Pacific Graphics (2014)
J. Keyser, Y. J. Kim, and P. Wonka (Editors)

Short Papers

Real-time Collision Detection with Two-level Spatial Hashing on GPU

Yang Hong[1], Wen Wu[1,2] and Hui Chen[3]

[1]Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, Macao
[2]Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong
[3]Beijing Key Lab of Human-Computer Interaction, Institute of Software, Chinese Academy of Science, Beijing, China

Abstract

In this paper, a two-level parallel spatial hashing method is presented for real-time collision detection of deformable objects based on modern GPU architecture. The second-level of spatial hashing is used to improve the culling efficiency. Moreover, a novel encoding method on GPU is proposed to compensate the inflexibility of the GPU memory system. It can efficiently determine the colliding pairs of primitives between deformable objects. The experimental results show that our method can perform high culling efficiency with low memory cost.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

1. Introduction

The goal of collision detection is to determine if any collisions occur between geometric models. For most simulation environments, for example, in motion planning, physics simulation and games, the final results are significantly influenced by the measurement of collision detection. In complex simulation scenarios, collision detection is always considered as the major computational bottleneck. Different from rigid bodies, the deformation of objects causes the change of the bounding volumes and sometimes leads to the self-collision problem. Therefore, it is more challenge to deal with the collision detection for deformable objects.

In this paper, a two-level parallel spatial hashing method for real-time collision detection between deformable objects on GPU is presented. Firstly, the geometric primitives in the possible overlapping region of the deformable objects are mapped into a coarse spatial grid using a general perfect hashing function. Subsequently, for those cells which include the geometric primitives of different objects, the subdivision of the cells is made in order to improve the culling efficiency. To take full advantage of the parallel accessibility and to reduce the cost of the limited global memory on GPU, a novel encoding method in the GPU shared memory is presented to determine the colliding pairs of primitives between the deformable objects. Compared to the traditional spatial algorithms, test results demonstrate that our method perform high culling efficiency with low memory cost.

The details of our method are presented in section 3. The experimental results and discussion are presented in section 4. Finally, a conclusion is made in section 5 with the further work.

2. Related Work

A good survey of collision detection for deformable objects may be found in [TKH*05]. Here, we focus on the parallel collision detection methods for deformable objects.

The parallel collision detection methods for deformable objects can be roughly categorized into three classes: the Bounding Volume Hierarchies (BVH), the spatial hashing methods, and the image-based methods.

In [ZK07], a streaming-based BVH method was proposed on a GeForce 7800GTX graphics card. Their method performed parallel BVH update and traverse by invoking the pixel shader. However, the method cannot handle large models due to the limitation of texture size on GPU. In order to reduce the cost of renewing AABB structure during the deformation, various front-based methods have been presented on multi-core CPU or GPUs [TMT10, TMLT11, ZK14]. A front-based decomposition method on a multi-core CPU for fast parallel BVH traversal can be found in [TMT10]. The

DOI: 10.2312/pgs.20141249

method highly improved the traversal performance. However, the memory cost was enormous when handling complex objects. In [TMLT11], a GPU-based front method was proposed. The collision streams were exploited to avoid the lock operations. Moreover, the deferred front node was provided in order to decrease the high memory cost. A recent published method in [ZK14] provided a great improvement of the front-based method on many-core processors. The p-partition front nodes were used instead of the exact BVTT front nodes in order to reduce the high memory cost. However, the scalability of their method is sensitive to BVTT and it is less efficient for unbalanced BVTT. Moreover, the time spent in finding p-partition is non-trivial for high number of processor cores.

The image-based and the spatial-based collision detection methods for deformable objects do not required any precomputation. Various image-space collision detection techniques have been discussed in [HTG04]. In [BW03], for instance, the depth and stencil buffers were used to check the intersections between deformable objects in image space by using the rasterization capability of the graphics card. However, the accuracy of these methods is limited by the viewing direction and the resolution of viewport. Moreover, transferring data of depth buffer between CPU and GPU highly influence the performance.

