GPU Based Simulation of Collision Detection of Irregular Vessel Walls

Binbin Yong, Jun Shen, Hongyu Sun, Zijian Xu, Jingfeng Liu and Qingguo Zhou

Abstract Collision detection is a commonly used technique in the fields of computer games, physical simulation, virtual technology, computing and animation. When simulating the process of particle collision of ADS (Accelerator Driven Sub-Critical) system, complex and irregular vessel walls need to be considered. Generally, an irregular vessel wall is a curve surface, which cannot be defined as an exact mathematical function, and it is difficult to calculate the distance between particles and the wall directly. In this paper, we present an algorithm to perform collision detection between particles and irregular wall. When the number of particles reaches the level of 10^6, our algorithm implements a considerable improvement in performance if running on GPU, nearly 10 times faster than running on CPU. Results have demonstrated that our algorithm is promising.

Keywords Collision detection · Irregular vessel · Physical simulating · GPU

B. Yong · H. Sun · Z. Xu · Q. Zhou (✉)
School of Information Science and Engineering, Lanzhou University,
Lanzhou, China
e-mail: zhouqg@lzu.edu.cn

B. Yong
e-mail: yongbb14@lzu.edu.cn

H. Sun
e-mail: sunhy13@lzu.edu.cn

Z. Xu
e-mail: xuzj15@lzu.edu.cn

J. Shen
School of Computing and Information Technology, University of Wollongong,
Wollongong, Australia
e-mail: jshen@uow.edu.au

J. Liu
Linksprite, Wuhan, China
e-mail: jingfeng.liu@linksprite.com

N.Y. Yen and J.C. Hung (eds.), *Frontier Computing*, Lecture Notes
in Electrical Engineering 422, DOI 10.1007/978-981-10-3187-8_42

1 Introduction

With the rapid development of GPU (Graphics Processing Unit) technology, we are more likely able to implement complex and realistic system. In our research on nuclear simulations, the accelerated particles are generally made to collide with other particles and the irregular wall. The collision detection between particles is a hot research topic [1]. Hence, we need to simulate the collision between various particles and different walls.

In this paper, we design an algorithm to implement the collision detection between the particles and irregular wall by dividing the wall into very small triangles. Next, we assign the triangles and particles into uniform grids by space subdivision, and make collision detection between these triangles and particles (these particles are considered as spheres) in parallel with the help of GPU. We also implement the algorithm with CPU and make a comparison between these two computing architectures. Consequently, experimental results demonstrate that our parallel algorithm has a good feasibility and obvious effect in accelerating the simulation.

The rest of this paper is organized as follows. Section 2 reviews the related work. We present our collision algorithm in Sect. 3. In Sect. 4, we show the experimental results, and the effects of different parameters are discussed. At last in Sect. 5, we conclude the work and address the future work.

2 Related Work

General Purpose GPU (GPGPU) is a relatively new research area. But the idea of using GPU in collision detection has been researched longer than the emergence of GPGPU.

Zheng et al. [2] showed a contact detection algorithm based on GPU and they used the uniform grid method in detection. Based on the vector relation of point, line segment and rectangle, Shen et al. [3] implemented a rapid collision detection algorithm. Li and Suo [4] researched the application of particle swarm optimization in randomly collision detection algorithm and increased real-time capacity, compared to the classic OBB (Oriented Bounding Box) bounding box algorithm. Similarly, Qu et al. used parallel ant colony optimization algorithm in randomly collision detection algorithm to improve the real-time characteristic and precision in collision detection [5]. With spatial projection transformation method, Li and Tao mapped irregular objects from three dimensional space to regular two-dimensional objects to carry on collision detection [6]. Based on MPI (Message Passing Interface) and spatial subdivision algorithm, Huiyan et al. researched an advanced algorithm to improve the performance and accuracy of collision detection [7]. Tang et al. [8] proposed a GPU-based streaming algorithm to perform collision queries between deformable models by using hierarchical culling. Zhang et al. presented a parallel collision detection algorithm with multiple-core computation by CPUs or GPUs [9]. Wang et al. proposed an image-based optimization algorithm for collision detection [10].

Although many of these GPU-based collision detection algorithms have been researched, the research on the collision of irregular walls has been seldom discussed so far.

