2017

GVM Based Intuitive Simulation Web Application for Collision Detection

Binbin Yong
Lanzhou University, yongbb14@lzu.edu.cn

Jun Shen
University of Wollongong, jshen@uow.edu.au

Zebang Shen
Lanzhou University

Huaming Chen
University of Wollongong, hc007@uowmail.edu.au

Xin Wang
Lanzhou University

See next page for additional authors

GVM Based Intuitive Simulation Web Application for Collision Detection

Abstract

Computer simulation, which has been proved to be an effective approach to problem solving, is nowadays widely used in modern science. However, it requires a lot of computing resources, which are difficult for general users to acquire. In this paper, we design a Web based system to implement on-line simulation system for ordinary users. As a useful example, the simulation of one type of collision detection model is presented in this paper. Moreover, the software application of simulation is offered as a service on Web. Meanwhile, the incorporation of general vector machine (GVM, a type of neural network) to intelligently predict the relationship between simulation parameters and computation resources is presented, which could further provide more information for system monitoring and scheduling. The system has demonstrated efficiency and intuitiveness for users of this type of applications.

Keywords

collision, detection, simulation, intuitive, web, gvm, application

Publication Details

Yong, B., Shen, J., Shen, Z., Chen, H., Wang, X. & Zhou, Q. (2018). GVM Based Intuitive Simulation Web Application for Collision Detection. Neurocomputing, 279 63-73

Authors

Binbin Yong, Jun Shen, Zebang Shen, Huaming Chen, Xin Wang, and Qingguo Zhou

GVM Based Intuitive Simulation Web Application for Collision Detection

Binbin Yong[a], Jun Shen[b], Zebang Shen[a], Huaming Chen[b], Xin Wang[a], Qingguo Zhou[a,*]

[a]*School of Information Science and Engineering, Lanzhou University, Lanzhou, Gansu, China*
[b]*School of Information Systems and Technology, University of Wollongong, NSW, Australia*

Abstract

Computer simulation, which has been proved to be an effective approach to problem solving, is nowadays widely used in modern science. However, it requires a lot of computing resources, which are difficult for general users to acquire. In this paper, we design a Web based system to implement on-line simulation system for ordinary users. As a useful example, the simulation of one type of collision detection model is presented in this paper. Moreover, the software application of simulation is offered as a service on Web. Meanwhile, the incorporation of general vector machine (GVM, a type of neural network) to intelligently predict the relationship between simulation parameters and computation resources is presented, which could further provide more information for system monitoring and scheduling. The system has demonstrated efficiency and intuitiveness for users of this type of applications.

Keywords: computer simulation, Web, collision detection, GVM

1. Introduction

With the rapid development of graphics processing unit (GPU) technology, realistic high performance computing systems become more powerful, and at the same time, more complex. For example, in the research field of nuclear simulation, we need to simulate the collision between various particles and different walls. In the field of particle accelerator field, the accelerated particles collide with other particles or the irregular wall. One of the most important factors to generate this type of simulative physical systems is to realize collision detection, especially when vessels are irregular. The term *irregular vessel wall* is the container used to store the particles to be accelerated and collided. In this paper, it is equivalent to *irregular object*, *object*, *wall*, and *model*.

Limited to the computing ability, traditional algorithms could only simulate collisions of tens of thousands particles. The reason was that with the growth of the particles number, the cost of computing would increase exponentially. Nowadays, GPU is more widely used in simulating computation to accelerate this type of simulation for its parallel property. In this paper, we design an algorithm to implement the collision detection between the particles and irregular walls. These particles are generally considered as spheres. Firstly, we divide the wall into small triangles that are surrounded by spheres. And then we make collision detection between these spheres and particles in parallel by utilizing GPU and space subdivision.

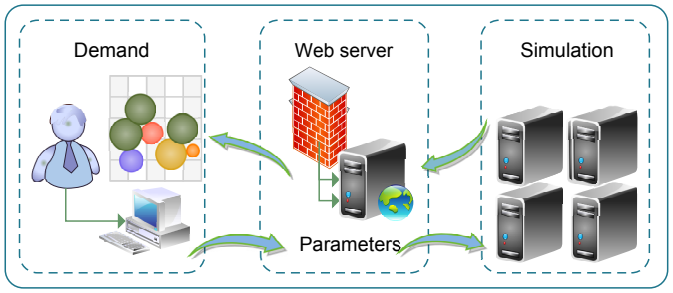

Fig. 1. Web-based simulation platform.

At the same time, as shown in Fig. 1, we implement a Web based platform to provide on-line simulation. Users will be able to share their simulation results and computation resources in this platform. The whole system is efficient and intuitive to end users who have a vision to share or present the simulations of large scale physics experiments. The computation resources of new simulation task is predicted by an artificial neural network (ANN) based general vector machine (GVM), which is instructive for resources scheduling.

The rest of this paper is organized as follows. Section 2 reviews the basic concepts and prior work. We present our design and implementation of the collision algorithm in Section 3. Section 4 shows the design of the Web simulation service. In Section 5, we firstly present the experimental results of the collision algorithm, then give the analysis of our Web based system. The effects of different parameters on the algorithm are discussed. At last in Section 6, we conclude our research and address the future work.

*Corresponding author at: School of Information Science and Engineering, Lanzhou University, Lanzhou, China. Tel: +86-0931-8912025; Fax: +86-0931-8912025

Email addresses: yongbb14@lzu.edu.cn (Binbin Yong), jshen@uow.edu.au (Jun Shen), shenzb12@lzu.edu.cn (Zebang Shen), hc007@uowmail.edu.au (Huaming Chen), xwang2016@lzu.edu.cn (Xin Wang), zhouqg@lzu.edu.cn (Qingguo Zhou)

2. Preliminary and Related Work

Generally in physics simulations, to detect collision is to calculate the distance between objects. It is easy to get the distance from a sphere center to a regular wall. For example, to get the distance between a sphere and a cube wall, we can merely calculate the distance between center of the sphere and the airplane of the cube. In this paper, we propose a new method and focus on the collision detection between irregular wall and particles. For simply, we omit the unit of length and keep the relative sizes of particles and models. The particles are seen as spheres, and the irregular wall is represented by a STereo Lithograph (STL) 3-dimensional (3D) model file, by which a spatial model is often approximated as many triangle meshes. This type of triangle mesh has a collision area which is different from a general meaning of triangle consists of three lines. The basic collision detection is based on uniform grid spatial subdivision and the bounding box is sphere box. The algorithm is implemented on different GPUs as well as on CPU.

2.1. Collision detection

The technology of collision detection, which studies the problem that whether two or more objects collide or not, when and where the collisions occur in the virtual scene, has been widely used in the field of computer games, physic simulation, virtual reality and animation. In the past few decades, a few collision detection algorithms have been proposed and proved to be effective. In general, there are two types of collision detection algorithms, spatial subdivision and bounding box. In reality, the bounding box method is more widely used than spatial subdivision method. The most widely used boxes are axis aligned bounding boxes (AABB) [1], sphere, oriented bounding box (OBB) [2], fixed directions hulls (FDH) [3]. Most of collision detection algorithms focus on the collision detection between regular objects. In this paper, we use the spatial subdivision combined with box technology to implement the parallel irregular wall collision detection in a more efficient way.

