Parallel Multiscale Contact Dynamics for Rigid Non-spherical Bodies

KRESTENITIS, KONSTANTINOS

Parallel Multiscale Contact Dynamics for Rigid Non-spherical Bodies

by

Konstantinos Krestenitis

A thesis presented for the degree of
Doctor of Philosophy

Innovative Computing Group,
Department of Computer Science
Durham University
United Kingdom

November, 2018

This thesis is dedicated to

my family

A Thesis Presented for the Degree of Doctor of Philosophy

Konstantinos Krestenitis

Submitted on
November, 2018

Abstract

The simulation of large numbers of rigid bodies of non-analytical shapes or vastly varying sizes which collide with each other is computationally challenging. The fundamental problem is the identification of all contact points between all particles at every time step. In the Discrete Element Method (DEM), this is particularly difficult for particles of arbitrary geometry that exhibit sharp features (e.g. rock granulates). While most codes avoid non-spherical or non-analytical shapes due to the computational complexity, we introduce an iterative-based contact detection method for triangulated geometries. The new method is an improvement over a naive brute force approach which checks all possible geometric constellations of contact and thus exhibits a lot of execution branching. Our iterative approach has limited branching and high floating point operations per processed byte. It thus is suitable for modern Single Instruction Multiple Data (SIMD) CPU hardware. As only the naive brute force approach is robust and always yields a correct solution, we propose a hybrid solution that combines the best of the two worlds to produce fast and robust contacts. In terms of the DEM workflow, we furthermore propose a multilevel tree-based data structure strategy that holds all particles in the domain on multiple scales in grids. Grids reduce the total computational complexity of the simulation. The data structure is combined with the DEM phases to form a single touch tree-based traversal that identifies both contact points between particle pairs and introduces concurrency to the system during particle comparisons in one multiscale grid sweep. Finally, a reluctant adaptivity variant is introduced which enables us to realise an improved time stepping scheme with larger time steps than standard adaptivity while we still minimise the grid administration overhead. Four different parallelisation strategies that exploit multicore architectures are discussed for the triad of methodological ingredients. Each parallelisation scheme exhibits unique behaviour depending on the grid and particle geometry at hand. The fusion of them into a task-based parallelisation workflow yields promising speedups. Our work shows that new computer architecture can push the boundary of DEM computability but this is only possible if the right data structures and algorithms are chosen.

Declaration

The work in this thesis is based upon research carried out at the Innovative Computing Group, Department of Computer Science, Durham University, United Kingdom. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Statement of Copyright

Acknowledgements

The following PhD research would not have been possible without the funding from both the Engineering and Physical Sciences Research Council (EPSRC) and EDF Energy Generation (Dr. McLachlan Neil and Dr. Steer Alan).

I would like to thank Durham University for the supercomputing resources and technical assistance. This work was made possible with the use of Durham University's local HPC facilities Hamilton and Intel Xeon Phi co-processor cluster. I appreciate the support from Intel through Durham's Intel Parallel Computing Centre (IPCC) which gave me access to latest Intel software. I also made use of the facilities of N8 HPC provided and funded by the N8 consortium and EPSRC (Grant No. N8HPC _DUR _TW _PEANO). The Centre is coordinated at the time by the Universities of Leeds and Manchester. I'm also grateful for the access to ATHOS HPC resources provided by EDF R&D in France and Advanced Research Computing High End Resource (ARCHER) HPC services at the Edinburgh Parallel Computing Centre.

In addition to the available teaching resources at Durham, I'm grateful to have been invited to visit the International HPC training in Ljubljana, Slovenia. The experience and the knowledge I gained there is invaluable. The visit was only possible because of the sponsorship from The Partnership for Advanced Computing in Europe (PRACE).

I would like to thank Dr. Tobias Weinzierl and Dr. Tomasz Koziara for their time and effort to supervise me. Their support and guidance in all these years have helped me understand many concepts that would not be possible without them. Moreover, the conferences and hiking trips will always be something to remember for the future. I'm grateful to have met Tomasz Koziara in Oxford and I'm thankful to have a glimpse to his perspective on many aspects.

Finally, thank you to all the people that supported me during my time at Durham University. To all the people at the department, St. Chad's college, all the friends and family, I express my gratitude.

List Of Published Manuscripts Detailing Parts of This Thesis

Peer Reviewed Publications

Konstantinos Krestenitis & Tobias Weinzierl (2018). A Multi-Core Ready Discrete Element Method With Triangles Using Dynamically Adaptive Multiscale Grids. Concurrency and Computation: Practice and Experience

Krestenitis, Konstantinos, Weinzierl, Tobias & Koziara, Tomasz (2018), Fast DEM collision checks on multicore nodes, in Wyrzykowski, Roman, Dongarra, J. J., Deelman, Ewa & Karczewski, Konrad eds, Lecture Notes in Computer Science, 10777 12th International Conference on Parallel Processing and Applied Mathematics (PPAM) 2017. Lublin, Poland, Springer, Cham, 123-132.

Krestenitis, Konstantinos, Weinzierl, Tobias & Koziara, Tomasz (2016), A Contact Detection Code using Triangles for Non-Spherical Particle Simulations, 24th Conference on Computational Mechanics (ACME-2016). Cardiff, UK, 227-230.

Krestenitis, Konstantinos & Koziara, Tomasz (2015), Calculating the minimum distance between triangles on SIMD Hardware, in Gil, A. J. & Sevilla, R. eds, 23rd Conference on Computational Mechanics (ACME-UK 2015). Swansea, UK, Swansea University, Swansea, 55-58.

Software Guidebook

Krestenitis, Konstantinos (2018), Delta Guidebook: A Discrete Element Method Library for Tessellated Geometries. Durham University. Available at: http://www.peano-framework.org/delta/guidebook.pdf

Contents

List of Algorithms

Nomenclature

Delta-DEM Notation

Δt	Delta Step Distance
δ	Triangle/Particle-Pair Distance
κ	Spring Dash-pot Value
$\mathbb{C}$	Total Contacts
$\mathbb{T}, \mathbb{T}_A$	Total Triangles, Triangles in Particle A
ω	Angular Velocity Vector
ε	Epsilon Margin of Particle or Triangle
A, B, C	Vertices on a Triangle
A, B	Particle A and B
BF	Brute Force Method
DEM	Discrete Element Method
dt	Delta Step Distance
eps	Epsilon Regularisation Parameter
f	Force vector
hes	Hessian Matrix
$Hybrid$	Hybrid Method
I	Identity Matrix
m	Mass
N	N particles or N bodies
n	Contact Normal
P	P Point of Shortest Distance from Particle A
p	Particle Position
$Penalty$	Newton Penalty Method
PID	Particle ID
PQ	Distance δ Between Vertices P and Q
PT	Point to Triangle Distance
Q	Q Point of Shortest Distance from Particle B
r	Penalty Parameter
$SEGSEG$	Line Segment to Line Segment Distance
t	Torque vector
T_i	The i^{th} Triangle of a Particle

TID	Triangle ID
tol	Tolerance
TTD	Triangle-Triangle Distance
v	Linear Velocity Vector
v_t	Tangent Velocity Vector

HPC Abbreviation

ALU	Arithmetic Logic Unit
AMR	Adaptive Mesh Refinement
AoS	Arrays of Structures
ARCHER	UK National Supercomputing Service
AVX	Advanced Vector Extensions
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
DEM	Discrete Element Method
GPU	Graphics Processing Unit
HPC	High Performance Computing
KNL	Knights Landing
LB	Load Balance
MD	Molecular Dynamics
MPI	Message Passing Interface
NUMA	Non-Uniform Memory Access
OpenMP	Open Multi-Processing
PIC	Particle In Cell
SIMD	Single Instruction Multiple Data
SMT	Simultaneous Multi-Treading
SoA	Structures of Arrays
TBB	Intel's Thread Building Block

CHAPTER 1

Introduction

Discrete Element Methods (DEM) are techniques that model granular flows, the break-up of brittle material, ice sheets and many dynamic other phenomena that occur during particle collisions. The method describes the medium of interest as a set of rigid bodies that interact through collisions and contact points. The expressiveness of such a simulation is determined on the one hand by the accuracy of the physical interaction model. On the other hand, it relies on the accuracy of geometric scale. The more rigid bodies (particles) that can be simulated the more accurate and realistic is the outcome.

DEM is found in a wide range of engineering applications where the better understanding of a medium's behaviour is important. Granular rock aggregates, sediments or powders can be found in many disciplines such as bio-medicine, manufacturing, process engineering, tribology, machinery design. The models applied to simulate the aforementioned engineering applications focus on the simulation of contact behaviour between particles within a process (e.g. hopper, conveyor belt, shaking table, powder mixing). Successful simulation of particle processing supports the understanding of phenomena like landslides, avalanches, engine design (i.e. moving components that interact), aggregate processing, powder technology, rock blasting, planetary rover mobility (i.e. wheel-soil interaction). Non-engineering applications of DEM is commonly found in video game programming and animations.

The DEM simulation is often designed as a three-phase algorithm. The first part is the collision detection phase, interactions between particles are defined. The second part is the force derivation phases where the interaction forces are updated

according to the Newton's law. Lastly, particle positions are updated according to the interaction forces at hand. Most DEM simulations stick to explicit time integrator (cmp. [5] and references therein). A DEM model defines each of these algorithmic phases, within these phases we describe the behaviour of a system of particles for a given engineering scenario.

The improvement of DEM models and the reduction of simulation time matters in engineering applications. Advancement of the model promises to increase physical accuracy and a reduction to the time-to-solution for a given setup. Physical, geometric accuracy and the total number of particles utilised within a system are the determinant factors for computational complexity. For the physical representation, the appropriate mechanics are applied to simulate interactions of interest (e.g. lubrication, contact elasticity, friction). Realistic geometric representation of the mediums at hand is vital and inseparable to the choice of contact models (physical accuracy). A betterment of the represented geometry supports the case of a realistic simulation. Finally, as granular engineering applications require large number of particles to simulate certain dynamic phenomena (e.g. land slides), the increase of particles adds to the complexity of the simulation. Subsequently, any reduction in computational complexity increases an engineer's capacity to run a growing number of case studies at higher degrees of accuracy.

There are two important shortcomings in DEM interaction models: particle geometry and particle scales. For the particle geometry most simulation codes use only spheres and particles, uniformly sized and too few of them [3, 98]. The sphere based geometries minimise contact complexity requirements by employing a simplistic contact model. Most of the time spheres form the simplest form of granular representation. DEM codes restrict themselves to analytical shape models where particles are described by some analytical function. As simulations are constrained by the computational cost that comes with increased levels of complexity, interaction phenomena are only approximated according to the computational resources at hand. Therefore, the use of spheres to resemble physically non-spherical particles is a compromise between physically accuracy and computational cost. In large scale simulations it is common to neglect the underlying geometric error found in sphere-based geometries in favour of computational feasibility [65, 101]. Such design decision is reasonable for applications where users are interested in general macroscopic and statistical quantities. In these cases it is anticipated that geometric error is controlled by some alternation of the contact model. For example, spheres need to exhibit higher degrees of friction to simulate non-spherical particle contact (i.e. non-spherical particles can have more than one contact point). Although ef-

forts to decrease geometric approximation errors do increase the number of physical phenomena that can be represented, they are occurring at the expense of computation. The challenge is to increase physical realism to do better engineering but also manage the computational cost that comes with it.

The second shortcoming is found in the difference in particle size. Particle dynamics of geometries at different orders of magnitude in scale are more complex to implement than those that are uniformly sized. The scale difference in geometries has an impact in both the model (e.g. density difference) and in the data administration which can translate more complex interactions (i.e. a large particle spans over a larger area of interaction than a small particle). Multiscale particles distributions impose special design decisions over space discretisation which furthermore adds to the computational complexity [11, 36] of the algorithm.

It is important to tackle the shortcomings as they represent a significant subset of engineering applications (e.g. rock aggregate analysis). The introduction of both multiscale particles and non-spherical particles increases the realism of the model which can lead to representative particle behaviours. Whenever particles are close to each other (i.e. their distance under runs a given threshold), they are assumed to be in contact. The distance between two particles depends on their geometry. For spheres a single contact is created while in arbitrary geometries this not necessary true. Sphere-based models mitigates the fact that real particles are not sphere-based [41, 52, 66, 97]. Once explicit time-stepping is used, exact contact becomes impossible as we have to discrete time. Collision between different particle scales require more sophisticated interaction models in place.

Collision detection is a critical DEM phase, during this phase the code computes the distances between particles to determine if there is a contact. The use of spheres for contact detection in high performance computing environments is widely understood and implemented [3, 5, 7, 28, 33, 54, 66]. Whereas algorithms that do describe surfaces as polyhedrons fail to perform contact point generation robustly [38, 101]. Moreover there is lack of work on non-spherical elements that is include on non-convex geometries. The reason for this, is the requirement for additional computational costs and interaction models [28, 38, 63, 72, 74] to suit the underlying geometry. Contact point generators that rely on analytical polyhedron shapes often use variants of the Gilbert Johnson Kerthi (GJK) [101]. The GJK algorithm notoriously requires the interpenetration of the shapes which causes contact divergence issues [38]. Therefore we find reason to investigate the contact problem, by employing triangulated meshes to describe geometries and define contacts during collisions.

Triangulated meshes in terms of geometry are ideal for contact mechanics simulations because complex multi-facade shapes can be described using triangles. Spherical object representations are an exception because they are best described as sphere-based elements instead. Multi-facade objects are reduced to triangular descriptions that make up a mesh-based geometry. Since triangles are chosen as the primary element in meshes, we investigate the contact model based on triangles. A single contact point between two triangulated particles is the middle point of the minimum distance between all pairs of triangles. The state-of-the-art interaction potentials (forces) of two particles are parameterised over particle distances. Thus collision detection in DEM is reduced to the computation of the minimum distances between elements (triangles).

Faced by the challenges of geometric and computational complexity, time-to-solution in a simulation increases. A common practice to decrease the time-to-solution in DEM is to develop parallel DEM routines that exploit higher throughput of modern computational hardware. Parallel computation through classic domain decomposition is well understood and the codes scale [44], but most codes refrain from modelling particles as irregularly shaped objects. They eliminate the second role of the interaction model — geometric accuracy — and spend the majority of compute time in collision detection. Iglberger et al. [44] report 31–34% within a multi-physics setting, while Li [55] for example reports even 90%. Collision detection becomes significantly more complicated once we switch from sphere-to-sphere or ellipsoid-to-ellipsoid contacts to the comparison of thousands of triangles that represent meshed geometries. Notably, contact complexity is increased when no constraints are imposed on sharp featured particles or compounds of simpler convex shapes.

The injection of meshed particles into DEM is a single node challenge. Up-scaling a DEM code with respect to particle count and machine size is a non-trivial [44, 48] task. Computation between triangle pairs costs more than sphere pairs. Moreover, multiscale simulations that use triangle-based particles face additional complexity in coarse-fine scale interactions that arise of the auxiliary data structures and data communication. Thus the computation increases remarkably with any increase of particle count. The scalability of this is in turn not given by construction.

Recent developments in central processing unit architecture [25, 32, 67, 79, 85] enable the support for Single Instruction Multiple Data (SIMD) [85] data level parallelism. SIMD parallelisation in principle allows for four (double precision) and eight (single precision) times speedup on compatible CPU cores. These speedups are possible on modern architecture cores that support wide registers (256, 512

bits) with new hardware instruction sets (SSE, AVX2). In addition to single core speedups, processors nowadays host multiple cores. It is vital to exploit all up-to-date resources to maximise the number of particles we can afford to handle. In this project we propose techniques that exploit such modern hardware motivated by the handling of triangular DEM meshes.

Research conducted in the area of DEM contact detection — to the best of my knowledge — does not comprise studies on data parallelism of triangle-to-triangle distance computation for contact detection. Neither is there work that uses non-analytic methods to this geometric problem. We propose a new set of methods for the computation of the minimum distance between triangles.

Our proposal on computing the minimum distance between two triangles combines two strategies; one is the naive or brute force approach where the geometric primitives of the triangles are exhaustively compared. Geometric primitives of triangles include segments, points and the triangle plane. The alternative approach is to analytically describe the triangles based on their barycentric coordinates and then to create a non-linear minimisation. The alternative iterative method is more favourable in terms of its computational complexity. Compared to the brute force method, the iterative method is less prone to data dependency and due to its iterative nature, it is possible to avoid logical branching during computation.

The iterative approach is the vectorisable and ideal for exploiting the available architecture. However brute force approach is the robust method. It always yields correct results. Iterative variants may not converge because of ill-posed triangle configurations. Both strategies exhibit different memory footprints which led to the contribution of a new hybrid approach that is both robust and fast.

We additionally contribute to multiscale grid based DEM. The computational domain is discretised based on a Cartesian grid and a spacetree hierarchical depth first traversal. We propose a new scheme that promises to map DEM algorithmic phases onto a multiscale grid traversal. The grid give rise to four types of morphologies that exhibit various computational behaviours over time.

We show that the distinct morphological behaviour of the adaptive grids is vital to a grid based approach in DEM. Adaptive grids significantly reduce particle to particle comparisons. We introduce a new type of grid that is reluctant adaptive. We show that the reluctant adaptive grid variant minimises the required morphological grid changes as the particles move around the domain. This in effect reduces the adaptivity overhead per time step as we only refine when it is absolutely necessary based on the dynamics.

Moreover we show that the reluctant grid when combined with an adaptive time

stepping can yield better performance than the regular adaptive grid scheme. This is because the adaptive time stepping scheme makes use of grid information to both refine step size and coarsen it. As such we propose a simple scheme that minimises the number of steps required to reach the termination condition.

In terms of shared memory parallelisation we propose a novel task-based algorithmic scheme that is derived from three separate levels of abstraction. At the finest level, we base our computation on the proposed triangle-to-triangle comparison contact detection model where computation is issued as standalone tasks. At a coarser level, a particle-to-particle parallelisation level at clusters of grid vertices help us launch pairwise particle comparisons. The particle-to-particle level parallelisation is proven counter-intuitive as grid adaptivity is introduced to minimise particle-to-particle comparisons. So we utilise cell-to-cell concurrency to both launch and consume task-based workload in parallel. The launched contact detection tasks are intermixed with grid traversal task routines. We finally discuss scalability results for various runs.

The presented research contributes to areas of HPC DEM and we propose several new methods for contact interaction between rigid bodies. The proposed method differs from existing methods found in literature in terms of computational complexity, the implementation methodology, geometry and numerical output. We do not focus on several aspects of DEM due to both time constraints but also to minimise the overall complexity. We do not contribute in numerical approximation techniques neither in DEM contact mechanics or numerical validations against physical experiments. In terms of the methodological limitations, we clearly identify errors and possible alternative routes to be explored in the future.

The thesis is organised as follows: In Chapter 2 (Physical Model), we sketch the physical model used in DEM to represent bodies, generate contacts and forces. We pinpoint the importance of contact detection and discuss geometric representation of physical bodies found in mainstream DEM simulations. An enhanced meshing method using a boundary layer is discussed which promises to improve contact point definition and contact robustness. In Chapter 3, we present a preliminary study of DEM. The study showcases that for standard benchmarks the underlying geometry of the interacting bodies impact the end result of a simulation. Small changes to the particle shapes and size distributions conditions lead to different results. We use the example of particles flowing through a hopper. In Chapter 4, we discuss the recent developments in computer architecture and global trends towards exascale computing. According to HPC roadmaps, these developments create new challenges for developers. In terms of DEM we extract challenges that affect development and

the realisation of large-scale particle simulations on large machines. We showcase opportunities that do arise in contact dynamics simulations and methods that exploit new hardware architectures. In Chapter 5 (Algorithm Outline), we discuss the outline of our novel DEM algorithm. The chapter covers a brief overview of explicit time integration and collision detection routines for non-spherical particles. In Collision Detection we discuss the algorithm that defines contact points between two particles and the derivation of interaction forces from collision points. We cover the analytical contact detection approach for non-spherical particles and discuss in detail our contribution towards a new set of iterative contact detection methods. New hybrid methods are introduced that exhibit the advantages of both iterative and exact, comparison based solvers. Moreover we study the memory layout required by such new solvers according to new trends in hardware. We discuss optimisation techniques and performance benchmarks for comparison of the new algorithms. In Chapter 6 (Grid Meta Data Structure), a grid meta data structure for DEM is introduced [92]. The grid discretises space to reduce the overall complexity of collision detection. The grid data structure is coupled with the DEM algorithm phases and maps particles into a cascade of grid cells. The mapping of particles into a grid structure provides better access locality and out of the box parallelisation of particle traversals events. The morphology of the grid depends on the underlying dynamics and can be adaptive. Two types of space adaptivity are introduced and compared. Multiscale grids resulting from the cascades of cells give rise to new implementation challenges in DEM. We discuss techniques that implement a functional multiscale collision detection and topology in three dimensions. In Chapter 7 (A Dynamic Time Step Scheme), we propose a new algorithm for time-based adaptivity. We discuss a time stepping routine that dynamically refines or coarsens time step sizes as particles are approaching or separating. The adaptive time stepping significantly reduces the overall time-to-solution. Moreover the algorithm introduces stability to the system as a dynamic step size ensures that no particle collision is omitted. In Chapter 8 (Manycore Concurrency), we discuss four new variants of DEM parallelisation: mesh level, particle level, grid level and grid-task level. We present results of shared memory measurements as well as the best mix of variants. We use all algorithmic ingredients to run benchmarks that reveal the potential and impact on the hardware. Finally, the thesis closes with the conclusion of lessons learned and the future outlook.

CHAPTER 2

Physical Model

Introduction. A DEM (Discrete Element Method) physical model in principle is given by the geometric representation of different particles and a contact model. Then a differential equation of motion is subject to a numerical scheme. The DEM geometry comprises the geometric elements (e.g. spheres, triangles) that represent the real bodies in a simulation. In a DEM context, there are three major types of errors that occur: the numerical, the precision, and the geometric error. The numerical error is dominated by the stiffness in the explicit time integration scheme which is typically used in DEM codes [39, 81]. The precision error originates from the finite precision in computer hardware. Lastly, the geometric error which that is produced from the discrete geometric description of the real world. This thesis tackles the geometric error.

Literature review. The choice of the geometry is critical to simulate any dynamic interaction between objects as most for the computation and modelling is derived from the geometry [33, 39, 41, 46, 52, 66, 81, 97]. The collision model implements the contact points detection. The contact points are fed into the force model to yield the actual interaction forces and therefore motion dynamics. In accordance to Newtons' laws of motion the force decomposes into repulsive and frictional forces. All critical phases of the DEM simulation are driven by the geometry.

For this project, we use triangulated geometries to approximate granular and non-smooth objects that feature sharp edges. This is a pivotal choice compared to models that use spheres as it adds significant computational and modelling overhead [83, 88]. To render the simulation nevertheless feasible we design a new contact detection algorithm for these geometries. While we make a contribution on the geometry side, we restrict ourselves to an explicit Euler time integration scheme for simplicity.

Chapter Outline. The physical model lays the theoretical base of the Discrete Element Method. The Chapter breaks down the method into the following sections: Geometric Representation, Contact Point Generation and Contact Forces. We start with an overview of geometric representations found in DEM. The Contact Point Generation section discusses a newly proposed method to generate contact points between two triangulated bodies. Lastly, the section Contact Forces provides an overview of the rigid body force model that naturally arises from these contact points.

2.1 Geometric Representation of Rigid Bodies

The geometric approximation of a body defines the surface features of particles that are used in a simulation. In particle engineering the most widely used geometric primitive is the sphere and analytical shapes [40, 52, 53, 72, 101]. Bodies with sharp corners are best described as polygons [53]. In DEM-based particle simulations the employed geometric element is the determinant factor for geometric error. The geometric error is defined as the general geometric discrepancy between reality and simulation. The choice of elements that approximate a rigid body surface directly affects the realistic surface representation, the choice of contact model, the dynamics and the cost in computational complexity to simulate all of the above.

Different case studies dictate different choices of geometric shapes. At the molecular level spheres and ellipsoid shapes are predominately used in chemistry and biological studies. Spherical shapes at the molecular level commonly represent objects like blood cells, micro-organisms or micro-structures. At the macroscopic scale, spherical geometries lose their dominance as real geometries feature sharpness. Buildings, ridges, rock formations, organisms, minerals and many more objects give rise to complex non-spherical geometries. The discrete element method is found applied to a range of engineering fields (e.g. milling, granulation, blending, seismic analysis, tribology) where bodies are predominately irregular and non-smooth. In many cases, multi-body structures (e.g. robots, engines, assemblies) and surfaces of particles like rock aggregates, structural rods and grains particles are characterised by sharp features that are difficult to capture with smooth spherical elements. It is vital to accurately represent shapes in order to reproduce realistic behaviour and capture kinetic phenomena (e.g. rotation).

> The focus of this project is to simulate contact dynamics of particle geometries and structures at the macroscopic scale.

Macroscopic particles are nevertheless often simulated with spherical shapes because these are computationally cheaper than polygons [88]. Codes that employ sphere shapes can approximate non-spherical shapes by an assembly of spheres. In a sphere-based scenario, there are two critical considerations: firstly the geometric approximation error is to be kept under control, secondly interaction models are necessary to anticipate physical phenomena that occur in non-spherical geometries.

Unlike perfect spheres, non-spherical particles' geometries can be classified by a variety of shape factors (e.g. sphericity, crosswise sphericity, lengthwise sphericity,

[1] By US Department of Energy [public domain], via Wikimedia Commons

Figure 2.1: Photographs of granulate examples. Soil granulates can be found in nature in many shapes. Top: Gravel aggregate frequently used in construction. Left: A Tunnel boring machinery shield. Rock jamming can occur during phases of drilling, crushing, extraction. Right: Movement of rover Curiosity on Martian soil granulates. Non-spherical particle interaction (rolling resistance, particle displacement, bulldozing effect) with rover wheels is an important research area in the field of terra-mechanics and robotics where soil samples are not widely available for laboratory experiments. [1]

particle circularity, corey shape factor, drag shape factor, volumetric shape factor roundness, aspect ratio, particle shape factor) [101]. Depending on these factors, non-spherical particles require special interaction models [101] such as friction coefficients which then take into account that the real geometry has flat surfaces and sharp features. The surface area under contact in non-spheres can be larger than that of sphere-based interactions, as a pair of spheres can only collide at a single point. As such sphere-based contact models that model non-spherical collisions are altered to exhibit the analogous mechanical behaviour of particles with sharp features. To increase physical accuracy and to enable the simulation of sharp edged particles, we employ triangulated particles instead of spheres. The change in the DEM geometry promises to increase the simulation realism while relying on first principle physics. This aspect of geometry forms the basis of my DEM contribution.

Particle Shape	Pros	Cons
Sphere	Simplest to implement; requires the minimum CPU resources to describe a particle so it allows a maximum number of simulated particles.	Particle behaviour is unrealistic because of simplified shape.
Rigid sphere cluster	Relatively simple to implement. Approximates real particle shapes.	In order to approximate a real particle shape, many spheres of various sizes are required, significantly increasing CPU resources.
Ellipsoid	Better matches real particle shapes. Contact force detection has some similarities to that of a sphere. CPU resources required are comparable to that of a sphere.	Contact force detection is more complicated than that of a sphere. The ability to match real particles is much better than that of a sphere but is still limited. Contact detection is more difficult than for a sphere.
Superquadric	There are more simulated particle parameters in a closed form description which can better describe shapes. The particle parameters are analytically computed.	Similar difficulties found as in ellipsoids and contact detection and force derivation is difficult, in particular for concave descriptions.
Poly-ellipsoid	It is a subset of a eight-quadrant superquadric. No surface gradient discontinuities as such no sharp ridges/edges.	Contact detection problems are more severe than that of ellipsoids.
Polyhedra	The polyhedron-based shape is the most general and the most applicable to describe complex geometries.	Analytical contact detection methods are very difficult. Contact force derivation at edges is non-trivial.

Table 2.1: Comparison table of commonly employed geometric shapes to specify the physical geometry of particles [53]
.

DEM simulation codes that feature complex geometries take into account several considerations regarding the geometry representation. As specified by the particle shape overview Table 2.1 found in literature [2, 40, 46, 52, 53] there are pros and cons in the choice of a geometric shape. The most simplistic form, the sphere-based model utilises minimal resources as both the mathematical contact model and geometry is simplified. The reduction in model complexity exhibits unrealistic simulations. More complex shapes (see superquadrics, polyhedras in Table 2.1) offer more accurate physics but at the cost of computational complexity. The choice of shape has a direct implication to geometric approximation error, computational resource utilisation and contact model.

There are several unique properties in non-spherical particle. Polygonal elements that represent non-spherical granulates improve the volumetric approximation error which is important to compute the precise body mass. This allow the dynamics to display a range of kinematic phenomena that would otherwise be impossible to capture. A important feature of non-spherical particles is the location of the centre of mass. The centre of mass in a non-spherical particle affects the simulated notion of rotational phenomena. If a rotating object is asymmetric around its principal axis of rotation (i.e. centre of mass), then the moment of inertia towards each coordinate direction changes over time as angular velocities slow down with gravitational forces. A change in the principal axis of rotation creates a wobbling effect that is not possible to capture in a spherical particle because of its symmetrical mass distribution. A non-spherical particle allows for one or more arbitrary number of contact points to occur per interaction. Thus the coexistence of flat surfaces and non-smooth geometries allows capturing the effects of sliding and rolling frictional forces which could in principle utilise more than one contacts.

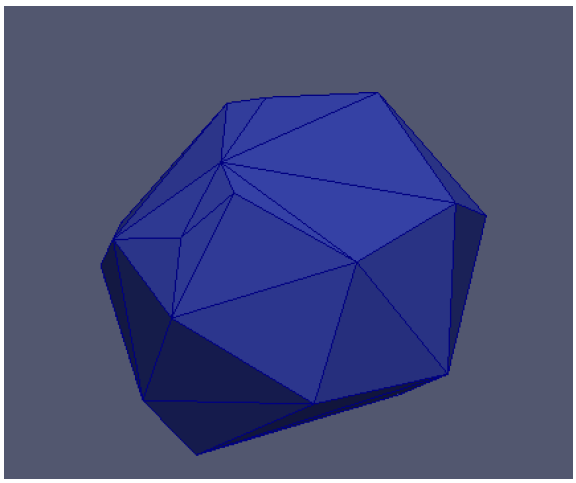


Figure 2.2: A non-spherical triangulated particle approximating a granular material (i.e crushed gravel aggregate).

The focus of this project is to create algorithms that allow users to work with triangulated non-spherical particles of arbitrary shape.

A second important aspect of DEM granular particles is their variation in scale. Extreme variation in particle scales occur in engineering applications (e.g. filtering, rock crushing, milling) [58]. Particles in some applications are not equally sized and there are cases of interaction where a larger structure (e.g. hopper, floor, machining, crusher) interacts with smaller particles. So a simulation should accommodate both coarse and fine particle sizes. Coarse-fine particle interactions yield important engineering phenomena like clogging, blocking, void filling, and granular flow rate variation. In terms of computation, heterogeneity in particle scales create heterogeneity in the data handling. The intermix of various scales of particles length add to the complexity of computation.

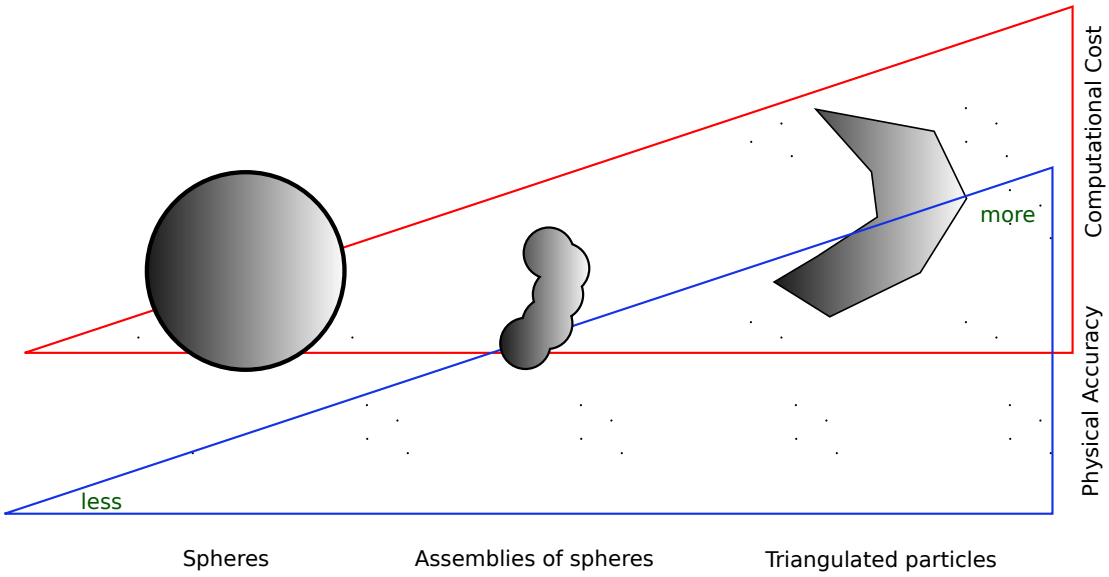


Figure 2.3: The relationship between computational cost and physical accuracy of geometries. Particles that are constructed using spherical shapes are more computationally feasible to simulate than those using triangulated shapes.

The geometry choice determines both the computational cost and physical accuracy of a DEM algorithm. As shown in Figure 2.3 and Table 2.1, the more geometric accuracy the more computationally complex and slow would be the DEM simulation. Geometric representation plays a crucial role in the representation of actual real world physics. The choice of geometry for a particular body affects both the

physics and the computational cost (Figure 2.3). With triangulation of particles we increase complexity in favour of geometric and model realism. Spherical particles are commonly used in DEM applications for algorithmic simplicity and the lower computational cost. Yet spherical particles cannot represent sharp edges without adding significantly larger number of spheres to approximate irregularities (e.g. sharp edges). The geometric approximation error that occurs with spheres that attempt to represent non-spherical particles does not allow the capture of important physics like granular rotation, non-smooth particle contact without fundamentally changing the spherical physical model to adapt to the requirements. It is therefore logical to employ triangulated bodies of particles to represent the non-spherical surface instead of spheres. Another dimension in DEM geometry is that multiscale interactions add additional complexity in order to enhance geometric realism.

2.2 Contact Point Model

Contact point generation. Collision detection and collision point generation are the prerequisite phases to derive the interaction force between two bodies. A collision is detected when two particles are in a "touch" state. The definition of "touch" or collision varies per DEM contact model. Contact detection for non-spherical particles varies widely [43, 100, 101]. It is also not uncommon to utilise clusters of spheres to approximate sharp featured geometries [2, 3, 5, 15, 66, 81, 101]. Rigid bodies collision detection algorithms distinguish themselves in overlapping and non-overlapping schemes. The codes that implement overlapped bodies allow geometries to cuddle over their real shape, GJK (Gilbert Johnson Keerthi) is found to be the predominantly used method for non-spherical particle collision detection in literature [27, 55, 88] (See Appendix Section GJK). Non-overlapping codes use a virtual geometry such that the real geometry is not allowed to overlap. In the latter case particles are in contact once they are closer than a prescribed critical threshold. The critical threshold is formed by an Minkowski sum around each particle (Figure 2.4) that creates a virtual or halo extension of the real geometry.

Contact models that do allow geometries to overlap over the real geometry make sure that the overlap gap is minimised [37, 64]. An explicit check of penetration rate or a fall back scheme is often put in place to control critical overlap. For non-overlapping schemes, the extra margin acts as a safe-guard area and allows for robust resolution of contact without relying on a fall back. Even if the real geometries do not overlap in a halo extension scheme, the halo extension does overlap. The minimisation of the gap in both cases goes hand in hand with the time step size

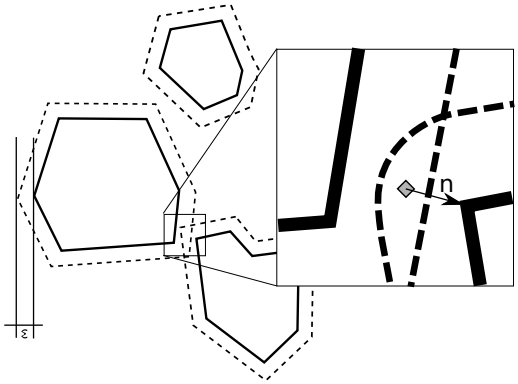


Figure 2.4: Three particles with their ϵ environment. The particles do not penetrate each other, but two particles plus their ϵ Minkowski sum penetrate and create one contact point with a normal.

and the force parameters. The infinitely short contact event thus is never resolved exactly.

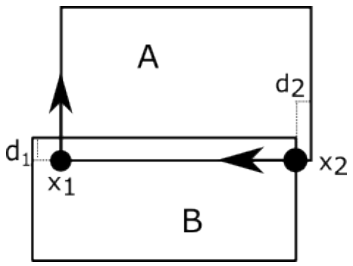


Figure 2.5: Contact divergence due to interpenetration. Convex bodies A, B with x_1, x_2 contact points and d_1, d_2 penetration depths. In the next time step object A will move upwards to the right and object B downwards left.

When a collision is detected, the collision overlap region is extracted and a unique contact point is defined. The extraction of the overlap region assume no mesh-based redundancies and thus should produce a single contact point. It is important that for each pair-wise collision the contact model produces unique contact points to avoid collision jitters and divergence [38]. Collision divergence occurs when particle geometries overlap and this can create contact normal's that are inconsistent (Figure 2.5. Due to discretisation errors (i.e. meshing) and geometric symmetries, contact point inconsistencies may arise [38], these need to be explicitly resolved by contact filtering or clustering [28, 49, 60]. Such models apply constraints at contact point generation or resolve mesh errors a priori to the simulation. Instead, in this project we choose to implement a contact detection model that uses virtual layer halo to detect collision [38] to increase contact robustness. In our scheme real geometries do not overlap, yet an extended geometry does overlap.

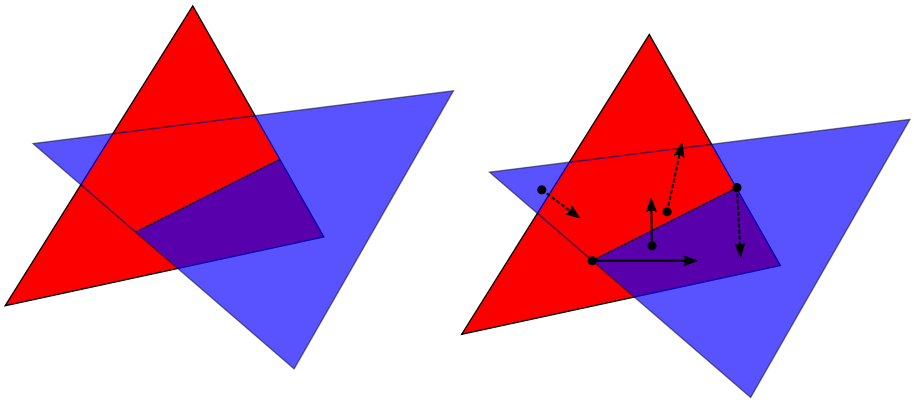


Figure 2.6: Example of triangles that intersect without boundary layers; the direction of shortest retraction is not consistent. The contact points are not robust as they diverge the collision.

An alternative collision detection approach is proposed where individual triangles of the mesh are used to determine collision between two bodies. Equipped with this method, non-overlapping collision detection contact divergence is avoided and a new contact model for unstructured meshed particles is proposed. Unstructured meshes represent geometries and are combined with an non-overlapping triangle-to-triangle contact detection approach. The idea of unstructured meshes for contact detection in DEM is inspired from robotics simulations [38]. In such an approach, a robust contact generation technique is possible when the boundaries of the bodies are fattened with a margin around them. This is especially useful for meshes that include features with noise. The boundary layer margin acts as a restriction constraint to the penetration depth during contact. The method is applied to rigid-bodies with the primary goal of producing stable (Figure 2.7) contact estimates for mesh-to-mesh collisions while tolerating the numerical error that comes with the introduced margin. However if triangles do penetrate, the simulator loses robustness (Figure 2.6). That is because of the undefined contact point, that can no longer be defined due to the penetration contact divergence problem, rendering impossible to pass the required contact normal information to the next time step of the simulation.

