Manuscript Number: AMC-D-14-02399

Title: Collision detection of convex polyhedra on the Nvidia GPU architecture for the Discrete Element Method.

Article Type: SI: ESCO 2014

Keywords: GPU, DEM, Polyhedra, CUDA, Collision Detection, NVIDIA.

Corresponding Author: Mr. nicolin govender,

Corresponding Author's Institution:

First Author: nicolin govender

Order of Authors: nicolin govender; Daniel Wilke, PhD; Schalk Kok, PhD

Abstract: Convex polyhedra represent granular media well. This geometric representation may be critical in obtaining realistic simulations of many industrial processes using the discrete element method (DEM). However detecting collisions between the polyhedra and surfaces that make up the environment and the polyhedra themselves is computationally expensive. This paper demonstrates the significant computational benefits that the graphical processor unit (GPU) offers DEM. As we show, this requires careful consideration due to the architectural differences between CPU and GPU platforms. This paper describes the DEM algorithms and heuristics that are optimized for the parallel NVIDIA Kepler GPU architecture in detail. This includes a GPU optimized collision detection algorithm for convex polyhedra based on the separating plane (SP) method. In addition, we present heuristics optimized for the parallel NVIDIA Kepler GPU architecture. Our algorithms have minimalistic memory requirements, which enables us to store data in the limited but high bandwidth constant memory on the GPU. We systematically verify the DEM implementation, where after we demonstrate the computational scaling on two large-scale simulations. We are able achieve a new performance level in DEM by simulating 34 million polyhedra on a single NVIDIA K6000 GPU. We show that by using the GPU with algorithms tailored for the architecture, large scale industrial simulations are possible on a single graphics card.

Dear Prof Solin

We thank you for taking the time to consider our paper.

In our paper we present the algorithms behind our GPU based discrete element method (DEM) code BLAZE-DEM that can model millions of polyhedral and spherical particles on a single desktop computer. This paper introduces collision detection algorithms for convex polyhedra based on the separating plane (SP) method optimized for the GPU. In addition, we present heuristics optimized for the parallel NVIDIA Kepler GPU architecture and validate that the correct numerical results is obtained on the GPU.

We show that by using the GPU with algorithms tailored for the architecture, large scale industrial simulations are possible on a single graphics card.

Regards

Nicolin Govender

Collision detection of convex polyhedra on the Nvidia GPU architecture for the Discrete Element Method.

Nicolin Govender*[1,2] , Daniel N Wilke[2], Schalk Kok[2]

[1] *Advanced Mathematical Modeling CSIR, Pretoria, 0001, South Africa*

[2] *University of Pretoria, Department of Mechanical and Aeronautical Engineering, Pretoria, 0001, South Africa*

Abstract

Convex polyhedra represent granular media well. This geometric representation may be critical in obtaining realistic simulations of many industrial processes using the discrete element method (DEM). However detecting collisions between the polyhedra and surfaces that make up the environment and the polyhedra themselves is computationally expensive. This paper demonstrates the significant computational benefits that the graphical processor unit (GPU) offers DEM. As we show, this requires careful consideration due to the architectural differences between CPU and GPU platforms. This paper describes the DEM algorithms and heuristics that are optimized for the parallel NVIDIA Kepler GPU architecture in detail. This includes a GPU optimized collision detection algorithm for convex polyhedra based on the separating plane (SP) method. In addition, we present heuristics optimized for the parallel NVIDIA Kepler GPU architecture. Our algorithms have minimalistic memory requirements, which enables us to store data in the limited but high bandwidth constant memory on the GPU. We systematically verify the DEM implementation, where after we demonstrate the computational scaling on two large-scale simulations. We are able achieve a new performance level in DEM by simulating 34 million polyhedra on a single NVIDIA K6000 GPU. We show that by using the GPU with algorithms tailored for the architecture, large scale industrial simulations are possible on a single graphics card.

Keywords: GPU, DEM, Polyhedra, CUDA, Collision Detection, NVIDIA.

1. Introduction

The discrete element method (DEM) approach which was first described by Cundall and Strack [1], is one of the most successful methods to simulate granular media (GM) [2, 3, 4, 5]. DEM simulations require that all particles in the system have to be checked for contact at each time step, which involves a considerable number of calculations depending on particle geometry and number of particles [6]. To reduce the computational cost, particle shape is often approximated by spheres, for which contact detection is trivial. This approximation however results in the system exhibiting unrealistic mechanical behavior, as discussed by Latham and Munjiza [7, 8]. Polyhedral shaped particles represent most GM well and hence exhibit realistic mechanical behavior to that of the actual system [9, 10]. However, the number of polyhedral particles that can be simulated on typical workstation computers in a realistic time frame is limited, as discussed by Mack [11], in which only 322 polyhedra are simulated. This limitation is due primarily to complex collision detection and storage costs of the polyhedra.

Driven by the demand for real-time 3D graphics, with prices kept low due to high selling volumes from the consumer gaming market, the programmable Graphic Processor Unit (GPU) offers cluster type performance at a fraction of the cost [12]. The parallel nature of the GPU allows a large number of simple independent processes to be executed in parallel. This results in significant speed ups over CPU implementations, for suitable algorithms.

2. The Graphical Processor Unit (GPU)

Figure 1 shows the hardware design of the CPU and GPU processor chips. We see a major difference in the number of cores and threads present on each chip. CPU cores are designed to be general purpose, hence they are able to do complex logical operations such as running an operating system while being able to perform arithmetical operations. The closest equivalent of a CPU core on a GPU is a Streaming Multi-Processor (SM) which has most of its transistors as dedicated Arithmetic Logic Units (ALU), rather than control and cache, as in the case of the CPU [13]. GPUs are designed to render graphics, which involves the simultaneous manipulation of millions of pixels. This requires many parallel algebraic operations, hence the large amount of ALUs. Thus applications which require a large number of parallel arithmetic operations see a performance benefit on the GPU. The GPU has two memory spaces: on-chip memory (shared and constant memory) which is very fast but limited in size and scope; and off-chip memory (global-memory) which is much more plentiful and can be accessed by all SMs as well as the CPU. Global-memory is however about one hundred times slower than on-chip memory and can cause major performance degradation if not used efficiently and correctly.

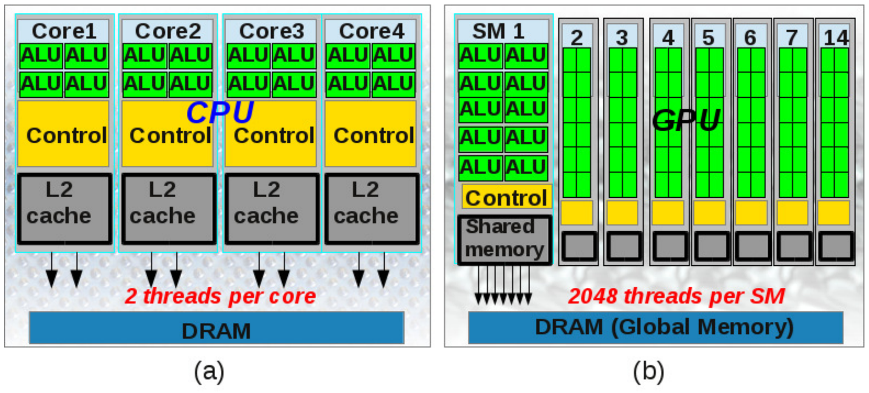


Figure 1: a) Quad core Intel CPU and b) NVIDIA Kepler GPU Chip layouts.

Figure 2 illustrates typical tasks at which each unit excels, in terms of computational performance. GPUs beat the flexible CPUs in spite of their considerably lower clock rate when processing similar data packets. Each CPU core is capable of launching two threads which can work independently and perform complex logical operations. Each SM on a Kepler GK110 GPU can launch 2048 threads which are only capable of performing the same task (Single Instruction Multiple Data (SIMD)) [14]. The NVIDIA Kepler architecture has a lower clock speed than the previous generation Fermi architecture but has more SMs, which benefits applications that have thread level parallelism such as DEM, provided we make certain assumptions to decouple the problem [15, 16].

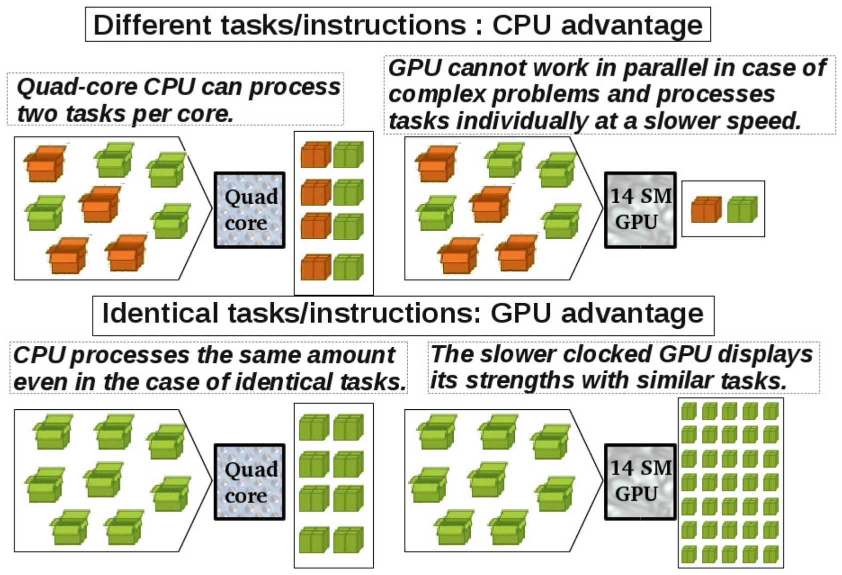

Different tasks/instructions : CPU advantage
Quad-core CPU can process two tasks per core.
GPU cannot work in parallel in case of complex problems and processes tasks individually at a slower speed.
Quad core
14 SM GPU
Identical tasks/instructions: GPU advantage
CPU processes the same amount even in the case of identical tasks.
The slower clocked GPU displays its strengths with similar tasks.
Quad core
14 SM GPU

Figure 2: Comparison between an i7 Quad-core CPU and NVIDIA GK110 GPU task processing.

In this paper we exploit the computational power of the GPU via the NVIDIA developed CUDA programming model [14], which allows us to issue commands to the GPU from C++ code as opposed to a graphics language like OpenGL. CUDA allows us to create thousands of thread blocks containing millions of threads which get scheduled for execution on the hardware as SMs become available (we don't have control over the execution order of the blocks). The execution of a block only completes once all the threads within the block have reached an end point. This is very important and requires us to design algorithms that require similar times to complete for all threads to best utilize the parallelism on the GPU.

3. Polyhedra Contact Detection

Contact detection in DEM usually consists of two phases. The first phase, referred to as the broad phase, is computationally cheap and aims to reduce the number of contact pairs to only the nearest neighbors, as only they can be in possible physical contact. A review of various broad phase algorithms is given by Jimenez and Segura [17]. We use a highly efficient hashed grid approach that executes in parallel on the GPU, that is part of the code BLAZE-DEM developed by the authors [16]. In the second phase, referred to as the narrow phase, pairs of contacting particles obtained from the first phase are considered in detail to determine if they are indeed in contact. Depending on the contact resolution algorithm either contact points [9], or a contact volume [18] can be computed to determine the contact forces.

Several approaches for narrow phase polyhedral collision detection can be found in literature [9, 18, 19]. However, these algorithms require either complex geometrical or algebraic operations with some requiring iterative procedures, as discussed by Nassauer and Liedke [18]. In order to achieve performance gains on the GPU we must formulate an algorithm that is well suited to the light-weight threading model of the GPU (Section 2) with minimal storage requirements.

3.1. Particle Representation

By restricting our analysis to only convex particles we can represent a polyhedron as a collection of half-spaces $f_i(\mathbf{n}, \mathbf{c})$, as illustrated in Figure 3. The half-space subtended by a face of a convex polyhedron partitions $\mathbb{R}^3$ space into two distinct regions. The first region $f_i(\mathbf{n}, \mathbf{c}) \leq 0$ contains the entire polyhedron, while the second region $f_i(\mathbf{n}, \mathbf{c}) > 0$ is an infinite half-space in the direction indicated by the normal to the face [20].

