Convex polyhedral chains: a representation for geometric data

O Günther and E Wong

A representation scheme for general polyhedra in arbitrary dimensions is presented. Each polyhedron is represented as a convex chain, i.e. an algebraic sum of convex polyhedra (cells). Each cell in turn is represented as the intersection of halfspaces and encoded in a vector. The notion of vertices is abandoned completely. It is shown how this approach allows the decomposition of set operations (such as intersection) on polyhedra into two independent steps. The first step consists of a collection of vector operations; the second step is a garbage collection where vectors that represent empty cells are eliminated. No special treatment of singular intersection cases is needed. This approach to set operations is significantly different from algorithms that have been proposed previously.

geometry, data management, solid modelling, constructive solid geometry

Modern database systems are no longer limited to business applications. Nonstandard applications such as computer-aided design, computer vision, or geographic data processing are becoming increasingly important, and geometric data play a crucial role in many of these new applications. For reasons of efficiency it is essential that the special properties of geometric data be fully utilized in the database management system. It is important to view geometric objects (such as points, lines, or polygons) as integral entities and not as tuples of numbers that may be used to represent them.

Furthermore, the special operators that are defined on these objects need to be supported. Common examples include set operators such as union or intersection, or search operators such as range search or point location. These operators are substantially different from the operators defined on numerical data. They are often harder to compute, and it is not trivial to determine the smallest domain on which they are closed. Even the regularized set operators, for example, are not closed on the set of simple polyhedra; see Figure 1. (The regularized set operators, as defined by Tilove[2], include intersection, union, and difference. They differ from the corresponding simple set operators by an additional step making the result regular, i.e. the closure of its interior. This way, the dimension of the result is equal to the lowest dimension of any of the operands. In this paper, all set operators are assumed to be regularized, unless stated otherwise.)

In short, to deal with geometric data effectively requires some recognition of geometry, and nowhere is this more important than in the representation of geometric objects, which can be interpreted as the mapping of the original objects into a set of objects that facilitates the computation of a particular class of operators. The significance of representation schemes for efficient geometric data management has been discussed by Requicha[3]. A survey of various representation schemes for two- and three-dimensional geometric data has been conducted by Besl and Jain[4].

In this paper, this theme is developed in connection with a particular representation scheme for an important class of geometric objects, that is, polyhedra. The restriction to polyhedra, rather than general point sets, is justified by the fact that those are commonly used to approximate general shapes in practice[5]. In most current applications the polyhedra to be represented are simple, i.e. self-intersections or holes are not allowed Polyhedra that are not simple, however, become more and more important in areas like computer-aided

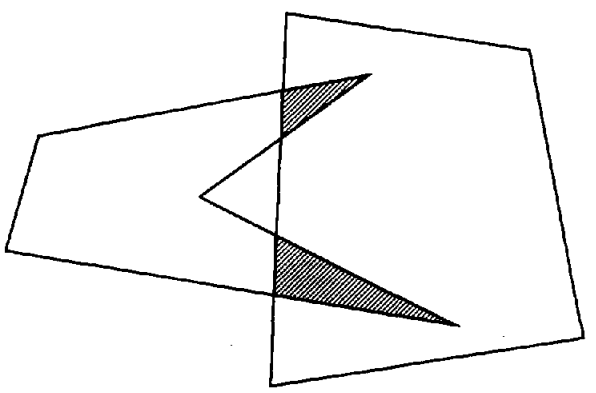

Figure 1. The intersection of two simple polyhedra is not necessarily a simple polyhedron

Department of Electrical Engineering and Computer Sciences, 231 Cory Hall, University of California, Berkeley, CA 94720, USA

This is an extended and revised version of a paper presented at the 13th International Conference on Very Large Data Bases, Brighton, England (September 1987)[1]. The work was sponsored under research contract DAAG29-85-0223 and, in the case of the first author, a scholarship from the German National Scholarship Foundation and an IBM graduate fellowship

 0010-4485/89/030157-08 $03.00

design or geographic data processing. Several examples for the applications of self-intersecting polygons in the area of IC mask description are given by Newell and Sequin[6]. Geographic applications very often need polygons with holes (for example, to represent areas whose altitude is within a given range). Some applications may require polygons that are folded and keep track of the resulting multiple layers. Also, there are numerous applications for higher-dimensional or unbounded polyhedra, such as linear programming[7] or logic databases where geometric objects are used to represent predicates[8].

A representation scheme for polyhedra should therefore include polyhedra in higher dimensions as well as polyhedra that are unbounded or not simple. Furthermore, it has to support some of the most common operators performed on geometric data, such as set and search operators. Finally, the representation scheme should be closed under set operators.

This paper presents the idea of polyhedral chains as a representation scheme for general polyhedra in arbitrary dimensions that meets these challenges. A general polyhedron may be self-intersecting or unbounded, it may have holes, and it may consist of several disjoint pieces. However, the polyhedra considered here are assumed to be regular[2] and therefore without degenerate parts, such as a dangling edge or face (one that bounds nothing).

General polyhedra are represented as polyhedral chains, i.e. algebraic sums of simple polyhedra (cells). The following section gives a definition of polyhedral chains and discusses their properties. Then a representation scheme that is based on convex polyhedral chains, i.e. for which all cells are convex, is described in detail. Each cell in turn is represented as the intersection of halfspaces and is encoded in a vector. The notion of vertices is abandoned completely as it is not needed for the set and search operators that are intended to be supported.