2.2. GPU and CUDA

In recent years, massive data computing is developing towards the CPU and GPU co-processing. In the year of 2006, Nvidia introduced Compute Unified Device Architecture (CUDA). It is a general-purpose embedded GPU parallel computing platform [4]. CUDA supports C, C++, Fortran and other programing languages. A GPU with CUDA support contains some Streaming Multiprocessors (SMXs), which are used to execute code in parallel. Hence, it is beneficial to reduce a lot of run time by parallel programming for some time-consuming tasks. Whereas, CPU is more capable of data caching and flow control, the GPU has more transistors and it is specialized for compute-intensive, highly parallel computation and data processing [5].

2.3. Web application

In the design of our system, Django[1], JSModeler[2] and Nginx[3] are used to structure our Web application. Django is a free and open source high-level Python based Web framework, which is designed to achieve rapid development of Web applications. In the design of our physic simulation system, Django is used to implement a Web application to invoke simulating computations on computing nodes. Meanwhile, users can design simple vessel models for collision simulation online with the Web application. These vessel models are created by the Javascript framework named JSModeler. In our system, JSModeler is used to generate intuitive 3D models online. It is embedded in Django Web framework to provide 3D modeling implementation. Nginx is one of the most popular open source Web server. It is also used as load balancer for Web servers.

2.4. GVM

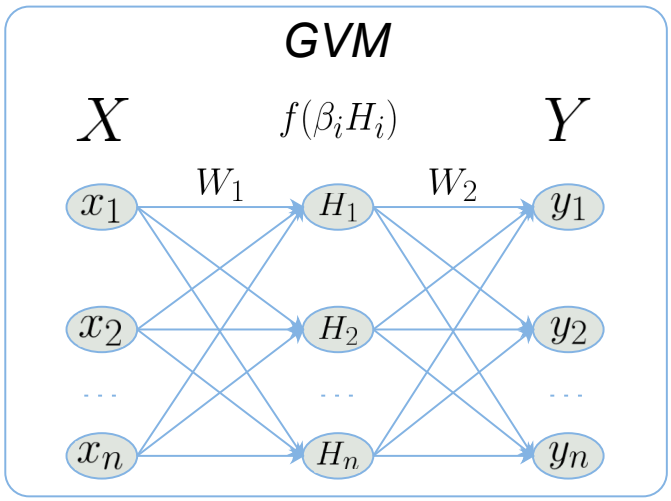

Fig. 2. The structure of GVM.

As shown in Fig. 2, GVM is a new type of learning machine based on 3-layer ANN, which is thought effective in finding relationship between training data [6]. Since proposed by Zhao [7] in 2016, GVM is proved to be effective for training models with small dataset [8, 9, 10, 11], which is suitable for our system to forecast simulation time when there is lack of training dataset. Instead of traditional back propagation (BP) algorithm [12], a GVM model is trained by Monte Carlo algorithm (MC), which endures GVM the ability to freely train itself with the increase of the training dataset collected by our system.

2.5. Related work

General Purpose GPU (GPGPU) is a relatively new research area. There are many researches using GPU for various types of collision detection. Purcell et al. [13] researched light and

[1] https://www.djangoproject.com/
[2] https://github.com/kovacsv/JSModeler/
[3] https://www.nginx.com/

triangle intersection test algorithm in ray tracing technology based on GPU. Baciu et al. [14] and Myszkowski et al. [15] researched convex body collision detection based on GPU by regarding pixels in each render buffer as a beam of light, which was perpendicular to the visual plane and testing intersection between the light and objects in the early days. Govindaraju et al. [16] presented an algorithm for collision detection between multiple deformable objects in a large environment using GPU. Kipfer et al. [17] presented a particle system engine for real-time animation and rendering. The system renders large particle sets using GPU and implements inter-particle collisions and visibility sorting. Zheng et al. [18] showed a contact detection algorithm on GPU and it used the uniform grid method in detection. S. Green et al. [19], showed a sample about the simulation of a particle system. The system uses a cube as the wall and concentrates on the parallelization of collision detection between the particles. In their simulation system, the particle's coordinates and the location of the cube wall were compared to deal with the collisions between particles and the wall. The algorithm proposed in this paper is different as we focus on the collision detection between particles and irregular walls, which is more common in physic simulation.

There are some other algorithms designed to optimize the computation of collision detection. Li and Suo [20] researched the application of particle swarm optimization in randomly collision detection algorithm. Similarly, Huiyan et al. used parallel ant colony optimization algorithm in randomly collision detection algorithm to improve the real-time and precision in collision detection [21]. By spatial projection transformation method, Li and Tao mapped irregular objects from 3D space to regular 2-dimensional (2D) objects to carry on collision detection [22]. Based on message passing interface (MPI) and spatial subdivision algorithm, Huiyan et al. researched an advanced algorithm to improve the performance and accuracy of collision detection [23]. Runtao et al. [24] designed a G-Octree based fast collision detection for large-scale particle systems. Zou et al. [25] designed a collision detection algorithm based on GPGPU. Fan et al. [26] explored the collision between two objects by finding intersections between a collection of line segments and a set of triangles. Even though collision detection algorithms based on GPU has been researched, research about the collision of irregular walls is rarely discussed. Therefore, we also provide the simulation services on the Web.

Web based simulation applications have been developed for a long time. Sakkopoulos et al. [27] developed an integrated system to provide health and information exchange services on Web. Jarno Ojala [28] studied an on-line sporting service which gathered users' data for analysis. Based on mobile sensors and Web technologies, Perttula et al. [29] developed Web applications to monitor heart rates.

3. Collision detection algorithm

In this section, we present the detailed steps of the algorithm to deal with the collision detection between particles and irregular walls. The algorithm is serviced on a Web site presented in the next section.

3.1. Overview

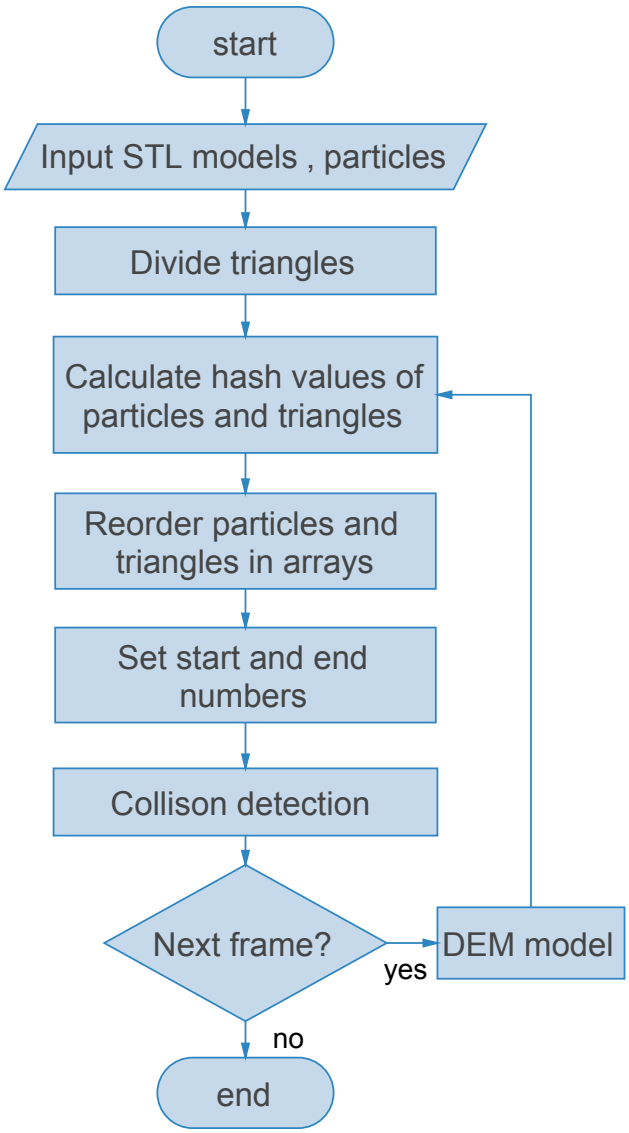

Fig. 3. The flow chart of collision detection based on GPU.

