Real-time Simulation of Granular Materials Using Graphics Hardware

Ren Yasuda
University of Tokyo
qq086315@iii.u-tokyo.ac.jp

Takahiro Harada
University of Tokyo

Yoichiro Kawaguchi
University of Tokyo

Abstract

We present a method to compute friction in a particle-based simulation of granular materials on GPUs and its data structure. We use Distinct Element Method to compute the force between particles. There has been a method to accelerate Distinct Element Method using GPUs, but the method does not compute friction. We implemented friction into the DEM simulation on GPUs and this leads to the real-time simulation of granular materials.

1 Introduction

Recently, there has been an increase in research of physical simulations using a graphics processing unit (GPU) for general purpose computations. Applications includes, for example, fluid simulations, rigid body simulations. One of these research topics is a particle-based simulation of granular materials. However, in the previous work, friction did not take into account and so that particles behave as if they were fluid. In this paper, we present a model and a data structure that computes friction for the distinct element method (DEM) using GPUs as stream processors.

2. Previous Work

2.1 Distinct Element Method

The distinct element method (DEM), introduced by Cundall *et al.*[3], is a method used to simulate the movement of a large number of rocks and sand particles in order to analyze the avalanche of rocks and earth.

In the computer graphics field, Bell *et al.* proposed a model that computes the interaction between rigid bodies and particles using DEM [2]. Tanaka *et al.* applied DEM for the computation of the motion of rigid bodies, and simulated the interactions between rigid bodies and fluids [10].

2.2 General Purpose Computation on GPUs

As the programmability of the GPU is increasing, they have become a generally-used tool for computation in several fields of research. In the computer graphics field, Amanda *et al.* and Harada *et al.* have accelerated the implementation of the smoothed particle hydrodynamics (SPH), one of the particle-based methods for the analysis of fluids, using GPUs [1][5]. In addition, Purcell *et al.* have accelerated photon mapping using one of the global illumination algorithms [9]. There are several researches for general computation on the GPU. A good survey can be found in [8].

2.3 Acceleration of DEM Using GPUs

Although Harada *et al.* proposed a method for accelerating DEM using GPUs, the friction between particles is not computed in their method [6].

3. Calculation Techniques

3.1 Collision Response

In the DEM, the force added to a particle is calculated as the summation of interparticle repulsion, damping, and frictional forces. The repulsion force is represented as a force proportional to the penetration between particles, and the damping force is modeled as a force proportional to relative speed of particles. The frictional force between two particles is modeled as a force proportional to the displacement from the position where two particles have collided, and so the process differs depending on whether the two particles collided or not at the previous time step [7]. If the two particles did not collide at the previous step, friction is not considered. However, it is necessary to save the relative position,

$$\boldsymbol{r}_{ij}^0 = \boldsymbol{r}_j^0 - \boldsymbol{r}_i^0, \tag{1}$$

because there is a possibility that these particles will collide later. On the other hand, if the two particles did collide

978-0-7695-3359-9/08 $25.00 © 2008 IEEE
DOI 10.1109/CGIV.2008.45

at the previous step, the displacement is computed from the first position of collision, and the frictional force is obtained from the displacement. Let $\delta \boldsymbol{r}_{ij}$ be the displacement, and therefore $\delta \boldsymbol{r}_{ij} = \boldsymbol{r}_{ij} - \boldsymbol{r}^0_{ij}$. The frictional force can then be calculated as:

$$[\boldsymbol{F}_s]_i = k_s[\delta \boldsymbol{r}_{ij}]_s, \tag{2}$$

where $[\delta \boldsymbol{r}_{ij}]_s$ is the shear component of $\delta \boldsymbol{r}_{ij}$. However, if $[\boldsymbol{F}_s]_i$ is larger than the maximum static friction force, then the kinetic frictional force is added instead. Therefore, $[\boldsymbol{F}_s]_i$ calculated as:

$$[\boldsymbol{F}_s]_i = \mu_k |\boldsymbol{F}_n| \frac{[\delta \boldsymbol{r}_{ij}]_s}{|[\delta \boldsymbol{r}_{ij}]_s|} \tag{3}$$

3.2 Boundary Conditions

Although all the elements of the simulation are usually modeled by particles, as in a particle-based method, a distance function is used as the boundary condition for the simulation so that collisions between particles and the boundary must be computed independently from particle-to-particle collisions. The distance function is a function that provides the minimum distance to the given boundary and the normal vector at a point on the boundary that minimizes the distance; therefore, the collision distance to the boundary can be easily obtained. Furthermore, use of the distance function to define the boundary can reduce the computational cost compared with the use of polygons or particles.