The boundary layer is created by extending the surface geometry by an epsilon value. For example, a set of triangles T_A in a mesh m_A, is covered at the boundary by the union of spheres of radius equal to epsilon ϵ, this forms a Minkowski sum and results to fattened triangles (Figure 2.8) at the edges. The extended bodies m_A and m_B (or their sets of triangles T_A, T_B) are in contact when the ϵ extended area of a pair of triangles T_i, T_j (of bodies m_A, m_A) overlaps. A function $d(T_i, T_j)$ is used to compute the exact minimum distance d between triangles, thus we check if $d(T_i, T_j) \leq \epsilon_A + \epsilon_B$. All possible pairwise triangles within distance $\epsilon_A + \epsilon_B$ are

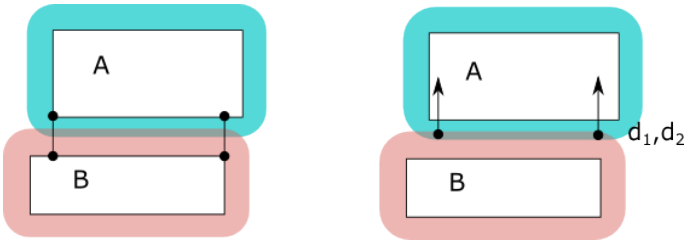


Figure 2.7: Contact divergence is avoided by introducing boundary layers in the two bodies, prohibiting penetration making contact point generation robust. Left: figure show contact points on A,B. Right: figure show contact normal caused by contact. d_1, d_2 are the boundary margin penetration depths. [38]

then used to produce contact points $\mathbb{C}$. Such an algorithm has the complexity $\mathcal{O}(|\mathbb{T}_A| \cdot |\mathbb{T}_B|)$.

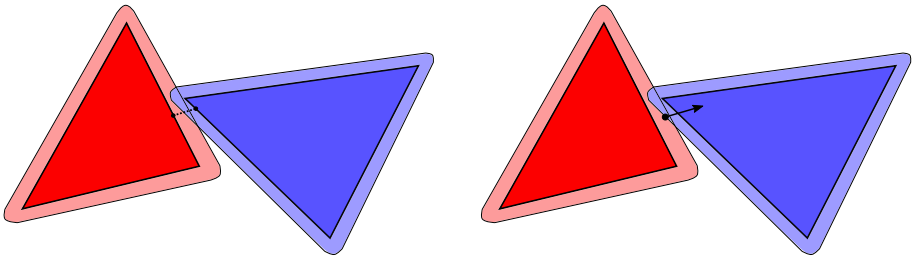


Figure 2.8: Contact points (left) and contact normal (right) with a boundary margin layer applied on triangles produces robust contacts.

The closest points on the triangles correspond to the deepest penetrating points of the extended triangles (Figure 2.8). The triangles are in contact when for all pairs of triangles there is a pair whose distances are less than the sum of the extended margin. This contact detection method is reduced to calculating the minimum distance between all pairs of triangles. To our knowledge there is only one strategy to obtain the distance between a pair of triangles robustly: a brute force method checks the distances of the triangles primitives, vertices and segments to determine a contact (Section 5.4).

The contact point is the centre of the overlap region (Figure 2.4). The distance d between a particle A and its contact point with a second particle is $0 < d \leq \epsilon$. Each contact point is equipped with a normal n pointing from the contact point to the surface of the contacting body's surface. As the distance is positive, we have $|n| \leq \epsilon$. Note that the normal direction depends on whether we read particle A is being hit by particle B or the other way round, each normal associated with a particle points along the outer normal of the body. Despite the fact that we employ a rigid body model, multiple contact points between two bodies may exist as particles can be concave and their Minkowsi sum may overlap in several places.

The Minkowski sum margin around the geometry allows the contact model to shift the penetration test onto a virtual boundary layer. A virtual boundary layer ensures that no real penetration is allowed and as a result prevents contact divergence found in penetration-based models [38]. A contact point is defined as the middle point of the minimum distance between two particles. Contact normal's point opposite to the particles. They are perfectly aligned along the triangle distance line.

2.3 Contact Forces

Force model. In a simple DEM setup two types of forces predominate the simulation when a contact is detected, repulsive forces to represent the third law of motion and frictional forces that capture aspects of the surface according to Coulomb's law of friction. In literature different models based on the spring-dash-pot elastic repulsion force has been already studied in depth (Zhong et. al. and references within [101]), there is no new finding or contribution there. In this project, we implement a linear visco-elastic particle model with spring-dash-pot forces where the actual particles are in-compressible.

2.3.1 Repulsive Force

We apply the linear spring-dash-pot force model from Cundall and Strack [13] to study the DEM and an efficient non-spherical implementation. In such DEM code, I chose to follow the simplest contact mechanics model following the Hertzian contact model. For a Hertzian type model there are few assumptions taken to the contact problem: The strains are small and within the elastic limit. The surfaces are continuous and non-conforming (implying that the area of contact is much smaller than the characteristic dimensions of the contacting bodies). Each body can be considered an elastic half-space.

The code follows the Hertzian theory but with the addition of friction and spring-dash-pot forces. So per particle A, the algorithm accumulates all contact points in $\mathbb{C}_A$ into one translational (f_{trans}) and one rotational repulsive force (f_{rot}) with some dissipation (damping). We make $\mathbb{C}_A$ subject to a post-processing stage which eliminates all collision point duplicates which are all duplicates that are closer than $min(h_{A,min}, h_{B,min})$, where h is the smallest length of the triangles that represent A and B (collision partner). No contact point may be closer than this value. The contact normal n of particle A is always opposite of B and vice versa. The normal vector shows the contact direction (opposite direction per body) and the distance

between the colliding bodies. On the preprocessed contact point set $\mathbb{C}_A$ we then determine

$$f(A,B)_{trans} = \sum_{c \in \mathbb{C}_A} \min\left(0, -k_s \cdot (\epsilon - |n|) + (k_d \cdot 2.0 \cdot (\sqrt{\frac{1.0}{\frac{1.0}{m_A} + \frac{1.0}{m_B}}}) \cdot k_s \cdot v_{AB})\right) \cdot \frac{n}{|n|}$$

$$f(A,B)_{rot} = l_A \times f(A,B)_{trans}$$

the single contact of collision per element per body where k_s and k_d are stiffness and dissipation coefficients, m is the mass. We use the normal's' norm to determine the model's penetration depth, while the change rate of this depth is derived from the projection of the relative velocity between the two particles onto the normal vector. We use the particles' velocities v_A and v_B here to get relative velocity v_{AB}. The min function ensures that the force always pulls particle away from each other. There is no particle attraction to model adhesive contact. The rotational force torque ($f(A,B)_{rot}$) is perpendicular to distance and is determined by the lever arm of A's centre of mass to the contact point. The halo size ϵ is set to 0.001, k_s to 1000, k_d to 0.1.

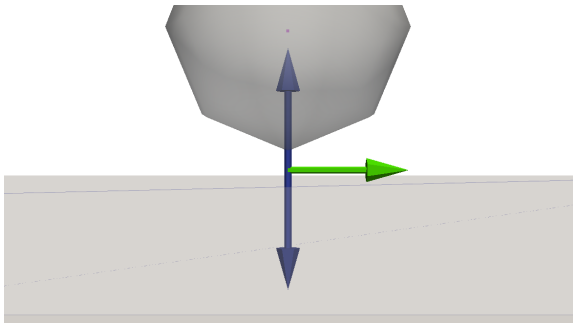


Figure 2.9: A spherical particle in contact with a floor, the contact normal is along the line defined by the distance. Contact forces point to the direction opposite to each particle according to the Newton's first law of motion (blue arrows). Friction is the green arrow and it is tangent to the contact normal.

2.3.2 Frictional Force

Friction is modelled at the contact point using Coulomb damping. A Coulomb friction model applies a friction coefficient to the force depending on the material parameters of the interacting objects. The applied friction is applied tangent and opposite to the relative velocities direction of particle pair during contact. The

frictional model captures the phenomena of stiction, sliding and rolling. Stiction is initiated at the first instance of contact and the collision trajectories can be opposite to each other. During contact sliding and rolling may emerge later on due to stiction, geometry, centre of mass, torque or with introduction of more contact points.

$$f_{friction} = -v_t \cdot mat \cdot f_{contact} \tag{2.1}$$

As friction (2.1) is applied per contact point, per contact point there is a unique contact force. In sphere particles only one contact force can exist per particle pair interaction. Taking the opposite of the relative velocity tangent $-v_t$ I can obtain the right direction to apply friction. For pure stiction (static friction) where particles are in contact with opposite directions (e.g. object on a table) it is straightforward as friction acts similarly to damping. Material parameters *mat* are included to take into account friction coefficient among different types of surface materials. Static friction also behaves to some extend like adhesion but is disabled during contact separation (see Johnson-Kendall-Roberts model of elastic contact).

2.3.3 Sliding Motion

Moving non-static friction occurs when an interaction overcomes stictional (or adhesive traction) forces. When two particles are sliding on one another, sliding friction occurs in opposite directions. Sliding friction coefficients model sliding contact between rough surfaces (corrosion, wearing, surface geometry), lubricated surfaces (oils, powders) and pseudo-elastic resistance (jelly on rough surface) of motion while undergoing deformation. The sum of frictional forces along an area of contact can be zero due to frictional forces annihilation or due to friction coefficients. When sliding friction is annihilated or disabled, slippage occurs. Slippage continues forever unless a new contact is introduced to the interaction.

2.3.4 Rolling Motion

Pure rolling occurs in sphere-based particles once initial static frictional forces are overcome by other kinetic forces. Rolling is the superposition of two motions, linear and angular translational motion. By definition, pure rolling of a axially symmetric object occurs along a reference frame in which a particle axis is parallel to a flat rolling surface at rest once linear velocities reach zero. The kinetic and contact force vector then initiate torque forces that instantaneously trigger angular velocity acceleration to the particle. The instantaneous angular velocity of all points of the

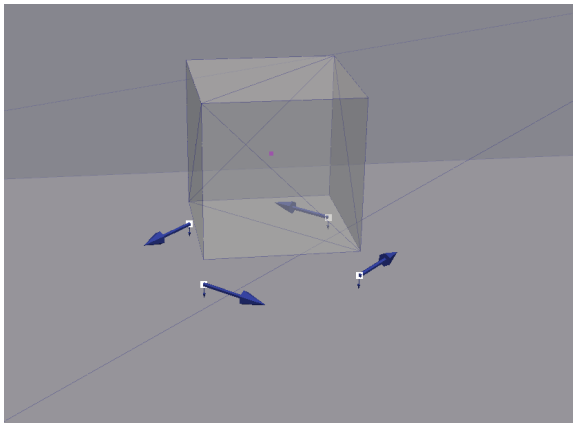


Figure 2.10: A cube particle on top of a table moving with a drilling motion creates friction vectors that apply resistant forces to the stationary (horizontal axis along centre of mass is not changing location) particle. The small vertical vectors are the forces that the particle exert towards the floor.

rolling object is the same as if it was rotating around an axis that passes through the point of contact with the same angular velocity. Due to small deformations near the contact area, sliding and energy dissipation occurs during rolling creating rolling resistance. Rolling resistance is important when modelling non-spherical particle with spheres in order to anticipate the non-existence of rough corners that exist in non-spherical particles. Without rolling friction, sphere particles can roll forever. In non-spherical particles rolling occurs naturally as axial asymmetric bodies induce both torque and bi-linear repulsive forces.

CHAPTER 3

Preliminary Study on the Impact of Triangulated Particles

Introduction. This chapter validates a fundamental claim about the impact of spherical and non-spherical geometries on the physical dynamics of simulation. We showcase by experiment and comparison to results found in literature that the geometric element employed to represent a non-spherical particle in DEM applications makes a difference in the output of a simulation. The hypothesis is that non-spherical particles and spherical particles do not produce the same output for a given engineering setup. This hypothesis can be experimentally validated in a granulates-based hopper flow simulation. We compare the flow of sphere and triangle-based bodies.

We show that varying the degree of geometrical approximation in the simulation makes a qualitative and quantitative difference to the simulation output. The comparison of a hopper flow setup (Figure 3.1) is fair only and only if as a prerequisite equal material parameters are used for computing the mass. The preservation of equal total mass in the domain is enforced in both scenarios.

Literature review. Relevant studies of shape impact in DEM simulations have been well documented in literature [12, 57, 63, 74, 102]. Specifically the hopper granulate flow is studied in many papers. Available studies observe motion differences between sphere, compounds of spheres, ellipsoid or polyhedral-based particles. The flow becomes increasingly concentrated in the narrow funnel above the hopper opening. DEM is able to predict important problems such as bridging and rat-holing. Increasing the blockiness or angularity of geometry of the particles also increases resistance to flow and reduces flow rates. According to Rigway [74] circular particles are a particularly special case that does not represent well real materials. Circular

particles have little resistance to shear or frictional forces. These forces cause the particle micro-structure to yield prematurely via a rolling mode of failure. This causes the flow rates to be over-estimated and always leads to excessively fluid-like mass flow in the hoppers. An extension of DEM to use non-circular particles is thus one amongst many important steps that are required before DEM can confidently predict all the phenomena that occur in hoppers [74]. Other effects still to be understood include the locking of three-dimensional micro-structures, the effect of particle asymmetry, the effect of cohesion, the effects of differing wall-particle to particle-particle friction properties and the effect of micro-roughness on the walls of the hopper [74].

Hopper experiments serve as benchmarks for many DEM codes. Reasonable particle counts yield insights on the flow behaviour, these range from 100 [42], 800 [64], over $3,125$ [12] to $6,000$ [81]. In contrast, experiments with spherical or analytical particle descriptions work with $100,000$ particles [56, 65, 72] and can even reach $2 \cdot 10^9$ particles in total with more than $15,000$ particles per rank/node [44]. However, triangulated particles of in-homogeneous sizes performance data for single node experiments is to the best of our knowledge new.

DEM or related codes that support arbitrary shapes and sizes are rare, and many papers omit the runtime impact discussion [8, 31, 42, 43, 101]. Notably, there is no mainstream code or study that examines arbitrarily shaped and sized triangulated particles. In particular concave meshed shapes are not commonly found [98]. Instead, most codes model complex geometries via assemblies (composites) of simpler/convex primitives [52, 53, 71, 101]. We instead attempt to make use of arbitrary shaped models straight from CAD models which are imported as triangulated objects.

Another dimension of geometry is size and particle scales variation within a single simulation [11, 15, 36, 82, 86]. Multiscale studies for short-range DEM-based simulations demonstrate that geometric representation of scale plays an important role in the dynamics [11, 15, 36]. In particular for milling processes the interaction of coarse particle with interstitial powders composed of fine particles allows predictions to be made of the effect of the local grinding environment [11]. Multiscale scenarios in DEM-based industrial grinding processes demonstrate the importance of the cushioning effects of high powder loads on the flow of coarse particles. Another study [15] demonstrates the importance of multiscale DEM-based porous non-spherical particles in hydrodynamics. Particles heterogeneity directly affects inter-particle interactions, the interstitial spaces (voids) as well as streamline flow if the material is porous. In the field of rock mechanics in general, literature highlights

the importance of shape plus size in various applications [53, 100].

In a hopper flow scenario, there are three major phases: firstly the initial drop configuration condition (Figure 3.1) where particles are above the hopper ready to be dropped by gravity, secondly, the flow dynamics phase resulting in particle-to-particle and particle-to-hopper walls interactions during the drop. Lastly, the simulation is said to be at its termination phase when particles come to rest on the floor (Figure 3.2). In the subsequent section we benchmark hopper flows with spheres against flows where the same number of particles is represented by triangulations. The aim of a flow comparison is to showcase that there is qualitative and quantitative difference, although this study does not prove which approach is the most realistic.

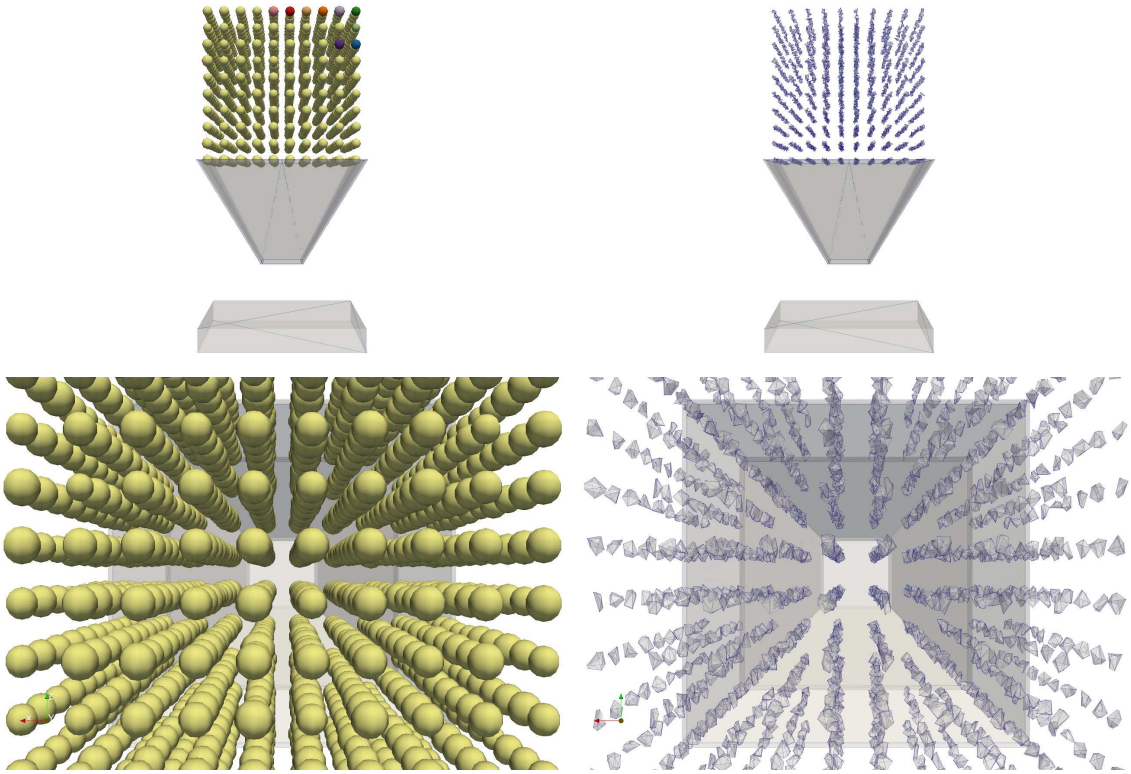


Figure 3.1: Depiction of the two hopper experiments at the starting point. Three types of particles dominate the simulation, a hopper, a floor and the particles. Gravity acceleration is applied to the particles to force them flow through the hopper top and bottom openings. Left: Sphere particles. Right: Triangulated particles (1000 particles made of 5 to 10 triangles).

Symmetries and asymmetries. In this study of geometry we focus on tracking positional symmetries of particles as they flow through the hopper structure, to determine that shape geometry influences the physics. The symmetries experiment is run using a set of scenarios where we use a fixed number of particles, a fixed given mass total of wood material, a fixed range of numbers of triangles for the granulates

(ten triangles for coarsest particle to forty triangles for smoothest particle). These scenarios are then further split into two categories: uniform distributed mass, and non-uniformly distributed mass.

In the uniformly distributed material mass case, the total mass is equally distributed across all homogeneously positioned particles. For the spheres scenario, the prescribed size and mass density are equal, i.e. the total mass is uniformly divided by the number of particles. In such a setup, the hypothesis is that the flow in principle should fall symmetrically along the centre of the domain across the x axis. Contrary to spheres, arbitrary non-spherical shaped triangulated particles of same volume but of random features with uniform mass, should would create a skewed asymmetry due to the irregular shape and non-uniform distribution of contacts. In both cases the hypothesis is that although the same physical phenomena are simulated the employed geometry produces a different output.

In the non-uniformly distributed mass case, the total mass is randomly distributed across homogeneously positioned particles. For the spherical particle scenario, particle sizes are different due to the random distribution of mass while preserving equal material density. In such scenario studies determine whether varying size make a difference compared to equally sized particles. In principle the variation in sizes for both spheres and triangulated non-spherical particle hypothetically would yield similar dynamic behaviour and both fundamentally create asymmetries but not due to geometry surface features but due to the variation in sizes.

Measurement formulation. Symmetry studies require metrics to quantify skewness relative to the mass and position of particles. Firstly, the mean position along the central X axis of the domain is computed by

$$IE_x = \sum_{n=1}^{N} x_n / N.$$

where x_n is the centre of mass (with frame of reference to the centre of the domain) of each particle on the X axis and N is the number of particles simulated. The weighted IE_x mean by mass is computed as

$$IE_{mass} = \sum_{n=1}^{N} x_n \cdot m_n / \sum_{n=1}^{N} m_n.$$

where m_n is the mass of each particle n. The maximum width in both directions from the mean centre of the pile to the pile edge is

$$M_x = max(|IE_x - x_n|).$$

The weighted maximum width in both directions from the mean centre is

$$M_m = max(|IE_{mass} - x_n|).$$

The variance value from the mean position of the pile is

$$Var_x = \sum_{n=1}^{N} (x_n - IE_x)^2/N.$$

The weighted variance from the mean position of the pile is

$$Var_m = \sum_{n=1}^{N} (x_n - IE_{mass})^2/N.$$

Type	$\mathbf{IE}_x$	$\mathbf{IE}_m$	$\mathbf{M}_x$	$\mathbf{M}_m$	$\mathbf{Var}_x$	$\mathbf{Var}_m$
Sphere	-0.000000	-0.000000	0.064117	0.000032	0.001059	0.000000
Triangle	-0.000277	-0.000734	0.079463	0.000772	0.000489	0.000001
*Sphere	0.002276	0.004271	0.070529	0.004289	0.000799	0.000018
*Triangle	-0.002392	-0.006890	0.126990	0.006944	0.001674	0.000047

Table 3.1: Uniform and non-uniform (* starred) mass distribution of sphere and triangle runs of a thousand granulates. A thousand particles are used for all four cases.

Spherical particles with uniformly distributed total mass. The experiment begins by dropping a thousand uniformly distributed mass particles over the hopper funnel (Figure 3.3, top left). The particles are dragged down by gravity. During the drop phase, the particles follow the shape of the hopper structure as they collide with the walls and they are mixed (Figure 3.3, top right, bottom left). During the mixing and compression phases the particles are interacting with each other. Particles are pushed towards the centre of the horizontal axis of the domain while at the same time particle interactions fill the voids through the funnel exit. In this set-up the first particles hit the ground structure around the centre of the domain axis with subsequent particles falling on top of the initial particles on their

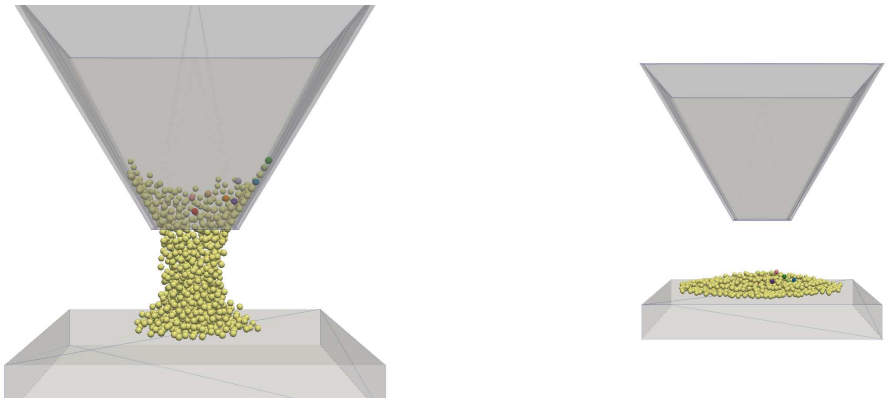


Figure 3.2: Left: A hopper with a thousand equally sized sphere particles flowing through it during the discharge phase. Right: A hopper with equally sized particles at the end of of the discharge phase, resting on top of the floor structure.

left and their right side. At this stage it is critical to observe that a pile is created as sliding and rolling friction coefficients are high enough to prevent rolling over the floor or rebounds.

At the end of the simulation, particle positions along the x axis indicate the degree of symmetry produced by the dynamics (Table 3.1). For equally massed and sized spheres, the mean position of the centre of mass of each particle along the x axis is zero. From the initial dropping up until the resting state all particles behave symmetrically left and right of the y axis, generating not only contact points at the same time but also equal repulsive forces. The mean position weighted with mass is also low as particles are equally massed. This scenario is the baseline of our next steps.

Non-spherical particles with uniformly distributed total mass. The same hopper experiment is executed on non-spherical triangulated particles. The geometry of triangulated particles are randomly generated using the convex hull algorithm (See Appendix) using ten triangles for a coarse surface. The radius of each triangulated particle closely resembles the uniformly distributed radii of spheres of the previous experiment. Although the surface of each particle is not exactly the same, the volume of each particle is preserved across all particles. As the geometries are arbitrary shaped, the centre of mass is not necessarily the same as the centre of geometry as in the case of spheres. For spheres the geometric center is the middlepoint whereas for arbitrary shapes it is the the point that is equidistant from each vertex (centroid). Due to rotational behaviour that non-spheres exhibit, formation patterns differ to spheres. Particle-to-particle contacts initiate rotations before the clogging/compression phase as any contact generated at a corner initiates torque forces. On top of the floor non-spherical particles pile up as long as there is

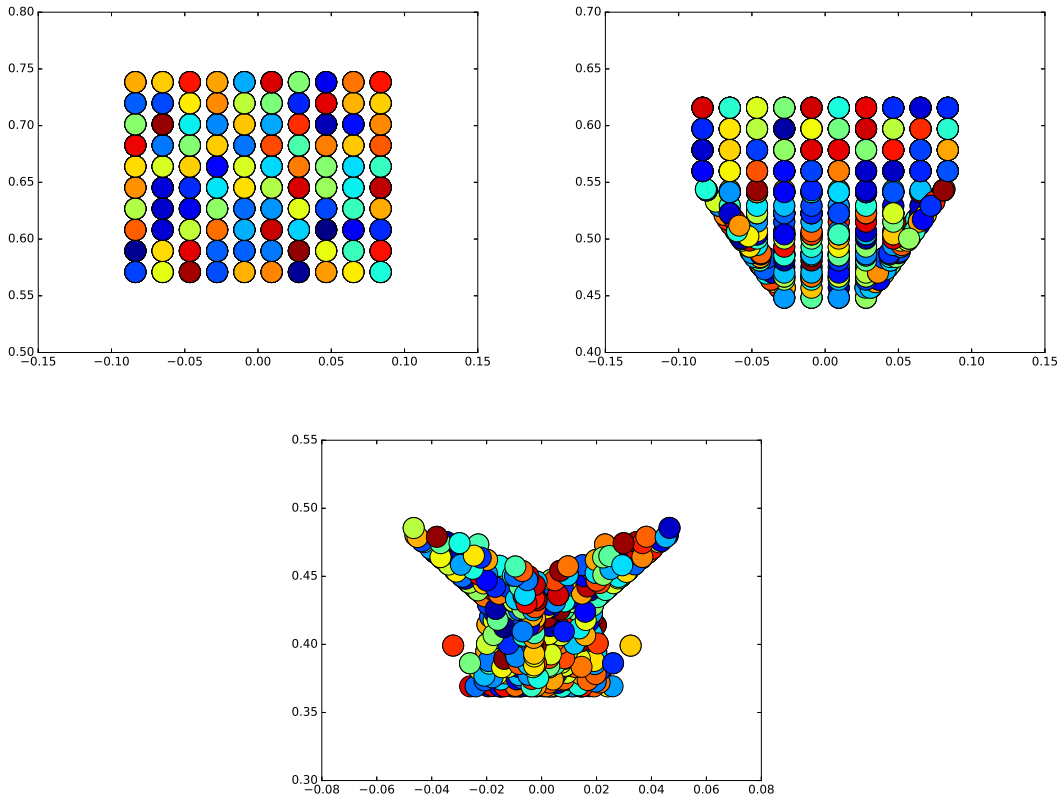


Figure 3.3: Schematic depiction of the two hopper experiments at the starting point. Three types of particles dominate the simulation, a hopper, a floor and the particles. Gravity applied to the particles forces them to flow through the hoppers top and bottom openings.

enough friction.

Type	$\mathbf{IE}_x$	$\mathbf{IE}_m$	$\mathbf{M}_x$	$\mathbf{M}_m$	$\mathbf{Var}_x$	$\mathbf{Var}_m$
Triangle 10	0.000120	-0.000159	0.075062	0.000193	0.000339	0.000000
Triangle 60	-0.000029	-0.000004	0.037941	0.000023	0.000272	0.000001

Table 3.2: Symmetry measurements of non-spherical particles of approximately ten and sixty triangles per particle respectively. A thousand particles are used. Increasing the number of triangles approximates the geometry and behaviour of a sphere.

At the end of the simulation (Table 3.1) the centre of mass positions are not as symmetric as in the uniform distributed mass spheres. The total mean mass position of the particle pile is skewed and its bisection along the x axis creates an asymmetry. The distance from the mean position (Table 3.1 is also slightly shorter than in the uniform sphere experiment. Non-spherical particles become slightly more symmetric and approach the uniform distributed mass of spheres behaviour when the geometry becomes more spherical. The sphericity of the geometry is increased

with the increase in the the triangle number which represent the mesh density at the surface (Table 3.2). When the mesh density is increased from 10 triangles to more, the geometry is becoming more spherical (there are still some random surface features) and the contact dynamics of the experiment exhibit a sphere-like behaviour.

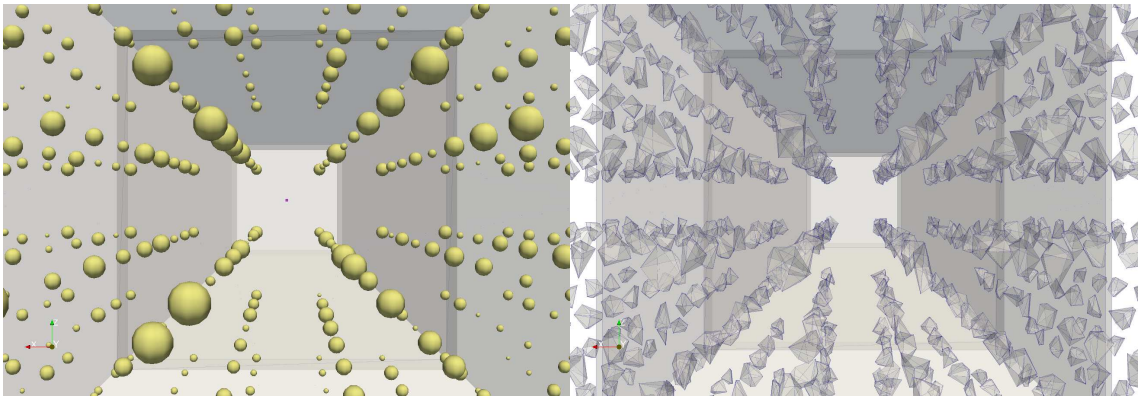


Figure 3.4: Multiscale particles of varying mass per particle. Left: A hopper flow simulation of sphere particles with non-uniform masses. Right: A hopper flow simulation with triangulated particles that have non-uniform mass values.

Sphere particles with non-uniformly distributed total mass. For the second category of experiments we investigate the effect of varying particle sizes. The experiment starts with particles falling off a drop zone above the hopper (Figure 3.4). The same total mass is used as in previous experiments to preserve similarly in the simulation. The interaction dynamics follows the one of spherical particles. As because mass is not distributed equally across the domain, the resulting interaction forces create an asymmetric dynamic interaction (Table 3.1). The spread around the pile bisection is also larger compared to the uniform mass experiment (Table 3.1). Heavier particles are less prone to be moved by lighter particles.

Non-spherical particles with non-uniformly distributed total mass. Finally, multi-scale non-spherical particles are tested. The total mass is non-uniformly distributed on top of the hopper similarly to aforementioned experiments. Non-spherical particles of varying scales inherit the physical characteristics of non-spherical dynamics but also the effects exhibited by multi-scale spheres due to the variation in sizes. Particles of varying scales are not creating a uniform pile (Table 3.1) and the spread is increased as in multi-scale spheres. Although triangle-based and sphere-based particles behave similarly due to varying sizes, they fundamentally differ due to the geometry. Non-uniformly distributed triangulated particles are asymmetric along the mean particle position on the x axis (Table 3.1).

Particle Shape Overview. We see from the preliminary experiment and literature review that the geometry plays an important role in the dynamic behaviour of

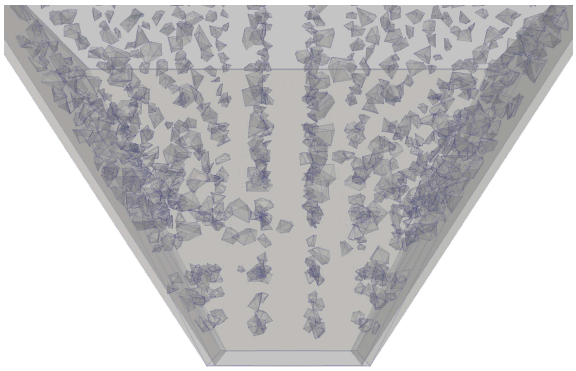


Figure 3.5: Multiscale non-spherical triangulated particles of varying size flowing through the hopper. The sizes vary significantly (up to 5x magnitude) among the dropped particles thus there are varying magnitudes of contact forces and angular velocities during the particle condensation phases.

a simulation. The geometric representation is the basis for the contact model. Both the scale and sharp features affect the employed contact model. For sphere-based models that represent geometries with sharp features, it is up to the contact model to anticipate for dynamic phenomena that inherently cannot exist (e.g. rotations, torques, more than one contact per particle pair) in spheres. Moreover the variation of scales add another dimension in the complexity of the simulation.

Macro-scale effects on large particle scenarios. At the macroscopic scale, minor perturbations in both sphere and non-sphere based geometries do fade way. This is observed in scenarios where small disturbances caused by either geometry or force irregularity (e.g. artificial force injection) do not affect the whole system. At the micro-scale level sphere-based contact models need to exhibit forces that simulate non-sphere-based geometries in order to exhibit on average identical flow behaviours as in non-sphere based geometries. Depending on the application, a compromise between geometry, contact model and computational complexity is reasonable. In this project, we attempt to handle multiscale shapes within a single simulation.

This preliminary experiment shows the effects produced in a hopper flow simulation given a specific amount of mass (kg) to simulate granular materials and surface geometries (sphere, triangulated). The original hypothesis is validated, to reveal that there are fundamental differences between sphere-based and triangulated-based non-spherical particles that both approximate granular material.

In sphere-based particles it is essential to manually find a friction coefficient that realistically attempts to simulate frictional forces of sharp edged, flat faced granu-

lates. Without these frictional parameters, sphere-based particles cannot behave as non-spherical particles neither during the flow phases nor during the piling phases. Without the required friction spherical particles even roll away on the floor. It is non-trivial to numerically anticipate the behaviour of non-spherical granulates using a sphere based model.

Another physical property difference is the phenomenon of non-spherical particles to rotate around an asymmetric centre of mass where axis of rotation is not at the centre of geometry. Contact points per particle pair is the primal cause for further interactions and energy exchange compared to a single point of contact that is the main characteristic of sphere-based particle pairs. These are all causes for the asymmetries observed on non-spherical particles of uniformly distributed mass. Lastly, size variations of particles is another source of difference that affect both sphere and triangle based models. Larger particles dominate in contact interaction and the variations in mass lead to variation in forces, dynamics, and then to the final settlement of the particles during the simulation.

All experiments have been conducted with my code base. The code and contact model employed makes a difference in the simulated physics. However, the differences in particles behaviour based upon geometry configuration is acceptable only if and only if compute work remains manageable. This is where we position our major contribution. In order to under to understand the trade off between the computational complexity and DEM dynamics we first have to review the computer architecture trends.

CHAPTER 4

Computer Architecture

Recent developments in High Performance Computing (HPC) research and future requirements drive the computer technology. The increased levels of parallelism in HPC hardware combined with the quest to solve larger and more accurate research problems shapes is the major challenge of HPC. New trends in hardware shape the best practices in application codes. It is critical that researchers are aware of architectural diversity and transitions in order to adopt early. Yet, the upcoming computation and scientific trends become drivers of HPC algorithm research and promise greater computational throughput.

Chapter Outline. In this Chapter we discuss technology and science drivers that underpin HPC technologies. At the first section Recent HPC Developments we discuss recent developments in HPC. In the second section The Path to Exascale Computing, we follow the exascale roadmap [17, 21, 26, 93] and we lay out implementation techniques and challenges towards the path to exascale. Lastly, section The Challenges and Opportunities in Discrete Element Method juxtaposes HPC trends with issues and opportunities that arise within the context a Discrete Element Method (DEM) implementation. They mark areas worth of algorithmic contributions.

4.1 Recent HPC Developments

Over the last decade, semiconductor technology obeyed Moore's law. Empirically, the density of transistors on a chip doubles every two years. The shrinkage of

semiconductors has been an important driver of global societal and economic change over decades. Chip architectures have been shrinking since the 1970s up to the recent length (as of 2017) of 10 nanometer, a length classification set by ITRS (International Technology Roadmap for Semiconductors) [1]. The shrinking size of components over time increases the computational performance in floating point operations per second as signals are processed faster in denser transistors. Shrinkage of the semiconductor technology is expected to continue but not at the same rate as in early the 1990s and 2000s. The transistor number continues to grow but at a decreasing rate, this is attributed to the limiting factors found in the semiconductor technology. There are manufacturing limits (e.g. modelling, manufacturing errors), physical density limits (e.g. issues with electrochemical components at nano-scale) and thermal side effects that need to be resolved before production.

Stagnation of silicon shrinkage has triggered the development of technologies that allow more floating point computations per chip, per socket and per machine nevertheless. In the HPC community, several parallel flavours have evolved and it is up to the computational scientists to utilise them. Modern computer architectures force scientific users to adapt simulation codes to new system configurations. Large scale simulations that rely on computer architecture goes in par with the underlying hardware evolution.

At the single chip level, a new parallelism paradigm has emerged with the introduction of new CPU instruction sets. Data level parallelism is classified as Single Instruction Multiple Data (SIMD) in the Flynn taxonomy [30] and it was first introduced in vector processors in the 70s. Vector processors gradually faced a decline as the supercomputing community turned into massively parallel commodity supercomputers at the time. Today SIMD/vector processing on the single chip has re-emerged [26] in commodity hardware and it is widely available on machines that expose wide registers in Arithmetic Logic Unit (ALU) through the latest instruction set commands (e.g. Advanced Vector Extensions (AVX)).

At the core data level, SIMD allows users to process data from memory with a single operation. From an implementation perspective, SIMD speedup doesn't come for free as data is required to be streamlined into the core and to be already in memory. Likewise, codes are subjected to fundamental data structure and data-to-memory mapping changes to operate. The code transformation is contradictory to common software readability and maintainability practices. The number of simultaneous operations performed by vector instructions depends on the width of the SIMD register and the binary size of each element that is processed. For example, vectors of 32 bit single precision elements can simultaneously be loaded or stored in

a 256 bit long register on a SIMD machine. This results in a automatic $256/32 = 8$ times speedup. Memory alignment and padding are important factors to streamline data to/from memory layers to fill the ALU registers. Vectorisation on a 256 bit SIMD machine theoretically yields 4 or 8 times speedup on single and double precision respectively. Vector lanes (i.e. register widths) become wider with the new generation of processors and new instruction sets AVX256, AVX512. In practice speedup however varies according to user optimisations and hardware. Heat protection within the chip can under-clock the core at different executions and registers may spill data into the cache when arithmetic intensity is too high.

At the intra-core level parallelism, a single chip can exhibit Simultaneous Multithreading (SMT) parallelism. This technique is also called hyper-threading. On a superscalar processor, a process is allowed to run two threads simultaneously to exploit otherwise inactive chip components. SMT does not always yield significant difference in computational throughput as memory resources are limited per core and often both threads require the same chip components. Due to single core limitations, manufacturers now increasingly design processors that accommodate more than one core at a single socket.

Recent computer generations increase cores per processor and it is not uncommon to see more than a dozen cores per compute node today. The expansion of cores per processor has made it necessary to multiple sockets on-board that are connected via an on-board socket interconnect. The scaling of problems to large number of cores requires significant effort by developers (i.e. latency, bandwidth, development and other constraints).

In recent years, compute nodes often start to incorporate co-processor units that are machines external to the main processors but connected to the main memory. These co-processors typically support higher core counts than standard main nodes and feature a separately owned layer of memory on a single socket. They promise high throughput manycore parallelism and give rise to hybrid computation. However as in vectorisation, speedups are only realised if the code is tailored properly to the architecture's compute units. A limiting factor in such systems is the data latency and bandwidth to and from main memory. It is up to the application to employ offload schemes that minimise latency and arithmetic intensity. Host core and co-processors together realise a hybrid HPC environment. The most prominent co-processor type is graphics cards (GPGPUs).

Significant developments in the architecture of processor design result in new challenges that programmers have to face. Recent developments in cores technology give rise to three type of parallelism: SIMD vectorisation, manycore multi-threading, hybrid CPU and co-processor computation. All these computational concepts are important to implement codes that run efficiently.

4.2 The Path to Exascale Computing

The HPC community charts a path to exascale [17] and clarifies that experiments through simulations will require higher resolutions on time scales that are infeasible at the moment. As such, the path to exascale is challenged by three general areas. Firstly, in the area of code development through which users program, debug and optimise code (i.e. frameworks, compilers, libraries, debuggers, performance analysis, fault tolerance). Secondly, in the area where application specific challenges happen (i.e. visualisation, data analysis, large data management, coupling techniques, algorithms). Thirdly, the cross cutting dimensions that affect HPC usage in general (i.e. resilience, power management, performance optimisation, programmability). In this section we focus on trends and challenges that affect modern DEM codes.