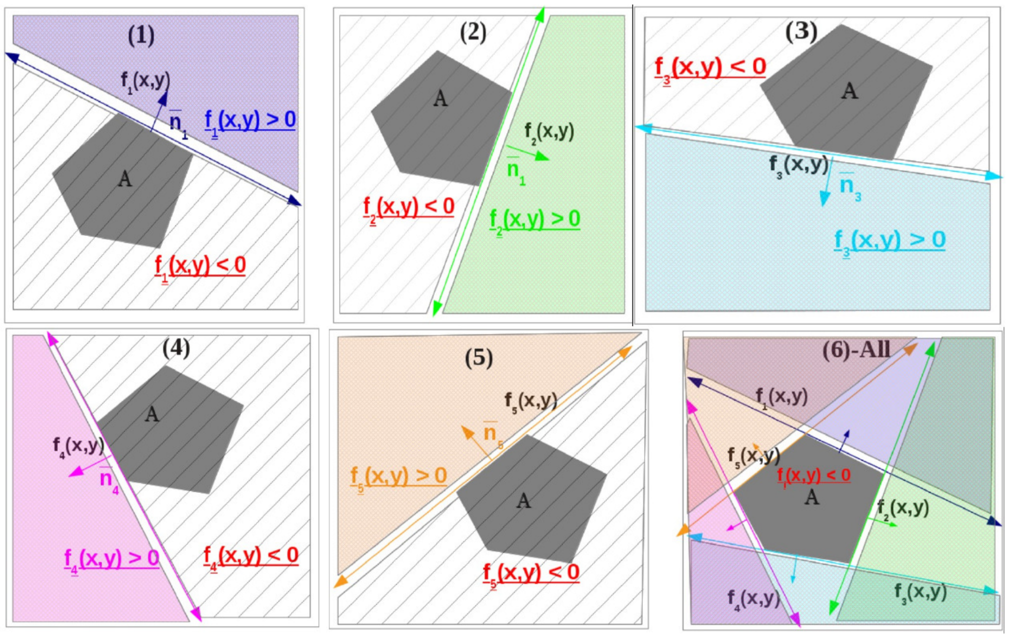
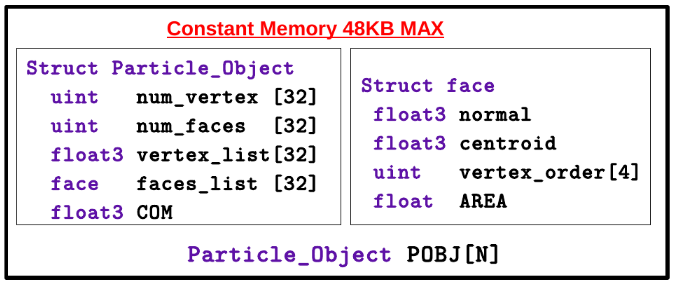


Figure 3: Planar polygon representation.

The choice of faces to define a polyhedron means that we only need to store the vertices $\mathbf{v_i}$, i $= 1, ..., $n, face normals $\mathbf{n_j}$, j $= 1, ...$m and face centroids $\mathbf{c_j}$, j $= 1, ...$m . This minimalistic representation allows us to place data structures in the faster limited constant memory (48 KB) on the GPU, as depicted in Figure 4. We hard code the maximum number of features to 32 due to the small size of constant memory.



Constant Memory 48KB MAX

```
Struct Particle_Object
  uint    num_vertex  [32]
  uint    num_faces   [32]
  float3  vertex_list [32]
  face    faces_list  [32]
  float3  COM
```

```
Struct face
  float3  normal
  float3  centroid
  uint    vertex_order[4]
  float   AREA
```

```
Particle_Object POBJ[N]
```

Figure 4: Particle object representation.

3.2. World Representation

A common approach in DEM simulations is to model the surfaces of the environment as actual particles [21]. This increases computational storage requirements and reduces the task level parallelism that can be exploited on the GPU. We adopt a similar approach as used in the gaming industry [16], in that we model the environment as separate geometric objects (world). World geometry is modeled as either a collection of planar quadrilaterals of the form $\mathrm{S}(\mathbf{n}, \mathbf{c})$, where $\mathbf{n}$ is the normal and $\mathbf{c}$ the centroid of the surface, or geometric objects that have a closed form expression such as a cylinder which we denote as "macro objects". A world can consist of a single object, as illustrated in Figure 5(a), or a combination of objects, as illustrated Figure 5(b), which shows a drum (labeled 1) and blades (labeled 2 and 3).

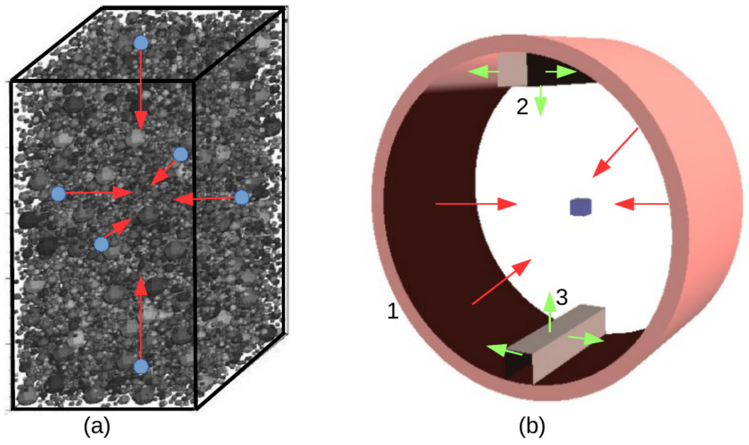


Figure 5: World Object Types.

We make a further distinction in that we split world objects in two classes:

1. Internal: all surface normals point inwards (Figure 5(a) and label 1 in Figure 5(b)).
2. External: surface normals can have any orientation (labels 2 and 3 in Figure 5(b)) .

This distinction allows us to reduce the computational cost of collision detection between a polyhedron and the world object. Consider Figure 6, which shows the different types of contact between polyhedra. Type 1 is the quickest to solve and the most frequent, while Type 2 is more complex but far less frequent in typical GM simulations (see for example Figure 33). For an internal object it is only necessary to check for Type 1 contact (vertex-face or face-face), as it is impossible to have edge-edge contact. Therefore we decompose non-convex boundaries into multiple convex boundaries where possible as shown in Figure 5(b), in which we model the blades as external objects and the drum as a cylinder, which is an internal object.

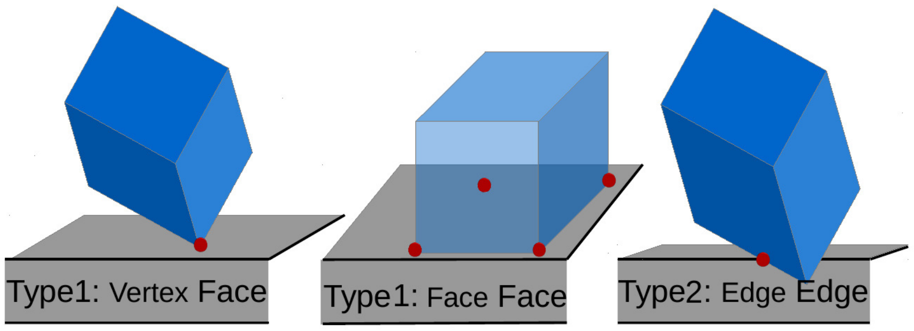


Figure 6: Polyhedra surface contact types.

3.3. Theoretical Formulation

The three scenarios depicted in Figure 7, represent all possible geometric configurations of a polyhedron $\mathcal{A}$ and a planar surface P. We represent the planar surface P by the half-space $S(\mathbf{n}, \mathbf{c}) > 0$. The distance between a vertex $\mathbf{v}$ and the planar surface P is given by:

$$d = \mathbf{n} \cdot (\mathbf{v} - \mathbf{c}) \tag{1}$$

where :

- $d > 0$ implies that the point is within the half-space $S(\mathbf{n}, \mathbf{c}) < 0$.
- $d = 0$ implies that the point is on the hyperplane boundary of the half-space.
- $d < 0$ implies that the point is penetrating the half-space $S(\mathbf{n}, \mathbf{c}) > 0$.

Figure 7(a) represents a scenario where there is clearly no penetration between $\mathcal{A}$ and S. For this case, we have $d_i > 0\,\text{for}\, i = 1, 2, ..., 6$. Figure 7(b) depicts part of $\mathcal{A}$ penetrating S, for which case we have $d_i < 0\,\text{for}\, i = 1, 2, 3$ and $d_i > 0\,\text{for}\, i = 4, 5, 6 > 0$. Figure 7(c) depicts $\mathcal{A}$ being completely past S, for which case we have $d_i < 0\,\text{for}\, i = 1, 2, ..., 6$. We will only encounter scenarios (a) and (b) where the center of mass (COM) point is always in the half-space of the plane, thus $d_i < 0,\text{for any}\, i \in (1, 2, ..., n)$ is indicative that contact has taken place. We summarize this as follows:

Theorem 1: If the $\perp$ distance $d_i > 0\,\text{for}\, i = 1, 2, ..., n$ (Equation (1)) between all vertexes $\mathbf{v}_i\,\text{for}\, i = 1, 2, ..., n$ of a polyhedron $\mathcal{A}$ and $S(\mathbf{n}, \mathbf{c})$, then there is no contact between $\mathcal{A}$ and S.

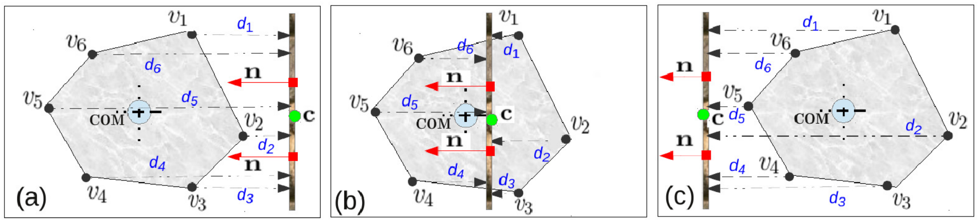


Figure 7: Illustration of Theorem 1 in 2D.

When applied to two polyhedra Theorem 1 is termed the separating plane (SP) method which was first described by Cundall [22]. The SP method reduces the expensive object-to-object contact detection problem to a less expensive plane-to-object contact problem. The aim is to find a plane between the two polyhedra that satisfies Theorem 1. If such a plane is found the two objects cannot be in contact. Many algorithms are based on this approach [9, 22]. However, they involve finding a SP by an iterative procedure, often minimizing a distance function [9], which is not suitable to the SIMD nature of the GPU. Furthermore, to minimize memory transactions our algorithms do not require a separate stage for finding points, an approach that is typical in numerous implementations [9, 18, 19]. It is computationally more efficient to do additional arithmetic operations rather than additional memory transactions on the GPU.

3.4. Polyhedron-Polyhedron Contact Algorithm

As discussed in Section 3.2, detecting vertex-face contact is computationally cheaper than edge-edge contact. Given two polyhedra $\mathcal{A}$ and $\mathcal{B}$, we just need to apply Equation (1) to the vertex set $\mathbf{v}_j^{\mathcal{B}}$ and half-planes $P_i^{\mathcal{A}}(x, y, z)$ and the vertex set $\mathbf{v}_i^{\mathcal{A}}$ and half-planes $P_j^{\mathcal{B}}(x, y, z)$ to determine if there is Type 1 contact. This is computationally efficient on the GPU, due to to the fact that by definition the half-planes $P_i(x, y, z)$ subtended by the faces, partition space into two distinct regions. Hence we can use the result of Theorem 1 to check if a SP exists. If a SP does not exist, then we check if a vertex has penetrated all half-spaces, which implies that it is contained within the other polyhedron and therefore it is a contacting vertex. If there is no SP or contact vertices then we check for Type 2 contact.

3.4.1. Edge-edge contact detection

Recall that for Type 2 contact we do not search for a SP we rather search for penetration between 2 edges and use the fact that if these edges are not penetrating there must be a SP and hence the polyhedra cannot be in contact. This is much cheaper computationally than searching for a SP between edges. Consider Figure 8 (a) which depicts a typical senario that will be a candiate for edge contact and Figure 8 (b) which depicts edge contact .

