Abstract

We present a scheme for collision detection of flexible models. This scheme relies on a multi-resolution technique of: 1) object location tracking to decrease time complexity, 2) the combination of bounding boxes (AABBs) and a hash grid function, and 3) computation of distance domain to determine collision with a contact surface. Experiments show that this approach can efficiently detect collisions in a large set of flexible objects.

Keywords: collision detection, multi-resolution method, hash grid function, location tracking, flexible body

Introduction

There are many collision detection problems addressed by many different techniques. Bounding volume hierarchies have been developed to speed up intersection tests of close bodies. For example: bounding spheres (Hubbard 1995, James and Pai 2004), axis-aligned bounding boxes (AABBs) (Bergen 1997, Teschner et al. 2003), oriented bounding boxes (OBBs) (Gottschalk et al. 1996), quantized orientation slabs with primary orientations (QuOSPOs) (He 1999), and discrete-oriented polytopes (K-DOPs) (Klosowski et al. 1998).

VCLIP (Mirtich 1998) and CLOD with dual hierarchy (Otaduy and Lin 2003) have been also proposed for polyhedral objects collision detection. Separation-sensitive collision detection (Erickson et al. 1999) was presented for convex objects while kinetic collision detection (Basch et al. 2004) is designed for simple polygon.

For many interactive 3D graphics, animation and visualization applications, many papers have also employed spatial decomposition for both screen and input models to reduce time complexity. The simple idea of screen spatial decomposition is to subdivide the screen into grid cells and determine the grid cell to animated objects from the positions, similarly for input models. The spatial decomposition techniques that have been proposed are octree (Moore and Wilhelms. 1988), BSP tree (Naylor et al. 1990), brep-indices (Bouma and Vanek, Jr 1991), k-d tree (Held et al. 1995), bucket tree (Ganovelli 2000), hybrid tree (Larsson et al. 2001), BVIT(Otaduy and Lin 2003), and uniform space subdivision (Teschner et al. 2003). Moreover, there are some collision detection algorithms for a large environment such as I- COLLIDE (Cohen et al. 1995) and CULLIDE (Govindaraju et al. 2003).

Most efforts have been in solving the collision detection problem for rigid bodies. This usually entails large sets of static data. Flexible objects however, have dynamically changing geometry. There may be few, if any, fixed points in the scene. Therefore, collision detection techniques optimized for rigid bodies are often not applicable, or less optimal for collision between flexible bodies. We propose a collision detection technique that can efficiently deal with a large set of flexible models.

The remainder of this paper proceeds as follows: first, an overview of the proposed algorithm, then a set of the experiments with the algorithm and results,
and finally, conclusions and discuss possibilities of further work.

**Algorithm Overview**

For collision detection with flexible objects in a large environment, we present the notion of tracking with bounding boxes, collision grid domain and contact surfaces by following:

1) Tracking with Bounding box (TB): all flexible objects are surrounded by a bounding box (AABB) for tracking. This tracking depends on the velocity gap between time periods. Two points, a minimum and a maximum, define the bounding box. The technique may be modified to make use of any bounding volume however.

2) Collision Grid Domain (CGD): we also present the notion of tracking with bounding boxes, collision grid domain and contact surfaces by following:

3) Contact surface (CS): to compute contact surface, distances of the flexible objects in same collision grid domain (CGD) are calculated. If the distances are less than the collision tolerance, collision or collisions will be detected.

**Tracking with Bounding box (TB)**

For computation in a linear system at each step time, we use location tracking. We put boundary boxes (AABBs) to determine the location of object in this step. We use the following notation:

**Geometry**

- $x^i$ is the geometric component in object $i$.
- $v^i_0, \ldots, v^i_n$ are the positions of vertices for object $i$.
- $v^i_{kx}$ is the position of vertex $v^i_k$ in x direction.
- $v^i_{ky}$ is the position of vertex $v^i_k$ in y direction.
- $v^i_{kz}$ is the position of vertex $v^i_k$ in z direction.
- $m^i$ is the mass of object $i$.

**Direction**

- $n^i v^i_k$ is the velocity of $v^i_k$ at time $n$.
- $n^i k$ is the normal vector of $v^i_k$ at time $n$.
- $f^i_k$ is the force of $v^i_k$ at time $n$.
- $VN^i_k$ is dot product of $v^i_k$ and $n^i_k$ at time $n$.

To animate the object, we have:

1) Rigid body has one external force.

2) Flexible body has one external force and one internal force.