Development Environment. Exascale systems are expected to consist of large number of compute nodes incorporating a mix of conventional multicore CPUs, many-core chips and accelerator hardware [17]. Therefore, heterogeneity in computation will be the source of increasingly complex memory management. It is expected that applications will combine distributed and shared memory computational phases. The maximisation of data locality by the programmer will be a vital design decision during code development.

Frameworks provide a common interface which exhibit modular components that can be used independently in more than one applications. These framework codes become increasingly meaningful as they provide the means to standardise software in application. Modularity of software components provide separation of concern to users and they make it easier for developers to isolate bugs. Multidisciplinary applications benefit from frameworks as less time is spent on development and reinvention of functionalities. Libraries offer users the potential to use computer systems without the need to understand the technicalities of the architecture. Frameworks promise at least to speed up scientific software development and increase reliability of execution.

Applications. Algorithms will be developed in line with the new architectural realities (such as wider vector lanes, NUMA). Scalability of algorithms on large sys-

tems is a key technological driver but it is faced with problems. Due to the required amount of threads per program, the creation of sufficient concurrency for one application is a major challenge. Non-Uniform Memory Access (NUMA) awareneness is necessary in code design. It is critical that future algorithms hide latency by overlapping computation with communication. Load balancing is a major issue that makes dynamic load balancing prominent [17]. Modelling in terms of data acquisition and storage technologies will produce large amounts of data as higher scalability becomes achievable [17].

Cross Cutting Dimensions. Exascale systems are expected to utilise hundreds of millions of cores and it is a challenge to achieve programmability. Key factors that affect the programmability of research codes include:

i. Parallelism: it is vital to expose the code to sufficient parallelism to sustain exascale operations. But at the same time, scalable concurrency has to be in par with latency, bandwidth and compute bounds.

ii. Distributed Resource Allocation and Locality Management: an algorithm must balance scattered workload and at the same time localise execution to sufficiently maximise resource utilisation. Task partition and execution has to be coupled with data locality to minimise latency, bandwidth and maximise operational ALU units.

iii. Latency Hiding: the computation needs to overlap communication in order to avoid blocked tasks and under-utilisation of resources.

iv. Hardware Idiosyncrasies: properties specific to computing resources such as memory hierarchies, instruction sets, accelerator design must be managed in a way that circumvents negative impacts. Adoption of architectural opportunities is necessary but without increasing the demands for explicit user control.

v. Portability: programs need to be portable across machine scales, generations and architectures. Developers have to minimise portability effects caused by small code perturbations.

vi. Synchronisation Bottlenecks: communication barriers and control procedures are required to exhibit lightweight synchronisation phases that overlap/pipeline computational phase. Data coherency bottlenecks in both shared and distributed memory need to be minimised.

4.3 The Challenges and Opportunities in Discrete Element Method

An understanding of the developments towards exascale is vital for the design of proper DEM-based particle simulations. We are confronted with machines that

feature the highest degree of concurrency, which is both a scalability challenge and a problem size up-scaling opportunity. These will enable DEM to incorporate higher number of particles, higher resolution surfaces and more complex geometries in contact dynamics simulations. To achieve this in practice the scientific community has to turn challenges into opportunities.

In this section we discuss the challenges and opportunities in the context of DEM. Table 4.1 lays out per challenge the problems and the opportunities. We categorise the challenges according to scalability, data migration, architectural complexity, programmability and higher resolution physics. For each of the challenges we cover the global issues, we discuss how problems could be tackled by DEM implementations and lastly we cover contributions by this thesis.

Challenges	**Problems**	**Opportunities/Tasks**
Scalability	very high concurrency, arithmetic intensity, communication and cache access tuning	Create models that increase arithmetic intensity, multicore runs, kernel fine tuning and vectorisation.
Data Migration	large volume of data, data locality, high latency and communication patterns	Introduce streamlined/vectorised data structures reduce latency, minimisation of data access.
Architectural Complexity	new CPUs, manycores, multi-socket, hybrid systems	Use established standards (OpenMP, TBB, OpenMPI, SIMD pragmas), load balancing, new models that exploit SIMD, manycore architectures.
Programmability	software modularity, algorithmic robustness, interoperability of libraries/frameworks	Creation of a modular library with minimal dependencies, interchangeable use of contact models and geometry manipulators.
Higher Resolution Physics	new and faster algorithms are required	Dense non-spherical geometries, new algorithmic model for DEM and a modern data storage scheme.

Table 4.1: The universal challenges, the specific problems and the opportunities that arise in the development of DEM codes.

DEM codes are faced both with challenges and opportunities. It is important to understand and adopt codes according to hardware changes in order to maintain high levels of throughput. The interoperability of software and exploitation of resources is critical to support better and faster physics.

The first challenge in terms of scalability of DEM codes (Table 4.1) is the adaptation of core routines to SIMD-enabled CPUs. The introduction of wide vector registers pushes developers to transform scalar-based operations into vector-based algorithms. High memory-to-core throughput is only possible when latency is minimised. The analytical and geometric computation found in DEM contact detection phase prohibit high arithmetic intensity as branching (i.e. poor streaming behaviour) or little floating points operations are required [51], whereas an iterative alternative method would typically exhibit higher arithmetic intensity. Ideally, an iterative contact detection method that exhibits the property of locally streamlined memory access along with lots of floating point operations per second would utilise the ALU resources. In addition to the contact detection which is common to take a high percentage of runtime [55], phases that benefit from SIMD are the particle position update and force derivation.

In this project we utilise SIMD parallelism to increase floating point operations per core. We create a novel set of iterative contact detection methods for non-spherical particles. To our knowledge this is the first attempt to create a vectorised non-spherical model and implementation with triangles. The employed memory layout allows for streamlined computation of contact points between meshed geometries. With vectorisation the contact detection phase runtime is reduced by magnitudes when compared to serial runs. We pack more floating point operations per core at lower latencies and it allows us to create a DEM code that is capable to simulate complex geometries at a larger scale.

An effective SIMD implementation that computes on more than one core is mandatory. DEM algorithms need to exploit the expanding concurrency levels. Particularly, DEM work phases for contact detection need to be shared among cores. We investigate various parallelisation concepts for clusters of particles and propose different parallelisation schemes. It is clear that it is critical to multi-thread the DEM. This is rather straightforward for localised particles in memory. Sparse particle clusters however are a challenge as data locality has to be ensured dynamically.

Standalone DEM algorithmic phases (i.e. contact, force derivation, position update and plotting) cannot always exploit all resources due to lack of work, latency, non-local memory accesses or branching. Resource allocation cannot be uniform throughout the whole simulation if the underlying geometry is very dynamic and

therefore affects the computational intensity. It is important to balance, intermix and partition algorithmic phases to achieve both spatial and temporal homogeneity for maximum throughput.

We propose a novel mix of parallelism for shared memory that tackles more than one DEM phase at a time. We pipeline contact detection, particle position update and force derivation phases following the work stealing model. Tasks are deployed intermixed to a work queue that then is used to execute work on various cores. The proposed multi-threading model realises a decoupled granularity at the triangle-to-triangle and particle-to-particle level. Futhermore, the tasked-based parallelisation is coupled with a parallel grid traversal. The concept of pipelined phases when combined with a grid finally allows us to read data only once per step.

An emerging challenge in HPC particle n-body simulations is the gradual adoption of co-processors. Co-processors raise new challenges in algorithmic design where offload routines become major phases of parallel computation. Hybrid computations give rise to computational phase segmentation, intra-node load balancing, CPU-co-processor communication and further fine tuning for algorithms. Offload incorporation between algorithmic phases and intermixing main core and co-processor cores introduces further complexity.

A challenge with respect to data migration can be found in multi node computation and communication too. The challenge in DEM is the reduction of communication latency and minimisation of data exchange volumes. Data exchange occurs during contact detection where halo particles that are located on neighbouring nodes need to derive interface contacts. The amount of data being exchanged is proportional to the data of pairwise halo particles. Moreover, a communication initiation overhead is imposed by data exchange. This contributes to the latency. In distributed memory DEM codes node level data structures finally are required to be moved into MPI buffers. The critical factor for overhead minimisation is the locality of particle data. There is no contribution here because this is an traditional challenge that is well understood [71].

Faced by implementation challenges, the current project contributes new methods and implementation techniques at the single node level. We introduce new novel contact detection methods for non-spherical particles that promise to increase the physics possible per core. We propose a new shared memory scheme that is based on a meshed-based contact detection coupled with pipelined DEM and grid traversal phases.

In terms of software development, productivity increases through the utilisa-

tion of modular frameworks. As our DEM implementation realises a grid-based approach, we rely on the Peano grid framework [92]. We use its mesh traversal and merge multiple computational phases into one DEM-grid mesh run-through. Such a formulation allows us to realise locality of particle data access, coarse-grained parallelism and dynamic load balancing. The modularity in software components and phases allow for loosely integrated functionality, localised errors in algorithms and it is an opportunity to create interchangeable codes within specialised research fields. In accordance to exascale roadmap, we propose a code design that is modular. The modularity allows our DEM code snippets to become a inter-operable library. As the project does not mean to re-invent space discretisation frameworks we focus on the DEM-grid interplay. Our abstract use of a grid and its interoperability with DEM algorithmic phases can be used a template example for other projects.

We propose a novel particles-in-grid scheme that support various scales of arbitrary shaped particles. The space decomposition scheme produces various grid morphologies that when combined with the contact model produce unique properties. Likewise, by exploiting the grid morphology we support bigger time steps.

CHAPTER 5

Algorithm Outline and Vectorisation

Introduction. In its basic form, a Discrete Element Method (DEM) algorithm is based on mainly three computational phases: collision detection, force derivation and time integration. The simulation starts from a geometry setup before time integration updates the position of particles and collision detection update interaction forces. A sequence of routines define the phases and update the physical properties of the geometry (position, velocities).

Chapter outlook. In this Chapter we discuss the mapping of DEM phases onto a general algorithmic outline. Section Time Integration discusses the employed time integration scheme executed at each time step. In the Particle Collision Detection Problem section we discuss the implementation of collisions i.e a set of new contact detection methods for triangulated geometries. Section Force Derivation discusses the generation of forces from the list of collisions. Lastly, we discuss vectorisation and memory layout of the contact detection methods. Although important for a implementation, we do not discuss preprocessing and post-processing in this Chapter.

5.1 Time Integration

Our time integration implements a simple scheme where the velocities are updated by the interaction forces before the positions are updated. This integration model is chosen for algorithmic simplicity. No contribution is made here.

Linear position updates using explicit time integration are straightforward. Positions are updated according to a force f, mass m and step size dt and the velocity

v. According to Newton's second law the velocity v is computed as $\Delta v = dt \cdot f/m$. In interactions that include collisions the total interaction force f_{ij} per particle pair i, j is computed based on contact points generated. If there are no collisions, then the velocities remain the same as no new forces are applied. Positions p at centre of mass of particles are then updated by $\Delta p = dt \cdot v$.

Rotational position updates work similar to linear position update but utilise angular velocities, torque and an inertia matrix. During collisions, contact points generate torque forces f_rot that contribute to the direction of the total acting force f on the bodies. Given a torque we update the angular velocity and position by step dt. We then construct the inertia matrix and mass using a material parameter (rho) and adding up a series of simplices composed by all the triangle points in the geometry. With an inertia matrix i, spatial position $\overline{x}$ (current position), referential position $\overline{X}$ (previous position), mass, angular velocity ω, a rotation operator R is constructed [48, 49]. All the triangle points on a particle are then updated as $x(X, dt) = R(dt) \cdot (X - \overline{X}) + \overline{x}(dt)$ where x, X are the spatial and the referential points on the geometry, and $\overline{x}$, $\overline{X}$ are the spatial and the referential mass centres. The current position is mapped into the next position by our timestep dt. The rotation specification is well documented in existing literature [48, 49], no contribution is made here.

Explicit time integration in DEM simulations is problematic in terms of numerical stability (it is a stiffness problem). The choice of step size dt is a critical factor in explicit time step simulations. Small time steps sizes allow the safe detection of collision events whereas simulations that employ a large step size risk skipping a collision event. The step size determines the degree of stability as well as the total number of time steps (i.e. cost required to reach termination). As such, two strategies are generally used: an optimistic and a pessimistic scheme. An optimistic scheme defines a constant dt time step size throughout the duration of the simulation. If the optimistic scheme fails it is possible to rollback to a previous step refine the step, the error is discarded and the simulation is repeated with a smaller step size. In a pessimistic approach a dynamic strategy is employed that adopts the time step size based on events within the simulation (i.e. chooses dt during runtime). Generally a dynamic dt minimises the number of iterations required [59] and subsequently the time to solution is shortened.

The explicit time integration is a naive approach to DEM implementation. Moreover the nature of particle collision dynamics enforces finer step sizes as collision occur regardless of the employed integration scheme. While the extension to implicit schemes is beyond scope, a more sophisticated adaptive time step scheme

combined with implicitly time integration could convert a time-driven scheme into an event-driven algorithm [59].

5.2 Force Derivation

Per contact, there's a shortest line between the two bodies. The line's midpoint is the contact point. Any contact point thus falls into the ϵ environment of the involved particles, and it is equipped with an outer normal vector n. Technically, we represent each contact point as two points with opposite normal directions. The normal vector's length is the distance from the contact point to the surface of the neighbouring particle. Its direction depends on the point of view: from which particle do we look at the contact point. We have $|n| \leq \epsilon$ where $\epsilon - |n|$ is the (halo layer) penetration depth.

The penetration depth determines the force arising from a contact. Its spatial position relative to the particles' centres of mass clarifies whether the contact induces rotation or translation or both. In the present experiments, we rely on spring-dashpot's force model with pseudo-elastic damping [13] between two particles (Chapter Physical Model Section Contact Forces).

5.3 The Particle Collision Detection Problem

Most DEM code spend a majority of their compute time in collision detection. Iglberger et al. [24] report 31-34% within a multiphysics setting, while Li [55] for example reports even 90%. Detection becomes significantly more complicated once we switch from sphere-to-sphere or ellipsoid-to-ellipsoid checks to the comparison of billions of triangles; notably if no constraints on convexity are made and if explicit time stepping stops us from modelling complex particle shapes as compound of simpler convex shapes subject to a non-decomposable constraint.

The basic DEM implementation is illustrated with Algorithm 1. For rigid body dynamics the main body of the algorithm consists of one outer time integration loop that host two inner particle traversal loops. The first inner loop consist of the collision detection phase (at line 2). The collision detection is an $O(n^2)$ operation where n is the number of particles. Contact points are detected and defined between particle triangles that come into contact. During collision detection (lines 2-9), all $\mathbb{T}$ triangles of $\mathbb{T}_A$ and $\mathbb{T}_B$ of two bodies A and B are compared to determine collision points (i.e. a $O(n^2)$ operation). If the distance between two particles is closer than a defined margin epsilon then the particles are in contact (Chapter 2). A second

Algorithm 1 Blueprint of the Discrete Element Method algorithm.

```
1:  for t < T do                                              ▷ T Total Simulated Time
2:      for i ← 0 ... 𝕋_A ∃ particles_n do
3:          for j ← i + 1 ... 𝕋_B ∃ particles_n do
4:              δ ← TTD(i, j)                  ▷ Triangle-to-Triangle Detection (TTD)
5:              if (δ ≤ ε) ∧ Particle(i)¬Particle(j) then
6:                  contact(PID(i)).add(point, normal)          ▷ Particle ID (PID)
7:              end if
8:          end for
9:      end for
10:     for z ← 0 ... particles_n do                                  ▷ Force Update
11:         for k ← 0 ... contacts(z)_n do
12:             f_z ← granular(velocity(z), position(z), contacts(z).getcontact(k))
13:         end for
14:         updatePosition(position(z), f_z)                          ▷ Position Update
15:     end for
16:     t ← t + Δt
17: end for
```

loop runs over detected contact points and converts them into contact forces. While contact points in principle could be translated into forces straight-away it makes sense to outsource the force derivation into a separate algorithm phase just before time integration.

Our DEM codes rely solely on pair-wise interactions. The complexity of force computation depends solely on contact point generation and the geometry. The number of contacts per particle typically is very small if particles are of reasonably similar size. If particle sizes vary dramatically, comparably large particles experience contact forces from many small particles. In this case, there are however few large particles compared to the overall number of particles.

It is therefore very important to create an efficient and flexible algorithm to resolve contacts. The fundamental problem in such DEM codes is the computation of the minimum distance between a pair of triangles in three dimensions. Given the minimum distance between two triangles and the corresponding closest points, a unique contact point can be generated. In the following sections I give a detailed overview of three triangle-to-triangle distance algorithms. The first algorithm is brute force, a naive approach to the triangle-to-triangle distance problem. The second algorithm is an iterative approach that promises to achieve better computational performance. It is however not robust, i.e. can yield wrong results. Lastly a hybrid approach is proposed. It combines both worlds into one solution.

An efficient collision detection algorithm co-determines performance in a DEM code. In triangulated non-spherical particle dynamics, the computation of distances between particles is reduced to computing the distances between triangle pairs. Alternative methods for collision detection do not allow for non-convex meshes.

5.4 The Brute Force Geometric Comparison

The computation of the minimum distance between two triangles is reduced into simpler geometric sub-problems. Since a triangle is composed by line segments and three vertices, it is sufficient to compare the distance between line segments to line segments and the distance between vertices to the triangle plane. This method is a brute force approach as it involves computing all possible combinatorial configurations. At the end of the series of checks, the solution is deterministically known. The algorithm is compiled with basic primitive geometric checks found in literature [19, 20, 28].

Algorithm 2 Brute Force Distance Computation Algorithm.

1: **function** BF(A, B, C, D, E, F) ▷ A ... F Triangle Points
2: $T_A \leftarrow [A; B; C;]$;
3: $T_B \leftarrow [D; E; F;]$;
4: $list[0] \leftarrow PT(T_B, T_A[0])$;
5: $list[1] \leftarrow PT(T_B, T_A[1])$;
6: $list[2] \leftarrow PT(T_B, T_A[2])$;
7: $list[3] \leftarrow PT(T_A, T_B[0])$;
8: $list[4] \leftarrow PT(T_A, T_B[1])$;
9: $list[5] \leftarrow PT(T_A, T_B[2])$;
10: $pt_{min} \leftarrow min(list)$;
11: $list[0] \leftarrow SEGSEG(T_A[0], T_A[1], T_B(0], T_B[1])$;
12: $list[1] \leftarrow SEGSEG(T_A[0], T_A[1], T_B[1], T_B[2])$;
13: $list[2] \leftarrow SEGSEG(T_A[0], T_A[1], T_B[2], T_B[0])$;
14: $list[3] \leftarrow SEGSEG(T_A[1], T_A[2], T_B[0], T_B[1])$;
15: $list[4] \leftarrow SEGSEG(T_A[1], T_A[2], T_B[1], T_B[2])$;
16: $list[5] \leftarrow SEGSEG(T_A[1], T_A[2], T_B[2], T_B[0])$;
17: $list[6] \leftarrow SEGSEG(T_A[2], T_A[0], T_B[0], T_B[1])$;
18: $list[7] \leftarrow SEGSEG(T_A[2], T_A[0], T_B[1], T_B[2])$;
19: $list[8] \leftarrow SEGSEG(T_A[2], T_A[0], T_B[2], T_B[0])$;
20: $ss_{min} \leftarrow min(list)$;
21: $pq_{min} \leftarrow min(ss_{min}, pt_{min})$;
22: $P \leftarrow p_{min}; Q = q_{min}$;
23: **end function**

Our brute force solver is shown in Algorithm 2. At line 1 the function accepts two triangles T_A and T_A with vertex points A, B, C and D, E, F respectively, in three dimensions. Function PT (lines 4-9) is the point-to-triangle distance algorithm (see Algorithm 4) that computes all distances between each vertex point one triangle against the pairwise partner triangle. As soon as the PT algorithmic phase is complete, the output is reduced into a pt_{min} which is the minimum of all point-to-triangle comparisons. The second phase starts with the execution of segment-to-segment ($SEGSEG$, lines 11-19) algorithm (Algorithm 3) which compares line segments of one triangle to line segments of the other triangle. At the end of the code (line 20,21) the minimum of both phases is the minimum distance between two triangles in three dimensions defined by vertices P and Q.

The brute force implementation runs per triangle pair and checks all possible minimum distance configurations (Algorithm 2). The method yields 15 geometric comparisons in total and it is a robust approach as it always yields the correct answer to the problem. A fusion of multiple steps is not possible as all geometric scenarios have to be evaluated separately and naively. The algorithmic (Algorithm 2) and the brute force character does not allow computational fusion of procedures as each geometric is evaluated separately in a naive scheme. The brute force method cannot be optimised for SIMD due to the branching that exists in its analytical solution of the distance (BF) routine.

The brute force approach, is a robust solution that involves rigorously checking every primitive of two triangles to determine the minimum distance. The main characteristic of the algorithm is that it always arrives to a correct solution but the drawback of that is that there are redundant checks. Redundancy in computation and logical branching renders the brute force method incapable to exploit SIMD hardware.

5.4.1 The Distance Between Two Line Segments

The calculation of the distance between line segments [19, 28] in three dimensions identifies the closest points by extending the line that they lie on until intersection. If the two lines intersect, then the closest point on the two segments is also within the boundaries of the line segments. The line segment $S_1 = [P_0, P_1]$ can be formulated as $P(s) = P_0 + s \cdot (P_1 - P_0) = P_0 + s \cdot u$ with a constraint on s, $0 <= s <= 1$. Similarly, the second segment $S_2 = [Q_0, Q_1]$ is written as $Q(t) = Q_0 + t \cdot (Q_1 - Q_0) = Q_0 + t \cdot u$ with a constraint $0 <= t <= 1$. Let s_c and t_c be the closest points on the extended segments lines L_1 and L_2. If s_c and t_c are within the boundaries of the line segments

then they are also the closest points on the respective segments. Whereas if s_c and t_c are located on points that are outside the range of either S_1 and S_2 then they do not also define a closest points. So it is necessary to determine points that minimise $w(s,t) = P(s) - Q(t)$ over the ranges of the segments using the corresponding constraints. The problem is formalised into a minimisation equation where w is the same as minimising $|w|^2 = w \cdot w = (P_0 + s \cdot u - t \cdot v) \cdot (Q_0 + s \cdot u - t \cdot v)$, w is a quadratic function of s and t. The relation of $|w|^2$ define a parabolic equation over a (s,t)-plane (see Figure 5.1) with a minimum at $C = (s_c, t_c)$, it is strictly growing along the (s,t)-plane with starting point from C. But because segments S_1 and S_2 are concerned and not their respective extended lines L_1 and L_2, the required minimum region is not C but it is located over a sub-region G of the (s,t)-plane. The global minimum at C may lie outside of G, however, in these cases, the minimum always occurs on the boundary of G, and in particular, on the part of G's boundary that is visible to C. That is, there is a line from C to the boundary point which is exterior to G, and it can be said that C can "see" points on this visible boundary of G (See Figure 5.1).

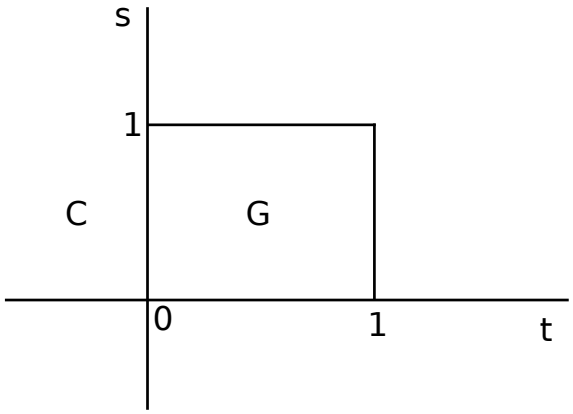


Figure 5.1: (s,t) parameter space with G boundary box C for global minimum

Suppose that we want the minimum distance between two finite segments S_1 and S_2, then $G = \{(s,t) \parallel 0 \leq s \leq 1, 0 \leq t \leq 1\} = [0,1] \times [0,1]$ is a unit square in (s,t) space. The four edges of the square are given by point at $s = 0, s = 1, t = 0, t = 1$. If $C = (s_c, t_c)$ is outside the G area, then it can "see" at most two edges of G. If $s_c < 0$, C can see the $s = 0$ edge, if $s_c > 1$, C can see the s = 1 edge, and similarly for tC, so in this way there is an enforcement of the required constraints. When C is not in G, at minimum 1 and at maximum 2 of constraints are active, and they determine which edges of G are candidates for a minimum of $|w|^2$.

For each candidate edge, to compute where the minimum that occurs on that edge, either in its interior or at an endpoint, it is possible to solve for the other

Algorithm 3 Line Segment to Line Segment Distance Algorithm.

```
1: function SEGSEG(S_1, S_2)
2:     if S_1 and S_2 are parallel then
           return any point P on S_1 and any point Q on S_2
3:     else
4:         Get the closest points s_c and t_c on the infinite lines L1, L2
5:         if s_c < 0 then                          ▷ the s = 0 edge is visible
6:             s ← 0
7:         end if
8:         if s_c > 0 then                          ▷ the s = 1 edge is visible
9:             s ← 1
10:        end if
11:        if t_c < 0 then                          ▷ the t = 0 edge is visible
12:            t ← 0
13:        end if
14:        if t_c > 0 then                          ▷ the t = 1 edge is visible
15:            t ← 1
16:        end if
17:    end if
           return P, Q points on two segments
18: end function
```

unknown parameter since at minimum one is to be found (either t or s). So using the derivative of the $|w^2|$ equation it is always possible to solve for the parameter that is in the interior of G. For example when $s = 0$, $|w|^2 = ((P_0 - Q_0) - tv) \cdot ((P_0 - Q_0) - t \cdot v)$. Taking the derivative with t it possible to get a minimum when $0 = \frac{d}{dt}|w|^2 = -2 \cdot v \cdot ((P_0 - Q_0) - t \cdot v)$. This gives a minimum on an edge at $(0, t_0)$ where $t_0 = \frac{(v \cdot (P_0 - Q_0))}{(v \cdot v)}$. If $0 <= t_0 < 1$, then this would be the minimum of $|w|^2$ on G, and P(0) and Q(t0) are the two closest points of the two segments. But in the case where t0 is outside G, then either $(0, 0)$ or $(0, 1)$ would be the minimum along that edge (since $s = 0$). Using this method it is possible to perform only a couple of checks to find the minimum of w(s,t) that correspond to the minimum distance between the segments.

5.4.2 The Distance Between a Point and a Triangle

The second part of the naive triangle-to-triangle minimum distance is the point-to-triangle distance [20] computation. The minimum distance between a point P and a triangle T can be described by its barycentric coordinates such that any point on its plane is defined by two parameters. We choose s and t such that

$T(s,t) = P_a + s \cdot E_0 + t \cdot E_1$ where $E_0 = P_b - P_a$, $E_1 = P_c - P_a$ for

$$(s,t) \in D = \{(s,t) : s \in [0,1], t \in [0,1], s+t \leq 1\}.$$

To solve the problem we need to find (s, t) values in a domain D.

The minimum distance is computed by determining the values (s,t) in the distance equation $Q(s,t) = |T(s,t) - P|^2$ where $T(s,t)$ corresponds to a point on the triangle closest to P. The function is quadratic written as

$$Q(s,t) = as^2 + 2bst + ct^2 + 2ds + 2et + f$$

where $a = E_0 \cdot E_0$, $b = E_0 \cdot E_1$, $c = E_1 \cdot E_1$, $d = E_0 \cdot (P_a - P)$, $e = E_1 \cdot (P_a - P)$, and $f = (P_a - P) \cdot (P_a - P)$. Quadratics are classified by the sign of $ac - b^2$ so for function Q, $ac - b2 = (E_0 \cdot E_0)(E_1 \cdot E_1) - (E_0 \cdot E_1)^2 = |E_0 \cdot E_1|^2 > 0$. The positivity is based on the assumption that the two edges E_0, E_1 are linearly independent, so their cross product is a nonzero vector. The minimum occurs at the interior where the gradient $Q = 2(as + bt + d, bs + ct + e) = (0,0)$ or at the boundary of D. The gradient of Q is zero only when $s = (be - cd)/(ac - b^2)$ and $t = (bd - ae)/(ac - b^2)$. If (s, t) is in region zero (Figure 5.2), then the point on the triangle closest to P is interior to the triangle, not on its edge. Otherwise, if the minimum is not in D then it resides on the boundary of the triangle. If (s,t) is in region one then the elliptic level curves of Q are those curves in the s,t plane for which Q is constant.

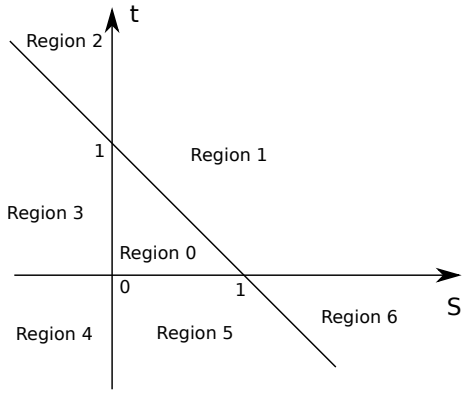


Figure 5.2: Regions based on the (s, t) parameters plane

The global minimum of $Q = (0,0)$ occurs in a level area V_{min} and levels V are growing from (s,t). There is a level value V_0 for which the corresponding ellipse is tangent to the triangle domain D edge $s + t = 1$ at a value $s = s_0 \in [0,1]$, $t_0 = 1 - s_0$. So for any level values $V < V_0$, the corresponding ellipses do not intersect D. However, any portion of D that intersect levels V must be $V > V_0$. Therefore,

in this case, point (s_0, t_0) gives the minimum distance between P and the triangle where $t_0 = 1 - s_0$ and s_0 is the only unknown to be solved. When the minimisation is happening at $\nabla Q(s, 1s) = 0$ there are three cases where $s > 1$ and s have to be restricted to $s = 1$ and the minimum occurs at $\nabla Q(1, 0)$ because of the barycentric triangle constraints, if $s < 0$ then the minimum occurs when $\nabla Q(0, 1)$ otherwise $s \in [0, 1]$ and has to be solved for a minimum at $\nabla Q(s, 1 - s)$.

Algorithm 4 Point to Triangle Distance Algorithm.

```
1:  function PT(T, P)
2:      if Point projection in region 0 then
3:          evaluate value of Q
4:      end if
5:      if Point projection in region 1 then
6:          determine (s, t) parametric space values
7:      end if
8:      if Point projection in region 2 then
9:          determine (s, t) parametric space values
10:     end if
11:     if Point projection in region 3 then
12:         determine (s, t) parametric space values
13:     end if
14:     if Point projection in region 4 then
15:         determine (s, t) parametric space values
16:     end if
17:     if Point projection in region 5 then
18:         determine (s, t) parametric space values
19:     end if
20:     if Point projection in region 6 then
21:         determine (s, t) parametric space values
22:     end if
        return Q point on triangle
23: end function
```

Similarly we follow the same method to determine whether the minimum occurs at the endpoints or at the interior of the constraints. For this we search region three and region five (Figure 4). In the case where (s, t) occur in region three then the minimum has to occur at $(0, t_0)$. If (s, t) is located in region five (Figure 4), then the minimum occurs at $(s_0, 0)$ where $s_0 \in [0, 1]$.

When (s, t) is located inside region two, there are three possibilities for the level to contact or intersect the boundaries:

1. edge where $s + t = 1$
2. edge where $s = 0$

3. at point where $t = 0$ and $s = 1$.

Although the global minimum is in region two, there is a level of Q that contacts the D but the contact point and the region inside of the level curve does not overlap. At these occurrences then the negative of the contacting level curve of Q cannot point inside D. For example for region two there could be the direction of $-\nabla Q(0, 1)$, $-\nabla Q(s, 1 - t)$ and $-\nabla Q(0, t)$, which points towards the inside of the level curve instead of inside D.

To determine which of the three cases occur, it is possible to check which areas are negative by eliminating area where contact doesn't occur. If $\nabla Q = (Qs, Qt)$ and Qs and Qt are the partial derivatives of Q, it should be the case where $(0, -1) \cdot \nabla Q(0, 1)$ and $(1, -1) \cdot \nabla Q(0, 1)$ are not both negative. The two vectors $(0, 1)$ and $(1, 1)$ are directions that correspond to the edges $s = 0$ and $s + t = 1$, respectively. The choice of edge $s + t = 1$ or $s = 0$ is decided based on the signs of $(0, -1) \cdot \nabla Q(0, 1)$ and $(1, -1) \cdot \nabla Q(0, 1)$. Similarly as for region three, the same technique is used for the regions four and six.

5.5 An Iterative Method

An alternative approach to solve the triangle-to-triangle distance problem is parameterise the triangles such that the distance between them is formulated as a quadratic function. The overall scheme then is an iterative solution that approximates the minimum distance between two triangles using the Newton method.

Let x and y be two points belonging to triangle T_A and T_B. Assuming that points A, B, C span T_A and that points D, E, F are points of T_B, x and y can be defined using the following equations over their barymetric parameters:

$$T_A : x(a, b) = A + (B - A) \cdot a + (C - A) \cdot b$$

and

$$T_B : y(g, d) = D + (E - D) \cdot g + (F - D) \cdot d.$$

To find the minimum distance between T_A and T_A we minimise

$$f(a, b, g, d) = \|x(a, b) - y(g, d)\|^2 .$$

It is important that x and y stay within the area of the two triangles. The four parameters of the function f have to thus comply with six inequality constraints

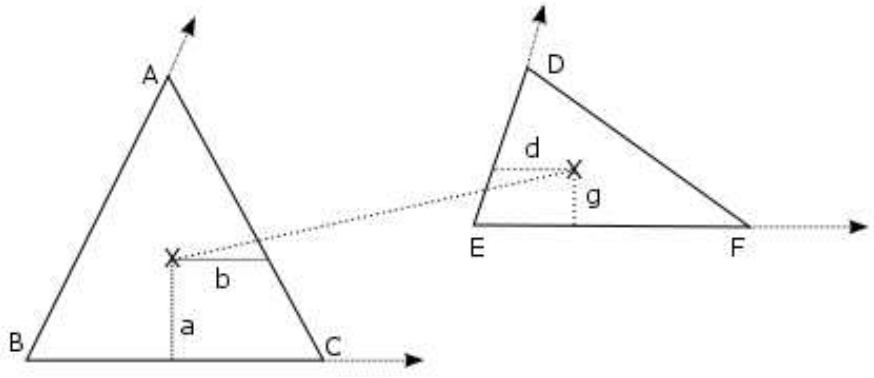


Figure 5.3: Example of minimum distance and the corresponding barycentric points (parameters of objective function) on a pair of triangles in 3D. Triangle X:T_A has points A, B, C where barycentric parameters a,b correspond to point x on the triangle. Triangle Y:T_B has points D,E,F where barycentric parameters g,d correspond to a point x. The two defined barycentric points define the minimum distance between the two triangles in 3D.

such that

$$\{a \geq 0, b \geq 0, a + b \leq 1, d \geq 0, g \geq 0, g + d \leq 1\}$$

.

The iterative method finds the minimum distance not primitive-wisely but rather using nonlinear constrained optimisation. A point on a triangle can be defined using triangle barymetric coordinates.

5.5.1 Penalty-Based Formalism

The penalty method enforces the constraints to the problem f using the penalising method. This approach adds a penalty term to the objective function to penalise the solution when outside of the feasible region:

$$P(x) = f(x) + r \cdot \sum_{i=1...6} max(0, c(x_i))^2 \tag{5.1}$$

where $r > 0$ is the penalty parameter and x is a, b, g, d. Newton iterations always converge to a solution but this solution might be slightly outside of the feasible region. Speed and "outsideness" can be controlled by the r parameter that controls the sharpness of the curve for the constraints. One aspect that requires care however is the invertibility of the Hessian $\nabla\nabla P$.

The problem is ill-conditioned, as the problem is based on the orientation of the

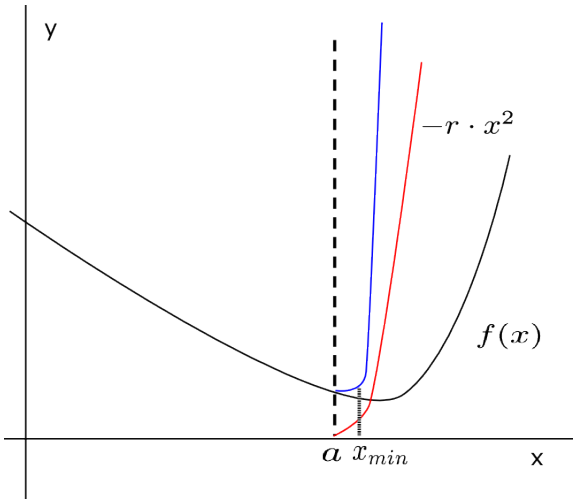


Figure 5.4: Illustration of a 2D problem showing the penalty function (red line) penalising the objective function (black line) f(x) under a constraint a (dash line) to create the feasible region (blue line).

two triangles and there can be more than one solution. As the Hessian matrix is not invertible, it is not possible to compute the Hessian and gradient. This illustrates the fact that f has multiple minima and $\nabla\nabla f$ is singular. Consequently, $\nabla\nabla P$ is also singular inside the feasible region. Because of the ill-conditioning, we use a quasi-Newton approach, where the Hessian is approximated by a perturbed operator $\nabla\nabla P + eps \cdot I$. I is an identity matrix and eps is suitably small.

The penalty algorithm as shown in Algorithm 5 accepts A, B, C, D, E, F vectors of triangles $T_A(A, B, C)$ and $T_B(D, E, F)$ as well as the required parameters for the algorithm to be solved. The penalty parameter r controls the steepness the P(x) function (5.1), `eps` is the perturbation parameter for the Hessian matrix, Tol is the tolerance for convergence (floating point accuracy). At line 23 of Algorithm 5 initial guess is chosen to be the centre of the two triangles, then the for loop initiates the Newton iterations to find the points on the triangle planes under the constraints c. For each of the six constraints (line 12) the max function of the penalty is determined so that every possible active constraint is detected. In line 17 and line 18 the gradient and Hessian of P is evaluated. Then the Gaussian elimination direct solver yield the Newton direction `DX`.

My optimised C implementation exploits matrix symmetries to reduce repetitions of calculations for matrix elements, for this reason `hf` in Algorithm 5 utilises symmetry. For the same reason, `X-Y` is only calculated once and stored into a variable `XY` so that its value is accessed instead of being repetitively calculated. Inside of the Newton loop, the optimised algorithm is totally different from the prototype.

Algorithm 5 Penalty Solver Algorithm.

```
1: function PENALTY(A, B, C, D, E, F, rho, tol)
2:     BA ← B − A; CA ← C − A; ED ← E − D; FD ← F − D;
3:     hf ← [2 · BA · BA′, 2 · CA · BA′, −2 · ED · BA′, −2 · FD · BA′;
4:     2 · BA · CA′, 2 · CA · CA′, −2 · ED · CA′, −2 · FD · CA′;
5:     2 · BA · ED′, −2 · CA · ED′, 2 · ED · ED′, 2 · FD · ED′;
6:     2 · BA · FD′, −2 · CA · FD′, 2 · ED · FD′, 2 · FD · FD′];
7:     x = [0.33; 0.33; 0.33; 0.33];
8:     for i ← 1 : 99 do
9:         X ← A + BA · x(1) + CA · x(2);
10:        Y ← D + ED · x(3) + FD · x(4);
11:        gf ← [2 · (X − Y) · BA′;
12:         2 · (X − Y) · CA′;
13:         −2 · (X − Y) · ED′;
14:         −2 · (X − Y) · FD′];
15:        h ← [ − x(1); −x(2); x(1) + x(2) − 1; −x(3); −x(4); x(3) + x(4) − 1];
16:        dh ← [ − 1, 0, 1, 0, 0, 0; 0, −1, 1, 0, 0, 0;
17:        0, 0, 0, −1, 0, 1; 0, 0, 0, 0, −1, 1];
18:        mask ← h′ >= 0;
19:        dmax ← dh. · [mask; mask; mask; mask];
20:        gra ← gf + ρ · dmax · max(0, h(:));
21:        hes ← hf + ρ · dmax · dmax′ + I(4, 4)/ρ²;
22:        dx ← hes \ gra;
23:        DX ← BA · dx(1) + CA · dx(2);
24:        DY ← ED · dx(3) + FD · dx(4);
25:        error ← sqrt(DX · DX′ + DY · DY′);
26:        if error < tol then
27:            BREAK;
28:        end if
29:        x ← x − dx;
30:    end for
31: end function
```

The derivatives of the constraints are stored in an array instead of the sparse matrix, so only the non-zeroes are used. Operator $max(0, h(:))$ is calculated without the use of the std library function $max()$ which is too generic for the code. It is possible to replace it with if statements and directly assign values to an array of active derivatives of constraints `dmax`, avoiding the masking operations in lines 10-11. The gradient `gra` is calculated so that any redundant operations are removed. The same techniques are used for the Hessian matrix `hes`. The point here is to end up with as few assignments as possible. The most significant aspect of the optimised algorithm is the linear solution in line 18. Unlike MATLAB which exploits a separate direct solver, our implementation merges individual operations of a 4x4 Gaussian elimina-

tion with the rest of the algorithm in a monolithic manner. This means that with Gauss elimination, calculation of the gradient and calculation of the Hessian are fused together into one compute kernel. Such solution streamlines the requirement of temporary variables assignments and floating point operations per iteration.

