Abstract

Efficient collision detection is important in many robotic tasks, from high-level motion planning in a static environment to low-level reactive behavior in dynamic situations. Especially challenging are problems in which multiple robots are moving among multiple moving obstacles. There is extensive work on the collision detection problem in robotics as well as in other fields. Many of the successful approaches exploit the continuity or coherence of the motion to reduce the collision checking overhead. In this paper, we present a number of collision detection algorithms formulated under the Kinetic Data Structures (KDS) framework, a framework for designing and analyzing algorithms for objects in motion. The KDS framework leads to event-based algorithms that sample the state of different parts of the system only as often as necessary for the task at hand. Earlier work has demonstrated the theoretical efficiency of KDS algorithms. In this paper we present new algorithms and demonstrate their practical efficiency as well, by an implementation and direct comparison with classical broad and narrow phase collision detection techniques.

1 Introduction

Collision detection is an algorithmic problem arising in all areas of computer science dealing with the simulation of physical objects in motion. Examples include motion planning in robotics, virtual reality animations, computer-aided design and manufacturing, and computer games. Though a physical simulation involves several other computational tasks, such as motion dynamics integration, graphics rendering, and collision response, collision detection remains still one of the most time consuming in such a system. Often the collision detection problem is broken up into two parts, the so-called broad phase, in which we identify the pairs of objects we need to consider for possible collision, and the narrow phase in which we track the occurrence of collisions between a specific pair of objects [13, 16]. For the broad phase, almost all authors use some kind of simple bounding volumes for the objects themselves, or for portions of their trajectories in space or space-time, so as to quickly eliminate from consideration pairs of objects that cannot possibly collide. Many different kinds of bounding volumes have been suggested and tried, including axis-aligned or oriented bounding boxes, spheres, etc. [19, 6, 11, 16].

The narrow phase is more specialized, according to the types of objects being considered. The simplest objects to consider are convex polytopes, and this case has been extensively studied in the literature [15, 16, 18, 10]. More complex objects are then broken up into convex pieces, which are tested pairwise. Algorithmically, the convex polytope intersection problem is a special case of linear programming; in two and three dimensions even more efficient techniques have been developed in computational geometry, that can be applied after a suitable preprocessing of the two polytopes [7]. The methods, however, that have proven to work best in practice exploit the temporal coherence of the motion to avoid doing an ab initio intersection test at each time step. Not surprisingly, the collision detection problem is closely related to the distance computation problem. Since the distance between two continuously moving polytopes also changes continuously, many well-known collision detection algorithms, such as those of Lin and Canny [14, 15], Mirtich [16, 17], and Gilbert et al. [10] (see also [5]), are based upon tracking the closest pair of features of the polytopes during their motion. The efficiency of these algorithms is based on the fact that, in a small time step, the closest pair of features will not change, or will change to some nearby features on the polytopes.

Collision detection need not require exact distance maintenance, however. Furthermore collisions, whether they be of bounding volumes or of the objects themselves, tend to be very irregularly spaced over time and thus a good time step is hard to choose. This is exactly the setting in which event-based methods come in handy. We present below a new collision detection algorithm for multiple moving convex bodies, taking advantage of these intuitions. The broad phase of our algorithm uses spheres as the bounding volumes and maintains the dual power diagram [2] of the spheres as they move; this is guaranteed to contain an edge between the closest pair of spheres, which are the two that can collide next. For the narrow phase, we maintain separation certificates between two moving convex polytopes. These certificates are updated as necessary during the motion. These
are generalization of the kinetic certificates used by [9] in two-dimensions.

In order to take the advantage of the temporal coherence of continuous motion, we study the algorithms under the Kinetic Data Structures (or KDSs for short) framework, introduced in [4]. In the kinetic setting we assume that the instantaneous motion laws for our polytopes are known, though they can be changed at will by appropriately notifying the KDS. Our sampling of time is not fixed, but is determined by the failure of certain conditions, called certificates. Using the known motion laws, these certificate failure times are estimated and placed in an event queue. In the plainest form of a KDS, the certificates being maintained prove (in the strict mathematical sense) that no collision has occurred. In our narrow phase implementation, for example, these are separation certificates, proving that the two polytopes are separated by a plane. The failure of a separation certificate need not mean that a collision has occurred; it can simply mean that the failed certificate has to be replaced by one or more others, still proving the non-intersection of the polytopes. In general, at a kinetic event corresponding to a certificate failure, the certificate set being maintained had to be repaired, and in the process the attribute computation possibly updated as well. A good KDS will choose a certificate set that can be repaired locally and efficiently when one of its members fails. This is possible because a KDS exploits continuity or coherence of the motion to obtain 'continuity' of the proof.