3 Collision Detection Algorithm

3.1 Overview

In this paper, a 3D irregular wall model is represented by STL (STereo Lithograph) file format, which contains many triangle meshes. The basic idea of our algorithm is to divide the triangles of the wall into very small triangles that are in the same scale as the particles. And then we assign these triangles into different grids by their center. At last, we make collision detection between these triangles and the particles in the same grid.

We implement both CPU-based and GPU-based codes to detect collision between particles and irregular vessels. However, the main idea of the algorithm is similar. The basic process of our algorithm is as follows: Firstly, we read the model file to get the position information of the triangle meshes. Next, we divide these triangles into smaller triangles, whose sizes are limited to particular range to make sure it can be contained by the uniform grids. The next phase is to make space subdivision. In this paper, we simply use the uniform grid to divide the space. Next we put the particles and the triangles into corresponding grid, and sort them to make it easy to implement data replication between CPU and GPU memory (no sort operation and data replication in CPU code). Finally, we perform the collision detection between particles and triangles in the same grid by calculating the distance between particles and triangles.

3.2 Data Structures

When making collision detection on GPU, we firstly copy the data structures of particles and triangles from CPU memory to GPU memory. (Note: The latest CUDA versions support (UMA) uniform memory access, but our algorithm is designed not only for new CUDA versions). The data structures are organized in array to copy quickly rather than use complex data structures such as linked list. The basic arrays are particles array and triangles array. The particles array is organized every four float data for each particle, including three coordinate values and a radius. The triangles array is organized every 12 float data for each triangle, including three points (9 coordinate values) and a normal vector (3 coordinate values). There are also auxiliary arrays, such as hash arrays, index arrays, cell start arrays and cell end arrays, both for particles and triangles.

3.3 Triangles Division

We read the triangles of the irregular wall model from an STL file. In general, these triangles are much larger than the particles. Firstly, we need to divide the triangles into smaller ones. In this paper, we divide the triangles by two methods to acquire a better solution. The two methods need a different minimum value of side length to divide the triangles.

The first method is based on the center of a triangle (centre of gravity), which is denoted as triangle-center method. We assign the divided triangles into different grids by the center of a triangle, and the minimum value of side length is set as the maximum radius of the particles.

Another method is based on the circumcenter of a triangle, which is denoted as circumcenter method. In this case, we should make sure that the area of every small triangle is as large as possible to reduce the number of triangles after dividing, so as to reduce the computation complexity. When the maximum value of the side length is fixed, an equilateral triangle has the largest area.

Hence, we get the side length by equation

$$d = r \times \cos(30^\circ) \times 2 \tag{1}$$

We assign these divided triangles into different grids by their circumcenters, and the minimum value of side length is set as Eq. (1).

But there is still an issue. If the divided triangle is an obtuse triangle, the circumcenter may be out of the triangle. In this situation, it is difficult to calculate which grid the obtuse triangle belongs to. Hence, we need to divide the obtuse triangle into smaller non-obtuse triangles. We simply draw a vertical line from the vertex of the obtuse angle to the opposite edge. By this way, the obtuse triangle is divided into two right triangles.

These two methods look similar. Nevertheless, the number of divided triangles grows exponentially. With the circumcenter method, the minimum side length is $\sqrt{3}$(Eq. (1)) times of the first method. However, the total number of triangles after dividing is no more than half of the triangles divided by triangle-center method.

The basic approach to dividing a triangle is to recursively split the triangle into two smaller triangles from the midpoint of this edge. Figure 1 shows the divided model. The image in the left is the original model, and the middle is the divided model.

