Algorithmica (2005) 43: 195–210
DOI: 10.1007/s00453-005-1153-2

Algorithmica

Efficient Collision Detection among Moving Spheres with Unknown Trajectories[1]

Ho Kyung Kim,[2] Leonidas J. Guibas,[3] and Sung Yong Shin[4]

Abstract. Collision detection is critical for applications that demand a great deal of spatial interaction among objects. In such applications the trajectory of an object is often not known in advance either since a user is allowed to move an object at his/her will, or since an object moves under the rules that are hard to describe by exact mathematical formulas. In this paper we present a new algorithm that efficiently detects the collisions among moving spheres with unknown trajectories. We assume that the current position and velocity of every sphere can be probed at any time although its trajectory is unknown. We also assume that the magnitude of the acceleration of each sphere is bounded. Under these assumptions, we represent the bounding volume of the sphere as a moving sphere of variable radius, called a *time-varying bound*. Whenever the time-varying bounds of two spheres collide with each other, they are checked for actual collision. Exploiting these bounds, the previous event-driven approach for detecting the collisions among multiple moving spheres with ballistic trajectories is generalized for those with unknown trajectories. The proposed algorithm shows an interactive performance for thousands of moving spheres with unknown trajectories without any hardware help.

Key Words. Collision detection, Event-driven approach, Interactive applications, Computer animation, Computational geometry.

1. Introduction

1.1. *Motivation*. Collision detection is critical for applications that involve spatial interactions among objects, and various collision detection algorithms have been proposed. Many of these [1]–[8] assume that the motion of an object is expressed as a closed-form function of time. In such a case a collision can be found using the equations of motion.

Applications, such as computer games, crowd simulations [9], and particle systems [10]–[13], have received extensive attention in recent years. In these applications either a user is allowed to move an object at his/her will, or an object moves under the rules that are hard to describe by exact mathematical formulas. In either case the trajectory of the object is hard to obtain in advance. Moreover, these applications may handle a large number of objects to generate a dynamic, complex scene. To simulate interactions among multiple objects efficiently, the objects are often approximated by their bounding spheres [14]–[17] to produce a visually pleasing animation in a feasible amount of time.

[1] This research has been supported by Korea Science and Engineering Foundation.

[2] Samsung Electronics Co., Ltd, Seoul, Korea. hkkim.kim@gmail.com.
[3] Computer Science Department, Stanford University, Stanford, CA 94305, USA. guibas@cs.stanford.edu.
[4] Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Korea. syshin@jupiter.kaist.ac.kr.

Received October 2, 2000; revised December 1, 2004. Communicated by D. T. Lee.
Online publication June 24, 2005.

This paper presents an efficient algorithm that detects collisions among multiple moving spheres with unknown trajectories. We assume that the maximum magnitude of the acceleration of each sphere is known in advance. This is reasonable in practice since an object in the real world cannot move arbitrarily fast. We also assume that the current position and velocity of the sphere can be probed at any time although its trajectory is unknown. The actual motion of the sphere is handled by the motion integrator, whose inner workings are not our concern.

1.2. *Related Work.* For environments consisting of multiple moving objects, a brute-force method checks for the collision between each pair of objects at every time step. Thus, the number of collision tests increases quadratically in that of objects. There has been an abundance of work aimed at reducing the number of collision tests, that is, to check collisions only between objects which are close to each other or to invoke a collision test when a pair of objects is highly likely to collide [18].

In practice, space subdivisions and bounding volumes have been widely used to accelerate collision detection. In space subdivisions the space is divided into cells, and collisions are checked only between pairs of objects that are in the same or nearby cells [1], [19]–[21]. Bounding volumes, such as spheres [15], [16] and axis-aligned boxes [7], [8], [22], are used due to the simplicity in their shape and motion description. Hierarchical bounding volumes, such as OBB-trees [23], sphere-trees [15], [24], [25], and BV-trees [26], are employed to enclose objects tightly. Recently, James and Pai suggested BD-trees [27] based on sphere-trees to detect collisions among multiple objects of restricted deformation.

Collisions are usually checked at a sequence of fixed time steps [19], [22], [23], [27]–[29]. In order not to miss any collisions, either the number of time steps is increased, or objects are assumed to move slowly. There have been attempts to fix this problem by adaptively changing the frequency of collision checks: a collision is more frequently checked between a pair of objects which are highly likely to collide.

Mirtich [7], [8] presented an algorithm that handles polyhedral objects with ballistic trajectories. At each time step the algorithm calculates the three-dimensional axis-aligned bounding box for each object, which is guaranteed to contain the object during a time interval. With a hierarchical hash table [30], the algorithm chooses, as the candidates for collision tests, the pairs of objects whose bounding boxes intersect each other. The collision between a candidate pair is checked by the algorithm of Lin and Canny [31] that tracks the closest features between two convex polyhedra. Mirtich also proposed a scheduling scheme to check adaptively for collisions between candidate pairs.

Kim et al. [1] suggested an event-driven approach that efficiently detects collisions among ballistic spheres. To localize collision detection, a space subdivision scheme is adopted. This approach traces the spatial distribution of spheres and their trajectories so as to focus only on the sphere pairs that are highly likely to collide.

If the trajectories of the objects are explicitly known between collisions, the framework of the *Kinetic Data Structures* of Basch et al. [32] and Guibas [33] facilitates a number of collision detection algorithms based on the idea of updating the system state only at moments when certain critical conditions fail. The set of the conditions, called *certificates*, is maintained incrementally as the system evolves. They showed that no intersection occurs unless any of the certificates fail. For example, Guibas and Zhang

presented an event-driven approach for ballistic spheres [34], exploiting the power diagram [35] of the balls. For disjoint balls, the closest pair are adjacent in the power diagram, which enables collision detection. In practice, the authors found that the power diagram changes only linearly many times for ballistic spheres. Guibas et al. [36] also described a kinetic data structure for maintaining a compact Voronoi-like diagram of convex polygons moving around in the plane, and this diagram can be used for collision detection among the moving convex polygons.

With an assumption that the maximum norm of the acceleration of each object is provided, Hayward et al. [14] and Hubbard [15], [16] independently proposed algorithms to detect collisions among objects with unknown trajectories.