The basic idea of the algorithm is to divide the triangles of the object model into smaller triangles that are in the same scale as the particles. Then, these triangles are put into bigger sphere boxes. At last, collisions between these sphere boxes and these particles are checked.

In this paper, we implement both CPU and GPU code to detect collisions between particles and irregular vessels. The basic process of collision detection algorithm is as follows: firstly, we read the model from STL file to achieve the positions of these triangles. Then we divide these triangles into smaller triangles, whose sizes are limited to a value to make sure the spheres of the particle size can contain them. These spheres are seen as bounding boxes. Next the space is divided. In this paper, we simply use the uniform grid to divide the space. Then we put the particles and the triangles into corresponding grids. Meanwhile, we sort the particles and the triangles to make it easy

to implement data replication between CPU and GPU memory. In the implementation of CPU code, sort and replication operations are not necessary. Finally, we process the collision detection between particles and the triangles by calculating the shortest distances between them. The flow chart of GPU algorithm is shown in Fig. 3.

3.2. Data structure

Before collision detection on GPU, data of particles and triangles is copied from CPU memory to GPU memory. In the latest CUDA versions that support UMA (uniform memory access), memory copy is automatically finished. However, our algorithm is not only suitable for new CUDA versions. Hence, we organize the data in array format to copy quickly instead of using complex data structures such as linked list. Another benefit of array is that all data is continuously stored in a fixed size in memory. The basic arrays include particle-array and triangle-array. The particle-array is organized every four float data for each particle, including three coordinate values and a radius value. The triangle-array is organized every 12 float data for each triangle, including 9 coordinate values of three vertices and 3 coordinate values of a normal vector. There are also auxiliary arrays such as hash-array for particles and triangles, index-array for particles and triangles, cell start-arrays for particles and triangles, cell end-arrays for particles and triangles. These arrays have an integer or one float data for each element.

3.3. Triangle division

3.3.1. Length of triangles

The triangles of the irregular walls are exported from STL files which are uploaded by users or generated online. In our simulation, these triangles are much larger than these particles in size. Firstly we need to divide these triangles into smaller ones, to make sure that these divided triangles can be contained in sphere boxes that have sizes of particle scale. In this paper, we divide these triangles by two methods to acquire a better solution. These two methods both calculate the maximum side length of these triangles after dividing. According to the value of maximum side length, we divide these triangles. The way to find the value is to set a sphere bounding box of the maximum particle size which justly contains these triangles. When detecting collisions, these triangles are assigned to different grids according to the center of their sphere bounding box. Meanwhile, the particles are also assigned to different grids by their cores. Then, for each particle or each triangle, we will calculate distances with the triangles or particles in its neighbouring grids to determine collision or not.

The first way is a gravity based method. As shown in Fig. 4, we calculate the triangle core (center of gravity) of one triangle (point G). In the 3D spatial coordinate system, the triangle core is calculated as:

$$\begin{aligned} x_G &= \frac{x1 + x2 + x3}{3} \\ y_G &= \frac{y1 + y2 + y3}{3} \\ z_G &= \frac{z1 + z2 + z3}{3} \end{aligned} \tag{1}$$

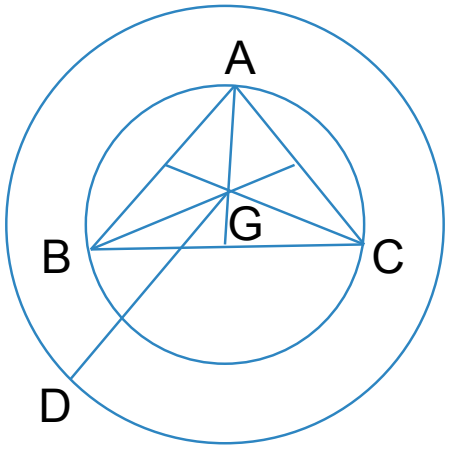

Fig. 4. Sizes of triangle and sphere in gravity based method.

By point G we can calculate which grid this triangle is assigned to. Because G is inside the triangle, the length between point G and any other points in the triangle is less than the longest side $|AB|$. Hence, using a sphere box whose radius is equal to $|AB|$ is able to contain the triangle.

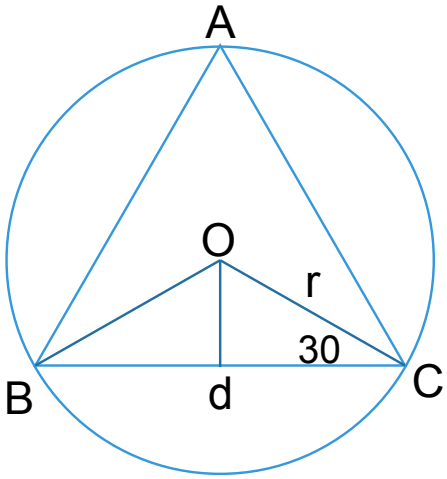

Fig. 5. Calculating side length of an equilateral triangle in circumscribed based method, r is the maximum radius of particles.

Another way proposed in this paper is circumscribed circle based method. In this case, we should make sure that the area of every triangle is as large as possible to reduce the number of triangles, so as to reduce computation complexity. When the maximum value of the side length of a triangle is fixed, equilateral triangle has the largest area. As shown in Fig. 5, we get the edge length of the equilateral triangle by Eq. (2), which is used to divide triangles.

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

We suppose the coordinates of the circle center is O (x, y, z), the coordinates of three points are A $(x1, y1, z1)$, B $(x2, y2, z2)$, C $(x3, y3, z3)$, and the length of three edges are a, b, c, the area is

S, the radius is r, $p = (a + b + c)/2$. According to the Helen area equation we get Eq. (3):

$$S = \sqrt{p(p-a)(p-b)(p-c)} \tag{3}$$

According to the Sine Theorem we get Eq. (4):

$$S = \frac{abc}{4r} \tag{4}$$

Meanwhile, as we know, $|OA|$ is equal to $|OB|$ and $|OC|$, hence we get Eq. (5):

$$\begin{aligned} (x1-x)^2 + (y1-y)^2 + (z1-z)^2 &= r^2 \\ (x2-x)^2 + (y2-y)^2 + (z2-z)^2 &= r^2 \\ (x3-x)^2 + (y3-y)^2 + (z3-z)^2 &= r^2 \end{aligned} \tag{5}$$

From Eq. (3, 4, 5), we can get the coordinates of circle center, by which we assign the triangle into a grid.

But there is still another problem. If the triangle is an obtuse triangle, the triangle center may be out of the triangle. In this situation, it is difficult to judge which grid the obtuse triangle belongs to. In this case, we must make sure the divided triangle is a non-obtuse triangle. We will review this problem once again in the next section.

These two approaches look similar. Nevertheless, the number of divided triangles grow exponentially with the maximum length. By the second method, the maximum side length is $\sqrt{3}$ (Eq. (2)) times of the first method. But the number of divided triangles is no more than half of the first method. It will reduce a lot computation time, which can be concluded from the experimental section.