A good KDS is called compact if it requires little space, responsive if it can be updated quickly after a certificate failure, local if it adjusts easily to changes in the motion plans of the objects, and efficient if the total number of events is small. Our kinetic collision-detection data structures have all these desirable properties; they maintain only a small constant number of certificates per object, and the cost for processing a certificate failure or a motion plan update is small. Most importantly, a KDS-based approach allows different parts of the system to be sampled only at the rate necessary for the motions in each part. For example, a fast moving bullet need not force a fine time step in parts of the environment away from the bullet’s path.

In Section 2, we first introduce our separation maintenance algorithm for two convex polytopes. In Section 3, we describe our power diagram maintenance algorithm for a set of moving balls. Those two algorithms together give us a kinetic collision detection algorithm for a set of moving convex bodies. The following Section 4 discusses the maintenance of the kinetic event queue and our methods for calculating event times. Section 5 presents the results of our experiments on those kinetic collision detection algorithms and compares kinetic event-based scheduling with fixed time-step methods.

2 Convex Polytopes

In this section we consider the problem of certifying the separation (equivalently, non-intersection) of two convex polytopes $P$ and $Q$ moving rigidly in 3D. Two non-intersecting convex polytopes always have a separating plane; such planes have been used previously in computer animation for collision detection [3]. It is also known that a separating plane can always be positioned so that it is either parallel to a facet of one of the polytopes, or parallel to a pair of edges, one from each polytope. If we make two copies of this separating plane and translate them towards $P$ and $Q$ respectively, till they each make contact, we will have one of the two situations depicted in Figure 1.

In both situations we end up with two parallel planes, one supporting $P$ and one supporting $Q$. The two planes partition space into three pieces, a halfspace containing $P$, the empty slab between the two planes, and a halfspace containing $Q$ — this is what we call the separation condition. In situation (a), the vertex-facet case, we have contact at a vertex $a$ of $P$ and a contact facet $bcd$ of $Q$ (this simplicity assumption can be easily removed), while in situation (b), the edge-edge case, we have contact with edge $ab$ of $P$ and edge $cd$ of $Q$. The separation condition is easily expressible in terms of the coordinates of the vertices of $P$ and $Q$ in contact with the two planes and their immediate neighbors.

We first assert that the plane supporting $P$ is above that supporting $Q$ — this is just a signed volume or orientation test on the tetrahedron $abcd$, in both situations (a) and (b). In terms of coordinates this is the determinant condition:

$$\left| \begin{array}{ccc}
    x_a & y_a & z_a \\
    x_b & y_b & z_b \\
    x_c & y_c & z_c \\
    x_d & y_d & z_d \\
\end{array} \right| > 0 .$$

![Figure 1](image-url)
We call this condition the *isolation certificate*. We also need to assert that the two planes support $P$ and $Q$ respectively, that $P$ is above the top plane and $Q$ below the bottom plane. Because $P$ and $Q$ are convex polytopes, it is sufficient to assert this condition locally for the vertices of $P$ and $Q$ that are neighbors of $a$, $b$, $c$, and $d$. We call these additional certificates, which can be expressed as orientation tests as well, the *support certificates*. There are four support certificates in the edge-edge case, while in the vertex-facet case the number of support certificates equals the degree of the vertex.

When polytopes $P$ and $Q$ move continuously as rigid objects, *as long as the isolation certificate and the relevant support certificates remain valid, $P$ and $Q$ cannot collide*. This is the motion coherence that our kinetic data structure will exploit. Note that we do not model a separating plane — the existence of such a plane is implied by the separation condition. Our most important task is to see how we can repair the separation condition when one of its certificates fails.

