A New Algorithm for computing Minimum Distance

K. Sundaraj , D. d'Aulignac & E. Mazer

INRIA Rhone-Alpes (SHARP Project)
ZIRST. 655 avenue de l'Europe
38330 Montbonnot Saint Martin
FRANCE

email : { *Kenneth.Sundaraj , Diego.d_Aulignac , Emmanuel.Mazer* } *@inrialpes.fr*

Abstract

This paper presents a new algorithm for computing the minimum distance between convex polyhedras. The algorithm of Gilbert-Johnson-Keerthi (GJK) and the algorithm of Lin-Canny (LC) are well-known fast solutions to the problem. We show how a mix between LC's idea and the GJK's algorithm can be used to solve the problem. In our algorithm, we use local methods to calculate the distance between features and new 'updating' conditions to add stability. These new conditions enable us to ensure more stability when compared to GJK. We also modify our terminating conditions to add robustness to our approach. Our experiments also show that the expected running time of our approach is constant, independent of the complexity of the polyhedra. We present some comparisons of our method with GJK.

1 Introduction

Recently, it has been shown that many problems in the robotics field and other related domains can be solved with efficient algorithms that solve the minimum distance problem. Solutions to these problems typically use the mathematical model of the objects to find a point on each object such that the distance between them is minimized. Minimum distance computation has found wide applications especially in collision avoidance, path modification, optimal path planning, dynamic simulation and virtual reality.

In certain cases, distance computation is used for collision detection. Collision treatment is considered fundamental in graphics animation and CAD application. Needless to say collision detection is a bottle neck for a large number of geometrical based algorithms and in particular, dynamic based simulations. In this area, distance computation becomes very critical. We have implemented such algorithms in our dynamic simulator - *AlaDyn3D*. The results of these tests can be found in [11], [1] and [7]

Two well known algorithms for efficient minimum distance calculations between convex polyhedra is the *Gilbert-Johnson-Keerthi* (GJK) algorithm [2] and the *Lin-Canny (LC)* algorithm [3]. Sometimes, both these algorithms suffer from several drawbacks. Nagle in [8] has noted that the GJK algorithm sometimes encounter configurations of objects that cause it to loop infinitely. This problem according to [6] is related to the *Minkowski set difference* of the objects. [6] also mentions that the main source of GJK's numerical problems is due to the rounding errors that accumulate in the *Distance Subalgorithm* of GJK. In fact, [6] goes on to explain the instability of the terminating conditions in the GJK algorithm. On the other hand, LC's algorithm which is based on the utilization of *voronoi regions* requires an exhaustive data structure (at least when compared to GJK) to model the objects. [6] finds that for large number of vertices, LC's algorithm will suffer from translational and rotational velocity variations as reported in [6]. A comparison of LC's performance over GJK under various density variations is also given in [6]. Nevertheless, with some minor modifications, both algorithms give almost constant time complexity for most applications as reported in [4] and [5].

In this paper, we present a new algorithm for finding and tracking the closest points on a pair of

convex polyhedra. The method works by finding and maintaining the pair of closest features (vertex, edge, or facet) as LC's algorithm. At this point, we avoid taking the *Minkowski set difference* of the cached features but use local methods to calculate the distance between them as in [3]. We note that GJK uses the *Minkowski set difference* and solves them using linear algebra. Unlike LC's algorithm which updates it's features after having failed certain *applicability criteria* using *boundary* and *coboundary* information, we update our features using GJK's *support function*. Here, we make some changes. Instead of modifying both features, we change only one and move on to the verification stage. We can verify the validity of the points returned by the local methods by treating them directly to the specification of the convex sets in terms of their support properties, which for polytopes, can be obtained easily from their vertices. Finally, as for terminating conditions, we use the ones suggested in [6]. This condition seems to solve most of GJK's terminating problems and as such we consider it suitable.