3.3.2. Divide the triangles

After getting the maximum length value (named *mlvalue*) of the triangles after dividing. We divided triangles until the longest edge of these triangles are smaller than *mlvalue*. We construct a binary tree to organize these divide triangles. At the beginning, it has only a root point, which is the original triangle. Firstly, we check whether the longest side of the triangle on the root node is larger than *mlvalue*. If true, we divide the triangle into two smaller triangles from the midpoint of the longest edge. And then these two triangles are added to the left-subtree and right-subtree of the binary tree. At last, we recursively divide these two subtrees until the largest side of each triangle is less than *mlvalue*. Fig. 6 shows the divided model. On the left is the original model, and on the right is the divided model.

In fact, as discussed above, if circumscribed circle based method is used, we must make sure these divided triangles are non-obtuse. Therefore, if we get an obtuse triangle after the last dividing, we still need to divide the obtuse triangle into two non-obtuse triangle. In our algorithm, a vertical from the point of the obtuse angle to the opposite side is added. And then, the obtuse triangle is divided into two right triangles.

The concrete methods and steps to divide these triangles are based on the specific needs. For example, we can take the midpoints of three sides of a triangle and connect these midpoints to divide the triangle into four triangles. This method is suitable for the condition that the original triangle is approximately regular. In these cases, the original triangle is divided rapidly and the divided triangles are also approximately regular.

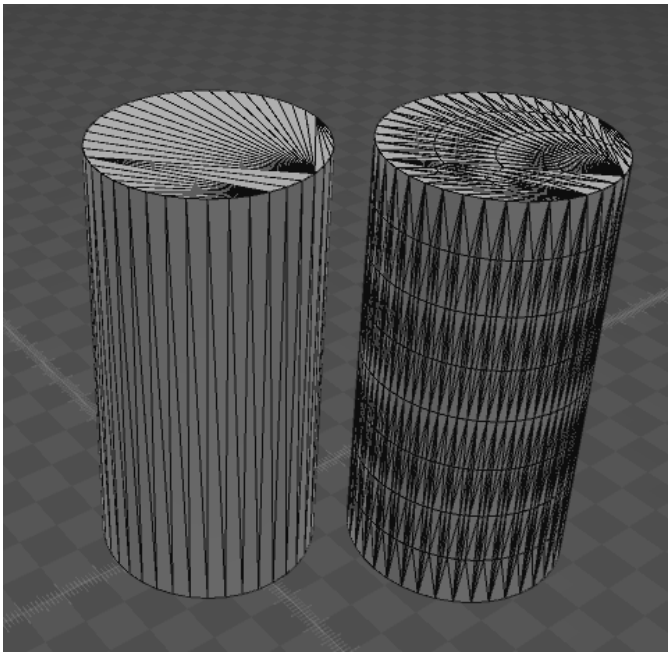

Fig. 6. The original model in the left and the divided model in the right.

There is a simple way to validate that the divided method is right. In fact, the total area of the divided triangles and the original triangles are the same. As Eq. (6) shows, we calculate the total areas of the triangles before and after division to determine this. It should be noted that if we use floating-point type in calculation, some deviations may appear for so many small triangles. Hence double precision type is recommended in verification.

$$S_{total} = \sum_{n=1}^{N_o} S_n = \sum_{p=1}^{N_d} S_p \tag{6}$$

3.4. Spatial subdivision

The world space is divided into uniform grids, whose sizes are denoted as (*sizeX*, *sizeY*, *sizeZ*). Hence, the number of grids is $sizeX \times sizeY \times sizeZ$. 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 Eq. (7).

$$hash = z \times sizeY \times sizeX + y \times sizeX + x \tag{7}$$

As shown in Fig. 7, for each divided triangle and particle, we judge the grid position by their sphere bounding box center coordinates. After getting the grid position, we calculate its grid number (hash value) by Eq. (7). In the implementation of GPU code, it is parallel to calculate the hash values.

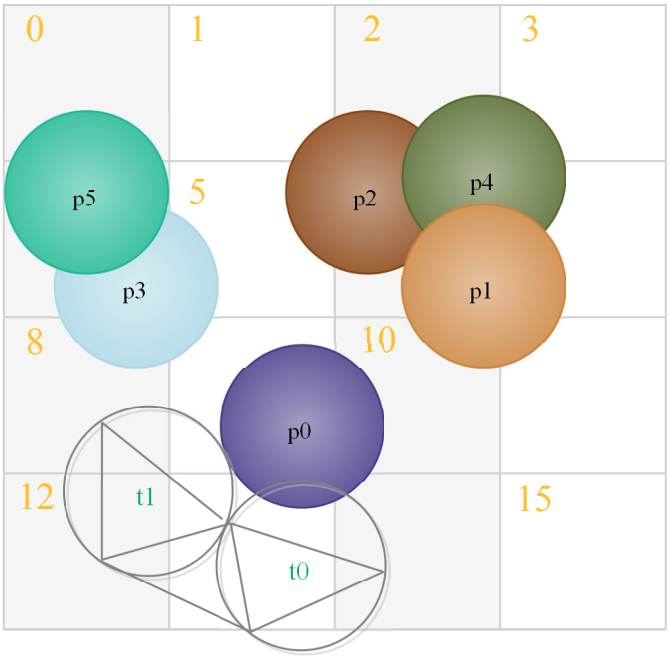

Fig. 7. The distribution of particles and triangles in grids.

Table 1
The parameters of particles in grids.

Index	(cell id,particle id)	sorted	start
0	(9,0)	(4,3)	0
1	(6,1)	(4,5)	
2	(6,2)	(6,1)	2
3	(4,3)	(6,2)	
4	(6,4)	(6,4)	
5	(4,5)	(9,0)	5

3.5. Sorting and data replication

This step is designed for GPU implementation. Because the numbers of particles and triangles in a grid are not fixed, we can not store these data in fixed memory size directly. One way to solve this problem is to reorder the particles array and the triangles array by the hash values (the grid number) of the particles and the triangles, then we can store the particles and the triangles by their start number and end number in a grid. We sort the data arrays by fast radix sort algorithm provided in the CUDPP libary [19].

After sorting the data, we still need to fill in the hash start-arrays and hash end-arrays for particles and triangles in each grid. Because these particles and triangles in the same grid get a same hash value, hence, each particle or triangle finds its start hash and end hash by comparing its grid index with the previous grid index. If there is a difference, it means a new grid and new start hash, as shown in Fig. 7, Table 1 and Table 2. The start and end hash values of particles are (0,1), (2,4), (5,5). The start and end hash values of triangles are (0,0), (1,2).

Table 2
The parameters of triangles in grids.

Index	(cell id,triangle id)	sorted	start
0	(13,0)	(12,1)	0
1	(12,1)	(13,0)	1
2	(13,2)	(13,2)	

3.6. Collision detection

Once the data structures are built and copied to GPU memory, it is used to detect particle-wall interactions. In the continuous collision simulation, we need to consider the specific physical model between particles and walls to obtain the next frame. For example, a discrete element method (DEM) [30, 31] may be used and the forces must be considered. In this case, there is a special case that the collision points are just the intersection points of multi-triangles. To deal with this, we divide the forces into these triangles. More generally, a label is tied to the divided triangles to represent its original triangle number when dividing these triangles. When dealing with these forces, the original triangles are used to replace the divided triangles in collision. In this paper, we focus on the collision detection implementation but not the interaction models.