Hayward et al. [14] proposed a collision predictor for moving spheres, which flags a collision if two spheres are close enough to collide. For each pair of spheres, they calculated the amount of time within which those spheres are guaranteed not to collide with each other. Then the algorithm adaptively pays more attention to highly urgent pairs. The algorithm needs $O(\log n \log\log n)$ time at each step and $O(n^2)$ space to flag collisions among n spheres.

Hubbard [15], [16] suggested a two-phase collision detection algorithm for interactive applications. The first phase, called the broad phase, detects collisions among the bounding spheres of the objects. The broad phase builds a four-dimensional space–time bound that contains a bounding sphere during a given time interval and detects the intersections among space–time bounds by the Bentley–Ottmann algorithm [37]. The results are used to identify the colliding pairs of bounding spheres. In the narrow phase the pairs of objects of which the bounding spheres collide are checked for collisions using the sphere trees that represent the objects.

1.3. *Overview.* For a moving sphere with an unknown trajectory, we can compute the bounding volume of the sphere using our knowledge of the maximum magnitude of its acceleration together with its initial position and velocity. This volume, called the *time-varying bound*, is represented as a moving sphere of variable radius, that is guaranteed to contain the sphere at any time instance in the future. Initially, the bounding volume has the same center position and radius as its corresponding sphere. As time passes, the uncertainty of the position of the sphere increases because of its unknown trajectory, and thus the radius of the volume monotonically grows in time.

Suppose that a pair of spheres collide with each other at some time instance in the future. Then their bounding volumes will certainly intersect beforehand. Thus, whenever the bounding volumes collide with each other, we check for the actual collision between the corresponding spheres. If the spheres collide with each other, we compute their new positions and velocities with a proper collision response. Otherwise, we probe the new positions and velocities of the sphere. Regardless of the actual collision between the spheres, we reset the colliding volumes to be the same as their corresponding spheres to resume the collision check with those new volumes. Hence, the collision detection problem between two spheres is transformed into a sequence of collision detection problems between their time-varying bounds.

The collisions among multiple spheres may be detected by repeating the collision checks between two time-varying bounds for every pair of spheres. However, as the number of spheres increases, the collision checks for all pairs of time-varying bounds

would be extremely time-consuming. We wish to reduce the number of collision checks as much as possible by focusing only on the pairs of time-varying bounds that are highly likely to collide. To achieve this goal, the event-driven approach to collision detection among multiple moving spheres of fixed radii [1] is generalized for time-varying bounds, that is, moving spheres of variable radii, exploiting their compact representation. We identify a set of events for time-varying bounds and properly manipulate those events to detect collisions among time-varying bounds.

The remainder of this paper is organized as follows. Section 2 describes a notion of a time-varying bound, that is, a moving sphere of variable radius. In Section 3 we present an event-driven approach for detecting the collisions among multiple time-varying bounds. We shows experimental results in Section 4, and finally conclude the paper in Section 5.

2. Time-Varying Bound. For efficient collision detection, a bounding volume should not only be simple but also fit an object tightly. Hubbard defined a four-dimensional bounding volume, called the parabolic horn, that contains a sphere with an unknown trajectory for a time interval [15], [16]. Since the computational cost for finding the intersections among these horns is expensive, the author built a simpler four-dimensional bounding volume, called a hypertrapezoid, which encloses a horn for the time interval. This volume grows rapidly as its time interval increases and thus is cumbersome to handle. To detect collisions among multiple spheres at an interactive rate, we need a more compact bounding volume.

For a sphere with an unknown trajectory, we represent its bounding volume with another sphere that satisfies our general criteria for a bounding volume. Let $\mathbf{x}(t)$, $\mathbf{v}(t)$, and $\mathbf{a}(t)$ denote the position, velocity, and acceleration of a sphere at time t, respectively. By assumption, we can probe the position $\mathbf{x}(t^0)$ and velocity $\mathbf{v}(t^0)$ of the sphere at the current time t^0. Suppose that an upper bound of $\|\mathbf{a}(t)\|$ is given for all $t \geq t^0$, that is,

$$\|\mathbf{a}(t)\| \leq A\,, \tag{1}$$

for some positive constant A. Then

$$\|\mathbf{x}(t) - (\mathbf{x}(t^0) + \mathbf{v}(t^0)(t - t^0))\| \leq \frac{A}{2}(t - t^0)^2. \tag{2}$$

A detailed derivation is available in the Appendix. According to Taylor's theorem, inequality (2) holds true when the trajectory $\mathbf{x}(t)$ is C^2-continuous. The trajectory $\mathbf{x}(t)$ is C^2-continuous except at the singular points where the sphere interacts with other spheres or the environment. By carefully identifying those singular points, we can use this inequality to bound the sphere. Equation (2) states that the center $\mathbf{x}(t)$ of the sphere for $t \geq t^0$ is within a distance of $(A/2)(t-t^0)^2$ from the known position $\mathbf{x}(t^0)+\mathbf{v}(t^0)(t-t^0)$. Let the radius of the sphere be r. Then the sphere at time t is contained in a bounding sphere of radius $\hat{r}(t)$ centered at $\hat{\mathbf{x}}(t)$, where

$$\hat{r}(t) \;=\; \frac{A}{2}(t - t^0)^2 + r, \tag{3}$$

$$\hat{\mathbf{x}}(t) \;=\; \mathbf{x}(t^0) + \mathbf{v}(t^0)(t - t^0). \tag{4}$$

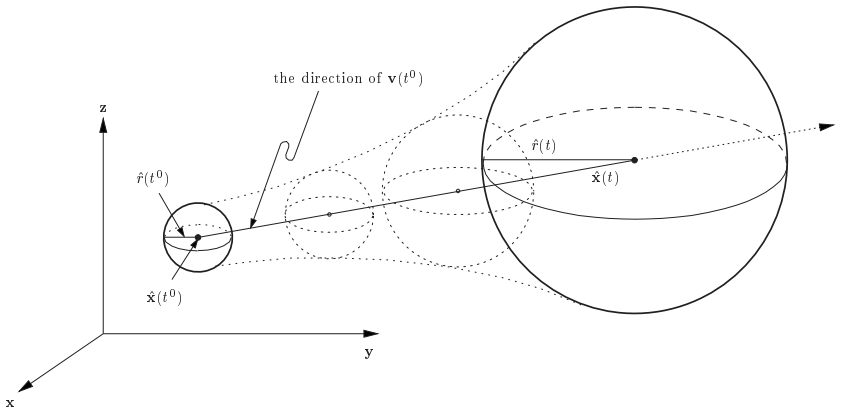


Fig. 1. A time-varying bound.