To exploit the high parallel computation power of modern GPUs, a hybrid CPU/GPU spatial method was proposed in [PKS10]. The broad phase culling was implemented by the parallel spatial hashing with multi-GPUs. However, the narrow phase of the triangle intersection test was simply handled by the brute-force method. The computational cost was affected by the cells which contain a large number of triangles. Therefore, the overall performance of the parallel processing was slowed down. In [FWZS11], the parallel radix sort and compaction computation were utilized to implement the parallel spatial hashing with multi-GPUs. Although the method is considered to achieve high parallelism, the requirement for high resolution of uniform grid causes the computation quite expensive.

3. Two-level spatial hashing

3.1. First level of parallel spatial division

For spatial hashing methods, the resolution of the grid highly influences the culling efficiency and the memory cost. The resolution of the first-level of the spatial grid in our method is determined as follows [WIK*06]

$$R_x = d_x\sqrt[3]{\frac{\lambda N}{V}}, R_y = d_y\sqrt[3]{\frac{\lambda N}{V}}, R_z = d_z\sqrt[3]{\frac{\lambda N}{V}} \qquad (1)$$

where d_x, d_y and d_z are the diagonal of the AABB of the object in x, y and z dimension, respectively; N represents the number of primitives of the object in the collision bounding volume; and λ is the density coefficient which determines the degree of the subdivision. λ was decided experimentally in our method. The grid is represented into a linear array by a general, perfect hashing function

$$H(x,y,z) = z * R_x * R_y + y * R_x + x \qquad (2)$$

where (x,y,z) are the coordinates of the cell's centroid and R_x, R_y and are the resolution of the uniform grid in x, y dimension, respectively. The AABB of each geometric primitive of deformable objects is mapped into the grid in parallel by a kernel function as follows

$$(l_i^{min}, l_i^{max}) = \left(\lfloor \frac{B_i^{min} - G_i^{min}}{c_i} \rfloor, \lfloor \frac{B_i^{max} - G_i^{min}}{c_i} \rfloor \right) i \in \{x,y,z\} \qquad (3)$$

where l is the overlapped region of the primitive's bounding volume in a cell; c is the cell size of the uniform grid; B and G represent the AABB of each geometric primitive and the bounding volume of the region covered by the grid, respectively. The atomic operation in CUDA is used to avoid the access conflict when multiple primitives are mapped into the same cell.

3.2. Second level of parallel spatial subdivision

For the cells which include the geometric primitives of different objects, the second-level of parallel spatial subdivision is processed. The hashing table of the subdivided grid is stored in the shared memory of GPU in order to reduce the memory cost and high access latency of the global memory. The shared memory can be simultaneously accessed and manipulated by multiple threads [NVI11]. The maximum size of the shared memory on the modern GPUs is 48KB. Owing to the limited size of the shared memory, the primitives cannot be directly mapped into the static hashing table. A novel encoding method in GPU is proposed in our method to resolve the problem.

The status of the primitive's AABB in a cell is encoded by a bit value. A value of 1 indicates the primitive's AABB exists in the cell. Otherwise, it does not. For each set of 4 by 4 by 4 uniform grid, the statuses of a primitive are packed into a 64-bit long integer. For simplicity, the encoding process for a 4 by 4 grid in a 2D case is illustrated in Figure 1. $T1$ and $T2$ are the primitives of body a, and $T3$ and $T4$ are the primitives of body b. The AABBs are displayed in the shaded rectangles. The encoding result of each primitive is shown individually for clarity.

Instead of a brute force test to find the possible colliding pairs, a further culling based on the endcoding information is applied. The primitives information of each cell is required from the encoding of each primitive. However, they are stored in multiple threads. To encoding them directly will cause the access confliction. Therefore, a transpose operation is applied to obtain the primitives information per cell from the primitives encodings. The cells, which are empty or only have primitives of one object, are culled. Then a logic AND operation is carried out on the bits of different objects

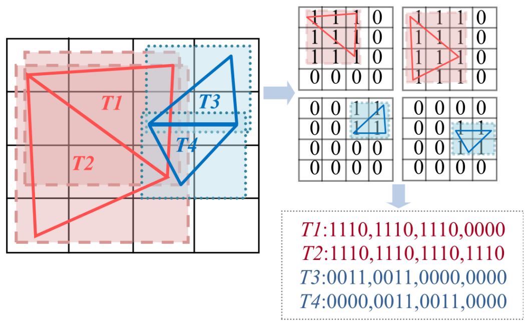

Figure 1: *An encoding example for 4 by 4 grid in a 2D case. If the AABB of triangle overlaps the ith cell, the ith bit of the bit mask is set to 1. Otherwise, the bit is set to 0.*

to acquire the possible colliding pairs (see Figure 2). Subsequently, these possible colliding pairs are further subject to the intersection test. In comparison to the time complexity of the brute force test which is $O(n^2)$, our method has a lower time complexity $O(1)$ and a higher parallelism.