4 Implementation

4.1 Data Structure

The data necessary to calculate friction to a particle is as follows: the index of collided particles, the relative position between each collided particle, and the number of collided particles. In the DEM, information regarding the collision at the previous time step is required to calculate the friction as described in 3.1. It is necessary to allocate the memory in advance because GPUs cannot allocate the memory dynamically. In the present study, a particle does not collide with more than 12 particles because all particles have the same diameter. And the first collision position used to calculate friction is obtained from the number of particles and the index of the particles at the previous time step. Memory is accessed based on the number of particles to obtain the first collision; therefore, this method accompanying with less memory access results in an increase in processing speed.

In this study, the distance function was used to represent walls. There has never been a model that computes the boundary using the distance function to compute frictional force between particles and the boundary. Therefore, we present a model that computes friction between particles and the walls by applying a model of interparticle friction. In this model, the frictional force between particles and walls is separately computed from the interparticle friction. Therefore, it is necessary to prepare two data for each particle as follows: collision position with the wall and the normal vector of the wall at the collision position. Whether it is necessary to compute the frictional force between particles and walls or not is determined by the normal vector. And if necessary, the frictional force is computed.

As noted in 3.2, distance function was used to define the boundary conditions. Therefore, data used to compute collision to the boundary, the first collision point, and the normal vector at the position are required, in addition to the data of the particle-to-particle collision described above. Computation of friction to the boundary requires data indicating whether or not collision occurred between the particles and boundary, as for the friction between particles and particles. However, incorrect friction may be computed if those particles collide with two walls (boundaries) that face each other for the second time in a row. To prevent such a case, friction between particles and the boundary is added only if the particle collides at a nearly continuous point on the boundary from the collision point at the previous time step. This is ascertained on the basis of the normal vectors at the present time step and the previous time step. Therefore, frictional force is added if the following equation is satisfied,

$$\boldsymbol{n}_{prev}\boldsymbol{n}_{cur} < 1 - \eta, \tag{4}$$

where $\boldsymbol{n}_{prev}$ and $\boldsymbol{n}_{cur}$ represent the normal vectors at the present time step and at the previous time step, respectively.

4.2 Grid Construction

Even though the computational power of the GPU is tremendous, the computing time is too expensive to compute collision for each particle with all other particles. Therefore, we detect collision more effectively by building a grid prior to collision detection. In this method, we have generalized the uniform grid developed by Harada *et al.*, and developed a model that does not use graphics API (Application Programming Interface) [5].

The process of grid construction using the uniform grid on CPUs is as described below.

1. Compute in which bucket particle i is.
2. Load the number N, the number of particles that was previously registered to the bucket.
3. Access the array of the bucket based on the number of particles and register i, the index of the particle.
4. Increment N.

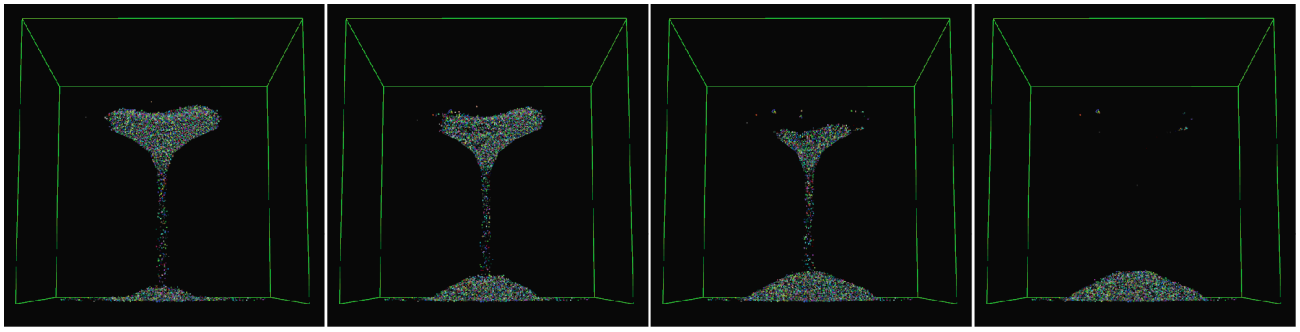

Fig. 1. 50000 particles fall into the bottom part of an hourglass through a hole in real-time.