Equations (3) and (4) represent a parabolic horn, whose cross section at time t is the sphere of radius $\hat{r}(t)$ centered at $\hat{\mathbf{x}}(t)$ [15], [16]. This horn can be interpreted as the volume swept by a time-varying sphere of variable radius $\hat{r}(t)$ moving from $\mathbf{x}(t^0)$ to $\mathbf{x}(t^0) + \mathbf{v}(t^0)(t - t^0)$. This sphere is said to be a *time-varying bound*. Initially at time t^0, the bound has the same center $\mathbf{x}(t^0)$ and radius r as its corresponding sphere. After $(t - t^0)$ time, the bound grows to a sphere of radius $\hat{r}(t)$ centered at $\hat{\mathbf{x}}(t)$. As t increases, its radius quadratically increases and its center moves linearly in the direction of $\mathbf{v}(t^0)$ from $\mathbf{x}(t^0)$ (see Figure 1). Note that a ballistic sphere is a special case of a time-varying bound in which $\|\mathbf{a}(t)\| = A$ for all t.

The collision between two time-varying bounds is easily calculated because of their simplicity in shape and motion. If a pair of spheres collide with each other, then their time-varying bounds also collide beforehand. Hence, we perform a collision check between two spheres whenever their time-varying bounds collide with each other.

3. Collision Detection among Time-Varying Bounds.

To detect the collisions among multiple time-varying bounds, we generalize the event-driven approach [1] that efficiently handles ballistic spheres. The efficiency of the approach is retained in its generalization for time-varying bounds, due to their compact representation.

3.1. *Event-Driven Approach.*

Kim et al. [1] presented an event-driven approach that efficiently detects collisions among ballistic spheres. This approach subdivides the whole space containing the spheres into a set of uniform cubical subspaces to localize the collision checks. The edge length of a subspace is greater than or equal to twice, but less than some fixed times, as large as the diameter of the largest sphere. Due to the assumption that the diameter of the largest sphere is bounded by a constant multiple of that of the smallest, each non-empty subspace has a constant number of intersecting spheres. These subspaces are maintained in a *subspace tree*, which is a binary search tree of bounded balance [38]. A leaf node of the tree represents a non-empty subspace, and every non-empty subspace has a list of spheres intersecting it.

There are at most eight subspaces intersecting a sphere, and thus the number of non-empty subspaces intersecting one or more spheres is $O(n)$, where n is the number of the spheres. Search, insert, and delete operations on the subspace tree can be done in $O(\log n)$ time. Hence, it takes $O(n \log n)$ time initially to construct the subspace tree.

The event-driven approach keeps track of the trajectories of spheres and their spatial distribution by identifying three types of events: entering, leaving, and colliding. The first two types are caused by a sphere crossing subspaces. The third type is due to a collision between a pair of spheres intersecting the same subspace. The algorithm computes the candidate collision events of each sphere with the others intersecting the same subspace as well as its candidate entering and leaving events while allowing each sphere to penetrate the other spheres. Those candidate events of every sphere are maintained in another search tree, called an *event tree*. Among the candidates, the earliest event actually occurs since no penetrations between spheres have ever happened. Whenever an event occurs, the event tree is properly modified by adding and removing candidate events according to the event type to enable the next event. Accordingly, new non-empty subspaces are added to and previous non-empty subspaces that become empty are removed from the subspace tree. Repeating this process, all events can be detected in their time sequence.

Each sphere causes $O(1)$ candidate events, and thus the total number of candidate events is $O(n)$, which are maintained in the event tree. Since it takes $O(\log n)$ time to add a candidate to the tree, the event tree can be constructed in $O(n \log n)$ time with $O(n)$ space. Whenever an event occurs, a constant number of search, insert, or delete operations are performed on the subspace and event trees, and thus it takes $O(\log n)$ time to handle an event.

The event-driven approach also subdivided the space with a hierarchy of regular grids to accelerate collision detection. Starting from the finest grid of the hierarchy, the size of the subspace is doubled at each subsequent coarser grid. Each sphere is assigned to a proper grid according to its radius. Since every grid partitions the same space, two spheres in different grids may collide with each other. For a sphere s intersecting a subspace C, the algorithm identifies all subspaces intersecting C, regardless of their grids, to generate the candidate colliding events for s with the spheres intersecting those subspaces. Note that only one subspace tree and one event tree are maintained, and all non-empty subspaces belong to the same subspace tree, regardless of their grids, and the same is true for the event tree.

After $O(n \log n)$ time preprocessing to construct the subspace and event trees, it takes $O(\bar{n}_c \log n + \bar{n}_e \log n)$ time and $O(n)$ space to detect collisions among n ballistic spheres, where $\bar{n}_c$ and $\bar{n}_e$ are the number of actual collisions and that of entering and leaving events during the simulation, respectively. As $\bar{n}_e$ depends on the subspace sizes, the performance of the event-driven approach also depends on the subspace sizes. The subspace sizes are obtained by optimizing the cost model that estimates the running time.

3.2. *Events for Time-Varying Bounds*. Given a set of spheres with unknown trajectories, we find collisions among the spheres using their time-varying bounds. We identify four types of events, *entering*, *leaving*, *resetting*, and *colliding*, each of which occurs when the trajectories of time-varying bounds or their spatial distribution change. By tracing the events in the order of their occurring times, we can detect collisions among time-varying bounds.

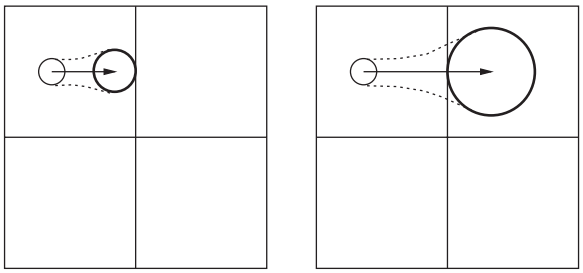


Fig. 2. An entering (a) and a leaving (b) event for a time-varying bound.

We first subdivide the whole space containing the spheres with a hierarchy of regular grids and assign every sphere to a proper grid. A collision between two spheres in different grids is detected by generating the candidate colliding events for their time-varying bounds in a similar way as given in [1]. Hence, in what follows we concentrate on collision detection of the time-varying bounds within one grid. Every non-empty subspace has a list of time-varying bounds intersecting it, and these subspaces are maintained in the subspace tree. The event tree maintains the candidate events for entering, leaving, resetting, and colliding of every time-varying bound.

