To appear at IROS 2003, Las Vegas, Oct. 2003

Tracking Minimum Distances between Curved Objects with Parametric Surfaces in Real Time

Zhihua Zou, Jing Xiao
Department of Computer Science
University of North Carolina – Charlotte
zzou2008@yahoo.com, xiao@uncc.edu

Abstract

This paper presents a new algorithm for real-time tracking of pairs of closest points as well as their corresponding surface features between certain general types of objects (which can be non-convex) with *parametric curved surfaces*. The fact that the algorithm works directly on accurate parametric descriptions of curved surfaces rather than polygonal approximation of surfaces (i.e., polygonal meshes) enables it to not only provide accurate collision detection among certain curved objects in real-time, but more importantly, also provide *accurate* description of the state of a collision, i.e., the actual regions of contact in real-time. Such capability is very useful in applications requiring high accuracy in real-time, including certain haptic rendering tasks for virtual prototyping or virtual training. Test results show that the algorithm achieves correct tracking in the rate of 1 *k*Hz.

1. Introduction

Collision avoidance or detection based on distance computation requires very efficient algorithms to be real-time. There is considerable research on the subject [1, 2]. One major approach to boost efficiency is by applying collision check to multi-levels of simpler bounding envelope approximations of objects (such as boxes, spheres, etc) in order to rule out non-collisions or localize areas of collisions quickly. In addition, spatial and time coherences are often exploited through tracking to speed up the computation [3, 4].

However, most of the real-time collision detection algorithms apply to polyhedral objects or polyhedral approximations of objects only (i.e., polygonal meshes) [1, 2]. Although polygons are simple to compute, polygonal approximation of curved objects often introduce undesirable artifacts especially in cases where the smooth nature of a curved surface needs to be maintained. Whereas, increasing the resolution of meshes tends to increase the burden of computation significantly. On the other hand, it is often the case that a curved surface that may require tens and thousands of polygons to approximate in order to reach certain level of smoothness can be described accurately in simple parametric forms. It is thus desirable to study collision detection and distance computation applied *directly* to smooth parametric surface expressions of curved objects, but there are relatively fewer related publications [5, 6, 7, 8]. In particular, Johnson and Cohen achieved real-time collision detection through effective multi-level representation of curved objects [7]. More recently, Patoglu and Gillespie developed an efficient real-time strategy to directly track the closest points between moving convex curved surfaces [8]. Their work presented an interesting new idea to achieve very efficient real-time distance computation and collision detection. However, the work can only be applied to a pair of natural geometrical surfaces, which, except for the closed ones such as spheres and ellipses, are often *infinite* surfaces.

In this paper, we present a different and more general strategy to directly track the closest points in real time between two rigid objects consisting of *finite* surfaces, i.e., either closed surfaces or finite surface patches, described parametrically. A general object with two or more parametric surface patches can be viewed as formed by a set of *features*, in terms of parametric *surfaces* and the surface bounding *curves* and *vertices*. Our algorithm tracks both the closest features between two objects and the closest points (on the closest features) at the same time.

Given initial pairs of the closest points between each pair of features between two objects as input, which can be computed or estimated off-line, our algorithm then tracks, in real-time, the time-changing closest features and points between the two objects as they move. The key of our approach is to repeatedly find good candidates of the closest features and good estimates of the closest points based on the results from the previous time frame so that the Newton's approach can be properly applied to quickly find the true results, i.e., the closest points and their features. In the following sections, we describe the algorithm in detail and present implementation and test results.

2. Object Representation and Types

Our algorithm requires that the boundary of each object be described by a set of features, i.e., surfaces, curves,

and vertices, and their adjacency relations arranged in a hierarchical structure with three levels. Surface features are at the top level of the feature hierarchy, their bounding (lower-dimensional) curve features are at the next level (where each curve feature bounds two or more surface features), and finally the bounding (lower-dimensional) vertices of the curve features are at the bottom level. Furthermore, each surface feature is represented by a parametric function $f(u,v)$ with bounded parameters u and v, and each curve feature is represented by a function $f(w)$ with a bounded parameter w. A vertex is represented by its coordinates with zero parameter.

