14th International Conference on Geometry and Graphics
August 5-9, 2010, Kyoto ISBN 978-4-9900967-1-7

A FAST CULLING SCHEME FOR DEFORMABLE OBJECT COLLISION DETECTION USING SPATIAL HASHING

Sunhwa JUNG and Min-Hyung CHOI
University of Colorado Denver, Colorado, USA

ABSTRACT: Fast culling for collision detection is one of the key components in dynamic simulation. While culling for collision detection works well when there is a smaller number of collisions compared to potential collision pairs in a given scene, it can be inefficient when most objects are close to each other and exact collision detections are required. Culling process must be simple and fast with minimal overheads. We describe a fast culling scheme using a novel spatial hashing method with inner-voxel culling (a culling process among primitives in each voxel). Previously, spatial hashing techniques have been used for culling, and exact collision detections were performed between all primitive pairs in a voxel. But due to hash collisions and the inefficiency of fixed grids, unnecessary exact collision checks are mandated, which substantially hamper the culling performance. We improved the culling performance of spatial-hashing by adding inner-voxel culling that consists of two levels: primitive level and element level based on 26-discrete oriented polytopes and surface normal. In addition, our algorithm is parallelized to utilize multi-core processors. Our method performs both inter-collision and self-collision in a linear runtime on the input primitives. Moreover, our method needs no preprocessing and has no assumption or limitations on topology and accuracy.

Keywords: Collision Detection, deformable objects, spatial hashing, parallel computing.

1. INTRODUCTION

Culling is a process to find potential collision pairs of primitives (e.g., triangles or polygons) for further computation that finds detailed collision information such as colliding time, colliding location, separating distance, and penetration depth. Culling is inevitable for a fast collision detection and resolution in many applications. Culling process for deformable objects is especially challenging due to the frequent changes of object shape and topology. There are mainly four different culling approaches for deformable objects. Bounding volume hierarchy (BVH) has been extensively employed for culling for inter-primitive collisions. The limitation of BVH is on handling fracturing or cutting because the hierarchy structure has to be rebuilt. Graphics process unit (GPU) based method utilizes powerful modern GPUs to accelerate culling process. But the major intrinsic limitations are the size of memory and the bottle neck between CPU and GPU because GPU is not flexible enough to perform all necessary computation. The precision of GPU floating point calculation might not be accurate enough for some applications. Although Voronoi computation itself is expensive, Voronoi based method demonstrated efficient discrete Voronoi region calculation on GPUs.
Unlike BVH-based culling and GPU-based culling, spatial subdivision method has no limitation in topology changes or in numerical accuracy (floating point computation), and it can perform self-collision check along with inter-collision check at the same time without extra cost. The previous huddle of spatial subdivision method has been the limitation of

memory size to hold the infinite three dimensional space and the computational cost of subdivision processing. But employing spatial hashing with a regular grid, subdivision processing became trivial and a smaller size of memory can be used to hold infinite cells.

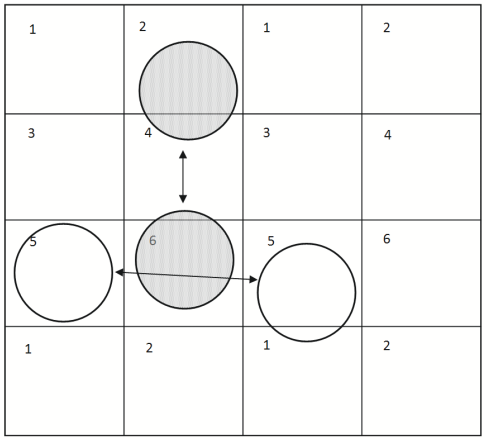

Figure 1: The voxel indexes (1-6) for grid cells are at the left top corner of each cell. Gray dots and white dots are unnecessary collision check pairs. The gray pair is well separated, but they are mapped into voxel 4. The white pair is mapped into voxel 5 due to hash collision.

Despite the recent improvement on spatial subdivision using spatial hashing, it is still considered slower than a Voronoi based method [15]. The inefficiency of spatial hashing based method is caused by the unnecessary collision checks. In Fig. 1, the cases of unnecessary collision checks are illustrated caused by hash collision and the inefficiency of a grid set-up. Hash collisions can be removed by using perfect hash functions, but it may require pre-processing and extra storage. The inefficiency of a grid set-up is originated from the nature of spatial hashing where objects are spread among neighboring regularly divided voxels. We resolved the inefficiency of spatial hashing by adding an inner-voxel culling, which substantially alleviates the effect of hash collision. Specifically our empirical studies in various setup consistently shows that our method improves the performance of a spatial hashing based culling by approximately three orders of magnitude. Our method has no limitations or assumptions of accuracy and topology change for both inter- and self-collisions. Furthermore, we utilized parallelism to take benefits of available multi-core architecture. Our method has been implemented on dual-core and quad-core using the open multi-processing (OpenMP) application program interface (API) [33].