The rest of the paper is arranged as follows. Section 2 outlines the required object representation and some basic definitions. Section 3 will give a brief description of the GJK and LC algorithms for the readers who are unfamiliar with them. The advantage of our local methods for computing the distance between features will be explained in Section 4. The general description of our algorithm will be presented in Section 5. Section 6 will present some results of our approach using problematic configurations of objects and timing comparisons with GJK. Section 7 will end this paper with the summary and conclusion.

2 Representations and Definitions

The approach described in this paper assumes the underlying model of each object as represented by a convex polyhedron or as a union of convex polyhedra. Surface representation is represented by polyhedral approximations which in turn depends on the resolution (number of vertices). Non-convex objects are subdivided into convex pieces and hierarchically represented. Each polyhedra is described by its 'Doubly Connected Edge List (DCEL)' data structure [9]. Nevertheless, our approach will even work for a data structure made up of a list of vertices alone, though a bit slower.

Let S_A and S_B define the set of points that form the surface of objects A and B respectively in R^3. $S_{A'}$ and $S_{B'}$ will then define the space exterior to A and B (the interior space is also defined in the same way, but is irrelevant to this analysis as we do not intend to treat collision at the moment). The positive distance between objects A and B is defined as the pair of features which contain the closest points. The distance between objects A and B is the shortest Euclidean distance, denoted by $d(A, B)$, defined by

$$d(A, B) = min \parallel x - y \parallel \ : \ x \in A, y \in B \qquad (1)$$

and let $X_A \in S_A, X_B \in S_B$ be such that

$$d(A, B) = \parallel X_A - X_B \parallel \qquad (2)$$

where X_A and X_B are a pair of closest points between objects A and B.

3 Overview of GJK and LC

GJK Algorithm

This algorithm is an iterative method for computing the distance between convex objects. It is fairly easy to implement, simple and applies well to general convex polytopes.

The most important information in this algorithm is a group of 1-4 pairs of vertices which is used to construct the *translational configuration space obstacle* (TCSO) [5]. The idea is that the minimum distance between objects A and B is given by the distance between the origin, O and the $TSCO$ in the *translational configuration space*. The $TSCO$ is constructed by taking the *Minkowski set difference* of these vertices. We can then express the distance between $x \in A$ and $y \in B$ in terms of their *Minkowski set difference*, $A - B$ as

$$d(A, B) = min \parallel v(A - B) \parallel \qquad (3)$$

where $v(TSCO)$ is defined as a point of the $TCSO$ nearest to the origin, such that

$$\parallel v(TCSO) \parallel \ = min \ \parallel z \parallel \ : \ z \in TCSO \qquad (4)$$

If we consider four or fewer vertices in the point set of the $TSCO$, then we will have a *simplex*. GJK then solves this *simplex* by using the *Distance Subalgorithm* to find a *witness point*, $\mathbf{u}$. Such a point then defines a function $g_u(z)$ where

$$g_u(z) = u \cdot u - u \cdot z \qquad (5)$$

If each vertex, z of the $TSCO$ satisfies the terminating condition

$$g_u(z) = 0 \tag{6}$$

then $\mathbf{u}$ is indeed the closest point from the origin to the $TSCO$. Otherwise, a point of the $TSCO$ that maximizes $-\mathbf{u} \cdot z$ will have to be found. The $support-function$ of GJK does exactly this and is expressed as

$$max\ (-\mathbf{u} \cdot z) = \ max\ (\mathbf{u} \cdot x) + max\ (\mathbf{u} \cdot y) \tag{7}$$

Readers who would like to know how the *Distance Sub-algorithm* of GJK solves each simplex can refer to [2] for more details. It is in this function that most of the numerical problems emerge and feasible solutions are proposed in [2] and [6]. Application of *hill-climbing* to GJK can be found in [5]. With *hill-climbing*, [4] reports that this algorithm has constant time complexity and roughly performs a total of 104 arithmetic operations.