Operations like division are limited because of their computational latency, operations like addition, subtraction and multiplication are preferred. Divisions are slow because they are solved iteratively in hardware (floating point unit of a general purpose processor), although the actual hardware-based method may vary by processor manufacturers [62]. The penalty method is well-suited for SIMD optimisation because I can concurrently determine the distance between multiple triangle pairs as long as we use the same number of Newton steps: Up to four or eight triangle pair distances can be determined at the same time; depending on the vector width. Such a speed-up statement however has to be read carefully. While the concurrency is high, it is not clear a priori how many Newton steps are required. A high number of Newton steps can render the penalty method slower than the brute force approach.

In the case of two parallel triangles in three dimensions brute force and the iterative scheme may produce different points that define the same distance. When two triangles are parallel to each other the iterative scheme is ill conditioned as multiple solutions exist. Our novel iterative implementation fuses the direct Gaussian elimination phase with the Newton solver into a single monolithic algorithm, thus removing all computational redundancy.

5.5.2 Penalty Method Parameter Tuning

The serial penalty-based algorithm forms the baseline our optimised implementation. For this reason, it is important to identify correlations between the optimisation parameters and individual triangle sizes. Such a relationship between the penalty parameter and output error is identified and exploited (Figure 5.5). The error is defined as the degree of difference in the solution of the penalty solver over the brute force solver.

For an optimum penalty parameter we decided to tune the calculation on random triangles sizes that reside in boundary boxed domain. To maintain consistency in the randomness of the triangle sizes, various length of triangles are tested with random rotations. A wide spectrum of cases is thus examined for the purpose of parameter tuning. Using different scales of boundary boxes, a linear relationship is found between the size of the boundary space and the size of triangle and the penalty parameter. Figure 5.5 shows that for different sizes of triangles, the penalty

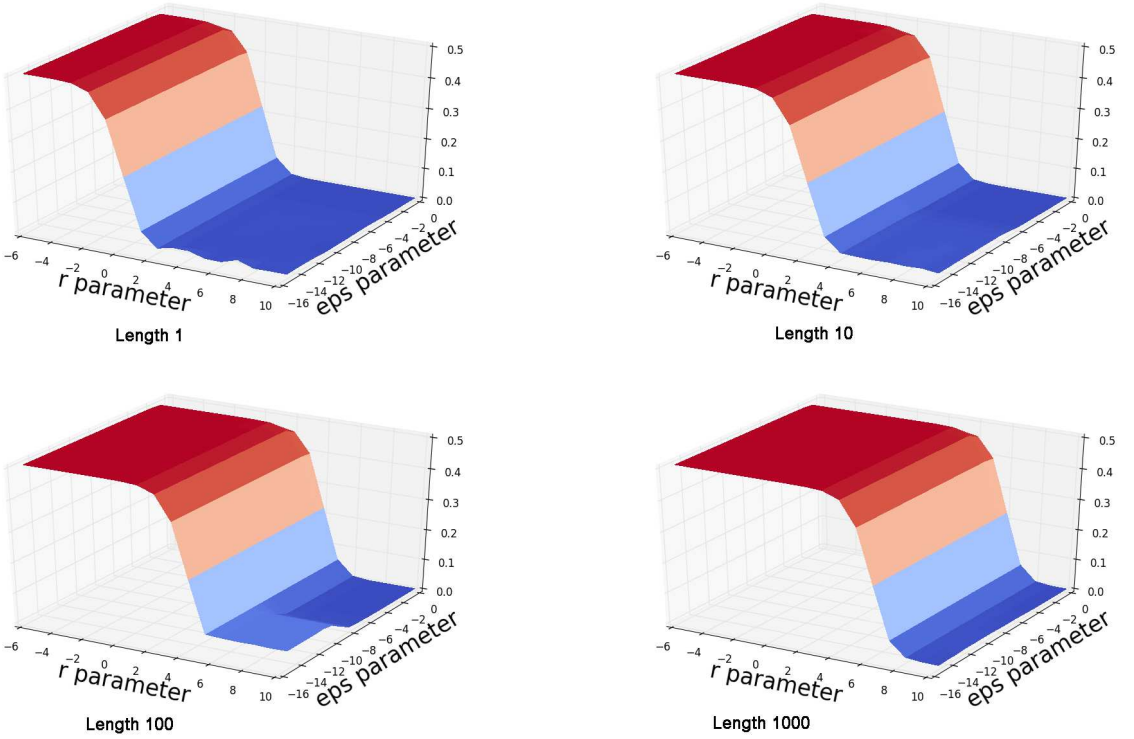


Figure 5.5: Tuning data for the penalty method to find a scaling relationship between the size (upper left to lower right figure) of problem, penalty parameter and error - using logarithmic scales

parameters correlate linearly with respect to the optimal equation that is derived based on the triangles size:

$$r_{optimal} = s \cdot 10^{(log10(s)+10)} \tag{5.2}$$

where s is the size of the triangle boundary (average side length). The error shows results for a hundred thousand of pairs of triangles.

The *eps* Hessian matrix regularisation parameter is tuned with respect to its effects on the failure rate at specific sizes of triangles but also the number of Newton iterations. To achieve a low on average 4-5 iterations (400,000-500,000 total iteration on average for 100,000 triangle pairs), eps parameter has to be small for small triangles. While the size of triangles increases, eps linearly increases. Using a low epsilon parameter doesn't mean optimal number of iterations, eps is adjusted using a coefficient of boundary size.

Debugging and experiments of the prototype code show that the first and second Newton iterations often oscillate around the convergence point, and the third converges to the solution (unless the point on the triangle is at a corner). The eps perturbation along the diagonal Hessian is increased at this point to complete

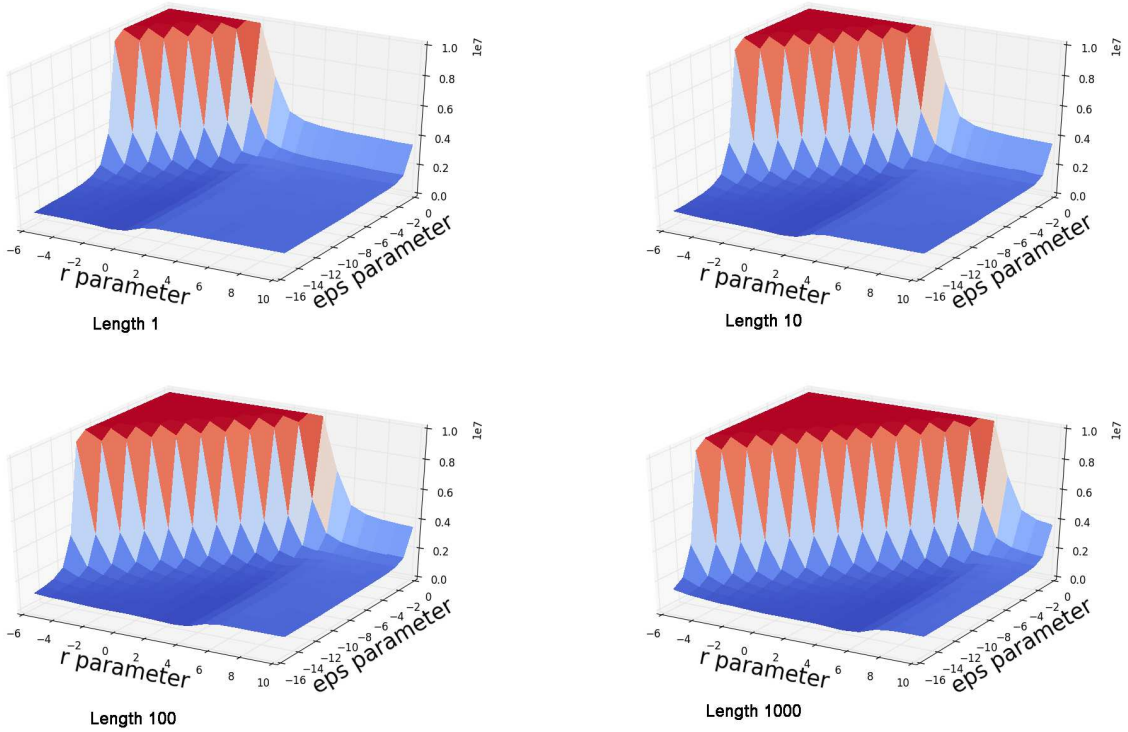


Figure 5.6: Relationship between eps and number of iterations for different lengths of triangles. Optimal eps should be low and increased depending on the r parameter.

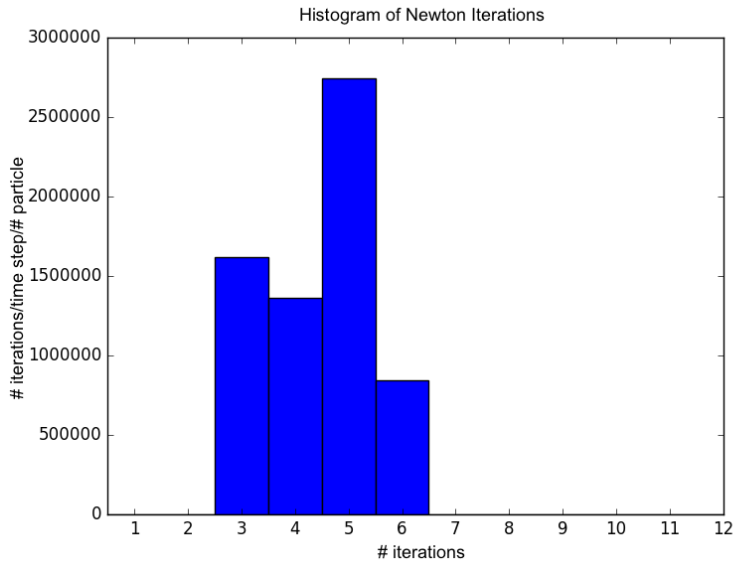


Figure 5.7: Histogram retrieved when tuning thousands of triangle pairs with optimal penalty parameters and epsilon perturbation of the problem. It shows that most problems converge within six iterations.

the convergence since it is known that the search is close to the convergence area. Statistically, the error is at the minimum at four iterations (Figure 5.7). Because penalty is used, it is inevitable that the solution has to be in a position spatially

outside (difference seen by the error values, see Figure 5.6). Epsilon initially cannot be too high but also not low because the solver cannot get the right direction (eps-r-iteration graph, Figure 5.6). The perturbation eps is increased after three steps for convergence with minimum error and minimum iterations.

The penalty parameters that act as a spring to the Newton step to keep it inside the feasible region is tuned against the error. Epsilon (eps) regularisation parameters are also tuned accordingly.

5.6 A Hybrid Method

The penalty-based triangle distance computation method is fundamentally not robust as it is an iterative solver approach. Our assumption is that there can be particle configuration cases where the penalty-based method under-perform in terms of numerically accuracy and time to solution (oscillations). As such, we propose a new method that combines both brute force and penalty methods into a single hybrid solution. The motivation is create a hybrid that is both robust and fast.

To create a hybrid solver I first assume that on average there are for a triangle pair only a few iterations required to arrive at a solution. Based on empirical studies and tuning of the penalty parameter and the regularisation variable, the majority of triangle pairs are solved within six iterations as shown in histogram Figure 5.7. I thus allow the algorithm to skip excessive iterations without convergence and carry on with the simulation. Secondly I introduce a user-defined tolerance of error, it acts as the switching point for the falling-back to brute force solver. If the number of brute force method fall-backs does not exceed the number of penalty-based solutions then our hybrid achieves a balance in terms of computational performance but also in error at the solution.

There are two variants of my hybrid method; the first is the hybrid-on-triangle-pairs and the second one is hybrid-on-triangle-batches. Both variants are developed to exploit the robustness of brute force while keeping the arithmetic intensity of the penalty method. The implementation of both methods takes into account the potential for shared memory scaling as well as data access continuity. In addition SIMD vectorised performance of the underlying methods and memory alignment are critical for the execution of both methods.

In a parallel execution setup the total number of pair-wise triangles are assigned to compute units (i.e CPU core). As the error already exists in the prescribed geometry configuration, the fall back frequency per core also depends on the distribution

of error in the geometry per triangle pair that located in memory. The frequency of fall-backs has to be uniform per core otherwise it can create execution imbalance in a manycore system. Imbalance can be solved with more sophisticated distribution of work (i.e. dynamic scheduling/allocation of computation), by optimising the granularity of computation (splitting the workload into finer assignments) and minimising the error.

Hybrid on Triangle Pairs. The first variant of hybrid is the hybrid-on-triangle-pairs. The hybrid-on-triangle-pairs method runs first through the penalty solver. If the solution is not within a user specified tolerance afterwards, it falls back to the brute force solution. The hybrid solver tolerance can be manually adjusted but in this implementation it is set to $1E-16$. The tolerance determines the distribution of error over a given set of triangle pairs. The input geometry and error tolerance are critical to parallel execution and load balancing.

Algorithm 6 Hybrid On Triangle Pairs

```
1: function hybridOnTrianglePairs(A, B, tol)
2:     for i ← 0 ... T_A do
3:         for j ← 0 ... T_B do
4:             triangleError ← penalty(A[i], B[j])
5:             if batchError < tol then
6:                 bruteForce(A[i], B[j])
7:             end if
8:         end for
9:     end for
       return P, Q                          ▷ points on the two triangles
10: end function
```

Hybrid on Triangle Batches. The second variant is the hybrid-on-batches. A hybrid method that is based on a per triangle batch comparison checks the error less frequently than the previous variant and uses an error for the whole batch. As in the hybrid-on-triangle-pairs variant, we run the penalty method on one batch of triangles and then fall back to brute on the whole batch if the error tolerance is not satisfied. A batch is a chunk of triangles that we decide to group to together to decrease the frequency of error checks. The batch size can be set by the user (batch size is also called granularity). For the current implementation the batch size is set to the number of triangles of the partner particle.

Both hybrid methods suffer by their nature by the granularity of the grain size whether that is singular size (triangle pair) or greater (batches of pairs). The memory distribution of pairs of triangles that do not converge within the mean number of Newton iterations are not known a priori because the solution depends on the

Algorithm 7 Hybrid On Triangle Batches

```
1: function hybridOnTrianglePairs(A, B, tol)
2:     for i ← 0 ... T_A do
3:         for j ← 0 ... T_B(batchSize) do
4:             batchError ← penalty(A[i], B[j])
5:         end for
6:         if batchError < tol then
7:             for j ← 0 ... B.T_B do
8:                 bruteForce(A[i], B[j])
9:             end for
10:        end if
11:    end for
       return P, Q                          ▷ points on the two triangles
12: end function
```

underlying geometry. Triangle pairs or triangle batches that do not converge within the set tolerance skew the overall error distribution margin, potentially creating the worse case scenario where the method becomes a worse than brute force solver with both penalty and brute force being executed in sequence. It is not possible to predict the sparsity distribution of non-convergent triangle pairs/batches during run-time so the tolerance value, penalty, regularisation parameters are vital. In our experiments those parameters are set based on empirical tuning and trial and error.

Various triangle distance methods are proposed, the brute force, the iterative penalty method, and a hybrid. The brute force method relies on geometric primitive computations, it is robust and a reliable solver. On the other hand the iterative scheme creates an non-linear optimisation problem that promises to deliver fast distance computation but lacks numerical robustness. The hybrid combines the two worlds into two hybrid variants that ensure robustness and a robust output.

5.7 Memory Organisation

The DEM algorithm implementation uses three sets of data types: geometric, behavioural and domain meta data. Geometric data handles particle properties and the physical geometry of the simulation. Behavioural data store pair-wise particle collision information. These additional data structures are required to support the contact detection, force and position update routines of the simulation. The DEM Grid Meta Data Chapter discusses the data structure optimisation in depth. In this section we discuss the geometric data and how to hold it into memory efficiently to

leverage SIMD hardware (i.e cache coherency, immediate addressing modes, locality of reference). The rest of the Section further discuss optimisation techniques and measurements.

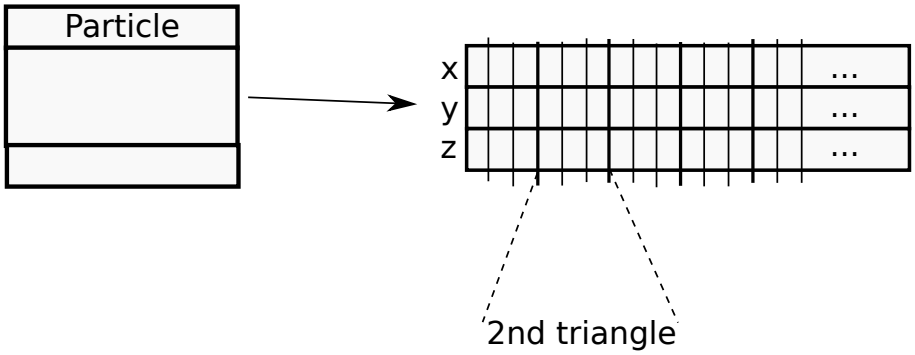


Figure 5.8: The two layer data layout of our DEM code with an AoS on the particle level but SoA for the vector entries with replicated vector entries.

We propose to realise the geometric data in two layers (Figure 5.8). A hull struct holds all particle properties such as velocities, rotation, mass, geometric centre and mass centre. Hulls link with pointers to the actual geometric data. This data is realised as structure of array (SoA), i.e. there is a sequence of x-coordinates, a sequence of y-coordinates and a sequence of z-coordinates, data sequences are not interleaved. These sequences are blown up with redundant data. The first three entries in the x array hold the x coordinates of the three vertices of the first triangle of the particle mesh. The entries four through six hold the coordinates of the second triangle and so forth. The degree of redundancy is determined by the particle mesh.

To leverage SIMD hardware and exploit the wide vector-based CPU registers structures of arrays are preferred. While counter intuitive for scalar programming it is an important design decision in vectorisation. A single triangle (three 3D vectors) stored in a single array (AoS) requires the execution of three separate load operators to transfer data to a CPU registers, while the same single triangle can also be stored in main memory as a set of arrays for each dimension (SoA, Figure 5.9). As such each, in SoA required data points fill SIMD vector registers with chunks of sequentially adjacent data. Streamlined data access exhibits a one-to-one (non-interleaved access) data access. We sketch the difference between the two AoS and SoA layouts in Figure 5.9 with the single triangle particle example.

We also present a detailed sketch of the data layout with respect to geometric data on Figure 5.10. It is noteworthy that we accept the increased memory consumption of such a structure. In return, we are able to avoid any indirect addressing. It is then possible to process all geometry data during the collision checks in a stream-

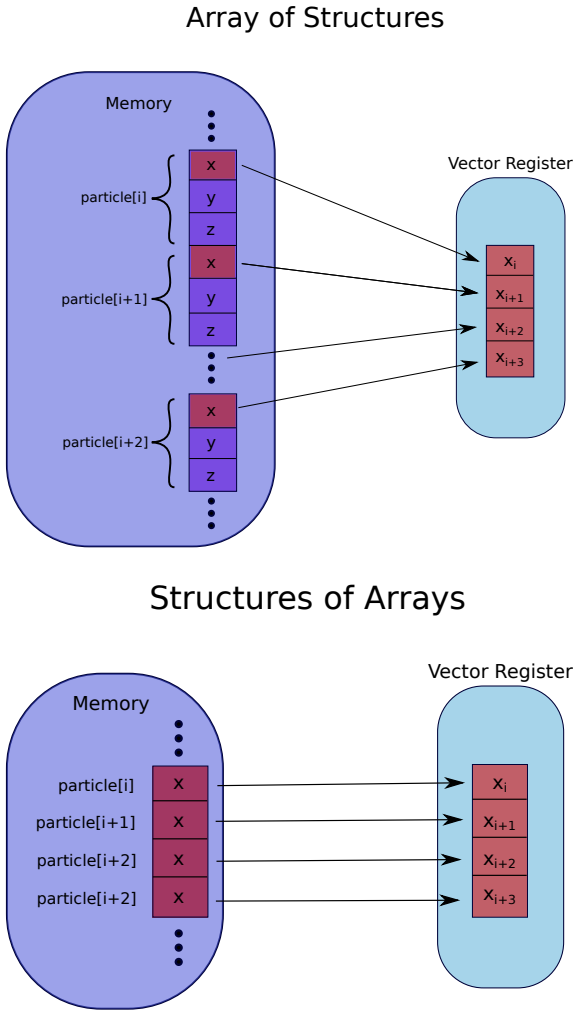


Figure 5.9: Top: Arrays of Structures (AoS) - Particle position data is interleaved (i.e. x,y,z coordinates are not sequential) so loads into CPU registers are not always streamlined. Bottom: Structures of Arrays (SoA) - Particle data is stored in separate arrays in memory (x,y,z dimensions), so CPU performs a streamlined load on a chuck.

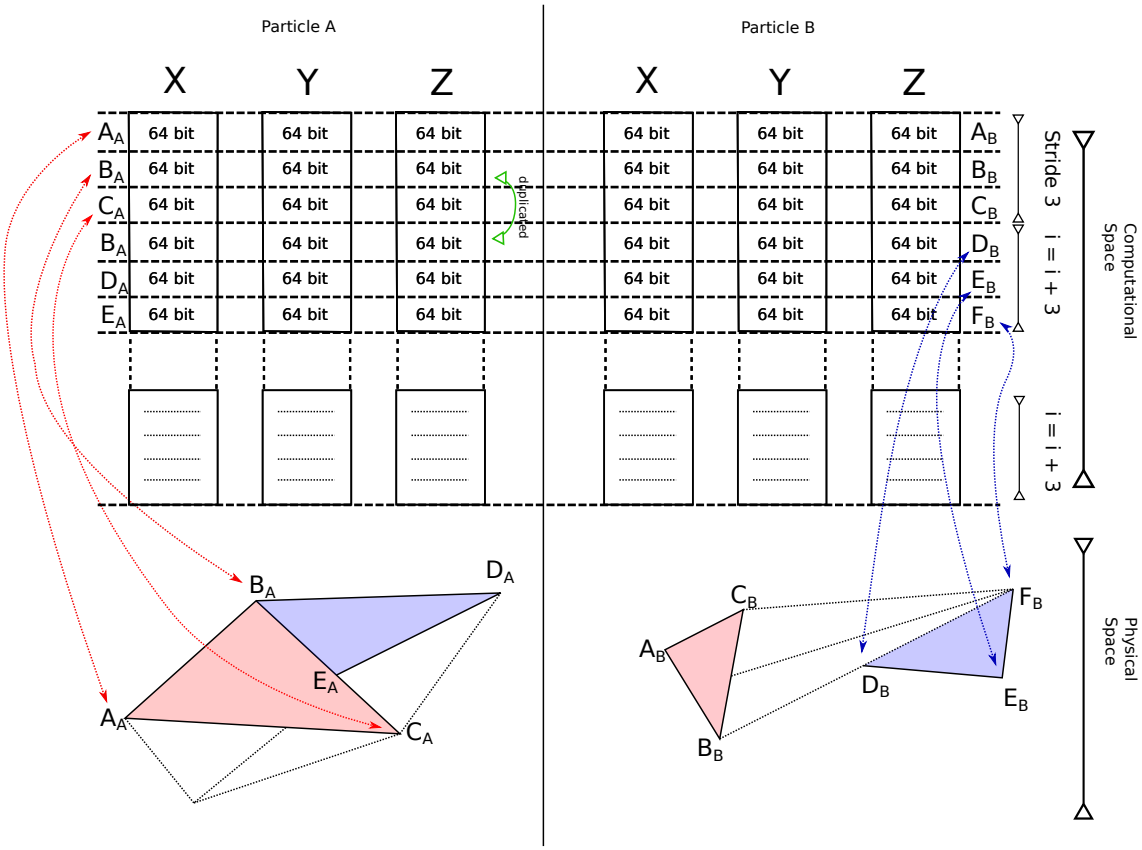


Figure 5.10: Two instances of triangulated particles A and B represented by points both on the real geometric Cartesian space (bottom section) and computational memory space (top section). Both particles are bisected by a separator line for illustration purposes. In both particles we highlight only two triangles out of the total triangles that their mesh is composed. Arrows to/from computational and physical space mark the the one-to-one point-wise correspondence relationship.

like fashion and we can align all vector entries. SoA data are notoriously difficult to handle if subsets of a data-set are to be transferred or data is to be reordered. In our particle handling, a particle is an atomic unit, it is never teared apart or resorted during the simulation run. A particle mesh is topologically invariant.

Behavioural data structure is accessed only throughout the phase of contact detection and force derivation. Behavioural data are composed of short length vectors (contact position) and scalars (distance, particle identities). Operations on these data are usually short because the information is required only temporary for scalar operations. Particle property-specific data like velocity and position are also loaded on short vectors.

5.8 Loop Footprint Optimisation

For the remainder of the chapter we discuss the execution of the DEM routine `findCollisionPoint` (Algorithm 8) which is utilised to identify contact points between two particles A and B. The identification of contacts between a pair of particles is the most computationally challenging step in DEM as the contact detection phase consumes the majority of the total runtime [44, 55, 72].

The quadratic complexity contact detection check is often wrapped into an additional check that compares bounding spheres or bounding boxes as a pre-check and thus may skip pair-wise comparisons all together. The sphere-to-sphere and box-to-box comparison is computationally cheaper than a mesh-to-mesh (triangles-to-triangles) comparison. Therefore, the pre-checks reduce the runtime when particles are not in collision. If the bounding areas overlap then we enter the phase of mesh-based collision check.

The mesh-based compute kernel is formed by a double nested loop (see Algorithm 8) in addition to an all-to-all n particle comparison. The outermost layer loop sweeps through all triangle vectorised points $\mathbb{T}_A$ of $\mathbb{A}$ and the inner loop runs over all triangles points $\mathbb{T}_A$ of $\mathbb{B}$. Within the innermost loop the contact detection is a SIMD operation.

Algorithm 8 Blueprint for Contact Definition Compute Kernel

```
1: for i ← 0 ... T_A ∃particles_n do
2:     for j ← i ... T_B ∃particles_n do
3:         findCollisionPoint(A[i]_x, A[i]_y, A[i]_z, B[j]_x, B[j]_y, B[j]_z)
4:     end for
5: end for
```

The contact detection functions (`findCollisionPoint`, Algorithm 8) is available

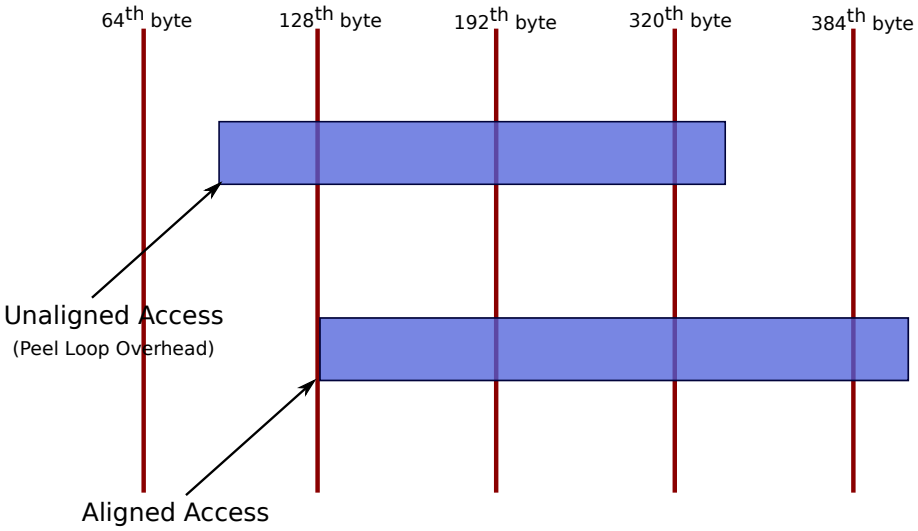


Figure 5.11: Data structures (arrays) are required to be aligned on specific memory locations as specified by the chip manufacturer. Data alignment and padding enable streamlined memory accesses as data is positioned at the natural hardware alignments.

as part of my *delta* library routines (i.e. brute force, penalty, hybrid on batches, hybrid on triangles pairs). Triangle vertices are passed as vectorised arrays which correspond to each x, y, z dimension. These are accessed continuously by `findCollisionPoint` with a stride-three iterator. The routine then returns the shortest distance between the two triangles A and B (Algorithm 8, line 3).

Streamlined and vectorised contact detection is enabled by our memory layout. Within a vector lane (i.e. SIMD inner loop) output vectors are stored at a equally sized vectors. As they carry the distance, we avoid SIMD reduction or scatter/gather operations, strided accesses between input and output vectors only in the end. The output vectors that store distances are then used to define unique pair-wise contacts within an ε margin.

Another aspect of loop optimisation is the explicit alignment of memory by padding accessed data to equally sized chunks, 32 or 64 bits depending on chip architecture (Figure 5.11). Within vector loops logical branching has to be avoided otherwise vectorised/streamlined memory access becomes unpredictable and often compilers are unable to switch on vectorisation.

We conclude with preliminary performance characteristics of the proposed algorithms. In Table 5.1, we show flops/s, cache miss rate, memory bandwidth of each proposed method. The measurement are taken by reading the hardware performance counters using Likwid [87]. The tests are performed on a single core Broadwell CPU E5-2650 v4, 2.20GHz. Our baseline is Stream [61], a matrix-vector multiplication

Table 5.1: Hardware counter results for characteristic single-core runs on the Broadwell chip. "BF" is brute force, "Hybrid T" is hybrid on triangle pairs, "Hybrid B" is hybrid on triangle batches.

Metric	**Stream**	**BF**	**Penalty**	**Hybrid T**	**Hybrid B**
Runtime (s)	6.71	0.906	0.917	0.894	0.907
MFLOPS/s	828.27	1094.59	1393.02	1293.26	1334.43
L2 Miss	0.1978	0.0001	0.0001	0.0001	0.0001
L3 Miss	0.1008	2.90E-08	5.53E-07	4.36E-07	1.32E-07
Bandwidth MB/s	8329.92	85.8905	96.081	78.2327	64.4084

Table 5.2: Hardware counter results for characteristic single-core runs on the Broadwell chip with SIMD enabled.

Metric	**BF**	**Penalty**	**Hybrid T**	**Hybrid B**
Runtime (s)	0.906	0.320	0.48	0.56
MFLOPS/s	1087.90	3529.57	2151.71	2410.75
L2 Miss	0.0001	0.0004	0.0002	0.0002
L3 Miss	8.28E-08	3.16E-06	2.40E-07	1.89E-07
Bandwidth MB/s	67.815	60.0901	44.6671	65.5665

benchmark that is memory bound (maximum memory bandwidth). For all methods the pressure on the memory subsystem is minimalistic when compared to the Stream baseline. AVX SIMD optimisations are visible via the higher levels of MFLOPS/s performed. We observe around three times speed up for the iterative penalty-based variant, and around two times for the hybrid methods (Table 5.2). Our penalty method exhibits high locality and high MFLOPS/s and is the fastest method while the hybrid methods are slower than the penalty iterative method, they are faster than the brute force method. All methods in SIMD version show increased MFLOPS/s except the brute force, this is due to the branching (if statements) that exist in the algorithm.

CHAPTER 6

Grid Meta Data Structure

Introduction. Space discretisation methods attempt to reduce the quadratic complexity of particle interaction from $O(n^2)$ to $O(n)$ where n is the particle count. Complexity reduction is possible through data storage schemes such as arrays, trees, hash tables, graphs. Various space discretisation techniques exist such as linked-cell lists [23] and Verlet lists [34]. Complexity reduction that use grid structures mirrors ideas in space discretisation that is commonly found in the molecular dynamics [23]. A grid divides the space domain into sub-domain cells, each sub-domain then handles its own local simulation information, this space discritisation strategy allows the decomposition of the interactions into local interaction areas.

For our DEM algorithm implementation, we rely on a generalised multilevel tree-based data structure that allows treating particles of a range of diameter sizes efficiently by exploiting a superimposed Cartesian grid. A tree data structure exhibits a tree-based hierarchical order (e.g. binary, quad, oct trees) within the grid. There are three observations that support the design decision of our grid.

First, particles that collide with one another are in close proximity in DEM simulations. It is thus sufficient to scan a certain environment around each particle for potential collision partners. We do not have to run through all possible particles pairs. Instead, for every particle we split up the domain into control volumes. The control volumes are cubic (in three dimensions) as it is a simpler implementation compared to alternative control volumes of other shapes.

Second, we make these control volumes no larger than the biggest particle diameter. It is sufficient to check the $3^d - 1$ (where d is the dimension) neighbouring

cells. Then through the traversal it is determined whether neighbouring cells host particles that might collide with the local particle.

Third, the previous decision is problematic if the particles in the domain are of extremely different size. The cell size determines the largest particle diameter. If we use a uniform cell size, many unnecessary collision checks are performed for small particles. If we use an adaptive grid, it is tricky to design the grid so that only direct neighbouring cells have to be studied. We thus propose to use a cascade of grids. If we have several grids embedded into each other then each particle is stored at different levels of the grid based on their suitable diameter. Particles of one grid cell then have to be checked against particles in their neighbouring cell as well as neighbouring cells on coarser grid resolution levels. There is no need to check that a particle of one grid resolution is in contact with particles of a finer grid resolution. If a particle A collides with a particle B, particle B also collides with particle A and such relations are thus already detected.

Literature review and alternative schemes. In grid-based space discretisation, when each particle is checked for possible collisions with other particles, in-homogeneous particle sizes pose challenges. As one particle might have to be compared to many other potential collision partners; the bigger the area a particle covers, the bigger the area we have to search for potential collision partners. This imposes significant throughput needs which might not necessarily translate into high computational load [9]. The state-of-the-art approach is to rely on a regular grid to check a particle within a cell only against particles residing in neighbouring (linked) cells, i.e. the regular grid yields the areas to be searched for collision partners. Such a linked-cell approach can be used as base for Verlet lists [35, 70] or more sophisticated check volumes [9]. Despite the fact that regular grids deteriorate for extremely in-homogeneous particle distributions [98] the largest particle dominates the chosen cell size, only [84] seem to use multiple meshes. Here, each particle is hosted within its "fitting" cell and cells are then compared to each other. Dynamically adaptive grids based upon recursive space decomposition also reduce the search area for collision partners. However, as long as particles reside on the finest resolution level [29, 34, 98], the deterioration around large particles is only localised but not eliminated. To the best of our knowledge, the combination of grid adaptivity with a hierarchical multilevel approach is not found in literature.

Alternative approaches found in literature select the grid cell length to accommodate the minimal particle diameter [55]. In this case, larger particles overlap multiple cells. Such an approach requires more sophisticated bookkeeping of particle-to-cell relationship. Another variant relies on unstructured grid space decomposition where

the domain is sliced non-equidistantly every-time, this strategy may also allow domain cuts to slice through particles. In unstructured grids, the depth of the tree hierarchy (depth of cell cascades) is shallower as the grid is tailored to particle shape and distribution [6]. In such variant, the mesh administration and communication scheme requires to merge or split objects during particle movement. The programming of such implementation becomes difficult and to our knowledge there is no clear design benefit. Instead for this project, we choose to employ a structured grid approach which prioritise algorithmic simplicity, a property that is notably enabled by structured tree-based decomposition.

DEM-based data structure schemes differ in terms of data decomposition. Codes are either based on the fixed assignment of particles to compute entities, either nodes and cores, or based upon the decomposition of the domain. In the latter approach, compute nodes or cores own domain fragments and implicitly own all particles residing within domains [34, 69, 99]. Studies on domain decomposition with multiscale, dynamically adaptive grids do — to the best of our knowledge — not exist for DEM. While a parallelisation of force contributions (i.e. check of particle pairs through pairwise interactions) is convenient in the molecular dynamics community [9, 69], DEM seems to be dominated by spatial parallelisation: different cells (sub-domain areas) are compared to each other concurrently. Our present work goes beyond selecting one parallelisation strategy as it compares spatial decomposition to particle-pair parallelisation. The latter is the equivalent parallelisation of forces in molecular dynamics codes.

While most approaches implement regular grids [71] or derive communication-minimising techniques from regular grids [9], such approaches can break down for strongly in-homogeneous particle distributions. Motivated by this, we emphasise on a realisation that relies on Peano [92] for which excellent memory and multi-node communication behaviour is validated [91]. We also emphasise that the particle administration is inspired from [95] where further communication-avoiding techniques are introduced. All paradigms proposed here however apply to any octree, quadtree, forest (clusters for trees) or spacetree data structure.

Chapter outline. In Chapter Grid Meta Data Structures we discuss the use of a Cartesian grid data structure within phases of the DEM algorithm. The Chapter is divided into four sections: Space Discretisation, Multiscale Grid Traversal, Multiscale Grid Morphology and Multilevel Data Inheritance. We start off with a discussion of Space Discretisation. Discretisation of space and the creation of hierarchies is the fundamental concept in grids. We show how a grid structure is applied to particle simulations. The Multiscale Grid Traversal section discusses the traversal

algorithm that sweeps through all cells. Grid Morphology defines the type of grid behaviour that arise from specific scenarios. The morphology give rise to properties that are at our disposal due to the grid dynamics. Multilevel Data discusses methods to keep information between different resolutions consistent.

6.1 Space Discretisation

A regular grid cuts the computational grid into cells (Figure 6.1). Every cell of the grid acts as a bucket for the particles. A bucket cell is the host cell that overlaps a particle's centre. Every cell has eight vertices in three dimensions (a cubic cell). Each vertex is traversed as part of the grid traversal algorithm. We then assign particles to their closest vertex. This assignment creates a particle-to-vertex association which is automatically inherited due its storage location in the data structure. We choose vertex-based association instead of cells as these allows us to further group particles within a cell but also exhibit better communication memory throughput during grid traversal [95]. Our grid-based space discretisation cuts space into sub-regions/sub-cells. To find potential collisions, it is sufficient to check solely against particles that are within the local and adjacent area of neighbour cells. The premise is that the search locality reduces the overall search cost. The space cuts effectively implicitly propagate sub-domain boundary information. Sub-domain information is used to infer the locality of particles.

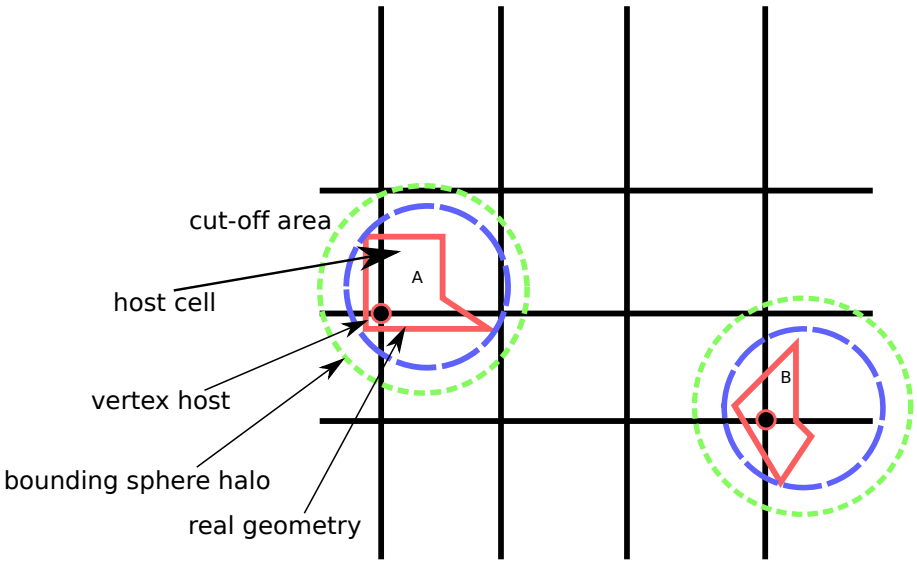


Figure 6.1: Two particles in 2D without hierarchical multiscale levels. In this configuration, the two particles are not compared, only if the two vertices are adjacent are the particles compared for collision.

Following the cell-based scheme it is then possible to extend this into a multiscale grid. A multiscale grid is a set of regular Cartesian grids that are embedded into each other. The embedded cells yield a hierarchical tree-based order, a parent cell has children cells. The cascade of cells allow us to treat particles of varying scales by placing them on different levels of the grid structure. We base the construction scheme on the unit cell, this is cut into three equidistant pieces along each coordinate axis which yield $3 \cdot 3 = 9$ cells in 2D. The nine cubes form the first level of elements/sub-domains/cells. The partitioned cells are called children of the bounding box cube which is the root, parent or coarse cube. For each of the children, the discretisation algorithm recursively splits again the cells if required. A leaf cell is a cube that is not refined further and it is locally the smallest in size in the hierarchy. The decision to cut into three parts is based upon the default three-partitioning design of the grid framework [92] at hand. Bi-partitioning also works on such setup which yields a binary tree in two dimensions.