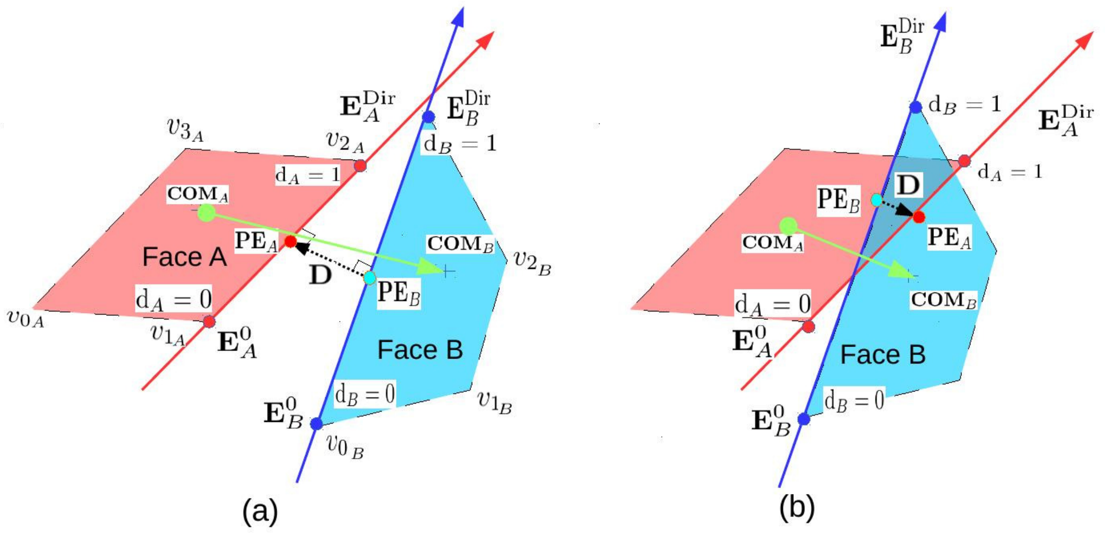


Figure 8: (a) Non penetrating Type 2 contact, (b) Penetrating Type 2 contact.

From Figure 8 (a) we use the parametric equations to represent the edges:

$$\mathbf{PE}_A = \mathbf{E}_A^0 + \mathrm{d}_A \mathbf{E}_A^{\mathrm{Dir}} \tag{2}$$

$$\mathbf{PE}_B = \mathbf{E}_B^0 + \mathrm{d}_B \mathbf{E}_B^{\mathrm{Dir}} \tag{3}$$

where $\mathbf{E}_{\mathrm{K}}^0$ is a vertex $\mathbf{v}_i$ on the edge , $\mathbf{E}_{\mathbf{K}}^{\mathrm{Dir}} = \mathbf{v}_j - \mathbf{v}_i$ is the direction of the edge where $\mathbf{v}_j$ is the other vertex on the edge. d_A and d_B are the scalar parameters for which we solve. We find d_A and d_B that yields the shortest distance between the two edges using the fact that the shortest distance between two edges is a vector $\mathbf{D} = \mathbf{PE}_A - \mathbf{PE}_B$ that is perpinducalar to both edges as indicated in Figure 8 (a).

$$\mathrm{d}_A = \mathrm{detJinv} * [(\mathbf{E}_B^{\mathrm{Dir}} \cdot \mathbf{E}_B^{\mathrm{Dir}}) * (\mathbf{E}_A^{\mathrm{Dir}} \cdot (\mathbf{E}_A^0 - \mathbf{E}_B^0)) - (\mathbf{E}_A^{\mathrm{Dir}} \cdot \mathbf{E}_B^{\mathrm{Dir}}) * (\mathbf{E}_B^{\mathrm{Dir}} \cdot (\mathbf{E}_A^0 - \mathbf{E}_B^0)] \tag{4}$$

$$\mathrm{d}_B = \mathrm{detJinv} * [(\mathbf{E}_A^{\mathrm{Dir}} \cdot \mathbf{E}_B^{\mathrm{Dir}}) * (\mathbf{E}_A^{\mathrm{Dir}} \cdot (\mathbf{E}_A^0 - \mathbf{E}_B^0)) - (\mathbf{E}_A^{\mathrm{Dir}} \cdot \mathbf{E}_A^{\mathrm{Dir}}) * (\mathbf{E}_B^{\mathrm{Dir}} \cdot (\mathbf{E}_A^0 - \mathbf{E}_B^0))] \tag{5}$$

where $\mathrm{detJinv} = 1/[(\mathbf{E}_A^{\mathrm{Dir}} \cdot \mathbf{E}_B^{\mathrm{Dir}}) * (\mathbf{E}_A^{\mathrm{Dir}} * (\mathbf{E}_A^0 \cdot \mathbf{E}_B^0)) - (\mathbf{E}_A^{\mathrm{Dir}} \cdot \mathbf{E}_A^{\mathrm{Dir}}) * (\mathbf{E}_B^{\mathrm{Dir}} \cdot \mathbf{E}_B^{\mathrm{Dir}})]$. Figure 9 describes our algorithm for polyhedra contact detection.

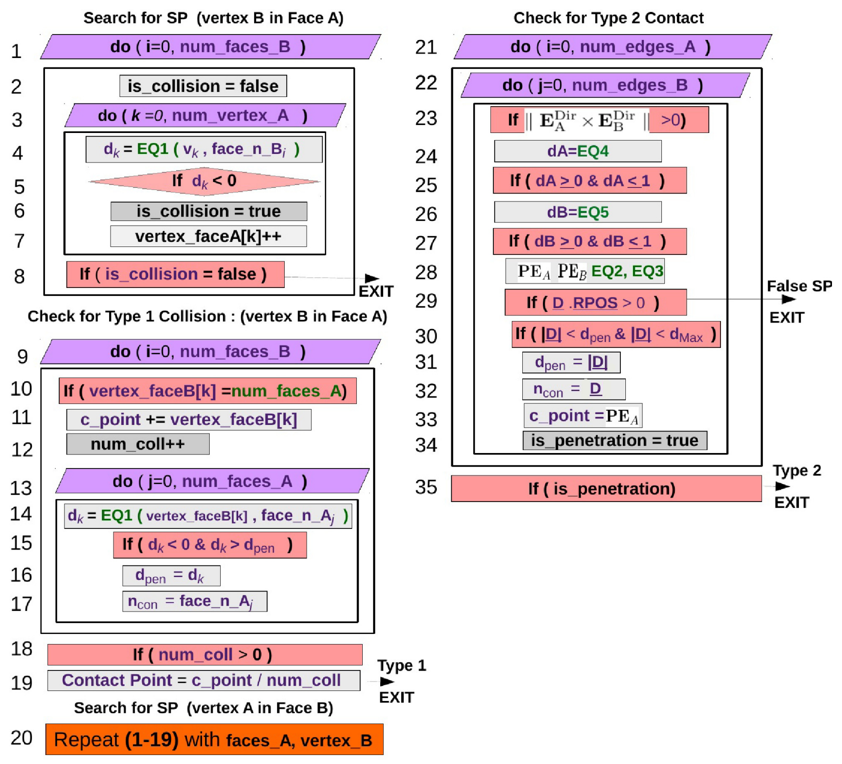

Search for SP (vertex B in Face A)
1 do (i=0, num_faces_B)
2 is_collision = false
3 do (k =0, num_vertex_A)
4 d_k = EQ1 (v_k , face_n_B_i)
5 If d_k < 0
6 is_collision = true
7 vertex_faceA[k]++
8 If (is_collision = false)
EXIT
Check for Type 1 Collision : (vertex B in Face A)
9 do (i=0, num_faces_B)
10 If (vertex_faceB[k] =num_faces_A)
11 c_point += vertex_faceB[k]
12 num_coll++
13 do (j=0, num_faces_A)
14 d_k = EQ1 (vertex_faceB[k] , face_n_A_j)
15 If (d_k < 0 & d_k > d_pen)
16 d_pen = d_k
17 n_con = face_n_A_j
18 If (num_coll > 0)
19 Contact Point = c_point / num_coll
Type 1
EXIT
Search for SP (vertex A in Face B)
20 Repeat (1-19) with faces_A, vertex_B
Check for Type 2 Contact
21 do (i=0, num_edges_A)
22 do (j=0, num_edges_B)
23 If ‖ E_A^Dir × E_B^Dir ‖ >0)
24 dA=EQ4
25 If (dA ≥ 0 & dA ≤ 1)
26 dB=EQ5
27 If (dB ≥ 0 & dB ≤ 1)
28 PE_A PE_B EQ2, EQ3
29 If (D .RPOS > 0)
False SP
EXIT
30 If (|D| < d_pen & |D| < d_Max)
31 d_pen = |D|
32 n_con = D
33 c_point =PE_A
34 is_penetration = true
35 If (is_penetration)
Type 2
EXIT

Figure 9: Algorithm 1: Polyhedron-Polyhedron contact detection.

Line 1 is a loop over all faces of polyhedron $\mathcal{B}$ to search for a SP. Line 2 is a flag, used to exit the loop once we find a SP. Line 3 is a loop over the vertices of polyhedron $\mathcal{B}$ and line 4 is the application of Equation (1) to each vertex. Line 6 sets the flag to false if there is no SP, while Line 7 increments a counter that stores how many faces a vertex has intersected. Line 8 is an exit condition as a SP has been found. Lines 9-19 check for a Type 1 collision by checking if the counter for a vertex equals the number of faces indicating the vertex is inside polyhedron $\mathcal{B}$ and therefore a contact point. Line 18 is an exit condition if contact points have been found. If there is no SP or contact points then Lines 1-19 are repeated with the faces of polyhedron $\mathcal{A}$. If the faces of polyhedron $\mathcal{A}$ does not yield a SP or contact point then we check for Type 2 contact (Lines 21-35). We also sort our data based on spatial location which increases cache hits and improves performance. This is described in the code BLAZE-DEM [16] developed by the authors.

3.5. World Polyhedron Contact

Equation (1) treats a surface as an infinite plane. We are however interested in finite surfaces, as illustrated in Figure 10 (a). We thus need to check that a vertex that is reported to be penetrating the plane is actually contained on the surface of the polygon described by that plane. We do this by tessellating the surface into triangles using the vectors L_i, $i = 1, 2, ..., m$ as described in Figure 10 (b). The area A_i, $i = 1, 2, ..., m$ for each triangle is calculated using the norm of the cross-product of two vectors describing the edges of the triangle. If the vertex is indeed contained within the finite surface, then $\Sigma_i^M A_i$ will equal the total area of the surface.

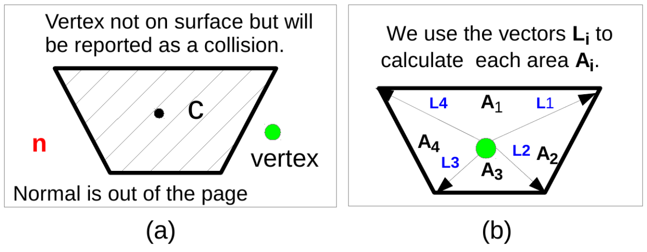
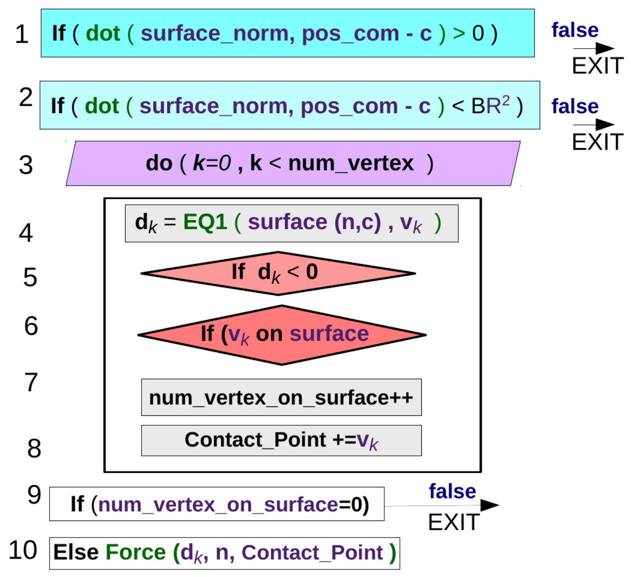

Vertex not on surface but will be reported as a collision.
C
n
vertex
Normal is out of the page
(a)
We use the vectors $\mathbf{L_i}$ to calculate each area $\mathbf{A_i}$.
L4
A_1
L1
A_4
L3
A_3
L2
A_2
(b)

Figure 10: Check for surface collisions.

