Lower Bound of Distance in 3D

by

Petr Konečný
Karel Zikan

FI MU Report Series FIMU-RS-97-01

January 1997

Lower Bound of Distance in 3D

Karel Zikan[*] Petr Konečný[†]

Abstract

The term "collision detection" refers to the task of determining whether, in a given set of objects, any two intersect. If they do, then common collision detection systems return either one such pair or all such pairs. The term "proximity computation" refers to a more general task where we determine the nearest or the "most overlapping" pairs of objects. In the article, we present a new method to rapidly compute lower bounds of distances. The lower bound decreases the complexity of collision detection (or proximity computation) by computing "candidates" of collision, i.e., pairs of objects that might intersect. It estimates the lower bound of their distance and rejects pairs that are too far from each other to collide.

1 Collision detection

Detection of collisions between virtual objects is one of the basic tasks in virtual reality type of applications. For instance, a program that maintains a virtual body (*"avatar"*) in a virtual environment often depends critically on knowing when and where the avatar collides with the environment. Similarly, a modern CAD system must be able to check if the given model satisfies engineering constraints, such as the required separation of the hydraulic systems of an aircraft from the aircraft's moving parts. Computations of lower bounds of distances between objects are used to ensure separations and to detect or eliminate collisions.

Clearly, a trivial method of finding a distance bound between two objects is to compute the exact distances between all pairs of relevant (convex) subregions; then the minimum of these distances is the best lower bound possible. However, such a brute force approach is too time consuming to

[*]Fulbright Visiting Professor (on leave from The Boeing Company), Masaryk University, Botanická 68a, 602 00 Brno, Czech Republic, *zikan@informatics.muni.cz*

[†]Student, Masaryk University, *pekon@informatics.muni.cz*

be practical in complex scenes. Even the computation of one exact distance between two *convex* objects can be relatively expensive; worse yet, too many such computations usually have to be performed. For instance, a jumbo jet like, say, the Boeing 777 consists of approximately five million generally non-convex parts (flying in a formation). Thus, the aforementioned virtual reality CAD system would indeed be ill advised to boldly charge forth by brute force. The CAD system would have to compute some twenty five trillion separations between pairs of generally non-convex parts.

1.1 Methods

Practically efficient collision detection methods rely on their ability to wholesale reject unpromising candidates. One way or another, these methods quickly find *one* implicit lower bound for a *large number* of distances and, if this lower bound is large enough, they discard these distances from further consideration.

Many auxiliary notions have been used by the various collision detection methods. They generally fall into one of the following three categories: purely geometrical, such as bounding boxes; purely data-structural, such as binary trees; and tight coupling of both, such as triangulations of the free space, or octrees in some form or another.

This paper is an outgrowth of and tries to improve upon the currently fastest known collision detection method, that is being put forth in [St-96]. The method, which was also presented at the recent SIGGRAPH'96 conference in New Orleans, uses hierarchical trees of bounding geometric (and algebraic) primitives. In the present paper, the primitives are called *fixed-directions hulls* (FDH).

Earlier, Kajiya and Kay [Ka-86] successfully used essentially the same primitives to compute radiosity. From a mathematical view, FDHs are points of a sublattice. As [Sa-89a, Sa-89b] attest, spatial hierarchies of bounding volumes have perhaps even longer and more distinguished history. In collision detection, spatial hierarchies have been used most successfully in the combination with bounding boxes (BB), especially with axes' aligned bounding boxes. The basic references include [Ba-96, Be-90, Za-94]. We also give [Ab-96] as an example of a highly successful industrial implementation of this collision detection paradigm—at least, when the environment is static and objects do not move against each other. Octrees were used in [Mo-88, No-89], sphere-trees in [Hu-95].

The above mentioned SIGGRAPH'96 published yet another paper [Go-96] with a very promising collision detection performance. The method uses successfully hierarchies of BBs in dynamic environments. The method does not enforce an alignment of the boxes: rather then realigning boxes after each transformation, the method looks for separating hyper-planes, choosing from a relatively small set of candidates. A simple but clever lemma states that if there is a hyper-plane that separates two boxes, then there is also one to be found in this limited set. Because the boxes need not be aligned, the method can use such orientation of each box that fits best. Unfortunately, there are also two weaknesses of the method: First, the intersect/no-intersect nature of the answer is often inadequate. Most virtual reality applications ask for more accurate separation information. Second, the limited set of separating planes grows too fast when polytopes other then the bounding boxes are used.

