Blaze-DEM: A GPU based large scale 3D discrete element particle transport framework

by

Nicolin Govender

A thesis submitted in partial fulfillment
of the requirements for the degree

Philosophiae Doctor (Mechanical Engineering)

in the

Department of Mechanical Engineering
Faculty of Engineering, the Built Environment and Information Technology

University of Pretoria
Pretoria

July 2015

Abstract

Understanding the dynamic behavior of particulate materials is extremely important to many industrial processes with a wide range of applications ranging from hopper flows in agriculture to tumbling mills in the mining industry. Thus simulating the dynamics of particulate materials is critical in the design and optimization of such processes. The mechanical behavior of particulate materials is complex and cannot be described by a closed form solution for more than a few particles. A popular and successful numerical approach in simulating the underlying dynamics of particulate materials is the discrete element method (DEM). However, the DEM is computationally expensive and computationally viable simulations are typically restricted to a few particles with realistic particle shape or a larger number of particles with an often oversimplified particle shape. It has been demonstrated for numerous applications that an accurate representation of the particle shape is essential to accurately capture the macroscopic transport of particulates.

The most common approach to represent particle shape is by using a cluster of spheres to approximate the shape of a particle. This approach is computationally intensive as multiple spherical particles are required to represent a single non-spherical particle. In addition spherical particles are for certain applications a poor approximation when sharp interfaces are essential to capture the bulk transport behavior. An advantage of this approach is that non-convex particles are handled with ease. Polyhedra represent the geometry of most convex particulate materials well and when combined with appropriate contact models exhibit realistic transport behavior to that of the actual system. However detecting collisions between the polyhedra is computationally expensive, often limiting simulations to only a few thousand of particles.

Driven by the demand for real-time graphics, the Graphical Processor Unit (GPU) offers cluster type performance at a fraction of the computational cost. The parallel nature of the GPU allows for a large number of simple independent processes to be executed in parallel. This results in a significant speed up over conventional implementations utilizing the Central Processing Unit (CPU) architecture, when algorithms are well aligned and optimized for the threading model of the GPU. This thesis investigates the suitability of the GPU architecture to simulate the transport of particulate materials using the DEM. The focus of this thesis is to develop a computational framework for the GPU architecture that can model (i) tens of millions of spherical particles and (ii) millions of polyhedral particles in a realistic time frame on a desktop computer using a single GPU.

The contribution of this thesis is the development of a novel GPU computational framework Blaze-DEM, that encompasses collision detection algorithms and various heuristics that are optimized for the parallel GPU architecture. This research has resulted in a new computational performance level being reached in DEM simulations for both spherical

and polyhedra shaped particles.

In terms of the particle shape there are currently no other freely available codes that can match the geometrical fidelity in terms of accurate particle shape representation on the GPU. To the authors knowledge there is only one study on the GPU that takes particle shape into account with a physics model of similar fidelity to Blaze-DEM. In that study by Longmore at al. the clumped sphere method is used. Blaze-DEM is able to simulate 2 orders of magnitude more particles compared to other published results while being 3 times faster. The only reported implementations for polyhedra are on the CPU platform. Blaze-DEM is hundreds of times faster compared to CPU codes with physics models of a similar fidelity and 24 times faster than CPU codes with physics models of a lower fidelity. For simulations involving spherical particles Blaze-DEM is 5 times faster than other GPU based codes that have physics models of a similar fidelity.

Acknowledgments

I would firstly like to thank my supervisors, Dr Daniel Wilke and Professor Schalk Kok for their time and dedication during the course of my PhD which often required late nights and weekends. I will always value the advice and knowledge you'll have imparted in my development as a scientific researcher. I would like to thank the CSIR for the finance which made this PhD possible. Thank you to my manger at the CSIR Dr Onno Ubbink for all your support. Thank you to Gail Mokgokong for all your help with administration at the CSIR. Thank you to Dr Igle Gledhill who guided my initial development into the research area and continued to motivate me during the course of my PhD. Thank you to Professor Simon Connell for the support and advice you gave me throughout my post graduate career. Thank you to Professor Rajamani for inviting me to the University of Utah and sharing your vast knowledge on discrete element modeling.

Thank you to my parents (Johnson and Margie Govender) for raising me with love and supporting me in all my endeavors. Thank you for teaching me the value of hard work and education. Thank you to my girlfriend Prenisha who stood by me throughout my studies. Thank you to god for the knowledge and grace he has bestowed upon me.

This work was supported in part by CSIR project SRP TA-2011-001. The support of the NVIDIA Corporation through the donation of GPUs is acknowledged gratefully. The support of the University of Utah in providing research funds for my stay at the university to do the work in Chapter 4 is appreciated. The support of Mines-Douai (France) in providing experimental results against which our of the hopper simulations in Chapter 2 could be validated is also appreciated.

Contents

1 Introduction **12**
1.1 Particulate materials 12
1.2 The discrete element method (DEM) 13
1.3 Computing Aspects 16
1.3.1 The Graphics Processor Unit (GPU) 16
1.3.2 Parallel computing 19
1.3.3 Computational implementations of the DEM 21
1.4 Overview 21

2 Development of a computational framework for the DEM on NVIDIA based GPUs **23**
2.1 Introduction 23
2.1.1 Background and Motivation 23
2.2 Collision Detection 25
2.2.1 Particle representation 25
2.2.2 Data Storage 26
2.2.3 Broad-phase Collision Detection 27
2.2.4 Narrow phase 29
2.3 Contact Resolution 30
2.3.1 Force Calculations 30
2.3.2 Numerical Integration 31
2.3.2.1 Angular Integration 32
2.4 Computational Implementation 34
2.4.1 Blaze-DEM framework. 34
2.5 Simulation examples with BLAZE-DEM 36
2.5.1 Numerical Verification of code 36

2.5.2 Gravity Packing 37
2.5.3 Hopper flow 39
2.6 Conclusions 42

3 Collision detection of convex polyhedra on the GPU architecture for the DEM 43
3.1 Introduction 43
3.2 Polyhedra Contact Detection 43
3.2.1 World Representation 44
3.2.2 Theoretical Formulation 45
3.2.3 Polyhedron-Polyhedron Contact Algorithm 46
3.2.3.1 Edge-edge contact detection 47
3.2.4 World-Polyhedron Contact 49
3.3 Contact Resolution 50
3.3.1 Normal Force 50
3.3.1.1 Restorative Force 50
3.3.1.2 Dissipative Force 53
3.3.1.3 Contact Evaluation for particle in free fall. 55
3.3.2 Tangential Force 56
3.4 Numerical Simulation 57
3.4.1 Polyhedra in a drum 57
3.4.2 Scaling with particle shape 58
3.4.3 Algorithm performance 61
3.5 Conclusion 63

4 Discrete Element Simulation of Mill Charge using the BLAZE-DEM GPU framework 65
4.1 Introduction 65
4.1.1 Background and Motivation 65
4.1.2 Computational Aspects 66
4.1.3 Additions to BLAZE-DEM framework for mill simulations 66
4.1.4 Force Model 68
4.1.4.1 Calculation of power drawn by a mill 70
4.1.5 Calibration of model parameters. 70
4.1.5.1 Effect of parameters on charge profile 71
4.2 Experimental validation of GPU DEM for mill simulations. 74
4.2.1 Three-dimensional mill 74
4.2.1.1 Calibration of model parameters 74
4.2.1.2 Charge motion and power draw 76

4.2.2 Charge motion and power draw for a slice mill 78
4.3 Industrial Mill Simulation 81
4.3.1 Performance scaling 84
4.4 Conclusions 86

5 Conclusion and Future work **87**
5.1 Concluding remarks 87
5.2 Future work 89

List of Figures

1.1 Examples of particulate materials on various scales. 12
1.2 (a) Discrete and (b) continuum views of particulate materials [1]. 13
1.3 (a) Single sphere, (b) clumped sphere and (c) polyhedron representations of a corn kernel. 14
1.4 DEM force assumption. 16
1.5 a) Quad core Intel CPU and b) NVIDIA Kepler GPU Chip layouts (196 cores per streaming multiprocessor (SM). 17
1.6 Comparison between CPU and GPU task processing for the case of (a) different incoming tasks and (b) identical incoming tasks. 18
1.7 Flow chart of DEM simulation procedure. 19
1.8 Domain decomposition on CPUs [2]. 20

2.1 Planar polygon representation. 25
2.2 Particle object representation. 26
2.3 AOS vs SOA data access patterns. 27
2.4 Data representation on the GPU. 27
2.5 Broad phase collision detection grid. 28
2.6 Polyhedra contact types. 30
2.7 Force interaction model. 30
2.8 Error in Energy Conservation. 32
2.9 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. 33
2.10 BLAZE-DEM Framework. 35
2.11 Verification of numerical integration scheme for different time steps. . . . 37
2.12 Gravity packing for 1 million polyhedra. 38

2.13 Energy dissipation of damped system packing under gravity (inital velocity of 1 m.s^{-1}in all directions). . . . 38
2.14 Performance scaling with number of particles on a Tesla K20 GPU for octahedra (8 face). . . . 39
2.15 (a) Experimental setup indicating the number of particles N in each layer, discharge angle β and dimensions of the inlet and outlet, (b) particle specifications. . . . 40
2.16 Flow rate and patterns for (a) polyhedra and (b) spheres. . . . 40
2.17 Hopper flow with 13824 corn shaped polyhedral particles. . . . 41
2.18 Polyhedra arching to restrict flow. . . . 42

3.1 World Object Types. . . . 44
3.2 Polyhedra surface contact types. . . . 45
3.3 Illustration of Theorem 1 in 2D. . . . 46
3.4 (a) Non penetrating Type 2 contact, (b) Penetrating Type 2 contact. . . . 47
3.5 Algorithm 1: Polyhedron-polyhedron contact detection. . . . 48
3.6 Check for surface collisions. . . . 49
3.7 Algorithm 2: Polyhedron-world planar surface contact. . . . 49
3.8 (a) Polyhedral particle contact model and (b) Simulation scenario for Section 3.3 50
3.9 Penetration distance as a function of velocity for a head-on elastic collision. ($K_n = 1 \times 10^9$ N.cm^{-1}, step size =10^{-5} s) 51
3.10 $\mathbf{F}_N^{elastic}$ as function of penetration depth for a head-on elastic collision. ($K_n = 1 \times 10^9$ N.cm^{-1}, step size =10^{-5} s) 52
3.11 Velocity as a function of time step for a head-on elastic collision. ($K_n = 1 \times 10^9$ N.cm^{-1}, step size =10^{-5} s) 52
3.12 Normal force as a function of penetration depth for a head-on in-elastic collision. ($K_n = 1 \times 10^9$ N.cm^{-1}, $K_D = 5 \times 10^4$ kg.s^{-1}, step size =10^{-5} s , $\mathbf{V} = 2$ m.s^{-1}) 54
3.13 $\| \mathbf{F}_N^{elastic} \| - \| \mathbf{F}_N^{diss} \|$ vs velocity ($K_n = 1 \times 10^9$ N.cm^{-1}, $K_D = 5 \times 10^4$ kg.s^{-1}, step size =10^{-5} s). . . . 54
3.14 Position vs Time for face contact ($K_n = 1 \times 10^9$ N.cm^{-1}, $K_D = 5 \times 10^4$ kg.s^{-1}, step size =10^{-5} s). . . . 55
3.15 Velocity vs Time ($K_n = 1 \times 10^9$ N.cm^{-1}, $K_D = 5 \times 10^4$ kg.s^{-1}, step size =10^{-5} s). . . . 55
3.16 (a) Position vs Time and (b) Velocity vs Time ($K_n = 1 \times 10^9$ N.cm^{-1}, $K_D = 2 \times 10^5$ kg.s^{-1}, $\mu = 1.54$, $K_T = 4 \times 10^3$ kg.cm.s^{-1}, step size =10^{-5} s). 56
3.17 $\| \mathbf{F}_T \|$vs Tangential Velocity ($K_n = 1 \times 10^9$ N.cm^{-1}, $K_D = 2 \times 10^5$ kg.s^{-1},$\mu = 1.54$, $K_T = 4 \times 10^3$ kg.cm.s^{-1}, step size =10^{-5} s). . . . 57

3.18 Dynamic packing of cubes in a drum. 58
3.19 Packing of cubes in a drum (symmetric). 59
3.20 Gravity packing simulation for performance benchmarking. 59
3.21 Computational scaling of polyhedra-world surface contact collision detection with world surfaces. 60
3.22 Effect of particle shape (number of faces) on the computational cost of collision detection. 60
3.23 Scaling of polyhedron-polyhedron collision detection with number of particles (N) for different polyhedra. 61
3.24 Scaling of polyhedron-polyhedron collision detection with number of particles (N) for Cubes (6 Face) on a K6000 GPU. 61
3.25 Scaling of polyhedra-world planar surface contact algorithm algorithm with number of particles (N) for Cubes (6 Face) on a K6000 GPU. 62
3.26 Scaling of Broad-Phase algorithm with number of particles (N) on a K6000 GPU. 62
3.27 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. 63

4.1 (a) Particle-lifter broad-phase collision detection and (b) detailed collision detection. 67
4.2 Normal and tangential force models depicted by a spring dash-pot system. 69
4.3 Charge profiles for CPU (a) $\mu = 0.70$ and GPU (b) $\mu = 0.70$ (c) $\mu = 0.60$ and (d) $\mu = 0.40$, N= 2916 (radius=2.5 cm). 72
4.4 GPU charge profiles for (a) $\epsilon = 0.25$ (b) $\epsilon = 0.45$ and (c) $\epsilon = 0.65$, N= 2916 (radius=2.5 cm). 72
4.5 (a) CPU and (b) GPU charge profiles. N= 5344 (radius=1.85 cm). . . . 73
4.6 (a) CPU and (b) GPU charge profiles. N= 11664 (radius=1.25 cm). . . 73
4.7 (a) Experiment (b) GPU charge profiles for different values of μ as indicated. N= 168 (20% filling), rpm = 32 (70% of critical speed). 75
4.8 GPU charge profiles for different values of ϵ as indicated. N= 168 (20% filling), rpm = 32 (70% of critical speed). 75
4.9 (a) Experiment [3] (b) GPU charge profiles. N= 168 (20% filling), rpm = 14 (30% of critical speed). 76
4.10 (a) Experiment [3] (b) GPU charge profiles. N= 168 (20% filling), rpm = 22 (50% of critical speed). 77
4.11 (a) Experiment [3] (b) GPU charge profiles. N= 243 (30% filling), rpm = 14 (30% of critical speed). 77

4.12 (a) Experiment [3] (b) GPU charge profiles. N= 243 (30% filling), rpm = 22 (50% of critical speed). 78
4.13 Power draw for (a) 25% and (b) 35% loading between experiment [4], GPU and CPU simulations. 79
4.14 (a) Experiment [4] (b) GPU and (c) CPU charge profiles. N= 169 (35% filling), rpm = 17.50 (30% critical speed). 80
4.15 (a) Experiment [4](b) GPU and (c) CPU charge profiles. N= 120 (25% filling), rpm = 40.81 (70% critical speed). 80
4.16 (a) Experiment [4] (b) GPU and (c) CPU charge profiles. N= 169 (35% filling), rpm = 58.30 (100% critical speed). 81
4.17 (a) Experiment [4] (b) GPU and (c) CPU charge profiles. N= 169 (35% filling), rpm = 93.30 (160% critical speed). 81
4.18 Lifter design Los Bronces semi-autogenous mill. 82
4.19 (a) Initial conditions N= 139392 (41% filling) (b) Charge profile of Los Bronces mill (colored by particle size as indicated in Table 4.6). 83
4.20 (a) Total power draw (b) Power distribution over time of Los Bronces mill. 84
4.21 (a) Initial conditions N=4×10^6 (35% filling) (b) steady state profile (orthogonal view) (c) steady state profile (isometric view). 84
4.22 Scaling of GPU code with number of particles for ball mills on a Kepler GPU. 85
4.23 (a) 2D steady state profile, N=6744 (b) Steady state profile for a slice 10% of the length, N=385534. 86

5.1 World geometry representations. 88
5.2 Incorrect surface selection. 90

List of Tables

1.1 Theoretical of CPU and GPU parallel solutions. 20

2.1 Hardware Specifications. 34

3.1 Analysis of particle velocities for a head-on elastic collision. 53
3.2 Parameters used in simulations for non-linear force model. 57
3.3 Comparison to other GPU codes (SD: Spring-Dashpot (Normal), NIT: Non-Incremental Tangent, IT: Incremental Tangent, IV: Impulse Velocity). 63

4.1 Untuned model parameters used in simulation for a 2D mill. 71
4.2 Tuned model parameters used in simulation for a 2D mill. 73
4.3 Average error in power for various parameter combinations for a 3D mill. 76
4.4 Power draw with experiment and GPU DEM for 3D mill. 78
4.5 Model Parameters used in simulation for slice mill. 79
4.6 Charge distribution for Los Bronces mill. 82
4.7 Model Parameters used in simulation of Los Bronces mill. 82

CHAPTER 1

Introduction

1.1 Particulate materials

From asteroids moving in space to sand covering the earth, particulate materials are all around us (Figure 1.1)[5]. Understanding the macroscopic behavior of particulate materials is essential to understanding the conditions that results in an avalanche or rock falls, potentially saving hundreds of human lives. Understanding this behavior could lead to reducing the energy consumption in many industrial processes such as the mixing of powders to make tablets to the crushing of ore in a ball mill.

Figure 1.1: Examples of particulate materials on various scales.

The macroscopic behavior of particulate materials is complex. Corn flowing in a hopper can behave like a liquid pouring out with ease, while having the ability to stop without any intervention, behaving like a solid. It comes as no surprise then that the underlying dynamics cannot be described by a single theoretical model as the system exhibits both fluid and solid behavior. Thus numerical simulation is an attractive option to gain insights into this complex macroscopic behavior.

Numerical simulation of particulate materials.

There are two general approaches than can be used to model particulate materials, namely continuum and discrete approaches.

1. Continuum approaches assume that the material being modeled completely occupies the space in which it is contained in as depicted in Figure 1.2(b). In a continuum approach the system is represented by a set of partial differential equations that are solved over a spatial domain resulting in physical properties being averaged over a finite volume. This averaging makes continuum methods only valid for particulate systems comprised of very small particles which responds to forces in a way that is typical of viscous fluids [6]. Sand is an example of a particulate system that can be accurately modeled with a continuum approach, as demonstrated by Herrmann and Luding [7]. However, if we want to predict when corn in a hopper will stop flowing, a continuum approach will predict smooth flow and thus not provide the required fidelity in the physics to understand the conditions that lead to blocking. Thus the application of continuum methods to model particle dynamics is limited as they do not model the inter-particle interactions which result in the complex behavior that governs particulate flow. However, the restrictive assumptions of a continuum model allows for large scale problems to be solved computationally efficiently.

2. Discrete approaches on other hand captures this complex macroscopic behavior as the material is modeled on the particle level as illustrated in Figure 1.2(a). However this detail comes at a very large computational cost compared to continuum methods. Thus the primary focus of this work is to decrease the computational cost of a discrete approach while maintaining a high fidelity model of the physics.

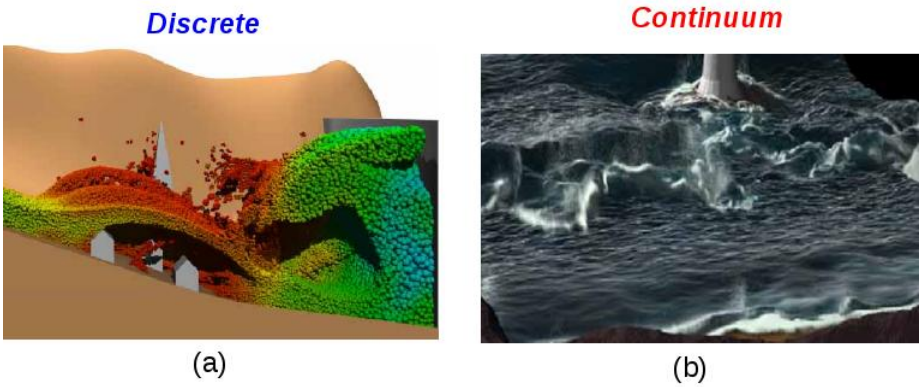

Figure 1.2: (a) Discrete and (b) continuum views of particulate materials [1].

1.2 The discrete element method (DEM)

The discrete element method (DEM), which was described by Cundall and Strack in 1979 [8], is one of the most successful discrete methods in simulating particulate materials. The

DEM was originally developed for solving problems in geotechnical engineering, but has been employed to model particulate materials in a variety of fields [9, 10, 11, 12].

The DEM requires all particles in the system to be checked for contact at each time step, which involves a considerable number of calculations depending on particle geometry and number of particles [13] in the system. To reduce the computational cost, the particle shape is often approximated by a sphere (Figure 1.3(a)), for which contact detection is trivial. This approximation however may result in the model exhibiting unrealistic mechanical behavior, as discussed by Latham and Munjiza [14, 15]. The clumped-sphere (Figure 1.3(b)) approximation [16] provides a better description of shape by using a number of spheres to represent a particle. However, this approach is limited in the number of particles and introduces non-physical artifacts into the simulation, as discussed by Horner [17]. Polyhedral shaped particles, depicted in Figure 1.3(c), can capture details in particle shape well and hence exhibit realistic transport behavior to that of the actual system [18, 19]. 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 et al. [20], in which only 322 polyhedra are simulated. This limitation is due primarily to the complexity of collision detection and larger memory storage requirements of the polyhedra.

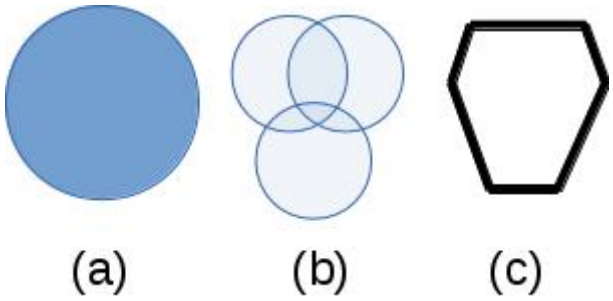

Figure 1.3: (a) Single sphere, (b) clumped sphere and (c) polyhedron representations of a corn kernel.

Once the particles that are in contact is determined, the resultant forces acting on the particles can be calculated. The combined Finite Element Method and Discrete Element Method (FEM-DEM) approach by Munjiza [21] offers a high fidelity model using the fully coupled inter-particle kinetics and particle material response to calculate the contact forces. As with all numerical simulations there is a trade-off between model accuracy and computational speed. In Munjiza's paper [21], the number of particles that can be modeled is limited due to the large computational cost of the associated FEM simulations and the solution of the coupled dynamic equilibrium problem associated with multiple bodies being simultaneously in contact. This allows for the simulation of weakly coupled problems in typical bulk flow analysis to strongly coupled problems

associated with the compaction of particles. This study focusses only on typical bulk flow problems that allows for some assumptions to be made without significant loss of accuracy, but drastically benefit the computational efficiency of the numerical model. In this study we assume only binary contact to resolve the associated contact forces on particles. In addition we consider the original model for force calculation in DEM simulations proposed by Cundall et al. [22]. The force is described by a spring-dashpot coupled in parallel, which offers an appropriate model fidelity for the problems under consideration that is computationally efficient as opposed to the computational expensive FEM-DEM numerical models [4, 8, 10, 20, 23]. Once the forces are calculated an explicit integration scheme such as velocity Verlet or forward Euler is often used to determine the resultant motion of all particles in the system. Since the aim of this study is to develop a modular framework that will allow us to easily implement different force models based on the type of problem being simulated we do not advocate the use of a particular force model in this thesis. In Chapter 2 we used a linear force model that was used by other researchers [24, 25] to conduct similar validation of their Graphical Processor Unit (GPU) DEM codes. In Chapter 3 we used a non-linear (Hertzian) force model that was suggested by Hromnik et al. [26] in his thesis on GPU DEM. Finally in Chapter 4 we used the force model [3, 10, 27] that has been widely used by researchers for tumbling mills.

Specifics of DEM model used in this study

An assumption in many DEM simulations is that particles are considered to be perfectly rigid for the duration of collision contact. In reality perfectly rigid particles do not exist, as all bodies will experience (to some extent) local deformations during contact. These deformations however occur on a time scale which is much smaller than what is required for capturing the macroscopic behavior of a system. Thus it is often sufficient to use a constitutive law, such as a linear spring, to model contact forces. Computing the time evolution of the system requires us to solve simultaneously Newton's equations of motion for all contacting particles, which on current hardware (2015) is only possible for a few thousand rigid bodies [28]. Hence, we assume that there are only binary contacts between particles at any given time as depicted in Figure 1.4, in addition to limiting ourselves to explicit time integration schemes. The total force acting on a particle is obtained by summing the individual contributions of all the binary contacts of a particle per time step. This is a good approximation of reality provided the particles are of a similar size and move very little during a time step.

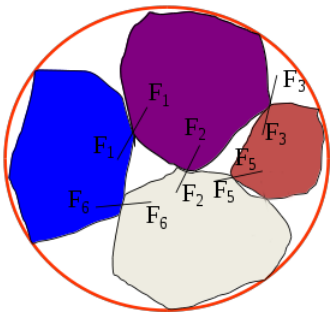

Figure 1.4: DEM force assumption.

In summary the assumptions that we make are:

1. Rigid bodies with point contact,
2. Explicit integration,
3. Local short-range interactions,
4. Non-incremental friction model and
5. Limited to explicit time integration.

These assumptions result in a system that is completely decoupled and can be expressed as a Lagrangian type process in which we are able to simulate the motion of individual particles independently of each other [8], resulting in our algorithm being ideally suited to the GPU.

1.3 Computing Aspects

A computer needs to perform two main tasks mainly (1) logical operations and (2) data processing. Traditionally the CPU handled both tasks. The introduction of the graphical based operating system windows with Graphical User Interfaced (GUI) applications and games in the early 90s resulted in a greater demand for more sophisticated graphical output. This resulted in the creation of a co-processor designed to execute some of the graphical tasks that was previously performed on the CPU. The NVIDIA GeForce graphics card released in 1999 was marketed as the worlds first consumer Graphics Processor Unit (GPU) and performed the remaining graphical tasks that was still performed on the CPU.

1.3.1 The Graphics Processor Unit (GPU)

Figure 1.5 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. The graphical

operations required to render images to a screen involves simple algebraic operations (addition, subtraction and multiplication) to be performed for every pixel on the screen. The GPU is required to be a massively parallel processor as the typical screen has millions of pixels that need to be simultaneously updated to render visual information for display to the user. Thus the GPU hardware design is such that it be made up of mostly Arithmetic Logic Units (ALUs), enabling it to perform these arithmetic operations in parallel. The CPU on the other-hand is designed to do a variety of different tasks such as running an operating system while being able to perform arithmetical operations. Thus its main goal is to complete a single task as quickly as possible so that it can process the next task.

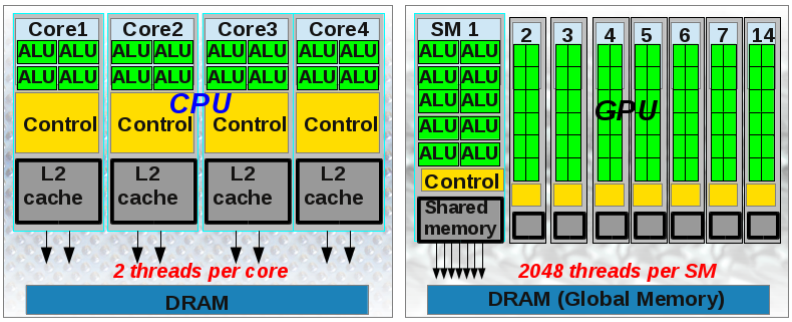

Figure 1.5: a) Quad core Intel CPU and b) NVIDIA Kepler GPU Chip layouts (196 cores per streaming multiprocessor (SM).

As an example, consider the analogy of transporting 60 people across the English channel (20km) in an hour. The options are to use a ferry or the road. Suppose the car can hold a maximum of 4 people traveling at a speed of 120km/h and the ferry 60 people at a speed of 20km/h. The car takes just 6 minutes to make the trip while the ferry takes an hour. However the car can only transport 40 people in an hour while the ferry takes 60 people in this time. This is termed as latency and throughput. The CPU, which is like the car, is designed to complete a single task in the shortest possible time (reduce latency). The GPU on the other hand is like the ferry and is concerned with how much work is done for a given amount of time (increase throughput). This difference is a direct result of the different native designs and purposes of these devices. It is important to note that in this analogy we are conducting the same task for each person i.e. transporting them across the channel.

In the mid 2000s with the introduction of the NVIDIA Geforce 8 series cards, consumer GPUs finally had their own dedicated memory and programmable units. This effectively made the GPU a large scale parallel processor that could perform operations other than graphics processing and gave rise to the term General Purpose Graphics Processor Units (GPGPU). In this thesis we exploit the computational power of the GPU via the NVIDIA developed CUDA programming model [29], which allows us to issue

commands to the GPU from C++ code as opposed to a graphics language like OpenGL. The CUDA programming model batches threads into blocks (max 1024 threads) for execution on a streaming multiprocessor (SM). Threads within blocks can access fast shared memory with each thread in turn having access to its own 32 bit registers (fastest memory available). 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 of the execution order of blocks). The execution of a block will only complete once all 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.