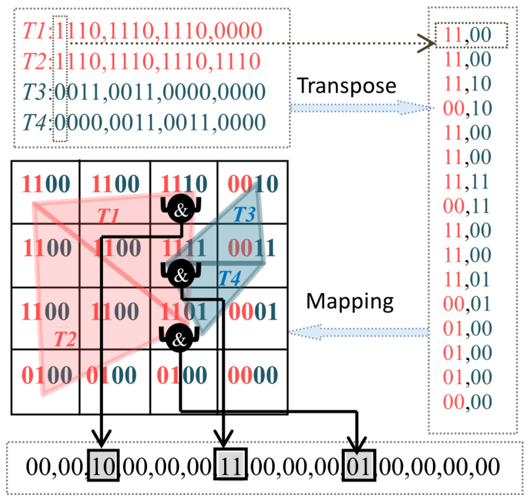

Figure 2: *To obtain the possible colliding pairs by the encoding information.*

3.3. Light weight dispatching scheme

In first level spatial division, there may have cells which include a large number of primitives.To process these cells by a single thread is very time consuming. In order to balance the workload of each thread, a lightweight dispatching scheme is adopted before starting the second level of spatial subdivision to guarantee the maximum number of the In order to balance the workload of each thread, a lightweight dispatching scheme is adopted to guarantee the primitives processed in each cell is 64. So each thread can deals with a single primitive.

For any cell with n primitives, the primitives of each deformable bodies expressed a set, can be divided into m groups, where $m = \lceil n/N_{max} \rceil$. N_{max} is the maximum number of primitives processed in each cell of the second level of spatial subdivision. N_{max} is 64 in our method. Subsequently, a Cartesian product operation was applied to these groups to generate the new groups of the possible colliding primitives. For two bodies a and b, let the sets of primitives of each body in any cell be A and B, respectively. The procedure can be represented as:

$$\begin{aligned} S = A \times B = \{(g_a, g_b) | g_a \in A \; and \; g_b \in B\} \\ a = 0,1,\ldots,m_a; \quad b = 0,1,\ldots,m_b \end{aligned} \tag{4}$$

where g_a and g_b are the groups of the primitives of body a and b, respectively; m_a and m_b are the number of groups of body a and b, respectively. At last, each new group of primitives is dispatched into a thread for further processed. An example in a 2D case is illustrated in Figure 3. Assume the maximum number of primitives processed in each cell of the second level of spatial subdivision is 2. $T1$ to $T4$ are the primitives of body a, and $T5$ and $T6$ are the primitives of body b in the i^{th} cell of the first level of spatial hashing. The lightweight dispatching scheme re-groups the primitives, guaranteeing that the number of primitives processed in each cell of the second level of spatial subdivision does not exceed the limitation.

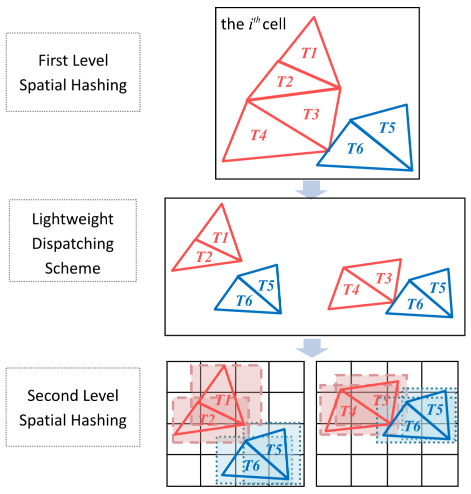
First Level Spatial Hashing
the i^{th} cell
T1
T2
T3
T4
T5
T6
Lightweight Dispatching Scheme
Second Level Spatial Hashing

Figure 3: *An example of our lightweight dispatching scheme in a 2D case.*

4. Experimental Results and Discussion

4.1. Experimental results

Our method was implemented by CUDA 5.0 on a PC with Intel i7 CPU, 3.2GHz, 16GB RAM and NVIDIA GeForce GTX680 graphics card. The deformation was generated by simulating the waving effect on the objects.