The [St-96] article should be consulted for an in-depth description of the overall method we use, for the analysis of many of the available design choices, and for comparisons of the method with other outstanding collision detection approaches. Here we present only a new, fast way to compute a lower bound of the distance between two differently oriented FDH. Our approach is based on a rarely used linear programming technique (So rare that even George Dantzig, the father of linear programming, claims to never actually encounter it.). The problem of computing the lower bound warrants an extraordinary attention. A routine that solves it efficiently can be used in the critical inner loop of the collision detection method. In turn, the inner query needs to be answered with utmost speed, several thousand times a second, if we are to achieve interactive collision detection speeds in large virtual reality environments.

1.2 Bounding volumes

We develop a sufficient intuitive understanding of the issues if we first consider the collision detection method with aligned bounding boxes (BBs) instead of higher level FDHs. Before computing any exact distance between the virtual objects, the program checks if their BBs are sufficiently separated. Only when this separation is deemed low, the program either checks separation of several pairs of smaller BBs or, eventually, the separation of the "real" objects.

Since computations with BBs are, in general, easy and fast, the only obstacle to excellent collision detection performance comes from the fact that BBs often provide pure-to-middling approximations of 3D objects: they add wasteful "empty corners" to the object.

We can generalize the former method by using some other convex bounding volumes (BV) that would enclose 3D objects more tightly. But when BV's get more complex the computation of distances becomes increasingly hard. An extremal example of a tight convex BV fit is the convex hull. But although there are reasonable algorithms for creating convex hulls, the problem of estimating distance between the hulls typically leads to large problems of linear or quadratic programming, with insufficient special structure to take advantage of. It is thus difficult to achieve the algorithmic efficiency that we need. (Not to mention the data structure and memory explosion problems that come up.) In this article, we deal with the special case of convex hulls called fixed directions hulls. The term "fixed directions" means that the surface of the hull consists of hyper-planes with normals from some fixed set of vectors. At one end of the spectra, BBs are an example of fixed direction hulls in 3D with the fixed set of 6 normals, $\{\pm e_1, \pm e_2, \pm e_3\}$. At the other end of the spectra, convex hulls are the "ultimate" (limiting) case of fixed direction hulls, where the fixed set of normals forms the whole surface of the unit sphere.

2 Fixed direction hulls

Let $D = \{\pm d_1, \pm d_2, \dots, \pm d_n\}$ be a fixed finite set of directions[1] in $\mathbb{R}^k$, so that $0 \notin D$ and all the directions are normalized $\|d\| = 1$. Given a set $X \subseteq \mathbb{R}^k$ and a direction $d \in D$, we define

$$c_d(X) = \begin{cases} -\infty & \text{if } X = \emptyset, \\ \sup\{d^\top x | x \in X\} & \text{otherwise.} \end{cases}$$

The value c_d becomes $+\infty$ when the set X is unbounded in the direction d. Points that satisfy the equation

$$d^\top x = c_d(X)$$

form the *supporting hyper-plane* of X in the direction d. Points that satisfy the inequality

$$d^\top x \leq c_d(X)$$

comprise the smallest closed half-space containing X with respect to the normal d.

The *collection* of all these half-spaces is the (D-induced) fixed direction hull FDH(X). The *intersection* of all of these half-spaces contains X, however, it is not the FDH(X). This is a subtle but important point. Most of the time, we can use these concepts interchangeably, but such expediency can cause the mathematics behind the computations to break down.

[1] All vectors in the article are supposed to be column vectors.

A good 2D example with which to illustrate the these concepts is the set of normals

$$D_8 = \{\pm(1,0)^\top, \pm(0,1)^\top, \pm(1,1)^\top/\sqrt{2}, \pm(1,-1)^\top/\sqrt{2}\}.$$

Consider the two dimensional set X defined by the three inequalities

$$0 \le x_1,\ 0 \le x_2 \text{ and } x_1 + x_2 \le 1.$$

The (D_8-induced) FDH(X) is the *set* of eight inequalities[2]

$$\begin{array}{cc} 0 \le x_1 \le 1 & 0 \le x_2 \le 1 \\ 0 \le x_1 + x_2 \le 1 & -1 \le x_1 - x_2 \le 1 \end{array}$$

Supporting hyper-planes of this FDH together with the set X are shown on Figure 1(a). The intersection of the hyper-planes of FDH(X) give us X back. The five supporting hyper-planes $x_1 \le 1$, $x_2 \le 1$, $0 \le x_1 + x_2$, and $-1 \le x_1 - x_2 \le 1$ are, in the language of linear programming, *redundant*. Yet, it would be a mistake to remove them from consideration when working with this FDH. Both, the efficiency and the accuracy of several computations depend on the presence of these seemingly "redundant" equations. On the other hand, the intersection of the eight inequalities

$$\begin{array}{cc} 0 \le x_1 \le 2 & 0 \le x_2 \le 2 \\ -2 \le x_1 + x_2 \le 1 & -2 \le x_1 - x_2 \le 2 \end{array}$$

also specifies the very same 2D region X, but this collection of hyper-planes does *not* form an FDH. The difference is in that the five redundant hyper-planes have been moved "outward" and they are no longer in supporting positions. The definition of FDH precludes such situations.

