MUTI-LEVEL SB COLLIDE: COLLISION AND SELF-COLLISION IN SOFT BODIES

Jaruwan Mesit                     Ratan K. Guha                   Erin J. Hastings
jmesit@cs.ucf.edu                  guha@cs.ucf.edu               hastings@cs.ucf.edu
Department of computer science, University of Central Florida
4000 Central Florida Blvd., Orlando, Florida 32826.

Keywords
soft body simulation, animation, spatial subdivision, spatial hashing

Abstract
In interactive 3D graphics collision detection of soft bodies in real time is a significant problem. It is time consuming because soft bodies are composed of possibly thousands of moving particles. Each time step all particles rearrange in new positions according to their behaviors and collision must be detected for each particle. To optimize collision detection in soft bodies, we introduce a solution called Multi-Level SB Collide. The method relies on the construction of subdivided bounding boxes, box hash functions, and contact surfaces. Multi-level SB collide applies multi-level subdivided bounding boxes (AABBs) into a box hash function and uses contact surface method to detect collision. This contact surface can be used to detect both collision with other objects and self-collision in soft bodies. Experimental results show that multi-level SB Collide is an accurate and efficient method for real-time collision detection in soft bodies.

1. Introduction
Collision detection is a fundamental issue in most all forms computer animation. Many algorithms are based upon types of bounding volume hierarchies. Some notable examples are: 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). For polyhedral objects, CLOD with dual hierarchy (Otaduy and Lin 2003) has been also proposed whereas Separation-sensitive collision detection (Erickson et al. 1999) was presented for convex objects.

Another method for collision detect is spatial subdivision. Numerous variations have been proposed such as: octree (Moore and Wilhelms. 1988), BSP tree (Naylor et al. 1990), brep-indices (Bouma and Vanecek, Jr 1991), k-d tree (Held et al. 1995), bucket tree (Ganovelli 2000), hybrid tree (Larsson et al. 2001), and uniform spatial subdivision (Teschner et al. 2003). Some additional subdivision methods specifically suited for collision detection in large environments are I- COLLIDE (Cohen et al. 1995) and CULLIDE (Govindaraju et al. 2003).

However, most of the methods that have been mentioned are to solve collision detection between rigid bodies and other rigid bodies, or between soft bodies and rigid bodies, which is slightly less complex problem. In this paper we present a new efficient algorithm for collision detection among soft bodies, which is extended from Mesit et al. 2004. The algorithm is based on the classical methods of subdivided bounding volume and spatial hashing. The major contribution of this paper includes: 1) subdivided bounding box method, 2) the specialized hash function, and 3) the contact surfaces.

The paper will proceed as follows: first, an overview of the proposed algorithm, next analysis of the algorithm, then a set of the experiments with the algorithm and results, and finally conclusions and discuss possibilities of further work.
1.1 Introduction to soft body simulation

A soft body simulation is usually a set of particles, each of which has its own position, velocity, and force. Particles are connected to neighbors by linear springs. Spring length, elasticity, and damping factor are defined by physical properties of the soft body. Figure 1 shows a skeleton of cloth and figure 2 presents two more types of springs called shear and bend to simulate a soft body. The shear and bend properties are required to allow the soft body to stretch in and out.

1.2 Springs and Dampers

Springs are the structures that connect between two vertices. The function that we use to stretch or compress is spring force by Hook’s law. This relates to the “rest length” of the spring and a “spring constant”. The spring constant determines the stiffness of the spring. Damper is related to velocity of vertices, since each vertex has its own force and velocity. Damper acts against this velocity. If there are two vertices connected with springs, dampers slow the relative velocity between those two vertices. Figure 3 shows compressed spring and stretched spring.

2. Algorithm Overview

For collision detection with soft bodies in a large environment, we introduce three steps of the Multi-Level SB Collide algorithm:

1) Multi-level Subdivided Bounding Box (Multi-level SB): all soft bodies are surrounded by a bounding box (AABB) for tracking. Two points, a minimum and a maximum, define the bounding box. Then, the bounding box is subdivided to n-levels of subdivision. We use 3-level subdivision for this simulation.

2) Box Hash Function (BHF): we apply the box hash function to each point that we use to create the subdivided bounding box. One subdivided bounding box has 8 points. Each point is hashed and given hash index by box hash function. List of subdivided bounding boxes is put in hash table related to hash index. Then, we compute contact surface from vertices that belong to subdivided bounding box in the list of hash table.

3) Collision for Flexible Models (CF): finally we detect collision for the models. If distance between points in the flexible models is less than collision tolerance, collision for flexible models is detected.