The GPU has two memory spaces: on-chip memory (shared memory and registers) which are very fast but limited in size (48 KB) and scope; and off-chip memory (global-memory) which is much larger (typically 2-24 GB) 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 1.6 illustrates the type of tasks that each unit excels at in computational performance. The limited GPU outperforms the versatile CPU in spite of the considerably lower clock rate when processing identical tasks.

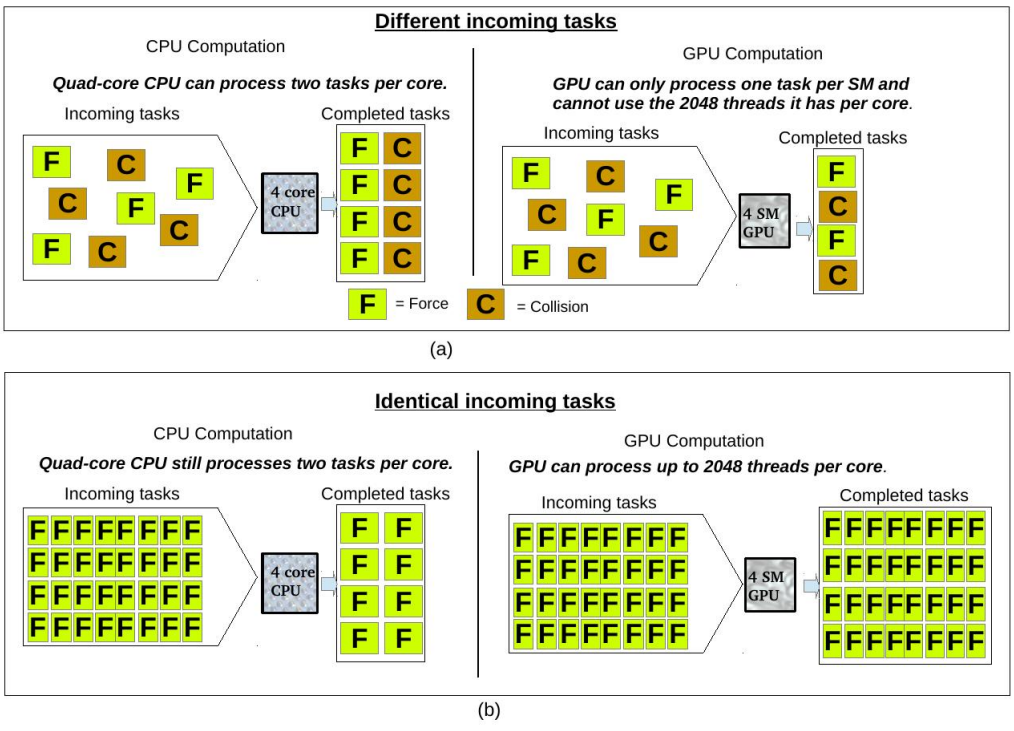

Figure 1.6: Comparison between CPU and GPU task processing for the case of (a) different incoming tasks and (b) identical incoming tasks.

Thus the parallel power of the GPU is only useful for a class of problems termed Single Instruction Multiple Data (SIMD) problems where there are a large number of tasks which are independent of each other. Furthermore memory transactions and CPU-GPU communication must be kept to a minimum as it can cause major bottlenecks.

1.3.2 Parallel computing

Since DEM simulations are compute bound, the number and complexity of simulated particles has scaled with increased computational power over the past three decades. In the last few years the trend of increasing clock speed has stopped due to the physical limitation of materials and physics. While computational power still scales with Moore's Law, this scaling is now achieved through increasing the number of computing cores on a single chip. This means that algorithms which execute in serial no longer see any performance benefit.

A general indication of the suitability of an algorithm to parallel execution is the presence of a loop where each iteration is independent of the other. These tasks can then be executed in parallel. The flow-diagram in Figure 1.7 describes the various steps in the DEM process that we model.

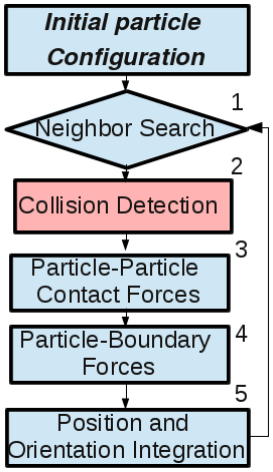

Figure 1.7: Flow chart of DEM simulation procedure.

The steps colored in blue have been implemented on the GPU by various groups [13, 26, 30, 31] as they are naturally independent and not memory intensive. They are therefore well suited to the threading and memory model of the GPU. However the collision detection step (red) which is required for polyhedra has not been implemented at all on the GPU while there are very few CPU implementations [18, 20, 32]. This is due to the computationally intensive algorithms which cannot be easily expressed as independent tasks and the additional associated memory requirements in the case of a GPU implementation. All the algorithms used in Blaze-DEM which are discussed in detail in

Chapter 3 are designed such that they can be executed as SIMD tasks. Even when a certain part of the algorithm can be executed more efficiently in serial, we use the GPU as the cost of memory transactions between the CPU and GPU is much greater than computation time required for executing the algorithm.

While most scientific CPU codes are now parallel, the CPU is still only a multi-core processor with the Intel Xeon chip having 12 cores on a single chip. Thus parallelism is limited to domain decomposition [33] as depicted in Figure 1.8. In a dynamic environment particles move through multiple domains requiring interchange of data between processors which is time consuming. Furthermore particles are still processed in a serial loop on each CPU core. The GPU however is a many core processor enabling parallelism at particle level, with each particle having its own thread.

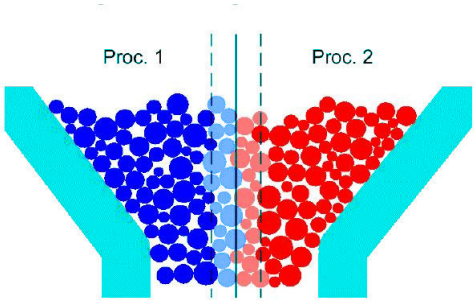

Figure 1.8: Domain decomposition on CPUs [2].

Table 1.1 depicts the theoretical performance of a Xeon CPU and Tesla GPU for the task of performing computations for 10 millions particles. In spite of the considerably higher clock rate the CPU can only launch 12 threads per cycle while the GPU can launch 53284 threads per cycle. This gives the GPU an enormous edge over the CPU in DEM calculations which are data parallel resulting in a speed up of 500 when taking into consideration cost and power consumption. Furthermore the thread scheduling is done automatically on the GPU by CUDA which removes the burden of ensuring effective parallel computation from the programmer. This also makes the same code scalable on future GPU hardware thus increasing performance without changing the code.

Table 1.1: Theoretical of CPU and GPU parallel solutions.

CPU	**GPU**
Intel® Xeon® Processor E7-8857	NVIDIA® Tesla® K80
3.0 GHz x 12 cores	1.0 GHz x 26 SM 53284 threads
Typical compute time for 10^6 particles	
subdomains: $10^6/12 = 83 \times 10^5$particles per core	10^6 threads are created (one per particle)
3 computations per cycle (3GHz)	1 computations per cycle (1GHz)
Time for 10^6 particles= $\frac{83\times10^5}{3} = 27777$ s	Time for 10^6 particles= $\frac{10^6}{53284} = 18.76$ s

1.3.3 Computational implementations of the DEM

Thirty years after the first DEM code BALL [8] simulated a few hundred two dimensional disks, current DEM codes are capable of simulating millions of particles in a variety of environments. There are several commercial DEM packages (PFC by Itasca, EDEM by DEM Solutions, and Rocky by Granular Dynamics) and in-house packages such as BLOCKS3D by the University of Illinois, Y-DEM by Queen Mary University and Millsoft [3] by the University of Utah that simulate particulate materials using the DEM. There are also open-source packages such as LIGGGHTS which are capable of simulating particulate materials.

While there has been some development from the academic sector with DEM implementations of spherical particles on the GPU [25, 26, 30, 31], there has been very little development in terms of polyhedra particles. To the best of the author's knowledge the only effort thus far has been using triangles in 2D by Zhang et al. [34]. Although the GPU is an ideal match for DEM simulations the current learning curve associated with GPU development is high as the technology is fairly new compared to the traditional CPU platform. Furthermore only an efficiently implemented GPU DEM solution will yield significant performance benefits over the CPU. To the best of the author's knowledge there are no published works on large scale simulations of polyhedra using the GPU.

The main contribution of this thesis is the development of a novel modular GPU computational framework Blaze-DEM encompassing heuristics and collision detection algorithms optimized for the parallel GPU architecture. This thesis demonstrates that by using the GPU with algorithms optimized for its parallel threading model (i) tens of millions of spherical particles and (ii) millions of polyhedral particles can be simulated in a realistic time frame on a single desktop computer. A major benefit of the Blaze-DEM framework is the modular nature in which the framework was designed. This allows us to easily implement different force models that are apporiate for the problem being simulated, as demonstrated by the three contact models used in this thesis. Furthermore this modular nature allows for integration with external codes to simulate multi-physics problems such as fluid interaction and particle breakage.

1.4 Overview

The following 4 chapters document the author's contribution as a postgraduate student in the Department of Mechanical and Aeronautical Engineering at the University of Pretoria. Each chapter is self-contained and is based on a published ISI journal paper. In Chapter 2 of this thesis we introduce the novel GPU framework Blaze-DEM and validate the numerical stability and physical accuracy of the code. We then demonstrate the importance of particle shape by comparing the spherical particle representation to that

of polyhedra. In Chapter 3 the detailed algorithms and heuristics for collision detection are presented followed by a detailed validation of the physics model and performance evaluation of the code. In Chapter 4 the flexibility of the framework is demonstrated with an application to an industrial problem. The industrial problem entails the simulation of tumbling mills to obtain power draw and profile of the dynamics within the mill. Finally, the study is concluded and recommendations for future work are offered in Chapter 5.

CHAPTER 2

Development of a computational framework for the DEM on NVIDIA based GPUs

2.1 Introduction

2.1.1 Background and Motivation

Transport processes involving Granular Media (GM) occur in many areas of science and engineering over a variety of length scales. Thus understanding the dynamical behavior of GM is central to a large number of engineering disciplines with applications in mining, agriculture and various other fields [9, 10, 11, 12]. Methods belonging to the Discrete Element Method (DEM) family which treats granular material as a system of individual particles, as opposed to a continuum description which averages particle properties, has shown the most promise [23]. The DEM approach which uses a local constitutive law to determine the forces between two contacting particles and consequently the resultant motion of all particles in the system, was described by Cundall and Strack [8]. The DEM is however computationally expensive as all particles in the system have to be checked for contact at each time step. This involves a considerable number of calculations depending on the particle geometry and number of particles [13]. To reduce computational cost, particle shape is often approximated using spheres, for which contact detection is trivial. This approximation however results in the system exhibiting different mechanical behavior to reality as discussed by Latham and Munjiza [14, 15]. The clumped-sphere approach [16] provides a better description of shape by using a number of spheres to represent a particle. However, this approach is limited in the number of particles and introduces non-physical artifacts into the simulation, as discussed by Horner [17].

In modeling GM correctly there are two general aspects that must be taken into con-

sideration:

1. Particle shape.

2. Detailed physics interaction between particles.

The Graphics Processor Unit (GPU) offers cluster type performance on a desktop computer at a fraction of the cost, and is well suited to computations that can be executed in parallel resulting in a performance benefit over the traditional CPU [35]. Radeke and Glasser [13] utilize the GPU to simulate powder mixing taking into account detailed particle interactions between spherical particles. They report that a one minute simulation of one million spherical particles requires 96 hours computing time using a single GPU. Longmore et al. [25] take into account particle shape by using multiple spheres to represent a sand grain with simple particle particle interactions and are able to simulate 256 thousand sand grains at 120 frames per second (FPS) on the GPU. This significant improvement in performance suggests that detailed particle interactions which requires particle contact history is costly on the memory constrained GPU. Thus our DEM framework focuses on the accurate representation of particle shape while using simplified interaction models that are suited to parallel implementation. Such a model finds application in particle flow problems where a simplified physics model can capture the dynamical bulk behavior of the system [24, 25].

Polyhedral shaped particles represent most GM accurately and hence exhibit similar mechanical behavior to that of the actual system [19, 20]. However, the number of polyhedral particles that can be simulated on current CPUs is limited [32, 36], with the largest simulations containing at most a few hundred thousand convex polyhedra [18]. In Nassauer and Liedke's work, 800 polyhedra are simulated with detailed particle interactions (1 FPS) using a parallel CPU implementation. To the best of the author's knowledge there has been no GPU implementations for polyhedral shaped particles. This chapter is intended to be a feasibility study to illustrate a new performance level of DEM by utilizing a physics model that is suited to the GPU while taking into account detailed particle shape. Blaze-DEM requires 3 minutes computational time for a one minute simulation of one million spheres (55 FPS), and 150 minutes for one million convex polyhedra (including graphics rendering) on a single GPU (0.9 seconds per time-step) using the simple physics model described by Longmore et al. [25] and Bell et al. [24]. In this chapter we develop a GPU orientated DEM environment and determine if it is useful in simulating hopper flow problems.

2.2 Collision Detection

2.2.1 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 2.1. 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 [37]. We summarize this result as :

> *The half-space subtended by a face of a convex polyhedra completely partitions space into two distinct regions:*
>
> *The region* $f_i(\mathbf{n}, \mathbf{c}) \leq 0$ *as indicated in Figure* 2.1*, containing the entire polygon.*
>
> *The region* $f_i(\mathbf{n}, \mathbf{c}) > 0$ *an infinite half-space in the direction indicated by the normal to the plane.*

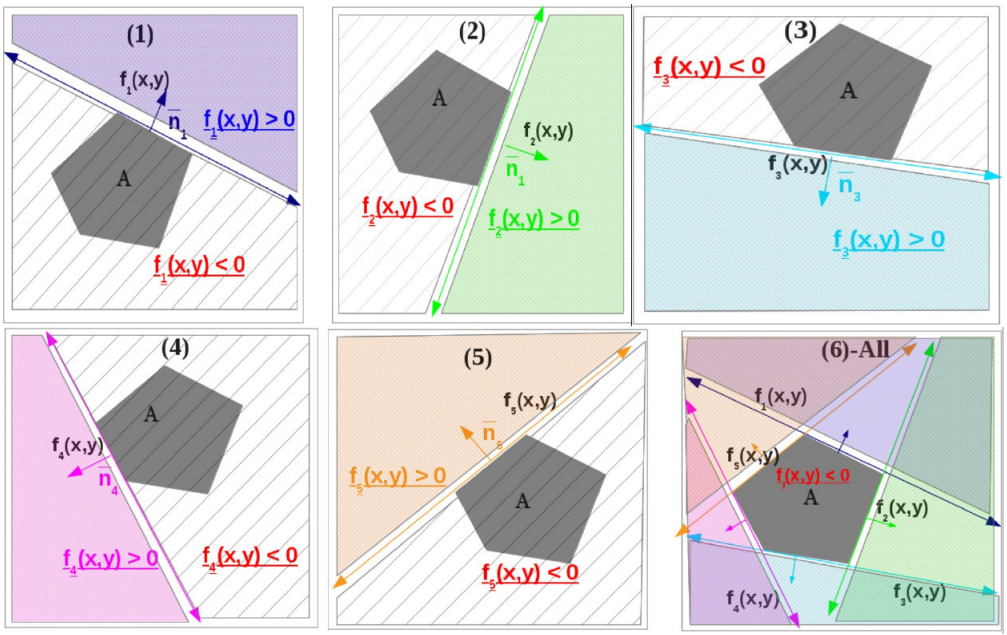

Figure 2.1: Planar polygon representation.

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. Each particle type is stored as a **Particle_Object** as illustrated in Figure 2.2. Vertex information is stored as an array of vectors (**vertex_list**). We store each face plane as **face** struct which contains the normal and centroid of the face. We also store the indices of the vertexes (**vertex_order**) that make up the face as references to **vertex_list**, which is

required for narrow phase collision detection. This minimalistic representation allows us to place data structures in the faster limited constant memory (48 KB) on the GPU, as depicted in Figure 2.2. We hard code the maximum number of features to 32 due to the small size of constant memory. The radius of a sphere that bounds the polyhedra is stored in the variable **bound_R** which we will use for culling in the broad phase.

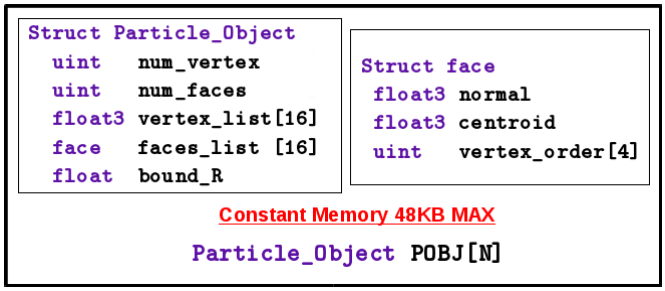

Figure 2.2: Particle object representation.

2.2.2 Data Storage

The major bottleneck on the GPU is memory utilization. We thus need to ensure that we keep memory transactions to a minimum and utilize the different memory spaces available on the GPU to achieve the best possible performance. For each particle we need to store 4 kinematic parameters (position $\mathbf{P}$, velocity $\mathbf{V}$, orientation $\mathbf{Q}$ and angular velocity $\boldsymbol{\omega}$).

The only option we have for storing this information on the GPU is global memory, which is very slow. However we can minimize the impact on performance by minimizing the number of transactions we have to make by ensuring memory transactions are coalesced. Consider Figure 2.3, which shows two types of commonly used data structures:

1. Array Of Structures (AOS): all kinematic parameters for each particle is stored in adjacent memory locations.

2. Structure Of Arrays (SOA): each kinematic array for all particles is stored in adjacent memory locations

To gauge the effective performance of the two representations on the GPU, we ran a simulation of 2 million particles and found that **AOS is three times slower** than SOA. We see better performance with SOA as it allows for coalesced thread access, in that neighboring threads access adjacent memory locations resulting in better utilization of cache, requiring fewer memory transactions.

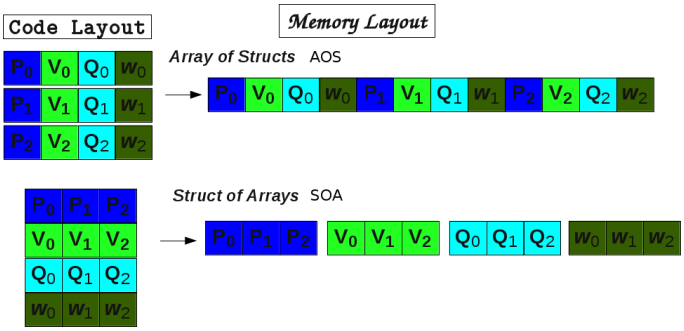

Figure 2.3: AOS vs SOA data access patterns.

In addition to the particle information we also store the force (acceleration) and information about nearest neighbors in global memory, as depicted in Figure 2.4. Information that remains fixed for the simulation is stored in high speed constant memory as described in Figure 2.2. We calculate the evolution of the inertia tensor as required instead of storing it, which is far more expensive. We also do not store geometric information for each particle as this will be very costly. Rather each particle has a reference to a **Particle_Object** (particle_type) which contains the geometric information of that particle type as illustrated in Figure 2.4.

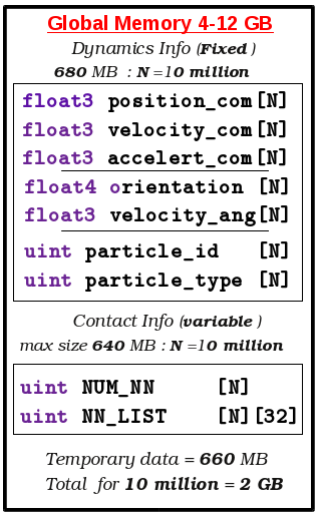

Figure 2.4: Data representation on the GPU.

2.2.3 Broad-phase Collision Detection

In DEM simulations particles only interact via mechanical forces. Hence, we only need to consider particles that can physically be in contact with each other. We are thus able to use spatial subdivision to limit the number of particle pairs that need to be checked for collision at each step. In choosing an algorithm to perform spatial subdivision, we have to take into consideration the following requirements:

1. Particles that cannot be in contact must be excluded with minimal computational

cost.

2. The algorithm must be suited to the SIMD nature of the GPU.

We thus use a collision grid approach [35] which discritizes geometry into a 3D grid as illustrated by Figure 2.5 in which a rectangular hopper is shown. Each particle is assigned a discrete grid position given by $\mathrm{GP^i_j} = \mathrm{floor}((\mathrm{P^i_j} - \mathrm{W_j})/\mathrm{Ncell_j})$, $\mathrm{j} = 1, 3$, where $P^{\mathrm{i}}(\mathrm{x}, \mathrm{y}, \mathrm{z})$ is the center of mass (COM) position of the i^{th} particle in the global coordinate system. Here **W** is the starting point of the grid and $(\mathrm{Ncell_x}, \mathrm{Ncell_y}, \mathrm{Ncell_z})$ number of cells in each dimension of the **W** system and j the Cartesian coordinate as illustrated in Figure 2.5. The minimum size of a cell is that of the largest particle bound radius in the simulation (particle size can vary by a factor of at most 4 without significantly impacting performance for flow problems). Based on the number of threads available on the GPU, the number of cells is optimized.

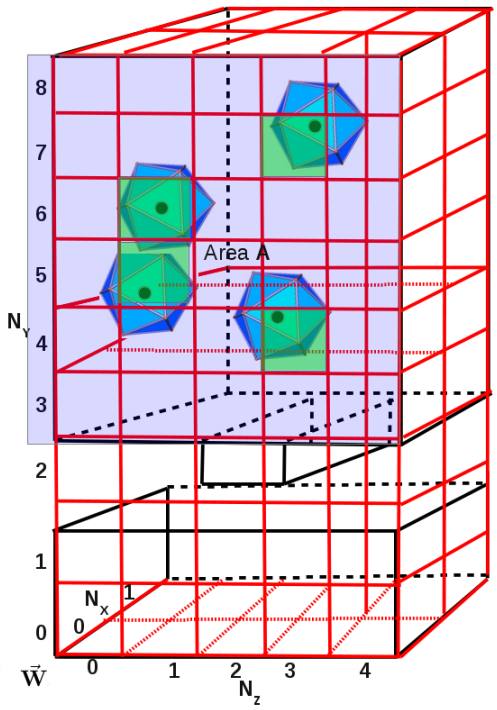

Figure 2.5: Broad phase collision detection grid.

To minimize memory costs we store each particle's grid position GP as a single integer value given by the mapping function "hashing":

$$\mathrm{P_{Hash}} = \mathrm{GP_{\mathbf{x}}} + \mathrm{GP_{\mathbf{z}}} \times \mathrm{Ncell_x} + (\mathrm{GP_{\mathbf{y}}} \times \mathrm{Ncell_z} \times \mathrm{Ncell_y}). \tag{2.1}$$

The mapping function we have chosen has the following properties:

1. Maps particles which are in the same grid cell to the same hash.

2. Maps particles that are close geometrically to each other, to hashes which numerically are close to each other.

3. The particle closest to the origin has the smallest hash and one furthest away the largest.

These properties allow for quick particle look-ups and improved caching. To determine potential contact pairs we only need to consider particles that are in the same cell or in the nearest neighboring cells relative to the particle of interest. Consider two adjacent cells **a** and **b**. Since we assign each particle to a thread and execute this in parallel, we cannot use the fact that cell **a** and **b** being neighbors are the same as cell **b** and **a,** which for serial calculations requires only 14 cells to be checked. On some parallel architectures it is possible to use atomic operations which allow different threads to write to the same location safely and thus exploit interaction symmetry typically only done in serial calculations. We also sort the dynamics information arrays for each particle described in Figure 2.4 according to the hash to improve memory coherence. We found a **thirty two times speed** up in run-time **when sorting** which includes sorting time. The size distribution primarily affects the cost of the broad phase neighbor search. If there is a very large size difference then the grid approach that we use is not computationally efficient to reduce the number of particles that need to be checked in detail for collision. Thus the feasible size ratio for our method is limited to between one and four. This size ratio represents granular media in hoppers and ball mills very well.

2.2.4 Narrow phase

While collision detection between spherical particles is resolved in the broad phase search, determining contact between polyhedra requires further checks as depicted in Figure 2.6 which we term narrow phase collision detection. This phase can account for as much as 70% of the total simulation time [20, 32, 36]. Thus Chapter 3 is devoted to the development of an efficient narrow phase collision detection algorithm for polyhedra. We use a multiphase heuristic approach that is based on the idea of a separating plane first described by Cundall [22] to determine if there is contact between convex polyhedra.

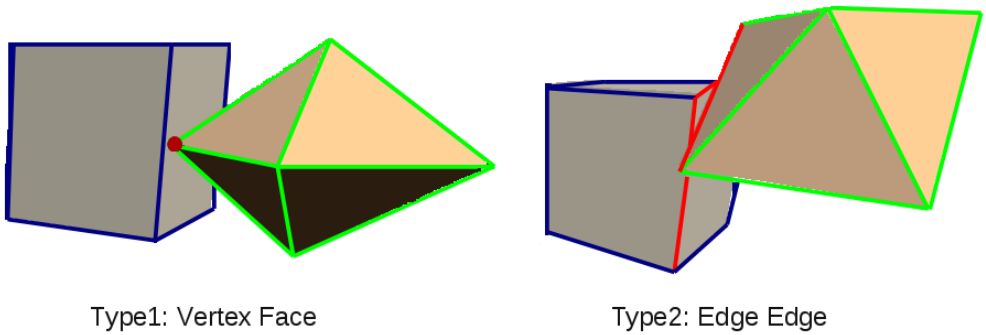

Figure 2.6: Polyhedra contact types.

If there is contact between two polyhedrons we obtain the point of contact $\mathbf{PC}(x, y, z)$, penetration distance δ and a normal direction $\mathbf{n}$. For Type 1 contact $\mathbf{n}$ is just the normal of the contacting face. For Type 2 contact if an edge is in contact with a face as depicted in Figure 2.6 $\mathbf{n}$ is also just the normal of the contacting face. For the case of two contacting edges there is no obvious choice for a normal [22]. In Section 3.2.4.1 we discuss our choice of normal for the case of edge-edge contact, which entails constructing a vector from the COM of each polyhedron to $\mathbf{PC}(x, y, z)$ and taking the average of the two vectors.

2.3 Contact Resolution

2.3.1 Force Calculations

The most common contact resolution model is the soft-sphere [24] approach of using the amount of inter-penetration between two contacting particles to determine a point force $\mathbf{F} = \mathbf{F}_{\mathrm{N}} + \mathbf{F}_{\mathrm{T}}$, as depicted in Figure 2.7.

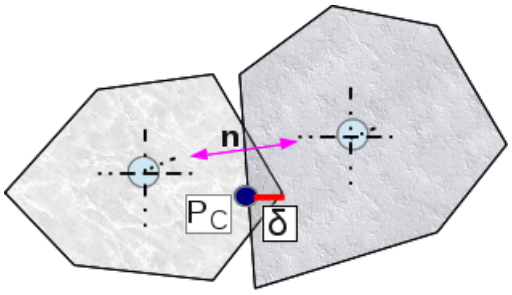

Figure 2.7: Force interaction model.

We use a linear spring to model the normal force as given by :

$$\mathbf{F}_{\mathrm{N}} = (\mathrm{K_n}\delta)\mathbf{n} - \mathrm{C_n}(\mathbf{V}_{\mathrm{R}} \cdot \mathbf{n})\mathbf{n} \tag{2.2}$$

where δ is the penetration depth, $\mathbf{V}_{\mathrm{R}} = \mathbf{V}_1 - \mathbf{V}_2$ is the relative translational velocity, $\mathrm{K_n}$

the spring stiffness, $C_n = \frac{2\ln(\epsilon)\sqrt{K_n m_{eff}}}{\sqrt{\ln(\epsilon)^2+\pi^2}}$ is the viscous damping coefficient, $\mathbf{n}$ the normal at contact, ϵ is the coefficient of restitution and $m_{eff} = (\frac{1}{m_1} + \frac{1}{m_2})^{-1}$ is the effective mass of the particles. There are two approaches to determine the tangential shear force, namely

(i) history independent models which require only knowledge of the current kinematic state and

(ii) history dependent models which require information about previous contacts.

Approach (i) is attractive as it is computationally cheap, but is limited in application. We limit ourselves to the history independent models as they are easily implemented on the GPU and is sufficient for the problems we wish to solve [24, 25].