The spacetree order of the discretisation exhibits a parent to child relationship property analogous to tree structures. Each cell besides the root has a reference to a unique parent cell. While cells could hold particles, we propose a vertex-based scheme [95]. A vertex is uniquely identified by its spatial position and the level it resides. The level of a cell indicates the number of refinement steps required to at least create one of its adjacent cells.

With the creation of the spacetree we superimpose a meta data structure [92] that run a space filling curve (SFC), a linearised array that runs through cells and vertices. By design, we choose each vertex to hold a list of particle associations. A particle is always stored at the finest grid level where the cells' edge length is still larger than its diameter. A particle is always associated to the vertex closest to its geometric centre. Links from the vertices to the particles are realised as pointers. If a particle migrates (i.e. position update) to a different vertex we update the associated vertex-to-particle pointers. Thus, there is no movement of geometric data in memory.

Our multi-scale definition exploits the hierarchical relationship between grid vertices (Figure 6.2). A vertex a is a child of a vertex b if all adjacent cells of a are children of adjacent cells of b. b is the parent vertex of a. A vertex a is a child of b if at least one adjacent cell of a is a child of an adjacent cell of b. b is an parents of a. This hierarchy of vertex links is vital throughout the simulation because it maintains consistent exchange of information between levels and particle scales. Just like in tree data structures our coarse particles are "inherited" by children vertices at finer levels via pointers. We name these particles "virtual particles" and their

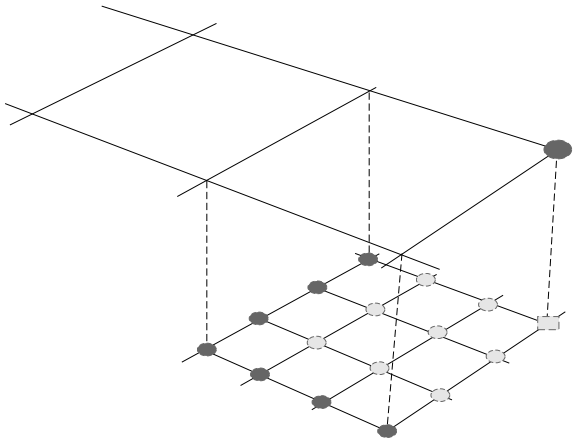


Figure 6.2: Particles are dropped from the coarse levels into the fine grid if new cell levels are added (inheritance). The bright round vertices are children of the marked dark coarse grid vertex. The smaller dark round markers' vertices are part of the children cells but may also belong to more than one parent/coarse cell.

hierarchical vertex-to-particle association links "virtual links" as they are inherited. Particle associations within a level are named "real links" and "real particles". We implement a list of actual held real particles and a list of virtual particles at every vertex. A local vertex is always read for the first time before any of its children and we loop through the virtual particle list first. We compare the locally held real particles with all other adjacent real particles and then those with all virtual particles. This scheme realises our multiscale particle-to-vertex association scheme.

The introduction of a grid over the computational domain in a DEM particle simulation pays off (Figure 6.1). The grid implicitly provides a cut off radius for the collision search of each particle. The discretisation allows a simulation to reduce its overall complexity to linear. A multiscale scheme allow us to position particles of varying scales on different levels of the grid. Grid cells form a cascade a sub-regions that map a hierarchical tree structure that can be traversed top down. The spacetree provides us with meta data (particle-to-vertices associations) that follow the hierarchical order of vertices. Every particle is stored at an associated vertex based on its size and position.

6.2 Multiscale Grid Traversal

With a grid at hand, the serial DEM algorithmic steps are translated into a top down grid traversal (Algorithm 9). We implement this strategy by running through the grid spanned by the spacetree, we combine a depth-first order with a space-filling

curve [90, 93] (Figure 6.3).

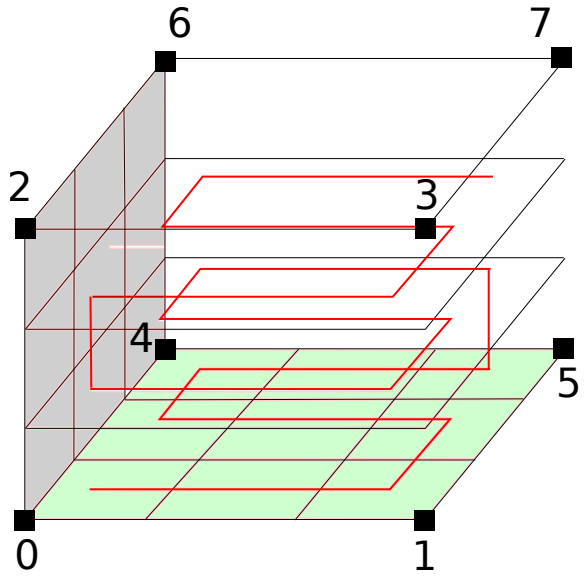


Figure 6.3: The Peano space filling curve is used by the grid to linearise the computational space. The traversal of the grid domain follows the curve as shown. Each sub-cell is located on the path of the curve.

The DEM algorithm is written down as a set of events [91, 94]. The set of events are based on traversal operations and specify which operations are to be performed. If a vertex is read for the very first time we invoke `touchVertexFirstTime`. If a cell is entered we invoke `enterCell` and so forth. The grid meta data structure hosts the particles. Hence, we make use of the traversal events to superimpose the DEM algorithmic phases. As shown in Algorithm 9 the DEM operations follow the grid sweeps and we build and maintain a set of collision points to implement the dynamics.

We start off with traversing the grid depth first. The traversal triggers some events. In those events we implement the DEM algorithm. On `touchVertexFirstTime` we derive all forces associated to each particle that reside at the vertex and we update the particle position. At this point we also inherit particles that reside on coarser levels with respect to the local vertex. Then on `enterCell` we reassign particle-to-vertex association locally once, re-assignment is performed only once locally but re-assignment checks may be performed more than once globally by neighbouring cells as vertices within the grid are shared with more than one cell. On `enterCell` we compare all pairs of each local vertex to detect possible contacts (inter-vertex comparison). Finally, contact detection is again performed on `touchVertexLastTime`, all pairs of local particles associated to the vertex are compared (intra-vertex comparison). The traversals continue until the termination condition is reached.

Algorithm 9 Grid-Based DEM Implementation.

```
1: function traverseGrid(C)
2:     C_old ← C
3:     C ← ∅
4:     while traversal continues do
5:         if touchVertexFirstTime then
6:             for all particles p associated to vertex do
7:                 for all contact points c ∈ C_old associated to p do
8:                     Update f_trans(p) through c
9:                     Update f_rot(p) through c
10:                end for
11:            end for
12:            for all particles p associated to vertex do
13:                Update particle incl. its triangles
14:            end for
15:        end if
16:        if enterCell then
17:            for all 2^d vertices do
18:                for all particles p associated to vertex do
19:                    if particle should be associated to different vertex then
20:                        Reassign particle
21:                    end if
22:                end for
23:            end for
24:            for all (p_i, p_j) particle pairs of vertex pairs (i, j) do
25:                C ← C∪ findCollisions(p_i, p_j)
26:            end for
27:        end if
28:        if touchVertexLastTime then
29:            for all particle pairs (p_i, p_j) associated to single vertex do
30:                C ← C∪ findCollisions(p_i, p_j)
31:            end for
32:        end if
33:    end while
34: end function
35: function main(T)
36:    C ← ∅
37:    t ← 0
38:    for t < T do
39:        traverseGrid(C)
40:        t ← t + Δt
41:    end for
42: end function
```

The grid-based realisation is characterised by few properties:

- When a vertex is loaded for the very first time, all particles associated to this vertex move if a force acts on them.
- Afterwards, all particles that are associated to a vertex are compared with each other and this implements the local collision detection that identifies possible contact points.
- The comparison of particles associated to different vertices is realised as we enter a cell event. All particles at this stage already have an updated position. All vertices at this point have been loaded already as they have been subject to `touchVertexFirstTime`.
- A particle may move at most one cell of its level at a time. Cell tunnelling occurs when a particle position is updated and a cell pass is skipped entirely, this is an issue that can potentially lead to missed contacts. Cell tunnelling [96] is prohibited. Tunnelling is possible when particles move fast. For some setups we experimentally do not allow jumping to non-neighbouring cells and vertices by manually setting a small time step size. However, this is not explicitly enforced but it is ensured by the proper time step choice (see Chapter 7).

Vertex traversal per cell. On the `enterCell` event, we perform pair-wise neighbouring vertex-to-vertex checks. As cell vertices may be adjacent to more than one cells, it is vital to avoid the redundant pair-wise vertex checks within a group of cells in the domain (Figure 6.4). Per cell, the maximum number of vertices that are checked is 27 (Figure 6.5). When a cell is visited then all its adjacent vertices are available to be checked. Yet, we avoid that vertex-pairs are checked twice by their left and right adjacent cells. Traversal of all cells and vertices realise a global linked-cell propagation along the domain and ensures that all particle pairs are evaluated.

The grid traversal yields a set of collision points that book-keeps case of pair-wise particle contacts. The collision point set is kept persistent for the subsequent traversal as one time step of the scheme is realised per two total grid traversals. The amortised cost however still is one time step per grid traversal. Such a scheme picks up the idea of pipelining [93]. By implementing a pipelined algorithm, we shift the DEM phases relative to the traversal to ensure that particles are accessed only once. Forces are derived from a single hash table of contact points using unique global particle identifiers. Force derivation is followed by position update at the following

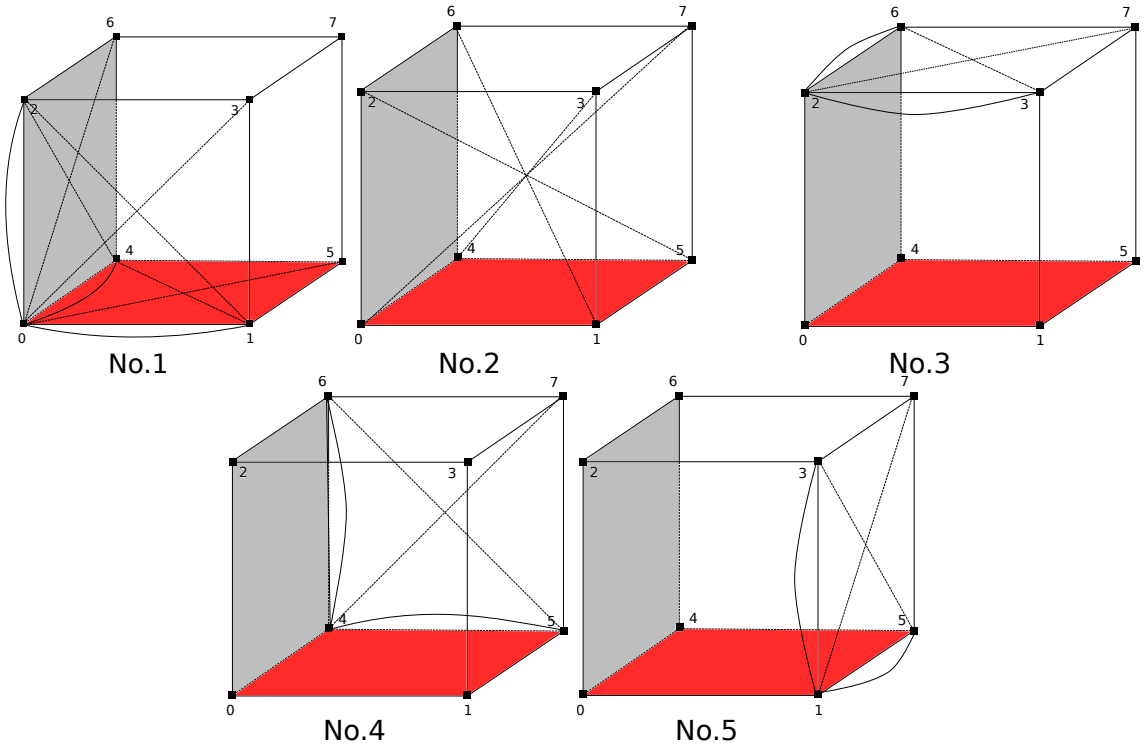


Figure 6.4: For the purpose of reference and debugging we group pairwise checks into vertex traversal groups; A traversal cell (no. 1), diagonal cell (no. 2), top boundary cell (no. 3), back boundary cell (no. 4), right cell (no. 5). The majority of cells visits perform the traversal cell vertex checks while the minority of cells are positioned at the boundary of the domain. At the end of a whole grid traversal all vertices are checked.

grid traversal (Figure 6.6). Collision data is mapped onto the particles when we read a particle for the first time in the subsequent traversal.

We maintain implementation simplicity by refraining from storing each contact point between two particles twice with inverse normals. Each contact is augmented with a copy of the corresponding partner particles' global data (mass, velocity, momentum). Otherwise, we would have to search for this data and build up global indices in addition to data being subject to a write constraint as particle's data already might have been subject to the next time step. Immediately after determining the forces that act on a particle, the physical and geometric properties of the particle are updated due to translation and rotation.

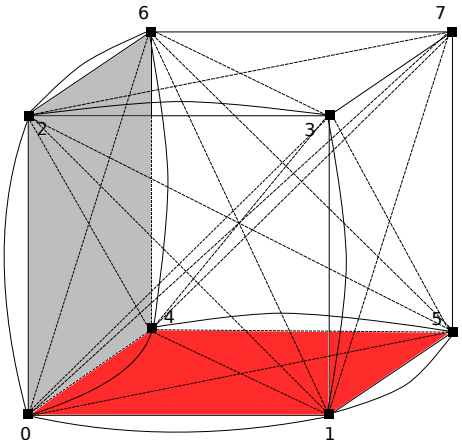


Figure 6.5: The sum of all unique vertex comparison minus the three comparisons that are left only for the last cell in the domain (6-7, 3-7, 5-7).

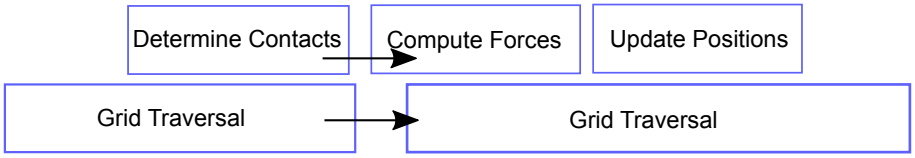


Figure 6.6: The contact forces are derived on the next traversal following the concept of pipelining.

The grid traversal exploits the space-filling curve to traverse all cells at all levels using a depth first traversal. Particles are stored at the vertices and all vertices need to be visited and compared against each other through a single-touch cell policy (single grid traversal). Our grid-DEM traversal follows the concept of pipelining, this allows our to pass visit particles for force derivation only once. It is important to detect pairwise redundancies in order to prevent more than one checks of the same particle pair.

6.3 Multiscale Grid Morphology

The formalism allows us to realise at least four grid variants: no-grid, regular, adaptive and reluctant adaptive grids. These grid variants form four types of morphologies that exhibit different properties for different particle setup scenarios. A deteriorated space grid (Figure 6.8) is the mono-cell grid which constitutes of a single cell of eight vertices in three dimensional space. All the remaining grids form hierarchies.

Regular grid. The regular grid version recursively refines all spacetree nodes until all children cell lengths are just larger than the smallest particle diameter

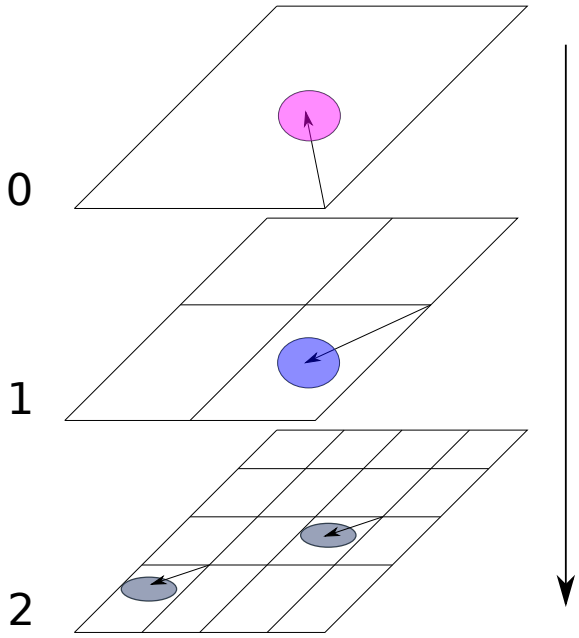


Figure 6.7: A single particle is dropped from a coarse cell into a fine cell. All fine cell are uniformly refined along the fine levels. The grid refined up until the particle is hosted at a cell no smaller than its diameter.

(Figure 6.7). This grid strategy yields a multiscale regular grid as every refinement is applied uniformly on all grid cells at each level. Each particle is assigned to the correct grid level as we drop and reassign them downwards the hierarchy levels until they fit into the correct cell length (Figure 6.8).

A regular grid is a naive approach to grid refinement as we allow the creation of large numbers of uniformly sized cells where no particles reside. The regular grid is a multiscale grid that refines every cell. The configuration of a regular grid is advantageous when the engineering scenario that defines a composition of particles that are equally distributed and sized.

The large number of cells become problematic when some particles are very small relative to the biggest one. If the simulation scenario exhibits clusters of particles of varying sizes then the regular grid by design realises redundant particle-to-particle comparisons. This is because the regular grid discretises space uniformly and it disregards particle scale and movement during runtime. When particles are not uniformly sized and distributed in clusters then an adaptive grid scheme is desirable. Another major disadvantage is the high number of particle comparisons ($\times$ triangle counts) required when particles cluster (Figure 6.10) into one region of the domain.

Adaptive grid. The dynamically adaptive grid variant is characterised by two elements, the mesh refinement control and inter-grid particle treatment as shown in Figure 6.9. The algorithm successively drops particles down the spacetree hierarchies

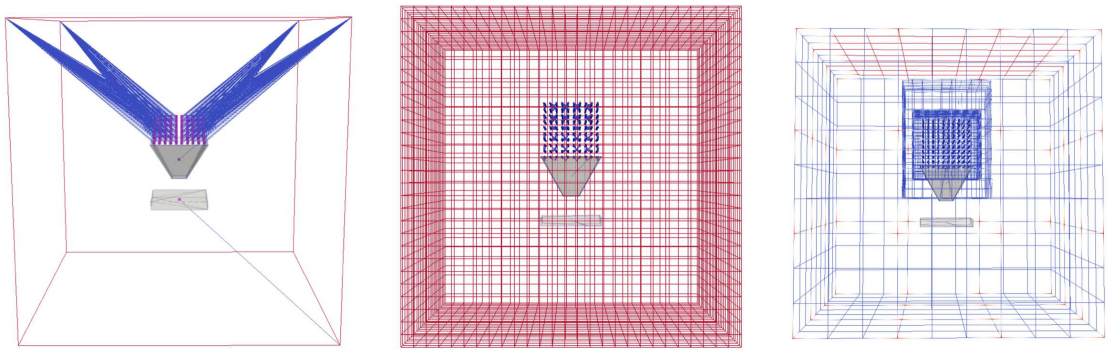


Figure 6.8: Left: A single cell hosting particles that are positioned on top of a hopper structure. All particles are associated to a cell vertex (blue line) at the root level. Middle: A regular grid that is refined uniformly across the finest level. Right: An adaptive grid adopts the grid around particles but does not refine cells that hold no particles. Both regular and adaptive grid types hold the hopper structure at a coarser level.

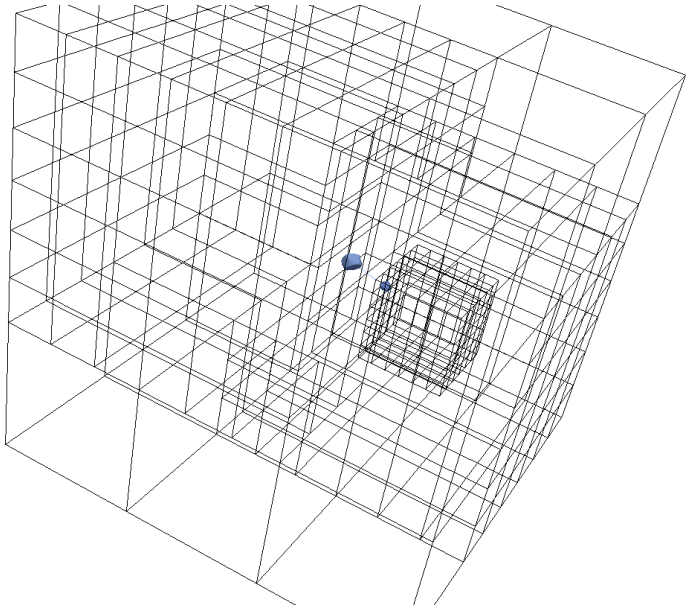


Figure 6.9: Two particles approach each other. As they are of different size they might be held at different spacetree resolution levels.

and it simultaneously refines coarse cells to match particles (Figure 6.8). The drop is performed during the first vertex visit on `touchVertexFirstTime` (Algorithm 9). The algorithm determines the smallest diameter of all the particles held by the vertex host. If the particle diameter is smaller than 1/3 of the mesh edge at the level of the vertex then the region around the vertex is refined. At each gird vertex, we check whether there are spatially coinciding vertices on a coarser level. If such vertices do exist, the particles that they hold are moved one level down as long as their diameter permits. The opposite hierarchical movement occurs when a cell is coarsened, this happens when particles move towards a neighbouring cell that is coarse, it is "lifted" and dropped to its level again.

In adaptive grids, we vary the morphology according to the dynamics of the particles under a set of conditions and rules. If we delete a vertex a that holds particles, its particles are moved to the next coarser level and reassigned. Each vertex holds a Boolean marker that is set $\perp$ before the vertex is read for the first time. If a vertex holds a particle, all the markers of the vertices where they are descendants of are set to $\top$. If a vertex whose adjacent cells are all refined holds $\perp$ at the end of the multi-scale traversal, we coarsen these refined adjacent cells. We rely on a top-down tree traversal. The refinement/coarsening procedure is evaluated on-the-fly and it is analogous to an analysed tree grammar [47].

A cascade of uniformly refined cells create a regularly refined grid. When we make changes to the refinement criteria then we can vary the refinement strategy from cell to cell. The selection of criteria per cell triggers refinement and coarsening events that form dynamically varying adaptive grids. The combined behaviour of criteria based on moving particles and grid rules can create a very dynamic grid morphology. To avoid undefined behaviour, it is important that particle-to-vertex associations are consistent with the grid behaviour at all times. From here on we do not discuss the mono-cell and we treat it as a no-grid variant that only holds the root cell.

The dynamic adaptive grid is a great improvement over the regular grid, as the number of particle-to-particle comparisons is reduced (Figure 6.10) and thus computation during contact detection is minimised. Particle sizes per level dynamically influence the space discretisation at this stage. The inheritance of particles from coarse to fine vertices becomes the key concept that enable such grid. We can construct and adopt the adaptive grid on the fly at every time step. However the adaptive grid only pays off when particle dynamics surpass grid administration overhead at every time step. The adaptive grid is great for reducing the total number of comparisons but it is also significantly more complex to implement and debug when

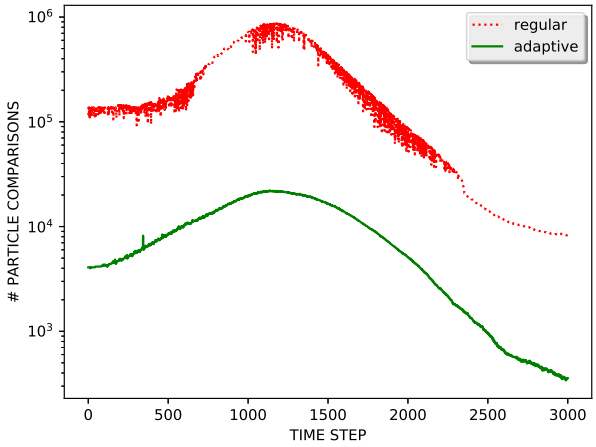


Figure 6.10: Particle comparisons of the hopper (1k particles) flow simulation using regular and adaptive grids. The regular grid make use of more cells/vertices than the adaptive variant and it refined uniformly on the whole domain. The adaptive grid refines only on the areas of particles thus the number of grid vertices is reduced significantly (and grid overhead), this also reduces the number of comparisons required as there are fewer vertices to compare at collisions areas. During the peak of hopper flow particle clustering phase (step 1200), the regular grid performs 850,000 comparisons while the adaptive grid only 25,000, this translates to 34 million versus 1 million triangle comparisons for granulates of 20 triangles for around 2000 actual collision points.

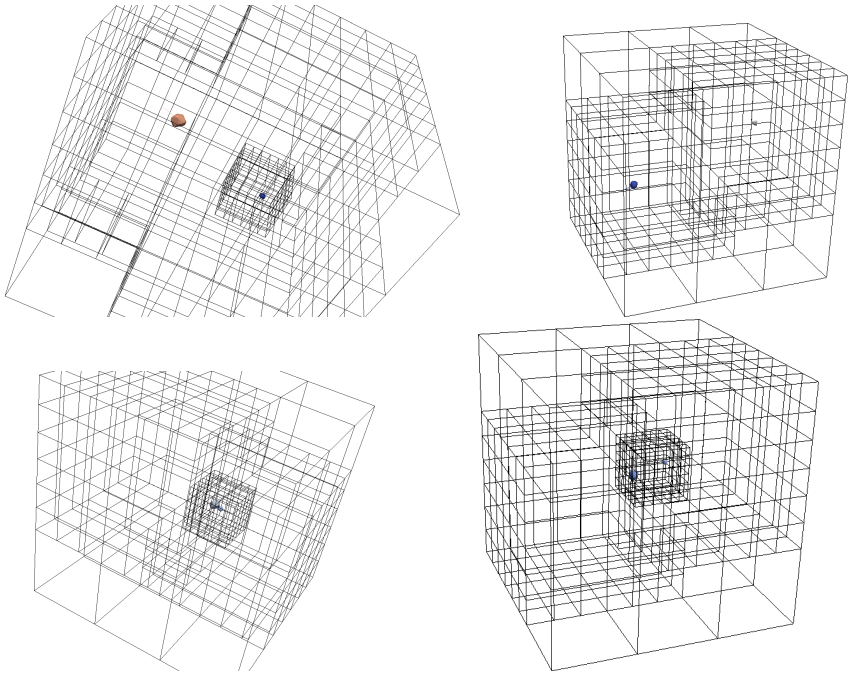


Figure 6.11: Two particles collide into each other. The adaptive grid refining around each particle while its diameter constrains the mesh size (left column - top and bottom left figure). The reluctant adaptive grid works with a coarser resolution as long as particles are far away from each other (right column - top and bottom right figure). Just before they collide, the grid is refined and particles are dropped down the resolution levels.

compared to the regular grid.

A reluctant adaptive grid. The dynamic adaptive grid refines aggressively, when a particle is smaller than the length of the cell that they are associated with we refine the grid and the particles are dropped to the finest refinement level. Adaptive grid regions follow particles (Figure 6.11) and the empty cells are coarsened. The downside of aggressive refinement is that it can create a larger number of fine cells than the minimum number of cell required for contact detection, this translates to grid administration overhead. Thus employing a relaxed-reluctant grid refinement strategy gives us the twin benefit of a minimal grid refinement administration and the minimal particle comparisons required from an adaptive grid.

The spacetree hierarchy allow us to realise four types of grids: no-grid, regular grid, adaptive grid and reluctant adaptive grid. These grids are subject to flag rules over time to maintain consistent multiscale morphological changes between levels.

The reluctant adaptive grid alters the behaviour of the standard adaptive grid

through two modifications of the refinement criterion. If only one particle is held by a vertex then no refinement is triggered as long as its particular diameter length is smaller than the cell size. Moreover, the refinement is triggered only if the vertex visited holds at least two particles that approach each other. Two particles A and B at dt step are approaching each other when the relative velocity difference

$$v_{BA} = (v_A \cdot d_{BA}) - (v_B \cdot d_{BA}).$$

along a particle distance line d_{BA} is less than zero. The distance line is defined as $d_{BA} = c_B - c_A$ where c_A and c_B are the centres of the two particles. A relative velocity difference v_{BA} that is greater than zero indicates particle separation. When particles move away from each other no refinement is required.

The reluctant adaptive variant ensures that the adaptivity is not as aggressive and the morphology behaves reluctantly with respect to the particle movement. The reluctant adaptive grid refines only when two particles are in close enough proximity (they exist in neighbouring cells) and the particles are approaching towards each other. An important property of reluctant adaptivity is that small particles are allowed to reside in coarse cells when they are lifted. Fine particles that are lifted and reside on coarse cells are not allowed to be dropped to finer cells unless the refinement criterion around their associated vertex is fulfilled (i.e. there are potential collision candidates at adjacent locations).

We present a novel use of the spacetree with the implementation of an adaptive and reluctant adaptive scheme. The adaptive grid premise is the reduction of the number of particle comparisons. Grid adaptivity pays off (Figures 6.11, 6.10) when particles of different size exist and when particles are irregularly distributed over the domain space. We employ a reluctant adaptive grid that aims at reducing the overhead associated with aggressive refinement and we show that the association of particles to coarser levels reduces grid administration overhead.

6.4 Multilevel Data

To implement complex grid morphologies, it is vital to rigorously address the multiscale data organisation between levels. We thus discuss the multilevel data hierarchy with respect to DEM particle data movement from fine to coarse vertices and vice versa. Particles relocate their data association within a cell to another vertex due to particle position updates and preserving consistency is important to avoid unde-

fined behaviour. Particle associations are dropped and lifted within levels of the grid as the grid morphology changes around them. Adaptive mesh refinement combined with multiscale collision dynamics add another dimension of complexity to the DEM design.

In this section we discuss the four components that define major data structure schemes within a DEM setup. Firstly, the particle vertex transition through the grid. Secondly, the transient virtual particle links. Thirdly, a geometry decomposition strategy. Fourthly, the filtering mechanism we have in-place to preserve robustness.

6.4.1 Particle Hierarchy Transition

Particles are dropped and lifted to and from specific vertices due to refinement and coarsening of sectors of the spacetree. On simulation initiation all particles placed in the hierarchy are dropped down to reside at vertices of cells whose side length fit the particle diameter. This initial drop ensures that the particles are fitted into the correct level of the grid. Then the simulation starts and the morphology start to change according to the dynamics.

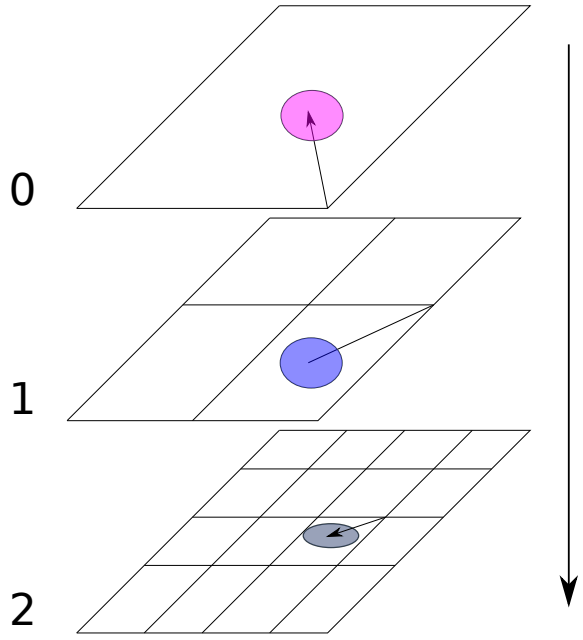


Figure 6.12: A particle is dropped down the grid levels due to its size.

In Figure 6.12, a particle is dropped from level zero to level one (pink circular particle). Its vertex association is reassigned to a different level because it fits into the next level of refinement. At level one the particle (light blue) is reassigned to a different Cartesian location due to vertex proximity. Eventually, the particle is dropped to the finest level (level two) and resides there.

When there is grid movement due to adaptivity, a lift call on a particle may be issued. In Figure 6.12 a particle at level two has to be lifted to level one if finer cells at level two are destroyed. A particle has to be lifted due to the destruction of hanging vertices. The lifted particle then need to be associated with a coarser vertex. A hanging vertex is a vertex that is required to form complete cell at the interface of adjacent cells of different refinement levels. Each vertex of a cell can accommodate unlimited number of particles as long as every particle rightfully belongs to the parent cell and remains within the physical constraints of non-penetrability enforced by DEM dynamics.

6.4.2 Virtual Particles

Our multiscale collision detection works top down. The top level coarse particles can only collide with equivalent level particles, while finer particles can collide with coarser particles. This approach removes the redundancy of a two way data exchange (downwards and upwards). We employ only a single way (downwards) data transfer which is coupled with the spacetree traversal. We derive coarse particle information at finer levels of the grid [89] on the fly.

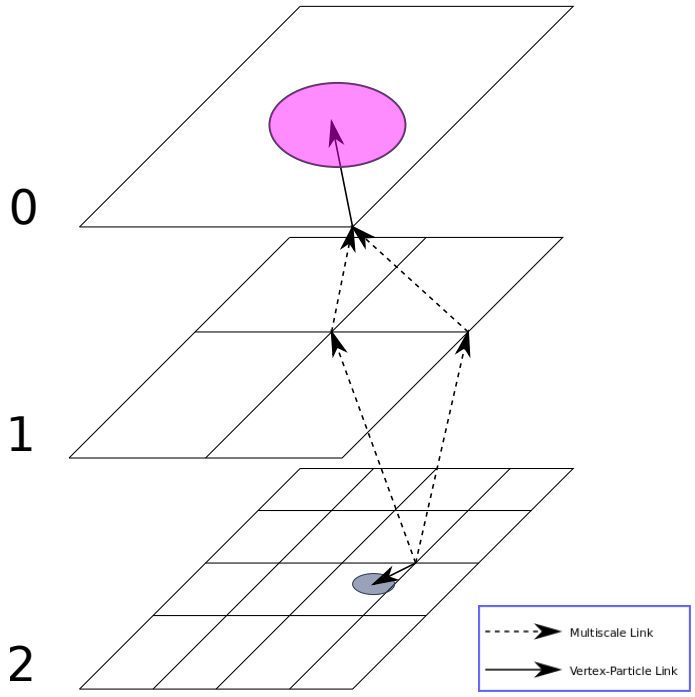


Figure 6.13: A virtual particle is a particle that is pointed by a vertex at a finer level. A virtual particle is inherited following the tree hierarchy of the grid.

The coarse particle set is derived at finer levels by the inheritance of coarse associations to the local vertex. Inherited particles are called "virtual particles" and

multiscale links between vertices are called virtual links. All virtual particle links are computed on-the-fly. The links are implemented with pointers. The virtual multi-level links to virtual coarse particles are important to multiscale contact detection as the links define the multiscale contact detection scheme (Figure 6.13). All vertices within the grid follow the hierarchical tree-like order which by design provides the coarse cell inheritance links to fine levels similar to a 3D tree structure where parent and children nodes exist.

Collision detection and inheritance between real and virtual particles in a grid traversal is realised as described by Algorithm 10 in `touchVertexFirstTime` and `enterCell` events. Both events summarise the collision detection phase of the whole DEM algorithm.

At `touchVertexFirstTime`, we derive the forces from the contact points that we defined during the previous iteration. We then use those forces to update the new particle positions for all particles associated at that vertex. Furthermore, we inherit virtual particles via pointers to coarser levels once per traversal at this point.

At `enterCell` traversal event, real and virtual particles associated to different vertices are checked for collisions. Every cubic cell in three dimensions require at most 28 vertex-to-vertex pairwise comparisons of real particle sets. The number of vertex-to-vertex comparisons rises when virtual particles of coarser vertices are inherited. For every cell the max number of real-to-virtual vertex comparisons is $RealParticles_n + VirtualParticles_n \cdot RealParticles_n + VirtualParticles_n$.

The neighbouring vertex-to-vertex pair-wise checks are required to inherit and retrieve lists of particles and perform particle-to-particle comparisons. For each vertex of a cell, it is not a necessarily required to inherit all virtual particles from all coarser vertices. Virtual particles are not always associated with the fine vertex closest to their centre. This may become possible when multiple virtual particles represent one real particle exist on a particular mesh level.

Another reason of misalignment of coarse to fine link associations is the possibility that the actual geometric shape and dimensions of the coarser particle disallows a fine vertex overlap. Fine vertices inherit all virtual particles that exist in coarser levels that overlap the fine vertices. Such inheritance realises a partial cell vertex inheritance. A detailed description of this scenario is discussed in sub-Section 6.4.4.

At `touchVertexLastTime` collision detection is performed pairwise between all real particles and virtual particles associated to a fine grid vertex. This step realises solely vertex-based particle-to-particle comparisons. All vertices within the grid that hold links to more than one particle perform collision detection.

Algorithm 10 Multiscale Grid-Based DEM Implementation.

```
1: function traverseGrid(C)
2:     C_old ← C
3:     C ← ∅
4:     while traversal continues do
5:         if touchVertexFirstTime then
6:             for all particles p associated to vertex do
7:                 for all contact points c ∈ C_old associated to p do
8:                     Update f_trans(p) through c
9:                     Update f_rot(p) through c
10:                end for
11:            end for
12:            for all particles p associated to vertex do
13:                Update particle incl. its triangles
14:            end for
15:            Inherit pointer links to virtual particles from coarser levels
16:        end if
17:        if enterCell then
18:            for all 2^d vertices adjacent of the cell do
19:                for all particles p associated to vertex do
20:                    if particle should be associated to different vertex then
21:                        Reassign particle
22:                    end if
23:                end for
24:            end for
25:            for all real and virtual (p_i, p_j) particle pairs of vertex pairs i, j do
26:                C ← C∪ findCollisions(p_i, p_j)
27:            end for
28:        end if
29:        if touchVertexLastTime then
30:            for all real and virtual particle pairs (p_i, p_j) at vertex do
31:                C ← C∪ findCollisions(p_i, p_j)
32:            end for
33:        end if
34:    end while
35: end function
36: function main(T)
37:    C ← ∅
38:    t ← 0
39:    for t < T do
40:        traverseGrid(C)
41:        t ← t + Δt
42:    end for
43: end function
```

6.4.3 Geometry Decomposition

A major overhead to the grid management arises when there is large variation in particle scales. When variations in particle scales are extreme, the number of virtual links becomes high. One solution is to decompose massive geometries into smaller particles so that the real-to-virtual link distance is minimised. For example, a hopper can be read either as one whole body or as individual composites that make up the original geometry of the hopper structure. We propose two variants of rigid body decomposition.

The first variant decomposes a coarse particle based on a cubic oct-section (Figure 6.14). This is a recursive uniform subdivision that decomposes portions of original particle mesh into smaller particles. Every sub-particle is the sum of mesh elements contained within the subdivision cube boundary. As it is a recursive method, the minimum sub-particle length is specified a-priori to produce sub-divisions of specific length.

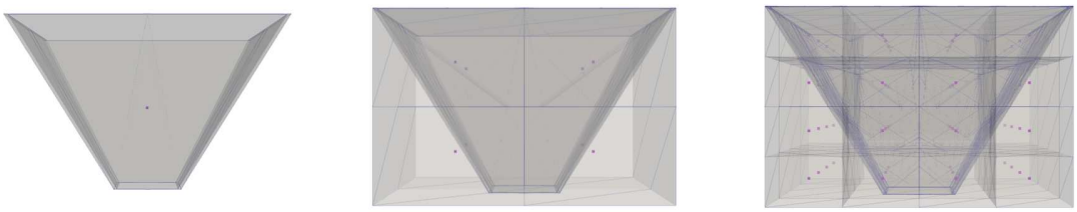


Figure 6.14: A particle is decomposed into smaller particles by recursive oct-sections. Each sub-particle holds a group of sub-elements of the original particle. Left: original structure, Middle: subdivision boundaries, Right: finer subdivision.

The alternative decomposition is based on triangle-based decomposition (Figure 6.15). A particle mesh is refined to fit certain scale ratios of the simulation and then each component is decomposed into smaller particles. The smaller particles copy the physical properties of the parent particle and forms virtual extensions of the original particle composition.

In both decomposition schemes, inseparability is implicitly ensured. All contact points of a decomposed particle are reduced at the end of the iteration in a hash table. Since we follow the idea of pipelining, we derive the total forces for every sub-particle at the next time step and perform the position update. Thus position updates are global to the whole particle while contact detection and contact point generation are local to each sub-particle.

The realisation of a particle decomposition scheme does not necessarily reduce computational cost. The finer virtual sub-particles reduce the multiscale overhead

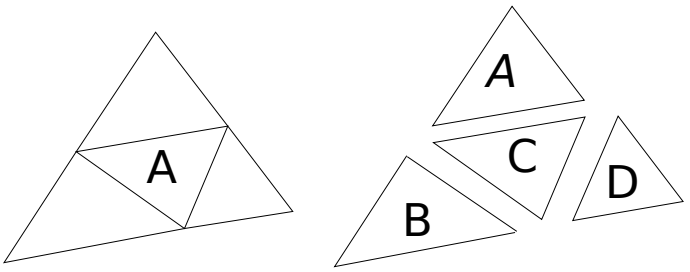


Figure 6.15: A particle can be decomposed into smaller particles or elements that are automatically dropped to finer levels like any other particle.

as the particle components of the original coarse particle is dropped into the fine level. The advantage of the scheme is a reduction of particle-to-particle comparisons in extreme scenarios when sizes between particles vary significantly. The drawback of these approaches is that memory footprint and particle count increase.