The collisions is checked in each grid. Hence, it is easy for parallel implementation. For a grid, there are two methods to detect collisions. One way is to traverse based on triangles in a grid, which is named wall based method. That is to say, for triangles in a grid, we firstly find the start triangle and the end triangle, and then we loop over these grids which are adjacent to these triangles. Next we check collisions between these triangles and the particles in these neighboring grids. Another way is to traverse based on particles in a grid, which is called particle based method. Similarly, we find start particle and end particle in a grid, and then we loop to detect collisions with triangles in neighboring grids. There are similarities between these two methods. However, the computational efficiencies are different. The detailed results can be seen in the experimental section.

The last issue of our collision detection algorithm is the intersection test between a sphere and a triangle (mesh). It is equivalent to calculate the shortest distance between the sphere core (set as P) and the triangle (set as ABC). If the distance is smaller than the sphere radius, it means that there is a collision.

To calculate the distance, one basic way is based on the idea: if the point is projected in the triangle, then the shortest distance must be the distance from the point to the plane of the triangle. If the projective point is outside the triangle, the nearest point in this triangle must be located in one side of the triangle. Hence, we can simply get the shortest distance to each edge of the triangle and select the smallest one. Next, we only need to determine whether the projection is inside or outside of the triangle. It is based on the spatial geometry relationship between these points. If the projection is inside of the triangle, for point A, the angles between vector $\vec{AO}$ and vector $\vec{AC}$, vector $\vec{AO}$ and vector $\vec{AB}$ must be both acute angles. Concretely, it

could be determined simply by the vector dot product:

$$\begin{cases} \vec{AO} \cdot \vec{AC} \geq 0 \\ \vec{AO} \cdot \vec{AB} \geq 0 \\ \vec{BO} \cdot \vec{BA} \geq 0 \\ \vec{BO} \cdot \vec{BC} \geq 0 \\ \vec{CO} \cdot \vec{CA} \geq 0 \\ \vec{CO} \cdot \vec{CB} \geq 0 \end{cases} \tag{8}$$

That is, if the points A, B, and C satisfy Eq. (8), point O is projected inside the triangle mesh ABC.

Some other approaches can be used to get the shortest distance between a point and a triangle. For example, Voronoi area method and vector calculus method were researched in [32]. But it is beyond the scope of this paper.

4. The design of Web based simulation service

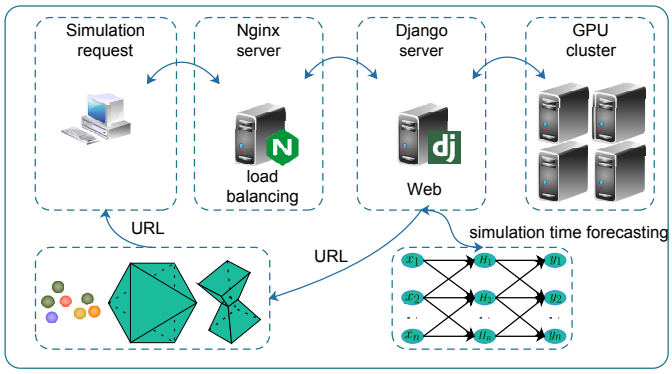

Fig. 8. The structure of simulation system on Web.

Our simulation system provides services over Web. Hence, the core of the system is a Web platform, which is developed by Django framework. The Web site runs on Nginx server, which is also responsible for load balancing of user's requests, as shown in Fig. 8.

The Web site provides basic 3D model functions online. As mentioned above, these 3D model functions are generated by using JSModeler library. The generated models are saved as STL format according to user's configurations. The algorithms described in this paper and other essential simulation algorithms are executed on the GPU cluster. For numerous users, it generates a heavy load for the computing cluster, which reduces the user experience obviously. Hence, a method of managing the computing resources is needed. For example, if a new task needs more resources than the remainder, the task should not be performed. In our implementation, as shown in Fig. 8, we use an ANN based GVM model to predict the computing resources, including GPU occupancy, occupied memory and simulation time of a frame.

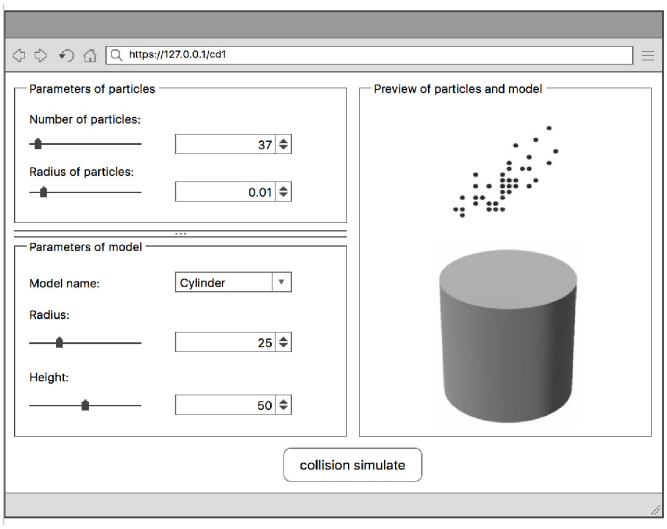

Fig. 9. The parameters configuration of collision algorithm on Web.

4.1. On-line modeling

Our Web site provides some basic vessel models for simulation of collision detection, as shown in Fig. 9. These models are used for general collision experiments, whose parameters are set online. However, the combinations of these models are also available. Users can also design some simple 3D models according to their requirements.

4.2. Tables of Web database

Our Web site uses MySql [4] as database. It mainly includes 5 relational tables.

(1) Admin table
This table is used to store the information of administrator users, including user id, user name and password etc.

(2) User table
The User table includes user id, user name, and password etc. fields for ordinary users.

(3) Model table
The simple models are stored in this table. It includes model id, model name, user name (of model), location (absolute path of the STL model file) etc.

(4) Algorithm table
This table stores the simulation algorithms. It includes the fields of algorithm id, name, category, description, path (execution route) etc.

(5) Log table
The Log table is used to record the history logs of simulation. It mainly includes some running parameters of simulation algorithms. For example, when simulating the algorithm proposed in this paper for collision detection between particles and irregular wall, the parameters are the number of particles, the number of triangles from STL models, memory usage, GPU usage and the computation time of one frame etc. These parameters will be used to train a prediction model for computation resources usage.

[4] https://www.mysql.com/

4.3. Computation resources prediction

In this paper, we utilize GVM to predict the computation resources on the GPU cluster. We take collision detection algorithm in this paper as an example. The GVM model uses the requested physics parameters of collision as the input vector, which include number of particles and number of triangles. These input vectors are exported from the log tables. When simulation tasks are finished on GPU cluster, the needed computation resources respective to different simulation parameters are collected and sent back to Web server as the output of the GVM model.

Table 3
The logs of simulation parameters and needed computation resources for simulation tasks.

Particles ($\times 10^3$)	Triangles	GPU (%)	Mem (M)	Time (s)
1	100	0.5	1.1	0.036
2.2	120	0.6	2.3	0.065
10	260	1.1	9.7	0.182

Table 3 shows some of the logs collected in Web server, in which *Particles* denotes the number of particles in simulation, $Triangles$ represents the number of triangles, GPU denotes the GPU occupancy and Mem denotes the occupied memory by simulation task. At last, $Time$ shows the simulation time. The system takes the parameters $Particles, Triangles$ as the input vector of the GVM model, and the parameters GPU, Mem, $Time$ as the output vector of GVM model. The GVM model adjusts its weight matrices according to these input-output datasets by MC algorithm [7]. After a period time of training, the GVM model is able to predict the computation resources of a new incoming task. With these predictions, the system is able to estimate the system load and determine whether start a new task or not. Meanwhile, the Web system could further be able to dispatch computation resources more efficiently.