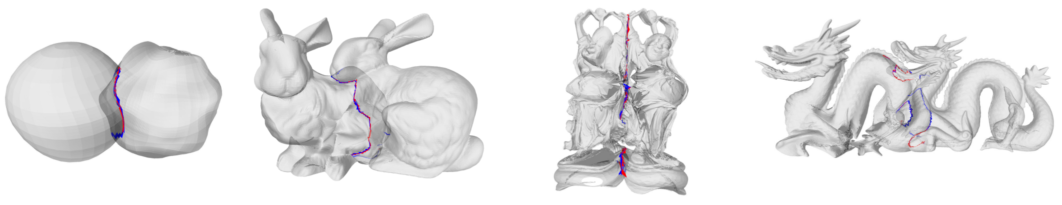

Figure 4: *The experimental results tested on four benchmarks. The colliding triangles of two objects are indicated in red and blue, respectively.*

Models	*#Tri*	*#Cells*	*#Sub*	*#CT*	*Time (ms)*
ball	15k x2	8,450	422	355	2.77
bunny	69k x2	40,068	1,014	1,202	3.66
buddha	100k x2	189,000	7,284	2,152	5.17
dragon	400k x2	229,900	4,290	3,571	8.31

Table 1: *The details of the benchmarks, the spatial subdivision information, and the time performance of the collision detection. #Tri is the number of triangles of the model. #Cells is the resolution of the first level of spatial grid. #Sub is the number of cells undergoing the second-level of subdivision. #CT is the average number of colliding triangles, and the last column is the average time of the collision detection.*

The tests of our method were carried out on four model benchmarks. Table 1 shows the details of the benchmarks, the spatial subdivision information, and the time performance of the collision detection. The colliding results testing on four benchmarks are shown in Figure 4.

The performance of our method was compared to the parallel spatial hashing method proposed in [FWZS11]. The comparison result is shown in Figure 5. In their method, the parallel spatial hashing was implemented by using the parallel radix sort and compaction computation with λ=5 for their best performance. In our method, λ was experimentally chosen to be 1. Therefore, the grid size is bigger in the first-level of spatial grid compared to their method. However, our method obtained the higher culling efficiency due to the use of the second-level spatial subdivision with the encoding scheme in the shared memory on GPU. The second-level of subdivision diminishes the occurence of high number of possible colliding pairs per cell. Furthermore, the use of the shared memory avoids the high latency of the global memory. The average time of our method for testing on four benchmarks is more than twice as fast.

Our method was also evaluated on the large deformation simulated by corotated finite element method [MG04] [ITF04]. The results are shown in Figure 6. The bunny and dragon models consist of 19,095 and 14,922 tetrahedrons, respectively. Table 2 shows the details of the spatial subdivision information, and the time performance of the collision detection.

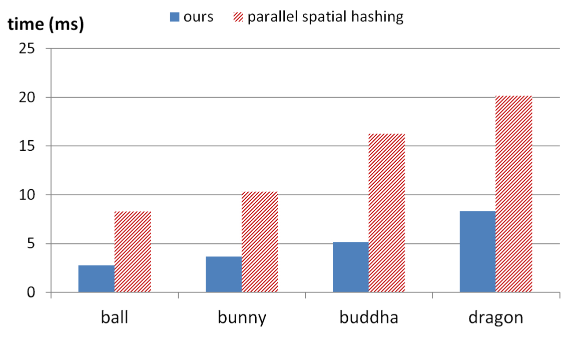

Figure 5: *The performance comparison of our method and the parallel spatial hashing.*