The tangential friction force is given by :

$$\mathbf{F}_T = -\min\left[\mu \parallel \mathbf{F}_N \parallel, K_T \parallel \mathbf{V}_T \parallel\right] \left(\frac{\mathbf{V}_T}{\parallel \mathbf{V}_T \parallel}\right) \tag{2.3}$$

where $\mathbf{V}_T = (\mathbf{V}_R - (\mathbf{V}_R \cdot \mathbf{n})\mathbf{n}) + \mathbf{r}_1 \times \boldsymbol{\omega}_R - \mathbf{r}_2 \times \boldsymbol{\omega}_R$ is the relative tangential velocity at the contact point, $\boldsymbol{\omega}_R = \frac{1}{2}(\boldsymbol{\omega}_1 + \boldsymbol{\omega}_2)$ is the relative angular velocity, $\mathbf{r}$ is the vector from the COM to the contact point $\mathbf{PC}(x, y, z)$, μ the coefficient of dynamic friction and K_T the viscous damping coefficient. Note that the physically correct tangential velocity computation requires both $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$, by using $\boldsymbol{\omega}_R$ we save on a memory transaction. The values for μ and K_T [24] should be such that K_T affects oblique impacts while μ affects rolling/sliding contact.

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

$$\boldsymbol{\Gamma} = (\mathbf{r} \times \mathbf{F}). \tag{2.4}$$

2.3.2 Numerical Integration

We use a modified explicit velocity Verlet algorithm for larger time steps and the forward Euler for smaller time steps, since we compute using single precision. Using single precision $\triangle t^2$ (velocity Verlet) may cause numerical inaccuracies whereas forward Euler avoids $\triangle t^2$ and only computes $\triangle t$, to obtain the position $\mathbf{x}$ and velocity $\mathbf{V}$ of a particle i at time t:

$$\mathbf{x}_{t+1}{}^i = \left[\mathbf{x}_t^i + \mathbf{V}_t^i \triangle t + \frac{1}{2}\mathbf{a}_t^i \triangle t^2\right], \tag{2.5}$$

$$\mathbf{V}_{t+1} = \left[\mathbf{V}_t^i + \frac{1}{2}(\mathbf{a}_t^i + \mathbf{a}_{t-1}^i)\triangle t\right] \tag{2.6}$$

The acceleration $\mathbf{a}$ at time t is given by $\mathbf{a}_{\mathrm{t}} = \frac{\mathbf{F}_{\mathrm{t}}^{\mathbf{net}}}{\mathrm{m}}$ where $\mathbf{F}_{\mathrm{t}}^{\mathbf{net}} = \sum_{j=1}^{\mathrm{z}} \mathbf{F}_{\mathrm{t}}^{\mathrm{ij}}$ is the sum of all Z contact forces experienced by the particle. The classical velocity Verlet algorithim requires the computation of $\mathbf{a}_{\mathrm{t}+1}$, which would require an additional evaluation of the contact forces. This explicit time integration is expected to be stable when $\triangle \mathrm{t} < \frac{2}{\sqrt{\frac{\mathrm{K_n}}{m}}}$. 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 2.8 shows the error in the total energy of the system for various time-steps. We firstly see the error remains within a bound for all time-steps sampled and decreases with a smaller time-step as expected. The largest time step size of 10^{-4} s introduces an error of $\approx 1\%$ into the simulation.

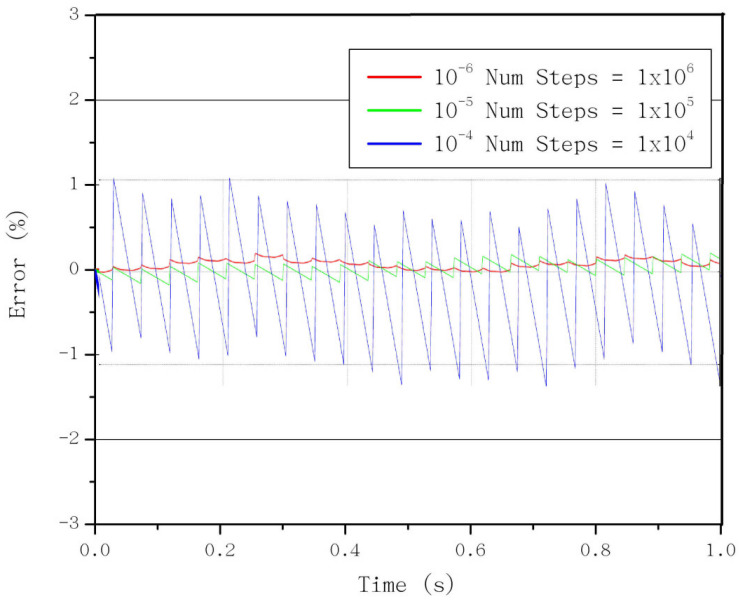

Figure 2.8: Error in Energy Conservation.

2.3.2.1 Angular Integration

The angular velocity $\boldsymbol{\omega}$ of particle i at time t is obtained using the forward Euler integration scheme:

$$\boldsymbol{\omega}_{\mathrm{t}+1} = \boldsymbol{\omega}_{\mathrm{t}} + \boldsymbol{\alpha}_{\mathrm{t}}^{\mathrm{ang}} \triangle \mathrm{t}. \tag{2.7}$$

The angular acceleration $\boldsymbol{\alpha}^{\mathrm{ang}}$ at time t is given by $\boldsymbol{\alpha}_{\mathrm{t}}^{\mathrm{ang}} = \mathbf{I}_{\mathrm{t}}^{-1} \boldsymbol{\Gamma}_{\mathrm{t}}^{\mathbf{net}}$ where $\boldsymbol{\Gamma}_{\mathrm{t}}^{\mathbf{net}} = \sum_{j=1}^{L} \boldsymbol{\Gamma}_{\mathbf{t}}^{\mathrm{ij}}$ is the sum of all L body contact torques experienced by particle i as given in Equation 2.4 and $\mathbf{I}_{\mathrm{t}}$ the inertia tensor at time t. 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 [38]. The orientation of a particle is represented by a unit quaternion $\mathbf{q}\{\mathrm{w}, \mathrm{x}, \mathrm{y}, \mathrm{z}\} = \{1, 0, 0, 0\}$, where w is an angle $[-1 : 1]$ and $(\mathrm{x}, \mathrm{y}, \mathrm{z})$ the axis of rotation. The relationship between a quaternion and axis angle representation (θ, x_1, y_1, z_1) is given

by:

$$\mathbf{q} = \{\cos(\theta/2),\ x_1\sin(\theta/2),\ y_1\sin(\theta/2),\ z_1\sin(\theta/2)\}. \tag{2.8}$$

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

$$\triangle\mathbf{q}_t = \{\cos(\|\boldsymbol{\omega}_t\|),\ \sin(\|\boldsymbol{\omega}_t\|)\frac{\boldsymbol{\omega}_t}{\|\boldsymbol{\omega}_t\|}\}, \tag{2.9}$$

The evolution of the angular orientation of the particle is just a multiplication [28] between the current quaternion $\mathbf{q}_{t-1}$ of a particle with $\triangle\mathbf{q}_t$:

$$\mathbf{q}_{t+1} = \mathbf{q}_{t-1} \times \triangle\mathbf{q}_t. \tag{2.10}$$

To verify that rotation using quaternions on the GPU is implemented correctly, we simulate a cube spinning with a constant velocity of $(1, 0, 0)$ rad.s^{-1} with a step-size of 10^{-5} s. Figure 2.9(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 1 s = 359.94406 degrees giving an error of 0.015% which is within error tolerances.

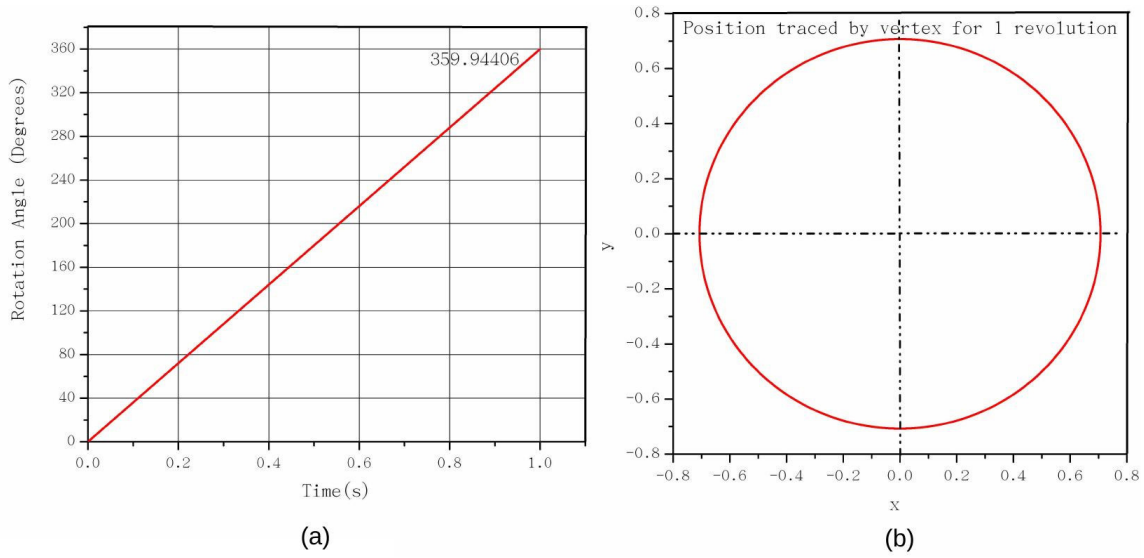

Figure 2.9: Results of angular integration depicting (a) angular displacement (rotation angle) as a function of time and (b) the position of a vertex on the rotating cube.

Figure 2.9(b) shows how the position of a vertex on the cube varies with time which gives an indication if any distortion from an expected circular path occurs. We see a perfect circular path which implies that there is no distortion occured. In Equation 2.10 as $\boldsymbol{\omega}_t \to 0$, $\triangle\mathbf{q} \to \{1, 0, 0, 0\}$ which leaves the orientation unchanged and hence does not result in numerical difficulties. Note that Anderson et al. [35] found no major difference in the round-off errors between CPU and GPU floating point operations.

2.4 Computational Implementation

We used two configurations of hardware for our simulations, which are listed in Table 2.1. The consumer grade workstation is a typical PC used by a scientist/engineer. The computing grade workstation has a TESLA computing graphics card which is dedicated for numerical computations.

Table 2.1: Hardware Specifications.

Consumer Grade		Computing Grade	
CPU	Intel i7 - 2.40 GHz (8 cores)	**CPU**	Intel i7 - 3.50 GHz (12 cores)
RAM	16 GB DDR3 - 1600 Mhz	**RAM**	32 GB DDR3 - 1600 MhZ
GPU	GTX 780M - 0.80GhZ (8 SM's)	**GPU**	TESLA K20 - 0.71GhZ (13 SM's)
VRAM	4GB GDDR5 - 2500 MhZ	**VRAM**	5GB GDDR5 - 2500 MhZ
HDD	120 GB SSD	**HDD**	120 GB SSD

2.4.1 Blaze-DEM framework.

In designing the framework for BLAZE-DEM we took the following features into consideration:

1. A modular environment that can be easily extended to simulate additional physics, such as fluid interactions.
2. A light-weight transparent class design than can be integrated easily into gaming and simulation environments.
3. A 3D graphics environment that is interactive and can display millions of polyhedra.
4. An interface between the numerical task computing and the DEM algorithm.
5. Portability to new architectures.

Figure 2.10 describes the BLAZE-DEM framework. The Data-Library is analogous to what is found in a typical computer game and allows complex simulation environments to be created using a combination of world and particles objects. The simulation information is stored in a text file:

1. Names of the world and particle objects.
2. Total number of particles for each particle object type.
3. Spatial location and orientation of particles.
4. Initial conditions and the values of the physical parameters.
5. Specification of the force models to be used for particle interaction.

6. Required statistics e.g. system energy and number of collisions.

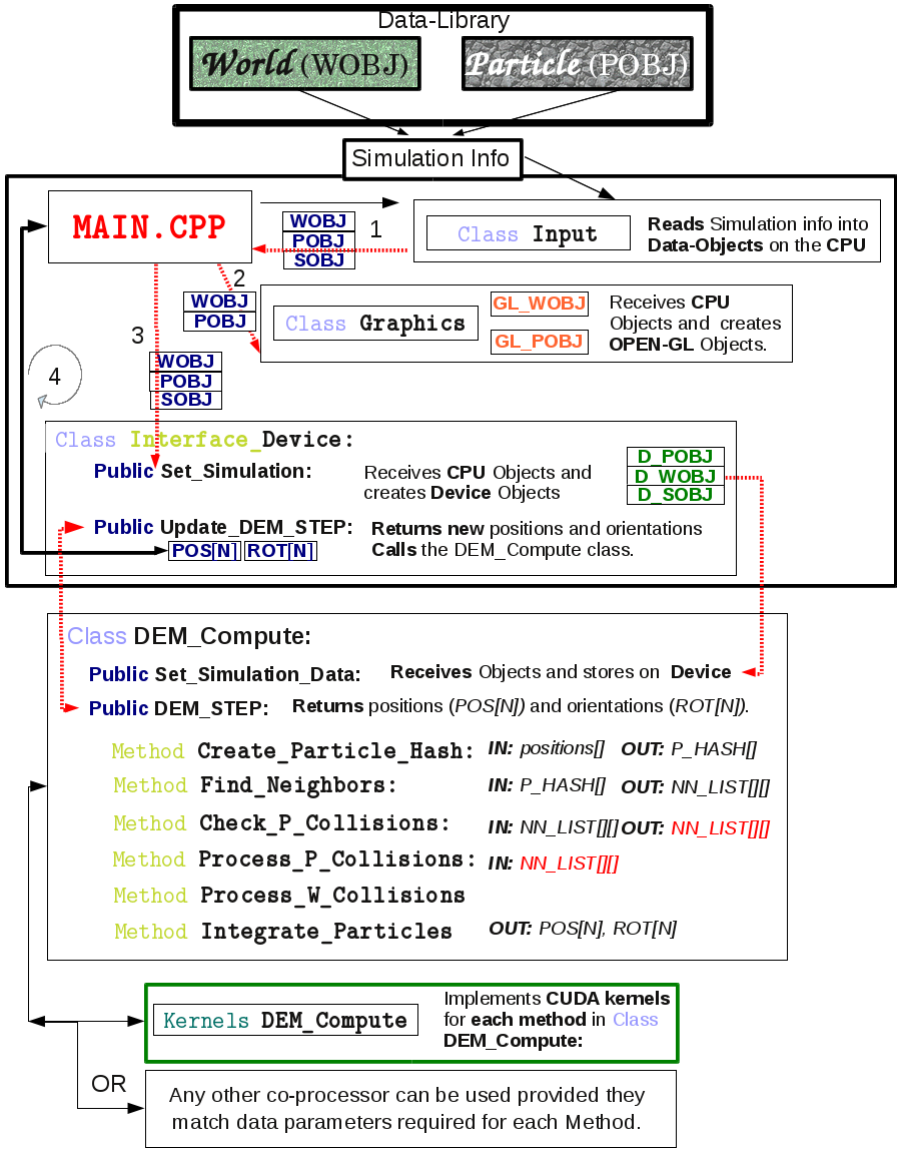

Figure 2.10: BLAZE-DEM Framework.

The code is object-orientated C++ using the OpenGL graphics library for visualization and QT for the graphical user interface. The *Input* class reads the simulation input and creates world, particle and simulation objects which are passed to the function *Main*. *Main* passes the world and particle objects to the graphics class which creates the required OpenGL graphics objects. After creation of the graphics objects, *Main* then passes all the objects to the *Set_Simulation* method in the *Interface* Class, which creates objects for the computing device which is currently only the GPU. A Kernel on the GPU stores the objects from the interface into GPU memory. Once the initial data objects are passed to the required classes, *Main* then makes a request to the *Interface* to advance the simulation and return new positions and orientations. The *Interface* then calls the

DEM_Compute class, which calls each method that in turn invokes a kernel which performs the computation on the GPU. The GPU implementation is separate from the CPU implementation with only communication between the interface. This allows us to easily do computations on different devices and implement new physics models, without having to change the entire code.

Algorithm 2.1 describes our GPU implementation. The argument in angle brackets launches **N** parallel threads on the GPU (no need to loop as in serial calculations). We use thread level parallelism, mapping a single particle to a single thread. The size of the blocks that we launch on the GPU is dependent on the problem size and GPU hardware used. We use the strategy of maximum SM occupancy so that the load is distributed evenly over all SMs on the GPU. At the start of the simulation we copy the initial data described in Figure 2.4 into GPU memory. No further memory transactions occur between the CPU and GPU, as we create a handle between OpenGL and the particle position and orientation GPU memory spaces. Each line (1-6) of Algorithm 2.1 is a CUDA kernel optimized for the Kepler architecture (see line comments for description).

Algorithm 2.1 Discrete Element GPU Implementation.

COPY (HOST-DEVICE): *dynamics data arrays* →**global memory** and *particle data arrays* →**constant memory** on the GPU.

1. **CalculateParticleHash** (**position_com**) <<N>> /* Calculate hash given by Equation 2.1*/
2. **SortParticleDynamics** (**p_hash[]**) <<N>> /* Sort the dynamics data arrays based on hash to improve memory access as discussed in Section 2.3.2 */
3. **Find_NN_Phase1** (**position_com**) <<N>> /* Create *NN_LIST[N][x]* containing **NumNN[N]** potentially contacting particles */
4. **Contact_Detection** (**position_com**) <<N>> /* Detailed contact detection as described in Section 3 */

```
for i=0 to NumNN [thread.id] do
  if particle i and thread.id are in contact.
      calculate force F^ij.
  end if
   F^i+ = F^ij
end for
update V^i,a^i
```

5. **Integrate_Position** (**position_com**, **velocity_com**, **accelrat_com**) <<N>> /* Numerical integration described in Section 2.3.2 */
6. **Check_WorldCollision** (**position_com**, **velocity_com**) <<N>> /* Ray-Trace between particle vertexes and world surfaces and resolve collisions */

2.5 Simulation examples with BLAZE-DEM

2.5.1 Numerical Verification of code

The numerical verification of a DEM code that simulates non-spherical 3D particle behavior is a complex matter and debugging is a non-trivial task as there are no standard tests that can be used to verify a model other than comparison with experimental or other

data [23]. Furthermore the debugging of GPU code is difficult and caution must be taken as C++ Object Orientation is not fully implemented on the GPU. To verify that the numerical integration on the GPU yields the expected results, we simulated the motion of a single particle falling under the effects of gravity and air resistance ($F = mg - \alpha v^2$), which is a 2nd order non-linear system and should be matched well by our numerical simulation. The analytical solutions for the velocity and position of the particle are $v(t) = \sqrt{\frac{mg}{\alpha}} \tanh\left(\frac{t}{\sqrt{\frac{m}{g\alpha}}}\right)$ and $x(t) = \frac{m}{\alpha} \ln\left[\cosh\left(\frac{t}{\sqrt{\frac{m}{g\alpha}}}\right)\right]$. Figure 2.11 shows the results with $\alpha = 0.5$. We see good agreement for both time-steps with the numerical error bound at 0.10% for a step size of 10^{-3} and 0.010% for a step size of 10^{-4}.

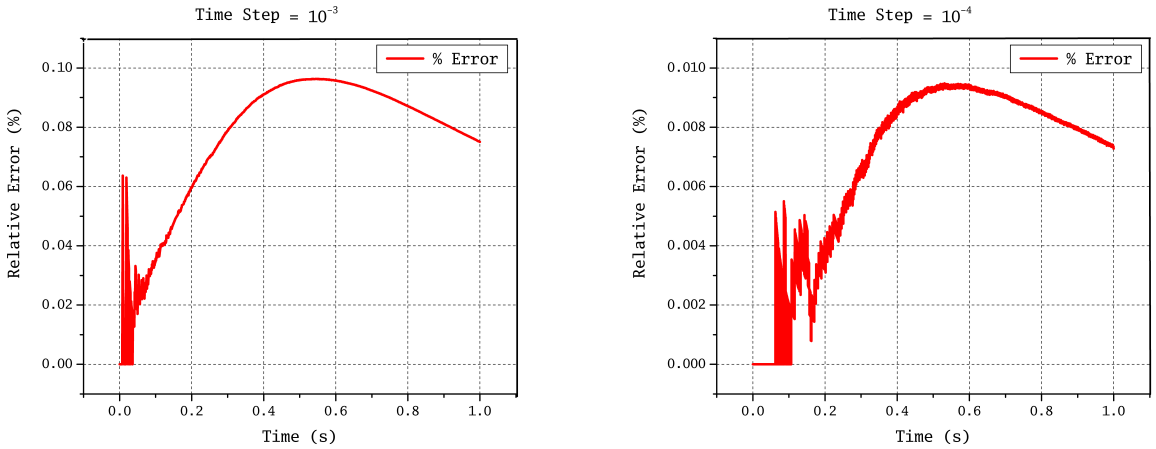

Figure 2.11: Verification of numerical integration scheme for different time steps.

2.5.2 Gravity Packing

To verify that the simulated bulk behavior of the system is satisfactory, we generated a grid of (128×128×64) polyhedra (0.001 cm spacing) with the grid starting at a height of 1 cm above the bottom of a (200×200×128) cm container. The particles have no initial velocity and fall under the influence of gravity into the container. Figure 2.12 shows the system at the start and after 1 second. This simulation tests the robustness of our contact detection algorithm and numerical stability as particles collide with each other and the world until they reach a final configuration at rest (quasi-static conditions). Simulating the gravity packing of particles have been used by numerous researchers [39] to test the robustness of algorithms in terms of numerical stability. Figure 2.13 shows a plot of the kinetic energy of the system over the duration of the simulation. We notice that the simulation does converge numerically as the energy decreases due to friction and damping in the system, which results from collisions between particles and particles and the container (world).

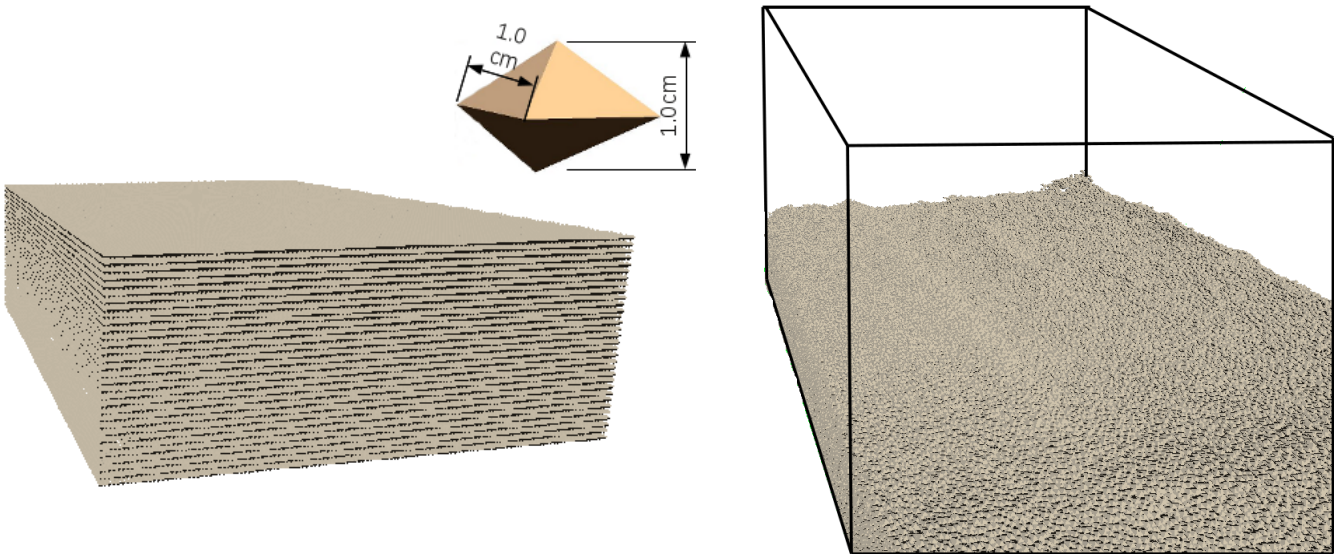

Figure 2.12: Gravity packing for 1 million polyhedra.

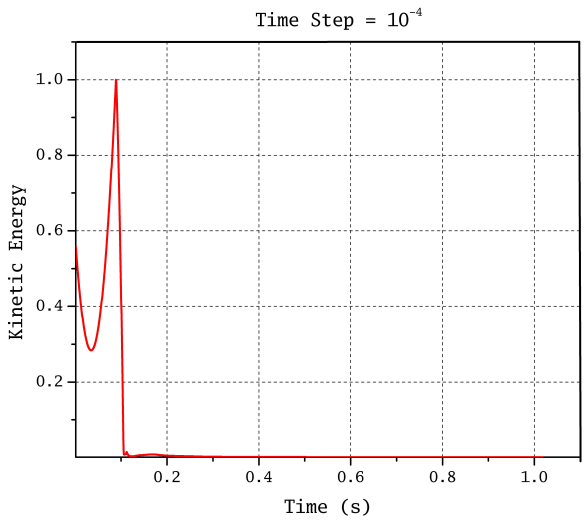

Figure 2.13: Energy dissipation of damped system packing under gravity (inital velocity of $1\,\mathrm{m.s^{-1}}$in all directions).

Figure 2.14 shows the scaling performance of the code as we increase the number of particles (8 face polyhedra) for the gravity packing simulation shown in Figure 2.12. We see that the computational time scales linearly with the number of particles N up to a million particles.

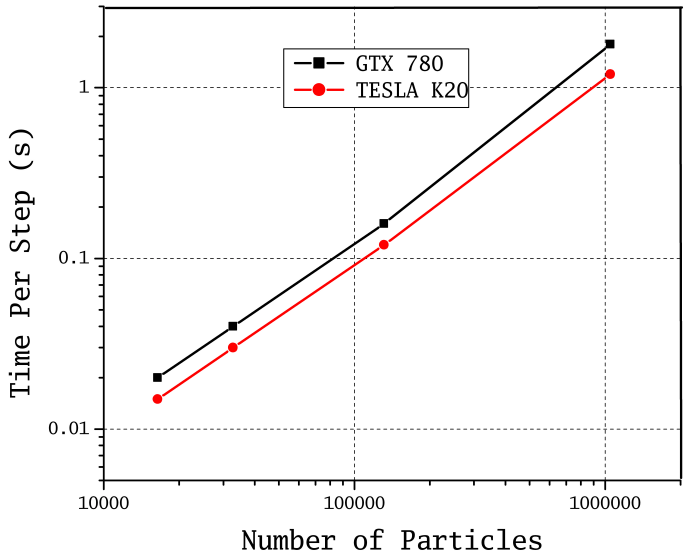

Figure 2.14: Performance scaling with number of particles on a Tesla K20 GPU for octahedra (8 face).

2.5.3 Hopper flow

In order to verify the suitability of the GPU based Blaze-DEM code for hopper simulation, we made comparisons to experiments conducted by Pizzete et al. [40, 41] using a lab-scale plexi-glass hopper with a discharge angle of $\beta = 90^o$ as depicted in Figure 2.15(a). Two thousand regular dodecahedron and spherical particles where used as depicted in Figure 2.15(b). They where packed in alternating colors as depicted in Figure 2.15(a) to observe the flow patterns. The hopper is filled by dropping particles down the center in order to obtain a random loose packing (we used the average of three runs). The friction coefficients between particles and particles with the boundaries was evaluated using a start angle experiment similar to that of Abriak et al. [42]. Note that for spheres, the test consists of a clump of three spheres to prevent rolling. The average frictional values was found to be $\mu_{particle} = 0.35$ and $\mu_{wall} = 0.30$. The average coefficient of restitution obtained by measuring the rebound height of a dropped particle was found to be $\epsilon = 0.4$. We used a spring stiffness of $K_n = 3.2 \times 10^5$ N.cm^{-1} with a time-step of 1×10^{-5} s. Note: The only difference between spheres and polyhedra is during collision detection, as spheres do not require the narrow-phase. Hence any differences is a result of the shape.

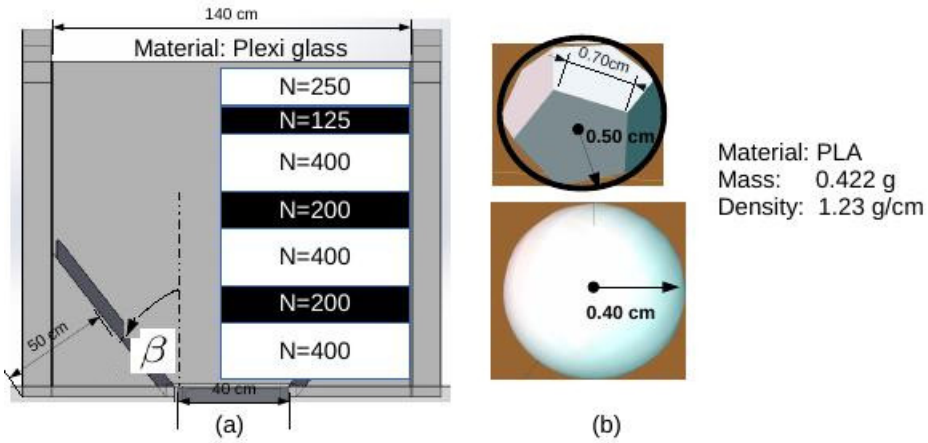

Figure 2.15: (a) Experimental setup indicating the number of particles N in each layer, discharge angle β and dimensions of the inlet and outlet, (b) particle specifications.