2. PREVIOUS WORK

Overall collision detection is surveyed by Jimenez et al [4] and collision detection among deformable objects is well summarized by Teschner et al [5]. In this section, we will highlight previous culling methods in four categories (BVH-based, GPU-based, Voronoi-based, and spatial-hashing-based) and consider parallel collision detection for deformable object simulation.

2.1 BVH-Based Methods

There are many approaches based on BVH for deformable object simulation such as aligned axis bounding box (AABB) [17], [18], [19], spheres [20], [21], oriented bounding box (OBB) [22], and discrete oriented polytope (DOP) [23]. The challenge of using BVH is the cost of updating the hierarchy. Hierarchy update [24] for limited-deformation models such as meshless deformation and modal analysis can be done very efficiently. But still it is not efficient for self-collision. In order to improve performance for self-collision an additional culling process using curvature of surface must be performed. It is quite expensive and inefficient for objects like cloth patches are folding itself with busy pattern.

2.2 GPU-Based Methods

Many researchers utilize advanced GPU power for collision detection. Image-based collision detection has been introduced for

convex objects [27] and extended for non-convex objects [28]. Layered depth image (LDI) has been proposed for objects with closed surfaces [29]. These methods highly depends the image resolution. The other trend of GPU-based methods is utilizing hierarchy [30], [31], [32]. All these methods did not deal with the limitation of GPU such as the size of memory and delay on read-back. Govindaraju et al [33] achieved a fast culling only for self-collision and recently Zhang and Kim proposed AABB streaming [26] to overcome the limited memory size and minimize the read-back data, but they ignore self-collision. Each method is fast, but cannot perform inter-object collision and self collision at the same time.

2.3 Voronoi-Based Methods

A fast culling method for deformable objects has been achieved using discrete Voronoi computation on GPUs [15]. This method has no limitations on topology change, but there is a minimum computation cost to set up GPU.

2.4 Spatial-Hashing-Based Methods

Spatial hashing has been successfully employed in the spatial sub division method for collision detection of rigid body simulation and deformable object simulation [1], [3], [6]. A spatial hashing for deformable objects proposed by Teschner et al. [3] used a limited size of data structure and a simple spatial hash function for tetrahedron. They computed only static collision contact triangle and barycentric coordinates. Recently Eitz and Lixu [16] introduced a hierarchical spatial hashing using one dimensional array with DJB2 hash function. Their method ran consistently for different cell sizes with a slight computational overhead, but the issue of hash collision was not considered seriously. A perfect spatial hashing was tried with pre-computation and applied to collision detection between objects by Lefebvre and Hoppe [7]. Although each voxel will take one spatial cell, a perfect hashing requires a large number of voxels to hold all the cell and it has to know the number of voxels a priori. For deformable objects simulation the number of required voxels changes at each step. Moreover, the irregularity of the number of triangles in voxels is inevitable in regular voxel grids because the mesh has different resolutions. All previous methods decomposed the problem for static collision detection without considering the unnecessary collision checks caused by hash collisions or by the result of inefficient grid set-up. However, our experiment shows that the unnecessary collision checks are a major problem on spatial-hashing-based culling.

2.5 Parallel Collision Detection

Many different approaches have been proposed for parallel collision detection. Voxel-based parallel collision detection [2] is implemented to determine colliding objects on CHARM++, which is the object-based parallel programming environment with a dynamic load balancer. Their design needs many communications between nodes because they used distributed memory for voxels. Especially for the objects overlapped in many places, communication overhead impedes the performance. While a linear scalability of parallel processing can be achieved, the performance of voxelization is slow because it is based on culling without computing crucial collision information such as collision points, penetration depth and colliding time. They have focused on collision detection only between static objects. The performance and the scalability can be lowered by applying exact collision detection for close primitive pairs in a highly dynamic environment. Overlapping axis aligned bounding box (OAABB) collision detection scheme [8] was proposed in parallel processing using OpenMP in a shared memory system. OAABB can be efficient in inter-objects collision detection, but it cannot work efficiently to self-collision. Parallel collision detection for cloth simulation [9] is proposed

on a distributed memory system creating local task pools of AABB collision checking using a hierarchy structure. Employing AABB in hierarchy will not perform efficiently for self-collision in parallel computing, since objects are often close enough, and therefore all recursive searches must reach to the leaf nodes of the hierarchy.