A failure of the isolation certificate is called a *push event* — corresponding to the determinant above becoming zero. We address the vertex-facet and edge-edge situations separately. Figure 2 (a) shows the situation when vertex $a$ of $P$ has become coplanar with facet $bcd$ of $Q$. When this event occurs we need to check if $a$ is inside the triangle $bcd$; if it is, $P$ and $Q$ have collided and the collision response code has to be invoked. If $a$ is outside the triangle $bcd$ then the line supporting at least one side of $bcd$, say $cd$, separates $a$ from the triangle. Now imagine a line through $a$ parallel to $cd$ and suppose we start rotating the $abcd$ plane around that line so that it does not enter $Q$. The rotation will be stopped by encountering either a vertex $e$ of $P$ or a vertex $f$ of $Q$. In the former case, $e$ must be a neighbor of $a$ and we can now use $ae$ and $cd$ as two edges giving us a new edge-edge isolation certificate. The latter case is symmetric. Figure 2 (b) shows the situation when the edge $ab$ of $P$ has become coplanar with edge $cd$ of $Q$. in this case as well, we can either report a collision or find a new isolation certificate by checking the features adjacent to $ab$ and $cd$.

We call a failure of a support certificate a *roll event*. The situation when a vertex support certificate fails is especially simple and is shown in Figure 3 (a): a neighbor $e$ of vertex $a$ of $P$ is now on the plane through $a$ and parallel to facet $bcd$. We need to replace the old $a$ against $bcd$ vertex-facet isolation certificate by one of $e$ against $bcd$, delete all the support certificates for $a$ and add the ones for $e$. An edge support certificate for edge $ab$ of $P$ fails when one of the facets adjacent to the edge, say $abe$, becomes parallel to the edge $cd$ of $Q$ — see Figure 3 (b). In this case the old edge-edge isolation certificate between $ab$ and $cd$ will be replaced by either $ae$ against $cd$ or $be$ against $cd$. The choice is dictated by whether the direction of $cd$ is further from that of $ae$ or $be$.

This completes our discussion of how the separation condition is to be maintained as the polytopes move. We note that, unlike algorithms, such as Lin-Canny [15], in which the closest pair of features of the polytopes is maintained, no preprocessing (such as the computation of a Voronoi diagram) whatsoever is required by our algorithm. Whenever one of our certificates fails, either a collision has actually occurred, or a straightforward local test repairs the separation condition so that the simulation can go on.

## 3 Power Diagram for Balls

Balls are among the most extensively used primitives in geometric modeling; for example, in several collision detection or ray tracing packages, objects are surrounded by bounding spheres so that preliminary intersection tests can be quickly performed. In this section, we describe a method to maintain the power diagram of a set of moving spheres.
and utilize this structure in performing the broad phase of our collision detection.

Power diagrams [2] are a generalization of Voronoi diagrams in which the sites defining the diagram are not points but balls. They derive their name from the fact that the distance used in their definition is not the standard Euclidean distance, but instead the classical notion of the power of a point with respect to a ball. In the following, we denote \( B(o, r) \) the ball centered at the point \( o \) and with radius \( r \). For a point \( p \) and ball \( B(o, r) \), the power distance \( \delta(p, B) \) is defined to be \( |op|^2 - r^2 \). For a set of balls \( \mathcal{B} \), the power diagram cell \( \mathcal{V}(B) \) of \( B \in \mathcal{B} \) then consists of all the points that are ‘closer’ to \( B \) than to any other balls in \( \mathcal{B} \), measured using the power distance. It is easy to verify that the bisector between two balls is a plane. Thus, each power diagram cell is a (possibly unbounded) convex polytopes. This property makes it easier to represent and manipulate power diagrams than the standard Voronoi diagrams for balls under the Euclidean distance.

The power diagram is useful as a proximity structure. In [12], it is shown that among a set of disjoint balls, the closest pair are always neighbors in the power diagram. Thus, if we can maintain the power diagram for disjoint balls, we are then able to maintain their closest pair as well. By tracking that pair, we can conveniently detect collisions. When balls may penetrate each other, we can then maintain the connected components of the moving balls and use them to filter the pairs needed to be passed to the narrow phase.