Our algorithm can be applied to general non-convex curved objects if the boundary of each object can be decomposed into a set of convex features (viewed from normal directions pointing into the object). See Fig. 1 for examples. With convex features, there is no more than one local minimum distance between any two features at any time, and there is no local maximum distance. Of course, between two objects at any time, there can be multiple local minima of distance values.

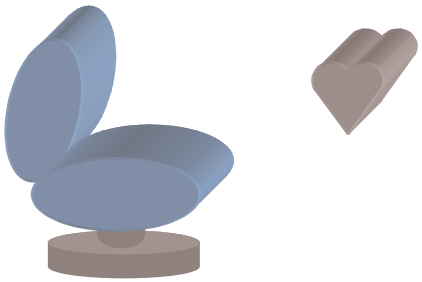

Figure 1: Non-convex curved objects made of convex features

In addition, our algorithm can also handle objects with certain monotonically concave and circular features, such as a spherical surface feature forming the inside of a bowl (Fig. 3) or a cylindrical surface feature forming the inside of a pipe, even though such a concave feature can cause local maximum distance between it and another feature.

3. The Algorithm

For real-time tracking, our algorithm discretizes the continuous time into a sequence of adjacent time frames. At each time frame, our algorithm decides the closest pair(s) of features between two rigid objects O_1 and O_2 and a corresponding closest pair of points on the two features (which determines the minimum distance between the two features) based on *known* pairs of closest points and features between O_1 and O_2 from the previous time frame and the locations (or configurations) of O_1 and O_2 in the current time frame.

It takes advantage of time and space coherence due to continuity: each time frame is made small enough so that a closest pair of points between two features of O_1 and O_2 in time frame i will be close to the known closest pair of points between the features in the previous time frame i-1, which is the estimate for the Newton method in our approach.

3.1. A top-down approach

Our algorithm uses a top-down approach to search for the closest pair of points of any feature pair between the objects O_1 and O_2 in time frame i by considering the two points in the order from high-dimensional to low-dimensional. For convenience, we measure the dimensionality of a feature pair as the *sum of parametric variables* in the two features. For example, a surface, curve, and vertex feature has 2, 1, and 0 parametric variables respectively. Hence, a surface-surface feature pair can be denoted as having a dimensionality 4; whereas, a surface-curve feature pair is of dimensionality 3, etc.

From a surface-surface feature pair (a, b) between O_1 and O_2 and a corresponding closest pair of points (p_a, p_b) between a and b found in the previous time frame i-1, our algorithm tries to find the minimum distance and a corresponding pair of closest points between the two features at time frame i by applying a general procedure `FindMin` to a and b. `FindMin` searches a closest pair of points between the two features based on the Newton optimization method with the initial value (p_a, p_b), as will be described in Section 3.2.

If `FindMin` returns no solution, it means that the closest points between a and b are on certain lower-dimensional feature pairs involving the boundary curves of a and/or b. Thus, `FindMin` will be applied to *every* such lower-dimensional feature pair (c, d) involving the boundary curves of a and b with the initial value (p^*_a, p^*_b), where p^*_a and p^*_b are points on c and d closest to p_a and p_b respectively. (p^*_a and p^*_b are found by applying Newton method again). In each case, if again no solution can be found, again `FindMin` will be applied to *every* one of the related even lower dimensional feature pairs (i.e., involving boundary vertices of the curves) in the same fashion. This top-down process starting from the surface-surface pair (a, b) will *always* end up with a solution in terms of a pair of features, which could be of lower dimension than (a, b), that determine the current minimum distance between a and b and a corresponding pair of current closest points (p^i_a, p^i_b) on them respectively (at frame i). This is because there always

exists a minimum distance between two finite surface patches including their bounding curves and vertices.

Conducting the above process from every surface-surface pair between O_1 and O_2 and comparing the distances between the obtained locally closest feature pairs, our algorithm can then determine the globally closest feature pair(s) between O_1 and O_2 at time frame *i*, with the corresponding pair(s) of closest points, each of which has the (same) globally minimum distance between them at time frame *i*.