2.1 Multi-level Subdivided Bounding Boxes (Multi-level SB)

The beginning of our algorithm is to do tracking with multi-level subdivided bounding box. We are tracking our soft bodies by using AABB bounding box. Minimum point and maximum point
are calculated to bound our soft body. The bounding box is subdivided in to \(2^n\) regions, where \(n\) is level of subdivision. Figure 4 shows 2-level of subdivision in which the second mid point is created from the first mid point.

**Figure 4: Mid points for subdivision**

These objects are subdivided by using the midpoint. Then, we find depth, height, and depth of each subdivided bounding box as shown in figure 5.

**Figure 5: 8 points for a subdivided bounding box**

### 2.2 Box Hash Function (BHF)

We use Box Hash Function (BHF). The idea is to use BHF and hash to 8 points that we use to create the subdivided bounding box. To facilitate hashing, initially a hash table is created which is based on grid size. The formula is as follows:

\[
\text{Index} = (\text{floor}(\text{Grid.min.x} / \text{length}) \times x_{\text{prime}} + \text{floor}(\text{Grid.min.y} / \text{height}) \times y_{\text{prime}} + \text{floor}(\text{Grid.min.z} / \text{width}) \times z_{\text{prime}}) \mod \text{bucketsize};
\]

where 
- \(\text{length}\) is grid length,
- \(\text{height}\) is grid height,
- \(\text{width}\) is grid width,
- \(\text{Grid.min.x}, \text{Grid.min.y}, \text{and Grid.min.z}\) are minimum points of each grid,
- \(x_{\text{prime}}, y_{\text{prime}}, \text{and} z_{\text{prime}}\) are any prime number for \(x, y,\) and \(z\).

Next, we apply BHF to 8 points of our subdivided bounding box shown in figure 6.

**Figure 6: Points of subdivided bounding box are hashed by hash function**

Since there are 8 points, \([0..7]\), in one subdivided bounding box, BHF can be written as:

\[
\text{HashIndex} = (\text{floor}(\text{point}[0..7].x / \text{length}) \times x_{\text{prime}} + \text{floor}(\text{point}[0..7].y / \text{height}) \times y_{\text{prime}} + \text{floor}(\text{point}[0..7].z / \text{width}) \times z_{\text{prime}}) \mod \text{bucketsize}.
\]

where \(x_{\text{prime}}, y_{\text{prime}},\) and \(z_{\text{prime}}\) are any prime number for \(x, y,\) and \(z\).

Length, height, and width are length, height, and width of grid cell.

When we have the hash index, we can put the subdivided objects to hash table. At this step, we create one hash table and we have the list of subdivided objects that have chances of collision. Figure 7 and 8 show that subdivided box 2 (SB2) and 3 (SB3) are in the same hash index which is index 6. Then, contact surface will be calculated.

**Figure 7: The combination of grid and box hash function**

These subdivided bounding box are in the same hash index.
2.3 Self-collision and Collision for flexible models (SCF and CF)

In this step we find the distances of vertices in the hash table and then we compare with a collision tolerance. This step returns true or false. True is colliding while false is not colliding. Contact surface can be determined as follows:

Given position of object A and B, d is the distance between them. If they are in same hash table, then we find for d. if $d < \text{collision tolerance}$, then they collide.

Time $t = n$, $d > \text{collision tolerance}$, they don’t collide.

Time $t = n+1$, if $d < \text{collision tolerance}$, they collide.

2.4 Self-collision in flexible object (SCF)

For every soft body object, it is necessary to solve self-collisions for internal constraints. Self-collision in soft bodies can be achieved by using the contact surfaces. The method of self-collision avoidance is to create collision tolerance around the points of flexible object. This collision tolerance force acts like a shield which rejects the points passing through the objects. Thus our algorithm solves for both self-collision detection and normal collision detection.

To summarize the algorithm, three major steps are computed for collision detection in every frame. First, the bounding boxes are subdivided into n-level of subdivision (3 levels in this simulation). Then, we apply the box hash function to subdivided bounding boxes and create hash table. Finally, for each box hash table, we compute the contact surface to the vertices in the subdivided bounding box. As a property hash table we consider more than a single pair of collisions. Thus multiple soft bodies are detected for collision in a large simulation environment.

3. Analysis

First we analyze the method with regard to time complexity. Since the proposed method is using hash table, performance depends on subdivided bounding box distribution in the scene.

- **Best Case**: $O(1)$. In this case every subdivided bounding box hashes to a different grid cell. No collision will be performed at all.
- **Worst Case**: $O(N^2)$. In this case, every subdivided bounding box in the scene is in the same grid cell. $N$ is number of vertices.

The benefit of subdivided bounding box is to reduce cost of hash function. In stead of applying hash function to $N$ vertices in the scene, the proposed method spend only $O \log_2 N$.