Given a time-varying bound $B(s_i)$ of a sphere s_i lying in a subspace, an entering event occurs when $B(s_i)$ touches a face of the subspace to enter its adjacent subspace sharing the face (see Figure 2(a)). A leaving event occurs when $B(s_i)$ moves apart from the face being touched (see Figure 2(b)). These types of events change the list of time-varying bounds for each subspace. Whenever $B(s_i)$ enters a subspace, we add $B(s_i)$ to the list for the subspace. Accordingly, we compute the new candidate collision events of $B(s_i)$ with the other bounds intersecting the subspace to add them in the event tree. We also compute the candidate events of $B(s_i)$ for entering and leaving the subspace. We handle a leaving event in a symmetrical way.

Initially, $B(s_i)$ is identical to s_i. As time passes, $B(s_i)$ grows and may intersect many subspaces. Accordingly, the candidate events for $B(s_i)$ would increase excessively to degrade the efficiency. We circumvent this problem by restricting the diameter of $B(s_i)$ from being larger than the side length of a subspace. This restriction forces a time-varying bound to intersect at most eight subspaces, and thus the total number of non-empty subspaces is $O(n)$, where n is the number of time-varying bounds. Since a subspace intersects a constant number of original spheres, we can easily show that the number of time-varying bounds intersecting a subspace is also constant.

Whenever the diameter of $B(s_i)$ reaches the side length of a subspace, a resetting event occurs (see Figure 3(a)). Then we probe both the current position $\mathbf{x}_i(t)$ and velocity $\mathbf{v}_i(t)$ of s_i to reset $B(s_i)$ to be the same as s_i. $B(s_i)$ may intersect different subspaces by resetting $B(s_i)$ (see Figure 3(b)). Hence, we remove $B(s_i)$ from the list of time-varying bounds of each subspace that does not intersect $B(s_i)$ anymore because of resetting. The position $\mathbf{x}_i(t_i^0)$ and velocity $\mathbf{v}_i(t_i^0)$ are also renewed (see Figure 3(b)). Hence, we recompute the candidate collision events of $B(s_i)$ with the other bounds intersecting the same subspaces as well as its candidate entering and leaving events. We also calculate

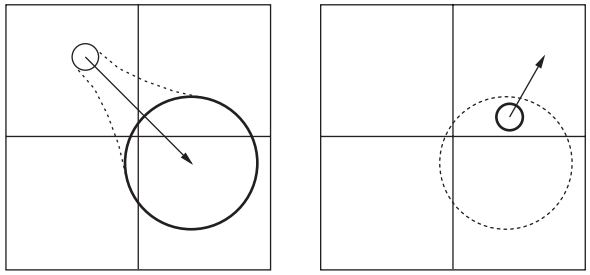


Fig. 3. (a) A resetting event for a time-varying bound and (b) after resetting the time-varying bound.

its next resetting event. These events are inserted into the event tree after the removal of old candidate events of $B(s_i)$.

A colliding event takes place when $B(s_i)$ collides with another time-varying bound $B(s_j)$ intersecting the same subspace (see Figure 4). Whenever a colliding event between $B(s_i)$ and $B(s_j)$ occurs, we check for the actual collision between spheres s_i and s_j. Regardless of the collision, we reset both $B(s_i)$ and $B(s_j)$ in the same way as on a resetting event to resume the collision check.

Now, we analyze the time needed to detect all collisions among the spheres. Since every time-varying bound intersects at most eight subspaces, the number of candidate entering and leaving events of each $B(s_i)$ is $O(1)$, which are maintained in the event tree. The number of its candidate colliding events is also $O(1)$ since each subspace intersects a constant number of time-varying bounds. In addition, each bound has one resetting event. Hence, $B(s_i)$ has a constant number of candidate events, and thus the total number of candidate events for the time-varying bounds in the event tree is $O(n)$.

It takes $O(\log n)$ time to add (delete) a candidate event to (from) the event tree. It also takes $O(\log n)$ time to update a non-empty subspace in the subspace tree. Since every actual event causes a constant number of candidate events to update, it takes $O(\log n)$ time to handle an event regardless of its type. Let $\bar{n}_t$ be the total number of actual entering, leaving, and resetting events of the time-varying bounds for a given time interval, and let $\bar{n}_c$ be that of actual colliding events among the bounds in the same time interval. Then we can detect the collisions among the spheres during the time interval in $O(\bar{n}_t \log n + \bar{n}_c \log n)$ time.

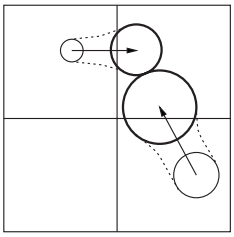


Fig. 4. A colliding event for two time-varying bounds.

3.3. *Efficiency Improvement.* The candidate events of a time-varying bound $B(s_i)$ are removed from the event tree upon its leaving, resetting, or colliding events. We can improve the efficiency of collision detection by reducing the number of such removal operations. For this purpose, we do not insert into the event tree the candidate events of $B(s_i)$ whose event times are later than that of its resetting event. Otherwise, we would remove those events anyway to recalculate them upon the resetting event. Hence, we maintain in the event tree only the candidate events of a time-varying bound, whose event times are earlier than its resetting event time. This also reduces the memory requirements for the event tree.

4. Experimental Results. To demonstrate the efficiency of our algorithm, we performed experiments on IRIS Indigo 2 (CPU: MIPS R10000, 195 MHz) to detect collisions among spheres with unknown trajectories moving in a given box. For collision response, we adopted an elastic collision model [39]. We first simulated a situation in which a user moves the spheres by randomly generating forces over the simulation. The magnitude of the force exerted on a sphere was chosen so that the maximum norm of the acceleration of the sphere is kept within a given bound. The algorithm does not use those forces for collision detection. In the second experiment we applied our algorithm to a particle system, where spheres move driven by velocity fields.