For the convenience of maintenance, we consider the dual simplicial complex, called the power complex, of the power program. For balls in general position, the power complex is a triangulation of the convex hull of the ball centers. For equal sized balls in two dimensions, the power complex is just the same as the well-known Delaunay triangulation of their centers. The crucial property is that the power diagram cells of two balls are adjacent if and only if there exists an edge between them in the power complex. In the following, we will first review the method for maintaining the Delaunay triangulation (in two dimensions) and then extend it to power diagrams in any dimension [1].

The \texttt{InCircle} test is the most important primitive used in the computation of Delaunay triangulations. For four points \( a, b, c, d \) in the plane, \texttt{InCircle}(\( a, b, c, d \)) is true if \( a \) is inside the oriented circumcircle of \( \triangle bcd \). Suppose that \( \triangle abc \) and \( \triangle bad \) are two triangles in a triangulation \( \mathcal{T} \). Then, the edge \( ab \) is said to be locally Delaunay if the test \texttt{InCircle}(\( a, b, c, d \)) is false. It is known that the Delaunay triangulation admits a local certification, i.e. \( \mathcal{T} \) is the Delaunay triangulation if each edge \( ab \) in \( \mathcal{T} \) is locally Delaunay. Such local certification makes it very easy to maintain the Delaunay triangulation as we can simply certify the local Delaunay-hood of each edge and fix the triangulation locally whenever such a certificate fails. The \texttt{InCircle} certificate fails when the points \( a, b, c, d \) become co-circular. A flip operation is then invoked to fix the triangulation, where the flip is to delete the edge \( ab \) and insert the edge \( cd \), or equivalently, to delete \( \triangle abc \) and \( \triangle bad \) and add \( \triangle dca \) and \( \triangle cdb \) (Figure 4(a)).

The above procedure extends to higher dimensions where the \texttt{InCircle} test is replaced by the \texttt{InSphere} test. But the flipping operation becomes more subtle. In three dimensions, when five points become co-spherical, there are exactly two ways to triangulate those five points. The local flip operation is then to replace a triangulation of five co-spherical points with its dual triangulation (Figure 4(b)). This property actually holds in any dimension, i.e. in \( d \) dimensions, there are exactly two ways to triangulate \( d + 2 \) points in convex position.

The power complex can be maintained in similar way to the Delaunay triangulation. First, we need to replace the \texttt{InCircle} tests by the \texttt{InPower} tests. In three dimension, we say that a ball \( B(a, r_a) \) is inside the power sphere of balls \( B(b, r_b), B(c, r_c), B(d, r_d) \) and \( B(e, r_e) \), if the following determinant condition holds.

\[
\begin{vmatrix}
  x_a & y_a & z_a & x_b^2 + y_b^2 + z_b^2 - r_b^2 & 1 \\
  x_b & y_b & z_b & x_c^2 + y_c^2 + z_c^2 - r_c^2 & 1 \\
  x_c & y_c & z_c & x_d^2 + y_d^2 + z_d^2 - r_d^2 & 1 \\
  x_d & y_d & z_d & x_e^2 + y_e^2 + z_e^2 - r_e^2 & 1 \\
  x_e & y_e & z_e & 1 & 1 \\
\end{vmatrix} < 0.
\]

For a set of disjoint balls, the same local property holds for power complexes. Thus, the power complex can be certified by asserting that each facet in the complex satisfies the local \texttt{InPower} test with respect to its two adjacent vertices. An important difference between power complexes and Delaunay triangulations is that a ball might not present in the power complex. For this to happen, it must be the case that the ball is completely contained in the union of four other balls. In practice, such situations only happen rarely. Thus, we can treat them separately. In our method, we track the balls that cover each absent ball and prevent it from presenting in the power complex. Such tracking can be better.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{flip.pdf}
\caption{Flip events (a) in two dimensions and (b) in three dimensions.}
\end{figure}
understood by a lifting map $L$, which we do not discuss here.

By maintaining the power complex, we can maintain the closest pair of balls for a set of disjoint balls, or the connected components of the balls when they can penetrate each other. This gives us a way to perform broad phase collision detection as we only need to invoke our narrow phase test for those objects with overlapping bounding spheres. A reason that we choose the power diagram as our broad phase structure is that it typically has linearly many edges and undergoes sub-quadratically many changes for algebraic motions (although, in the worst case, the number of edges can be quadratic, and the number of changes might be cubic or higher). Actually, the theoretical bound on the number of changes of the power complex is still a famous open question. Instead of performing theoretical analysis, we study those complexity measures by experiments and compare them to classical bounding-box methods in Section 5.1.