Figure 2.16(a) shows the flow rates and flow patterns for polyhedra obtained with DEM and experiment while Figure 2.16(b) shows the flow rates and patterns obtained for spheres with DEM and experiment. We see very good agreement with both spheres and polyhedra against experimental results. We notice that the hopper does not discharge completely as expected for both spheres and polyhedra, with spheres flowing much faster than the polyhedra, showing that particle shape affects flow rate. We also notice that a funnel flow regime (particles in the center discharge first and those in the sides last) is present in both DEM and experiment which further validates the code.

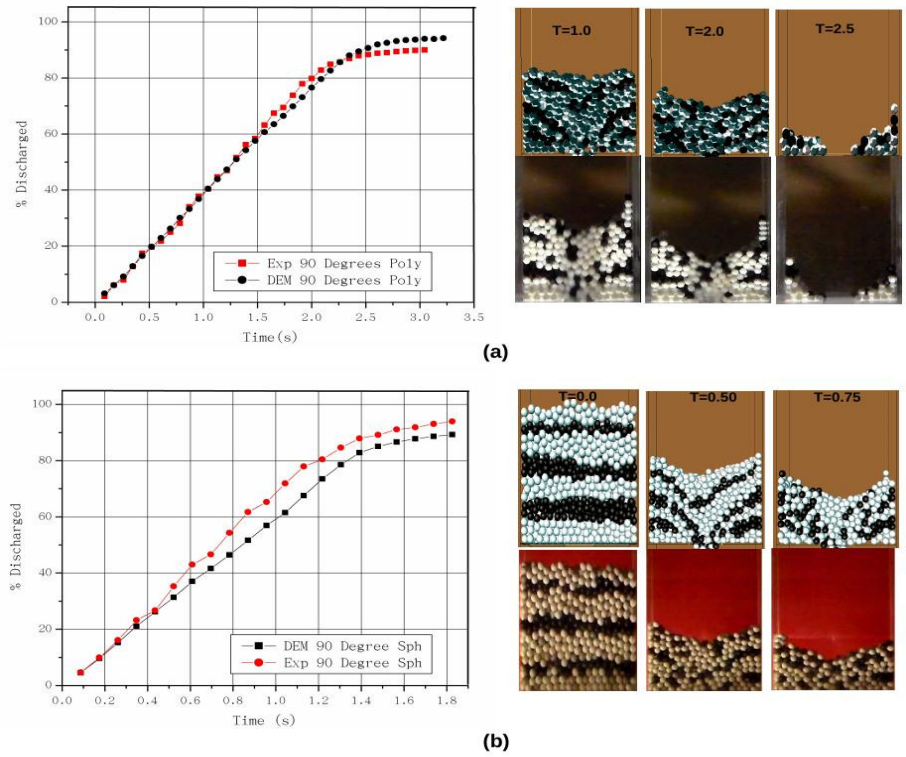

Figure 2.16: Flow rate and patterns for (a) polyhedra and (b) spheres.

Figure 2.17 shows corn shaped polyhedral particles flowing in a hopper. They are initially packed in a $(32 \times 27 \times 16)$ grid. We see that the polyhedra form a denser packing in the corners due to interlocking, as a result of their shape.

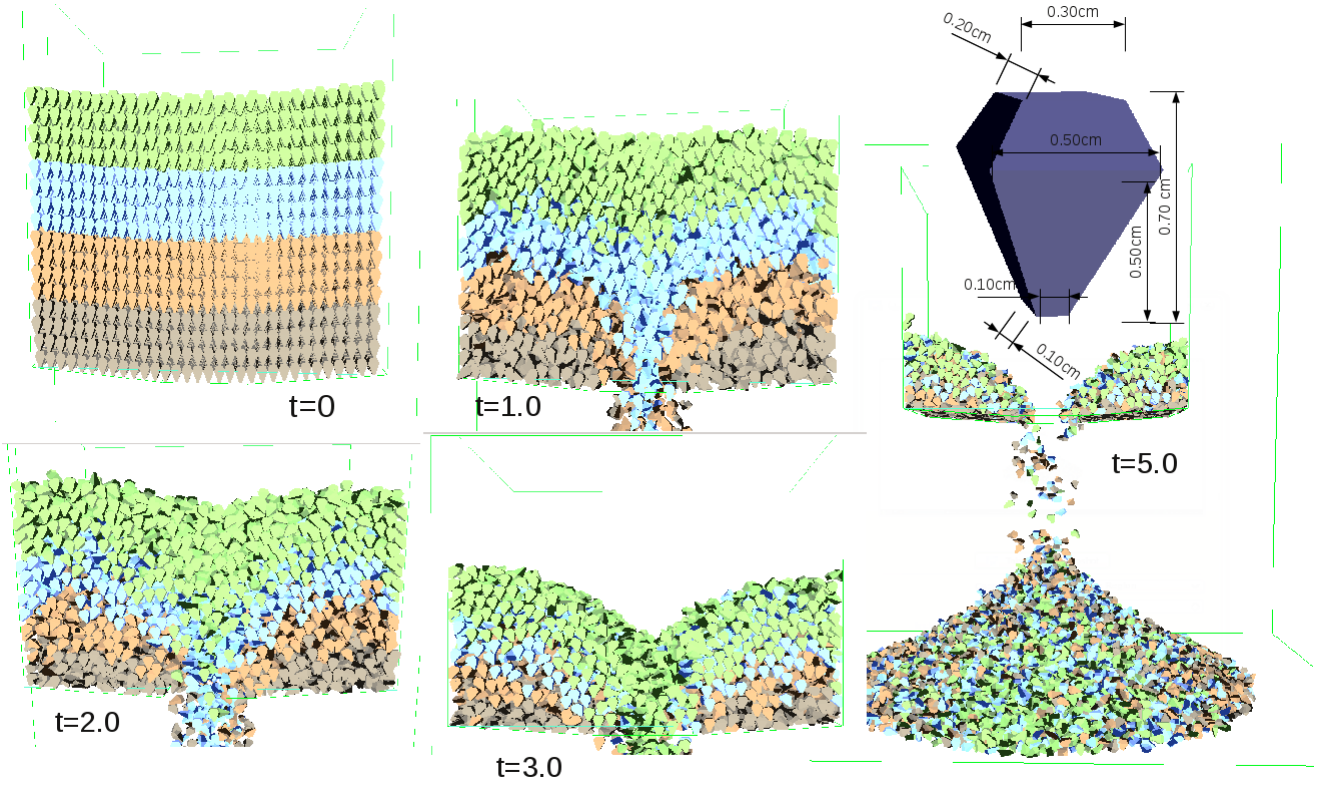

Figure 2.17: Hopper flow with 13824 corn shaped polyhedral particles.

Particle shape indeed has an effect on particle dynamics as polyhedra have the ability to restrict flow significantly compared to a spherical representation of particles as illustrated in Figure 2.18. This finding is consistent with other authors [14, 15, 20]. We achieved a frame rate of 167 FPS (0.006 s per step) with a step size of 10^{-4} s the simulation of 5 seconds required 5 minutes of computational time.

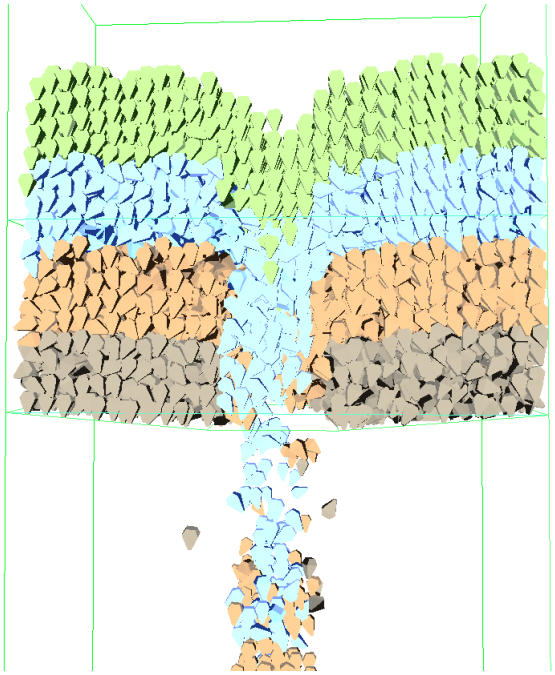

Figure 2.18: Polyhedra arching to restrict flow.

2.6 Conclusions

This chapter developed a computational framework for the discrete element modeling of convex polyhedral and spherical particles on the GPU. These developments are proposed to increase computational efficiency for large-scale granular material flow simulations where a simplified physics model is sufficient. Several simulations are presented to demonstrate the numerical stability and performance of Blaze-DEM. We conclude that the GPU is well suited to DEM simulations where a simple physics model can be used to expose parallelism in DEM, albeit to limited applications. In the next chapter we discuss the details of the collision detection algorithms that are utilized by the Blaze-DEM framework.

CHAPTER 3

Collision detection of convex polyhedra on the GPU architecture for the DEM

3.1 Introduction

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 polyhedra and the world surfaces is computationally expensive. This chapter demonstrates the significant computational benefits that the graphical processor unit (GPU) offers DEM for collision detection. As we show, this requires careful consideration due to the architectural differences between CPU and GPU platforms. This chapter 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.

3.2 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 [43]. We use a highly efficient hashed grid approach that executes in parallel on the GPU [44], as discussed in Chapter 2. 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 penetration distance [18], or a contact volume [36] can be computed to determine the contact forces.

Several approaches for narrow phase polyhedral collision detection can be found in literature [18, 32, 36]. However, these algorithms require either complex geometrical or algebraic operations with some requiring iterative procedures, as discussed by Nassauer and Liedke [36]. 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 with minimal storage requirements. To achieve this we need to consider both the particle and world representations.

3.2.1 World Representation

A common approach in DEM simulations is to model the surfaces of the environment as actual particles [23]. 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 [44], 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 3.1(a), or a combination of objects, as illustrated Figure 3.1(b), which shows a drum (labeled 1) and blades (labeled 2 and 3).

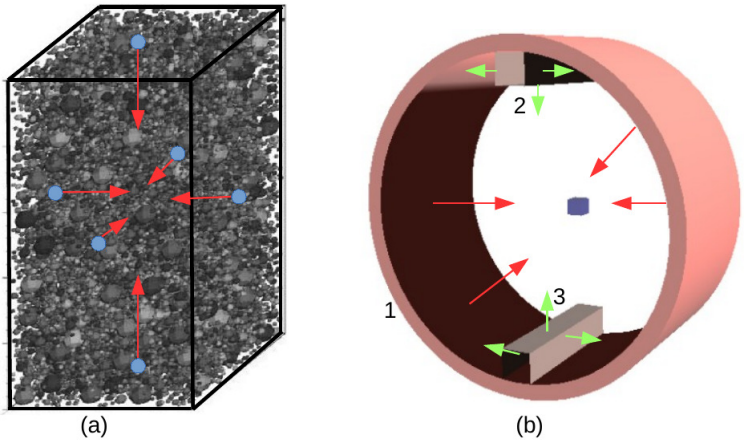

Figure 3.1: World Object Types.

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

1. Internal: all surface normals point inwards (Figure 3.1(a) and label 1 in Figure 3.1(b)).
2. External: surface normals can have any orientation (labels 2 and 3 in Figure 3.1(b)) .

This distinction allows us to reduce the computational cost of collision detection between a polyhedron and the world object. Consider Figure 3.2, 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 3.27). 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 3.1(b), in which we model the blades as external objects and the drum as a cylinder, which is an internal object.

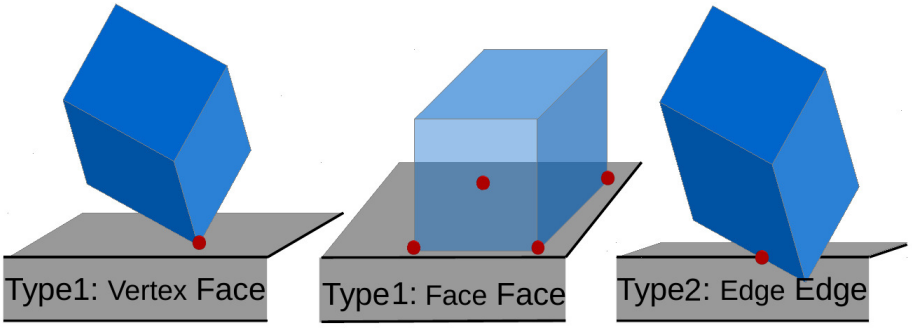

Figure 3.2: Polyhedra surface contact types.

3.2.2 Theoretical Formulation

The three scenarios depicted in Figure 3.2, 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:

$$d = \mathbf{n} \cdot (\mathbf{v} - \mathbf{c}) \tag{3.1}$$

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 3.3(a) represents a scenario where there is clearly no penetration between $\mathcal{A}$ and S. For this case, we have $d_i > 0$ for $i = 1, 2, ..., 6$. Figure 3.3(b) depicts part of $\mathcal{A}$ penetrating S, for which case we have $d_i < 0$ for $i = 1, 2, 3$ and $d_i > 0$ for $i = 4, 5, 6$. Figure 3.3(c) depicts $\mathcal{A}$ being completely past S, for which case we have $d_i < 0$ 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 $d_i < 0$, 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 $d_i > 0$ for $i = 1, 2, ..., n$ (Equation 3.1) between all vertexes $\mathbf{v}_i$ 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.

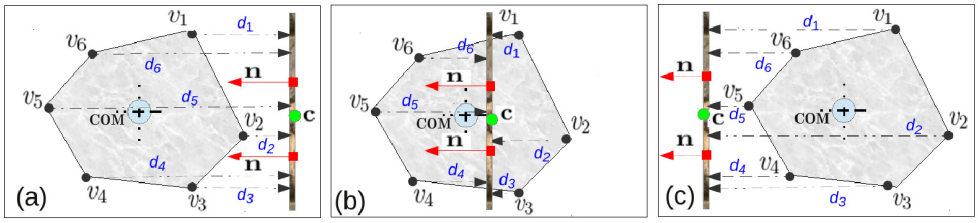

Figure 3.3: Illustration of Theorem 1 in 2D.

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 [18, 22]. However, they involve finding a SP by an iterative procedure, often minimizing a distance function [18], 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 [18, 32, 36]. It is computationally more efficient to do additional arithmetic operations rather than additional memory transactions on the GPU.

3.2.3 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 3.1 to the vertex set $\mathbf{v}_j^{\mathcal{B}}$ *for* $j = 1, 2, ..., k$ and half-planes $P_i^{\mathcal{A}}(x, y, z) = (\mathbf{n}_\mathrm{i}, \mathbf{c}_\mathrm{i})$ *for* $i = 1, 2, ..., m$ and the vertex set $\mathbf{v}_i^{\mathcal{A}}$ *for* $i = 1, 2, ..., l$ and half-planes $P_j^{\mathcal{B}}(x, y, z) = (\mathbf{n}_\mathrm{j}, \mathbf{c}_\mathrm{j})$ *for* $j = 1, 2, ..., u$ 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.

3.2.3.1 Edge-edge contact detection

For Type 2 contact we do not search for a SP. We rather search for penetration between two edges. This is much cheaper computationally than searching for a SP between edges. Consider Figure 3.4(a) which depicts a typical scenario that will be a candidate for edge contact and Figure 3.4(b) which depicts edge contact.

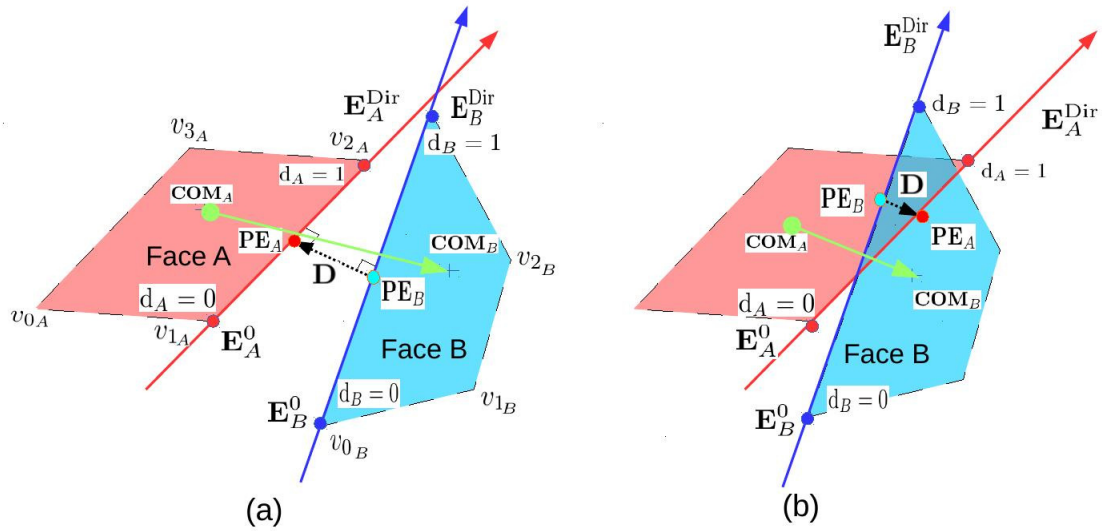

Figure 3.4: (a) Non penetrating Type 2 contact, (b) Penetrating Type 2 contact.

From Figure 3.4(a) we use the parametric equations to represent the edges:

$$\mathbf{PE}_A = \mathbf{E}_A^0 + \mathrm{d}_A \mathbf{E}_A^{\text{Dir}} \tag{3.2}$$

$$\mathbf{PE}_B = \mathbf{E}_B^0 + \mathrm{d}_B \mathbf{E}_B^{\text{Dir}} \tag{3.3}$$

where $\mathbf{E}_{\text{K}}^0$ is a vertex $\mathbf{v}_i$ on the edge, $\mathbf{E}_{\mathbf{K}}^{\text{Dir}} = \mathbf{v}_j - \mathbf{v}_i$ 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 perpendicular to both edges as indicated in Figure 3.4(a).

$$\mathrm{d}_A = \mathrm{J}[(\mathbf{E}_B^{\text{Dir}} \cdot \mathbf{E}_B^{\text{Dir}})(\mathbf{E}_A^{\text{Dir}} \cdot (\mathbf{E}_A^0 - \mathbf{E}_B^0)) - (\mathbf{E}_A^{\text{Dir}} \cdot \mathbf{E}_B^{\text{Dir}})(\mathbf{E}_B^{\text{Dir}} \cdot (\mathbf{E}_A^0 - \mathbf{E}_B^0))] \tag{3.4}$$

$$d_B = J[(\mathbf{E}_A^{Dir} \cdot \mathbf{E}_B^{Dir})(\mathbf{E}_A^{Dir} \cdot (\mathbf{E}_A^0 - \mathbf{E}_B^0)) - (\mathbf{E}_A^{Dir} \cdot \mathbf{E}_A^{Dir})(\mathbf{E}_B^{Dir} \cdot (\mathbf{E}_A^0 - \mathbf{E}_B^0))] \tag{3.5}$$

where $J = 1/[(\mathbf{E}_A^{Dir} \cdot \mathbf{E}_B^{Dir})(\mathbf{E}_A^{Dir}(\mathbf{E}_A^0 \cdot \mathbf{E}_B^0)) - (\mathbf{E}_A^{Dir} \cdot \mathbf{E}_A^{Dir})(\mathbf{E}_B^{Dir} \cdot \mathbf{E}_B^{Dir})]$. Edge to edge contact exists when the solved d_A and d_B are both between 0 and 1. Figure 3.5 describes our algorithm for polyhedra contact detection.

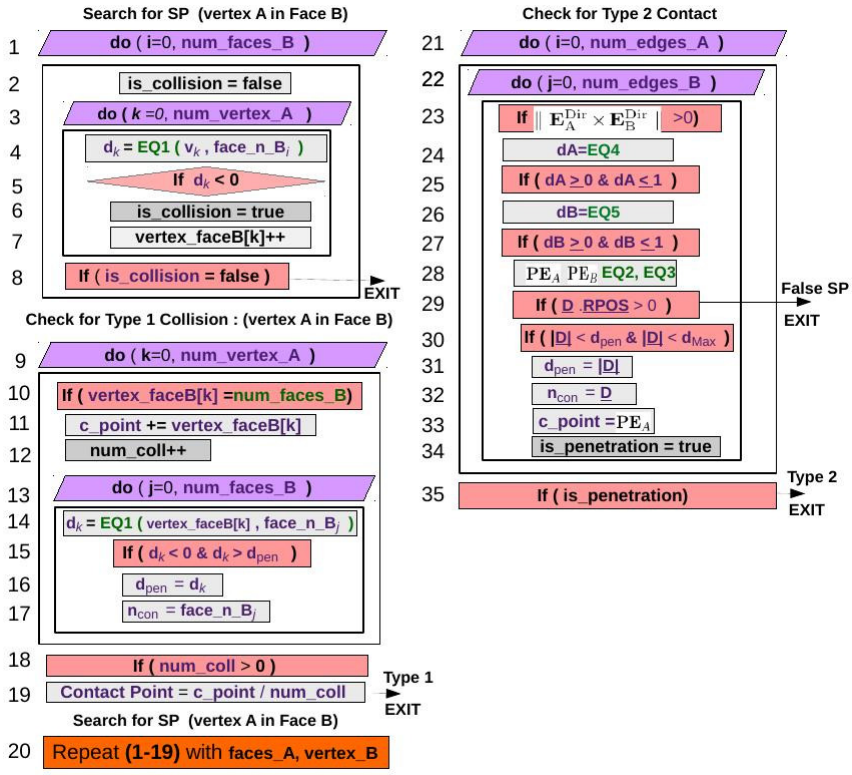

Figure 3.5: Algorithm 1: Polyhedron-polyhedron contact detection.

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 $\mathcal{A}$ and line 4 is the application of Equation 3.1 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 as described in Section 2.2.

3.2.4 World-Polyhedron Contact

Equation 3.1 treats a surface as an infinite plane. We are however interested in finite surfaces, as illustrated in Figure 3.6(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 3.6(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.

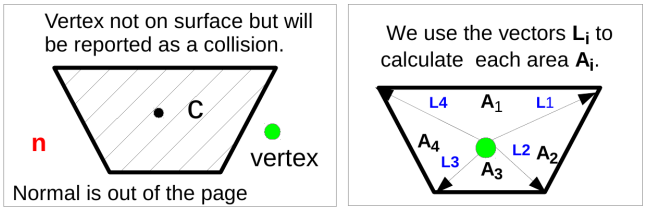

Figure 3.6: Check for surface collisions.

We summarize this result as:

> **Theorem 2:** If the $\perp$ distance $(d_i, i = 1, 2, ..., k) > 0$ (Equation 3.1) 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 3.7 describes our contact detection algorithm for polyhedron-planar surface contact.

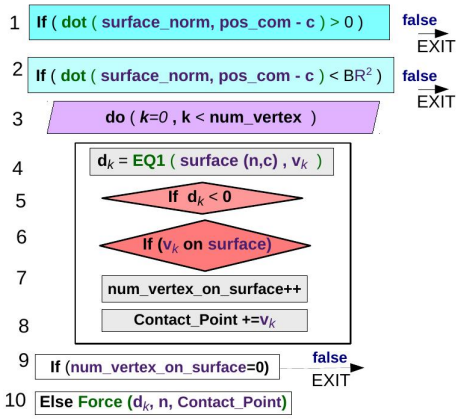

Figure 3.7: Algorithm 2: Polyhedron-world planar surface contact.

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 3.1 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 average d_i for all penetrating vertices. For spherical particles only lines 1-2 are required as the bounding sphere is the actual radius of the spherical particle.

3.3 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 3.8(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 two particles. Figure 3.8(b) depicts the scenario we simulate in this chapter. Note: the cube is stationary with the octahedron given an initial velocity towards the cube which we denote "downwards".

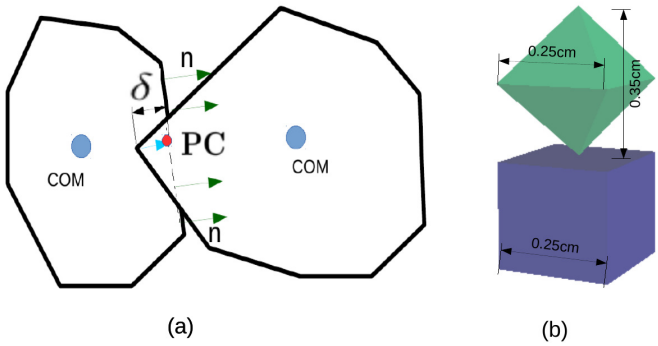

Figure 3.8: (a) Polyhedral particle contact model and (b) Simulation scenario for Section 3.3

3.3.1 Normal Force

3.3.1.1 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.

$$\mathbf{F}_{\mathrm{N}}^{\mathrm{elastic}} = (\mathrm{K_n}\delta^{\frac{3}{2}})\mathbf{n} \tag{3.6}$$

where $\delta = d_{pen}$ is the penetration depth and $\mathbf{n}$ the contact normal. The spring stiffness $\mathrm{K_n}$ can be chosen using the material properties as defined by Bell et al. [24] 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. The bounding radius for the simulations depicted in Figures (3.9-3.17) is 0.25 cm. The targeted maximum allowed penetration depth is 5% of the radius which is 0.0125 cm. Should the value be exceeded the user is warned in the log file.

Figure 3.9 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 3.6. This information can be used to tune $\mathrm{K_n}$ 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.

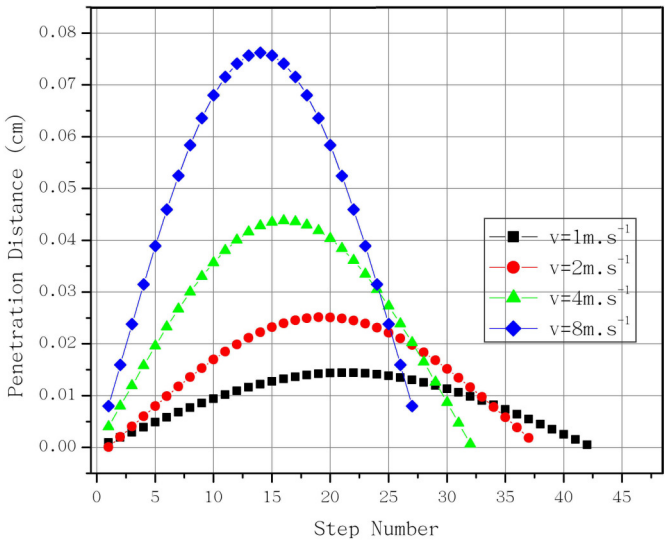

Figure 3.9: Penetration distance as a function of velocity for a head-on elastic collision. ($\mathrm{K_n} = 1 \times 10^9$ N.cm^{-1}, step size $=10^{-5}$ s)

Figure 3.10 shows how $\mathbf{F}_{\mathrm{N}}^{\mathrm{elastic}}$ varies as a function of the penetration depth for an incident velocity of $\mathbf{V} = 2$ 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 symmetric which further indicates that our algorithms are implemented correctly.

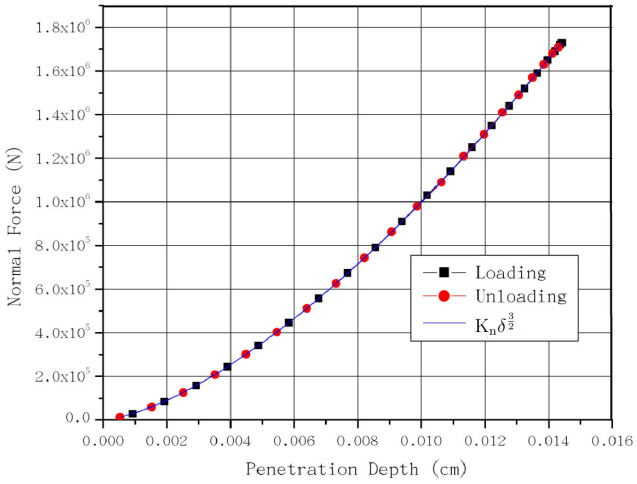

Figure 3.10: $\mathbf{F}_{\mathrm{N}}^{\mathrm{elastic}}$ as function of penetration depth for a head-on elastic collision. ($K_n = 1 \times 10^9$ N.cm^{-1}, step size $=10^{-5}$ s)

An important test of our contact detection and resolution algorithms is that for a head-on elastic collision kinetic energy must be conserved. Figure 3.11 shows how the velocity varies during contact. We see that for all incident velocities the pre and post collision velocities are the same.