We summarize this result as:

Theorem 2: If the $\perp$ distance (d_i, $i = 1, 2, ..., k$)> 0 (Equation (1)) between any vertex ($\mathbf{v}_i$, $i = 1, 2, ..., k$) of a polyhedral object $\mathcal{A}$ and half-space $S(\mathbf{n}, \mathbf{c})< 0$ then there is contact between $\mathcal{A}$ and S, provided the vertex is contained within the surface.

Typical world geometry consists of many surfaces. Thus, our algorithm must provide a quick rejection of surfaces with which the particle cannot be in contact. In addition, the algorithm must detect and identify contact in an efficient manner so that we can reduce thread divergence which has a major impact on parallel performance. Figure 11 describes our contact detection algorithm for polyhedron-planar surface contact.



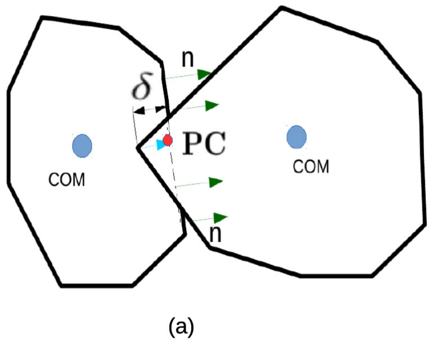
```
1   If ( dot ( surface_norm, pos_com - c ) > 0 )          false -> EXIT
2   If ( dot ( surface_norm, pos_com - c ) < BR^2 )       false -> EXIT
3   do ( k=0 , k < num_vertex )
4       d_k = EQ1 ( surface (n,c) , v_k )
5       If d_k < 0
6       If (v_k on surface
7       num_vertex_on_surface++
8       Contact_Point +=v_k
9   If (num_vertex_on_surface=0)                          false -> EXIT
10  Else Force (d_k, n, Contact_Point )
```

Figure 11: Algorithm 2: Polyhedra-World.

Lines 1-2 are heuristics. Line 1 checks if the COM point is on the correct side of the surface. Line 2 uses the radius from the COM point to the vertex furthest away to check if there is an intersection, since this sphere contains the entire polyhedron. If this sphere does not intersect with the surface, then neither does the polyhedron. These heuristics are computationally cheap and can be executed in parallel, thus quickly eliminating surfaces with which the particle cannot be colliding with . If a particle passes the heuristics, then we apply Equation (1) to all vertices of the particle (Line 4). If a vertex satisfies Theorem 2 then we increment the counter that we use to classify the collision (Line 5). The contact point **PC** is considered to be the average of the vertices which are in contact with the surface $\mathbf{PC} = (\sum_{k=0}^{m} \mathbf{v_k})/m$, where m is the number of contact points. The penetration distance $\mathrm{d_{pen}}$ is the averaged of d_i for all penetrating vertices.

4. Contact Resolution

The total force experienced by a particle during contact is given by $\mathbf{F} = \mathbf{F}_\mathrm{N} + \mathbf{F}_\mathrm{T}$, where $\mathbf{F}_\mathrm{N}$ and $\mathbf{F}_\mathrm{T}$ represent the normal and tangential forces respectively. Consider Figure 12(a) which depicts two contacting particles. We use a penalty approach in that the normal force has a dependance on the amount of interpenetration between 2 particles. Figure 12(b) depicts the the senario we simulate in Section 4.1. Note: the cube is stationary with the octahedron given an inital velocity towards the cube which we denote "downwards".



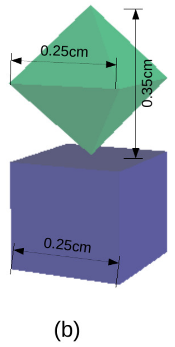


Figure 12: (a) Polyhedral particle contact model and (b) Simulation senario for Section 4.

4.1. Normal Force

4.1.1. Restorative Force

The normal force consists of an elastic $\mathbf{F}_\mathrm{N}^\mathrm{elastic}$ and dissipative part $\mathbf{F}_\mathrm{N}^\mathrm{diss}$. The elastic contribution is represented using a non-linear spring that acts to move particles out of contact.

$$\mathbf{F}_\mathrm{N}^\mathrm{elastic} = (\mathrm{K_r}\delta^{\frac{3}{2}})\mathbf{n} \tag{6}$$

where $\delta = d_{pen}$ is the penetration depth and **n** the contact normal. The spring stiffness $\mathrm{K_r}$ can be chosen using the material properties as defined by Bell et.al [15] or manually tuned to yield the desired response. For this study we manually tune parameters as we are not simulating a real material and only wish to verify our contact algorithms. (Note: we use cm as the unit of length). The bounding radius for the simulations depicted in Figures (13-24) is 0.25cm. The maximum allowed penetration depth is 5% of this which is 0.0125cm.

Figure 13 shows how the penetration depth varies with different incident velocities for a particle undergoing a head-on elastic collision. We see a parabolic curve, which becomes sharper as velocity increases which is the expected result of Equation (6). This information can be used to tune $\mathrm{K_r}$ for a range of velocities that is observed for a typical GM simulation based on experimental or simulation data, so that the desired maximum penetration depth, or a bound on the contact time can be obtained.

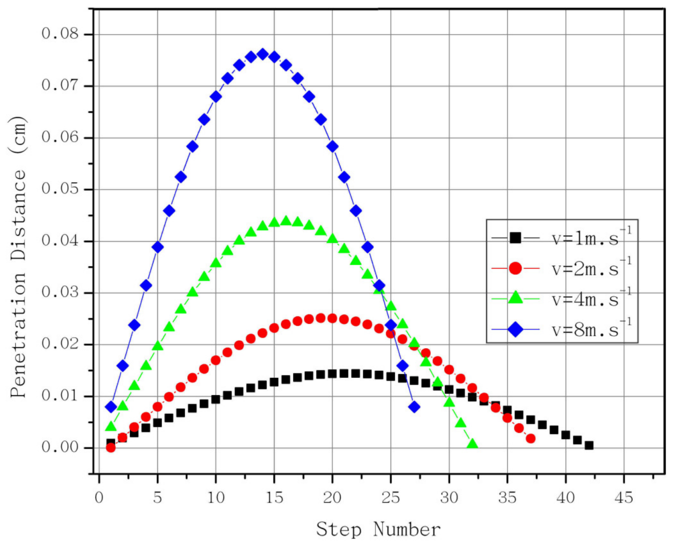


Figure 13: Penetration distance as a function of time step for a head-on elastic collision.($K_r = 1 \times 10^9$, step size $=10^{-5}$)

Figure 14 shows how $\mathbf{F}_N^{\text{elastic}}$ varies as a function of the penetration depth for an incident velocity of $\mathbf{v} = 2\text{m.s}^{-1}$. We see a non-linear dependance on penetration depth for both loading and unloading as expected. We also see that the curves for both stages are identical which is further indication that our algorithms are implemented correctly..

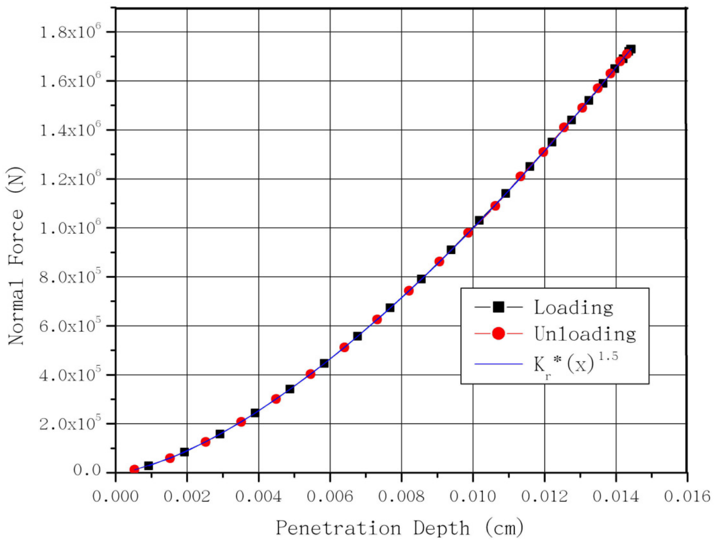


Figure 14: $\mathbf{F}_N^{\text{elastic}}$ as function of penetration depth for a head-on elastic collision.($K_r = 1 \times 10^9$, step size $=10^{-5}$)

An important test of our contact detection and resolution algorithms is that for a head-on elastic collision kinetic energy must be conserved. Figure 15 shows how the velocity varies during contact. We see that for all incident velocities the pre and post collision velocities are the same.

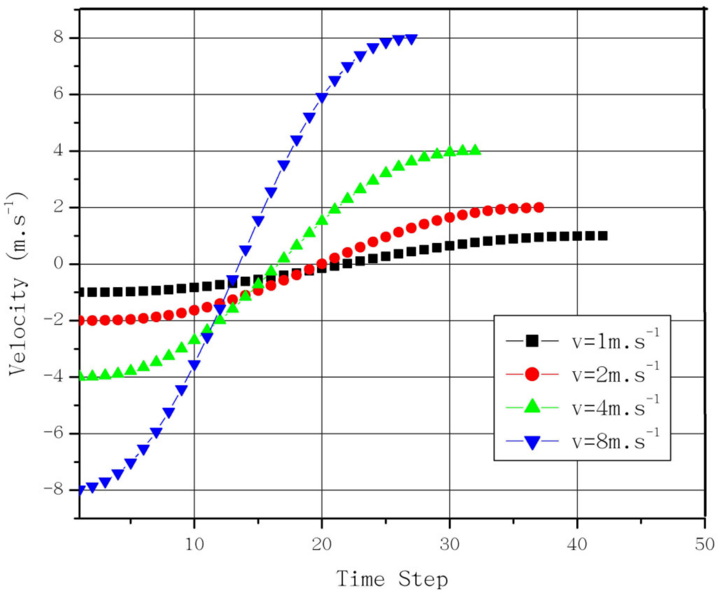


Figure 15: Velocity as a function of time step for a head-on elastic collision.($K_r = 1 \times 10^9$, step size $=10^{-5}$)

Table 1 gives us a quantitative estimate on the accuracy of our code by analyzing the pre and post collision velocities which must be conserved. Here, $\mathbf{V}$ indicates the expected result and $\mathbf{V}'$ the numerically computed result.We see excellent agreement with a maximum difference of 0.33%.

Table 1: Analysis of particle velocities for a head-on elastic collision.

$\mathbf{V}$(m.s^{-1})	$\mathbf{V}'$(m.s^{-1})	**% Diff**
0.062500	0.062582	-0.1312
0.125000	0.125249	-0.1992
0.250000	0.250152	-0.0608
0.500000	0.501799	-0.3595
1.000000	0.996799	0.32010
2.000000	1.997261	0.13695
4.000000	4.013109	-0.32772

4.1.2. Dissipative Force

Dissipation during contact is given by:

$$\mathbf{F}_{\mathrm{N}}^{\mathrm{diss}} = -\mathrm{K_D}\delta^{\gamma}\mathbf{vrel}_{\mathrm{n}}, \tag{7}$$

where $\mathrm{K_D}$ is the damping coefficient and $\mathbf{vr}\mathrm{el_n} = ((\mathbf{v}_1 - \mathbf{v}_2) \cdot \mathbf{n})\mathbf{n}$ is the relative normal translational velocity. $\gamma = \frac{1}{2}$ was found to match experimental data the best by Bell et.al [15], as the energy dissipation increases with increasing impact velocity. Figure 16 shows the force hysteresis for $\mathbf{F}_{\mathrm{N}}$. We see that adding energy dissipation results in an elliptical shape for the force that is asymmetric. This shape is consistent with that of other authors [18]. We firstly see that at the start of loading (0 penetration distance) the force points upwards opposing the downwards motion of the octahedron, which is a result of the dissipative force as there is no elastic contribution. The dissipative force once again exceeds the elastic force contribution towards the end of the unloading stage resulting in a negative total force that is decelerating the body which is now moving in an upwards direction. This negative force might seem non-physical but none of authors [15, 23] have reported non-physical effects such as the particle being attracted to the surface during contact. Nevertheless we verify the realism of the force model for a particle in free fall as illustrated in Section 4.1.3.