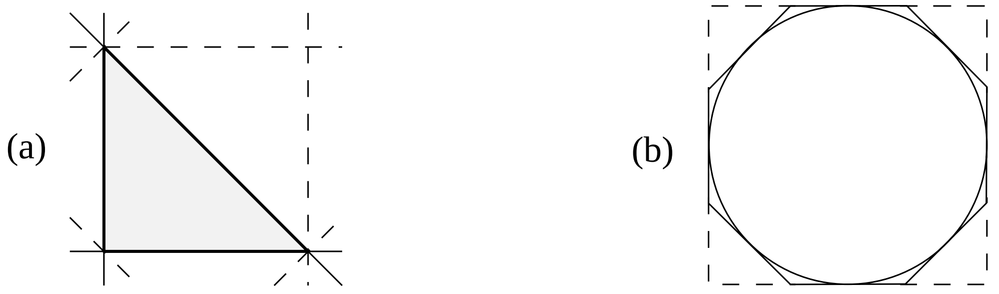

Figure 1: D_8 induced hulls

[2]For a better readability, these inequalities are presented in this more readable form where the directions are not normalized.

2.1 Vector representation

We can fix the order of directions and use them as rows of a $2n \times k$ matrix M. The corresponding ordering of the values $c_i = c_{d_i}$ gives us the vector $c = (c_i) \in \mathbb{R}^{2n}$. The matrix equation

$$Mx \leq c$$

describes succinctly the intersection of the collection of half-spaces $\{x | d_i^\top x \leq c_i\}$. The matrix M is fixed and facilitates the translations from geometry to algebra and back. The distinctions between the individual FDH are encoded into the vectors c and, in fact, many computations with FDH are performed solely with c's, without regard to M. One such operation is *join*, that is, the point-wise maximum of two vectors

$$c = c^1 \vee c^2 = (\max\{c_i^1, c_i^2\}).$$

Another such operation is *meet*, the point-wise minimum of the two vectors

$$c = c^1 \wedge c^2 = (\min\{c_i^1, c_i^2\}).$$

The join is related to the geometric union operation: "The join of hulls is the smallest hull that contains them." The meet is related to the geometric intersection operation: "When their intersection is not empty, hulls intersect in their meet." The inclusion properties with respect to the original sets are also preserved under the intersection/meet and union/join correspondences. Even a more accurate theorem can be proved, but there is no need to digress.

Whereas each vector $c \in \mathbb{R}^{2n}$ corresponds in the obvious one-to-one, onto fashion with a set of half-spaces $\{d_i^\top x \leq c_i\}$, we have already seen an example where the set of half-spaces did not form an FDH. Therefore the set $\mathcal{H}$ of all D-induced FDH forms a proper subset of $\mathbb{R}^{2n}$. One suitable definition of $\mathcal{H}$ is to build it constructively in two steps: First, we put into $\mathcal{H}$ all transforms of $\mathbb{R}^k$ by the matrix M; consequently

$$\{c = Mx | x \in \mathbb{R}^k\} \subset \mathcal{H}.$$

Second, we close $\mathcal{H}$ under the meet and join operations; whenever c^1 and c^2 are members of $\mathcal{H}$, so are $c^1 \wedge c^2$ and $c^1 \vee c^2$.

In the first step, $\mathcal{H}$ was made a k-dimensional linear manifold, embedded in $\mathbb{R}^{2n}$. In the second step, $\mathcal{H}$ was enlarged into the smallest sublattice containing the manifold.

Remark: A subtle, yet intrinsically important point is that FDH algebraically exists even when the intersection of its half-spaces is empty. At this level, however, we shall concern ourselves primarily with FDH geometric existence.

Remark: In order to preserve the tight lower bounds for the entire given class of D-hulls, it is important to algebraically keep around even the supporting redundant hyper-planes (if there are redundant hyper-planes), even though these do not influence the geometric shape of the FDH [ZC-93] This is another subtle point that can be easily missed in a simplified geometric discussion.

2.2 Distance of hulls

This paper employs FD-hulls for computing lower bounds of distances of objects. We use a quick method for bounding the euclidean separation of the hulls to bound also the distance between the objects. The method is based on a (fixed-directions) distance functional that closely approximates and bounds below the usual euclidean metric. Since set-separation is the natural context for our setup, we define the notion in set-separation terms.