We extended spatial-hashing-based method for parallel computation to handle general deformable objects. Our parallel scheme is implemented independently from modeling or numerical solvers, so that it can be coupled with any other parallel simulation techniques. We examined our method by conducting three benchmarks with many collisions and contacts.

3. OVERVIEW

The key idea of our method is working over the input primitives directly regardless of the number of objects, so the runtime complexity of our method is O(n) where n is the number of primitives (e.g., triangles or tetrahedral). Our algorithm consists of four steps: updating the 26 DOP of each triangle, spatial hashing (mapping AABB of triangles into the fixed number of voxels), culling of the primitives in each voxel using 26-DOPs overlapping test and surface normal test, and exact collision detection with 26-DOPs overlapping tests for the element pairs. Compared to a traditional spatial-hashing-based method, the two culling steps are added. No preprocessing or extra data structure is needed for our method. The whole process is illustrated in Fig. 2.

4. METHOD

4.1 26 DOP Update

The planes of 26-DOPs have following normal vectors; <1,0,0>, <0,1,0>, <0,0,1>, <1,1,0>, <1,-1,0>, <0,1,1>, <0,-1,1>, <1,0,1>, <1,0,-1>, <1,1,1>, <1,-1,1>, <-1,1,1>, <-1,-1,1> and their opposite vectors. Since many applications require exact collision detection for both continuous collision detection and distance calculation, the 26-DOPs of each primitive cover the entire trace of the primitive in a given period. The threshold of distance calculation (e.g., the thickness of repulsion force field) is added to the total DOPs. Overall DOP update is quick because there is no hierarchy to traverse. For further optimization, the DOPs of static objects in a given scene are not updated.

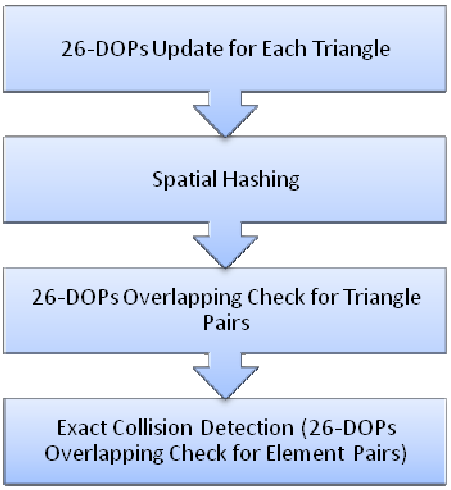

Figure 2: Four steps of culling based on spatial hashing are shown. The proposed two-level culling is at the third and the fourth step. The element level culling and exact collision detection are combined to save the storage of potential collision pairs.

4.2 Spatial Hashing

In order to perform spatial hashing on objects, the size of a cell and the number of voxels for a given scene must be known before mapping primitives to voxels. The size of a cell and the number of voxels do play an important role in performance optimization but there is no definitive formula for optimal values due to the diversity of geometric structures, granularities of primitives, and topology changes of deformable structures. As Teschner et. al [3] suggested, a large hash table size would contribute to the reduction of hash collisions, but too large hash table size could incur memory management issues to actually decrease the overall performance.

The cell grid size should also strike a balance; the number of primitives in a cell should be reasonably controlled and a primitive should not be disperses to many neighboring cells.
In our algorithm, the size of cell and the number of voxels are computed in the beginning of each simulation once and kept constant during the simulation. The length of the largest axis of 26-DOPs of triangles is used for the cell size, and 20 percent of the number of triangles is used for the number of voxels. While these parameters can be further tuned, it is rather a conservative arrangement to warrant general applicability and objective performance evaluation. When objects in the scene do not move, the objects are voxelized once and keep the information for each voxelization.

4.3 Inner Culling

In conventional spatial hashing method, all candidate primitive pairs are thrown into exact collision if culling is failed on them. Inner culling method goes one step further to see if the primitive pairs do require exact collision detection before performing exact collision detection. Inner culling is performed in two levels: the 26-DOPs of primitives with surface normal test and for the 26-DOPs of elements (point, and edge). 26-DOPs of each primitive are updated and stored at the beginning. 26-DOPs of elements are calculated later for the element level culling.

