PERGAMON Computers and Mathematics with Applications 40 (2000) 971–983

An International Journal
computers & mathematics
with applications

www.elsevier.nl/locate/camwa

An Iterative Algorithm for Finding a Nearest Pair of Points in Two Convex Subsets of R^n

B. LLANAS AND M. FERNANDEZ DE SEVILLA
Dpto. de Matemática e Informática, E.T.S.I. de Caminos
Ciudad Universitaria, 28040, Madrid, Spain

V. FELIU
Dpto. de Ingeniería Eléctrica, E.T.S.I de Industriales
Campus Universitario s/n, 13071, Ciudad Real, Spain

(Received and accepted January 2000)

Abstract—We present an algorithm for finding a nearest pair of points in two convex sets of R^n, and therefore, their distance. The algorithm is based on the fixed-point theory of nonexpansive operators on a Hilbert space. Its practical implementation requires a fast projection algorithm. We introduce such a procedure for convex polyhedra. This algorithm effects a local search in the faces using visibility as a guide for finding the global minimum. After studying the convergence of both algorithms, we detail computer experiments on polyhedra (projection and distance). In the case of distances, these experiments show a sublinear time complexity relative to the total number of vertices.

Keywords—Polyhedra, Local search, Projection algorithms, Euclidean distance, Nonexpansive operators.

1. INTRODUCTION

The efficient computation of distances between two bodies is a crucial element of many path planners in robotics and other applications. As polyhedra are good models of real objects, fast methods for computing the distance between them are necessary.

Most of the algorithms proposed to this date, the fastest ones included, have a time complexity linear or superlinear (in relation to the sum of vertices of both polyhedra). This implies that their performance for polyhedra having a great number of faces (for example, in advanced virtual reality simulations [1]) can be poor.

We can classify most of the known (nonrandomized) algorithms according to the dimension n and the type of sets they handle.

- $n = 2$.
  - Algorithms for finding the distance between planar polygons have been studied in [2,3].

The research reported in this paper was financially supported (I+D 0105/94) by the Consejería de Educación y Cultura de la Comunidad de Madrid and the electric company IBERDROLA.

0898-1221/00/$ - see front matter Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX
PII: S0898-1221(00)00212-1

- $n = 3$.
  - *Polyhedra.* (Intersection of a finite number of closed half spaces.)
    * Methods based on a hierarchical decomposition of the polyhedra as proposed in [4].
    * Methods based on an exhaustive computation of the distance between all the pairs of faces of both polyhedra (brute force method) [5].
    * Methods based on a sequence of constrained minimizations [6].
    * Local methods that only process a subset of every polyhedron [7,8].
  - *General Sets.* Gilbert *et al.* [9] have generalized the algorithm presented in [10] to the case of objects with curved boundary. This problem is also studied in [7]. Related collision detection problem methods which use bounding volume hierarchies ("BV-trees") are described in [11].
- n **arbitrary**
  - *Polytopes.* (Intersection of a finite number of closed half spaces.)
    * Methods based on a descent procedure which works on the distance between elementary polytopes contained in the convex sets [10].
    * Methods based on duality [12].
    * Local Methods [13].
  - *General Sets.* An algorithm for the case of convex sets defined as the intersection of an infinite number of closed half spaces was studied in [14].

In Section 2 of this article, we give the theoretic foundation of an iterative algorithm for finding a nearest pair of points in two convex, bounded, and closed subsets of R^n. We will call it "swap" algorithm (SA). This method differs from those cited above. To be practical, it must use a fast algorithm for projecting points onto a convex set of R^n.

In Section 3, we introduce such a fast projection algorithm onto a convex polyhedron. This algorithm performs a local search over the faces visible from the point to be projected. Convergence to the nearest face is proved. From now on, we denote this algorithm by "local search algorithm based in faces" (LSABF).

In Section 4, we present numerical experiments on a wide variety of polyhedra, including large ones (more than 1000 faces).

The experiments show that LSABF is a fast projection procedure. If we combine it with SA, we get an algorithm for finding the distance between polyhedra which has a sublinear behavior.

2. THE "SWAP" ALGORITHM

From now on, R^n will denote the Euclidean n-space.

Let $\mathcal{P}$ and $\mathcal{Q}$ be two convex, bounded, and closed sets in R^n. Our problem is to find a point $\mathbf{a} \in \mathcal{P}$ and other point $\mathbf{b} \in \mathcal{Q}$ such that

$$d(\mathbf{a}, \mathbf{b}) = \min_{\substack{\mathbf{x} \in P \\ \mathbf{y} \in Q}} d(\mathbf{x}, \mathbf{y}) \equiv d(\mathcal{P}, \mathcal{Q}). \tag{1}$$

($d(.,.)$ denotes the Euclidean distance.)

LEMMA 1. *Let $\mathcal{P}$ and $\mathcal{Q}$ be convex, bounded, and closed subsets of R^n and $\pi_\mathcal{P}$ and $\pi_\mathcal{Q}$ projection operators on $\mathcal{P}$ and $\mathcal{Q}$, respectively.*

(a) *If $\mathbf{a} \in \mathcal{P}$ is such that there exists $\mathbf{b} \in \mathcal{Q}$ such that $d(\mathbf{a}, \mathbf{b}) = d(\mathcal{P}, \mathcal{Q})$, then $\mathbf{a}$ is a fixed point of the operator: $\pi_\mathcal{P} \circ \pi_\mathcal{Q} : \mathcal{P} \to \mathcal{P}$.*

(b) *Conversely, if $\mathbf{a} \in \mathcal{P}$ is a fixed point of $\pi_\mathcal{P} \circ \pi_\mathcal{Q}$, then $d(\mathbf{a}, \pi_\mathcal{Q}(\mathbf{a})) = d(\mathcal{P}, \mathcal{Q})$.*