4.1. *Event Time Calculation.* In this section we describe how to predict an event. Suppose that we reset a time-varying bound $B(s_i)$ at time t_i, and the position and velocity of s_i at t_i^0 are $\mathbf{x}_i(t_i^0)$ and $\mathbf{v}_i(t_i^0)$, respectively. Then the position $\hat{\mathbf{x}}_i(t)$ and radius $\hat{r}_i(t)$ of $B(s_i)$ at time t, $t \geq t_i^0$, are as follows:

$$\hat{r}_i(t) = \frac{A_i}{2}(t - t_i^0)^2 + r_i, \tag{5}$$

$$\hat{\mathbf{x}}_i(t) = \mathbf{x}_i(t_i^0) + \mathbf{v}_i(t_i^0)(t - t_i^0), \tag{6}$$

where A_i and r_i are the maximum magnitude of the acceleration and radius of s_i, respectively.

Entering and leaving events occur when $B(s_i)$ touches a face of a subspace. Let $\mathbf{n}_\mathrm{f}$ be the unit normal vector of a plane containing the face and let $\mathbf{p}_\mathrm{f}$ be a point on that plane. Then we predict the touching time by solving

$$|(\hat{\mathbf{x}}_i(t) - \mathbf{p}_\mathrm{f}) \cdot \mathbf{n}_\mathrm{f}| = \hat{r}_i(t). \tag{7}$$

Let L_i denote the side length of the subspace in which $B(s_i)$ is assigned. Then the resetting event of $B(s_i)$ occurs when its diameter grows to L_i. We solve $2\hat{r}_i(t) = L_i$ to predict the resetting time $t = t_i^0 + \Delta t$, where

$$\Delta t = \sqrt{\frac{L_i - 2r_i}{A_i}}. \tag{8}$$

A collision time between two time-varying bounds, $B(s_i)$ and $B(s_j)$, is the solution of

$$\|\hat{\mathbf{x}}_i(t) - \hat{\mathbf{x}}_j(t)\| = \hat{r}_i(t) + \hat{r}_j(t). \tag{9}$$

For simplicity, we represent the collision time t as $t^0 + \delta_t$, that is, $t = t^0 + \delta_t$, where t^0 is the current time and $\delta_t > 0$ is the time elapsed after t^0. To predict the collision time t, we need to compute δ_t. Squaring and rearranging (9) yields

$$(10) \qquad -\frac{a^2}{4}\delta_t^4 - ab\delta_t^3 + (\|\mathbf{v}_r\|^2 - ar - b^2)\delta_t^2 + 2(\hat{\mathbf{x}}_r \cdot \mathbf{v}_r - br)\delta_t + \|\hat{\mathbf{x}}_r\|^2 - r^2 = 0,$$

where $a = A_i + A_j, b = A_i(t^0 - t_i^0) + A_j(t^0 - t_j^0), r = \hat{r}_i(t^0) + \hat{r}_j(t^0), \hat{\mathbf{x}}_r = \hat{\mathbf{x}}_i(t^0) - \hat{\mathbf{x}}_j(t^0)$, and $\mathbf{v}_r = \mathbf{v}_i(t_i^0) - \mathbf{v}_j(t_j^0)$. In theory the closed-form formula for the roots of a quartic equation can be used to compute δ_t in a constant time.

To keep a sphere within the box, we have to bounce it back into the box when it collides with a face of the box. We check this collision when its time-varying bound collides with the face. We regard such a collision as another type of colliding event and compute the collision time in a similar way to that of an entering or a leaving event.

4.2. *Performance*. We measured the actual processing times for collision detection among spheres moving in a box of $200 \times 200 \times 200$. Radii of the spheres were chosen between 0.1 and 10.0. Their initial speeds were uniformly distributed between $2.5\sqrt{3}$ and $25\sqrt{3}$, and the maximum norms of acceleration were between 10.0 and 20.0. We performed 15-second simulations, varying the number of spheres from 500 to 3500. In Figure 5 we plot average processing times to simulate 1-second evolutions of moving spheres in a real world clock. It took 1.372 seconds for 1000 spheres in our computer clock, which shows that our algorithm performs the simulation of 1000 moving spheres at a speed almost comparable with the actual speed in the real world. Figure 6 shows the number of actual events that occur during the simulations. The numbers of entering, leaving, and resetting events are bounded by the number of colliding events.

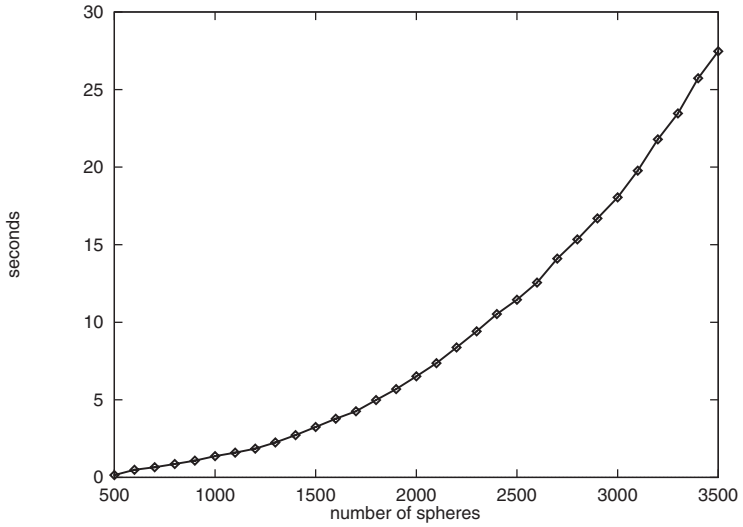


Fig. 5. Average processing time.

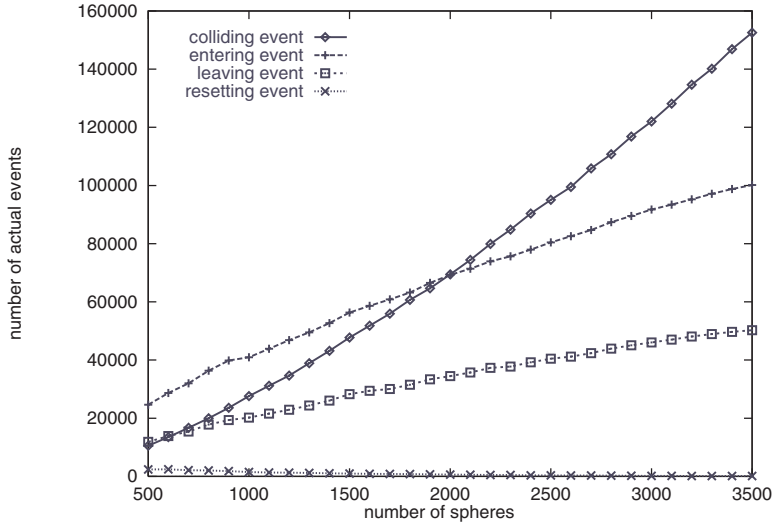


Fig. 6. The number of actual events.