LC Algorithm

For this algorithm, object representation must follow [3]. Basically, given two initial points, say p and q, which belong to object A and B, let

$$(S_{A'}\ ,\ S_{B'}) \subset (V_{Fp}\ ,\ V_{Fq}) \tag{8}$$

where F_x is the feature that point x belongs to, V_x is the $Voronoi\ Region$ [9] of feature x and S_x, the set of points that define V_x, LC's algorithm check if

$$p \in S_{B'}\ and\ q \in S_{A'} \tag{9}$$

LC uses three *applicability criterion* functions namely Point-Vertex, Point-Edge and Point-Face to verify (9) and if any of these tests fails, with some additional returned information using *boundary* and *coboundary* features, LC is able to $'walk'$ to a new pair of features whose minimum distance is guaranteed to be closer than the old one. Thus, it is clear that this algorithm will terminate in a number of steps at most equal to the number of feature pairs. [4] reports that LC has the same expected cost as the $Enhanced - GJK$ and performs roughly a total of 111 arithmetic operations. A detailed explanation of this algorithm can be found in [3].

4 Local Method Computations

Since the only possible feature pairs that could realize the minimum distance is point-point, point-edge, point-facet and edge-edge, we only need to formulate efficient and robust methods to solve them. We have found the algorithms by [10] to be suitable for our applications. We extract some important notes from the web site and cite them here. The advantage is that we arrive at the solution for each possible feature pair with at most performing 1 division. Except for the point-point case, the others need detailed analysis. We begin with the analysis of point-point.

Point-Point

The divisionless distance function Q, between point P_0 and P_1 is given by

$$Q = \parallel P_0 - P_1 \parallel \tag{10}$$

Point-Edge

Let the point be P. The edge, E can be expressed as a line segment by

$$E(t) = B + tM \tag{11}$$

where B is a point on the line, M is the line direction with $t \in [0,1]$. If $\parallel M \parallel^{\ 2} \leq \epsilon$, where ϵ is a user defined value to denote minimum edge length, then $t = 0$ and we consider the edge as a point and use point-point analysis. Otherwise, let

$$t' = M \cdot (P - B)\ ,\ t'' = M \cdot M \tag{12}$$

The distance function $Q(t)$, from point P to the edge E is given as

$$Q(t) = \begin{cases} \parallel P - (B + M) \parallel & \text{if } t' > 1, t'' < t' \\ \parallel P - B \parallel & \text{if } t' \leq 0 \\ \parallel P - (B + \frac{t'}{t''}M) \parallel & \text{otherwise} \end{cases} \tag{13}$$

The operation $\frac{t'}{t''}$ is left to the end and is only performed if necessary.

Point-Facet

The problem of finding the minimum distance between a point P and a facet F, defined as

$$F(s,t) = B + sE_0 + tE_1 \tag{14}$$

where E_0 and E_1 are two edges of the triangle, $(s,t) \in R = \{(s,t) : s \in [0,1], t \in [0,1], s + t \leq 1\}$ is obtained by computing $(s',t') \in R$ corresponding to a point on the facet closest to P.

Let $a = E_0 \cdot E_0$, $b = E_0 \cdot E_1$, $c = E_1 \cdot E_1$, $d = E_0 \cdot (B - P)$, $e = -E_1 \cdot (B - P)$ and $f = (B - P) \cdot (B - P)$. If $ac - b^2 < \mu$, where μ is a user defined value, then the two edges of the facet are not linearly independent and we will then treat the facet as an edge and use point-edge to solve the problem. Else, the minimum squared-distance function is given as

$$\begin{aligned} Q(s,t) &= \| E_0(s) - E_1(t) \|^2 \\ &= as^2 + 2bst + ct^2 + 2ds + 2et + f \end{aligned} \quad (15)$$

The aim is to minimize $Q(s,t)$ in R. The method to obtain the minimum distance for each region can be obtained from [10]. But, we would like to note that at most 1 division is computed with the denominator as $ac - b^2$.