3.2. The `FindMin` procedure

Given two features *f* and *g* belonging to two objects O_1 and O_2 respectively, the `FindMin` procedure searches the pair of closest points on *f* and *g* respectively (which determine the minimum distance between *f* and *g*) based on the Newton optimization method, starting from a given initial estimate pair of closest points.

Newton's methods are common optimization techniques. To find a minimal $F(\vec{X})$, where $\vec{X}$ is a multi-variance vector, the procedure starts from an initial solution $\vec{X}_0$ and proceeds by minimizing the second order Taylor series approximation for *F* through iterations. In each iteration, the solution $\vec{X}_k$ is updated by:

$$\vec{X}_k = \vec{X}_{k-1} - J^{-1}\nabla F(\vec{X}_{k-1}) \qquad (1)$$

where *J* is the Hessian matrix of *F*, and $\nabla F(\vec{X}_{k-1})$ is the first derivative of *F*. Such a method has second order convergence rate so that the solution can be reached very fast if the initial solution $\vec{X}_0$ is close by.

In `FindMin`, the objective function for minimization is the square of the distance between two features *f* and *g*:

$$F(\vec{X}) = \left\|\vec{d}\right\|^2 = \min_{\vec{X}} \left\|\vec{f}(\vec{p}) - \vec{g}(\vec{q})\right\|^2 \qquad (2)$$

where $\vec{p}$ and $\vec{q}$ are the vectors of parameter(s) of *f* and *g*, with 0, 1 or 2 elements if the corresponding feature is a vertex, a curve or a surface respectively, $\vec{X} = [\vec{p}\ \vec{q}]^T = [x_0 \cdots x_n]^T$, and *n* is the dimensionality of the feature pair as defined earlier. Note that when *n* is zero, $\vec{X}$ is zero vector, and both features are vertices with a unique distance between them.

From multi-variant calculus, for a non-zero $\vec{X}^*$ to be a local minimum, the necessary condition is that

$$\nabla F(\vec{X}^*) = 0 \qquad (3)$$

i.e., the first derivatives of *F* with respect to the elements in $\vec{X}$ must be zeros. When Eq. (3) is satisfied, the vector connecting two points on the two features respectively is orthogonal to the gradient at each point in each feature. An example is shown in Fig. 2 with a curve feature and a surface feature, where the solution $\vec{X}^* = [w_0\ u_0\ v_0]$ satisfies Eq. (3), and the vector $\vec{d}$ is orthogonal to all gradient vectors $\vec{g}_w$, $\vec{f}_u$ and $\vec{f}_v$.

If at least one of the features *f* and *g* is an *open surface* such that its boundary curve(s) are not counted or an *open curve* such that its boundary vertices are not counted, there are three possible scenarios:

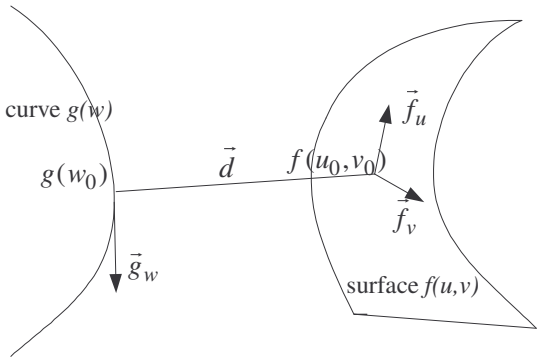

Figure 2: the vector $\vec{d}$ is orthogonal to all gradients of both surface feature *f* and curve feature *g*.

1. There is no extreme (i.e., minimum or maximum) solution of $\vec{X}$ satisfying Eq. (3).
2. A solution of $\vec{X}$ satisfying Eq. (3) exists, but it is not a minimum solution between the two bounded features *f* and *g*.
3. There exists a minimum solution of $\vec{X}^*$ (as illustrated in Fig. 1) between the bounded *f* and *g*.

