GPCD:Grid-based Predictive Collision Detection for Large-scale Environments in Computer Games

Zhiwen Yu
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Hong Kong
yuzhiwen@cs.cityu.edu.hk

Hau-san Wong
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Hong Kong
cshswong@cityu.edu.hk

Abstract

Given a time horizon parameter h and an object set O, predictive collision detection finds all the object pairs $< o_i, o_j, t_i >$ which will collide in the future time interval $[t, t+h]$ (where $1 \leq i, j \leq n$, $t_i \in [t, t+h]$). Although there are a number of state-of-the-art approaches to solve collision detection problems, predictive collision detection is addressed for the first time. In this paper, we propose a grid-based predictive collision detection algorithm (GPCD), which is a general technique for the efficient detection of the collision of object pairs in a future time interval. GPCD first determines a candidate list which stores the object pairs having a non-zero probability to collide in a future time. Then, GPCD achieves low running time based on two pruning strategies: (i) space intersection test and (ii) time intersection test. These two pruning strategies eliminate most of the false collision cases in an initial filtering phase. In the refinement phase, a bounding-volume tree is applied to refine the detection results. Our experiments show that GPCD works well for the purpose of predictive collision detection.

1. Introduction

Collision detection is a fundamental problem in robotics, computer graphics, computational geometry, computer animation and physics based modeling ([1], [2], [3], [4], [5], [6], [7], [8], [10], [11]). The collision detection problem is divided into two subproblems from the perspective of the number of objects considered: (i) two objects collision detection and (ii) n objects collision detection. Predictive collision detection belongs to the latter which is addressed for the first time. The objective of predictive collision detection is to avoid future collision, which represent a major difference from traditional collision detection. There are many applications for predictive collision detection, such as simulating a group of random walking robots, ships going in or out of the harbor, cars driving in or out of a car park, and so on.

The motivation of predictive collision detection comes from predictive queries in the database area([9]). Predictive query retrieves the objects which satisfy some conditions in the future time, such as "Which bus is expected to reach the bus station in the next 5 minutes", "Which ship is close to the dock in the next 10 minutes", and so on. There are three kinds of predictive queries: (i) predictive window query, (ii) predictive nearest neighbor query, (iii) predictive spatial join query. The query that retrieves all the object pairs $< o_i, o_j, t_i >$ which will collide in the future time interval $[t, t+h]$ is a novel type of query which belongs to the class of predictive spatial join queries (where $1 \leq i, j \leq n$, $t_i \in [t, t+h]$).

In this paper, we propose a grid-based predictive collision detection (GPCD) algorithm to retrieve all the object pairs which will collide in the future time interval $[t, t+h]$. The objects which are bounded by a minimum bounding box(MBB) are indexed by the grid G. GPCD first determines a candidate list for the object o. Then, GPCD achieves high efficiency by two pruning strategies: (i) a space intersection test which eliminates the false candidates by checking the overlapping of the projected interval, (ii)a time intersection test which removes the false candidates through examining the overlapping of the time intervals. Finally, GPCD computes the exact time of the intersection for the object pairs which are not pruned. Here we only focus on retrieving collision pairs. To keep track of collision pairs and dynamically maintain the system will be the topics of our future work.

The contribution of this paper are threefold. First, we address predictive condition detection for the first time. Second, we propose a grid-based predictive collision detection (GPCD) algorithm to retrieve all the object pairs which will collide in future. Third, we provide a space intersection test and a time intersection test to reduce the computational cost.

The rest of the paper is organized as follows. Section 2 surveys the related work on predictive query and collision detection. Section 3 describes the grid-based predictive collision detection algorithm. Section 4 evaluates GPCD experimentally. Section 5 is conclusion and future work.

2. Related work

For an n object collision detection problem, we need to perform $O(n^2)$ pairwise interference checks, which is computationally intensive, especially when n is large. In order to eliminate useless pairwise checks, several techniques are proposed ([2], [3], [8]). The most related work to us is the sorting-based sweep and prune algorithm([8]).