We also compared our algorithm with the broad phase of Hubbard's algorithm [15], [16]. As explained in Section 1.2, the broad phase detects collisions among objects with unknown trajectories using their space–time bounds, which are four-dimensional hypertrapezoids. We modified the broad phase so that it can also detect collisions between spheres and box faces. We chose the radius, initial speed, and maximum norm of acceleration of each sphere as in the previous experiment. The volume of the box is also 200^3. We ran 50 simulations for each of 30, 100, 300, and 1000 spheres.

Table 1 shows that our algorithm outperforms Hubbard's. This is mainly due to the following reasons. First, a time-varying bound is more compact than a space–time bound. Hence, space–time bounds collide with each other more frequently than time-varying bounds. Second, the broad phase rebuilds *all* space–time bounds upon each collision to reinitiate the next collision check. For each α axis, $\alpha \in \{\mathbf{x}, \mathbf{y}, \mathbf{z}\}$, a space–time bound has a pair of faces that are normal to the α axis. The broad phase projects the pair of faces of each space–time bound onto the two-dimensional αt-plane, where t represents the time dimension. This results in three sets of line segments, each of which is in the αt-plane. To obtain the collision time, the broad phase finds intersections for each set

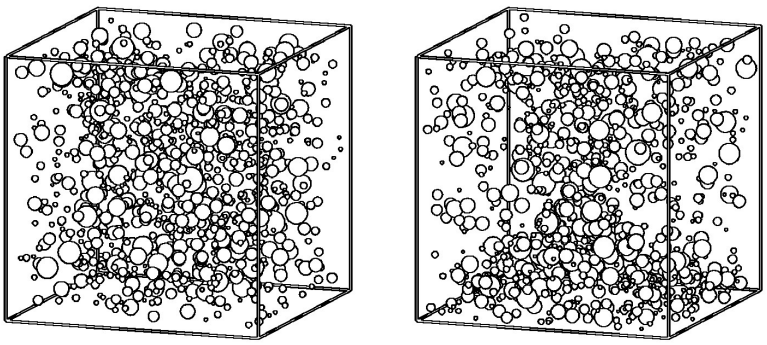
Table 1. The processing time of our algorithm and Hubbard's broad phase.

	Average processing times (in seconds)		
Number of spheres	The proposed algorithm	Hubbard's broad phase	Ratio
30	0.00576	0.94064	1 : 163.3056
100	0.02776	13.36050	1 : 481.2860
300	0.15844	150.28884	1 : 948.5537
1000	1.37200	2767.82990	1 : 2017.3687

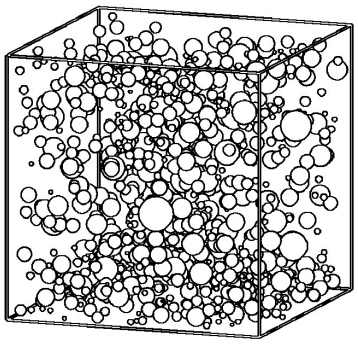
of line segments using the Bentley–Ottmann algorithm [40], [41]. This algorithm takes $O((n+k)\log n)$ time, where n is the number of line segments and k is the number of intersections among the segments. For accuracy, our algorithm computes each collision time by numerically solving the quartic equation in (10), although there is a closed-form formula.

We effectively detected collisions among the spheres, using their time-varying bounds. For efficiency, we localized collision detection by keeping track of the spatial distribution of the time-varying bounds over the subspaces. It takes $O(\log n)$ time to handle an event regardless of its type. As stated in Section 3.2, the performance of our algorithm depends on the number of events which is a function of the subspace sizes. We adapted the cost model in [1] to determine the subspace sizes.

4.3. *Example*. We applied our algorithm to detect collisions among particles circulating in a box due to velocity fields [42]. The particles moved upward along the line connecting two center points on each of the top and bottom faces of the box, and they moved downward along the side faces. Figure 7 exhibits some snapshots taken during



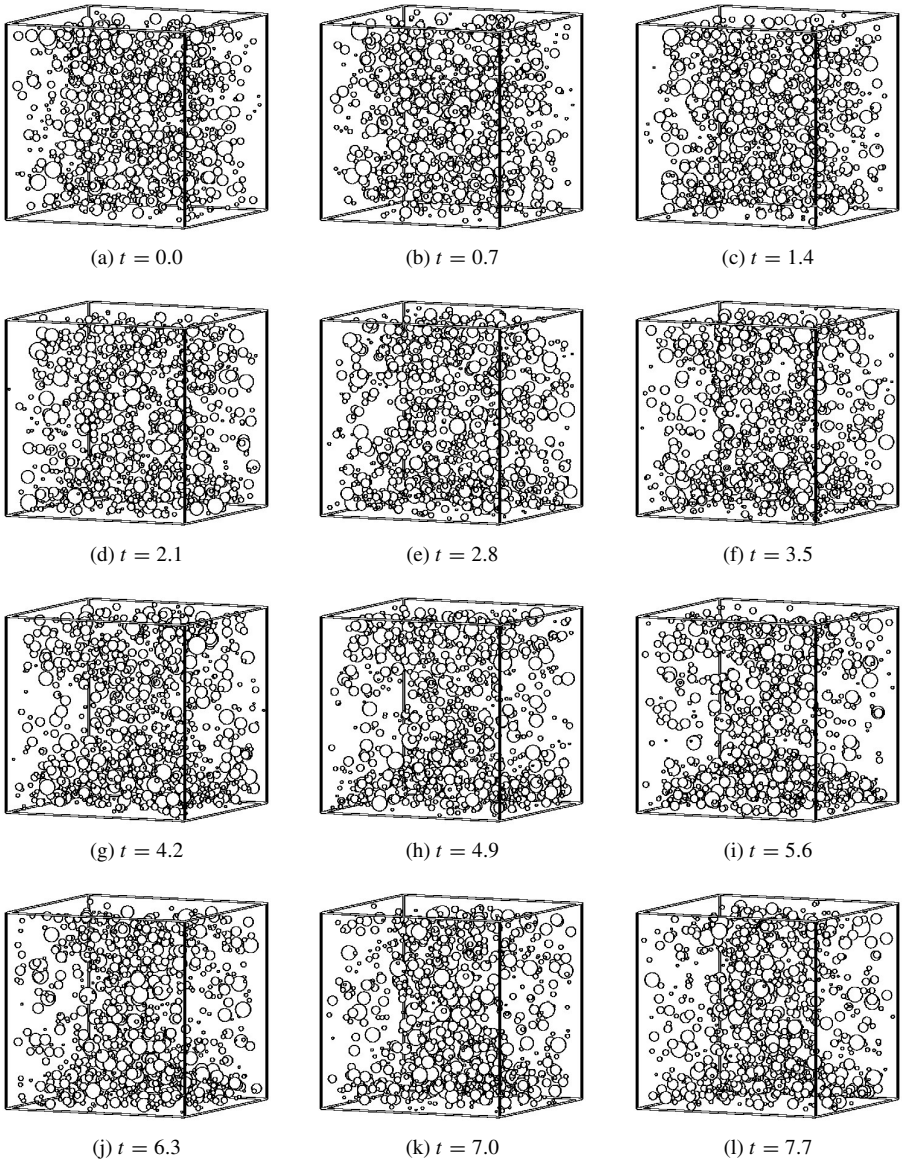
(a) Initial particles (b) Moving particles