Our `FindMin` routine first runs the Newton's iterations based on Eqs. (1)–(3). If a solution satisfying Eq. (3) cannot be found after a sufficient *m* number of iterations, then the above Scenario 1 can be assured because the Newton's method converges very quickly if there is a solution and the given initial solution $\vec{X}_0$ is close by (due to time and space coherence, see Section 3.1). In such a case, `FindMin` returns "no solution" and stops.

Otherwise, if a solution of $\vec{X}$ satisfying Eq. (3) is found, `FindMin` will then check if every parameter value in the solution is within the bounds as defined by the corresponding feature. If not, it means that the solution is on the corresponding infinite features of *f* and/or *g* but not on the bounded *f* and *g*, and it is Scenario 2. `FindMin` thus returns "no solution" and stops.

If the solution $\vec{X}^*$ satisfying Eq. (3) is within the bounds of features *f* and *g*, and if *f* and *g* are convex, then it is Scenario 3, and `FindMin` returns $\vec{X}^*$ as the solution.

If either *f* or *g* is a monotonically concave and circular feature (Section 2), `FindMin` will further check whether $\vec{X}^*$ is a minimum solution or not. If *f* is a curve feature *f(w)*, then $\vec{X}^* = [w_0 \ \vec{q}_0 \ ]^T$, and the sufficient condition for $f(w_0)$ to be the closest point of *f* to *g* is:

$$\vec{f}_w \bullet \vec{f}_w + \vec{f}_{ww} \bullet (\vec{f}(w) - \vec{g}(\vec{q}))\Big|_{\substack{w=w0 \\ \vec{q}=\vec{q}0}} > 0 \qquad (4)$$

If *f* is a surface feature *f(u, v)*, then $\vec{X}^* = [u_0 \ v_0 \ \vec{q}_0 \ ]^T$, and the sufficient condition for $f(u_0, v_0)$ to be the closest point of *f* to *g* is:

$$\begin{aligned} A &= \vec{f}_u \bullet \vec{f}_u + \vec{f}_{uu} \bullet (\vec{f}(u_0, v_0) - \vec{g}(\vec{q}_0)) \\ B &= \vec{f}_u \bullet \vec{f}_v + \vec{f}uv \bullet (\vec{f}(u_0, v_0) - \vec{g}(\vec{q}_0)) \\ C &= \vec{f}_v \bullet \vec{f}_v + \vec{f}vv \bullet (\vec{f}(u_0, v_0) - \vec{g}(\vec{q}_0)) \\ B^2 - A \cdot C < 0 &\text{ and } A > 0 \end{aligned} \qquad (5)$$

Similar conditions can be written for the point $g(\vec{q}_0)$ to be the closest point of *g* to *f*. Since the partial derivative vectors in Eq. (4) and Eq. (5) are already computed and used in the original Newton's iterations, `FindMin` can check these conditions very quickly to decide if the solution $\vec{X}^*$ is a minimum solution or not. If yes, it is Scenario 3, and `FindMin` returns $\vec{X}^*$ as the solution and stops.

Otherwise, `FindMin` quickly finds a new initial estimate $\vec{X}_0$ based on $\vec{X}^*$ and the cyclosymmetry and runs the Newton's iterations again for solution. The obtained solution will be a minimum solution, and if Scenario 3 holds, `FindMin` will return it and stops. If not, `FindMin` will report "no solution" and stops.

As described in Section 3.1, when `FindMin` returns no solution, our top-down algorithm will apply `FindMin` to a lower-dimensional feature pair involving either or both boundary features of *f* and *g*, and so on, until a solution is found (and there is *always* a solution eventually).

3.3. Efficient computation

The algorithm as described can be efficiently implemented by avoiding redundant checking of the feature pairs. Recall that our top-down approach goes from higher dimensional feature pairs to lower dimensional feature pairs. Since a lower-dimensional feature pair is related to more than one higher-dimensional feature pair (e.g., a curve-surface pair is related to two surface-surface pairs), it can be visited more than once from different higher dimensional feature pairs. In such a case, our algorithm only applies `FindMin` to it once and stores the found results for possible future use.