3. Grid-based predictive collision detection algorithm

3.1. The overview of the algorithm

Grid-based predictive collision detection algorithm (GPCD) is proposed to detect all the object pairs $< o_i, o_j, t_i >$ which have a non-zero probability to collide in the future time interval $[t, t+h]$,where $1 \leq i, j \leq n$, $t_i \in [t, t+h]$. We first consider the case where the object moves non-linearly in space. The motion function of the object is

$$o(t_i) = o(t) + v \cdot (t_i - t) + \frac{F}{2m} \cdot (t_i - t)^2 \ (t_i \in [t, t+h]) \quad (1)$$

where $o(t_i)$ and $o(t)$ are the position vectors of the object o at time t_i and the current time t, v is the current velocity of the object, F is the force vector, m is the mass of the object o, and h is a time horizon.

An overview of GPCD is provided in figure 1 and figure 2. GPCD first computes the displacement for all the objects in the object set. In order to avoid picking up an object pair twice, GPCD only considers an object pair $(o(t), o_i(t))$ with $o_i(t).x \leq o(t).x$ for every object o ($o_i(t).x$ and $o(t).x$ are the x coordinates of the center point of the object o). Then, GPCD determines the candidate list for the object o. In the third step, the space intersection test and the time intersection test are applied to eliminate the false candidates. Finally, GPCD returns the object pairs which will collide in the future time interval $[t, t+h]$. The size of the arrow in figure 1 reflects the size of the candidate list.

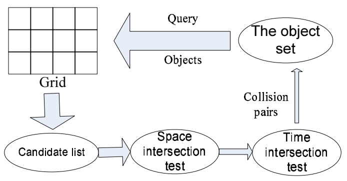


Figure 1. The overview of the system

3.2. Determination of the candidate list

GPCD first selects $\prod_{i=1}^{d} c_i \cdot \bar{I}_i$ as the grid cell size(where d is the number of dimensions, $\bar{I}_i$ is the average projected intervals of the objects and their trajectories in the ith dimension, and c_i is a constant parameter. **Note that**, in order to save computational cost, the bounding box is applied to bound the object and its trajectory. Then, the algorithm calculates the intersection of the grid cell with the bounding rectangle. If the grid cell intersects the bounding rectangle, (i)the grid cell stores the object id and (ii) the object adds the grid cell id. The candidate list of the object o consists of all the objects whose bounding boxes share the same grid cell with that of the object o.

```
Algorithm **GPCD**(object set $O$)
1.  Initialize a collision pair list;
2.  compute the trajectories for all the objects;
3.  Project the objects and their trajectories into all the dimensions;
4.  Calculate average projected interval ($\bar{I}_i$) in all the dimensions;
5.  Choose the grid cell size $\prod_{i=1}^{d} c_i \cdot \bar{I}_i$;
6.  Hashes all the objects and their trajectories into the grid;
7.  The grid cell stores the id of the objects intersected;
8.  The object stores the id of the grid cell intersected;
9.  For each object $o$
10.   Add all the objects in the grid cells which contain the object $o$
        into the candidate list;
11.   For each candidate $o_i$ in the candidate list
12.     If (! projection overlap condition)
13.       the candidate $o_i$ is pruned;
14.     If (center point test);
15.       add $< o, o_i >$ into collision pair list; continue;
16.     If (! Time interval condition)
17.       the candidate $o_i$ is pruned;
18.     If (! trajectory intersection test)
19.       the candidate $o_i$ is pruned;
20.     Compute exact time of intersection;
21.     add $< o, o_i >$ into collision pair list;
22. Refi nement by bounding volume tree;
23. If the collision pair list is not empty
24.   Take measures to avoid collision;
```

Figure 2. Grid-based predictive collision detection algorithm

3.3. Space intersection test

After the algorithm obtains the candidate list of the object o, space intersection test is applied to eliminate the false candidates. Space intersection test includes two phases: projection overlap test and trajectory intersection test.