Finally, it is shown how this approach allows the decomposition of the computation of set operators on polyhedra into two steps. The first step consists of a collection of vector operations; the second step is a garbage collection where vectors that represent empty cells are eliminated. No special treatment of singular intersection cases such as dangling edges or faces is needed. This approach to set operations is significantly different from algorithms that have been proposed in the past[9–14]. Most of those approaches are based on a space partitioning to localize the set operations and on the subsequent use of vertices, edges, and adjacencies. They require special treatment for singular intersection cases and are difficult to generalize to higher dimensions. A first algorithm for the treatment of higher-dimensional polyhedra has appeared recently[15]. Like some of the algorithms mentioned above, it uses a complicated boundary classification scheme, which involves special treatment of the singular intersection cases.

POLYHEDRAL CHAINS

The notion of polyhedron is extended in the following way. A polyhedral chain in d-dimensional Euclidean space E^d, as defined by Whitney[16], is an expression of the form

$$\sum_{i=1}^{m} \alpha_i p_i$$

where α_i are integers and p_i are simple d-dimensional polyhedra in E^d, called cells. Note that the cells are not necessarily bounded. The algebraic convention is as follows:

$$\alpha p_i + \beta p_i = (\alpha + \beta) p_i$$
$$p_i + p_i = p_i \cup p_i \Leftrightarrow p_i \cap p_i = \varnothing$$
$$0 \cdot p_i = \varnothing$$

Two polyhedral chains are equivalent if they can be transformed into each other using these conventions.

The semantics assigned to a polyhedral chain are as follows. The polyhedral chain can be viewed as a function that maps each point $t \in E^d$ into an integer number that indicates the number of cells present at this point. More formally, the function f_x corresponding to a chain $x = \Sigma_{i=1}^{m} \alpha_i p_i$ may be defined as

$$f_x(t) = \sum_{t \in p_i} \alpha_i, \quad t \in E^d$$

From the algebraic conventions for polyhedral chains it follows that two chains are equivalent if and only if they correspond to the same function $f_x(t)$.

Polyhedral chains are a simple and powerful tool to describe various kinds of polyhedra. They may be used to describe any polyhedral point set in E^d (Figure 2), as well as self-intersecting polyhedra of any shape (Figures 3 and 4).

In many applications, a distinction is drawn between the inside and the outside of a polyhedron. Given a polyhedral chain x_p, there are several conventions in common use to determine whether a given point $t \in E^d$ is to be considered inside or outside of the corresponding polyhedral point set $P \subseteq E^d$. These include the parity, the oriented multiply-covered, and the nonzero winding number cnvention[6].

The parity convention determines the state of a point by the parity of the number of intersections between faces of the polyhedron P and a straight line drawn from the point to infinity in any direction (Figure 5). Therefore