Our search process does not need to check all non surface-surface feature pairs (i.e., with dimensionality ≤3). Such a lower-dimensional feature pair will be examined only if it is reached from a higher dimensional feature pair during the top-down process.

4. Implementation and Test Results

We have implemented our algorithm using C++. The implementation achieves real-time computation with an update rate of ~1*k* Hz on a Pentium IV processor (1.4G CPU, 256M RAM). In our test cases, a solution of `FindMin`, if existing, is usually found in 2—3 iterations, and the maximum number of Newton iterations is set to 12.

Our algorithm requires knowing object features as well as their parametric representations. To facilitate implementation, we currently build a feature database with some basic types of features, such as planes, ellipsoidal surface, cylindrical surfaces, conical surfaces, parabolic surfaces, straight lines, ellipses, parabolic curves, and vertices. We plan to extend the database to include other types of features such as polynomial curves and NURBS surfaces.

Our testing program simulates object movements by taking as inputs the trajectories of moving objects in order to know the location of each object at each time frame. The locations of interested feature pairs are then obtained by proper coordinate transformations to facilitate needed computation.

Fig. 3 shows a test example with two objects, a pen and a bowl. The pen has three convex surface features, (a top plane *S5*, a cylindrical surface *S4*, and a cone *S3*), two boundary circles (*C2* and *C3*), and a vertex *V1*. The bowl has two surface features (a convex parabolic surface *S1*, and a concave spherical surface *S*2), and a circle (*C1*).

The frames of the bowl and the pen are shown as $X_bY_bZ_b$ and $X_pY_pZ_p$ respectively. The global frame is overlapped with the frame of the pen. The representations of the surfaces in their part frames are listed in Eq. (6) (Note that we do not provide parametric representations here to be concise). The representations of the curve features can be derived from those of the surfaces.

$$\begin{aligned} &S1: x^2 + y^2 = 4z \quad S2: x^2 + y^2 + (z-7)^2 = 5^2 \\ &S3: 4z^2 = x^2 + y^2 \quad S4: x^2 + y^2 = 1 \\ &S5: z = 7 \end{aligned} \qquad (6)$$

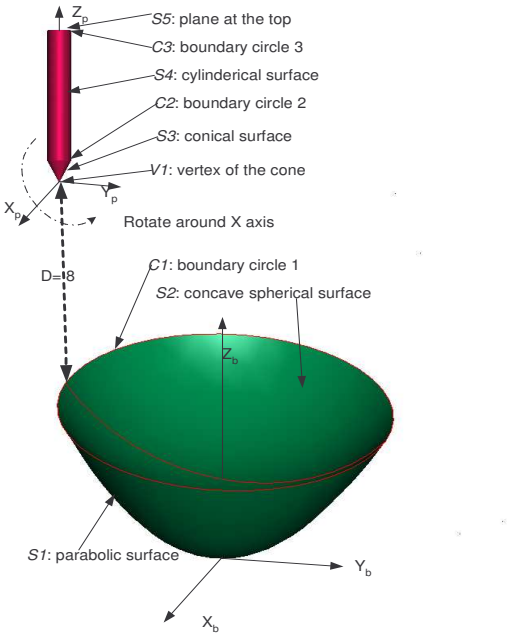

Figure 3: two objects (a bowl and a pen) in their initial locations with a minimum distance 8

The relative location of the two frames in Fig. 3 is described by the following homogenous transformation matrix:

$$ {}^{Bowl}T_{Pen} = \begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -4 \\ 0 & 0 & 1 & 12 \\ 0 & 0 & 0 & 1 \end{Bmatrix} \qquad (7) $$

In one test, we make the pen move along a straight line that locates on the YZ plane (x=0) as shown in Fig. 4 (so that they can be projected into the YZ plane for convenience). In the left plot of Fig. 4, the pair of points that decide the global minimum distance are shown in 9 time frames. The right plot of Fig. 4 shows the values of the global minimum distance and the corresponding pair of features as the function of the pen's y coordinates when the pen moves. For example, when the pen moves to y=22, the minimal distance is decided by the pair of features (*S1*, *C5*) with the value 15.40764. Table 1 lists the coordinates and distance of the closest pair of points with respect to their own object frames as the pen moves.