Let X and Y be two convex subsets of $\mathbb{R}^k$, and let $d \in \mathbb{R}^k$, $\|d\| = 1$ be a fixed direction. Define *(oriented) distance from X to Y in the direction d* by

$$\delta_d(X,Y) = \inf_{x \in X, y \in Y} d^\top (y - x) = \inf_{y \in Y} d^\top y - \sup_{x \in X} d^\top x.$$

If we consider the projections of X and Y onto the line $\{\theta \cdot d | \theta \in \mathbb{R}\}$, then $\delta_d(X,Y)$ measures whether (and by how much) the projection of Y is completely "beyond" the projection of X. Intuitively, the functional $\delta_d(X,Y)$ is zero exactly when the supremum of the projection of X and the infimum of the projection of Y agree. This is the tightest case when Y is beyond X with respect to the direction d. If $\delta_d(X,Y)$ is positive, then Y is $\delta_d(X,Y)$ units beyond X. On the other hand, if $\delta_d(X,Y)$ is negative, then Y would have to be translated by $-\delta_d(X,Y)$ units in the direction d to get beyond X.

Of course, $\delta_d(X,Y) = \delta_{-d}(Y,X)$. Therefore, if the maximum of $\delta_d(X,Y)$ and $\delta_{-d}(X,Y)$ is positive, then it is the euclidean distance between the projections of X and Y. Moreover, it is also a lower bound of the euclidean separation between the original sets X and Y.

We define the *(fixed-directions) distance between X and Y* by

$$\delta(X,Y) = \max_{d \in D} \delta_d(X,Y).$$

Of course, this distance functional is defined with respect to the usual set of fixed directions D. Since each $\delta_d(X,Y)$ is a lower bound of the euclidean separation between X and Y, functional $\delta(X,Y)$ also bounds this separation. Since X and Y are convex, and the directions of D are spatially well chosen, the bound is quite accurate.

When $X = \{x\}$ and $Y = \{y\}$ are singleton sets, we get the previously mentioned *fixed-directions distance* between the points x and y. The distance is $\delta(x, y) = \delta(X, Y)$, and when $y = 0$, the fixed-direction distance functional $\delta(x, 0)$ becomes a norm on x. Figure 1(b) illustrates the concept by simultaneously showing the unit balls of the euclidean norm and of the fixed-direction norm of D_8 and D_4.

Assume that $c^1, c^2 \in \mathcal{H}$ and that the corresponding FDH regions X^1, and X^2 of $\mathbb{R}^k$ are not empty. The formula for computing $\delta(X^1, X^2)$ is simple. Recall that we insisted that D is composed of pairs of directions, $\pm d$; thus we can pair up these directions—and permute indices, if necessary—so that $d_i = -d_{i'}$ for $i = 1, 2, \ldots, n$. We thus have n pairs of inequalities $d_i^\top x \leq c_i$ and $-d_i^\top x \leq c_{i'}$, alas, in a natural shorthand,

$$-c_{i'} \leq d_i^\top x \leq c_i.$$

We compute $\delta(X^1, X^2)$ using

$$\delta(X^1, X^2) = \max_{1 \leq i \leq n} \{-c^1_{i'} - c^2_i, -c^2_{i'} - c^1_i\}.$$

When $\delta(X^1, X^2) = 0$, then the regions touch. When $\delta(X^1, X^2) \geq 0$, then the region are separated by at least $\delta(X^1, X^2)$ units. When $\delta(X^1, X^2) \leq 0$, then the region intersect and no translation of less then $-\delta(X^1, X^2)$ units dislodges them. However, there is a translation of exactly $-\delta(X^1, X^2)$ units which dislodges the regions into a touching position.

2.3 Linear transformation of the hull

In many applications, our collision detection method must cope with transformations of objects. This means that, together with scaling, moving, and rotating those objects, we must also transform their hulls properly. We need to compute hulls of the transformed objects. To deal with scaling and translating an FDH is quite simple. It is sufficient to apply an independent linear transformation to each of c_d. Rotation, however, is not as simple to handle.

There are three possible solutions:

1. compute the D-hull of the transformed object directly,
2. compute distance bounds using hulls with differently oriented sets D,
3. compute the D-hull of the transformed hull.

The first alternative can obviously be too expensive. The process of computing FDH of some complex object, defined for example by triangle meshes, can be arbitrarily time consuming. Only, if the rotations are rare, we may want to use this first approach. The resulting hull is then the closest approximation to the transformed object and the time spent recomputing the tightest FDH may pay off in time savings elsewhere in the algorithm.

The second alternative still remains to be fully explored. There are several promising methods of computing lower bounds on distances using linear or quadratic programming [ZC-93], and even some other approaches.