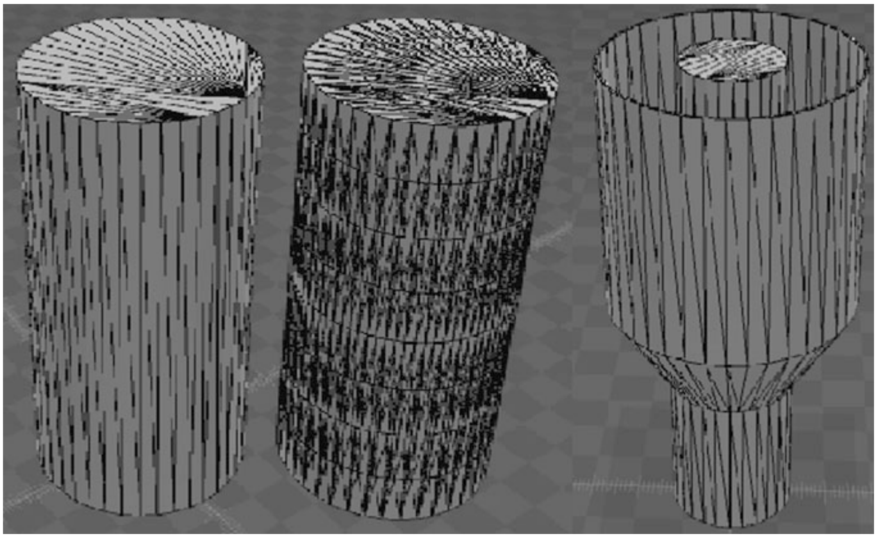

Fig. 1 The original, divided (wall1) and spallation target model (wall2) walls

3.4 Spatial Subdivision

We split the world space into uniform grids, and the number of grids in three dimensions is denoted as (*sizeX*, *sizeY*, *sizeZ*). Overall, there are $sizeX \times sizeY \times sizeZ$ grids. Each grid has the same scale size as the particles. For each particle we calculate the grid position (x, y, z) by its center position. And then we calculate the grid number as its hash value simply through equation

$$hash = z \times sizeY \times sizeX + y \times sizeX + x \tag{2}$$

For each triangle (after dividing) and particle, we judge the grid position by its center point. For a triangle, we calculate its triangle core or circumcenter, which is used to get the number of grid. We calculate the grid number (hash value) by Eq. (2).

3.5 Sorting and Data Replication

For a grid, the number of particles and triangles are not fixed. We herein can not store these data in fixed memory size. One way is to reorder the particles array and the triangles array by its hash value (the grid number), and then we store the particles and the triangles of each grid by their start number and end number. We sort the data via fast radix sort in the CUDPP library.

After sorting the data, we still need to fill in the hash start arrays and hash end arrays for particles and triangles. Each particle or triangle gets its start hash and end hash by comparing its cell index with the previous cell index. If there is a difference, it indicates a new grid and new start hash (the particles or triangles in one grid have a same hash value).

3.6 Collision Detection

Once the grid structures are built on GPU memory, it is used to detect particle-wall interactions. In the continuous simulation, we must consider the specific collision physical model between particles and walls to obtain the next frame. For example, a DEM (Discrete Element Method) [11, 12] may be used and the forces should be considered. In this paper, anyhow, we focus on the collision detection rather than the interaction model.

There are two methods to traverse collision detection. The first method is a wall-based method. For triangles in a grid, we get the start triangle and the end triangle, and then we loop over the neighboring grid cells of each triangle to check for collisions with each particle in these cells. The second method is particle-based method. Similarly, we find start particle and end particle in a grid, and then loop to detect neighboring collisions. There are similarities between these two methods. However, the computational efficiency is different. The detailed results can be seen in Sect. 4.

The last issue of basic collision detection is the test between a sphere and a triangle, which is equivalent to calculate the shortest distance between the sphere core and the triangle. If the distance is smaller than the sphere radius, it means a collision occurs.

4 Experiments

4.1 Experimental Environment

The experiments were basically performed on a computer with Intel Core i3 processor and 4.00 GB RAM and a GTX480 GPU. We also run the code on a computer with Tesla K80 GPU to find the improvement on different GPU unit. We used a 3D modeling software to draw a cylinder (height 100 and radius 50, the left and middle of Fig. 1, name wall1) as one simple collision wall, which was easy to verify collision detection. We also experimented on a spallation target model (the right of Fig. 1, name wall2) used in our project. The particles are different in size, but the maximum radius is set as 1. The length of uniform grid is set as 2.

Next, we mainly focus on the experiments between CPU and GPU, the differences between triangle-center and circumcenter triangulation method, wall-based and particle-based traversal methods, the influence of number of triangles and the differences on different GPUs.