During the process of projection overlap test, the trajectories of the object o and its candidates are projected to all the dimensions. If the projected interval of the candidate does not intersect that of the object o in one of the dimensions, the candidate will be removed from the candidate set. As a result, the candidates o_i retained in the candidate list should satisfy the following projection overlap condition:

$$[o.I_j(min), o.I_j(max)] \bigcap [o_i.I_j(min), o_i.I_j(max)] \neq \emptyset \quad (2)$$

where $o.I_j(min)$, $o.I_j(max)$, $o_i.I_j(min)$, $o_i.I_j(max)$ are the minimum and maximum coordinates of the projected interval of object o and the candidate o_i in the jth dimension. j is the dimension ($1 \leq j \leq d$, d is the total number of the dimensions).

There are four outcomes of the trajectory intersection test: (i)the center of the object and that of the candidate intersect; (ii) The trajectory of the object and that of the candidate intersect ; (iii) The MBR of the object and that of the candidate intersects ; (iv) The object does not intersect with the candidate.

3.4. Time intersection test

For the time intersection test, the algorithm considers the candidates one by one. Time intersection test consists of three stages: (i) center point test, (ii) time interval test, (iii) exact time computation.

The algorithm first computes the time t_i at which the center of the object o and that of the candidate o_i intersects by solving the following quadratic equations:

$$\begin{cases} o(t)_j + o.v_j(t_i - t) + \frac{o.F_j}{2o.m}(t_i - t)^2 \\ = o_i(t)_j + o_i.v_j(t_i - t) + \frac{o_i.F_j}{2o_i.m}(t_i - t)^2 \end{cases} \quad (3)$$

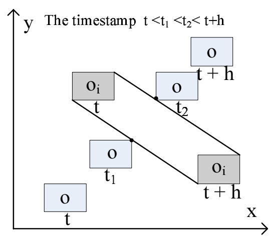
where $o(t)_j$ and $o_i(t)_j$ are the coordinates of the positions of the center of the object and the candidate in the jth dimension at time t respectively. v_j, F_j are the corresponding component of the velocity and the force. t and t_i are the current time and the future time, and j is the dimension ($1 \leq j \leq d$, where d is the total number of dimensions).

If $t_i \in [t, t+h]$, the object o and the candidate o_i is a collision pair. Otherwise, the time interval test and exact time test are performed. If the candidate does not satisfy the time interval test or the exact time test, the candidate is pruned.

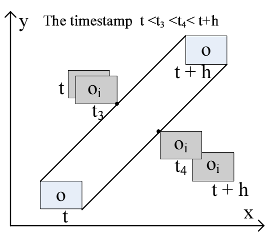
During the time interval test, the algorithm calculates the times t_1 and $t_2 (t_1, t_2 \in [t, t+h])$ of which the object o enters and leaves the trajectory of the candidate o_i, as shown in Figure 3(a), by solving the quadratic equations which links the trajectory functions of the vertices of the bounding boxes. The times t_3 and t_4 ($t_3, t_4 \in [t, t+h]$) at which the candidate intersects the trajectory of the object o (Figure 3(b)) are obtained in a similar way. The candidate o_i is pruned if it does not satisfy the time interval intersection condition:

$$[t_1, t_2] \bigcap [t_3, t_4] \neq \emptyset \quad (4)$$

Finally, the algorithm computes the exact time of intersection for the remaining candidates.



(a)



(b)

Figure 3. Time intersection test in 2D

3.5. Collision Avoidance

If the candidate o_i is not pruned, the collision pair $< o, o_i >$ is added to the collision pair list. All the object pairs in the collision pair list are refined by a bounding volume tree(BVT). Finally, if the collision pair list is not empty, the algorithm considers other measures to avoid collision. These measures include acceleration, deceleration or diverting the direction of movement. Acceleration and deceleration allow the object to break the time intersection condition, while diverting the direction of movement helps the object to break the space intersection condition.

4. Experiment

4.1. Experimental Setting and Data Set

All the experiments presented are executed with a Pentium 2.8 GHz CPU with 1 GByte memory. Our data simulate a battle in a computer game which includes a group of random running soldiers based on information from 3D computer animation. We only consider the movement of random running soldiers in a 3D scene with size $[10000 \times 10000 \times 50]$. In our experiments, every soldier is bounded by a minimum bounding box. The distribution of the soldiers in different regions are shown in Figure 4.