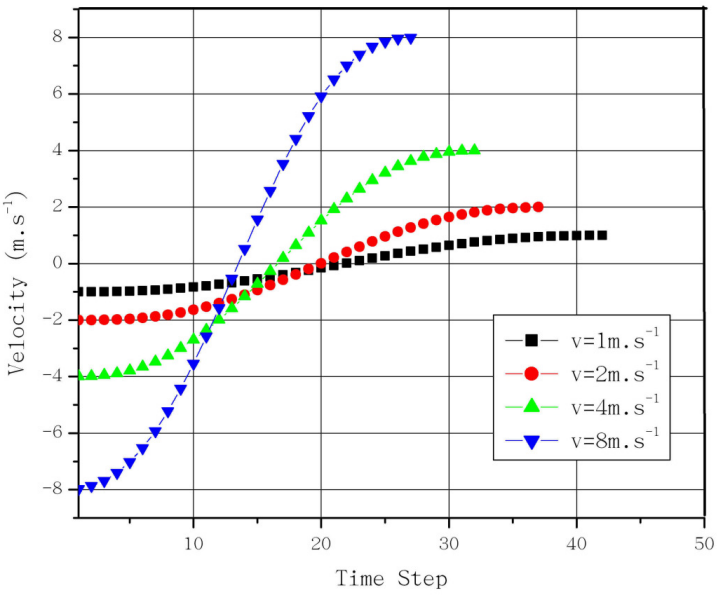

Figure 3.11: Velocity as a function of time step for a head-on elastic collision. ($K_n = 1 \times 10^9$ N.cm^{-1}, step size $=10^{-5}$ s)

Table 3.1 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 3.1: Analysis of particle velocities for a head-on elastic collision.

$\mathbf{V}(\mathrm{m.s}^{-1})$	$\mathbf{V}'(\mathrm{m.s}^{-1})$	% 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

3.3.1.2 Dissipative Force

Dissipation during contact is given by:

$$\mathbf{F}_{\mathrm{N}}^{\mathrm{diss}} = -\mathrm{K_D}\delta^{\alpha}\mathbf{V}_{\mathrm{R}}^{normal}, \tag{3.7}$$

where $\mathrm{K_D}$ is the damping coefficient and $\mathbf{V}_{\mathrm{R}}^{normal} = ((\mathbf{V}_1 - \mathbf{V}_2)\cdot\mathbf{n})\mathbf{n}$ is the relative normal translational velocity. $\alpha = \frac{1}{2}$ was found to match experimental data the best by Bell et al. [24], as the energy dissipation increases with increasing impact velocity. Figure 3.12 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 [36]. 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 [24, 25] 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 3.3.1.3.

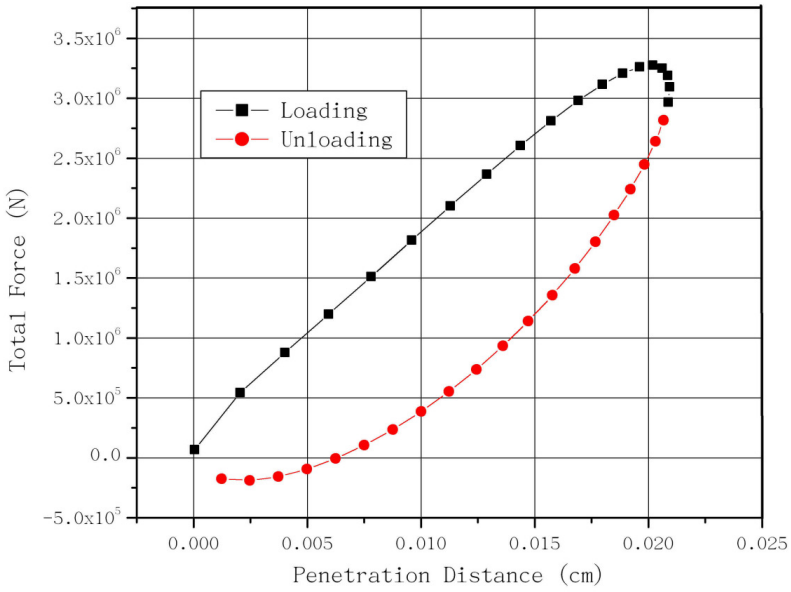

Figure 3.12: Normal force as a function of penetration depth for a head-on in-elastic collision. ($K_n = 1 \times 10^9$ N.cm^{-1}, $K_D = 5 \times 10^4$ kg.s^{-1}, step size $=10^{-5}$ s , $\mathbf{V} = 2$ m.s^{-1})

Figure 3.13 shows the difference between $\| \mathbf{F}_N^{elastic} \|$ and $\| \mathbf{F}_N^{diss} \|$ as a function of velocity (vertical lines indicate the onset of contact). We firstly see that for high initial velocity collisions that $\mathbf{F}_N^{diss}$ dominates at the start and end of the collision, resulting in a large dissipation of energy. This is consistent with experimental data [24] and explains the negative total force in Figure 3.12 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.

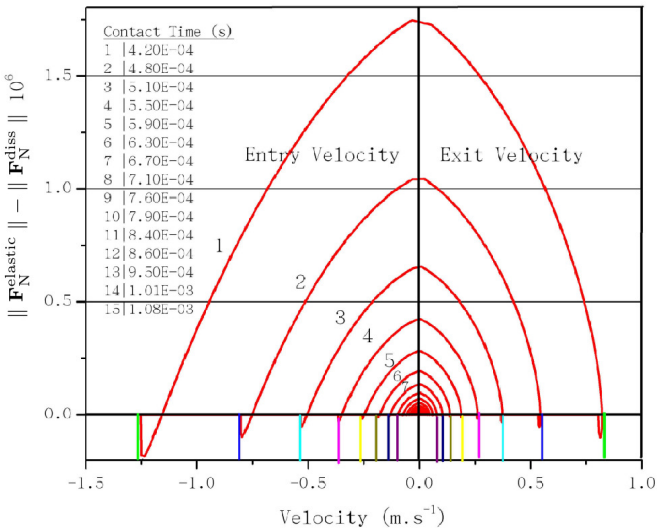

Figure 3.13: $\| \mathbf{F}_N^{elastic} \| - \| \mathbf{F}_N^{diss} \|$ vs velocity ($K_n = 1 \times 10^9$ N.cm^{-1}, $K_D = 5 \times 10^4$ kg.s^{-1}, step size $=10^{-5}$ s).

3.3.1.3 Contact Evaluation for particle in free fall.

Figure 3.14 shows the evolution of the position for a cube dropped vertically on its face undergoing inelastic contact with a surface. We see the particle undergoing a number of contacts with the surface before reaching an equilibrium as expected.

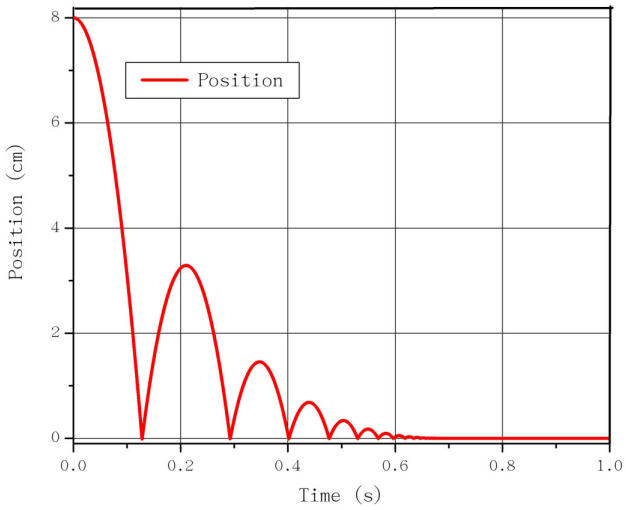

Figure 3.14: Position vs Time for face contact ($K_n = 1 \times 10^9$ N.cm^{-1}, $K_D = 5 \times 10^4$ kg.s^{-1}, step size $=10^{-5}$ s).

Figure 3.15 shows the corresponding velocity for Figure 3.14. We see that the velocity also reaches an equilibrium as expected.

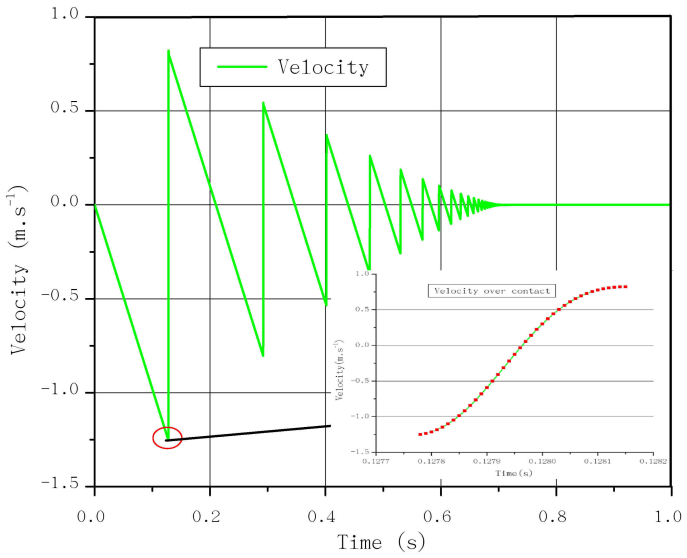

Figure 3.15: Velocity vs Time ($K_n = 1 \times 10^9$ N.cm^{-1}, $K_D = 5 \times 10^4$ kg.s^{-1}, step size $=10^{-5}$ s).

3.3.2 Tangential Force

The same tangential force model as discussed in Section 2.3.1 is used. To illustrate the effect of the tangential force we simulate the motion of the octagon shaped particle with an initial velocity of $2\,\mathrm{m.s^{-1}}$ in the tangential direction and $-0.5\,\mathrm{m.s^{-1}}$ in the normal direction ("downwards") in a gravity field initially placed 0.75 cm above the surface (no rotation). Figure 3.16 shows the evolving particle position and tangential velocity with time. In Figure 3.16 (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 3.16 (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.

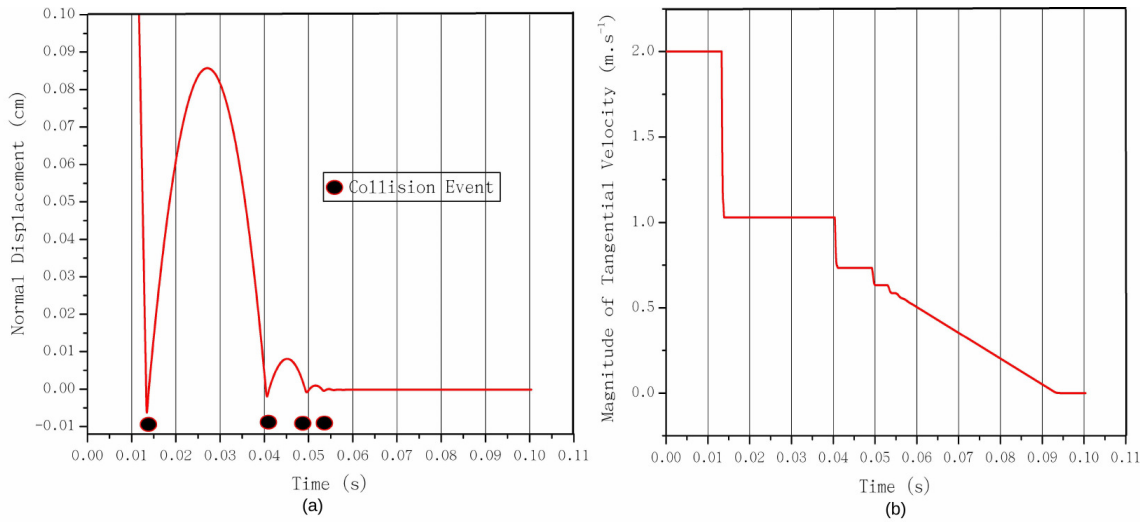

Figure 3.16: (a) Position vs Time and (b) Velocity vs Time ($K_n = 1\times10^9$ N.cm^{-1}, $K_D = 2\times10^5$ kg.s^{-1}, $\mu = 1.54$, $K_T = 4 \times 10^3$ kg.cm.s^{-1}, step size $=10^{-5}$ s).

Figure 3.17 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 $\mu \| \mathbf{F}_N \|$ dominates.

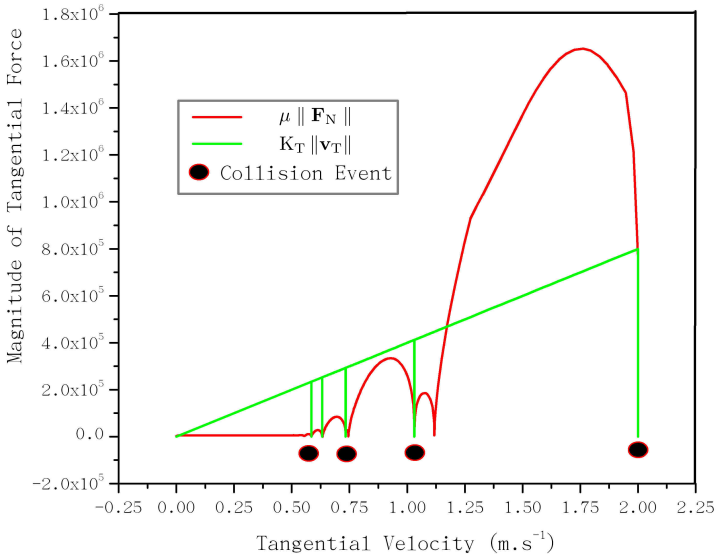

Figure 3.17: $\| \mathbf{F}_T \|$vs Tangential Velocity ($K_n = 1 \times 10^9$ N.cm^{-1}, $K_D = 2 \times 10^5$ kg.s^{-1},$\mu = 1.54$, $K_T = 4 \times 10^3$ kg.cm.s^{-1}, step size $=10^{-5}$ s).

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

$$\mathbf{\Gamma} = (\mathbf{r} \times \mathbf{F}_{CP}) \tag{3.8}$$

where $\mathbf{r}$ is the vector from the COM to the contact point $\mathbf{PC}(x, y, z)$ and $\mathbf{F}_{CP}$ the force exerted onto the particle due to contact.

3.4 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 3.2.

Table 3.2: Parameters used in simulations for non-linear force model.

Parameter	$\triangle t$	K_n (N.cm^{-1})	K_D (kg.s^{-1})	μ	K_T (kg.cm.s^{-1})
Value	10^{-6}	1×10^8	5×10^3	0.154	4×10^2

3.4.1 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 3.18. Notice that all the cubes are given an initial velocity of 0.2 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 a qualitative indication that our algorithms correctly detect collisions.

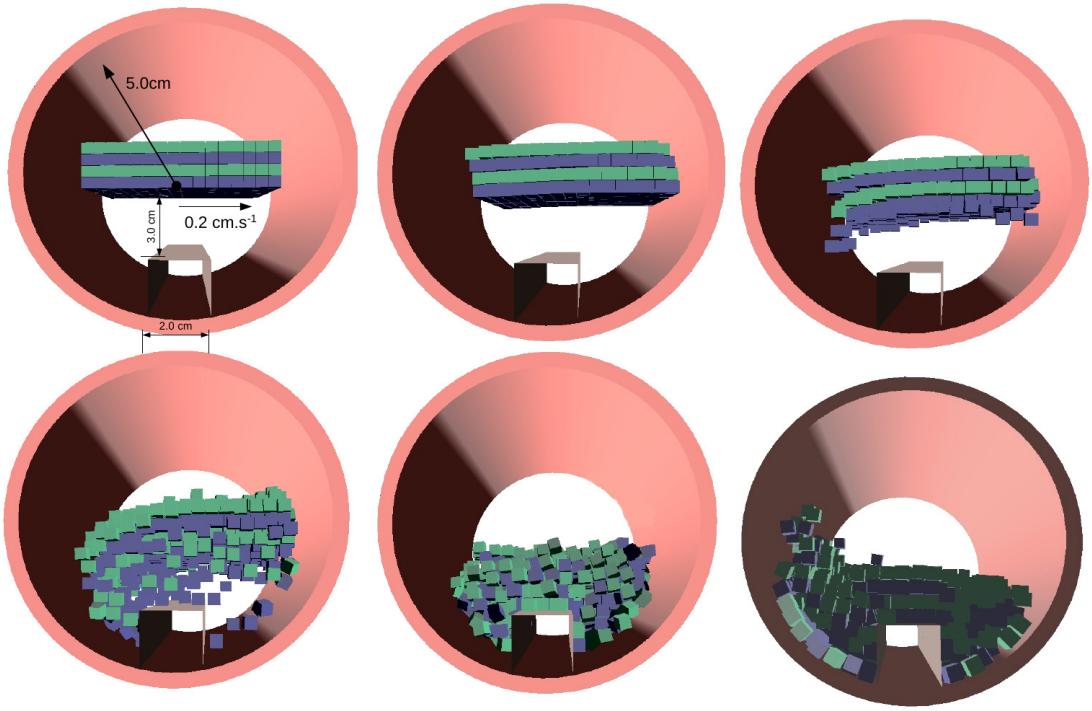

Figure 3.18: Dynamic packing of cubes in a drum.

In Figure 3.19 the cubes are arranged symmetrically in the drum with an initial height of 1 cm 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.

3.4.2 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 3.20.

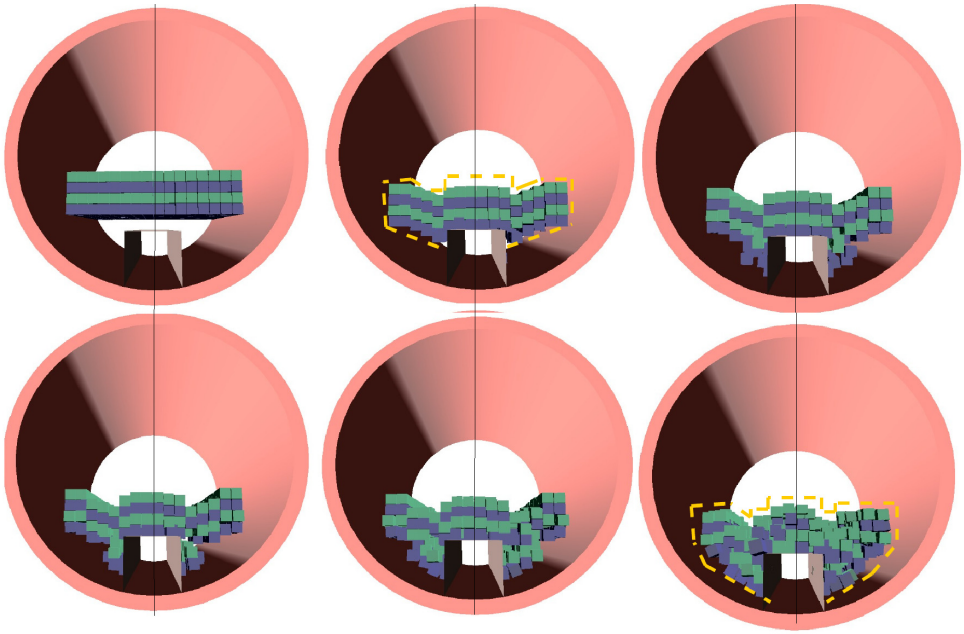

Figure 3.19: Packing of cubes in a drum (symmetric).

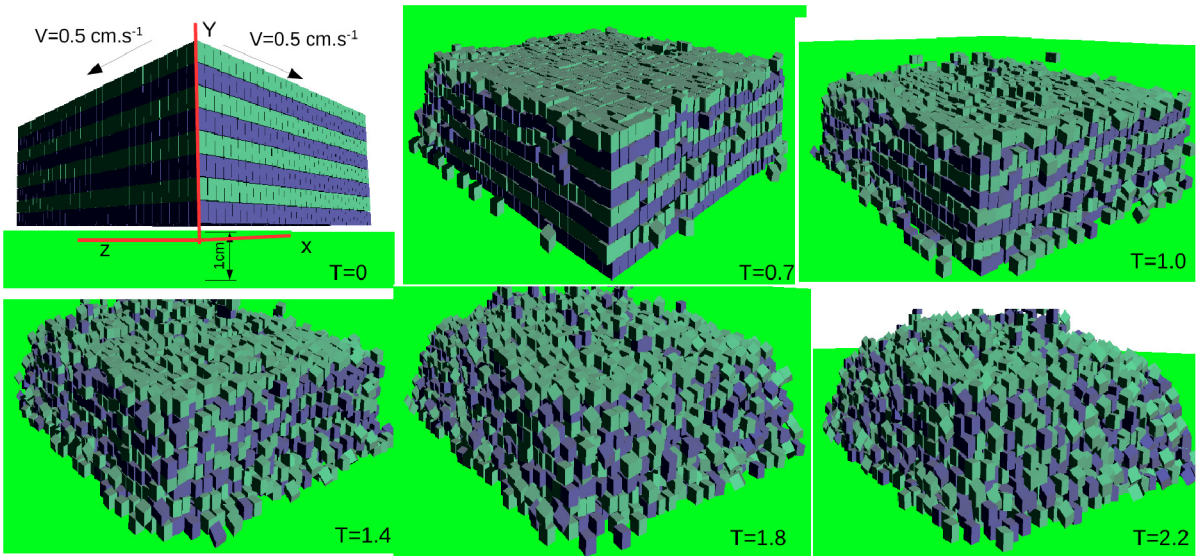

Figure 3.20: Gravity packing simulation for performance benchmarking.

Figure 3.21 shows the scaling of our polyhedra-world planar surface contact 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.

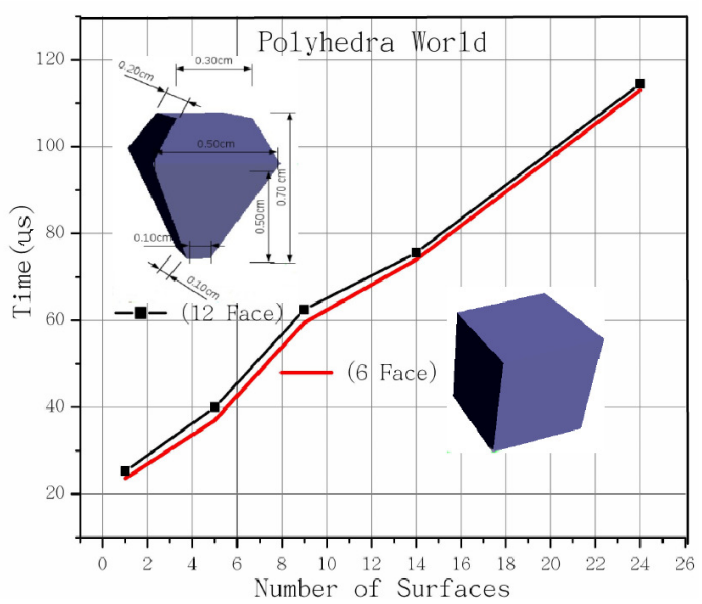

Figure 3.21: Computational scaling of polyhedra-world surface contact collision detection with world surfaces.

Figure 3.22 shows the scaling of the polyhedra-world planar surface contact 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.

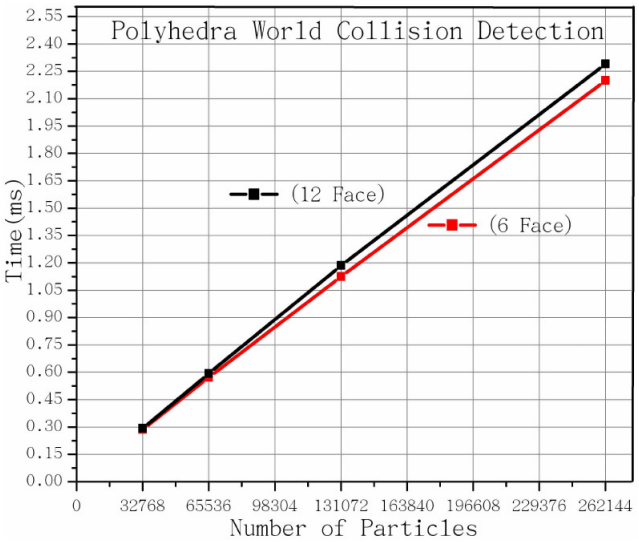

Figure 3.22: Effect of particle shape (number of faces) on the computational cost of collision detection.

Figure 3.23 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.

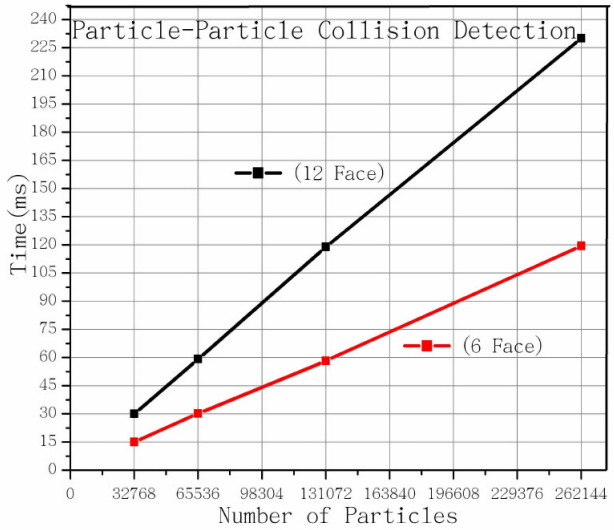

Figure 3.23: Scaling of polyhedron-polyhedron collision detection with number of particles (N) for different polyhedra.

3.4.3 Algorithm performance

Figures 3.24-3.26 illustrate the large-scale performance of our algorithms for the simulation described in Figure 3.20. We firstly analyze Polyhedron-Polyhedron collision detection, which is computationally the most expensive, in Figure 3.24. We see that the trend of linear scaling holds up to 34 million particles, which is only limited by the current Nvidia NVCC compiler .

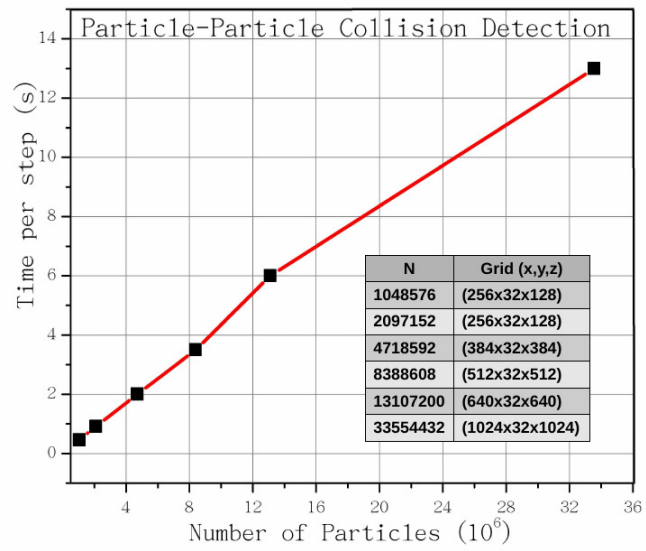

Figure 3.24: Scaling of polyhedron-polyhedron collision detection with number of particles (N) for Cubes (6 Face) on a K6000 GPU.

The trend of linear scaling continues with the polyhedra-world planar surface contact algorithm. However, it is significantly faster and takes 0.25 seconds for 34 million particles.

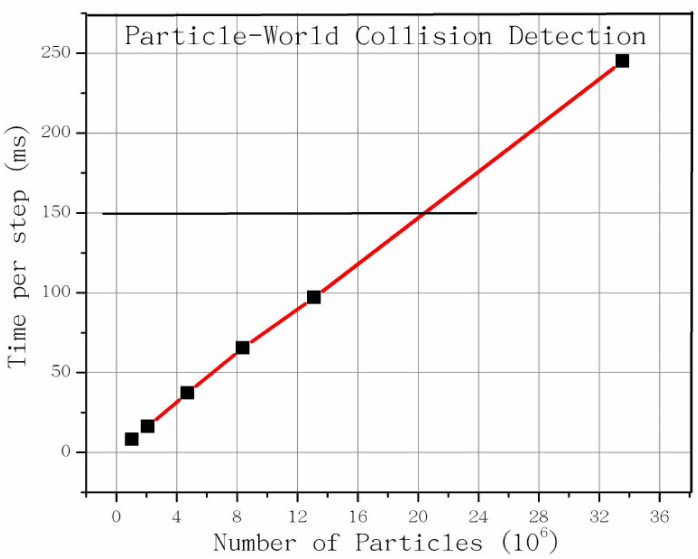

Figure 3.25: Scaling of polyhedra-world planar surface contact algorithm algorithm with number of particles (N) for Cubes (6 Face) on a K6000 GPU.

The broad-phase search algorithm [44] takes under one second a step for 34 million particles, with linear scaling as well, indicated in Figure 3.26. Our results set a new performance level in contact detection for polyhedra in DEM.