$$t \in P \Leftrightarrow f_{xp}(t) \text{ is odd.}$$

The oriented multiply-covered convention defines an

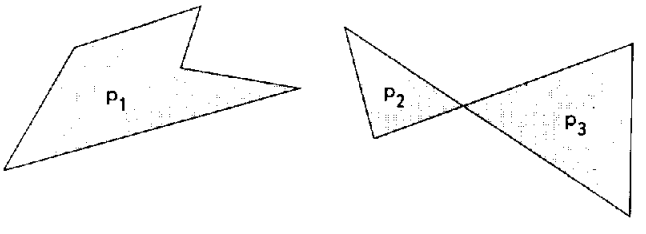

Figure 2. $p_1 + p_2 + p_3$

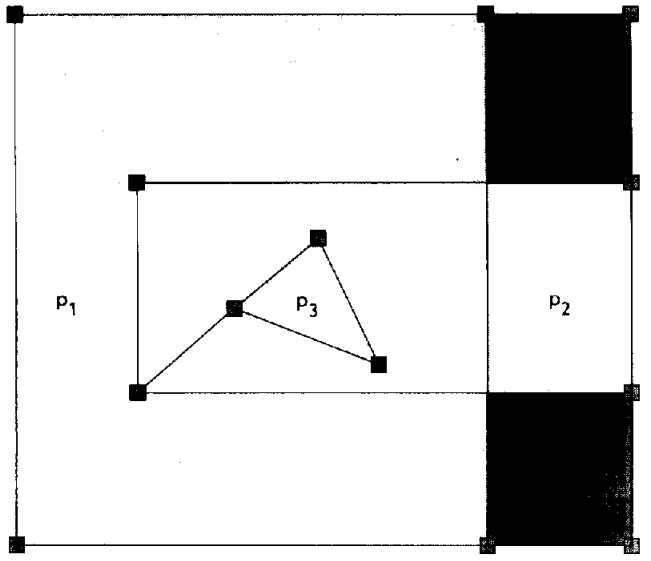

Figure 3. $p_1 + p_2 + p_3$

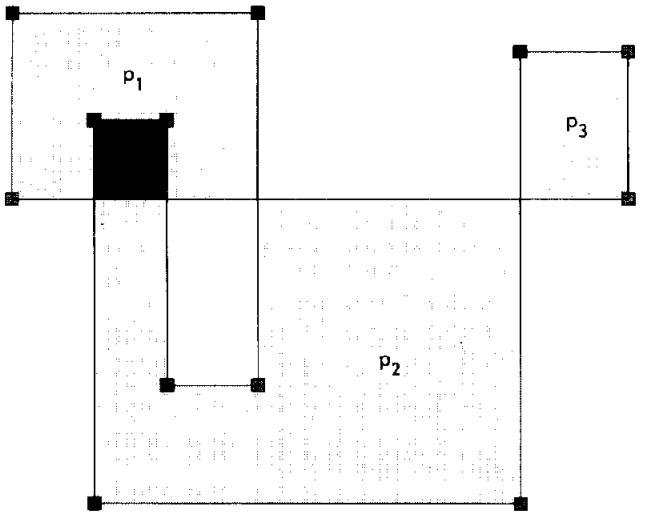

Figure 4. $p_1 + p_2 + p_3$

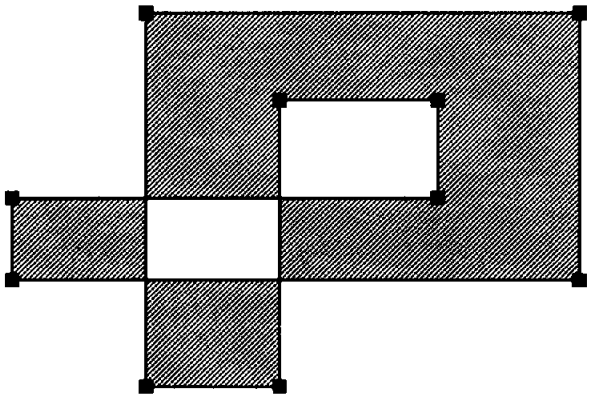

Figure 5. Parity convention[6]

orientation for the boundary of a polyhedron such that one side of each boundary segment defines material (i.e. inside) and the other side defines holes (outside), as in Figure 6. Material that overlaps material is simply material. Each hole is able to annihilate exactly one

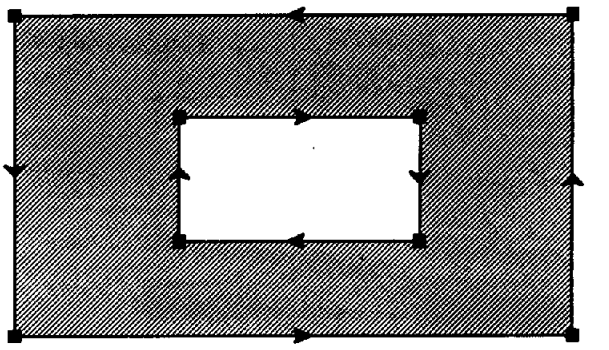

Figure 6. Oriented multiply-covered convention[6]

layer of material. Moreover, holes in space are ignored. It is

$$t \in P \Leftrightarrow f_{xp}(t) > 0$$

Applications of this convention have been given by Newell and Sequin[6].

For polygons, another way of describing the oriented multiply-covered convention is to say that a point is considered to be inside the polygon if the winding number of the boundary with respect to that point is greater than zero. The winding number of a polygon boundary with respect to a given point is defined as the net number of times that a point on the boundary wraps around the given point while the boundary point makes one complete traversal of the boundary.

To eliminate some of the flaws of this convention, Newell and Sequin[6] propose yet another convention, the so-called nonzero winding number convention. According to this convention, a point is considered to be inside a polygon if the winding number of the boundary with respect to that point is nonzero (Figures 7 and 8). Using the notation of polyhedral chains, this approach can be generalized to arbitrary dimensions and described in a much simpler way. The winding number with respect to a point t is simply $f_{xp}(t)$. According to the nonzero winding number convention, it is

$$t \in P \Leftrightarrow f_{xp}(t) \neq 0$$

According to any of the above inside–outside

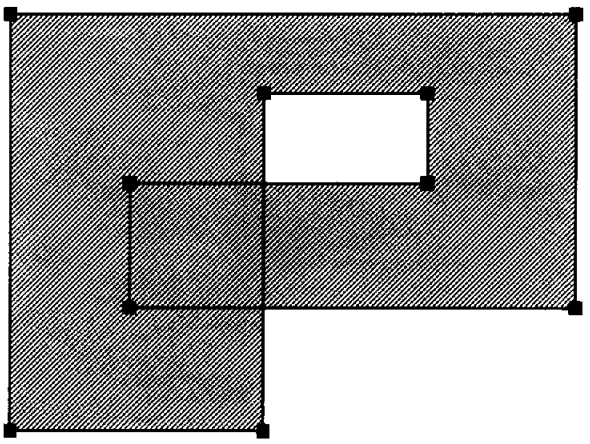

Figure 7. Nonzero winding number convention[6]

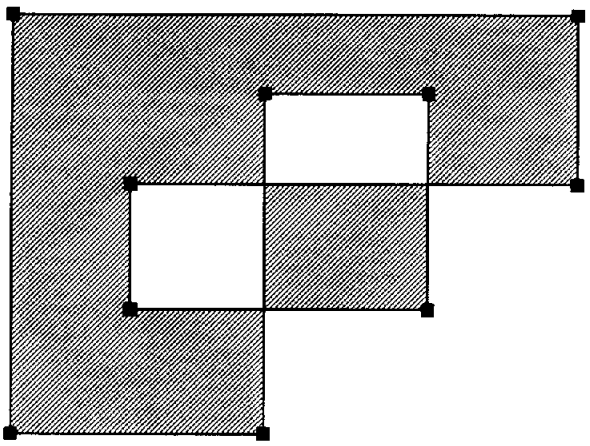

Figure 8. Nonzero winding number convention[6]

conventions, all equivalent chains correspond to the same point set in E^d.

If multiple layers or negative space are not needed, the polyhedral point sets may as well be represented by means of non-negative polyhedral chains x_p where $f_{xp}(t)$ is 0 for outside points and positive otherwise. Convex polyhedral chains, where all cells p_i are convex, also have important applications because the cells can be described in a very simple way as an intersection of halfspaces. The following discussion will focus on convex, non-negative polyhedral chains, which will also be referred to as convex chains. Unlike simple polyhedra, convex chains are closed under all set operators such as intersection (Figure 9). Note also that for any polyhedral chain x in E^d and a given inside–outside convention, there is a convex chain x′ in E^d, such that x and x′ represent the same polyhedral point set.