6.4.4 Multiscale Filter Mechanisms

The realisation of a multiscale grid in DEM introduces redundant pointers by design. The root of the problem is the pointer inheritance that is associated vertices between hierarchy levels. We categorise link redundancies into vertex-based and cell-based.

Vertex-based redundancies. On `touchVertexFirstTime` event we inherit particles from all coarser vertices using pointers. At this stage there are two types of inheritance. The inheritance of all real particles of coarser vertices and the inheritance of all virtual particles of the coarser vertices. The fundamental property of vertex inheritance is that all vertices of a coarser cell are associated with all vertices of the finer cell. This creates duplicate particle pointers.

To alleviate redundancies in vertex pointers (Figure 6.16), we filter out the duplicates of coarser vertices after each vertex inheritance phase. Multiscale vertex-to-vertex links are minimised by utilising particle size information. Virtual particle inheritance occurs if and only if the radius of a coarse particle residing at a coarser vertex overlaps the vertex of a finer vertex. A radius of a virtual particle is defined as

$$R = r + 2 \cdot \epsilon + fineGrid_H \cdot \sqrt{dimensions}$$

where r is the radius of the coarse particle, ϵ is the DEM halo margin, $fineGrid_H$ is the H length of the fine cell side, $dimensions$ is the dimensions of the cell. The square root portion of the equation defines the diagonal of the fine cell, thus R covers an the area around the particle by one fine cell. If d is the distance between the coarse particle centre and the fine vertex then if $d \leq R$ then the fine vertex overlaps

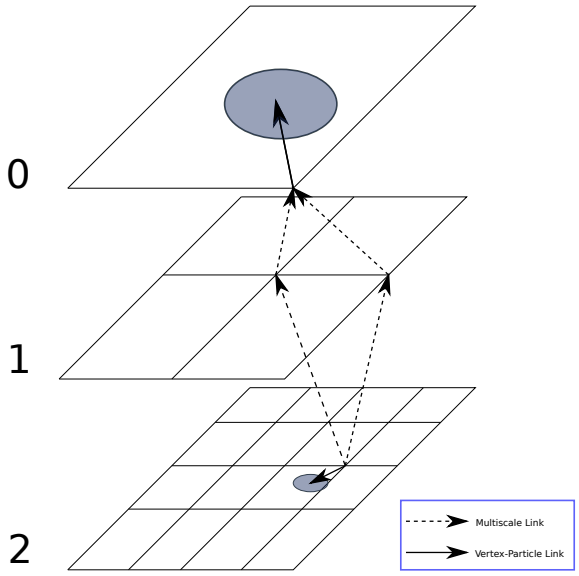


Figure 6.16: Every vertex residing on a finer level inherits all virtual particles residing on the coarser levels. Particle inheritance duplicates on a per vertex basis.

the virtual particle radius and thus it is inherited.

Cell-based redundancies. On `enterCell` we check every vertex with its neighbours to determine collision between particles. At this stage, all vertices of a cell have already inherited unique links to coarser virtual particles. These virtual particles are unique per fine vertex but not unique to the cell as shown on Figure 6.17. Therefore, it is important to remove duplicate links to particles at a cell-basis. All virtual particles per vertex are reduced to unique sets of virtual particles and assigned to a local virtual vertex that is then compared against all vertices of the cell to determine real to virtual particle contacts.

Filled cells, empty cells and ghost cells. Cells can be categorised into three types depending on what type of particles they host (Figure 6.18). Filled cells host real and virtual particles and empty cells host neither real nor virtual particles. Ghost cells are cells that support cells at finer levels by providing the virtual links to coarse particles.

Ghost cells exist to support the hierarchical order between coarse and fine levels. When the grid is traversed these ghost cells are also traversed. As these particular cells do not hold any real particles, nothing is to be calculated on them. There is no need to invoke any computation between regions that hold virtual particles. We reserve DEM computational phases for cells that hold at least one real particle.

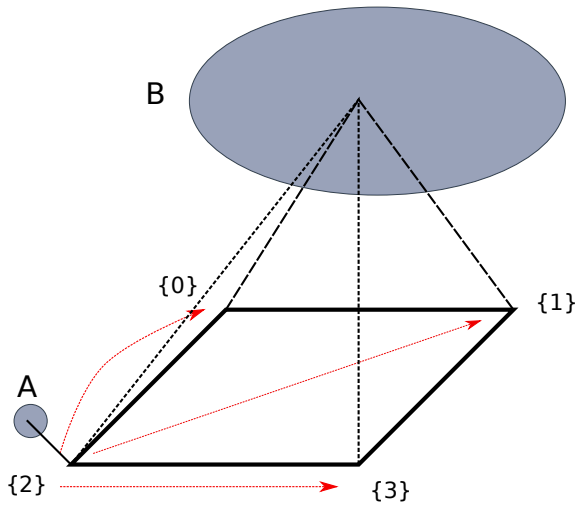


Figure 6.17: Every vertex residing on a finer level inherits all virtual particles residing on the coarser levels. Particle inheritance creates collision detection redundancies on a per a cell basis.

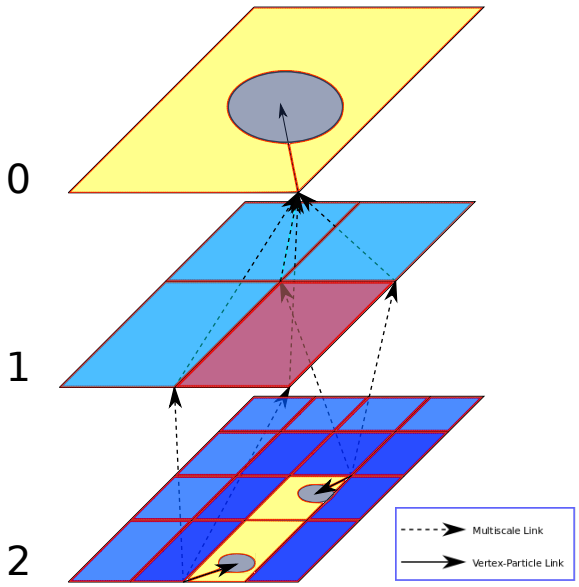


Figure 6.18: In terms of collision detection and inheritance, there are four types of cells: a. yellow cells where real particles reside, b. light blue cells that are empty cells with neither real nor virtual particles, c. dark blue cells that are partially linked to virtual particles, d. pink cells store no particles but exist between cells that store real particles at different levels.

CHAPTER 7

A Dynamic Time Step Scheme

Introduction. To realise global adaptive time stepping for particle collisions, we exploit both the grid properties and the particle dynamics to adjust the time step size. Step adjustment allows us to speed up the simulation's time-to-solution. When combined with Adaptive Mesh Refinement (AMR) schemes, a dynamic step exploits the morphology of the grid which yields a cut-off radius for the neighbour collision check and dictates the maximum step size. The study of the interplay between the adaptivity criteria with an admissible time step sizes are to the best of our knowledge not available. Reluctant adaptivity provides the opportunity to adjust particle time step constraints but reduce the grid overhead per traversal. Given the one-cell-per-time-step constraint on the particle velocity, we restrict the maximum step. While we preserve a global step size, we coarsen the grid when possible which allow us to facilitate larger time step sizes.

In this adaptive space and time scheme, when two particles that are within close proximity we trigger grid refinement and immediately reduce time step Δt. When a pair of particles are within a critical close distance we explicitly restrict Δt such that there is no interpenetration. Similarly, during collision separation our grid/step flags allow the incremental increase of Δt up to a set Δt_{max} step.

Chapter outlook. In this chapter we discuss the concept of a global dynamic time step scheme that derives information from two relationships. We therefore split the discussion into Particle-to-Grid Relationship and Particle-to-Particle Relationship sections. In Particle-to-Grid Relationship, we discuss the interplay between the grid and a particle that can be exploited to determine a suitable step size locally.

The local grid morphology varies over time based on particle movements and collision states. In section Particle-to-Particle Relationship, we discuss the information exchange that define three collision states within a pairwise interaction. We define the "non-critical approach", "critical approach" and "separation" states. Each state is triggered based upon pairwise relative velocity direction. Finally in section Particles that Reside on Different Grid Levels, we show a working example of a two-particle-collision of different scale.

We design a dynamic mesh refinement criterion that fuses the traditional AMR advantages (reduction of computational cost) with the maximisation of the admissible time step size. Such scheme allows us to utilise large step sizes while minimising the total number of steps required to simulate particle interactions.

7.1 Particle-to-Grid Relationship

Within the grid, every particle is subject to spatial and temporal restrictions to preserve numerical stability and reduce computational complexity while at the same time we want to utilise large time steps. The principal rule is that particles are not allowed to tunnel through neighbouring cells to safeguard against contact misses. Every particle updates its position given an initial step size Δt. Our simulation is given an arbitrary set Δt_{min} while a maximum step size Δt_{max} is determined by the global maximum velocity and the minimum cell side length.

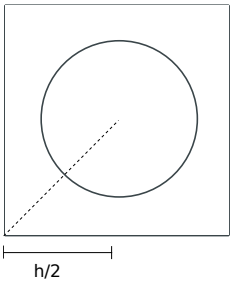


Figure 7.1: A particle residing within a cell is associated with a vertex and is not allowed to update its position more than the length of its cell.

We define the particle-to-grid relationship based upon three points:

i. In an adaptive grid, a cell side cannot be smaller than the particle diameter that it holds.

ii. Particles are hosted within cells as long as their radius fit, we do not force particles to reside at the finest level, instead particles are allowed to travel

within coarser levels.

iii. A particle may not update its position along step size Δt further than half its cells' length.

It is a precondition that no particle resides in a cell with a side length smaller than the particle itself. A cell may hold more than one particle at a given time. Particles are dropped to finer cells as long as their diameter permits. Particles are not always pushed to a finer level, with a reluctant grid we allow particles to reside on coarse levels as long as they are not in a collision course with another particle. The grid refines and drops particles to finer levels only when there are two particles that approach each other. Such constraint allows us keep the grid as coarse as possible. Large time steps are enabled when particles reside on a coarser cell relative to their diameter length. The step size is incremented by $\Delta t = \Delta t \cdot 1.1$ up until the maximum $\Delta t_{max} = h_{max}/(2.0 \cdot 1.1 \cdot V_{max})$ where h_{max} is the maximum cell length in the grid and V_{max} is the maximum current velocity among all particles.

7.2 Particle-to-Particle Relationship

The dynamic step scheme adheres to the collision detection phase of particles. Increments and decrements depend on three collision states. These three configuration states "non-critical approach", "critical approach" and "separation" of a local particle pair determines the global variation in step size when collisions occur. Collision states are defined by the sign of the relative velocity direction of a particle pair. For an 'non-critical approach' state two particles are moving towards each other but are sufficiently away to not trigger 'critical approach' (Figure 7.2). A critical approach is activated when during the following iteration, the virtual halo of the particles might overlap. If two particles are within the critical approach state then Δt is explicitly restricted such that there is no interpenetration and undefined behaviour. Finally the separation state is the flag that safely increments Δt up to the Δt_{max} step.

When there is no contact detection between a pair of particles then we define no collision state. The step size increases over time by our constant factor $\times 1.1$ but constrained by the reluctant grid. All dynamically defined Δt are smaller than a max step size

$$\Delta t_{max} = h_{min}/(2.0 \cdot inc \cdot V_{max})$$

where inc is the increment factor 1.1 and h_{min} is the minimum cell length. The global step size is kept constant at Δt_{max} and is decreased only and only if at least

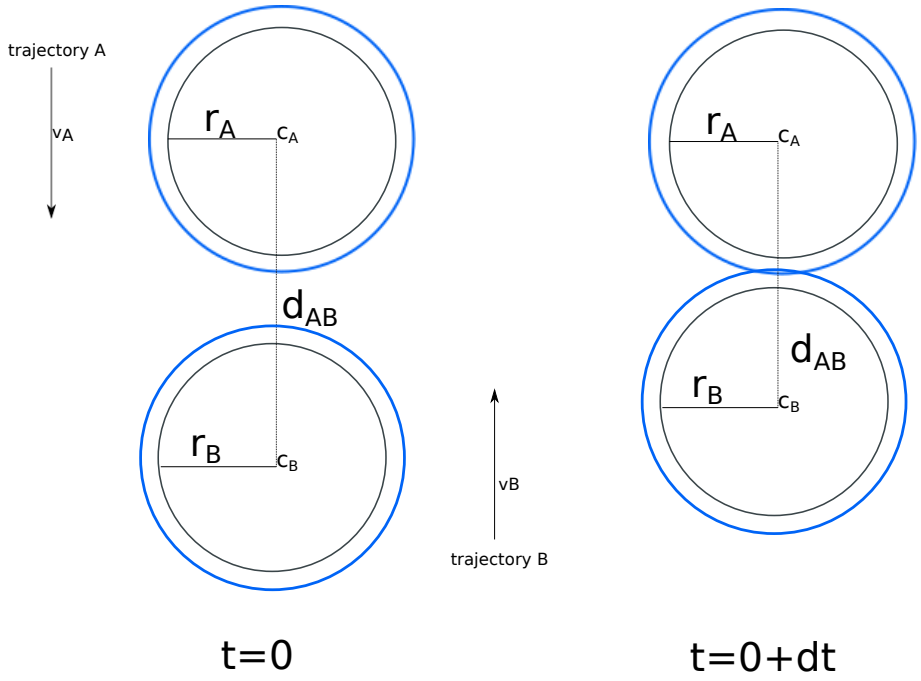


Figure 7.2: Two particles A and B are on approach when their velocities point towards each other and relative velocity is negative. When two halos overlap then there within critical approach, otherwise approach is non-critical.

one particle pair trigger an "approach" collision state.

Particles in critical and non-critical approaches. The non-critical approach proceeds the critical collision state and step increments are continued up until a critical approach is triggered (Figure 7.2). During all non-critical approaches, we preserve the global maximum collision velocity Δt_{max}. The maximum collision velocity together with cell size sets the global upper limit for step size which preserves global robustness of pair-wise position updates. A critical approach triggers a "particles are too close" condition which explicitly dictates the maximum step that doesn't trigger real interpenetration. Bounding spheres of particles are sufficient to trigger collision states and implement the dynamic time stepping. Step computation is performed during the halo sphere-to-sphere pre-check. This is an explicit filter check before the expensive triangle-to-triangle collision detection.

We determine approach states a priori to contact detection by using the following quantities:

i. Vector d_{AB} is the line segment between the centres of mass of two particles A and B.

ii. Distance $\|d_{AB}\|$ is the scalar distance between the two particles from the centres of mass.

iii. The relative velocities $v_{AB} = (v_B \cdot d_{AB}) - (v_A \cdot d_{AB})$ define the approach velocities.

Based on the v_{AB} sign we determine whether the particles move towards an approach or a separation trajectory.

For two particles A and B, the halo radius is r_A, r_B, the centre of mass is c_A and c_B respectively. The halo radii r_A and r_B is defined as $1.1 \cdot e_{A|B}$ where $e_{A|B}$ is the epsilon margin. The distance vector from the two centres of mass points is $d_{AB} = c_B - c_A$, while the scalar distance d is $\|d_{AB}\|$. The minimum distance between two sphere particles is $d - r_A - r_B$. The velocities of two particles A and B are v_A and v_B respectively. The difference between the two vectors is then projected to the distance line d_{AB} so that the relative velocity is defined as

$$v_{BA} = (v_A \cdot d_{AB}) - (v_B \cdot d_{AB}).$$

The relative velocity v_{BA} sign indicates the collision direction. When positive, v_{BA} indicates separation, particle pairs are ignored for the time stepping. If the direction is negative then the particles are moving towards each either in non-critical or critical approach states.
During a non-critical approach, it is crucial to trigger the critical condition to pause step increments. In order to identify critical approach, we predict halo overlap at a future step. We use the predicted relative velocities pv_{BA}, predicted distances of particle pairs pd_{BA} and predicted step sizes $pdt = \Delta t + \Delta t \cdot inc$ to safely extrapolate particle movement along the velocities direction.

A critical approach is triggered when particle are virtually updated by pdt and their halo's overlap (Figure 7.2 right). In critical approach, the local step size is set to

$$\Delta t_{AB} = (\|d_{AB}\| - rr_A - rr_B - e_A - e_B)/2 \cdot v_{BA}$$

where rr_A, rr_B are the real geometry boundary radii. The global minimum of all pair-wise Δt is then used as the global time step size.
When pair-wise velocities point away from each other, v_{BA} is positive and the pair is regarded in a separation phase. Particles in separation are safely ignored.

The algorithm of dynamic time stepping is divided into critical approach and the rest time stepping types (Algorithm 11). At the start of the simulation we initialise Δt with a small number. When two particles meet the precondition of comparison (i.e. adjacent positions in the grid), we determine their approach state. To capture a critical approach state (Algorithm 11), we predict particle movement using the current constant step increment. The current position is extrapolated by pdt (Algorithm 11). We derive the predicted relative velocity pv_{BA}, predicted

Algorithm 11 Multiscale Grid-Based DEM Dynamic Time Step Algorithm.

1: Δt = initial step size
2: $inc \leftarrow 1.1$
3: $\Delta t_{min} \leftarrow 1E-4$
4: **for** $i \leftarrow 0 \ldots numberOfsteps$ **do**
5: **if** contact detection phase entered between particle A and B **then**
6: **if** $v_{BA} \geq 0.0$ - indicates separation **then**
7: Trigger Separation
8: **end if**
9: **if** $-v_{BA} \geq 0 v_{max}$ - indicates non-critical approach **then**
10: Set v_{max} to max($-v_{AB}$)
11: $\Delta t_{max} \leftarrow h_{min}/(2.0 \cdot inc \cdot V_{max})$
12: **end if**
13: $pdt \leftarrow \Delta t + \Delta t \cdot pdt$
14: Predict next particle position using predicted step size pdt.
15: $pd_{min} \leftarrow (pd_{BA} - r_A - r_B)$
16: $pdPerStep \leftarrow pd_{min}/pdt$
17: **if** $-pv_{BA} \geq (pdPerStep)$ - trigger critical approach **then**
18: $\Delta t_{localMax} \leftarrow (d - rr_A - rr_B - e_A - e_B)/2 \cdot v_{BA}$
19: **if** $\Delta t_{localMax} < 0.0$ **then**
20: $\Delta t_{localMax} \leftarrow -1 \cdot \Delta t_{localMax}$;
21: **end if**
22: Trigger Critical Approach
23: **end if**
24: **end if**
25: Update current particle position by Δt.
26: Enter Update Δt.
27: **end for**

distance $\|pd_{BA}\|$, and predicted minimum distance $pd_{min} = pd_{BA} - r_A - r_B$ where $r_{A|B}$ are the prescribed halo radii. If the relative velocity is greater or equal to the predicted distance per predicted step size then the particle critical approach is triggered. As soon as the critical approach state is triggered, our step size is shortened to $\Delta t_{localMax}$ the remaining distance is divided by the relative velocities. If all relative velocities in the domain indicate separations then we increment the global step by Δt.

Our dynamic evaluation of Δt is only set at the end of the simulation (Algorithm 12. First we check if Δt has fallen below our constant minimum Δt else we decrement Δt to the critical $\Delta t_{localMax}$ we've logged during the simulation. Otherwise, if the global state is non-critical approach or in separation, we increment Δt by inc constant factor. If Δt exceeds Δt_{max} then we reset it back to Δt_{max}.

Algorithm 12 Dt Variance & Bookkeeping Pseudocode.

```
1: Δt_min ← 1E − 4
2: if Critical Approach then
3:     if Δt ≥ Δt_min then
4:         Δt ← Δt_localMax
5:     end if
6: else
7:     Δt ← Δt + Δt · inc;
8: end if
9: if Δt >Δt_max then
10:    Δt ← Δt_max
11: end if
```

The presented adaptive time step scheme increases the step size when there are no collisions and decreases the step size when particles are approaching. This reduces the total number of iterations required to simulate an interaction.

7.3 Particles that Reside on Different Grid Levels

We begin with an experiment for the subsequent observations. We take two particles and start the behavioural study of the grid and step size. For this experiment the two particles are placed at $(0.2\ 0.2\ 0.2)^T$ and $(0.8\ 0.8\ 0.8)^T$ over the unit cube domain. The velocities are set to $(0.1\ 0.1\ 0.1)^T$ and $(-0.1\ -0.1\ -0.1)^T$. Trajectories are set towards collision. The two particles have different sizes, i.e. a diameter of 0.02 vs. a diameter of 0.2.

The experiment compares runtimes between the three grid types. For the regular grid since two particles vary in size they reside on different levels. All particle-to-particle comparisons result from virtual links. Once the two particles are close, a large spike in comparisons is seen (Figure 7.3, top). There are more comparisons as the bigger particle induces multiple virtual particles associations towards more than one fine grid vertex, we thus end up with up to $2^d = 8$ comparisons. During particle comparisons, we check to determine whether the two particles approach or move away from each other. These comparisons are persistent throughout the duration of the collision. We thus observe a small plateau of comparisons before and after collision.

The adaptive grid yields a similar spike of particle comparisons which is preceded by a fewer comparisons as the grid refines towards the collision area. The comparisons do not coincide with the actual collision around $t \approx 2.5$, this is a result of aggressive adaptivity and an implementation property which makes support/hang-

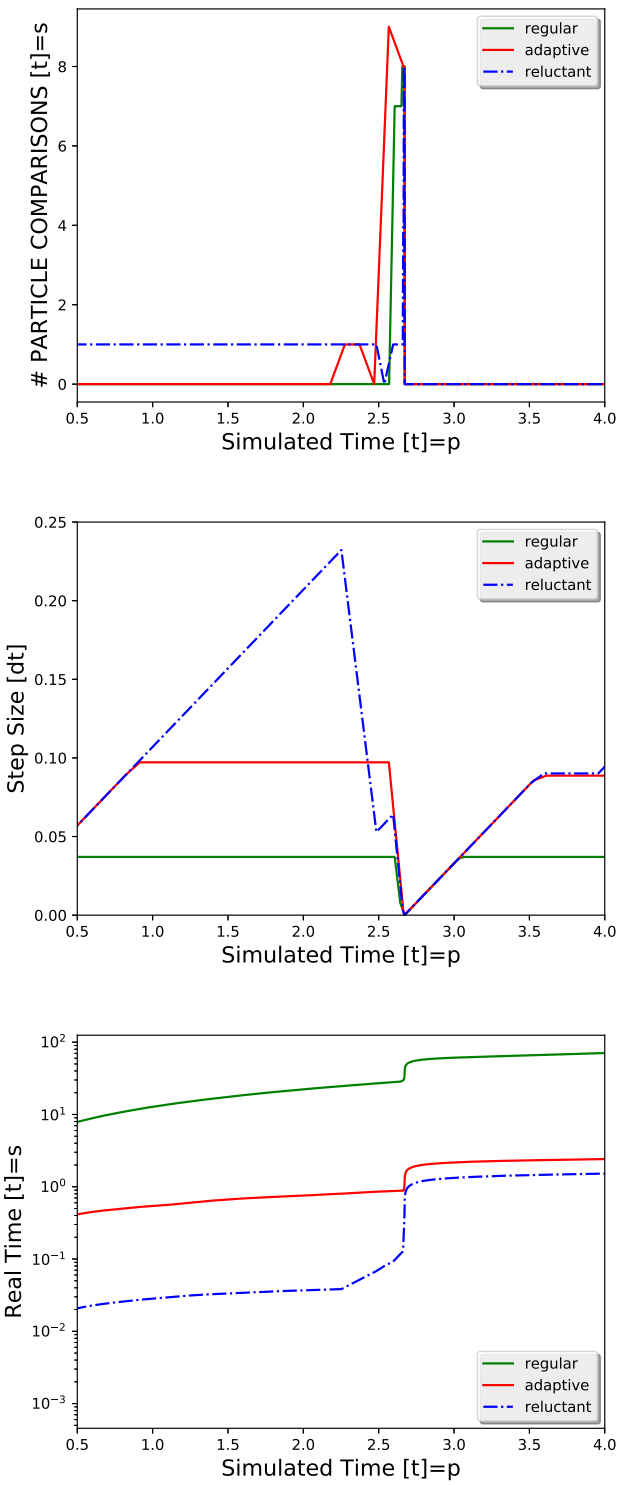


Figure 7.3: Zoom into collision behaviour for two particles and different grid types. We present number of collisions over simulated time (top). The time step sizes creeps towards the maximum time step size and then goes close to zero when the two fast particles collide (middle). Regular grids yield the same time step size pattern as the adaptive mesh. The compute time on a single core is given for all three grid choices (bottom).

ing vertices always inherit all parent particle data [95]. The reluctant grid yields a slightly higher comparison count from the start as it utilises a coarse grid and triggers comparisons only to find out that it has to refine. These additional comparisons are cheap as they only indicate a sphere-to-sphere pre-check comparison which indicate no overlap.

All setups start with a small time step size (Figure 7.3, middle). The regular grid variant reaches the constraint imposed by the finest grid so there is no increment of its step size. Adaptive and reluctant grids slightly increase the initial step size during the first few steps. The reluctant grid starts from a way coarser mesh width thus its time step size constraint it more relaxed than the adaptive variant. All approaches reduce the time step size once the approach collision state is triggered, the time step size is then relaxed again after collision.

In terms of computational runtime, we see that our adaptive time step scheme pays off (Figure 7.3, bottom). All adaptive grid variants exhibit a faster runtime than the regular grid. The reluctant grid variant in particular performs the best as it employs larger time step sizes. Although the reluctant has an increased number of comparisons it makes up in terms of time. When a collision is detected the reluctant grid shifts the performance curve. However, the reluctant variant when compared to the adaptive grid only makes a difference in performance if particles are spread away from each other i.e. they are not grouped in clusters. While for sparse particle configurations, we assume the performance improvement is significant.

The choice of grid dynamics plays an important role in DEM. The size of the cell that particles are allowed to reside in is directly related the maximum admissible time step size. We propose a reluctant grid that benefit from non-aggressive adaptivity and allow us to set a larger time step sizes that reduce the total simulation runtime.

CHAPTER 8

Manycore Concurrency

Introduction. Given the dominance of spatial decomposition schemes in the realisation of MPI parallelisation (distributed memory) in DEM, it is well-understood in literature [12, 29, 44, 45, 49, 71, 72, 75, 77, 99, 101]. Roadmaps [18] predict that the gain of simulation performance in future supercomputers will be derived from the increasing number of shared memory cores. However, current literature on shared memory parallelisation in the context of DEM is rare. Shared memory is where we make a contribution, though this impacts distributed memory parallelisation designs. Papers found in HPC DEM particle-based simulations literature [72, 88, 99] point out that distributed memory parallelisation is not the big challenge anymore, instead methods are required to minimise or overlap the communication [9] exchanged between node ranks. Thus to aid this, our adaptive grid variants motivates this purpose, we exclude as early as possible in the simulation all redundant particle collision checks. In addition, our DEM-grid traversal data movement is minimised through the single-touch algorithm where each particle is only read once per time step. This implies that particle data has to be exchanged only once in a distributed memory DEM scheme. We propose a shared memory strategy that could exist in existing DEM distributed memory algorithmic models.

Algorithm 13 Parallel Multiscale Grid-Based DEM Implementation.

```
1: function traverseGrid(C)
2:     C_old ← C
3:     C ← ∅
4:     while traversal continues do
5:         if touchVertexFirstTime then                    ▷ Grid-based Concurrency
6:             for all particles p associated to vertex do
7:                 for all contact points c ∈ C_old associated to p do
8:                     Update f_trans(p) through c
9:                     Update f_rot(p) through c
10:                end for
11:            end for
12:            for all particles p associated to vertex do
13:                Update particle incl. its triangles
14:            end for
15:            Inherit pointer links to virtual particles from coarser levels
16:        end if
17:        if enterCell then                               ▷ Grid-based Concurrency
18:            for all 2^d vertices adjacent to cell do
19:                for all particles p associated to vertex do
20:                    if particle should be associated to a different vertex then
21:                        Reassign particle
22:                    end if
23:                end for
24:            end for
25:        end if
26:        if LeaveCell then                               ▷ Grid-based Concurrency
27:            for all 2^d vertices adjacent to cell do    ▷ Particle-based Concurrency
28:                for all real, virtual (p_i, p_j) particle pairs of vertex pairs i, j do
29:                    C ← C∪ findCollisions(p_i, p_j)     ▷ Triangle-based Conc.
30:                end for
31:            end for
32:        end if
33:        if touchVertexLastTime then                     ▷ Grid-based Concurrency
34:            for all real, virtual particles (p_i, p_j) do   ▷ Particle Concurrency
35:                C ← C∪ findCollisions(p_i, p_j)         ▷ Triangle-based Concurrency
36:            end for
37:        end if
38:    end while
39: end function
40: function main(T)
41:    C ← ∅
42:    t ← 0
43:    for t < T do
44:        traverseGrid(C)
45:        t ← t + Δt
46:    end for
47: end function
```

The DEM algorithm mapped to a grid traversal presents the potential for a three-fold cascade of shared memory parallelisation. At the finest level of concurrency we exploit the triangle-based tessellation between each meshed particle pairs. Within each grid-cell's vertex, a particle-to-particle collision detection phase realises the local all-to-all $O(n^2)$ sweep on an intra and inter-vertex basis. At the coarsest level, sections of the grid are traversed and processed independently in parallel on a set of pre-allocated threads. We finally combine all these layers into a unified parallel task-based DEM scheme.

The presented algorithm realises three layers of multicore parallelisation:

i. Triangle-to-triangle concurrency.

ii. Particle-to-particle concurrency.

iii. Grid traversal concurrency.

We present a combination of manycore setups using the hybrid solver and a variation of the grid-based DEM algorithm from Chapter 6. As shown in Algorithm 13 each grid traversal event (`enterCell`, `leaveCell`, `touchVertexFirstTime`, `touchVertexLastTime`) is run concurrently and forms our *grid-based parallelisation* layer. Enclosed within these events we have the *particle-based* and the *triangle-based parallelisation* layers. Each distinct layer of computation does not affect other concurrency levels as they are completely independent. In this Chapter, we discuss each individual parallelisation layer as well as we propose a intermixed scheme where all layers combined into a task-based parallel algorithm.

We base our performance experiments on the simulation of particles flowing through a hopper structure. We represent each rock granular particle with 10 , 20 and 40 triangles. The geometries are created using a Delaunay triangulation [16] based upon a random point cloud (see Appendix for particle construction). Mesh density and thus computational grain density is controlled by the variation in cloud points employed during the creation of the geometry. We scale the contact problem in terms of number of non-spherical particles and number of triangles over number of cores.

Chapter outline. This Chapter is organised based upon our proposed levels of concurrency. We start off the discussion with the innermost triangle-based, the particle-based parallelisation and the grid-based parallel traversal. Then we benchmark the performance of the proposed methods on the multicore system. At the end of the Chapter, we finish the discussion with the impact of grain size to the performance and the limitations of each method.

8.1 Triangle Mesh-Based Parallelisation

At the the innermost loop when a pair of particles p_i and p_j approach each other and are reasonably close we execute `FindCollisions()`. A $\mathcal{O}(\mathbb{T}^2_{max})$ triangle comparison is performed in Algorithm 13, lines 27, 33. The invoked call is executed at vertices where there are particle associations throughout the spacetree traversal.

These triangle comparisons can be parallelised via a plain `parallel for` in OpenMP [14]. Let $|\mathbb{T}_i|$ and $|\mathbb{T}_j|$ be the number of triangles of two particles p_i and p_j. The collision detection between p_i and p_j has a concurrency level of $|\mathbb{T}_i| \cdot |\mathbb{T}_j|$. It is convenient to check the first triangle of particle p_i against all triangles from p_j while we run concurrently the checks for the second triangle of p_i. We refer to this intra-triangle concurrency as *triangle-based parallelism*.

When two particles' boundary spheres augmented by ϵ overlap, then the triangles are closer than $2 \cdot \epsilon$. This yields a list of contact points. Every contact point per particle pair per traversal is inserted into a collision set which is protected by a thread lock. For this standalone phase, the concurrency level is determined by the number of triangles per particle pair and the number of contact points.

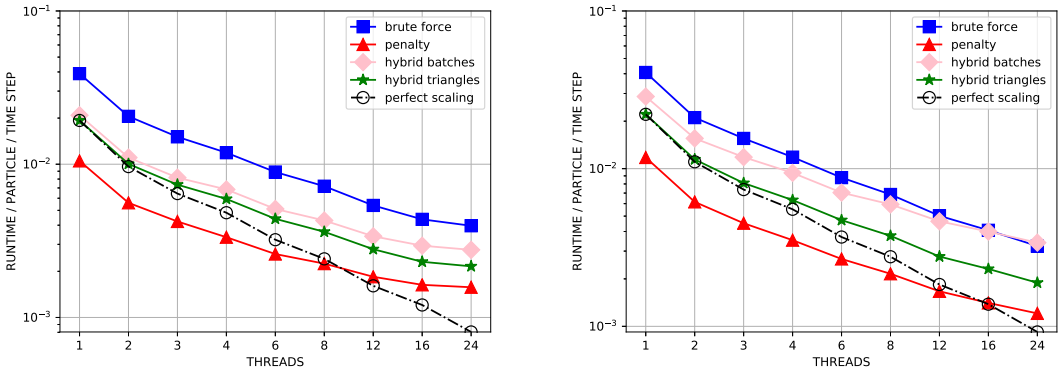


Figure 8.1: Scaling of the various methods. Particles with 20 (left) or 40 (right) triangles each.

In Figure 8.1 we compare our contact methods runtime over a two-socket Broadwell chip. As expected the penalty-based method is leading as in the serial version. The performance of all methods is acceptable on low triangle counts (Figure 8.1, left) on at least the first six core of one socket. The performance gets better for higher triangle counts (Figure 8.1, right) as higher levels of arithmetic intensity exist there. The hybrid on triangles variant performs better than the hybrid on batches.

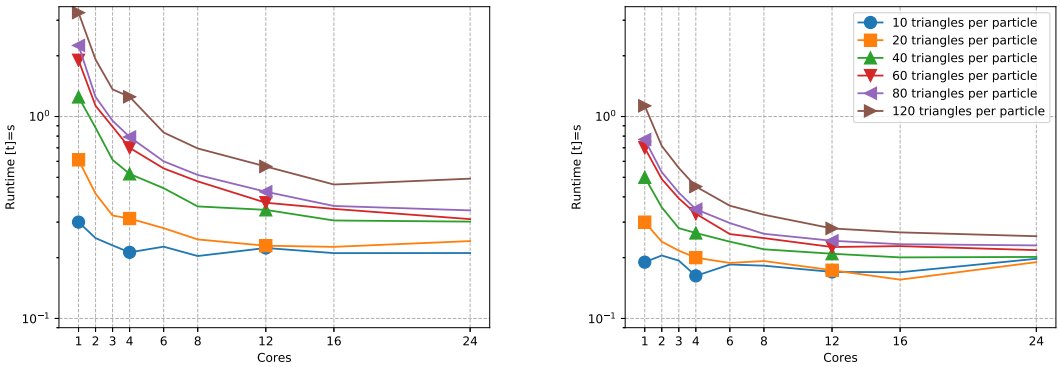


Figure 8.2: Collision behaviour of two particles that are discretised with different triangle counts. Left: we track the scalability of a non-vectorised code. Right: scalability graph with vectorisation and triangle-based parallelism switched on.

For the vectorisation benchmark we run a two-particle setup with the hybrid contact solver (Figure 8.2). We focus solely on the contact event where there are actual triangle-based operations. There is minimal interference from the remaining DEM algorithmic phases; contact detection takes the majority of the total scalar/vector computation and memory bandwidth. We vary the mesh density over 10, 20, 40 triangles and see a runtime improvement through vectorisation (Figure 8.2) of up to a factor of two, if a sufficient number of triangles exists per particle. Even though we have validated that the compiler does vectorise all triangle-to-triangle comparisons and does align all geometric data structures properly, the fewer triangles the lower the speedup. Larger triangle counts are required but in practice granulates are composed with less than forty triangles. The higher the number of triangles per particle in a geometry setup the better the arithmetic intensity (Flop's/Bytes processed) and eventual exploitation of the whole register width.

The contact detection code benefits from vectorisation (AVX) of modern microprocessors, but when combined with our grid data structures that minimise the redundant computation and a collision pre-check which minimises actual mesh-to-mesh contact detection, we also reduce the available floating point operations. Together, these properties imply that a very limited shared memory up-scaling potential does exist on the triangle level. Scalability has to stem from other approaches. Yet, once two particles are compared, more than one core should be used. Unless we omit vectorisation, it does not make sense to exploit more than four cores solely for contact detection comparisons. Beyond this, strong scaling is not efficient any more. For the end-to-end simulation, four core is even too optimistic as only very few particle pairs actually become subject to triangle-to-triangle comparisons due to the grid discretisation. The triangle-to-triangle concurrency benefits from increased

number of triangles per particle, but this improvement does not translate into the whole simulation as the number of collision detection invocations are not necessarily constant throughout a simulation run.

8.2 Particle-to-Particle Parallelisation

The second layer of concurrency, *particle-based parallelisation*, arises when multiple particles are associated to a single vertex. A single vertex associated to a single particle with no neighbouring vertices that hold particles does not yield a particle-to-particle comparison (Figure 8.3). If a vertex holds more than two particles, the collision check between these particles is parallelised. Neighbouring vertices that hold at least one particle yield local particle comparisons. Adjacent vertices may hold more than two particles in total as they include particles associations located at coarser scale levels.

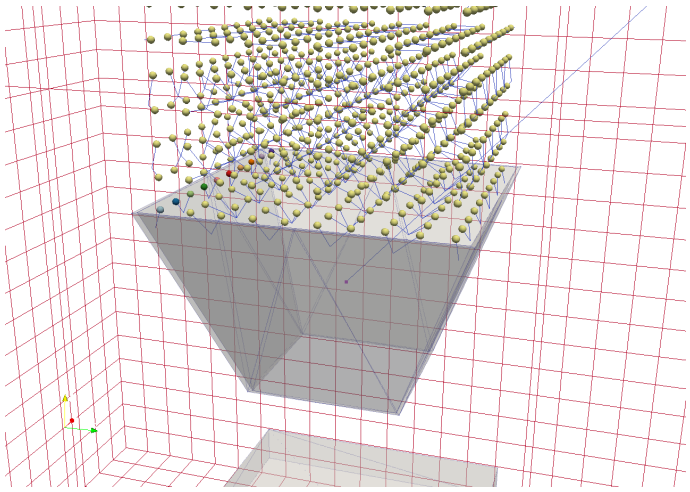


Figure 8.3: The initial particle arrangement of a hopper flow scenario. Each line segment of each particle indicates the association between the particle itself and the grid vertex. Grid vertices may be associated with one or more particles at a time.

Parallelisation at the particle level starts if a single vertex holds or if a pair of neighbouring vertices hold more than one particle. Parallel contact detection at a vertex is triggered when the vertex is touched throughout a grid sweep (see Algorithm 13). When we enter a cell and compare different neighbouring vertices' particles with each other, particle-to-particle comparisons are triggered in parallel, too. The concurrency level is determined by the number of particles associated with a vertex or adjacent vertices.

Concurrency at this level is affected by two factors: grid scheme and simulation scenario. The simulation scenario (initial parameters, geometry arrangement,

scales) sets the degree of particle clustering/disparity in the domain. Then geometric data locality is determined by the employed grid scheme (no-grid, regular, adaptive, reluctant grid). Geometric data of particles that are clustered yield higher particle-to-particle concurrency at the cost of redundant computation. Particle-based parallel computation and grid adaptivity is mutually opposed. The adaptive grid variants minimise the number of particles that are held within a cell. So they are not in favour of particle-to-particle parallel contact detection.

A different strategy is to combine particle and triangle parallelism in a parallel loop nest. Both particle and triangle-based shared memory threads are launched simultaneously when particle comparisons are triggered. In such variant the runtime to solution is slightly slower than particle-based parallelism alone due to the thread overhead proportional to arithmetic intensity in the granulate particle geometry scenario (low triangle count). Nevertheless, it has an impact on brute-force contact detection method as it scales smoother than particle-based only parallelism due to the aforementioned overhead. Since particle comparison counts vary throughout different simulation scenarios, we do not rely on this scheme.

8.3 Grid-Based Producer-Consumer Parallelisation

Poor performance at the particle level calls for a coarser level parallelisation model at the grid level. As the grid owns the particles through particle association pointers and since the traversal triggers particle comparisons, we choose to parallelise the grid traversal itself with (*grid-based parallelism*). The actual particle position updates (Algorithm 13) are performed in parallel during the `touchVertexFirstTime` event. Vertices are visited concurrently so are the updates of velocities and configurations of the corresponding particles. Concurrent contact detection routines are invoked in `ttouchVertexLastTime` and `enterCell` lines (Algorithm 13).