$fg$ is the external force (gravity)

$fn$ is the internal force (mass spring)

$\Delta x^i$ indicates the time beginning with an arbitrary time $t0$.

$\Delta x^i = x^i (t0 + n\Delta t)$

$\Delta d(x^i, x^j)$ is the distance between observed object $i$ and $j$ at time $n$.

$\Delta d(x^i, x^j) = \sqrt{\sum (\Delta p^i - \Delta p^j)^2}$

$\Delta x$, $\Delta y$, and $\Delta z$ denote the distance of moving object in x direction, y direction, and z direction in each step time.

$|n-1 x^i_x - nx^i_x| - \Delta x$

$|n-1 x^i_y - nx^i_y| - \Delta y$

$|n-1 x^i_z - nx^i_z| - \Delta z$

To display objects moving smoothly, we offer the constraint, $\Delta x + \Delta y + \Delta z \leq 1$ for each time step.

**Euler integration**

We use Euler 1st order integration to animate an object. Euler integration consists of the following steps. First, set time to the initial value $t0$. Next, compute the level of time step by using an interval between frame to frame. Then, contribute the rate of change from time $n$ to time $n+1$. Euler integration is appropriate for ease of implementation (Desbrun et al 1999).

From Euler 1st integration we have

$\frac{dve^i_k}{dt} = f^i_k/m^i$

$\frac{dve^i_k}{dt} = f^i_k dt/m^i$

Next, we have

$n+1 v^i_k = n v^i_k + \left(\frac{f^i_k}{m^i}\right) dt$

Then, we have

$n+1 v^i_k = n v^i_k + n+1 v^i_k dt$

From this equation we get the position of each vertex in the flexible objects for each time step. We bounded the object with a bounding box (AABB) by finding the minimum point and maximum points in each flexible object. Then we can find the overlapping between the bounding boxes (AABBs). Next, we put the position of each vertex of object i to hash grid function.
Lemma 1: An object $x^i$ is not colliding with any object if they are not overlapping of bounding box volume.

**Proof:** This is obvious by definition of bounding box. The objects bounded with a box will collide if there is the intersection between bounding boxes.

**Collision Grid Domain (CGD)**

It is very efficient to use a hash grid function for spatial subdivision (Teschner et al. 2003). For each time step, we applied hash grid function to each vertex in the overlapping objects to determine the hash index for the animated objects with respect to a user-defined cell size. We can present it by following:

$$h = \text{hash}(nv^i_k, xl, yl, zl)$$

$$h = \left\lfloor \frac{\text{floor}(v^i_{kx}/xl) \times x_{\text{constant}} + \text{floor}(v^i_{ky}/yl) \times y_{\text{constant}} + \text{floor}(v^i_{kz}/zl) \times z_{\text{constant}}}{\text{hash index size}} \right\rfloor$$

Lemma 2: Given $n$ vertices $v_i^0, \ldots, v_i^n$, $v_i^j$ does not belong to CGD if object $x^i$ does not intersect with $x^0, \ldots, x^{i-1}, x^{i+1}, \ldots, x^m$ where $0 \leq i \leq m$ and $v_i^j$ is not in the considering grid cell.

**Proof:** Follows trivially from the definition of CGD. We use Lemma 2 to check if a vertex belongs to the CGD.

After the hash index and CGD have been determined, this technique computes the distance of vertices in the CGD. Using the distance, the algorithm can detect touching, hitting or throwing.

**Contact Surface (CS)**

This is the step that we find the distances of vertices in CGD and then we compare with collision tolerance. This step returns true or false. True is colliding while false is not colliding.

Lemma 3: Given $v_k^i$ as a point $p_i^k$ and $v_k^j$ as a point $p_j^k$. $v_k^i$ and $v_k^j$ belong to CGD. $v_k^i$ and $v_k^j$ are collide if $d(p_i^k, p_j^k) < \text{collision tolerance}$ and $v_{N_k}^i < \text{velocity tolerance}$ at time $n$.

**Proof:** By the definition of distance computation, we determine collision tolerance. If the distance between $p_i^k$ and $p_j^k$ is less than the collision tolerance, the vertices are going to touch each other.

Suppose there exist $m$ objects $(x^0, \ldots, x^m)$ in the simulation frame. Also, suppose each object is composed of vertices $v_i^0, v_i^1, \ldots, v_i^n$ where $0 \leq i \leq m$.