Now consider a database consisting of a collection of general d-dimensional polyhedra in Euclidean space E^d. To support set and search operators, let the polyhedra be represented as convex chains. Formally, each polyhedron P is represented as a convex chain in E^d,

$$x_P = \sum_{i=1}^{m} p_i$$

with all p_i being convex. A point is inside P if and only

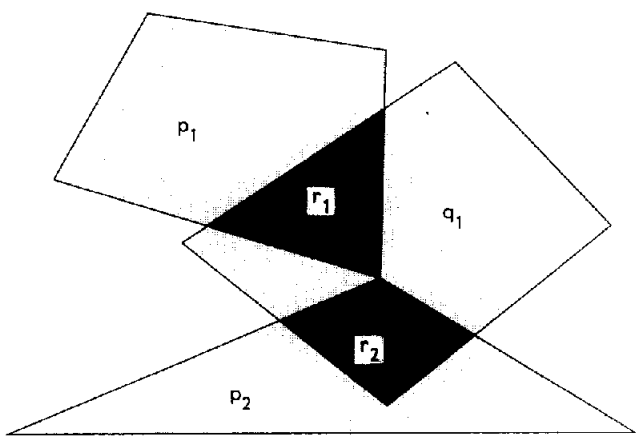

Figure 9. $x_P = p_1 + p_2$, $x_Q = q_1$, $x_{P \cap Q} = r_1 + r_2$

if it is inside any of the cells p_i, i.e.

$$t \in P \Leftrightarrow t \in p_i \text{ for some } i = 1, \ldots, m$$

Note that it is not required for the p_i to be mutually disjoint. Disjointness is hard to maintain and provides no particular advantages for the operators we intend to support.

Convex chains can be viewed as a special case of the constructive solid geometry (CSG) approach proposed by Requicha[3]. CSG represents a geometric object by a binary tree whose leaves correspond to geometric primitives and whose internal nodes correspond to set operators. In the case discussed here, the cells are the primitives, and all internal nodes correspond to the union operator.

h VECTOR

The next question is how to represent the convex cells p_i. It is well known that any convex polyhedron in E^d is the intersection of closed halfspaces in E^d. Each halfspace in turn can be represented as a product $h \cdot H$ where H is an oriented $(d-1)$-dimensional hyperplane and h is an integer. In particular, let $a \in E^d - \{0\}$ and $c \in E^1$; then the $(d-1)$-dimensional set $H(a, c) = \{x \in E^d : x \cdot a = c\}$ defines a hyperplane in E^d. A hyperplane H(a, c) defines two closed halfspaces $1 \cdot H(a, c) = \{x \in E^d : x \cdot a \geqslant c\}$ and $-1 \cdot H(a, c) = \{x \in E^d : x \cdot a \leqslant c\}$. For completeness, we define $0 \cdot H(a, c)$ as E^d. A hyperplane H supports a convex cell p if $H \cap p \neq \varnothing$ (where in this and the following two expressions the operator $\cap$ denotes the simple intersection operator and not the regularized one, as defined by Tilove[2]) and it is $p \subseteq 1 \cdot H$ or $p \subseteq -1 \cdot H$. If H is any hyperplane supporting p, then $H \cap p$ is a face of p. The faces of dimension 1 are called edges; those of dimension 0 are called vertices. A supporting hyperplane H is called a boundary hyperplane if the face $H \cap p$ is of dimension $d-1$.

Let $\mathbf{H} = H_1 H_2 \ldots H_{|\mathbf{H}|}$ denote a vector of $(d-1)$-dimensional oriented hyperplanes such that H_i is in $\mathbf{H}$ if and only if H_i is a boundary hyperplane of some cell p in the database. For simplicity, we require that for each $(d-1)$-dimensional face f of any convex cell p there be a $(d-1)$ dimensional face g of a polyhedron P in the database, such that f and g are both subsets of the same hyperplane. Then $\mathbf{H}$ can be restricted to include only those hyperplanes that are boundary hyperplanes of some polyhedron P in the database.

Now each cell in the database can be represented as an $|\mathbf{H}|$-dimensional vector, consisting of the digits 1, 0 and −1. Each 1 or −1 selects a hyperplane from $\mathbf{H}$, and associates an orientation with it. The resulting set of halfspaces represents a convex polyhedron. More formally, each cell p is represented as a ternary vector $\mathbf{h}_p = \{0, 1, -1\}^{|\mathbf{H}|}$, such that $p = \cap_{i=1}^{|\mathbf{H}|} (\mathbf{h}_p)_i \cdot H_i$. an example is given in Figure 10.

Note that for a given cell p, $\mathbf{h}_p$ is by no means unique. For example, suppose that the hyperplane H_i is not a boundary hyperplane of cell p, but p is a subset of the halfspace $1 \cdot H_i$. Then it makes no difference whether $(\mathbf{h}_p)_i$ is 0 or 1; the hyperplane H_i is redundant with respect to p. For a given p, the set of all possible $\mathbf{h}_p$

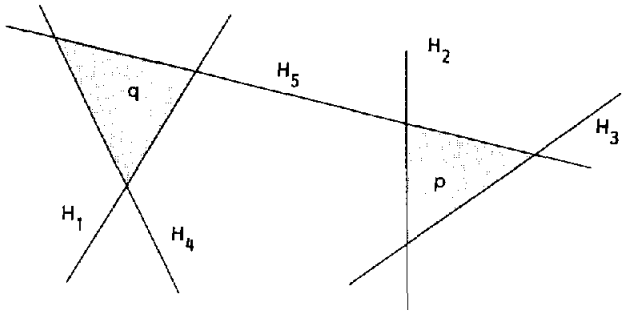

Figure 10. $\mathbf{h}_p = (0, 1, -1, 0, -1)$, $\mathbf{h}_q = (-1, 0, 0, 1, -1)$

vectors is an equivalence class which contains a unique vector with the minimum number of nonzero components. For this unique minimum $\mathbf{h}_p$ every nonzero component corresponds to a boundary hyperplane of p. Note that there is no unique minimum vector to represent the empty set. On the other hand, there is a unique minimum vector to represent the whole space E^d, i.e. the vector $0^{|\mathbf{H}|}$.