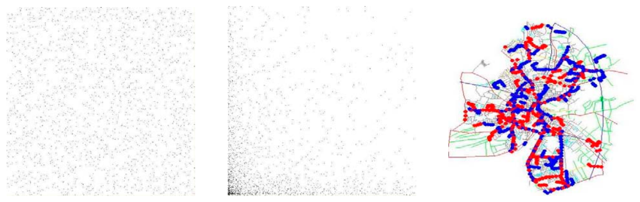

(a) Uniform (b) Zipf (c) City

Figure 4. dataset

Table 1 summarizes the parameters along with their default values and the ranges. In the following experiments, we vary the value of one parameter, while setting the values of other parameters as default values. The parameters of the grid size in GPCD algorithm are $c_1 = 1$, $c_2 = 1$ and $c_3 = 1$ in 3D. We compare the collision detection performance based on the grid structure with that of a TPR-tree using the self-intersection join algorithm(SIJ-TPR).

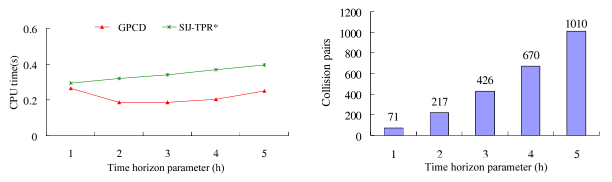
Parameter	Default	Range
Horizontal parameter(h)	3	$1, 2, 3, 4, 5$
Maximum velocity (v_{max})	3	$1, 2, 3, 4, 5$
Maximum accelerate (a_{max})	0.5	$0, 0.5, 1, 1.5, 2$
Object population(n)	$2k$	$1k, 2k, 3k, 4k, 5k$
Object distribution	uniform	uniform, zipf, City
Algorithms	GLOBAL	GPCD, SIJ-TPR
Motion function	linear	linear, non-linear

Table 1. Parameters

4.2. The influence of the parameters

According to the parameters in Table 1, we process the queries that find all the collision pairs by the algorithms, and measure the running time and collision object pairs by varying one of the parameters.

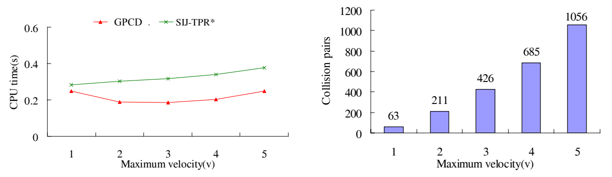
Figure 5(a) illustrates the running time with respect to the time horizon h. As a whole, the running time increases when h becomes larger. The running time of GPCD first decreases, then increases. When h is small, the corresponding average projected interval is small as well. This in turn leads to a small grid size and a large number of grid cells which degenerate the performance of GPCD. Figure 5(b) shows the number of collision pairs corresponding to the time horizon parameter h. The number of collision pairs increase quickly with h.



(a) Running time (b) Collision pairs

Figure 5. Effect of time horizon parameter h

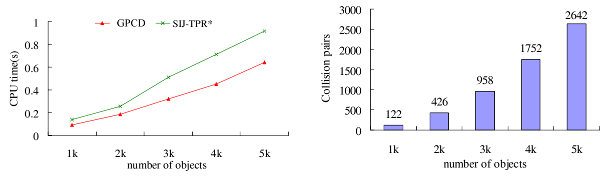
Figure 6(a) shows the running time as a function of the maximum velocity. GPCD is rather insensitive to the maximum velocity, since the performance of GPCD is mainly related to the average velocity, not the maximum velocity. Figure 6(b) measures the effect of the maximum velocity on the number of collision pairs.



(a) Running time (b) Collision pairs

Figure 6. Effect of maximum velocity

Figure 7(a) compares the performance of the algorithms versus the number of objects. It is reasonable that all the algorithms are sensitive to the number of objects. With an increased number of objects, the algorithms process more pairwise interference checks, and the corresponding collision pairs increase as well, as shown in Figure 7(b).