PROOF a. By (1), $\pi_\mathcal{Q}(\mathbf{a}) = \mathbf{b}$ and $\pi_\mathcal{P}(\mathbf{b}) = \mathbf{a}$. So we have

$$\pi_\mathcal{P}(\pi_\mathcal{Q}(\mathbf{a})) = \pi_\mathcal{P}(\mathbf{b}) = \mathbf{a}.$$

PROOF b. We have to prove that

$$d(\mathbf{a}, \pi_{\mathcal{Q}}(\mathbf{a})) \leq d(\mathbf{x}, \mathbf{y}), \qquad \forall \mathbf{x} \in \mathcal{P}, \quad \forall \mathbf{y} \in \mathcal{Q},$$

note that for arbitrary $\mathbf{x} \in P$ and $\mathbf{y} \in Q$,

$$\begin{aligned} |\mathbf{x} - \mathbf{y}|^2 &= |\mathbf{x} - \mathbf{a} + \mathbf{a} - \pi_{\mathcal{Q}}(\mathbf{a}) + \pi_{\mathcal{Q}}(\mathbf{a}) - \mathbf{y}|^2 \\ &= |\mathbf{x} - \mathbf{a} + \pi_{\mathcal{Q}}(\mathbf{a}) - \mathbf{y}|^2 + |\mathbf{a} - \pi_{\mathcal{Q}}(\mathbf{a})|^2 + 2(\mathbf{x} - \mathbf{a}).(\mathbf{a} - \pi_{\mathcal{Q}}(\mathbf{a})) \\ &\quad + 2(\pi_{\mathcal{Q}}(\mathbf{a}) - \mathbf{y}).(\mathbf{a} - \pi_{\mathcal{Q}}(\mathbf{a})). \end{aligned} \tag{2}$$

The hypothesis $\pi_{\mathcal{P}}[\pi_{\mathcal{Q}}(\mathbf{a})] = \mathbf{a}$ implies that

$$(\mathbf{x} - \mathbf{a}).(\mathbf{a} - \pi_{\mathcal{Q}}(\mathbf{a})) \geq 0, \qquad \forall \mathbf{x} \in \mathcal{P}. \tag{3}$$

Also, since $\pi_{\mathcal{Q}}(\mathbf{a})$ is the projection of $\mathbf{a}$ onto $\mathcal{Q}$,

$$(\pi_{\mathcal{Q}}(\mathbf{a}) - \mathbf{y}).(\mathbf{a} - \pi_{\mathcal{Q}}(\mathbf{a})) \geq 0, \qquad \forall \mathbf{y} \in Q. \tag{4}$$

From (2)–(4), we have

$$|\mathbf{x} - \mathbf{y}| \geq |\mathbf{a} - \pi_{\mathcal{Q}}(\mathbf{a})|, \qquad \forall \mathbf{x} \in \mathcal{P}, \quad \forall \mathbf{y} \in \mathcal{Q}. \qquad \blacksquare$$

From now on, we shall use the notation $\pi_{\mathcal{PQ}} \equiv \pi_{\mathcal{P}} \circ \pi_{\mathcal{Q}}$. A fixed point of this operator can be found by means of the following algorithm (we give a FORTRAN 90-like pseudocode).

Swap Algorithm (SA)

Step 1: We choose $\mathbf{x}_0 \in \mathcal{P}$, the stopping criterion (EPS)
and $t \in (0, 1)$

Step 2: $\mathbf{x}_{n+1} = t\, \mathbf{x}_n + (1 - t)\, \pi_{\mathcal{P}}\, \mathcal{Q}(\mathbf{x}_n)$

Step 3: **if** $|\mathbf{x}_{n+1} - \mathbf{x}_n| < \text{EPS}$ **then**
go to Step 4
else
$\mathbf{x}_n = \mathbf{x}_{n+1}$
go to Step 2
endif

Step 4: $\mathbf{x}_n$ is an approximation to $\mathbf{a}$ and $\pi_{\mathcal{Q}}(\mathbf{x}_n)$
is an approximation to $\mathbf{b}$ (STOP)

LEMMA 2. *$\pi_{\mathcal{PQ}} : \mathcal{P} \to \mathcal{P}$ is a nonexpansive operator.*

PROOF. The projection operator onto a closed convex set of a Hilbert space is nonexpansive [15], therefore,

$$|\pi_{\mathcal{P}}(\pi_{\mathcal{Q}}(\mathbf{x})) - \pi_{\mathcal{P}}(\pi_{\mathcal{Q}}(\mathbf{x}'))| \leq |\pi_{\mathcal{Q}}(\mathbf{x}) - \pi_{\mathcal{Q}}(\mathbf{x}')| \leq |\mathbf{x} - \mathbf{x}'|$$

for all $\mathbf{x}, \mathbf{x}' \in \mathcal{P}$. $\blacksquare$

THEOREM 1. *The swap algorithm (SA) converges to a point $\mathbf{a} \in \mathcal{P}$ such that $\mathbf{a}$ and $\mathbf{b} = \pi_{\mathcal{Q}}(\mathbf{a})$ verify expression (1).*

PROOF. From the theorem of Browder, Göhde and Kirk [16, pp. 478–479], and Lemma 2, we can conclude that the set of fixed points of the operator $\pi_{\mathcal{PQ}}$ is a nonempty, closed, and convex set.

We can now apply the following result [16, p. 481].

Let $T : M \subset H \to H$ be a nonexpansive operator on a set M in a real Hilbert space H, M being nonempty, convex, bounded, and closed. Then, for a fixed $t \in (0, 1)$, the iterative sequence

$$\mathbf{x}_{n+1} = (1 - t)\, T\mathbf{x}_n + t\, \mathbf{x}_n$$

converges weakly to a fixed point of T.

Finally, we use the fact that in a Hilbert space of finite dimension strong and weak convergence are equivalent [15, p. 36]. Then we apply the second part of Lemma 1. $\blacksquare$

3. A FAST METHOD FOR PROJECTING A POINT ON A CONVEX POLYHEDRON: THE LOCAL SEARCH ALGORITHM BASED ON FACES

The computation of the projection of a point onto a subset of R^n has practical importance in many applications: obstacle avoidance algorithms based on potential functions, pattern recognition, image reconstruction (computerized tomography),....