Computation time on sever system, specifically on our GPU cluster, is an important parameter for the simulation process. With our GVM model deployed, we are able to provide instant feedback when a new task is uploaded. Moreover, the resource scheduling based on our GVM model prediction would be interesting, though it is not in the scope of this paper.

4.4. Compute nodes

In reality, compute nodes share the similar physics simulation algorithms. One compute node mainly handles three tasks.

(1) Simulation computation
It executes the simulation algorithms designed in this paper with CPU or GPU.

(2) Parameter collection
It collects the parameters of computation resources and sends these parameters to Web site for training GVM model of load prediction.

(3) Results management
The results are managed on the compute node. It provides hyperlink of the simulation results to users. The results are stored as text files, images or videos.

5. Experimental Results and Analysis

In this section, we discuss the experimental results of the collision detection algorithms designed in this paper.

5.1. Experimental environment

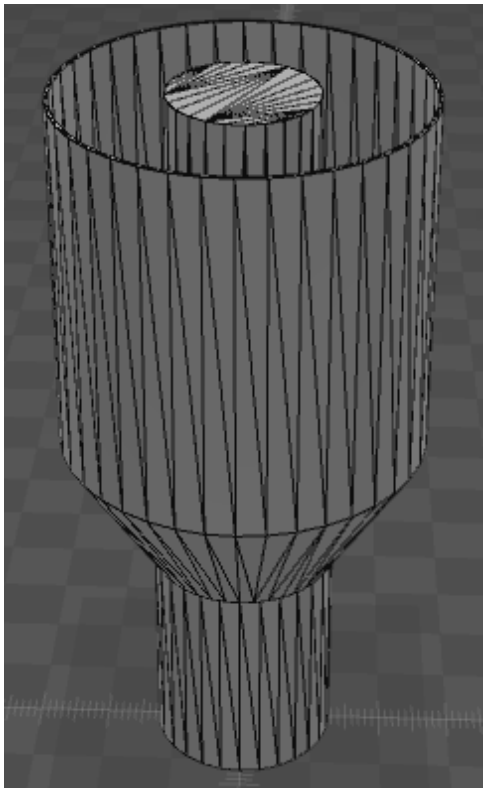

Fig. 10. A complex collision wall, named *wall*2.

The experiments were performed on a compute node with GTX480 GPU and a compute node with Tesla K80 GPU, which represent the low-end and high-end GPUs respectively. We generate a cylinder model whose height is 100 and radius is 50 online as a simple collision wall, as shown in Fig. 6. We name it *wall*1. We also generated a complex model as shown in Fig. 10, which is similar to the spallation target in the simulation of ADS. We name it *wall*2. The sizes of particles are different, but the maximum radius of these particles is set as 1.

Next, our experiments mainly focus on the different efficiencies between CPU and GPU code, the difference between two traversal methods, the difference between gravity based method and circumcircle based method for triangle division, the influence of the number of triangles, the influence of grid length and the difference on different GPUs. The relationship between parameters of collision algorithm and computation time is explored at last.

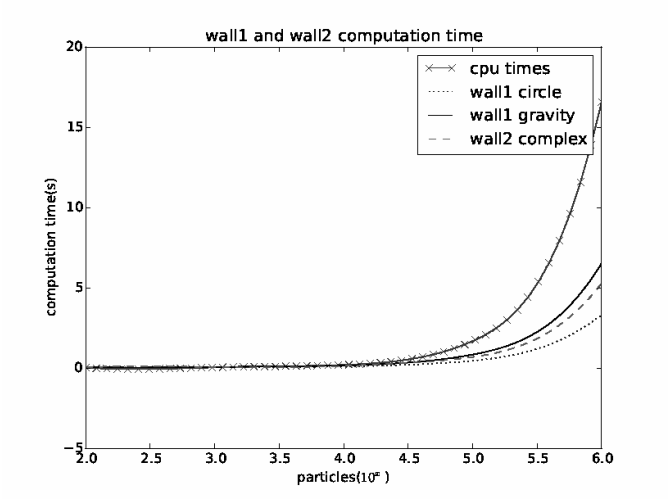

Fig. 11. Experimental results of computation time: CPU, *wall*1 circumcircle based method, *wall*1 gravity based method, *wall*2 circumcircle based method.

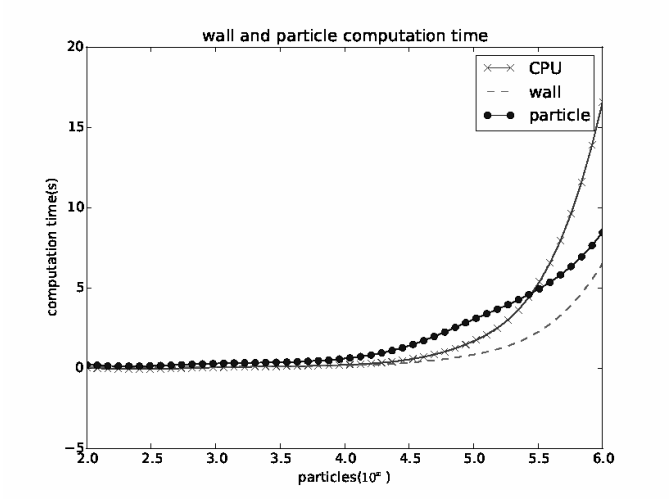

Fig. 12. Experimental results: the comparison between wall based method and particle based method.

5.2. CPU and GPU

Table 4
Experimental results of computation time.

number	10^2	10^3	10^4	10^5	10^6
CPU time(s)	0.025	0.038	0.195	1.674	16.56
GPU gra (s)	0.056	0.068	0.159	0.836	6.518
GPU cir (s)	0.037	0.044	0.103	0.453	3.315
GPU cir2(s)	0.045	0.057	0.138	0.683	5.265

In order to measure the performance of the algorithm, we choose different particle numbers as the input variable. We make the experimental on CPU and GPU platforms. As shown in Fig. 11 and Table 4 (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 less than 10^3, the CPU and GPU codes have similar computation timings. However, when the number of particles is more than 10^4, the computation time of CPU code increases considerably. When the number of particles are 10^4, 10^5 and 10^6, compared to CPU code, the speed-up ratios of GPU by circumcircle based method are 1.89, 3.69 and 4.99, respectively. Hence, the services of physic simulations service are implemented on GPU.

5.3. Two traversal methods

As discussed above, there are two methods to detect collision based on GPU, wall based method and particle based method. Fig. 12 draws the different results of these two methods. For particle based method, when the number of particles is less than $10^{5.5}$, the method is even worse than CPU code. When the particles number is larger than $10^{5.5}$, the particles based method performs better than CPU algorithm, but no better than wall based algorithm. The reason is that, the number of triangles is much smaller than the number of particles, so the particle based algorithm needs more concurrent threads to finish collision detection.

5.4. Gravity based method and circumcircle based method

In the section of triangle mesh division, we have discussed gravity based and circumcircle based division methods. Now we compare these two methods on experimental data. For the simple model *wall*1, as shown in Fig. 6, the original number of triangles is 156. By gravity based method, the minimum length of the divided triangles is set as 1 and we get 226056 triangles after division. By circumcircle based method, the minimum length is set as $\sqrt{3}$ and the number of divided triangles is 99612 which is less than half of the number of gravity based method.

