Fast Collision Detection among Multiple Moving Spheres *

Dong Jin Kim† Leonidas J. Guibas‡ Sung Yong Shin†

Abstarct

This paper presents an event-driven approach that efficiently detects collisions among multiple moving spheres of uniform radius. We divide the space containing the spheres into uniform subspaces of cell structure. Each sphere intersecting a subspace is tested against the others intersecting the same subspace for possible collisions. We identify three types of events to detect the sequence of all collisions during our simulation: collision, entering, and leaving. The first type of events is due to actual collisions, and the other two types occur when spheres move from subspace to subspace. By tracing all such events in the order of their occuring times, we are able to simulate n moving spheres with proper collision response[1, 5, 10] in $O(n_c \log n + n_e \log n)$ time with $O(n)$ space after $O(n \log n)$ time preprocessing, where n_c and n_e are the number of actual collisions and that of entering and leaving events during the simulation, respectively. Since n_e depends on the size of subspaces, we adopt the collision model from kinetic theory for molecular gas[3] to determine the subspace size that minimizes simulation time. Experimental results show that collision detection can be done in linear time in n over a large range.

1 Introduction

Spheres are widely used in computer graphics and robotics since they are simple and effective to model a variety of objects. They can not only represent objects such as atoms, balls, and sand[9, 12, 13, 14, 15] but also be used to approximate other 3D objects such as desk lamps, piles, and clumps[7, 11]. Particle systems use spheres to represent point masses for modeling complex natural phenomena such as fire, clouds, and water[12, 13, 14, 15]. Molecular graphics use spheres to describe the behavior of molecules composed of atoms[6, 8, 16]. Polyhedral objects are approximately represented by hierarchies of spheres[7, 11]. In order to consider interactions among spheres in such cases, it is a fundemental problem to efficiently detect collisions among spheres. To solve this problem, the number of collision checks must be minimized[1, 5, 10, 17, 18].

*This work was supported in part by Samsung Advanced Institute of Technology(project Research on Multimedia Creation Technology)

†Department of Computer Science, Korea Advanced Institute of Science and Technology, Taejon, Korea 305-701. e-mail:{djkim, syshin}@jupiter.kaist.ac.kr

‡Computer Science Department, Stanford University, Stanford, USA CA 94305. e-mail:guibas@cs.stanford.edu

In physical simulation of moving 3D objects, collision checks are done at a sequence of sampled times called time steps[7, 10, 17, 18]. In order not to miss any collision, the number of steps are increased, or objects are assumed to move sufficiently slowly to hit other objects no more than once between two consecutive time steps[4]. The latter sacrifices generality to restrict the domain of simulation. Meanwhile, the former requires heavy computation time. Assuming that objects are represented as a set of spheres, we propose an efficient event-driven approach for detecting the collisions among multiple moving spheres to avoid excessive collision checks.

In general, the motion of a solid is a mixture of translation and rotation, which makes it hard to predict an exact collision. A sphere is extremely simpler in shape than a general solid, and its motion can be described by translation only. Due to those nice properties, the collision time between two spheres is much easier to compute. In some sense, a simulation of moving spheres may be considered to be too domain-specific. However, its efficiency makes up for the loss of generality. The popularity of bounding spheres and a hierarchy of spheres to approximate a 3D object justify efficient methods for fast collision checks among spheres. Moreover, there are a wide variety of applications of particle modeling for which spheres themselves are massively floating around. The interaction among spheres is important to support those applications.

2 Algorithms and Analysis

We take an event-driven approach instead of time-consuming collision check at every time step. In order to localize collision checks, we decompose the space containing moving spheres into a set of uniform subspaces of cell structure. We keep track of the path of every moving sphere and the list of spheres intersecting each subspace. Those paths and lists are changed when a pair of spheres collide and when a sphere in a subspace moves to another subspace, respectively. Their occuring times are kept in a global queue. Thus our approach most closely resembles sweep-line and -plane methods in computational geometry, except that in our approach the dimension being swept over is time. To efficiently detect such events in the order of their occuring times, we first need nice data structure for non-empty subspaces intersecting one or more spheres and candidate pairs of spheres for collision. This idea is further explored in [2] from the theoretical point of view to maintain various attributes of mobile data.