In addition to the methods that find a nearest pair of points in two sets, the following algorithms for solving this problem have been described in the following literature:

1. methods based on the inequality of Wolfe that use techniques inspired by linear programming [17];
2. other iterative methods [18–20];
3. exhaustive computation of the distance of the point to the faces of the polyhedron (brute force).

To apply SA, we need a fast algorithm for projecting a point $\mathbf{p}$ on a convex polyhedron. In this section, we introduce a local search procedure which can be used. This algorithm only examines the distance of selected faces of the polyhedron to the point $\mathbf{p}$. Local descent along visible faces is used to find the point nearest to $\mathbf{p}$.

3.1. The Local Search Algorithm

In this section, we introduce an algorithm for solving the projection problem. Its convergence will be a consequence of the following fact (Corollary 1): a local minimum of $d(A, \mathbf{b})$ on a polyhedron face visible from $\mathbf{b}$ is, in fact, the global minimum.

To prove this, we need some definitions.

R^n will denote the Euclidean n-space and . the usual dot product. A closed segment in R^3 is defined by

$$[\mathbf{a}, \mathbf{b}] = \{\mathbf{x} \in R^3 / \mathbf{x} = \mathbf{a} + t(\mathbf{b} - \mathbf{a}), \ t \in (0,1)\}.$$

Let $(A, \mathbf{n}_A)$ be an oriented polygon that forms a polyhedral face of $\mathcal{P}$. The unit vector $\mathbf{n}_A$ is supposed to be directed towards the exterior of $\mathcal{P}$. We shall say that the face $(A, \mathbf{n}_A)$ is *visible* from $\mathbf{b}$ if for any $\mathbf{a} \in A$ we have

$$(\mathbf{a} - \mathbf{b}).\mathbf{n}_A \leq 0.$$

Let us denote by $\mathrm{Vis}(\mathbf{b})$ the set of faces of $\mathcal{P}$ which are visible from $\mathbf{b}$.

The following lemma implies that the search of the nearest face to $\mathbf{b}$ need only consider faces visible from $\mathbf{b}$.

LEMMA 3. *If $\mathbf{p}$ is the projection of a point $\mathbf{b}$ (external to $\mathcal{P}$) on $\mathcal{P}$, then $\mathbf{p}$ belongs to a face of $\mathcal{P}$ visible from $\mathbf{b}$.*

PROOF. This result is a particular case of the following lemma.

LEMMA 4. *Let $\mathbf{b}$ be a point external to the convex polyhedron $\mathcal{P}$ and let $\mathbf{a} \in \mathcal{P}$. The intersection point $\mathbf{c}$ of the segment $[\mathbf{a}, \mathbf{b}]$ with the boundary $\partial\mathcal{P}$, which is nearest to $\mathbf{b}$, has the following property.*

There exists at least one face $(A, \mathbf{n}) \in F(\mathcal{P})$, such that

- $\mathbf{c} \in A$,
- $\mathbf{bc}.\mathbf{n} \leq 0$ *(A is visible from $\mathbf{b}$).*

PROOF. We have that

$$\mathcal{P} = \{\mathbf{x} \ / \ \mathbf{x}.n_i \leq c_i, \ i = 1, \ldots, m\},$$
$$\partial\mathcal{P} = \{\mathbf{x} \in \mathcal{P} \ / \exists i \ / \ \mathbf{x}.n_i = c_i\}$$

$\mathbf{x} \in [\mathbf{a}, \mathbf{b}]$ can be expressed as $\mathbf{x} = \mathbf{a} + t(\mathbf{b} - \mathbf{a})$ with $t \in [0, 1]$.

Now consider the following auxiliary functions:

$$h_i(t) = [\mathbf{a} + t(\mathbf{b} - \mathbf{a})].\mathbf{n}_i - c_i, \qquad i = 1, \ldots, m.$$

Since $\mathbf{b}$ is external to $\mathcal{P}$: $h_i(1) > 0$ for $i = 1, \ldots, k$ (where $0 < k < m$, renumbering faces if necessary).

Since $\mathbf{a} \in \mathcal{P}$, we have $h_i(0) \leq 0$ for $i = 1, \ldots, m$.

For $i = 1, \ldots, k$, let $t_i \in [0, 1]$ be a value of t for which $h_i(t_i) = 0$.

Let $t_0 = \min_{i=1,\ldots,k}\{t_i\}$. Then there exists at least one index s $(1 \leq s \leq k)$ such that $h_s(t_0) = 0$ and $h_i(t_0) \leq 0$, $\forall i = 1, \ldots, m$. Therefore, we put

$$\mathbf{c} = \mathbf{a} + t_0(\mathbf{b} - \mathbf{a}) \in \partial\mathcal{P}.$$

Subtracting the formulae

$$h_s(t_0) = \mathbf{c}.\mathbf{n}_s - c_s = 0$$

and

$$h_s(1) = \mathbf{b}.\mathbf{n}_s - c_s > 0,$$

we find $(\mathbf{c} - \mathbf{b}).\mathbf{n}_s < 0$. ∎

Let $F(\mathcal{P})$ denote the set of oriented polygons that form the faces of $F(\mathcal{P})$. If $A \in F(\mathcal{P})$, let $N(A)$ denote the set of faces of $\mathcal{P}$ which have a point or an edge in common with A excluding A itself. The number of elements of $N(A)$ will be called NA.