In this article, we concern ourselves with how to best execute the third alternative. We think of the hull as of an object in $\mathbb{R}^k$ and we simply compute the hull of it. Obviously, the new hull is also a bounding volume of the transformed original object, albeit not the tightest possible. Therefore the distance of the transformed objects is greater or equal to the distance of their new hulls. Thus we still have a lower bound.

Let $H = \{x \in \mathbb{R}^k | Mx \leq c\}$ and consider the affine transformation of $\mathbb{R}^k$

$$f : x \mapsto Ux + t,$$

where U is an invertible matrix, and $x, t \in \mathbb{R}^k$. Transformation f applied to H gives

$$\begin{aligned} f(H) &= \{y | x \in \mathbb{R}^k, y = Ux + t, Mx \leq c\} \\ &= \{y \in \mathbb{R}^k | MU^{-1}(y - t) \leq c\} \end{aligned}$$

The FDH of this object is defined by the vector $\hat{c} = (\hat{c}_d)$, where

$$\hat{c}_d = \max\{d^\top x | MU^{-1}(x - t) \leq c\},$$

which can be simplified to

$$\begin{aligned} \hat{c}_d &= \max\{d^\top t + d^\top (x - t)| \\ &\qquad M(U^{-1}(x - t)) \leq c\} \\ \hat{c}_d &= d^\top t + \max\{(d^\top U)x | Mx \leq c\}. \end{aligned} \tag{1}$$

We can find $\hat{c}_d$ by solving this linear program. To compute the FDH, we find value of $\hat{c}_d$ for all $d \in D$.

3 Special case - FDH$_{14}$

In Figure 1, we have seen that to get good approximation of an object, we can "cut off the corners" of its bounding box. We can use the same principle to make FDH of 3D objects. Again, we start with the object's BB and then we cut off the BB's corners. To this, we need altogether 14 normals of the cutting planes—6 normals correspond to BB sides; 8 normals correspond to BB corners. Let us set up the matrices

$$A = \begin{pmatrix} 1 & 0 & 0 & \nu & -\nu & \nu & \nu \\ 0 & 1 & 0 & \nu & \nu & -\nu & \nu \\ 0 & 0 & 1 & \nu & \nu & \nu & -\nu \end{pmatrix}^\top,$$

where $\nu = 1/\sqrt{3}$, and

$$M = \begin{pmatrix} A \\ -A \end{pmatrix}.$$

Then we can write every hull $H \in \mathcal{H}$ in the form

$$\begin{aligned} H &= \{x \in \mathbb{R}^3 | Mx \leq b\} \\ &= \{x \in \mathbb{R}^3 | \ell \leq Ax \leq u\}, \end{aligned}$$

where $b = \left(\begin{smallmatrix} u \\ -\ell \end{smallmatrix}\right)$. The vectors ℓ and u satisfy $\ell \leq u$ whenever H is not empty—the only cases that matter here.

Examples of a FDH$_{14}$ are shown on Figure 2. Among other features, the examples also show that the topology of the graph of edges of FDH may change with ℓ and u.

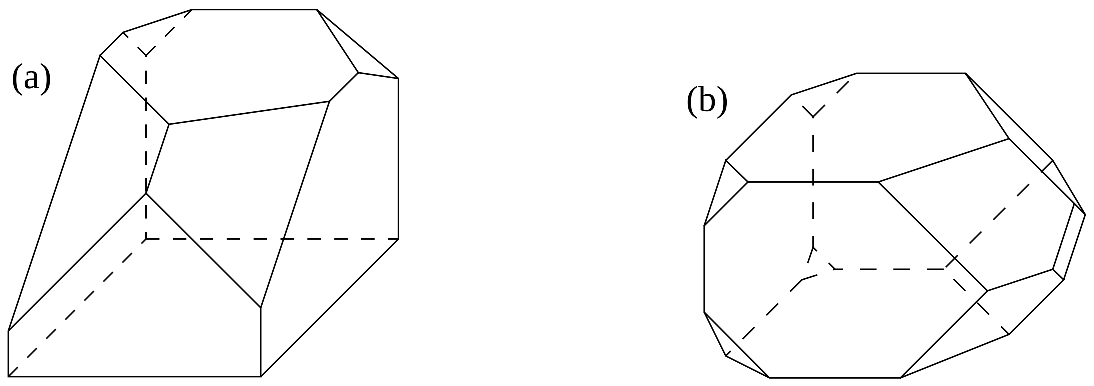

Figure 2: Three dimensional FDH

We will now present an algorithm for computing FDH$_{14}$ of a rigidly transformed FDH$_{14}$. The algorithm is based on (1) and uses *implicit dual-feasible pivots* for solving the linear program. We can identify the following tasks:

1. change w and the constraints of LP to a canonical form,
2. translate hull to the origin of coordinate system,
3. discard the irrelevant constraints, and
4. find the maximal point.