As shown in Fig. 11 and Table 4, we find that the circumcircle based method is more efficient than the gravity based method in computation. A plausible explanation is that the circumcircle based method uses the collision space more effectively and gets fewer divided triangles. In fact, the computation time is relevant to the number of triangles, we can see this from next section. In our service system, circumcircle based method is used for physic simulation.

5.5. The influence of number of triangles

We tested the influence of the number of triangles based on fixed number of particles. In Fig. 13, we set the number of particles as fixed 10^5 and 10^6 to test the relationship between number of triangles and computation time. We roughly draw a conclusion that the computation time is proportional to the the number of triangles. We get the following linear fitting equations.

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

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

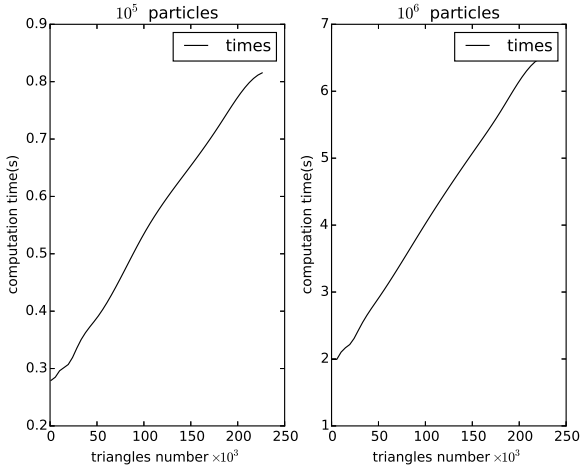

Fig. 13. Experimental results: the influence of the number of triangles.

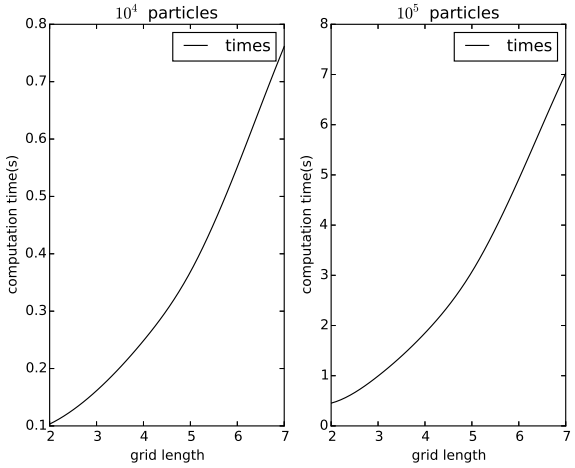

Fig. 14. Experimental results: the influence of grid length.

From Eq. (9) and Eq. (10), we can conclude that: to effectively make collision detection by our algorithm, the number of triangles can not be too large. That is, the model can not too large compared to the sizes of particles. For example, if there are 10^6 particles and 10^8 particles, the forecasting simulation time of a collision frame will be 2080 seconds, which is not acceptable for users.

We also tested the complex model *wall*2 by circumcircle based method. The original number of triangles of model *wall*2 is 1600. The number of divided triangles is 151424. As shown in Fig. 11, it even runs faster than the simple gravity based method of *wall*1. The reason is that the number of divided triangles of *wall*1 based on gravity method is 226056, larger than the number of divided triangles of *wall*2 based on circumcircle method. Therefore, using a larger bounding sphere box to reduce the number of divided triangles is a better way to improve computational efficiency.

5.6. The influence of grid length

Another concern is the effect of the grid length. Given a fixed irregular wall, the detecting boundary is fixed. So the number of grid is inversely proportional to the length of the grid. As shown in Fig. 14, the computation time increases with the rise of grid length. As discussed above, the length of the grid is set as twice of the maximum particle radius in this paper. What obvious is that the smaller the grid length is, the more grids and memory the code occupies.

5.7. On different GPUs

We tested our algorithm on GTX480 and Telsa K80 GPU. As shown in Fig. 15, when the number of particles is less than 10^4, the code on GTX480 computes faster. However, once the number of particles is more than 10^4, for instance, 10^4, 10^5 and 10^6, compared to GTX480, the speed-up ratios of K80 are 1.15, 2.14 and 3.04. Hence, we conclude that with the increase of the number of particles, GPU with higher computation ability will obviously improve computation efficiency.

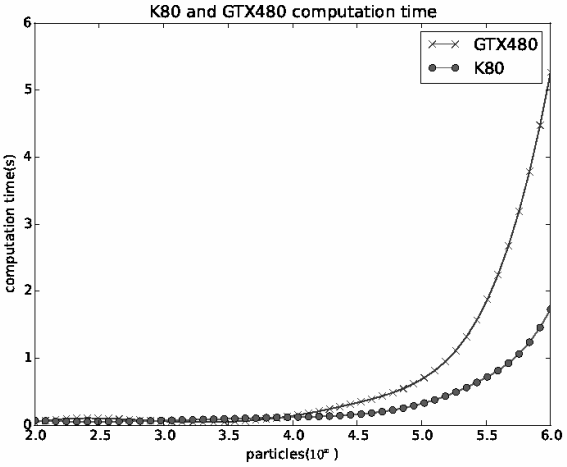

Fig. 15. Experimental results: the comparison between GTX480 and K80.

5.8. Computation resources prediction

Overall, we have explored the effects of various factors on the time of simulation computation. However, when fixing GPU and algorithm, number of particles and number of triangles are the main factors influencing the computation resources. We collect 100 data on various number of particles, triangles and their corresponding computation resources in a frame. 70 data are used as our training dataset and 30 data are used as the validation dataset. As shown in Fig. 16, the validation loss decreases rapidly along with the train loss and reach a minimal value after 100 epochs. Both train loss and validate loss have a similar convergent trend, and the loss curves are smooth, which indicates that the GVM model is able to effectively predict the relationship between collision parameters and needed computation resources.

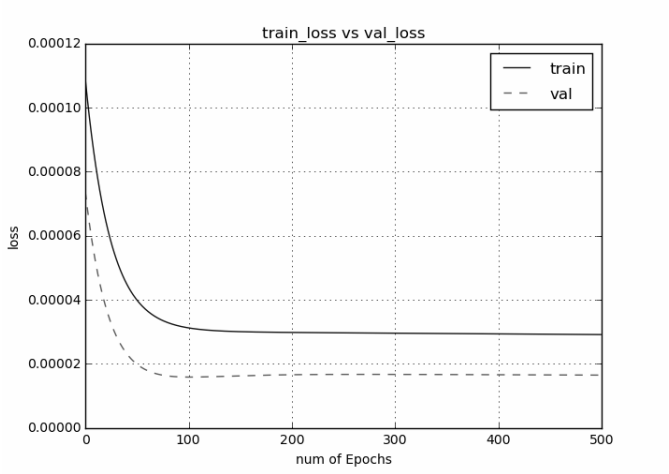

Fig. 16. Experimental results: train loss and validate loss.

6. Conclusion

In our current work, it is more meaningful to focus on collision detection between irregular walls and particles by using GPU to accelerate this computation process. In this paper, we have designed some algorithms to achieve this goal. In the experiment part, we have tested and compared the efficiencies between CPU and GPU based algorithm, particles and triangles based traverse methods, circumcircle based and gravity based method. At the same time, we discuss the influences of different number of particles, grid length, different GPUs and relationship between these parameters and computation time. The experiment results prove that the algorithms proposed in this paper are not only feasible and effective, but also efficient.