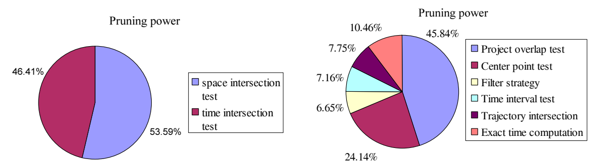
(a) Running time (b) Collision pairs

Figure 7. Effect of n

The prune power of the pruning strategies is defined by the following equation:

$$prunepower(\%) = \frac{N_{beforeprune} - N_{afterprune}}{N_{beforeprune}} \cdot 100\% \quad (5)$$

where $N_{beforeprune}$ and $N_{afterprune}$ denote the cardinality of the candidate list before pruning and after pruning. Figure 8(a) compares the prune power of space intersection test and time intersection test. Figure 8(b) illustrates the prune power of the different strategies.



(a) Prune strategies (b) Concrete strategies

Figure 8. Prune power

5. Conclusion and future work

The paper investigates the problem of predictive collision detection. Given an object set and a time horizon parameter, the query retrieves all the object pairs which will collide in future. Although there are a number of collision detection approaches, predictive collision detection is addressed for the fi rst time. The objective of prediction is to avoid collision. Our major contribution is the introduction of a grid-based predictive collision detection algorithm(GPCD) to predict the collision in future. GPCD is divided into three main steps: (i) determination of the candidate list, (ii) space intersection test, (iii) time intersection test. We compare GPCD with SIJ-TPR in the experiments which show that GPCD outperforms SIJ-TPR.

There are still a lot of interesting work to be done in the future. First, it will be promising to establish an integrated predictive collision detection system and to maintain the system dynamically. Second, handling updates is a challenge in the predictive system. Third, we plan to apply this approach to explore the motions of robots.

Acknowledgments

The work described in this paper was partially supported by a grant from City University of Hong Kong [Project No. 7001965].

References

[1] Naga K. Govindaraju, David Knott, Nitin Jain, Ilknur Kabul, Rasmus Tamstorf, Russell Gayle, Ming C. Lin, Dinesh Manocha,Interactive Collision Detection between Deformable Models using Chromatic Decomposition, ACM SIGGRAPH, 2005

[2] Stefan Gottschalk, Ming C. Lin, Dinesh Manocha. OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. Proc. ACM SIGGRAPH, 1996.

[3] JAMES, D. L., AND PAI, D. K. BD-Tree: Output-sensitive collision detection for reduced deformable models. Proc. of ACM SIGGRAPH.2004.

[4] BRIDSON, R., FEDKIW, R., AND ANDERSON, J. Robust treament for collisions, contact and friction for cloth animation. Proc. of ACM SIGGRAPH.2002.

[5] Sung-Eui Yoon, Brian Salomon, Ming C. Lin, Dinesh Manocha. Fast Collision Detection between Massive Models using Dynamic Simplifi cation. Proc. of Symposium on Geometry Processing, 2004

[6] Stephane Redon, Ming C. Lin, Dinesh Manocha, Young J. Kim. Fast Continuous Collision Detection for Articulated Models. Proceedings of ACM Symposium on Solid Modeling and Applications, 2004.

[7] Naga K. Govindaraju, Ming C. Lin, Dinesh Manocha. Fast and Reliable Collision Culling using Graphics Processors. IEEE Transactions on Visualization and Computer Graphics , 2005

[8] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, Madhav Ponamgi. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scaled Environments. Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, Madhav Ponamgi Proc. ACM Symposium on Interactive 3D Graphics, pp. 189-196, 1995.

[9] Saltenis, S., Jensen, C., Leutenegger, S., Lopez, M. Indexing the Positions of Continuously Moving Objects. SIGMOD, 2000.

[10] Vincent Hayward,Stphane Aubry, Andr Foisy, Yasmine Ghallab. Effi cient collision prediction among many moving objects. International Journal of Robotics Research archive Volume 14 , Issue 2 (April 1995), Pages: 129 - 143.

[11] Kim, B. and Rossignac, J. Collision Prediction for Polyhedra under Screw Motions. Proc. of the ACM SM'03. pp 4-10. Seatle, Washington. ISBN:1-58113-706-0. June, 2003.