Edge-Edge

Let us denote the two edges as line segments represented by

$$E_0(s) = B_0 + sM_0 \ , \ E_1(t) = B_1 + tM_1 \quad (16)$$

for $s \in [0,1]$, $t \in [0,1]$. In this case the minimum squared distance function $Q(s,t)$ for $(s,t) \in [0,1]^2$ is given as

$$\begin{aligned} Q(s,t) &= \| E_0(s) - E_1(t) \|^2 \\ &= as^2 + 2bst + ct^2 + 2ds + 2et + f \end{aligned} \quad (17)$$

where $a = M_0 \cdot M_0$, $b = -M_0 \cdot M_1$, $c = M_1 \cdot M_1$, $d = M_0 \cdot (B_0 - B_1)$, $e = -M_1 \cdot (B_0 - B_1)$ and $f = (B_0 - B_1) \cdot (B_0 - B_1)$. For this function Q

$$ac - b^2 = \| M_0 M_1 \|^2 \geq 0 \quad (18)$$

If $ac - b^2 > 0$, then the two edges are not parallel, else if $ac - b^2 = 0$, then the two edges are parallel. For the non-parallel case, our aim is to minimized $Q(s,t)$ in $[0,1]$. Readers are referred to [10] for solutions to the parallel and non-parallel case. In this case also, at most 1 division is required and the denominator in this division is $ac - b^2$.

We end this section by noting that for the edge-facet case, we use methods as explained in [3] to reduce the dimension of the combined feature. [10] gives a method to perform a similar operation for facet-facet but in all our experiments, we have never encountered such a case and thus we consider it irrelevant for our application. Another method which we use is to recycle the features to obtained the correct minimum distance feature.

5 Description of the Algorithm

In this section, we will explain how our approach is executed. We assume that the objects A and B are initially separated. We begin with a random point from each object. Let F_A and F_B define the feature formed by the set of points taken from A and B respectively. Since our models are represented by points and form features of vertices, edges and facets, it is clear that for convex objects, we can uniquely express $d(F_A, F_B)$ as distance between one of the following feature pairs : point-point,point-edge,point-facet and edge-edge. The minimum distance between an edge and a facet or a facet and another facet can always be expressed as one of the above feature pairs.

Depending on the kind of features that we cache at each iteration, the local distance method will calculate the minimum distance between these features and if necessary, update the cached features. It is here that we make some changes with respect to GJK. As we have mentioned before, we use GJK's *support function* to update our features. In our approach, we apply this function to one object at a time. After applying, if we find a new point, we update this feature and verify if the new formed feature is *stable* or not. If the formed feature is *unstable*, there could only be 2 possibilities: the object size is too small or the feature of that object is too small. In either case we terminate immediately without moving on to the verification stage. If the feature is *stable*, then we apply our local distance methods to get the distance. If this distance is not the minimum, we proceed to the other object and repeat the process. Such a method ensures that all the features formed are entirely *stable*. Then our local distance methods have no problems getting the distance values. Hence, the entire approach becomes stable.

This minimum distance vector when obtained, is then verified using the modified terminating condition mentioned in [6]. We repeat it here for clarity sake.

```
if { d(Fa,Fb) <= Ae } end;
else if { d(Fa,Fb)-Me <= d(Fa,Fb)*Re } end;
else update (Fa) or (Fb)
```

where A_e is the user defined absolute error that denotes collision and R_e is the relative error at the *k-th* iteration derived by

$$\begin{aligned} R_e(k) &= \frac{d(A,B)_{k-1} \ \cdot \ d(F_A, F_B)_k}{\| \ d(A,B)_{k-1} \ \|} \\ M_e(k) &= max \ \{ \ R_e(k-1) \ , \ R_e(k) \ \} \end{aligned} \quad (19)$$

If the terminating condition is not satisfied, then we use the *support function* of GJK to get the new support points. These points are then added to the set F_A *or* F_B and the process is repeated. We note here that with some neighbouring adjacency information of a polytope the cost of computing a support point of a convex polyhedra can be reduced to almost constant time. This technique is well known as *hill climbing* and has been implemented in the Enhanced-GJK version [5]. Our experiments were carried out with this option available. Finally, at termination, X_A and X_B, a pair of closest points is computed as follows.

$$X_A = \sum_{i=1}^{n} \lambda_i F_{A_i} \ , \ X_B = \sum_{j=1}^{m} \gamma_j F_{B_j} \qquad (20)$$

where F_{A_i} and F_{B_j} are points that make up the set F_A and F_B respectively whereas λ_i and γ_j are coefficients that we derived previously from the local distance method. They are obtained by solving equation (10),(13),(15) or (17) depending on the features that realize the minimum distance.