The insertion of new polyhedra is performed by adding new hyperplanes to **H**, if necessary. For simplicity it is assumed that the components of the ternary vectors $\mathbf{h}_p$ default to zero if they are not explicitly specified. Under this assumption an insertion does not change the representations of existing cells.

The deletion of polyhedra may cause some hyperplanes in **H** to become redundant with respect to all cells in the database. The deletion of such a hyperplane from **H** corresponds to a compression of each vector $\mathbf{h}_p$ by one component. Although it may not be efficient to perform this update after each single deletion, it might be worthwhile to do such a clean-up after a certain number of deletions. Otherwise a large number of redundant hyperplanes will inflate the representations unnecessarily.

If **H** contains many hyperplanes, as may well be the case, the explicit storage representation of $\mathbf{h}_p$ is not feasible. However, the simple structure of $\mathbf{h}_p$ allows many alternative data structures to be used. As one example, $\mathbf{h}_p$ could be represented by a set of (signed) pointers, pointing to those hyperplanes that correspond to the nonzero elements.

Note that this approach to represent polyhedra abandons the notion of vertex completely. Representation of cells by **h** vectors has both conceptual and computational advantages. To represent cells in terms of boundary hyperplanes rather than in terms of vertices is usually the most space-efficient way because no adjacency relations need to be stored. This becomes especially important in higher dimensions as the number of adjacencies may grow exponentially in the dimension; see Preparata and Shamos[17], pp 89–93. Furthermore, it seems that vertices are not necessary for the search and set operators we intend to support. Search operators such as point location or range search can be supported efficiently by search structures that are based on hyperplanes rather than vertices; examples for such structures are the binary space partitioning tree[18] or the cell tree[19]. All set operators can be computed efficiently without using vertices by decomposing them into two parts:

- an operation on the **h** vectors without references to the geometric coordinates of the hyperplanes
- a generic operation that tests whether a vector $\mathbf{h}_p$ is null, i.e. whether the intersection of the halfspaces specified by $\mathbf{h}_p$ is empty

This decomposition will be described in detail in the following section.

SET OPERATORS

Let P and Q be two general polyhedral point sets. We now show that the computation of any set operator on P and Q can be decomposed into

- operations on the corresponding **h** vectors
- deleting the null vectors from the set of resulting **h** vectors

The following propositions are easily verified with the definitions of set operators and of polyhedral chains.

Proposition 1

Let P and Q be represented by convex chains $x_P = \Sigma_{j=1}^{m} p_j$ and $x_Q = \Sigma_{k=1}^{l} q_k$. Then

$$x_{P \cup Q} = x_P + x_Q$$

$$x_{P \cap Q} = \sum_{j,k} (p_j \cap q_k)$$

$$x_{\bar{p}} = x_{\bar{p}_1 \cap \ldots \cap \bar{p}_m}$$

$$x_{P-Q} = x_{P \cap \bar{Q}}$$

Proposition 2

Let $\mathbf{h}_p$ denote an **h** vector of a cell p. Then $x_{\bar{p}} = -\mathbf{h}_p \cdot \mathbf{H}$. (For an example see Figure 11. Note that the length of this chain equals the number of nonzero components of the vector $\mathbf{h}_p$. It is therefore desirable to keep this number low, possibly at its minimum.)

Proposition 3

Let $\mathbf{h}_p$ and $\mathbf{h}_q$ denote the **h** vectors for two cells p and q respectively. Then $\mathbf{h}_{p \cap q}$ can be computed using Table 1 for each component $(\mathbf{h}_{p \cap q})_i$. In those cases denoted by *, the hyperplane H_i separates p and q, i.e. $p \subseteq 1 \cdot H_i$ and $q \subseteq -1 \cdot H_i$, or vice versa, and therefore $p \cap q = \varnothing$. (Note that both the intersection and the complementation

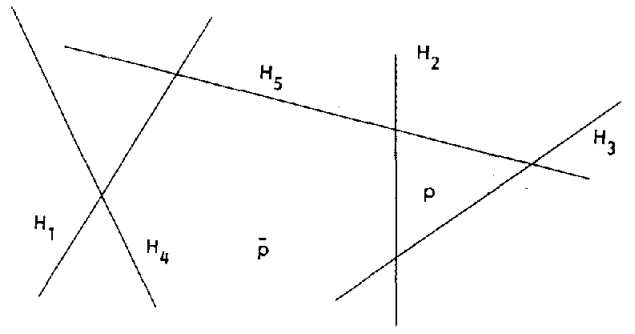

Figure 11. $\mathbf{h}_p = (0, 1, -1, 0, -1)$, $x_{\bar{p}} = -1 \cdot H_2 + 1 \cdot H_3 + 1 \cdot H_5$

Table 1. $(\mathbf{h}_{p\cap q})_i$

$(\mathbf{h}_{p\cap q})_i$		$(\mathbf{h}_q)_i$		
		0	1	−1
$(\mathbf{h}_p)_i$	0	0	1	−1
	1	1	1	*
	−1	−1	*	−1

operator are defined on the components of the **h** vector. The components are independent of each other and can therefore be processed in parallel. In particular, a systolic array[20] seems to be promising for an efficient implementation.)