Lemma 2: Given $n$ vertices $v_i^0, \ldots, v_i^n$. $v_i^j$ does not belong to CGD if object $x^i$ does not intersect with $x^0, \ldots, x^{i-1}, x^{i+1}, \ldots, x^m$ where $0 \leq i \leq m$ and $v_i^j$ is not in the considering grid cell.

**Proof:** Follows trivially from the definition of CGD. We use Lemma 2 to check if a vertex belongs to the CGD.

After the hash index and CGD have been determined, this technique computes the distance of vertices in the CGD. Using the distance, the algorithm can detect touching, hitting or throwing.

**Contact Surface (CS)**

This is the step that we find the distances of vertices in CGD and then we compare with collision tolerance. This step returns true or false. True is colliding while false is not colliding.

Lemma 3: Given $v_k^i$ as a point $p_i^k$ and $v_k^j$ as a point $p_j^k$. $v_k^i$ and $v_k^j$ belong to CGD. $v_k^i$ and $v_k^j$ are collide if $d(p_i^k, p_j^k) < \text{collision tolerance}$ and $v_{N_k}^i < \text{velocity tolerance}$ at time $n$.

**Proof:** By the definition of distance computation, we determine collision tolerance. If the distance between $p_i^k$ and $p_j^k$ is less than the collision tolerance, the vertices are going to touch each other.
Our algorithm uses three passes to compute collision detection in each time step.

- First pass: we check if \( x_i \) overlaps with at least one of the objects \( x_0, x_1, \ldots, x_{i-1} \) or \( x_{i+1}, x_{i+2}, \ldots, x_m \) by using intersection of bounding boxes.
- Second pass: In case bounding box of object \( x_i \) overlaps with bounding box of object \( x_j \), we check if vertices \( v_{i1}, v_{i2}, \ldots, v_{in} \) and \( v_{j1}, v_{j2}, \ldots, v_{jn} \) are in the same grid cell or not. If so, we determine those vertices are in the CGD.
- Third pass: We calculate to see if a surface is contacted by another object.

In this algorithm, we can consider more than one pair of collision so that multiple flexible objects are detected for a large environment of simulation.

**Experiments and results**

We compared our algorithm with two other algorithms using five different simulations (A, B, C, D and E) to exam the relative performance of our algorithm. Simulation A, B, and C are composed of 315 vertices in each flexible object whereas D and E are created from 1189 vertices. For simulation A and D, we made 10 flexible objects in simulation, presented in Figure 2. Each object can be touching each other and colliding with a sphere. Simulation B is created with 100 flexible objects with one sphere while C and D are made from 100 flexible objects with 100 spheres shown in Figure 3 and 4 respectively. In case of D and E, we present how these features can affect our algorithms if there are more vertices in the flexible objects. Our scenarios are performed in an update method and the results of our algorithms are compared to the results of other algorithms.

<table>
<thead>
<tr>
<th>Model</th>
<th>Number of flexible objects</th>
<th>Number of spheres</th>
<th>Number of vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>1</td>
<td>3150</td>
</tr>
<tr>
<td>B</td>
<td>100</td>
<td>1</td>
<td>31500</td>
</tr>
<tr>
<td>C</td>
<td>100</td>
<td>100</td>
<td>31500</td>
</tr>
<tr>
<td>D</td>
<td>10</td>
<td>1</td>
<td>11890</td>
</tr>
<tr>
<td>E</td>
<td>100</td>
<td>100</td>
<td>118900</td>
</tr>
</tbody>
</table>

**Table 1: The specification of each test model**

In our experiments, we used our simulations with grid hash function algorithm, bounding Box (AABB) algorithm and our algorithm.
The results show that our algorithm can reduce time complexity even though there are more flexible objects or vertices in them.

### Conclusion and Future work

In this research, we have introduced the concept of tracking with bounding box (TB), collision grid domain (CGD), and contact surface (CS). Our method essentially extends the benefits of two leading existing approaches with concepts of contact surface. All flexible objects are surrounded by a bounding box with maximum point and maximum point. Then, vertices in overlapping bounding boxes are determined into collision grid domain by hash grid function. The vertices in same collision grid domain are, finally, observed for surface contact. The result of this work performed with 100k vertices showed that this algorithm is efficient collision detection for flexible bodies in real-time considering the time spent for animation.

This research can be extended to several ways:

- Simplify the operation process to optimize the algorithm that has been implemented in this paper.
- Simulate the structure of tree to determine the area of collision in the object body and perform the collision detection in the overlapped area to reduce time complexity.
- Create the suitable test cases to compare with different algorithm.

### References