The particle to grid association maintenance exhibits a lower concurrency level due to the on-the-fly particle-to-vertex re-assignments which are driven by position updates. A particle is allowed to move at most within the list of a cell-connected vertices. We realise these moves while the algorithm traverses through the grid in parallel. Particle reassignments modify records at associated vertices. We may not run the particle-grid maintenance on two vertex-connected cells concurrently due to thread safety. Instead, we run through the cells per tree level in a red-black Gauß-Seidel fashion. Multiscale traversal thread safety is ensured with the colouring of every second cell to demarcate multilevel cell inter-dependencies. The colouring along every coordinate axis ensures that particle re-assignments do not induce read-

write race conditions at our vertices. Along with particle position updates, collision checks per vertices and per vertices pairs (Algorithm 13) are executed in parallel, the collision points are safeguarded in a shared memory container.

Similarly, the evaluation of the adaptivity criteria requires additional synchronisation and colouring. Grid coarsening phases rely on data movement restrictions to ensure the consistency of the grid morphology over time. No two children vertices properties (associated particles, refinement control parameters) are lifted into their parent concurrently on cell coarsening as this triggers undefined behaviour. These events are safeguarded with atomic lock operators. Nevertheless, these meta data operations are negligible in terms of computational cost. During grid morphology changes, we continue to traverse the grid in serial mode up until the grid geometry becomes stationary. Although we run in parallel the lift and drops of particles through the grid levels, the updates of virtual particles links, the initialisation of data structures, the allocation of memory, many of these operations contain synchronisation constraints through operating system calls. Thus it is convenient to wait until part of the grid geometry becomes stationary and skip the parallel treatment of the affected grid regions by one grid sweep. This results in a pipelined parallel DEM-grid traversal that is thread-safe.

The parallel DEM-grid promises a coarser level of parallelism based upon the grid discretisation but this is often unnecessary when the majority of grid vertices are unoccupied due to refinement or particle clustering. A better solution is to utilise multiple layers of parallelism, this promises better computational granularity as computational work/geometry is often not equality distributed in the domain. For this we rely on a task-based realisation which follows a producer and consumer model. Through Intel's Threading Building Blocks (TBBs) [68, 73] unlike traditional OpenMP-based [14] algorithms we do not assign stationary compute resources to particular algorithmic steps. Instead the parallelisation model produces and consumes jobs as tasks. We base our parallel formulation on the outermost grid traversal routines to launch tasks using the peano-framework [91]. In our tasked-based models, all our proposed shared memory parallelisation layers are combined. Although the three levels are conceptionally different to each other, the computational efficiency of one level might depend on the others. With the layers combined, the shared memory output (i.e. the storage of contact points) has to be protected by global semaphores. The shared memory lock frequency depends on the physical geometry configuration and dynamics at hand. The identification of unique contact points occurs infrequently when compared to the total triangle count memory accesses. As such, our total synchronisation penalty is negligible during a traversal.

The parallel traversal runs through vertex/cell to evaluate all local refinement criteria and identifies collision candidates. At this point the algorithm does not trigger an actual particle-to-particle comparison but instead a task-producer model wraps around each pair-wise collision candidate into tasks. The tasks are then launched, the grid traversal continues immediately. Such an approach relies on task stealing [73] to keep all cores that are not used by the grid traversal busy. A scheduling subsystem consumes the actual particle-to-particle comparison tasks, executes them, and eventually stores the output contact points. Cores on the machine act as task consumers. At the end of a grid traversal we employ one global synchronisation point, the traversing core waits until all of the launched tasks have completed.

The contact detection tasks that are produced during the parallel grid traversal are marked based on execution priority. The traversal itself is set to the highest priority but grid traversal is often computationally empty due to the underlying geometry. Our algorithm has the capacity to intermix the traversal with contact detection tasks to keep the machine busy. The contact detection tasks are launched as lower priority background tasks and are invoked to be executed during the traversal at no particular order. A high number of background tasks per core would indicate task over-subscription and the stacking of tasks at the end of the traversal, whereas an under-subscription would indicate lower task consumption. The number of background tasks launched by the producer at a time can be specified by the user, however we stick to backgrounds tasks that are equal to the number of hardware cores which is the ideal setting for many applications [10].

A producer-consumer task-based parallelisation model allows us to utilise all levels of parallelism. We treat computational work as independent task units that are allowed to be intermixed and consumed by any thread unit. This scheme allows the algorithm to execute both the efficient DEM collision phases but also the traversal itself in parallel following a task stealing paradigm.

8.4 Impact of Grain Size on Resources, Limitations and Side Effects

The parallelisation of the whole DEM pipeline is highly dependent on the geometry and simulation dynamics over time. Because of the dependency in the physical geometry the impact on resource utilisation is significant. Grain size described as an abstract unit of computation plays a vital role in every parallelisation layer in algorithmic design. In the classic realisation, the grain size of triangle-based com-

parisons in pair-wise particle checks define the arithmetic intensity (FLOPS/byte). Whereas on the task-based implementation task grains are consumed intermixed on demand by the system. High arithmetic intensity per thread maximises the utilisation of compute and communication channels and weak scaling over several sockets and cores is straightforward. The remainder of the Section discusses the impact of grain sizes and the limiting factors observed at runtime.

Classical particle-based parallelisation suffers from the idea of multiscale adaptive meshes. The vision behind the spacetree construction is to obtain grid geometries that morph around the hosted particles with the objective to spatially isolate as many particles as possible. As a result, the number of particles per vertex and the number of vertex neighbours that hold particles is limited. Furthermore, it is convenient per cell to build up particle comparison lists prior to triggering any actual comparison (sphere pre-check). Therefore, the impact of particle-based parallelism is limited in dispersed dynamics. On the contrary, packed configurations with limited grid adaptivity enforced show increased effectiveness of parallel computation. In the dense scenario the total number of comparisons divided by the number of cores define our grain size.

Triangle-based parallelism alone is promising for detailed particle meshes where the number of comparisons is high. However it suffers when the high comparison workload is raised infrequently when a grid is in place. A grid reduces the frequency of triangle-based phases as they depend on the total number of particle comparisons. The reduction in the total triangle comparisons minimises the total speedup gain from vectorised contact detection phases. An excellent vectorisation efficiency [51] relies upon the decent triangle count. Furthermore, if a particle decomposition scheme is applied too aggressively then this also affects the concurrency level, as vectors become smaller. In practice, the pay-off between parallelisation and vectorisation is balanced empirically by the granulate mesh density. The parallelisation strategy that is most promising is also of limited availability in reduced particle comparison configurations.

With respect to the granularity, batches of triangle pairs are statically distributed over the threads. Heterogeneity in mesh density over a pair of particles does not affect triangle-based shared memory load balance. Yet, thread imbalance occurs in the particle-based setup as triangle heterogeneity propagates to the particle-based parallelisation granularity. A dynamic load balancing for classic shared memory is then applicable. Such case is not further studied as it is setup dependant and out of our scope.

Naive mesh-based and particle-based parallel codes alone do not necessarily yield

uniform speedup over the total duration of the simulation. In a regular grid setup while space is discretised into equal parts, the corresponding computation within the grid traversal is not. Adaptive grids outperform the regular counterpart in terms of time-to-solution and weak scaling over the available cores is worse. Moreover in Adaptive Mesh Refinement (AMR), despite high concurrency phases that are under colouring constraints, the traversals are interrupted by tiny close-to-serial fractions where grid refinement evaluations kick in at the end (Figure 8.7). The grain size of computational workload equals a work task, we define the grain size statically over the available threads. A machine learning scheme is available by the Peano grid framework [91] that dynamically changes the granularity during runtime. This is switched off in our experiments due to the scheduling overhead and CPU spinning that can occur during performance measurements. A dynamic grain size balance is useful when executed in repeated test runs (machine training approach) which then yield empirical grain size information that is then applied on the same engineering scenario.

In our task-based scheme where all layers are enabled computationally heavy triangle-to-triangle steps take turns with the computationally cheap grid traversal steps. The grid traversal steps tend to be solely bandwidth-bound and do not yield lots of numerical operations per second. For the sustainable performance of the model, we balance the two phases by launching contact detection phases as standalone background tasks. The grain size of these background tasks vary according to the geometry vector lengths. The number of background tasks executed on a given processor is fixed to the allocated number of threads which yield the best performance. The consumption of background tasks is intermixed with high priority traversal task that coexist in the task queue. The grain size of contact detection tasks is subject to automatic static division over the number of available cores. Therefore it is critical to have uniform contact grain distribution over cores, this is ensured in uniform geometries (i.e. no extreme mesh densities in the system). For extremely heterogeneous geometry problems where few individual tasks take magnitudes time longer to execute, the execution is performed in the waiting phase at the end of the traversal. The proposed parallelisation scheme results in a DEM simulation where most cores are busy with actual triangle-to-triangle checks while the grid geometry is continuously traversed through the machine.

8.5 Many-Particle Systems

We conclude our shared memory performance experiment study with the simulation of the hopper flow scenario (Figure 8.4). We keep the hopper setup as simple as possible and focus on the DEM behaviour, the grid and particle geometry. We make use of $1,000$ to $10,000$ particles that are arranged in a lattice layout above a hopper structure. The particles are dropped with the force of gravity into the hopper geometry, they squeeze through it and eventually fall on top of a floor structure on the bottom.

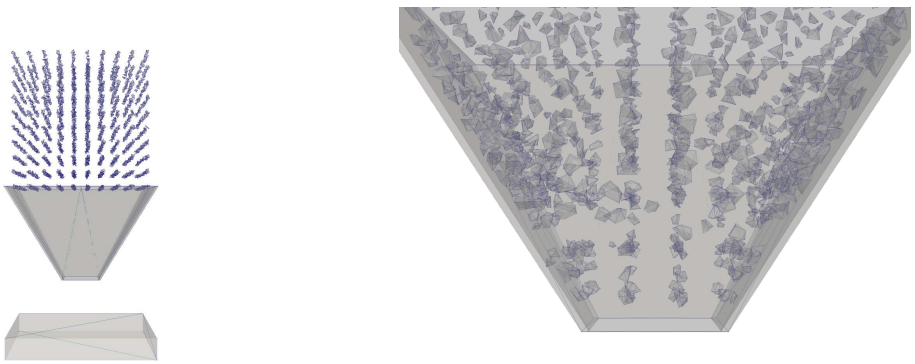


Figure 8.4: Depiction of the one thousand particles hopper flow experiment.

All single node experiments are ran on an Intel Xeon E5-2650 with two times 8 cores running at 2.0 GHz. On this system, we use Likwid [87] to read performance counters. All manycore experiments are done on an 64 core Intel Xeon Phi 5110P KNL (Knights Landing) with 8 GByte of memory running at 1053 GHz in native mode. Parallel experiments are ran on Durham University Hamilton supercomputer where per node there are 2 x Intel Xeon E5-2650 v2 (Ivy Bridge) 8 cores, 2.6 GHz processors, 64 GB DDR3 memory, 1 x TrueScale 4 x QDR single-port InfiniBand interconnect. We use Intel(R) MPI Library for Linux* OS, 64-bit applications, Version 5.0 Update 3 Build 20150128. For performance statements, we rely on the Intel 16 compiler.

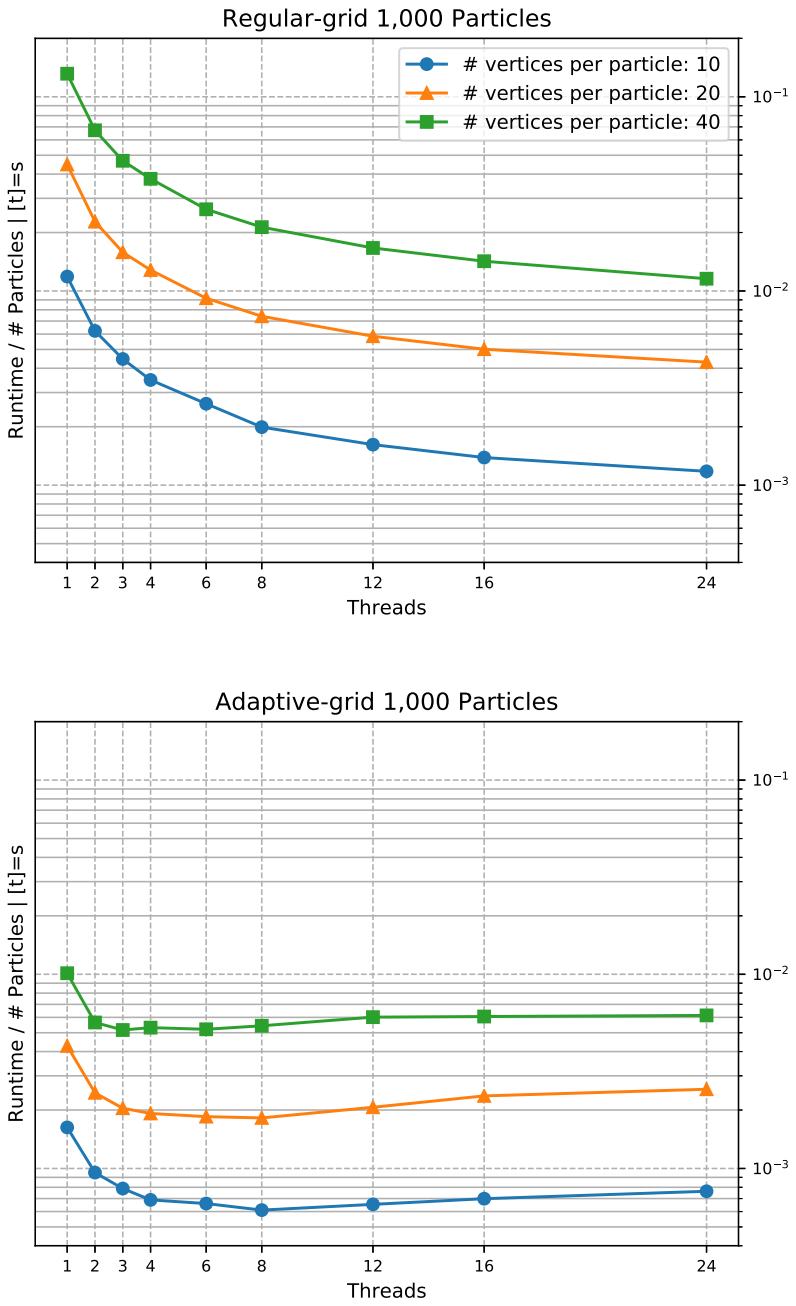


Figure 8.5: Scalability graph with different thread counts, a plain realisation of the grid-based parallelism. We compare the regular (top) grid and adaptive grid types (bottom).

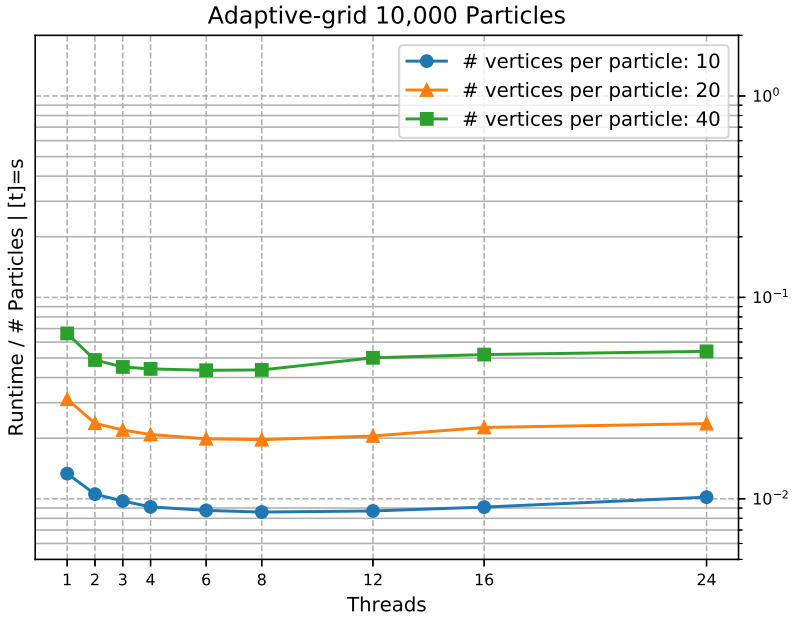


Figure 8.6: Scalability graph of an adaptive grid with high particle count. The higher particle count does not reflect an improvement in shared memory performance in the plain parallelisation scheme.

We start off the experiments with the classic grid-based parallelisation where additional concurrency layers are disabled. This yields mixed results [51]. Experiments with regular grids (Figure 8.5, top) give us timings that show the correlation between triangle count and number of utilised threads; we see good scaling on all triangle counts. However, grid adaptivity exhibits almost no scaling (Figure 8.5, bottom), this is due to the reduction of redundancy. An increase in the number of particles in the experiment in this grid variant does not improve the runtime scalability (Figure 8.6). The reluctant grid does not yield any different results from the adaptive grid, this behaviour is to be expected given the dense packing of all particles in the hopper setup. This regular resource allocation forms our baseline for the proposed task-based parallelisation variant.

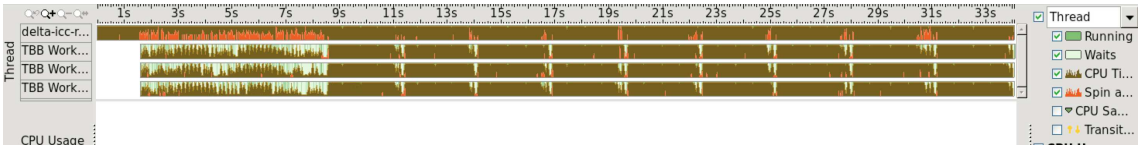


Figure 8.7: Screenshot of Intel's VTune validating that the task producer-consumer pattern pays off after an initialisation phase of 9s where the grid is constructed and the particle shapes are built up.

By focusing on the combined scheme where all parallelisation levels are active, a producer-consumer pattern changes the scalability characteristics (Figure 8.7). A tasked-based scheme deploys the actual collision detection to a series of "worker" tasks which are stolen by idle threads of the processor, while a "main" thread continues the grid traversal. At the end of the traversal, the main thread joins the worker threads to complete all contact comparisons. Figure 8.7 shows measurements from a four thread run; the green intervals indicate the sharp time discrepancy in tasks ending towards the end of the traversal. The traversal ends and a brief serial phase follows. Then we remove duplicate contact points, derive the forces from the collisions, and kick off the subsequent mesh traversal. All worker threads continue to be idle until new collision task is issued.

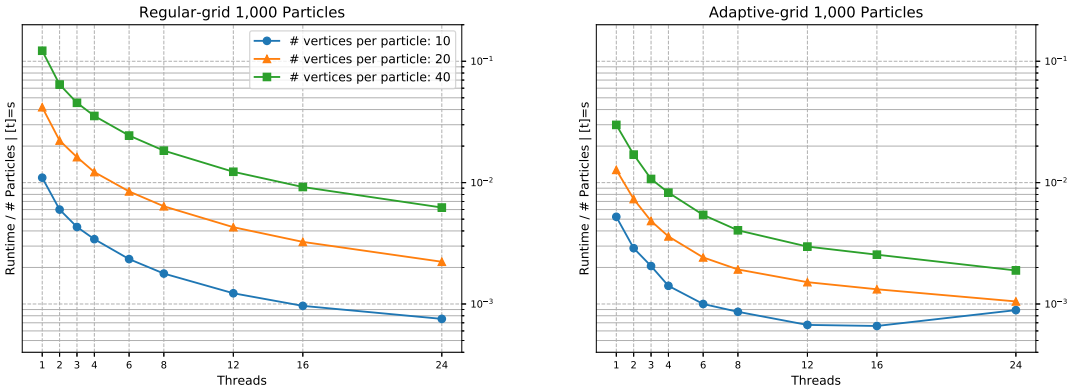


Figure 8.8: Experiments from Figure 8.5 rerun with a task-based consumer-producer realisation of the grid-based parallelism. The both grid types scale better over the assigned number of threads. The adaptive variant shows a significant improvement.

We observe that a task-based realisation of the grid parallelism makes our algorithm scale reasonably well (Figure 8.8). The task-based formalism pays off both in the regular and adaptive grid variants. For the regular grid (Figure 8.8, left), we observe better speedup at runs that utilise more than twelve threads. The runs beyond twelve threads are executed on the second CPU socket, arithmetic intensity per thread is increased with task stealing. The full potential of the producer-consumer scheme becomes apparent when adaptivity is switched on. Although with grid adaptivity the number of particle collisions is reduced, work stealing fills the allocated threads with enough computation to scale over the first CPU socket.

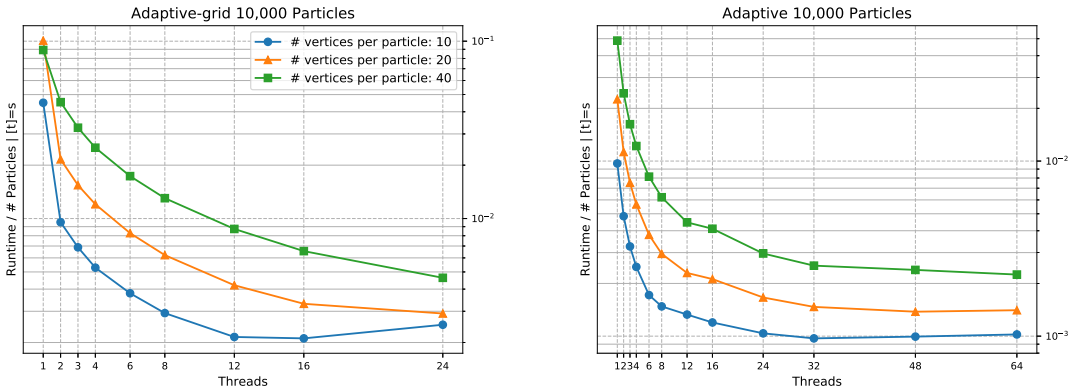


Figure 8.9: Experiments from Figure 8.8 with 10,000 particles on the Broadwell (left) and a KNL - Knights Landing (right).

To stress the task formalism, these experiments process the actual triangle-to-triangle comparisons concurrently [51]. The actual comparisons thus pass through quickly as they are small due to triangle count. Although the actual particle-to-particle comparisons are few in an adaptive grids, weak scaling behaviour remains observable (see Figure 8.9, left). An increase in the number of particles does improve the performance of the adaptive grid. The heavier the workload per particle pair (the higher the triangle count) the better the scaling. We finally run the same experiment on the Intel KNL and we see similar runtimes but with better scaling over 24 threads due to the lower clock frequency. We conclude that the combined three-layer task formalism brings together the advantages of adaptive grids with multi-threading.

CHAPTER 9

Conclusion

In this thesis we've revisited the major steps of a classic DEM algorithm implementations from a HPC point of view and highlighted the importance of mesh-based geometries in a simulation. We contributed to a new set of contact detection methods suitable for SIMD hardware. Furthermore we introduced three grid types that go in par with the dynamics of a DEM simulation. We achieved to map traditional DEM algorithmic phases into a structured multiscale grid traversal. Lastly, we demonstrated several shared memory strategies.

Chapter outlook. In this Chapter we conclude the discussion with the summary of what was achieved in this project. We split the areas of contribution by topic and with respect to the simulation code. We highlight the difficulties encountered over the duration of the research. Finally, we conclude with some open questions and the future outlook.

9.1 Summary

Contact Methods. We described in-depth the motivation behind our focus on the contact detection phase of the simulation. DEM algorithms found in literature report that the majority of the runtime is spent on contact detection phases both in sphere and non-sphere based geometries. It is clear that our challenge is to both increase the physical approximation of real geometry and to minimise the total duration of collision detection phases. The increased levels of geometric representation accuracy promises to allow for more complex and realistic phenomena.

We propose an iterative contact detection method which is designed to exploit SIMD vector lanes found in modern computing hardware. The counter-part of the iterative method is the numerically robust brute force approach which looks up naively all possible distances of each geometric primitive of a triangle pair (line segments, vertices, plane). The drawback of the naive method is the cost through branching. We propose a hybrid version that combines the best of the two worlds. The hybrid algorithm first solves the problem with the iterative solver but then falls back to the brute force if the solution is not within the set ϵ error range.

Multiscale Grid. The complexity of collision detection between a set of particle is $O(n^2)$ as each particle needs to be compared with all other particles. A grid is employed to reduce the overall collision detection complexity. We achieve this by mapping all DEM algorithmic phases (contact detection, particle position update, force derivation, particle creation) into a grid traversal. The grid enforces a single touch policy which allows us to reduce the redundant collision checks. Furthermore, a cascade of grids allow us to accommodate different scales of particles within the hierarchical levels of the Cartesian grid. Coarse and fine levels of the grid tree communicate via parent-child relationships of the grid vertices.

Three types of grids are proposed: the regular, adaptive, and reluctant adaptive variants. The naive regular grid uniformly refines across the whole domain up to the length of the smallest particle diameter. An adaptive variant reduces the refinement requirement across the domain by refining only the special regions of interests (where there are particles there are contacts). Moreover the refinement and coarsening policy is time dependent on the particle positions. We propose an improved version of adaptivity, where we attempt to reduce any grid overhead by minimisation of the number of grid changes per step. The reluctant adaptive scheme wraps around the adaptive scheme and triggers refinement only when particles approach each other at adjacent vertices. Similarly we trigger cell coarsening when two particles separate.

We can make certain on-the-fly assumptions about possible collisions and maximum step sizes based on their hierarchical position in the data structure. We make use of the reluctant schemes' refinement and coarsen policy to define local particle pair-wises states: approach and separation. During the approach state the global step size is reduced towards a contact critical length. When a particle pair separates then both the grid and the time step size coarsens up to the maximum admissible. The underlying contact dynamics of the simulation define the behaviour of both the grid and step size during runtime. This scheme minimises the overall number of particle comparisons as well as the number of steps, this results to a improved time-to-solution.

Parallel DEM. Equipped with a DEM routine mapped to a grid traversal, we propose a novel three layer parallelisation scheme. The triangle, particle and grid-based layers exploit several levels of granularity. At the innermost level a triangle-based parallelisation exploits the tessellation density of the individual particle meshes. This shows excellent scaling behaviour when lots of triangles are compared. At the vertex level, local particle pair comparisons are executed in parallel. Since the grid is employed to minimise particle-to-particle comparisons this level of parallelisation is counter-intuitive but it also act as a safeguard in clustered particle scenarios where refinement is skipped. Finally, at the coarsest level, a parallel task producer/consumer scheme traverses the whole multiscale grid. The intermix of all levels of parallelisation offers reasonable scaling on manycore machines and it is suitable for a contact detection library.

9.2 Outlook

Throughout this project a few ideas for future research did arise. Future work will expand in three areas: DEM geometry physical representation, the generalisation of algorithmic ingredients and many-node HPC.

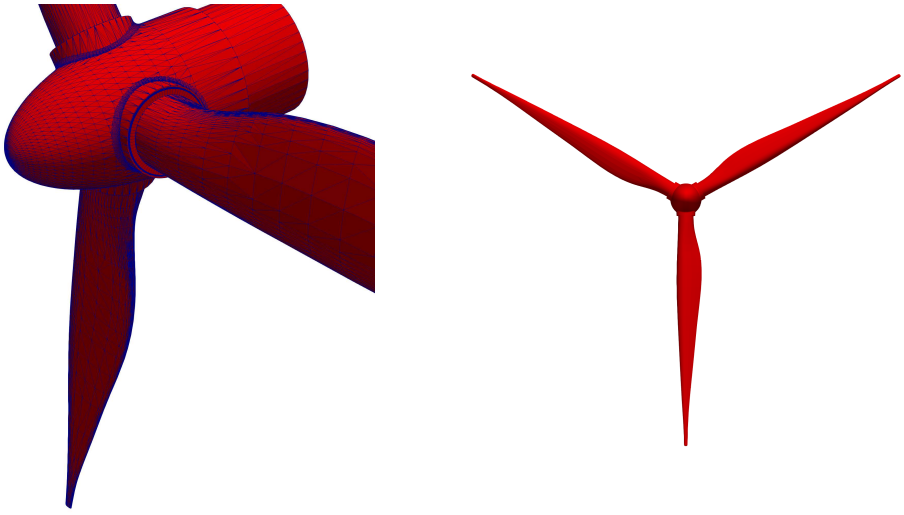


Figure 9.1: Complex geometries imported from CAD software would allow for more realistic DEM simulations.

The current code supports spheres and triangle elements. The addition of further geometric elements and the potential intermix of geometries would lead to new contact models. Interesting interactions would include shapes of ellipsoids, clusters of spheres/ellipsoids and Reuleaux triangles. Additional complex physical phenomena could be simulated (e.g. lubrication models, advanced frictional forces) that are

currently not available. These would allow us to expand the use of the software in a variety of applications.

With respect to algorithmic features, we plan to separate the DEM core code from the grid traversal. The underlying spacetree meta data is based upon tree-partitioning as we rely on the AMR framework Peano [92]. The detachment from the Peano grid framework makes the DEM code grid framework independent and modular. The source code should be directly applicable for any tree-based data structure without any dependencies (see Appendix). The whole DEM algorithm should be designed as a single callable class. At the highest level of abstraction the DEM workflow should be categorised into: geometry import (Figure 9.1) routines, the world creation, simulation and post-processing.

The parallelisation strategies in our manuscript are studied by means of a many-core shared memory. However, the grid-based parallelisation paradigm is also applicable to distributed memory environments. The grid-based parallelism describes a classic space domain decomposition. We could process cells in parallel by assigning each tree region to a CPU. The grid partitioning yields a particle distribution and we can ensure that all contact points are found per each domain chunk by augmentation of the boundaries with one ghost cell layer. In such a scheme, computation of particles within the ghost cells is replicated but global elimination of replication is straight-forward. With such redundancy along sub-domain boundaries, it mandates a balanced distributed memory rank communication and computation. A discussion of the distributed particle administration which includes parallelised multiscale lifts and drops can be found in [95]. Computation with lower numerical precision in distributed memory environment would also be interesting to investigate; notably following the scheme found in [22].

Further Application Areas. A generalised DEM library is potentially useful for non-DEM codes too. for handling the preprocessing of geometries, to setup particle positions, particle orientation and initial parameters of the simulation in addition to contact detection routines. Moreover, a generalised code within the context of structured grids is a useful tool for distance map definition between the grid vertices and geometries (Figure 9.2). Wherever the distance map defines values within the superimposed grid greater than zero then its cells represent the voxelisation of the real geometry.

Voxelisation is a interesting research area for both DEM and Computational Fluid Dynamics (CFD) studies. Voxels of the real geometry superimpose a "shadow" onto the underlying spacetree. This results in the pixelisation of a given real geometry (Figure 9.2, right). According to [101] there are several studies in literature where

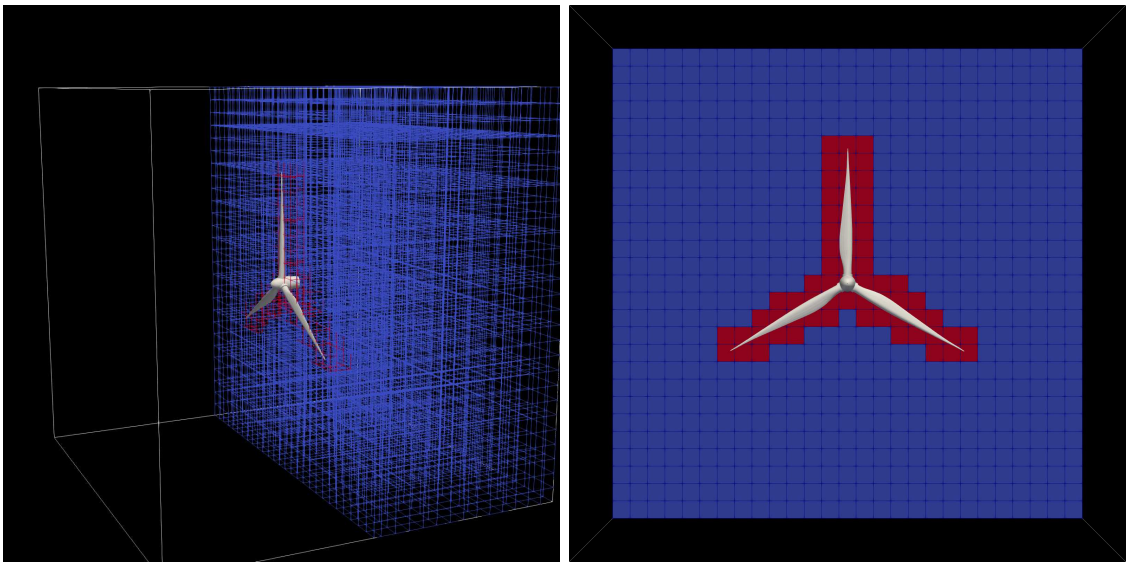


Figure 9.2: A complex geometry is used as a reference to create voxels.

pixelisation techniques realised by a contact detection method. It would interesting to investigate contact detection as an add-on pre-check routine for non-spherical meshed particles.

For CFD applications distance maps between mesh nodes and the triangulated rigid body geometry would be useful. Fluid-structure interactions would benefit from such interface data at the boundary. The integration of the DEM into CFD codes comes out of the box, the DEM dynamics are completely separate by design and they could be extended to accept additional inputs from the CFD code.

Contact detection between particles with dense triangulated geometries does not make sense due to the increased triangle-to-triangle check redundancy. We propose to reduce the redundancy by discretising the bodies into search areas that are built upon hierarchical cubic cuts of an octree. Contact detection thus can be augmented with group of octrees before the actual mesh-to-mesh contact detection. This strategy is an improvement over a sphere-based pre-check based on cubes. A preliminary example of the octree-based method is shown in Figure 9.3. The octree is applied to every geometry separately. Both regular or adaptive cuts can be applied, the adaptive variant refines along the edges/curves of mesh. At contact detection the triangle group closest to a potential contact is extracted with a pre-check routine, next the actual triangle-to-triangle distance computation is performed. The octree acts as a on-the-fly adaptive bounding box pre-check phase.

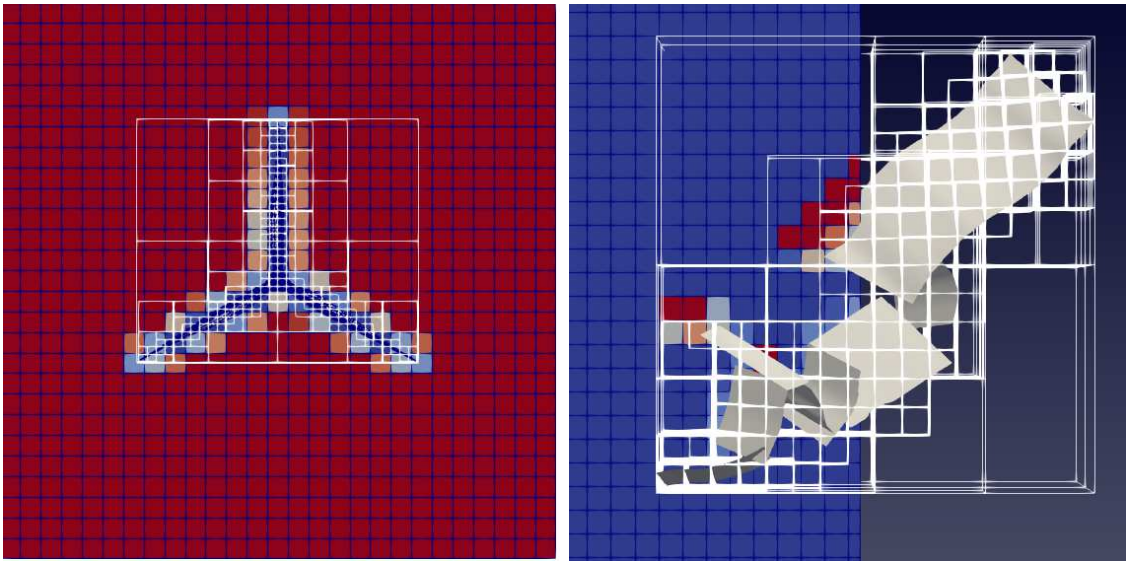


Figure 9.3: Two complex geometries are used to superimpose a octree bounding box prior to the contact detection and voxelisation. The colours at the boundary of the objects indicate the distance maps between the grid nodes and the body geometry.

Bibliography

[1] A. Allan, D. Edenfeld, William H. Joyner, J., A. B. Kahng, M. Rodgers, and Y. Zorian. 2001 technology roadmap for semiconductors. *Computer*, 35(1):42–53, January 2002.

[2] M. F. Alonso, G. A. Ramírez, M. C. González, N. Balaam, D. A. H. Hanaor, E. A. Flores, J., Y. Gan, S. Chen, and L. Shen. Experimental and numerical determination of mechanical properties of polygonal wood particles and their flow analysis in silos. *Granular Matter*, 15(6):811–826, 2013.

[3] S. Amberger, M. Friedl, C. Goniva, S. Pirker, and C. Kloss. Approximation of Objects by Spheres for Multisphere Simulations in DEM. *European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2012)*, (Eccomas), 2012.

[4] C. Bradford Barber, David P. Dobkin, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, December 1996.

[5] J. M. Boac, K. P. R. Ambrose, M. E. Casada, R. G. Maghirang, and D. E. Maier. Applications of Discrete Element Method in Modeling of Grain Postharvest Operations. *Food Engineering Reviews*, 6(4):128–149, 2014.

[6] E. G. Boman, U. V. Catalyurek, C. Chevalier, and K. D. Devine. The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering, and coloring. *Scientific Programming*, 20(2):129–150, 2012.

[7] C. W. Boon, G. T. Houlsby, and S. Utili. Computers and Geotechnics A new algorithm for contact detection between convex polygonal and polyhedral

particles in the discrete element method. *Computers and Geotechnics*, 44:73–82, 2012.

[8] C. W. Boon, G. T. Houlsby, and S. Utili. A new contact detection algorithm for three-dimensional non-spherical particles. *Powder Technology*, 248:94–102, 2013.

[9] K. J. Bowers, R. O. Dror, and D. E. Shaw. Zonal methods for the parallel execution of range-limited N-body simulations. *Journal of Computational Physics*, 221(1):303–329, 2007.

[10] D. E. Charrier, B. Hazelwood, and T. Weinzierl. Enclave Tasking for Discontinuous Galerkin Methods on Dynamically Adaptive Meshes. *arXiv:1806.07984 [cs]*, June 2018. arXiv: 1806.07984.

[11] P. W. Cleary. A multiscale method for including fine particle effects in DEM models of grinding mills. *Minerals Engineering*, 84:88–99, 2015.

[12] P. W. Cleary and M. L. Sawley. DEM modelling of industrial granular flows : 3D case studies and the effect of particle shape on hopper discharge. *Applied Mathematical Modelling*, 26:89–111, 2002.

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

[14] L. Dagum and R. Menon. Openmp: An industry-standard api for shared-memory programming. *IEEE Comput. Sci. Eng.*, 5(1):46–55, January 1998.

[15] S. Das, N. G. Deen, and J. A. M. Kuipers. Multiscale modeling of fixed-bed reactors with porous (open-cell foam) non-spherical particles: Hydrodynamics. *Chemical Engineering Journal*, 334(August 2017):741–759, 2018.

[16] B. Delaunay. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.

[17] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, Xuebin Chi, a. Choudhary, S. Dosanjh, T. Dunning, S. Fiore, a. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, a. Hoisie, K. Hotta, Zhong Jin, Y. Ishikawa, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, a. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M. S. Mueller, W. E. Nagel, H. Nakashima, M. E.

Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, a. Trefethen, M. Valero, a. van der Steen, J. Vetter, P. Williams, R. Wisniewski, and K. Yelick. The International Exascale Software Project roadmap. *International Journal of High Performance Computing Applications*, 25(1):3–60, 2011.

[18] J. Dongarra, J. Hittinger, et al. Applied Mathematics Research for Exascale Computing, 2014. DOE ASCR Exascale Mathematics Working Group. [Online] Available from:http://www.netlib.org/utk/people/JackDongarra/PAPERS/doe-exascale-math-report.pdf.

[19] D. Eberly. Intersections of a Set of Segments. Geometric Tools. "`http://geomalgorithms.com/a09-_intersect-3.html`", 2008. [Online; accessed 1-July-2015].

[20] D. Eberly. Planes and Distance of a Point to a Plane. Geometric Tools.. "`http://geomalgorithms.com/a04-_planes.html`", 2008. [Online; accessed 1-July-2015].

[21] W. Eckhardt. Efficient HPC Implementations for Large-Scale Molecular Simulation in Process Engineering. 2014.

[22] W. Eckhardt, R. Glas, D. Korzh, S. Wallner, and T. Weinzierl. On-the-fly memory compression for multibody algorithms. In *Advances in Parallel Computing*. IOS Press, 2015. (in press).