In another test, we make the pen rotating about its x axis by 1° in each time frame. The result is shown in Fig. 5, where the global minimum distance and the corresponding pairs of closest features are shown as functions of the angles rotated as the pen moves. For example, the minimal distance is decided by the pair of features (*C*1, *S*5) when the pen has rotated 180°, and by the pair of features (*S*2, *C*3) when the pen has rotated 210°. Table 2 lists the coordinates of the closest pairs of points and the corresponding distance when the pen rotates around its x axis. The points are represented in their own object frames.

Table 1: Pairs of closest points in several time frames

Y	Min_Dist	Point (Pen)	Point (Bowl)
0	8.0	(0,0,0)	(0,-4,4)
4	5.65685	(0,0,0)	Full C1
8	0	(0,0,0)	(0,4,4)
8.1	0.04472	(0,-0.03,0.06)	(0,4,4)
8.8	0.35777	(0,-0.48,0.96)	(0,4,4)
9.5	1.0	(0,-0.5,1.5)	(0,4,4)
14.8	6.3	(0,-0.5,6.8)	(0,4,4)
15.5	7.01783	(0,-0.5, 7)	(0,4,4)
22	15.40764	(0,-0.5, 7)	(0,3.923 3.84)

Table 2: Pairs of closest points when the pen rotates

°	Min_Dist	Point (Pen)	Point (Bowl)
45	8	(0,0,0)	(0,-4,4)
66	7.99198	(0,-.16,.32)	(0,-4,4)
90	7.56637	(0,-.5,1)	(0,-4,4)
120	6.42820	(0,-.5, 4)	(0,-4,4)
170	1.24994	(0,-.5,7)	(0,-4,4)
178	0.99513	(0,-0.279,7)	(0,-4,4)
182	0.99513	(0,0.279,7)	(0,-2.897,2.925)
210	3.38993	(0,0.5,7)	(0,4,4)
225	4.31371	(0,0,7)	(0,4,4)
270	7.56637	(0,0.5,7)	(0,4,4)
293	7.98452	(0,.222,.445)	(0,-4,4)

From the two sets of test results, one can see that the minimum distance changed smoothly until when the pair of features that decide the minimum distance changes from one frame to the next. Since the same pair of features could decide the minimum distance most likely in more than one time frame, we can speed up the tracking process by considering the same pair of features over a number of consecutive time frames in most cases. We also tested our algorithm in haptic rendering where the held object was moved arbitrarily by a user via a PHANToM 6DOF device and the min. distances to the other objects were shown in real-time – see video.

5. Discussions

Our algorithm can be evaluated in three aspects: speed, accuracy, and robustness. The current implementation of the algorithm already achieves real-time updating rate, but it can be further speeded up in a number of ways. One is that not all surface-surface feature pairs need to be examined in every time frame because a pair of such features with a distance significantly greater than the minimum distance between the objects at time frame i-1 are not likely to become the closest pair of surface features at time frame i (taking into account the boundary curves and vertices). We can set up a threshold based on the maximal speed of the objects and use it to