Local Search Algorithm Based on Faces (LSABF)

Step 1: Choose $A_0 \in F(\mathcal{P})$ such that $A_0 \in \mathrm{Vis}\,(\mathbf{b})$
Compute $d(A_0, \mathbf{b})$.
$A = A_0$

Step 2: **do** $k = 1$, NA $\quad (A_k \in N(A))$
if ($A_k \in \mathrm{Vis}\,(\mathbf{b})$ and $d(A_k, \mathbf{b}) < d(A, \mathbf{b})$) **then**
$A = A_k$
go to the start of Step 2
endif
enddo

Step 3: $d(\mathcal{P}, \mathbf{b}) = d(A, \mathbf{b})$ $\qquad$ (STOP)

Before showing the convergence of this algorithm, we need to derive some technical results.

Let $\mathbf{b}$ be a point external to $\mathcal{P}$ and let $\mathbf{x}$ be a point belonging to $\mathcal{P}$. We shall use the auxiliary function f, defined as follows.

Define $I \equiv [\mathbf{x}, \mathbf{b}] \cap \partial\mathcal{P}$, and let f map $\mathbf{x}$ to the point of I closest to $\mathbf{b}$.

Using Lemma 4, the following properties of f can be verified.

1. f is continuous on $\mathcal{P}$.
2. If $\mathbf{x} \in \mathrm{Vis}\,(\mathbf{b})$, then $f(\mathbf{x}) = \mathbf{x}$.
3. For all $\mathbf{x} \in \mathcal{P}$, we have that $f(\mathbf{x}) \in \mathrm{Vis}\,(\mathbf{b})$.
4. If $\mathbf{x} \in \partial\mathcal{P}$ and $\mathbf{x} \notin \mathrm{Vis}\,(\mathbf{b})$, then $f(\mathbf{x}) \neq \mathbf{x}$.

LEMMA 5. *If* $\mathbf{a}, \mathbf{b} \in R^n$ *and* $|\mathbf{a}| < |\mathbf{b}|$, *then*

$$|\mathbf{a} + t(\mathbf{b} - \mathbf{a})| < |\mathbf{b}|, \qquad \text{for all } t \in [0, 1).$$

PROOF. By the properties of the norm

$$\begin{aligned} |\mathbf{a} + t(\mathbf{b} - \mathbf{a})| &= |\mathbf{a}(1 - t) + \mathbf{b}t| \leq |\mathbf{a}(1 - t)| + |\mathbf{b}t| \\ &= |\mathbf{a}|(1 - t) + |\mathbf{b}|t < |\mathbf{b}|(1 - t) + |\mathbf{b}|t = |\mathbf{b}|. \end{aligned}$$

∎

LEMMA 6. *Let* $\mathbf{b}$ *be a point external to* $\mathcal{P}$ *and let* $A \in \mathrm{Vis}(\mathbf{b})$. *Suppose that there exists* $A' \in F(\mathcal{P})$ *such that* $d(A', \mathbf{b}) < d(A, \mathbf{b})$. *Then there exists* $\tilde{A} \in N(A) \cap \mathrm{Vis}(\mathbf{b})$ *such that* $d(\tilde{A}, \mathbf{b}) < d(A, \mathbf{b})$.

PROOF. Let $\mathbf{a}$ and $\mathbf{a}'$ be the points of A and A', respectively, nearest to $\mathbf{b}$. By hypothesis $d(\mathbf{a}', \mathbf{b}) < d(\mathbf{a}, \mathbf{b})$. Using Lemma 5, we see that for all $x \in (\mathbf{a}, \mathbf{a}']$,

$$d(\mathbf{x}, \mathbf{b}) < d(\mathbf{a}, \mathbf{b}). \tag{5}$$

The continuity of the auxiliary function f, implies that given $\epsilon > 0$ there exists $\delta > 0$ such that

$$f[B_\delta(\mathbf{a}) \cap \mathcal{P}] \subset B_\epsilon(f[\mathbf{a}]) \cap \partial\mathcal{P}.$$

(We denote by $B_\gamma(\mathbf{c})$ the open ball of radius γ centered in $\mathbf{c}$.)

Since $\mathbf{a}$ belongs to a visible face, this second property of f implies that $f(\mathbf{a}) = \mathbf{a}$. Therefore,

$$f[B_\delta(\mathbf{a}) \cap \mathcal{P}] \subset B_\epsilon(\mathbf{a}) \cap \partial\mathcal{P},$$

that is, if we take $\mathbf{x} \in (\mathbf{a}, \mathbf{a}'] \cap (B_\delta(\mathbf{a}) \cap \mathcal{P})$, then $f(\mathbf{x}) \in B_\epsilon(\mathbf{a}) \cap \partial\mathcal{P}$ (arbitrarily close to $\mathbf{a}$). Furthermore, from (5) we have

$$d(f(\mathbf{x}), \mathbf{b}) \leq d(\mathbf{x}, \mathbf{b}) < d(\mathbf{a}, \mathbf{b}).$$

Therefore, there exist points on the boundary $\partial\mathcal{P}$ arbitrarily close to the face A which belong to visible faces ($\tilde{A}$) which are nearer to $\mathbf{b}$ than A. ∎

COROLLARY 1. *Suppose that for all* $\tilde{A} \in N(A) \cap \mathrm{Vis}(\mathbf{b})$, *we have* $d(\tilde{A}, \mathbf{b}) \geq d(A, \mathbf{b})$. *Then for all* $A' \in F(\mathcal{P})$, *we have*

$$d(A', \mathbf{b}) \geq d(A, \mathbf{b}).$$

PROOF. This statement is equivalent to Lemma 6. ∎

The above results imply we can formulate the following convergence theorem.

THEOREM 2. *LSABF yields the global minimum of* $d(A, \mathbf{b})$ *in a finite number of steps.*

PROOF. Corollary 1 gives a stopping criterion. Since the number of faces is finite, and the distance is strictly decreased in every iteration, the algorithm obtains the nearest face in a finite number of steps. Moreover, Lemma 6 shows that we need only consider faces visible from $\mathbf{b}$ in the descent procedure. ∎

3.2. A Criterion for Choice of Initial Face

The following result gives a simple and fast procedure for choice of initial face in LSABF (a face visible from $\mathbf{b}$ and near it).