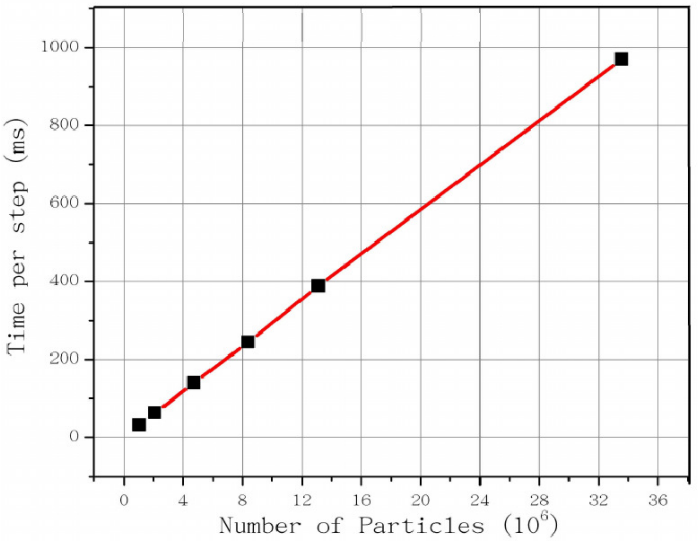

Figure 3.26: Scaling of Broad-Phase algorithm with number of particles (N) on a K6000 GPU.

Table 3.3 shows the comparison of our code to that of other authors using the Cundall Number $C = \mathrm{N} \times \mathrm{FPS}$, 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 author's 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 et al.) in Table 3.3. Although the code GRPD by

Radake et al. [13] performs the best it uses only spherical particles. BLAZE-DEM[44] using spherical particles is about 3 times faster than other codes as indicated in Table 3.3. Note that we achieve an average GPU utilization of 77% of the peak theoretical performance.

Table 3.3: Comparison to other GPU codes (SD: Spring-Dashpot (Normal), NIT: Non-Incremental Tangent, IT: Incremental Tangent, IV: Impulse Velocity).

Author	**Shape(mono size)**	Physics	GPU	**N particles**	**C Number**
Harida et al.[28]	Clumped(4 sphere)	IV	Fermi (8800)	1.64×10^4	0.66×10^6
Longmore et al. [25]	Clumped(4 sphere)	SD,NIT	Fermi (8800)	2.56×10^5	1.49×10^6
BLAZE-DEM	Poly (6 face)	SD,NIT	Kepler(G110)	34×10^6	2.62×10^6
GRPD [13, 45]	Sphere	SD,NIT	Kepler(G110)	20×10^6	20×10^6
Govender et al. [46, 47]	Sphere	SD,NIT	Kepler(G110)	50×10^6	55×10^6

Figure 3.27 shows the frequency of the two collision types depicted in Figure 3.2. We see that indeed vertex-face is the dominant contact type and justifies our choice in searching for it first. In Figure 3.27(a) we see that edge-edge contact is at most 10%. We note a similar trend in Figure 3.27(b) where a slightly higher percentage of edge-edge contact can be attributed to the geometrical effect of the drum. In Figures 3.27(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.

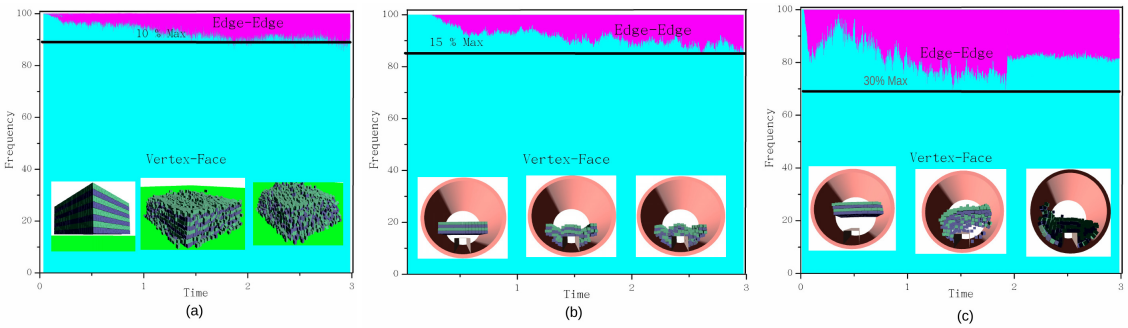

Figure 3.27: Frequency plot of the collision type against simulation type for (a) gravity packing problem, (b) gravity packing in drum without initial velocity, (c) gravity packing in drum with initial velocity.

3.5 Conclusion

In this chapter 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. With the overall framework described in detail in Chapter 2, and the details of the collision detection aglorithims described in detail in Chapter 3, Chapter 4 now focuses on an industrial scale application.

CHAPTER 4

Discrete Element Simulation of Mill Charge using the BLAZE-DEM GPU framework

4.1 Introduction

4.1.1 Background and Motivation

Since the first application of the discrete element method (DEM) for the simulation of grinding mills by Rajamani [48] in 1990 there has been a phenomenal growth in the variety of ways this technique is used in the mining industry. Prior to DEM, Powell's [49] single ball trajectory in rotary mills was a key advancement in understanding lifter relief angle on the trajectory of charge. This approach continues to serve the mining industry even today.

In the late 1990s two-dimensional DEM codes were the norm, due to the ease of execution on a personal computer with a single central processing unit (CPU). On the other hand, three-dimensional simulations promise greater accuracy of simulated results at the expense of computing time. At the outset it is useful to discuss the merits of 3D codes in comparison to 2D codes. The 2D code executes in a matter of hours on a CPU. It has been heavily used in hundreds of mining operations for annual or semiannual replacement of shell lifters [50]. This code has impacted the production, capacity and liner life of ball mills, autogenous mills and semi-autogenous (SAG) mills. The three-dimensional simulations are more accurate because the momentum transfer between balls and rock particles in the axial direction of the mill is accounted for. This opens the door for new insights when utilizing large-scale 3D simulations. 3D simulations has not been readily available to researchers and mill designers since execution times are of the order of weeks on a single CPU for a typical plant size mill. This then becomes impractical to pursue

on a routine basis.

Regardless of the severe computational burden of 3D simulations, a number of successes have been reported. In 2001 Venugopal and Rajamani [3] presented a 3D DEM computational framework and compared it against the power draft in a laboratory scale 90 cm diameter mill. In the same year Rajamani and Mishra [27] used the same 3D code for the prediction of power draft in plant scale mills. Herbst and Nordell [51] combined 3D DEM with smoothed particle hydrodynamics (SPH) and the finite element method (FEM) to simulate slurry and solid charge motion, ore particle breakage and liner wear. Cleary [52] demonstrated the sensitivity of charge behavior and power draft of a 5m diameter ball mill to liner geometry and charge composition using a 3D code. There are continued advances in the simulation of breakage and slurry flow incorporating all the details in three-dimensional simulations. Morrison and Cleary [53] describe the evolution of "Virtual Comminution Machine", a simulation code that simulates breakage and slurry transport in tumbling mills. In their simulation both the discrete element method and smoothed particle hydrodynamics are employed for slurry and pebble flow through the grate slots and the pulp lifter. Cleary and Morrison [54] show that 3D DEM combined with SPH is a viable tool for analyzing mineral processing equipment such as mills and twin deck screens. In a more recent study Alatalo et al. [55] compared the experimental deflection of a lifter in a ball mill with 3D predictions made with EDEM [56], a commercial DEM code. They concluded that 3D simulations agree better with predicted experimental values than 2D simulations.

4.1.2 Computational Aspects

A full 3D simulation of a mill will give valuable insights into the dynamics within a mill which can improve energy efficiency resulting in savings of thousands of dollars. An emerging trend of the past few years is the implementation of scientific and engineering solutions on a new class of processors termed General Purpose Graphical Processor Units (GPGPU) [25, 26, 28], which offers CPU cluster computing performance at a fraction of the cost. Rajamani et al. [57] shows a speed increase of up to 50x over CPU implementations for mill charge motion while Govender et al. [58] showed a speed increase of up to 132x for polyhedral particles.

4.1.3 Additions to BLAZE-DEM framework for mill simulations

Our mill simulation code is built on the BLAZE-DEM GPU framework developed by Govender et al. [44, 58]. The collision detection algorithm is based on the geometry class classification as described by Govender et al. [58] for the GPU architecture:

1. We firstly do a broad phase check if the particle is beyond the cylinder with radius

r that bounds all the lifters (determined by lifter with largest radius).

2. If a particle passes this check we then loop over all lifters to check if there is intersection between the particle and the bounding cylinder with radius r_{lifter} of a lifter, as depicted in Figure 4.1(a).

Heuristic 1 requires $\mathcal{O}(N)$ computations and heuristic 2 requires $\mathcal{O}(K)$ computations where K is the number of lifters and N is the number of particles.

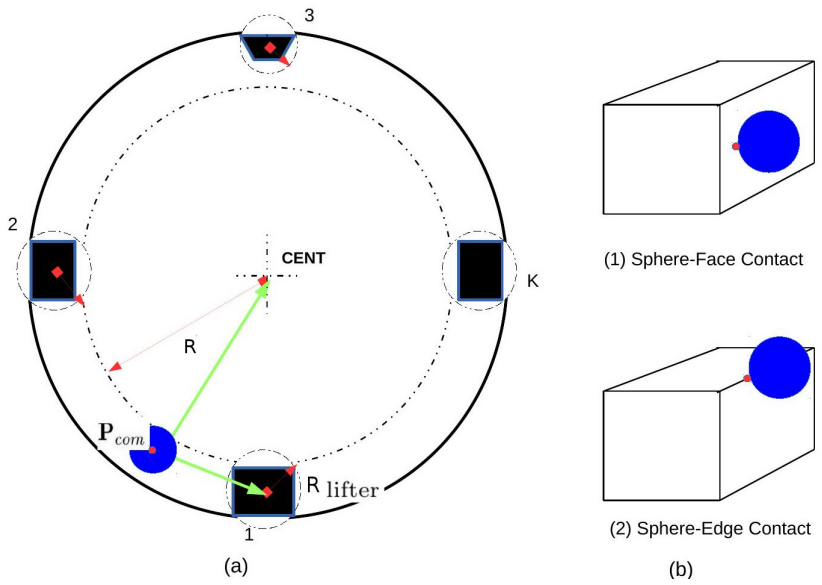

Figure 4.1: (a) Particle-lifter broad-phase collision detection and (b) detailed collision detection.

If there is an intersection between a ball and a lifter we then find the contact normal and penetration distance as described in Algorithm 4.1.

Algorithm 4.1 Particle-lifter detailed collision detection.

1. **Loop** over all **lifter faces**
   a) Compute the distance $\mathrm{d} = \mathbf{n}_{\mathrm{face}} \cdot (\mathbf{P}_{\mathrm{com}} - \mathbf{C})$ between the lifter face and particle. Here $\mathbf{P}_{\mathrm{com}}$ is the center of mass position of the particle, $\mathbf{C}$ is the center of the face and $\mathbf{n}_{\mathrm{face}}$ the face normal.
   b) If this distance d is less than the particle radius we have possible contact.
      i. The contact point is given by $\mathbf{P}_{contact} = \mathbf{P}_{com} + \mathrm{d}.\mathbf{n}_i$.
      ii. We now check if the contact point is actually on the face as (a) indicates contact for an infinite plane.
      iii. We check if the penetration $\delta = \mathrm{d} - \mathrm{r}$ is valid (max 5% of radius) ($\delta < 0.05\mathrm{r}$). If this is exceeded a warning is printed in the log file.
      iv. We have contact at point $\mathbf{P}_{\mathrm{contact}}$ with normal $\mathbf{n}_{\mathrm{face}}$ and penetration distance δ.
2. If there is no contact with the faces we do a check for contact with edges, which is computationally more expensive.
3. **Loop** over all **lifter edges**
   a) Compute the vector $\mathbf{L}_{\mathrm{PE}} = (\mathbf{P}_{\mathrm{com}} - \mathbf{E}_{\mathrm{i}}^{0})$ that gives the shortest distance between the particle and lifter edge, where $\mathbf{E}_{\mathrm{i}}^{0}$ is a vertex on the lifter edge.
   b) We now check if this vector is valid. If $(\mathbf{E}_{\mathrm{i}}^{\mathrm{Dir}} \cdot \mathbf{L}_{\mathrm{PE}}) > 0$, where $\mathbf{E}_{\mathrm{i}}^{\mathrm{Dir}}$ is the direction along the lifter edge, then
      i. We compute the contact point $\mathbf{P}_{\mathrm{contact}} = (\mathbf{E}_{\mathrm{i}}^{0} + \parallel \mathbf{L}_{\mathrm{PE}} \parallel \mathbf{E}_{i}^{\mathrm{Dir}})$.
      ii. We check if the distance $\mathrm{d} = \parallel \mathbf{P}_{\mathrm{com}} - \mathbf{P}_{\mathrm{contact}} \parallel$ between the point and the particle is less than the radius.
      iii. We check if the penetration $\delta = \mathrm{d} - \mathrm{r}$ is valid (max 5% of radius) ($\delta < 0.05\mathrm{r}$). If this is exceeded a warning is printed in the log file.
      iv. We have contact at point $\mathbf{P}_{\mathrm{contact}}$ with normal $\mathbf{n} = (\mathbf{P}_{\mathrm{com}} - \mathbf{P}_{\mathrm{contact}})/\mathrm{d}$ and penetration distance δ.

4.1.4 Force Model

Figure 4.2 shows the normal and tangential force models which are commonly used in DEM simulations of Ball Mills [10].

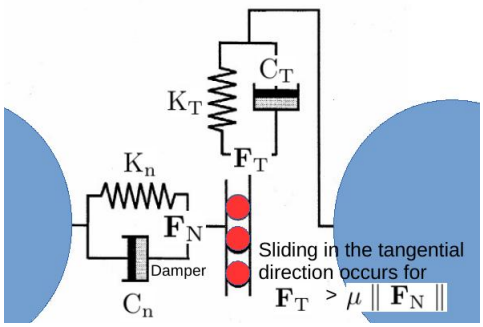

Figure 4.2: Normal and tangential force models depicted by a spring dash-pot system.

A linear spring dash-pot model is used to calculate the normal force between particles given by :

$$\mathbf{F}_N = (K_n \delta)\mathbf{n} - C_n(\mathbf{V}_R \cdot \mathbf{n})\mathbf{n}, \tag{4.1}$$

where δ is the penetration depth, $\mathbf{V}_R = \mathbf{V}_1 - \mathbf{V}_2$ is the relative translational velocity, $K_n = \frac{m_{eff}}{t_{contact}^2} \ln(\epsilon)^2 + \pi^2$ is the spring stiffness, $C_n = \frac{2\ln(\epsilon)\sqrt{K_n m_{eff}}}{\sqrt{\ln(\epsilon)^2+\pi^2}}$ is the viscous damping coefficient, $\mathbf{n}$ the normal at contact, ϵ is the coefficient of restitution and $m_{eff} = (\frac{1}{m_1} + \frac{1}{m_2})^{-1}$ is the effective mass of the particles. The contact time $t_{contact}$ is determined by the properties of the material. However in most cases experimental data is not readily available for a particular material. For such cases K_n is chosen such that that physical quantities of interest (such as energy) are conserved during integration for the typical range of velocities observed in tumbling mill simulations [3, 10, 26].

Typical DEM simulations use a spring stiffness and time step that limits the maximum penetration depth to $\delta_{max} \leq 0.05r$ where r is the radius of the smallest particle [18, 24, 25, 52].

Tangential Contact In CPU DEM codes a linear spring dash-pot model is also used to calculate the tangential force given by:

$$F_T = -\min\left[\mu \parallel \mathbf{F}_N \parallel, (K_T \int (\parallel \mathbf{V}_T \parallel dt) - C_T \parallel \mathbf{V}_T \parallel)\right], \tag{4.2}$$

where $\mathbf{V}_T$ is the relative tangential velocity at the contact point as calculated in Section 2.3.1, μ the coefficient of friction, K_T the tangential spring stiffness and $C_T = \frac{2\ln(\epsilon)\sqrt{K_T m}}{\sqrt{\ln(\epsilon)^2+\pi^2}}$ the tangential damping coefficient. The integral of the tangential velocity over the duration of the contact behaves as an incremental spring that stores energy from the relative tangential motions between the particles. This represents the elastic tangential deformation of the contacting particles while the dash-pot dissipates a proportion of energy. The magnitude of the tangential force is limited by $\mu \parallel \mathbf{F}_N \parallel$ at which point the particles begin to slide over each other.

However, due to the sorting of data on the GPU [44], maintaining contact history for the duration of the collision will result in a drop in performance due to uncoalesced memory transactions. Thus currently GPU codes use a history independent tangential model [24, 25, 58] that ignores the elastic tangential deformation in the tangential direction during contact and only accounts for energy dissipation. Therefore the GPU friction model [25, 44, 45] is simplified to:

$$\mathrm{F_T} = -\min\left[\mu \parallel \mathbf{F_N} \parallel, \frac{\min\left[\min\left[\|\mathbf{V_{1T}}\|, \|\mathbf{V_{2T}}\|\right], \mu\|\mathbf{V_T}\|\right]\mathrm{m_{eff}}}{\triangle \mathrm{t}}\right], \tag{4.3}$$

where $\mathbf{V_{1T}}$ and $\mathbf{V_{2T}}$ are the tangential velocities of each particle. This model has been shown to match experiments very well for simulations where the particles are constantly in motion in the previous works by the authors [44].

4.1.4.1 Calculation of power drawn by a mill

Grinding in the mineral processing industry is performed primarily by a ball mill which consumes a vast amount of energy and can account for as much as half of the processing cost. Thus understanding grinding mechanisms and estimating the power drawn by a mill can give guidance to improve the operational energy efficiency. The harsh environment inside the mill makes obtaining experimental data difficult. Thus the physical quantities calculated in a DEM simulation provide valuable insight to improve the efficiency of a mill. Since the power drawn by a mill is largely determined by the dynamics of the charge within the mill, we can obtain a good estimate of the power required by analyzing the energy loss mechanisms in a DEM simulation.

The total energy consumed by a mill is simply the net sum of the energy dissipated through contact $\mathrm{E_{diss}} = \sum_i^{\mathrm{L}}(\parallel \mathbf{F}_{\mathrm{diss}} \parallel \triangle \mathrm{x})$ where L is the number of contacts. Here $\mathbf{F}_{\mathrm{diss}}$ is given by the damping and friction forces in Equations 4.1, 4.2 and 4.3 respectively and is assumed to be constant over the distance $\triangle$x that the force acts. $\triangle$x is estimated by $\parallel \mathbf{V} \parallel \triangle$t since we do not store contact history in our GPU implementation. Thus the power consumed can be estimated by $\mathrm{Power} = \frac{\mathrm{E}_{diss}}{\mathrm{t}}$ where t is the duration over which we wish to calculate the power.

4.1.5 Calibration of model parameters.

In a DEM simulation the model parameters are chosen to either match experimental results or to reproduce a desired behavior. Tuning these parameters is a tedious task with the plethora of different models used in DEM simulations. Little guidance can be found in literature as the problems being simulated are quite often unique. In the area of ball mill simulations the 2D DEM code Millsoft developed by Rajamani et al. [48] was the first code to be employed in the simulation of mills and has been validated extensively

over the past two decades. Thus we verify our GPU code using the non-incremental tangential force model given by Equation 4.3 , against Millsoft using the incremental tangential force model given by Equation 4.2. In mill simulations the dynamics of the system is governed by the rotation speed Ω of the lifters. The distance covered by a lifter during a time-step is given by $x_{\text{Lifter}} = R.\omega.\triangle t$ where R is the radius of the mill drum and $\omega = \frac{\pi}{30}.\Omega$ is the angular velocity of the mill. Thus the maximum time step ($\triangle t_{\max}$) for a mill rotating at Ω rpm having a maximum penetration distance of δ_{max} is given by:

$$\triangle t_{\max} = \frac{\delta_{max}}{R.\omega} \tag{4.4}$$

4.1.5.1 Effect of parameters on charge profile

In this simulation we use a mill with diameter of 516 cm and length set equal to the ball diameter for simulating motion in 2D when using a 3D code. Thirty two rows of rectangular lifters with a height of 9.5 cm and width of 15 cm is used with the mill rotating at 14 rpm. We attempt to tune the model parameters of the GPU code to reproduce the charge profile obtained using Millsoft. We start of by using the same parameters with the GPU code as given in Table 4.1. Note that the maximum allowed time-step given by Equation 4.4 is $\triangle t_{\max} = 1.65 \times 10^{-5}$ s using a maximum penetration distance of 0.025r.

Table 4.1: Untuned model parameters used in simulation for a 2D mill.

Parameter	K_n (N.m^{-1})	ϵ	K_T (N.m^{-1})	μ	$\triangle t$ (s)
GPU	4×10^5	0.45	-	0.70	1×10^{-5}
CPU	4×10^5	0.45	3×10^5	0.70	1×10^{-5}

Figure 4.3(a) shows the charge profile obtained with Millsoft and sub-figures (b)-(d) the GPU profiles for various values of μ. We notice in Figure 4.3(b) using the same frictional value that there is a good match with the shoulder position and the release point of the lifters. However there is a significant difference with the toe position in the GPU simulation at 6 o'clock 30' while it is at 7o'clock 30' in the CPU simulation as indiacted by the clock numbers on the cross section. This difference is attributed to the absence of a restorative force in the GPU tangential model, which results in a greater resistive force being experienced by the particles for the same value of μ. This keeps the center of mass of the distribution at a higher point as illustrated by the belly position. In Figure 4.3(c) we use a lower friction value of $\mu = 0.6$ in the GPU simulation. We notice that the toe is lower but the frictional force is still too high. In Figure 4.3(d) we reduce the friction value to $\mu = 0.4$ in the GPU simulation. We now see a much better agreement of the toe, shoulder and belly positions.

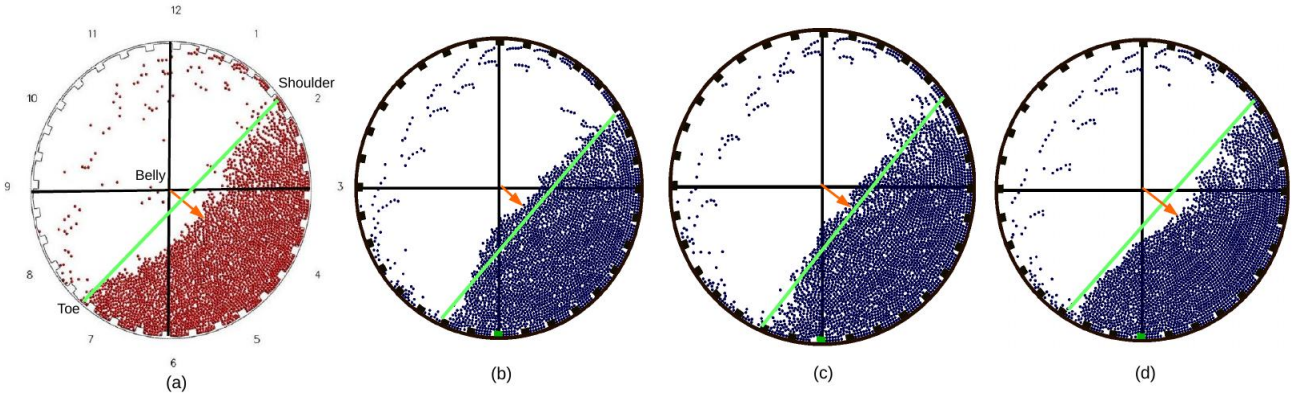

Figure 4.3: Charge profiles for CPU (a) $\mu = 0.70$ and GPU (b) $\mu = 0.70$ (c) $\mu = 0.60$ and (d) $\mu = 0.40$, N= 2916 (radius=2.5 cm).

The second parameter that we can tune is the coefficient of restitution ϵ. In Figure 4.4 we vary the value of ϵ while maintain the frictional value of $\mu = 0.4$. We notice that for $\epsilon = 0.25$ (Figure 4.4(a)) the distribution is packed much tighter as a larger fraction of energy is lost during contact. For $\epsilon = 0.65$ (Figure 4.4(c) the packing is less dense producing a greater dispersion when particles are released that provides a closer match to the distribution obtained with Millsoft (Figure 4.3(a)).

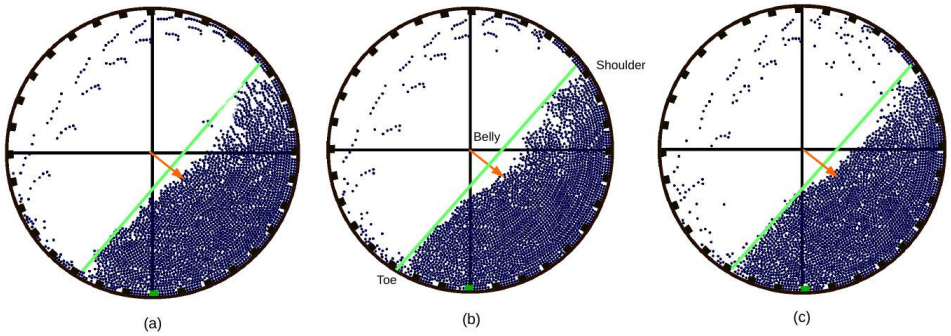

Figure 4.4: GPU charge profiles for (a) $\epsilon = 0.25$ (b) $\epsilon = 0.45$ and (c) $\epsilon = 0.65$, N= 2916 (radius=2.5 cm).

Now that we have calibrated the GPU parameters, we vary the number of particles (N) and compare the charge profiles using the same parameters as summarized in Table 4.2.

Table 4.2: Tuned model parameters used in simulation for a 2D mill.

Parameter	K_n (N.m^{-1})	ϵ	K_T (N.m^{-1})	μ	$\triangle$t (s)
GPU	4×10^5	0.65	-	0.40	1×10^{-5}
CPU	4×10^5	0.45	3×10^5	0.70	1×10^{-5}

The charge profiles for N= 5344 are compared in Figure 4.5, while Figure 4.6 compares the charge profiles for N= 11664. We notice a good match using the same set of parameters.

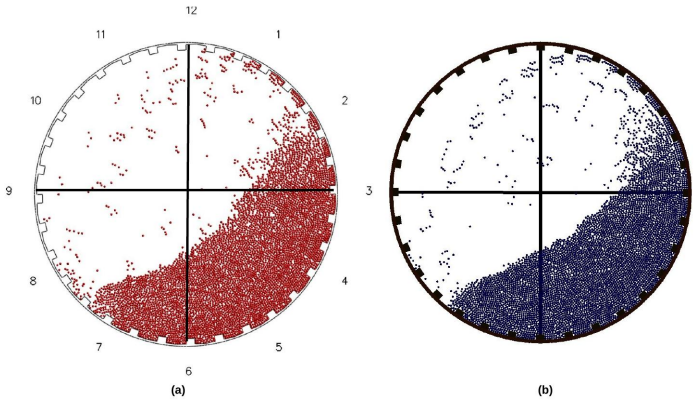

Figure 4.5: (a) CPU and (b) GPU charge profiles. N= 5344 (radius=1.85 cm).

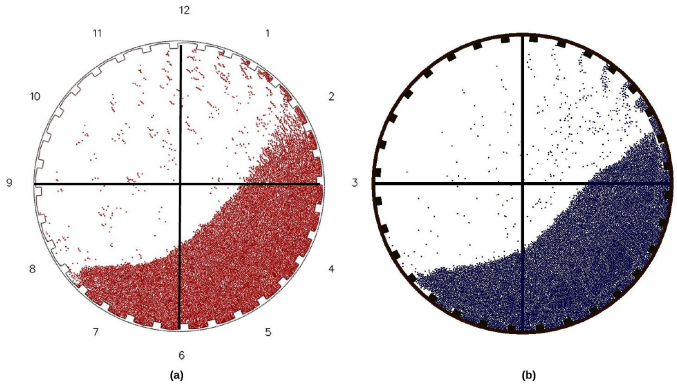

Figure 4.6: (a) CPU and (b) GPU charge profiles. N= 11664 (radius=1.25 cm).

4.2 Experimental validation of GPU DEM for mill simulations.

4.2.1 Three-dimensional mill

We use the experimental data obtained by Venagopal and Rajamani [3] using a 90 cm diameter by 15.0 cm length mill containing eight 4.0 cm square lifters. The face of the mill was made of Plexiglas^{TM} as to enable photographing of the tumbling charge, with the shell and lifters being made of steel (density of 7800 kg.m^{-3}). The mill was operated at 30%, 50% and 70% of critical speed for two levels of mill filling, 20% and 30% by volume respectively using steel balls with a radius of 2.5 cm. The critical speed of a mill is the speed at which particle motion changes from cascading to centrifuging. Note that the maximum allowed time-step given by Equation 4.4 is $\triangle t_{max} = 4.14 \times 10^{-5}$ s using a maximum penetration distance of 0.025r.