However, this process would cause errors if implemented on GPUs, because streaming processors on GPUs run independently. Therefore, we have to modify the algorithm when it is implemented on GPUs. Particle indices are stored to the grid by four kernels. The procedure in a kernel is

1. Compute in which bucket particle i is.
2. Load the indices of particles that were previously registered to the bucket, and if i is found in those indices, switch to step 4.
3. Store i to pth element of the array of the bucket.

The number p represents how many times the process repeated. We set the size of each bucket of the grid not to be able to contain more than 4 particles, and then it is ensured that all particles are registered to the grid.

5 Results and Discussion

The programs were written in C++ and Compute Unified Device Architecture(CUDA) on a 3.00 GHz Core 2 Duo PC and Geforce 8800 ultra GPU. Fig.1 shows the screenshots of the simulation using an hourglass-like boundary condition. The particles generate a mound if friction is computed, as shown in Fig.2. On the other hand, if friction is not computed, the particles do not generate a mound, and this behavior is evidently unnatural for granular materials.

Fig.3 shows the transition of computation time per step until all particles fall into the bottom part of the hourglass. As shown in Fig.3, the computation time increases with time, and this is attributed to lowering of the cache hit ratio. At first, the physical values of particles have locality in the memory space, and therefore the cache hit ratio remains high. However, particles are scattered in the hourglass over time, and this causes lowering of the cache hit ratio.

Fig.4 shows transition of computation time per step in a scene. In this scene, a slug of particles falls freely in a cube-like boundary condition at first. Then, they pile up after collision against the wall. In the present architecture of the GPU, memory access is much expensive than floating point computation. As described above, memory access increases as more particles collide and leads to the delay. In the scene shown in Fig.4, number of colliding particles is small at first because particles. However, the particles are scattered soon after colliding against the wall, and so a pulse-like wave can be seen in the graph of the simulation with friction in Fig.4. In addition, in the graph showing the simulation without friction, we can see a gentle curve instead of a pulse-like wave, which is due to lowering the cache hit ratio as shown previously Fig.3. The computation time of the simulation with friction then increases gradually because the number of collisions increases as particles accumulate. On the other hand, the computation time of the simulation without friction decreases with time. This is because particles become less scattered, increasing the cache hit ratio with time.

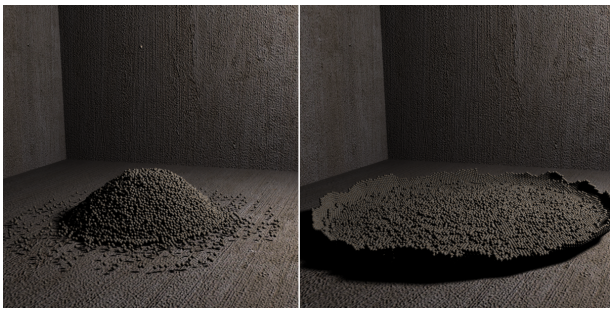

Fig. 2. Screenshots of simulation with friction (left) and without friction (right).

In the present study, texture memory and the cache were used to accelerate the process of searching for neighbor particles by reducing the overhead of memory access. However, the computational time increases as the particles scatter and the cache hit ratio increases; therefore, it is undesirable that the computation time changes with the state of the