4.2 CPU and GPU

In order to measure the performance of our algorithm, we choose different particle numbers as the input variable. We conduct experiments on CPU and GPU platform. As shown in the left of Fig. 2 and Table 1 (only 10^2, 10^3, 10^4, 10^5, 10^6 level of magnitudes are shown), we can see that if the number of particles is small (less than 10^3), the CPU and GPU code has a similar computation time, and CPU code may even achieve a better performance. However, with the increase of the number of particles (more than 10^4), the computation time of CPU code increases sharply. When the number of particles is 10^4, 10^5 and 10^6, compared to CPU code, the speed-up ratio of GPU code by circumcenter method is 1.89, 3.69 and 4.99.

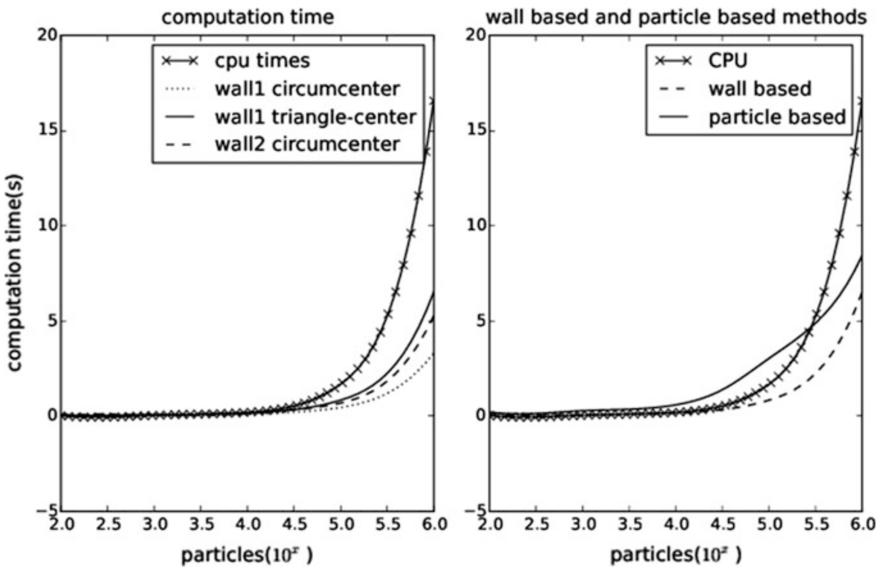

Fig. 2 The *left* figure shows the computation time based on CPU, and three different methods based on GPU: wall1 circumcenter method, wall1 triangle-center method, wall2 circumcenter method. The *right* figure illustrates the comparison between wall based method and particle based method

Table 1 Experimental results of computation time

Number	10^2	10^3	10^4	10^5	10^6
CPU time(s)	0.02577	0.03819	0.19552	1.67494	16.5695
wall1 tri-center(s)	0.05659	0.06805	0.15951	0.83642	6.51862
wall1 circumcenter(s)	0.03703	0.04499	0.10320	0.45318	3.31567
wall2 circumcenter(s)	0.04515	0.05750	0.13836	0.68392	5.26571

4.3 Two Traversal Methods

As discussed above, there are two methods to detect collision based on GPU, wall based and particle based methods. In the right of Fig. 2, the computation time of these two methods was drawn. If we use particle based method, when the number of particles is less than $10^{5.5}$, the GPU code is even worse than CPU code. When the number of particles is larger, the particle based GPU code performs better than CPU code. However, the particle based code is no better than triangle based code. In general, the number of triangles is much smaller than the number of particles. Hence, the particle based code needs more concurrent threads to finish collision detection.

4.4 The Circumcenter and the Triangle-Center Methods

In Sect. 3.3, we discussed two type of triangle division methods, circumcenter method and triangle-center method. Now we compare these two methods on experimental data.

For the model wall1, the original number of input triangles is 156. If we use the triangle-center method to divide these triangles and the minimum value of side length is set as 1, we get 226056 triangles after division. When we use the circumcenter method, the minimum value of side length is set as $\sqrt{3}$. The number of triangles is 99612 after division, which is less than half of the triangles by triangle-center method.