LEMMA 7. *Let* $\mathcal{P}$ *be a polyhedron and* $V(\mathcal{P})$ *the set of its* N *vertices. If* $\mathbf{c}, \mathbf{b} \in R^n$ $(\mathbf{c} \neq \mathbf{b})$ *and* $\mathbf{v}_0 \in V(\mathcal{P})$ *is such that*

$$\mathbf{v}_i.\mathbf{cb} \leq \mathbf{v}_0.\mathbf{cb}, \tag{6}$$

for all $\mathbf{v}_i \in V(\mathcal{P})$, *and*

$$(\mathbf{c} - \mathbf{b}).(\mathbf{v}_0 - \mathbf{b}) > 0, \tag{7}$$

then there exists a face $A \in F(\mathcal{P})$ *such that*

- $\mathbf{v}_0 \in A$,
- $A \in \mathrm{Vis}(\mathbf{b})$.

PROOF. We shall prove that $[\mathbf{b}, \mathbf{v}_0) \cap \mathcal{P} = \emptyset$. Then we need only apply Lemma 4. Define $\mathbf{d} \equiv (c\mathbf{b}/|c\mathbf{b}|)$ and $k \equiv \mathbf{v}_0.\mathbf{d}$.

Then (6) can be written as $\mathbf{v}_i.\mathbf{d} \leq k$. Define the following half spaces:

$$H^+ = \{\mathbf{x}/\mathbf{x}.\mathbf{d} > k\},$$
$$H^- = \{\mathbf{x}/\mathbf{x}.\mathbf{d} \leq k\}.$$

Obviously, $H^+ \cap H^- = \emptyset$. We have that $\mathcal{P} \subset H^-$, in effect, if $\mathbf{p} \in \mathcal{P}$ then $\mathbf{p} = \sum_{i=1}^{N} \alpha_i \mathbf{v}_i$ (being $\alpha_i \geq 0$, $i = 1, \ldots, N$, and $\sum_{i=1}^{N} \alpha_i = 1$), therefore,

$$\mathbf{p}.\mathbf{d} = \sum_{i=1}^{N} \alpha_i \mathbf{v}_i.\mathbf{d} \leq k \sum_{i=1}^{N} \alpha_i = k.$$

We shall prove now that $[\mathbf{b}, \mathbf{v}_0) \subset H^+$, in effect, from (7) we have that $\mathbf{b}.\mathbf{d} > k$. As

$$[\mathbf{b}, \mathbf{v}_0) = \{\mathbf{x}/\mathbf{x} = \mathbf{b} + (\mathbf{v}_0 - \mathbf{b})t,\ t \in [0, 1)\}.$$

Then, if $\mathbf{x} \in [\mathbf{b}, \mathbf{v}_0)$, we have

$$\mathbf{x}.\mathbf{d} = \mathbf{b}.\mathbf{d}(1 - t) + \mathbf{v}_0.\mathbf{d}t > k(1 - t) + kt = k. \qquad \blacksquare$$

In practice, we take $\mathbf{c}$ as the barycenter or the center of the bounding box of $\mathcal{P}$ [21] the sides of which are parallel to the coordinate axes.

Then, when the point $\mathbf{b}$ is far enough of the polyhedron, inequality (7) is always fulfilled and Lemma 7 guarantees the existence of a visible face with $\mathbf{v}_0$ as vertex.

For some polyhedra, the procedure can fail in some points $\mathbf{b}$ near $\mathcal{P}$. The following algorithm keeps in mind this possibility, performing an exhaustive search when it is necessary.

We denote by NF the number of faces of the polyhedron, and by $\mathbf{v}_{1i}, \mathbf{v}_{2i}, \mathbf{v}_{3i}$ the three first vertices (counter-clockwise) of the face A_i.

Algorithm for Finding an Initial Visible Face

1. Compute $\mathbf{c}$ (barycenter or center of the bounding box)
2. **if** $(\mathbf{b} = \mathbf{c})$ **then**
   $\mathbf{b}$ is inside $\mathcal{P}$: "the projection is b" (STOP)
   endif
3. Compute $m = \min_{\mathbf{v}_i \in V(\mathcal{P})} \mathbf{v}_i.c\mathbf{b}$
   (we call $m_i \equiv \mathbf{v}_i.c\mathbf{b}$)
4. **do** $i = 1, NF$
   if ($m_i = m$ **and** there exist $A \in F(\mathcal{P})$ such that
   $A \in \text{Vis}(\mathbf{b})$ **and** $\mathbf{v}_i \in A$) **then**
   $A_0 = A$
   go to LSABF
   endif
   enddo
5. **do** $i = 1, NF$
   $\mathbf{N}_i = (\mathbf{v}_{2i} - \mathbf{v}_{1i}) \times (\mathbf{v}_{3i} - \mathbf{v}_{2i})$
   if $((\mathbf{v}_{1i} - \mathbf{b}).\mathbf{N}_i \leq 0)$ **then**
   $A_0 = A$
   go to LSABF
   endif
   enddo
   $\mathbf{b}$ is inside $\mathcal{P}$: "the projection is $\mathbf{b}$" (STOP)

4. EXPERIMENTS IN THREE DIMENSIONS

To calculate projections onto different types of polyhedra of increasing degree of complexity, we have used LSABF and have applied SA to calculate the distance between several examples of polyhedra.

LSABF and SA were programmed using Watcom C, and the computer experiments have been realized using a Pentium (133 Mhz) processor.

Experiments are shown in Figures 1–4. They are as follows.

- Tetrahedron $\mathcal{A}$: four vertices and four faces.
- Parallelepiped $\mathcal{B}$: eight vertices and six faces.
- Icosahedron $\mathcal{C}$: 12 vertices and 20 faces.
- Pyramid $\mathcal{D}$: 21 vertices and 21 faces.
- Polyhedron $\mathcal{E}$: 34 vertices and 40 faces.
- Polyhedron $\mathcal{F}$: 72 vertices and 80 faces.
- Polyhedron $\mathcal{G}$: 409 vertices and 409 faces.
- Polyhedron $\mathcal{H}$: 1152 vertices and 1122 faces.