Let U be a sufficiently big cube of volume D^3 that contains n moving spheres, where D is the length of edges of the cube. The cube U represents the space in which the moving spheres are simulated. We decompose this space into a set of uniform subspaces of cell structure. Each of the subspaces is a cube of volume L^3. For simplicity, D is taken to be a multiple of L, that is, $D = mL$. The size L^3 of the volume of a subspace is determined to be a constant multiple c of

that of a sphere so that each subspace is able to accomodate two or more spheres at a given instance.

Without loss of generality, U is assumed to be contained in the octant in which the three coordinate values of every point are all non-negative. Moreover, a vertex of U is coincident with the origin, and each edge of U is parallel with a principle axis. Let each subspace be denoted by $u(u_x, u_y, u_z)$ for $0 \leq u_x, u_y, u_z < m$ such that

$$\begin{aligned} u(u_x, u_y, u_z) = & \quad [u_x L, (u_x + 1)L] \\ & \times \quad [u_y L, (u_y + 1)L] \\ & \times \quad [u_z L, (u_z + 1)L]. \end{aligned} \tag{1}$$

There are m^3 subspaces. However, many of them do not intersect any sphere. Since the size of a subspace is at least twice as large as that of a sphere, each sphere intersects at most eight subspaces. Therefore, there are $O(n)$ non-empty subspaces. Employing a bounded-balanced tree[19], we build a search tree, called a *subspace tree*, to keep those subspaces. A leaf node of the tree represents a non-empty subspace and has a list of spheres intersecting the subspace represented by the node. Search, insert, and delete operations on the subspace tree can be done in $O(\log n)$ time each. As auxiliary data structures, every sphere maintains a list of subspaces intersecting itself. This information is linked to the corresponding leaf nodes of the subspace tree.

In order to efficiently access a subspace in the tree, we assign the unique key to every subspace $u(u_x, u_y, u_z)$ as follows:

$$key(u(u_x, u_y, u_z)) = u_x m^2 + u_y m + u_z, \tag{2}$$

where $0 \leq u_x, u_y, u_z < m$. The function $key(\cdot)$ associates $u(u_x, u_y, u_z)$ with a unique integer in the range from 0 to $m^3 - 1$. It is clear that $key(\cdot)$ is one to one and onto.

Given a sphere with its position and radius, it takes a constant time to find the intersecting subspaces with it. It also takes a constant time to compute the key of each subspace. There are at most eight subspaces intersecting a sphere. Thus, we can compute all keys for a sphere in a constant time. It takes $O(\log n)$ time to insert and update each subspace. Since there are a maximum of eight subspaces to handle for a sphere, each sphere needs $O(\log n)$ time. Hence, it takes $O(n \log n)$ time to initially construct a subspace tree. Moreover, the number of leaf nodes is $O(n)$ which immediately gives $O(n)$ space to keep the tree[19].

We identify three types of events to detect the sequence of collisions during a simulation: collision, entering, and leaving. The first type of events is due to actual collisions, and the other two types occur when spheres enter new subspaces and leave the current subspace(s), respectively. Allowing penetration with other spheres, we can compute the times for candidate collision events of each sphere with the others in the same subspace as well as those candidates for leaving and entering subspaces. Under this condition, the candidate events seem to have nothing to do with actual ones. However, given all such events for n spheres, no penetration occurs until the earliest among them. Therefore, an actual event must occur at the time of the earliest candidate, and no others do by this time. Assuming that the subspace tree is correctly updated to reflect the actual event, we can again catch the next earliest event in the same manner. Repeating this process, all the events can be generated in their time sequence.