4.2.1.1 Calibration of model parameters

In the previous simulation we studied the effect of parameters on the charge profile in the mill. In this simulation we investigate the effect of the parameters on power draw as well for a mill with 20% loading at a mill speed of 32 rpm (70% of critical speed) drawing 532 W of power. We firstly match the charge profile by varying the frictional value as we saw in the previous simulation that this had the largest effect on the charge profile. Figure 4.7 shows the charge profile for various values of μ. We notice that the shoulder, toe and belly positions are lower as expected as we decrease the value of μ. Figures 4.7(f) and (g) having values of $\mu = 0.20$ and $\mu = 0.15$ respectively gives the best match to the experimental profile depicted in Figure 4.7(a). Thus the ideal value of μ seems to be in the range $[0.15 : 0.20]$

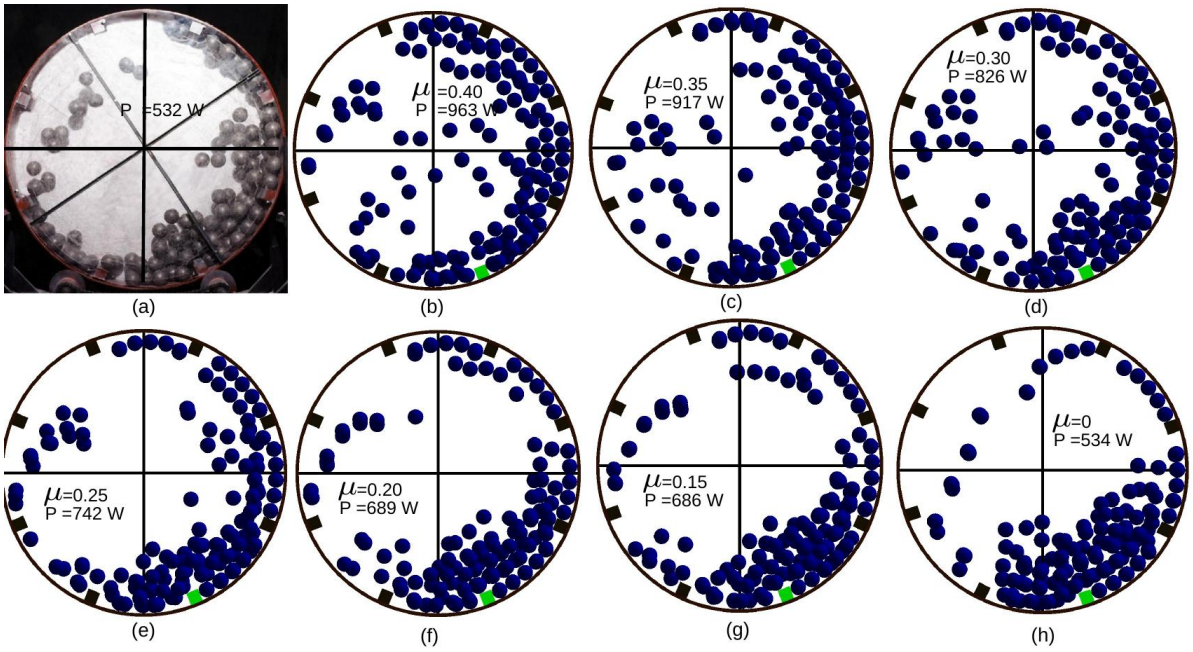

Figure 4.7: (a) Experiment (b) GPU charge profiles for different values of μ as indicated. N= 168 (20% filling), rpm = 32 (70% of critical speed).

Figure 4.8 depicts the charge profile for varying values of ϵ using $\mu = 0.20$. We firstly notice that there is little change with the shoulder and toe positions with the belly becoming less dense as we increase ϵ. The ideal value of ϵ seems to be in the range $[0.80 : 0.85]$ as power at the limits of the range bounds the experimental power of 532 W.

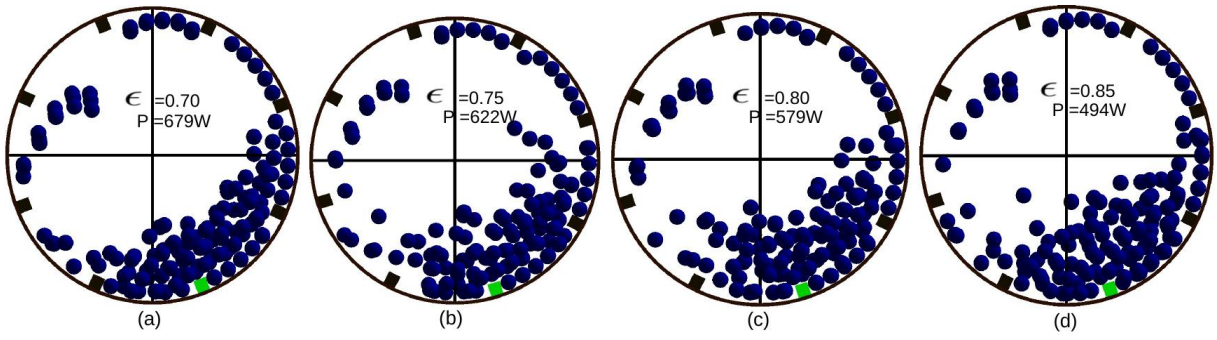

Figure 4.8: GPU charge profiles for different values of ϵ as indicated. N= 168 (20% filling), rpm = 32 (70% of critical speed).

Now that we have a reasonable match for the charge profile we tune the parameters to yield the desired power values. We use a finite number of combinations of μ and ϵ in the ranges that we deemed to yield the best charge profiles in Figure 4.7 and 4.8. Table 4.3 contains the four combinations we use as well as the average and maximum errors

obtained for mill speeds of 14, 22 and 32 rpm. Ideally the range of values should be more than the four we have chosen to get the best possible match. However we only wish to show the effect of DEM parameters on mill simulations rather than do a detailed calibration of DEM parameters. Combination 3 gives the lowest error and will thus be used to predict power and charge profiles for the rest of this section.

Table 4.3: Average error in power for various parameter combinations for a 3D mill.

Combination	ϵ	μ_T	Error at highest RPM	Average Error
1	0.80	0.15	7.88%	7.09%
2	0.825	0.20	7.34%	5.23%
3	0.825	0.15	5.99%	4.61%
4	0.85	0.175	8.01%	5.77%

4.2.1.2 Charge motion and power draw

Representative snapshots of the GPU DEM predicted charge profiles alongside the still camera images for each of the experiments are shown in Figures 4.9 to 4.12 with the associated power draft in Table 4.4. Charge profiles predicted by the GPU DEM code are consistent with observed charge profiles. The positions of the toe and shoulder of the charge are also reasonable with a slightly lower toe position and lower mass distribution which is expected due to the simpler tangential force model used. An exact match between charge trajectories is difficult to obtain due to the stochastic nature of the problem for slight deviations in the initial setup. In addition to the simplifying assumptions made in the DEM model, there are unaccounted mechanical losses and the geometry of the mill is not exactly the same due to wear and manufacturing processes.

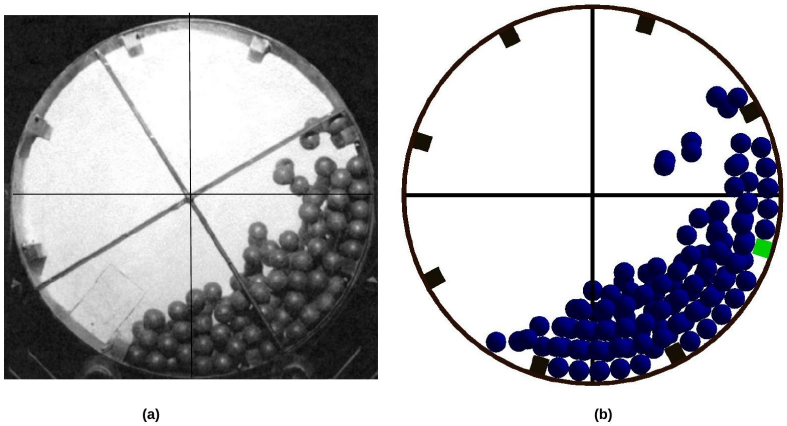

Figure 4.9: (a) Experiment [3] (b) GPU charge profiles. N= 168 (20% filling), rpm = 14 (30% of critical speed).

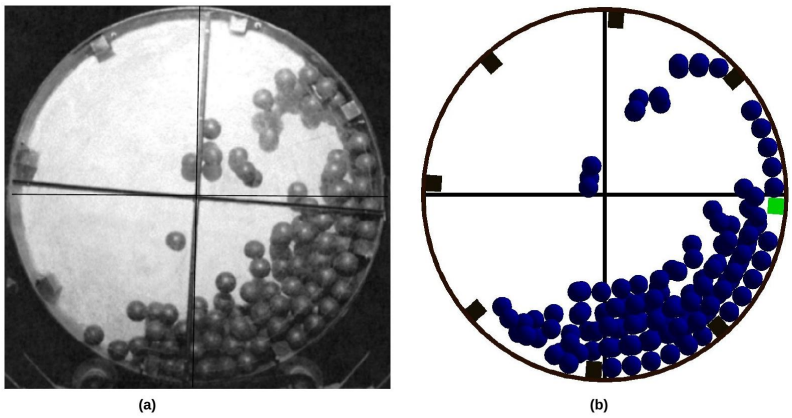

Figure 4.10: (a) Experiment [3] (b) GPU charge profiles. N= 168 (20% filling), rpm = 22 (50% of critical speed).

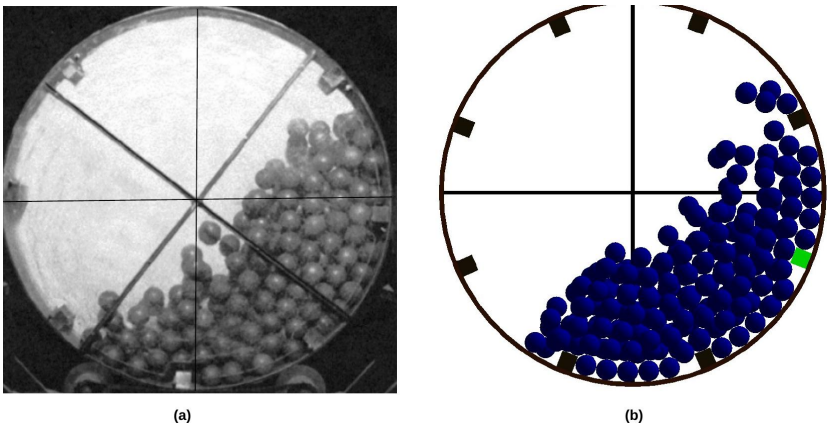

Figure 4.11: (a) Experiment [3] (b) GPU charge profiles. N= 243 (30% filling), rpm = 14 (30% of critical speed).

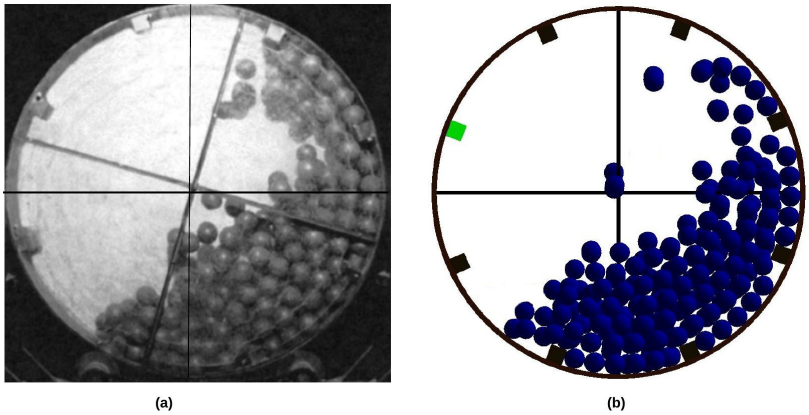

Figure 4.12: (a) Experiment [3] (b) GPU charge profiles. N= 243 (30% filling), rpm = 22 (50% of critical speed).

Table 4 summarizes the power values obtained. We see a good match for the 30% loading scenario as the error is only 4.07%. Note that we only tuned the parameters for the 20% loading scenario. This gives confidence in the selection of the model parameters as we can predict the effects of mill-load on the power draw.

Table 4.4: Power draw with experiment and GPU DEM for 3D mill.

RPM	Filling (20%)		Filling (30%)	
	Power(W)			
	Experiment	GPU DEM	Experiment	GPU DEM
14	301	336	393	409
22	459	437	617	636
32	532	520		

4.2.2 Charge motion and power draw for a slice mill

In this simulation we use the experimental data as given by Moyes et al. [4] for a pilot mill with a diameter of 55 cm and length of 2.35 cm containing twelve rows of 22 cm square lifters. The mill is loaded to 25% and 35% respectively with steel balls having a radius of 2.2 cm and density of 7800 kg.m^{-3}. For their DEM simulations Moyes et al. use the same model parameters for different mill-speeds. However as discussed in Section 1.6 the model parameters (specifically the time-step) are determined by the rotation speed of the mill. The maximum allowed time-step using Equation 4.4 for a maximum penetration distance of 0.10r per step at 160% of critical speed (93.30 rpm) is $\Delta t_{max} = 4.14 \times 10^{-6}$ s. To allow comparison with published results, we use the same Δt for increasing speed but also include results using a time-step given by Equation 4.4 as mill speed increases. Table 4.5 summarizes the model parameters used.

Table 4.5: Model Parameters used in simulation for slice mill.

Parameter	K_n (N.m^{-1})	$\epsilon_{particle}$	ϵ_{shell}	K_T (N.m^{-1})	$\mu_{particle}$	μ_{shell}	Δt (s)
GPU	4×10^5	0.85	0.80	-	0.15	0.20	2×10^{-5}
CPU	4×10^5	0.66	0.36	4×10^5	0.14	0.39	2×10^{-5}

Figure 4.13 shows how the power varies as a function of rotation speed. We see a good match for sub-critical speeds which is the normal operation mode of a mill between both DEM codes and experiment. At super-critical speeds there is a slight difference due to CPU and GPU codes using a constant time step. However the results using the time step given by Equation 4.4 as we increase mill speed shows a better match to experiment.

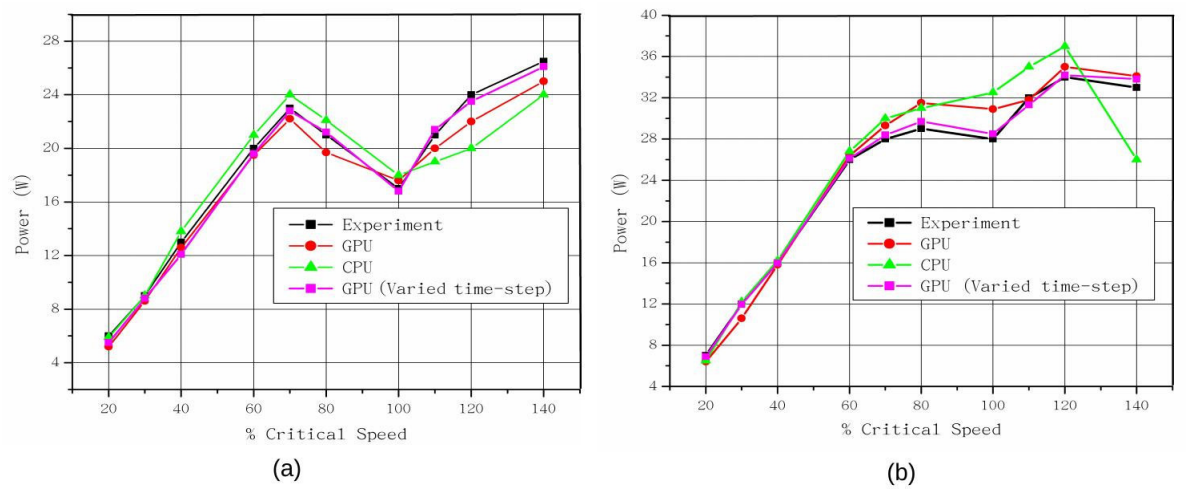

Figure 4.13: Power draw for (a) 25% and (b) 35% loading between experiment [4], GPU and CPU simulations.

Figures 4.14 and 4.15 depict the charge profile for sub-critical speeds. We see a good match which is expected as the power values are similar. Note that it is difficult to do an accurate frame matching.

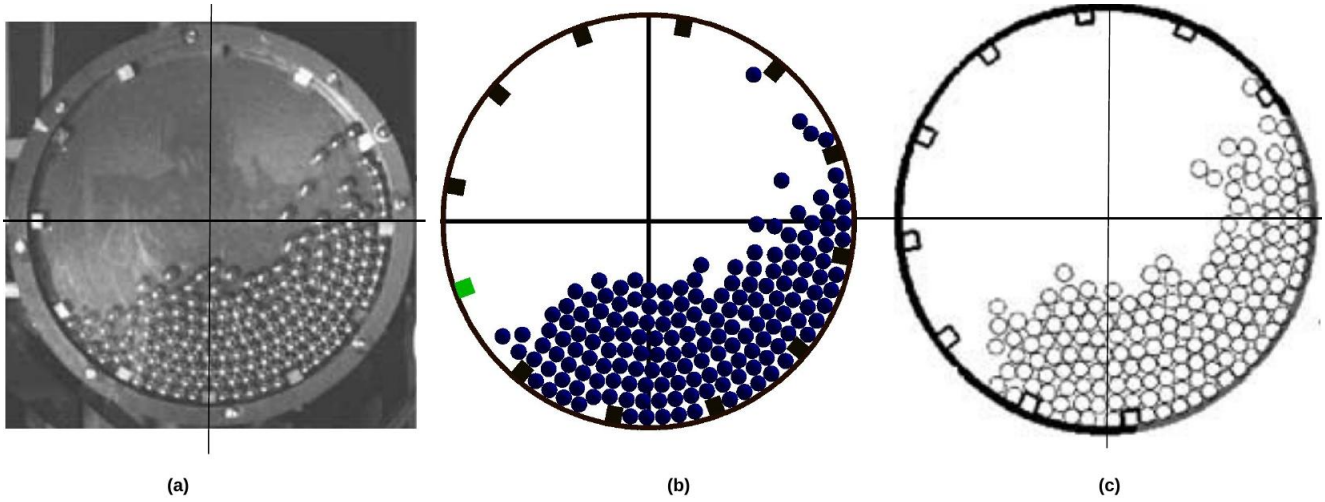

Figure 4.14: (a) Experiment [4] (b) GPU and (c) CPU charge profiles. N= 169 (35% filling), rpm = 17.50 (30% critical speed).

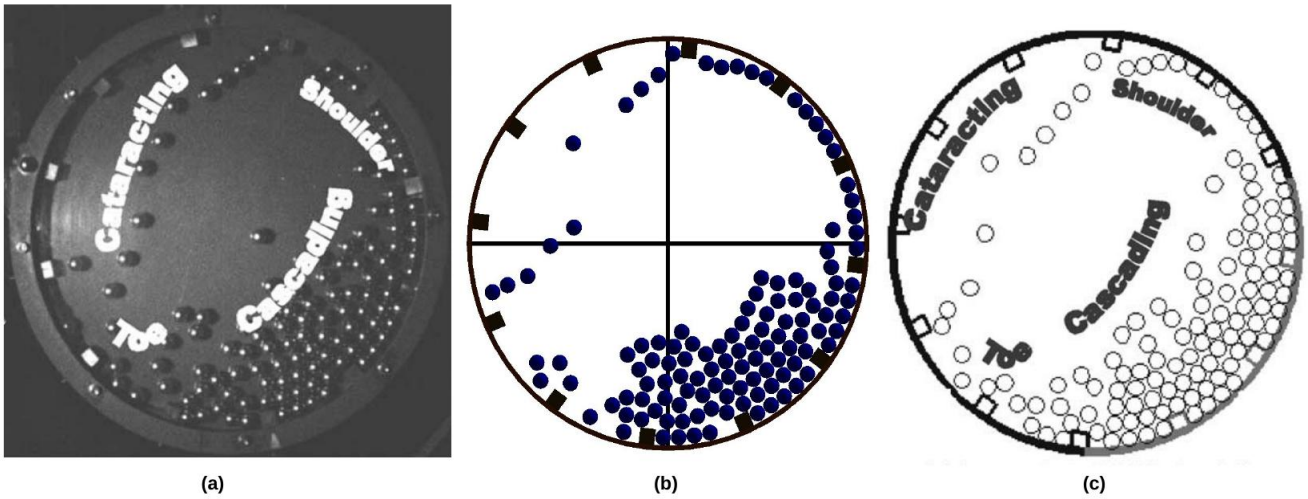

Figure 4.15: (a) Experiment [4](b) GPU and (c) CPU charge profiles. N= 120 (25% filling), rpm = 40.81 (70% critical speed).

Figures 4.16 and 4.17 depict the charge profile for super-critical speeds. We note that DEM correctly predicts 1 and 2 layers of centrifuging particles for 100 and 160 percent of critical speed respectively.

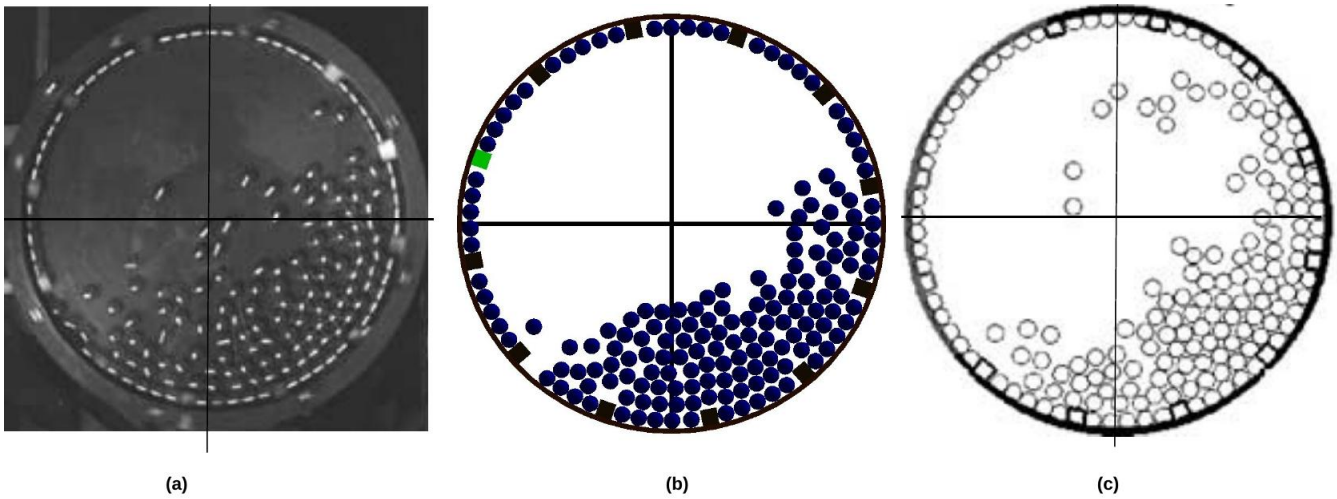

Figure 4.16: (a) Experiment [4] (b) GPU and (c) CPU charge profiles. N= 169 (35% filling), rpm = 58.30 (100% critical speed).

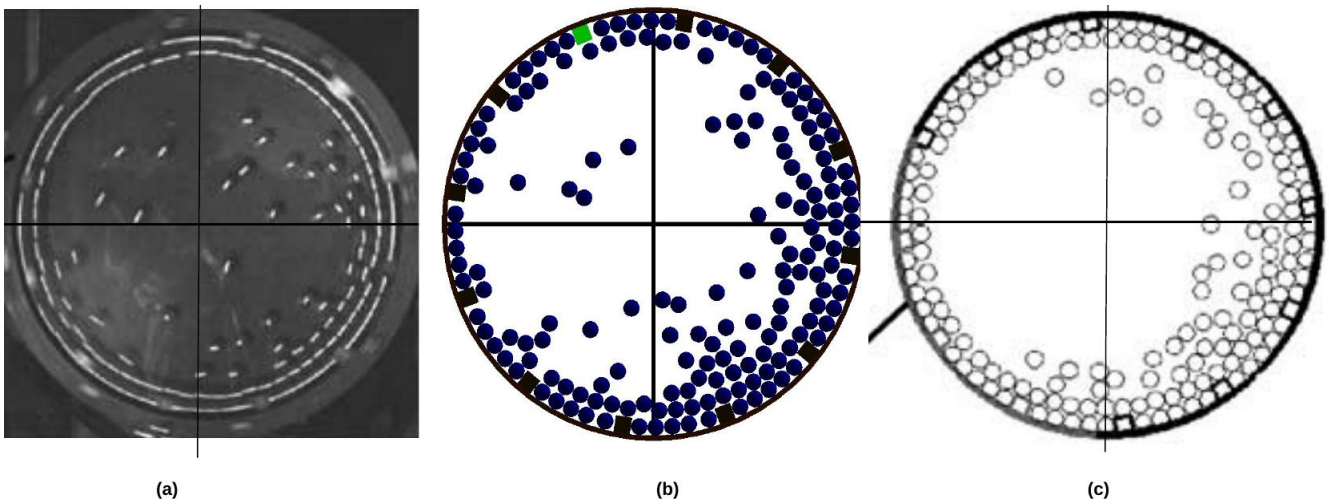

Figure 4.17: (a) Experiment [4] (b) GPU and (c) CPU charge profiles. N= 169 (35% filling), rpm = 93.30 (160% critical speed).

4.3 Industrial Mill Simulation

The Los Bronces SAG mill is a 10.12 m diameter by 4.7 m long mill rotating at 10 rpm. The mill charge is made up of 31% ore and 10% balls by volume. The power draft reported for this mill [59] is 7.1 MW. Since no further information about mill internals was available, typical values of operating parameters for this type of mill was used. It is presumed that the mill would be fitted with 64 rows of high low lifters as shown in Figure 4.18.

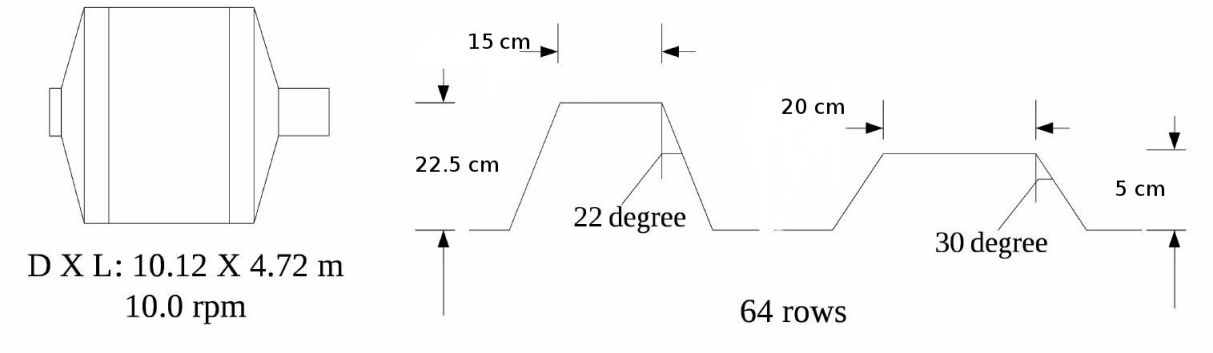

Figure 4.18: Lifter design Los Bronces semi-autogenous mill.

Equilibrium ball size distribution with top ball size of 12.5 cm was used in the simulation. The ore in the mill charge was approximated as a Gaudin-Schuhmann [60] distribution with slope 0.6 and top size of 15.0 cm. The combined weight distribution and number of ore and ball particles in the charge mix are shown in Table 4.6. The aim of this study is to do a qualitative investigation as opposed to a quantitative investigation due to the uncertainties in the experimental setup. We investigate how the power consumption varies over revolutions, as well as how the power gets dissipated inside the mill.

Table 4.6: Charge distribution for Los Bronces mill.

Diameter (cm)	Density (kg/m^3)	Weight (%)	Number	Color
	ORE			
15.0	2850	9	6320	red
12.5	2850	9	9020	green
10.1	2850	30	81881	blue
	BALLS			
12.4	7800	21	9800	magenta
10.0	7800	19	16171	yellow
8.7	7800	12	16220	cyan
TOTAL			**139392**	

Table 4.7 summarizes the model parameters used in the simulations in Section 4.3.

Table 4.7: Model Parameters used in simulation of Los Bronces mill.

Parameter	K$_n$ (N.m^{-1})	$\epsilon_{particle}$	ϵ_{lifter}	K$_T$ (N.m^{-1})	$\mu_{particle}$	μ_{lifter}	$\triangle$t (s)
GPU	1.5×10^6	0.75	0.75	-	0.40	0.70	2×10^{-5}

Figure 4.19 shows the charge profile in the mill. The charge profile is what one would anticipate, for such large filling of 41%. The charge shoulder is nearly at 2 o'clock and the toe is between 7 and 8 o'clock points on the mill circle. We also see radial segregation with the smallest particles moving to the mill shell as predicted by theory and experiment.