4.1. Algorithm LSABF

- The algorithm needs the following topological data for every polyhedron processed.
  - Number of vertices and faces of $\mathcal{P}$.
  - Number and description of the vertices of every face.
  - Number and description of the faces adjacent to every face.
  - Number and description of the faces intersecting in every vertex.

  It also uses the coordinates of the vertices of the polyhedron.
- Compute time for a given polyhedron and a given point is calculated as the average of 1000 repetitions of the experiment and expressed in milliseconds. The time that appears in Table 1 is averaged from six positions of the point to be projected on a fixed polyhedron.
- We compare LSABF with two standard algorithms: the brute force method (BF), which calculates the distance to the point of every face of the polyhedron and selects the minimum one and a second procedure, a nonrecursive version of the Sekitani-Yamamoto algorithm [22]. Both these procedures were tests on the same examples than LSABF. The results can be seen in Table 1.
- Polyhedra are found in [22].

The table below uses the following.

- N: Number of vertices of the polyhedron.
- NDP: Mean number of distances point-polygon calculated by LSABF.
- TLSABF: CPU time of the LSABF.
- TBF: CPU time of the brute force algorithm.
- TSYNR: CPU time of the nonrecursive Sekitani-Yamamoto algorithm.

Table 1. Projection experiments (time $\times$ 10^{-3} s).

Polyhedron	N	NDP	TLSABF	TBF	TSYNR
$\mathcal{A}$	4	4	0.14	0.07	0.08
$\mathcal{B}$	8	5	0.22	0.14	0.19
$\mathcal{C}$	12	13	0.68	0.55	0.36
$\mathcal{D}$	21	21	1.33	0.94	0.64
$\mathcal{E}$	34	11	0.83	1.31	1.24
$\mathcal{F}$	72	13	1.19	2.83	3.19
$\mathcal{G}$	409	36	4.10	14.71	50.24
$\mathcal{H}$	1152	25	5.45	33.71	-

Figure 1. Tetrahedron $\mathcal{A}$ and Parallelepiped $\mathcal{B}$.

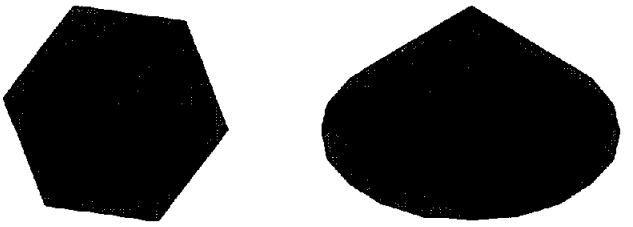

Figure 2. Icosahedron $\mathcal{C}$ and Pyramid $\mathcal{D}$.

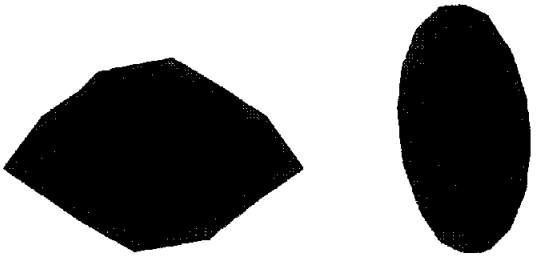

Figure 3. Polyhedron $\mathcal{E}$ and Polyhedron $\mathcal{F}$.

Figure 4. Polyhedron $\mathcal{G}$ and Polyhedron $\mathcal{H}$.

4.2. Algorithm SA

SA (based on LSABF) was applied to several pairs of polyhedra. In these experiments, we have the following.

- We take as initial point (x_0) of the algorithm, the center of the bounding box of $\mathcal{P}$ [21], the sides of which are parallel to the coordinate axes.
- Performance times are calculated for two given polyhedra in a fixed position as the mean of 100 repetitions of every experiment and expressed in milliseconds. Table 2 shows time which is the average of the time obtained considering four different relative positions (and distances) of the two polyhedra.
- We have compared our "swap" algorithm with two alternative techniques. One is the algorithm proposed by Red in [5]. This procedure calculates the distance between all the pairs of faces of both polyhedra and selects the minimum one. The second technique is the nonrecursive version of the Sekitani-Yamamoto algorithm mentioned above. Both cases were tested on the same examples that SA and the results can be seen in Table 2.

The table below uses the following notation.

- $\mathcal{P}$, $\mathcal{Q}$: Polyhedra considered in each experiment.
- N: Sum of the vertices of the polyhedra.
- PREC: $|computed\ distance - exact\ distance|$.
- t: Parameter of the "swap" algorithm.
- NISA: Number of iterations of the "swap" algorithm.
- EPS: Stopping criterion.
- TSA: CPU time of the "swap" algorithm.
- TRED: CPU time of the Red algorithm.
- TSYNR: CPU time of the nonrecursive algorithm of Sekitani-Yamamoto.

In all experiments, we have used the following values of the constants: PREC $< 10^{-6}$, $t = 0$, EPS $= 10^{-6}$.

Table 2. Distance experiments (time $\times$ (10^{-3} s)).