Every sphere has a constant number of candidate events: $O(c)$, $O(1)$, and $O(1)$ candidates for collision, entering, and leaving, respectively. Thus, the total number of candidates are $O(n)$. We again adopt a bounded-balanced tree to maintain all candidate events. A leaf node of the event tree represents a candidate event. Its occuring time is used for the search key. Occuring times may not to be unique. However, we can always use sphere labels and event types in addition to those times to break any ties. The candidate events are arranged in the ascending order of their keys. By a similar analysis for the subspace tree, we can initially construct the event tree in $O(n \log n)$ time with $O(n)$ space.

According to the type of the current event, the change of distribution of the spheres over the subspaces together with its resulting change of candidate events are correctly updated in the data structures such as subspace and event trees. This change includes:

1. the candidate events of each sphere that is involved in the current event.

2. the subspaces of each sphere of the current event.

Suppose that the current event is caused by an actual collision between two spheres. Then, the portion of the moving path of each of those spheres after the collision is changing due to the collision response of one sphere to the other. Collision response is beyond the scope of this paper although it itself is very interesting. There are several models for collision response available in computer graphics literature[1, 5, 10]. By renewing the paths of two colliding spheres, we can compute the new occurring times of the candidate events of each sphere. They include the times for candidate collisions with the other spheres in the same subspace(s) as well as for the candidate entering and leaving events of the sphere. There are a pair of spheres that are involved in a collision. Moreover, every sphere intersects at most eight subspaces, each of which may have a constant number c of intersecting spheres. Therefore, every collision causes a constant number of candidate events to be renewed. Since each of those events needs $O(\log n)$ time to be updated in the event tree, it takes $O(\log n)$ time to handle a collision event.

Now, consider the case that the current event occurs due to a sphere entering a subspace. The subspace has the entering sphere as a new member in its list of intersecting spheres. That sphere needs adding to the list for the subspace in the subspace tree, which takes $O(\log n)$ time. Accordingly, the new candidate events for collisions between the sphere and the others in the subspace are added to the event tree. This can also be done in $O(\log n)$ time since the subspace intersects a constant number of spheres. We finally renew the candidate event time for entering another subspace in $O(\log n)$ time. Therefore, it takes $O(\log n)$ time to handle an entering event.

Finally, suppose that the current event is caused by a sphere leaving a subspace. Then, that sphere needs removing from the subspace, and the candidate events for collisions between the sphere and the others in the subspace are also required to be removed. Clearly, it takes $O(\log n)$ time to update the subspace and event trees to reflect those removals, respectively. It also takes $O(\log n)$ time to renew the candidate event time of the sphere for leaving. Therefore, we need $O(\log n)$ time to handle a leaving event.

A brute-force approach needs $O(n_t n^2)$ operations for collision detection, where n_t denotes the number of time steps during a simulation. Let n_c and n_e be the number of actual collisions and that of actual entering and leaving events for the simulation, respectively. Since it takes $O(\log n)$ time to handle an event regardless of its type, we need $O(n_c \log n +$

$n_e \log n)$ operations for the whole simulation of n moving spheres after $O(n \log n)$ initial investment for constructing subspace and event trees. It is clear that n_c cannot be further reduced. However, n_e is a function of the size of subspaces, and thus needs more attention to improve.

3 Determining the Size of Subspaces

For a given n, D, a radius of spheres, and their mean velocity, we can choose L by solving the following equation:

$$6vTL^{-2}\left[2n\frac{(L+2r)^2}{D^3}\left\{c_1\pi r^2 n_0 L^2 + c_3(L-r)\right\} - c_4\right] = 0$$

This equation is derived using the collision model of kinetic theory for molecular gas[3]. The obtained subspace size is not optimal. However, experiments support that it is a good approximation of the optimal subspace size.

4 Experimental Results

In order to show the efficiency of the proposed event-driven approach, experiments are performed on an IRIS Indigo2 machine(128MHz, R10,000CPU, 128MB) for multiple spheres of uniform radii freely moving in a given box. Assuming that a gravity force is zero, the motion of each sphere is given by its initial position and velocity. A path of a sphere is changed whenever it undergoes a collision with other spheres or collides with a face of the box.

Varying the number of spheres for a fixed radius and mean speed while optimizing the subspace size, we measure actual running times(Figure 1). The vertical axis gives an actual running time for a unit time of simulation of given data. As plotted in the figure, it is nearly linear in n over a large range.