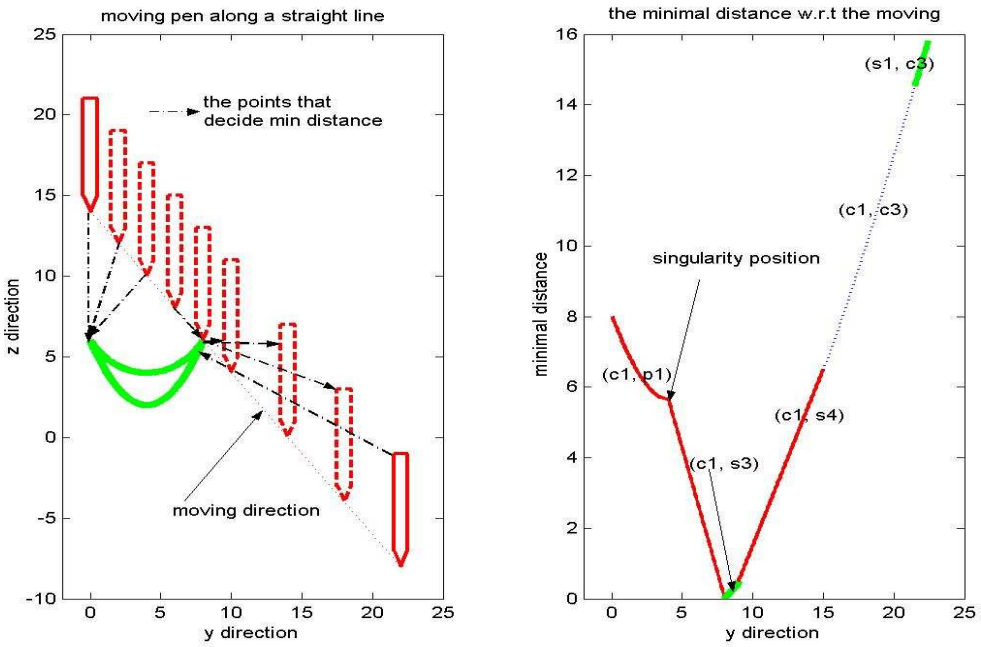

Figure 4: The closest pairs of points and features and the minimum distance when pen moves along a straight line

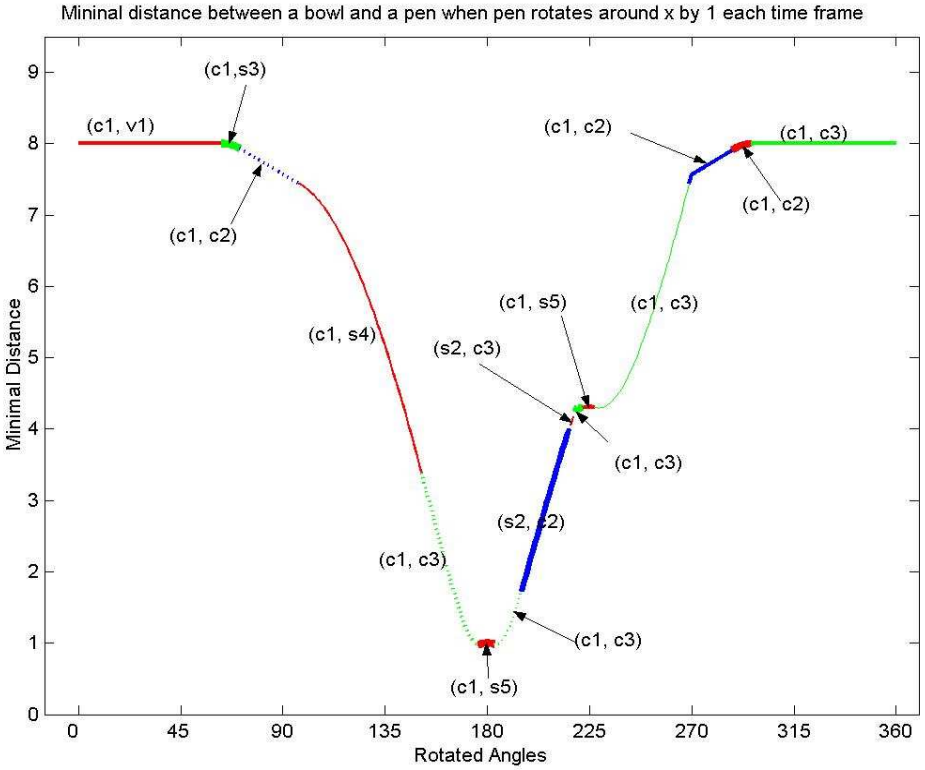

Figure 5: The closest pairs of features and the minimum distance when pen rotates along its x axis

select pairs of surface features that are likely candidates to produce the closest feature pairs in each time frame. Distances between other surface feature pairs can be updated in slower rates. Other strategies introduced in the literature for such selection can also be applied [1,2]. Note that for convex objects, candidate surface-surface feature pairs can be selected as only those neighboring to the closest pair of features in the previous time frame.