6 Experimental Results

We present some experimental results here to shown that our approach is workable. We compare 2 main aspects of the algorithm: Robustness and Timing. **Robustness :** There are basically 3 problems in GJK as reported by [5]. They are the terminating conditions, the Distance Sub-Algorithm and the geometry of the objects. As stated before, we have improved our terminating conditions by using the ones presented in [5]. The Distance Sub-algorithm as used in GJK has been replaced by our local methods which is able to handle unstable simplices such as: an edge too small, a facet with 2 dependent edges and parallel edges. Further constraints can also be added such as when the objects are too far away. Thus what remains to be verified in our approach is for certain configurations of objects. According to [9], two problematic configurations are when the objects are too close and when they differ in a few orders of magnitude. We used 2 cubes and 2 tetrahedrons to test our approach. The results are given below

Experiment 1 : 2 very small tetrahedrons very near

Object	Edge Length	Vertices
Tetra1	1e-23	4
Tetra2	1e-23	4
	Distance apart : 20e-23	

Experiment 2 : 2 cubes very close and of different orders in magnitude

Object	Edge Length	Vertices
Cube1	2000	8
Cube2	1e-15	8
	Scaling Factor : 10e39 Distance apart : 1e-10	

Each of the above experiment was repeated about 100 times under different rotational velocities. We encountered no numerical problems for any of the above using our approach. **Timing :** We compared the execution times of the algorithm with Cameron's enhanced GJK hill-climbing algorithm[1]. he objects tested are randomly generated convex polyhedra. For a pair of objects one is placed at the origin while we apply a continuous translation to the other object through 10e4 little incremental displacements.

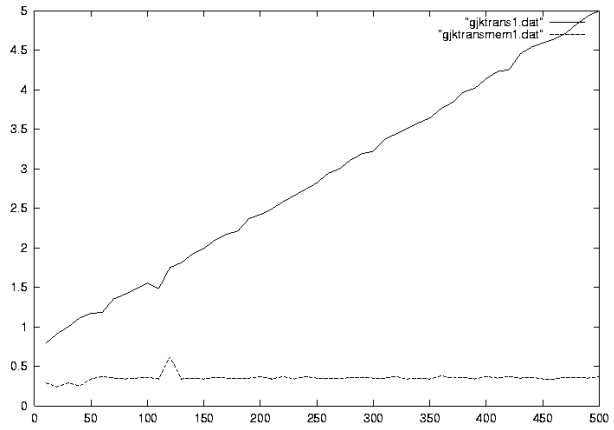

Figure 1: *Enhanced GJK: Time for continuous translation with (dashed) & without (solid) tracking info.*

Figure 1 shows how the timings evolve with increasing object complexity. When no tracking information is used we obtain a linear relationship between complexity and execution time. However, when making use of this information the time is almost constant ($0.36s$ on average). When repeating this experiment with our approach we obtain the plot given in Figure 2. We observe the same linear and near-constant relationship without and with tracking information. However, with our approach the time increase is slower with rising complexity of the objects,

[1]The code used was downloaded from http://www.comlab.ox.ac.uk/ cameron/distances.html

and the near-constant time drops to 0.24s on average. For the experimental results shown in Figure 3 we placed two objects with 50 points each a given distance apart and then applied a continuous rotation over 10e4 steps to one of them. The plot shows how the timings evolve for different rotational velocities. The tracking information is used, of course. We observe that our approach gives lower timings even at high rotational speeds as compared to the enhanced GJK method. These results were obtained using a *SGI Octane (195MHz)*, and the code was compiled using the *CC* compiler with *-Ofast* optimisation. However, some caution should be taken when interpreting these results, since they represent a purely practical analysis.