Primitive Level Culling: As the first step in inner culling procedure, 26-DOPs overlapping tests for the primitives in each voxel are performed. Previous approaches performed direct collision checks for all primitives in a voxel. Since 26-DOPs are tightly approximating the primitive, the number of possible collision pairs for the exact collision detection has been reduced significantly. The primitive pairs mapped into the same voxel due to the hash collisions can be culled out efficiently in this step. When building the list of the potential collision pairs, there might be duplicated pairs. However since the primitives are well culled by spatial hashing, the number of the duplicated potential collision pairs are relatively small. If we want to completely eliminate the redundancy, the algorithm becomes very expensive while the expected performance advantages would be relatively small to justify the overhead.

Surface Normal Test: 26-DOPs overlapping tests for the primitives are not efficient for self-collision among thin objects like cloth because neighboring triangles are always overlapping. After performing 26-DOPs primitives overlapping test, if the potential collision pairs are in neighboring each other (sharing at least one node), the dot product of the surface normal vectors of two adjacent primitives is calculated. If the dot product of the surface normal vectors is greater than the threshold (-0.9 or close to -1.0), the pair is culled out from exact collision detection. The threshold should be different for the thickness of the repulsion force field. This may allow self collisions by degenerate elements (see figure 3) undetected, but this exceptional case should be better handled and prevented by the more accurate modeling.

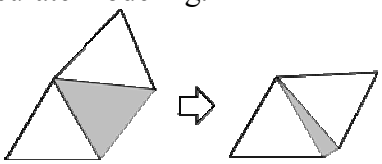

Figure 3: The gray degenerate (very thin and long) triangle is located between two normal neighboring triangles.

Element Level Culling: For further culling after the primitive level culling, we employed the element level culling which is inspired by the work of Hutter and Fuhrmann [14]. Before performing 15 triangle/point and edge/edge collision checks between two triangles or any other primitive pairs, we calculated the 26-DOPs of points and edges. 26-DOPs overlapping tests were performed between the 26-DOPs of primitive and the 26-DOPs of

element, and between the 26-DOPs of element and the 26-DOPs of element. Hutter and Fuhrmann [14] employed lower order 18-DOPs for the element culling, but we found that the 26-DOPs are more efficient and the overhead for calculating more degrees of DOP is trivial.

4.4 Exact Collision Detection

The possible element pairs are checked to find the exact collision time and the collision location (barycentric coordinates) by the coplanar condition method described by Bridson et al [18]. The exact collision detection has been performed with the element level culling of the 26-DOPs overlapping test to save the storage for the possible collision pairs. If a collision is found, the redundant check is performed over the collision result (the list of triangle/point pair and edge/edge pair) to remove the redundant collision detection pairs. After this process, the collision detection result is reported for collision resolution (e.g., the list of triangle/point and edge/edge pairs with collision time and collision location: barycentric coordinates)

5. PARALLELIZATION

The parallelization of our method has been done in four separate steps because the shared memory for voxels and collision results must be synchronized for the next step after each step is finished. In the 26-DOPs update step, there is no shared data. Only one time synchronization is required after each processor performs update for the 26-DOPs of primitives. In the spatial hashing step, the memory of voxels is shared and maintained by applying critical regions for tagging a cell to its voxel and adding a primitive to the voxel. In the primitive level culling step, the memory of the potential collision pair list is shared. In the element level culling and exact collision detection, the list of the collision pairs is shared and maintained by applying critical regions on add-operations. After each step, the shared memory is synchronized among all the participating processors.

6. EXPERIMENTS

6.1 Benchmarks

Our method was applied to three different simulation scenarios involving many collisions and sustained contacts to analyze the performance in various conditions.

Benchmark 1(14 tori simulation): each torus consists of 800 triangles and 14 tori are located randomly at the initial state and piled up on the floor. The 14 tori are modeled by mass-spring system similar to the model described by Ko and Choi [35]. This example has an increasing number of collisions (for both inter/self collision) as the 14 tori pile up. The total simulation step is 1,900 and the snapshots of 14 tori simulation are shown in figure 4.