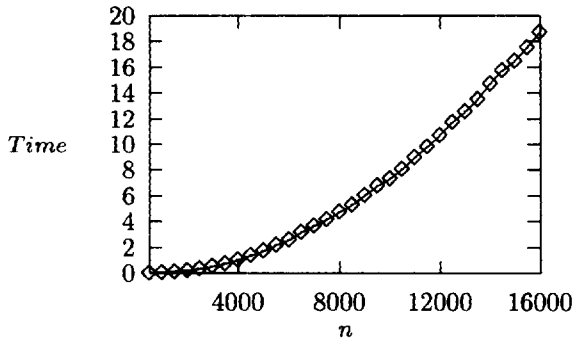

Figure 1: The radius of spheres is 1.5, their mean speed is $10\sqrt{3}$, and the size of the space is 200×200×200,

5 Conclusions

This paper proposes an event-driven approach for efficient collision detection among multiple moving spheres of uniform size. The proposed approach enables us to simulate in real time several thousands of spheres freely moving in a 3D box. Although we assume that all spheres are of uniform size, our approach can also be applicable to spheres of non-uniform radii if the size of the largest sphere is a small constant multiple of that of the smallest one.

Acknowledgements

We would like to thank Mr. Chan-Su Shin for his careful reading.

References

[1] David Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *Computer Graphics*, 24(4):19–28, August 1990.

[2] Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. In *ACM-SIAM Symposium on Discrete Algorithms*, 1997(to appear).

[3] Feynmann, Leighton, and Sands. *The Feynmann Lectures on Physics*, volume 1, pages 43.1–43.4. Addison-Wesley, 1963.

[4] Stefan Gottschalk, Ming Lin, and Dinesh Manocha. Obbtree: A hierarchical structure for rapid interferece detection. *Computer Graphics Proceedings, Annual Confernece Series*, pages 171–180, 1996.

[5] James K. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):299–308, August 1988.

[6] Dan Halperin and Mark H. Overmars. Spheres, molecules, and hidden surface removal. In *Proc. 10th Annual ACM Symposium on Computational Geometry*, pages 113–122, 1994.

[7] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *IEEE Transactions of Graphics*, 15(3):179–210, July 1979.

[8] Cyrus Levinthal. Molecular model-building by computer. *Scientific American*, pages 42–52, June 1966. 213(6).

[9] Victor J. Milenkovic. Position-based physics: Simulating the motion of many highly interacting spheres and polyhedra. *Computer Graphics Proceedings, Annual Conference Series*, pages 129–136, 1996.

[10] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. *Computer Graphics*, 22(4):289–298, August 1988.

[11] Joseph O'Rourke and Norman Badler. Decomposition of three-dimensional objects into spheres. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(3):295–305, July 1979.

[12] W. T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. *ACM Transaction on Graphics*, 2(2):91–108, April 1983.

[13] Craig W. Reynolds. Flocks, herbs, and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[14] Karl Sims. Particle animation and rendering using data parallel computation. *Computer Graphics*, 24(4):405–413, 1990.

[15] Richard Szeliski and David Tonnesen. Sufrace modeling with oriented particle systems. *Computer Graphics*, 26(2):185–194, July 1992.

[16] G. Turk. Interactive collision detection for molucular graphics. Technical report, 90-014, Dept. of Computer Science, Univ. of North Calolina at Chapel Hill, Jan. 1990.

[17] Vincent Hayward, Stéphane Aubry, André Foisy, and Yasmine Ghallab. Efficient collision prediction among many moving objects. *The International Journal of Robotics Research*, 14(2):129–143, 1995.

[18] Robert Webb and Mike Gigante. Using dynamic bounding volume hierarchies to improve efficiency of rigid body simulation. *Visual Computing*, pages 825–842, 1992. (CGI Proc).

[19] Dan E. Willard and George S. Lueker. Adding range restriction capability to dynamic data structure. *Journal of The Association for Computing Machinery*, 32(3):pp. 597–617, July 1985.