4.2. Discussion

The primitive pairs occupying the same cells may be checked multiple times. For the four model benchmarks in our experiments, 62% to 83% primitive pairs undergo multiple checking. The average checking times are 2.11 to 3.72 as shown in Table 3. Besides the model scales, the checking times are also related with the cell size of the spatial division. The smaller the cell size is, the more the duplication checking of the primitive pairs occurs. One solution is to handle the possible colliding pairs by a hashing function. In our experiments, the DJB2 hashing function was tested. The time

Models	*#Time (without hash)*	*#Time (with hash)*	*#CT (without hash)*	*#CT (with hash)*	*#Missing rate*	*#MCT*	*#Checking times*
ball	2.77	2.43	355	345	0.027	0.83	3.51
bunny	3.66	3.53	1,202	1,158	0.037	0.63	2.11
buddha	5.17	4.42	2,152	2,060	0.043	0.72	3.72
dragon	8.31	8.09	3,571	3,452	0.033	0.62	2.31

Table 3: *The comparison with and without the hash function for testing the possible colliding pairs. #Time is the average time of the collision detection. #CT is the average number of colliding primitives. #Missing rate is the ratio of missing collisions to colliding primitives. #MCT is the precentage of multiple checking triangles, and #Checking times is the average checking times.*

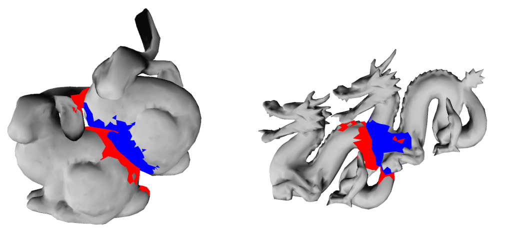
(a) bunny (b) dragon

Figure 6: *The experimental results tested on the large deformable simulation by corotated finite element method. The colliding tetrahedrons of two objects are indicated in red and blue, respectively.*

Models	*#Tetra*	*#Cells*	*#Sub*	*#CT*	*Time (ms)*
bunny	8,022 x2	647	469	2,452	7.61
bunny	19,095 x2	1,741	649	4,645	9.67
dragon	8,265 x2	676	277	1,063	6.31
dragon	14,922 x2	1,316	337	1,793	7.27

Table 2: *Collision detection on large deformable objects. #Tetra is the number of tetrahedrons in the model. #Cells is the resolution of the first level of spatial grid. #Sub is the number of cells undergoing the second level of subdivision. #CT is the average number of colliding tetrahedrons, and the last column is the average time of the collision detection.*

performance can be improved by eliminating the unnecessary duplication of the colliding test (see Table 3). However, due to the confliction problem of the hash function, some of the colliding primitive pairs have not been detected. The missing rates of the benchmarks are 0.027 to 0.043.

The missing rate can be further reduced by increasing the size of hash table (see Figure 7). However, a large size of the hash table results in a high memory cost. Thus, the size of the hashing table was chosen as the total number of primitives in our experiments to balance the memory cost and the collision missing rate.

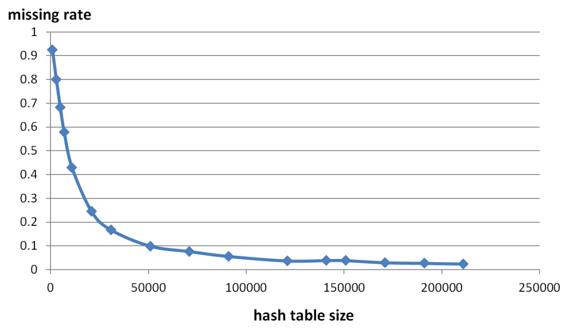

Figure 7: *The relationship of the missing rate and the size of the hashing table of the bunny benchmark.*

5. Conclusions and future work

We have presented a two-level spatial hashing method for real-time collision detection between deformable objects on GPU in the paper. The application of two-level structures enhances culling out the non-intersection primitives without sacrificing the overall culling efficiency. Moreover, a novel encoding method has been proposed to manage the primitive-cell mapping in the share memory of GPU. We have tested our method on four benchmarks and compared the performance with the parallel spatial hashing method [FWZS11]. Our method can process complex deformable objects involving a large amount of collisions in a few milliseconds.