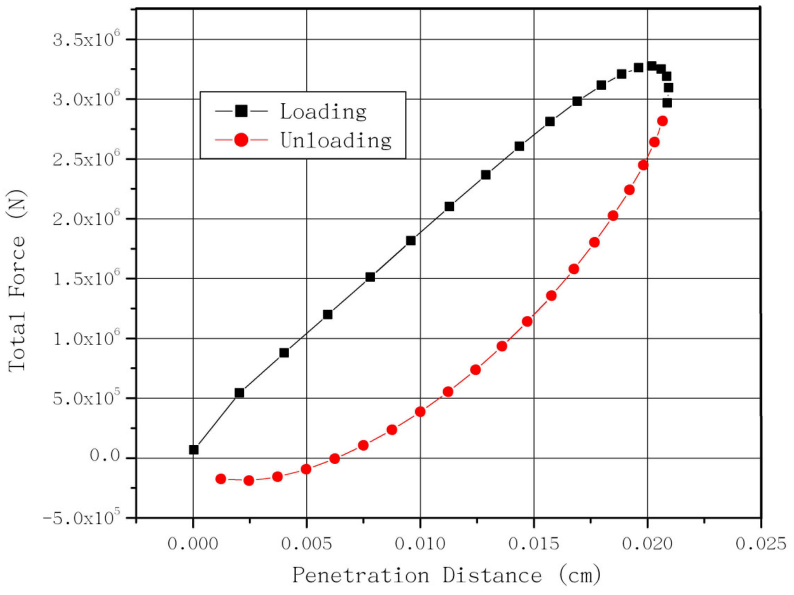


Figure 16: Normal force as a function of penetration depth for a head-on in-elastic collision.($\mathrm{K_r} = 1 \times 10^9$,$\mathrm{K_D} = 5 \times 10^4$, step size $=10^{-5}$,$\mathbf{v} = 2\mathrm{m.s}^{-1}$)

Figure 17 shows the difference between $\| \mathbf{F}_{\mathrm{N}}^{elastic} \|$ and $\| \mathbf{F}_{\mathrm{N}}^{\mathrm{diss}} \|$ as a function of velocity (vertical lines indicate the onset of contact). We firstly see that for high initial velocity collisions that $\mathbf{F}_{\mathrm{N}}^{\mathrm{diss}}$ dominates at the start and end of the collision, resulting in a large dissipation of energy. This is consistent with experimental data [15] and explains the negative total force in Figure 16 at the initial stages of contact. We also see that the contact time increases with decreasing velocity, while there is effectively no damping for small velocity impacts that occur when the particle is in persistent contact with the surface.

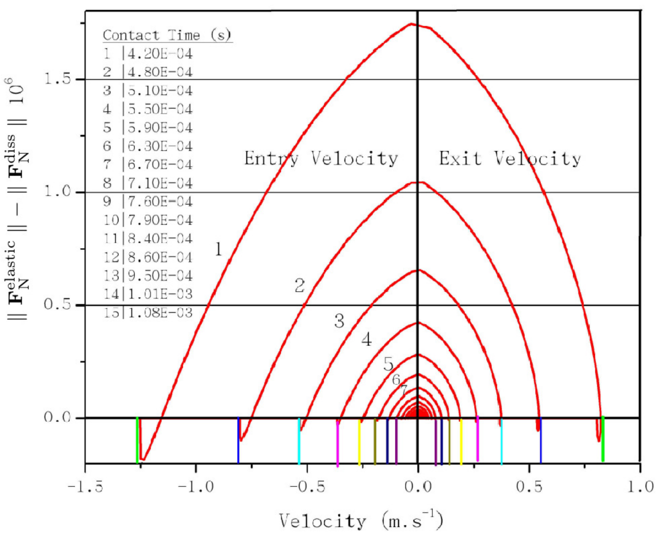


Figure 17: $\| \mathbf{F}_{\mathrm{N}}^{\mathrm{elastic}} \| - \| \mathbf{F}_{\mathrm{N}}^{\mathrm{diss}} \|$ vs velocity ($\mathrm{K_r} = 1 \times 10^9$,$\mathrm{K_D} = 5 \times 10^4$, step size $=10^{-5}$).

4.1.3. Contact Evaluation for particle in free fall.

Figure 18 shows the evolution of the position for a particle dropped from rest undergoing inelastic contact with a surface. We see the particle undergoing a number of contacts with the surface before reaching an equilibrium as expected.

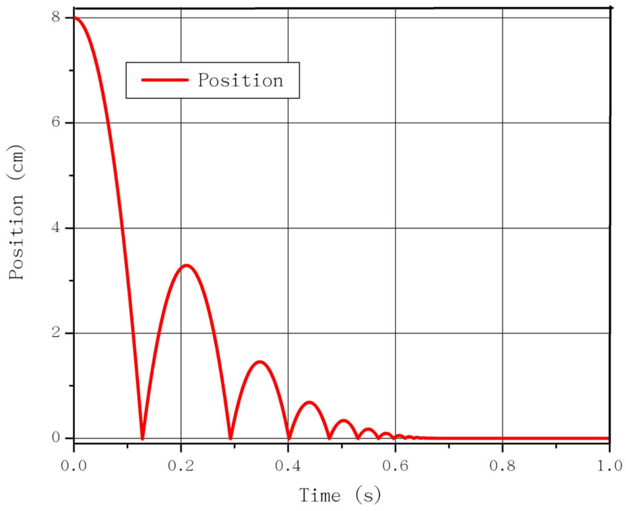


Figure 18: Position vs Time ($\mathrm{K_r} = 1 \times 10^9$,$\mathrm{K_D} = 5 \times 10^4$, step size $=10^{-5}$).

Figure 19 shows the corresponding velocity for Figure 18. We see that the velocity also reaches an equilibrium as expected.

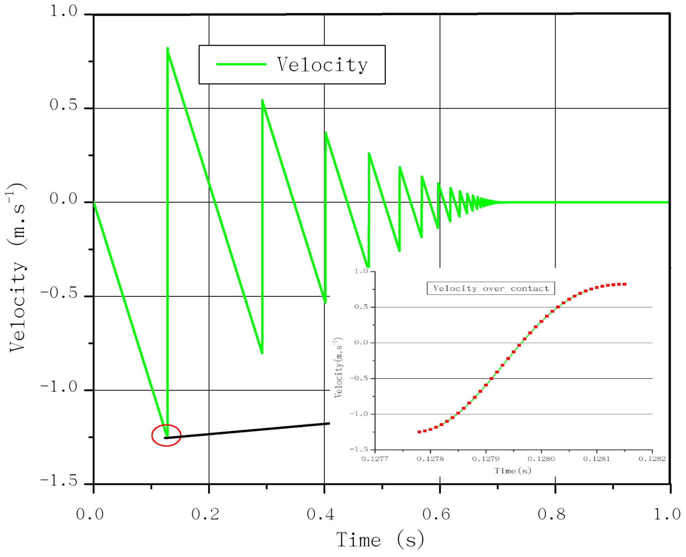


Figure 19: Velocity vs Time ($K_r = 1 \times 10^9$,$K_D = 5 \times 10^4$, step size $=10^{-5}$).

4.2. Tangential Force

The tangential force is given by :

$$\mathbf{F}_T = -\min\left[\mu \parallel \mathbf{F}_N \parallel, K_T \|\mathbf{v}_T\|\right] \left(\frac{\mathbf{v}_T}{\|\mathbf{v}_T\|}\right) \tag{8}$$

where $\mathbf{v}_T = (\mathbf{v}_R - (\mathbf{v}_R.\bar{\mathbf{n}}_s))\bar{\mathbf{n}}_s$ is the relative tangential velocity , μ the coefficient of friction and K_T the viscous damping coefficient [15]. The values for μ and K_T should be such that K_T affects oblique impacts while μ affects rolling/sliding contact. To illustrate this we simulate the motion of the octagon shaped particle with an initial velocity of 2m.s^{-1} in the tangential direction and -0.5m.s^{-1} in the normal direction ("downwards") in a gravity field initially placed 0.75cm above the surface (no rotation). Figure 20 shows the evolution particle position and tangential velocity with time. In Figure 20 (a) we see that the particle undergoes 4 oblique collisions with the surface before the rebound height becomes sufficiently small such that the particle can be considered to be moving on the surface. Figure 20 (b) shows the corresponding tangential velocity which has two distinct patterns. The velocity firstly decreases in steps for $t < 0.055$ which is indicative of viscous damping and thereafter it decreases in a linear fashion which corresponds to a constant frictional force.

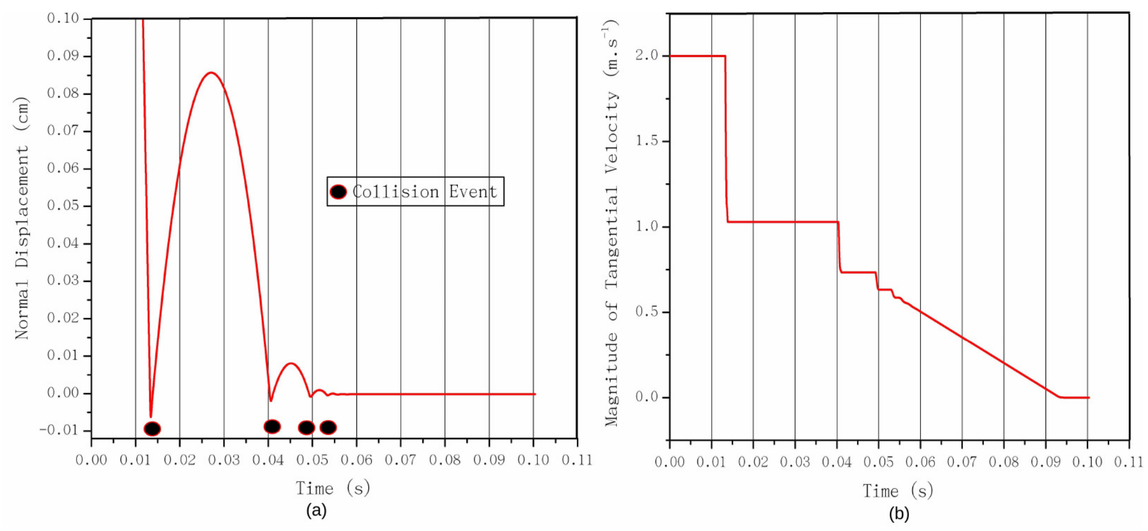


Figure 20: (a) Position vs Time and (b) Velocity vs Time ($K_r = 1 \times 10^9$,$K_D = 2 \times 10^5$,$\mu = 1.54$,$K_T = 4 \times 10^3$, step size $=10^{-5}$).

Figure 21 shows the tangential force as a function of velocity, we see that indeed for high velocity oblique collisions $K_T \|\mathbf{v}_T\|$ dominates while for low velocity collisions $\mu \parallel \mathbf{F}_N \parallel$dominates.

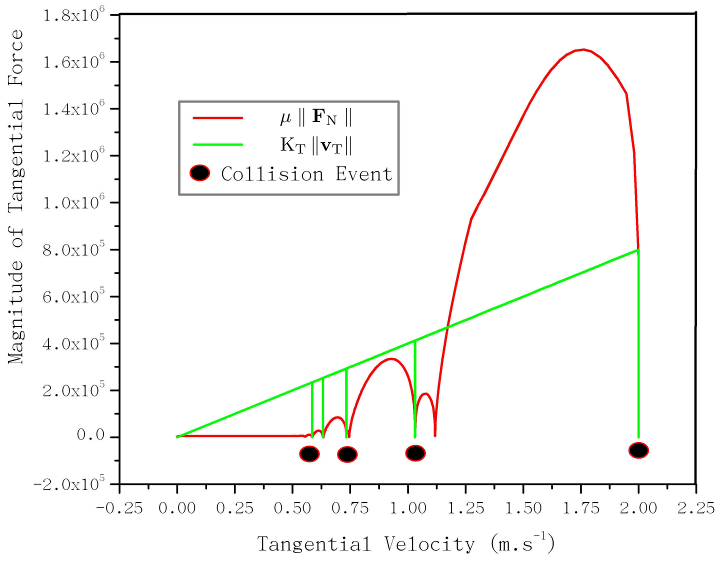


Figure 21: $\parallel \mathbf{F}_T \parallel$vs Tangential Velocity ($K_r = 1 \times 10^9$,$K_D = 2 \times 10^5$,$\mu = 1.54$,$K_T = 4 \times 10^3$, step size $=10^{-5}$).