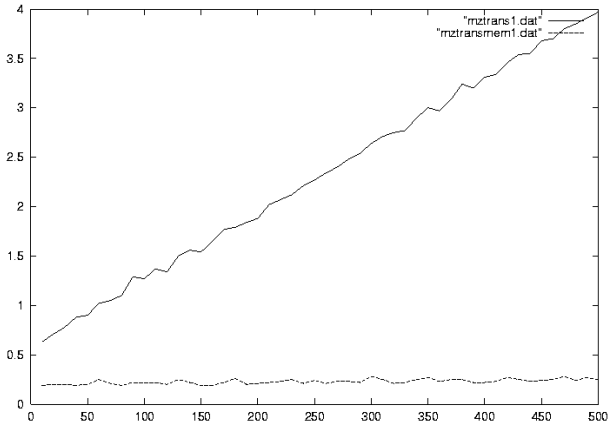

Figure 2: *Our approach: Time for continuous translation with (dashed) & without (solid) tracking info.*

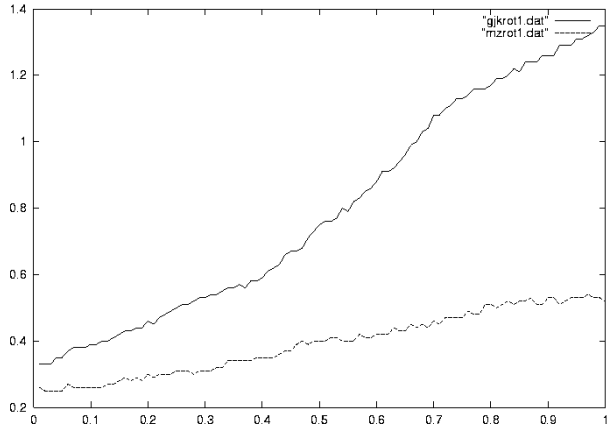

Figure 3: *Evolution of the time for a continuous rotation with different rotational velocities (x-axis).*

7 Summary and Conclusions

We have shown in this paper a new algorithm to solve the minimum distance problem. This can be done by using the caching of features method like in LC's algorithm and using local methods to compute the distance between them. With our local methods and the 'updating conditions' using GJK's *support function*, we are able to add more stability. With experiments using problematic configuration of objects as described in [6], we have no numerical problems.

References

[1] A.Joukhadar and C.Laugier, "Dynamic Modeling and its Robotic Application," *IEEE/ICRA,* May 1995.

[2] E.G.Gilbert, D.W.Johnson and S.S.Keerthi, "A Fast Procedure for Computing the Distance Between Complex Objects in Three Dimensional Space," *IEEE Journal of Robotics and Automation,* 4(2), 1988.

[3] M.C.Lin and J.F.Canny, "A Fast Algorithm for Incremental Distance Calculation," *Proc. of IEEE International Conference on Robotics and Automation,* pp. 1008-1014, 1991.

[4] Stephen Cameron, "A Comparison of Two Fast Algorithms for Computing the Distance between Convex Polyhedra," *Proc. of IEEE Transactions on Robotics and Automation,* 1996.

[5] Stephen Cameron, "Enhancing GJK: Computing Minimum and Penetrating Distances between Convex Polyhedra," *Proc. of IEEE International Conference on Robotics and Automation,* pp. 3112-3117, 1997.

[6] Gino Van der Bergen, "A Fast and Robust GJK Implementation for Collision Detection of Convex Objects," *To appear,* 1999.

[7] A.Joukhadar, A.Wabbi and C.Laugier, "Fast Contact Localisation between Deformable Polyhedra in Motion," *Computer Animation,* 1996.

[8] J.Nagel, "GJK collision detection algorithm wanted," *comp.graphics.algorithms newsgroup,* 1998.

[9] F.P.Preparata and M.I.Shamos, "Computational Geometry," *An Introduction,* 1985.

[10] Dave Eberly, "Source Code for Computer Graphics, Image Analysis and Numerical Methods," *http://www.magic-software.com*

[11] A.Joukhadar and C.Laugier, "Dynamic Modeling of Rigid and Deformable objects for Robotic Tasks," *International Conference ORIA,* 1994.