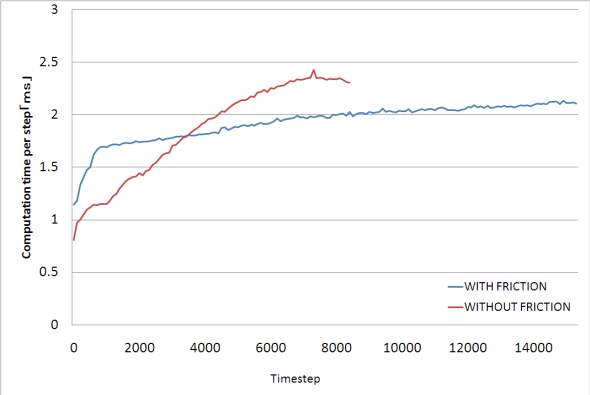

Fig. 3. Transition of computation time

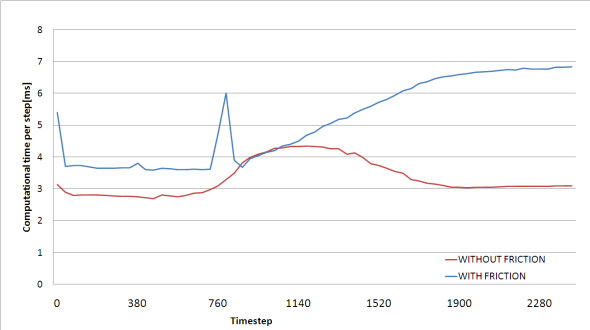

Fig. 4. Computation time

scene if this model is applied to a real-time application. As a solution to this problem,sorting particle data would be useful to ensure that nearby particles have a close index number to maintain a high cache hit ratio. However, an introduction of sorting does not improve the performance when the overhead of sorting is larger than the performance gain. even when using the fastest method of sorting on GPUs that has been reported to date [4]. Even the radix sort on the GPU is too slow to use for this purpose[4]. Studying much faster sorting algorithm on the parallel architecture is a future work.

On the other hand, shared memory can be used as an alternative to texture memory, in order to make the computation more efficient. Although much data should be shared between threads to exploit shared memory, this can also be achieved by sorting particles, with the overhead. Moreover, access to the shared memory has various limitations, and these limitations makes it difficult to use shared memory for acceleration of particle-based simulations. For these reasons, GPUs with large-capacity cache memory, as opposed to those with shared memory, are the architectures suited for accelerating particle-based methods.

In this research, there was no point to share the data used to compute friction, because these data are the relative positions between each particle, and so memory access would cause delay, even if the issue of the cache hit ratio and shared memory was solved. The memory access results in an unavoidable delay in the computation of vast amounts of data on GPUs, such as the particle-based model simulated in this paper. As a result, improvement memory access latency is absolutely essential for physically-based simulations.

Acknowledgements

This research was supported by Core Research for Evolution Science and Technology (CREST) of Japan Science and Technology Agency(JST).

References

[1] T. Amada, M. Imura, Y. Yasumuro, Y. Manabe, and K. Chihara. Particle-based fluid simulation on gpu. *ACM Workshop on General-Purpose Computing on Graphics Processors and SIGGRAPH*, 2004.

[2] N. Bell, Y. Yu, and P. Mucha. Particle-based simulation of granular materials. *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 77–86, 2005.

[3] P. Cundall and O. Strack. A discrete numerical model for granular assemblies. *Geotechnique*, 29(1):47–65, 1979.

[4] S. L. Grand. *GPUGems3*. Addison-Wesley, 2007. ISBN: 0321515269.

[5] T. Harada, S. Koshizuka, and Y. Kawaguchi. Smoothed particle hydrodynamics on gpus. *In Proc. of Computer Graphics International*, pages 63–70, 2007.

[6] T. Harada, M. Tanaka, and S. Koshizuka. Acceleration of distinct element method using graphics hardware. *Transactions of JSCES*, 2007, 2007.

[7] J. Lee and H. Herrmann. Angle of Repose and Angle of Marginal Stability: Molecular Dyanmics of Granular Particles. *Arxiv preprint cond-mat/9211016*.

[8] J. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. Lefohn, and T. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.

[9] T. Purcell, C. Donner, M. Cammarano, H. Jensen, and P. Hanrahan. Photon mapping on programmable graphics hardware. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 41–50, 2003.

[10] M. Tanaka, M.Sakai, and S. Koshizuka. Particle-based rigid body simulation and coupling with fluid simulation. *Transactions of JSCES*, 2007, 2007.