It follows from propositions 1–3 that for any set operation &, the **h** vector representation of P&Q can be computed from the **h** vector representations of P and Q. However, the **h** vectors in the resulting representation may not be minimal. Also, some vectors may define empty sets, as condition * is a sufficient, but not a necessary condition for nonintersection. Two cells p and q may not intersect, but there is no component $(\mathbf{h}_{p\cap q})_i$ where condition * occurs. In that case, the resulting vector $\mathbf{h}_{p\cap q}$ defines an empty set. Although that case is consistent with the data model given here, it is not desirable. A large number of empty cells p_j in the convex chains $x_p = \sum_{j=1}^{m} p_j$ representing the polyhedra in the database may slow down the system performance considerably. An efficient means for detecting empty cells is therefore needed.

One approach would be to abandon the concept of minimality and to increase the number of nonzero components in the **h** vector, possibly to its maximum, i.e.

$$(\mathbf{h}_p)_i = \begin{cases} 1 & \text{if } p \subseteq \ \ 1 \cdot H_i \\ -1 & \text{if } p \subseteq -1 \cdot H_i \\ 0 & \text{otherwise} \end{cases}$$

Each nonzero component increases the chance that a separating hyperplane is found, i.e. that condition * is met if two polyhedra do not intersect. If each **h** vector had a maximum number of nonzero components, then a separating hyperplane would be detected immediately; i.e. condition * would be a necessary and sufficient condition for nonintersection. On the other hand, this approach makes the identification of boundary hyperplanes, and therefore the cell complementation and boundary retrieval operations, much more difficult. Also, computing the above function for each cell p in the database requires an immense amount of computation and produces a lot of data that is probably never needed.

A garbage collector seems to be a better solution. Each time a new cell is computed as the intersection of two cells, the new cell is tagged. A background process (the garbage collector) keeps checking the tagged cells in the database for emptiness. If a cell is found to be not empty, it is untagged. Otherwise, it is deleted from storage and from the chains that contain that cell. Unfortunately, the representation of cells by means of their **h** vectors does not lead to an efficient algorithm to check cells for emptiness. A better approach to this problem, based on geometric duality, is presented in a theoretical companion to this paper[21]. There it is shown that the time complexity to check two cells for intersection is polylogarithmic and therefore sublinear in the number of boundary hyperplanes of any of the cells.

To avoid duplicating computational effort and losing information, it is proposed that the results obtained by the garbage collector be cached. Whenever a cell intersection $p \cap q$ is computed a second time, it should be immediately clear from the vectors $\mathbf{h}_p$ and $\mathbf{h}_q$ if the intersection $p \cap q$ is empty or not. Whenever the garbage collector checks a new cell $r = p \cap q$, it either discovers a separating hyperplane (if p and q are disjoint) or it discovers that there are no separating hyperplanes (if p and q intersect). This result can be cached by extending the notion of the **h** vector to capture more information in the following way.

Given a cell p and a hyperplane H_i, there are two pieces of information about the relationship between p and H_i that are of interest and that should be cached in the component $(\mathbf{h}_p)_i$:

- Which side of H_i is p on? Possible answers are to the left (−1), to the right (+1), H_i intersects the interior of p (I), or unknown (0).
- Is H_i a boundary hyperplane of p? Yes (Y), No (N), or unknown (0).

Clearly, if H_i intersects the interior of p then it can not be a boundary hyperplane of p. Also, if it is not known on which side of H_ip is on, then H_i must not be a boundary hyperplane; otherwise, *p* would not be defined properly. Hence, of the twelve possible combinations, only eight combinations make sense; see Table 2.

Each cell p is represented as a vector $\mathbf{h}_p^+$ with the following semantics. Each component is one of the eight combinations described in Table 3. Components that are not explicitly specified default to (0, N). It turns out that these $\mathbf{h}_p^+$ vectors are closed with respect to intersection of two cells. $(\mathbf{h}_{p\cap q}^+)_i$ is given by Table 4.

In those cases denoted by * in Table 4, the hyperplane H_i separates p and q, i.e. $p \subseteq 1 \cdot H_i$ and $q \subseteq -1 \cdot H_i$, or vice versa, and therefore $p \cap q = \varnothing$. Then there must be at least one separating hyperplane H_j that is a boundary hyperplane of p or q. In this case $(\mathbf{h}_{p\cap q})_j$ corresponds to one of the cases denoted by * or by †. Therefore, a new cell $r = p \cap q$ is certainly empty if

Table 2. Combinations of information possible

		Side			
		1	−1	I	0
Boundary hyperplane?	Y	ok	ok	–	–
	N	ok	ok	ok	ok
	0	ok	ok	–	–

any component $(\mathbf{h}_{p\cap q})_i$ corresponds to one of the cases denoted by *. Otherwise, it needs to be tagged if and only if there is at least one component $(\mathbf{h}_{p\cap q})_i$ that corresponds to one of the cases denoted by †. Then the garbage collector will check whether the cell $p \cap q$ is in fact empty.