Currently, our method handles the collision detection between two deformable objects. In many scenarios, collisions would occur to a large amount of objects. Moreover, intra-object collisions often occur to the objects with large deformation. Methods to handle these cases will be investigated as future work.

Acknowledgements

This research is supported by Multi-Year Research Grant of the University of Macau (MYRG150 (Y1-L2)-FST11-WW and MYRG2014-00139-FST) and NSFC Grant (61272326, 61135003 and 81201438).

References

[BW03] BACIU G., WONG W.: Image-based techniques in a hybrid collision detector. *IEEE Transactions on Visualization and Computer Graphics 9*, 2 (April 2003), 254–271. doi:10.1109/TVCG.2003.1196011. 2

[FWZS11] FAN W., WANG B., ZHOU J., SUN J.: Parallel spatial hashing for collision detection of deformable surfaces. In *Computer-Aided Design and Computer Graphics (CAD/Graphics), 2011 12th International Conference on* (Sept 2011), pp. 288–295. doi:10.1109/CAD/Graphics.2011.28. 2, 4, 5

[HTG04] HEIDELBERGER B., TESCHNER M., GROSS M.: Detection of collisions and self-collisions using image-space techniques. In *JOURNAL OF WSCG* (2004), pp. 145–152. 2

[ITF04] IRVING G., TERAN J., FEDKIW R.: Invertible finite elements for robust simulation of large deformation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2004), SCA '04, Eurographics Association, pp. 131–140. URL: http://dx.doi.org/10.1145/1028523.1028541, doi:10.1145/1028523.1028541. 4

[MG04] MÜLLER M., GROSS M.: Interactive virtual materials. In *Proceedings of Graphics Interface 2004* (School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004), GI '04, Canadian Human-Computer Communications Society, pp. 239–246. URL: http://dl.acm.org/citation.cfm?id=1006058.1006087. 4

[NVI11] NVIDIA CORPORATION: *NVIDIA CUDA C Programming Guide*, June 2011. 2

[PKS10] PABST S., KOCH A., STRASSER W.: Fast and scalable cpu/gpu collision detection for rigid and deformable surfaces. In *Computer Graphics Forum* (2010), vol. 29, Wiley Online Library, pp. 1605–1612. 2

[TKH*05] TESCHNER M., KIMMERLE S., HEIDELBERGER B., ZACHMANN G., RAGHUPATHI L., FUHRMANN A., CANI M.-P., FAURE F., MAGNENAT-THALMANN N., STRASSER W., ET AL.: Collision detection for deformable objects. In *Computer Graphics Forum* (2005), vol. 24, Wiley Online Library, pp. 61–81. 1

[TMLT11] TANG M., MANOCHA D., LIN J., TONG R.: Collision-streams: Fast gpu-based collision detection for deformable models. In *Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2011), I3D '11, ACM, pp. 63–70. URL: http://doi.acm.org/10.1145/1944745.1944756, doi:10.1145/1944745.1944756. 1, 2

[TMT10] TANG M., MANOCHA D., TONG R.: Mccd: Multi-core collision detection between deformable models using front-based decomposition. *Graphical Models 72*, 2 (2010), 7–23. 1

[WIK*06] WALD I., IZE T., KENSLER A., KNOLL A., PARKER S. G.: Ray tracing animated scenes using coherent grid traversal. In *ACM Transactions on Graphics -proceedings of ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), vol. 25 of *SIGGRAPH '06*, ACM, pp. 485–493. URL: http://doi.acm.org/10.1145/1179352.1141913, doi:10.1145/1179352.1141913. 2

[ZK07] ZHANG X., KIM Y. J.: Interactive collision detection for deformable models using streaming aabbs. *IEEE Transactions on Visualization and Computer Graphics 13*, 2 (Mar. 2007), 318–329. URL: http://dx.doi.org/10.1109/TVCG.2007.42, doi:10.1109/TVCG.2007.42. 1

[ZK14] ZHANG X., KIM Y. J.: Scalable collision detection using p-partition fronts on many-core processors. *IEEE Transactions on Visualization and Computer Graphics 20*, 3 (March 2014), 447–456. doi:10.1109/TVCG.2013.239. 1, 2