)$.

Hint: You may use the notion of an algebraic decision tree: an algebraic tree of degree a is a decision tree where the test at any node evaluates the sign of some algebraic function of degree a for the inputs. Loosely stated, a result by Ben-Or (see also subsection 1.2.2) says that any algebraic decision tree that decides whether a point in $\mathbb{E}^k$ belongs to some connected component W of $\mathbb{E}^k$ must have a height $h = \Omega(\log c(W) - k)$, where $c(W)$ is the number of connected components of W.

Exercise 8.2 (Adjacency graph) Let a simplicial d-polytope be defined as the convex hull of n points. Show that knowledge of the facets of the graph (given by their vertices), along with their adjacencies, suffices to reconstruct the whole incidence graph of the polytope in time linear in the size of the adjacency graph, which is $O(n^{\lfloor d/2 \rfloor})$.

Exercise 8.3 (1-skeleton) This problem is the dual version of its predecessor. Let a simple d-polytope be defined as the intersection of n half-spaces. Suppose that the 1-skeleton is known, that is the set of its vertices and the arcs joining them. Each vertex is given as the intersection of d bounding hyperplanes. Show that the whole incidence graph of the polytope may be reconstructed in time $O(n^{\lfloor d/2 \rfloor})$.

Exercise 8.4 (Degenerate cases) Generalize the incremental algorithm described in section 8.4 to build the convex hull of a set of points which is not assumed to be in general position.

Exercise 8.5 (On-line convex hulls) Give an algorithm to compute on-line the convex hull of a set of points in $\mathbb{E}^d$, by using an influence graph whose nodes correspond to regions which are half-spaces. Give the randomized analysis of this algorithm.

Hint: Each region, or half-space, is now determined by a subset of d affinely independent points that generates its bounding hyperplane. A point conflicts with a half-space if it lies inside. The regions defined and without conflict over a set $\mathcal{S}$ are in bijection with the facets, or $(d-1)$-faces, of the convex hull $conv(\mathcal{S})$ of $\mathcal{S}$.

Upon inserting a point P into $\mathcal{S}$, the regions killed by P correspond to the facets of $conv(\mathcal{S})$ that are red with respect to P, and the regions created by P correspond to the facets $conv(G \cup P)$ of $conv(\mathcal{S} \cup P)$ where G is any $(d-2)$-face of $conv(\mathcal{S})$ that is purple with respect to P.

1. Let F_1 and F_2 be the facets of $conv(\mathcal{S})$ incident to a $(d-2)$-face G, which is purple with respect to P. Show that the node of the influence graph that corresponds to the facet $conv(G \cup P)$ must have both nodes corresponding to F_1 and F_2 as parents.

2. In this manner, a node in the graph may receive a child without being killed, therefore the number of children of a node is not bounded any more. The maximum number of parents is two, however. For this particular problem, define and use the notion of a biregion that was introduced in exercise 5.7, and show that the expected complexity of the algorithm is $O(n \log n + n^{\lfloor \frac{d}{2} \rfloor})$.

Exercise 8.6 (Intersection of half-spaces) Give a randomized incremental algorithm that uses a conflict graph to build the intersection of n half-spaces in $\mathbb{E}^d$ whose bounding hyperplanes are in general position. Try to achieve an expected running time of

$O(n \log n + n^{\lfloor \frac{d}{2} \rfloor})$. Give an on-line version of the preceding algorithm that uses an influence graph.

Show that in the version of the algorithm that uses a conflict graph, the storage requirements may be lowered if only one conflict is stored for each half-space.

Hint: Objects are half-spaces, regions are segments. A segment is determined by $d+1$ half-spaces, or rather by the $d+1$ hyperplanes which bound these half-spaces. The line that supports the segment is the intersection of $d-1$ hyperplanes, and the endpoints of the segment are the intersections of this line with the two remaining hyperplanes. A segment conflicts with a half-space if it has an intersection with the (open) complementary half-space. The segments defined and without conflict over these half-spaces are precisely the edges of the polytope, obtained as the intersection of the half-spaces.

When a new half-space $\overline{H^+}$ bounded by a hyperplane H is inserted, the conflict graph identifies all the edges that lie in H^-, which disappear, and those that intersect H. An edge E that intersects H gives a shorter edge $E' \subset E$, and the conflict list of E' is set up by traversing that of E. To obtain the new edges that lie in H, it suffices to follow, for each 2-face F incident to each edge E that intersects H, the edges of F that conflict with $\overline{H^+}$ until the second edge E' of F that intersects H is found. The new edge $F \cap H$ has vertices $E \cap H$ and $E' \cap H$. Its conflict list can be obtained by traversing the conflict lists of the edges of F killed by $\overline{H^+}$. Knowing the 1-skeleton, the whole incidence graph of the intersection may be updated.

8.8 Bibliographical notes

The incremental deterministic convex hull algorithm described in section 8.4 is due to Seidel [201]. This algorithm is also described in detail in Edelsbrunner's book [89] where degenerate cases are also handled (see exercise 8.4).

The first randomized algorithm to build convex hulls was due to Clarkson and Shor [71]. This algorithm uses a conflict graph and in fact solves the dual problem of computing the intersection of half-spaces (see exercise 8.6). The on-line algorithm that uses an influence graph is due to Boissonnat, Devillers, Teillaud, Schott and Yvinec [28]. The dynamic algorithm presented in section 8.6 is due to Dobrindt and Yvinec [86]. Clarkson, Mehlhorn, and Seidel [70] and independently Mulmuley [176, 177] proposed similar solutions for dynamically maintaining convex hulls.

Chazelle [46] proposed a deterministic algorithm that is optimal in any dimension greater than 3. This algorithm is a *derandomized* incremental algorithm, and uses the method of conditional probabilities to determine which point must be inserted next. Brönnimann, Chazelle, and Matoušek [36] and Brönnimann [35] give a simpler version which works in any dimension.

The lower bound proposed in exercise 8.1 to identify the extreme points in a set of points in the plane is due to Yao [220]. The solution to exercise 8.1 can be found in the book by Preparata and Shamos [192].