If a tagged cell $r = p \cap q$ is found empty, this result can be cached by the following updates. Let H_i be a separating hyperplane and without loss of generality, let $p \subseteq 1 \cdot H_i$ and $q \subseteq -1 \cdot H_i$.

if $(\mathbf{h}_p^+)_i = (0, N)$ **then** $(\mathbf{h}_p^+)_i := (1, N)$

if $(\mathbf{h}_q^+)_i = (0, N)$ **then** $(\mathbf{h}_q^+)_i := (-1, N)$

If, on the other hand, a tagged cell $r = p \cap q$ is found not empty, this result can be cached as follows. There are no separating hyperplanes between p and q, i.e. for any hyperplane H_i that may be a boundary hyperplane of p, either q lies on the same side of H_i as p, or H_i intersects the interior of q. A similar condition holds for any hyperplane H_i that may be a boundary hyperplane of q. Therefore,

if $\{(\mathbf{h}_p^+)_i = (\pm 1, Y)$ **or** $(\mathbf{h}_p^+)_i = (\pm 1, 0)\}$ **and** $(\mathbf{h}_q^-)_i = (0, N)$
then if $H_i \cap q = \varnothing$ **then** $(\mathbf{h}_q^+)_i := (\pm 1, N)$
else $(\mathbf{h}_q^+)_i := (I, N)$
if $\{(\mathbf{h}_q^+)_i = (\pm 1, Y)$ **or** $(\mathbf{h}_q^+)_i = (\pm 1, 0)\}$ **and** $(\mathbf{h}_p^+)_i = (0, N)$
then if $H_i \cap p = \varnothing$ **then** $(\mathbf{h}_p^+)_i := (\pm 1, N)$
else $(\mathbf{h}_p^+)_i := (I, N)$

Whenever $p \cap q$ is computed again, the vectors $\mathbf{h}_p^+$ and $\mathbf{h}_q^+$ indicate whether p and q intersect or not. If they do intersect, the intersection cell will not have to be tagged again.

When a new cell is inserted into the database, most of the components of its **h** vector are zero. As set operations are performed on the stored polyhedra, the database evolves. More and more zero components of the **h** vectors are replaced, and the vectors carry more and more information. Therefore, it will happen less and less frequently that a new cell has to be tagged and checked for emptiness. Also, at some point it may be more efficient to test a new cell $r = p \cap q$ for emptiness by checking the hyperplanes that may be separating ones (i.e. the ones that correspond to components with a † in Table 4) one by one if they are actually separating. If there are few enough components with a †, this may be simpler and faster than using the dual approach proposed in the companion to this paper[21].

Problems such as complementation, point location or boundary retrieval may be solved by looking at only those hyperplanes that may be boundary hyperplanes, i.e. the hyperplanes H_i where $(\mathbf{h}_p^+)_i$ is $(\pm 1, Y)$ or $(\pm 1, 0)$.

There are variations to this approach. First, it may be preferable to always identify the boundary hyperplanes of each cell, i.e. to avoid vector components $(\pm 1, 0)$. This can be achieved by extending the garbage collector, such that each time an intersection cell is found not empty, its boundary hyperplanes are computed and the **h** vector is updated accordingly. Second, a decision may be taken to simplify the update procedure above by introducing additional aggregation states (1I, N) and (−1I, N) which represent $\{(1, N)$ or $(I, N)\}$ and $\{(-1, N)$ or $(I, N)\}$, respectively. Then the set of updates for the case that p and q intersect can be simplified to

if $\{(\mathbf{h}_p^+)_i = (\pm 1, Y)$ **or** $(\mathbf{h}_p^+)_i = (\pm 1, 0)\}$ **and** $(\mathbf{h}_q^+)_i = (0, N)$
then $(\mathbf{h}_c^+)_i := (\pm 1I, N)$
if $\{(\mathbf{h}_q^+)_i = (\pm 1, Y)$ **or** $(\mathbf{h}_q^+)_i = (\pm 1, 0)\}$ **and** $(\mathbf{h}_p^+)_i = (0, N)$
then $(\mathbf{h}_p^+)_i := (\pm 1I, N)$

In particular, it is not necessary any more to check any hyperplane H_i that is a boundary hyperplane of p (q) if it intersects the interior of q (p), i.e. if $H_i \cap q$ $(H_i \cap p) = \varnothing$. New vectors $\mathbf{h}_p^+$ and $\mathbf{h}_q^+$ still indicate whether p and q intersect or not. If they do intersect, the intersection cell will not have to be tagged again. As proven in the companion to this paper[21], the time

Table 3. Possible combinations and meanings

$(\mathbf{h}_p^+)_i$	Meaning
(1, Y)	$p \subseteq 1 \cdot H_i$, H_i is a boundary hyperplane of p
(−1, Y)	$p \subseteq -1 \cdot H_i$, H_i is a boundary hyperplane of p
(1, 0)	$p \subseteq 1 \cdot H_i$, H_i may or may not be a boundary hyperplane of p
(−1, 0)	$p \subseteq -1 \cdot H_i$, H_i may or may not be a boundary hyperplane of p
(1, N)	$p \subseteq 1 \cdot H_i$, H_i is not a boundary hyperplane of p
(−1, N)	$p \subseteq -1 \cdot H_i$, H_i is not a boundary hyperplane of p
(I, N)	H_i intersects the interior of p
(0, N)	H_i is not a boundary hyperplane of p

Table 4. $(\mathbf{h}_{p\cap q}^+)_i$

$(\mathbf{h}_{p\cap q}^-)_i$		$(\mathbf{h}_q^+)_i$							
		(1, Y)	(−1, Y)	(1, 0)	(−1, 0)	(1, N)	(−1, N)	(I, N)	(0, N)
$(\mathbf{h}_p^+)_i$	(1, Y)	(1, 0)	*	(1, 0)	*	(1, 0)	*	(1, 0)	(1, 0)†
	(−1, Y)	*	(−1, 0)	*	(−1, 0)	*	(−1, 0)	(−1, 0)	(−1, 0)†
	(1, 0)	(1, 0)	*	(1, 0)	*	(1, 0)	*	(1, 0)	(1, 0)†
	(−1, 0)	*	(−1, 0)	*	(−1, 0)	*	(−1, 0)	(−1, 0)	(−1, 0)†
	(1, N)	(1, 0)	*	(1, 0)	*	(1, N)	*	(1, N)	(1, N)
	(−1, N)	*	(−1, 0)	*	(−1, 0)	*	(−1, N)	(−1, N)	(−1, N)
	(I, N)	(1, 0)	(−1, 0)	(1, 0)	(−1, 0)	(1, N)	(−1, N)	(0, N)	(0, N)
	(0, N)	(1, 0)†	(−1, 0)†	(1, 0)†	(−1, 0)†	(1, N)	(−1, N)	(0, N)	(0, N)

complexity to check the condition $H_i \cap q\,(H_i \cap p) = \varnothing$ for a particular hyperplane H_i is logarithmic in the number of boundary hyperplanes of q (p).