- **Without subdivided bounding box**: $O(N)$. In this case every vertex has to be applied to hash function.
- **With subdivided bounding box**: $O(\log_2 N)$. In this case every vertex doesn’t have to apply to hash function. Only 8 points of subdivided bounding box have to be applied to hash function. $N$ is the number of vertices and $n$ is level of subdivision.
4. Experiments and Results

A simulation was developed using: Code (C++), Graphics (OpenGL), and Compiler (Microsoft Visual Studio). Figure 9 presents a soft body which is created from 162 vertices and 320 faces. We compare our proposed method to Grid Hash Function, and Bounding Box, Bounding Box and Grid Hash Function. Four different scenarios are created and called A, B, C, and D, in which all soft bodies are falling down and colliding with a sphere and ground surface shown in figures 10, 11, 12, and 13. There are 1k, 5k, 10k, and 15k faces in scenario A, B, C, and D respectively. For the hash table a bucket size of 101 is applied in scenario A and B, while a bucket size of 1001 is created in scenario C and D.

Initially in Figure 10 there are a solid sphere (below) and a soft body (above). Then the soft body collides with the sphere as shown in figure 11. Later on, there are more soft bodies falling down and colliding with other soft bodies presented in figure 12 and 13. They deform with vertices moving based on the laws of physics, according to detail in Maciej and Ollila, 2003. The simulation has been captured at frame 100, 400, 600, and 800.

Figure 9: structure of a soft body

Figure 10: Frame 100 showing a solid body sphere (below) and a falling soft body object (above)

Figure 11: At frame 400 a soft body object has collided with the solid body sphere

Figure 12: At frame 600 multiple soft body spheres have dropped upon each other, showing both soft-soft and soft-solid collision

Figure 13: At frame 800 gravity and friction eventually drop the soft body objects to the floor
The experiment shows that using only a grid hash function yielded worst performance for simulations A, B, C, and D. The bounding box algorithm and combined bounding box with a grid hash function gave slightly better results. Multi-Level SB Collide gave the best performance, achieving significantly better frame rate in all four scenarios. The results of all experiments show that our algorithm can reduce time computation in comparison to the other methods.

5. Conclusions and Future Work

Multi-level SB collides presents the concept of tracking with Multi-level Subdivided Bounding box (Multi-level SB), box hash function (BHF), Self-collision and collision in contact surface (SCF and CF). The main strength of the multi-level SB collide lies in the efficiency in collision detection in soft bodies, only requiring multi-level bounding boxes and applying into box hash function. Our method essentially extends the existing approaches, multi-level subdivided bounding box and box hash function, with the concept of contact surfaces. All soft bodies are surrounded by a bounding box with maximum point and maximum point. Then, box hash table is defined by subdivided bounding box and box hash function. The vertices in same box hash list are, finally, observed for surface contact for self-collision and collision. Another benefit of this algorithm is that we can consider more than one pair of objects so that multiple, simultaneous soft body collisions are detected in a large environment of simulation. The result of this work performed with 10-15k faces showed that this algorithm is efficient for detecting collision for soft bodies in real-time considering the time spent for animation.

As for future work, it is possible to simplify the operation process to optimize the algorithm that has been implemented in this paper. Also, a tree structure could possibly be implemented to determine the area of collision in the object body and perform the collision detection in the overlapped area. However, a tradeoff would be introduced of additional time required to modify the tree structure in each time step.

6. References


---

Table 1: The average running time (frame per second) in each model on a typical laptop: Pentium-4 3.2Ghz, 1Gb RAM, Nvidia GForce Go GPU.

<table>
<thead>
<tr>
<th>Model</th>
<th>Grid Hash Function (frame per second)</th>
<th>Bounding Box (frame per second)</th>
<th>Bounding Box and Grid Hash Function (frame per second)</th>
<th>Multi-level SB collide (frame per second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>20</td>
<td>22</td>
<td>40</td>
<td>70</td>
</tr>
<tr>
<td>B</td>
<td>18</td>
<td>16</td>
<td>35</td>
<td>57</td>
</tr>
<tr>
<td>C</td>
<td>14</td>
<td>13</td>
<td>31</td>
<td>49</td>
</tr>
<tr>
<td>D</td>
<td>13</td>
<td>14</td>
<td>25</td>
<td>41</td>
</tr>
</tbody>
</table>

---

As for future work, it is possible to simplify the operation process to optimize the algorithm that has been implemented in this paper. Also, a tree structure could possibly be implemented to determine the area of collision in the object body and perform the collision detection in the overlapped area. However, a tradeoff would be introduced of additional time required to modify the tree structure in each time step.

6. References