3.1 Canonical form

In the following, we suppose that we have the constant $\tau = d^\top t$ and the vector $w = d^\top U$ from (1). We show that we can transform the optimization problem into an equivalent form where all the coordinates of the vector

w are nonnegative and non-decreasing. The salient point is that, while we transform the equations accordingly, we never actually change the constraint matrix of the problem. Because of all the geometric symmetries of the matrix, we end up with the same matrix as we started from.

Let s be the vector of "signs" of components of the vector w ($|s_i| = 1$, $w_i = s_i|w_i|$ for $i = 1, 2, 3$) and let p be the permutation of indices $1, 2$ and 3 such that

$$|w_{p_1}| \leq |w_{p_2}| \leq |w_{p_3}|.$$

Then set up $\bar{w} = (|w_{p_1}|, |w_{p_2}|, |w_{p_3}|)$. Suppose that S is the diagonal matrix with the values of s on the diagonal and that P is the matrix representing the permutation p. Then we have $P^{-1} = P^\top$, $S = S^{-1} = S^\top$ and $\bar{w} = PSw$; thus

$$w^\top = (S^{-1}P^{-1}\bar{w})^\top = \bar{w}^\top PS.$$

Let $c = \hat{c}_d - \tau$, then we get a new optimization problem

$$\begin{aligned} c &= \max\{w^\top x | Mx \leq b\} \\ &= \max\{\bar{w}^\top PSx | Mx \leq b\} \\ &= \max\{\bar{w}^\top y | MPSy \leq b\}. \end{aligned}$$

From algebraic symmetries of matrix M, we know that there exists permutation matrix Q such that $MPS = QM$. Thus

$$c = \max\{\bar{w}^\top x | QMx \leq b\} = \max\{\bar{w}^\top x | Mx \leq Q^\top b\}.$$

The first task of our algorithm transforms the linear program into this canonical form. The transformation can be encoded into a table of permutations indexed by the vectors s and the permutations p. There are eight possible sign patterns of w and six available permutations of its coordinates. Therefore there is exactly forty-eight permutation patterns Q to keep, one for each pair (s, p). As one would expect, it is also possible to efficiently encode all forty-eight permutations patterns in terms of their s and p pattern "components"; only fourteen, instead of forty-eight, patterns thus need be kept.

For the the second task, we let $w = \bar{w}$ and $o = (b_1, b_2, b_3)^\top$ then

$$\begin{aligned} c_d &= \max\{w^\top x | Mx \leq b\} \\ &= \max\{w^\top (x + o) | M(x + o) \leq b\} \\ &= w^\top o + \max\{w^\top x | Mx \leq \hat{b}\} \end{aligned}$$

where $\hat{b} = b - Mo$. It ought to be transparent that $\hat{b}_1 = \hat{b}_2 = \hat{b}_3 = 0$, and that we have achieved the translation of the problem.

3.2 Implicit pivoting

We simultaneously dispose of the third and fourth task of the list, when we perform implicitly dual-feasible simplex pivots until we arrive to the optimal solution.

The most remarkable feature of our linear programming formulation is that *all* of our problems have the same constraint matrix! We have further increased the commonality between the problems by bringing all of them into the canonical form. Not only the constraint matrix is the same, but we now also have $0 \le w_1 \le w_2 \le w_3$ and $\hat{b}_1 = \hat{b}_2 = \hat{b}_3 = 0$.

Consider a linear program in this form. Those versed in linear programming will be able to verify for themselves the correctness of the next steps.

First, we see that $x^{(0)} = (0,0,0)^\top$, the upper corner of the constraint box ($\hat{\ell}_i \le x_i \le 0, i = 1,2,3$), is a super-optimal point of our linear program. Therefore we can use $x^{(0)}$ as the starting point of the simplex algorithm.

The point $x^{(0)}$ is either primal-feasible and thus optimal, or a sequence of dual-feasible pivots will produce such a point. In the former (degenerate) case, the optimal value is $w^\top o$. In the latter case, the value needs to be decreased through pivoting. In the first pivot, the violated constraint $x_1 + x_2 + x_3 \le \sqrt{3}\hat{b}_4$ is brought into the base and, because of the condition $0 \le w_1 \le w_2 \le w_3$, the constraint $x_1 \le 0$ drops out through the min ratio test.