CONCLUSIONS

In this paper, the concept of polyhedral chains has been introduced as a representation scheme for polyhedra in arbitrary dimensions, and a scheme that is based on convex chains has been presented in detail. The scheme is conceptually simple, facilitates the efficient computation of set operators, and seems well suited for parallel processing. Each cell is represented as an intersection of halfspaces, encoded in a vector. The notion of vertices is abandoned completely as it is not needed for the set and search operators that are intended to be supported.

Based on this representation, a scheme to decompose the computation of set operators into two steps is described. The first step consists of a collection of vector operations; the second step is a garbage collection where vectors that represent empty cells are eliminated. All results of the garbage collection are cached in the vectors in such a way that no computations have to be duplicated. As the database is learning more and more information through the garbage collector, it will be able to detect empty cells immediately such that no additional test for emptiness is required. No special treatment of singular intersection cases is needed. This approach to set operations is significantly different from algorithms that have been proposed in the past. It makes no use of vertices, edges or adjacencies, it does not induce a space partioning to localize the set operations, and it is applicable to polyhedra in arbitrary dimensions.

Future work will focus on an experimental implementation of this scheme. Clearly, a systolic array[20] seems to be promising for an efficient implementation. Also, this approach is believed to be more amenable to parallel processing than a vertex-based approach. In particular, the components of the **h** vectors are processed independently from each other. Therefore, it seems possible to assign one processor to each hyperplane in **H** and to carry out a significant fraction of the necessary computations locally without interprocessor communication.

REFERENCES

1 **Günther, O and Wong, E** 'A dual space representation for geometric data' in *Proc. 13th Int. Conf. Very Large Data Bases* Brighton, UK (September 1987)

2 **Tilove, R B** 'Set membership classification: a unified approach to geometric intersection problems' *IEEE Trans. Comput.* Vol C-29 No 10 (1980) pp 874–883

3 **Requicha, A A G** 'Representations for rigid solids: theory, methods, and systems' *Comput. Surveys* Vol 12 No 4 (1980)

4 **Besl, P J and Jain, R C** 'Three-dimensional object recognition' *Comput. Surveys* Vol 17 No 1 (1985)

5 **Faux, I D and Pratt, M J** *Computational geometry for design and manufacture* Ellis Horwood, Chichester, UK (1979)

6 **Newell, M E and Sequin, C H** 'The inside story on self-intersecting polygons' *LAMBDA* (Second Quarter 1980)

7 **Dantzig, G B** *Linear programming and its extensions* Princeton University Press, Princeton, NJ, USA (1963)

8 **Stonebraker, M, Sellis, T and Hanson, E** 'An analysis of rule indexing implementations in data base systems' in *Proc. 1st Int. Conf. Expert Data Base Systems* (April 1986)

9 **Franklin, W R** 'Efficient polyhedron intersection and union' *Graphics Interface 82* (May 1982) pp 73–80

10 **Mantyla, M and Sulonen, R** 'GWB: A solid modeler with Euler operators' *IEEE Comput. Graph. Appl.* (September 1982) pp 17–31

11 **Mantyla, M and Tamminen, M** 'Localized set operations for solid modeling' *Comput. Graph.* (July 1983) pp 279–288

12 **Tilove, R B, Requicha, A A G and Hopkins, M R** 'Efficient editing of solid models by exploiting structural and spatial locality' *Comput. Aided Geom. Des.* Vol 1 (1984) pp 227–239

13 **Requicha, A A G and Voelcker, H B** 'Boolean operations in solid modeling: boundary evaluation and merging algorithms' in *Proc. IEEE* (January 1985), pp 30–44

14 **Carlbom, I** 'An algorithm for geometric set operations using cellular subdivision techniques' *IEEE Comput. Graph. Appl.* (May 1987) pp 44–55

15 **Putnam, L K and Subrahmanyam, P A** 'Boolean operations on n-dimensional objects' *IEEE Comput. Graph. Appl.* Vol 6 No 6 (June 1986)

16 **Whitney, H** *Geometric integration theory* Princeton University Press, Princeton NJ, USA (1957)

17 **Preparata, F P and Shamos, M I** *Computational geometry* Springer-Verlag, New York, NY, USA (1985)

18 **Fuchs, H, Kedem, Z and Naylor, B** 'On visible surface generation by a priori tree structures' *Comput. Graph.* Vol 14 No 3 (June 1980)

19 **Günther, O** *Efficient structures for geometric data management, Lecture Notes in Computer Science No 337* Springer-Verlag, Berlin, FRG (1988)

20 **Kung, H T** 'Systolic arrays' *Computer* Vol 11 No 4 (1979) pp 397–409

21 **Günther, O and Wong, E** 'A dual approach to detect polyhedral intersections in arbitrary dimensions' in *Proc. 25th Annual Allerton Conf. Communications, Control and Computing* (October 1987)