Polyhedra	N	NISA	TSA	TRED	TSYNR
$\mathcal{A},\mathcal{A}$	8	2	0.55	3.27	0.60
$\mathcal{B},\mathcal{B}$	16	2	0.70	10.30	2.75
$\mathcal{C},\mathcal{C}$	24	2	1.93	83.20	6.75
$\mathcal{B},\mathcal{D}$	29	8	8.65	35.98	12.10
$\mathcal{E},\mathcal{E}$	68	2	2.48	434.00	269.00
$\mathcal{F},\mathcal{F}$	144	2	3.55	1876.00	5538.00
$\mathcal{G},\mathcal{G}$	818	2	11.70	54389.25	-
$\mathcal{H},\mathcal{H}$	2304	2	25.65	298892.50	-

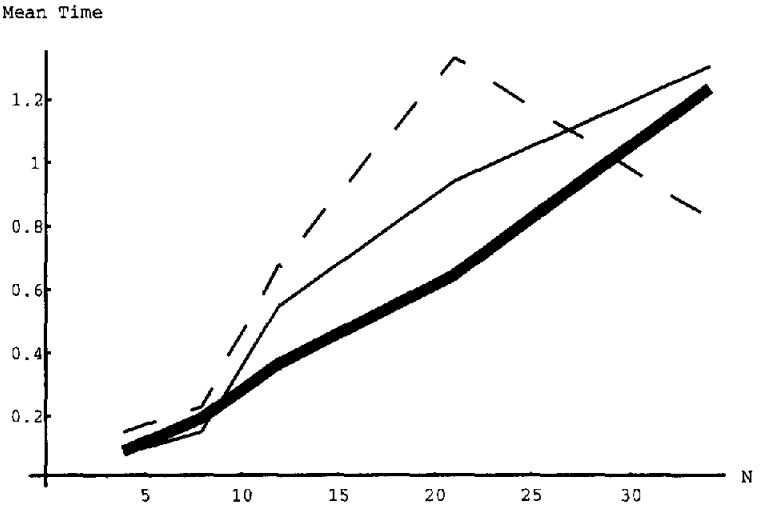

Figure 5. Low complexity.

5. CONCLUDING REMARKS

5.1. Observed Performance of LSABF

The results of Table 1 can be shown as a graph of input complexity versus runtime. Here 'input complexity' is simply the number of vertices of the polyhedron considered. In the figures, the thick line represents the nonrecursive Sekitani-Yamamoto algorithm and the thin line represents the brute force method. The dashed line represents LSABF. Time is measured in milliseconds.

This graph indicates that for low-complexity problems, the nonrecursive Sekitani-Yamamoto algorithm performs best.

In contrast, for problems of mean-high complexity the best performance is attained by LSABF (Figure 6). Its empirically measured time is $O(\sqrt{N})$ while the compute time of a brute force approach is $O(N)$.

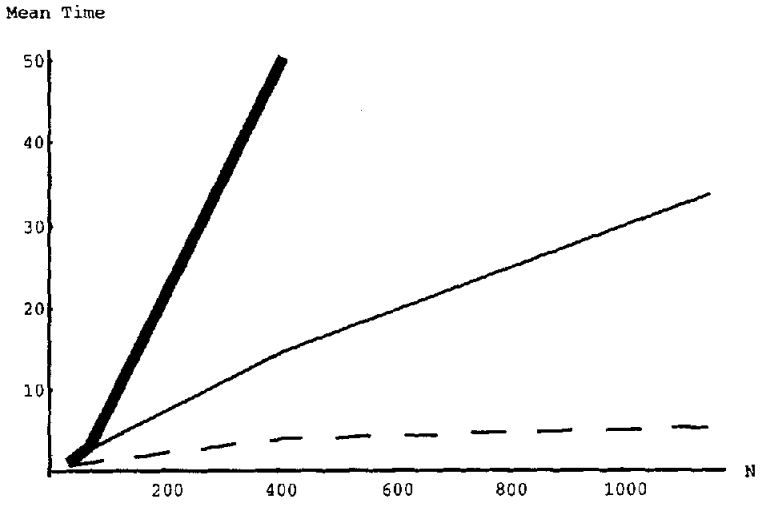

Figure 6. Mean-high complexity.

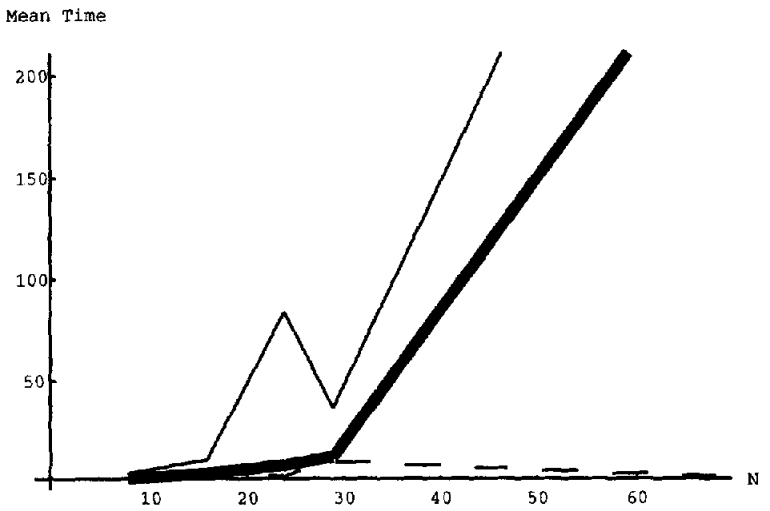

Figure 7. Low complexity.

5.2. Experimental Performance of Algorithm SA

The results of Table 2 can be shown as a graph of complexity versus runtime. The complexity N is expressed as the sum of the number of vertices of both polyhedra. The time is the average of the experiments performed for every pair of polyhedra. The thick line in the figures represents the nonrecursive Sekitani-Yamamoto algorithm and the thin line represents the Red method. The dashed line represents the "swap" algorithm. The unit of time is hundredths of a millisecond. The optimal value found for the parameter t is 0. The number of iterations increases if t increases.

For low-complexity problems, the "swap" algorithm has better performance than the Red and nonrecursive Sekitani-Yamamoto algorithms (Figure 7).

For problems of mean-high complexity (Figure 8), we have found for SA an empirical computational time of $O(N^{\epsilon})$ where $1/2 < \epsilon < 1$. The computational time of the Red method is $O(N^2)$.