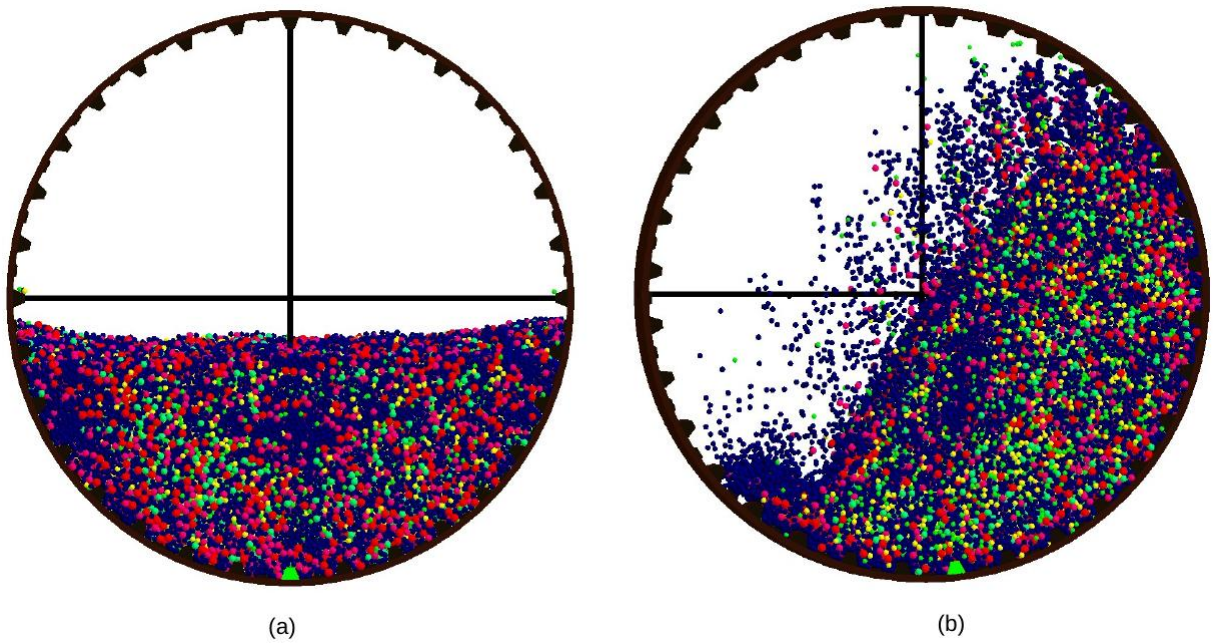

Figure 4.19: (a) Initial conditions N= 139392 (41% filling) (b) Charge profile of Los Bronces mill (colored by particle size as indicated in Table 4.6).

The computed power seems to stabilize after six revolutions as shown in Figure 4.20(a). The large number of spheres in the simulation smooth out the energy consumed in collisions per revolution. Hence, accurate dependence of contact parameters on contact velocity is unnecessary here. The computed power consumption is 6.8 MW, which is slightly lower than the experimental value of 7.1 MW. Note, this value is arbitrary as we showed we could match the power consumption exactly by changing the coefficient of restitution. The important observation is how the power consumption varies over revolutions, as well as how this power is dissipated. Figure 4.20(b) shows the contribution to power draw by particle to particle collisions, particle to mill shell collisions and particle to lifter collisions. Around 68% of the power draw comes from the lifter since it lifts the majority of the load to the shoulder of the mill and hence excessive forces occur at the points of contact on the lifters. Likewise, around 18% is contributed by the mill shell as it supports the load between two lifters. Thus less than 14% is contributed by particle-particle collisions. In addition, as the particle lifter power decreases slightly over revolutions, the mill shell power draw increases slightly. The particle-particle power dissipation remains virtually constant over the revolutions.

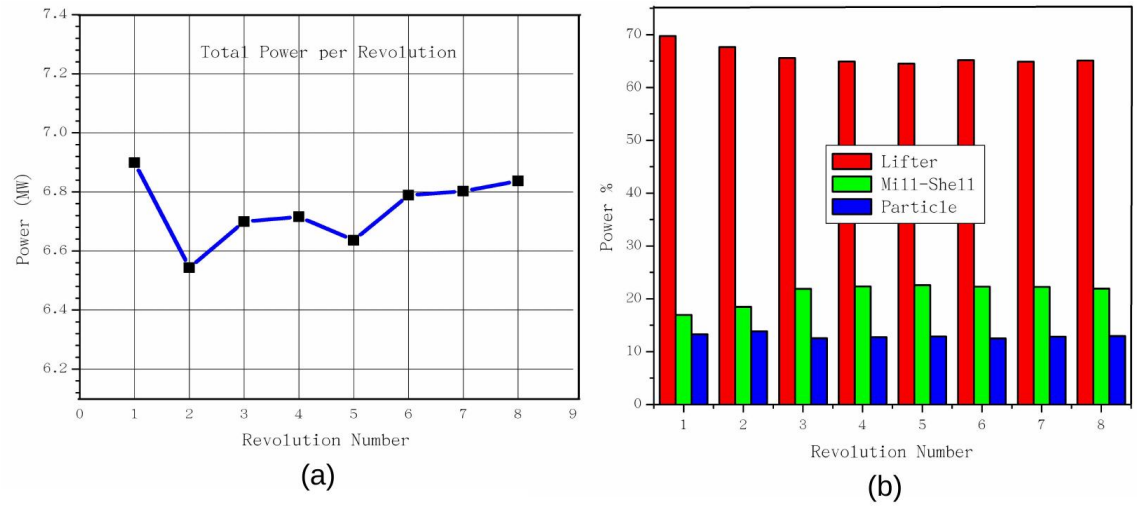

Figure 4.20: (a) Total power draw (b) Power distribution over time of Los Bronces mill.

4.3.1 Performance scaling

To gauge the performance of our code we increased the length of the mill to 2800 cm to accommodate four million mono-sized steel balls with a diameter of 6 cm. Figure 4.21 shows the charge profile.

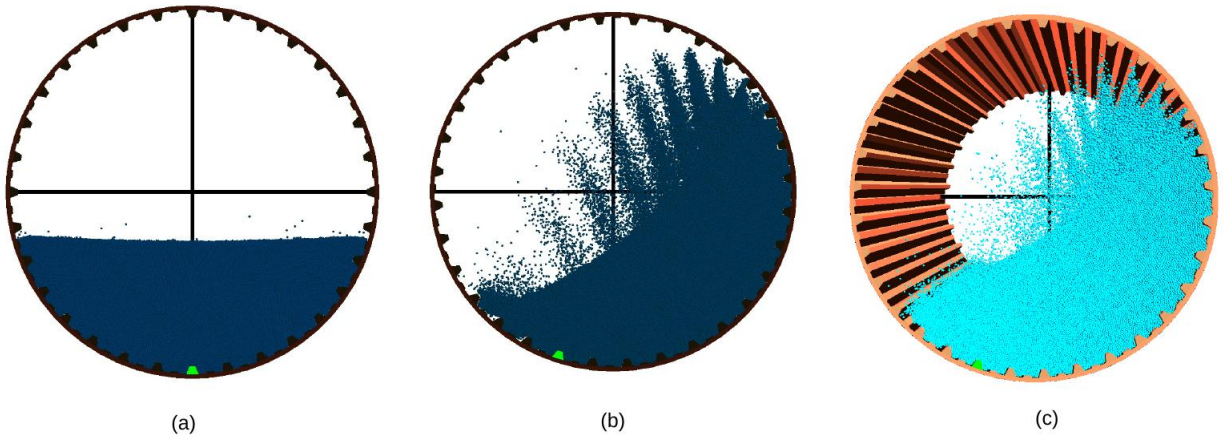

Figure 4.21: (a) Initial conditions N=4×10^6 (35% filling) (b) steady state profile (orthogonal view) (c) steady state profile (isometric view).

The power draft is distributed with 47% to particle-particle collisions, 44% to particle-lifter collisions and 9% to particle mill-shell collisions. The shear number of particles simulated results in particle-particle collisions consuming the most amount of energy. The GPU compute time for one revolution (6 seconds) using a time step of 2×10^{-5} s

with a NVIDIA Kepler GPU is 7 hours (12 FPS). Figure 4.22 shows the scaling of our code with increased number of particles. We observe a trend of linear scaling which is a good indication of the scalability of our code. The GPU compute time for one revolution (6 seconds) of a simulation consisting of 10 million particles using a time step of 2×10^{-5} s with a NVIDIA Kepler GPU will take just 18.5 hours with a simulation of 100 million particles taking just over a week.

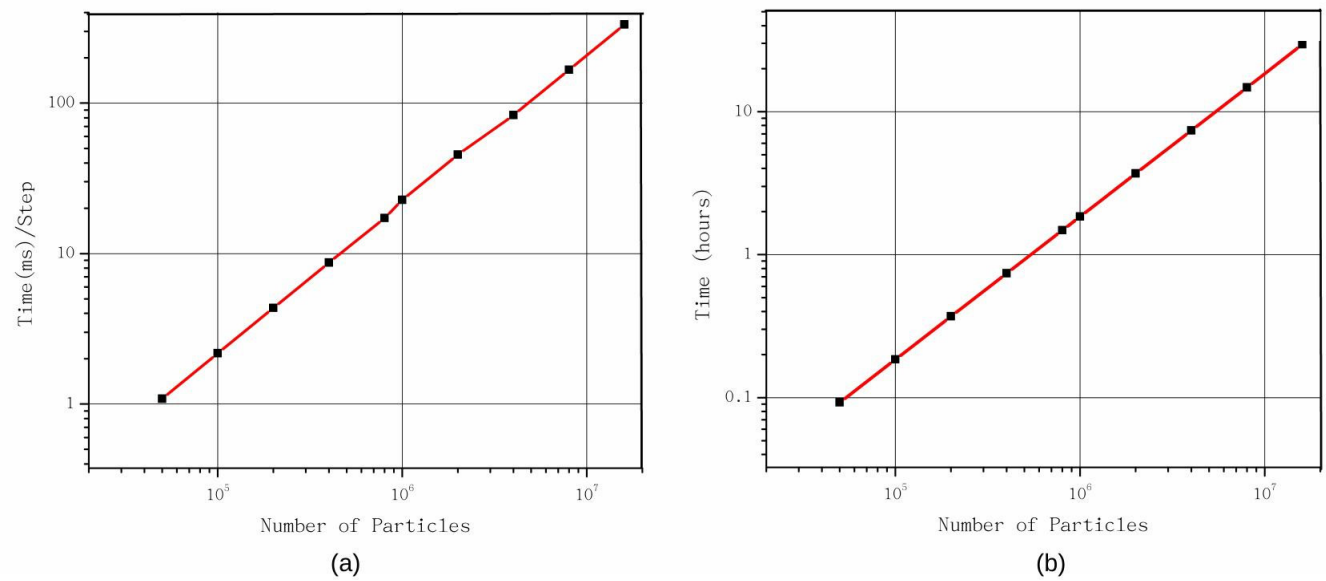

Figure 4.22: Scaling of GPU code with number of particles for ball mills on a Kepler GPU.

To show the benefits of a full 3D simulation we performed the same simulation in 2D with the same parameters and obtained a very different charge profile as depicted in Figure 4.23(a). Figure 4.23(b) depicts the charge profile for a slice of 10% of length. We notice that the pattern is very similar to the full length of the mill. Clearly the effect of the axial direction cannot just be neglected as the 2D case cannot fully reproduce the dynamics.

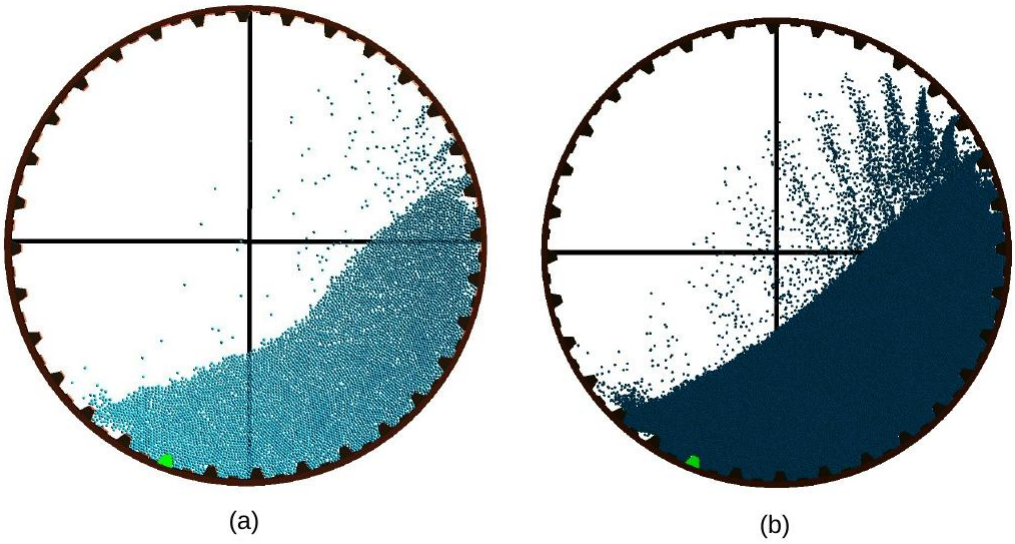

Figure 4.23: (a) 2D steady state profile, N=6744 (b) Steady state profile for a slice 10% of the length, N=385534.

4.4 Conclusions

In this chapter we have presented a novel approach for modeling tumbling mills utilizing the GPU architecture. The modular approach of our code allows us to analyze the distribution of power amongst different collision types in the mill which lends insight that may be exploited to improve the energy efficiency of mills. We achieve a new performance level in DEM modeling of mills by simulating 16 million particles at 3 FPS on a laptop GTX 880 GPU. In the next chapter we conclude this study and offer recommendations for future work.

CHAPTER 5

Conclusion and Future work

5.1 Concluding remarks

In this thesis the suitability of the Graphical Processing Unit (GPU) architecture to simulate particulate materials using the Discrete Element Method (DEM) was investigated. This thesis contributed to the field of particulate material transport through developments in the computational implementation of the DEM. The developments in this thesis allows for increased accuracy in particle shape by using polyhedral particles with a typical physics model fidelity used in DEM at a reduced computational run-time. The reduced computational cost and increased shape fidelity may also allow engineers to routinely conduct analysis in DEM simulations, expanding the value the technology adds to practical engineering and enhancing the insight in the underlying transport processes. In addition, the reduced computational cost enables potential sensitivity analysis and design optimization using DEM technology. Furthermore the number of particles that can be simulated on a single computer has been increased from tens of millions to hundreds of millions for spherical particles and from hundreds of thousands to millions for polyhedral particles for the same duration of computational run time. This increase in particle number improves the predictive capability of DEM when simulating industrial particulate material transport which often consists of hundreds of millions of particles.

In this thesis a computational framework for the implementation of the DEM on the Graphical Processing Unit (GPU) was developed. This aim of this research was to develop a computational framework for the GPU architecture that can model

1. hundreds of millions of spherical particles and
2. tens of millions of polyhedral particles in a realistic time frame on a desktop computer using a single GPU without sacrificing the fidelity of the physics model.

In developing our framework we followed a typical gaming dynamics engine approach, that separates the geometry description and physics model into modules. This gave us the flexibility to increase the geometrical complexity from spheres to detailed polyhedra using the same physics engine with the exception of the contact detection, ensuring that a fair comparison between spherical and polyhedral simulations can be made. Since Blaze-DEM is a novel implementation of DEM on the GPU architecture we verified our code against experiment for the problem of silo flow of particles for which the dynamics is well known. We found good agreement between the simulated and experimental flow patterns at various time snap shots. Due to the lack of published results owing to the large computational cost [20, 32] associated with a 3D polyhedral DEM simulation we could not make direct comparisons to other codes for the case of corn flowing in a silo. The flow rates was also found to be consistent with that of experiment. We further noticed that the polyhedral shaped particles did not flow smoothly with arching occurring due the nature of polyhedra as opposed to the smooth flow of spherical particles. This highlighted the need for the inclusion of particle shape in large scale 3D DEM simulations as pointed out in Chapter 2.

While collision detection for spheres are trivial, polyhedra are much more complex and can account for up to 90% of simulation time. Thus importance was placed on the development of contact algorithms that suited the parallel nature of the GPU without the loss of simulation fidelity and generality in terms of geometry. This generality in terms of world geometry allows for a variety of problems to be simulated using the code. Furthermore our representation of world geometry by primitives reduces memory storage and computation effort when compared to the typical particle/triangulation [16, 25, 26] representation of world geometry in other DEM codes as depicted in Figure 5.1.

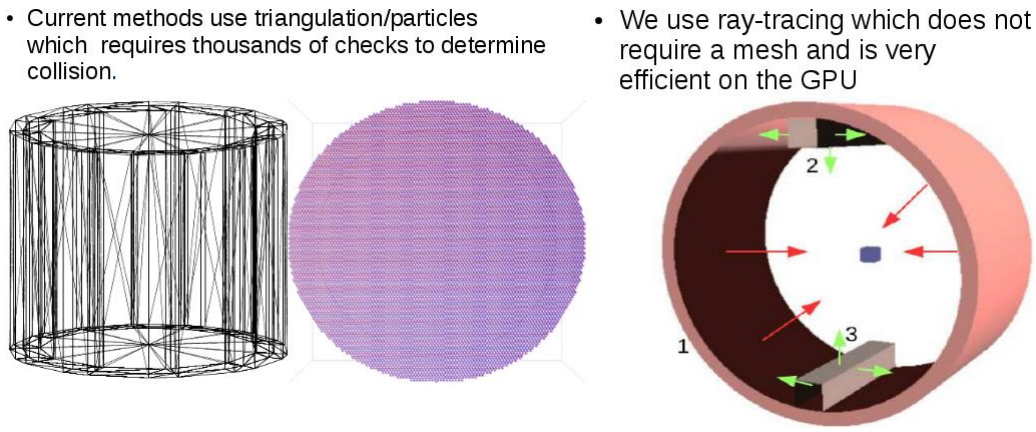

Figure 5.1: World geometry representations.

We verified the algorithms for the case of vertex-face contact between two particles which is analogous to the single point contact of spheres for which the contact mechanics is well known for various force models. We obtained the theoretically expected results

verifying the correctness of our algorithms. We demonstrated that with algorithms and heuristics designed for the parallel GPU architecture, we can achieve significant performance gains over CPU based simulations and other GPU implementations. We evaluated the scaling of our algorithm and found favorable results in that time does not necessarily increase linearly with increased geometrical complexity for the systems we have analyzed due to efficient memory access and exploitation of low-level parallelism as pointed out in Chapter 3. We achieved a new performance level in high fidelity modeling of shape in DEM by simulating 34 million polyhedra (13 seconds per time step) on a single Nvidia K6000 GPU.

Finally in Chapter 4 we demonstrated the flexibility and physical fidelity of our framework by simulating a variety of industrial problems. We obtained favorable results compared to experimental data and other CPU based computational codes. The modular approach of the framework allowed us to employ different physics models that was applicable to the range of systems simulated. The modular approach of our code also allowed us to easily include functions to analyze the distribution of energy and power in industrial processes which lends insight that may be exploited to improving the energy efficiency of such processes. We also achieved a new performance level in DEM modeling of mills by simulating 16 million particles at 3 FPS on a laptop GTX 880 GPU.

5.2 Future work

The major effort of this research was the development of a framework consisting of various collision detection methods and functions that facilitate the implementation of the DEM on the GPU with emphasis placed on flexibility. This allows low fidelity shape modeling using spherical particles and high fidelity shape modeling using polyhedral particles with a variety of physics models to be used. While extensive validation has been done for spherical particles, limited validation has been done for polyhedral particles and validation against experimental data needs to be done.

We noticed that occasionally the reported penetration distance for polyhedra-polyhedra contact is larger than reality which is the result of the incorrect contacting surface being selected. Consider Figure 5.2. By design our algorithm will return the largest penetration distance between two objects which is at surface 1 in Figures 5.2(a)-(c). However as the objects are being moved apart at some point the largest penetration distance is at surface 2 as depicted in Figures 5.2(d). This results in a contact force in the incorrect direction. For the simulations in this thesis we limited the extent of this error on particle dynamics by using a small time step and limiting the maximum allowed penetration distance. Solving this problem will allow the time-step to be increased resulting in further reduced simulations times. However the solution is not trivial and will require storage of the

contact history between contacting particles in an efficient manner in GPU memory.

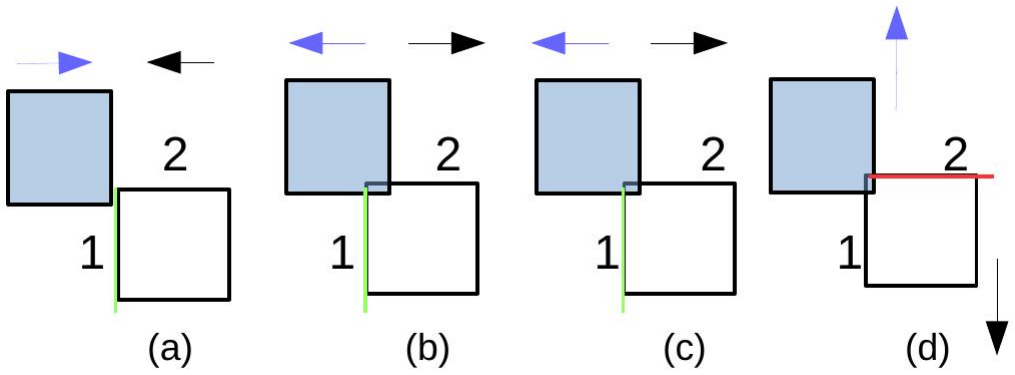

Figure 5.2: Incorrect surface selection.

In this thesis we used contact models that has been used by others in the DEM community which was suited for the parallel nature of the GPU. For the types of problems that we solved these models where sufficient. However for problems where particles are in quasi-static motion a tangential force model using an incremental spring model is needed to capture the complex frictional behavior. This requires the history for each contact to be stored, which is not possible as the sorting that we perform to reduce memory access time on the GPU makes it impossible to track particle pairs from one step to another. A possible solution is to devise a mathematical function that maps particle pairs to unique integers which will allow them to be tracked. The computational cost of this could also significantly impact the speed-up achieved on the GPU. Our recommendation is that GPU DEM be applied to problems that involve modeling the dynamical behavior of particles only where history independent contact models suffice. While this thesis focused on making efficient use of a single GPU, Blaze-DEM can be easily extended to make use of multiple GPUs. This is achieved by launching $\frac{N}{d}$ threads on each GPU where N is the number of particles and d the number of GPU devices. Each GPU will be initialized with particle and world objects at start-up, and only the 8 arrays that contains the particle dynamics information need to be transferred to all GPUs after each step.

Bibliography

[1] S. Green, Particle based Fluid Simulation, NVIDIA, 2008.

[2] P. Cleary, M. Sawley, Three-dimensional modeling of industrial granular flows, Proceeding of CFD in the minerals and process industries, CSIRO, Melbourne, Australia, 1999.

[3] R. Venugopal, R. Rajamani, 3d simulation of charge motion in tumbling mills by the discrete element method., Powder Technology 115 (2001) 157–166.

[4] O. Hlungwani, J. Rikhotso, H. Dong, M. Moys, Further validation of DEM modeling of milling: effects of liner profile and mill speed, Minerals Engineering 16 (2003) 993–998.

[5] D. Hiskey, Today i found out (2014).
URL `www.todayifoundout.com`

[6] N. Climent, Sand production simulation coupling DEM with CFD, European Journal of Environmental and Civil Engineering 18 (2014) 983–1008.

[7] H. J. Herrmann, S. Luding, Modeling granular media on the computer, Continuum Mechanics and Thermodynamics 10 (1998) 189–231.

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

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

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

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

[12] 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.

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

[14] 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.

[15] 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.

[16] H. Abou-Chakra, J. Baxter, U. Tuzun, Three-dimensional particle shape descriptors for computer simulation of non-spherical particulate assemblies, Advanced Powder Technology 15 (2004) 63–77.

[17] D. Hohner, S. Wirtz, V. Emden, H.K. Scherer, Comparison of the multi-sphere and polyhedral approach to simulate non-spherical particles within the discrete element method, Powder Technology 208 (2011) 643–656.

[18] 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.

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

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

[21] A. Munjiza, A combined finite-discrete element method in transient dynamics of fracturing solids., Int. J. Eng. Computation 12 (1995) 145–174.

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

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

[25] J. Longmore, P. Marais, M. Kuttel, Towards realistic and interactive sand simulation: A GPU-based framework, Powder Technology 235 (2013) 983–1000.

[26] M. Hromnik, Masters Thesis: A GPGPU implementation of the discrete element method applied to modeling the dynamic particulate environment inside a tumbling mill, University of Cape Town, www.uct.ac.za, 2013.

[27] B. K. Mishra, R. Rajamani, Three dimensional simulation of plant size SAG mills, International Conference on Autogenous and Semiautogenous Grinding Technology 31 (2001) 48–57.

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

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

[30] Shigeto, Sakai, Parallel computing of discrete element method on multi-core processors, Particuology 9 (2011) 389–405.

[31] J. Xu, et.al., Quasi-real-time simulation of rotating drum using discrete element method with parallel GPU computing, Particuology 9 (2011) 446–450.

[32] 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.

[33] J. H. Walther, F. Sbalzarini, Large-scale parallel discrete element simulations of granular flow, Engineering Computations 26 (2009) 688–697.

[34] Zhang, et.al., A fast scalable implementation of the two-dimensional triangular Discrete Element Method on the GPU platform, Advances in Engineering Software 60 (2013) 70–80.

[35] J. Anderson, et.al, General Purpose Molecular Dynamics Simulations Fully Implemented on Graphics Processing Units, Journal of Computational Physics 47 (2008) 1–17.

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

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

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

[39] Sanni.I, A reliable algorithm to solve 3D frictional multi-contact problems: Application to granular media, Journal of Computational and Applied Mathematics 234 (2010) 1161–1171.

[40] P. Pizette, N. Abriak, Particle image velocimetry analysis on 2D silo flows, Proceedings of Africomp 2015 Morocco, 2015.

[41] N. Govender, P. Pizette, D. Wilke, N. Abriak, Validation of the GPU based BlazeDEM framework for hopper discharge, Proceedings of the International Conference on Particle-based Methods 2015 Spain, 2015.

[42] N. Abriak, Local friction effect of the global behaviour of granular media, Mathematical and Computational Modelling 28 (1998) 121–133.

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

[44] N. Govender, D. Wilke, S. Kok, R. Els, Development of a convex polyhedral discrete element simulation framework for NVIDIA Kepler based GPUs, JCAM 270 (2014) 63–77.

[45] G. Neubauer, C. A. Radek., GPU Based Particle Simulation Framework With Fluid Coupling Ability, NVIDIA GTC 2014, San Jose,USA, 2014.

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

[47] N. Govender, D. Wilke, S. Kok, A GPU based polyhedral particle DEM transport code, NVIDIA GTC 2014, San Jose,USA, 2014.
URL `http://on-demand.gputechconf.com/gtc/2014/poster/pdf/P4126`

[48] B. Mishra, R. Rajamani, Numerical simulation of charge motion in a ball mill., 7th European Symposium on Comminution. 1 (1990) 555–563.

[49] M. Powell, The effect of liner design on the motion of the outer grinding elements in a rotary mill, International Journal of Mineral Processing 31 (1991) 163–193.

[50] M. Dennis, R. Rajamani, Evolution of the perfect simulator, International Autogenous and Semiautogenous Grinding Technology Proceedings 31 (2001) 163–193.

[51] J. Herbst, L. Nordell, Optimization of the design of SAG mill internals using high fidelity simulation, International Conference on Autogenous and Semiautogenous Grinding Technology 31 (2001) 150–164.

[52] P. W. Cleary, Recent advances in DEM modelling of tumbling mills, Minerals Engineering 14 (2001) 1295–1319.

[53] R. Morrison, P. W. Cleary, Towards a virtual comminution machine, Minerals Engineering 21 (2008) 770–781.

[54] P. W. Cleary, R. Morrison, Particle methods for modelling in mineral processing, International Journal of Computational Fluid Dynamics 23 (2009) 137–146.

[55] J. Alatalo, K. Tano, Comparing experimental measurements of mill lifter deflections with 2D and 3D DEM predictions, DEM5 Proceedings, Queen Mary University, London, UK, 1 (2010) 194–198.

[56] J. Favier, EDEM (2014).
URL `http://www.dem-solutions.com/`

[57] R. Rajamani, S. Callahan, J. Schreiner, DEM Simulation of mill charge in 3D via GPU computing, Proceeding of the SAG conference, Vancouver, 2011.

[58] N. Govender, D. Wilke, S. Kok, Collision detection of convex polyhedra on the NVIDIA GPU architecture for the discrete element method, Journal of Applied Mathematics and Computation http://dx.doi.org/10.1016/j.amc.2014.10.013.

[59] S. Koski, J. Vanderbeck, J. Eriques, Cerro Verde Concentrator-Four Year Operating HPGRs, Proceeding of the SAG conference, Vancouver, 2011.

[60] A. Macias-Garcia, Eduardo, M. Cuerda-Correa, M. Diaz-Diez, Application of the Rosin-Rammler and Gates-Gaudin-Schuhmann models to the particle size distribution analysis of agglomerated cork, Materials Characterization 52 (2004) 159–164.