As shown in Fig. 2 and Table 1, we find that the circumcenter method is more efficient than the triangle-center method in computation. A plausible explanation is that the circumcenter method uses the collision space more effectively and gets fewer triangles after division. In fact, the computation time is relevant to the number of triangles, which can be seen from next section.

4.5 The Influence of Triangles Number

We tested the influence of number of triangles based on fixed number of particles. In Fig. 3, we set the number of particles as 10^5 and 10^6 and test the relationship between number of triangles and computation time. We roughly draw a conclusion that the computation time is proportional to the number of triangles. We calculate the following linear fitting function. Where t_5 and t_6 stand for the time for 10^5 and 10^6 particles, and n_{tri} stands for the number of triangles.

$$t_5 = 2.477 \times 10^{-9} \cdot n_{tri} + 0.2712 \tag{3}$$

$$t_6 = 2.078 \times 10^{-5} \cdot n_{tri} + 1.8970 \tag{4}$$

We also tested the spallation target model wall2 based on circumcenter triangle division method. The original number of triangles is 1600. The number of triangles after division is 151424. As shown in Fig. 2, it even runs faster than the simpler model wall1 based on triangle-center method. The reason is that the number of divided triangles of wall1 based on triangle-center method is 226056, larger than number of divided triangles of wall2 based on circumcenter method. Therefore,

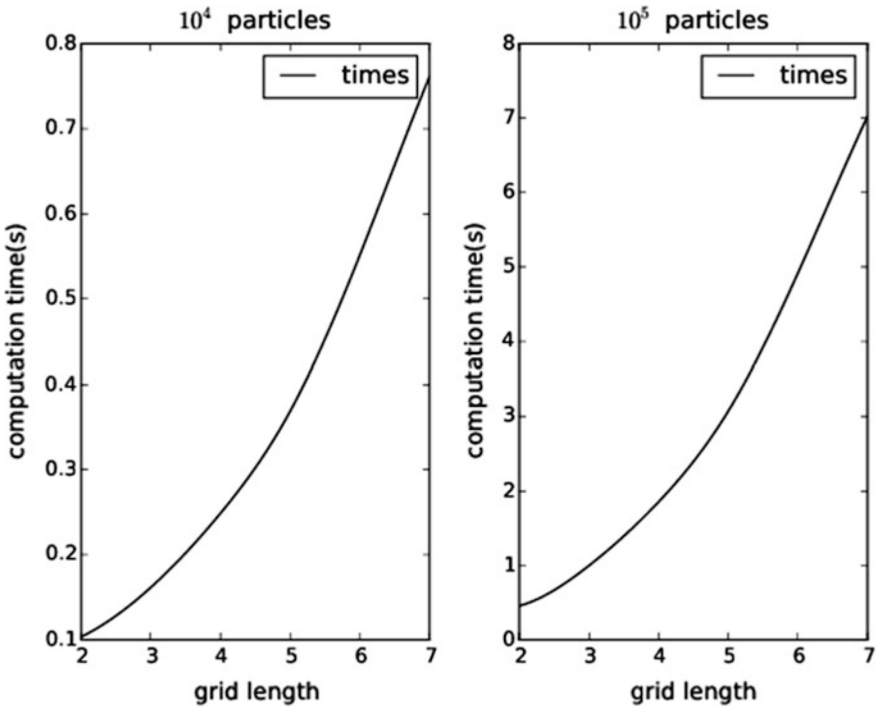

Fig. 3 Experimental results: the influence of the number of triangles

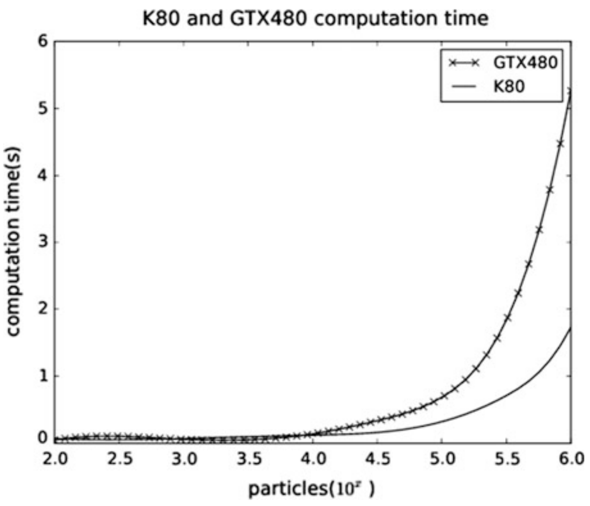

Fig. 4 Experimental results: the comparison between GTX480 and K80

using a better method to reduce the number of divided triangles is a better way to improve computational efficiency.