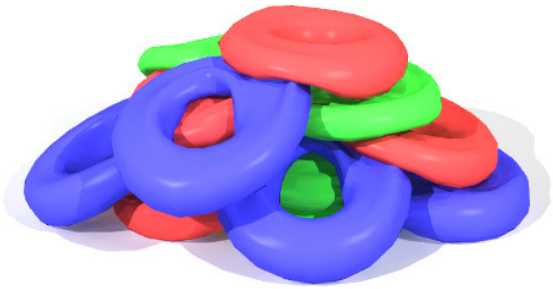

Figure 4: 14 deformable Tori are piled up

Benchmark 2(Bunny and cloth simulation): a cloth patch modeled by 100 by 100 particles and 20K triangles is falling on the Stanford bunny model with 60K triangles. The cloth patch is also modeled with mass-spring system similar to the 14 tori simulation. The total simulation step is 4,500. The snapshots of bunny and cloth simulation are shown in figure 5. In this example, the bunny model is once mapped to the voxels and the resulted voxels are kept throughout simulation without update.

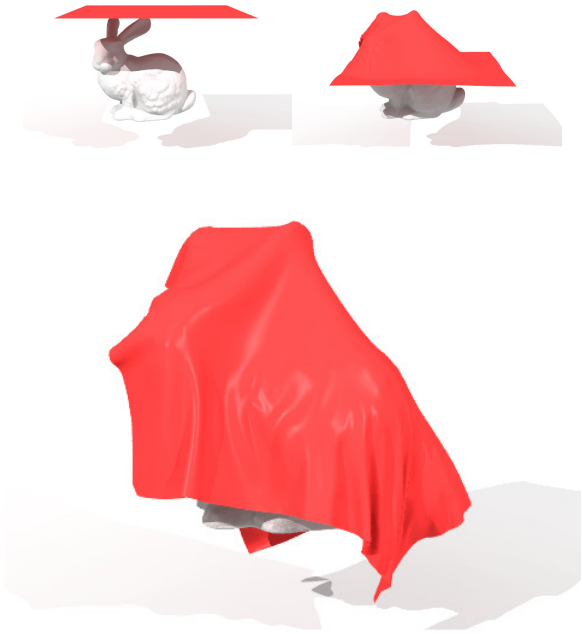

Figure 5: Bunny and cloth simulation: at the time step 1 (top left), 2050 (top right), 6000 (bottom).

Benchmark 3(1,014 alphabets simulation): each letter is modeled with approximately 900 triangles and the total triangles in the given scene are about one million. 1,014 letters are falling to the floor with random initial condition as shown in Figure 6. The letters are modeled by meshless deformation using FastLSM [34] and free-form deformation (FFD) AABB is used as the primitives for collision detection and resolution. The letters have 11 lattices, and the total primitives are 11,000. Collision detection and resolution is done by using FFD AABB which is distance calculation for each FFD AABB.

Figure 6: 1,014 deformable alphabets

6.2 Primitive Level Culling Result

The primitive culling result is compared with the spatial hashing result (the sum of the collision pairs in each voxel: n(n-1)/2, where n is the number of primitives in a voxel). After the primitive level culling, there might be redundant pairs in the list, but still our method consistently reduces the number of collision pairs by the minimum of two orders of magnitude for all three benchmarks. It shows that the unnecessary collision checks are major bottleneck of spatial-hashing-based culling and our inner culling method plays a central role in improving the overall efficiency of algorithm. The numbers of potential colliding pairs are compared and shown in figure 7 in log scale. Benchmark 2 and 3 results are almost identical in terms of the performance improvement rates. The hash table sizes are 2,239, 17,889 and 2,000 relatively for benchmark 1, 2, and 3. Culling is very efficient at the first iteration because there is no collision. After the primitive level culling, the number of potential colliding pairs is significantly reduced. While collisions are increasing, the speed-up of the primitive culling is maintained roughly at two orders of magnitude.

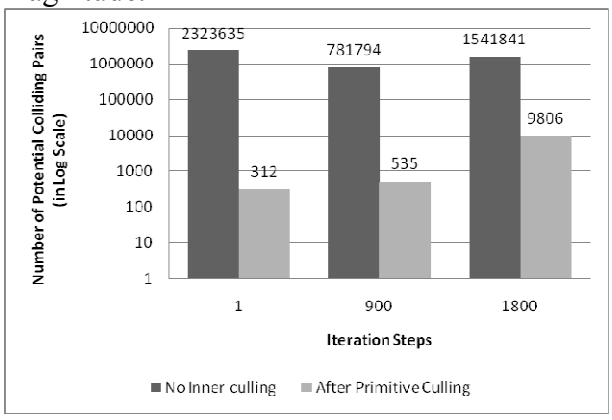

Figure 7: Primitive level culling speed-up result for benchmark 1.

6.3 Element Level Culling Result

When exact collision detection is performed for the potential colliding pairs of primitives without the element level culling, the number of element pairs for exact collision detection