4 Scheduling Kinetic Events

The main loop of a kinetic data structure consists of detecting and processing kinetic certificate failures. These pending kinetic events can be organized in a number of ways, of which the simplest is to just maintain them in a global priority queue. Each active kinetic certificate schedules an event in the queue, which its earliest predicted failure time larger than the current time. When a certificate fails, the proof, and with it the associated certificate set as well as the event queue, needs to be updated. The calculation of an event time is frequently a non-trivial computational task. In the example certificates of the previous section, the certificate itself is an algebraic inequality on the coordinates of a few feature points (vertices) of the objects. Yet even in the simplest case of ballistic rigid motions for the objects with no outside forces (such as gravity), vertex trajectories are still polynomial mixtures of algebraic and trigonometric functions of time. The evaluation of the roots of algebraic forms on these transcendental functions is a challenging numerical problem. Furthermore, the cost of such computations cannot always be justified in terms of the final result that needs to be computed. For example, almost all kinetic simulations involve the de-scheduling of events — these are events that will not happen because the associated certificates were removed from the proof, and the computational resources that went into their event time calculation will be wasted.

We address this problem by the use of conservative approximations to the certificate failure times. We exploit the geometric structure of our certificates to get easier to compute approximate failure times that are guaranteed to be less than or equal to the actual failure time (this is what we mean by a conservative approximation). We call these approximations reconfirmation times. Each certificate then schedules a reconfirmation time in the event queue. When that reconfirmation time is reached, the certificate is evaluated again, a new reconfirmation time is computed, and the certificate is rescheduled. This continues until we are close enough to the actual failure time by some criterion. In the simplest case we just advance to the next frame time for the simulation when the reconfirmation interval is less than the interframe interval. In more elaborate cases, since we are by now very close to the actual failure time, we switch to an analytic approximation and estimate that time more precisely. We have experimented with polynomial approximations to the trigonometric functions arising in our motion plans and obtained good results. In both cases an event may be processed at a time slightly different from the actual time when it occurs; this can cause certain difficulties that we have to address. Overall, however, this method has many advantages because a few easy conservative steps get us quite close to the actual event time.

4.1 Computing Certificate Failure Times

To illustrate the use of reconfirmation events we focus on the convex polytope separation certificates presented in Section 2. We need to treat isolation and support certificates differently, as the latter depend only on the rotational and not the translational motion of the convex bodies. Let us look again in Figure 1, where we have two parallel planes, one supporting $P$ from below and the other $Q$ from above. Suppose this is snapshot taken at the time when the isolation certificate was first created. Let us imagine in addition an intermediate plane parallel to these two and half-way between them. For now imagine this intermediate plane as fixed in space, while $P$ and $Q$ continue their motion from the snapshot shown. It is clear that as long as none of the vertices $a$, $b$, $c$, and $d$ involved in the isolation certificate cross this plane, the isolation certificate cannot fail. If we have upper bounds on the translational and rotational velocities of $P$ and $Q$, we can use them to easily derive upper bounds on the speeds of the four vertices in the certificate. If $D$ denotes the separation of the two parallel planes at the snapshot in Figure 1 and $v_a$, $v_b$, $v_c$, and $v_d$ the maximum speeds of $a$, $b$, $c$, and $d$ respectively, then a conservative bound on the isolation certificate failure time is

$$t = \min\left\{ \frac{D}{2v_a}, \frac{D}{2v_b}, \frac{D}{2v_c}, \frac{D}{2v_d} \right\}$$

for cases (a) and (b) respectively. Better bounds can be obtained by giving the intermediate plane some motion as well, say by averaging the linear speeds of the two bodies at the time of the snapshot, or by better balancing the position of the intermediate plane with respect to the relative speeds of the two bodies.