4.6 *On Different GPUs*

We also tested our algorithm on GTX480 and Tesla K80 GPU. As shown in Fig. 4. When the number of particles is less than 10^4, the code on GTX480 runs faster. However, once the number of particles is more than 10^4, for instance, 10^4, 10^5, 10^6, compared to code on GTX480, the speed-up ratio of code on K80 are 1.15, 2.14, 3.04. Hence, we can conclude that with the increase of the number of particles, a GPU with higher computation ability will obviously improve computation efficiency.

5 Conclusion

In our current process, it is meaningful to focus on collision detection between irregular walls and particles. In this paper, we designed and implemented an algorithm to achieve this goal. With the help of a K80 GPU, we can detect collision between a million particles and a spallation target model in 2s. The experiment results prove that the algorithm proposed in this paper is feasible and effective.

Acknowledgements This work was supported by Dongguan's Recruitment of Innovation and entrepreneurship talent program, National Natural Science Foundation of China under Grant No. 61402210 and 60973137, Program for New Century Excellent Talents in University under

Grant No. NCET-12-0250, Strategic Priority Research Program of the Chinese Academy of Sciences with Grant No. XDA03030100, Gansu Sci. and Tech. Program under Grant No. 1104GKCA049, 1204GKCA061 and 1304GKCA018, Google Research Awards and Google Faculty Award, China.

References

1. NVIDIA, *Particle Simulation using CUDA*, 1st ed., NVIDIA, 9 2013.
2. J. Zheng, X. An, and M. Huang, "Gpu-based parallel algorithm for particle contact detection and its application in self-compacting concrete flow simulations," *Computers & Structures*, vol. 112, pp. 193–204, 2012.
3. Y. Shen, Q. Jia, G. Chen, Y. Wang, and H. Sun, "Study of rapid collision detection algorithm for manipulator," in *Industrial Electronics and Applications (ICIEA), 2015 IEEE 10th Conference on*, Jun. 2015, pp. 934–938.
4. S. Xue-li and Z. Ji-suo, "Research of collision detection algorithm based on particle swarm optimization," *Computer Design and Applications (ICCDA)*, vol. 1, 2010.
5. H. Qu and W. Zhao, "Fast collision detection algorithm based on parallel ant," *Virtual Reality and Visualization (ICVRV)*, pp. 261–264, 2013.
6. S. Xue-li and L. Tao, "Fast collision detection based on projection parallel algorithm," *Future Computer and Communication (ICFCC)*, vol. 1, 2010.
7. H. Qu and W. Zhao, "Fast collision detection of space-time correlation," *Computer Science and Electronics Engineering (ICCSEE)*, vol. 3, pp. 567–571, 2012.
8. M. Tang, D. Manocha, J. Lin, and R. Tong, "Collision-streams: Fast GPU-based collision detection for deformable models," in *I3D '11: Proceedings of the 2011 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 2011, pp. 63–70.
9. X. Zhang and Y. J. Kim, "Scalable collision detection using p-partition fronts on many-core processors," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 3, pp. 447–456, Mar. 2014.
10. L. Wang, Y. Shi, and R. Li, "An image-based collision detection optimization algorithm," in *Signal and Information Processing (ChinaSIP), 2015 IEEE China Summit and International Conference on*, Jul. 2015, pp. 220–224.
11. H. Karunasena, W. Senadeera, Y. Gu, and R. Brown, "A coupled sph-dem model for fluid and solid mechanics of apple parenchyma cells during drying," in *18th Australian Fluid Mechanics Conference*. Australasian Fluid Mechanics Society Launceston, Australia, 2012.
12. M. Rhodes, X. S. Wang, M. Nguyen, P. Stewart, and K. Liffman, "Study of mixing in gas-fluidized beds using a dem model," *Chemical Engineering Science*, vol. 56, no. 8, pp. 2859–2866, 2001.