In addition to translation forces a particle also experiences a torque as a result of contact given by :

$$\mathbf{\Gamma} = (\mathbf{r} \times \mathbf{F}) \tag{9}$$

where $\mathbf{r}$ is the vector from the COM to the contact point $\mathbf{PC}(x, y, z)$.

4.3. Numerical Integration

We use the explicit velocity Verlet algorithm, which is second order accurate, to obtain the position $\mathbf{x}$ and velocity $\mathbf{v}$ of a particle i at time k:

$$\mathbf{x}^{\mathrm{i}}{}_{\mathrm{k}} = \left[\mathbf{x}^{\mathrm{i}}_{\mathrm{k}-1} + \mathbf{v}^{\mathrm{i}}_{\mathrm{k}-1}\triangle \mathrm{t} + \frac{1}{2}\mathbf{a}^{\mathrm{i}}_{\mathrm{k}-1}\triangle \mathrm{t}^2\right] \tag{10}$$

$$\mathbf{v}_{\mathrm{k}} = \left[\mathbf{v}^{\mathrm{i}}_{\mathrm{k}-1} + \frac{1}{2}(\mathbf{a}^{\mathrm{i}}_{\mathrm{k}-1} + \mathbf{a}^{\mathrm{i}}_{\mathrm{k}})\triangle \mathrm{t}\right] \tag{11}$$

The acceleration $\mathbf{a}$ at time k is given by $\mathbf{a}_{\mathrm{k}} = \frac{\mathbf{F}^{\mathbf{net}}_{\mathrm{k}}}{\mathrm{m}}$ where $\mathbf{F}^{\mathbf{net}}_{\mathrm{k}} = \sum_{j=1}^{L} \mathbf{F}^{\mathrm{ij}}_{\mathrm{k}}$ is the sum of all L contact forces experienced by the particle. To ensure that the integration scheme we employ behaves as expected we simulate the motion of a system of particles falling under the influence of gravity for 1 second onto a planar surface. Figure 23 shows the error in the total energy of the system for various time-steps. We firstly see the error remains within a bound for the time-steps sampled and decreases with a smaller time-step as expected. While a time step size of 10^{-5} is satisfactory we will use a time step size of 10^{-6} for our simulation examples as it is typically used in DEM simulations by other authors and will allow for unbiased comparisons.

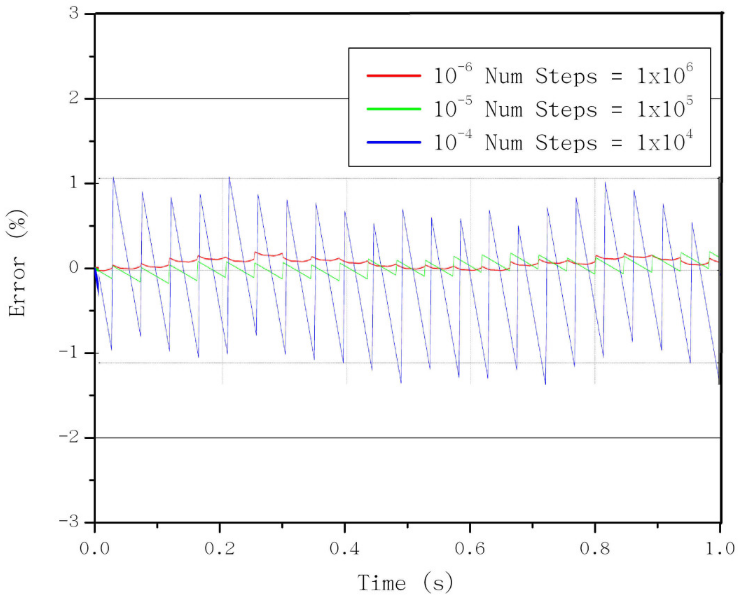


Figure 22: Error in Energy Conservation.

4.3.1. Angular Integration

The angular velocity ω of particle i at time k is obtained using the forward Euler integration scheme.

$$\boldsymbol{\omega}_{\mathrm{k}} = \boldsymbol{\omega}_{\mathrm{k-1}} + \boldsymbol{\alpha}^{\mathrm{ang}}_{\mathrm{k}} \triangle \mathrm{t}. \tag{12}$$

The angular acceleration $\boldsymbol{\alpha}^{\mathrm{ang}}$ at time k is given by $\boldsymbol{\alpha}^{\mathrm{ang}}_{\mathrm{k}} = \mathbf{I}^{-1}_{\mathrm{k}} \boldsymbol{\Gamma}^{\mathbf{net}}_{\mathrm{k}}$ where $\boldsymbol{\Gamma}^{\mathbf{net}}_{\mathrm{k}} = \sum_{j=1}^{L} \boldsymbol{\Gamma}^{\mathrm{ij}}$ is the sum of all L body contact torques experienced by particle i as given in Equation (9) and $\mathbf{I}_{\mathrm{k}}$ the inertia tensor at time k . Quaternions have minimal storage requirements and are thus well suited to the GPU, and they are also more robust than other representations such as Euler angles [24]. The orientation of a particle is represented by a unit quaternion $\mathbf{q}\{\mathrm{w,x,y,z}\} = \{1,0,0,0\}$, where w is an angle $[-1:1]$ and$(\mathrm{x,y,z})$ the axis of rotation. The relationship between a quaternion and axis angle representation (θ, x_1, y_1, z_1) is given by:

$$\mathbf{q} = \{\, \cos(\theta/2),\ x_1 \sin(\theta/2),\ y_1 \sin(\theta/2),\ z_1 \sin(\theta/2)\,\}. \tag{13}$$

Given an angular velocity vector $\boldsymbol{\omega}$ the quaternion representing that rotation is given by:

$$\triangle \mathbf{q} = \{\, \cos(\|\boldsymbol{\omega}_{\mathrm{k}}\|),\ \sin(\|\boldsymbol{\omega}_{\mathrm{k}}\|) \frac{\boldsymbol{\omega}_{\mathrm{k}}}{\|\boldsymbol{\omega}_{\mathrm{k}}\|}\,)\,\}. \tag{14}$$

The evolution of the angular orientation of the particle is just a multiplication [25] between the current quaternion $\mathbf{q}_{\mathrm{k-1}}$ of a particle with $\triangle \mathbf{q}$:

$$\mathbf{q}_{\mathrm{k}} = \mathbf{q}_{\mathrm{k-1}} \times \triangle \mathbf{q}. \tag{15}$$

To verify that rotation using quaternions on the GPU is correct, we simulate a cube spinning with a constant velocity of $(1,0,0)$ rads^{-1} with a step-size of 10^{-5}. Figure 23(a) shows the angular displacement as a function of time. We see that it varies in a linear fashion as expected with the total angle of rotation at the end of $1s = 359.94406$ giving an error of 0.015% which is within error tolerances.

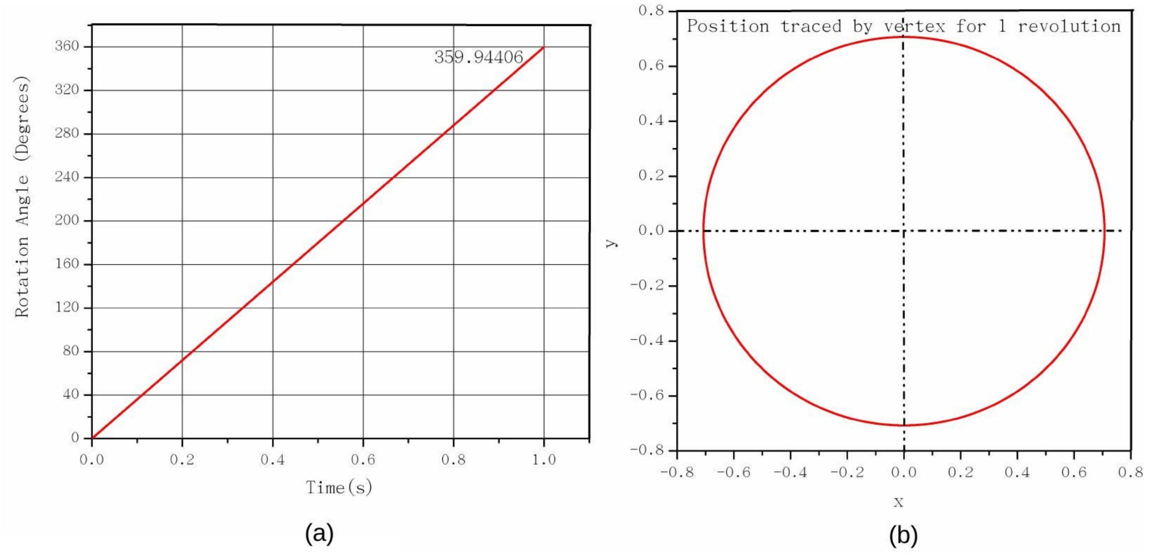


Figure 23: Results of angular integration depicting (a) angular displacement (rotation angle) as a function of time and (b) the position of a vertex on the rotating cube.

Figure 23 (b) shows how the position of a vertex on the cube varies with time which gives an indication if any shearing of the shape occurs. We see a perfect circular path which implies that there is no shearing of shape occurring. Note: in Equation (15) as $\boldsymbol{\omega}_{\mathrm{k}} \rightarrow 0$,$\triangle \mathbf{q} \rightarrow \{1,0,0,0\}$ which leaves the orientation unchanged and hence does not result in numerical difficulties.

5. Numerical Simulation

All simulations are done using a Nvidia Quadro K6000 GPU (30720 physical threads) on an Intel i7 3.5 GHZ Extreme Edition CPU with 32 GB of RAM under OpenSuse Linux 13.1. The values for all parameters are given in Table 2.

Table 2: Parameters used in simulations for non-linear force model.

Parameter	$\triangle t$	$\mathrm{K_r}$	$\mathrm{K_D}$	μ	$\mathrm{K_T}$
Value	10^{-6}	1×10^{8}	5×10^{3}	0.154	4×10^{2}

5.1. Polyhedra in a drum

To verify that our algorithms correctly detects collisions, we model the gravity packing of 1024 tightly packed cubes (edge length 0.50 cm) in a drum with a blade as depicted in Figure 24. Notice that all the cubes are given an initial velocity of $0.2\mathrm{cm.s}^{-1}$ to the right. We see that the particles take the shape of the container with the faces resting on the cylinder and around the obstacle (blade) which is what we expect. This gives as a qualitative indication that our algorithms correctly detect collisions.

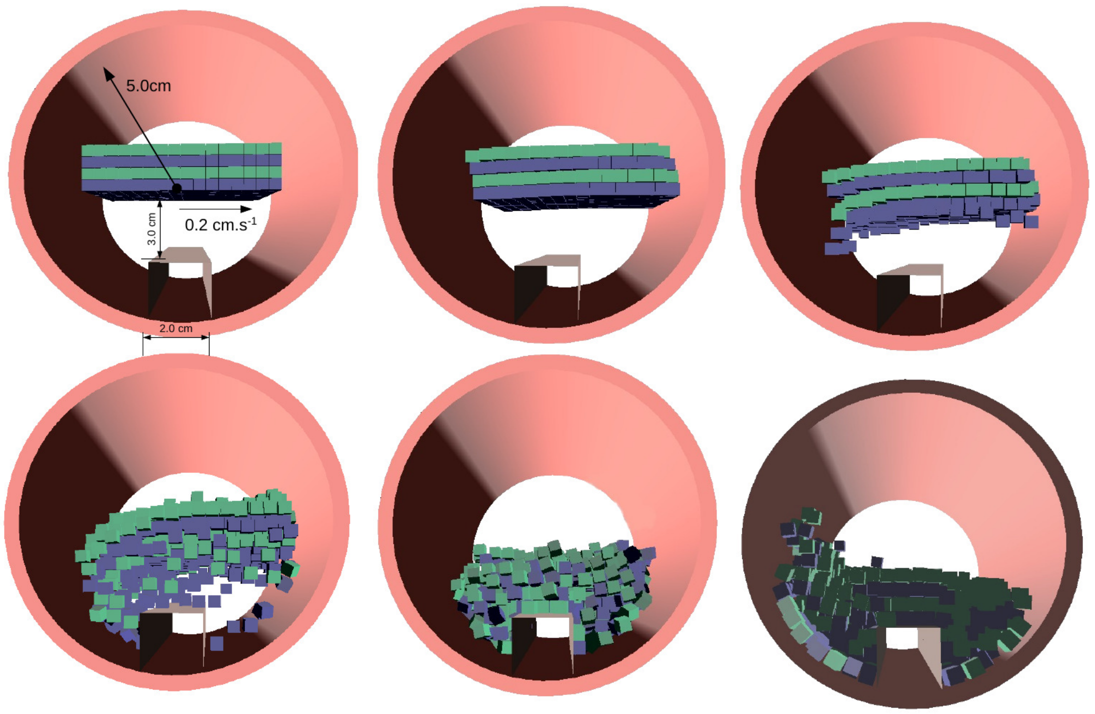


Figure 24: Packing of dynamic cubes in a drum.