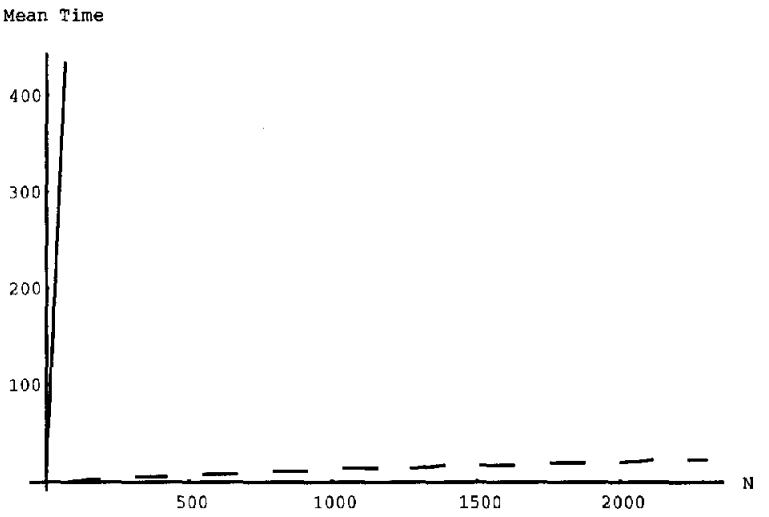

Figure 8. Mean-high complexity.

Indirect comparison with other methods is difficult due to several factors: different computers, different programming languages, and operating systems. On the other hand, the complexity of the polyhedra studied by other authors has generally been smaller than those presented here.

Although no firm conclusion should be drawn from such limited experiments, the algorithms studied exhibit sublinear empirical computational complexity. Some other algorithms [8,12,13,23], with this property have been reported. SA converges very rapidly (two iterations) in most of the examples studied. This opens a way to get faster algorithms if projection methods faster than LSABF can be found.

When deformable polyhedra move, the topological data given at the start do not change.

A drawback of SA is the increase of the number of iterations that can occur when the polyhedra are very near and the points of minimum distance lie in almost parallel faces (Table 2 ($\mathcal{B} + \mathcal{D}$)). This can be avoided by demanding less precision or using a different strategy in applying SA [8].

REFERENCES

1. G.C. Burdea, *Force and Touch Feedback for Virtual Reality*, Wiley, (1996).
2. J.T. Schwartz, Finding the minimum distance between two convex polygons, *Info. Proc. Lett.* **13**, 168–170, (1981).
3. F. Chin and C.A. Wang, Optimal algorithms for the intersection and the minimum distance problems between planar polygons, *IEEE Trans. Comp.* **C-32**, 1203–1207, (1983).
4. D.P. Dobkin and D.G. Kirkpatrick, Determining the separation of preprocessed polyhedra—A unified approach, In *Proc. 17th Internat. Colloq. Automata Lang. Program*, (Edited by M.S. Paterson), pp. 400–413, Springer-Verlag, New York, (1990).
5. W.E. Red, Minimum distances for robot task simulation, *Robotica* **1**, 231–238, (1983).
6. J.E. Bobrow, A direct minimization approach for obtaining the distance between convex polyhedra, *International Journal of Robotics Research* **8**, 65–76, (1989).
7. M. Lin, *Efficient Collision Detection for Animation and Robotics*, Ph.D. Thesis, University of California at Berkeley, (1993).
8. B. Llanas, M. Fernández de Sevilla and V. Feliú, A quasilocal descent-projection method for finding the distance between two convex polyhedra, (submitted).
9. E.G. Gilbert and C.-P. Foo, Computing the distance between general convex objects in three-dimensional space, *IEEE Transactions on Robotics and Automation* **6** (1), 53–61, (1990).
10. E. Gilbert, D.W. Johnson and S.S. Keerthi, A fast procedure for computing the distance between complex objects in three-dimensional space, *I.E.E.E. Journal of Robotics and Automation* **4**, 193–203, (1988).
11. J. Klosowski, M. Held, J. Mitchell, H. Sowizral and K. Zikan, Efficient collision detection using bounding volume hierarchies of k-DOPs, *IEEE Trans. on Visualization and Computer Graphics* **4** (1), 21–36, (1998).
12. S. Fujishige and P. Zhan, A dual algorithm for finding a nearest pair of points in two polytopes, *J. of the Operations Research Society of Japan* **35**, 353–365, (1992).
13. K. Sekitani and Y. Yamamoto, A recursive algorithm for finding the minimum norm point in a polytope and a pair of closest points in two polytopes **61**, 233–249, (1993).

14. P.J. Laurent, Un algorithme dual pour le calcul de la distance entre deux convexes, In *Optimization and Optimal Control*, (Edited by A. Dold and B. Eckmann), pp. 202–228, Springer-Verlag, New York, (1974).
15. H. Brezis, *Analyse Fonctionelle*, Masson, (1983).
16. E. Zeidler, *Nonlinear Functional Analysis and its Applications, Volume 1: Fixed-Point Theorems*, Springer-Verlag, (1986).
17. Ph. Wolfe, Finding the nearest point in a polytope, *Mathematical Programming* **11**, 128–149, (1976).
18. H.H. Bauschke and J.M. Borwein, On projection algorithms for solving convex feasibility problems, *SIAM Review* **38** (3), 367–426, (1996).
19. B. Llanas and C. Moreno, Finding the projection on a polytope: An iterative method, *Computers Math. Applic.* **32** (8), 33–39, (1996).
20. B. He and J. Stoer, Solution of projection problems over polytopes, *Numerische Mathematik* **61** (1), 73–90, (1992).
21. G. Glaeser, *Fast Algorithms for 3D-Graphics*, Springer-Verlag, (1994).
22. B. Llanas and M. Fernández de Sevilla, Una versión no recursiva del algoritmo de Sekitani-Yamamoto para encontrar la distancia entre dos poliedros, *Sistema de Teleoperación basado en Realidad Virtual*, Research Report, Comunidad Autónoma de Madrid, (1996).
23. S. Cameron, A comparison of two fast algorithms for computing the distance between convex polyhedra, *IEEE Transactions on Robotics and Automation* **4**, 915–920, (1997).