Other methods for further speeding up include parallel processing of surface-surface feature pairs (this will be straightforward since our algorithm is naturally parallel) and using multilevel representation of objects, which is a common approach in distance computation algorithms.

The accuracy of our algorithm mainly depends on the resolutions in testing the convergence conditions in `FindMin`, which include (a) the orthogonal condition between the distance vector connecting two points of two features respectively and the tangent planes through those points, and (b) the condition for termination based on parameter change rate. The higher the resolutions are, the more accurate the solution can be (the drawback is that the number of Newton iterations will increase).

Our algorithm is robust for the types of objects described in Section 2. If, for example, an object involves a non-monotonically concave feature, then there can be multiple local minimum distances between such a feature and another feature of another object, and the algorithm cannot readily find all of them to determine the global minimum distance between the two features. This limitation is due to the Newton's approach for always finding the local minimum close by.

The robustness of our algorithm also depends on the Hessian matrix *J* in Eq. 1. The algorithm will not be able to find a solution when *J* is singular, i.e., there is no inverse of *J*. Different parametric representations for the same surface affects the robustness of the algorithm differently. Our experience is that robustness is the best if $\vec{f}_u$ and $\vec{f}_v$ of a surface are always perpendicular.

6. Conclusions and Future Work

The paper presents an efficient algorithm for tracking in real-time the minimum distance and the corresponding closest pairs of features and points between some general types of curved objects with parametric representations. The algorithm uses a top-down approach to organize object features, adopts the efficient Newton's approach for distance minimization, and takes advantage of the small change in the relative location between two objects in two consecutive time frames to provide good initial estimates for distance minimization. The algorithm achieves real-time speed with good accuracy and robustness.

As the next step of our work, we will further investigate the extension of the algorithm so that it can be applied to even more general curved objects. We will also extend the algorithm to compute penetration distances between objects. Currently the search process (for maximum penetration distance) may stop at the intersection points between features, but we expect to solve the problem soon. Other issues such as further increasing the efficiency of the algorithm and the stability of the Newton's method are also in our list for improving our algorithm.

Acknowledgement

This project is partially supported by the National Science Foundation grant IIS-9700412. The authors also thank Qi Luo for producing the video.

References

[1] Jimenez, P., Thomas, F., and Torras, C., "3D Collision Detection: A Survey," *Computers and Graphics*, 25(2): 269-285, 2000.

[2] Lin, M.C., Gottschalk, S., "Collision Detection between Geometric Models: A Survey," IMA Conference on Mathematics of Surfaces, Vol. 1, pp. 602-608, May 1998.

[3] Lin, M.C., and Canny, J.F. "A Fast Algorithm for Incremental Distance Calculation," *Proc. IEEE International Conf. Robotics & Automation (ICRA*), 1991, Sacramento, CA.

[4] Gilbert, E. G, Johnson, D.W., and Keerthi, S.A., "A Fast Procedure for computing the Distance between Complex Objects in Three-Dimensional Space", *IEEE Trans. Robotics and Automation*, 1998, 4(2): pp.193-203.

[5] Gilbert, E., Foo, C., "Computing the Distance between General Convex Objects in Three-Dimensional Space," *IEEE Transactions on Robotics and Automation*, 6(1):53-61, 1990.

[6] Turnbull, C., Cameron, S. "Computing Distances between NURBS-defined Convex Objects,'' *Proc. ICRA*, pp. 3686-3690, Leuven, Belgium, 1998.

[7] Johnson, D., Cohen, E., "Bound Coherence for Minimum Distance Computation," *Proc. ICRA*, pp. 1843-1848, Detroit, MI, May 1999.

[8] Patoglu, V., Gillespie, R. B., "Extremal Distance Maintenance for Parametric Curves and Surfaces," *Proc. ICRA*, pp. 2817-2823, Washington D.C., May 2002.