We note that when the bodies $P$ and $Q$ have no rotational motion, roll events do not occur and the support certificates cannot fail. Thus each support certificate is best considered in the relative frame of the body where the roll event happens.
Consider the case of vertex-facet roll event, as in Figure 3 (a), in the local framework of body \( P \). Let \( \theta \) be the angle between the edge \( ae \) and its projection on the plane supporting \( P \) at \( a \) when the support certificate was created. Though the bodies \( P \) and \( Q \) are rotating about different axes, we focus on what happens locally at vertex \( a \) and specifically around an axis through \( a \) on the support plane and perpendicular to \( ae \). If \( \omega_P \) and \( \omega_Q \) are upper bounds on the angular velocities of bodies \( P \) and \( Q \) respectively, then it is not hard to show that \( \theta/(|\omega_P| + |\omega_Q|) \) is a conservative bound on the time when \( e \) can roll onto the support plane. By taking the minimum of these times over all neighbor vertices of \( a \), a conservative bound for the roll event can be obtained. In the edge-edge case of Figure 3 (b), a conservative bound can be similarly derived.

4.2 Initialization, Composition, and Recovery

In our convex polytope collision detection system, the power diagram of bounding boxes for the polytopes is used for the broad phase, and the separation certificates between pairs of convex polytopes for the narrow phase. The broad phase passes a pair of polytopes to the narrow phase when the distance of the bounding balls drops below some specified fraction of the sum of their radii. For two disjoint balls the plane of points having equal power to the balls separates the balls, and \textit{a fortiori} their enclosing polytopes. This plane can be used to initialize the separation certificate between the polytopes — we omit the straightforward details. A polytope pair is dropped out of the narrow phase when the distance of their bounding balls exceeds some other larger specified fraction of the sum of their radii (so as to build some hysteresis into the system). Initialization for the narrow phase may also be reinvoked in mid-simulation, but due to lack of space we omit this discussion.

5 The Experiments

We have implemented our algorithms on a Pentium II PC using Microsoft Visual C++. We have also compared our 3D convex polytope methods to some of the state-of-the-art collision detection packages available today. Because of the clear distinction between the broad and narrow phases, we have conducted our comparisons for the two phases both jointly and separately. Animations based on our algorithms can be seen at

http://graphics.stanford.edu/~feng/KDS_clsn/

5.1 Broad Phase

We can always perform pairwise checking to detect all object pairs with overlapping bounding volumes. Such a brute-force method works well for small number of objects and requires no data structures, but its quadratic growth rate limits its scalability. To reduce the number of pairs being checked, many algorithms use axis-aligned bounding boxes and test the intersection of the projection of the objects in lower (one or two) dimensions \([6, 16]\). Only those pairs that have intersecting projections need to be checked further. We have implemented a method to maintain the sorted order of the projection of the boxes on each of the \(x\)-, \(y\)- and \(z\)-axes — thus detecting potential collisions among the boxes when the ordering in one projection changes.

As for the test data, we first generated a set of disjoint balls with variable radii randomly, with uniform distribution. These balls were used as the input data to test the power diagram methods. Then, for each ball, we created an axis-aligned box with the same position and velocity, and comparable size. These boxes form the input data to the bounding box methods. During the motion, objects bounce against each other when they collide. We then ran the algorithms on data sets ranging from 10 to 1000 objects for sufficiently long simulation times to obtain reliable average number of events and collisions per second. The data are summarized in Figure 5. In the three figures, the \(x\)-axis measures the number of objects in the data file, and the \(y\)-axis measures (a) the number of events per second, (b) the number of certificates in the structure, and (c) the number of collisions per second, respectively.

From Figure 5(a), we can see that the number of events when using the bounding box method exhibits a quadratic growth in the number of objects, while in the power diagram method it is significantly smaller and approximately linear. However, the time savings are not as significant as the reduction in events would indicate. In the power diagram method, we have to compute a determinant, which is a degree five polynomial for linear motions, and then solve the resulting equation by using numerical methods. Also, it takes more time to update the power diagram when an event happens. In our experiments, each event updates about 10 certificates on the average and for each certificate, it takes about \(10^3\) floating point operations to compute the failure time. Still the power diagram method becomes superior when there are more than a few hundred objects, as is the case in most uses of such simulations. Furthermore, in applications such as computational molecular biology, spheres are naturally the preferred bounding volumes.

Figure 5(b) shows that although the number of certificates used in the power diagram structure is larger than that in the bounding box method, both of them are linear as a function of the number of objects. Figure 5(c) shows the number of collisions (per second) are about the same in the two experimental models, which justifies our experimental setup. It is also clear from the data that in both cases, the number of collisions per second has a linear relationship to the number of objects for the random distributions tested.