(c) Time-varying bounds of the particles in (b)

Fig. 7. Circulating particles in a box.

a simulation of 1000 particles. The radius of a particle was chosen between 1.0 and 10.0, and the maximum norm of acceleration was 30.0. The volume of the box was 200^3. Figure 7(a) is the initial state of the particles in the box. Figure 7(b) shows the moving particles, and their time-varying bounds at the same time instance are shown in Figure 7(c). As expected, the time-varying bounds in Figure 7(c) are larger than their corresponding spheres in Figure 7(b). Figure 8 shows a 7-second simulation of particles.



(a) $t = 0.0$ (b) $t = 0.7$ (c) $t = 1.4$

(d) $t = 2.1$ (e) $t = 2.8$ (f) $t = 3.5$

(g) $t = 4.2$ (h) $t = 4.9$ (i) $t = 5.6$

(j) $t = 6.3$ (k) $t = 7.0$ (l) $t = 7.7$

Fig. 8. Simulation for 7 seconds.

Initially, the particles were randomly placed in a box. As time passed, they gathered around the center line of the box and spread out on the top face due to velocity fields. Then they circulated in the box. It took 2.846 seconds on average to perform 1-second simulation of 1000 circulating particles. That is, our algorithm showed a near real-time performance to detect collisions among 1000 particles.

5. Conclusions. In this paper we have presented a new algorithm that efficiently detects collisions among multiple moving spheres with unknown trajectories. To trace the movement of a sphere, we represent its time-varying bound as a moving sphere of variable radius, which is guaranteed to contain the sphere. Due to the compact representation of our time-varying bounds, we can generalize the event-driven approach [1] for collision detection among multiple such bounds. Since a ballistic sphere is a special case of a time-varying bound, our algorithm can handle not only spheres with unknown trajectories but also ballistic spheres. Our experimental results show that the proposed algorithm simulates several thousands of moving spheres with unknown trajectories at an interactive rate. For future work, we intend to apply the algorithm to various applications such as crowd simulations and interactive applications.

Acknowledgement. We thank Prof. Hong Oh Kim at the Applied Mathematics Department of KAIST (Korea Advanced Institute of Science and Technology) for his help in deriving (2) as given in the Appendix.

Appendix. Set $F(s) = u \cdot X(s)$, where $X(s)$ is the position of an object at time s. By *Taylor's theorem*,

$$F(s) = F(t_0) + \frac{F'(t_0)}{1!}(s - t_0) + \frac{F''(m(s))}{2!}(s - t_0)^2$$

for some $t_0 < m(s) < s$. Note that

$$F'(s) = u \cdot \dot{X}(s) \quad \text{and} \quad F''(s) = u \cdot \ddot{X}(s).$$

Evaluating $F(s)$ at $s = t > t_0$,

$$F(t) = u \cdot X(t) = u \cdot X(t_0) + \frac{u \cdot \dot{X}(t_0)}{1!}(t - t_0) + \frac{u \cdot \ddot{X}(m(t))}{2!}(t - t_0)^2,$$

or

$$u \cdot \left(X(t) - X(t_0) - \frac{\dot{X}(t_0)}{1!}(t - t_0) \right) = \frac{u \cdot \ddot{X}(m(t))}{2!}.$$

Take

$$u = X(t) - X(t_0) - \frac{u \cdot \dot{X}(t_0)}{1!}(t - t_0).$$

Then

$$\|u\|^2 \leq \frac{\|u\| \cdot \|\dot{X}(m(t))\|}{2!}(t - t_0)^2,$$

or

$$\left\| X(t) - X(t_0) - \frac{\dot{x}(t_0)}{1!}(t - t_0) \right\| \leq \frac{\|\ddot{X}(m(t))\|}{2!}(t - t_0)^2.$$

Since $\|\ddot{X}(m(t))\| \leq A$ for all t,

$$\left\| X(t) - X(t_0) - \frac{\dot{X}(t_0)}{1!}(t - t_0) \right\| \leq \frac{A}{2!}(t - t_0)^2.$$

References

[1] D. Kim, L. J. Guibas, and S. Y. Shin, Fast Collision Detection among Multiple Moving Spheres, *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 3, pp. 230–242, 1998.

[2] J. W. Boyse, Interference Detection among Solids and Surfaces, *Communications of the ACM*, vol. 22, no. 1, pp. 3–9, 1979.

[3] S. A. Cameron, Collision Detection by Four-Dimensional Intersection Testing, *IEEE Transactions on Robotics and Automation*, vol. 6, no. 3, pp. 291–302, 1990.

[4] J. Canny, Collision Detection for Moving Polyhedra, *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 8, no. 2, pp. 200–209, 1986.

[5] T. Duff, Interval Arithmetic and Recursive Subdivision for Implicit Functions and Constructive Solid Geometry, *Computer Graphics*, vol. 26, no. 2, pp. 131–138, 1992.

[6] B. V. Herzen, A. H. Barr, and H. R. Zatz, Geometric Collisions for Time-Dependent Parametric Surfaces, *Computer Graphics*, vol. 24, pp. 39–48, August 1990.