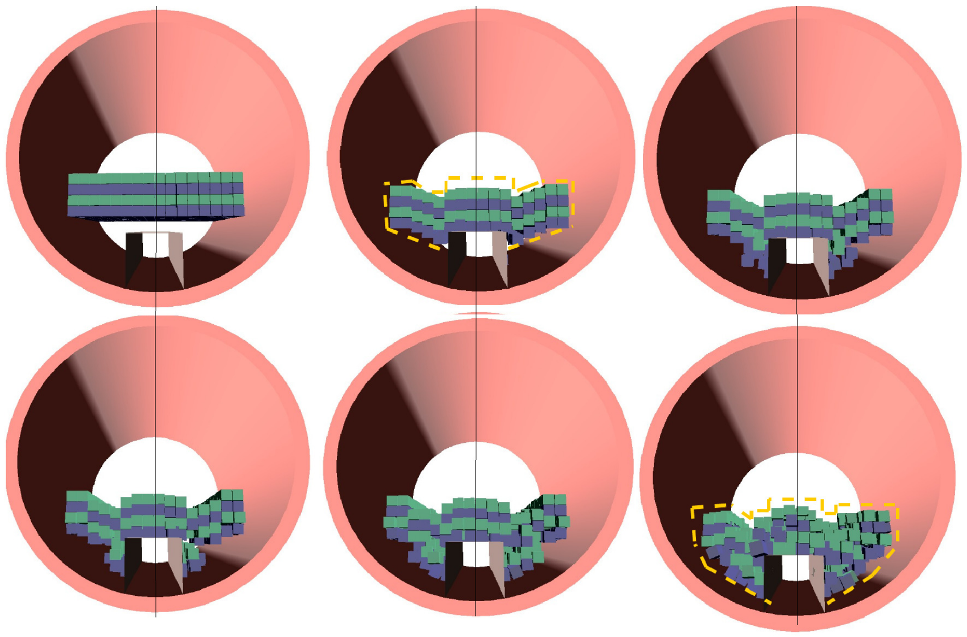


Figure 25: Packing of cubes in a drum (symmetric).

In Figure 25 the cubes are arranged symmetrically in the drum with an initial height of 1cm above the horizontal section of the blade.). The cubes fall under the influence of gravity only (zero initial velocity). We see that the packing is symmetric, which is what we expect, further validating our algorithms.

5.2. Scaling with particle shape

To further evaluate the performance of our algorithms, we simulate the motion of tightly packed identical cubes (edge length 0.50 cm) arranged in a rectangular grid, falling under the influence of gravity with an initial velocity as indicated in Figure 26.

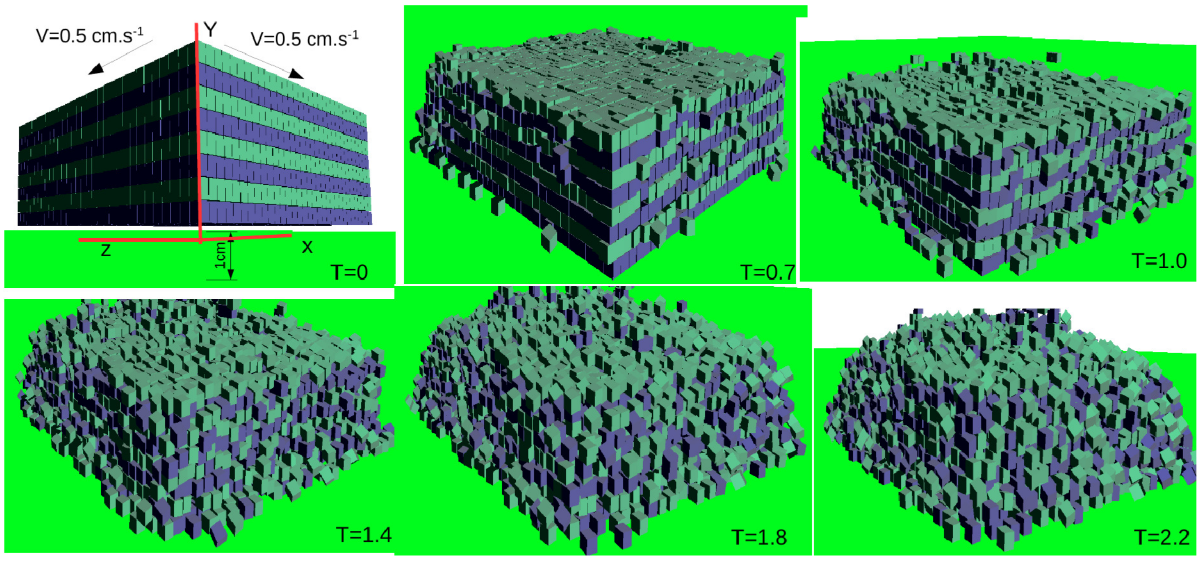


Figure 26: Gravity packing simulation for performance benchmarking.

Figure 27 shows the scaling of our World-Polyhedra algorithm with the number of surfaces in the world. We firstly see that there is a small computational penalty for doubling the number of faces. We also see linear scaling with an increase in world surfaces.

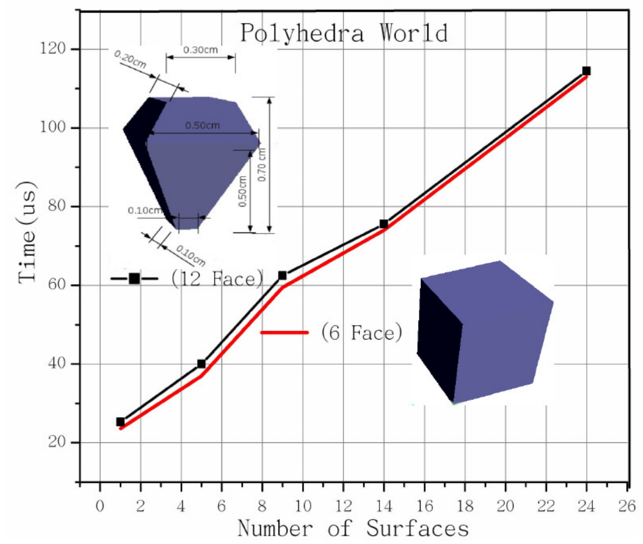


Figure 27: Computational scaling of World-Polyhedron collision detection with world surfaces.

Figure 28 shows the scaling of the World-Polyhedron algorithm with increased particle number. We see that again there is a small computational penalty for doubling the number of faces, which can be attributed to the SIMT of the GPU. The scaling with increasing particle number is once again linear.

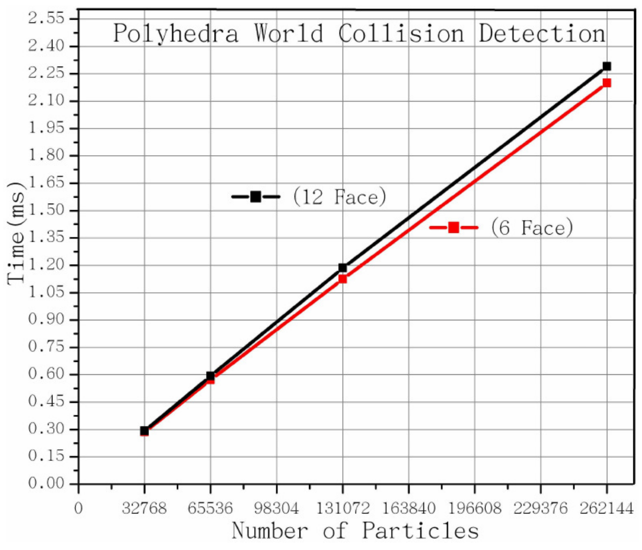


Figure 28: Effect of particle shape (number of faces) on the computational cost of collision detection.

Figure 29 shows the scaling of the computational cost of Polyhedron-Polyhedron contact detection. We observe linear scaling with the number of particle faces and the number of particles.

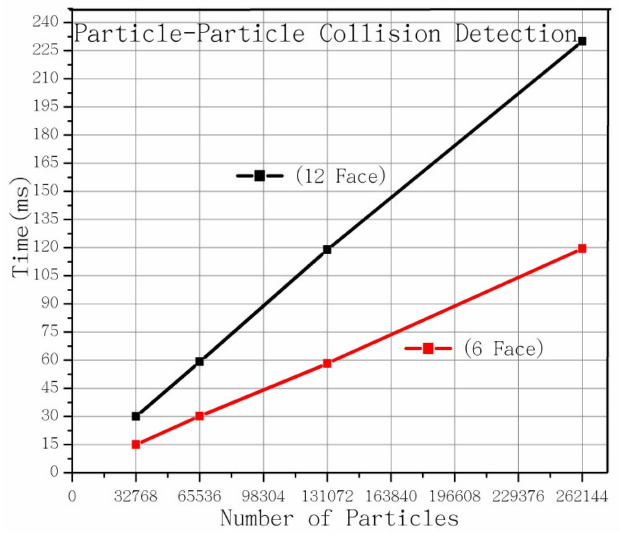
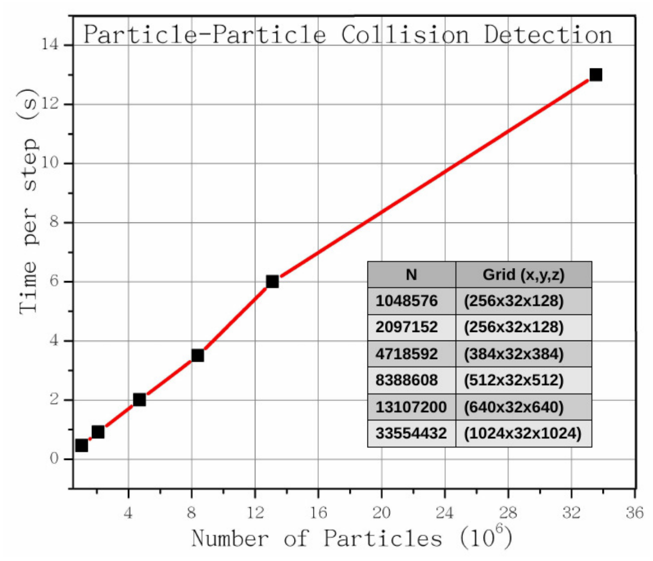


Figure 29: Scaling of Polyhedron-Polyhedron collision detection with number of particles (N) for different polyhedra.

5.3. Algorithm performance

Figures 30-32 illustrate the large-scale performance of our algorithms for the simulation described in Figure 27. We firstly analyze Polyhedron-Polyhedron collision detection, which is computationally the most expensive, in Figure 31. We see that the trend of linear scaling holds up to 34 million particles, which is only limited by the current Nvidia NVCC compiler .



N	Grid (x,y,z)
1048576	(256x32x128)
2097152	(256x32x128)
4718592	(384x32x384)
8388608	(512x32x512)
13107200	(640x32x640)
33554432	(1024x32x1024)

Figure 30: Scaling of Polyhedron-Polyhedron collision detection with number of particles (N) for the six face polyhedron.