After the first pivot, we thus reach the point

$$x^{(1)} = (\sqrt{3}\hat{b}_4, 0, 0)^\top.$$

Again, the point $x^{(1)}$ is super-optimal and if this point is also primal-feasible, then it produces the optimal value of the problem (see Figure 3(a)). If $x^{(1)}$ is not primal-feasible, we need to perform one more pivot to further reduce the objective value. The second pivot brings into the base the constraint $-x_1 + x_2 + x_3 \le \sqrt{3}\hat{b}_5$ and drops from the base the constraint $x_2 \le 0$. The corresponding point is

$$x^{(2)} = (\frac{\sqrt{3}}{2}(\hat{b}_4 - \hat{b}_5), \frac{\sqrt{3}}{2}(\hat{b}_4 + \hat{b}_5), 0)^\top.$$

Because of the way each 14-hull is formed by cutting off the corners of its bounding box by its (topologically dual) octahedral planes, one can show that either $x^{(1)}$ or $x^{(2)}$ is a primal-feasible point (see Figure 3(b)). This is a special case of a more general theorem which asserts a similar property for all 3-dimensional FDH built using the normals of dual pairs of polytopes. We eschew the proof of the theorem because of tedious exposition.

As no more than these two pivots are ever needed, we can discard all the superfluous data, i.e., the data not involved in the pivoting. Since we

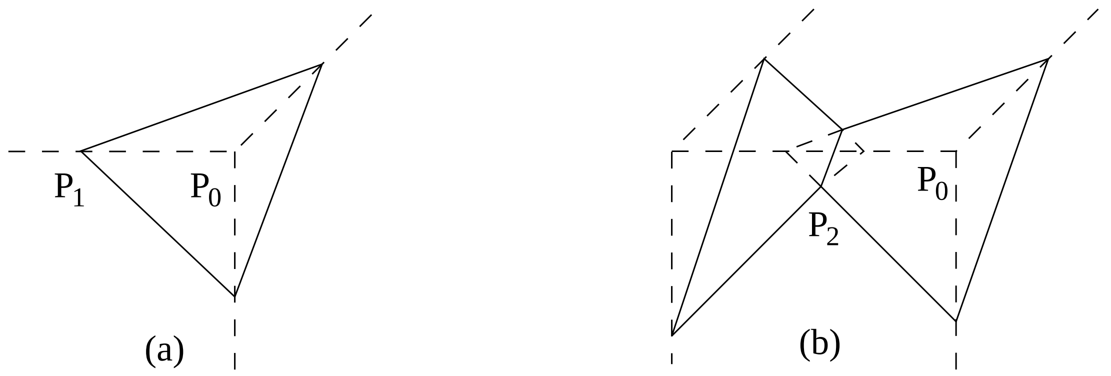

Figure 3: Pivoting points

have also already precomputed the pivots, we can now also drop all of the auxiliary constraint matrix structure.

The initial point, $x^{(0)}$, is optimal only in the degenerate case. Because $x_1 \leq 0$, $x_2 \leq 0$, and $x_3 \leq 0$, we see that $\hat{b}_4 \leq 0$. Thus if $\hat{b}_4 \geq 0$ then $\hat{b}_4 = 0$ and $x^{(0)} = x^{(1)}$. Consequently, we actually need not examine the optimality at $x^{(0)}$, because the initial point can be optimal only when $x^{(1)}$ is also optimal. This observation and a grouping by variables yields a simple closed-form formula

$$c = w^\top o + \begin{cases} \sqrt{3} w_1 \hat{b}_4 & \text{if } \hat{b}_4 + \hat{b}_5 \geq 0, \\ r_1 + r_2 & \text{otherwise,} \end{cases}$$

where $r_1 = \hat{b}_4 \frac{\sqrt{3}}{2}(w_1 + w_2)$ and $r_2 = \hat{b}_5 \frac{\sqrt{3}}{2}(w_2 - w_1)$.

The translation of the hull into the origin of the coordinate system, although useful for exposition, somewhat hinders the performance of the algorithm. Without the translation (i.e., with the original values of b instead of $\hat{b}$), we get the next formula for computing $\hat{c}_d$

$$\hat{c}_d = d^\top t + \begin{cases} v^1 s^1 & \text{if } \frac{\sqrt{3}}{2}(b_4 + b_5) \geq b_2 + b_3, \\ v^2 s^2 & \text{otherwise.} \end{cases}$$

In this formula the vectors v^1, v^2 are defined as follows:

$$v^1 = (w_2 - w_1, w_3 - w_1, \frac{w_1}{\sqrt{3}})$$

$$v^2 = (w_3 - w_2, \frac{\sqrt{3}}{2}(w_1 + w_2), \frac{\sqrt{3}}{2}(w_2 - w_1))$$

and vectors s^1, s^2 depend solely on the original FDH:

$$s^1 = (b_2, b_3, b_4)^\top \quad s^2 = (b_3, b_4, b_5)^\top.$$

Repeated use of this formula gives us all the parameters of the “outer” FDH and, in extension, the desired lower bound of the distance. The vectors v^1 and v^2 are functions of the rotation U only; they are common to all FDH that were realigned by this U. A good speedup is thus achieved for sets of commonly transformed hulls by precomputing v^1, v^2 and $d^\top t$ (together with the permutations Q) for each $d \in D$.