[7] B. Mirtich, Impulse-Based Dynamic Simulation of Rigid Body Systems, Ph.D. thesis, University of California, Berkeley, CA, December 1996.

[8] B. Mirtich, Efficient Algorithms for Two-Phase Collision Detection, Tech. Rep. TR-97-23, Mitsubishi Electric Research Laboratory, Cambridge, MA, Dec. 1997.

[9] http://www.antz.com/technology, 1998.

[10] W. T. Reeves, Particle Systems—A Technique for Modeling a Class of Fuzzy Objects, *ACM Transaction on Graphics*, vol. 2, pp. 91–108, April 1983.

[11] C. W. Reynolds, Flocks, Herbs, and Schools: A Distributed Behavioral Model, *Computer Graphics*, vol. 21, no. 4, pp. 25–34, 1987.

[12] K. Sims, Particle Animation and Rendering Using Data Parallel Computation, *Computer Graphics*, vol. 24, no. 4, pp. 405–413, 1990.

[13] D. Tonnesen, Modeling Liquids and Solids Using Thermal Particles, in *Proc. of Graphics Interface '*91, pp. 255–262, 1991.

[14] V. Hayward, S. Aubry, A. Foisy, and Y. Ghallab, Efficient Collision Prediction among Many Moving Objects, *The International Journal of Robotics Research*, vol. 14, no. 2, pp. 129–143, 1995.

[15] P. M. Hubbard, Collision Detection for Interactive Graphics Applications, Ph.D. thesis, Department of Computer Science, Brown University, Providence, RI, Oct. 1994.

[16] P. M. Hubbard, Collision Detection for Interactive Graphics Applications, *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, pp. 218–230, September 1995.

[17] I. J. Palmer and R. L. Grimsdale, Collision Detection for Animation Using Sphere-Trees, *Computer Graphics Forum*, vol. 14, no. 2, pp. 105–116, 1995.

[18] P. Jimenez, F. Thomas, and C. Torras, 3D Collision Detection: A Survey, *Computers and Graphics*, vol. 25, no. 2, pp. 269–285, 2001.

[19] M. Moore and J. Wilhelms, Collision Detection and Response for Computer Animation, *Computer Graphics*, vol. 22, pp. 289–298, August 1988.

[20] G. Turk, Interactive Collision Detection for Molecular Graphics, Tech. Rep. TR90-014, Department of Computer Science, University of North Carolina, Chapel Hill, NC, Jan. 1990.
[21] R. Webb and M. Gigante, Using Dynamic Bounding Volume Hierarchies to Improve Efficiency of Rigid Body Simulation, *Visual Computing* (*Proceeding of Computer Graphics International*), pp. 825–842, 1992.
[22] J. D. Cohen, M. C. Lin, D. Manocha, and M. Ponamgi, I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments, in *Proc. of ACM Symposium on Interactive* 3*D Graphics*, pp. 189–196, 1995.
[23] S. Gottschalk, M. C. Lin, and D. Manocha, OBBTree: A Hierarchical Structure for Rapid Interference Detection, in *Proc. of ACM SIGGRAPH Conference on Computer Graphics*, pp. 171–180, 1996.
[24] P. M. Hubbard, Approximating Polyhedra with Spheres for Time-Critical Collision Detection, *ACM Transactions on Graphics*, vol. 15, pp. 179–210, July 1996.
[25] L. Guibas, A. Nguyen, D. Russel, and L. Zhang, Collision Detection for Deforming Necklaces, in *Proc. of the Eighteenth Annual Symposium on Computational Geometry*, pp. 33–42, 2002.
[26] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan, Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs, *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21–36, 1998.
[27] D. L. James and D. K. Pai, BD-Tree: Output-Sensitive Collision Detection for Reduced Deformable Models, *ACM Transactions on Graphics* (*SIGGRAPH* 2004), vol. 23, pp. 393–398, Aug. 2004.
[28] D. Baraff, Curved Surfaces and Coherence for Non-Penetrating Rigid Body Simulation, *Computer Graphics*, vol. 24, pp. 19–28, August 1990.
[29] J. K. Hahn, Realistic Animation of Rigid Bodies, *Computer Graphics*, vol. 22, pp. 299–308, August 1988.
[30] M. H. Overmars, Point Location in Fat Subdivisions, *Information Processing Letters*, vol. 44, pp. 261–265, December 1992.
[31] M. C. Lin and J. F. Canny, A Fast Algorithm for Incremental Distance Calculation, in *Proc. of IEEE International Conference on Robotics and Automation*, pp. 1008–1024, 1991.
[32] J. Basch, L. J. Guibas, and J. Hershberger, Data Structures for Mobile Data, *Journal of Algorithms*, vol. 31, pp. 1–28, 1999.
[33] L. J. Guibas, Kinetic Data Structures—A State of the Art Report, in *Proc. of* 3*rd Workshop on Algorithmic Foundations of Robotics*, pp. 191–209, 1998.
[34] L. J. Guibas and L. Zhang, Euclidean Proximity and Power Diagrams, in *Proc. of* 10*th Canadian Conference on Computational Geometry*, http://cgm.cs.mcgill.ca/cccg98/proceedings/welcome.html, 1998.
[35] F. Aurenhammer, Power Diagrams: Properties, Algorithms and Applications, *SIAM Journal on Computing*, vol. 16, pp. 78–96, 1987.
[36] L. Guibas, J. Snoeyink, and L. Zhang, Compact Voronoi Diagram for Moving Convex Polygons, *Algorithm Theory - SWAT*, vol. 1851, pp. 339–352, 2000.
[37] J. L. Bentley and T. A. Ottmann, Algorithms for Reporting and Counting Geometric Intersections, *IEEE Transactions on Computers*, vol. 28, pp. 643–647, 1979.
[38] T. H. Cormen and C. H. Leiserson, *Introduction to Algorithms*, The MIT Press, Cambridge, MA, 1990.
[39] K. R. Symon, *Mechanics*, 3rd edn., Addison-Wesley, Reading, MA, 1979.
[40] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry*: *Algorithms and Applications*, Springer-Verlag, New York, 1997.
[41] F. P. Preparata and M. I. Shamos, *Computational Geometry*: *An Introduction*, Springer-Verlag, New York, 1985.
[42] J. Wejchert and D. Haumann, Animation Aerodynamics, *Computer Graphics*, vol. 25, no. 4, pp. 19–22, 1991.