The above experiments provide some insight on how our method compares to fixed time step methods as well. One difficulty in comparing event-based methods with fixed time-
step methods is the choice of the time step size. For a fixed time-step method to be completely precise, we would have to choose the time step according to the minimum inter-collision time (the time between two collisions) in order to allow all the collisions to be captured. While most applications do not require that amount of precision, a reasonable time step choice would still need to be at least proportional to the average inter-collision time. We then expect the cost of the fixed time-step method to be proportional to the number of collisions times the update cost of the structure at each step. In the methods given in [6, 16], the update cost is typically linear in the number of objects, if the motion of the objects is slow relative to the time-step size, and worse otherwise. Furthermore, as our experiments show (Figure 5(c)), the number of collisions per second is linear in the number of objects, and therefore the overall running time of the bounding box method using fixed time steps will be at least quadratic in the number of objects.

5.2 Narrow Phase

We have compared our separation based method to some of the well-known proximity based algorithms. The closest pair tracking methods presented in [15, 17] have been quite successful in exploiting the temporal coherence of motion. Our separation based method is closely related to these proximity methods, since both they and we maintain a pair of features, one from each polytope, that can be used to obtain a separating plane. In all methods, the most important quality measure is the number of times the pair of features used to obtain the separating plane changes during the simulation. Although the methods in [15, 17] do not track every change in the closest pair of features, the number of changes in the closest pair is a reasonable measure of the cost of these algorithms, because at each time step they perform a ‘local walk’ along the polytopes to the next closest pair, whose length can be expected to be comparable to the number of changes to that pair (were it to be tracked).

We have conducted experiments to compare the number of changes of the closest pair and the number of changes in the separation certificates maintained by our method. We ran data in two settings. In one set of data, the objects have the same initial velocity but different combinatorial sizes, ranging from 20 to 100 vertices. In the other set of data, the objects always have 40 vertices, but different velocities were used. We counted the number of events (i.e. number of feature pair changes for the closest pair method and number of separation certificate failures for the separation plane method) per collision for each method. The experimental results are summarized in Figure 6.

As can be seen from the data, the number of events in our method is consistently smaller than the number of events.
in the closest pair method. What is more interesting is that the rate of changes to the separating pair is very stable and mostly independent of the size and speed of the objects, while the rate of changes to the closest pair is significantly more variable and unstable. A possible explanation of this phenomenon is that the closest pair asserts a stronger geometric condition on the pair of features involved than our separation certificates do. This relative stability makes our kinetic method attractive for real-time environments.

On a Pentium II PC, with our unoptimized implementation we obtained the following running time statistics for tracking the separation plane between two convex bodies. For each data set, the initial velocities, aspect ratio and bounding sphere are the same but the complexity (number of vertices) differs. The results are summarized in Table 2. The simulation is for 40 objects with average size 3, average separation planes are tracked in the narrow phase. The total average time per frame is 0.386 seconds when a simple pairwise test is used for the broad phase and separation planes are tracked. The total average time per frame is 0.316 seconds when the power diagram vertices, bouncing around in a cube of size 60. The total average time per frame is 0.316 seconds when the power diagram vertices, bouncing around in a cube of size 60. The total average time per frame is .316 seconds when the power diagram vertices, bouncing around in a cube of size 60. The total average time per frame is .316 seconds when a simple pairwise bounding sphere test (n x n) is used in the broad phase while separation planes are tracked in the narrow phase.

Table 1. Separation running time for sample data sets.

<table>
<thead>
<tr>
<th>number of vertices</th>
<th>average time per frame (in milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>33.6</td>
</tr>
<tr>
<td>60</td>
<td>47.9</td>
</tr>
<tr>
<td>110</td>
<td>79.7</td>
</tr>
</tbody>
</table>

6 Conclusions

We have presented new results on the design and implementation of collision detection algorithms, using kinetic data structures. These structures lead to novel methods for exploiting temporal coherence in collision detection and related proximity problems. For the first time, kinetic data structures make it possible to rigorously compare alternate strategies of maintaining attributes of an evolving system in the manner that has been so successfully used in the analysis of algorithms. We expect to see many more KDS applications in robotics in the future.

References