4 Summary

We have shown a method for computing accurate lower bounds of distance between 3D objects using a special form of bounding volumes—fixed directions hulls (FDH). In Section 2, we have developed those aspects of the technique that are common to all FDH types. In Section 3, we give a detailed analysis of the FDH_{14} case because sets of FDH_{14} generally give concise close approximations of 3D objects. The resulting formulas can be used to increase the speed of collision detection methods that work with bounding volumes. In particular, the formulas can be (and have been) used as the base for an algorithm using FDH_{14} within complex bounding hierarchies of virtual objects and virtual environments.

This report pertains to a work in progress. Our current implementation of the method with FDH_{14} on SGI Indigo^2 achieves approximately 2000 proximity tests a second in an environment consisting of two complex, mutually moving objects of 200 and 20000 triangles, respectively.

The slightly slower speed of the distance query—when FDH_{14} instead of the more common axes' aligned bounding boxes is used—seems to be amply compensated by the gains in fit and accuracy that FDH_{14} provides. The much closer geometric approximations and more accurate distance bounds result in improved tree-pruning properties.

Acknowledgements

We thank many of our colleagues for their help, encouragement, and insights. J. Sochor and his team of Masaryk University students built the collision detection system that incorporates our query. H. Sowizral of Sun and the Stonybrook University group of M. Held (also of Salzburg U.), J. Klosowski, and J. Mitchell have coinvented the general approach based on FDH. Much of the insight, encouragement, and helpful discussion came from W. McNeely and J. Heisserman of Boeing. And, of course, the lion's share of credit for inventing and developing the art and techniques of linear programming goes to G.B. Dantzig, Professor Emeritus of Stanford University.

References

[Ab-96] B. Abarbanel, W. McNeely, and E. Brechner. FlyThru the Boeing 777. SIGGRAPH'96 Visual Proceedings; also FlyThru User Guide (M. Silverblatt ed.) D6-56403-600 Boeing document, http://superfly.rt.cs.boeing.com/FlyEtc/FlyEtc.HomePage, 1996.

[ZC-93] K. Zikan and D. Curtis. Intersections and separations of polytopes (A note on collision and interference detection). Boeing Technical Report BCSTECH-93-031, 1993.

[Sa-89a] H. Samet. The Design and Analysis of Spatial Data Structures. Addison-Wesley, 1989.

[Sa-89b] H. Samet. The Applications of Spatial Data Structures. Addison-Wesley, 1989.

[Ka-86] T.L. Kay and J.T. Kaiya. Ray tracing complex scenes. SIGGRAPH'86 Proceedings, pp. 296–278, 1986.

[Go-96] S. Gottschalk, M.C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. SIGGRAPH'96 Proceedings, pp. 171–180, 1996.

[Ba-96] G. Barequet, et al. BOXTREE: A hierarchical representation for surfaces in 3D. Eurograhics'96 Proceedings, 1996.

[Be-90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B.Seeger. The R*-tree: An efficient and robust access method for points and rectangles. Proceedings of ACM SIGMOD International Conference on Management of data, pp. 322–331, 1990.

[Za-94] G. Zachmann. Exact and fast collision detection, Diploma Thesis. Technische Hochschule Darmstadt, Fachbereich Informatik, 1994.

[Mo-88] M. Moore and J. Willhelms. Collision detection and response for computer animation. SIGGRAPH'88 Proceedings, pp. 289–298, 1988.

[No-89] H. Noborio, S. Fukuda, and S. Arimoto. Fast interference check method using octree representation. Advanced Robotics, pp. 193–212, 1989.

[Hu-95] P.M. Hubbard. Collision detection for interactive graphics applications. IEEE Transactions of Visualization and Computer Graphics, pp. 218-230, 1995.

[St-96] M. Held, J. Klosowski, J.S.B. Mitchell, H. Sowizral, and K. Zikan, “Real-Time Collision Detection for Motion Simulation within Complex Environments”, Manuscript, 1996. (An abbreviated version appears as a Technical Note, *ACM SIGGRAPH'96 Visual Proceedings*, New Orleans, LA, Aug 4–9, 1996.)