The trend of linear scaling continues with the World-Polyhedron algorithm. However, it is significantly faster and takes 0.25 seconds for 34 million particles.

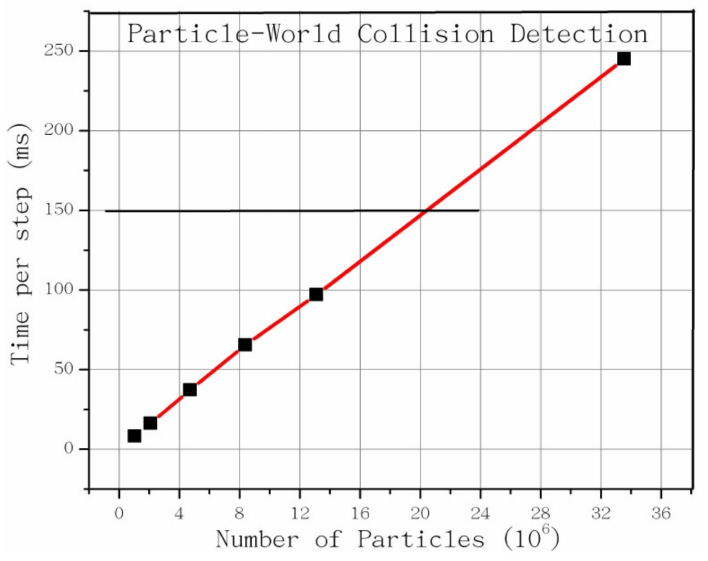


Figure 31: Scaling of Polyhedron-World algorithm with number of particles (N).

The broad-phase search algorithm [16] takes under one second a step for 34 million particles, with linear scaling as well, indicated in Figure 32. Our results set a new performance level in contact detection for polyhedra in DEM.

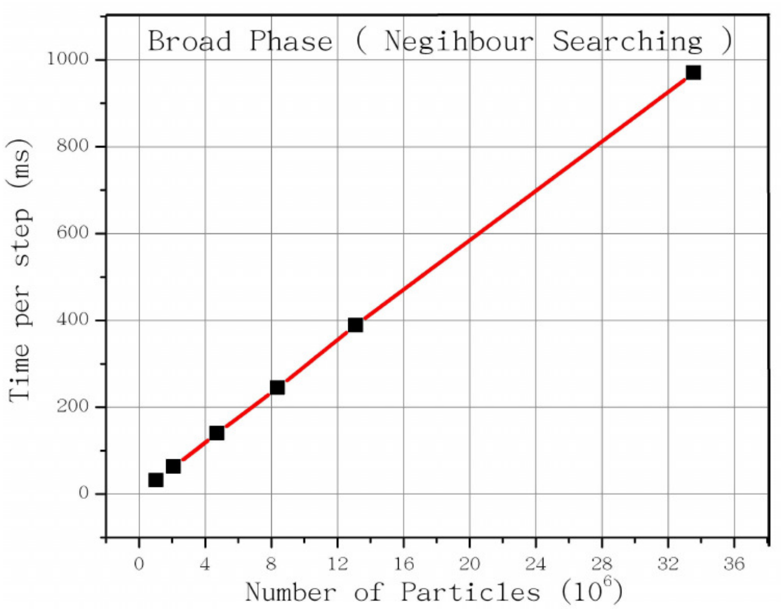


Figure 32: Scaling of Broad-Phase algorithm with number of particles (N).

Table 3 shows the comparison of our code to that of other authors using the Cundall Number $C = \text{N} \times \text{FPS}$, where a a higher number represents better performance. Here FPS is the number of frames/steps that can be performed per second. Note: to the best of the authors knowledge there are no other GPU codes using polyhedral particles. Our code BLAZE-DEM (polyhedra) performs better than the others which use a non-spherical particle representation, while being able to simulate 136 times more particles than the fastest GPU code (Longmore at.al) in Table 3. Although the code by Radake et.al [6] performs the best it uses only spherical particles. Most of the algorithms presented in this paper is the same for spheres and our code BLAZE-DEM[16] using spherical particles is about 3 times faster as indicated in Table 3.

Table 3: Comparison to other GPU codes (mono sized particles) *Kepler class GPU.

Author	**Shape**	**N particles**	**C Number**
Harida et.al[25]	Clumped	1.64×10^4	0.66×10^6
Longmore et.al [23]	Clumped	2.56×10^5	1.49×10^6
This paper	***Poly**	34×10^6	2.62×10^6
Radake et.al[6, 26]	*Sphere	20×10^6	20×10^6
Govender et.al [12, 27]	*Sphere	50×10^6	55×10^6

Figure 33 shows the frequency of the two collision types depicted in Figure 6. We see that indeed vertex-face is the dominant contact type and justifies our choice in searching for it first. In Figure 33 (a) we see that edge-edge contact is at most 10%. We note a similar trend in Figure 33 (b) where a slightly higher percentage of edge-edge contact can be attributed to the geometrical effect of the drum. In Figures 33 (c) edge-edge contact increases to 30% as the initial velocity of particles causes an impact with the drum resulting in a more random orientation. As the particles reach a macroscopic steady state, the edge-edge contact frequency decreases to be similar to the other simulations.

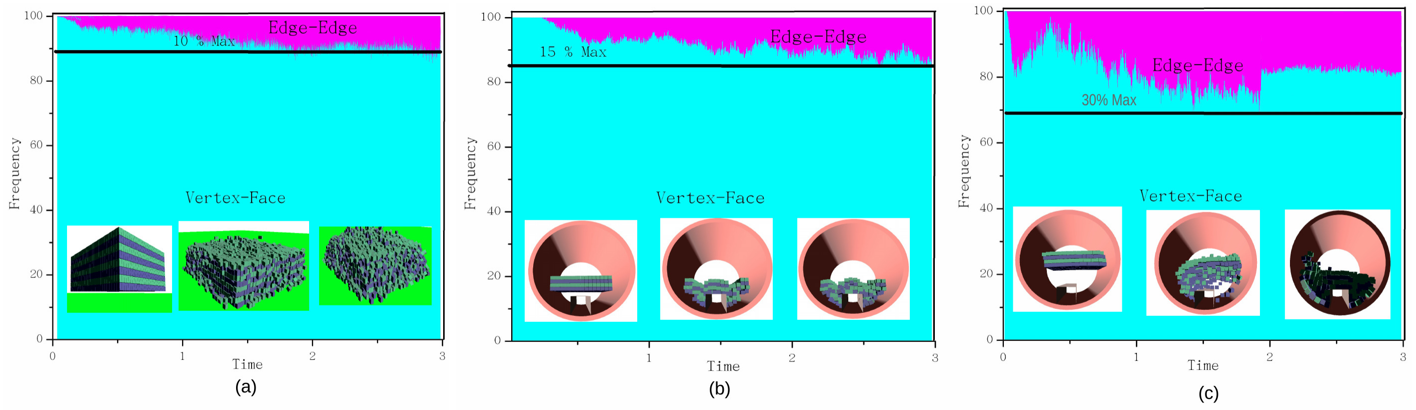


Figure 33: Frequency plot of the collision type against simulation type for (a) gravity packing problem, (b) gravity packing in drum without initial velocity, (c) gravity packing in drum with initial velocity.

6. Conclusion

In this paper we have presented a novel approach for collision detection that is optimized for the GPU architecture. We evaluated the scaling of our algorithm and found favorable results in that the time does not necessarily scale with increased geometrical complexity for the system we have analyzed. We achieve a new performance level in DEM by simulating 34 million polyhedra (13 seconds per time step) on a single Nvidia K6000 GPU.

7. Acknowledgments

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the GPU used for this research.

References

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

[2] P. Cleary, The filling of dragline buckets, Math. Eng. Ind. 29 (1998) 1–24.

[3] B. Mishra, R. Rajamani, Simulation of charge motion in ball mills. part 1: experimental verifications, Int. J. Mineral Process 40 (1994) 171–186.

[4] W. Ketterhagen, J. Curtis, C. Wassgren, Predicting the flow mode from hoppers using the discrete element method, Powder Technology 195 (2009) 1–10.

[5] M. Moakher, T. Shinbrot, F. Muzzio, Experimentally validated computations of flow, mixing and segregation of non-cohesive grains in 3d tumbling blenders, Powder Technology 109 (2000) 58–71.

[6] C. Radeke, B. Glasser, J. Khinast, Large-scale powder mixer simulations using massively parallel gpu architectures, Chemical Engineering Science 65 (2010) 6435–6442.

[7] J. Latham, A. Munjiza, The modelling of particle systems with real shapes, Philosophical Transactions of The Royal Society of London Series A: Mathematical Physical and Engineering Sciences 362 (2004) 1953–1972.

[8] P. Cleary, M. Sawley, Dem modelling of industrial granular flows: 3d case studies and the effect of particle shape on hopper discharge, Applied Mathematical Modelling 26 (2002) 89–111.

[9] D. Zhao, E. Nezami, Y. Hashash, J. Ghaboussi.J., Three-dimensional discrete element simulation for granular materials, Computer-Aided Engineering Computations: International Journal for Engineering and Software 23 (2006) 749–770.

[10] D. Markauska, Investigation of adequacy of multi-sphere approximation of elliptical particles for dem simulations, Granular Matter 12 (2010) 107–123.

[11] S. Mack, P. Langston, C. Webb, York.T., Experimental validation of polyhedral discrete element model, Powder Technology 214 (2011) 431–442.

[12] NVIDIA, Cuda 6 (May 2014).
URL `http://www.nvidia.com/cuda`

[13] NVIDIA, Nvida kepler gk110 architecture whitepaper (2012).
URL `http://www.nvidia.com/NVIDA KEPLER GK110 Architecture Whitepaper`

[14] J. Sanders, E. Kandrot, CUDA by example, Vol. 12, 2010.

[15] N. Bell, Y. Yu, Particle-based simulation of granular materials, Eurographics/ACM SIGGRAPH Symposium on Computer Animation 25 (2005) 29–31.

[16] N. Govender, D. N. Wilke, S. Kok, R. Els, Development of a convex polyhedral discrete element simulation framework for nvidia kepler based gpus, Journal of Computational and Applied Mathematics.

[17] J. J. Jimenez, R. J. Segura., Collision detection between complex polyhedra, Computers and Graphics 32 (2008) 402–411.

[18] B. Nassauer, T. Liedke, Polyhedral particles for the discrete element method, Granular Matter 15 (2013) 85–93.

[19] C. Boon, G. Houlsby, S. Utili, A new algorithm for contact detection between convex polygonal and polyhedral particles in the discrete element method, Computers and Geotechnics 44 (2012) 73–82.

[20] B. Grunbaum, Convex Polytopes, 2nd edition, Volker Kaibel, ISBN 978-0-387-40409-7, 2003.

[21] P. Langston, Distinct element modelling of non-spherical frictionless particle flow, Chemical Engineering Science 59 (2004) 425–435.

[22] P. Cundall, Formulation of a three-dimensional distinct element model - part i: a scheme to detect and represent contacts in a system composed of many polyhedral blocks, Int. J. of Rock Mech 25 (1988) 107–116.

[23] J. Longmore, et.al, Towards realistic and interactive sand simulation: A gpu-based framework, Powder Technology 235 (2013) 983–1000.

[24] E. Battey-Pratt, T. Racey, Geometric model for fundamental particles, International Journal of Theoretical Physics 19 (1980) 6.

[25] T. Harada, GPU Gems 3: Real-time rigid body simulation on GPUs, Vol. 3, 2008.

[26] G. Neubauer, C. A. Radeke., Gpu based particle simulation framework with fluid coupling ability (March 2014).
URL `http://on-demand.gputechconf.com/gtc/2014/poster/pdf/P4143`

[27] N. Govender, D. Wilke, S. Kok, A gpu based polyhedral particle dem transport code (March 2014).
URL `http://on-demand.gputechconf.com/gtc/2014/poster/pdf/P4126`

Potential Reviewers

Dr Raj Rajamani
University of Utah
rajkrajamani@gmail.com
Expert in DEM and GPU

Samiullah Baig
Montan university Austria
samiullah.baig@unileoben.ac.at
Expert in GPU and SPH which is similar to DEM