The collision detection of a large number of particles in the physic simulation is still very challenging, especially the real-time display of a large number of particles is still an open issue. With the help of a K80 GPU, we can detect collision between a million particles and a fairly complex model in no more than 2 seconds. We can also conclude that one potent way to improve computing efficiency in this algorithm is to reduce the number of the divided triangles. In this paper, we present two methods for triangle division. However, as we know, when the length of a triangle's edge is limited, an equilateral triangle is the biggest triangle. Therefore, dividing triangles into equilateral triangles will reduce the number of triangles after dividing, so as to improve the computation efficiency considerably. Delaunay triangulation is a very important technology in the field of finite element analysis and computer graphics [33, 34]. The triangles divided by this method are approximately regular. So Delaunay triangulation is instructive in triangle division. However, this method is used for polygon triangulation rather than triangle division. More generally, how to divide a large triangle into small triangles, whose side length is limited to a fixed maximum value, and make sure that the number of divided triangles is minimum, is still an issue.

On the other hand, providing simulation services on Web is one of the future development directions of next generation Web application. In general, more simulation services could be provided to users who are lacking high cost simulation environment, such as chemical reaction simulation, biological evolution simulation and more physic simulation etc. Such systems can be implemented as cloud platforms. With the increase of the number of users, task scheduling algorithm is more and more important to intelligently assign compute nodes. Although a prototype system was implemented in our paper, further explorations on how to optimize such on-line simulation system still need to be performed.

Acknowledgment

This work was supported by 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] Gino van den Bergen, Efficient Collision Detection of Complex Deformable Models using AABB Trees[J], in: Journal of Graphics Tools, 1997, 2(4):1-13.
[2] Chang J W, Wang W, Kim M S, Efficient collision detection using a dual OBB-sphere bounding volume hierarchy[J], in: Computer-Aided Design, 2010, 42(1):50-57.
[3] Tan R, Zhao W, Li J, The Study of Parallel Collision Detection Algorithms[C], in: International Conference on Multimedia Technology. IEEE, 2010:1-4.
[4] M. Garland, Parallel computing with CUDA, in: IEEE Computer Society, 2010,
[5] NVIDIA, CUDA C Programming Guide, in: NVIDIA, 2013
[6] H. Wang, B. Raj, On the origin of deep learning, in: arXiv preprint, arXiv: 1702.07800, 2017
[7] H. Zhao, General vector machine, in: arXiv preprint, arXiv: 1602.03950, 2016
[8] H. Chen, H. Zhao, J. Shen, R. Zhou, Q. Zhou, Supervised machine learning model for high dimensional gene data in colon cancer detection, in: IEEE International Congress on Big Data, 2015, pp. 134-141
[9] Q. Zhou, H. Chen, H. Zhao, J. Shen, A local field correlated and monte carlo based shallow neural network model for nonlinear time series prediction, in: EAI Endorsed Transactions on Scalable Information Systems, 2016, pp. 1-7
[10] L. Wang, J. Shen, Q. Zhou, Z. Shang, H. Chen, H. Zhao, An evaluation of the dynamics of diluted neural network, in: International Journal of Computational Intelligence Systems, 2016, pp. 1191-1199
[11] B. Yong, Z. Xu, J. Shen, H. Chen, Y. Tian, Q. Zhou, Neural network model with monte carlo algorithm for electricity demand forecasting in queensland, in: Australasian Computer Science Week Multiconference, 2017,
[12] M. T. Hagan, H. B. Demuth, M. H. Beale, Neural Network Design, in: PWS Pub. Co., 1995
[13] T. J. Purcell, I. Buck, W. R. Mark, P. Hanrahan, Ray tracing on programmable graphics hardware, ACM Transactions on Graphics (TOG), in: Acm Transactions on Graphics, 2002, pp. 703-712
[14] G. Baciu, W. S.-K. Wong, H. Sun, Recode: an image-based collision detection algorithm, in: Computer Graphics and Applications, 1998. Pacific Graphics' 98. Sixth Pacific Conference on, IEEE, 1998, pp. 125-133
[15] K. Myszkowski, O. G. Okunev, T. L. Kunii, Fast collision detection between complex solids using rasterizing graphics hardware, The Visual Computer, in: The Visual Computer, 1995, pp. 497-511

[16] N. K. Govindaraju, S. Redon, M. C. Lin, D. Manocha, Cullide: Interactive collision detection between complex models in large environments using graphics hardware, in: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, Eurographics Association, 2003, pp. 25-32
[17] P. Kipfer, M. Segal, R. Westermann, Uberflow: a gpu-based particle engine, in: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware, ACM, 2004, pp. 115-122
[18] J. Zheng, X. An, M. Huang, Gpu-based parallel algorithm for particle contact detection and its application in self-compacting concrete flow simulations, in: Computers & Structures, 2012, pp. 193-204
[19] NVIDIA, Particle Simulation using CUDA, NVIDIA, 1st Edition
[20] S. Xue-li, Z. Ji-suo, Research of collision detection algorithm based on particle swarm optimization, in: Computer Design and Applications (ICCDA), 2010
[21] H. Qu, W. Zhao, Fast collision detection algorithm based on parallel ant, in: International Conference on Virtual Reality and Visualization, 2013, pp. 261-264
[22] S. Xue-li, L. Tao, Fast collision detection based on projection parallel algorithm, in: Future Computer and Communication (ICFCC), 2010
[23] H. Qu, W. Zhao, Fast collision detection of space-time correlation, in: Computer Science and Electronics Engineering (ICCSEE), 2012, pp. 567-571
[24] R. Xu, L. Kang, H. Tian, A g-octree based fast collision detection for large-scale particle systems, in: Computer Science and Electronics Engineering (ICCSEE), 2012, pp. 269-273
[25] Z. Yisheng, Z. Xiaoli, D. Guofu, H. Yong, J. Meiwei, A GPGPU-based collision detection algorithm, in: International Conference on Image and Graphics, 2009, pp. 938-942
[26] H. G. W. Z. W. Fan, S. Gao, Streaming real time collision detection using programmable graphics hardware, in: Journal of Software, 2004, pp. 1505-1513
[27] E. Sakkopoulos, E. Sourla, A. Tsakalidis, M. D. Lytras, Integrated system for e-health advisory web services provision using broadband networks, in: International Journal of Social and Humanistic Computing (IJSHC), 2008, pp. 36-52
[28] J. Ojala, Personal content in online sports communities: motivations to capture and share personal exercise data, in: International Journal of Social and Humanistic Computing (IJSHC), in: 2013, pp. 68-85
[29] A. Perttula, P. Tuomi, K. Kiili, M. Suominen, A. Koivisto, J. Multisilta, Enriching shared experience by collective heart rate, in: International Journal of Social and Humanistic Computing (IJSHC), 2013, pp. 31-50
[30] H. Karunasena, W. Senadeera, Y. Gu, R. Brown, A coupled sph-dem model for fluid and solid mechanics of apple parenchyma cells during drying, in: 18th Australian Fluid Mechanics Conference, 2012
[31] M. Rhodes, X. S. Wang, M. Nguyen, P. Stewart, K. Liffman, Study of mixing in gas-fluidized beds using a dem model, in: Chemical Engineering Science, 2001, pp. 2859-2866
[32] C. Ericson, Real-Time Collision Detection, in: ELSEVIER, 2010
[33] P.-L. George, H. Borouchaki, Delaunay triangulation and meshing, in: Hermes, 1998
[34] B. Joe, Construction of three-dimensional delaunay triangulations using local transformations, in: Computer Aided Geometric Design, 1991, pp. 123-142