[23] W. Eckhardt, R. Glas, D. Korzh, S. Wallner, and T. Weinzierl. On-the-fly memory compression for multibody algorithms. *Advances in Parallel Computing*, 27:421–430, 2016.

[24] W. Eckhardt, A. Heinecke, R. Bader, M. Brehm, N. Hammer, H. Huber, H. G. Kleinhenz, J. Vrabec, H. Hasse, M. Horsch, M. Bernreuther, C. W. Glass, C. Niethammer, A. Bode, and H. Bungartz. 591 TFLOPS multi-trillion particles simulation on SuperMUC. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7905 LNCS:1–12, 2013.

[25] A. E. Eichenberger, P. Wu, and K. O'Brien. Vectorization for SIMD architectures with alignment constraints. *ACM SIGPLAN Notices*, 39(6):82, 2004.

[26] A. Elafrou and K. Kourtis. SparseX: a Library for High-Performance Sparse Matrix-Vector Multiplication on Multicore Platforms. *ACM Trans. Math. Softw*, 1(1):1–32, 2017.

[27] C Ericson. The Gilbert-Johnson-Keerthi algorithm, 2005.

[28] Christer Ericson. Real-time collision detection. In Christer Ericson, editor, *Real-Time Collision Detection*, The Morgan Kaufmann Series in Interactive 3D Technology, pages 1 – 6. Morgan Kaufmann, San Francisco, 2005.

[29] F. Fleissner and P. Eberhard. Parallel Load Balanced Particle Simulation with Hierchical Particle Grouping Strategies. *Prof. of IUTAM 2006, Multiscale Problems in Multibody System Contacts*, pages 33–44, 2006.

[30] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, 21(9):948–960, September 1972.

[31] F. Y. Fraige, P. A. Langston, and L. A. Al-Khatib. Polyhedral Particles Hopper Flowrate Predictions using Discrete Element Method. *Chemical Product and Process Modeling*, 6(1), 2011.

[32] F. Franchetti and M. Püschel. Generating SIMD vectorized permutations. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4959 LNCS, pages 116–131, 2008.

[33] X. Garcia, J. P. Latham, J. Xiang, and J. P. Harrison. A clustered overlapping sphere algorithm to represent real particles in discrete element modelling. *Géotechnique*, 59(9):779–784, 2009.

[34] P. Gonnet. Efficient and Scalable Algorithms for Smoothed Particle Hydrodynamics on Hybrid Shared/Distributed-Memory Architectures. *SIAM Journal on Scientific Computing*, 37(1):C95–C121, 2014.

[35] M. Griebel, S. Knapek, and G. Zumbusch. *Numerical Simulation in Molecular Dynamics*. Springer, Berlin, Heidelberg, 2007.

[36] N. Guo and J. Zhao. 3D multiscale modeling of strain localization in granular media. *Computers and Geotechnics*, 80:360–372, 2016.

[37] D. Harmon, E. Vouga, R. Tamstorf, and E. Grinspun. Robust treatment of simultaneous collisions. *ACM SIGGRAPH 2008 papers on - SIGGRAPH '08*, 27(3):1, 2008.

[38] K. Hauser. Robust contact generation for robot simulation with unstructured meshes. In *Springer Tracts in Advanced Robotics*, volume 114, pages 357–373, 2016.

[39] D. Höhner, S. Wirtz, and V. Scherer. Experimental and numerical investigation on the influence of particle shape and shape approximation on hopper discharge using the discrete element method. *Powder Technology*, 235:614–627, 2013.

[40] D. Höhner, S. Wirtz, and V. Scherer. A study on the influence of particle shape and shape approximation on particle mechanics in a rotating drum using the discrete element method. *Powder Technology*, 253:256–265, 2014.

[41] X. Hou, T. Ding, Z. Deng, Z. Yu, P. Xue, P. Cao, and T. Tang. Study of the creeping of irregularly shaped Martian dust particles based on DEM-CFD. *Powder Technology*, 328:184–198, 2018.

[42] G. T. Houlsby. Computers and Geotechnics Potential particles : a method for modelling non-circular particles in DEM. *Computers and Geotechnics*, 36(6):953–959, 2009.

[43] L. Hu, G. M. Hu, Z. Q. Fang, and Y. Zhang. A new algorithm for contact detection between spherical particle and triangulated mesh boundary in discrete element method simulations. *International Journal for Numerical Methods in Engineering*, (March):787–804, 2013.

[44] K. Iglberger and U. Rüde. Massively parallel granular flow simulations with non-spherical particles. *Computer Science - Research and Development*, 25(1-2):105–113, 2010.

[45] K. Iglberger and U. Rüde. Large-scale rigid body simulations. *Multibody System Dynamics*, 25(1):81–95, 2011.

[46] A. Jensen, K. Fraser, and G. Laird. Improving the Precision of Discrete Element Simulations through Calibration Models. *13 th International LS-DYNA Users Conference*, 7(88):405–410, 2014.

[47] D. E. Knuth. Semantics of context-free languages. *Mathematical systems theory*, 2(2):127–145, Jun 1968.

[48] T. Koziara and N. Bićanić. Simple and efficient integration of rigid rotations suitable for constraint solvers. *International Journal for Numerical Methods in Engineering*, 81(9):1073–1092, 2010.

[49] T. Koziara and N. Bićanić. A distributed memory parallel multibody Contact Dynamics code. *International Journal for Numerical Methods in Engineering*, 87(1-5):437–456, 2011.

[50] K. Krestenitis, T. Koziara, and T. Weinzierl. Delta, 2016. github.com/KonstantinosKr/delta.

[51] K. Krestenitis, T. Weinzierl, and T. Koziara. Fast dem collision checks on multicore nodes. In R. Wyrzykowski, J. J. Dongarra, E. Deelman, and K. Karczewski, editors, *Parallel processing and applied mathematics : 12th International conference, PPAM 2017, Lublin, Poland, September 10-13; revised selected papers. Part 1.*, number 10777 in Lecture Notes in Computer Science, pages 123–132. Springer Science, 2018.

[52] H. Kruggel-Emden, S. Rickelt, S. Wirtz, and V. Scherer. A study on the validity of the multi-sphere Discrete Element Method. *Powder Technology*, 188(2):153–165, 2008.

[53] J. E. Lane, P. T. Metzger, and R. A. Wilkinson. A Review of Discrete Element Method (DEM) Particle Shapes and Size Distributions for Lunar Soil. *Nasa Tm*, (December):43, 2010.

[54] C. Q. Li, W. J. Xu, and Q. S. Meng. Multi-sphere approximation of real particles for DEM simulation based on a modified greedy heuristic algorithm. *Powder Technology*, 286:478–487, 2015.

[55] T. Li and J. Chen. Incremental 3D collision detection with hierarchical data structures. *Proceedings of the ACM symposium on Virtual reality software and technology 1998 - VRST '98*, 1998:139–144, 1998.

[56] Y. Li, N. Gui, X. Yang, J. Tu, and S. Jiang. Effect of friction on pebble flow pattern in pebble bed reactor. *Annals of Nuclear Energy*, 94:32–43, 2016.

[57] S. D. Liu, Z. Y. Zhou, R. P. Zou, D. Pinson, and A. B. Yu. Flow characteristics and discharge rate of ellipsoidal particles in a flat bottom hopper. *Powder Technology*, 253:70–79, 2014.

[58] E. Loth. Drag of non-spherical solid particles of regular and irregular shape. *Powder Technology*, 182(3):342–353, 2008.

[59] L. Lu, A. Rahimian, and D. Zorin. Contact-aware simulations of particulate Stokesian suspensions. 2016.

[60] Matuttis. *Understanding the discrete element method : simulation of non-spherical particles for granular and multi-body systems.* Wiley, Singapore, 2014.

[61] J. D. McCalpin. Stream: Sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, Charlottesville, Virginia, 1991-2007. A continually updated technical report. http://www.cs.virginia.edu/stream/.

[62] M. McCann and N. Pippenger. Srt division algorithms as dynamical systems. *SIAM Journal on Computing*, 34(6):1279–1301, 2005.

[63] G. Mollon and J. Zhao. The influence of particle shape on granular Hopper flow. *AIP Conference Proceedings*, 1542(1):690–693, 2013.

[64] B. Nassauer, T. Liedke, and M. Kuna. Polyhedral particles for the discrete element method: Geometry representation, contact detection and particle generation. *Granular Matter*, 15(1):85–93, 2013.

[65] K. Nidhi, S. Indrajeet, M. Khushboo, K. Gauri, and D. J. Sen. Hydrotropy: A promising tool for solubility enhancement: A review. *International Journal of Drug Development and Research*, 3(2):26–33, 2011.

[66] E. J. R. Parteli. DEM simulation of particles of complex shapes using the multisphere method: Application for additive manufacturing. In *AIP Conference Proceedings*, volume 1542, pages 185–188, 2013.

[67] M. Pharr and W. R. Mark. ispc: A spmd compiler for high-performance cpu programming. In *2012 Innovative Parallel Computing (InPar)*, pages 1–13, May 2012.

[68] C. Pheatt. Intel® threading building blocks. 23:298–298, 04 2008.

[69] S. Plimpton. Fast Parallel Algorithms for Short – Range Molecular Dynamics. *Journal of Computational Physics*, 117(June 1994):1–19, 1995.

[70] T. Pöschel and T. Schwager. *Computational Granular Dynamics—Models and Algorithms.* Springer, 2005.

[71] A. D. Rakotonirina and J. Delenne. A parallel discrete element method to model collisions between non-convex particles. In *Powders & Grains*, volume 06004, pages 1–4, 2017.

[72] A. D. Rakotonirina and A. Wachs. Grains3D, a flexible DEM approach for particles of arbitrary convex shape - Part II: Parallel implementation and scalable performance. *Powder Technology*, 324:18–35, 2018.

[73] J. Reinders. *Intel Threading Building Blocks*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, first edition, 2007.

[74] K. Ridgway and R. Rupp. The effect of particle shape on powder properties. *Journal of Pharmacy and Pharmacology*, 21(1 S):30S–39S, 1969.

[75] V. Rintala, H. Suikkanen, J. Leppänen, and R. Kyrki-Rajamäki. Modeling of realistic pebble bed reactor geometries using the Serpent Monte Carlo code. *Annals of Nuclear Energy*, 77:223–230, 2015.

[76] A. Ruszczynski. *Nonlinear Optimization*. Princeton University Press, Princeton, 2011.

[77] C. H. Rycroft, A. Dehbi, T. Lind, and S. Güntay. Granular flow in pebble-bed nuclear reactors: Scaling, dust generation, and stress. *Nuclear Engineering and Design*, 265:69–84, 2013.

[78] W. Schroeder. *The visualization toolkit : an object-oriented approach to 3D graphics*. Kitware, Clifton Park, N.Y, 2006.

[79] A. Shahbahrami, B. Juurlink, and S. Vassiliadis. Performance Impact of Misaligned Accesses in SIMD Extensions. *Proceedings of the 17th Annual Workshop on Circuits, Systems and Signal Processing*, pages 334–342, 2006.

[80] D. E. Shaw. A fast, scalable method for the parallel evaluation of distance-limited pairwise particle interactions. *Journal of Computational Chemistry*, 26(13):1318–1328, 2005.

[81] B. Soltanbeigi, A. Podlozhnyuk, S. A. Papanicolopulos, C. Kloss, S. Pirker, and J. Y. Ooi. DEM study of mechanical characteristics of multi-spherical and superquadric particles at micro and macro scales. *Powder Technology*, 329:288–303, 2018.

[82] E. Somogyi, A. A. Mansour, and P. J. Ortoleva. ProtoMD: A prototyping toolkit for multiscale molecular dynamics. *Computer Physics Communications*, 202:337–350, 2016.

[83] M. Spellings, R. L. Marson, J. A. Anderson, and S. C. Glotzer. GPU accelerated Discrete Element Method (DEM) molecular dynamics for conservative,

faceted particle simulations. *Journal of Computational Physics*, 334:460–467, 2017.

[84] A. R. Thornton, Dinant L., A. Voortwis, V. Ogarko, S. Luding, R. Fransen, S. Gonzalez, O. Bokhove, I. Olukayode, and Weinhart T. *A review of recent work on the Discrete Particle Method at the University of Twente: An introduction to the open-source package MercuryDPM*, pages 50–56. Colorado School of Mines, 2013.

[85] X. Tian, H. Saito, S. V. Preis, E. N. Garcia, S. S. Kozhukhov, M. Masten, A. G. Cherkasov, and N. Panchenko. Effective SIMD Vectorization for Intel Xeon Phi Coprocessors. *Scientific Programming*, 2015, 2015.

[86] Q. Tong and S. Li. Multiscale coupling of molecular dynamics and peridynamics. *Journal of the Mechanics and Physics of Solids*, 95:169–187, 2016.

[87] J. Treibig, G. Hager, and G. Wellein. LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments. In *Proceedings of the 2010 39th International Conference on Parallel Processing Workshops*, ICPPW '10, pages 207–216. IEEE Computer Society, 2010.

[88] A. Wachs, L. Girolami, G. Vinay, and G. Ferrer. Grains3D, a flexible DEM approach for particles of arbitrary convex shape - Part I: Numerical model and validations. *Powder Technology*, 224:374–389, 2012.

[89] T. Weinhart, R. Hartkamp, A. R. Thornton, and S. Luding. Coarse-grained local and objective continuum description of three-dimensional granular flows down an inclined surface. *Physics of Fluids*, 25(7):070605, 7 2013.

[90] T. Weinzierl. *A Framework for Parallel PDE Solvers on Multiscale Adaptive Cartesian Grids*. Verlag Dr. Hut, München, 2009.

[91] T. Weinzierl. The peano software—parallel, automaton-based, dynamically adaptive grid traversals. *ACM Transactions on Mathematical Software*, 2018. (submitted; arXiv:1506.04496).

[92] T. Weinzierl et al. Peano—a Framework for PDE Solvers on Spacetree Grids, 2016. www.peano-framework.org.

[93] T. Weinzierl and M. Mehl. Peano – A Traversal and Storage Scheme for Octree-Like Adaptive Cartesian Multiscale Grids. *SIAM Journal on Scientific Computing*, 33(5):2732–2760, October 2011.

[94] T. Weinzierl and M. Mehl. Peano – a traversal and storage scheme for octree-like adaptive cartesian multiscale grids. *SIAM Journal on Scientific Computing*, 33(5):2732–2760, October 2011.

[95] T. Weinzierl, B. Verleye, P. Henri, and D. Roose. Two particle-in-grid realisations on spacetrees. *Parallel Computing*, 52:42–64, 2016.

[96] T. Weinzierl, B. Verleye, P. Henri, and D. Roose. Two particle-in-grid realisations on spacetrees. *Parallel Computing*, 52:42–64, 2016.

[97] C. M. Wensrich and A. Katterfeld. Rolling friction as a technique for modelling particle shape in DEM. *Powder Technology*, 217:409–417, 2012.

[98] J. R. Williams and R. O'Connor. Discrete element simulation and the contact problem. *Archives of Computational Methods in Engineering*, 6(4):279–304, 1999.

[99] B. Yan and R. A. Regueiro. A comprehensive study of MPI parallelism in three-dimensional discrete element method (DEM) simulation of complex-shaped granular particles. *Computational Particle Mechanics*, 2018.

[100] Y. You, M. Liu, H. Ma, L. Xu, B. Liu, Y. Shao, Y. Tang, and Y. Zhao. Investigation of the vibration sorting of non-spherical particles based on DEM simulation. *Powder Technology*, 325:316–332, 2018.

[101] W. Zhong, A. Yu, X. Liu, Z. Tong, and H. Zhang. DEM/CFD-DEM Modelling of Non-spherical Particulate Systems: Theoretical Developments and Applications. *Powder Technology*, 302:108–152, 2016.

[102] Zongyan Zhou, Sida Liu, R P Zou, Aibing Yu, David Pinson, and Paul Zulli. Numerical investigation of piling and hopper flow of ellipsoidal particles. 10 2018.

APPENDIX A

Appendix

A.1 Software

All underlying software is free and open source C++ code, `Delta` Δ is available from [50] and offers all the functionality mentioned in this project. All spacetree and adaptive mesh refinement routines (as mentioned in Chapter 6) used in the present work rely on the framework Peano [90, 92, 93]. All geometric operations as well as DEM-specific compute kernels however are independent of Peano and can be used as standalone function in any other spacetree or simulation software.

We offer `Delta` Δ with single (`float`) and double (`double`). The accuracy is controlled via a compile flag `-DiREAL=`. Subject to studies conducted here we exclusively set precision to `double`. Studies on reduced accuracy computations are beyond the scope of the present work.

A.1.1 Make & Execute Delta

There are two source code folders delta and DEM. Delta holds the delta library code while DEM holds the glue code is the interface between DEM and Peano instructions calls to delta. Peano framework can be downloaded separately and be linked with "ln -s ../../directory/peano/src/peano peano". In addition to Peano the DEM code also uses a c++ toolkit provided by Peano called tarch. The tarch directory can be linked by doing "ln -s ../../directory/peano/src/tarch tarch".
Quick Make Command

```
cd gitpathdirectory/delta_peano/
ln -s ../../directory/peano/src/peano peano
ln -s ../../directory/peano/src/tarch tarch
make dim3-icc-release-vec
```

The make command follows the following format: make [dimensions]-[compiler]-[release/debugl-[parallelism]. The dimensions component takes dim3 or dim2 for two and three dimensions, currently only three dimensions is supported. The compiler components takes icc or gcc. The release/debug component switches between optimisations for optimal performance and debug mode. The parallelism component takes vec, novec, tbb, mpi, omp, omp-triangle, omp-particle, tbb-omp-triangle, tbb-omp-particle. The Makefile contains all commands for all make combinations.

The execution command format is as follows:
./[dem-execname] [maxGridH] [scenario] [iterations] [gridType] [step size] [plot frequency] [simulated time] [gravity] [scheme] [mesh density]

i. The [dem-execname] component is the executable file that is compiled with the

make command.

ii. The [maxGridH] component specifies the maximum cell size.

iii. The [scenario] component specifies the predefined/build-in collision scenario.

iv. The [iterations] component specifies the number of time iterations that will take place in the simulation.

v. The [gridType] component specifies the type of grid to be used (no-grid, regular-grid, adaptive-grid, reluctant-grid).

vi. The [step size] component specifies the size of the time step.

vii. The [plot frequency] component specifies the frequency of plotting (every-iteration, every-batch (50 iterations), never).

viii. The [simulated time] component specifies the maximum simulated time.

ix. The [gravity] component specifies a binary switch (true/false) for gravity.

x. The [scheme] component specifies the collision model (bf (brute force), penalty, sphere, hybrid-with-triangles, hybrid-with-batches).

xi. The [mesh density] component specifies the number of points that are used to generate non-spherical triangulated granulates. This option is only used when non-sphere schemes are executed.

For example, running the executable produced by make dim3-icc-release-vec would be:

```
./dem-3d-release-vec 0.1 hopperUniform 1500 no-grid 0.0001
never 1 true sphere 10
```

If there is an error within the execution command then a pop up appears with the user manual with all scenarios, grid types, plot frequencies that are available.
A nested grid and particle hybrid parallelism setup can be exploited (12CELL_THREADS*_PAR_THRE
using the following openMP [14] affinity commands:

```
setenv OMP_NESTED true : nesting enabled
setenv OMP_NUM_THREADS x : thread counter
setenv OMP_PROC_BIND close : affinity binding
```

A.1.2 Source Code Layout

The source code that is found in the "delta" folder. All functions are documented in the header files of the source code. The source code is structured as follows within five main namespaces. Detailed information can be found in www.peano-framework.org/delta/ and the source code.

- Contact (Contact Detection Routines and Collision Handlers)
- Core (Core Simulation Handlers, Trackers, Metrics, I/O)
- Dynamics (Physics Operators For Orientation Matrix)
- Geometry (Geometry Operators and Preprocess Body Manipulators)
- World (World Entities Layout, Scenario Creators)

Contact. In this namespace we define all functionality relevant to contact detection between two meshes. We define contact points as a structure to conveniently summarise information per contact point. An additional sub-namespace contact::detection defines all contact detection methods for meshes as well as spheres.

Core. In this namespace we define all the IO functionality of the code. We have read/write namespaces to import and export the geometry. In order to support multiple data formats, Assimp library is used. At its basic functionality the code supports a simplified VTK [78] import and export routine. Moreover the core holds a data structure that in the DEM context can hold all the data in one single SoA data structure. Statistics and time measurements can be implemented here.

Dynamics. In Dynamics namespace we define the functionality that is relevant to physics computations (inertia, mass, centre of mass, volume, energy) and energy. Time integration routines are also included here to update angular velocities and orientation matrices. This namespace is relevant to DEM implementations.

Geometry. In Geometry namespace we define all functionality that is relevant to geometry. Here we define the triangle and mesh class as well as objects. A mesh is composed of triangles and an object holds a mesh. Any geometry can be described within a mesh object, that is position, bounding boxes. Routines of meshes implement common procedures that is common to operate on meshes during the initiation phase of the simulation (moveTo(), scale(), getBoundingBox(), etc.). The object construct builds an object that holds in addition to mesh object physical properties (inertia, mass, velocities, etc.). Lastly the namespace holds sub-namespace geometry::operators that hold routines that operate on an unstructured mesh, these

are required to setup the scene of the initial geometry. The geometry::hardcoded holds predefined geometries (hopper, blender, granulates, cube).

World. The world namespace defines the "world. That is all particles that are involved in the simulation. In this namespace we include routines to create assembles of particles. Layout sub-namespace routines define function that align a set dimensional array of particles at a position. The layout routine can be used to stack a group of 10x10x10 particles above a hopper structure. The configuration sub-namespace is used to preconfigure geometry into place and scenario sub-namespace define hardcoded simulation scenarios using the imported or pre-defined geometries.

General remarks. There are a few ways to use the library. Individual functions can be called irrespectively of the DEM workflow as required by the user application. Alternatively, a user can follow the workflow of DEM without looking into the hierarchy of the DEM source code. Lastly, different phases of the DEM workflow can be selected or opt out on demand with pragmas and comments either manually or using the flags provided.

All DEM phases are derived from the basic namespaces that the source code is based upon. Therefore each namespace (except the core) represents the algorithmic phases. More specifically there are only four phases out of which three are repeated per time step.

The DEM algorithm starts off by the creation of the geometric domain. In this phase, the user can choose to either import a geometry or create it manually by hard-coding each triangle. Then to run the simulation, it is required to call a contact detection routine in order to define contact points. The contact points are eventually used to derive a force and to update the position of the geometry utilised per time step.

- World Creation (The geometry is created only once.)
- Contact Detection (Performed if collisions are enabled.)
- Force Derivation (Forces are derived from contacts.)
- Position Update (Particles are translated at every time step.)

World Creation. The library under world::scenarios namespace holds predefined scenarios that can be called and run. For example, the "two particle collision" scenario is invoked by passing the epsilon parameter and mesh density. The function call returns a list of objects in which case the "two particle collision" scenario returns two objects. The list of objects are then passed to the simulation.

Contact Detection. All contact detection methods are implemented in atomic functions that are selected according to the contact model used. Contact detection is

performed between two particles and two primitive geometric elements. All collisions are stored locally in a pair-wise map.

Forces. The contact points collision map is then used to define forces at the contact point. These forces are then transferred to the bodies as velocities.

Position Update. The position of the particles is changed by delta x using the velocities that are derived from the forces applied. The position update is performed at every step of the simulation.

A.2 Randomised Granulate Particle Generation

For our particle generation, we rely on the convex hull algorithm [4]. We place 50 points random on the unit sphere They act as input to the computation of a convex hull which yields around 60 triangles typically. Each vertex of the resulting mesh then is scaled with a random scalar from $[0.75, 1]$. Thus, we obtain distorted particles that do not resemble a sphere and have sharp features. The resulting mesh is finally suitably dilated and scaled.

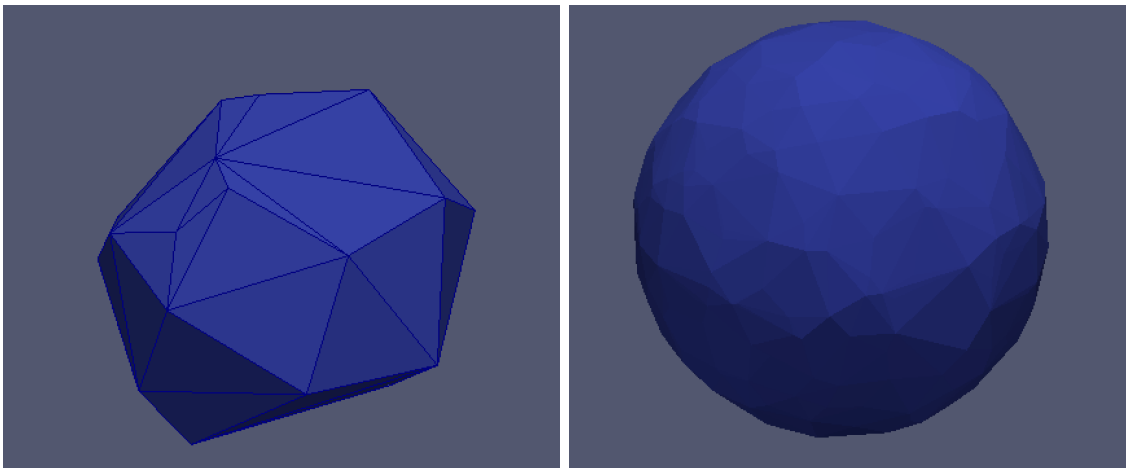


Figure A.1: Coarse and fine spherical particle triangulations using the convex Hull and Delauney algorithms.

The convex particle can then be used in simulation scenarios to resemble a rock granulate. Any large variation in number of points on the unit sphere controls the surface smoothness. The more point are used the more triangles are created to triangulate all points as we do not run a geometry simplification algorithm.

A.3 Application Scenarios

Chapter Outline. In this section we discuss certain simulation scenarios that proved crucial for the development of the DEM code. All simulation scenarios are automatically instantiated during the preprocessing phases of the simulation. Since we construct the object environment within a grid from the start it is convenient to exploit the hierarchical traversal to assign particles to cells during the grid creation. As such, at world creation we distinguish scenarios in two categories: particles only (granulates) and structure-based (e.g. hopper, multi-body structures).

Each simulation scenario is constructed top-down (coarsest level) following the hierarchical traversal. Object-to-vertex associations are automatically assigned by our meta data algorithm as soon as they are instantiated in memory. We distinguish particles based on scale, so coarse particles are created first. When fine particle are created at the coarse level then as the traversal continues the particles fall automatically into place. We instead refrain placing all particles at the coarsest cell to avoid memory movement overhead. An efficient world creation requires as to distinguish fine particles a priori and instantiate them last within the traversal. At the end of the initial traversal all particles are placed in the simulation domain.

As soon as particles are settled at the corresponding levels, we run a second preprocessing phase to prepare the grid. This is mandatory as initial floating point offsets of particle positions may be at the boundary of vertex-to-particle cut-off range (centre of the cell) where any movement towards any direction triggers a vertex association update. The first initial particle movement in a perfectly refined grid triggers large movement in memory as vertex-to-particles associations are updated. To avoid large memory shifts during the actual simulation, we perturbate the affected particle positions towards one direction (e.g. gravity) and update vertex associations in several sweeps. We ensure that positions are not too further away from the original values at machine precision. When adaptivity is enabled, refinement sweeps are also triggered at the same time to stabilise the grid morphology before the first step. The preprocessing phase helps us shift data preparation time before the first time step. We then finally run the DEM simulation.

A.3.1 Particle Studies

The Single particle. To test the grid and the physics algorithms we start off with the single particle setup. A single sphere particle is our baseline test to debug position update routines, multiscale vertex association reassignments and material parameters. Each granulate particle is created by an algorithm that places a random number of points within the unit sphere. The convex hull algorithm [4] creates the mesh that is shrinked to fit into our prescribed diameter size.

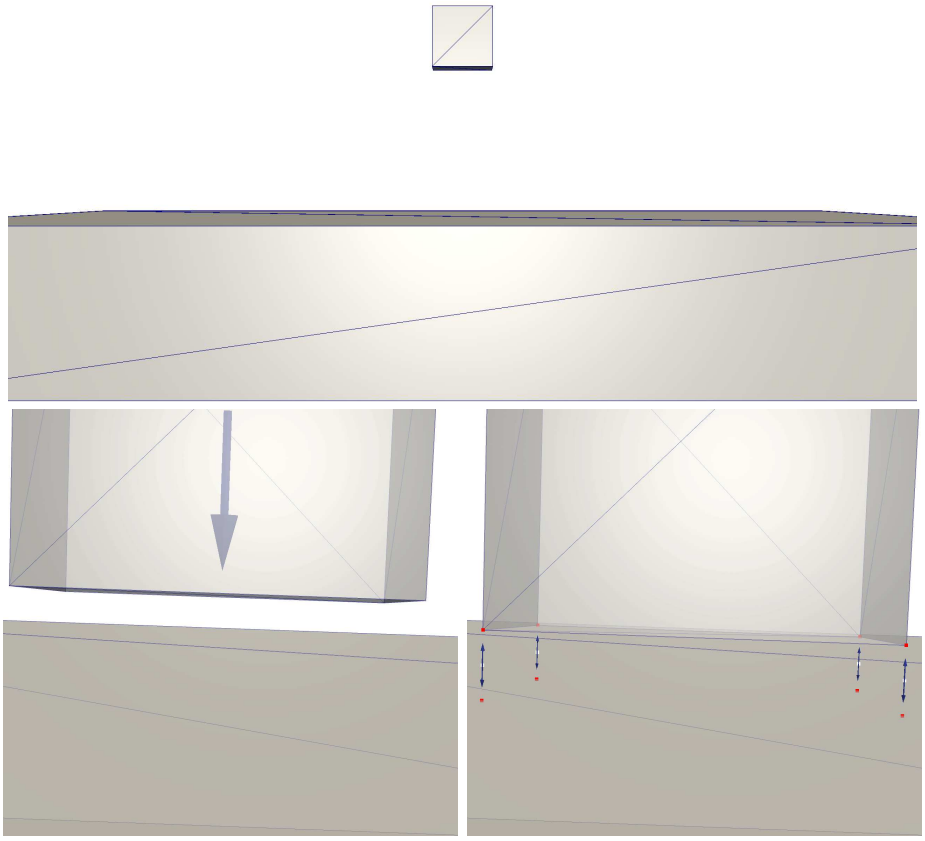


Figure A.2: The cube particle is falling vertically on to a floor. Red points indicate the shortest distance between two triangles while the middle point indicates the middle contact point.

We test single particle interaction against a floor structure (a triangulated box) where gravity is added to the system (see Figure A.2). The floor unlike granular particles is treated as a rigid obstacle object, it only exerts interaction forces but does not update its position. For the floor-based interaction, both spheres and meshed granulates are tested for physical and algorithmic behaviour at the single contact (damping, force, margin parameters). Finally, the same single particle drop setup is employed for more complex phenomena validations like sliding, rolling and friction.

Two particle crash. The next group of tests is the study of interaction between the two granular particle setup. We place the two granulates at a close distance to each other and apply opposite linear velocities. As in all cases, two particles are assumed to collide as soon as they get closer than ϵ. This simple collision setup provides the validation data for all interaction variants (single/multiple contacts) but it also forms the basis of measurement for our algorithmic methods. The impact of contact detection between a single pair of triangulated particles is measured in this scenario. We categorise the algorithmic methods applied in this setup into three parts: triangle-based contact, multiscale grid, dynamic time step.

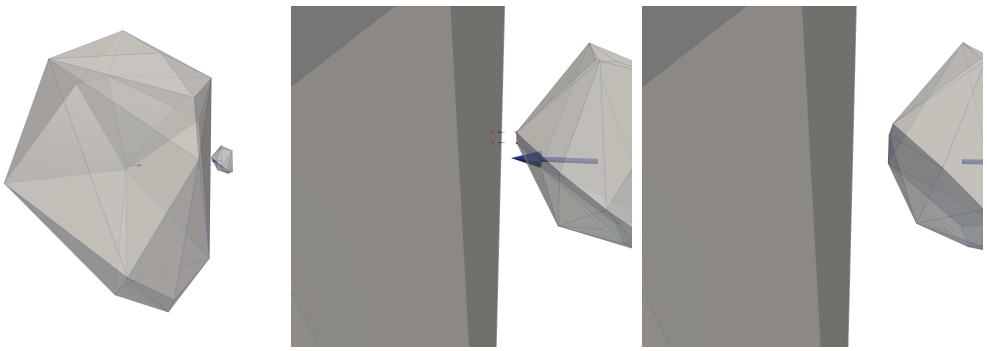
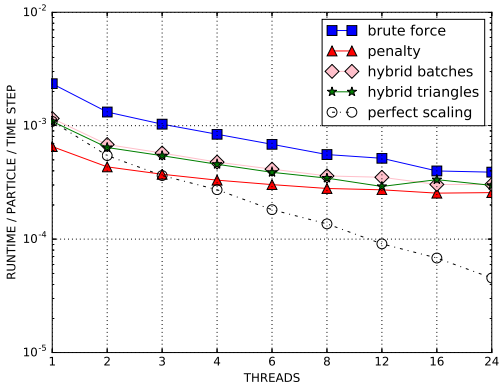


Figure A.3: Two granular particles in collision course (left). At collision we obtain the contact points and derive the forces (middle). The particle separate after collision (right).

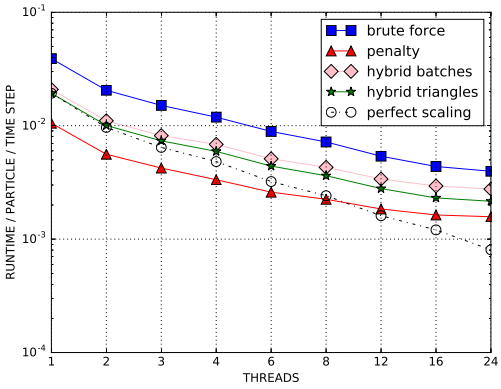
Firstly, the two particle crash (Figure A.3) at close proximity helps us remove all meta-data overhead, computation is reduced only to the comparison of two granulates. This ideal setup allow us to isolate the benchmarks of contact detection methods. The contact models are measured and validated both for the single core and the shared memory (triangle-based parallelisation level) runs. Strong scaling is possible with our granulate generation scheme which allow us to control the problem size by varying the mesh density between runs.

Mesh-based parallelisation scalability tests reveal that on few triangles (e.g. 10) per particle the computation does not scale more than 2-3 cores but this behaviour changes as the number of triangles increases towards 40. In terms of contact method; the penalty method is the fastest followed by hybrid-on-triangles, the slightly slower hybrid on batches, and the slowest brute force method. The larger the triangle number, the better and closer scalability towards perfect scaling we get.

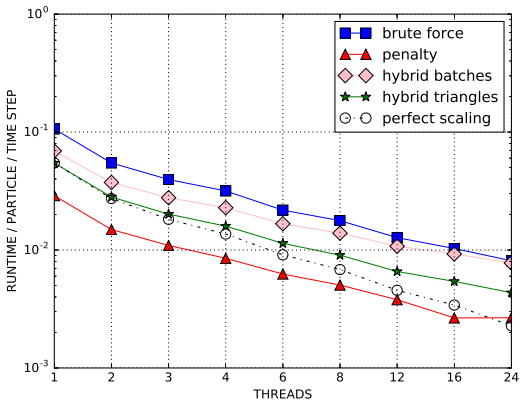
Secondly, the two particle crash is an excellent demonstrator of our proposed set of multiscale grid morphologies. The regular grid exhibits redundancy in vertex comparisons and vertex-to-particle reassignments as the particle position is updated through the grid. The uniformly refined grid disregards particle sizes and adaptive



(a) 10 triangle mesh-based parallelisation



(b) 20 triangle mesh-based parallelisation



(c) 40 triangle mesh based parallelisation

Figure A.4: Mesh-based (triangle-to-triangle) parallelisation using OpenMP.

behaviour is prohibited. The particles eventually collide within a static grid lattice.

Equipped with the adaptive grid, the setup is clearly distinguished from the regular variant. The adaptive morphological behaviour demonstrates reduced number of collision comparisons as the grid applies the local maximum refinement per particle that corresponds to their diameter length. The setup works as well for particles of large scale ratio, they are by default sorted hierarchically in the tree. The drawback of this setup is that the geometry follows both particles throughout and after the collision phase, this induces an administration overhead. The same simulation setup is optimal when the morphology changes only according to the collision dynamics. The relaxed-reluctant variant invokes grid refinement only when there is at least one neighbouring particle in collision course. The two particle crash setup allows us to observe the behaviour of the proposed reluctant scheme.

Lastly, the same two particle setup forms the benchmark scenario for our adaptive time step scheme. The multilevel hierarchy combined with the reluctant grid scheme allows for large time steps in void areas while at same time it minimises morphology changes. Both particles reside in cells that only refine or coarsen when particles are in the state of approach or separation. We match grid refinement and coarsening with the global time step variance. This results to a scheme that minimises the number of total simulation steps which leads to a fast time-to-solution.

Particle freefall. Throughout this project the two particle crash case studies form the basis all particle interactions. With the behavioural studies at hand we create large granulate setups where the whole domain is filled. In our particle freefall (Figure A.5) scenario granulates of various diameters between a given d_{min} and d_{max} are randomly distributed in the unit cube. If particles bump into the unit cube wall, we apply a velocity reflection to keep them within the system. This particular setup is studied to evaluate the robustness of the grid meta data structure during multilevel reassignments.

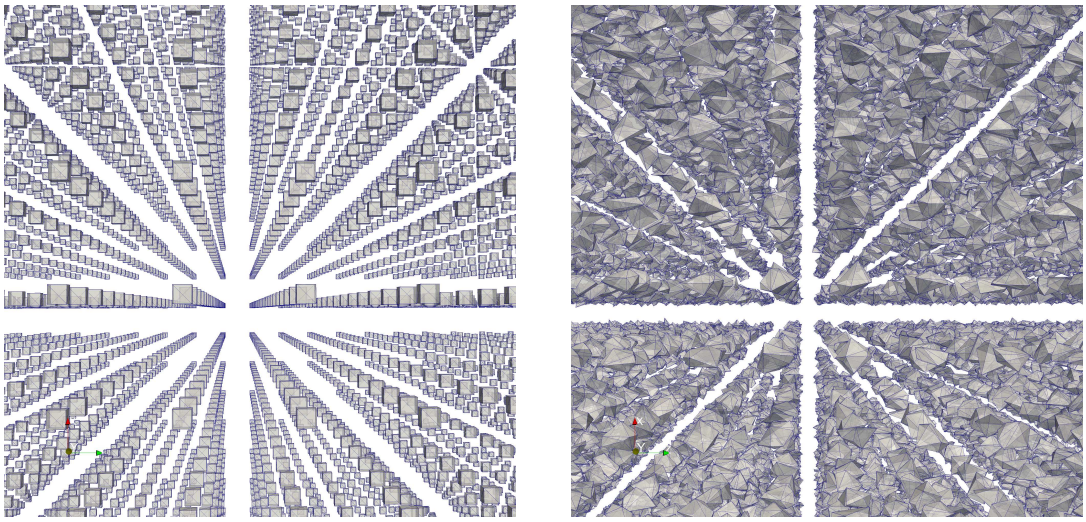


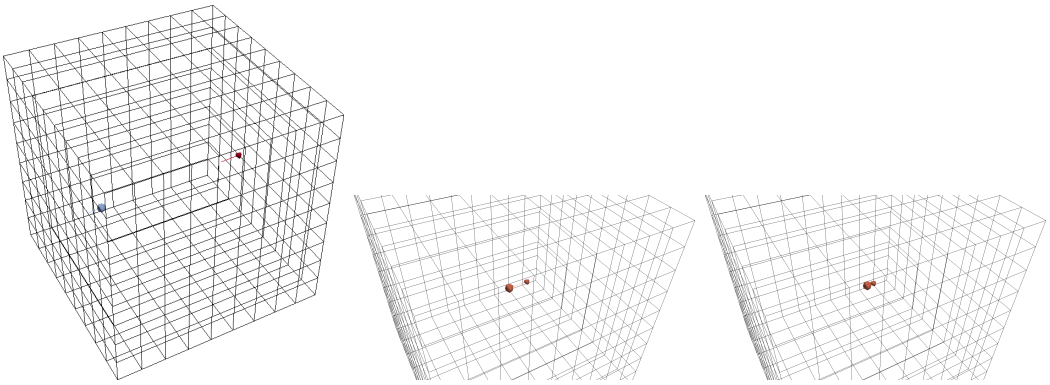
Figure A.5: The freefall scenarios using cubes (left) and granulate (right) particles.

Two different setups are realised: In the first experiment, we do not imply any external force. The particles fly through the unit cube and collide with each other. In the second experiment, we apply uniform gravity. The first setup yields an estimate on how our code performs if the particle collision on the long term is time invariant while the particle distribution is stationary. The second setup yields an estimate on how the code performs when particles cluster and run into a stationary setup. Both experiments thus cover an extreme of the geometric challenge.

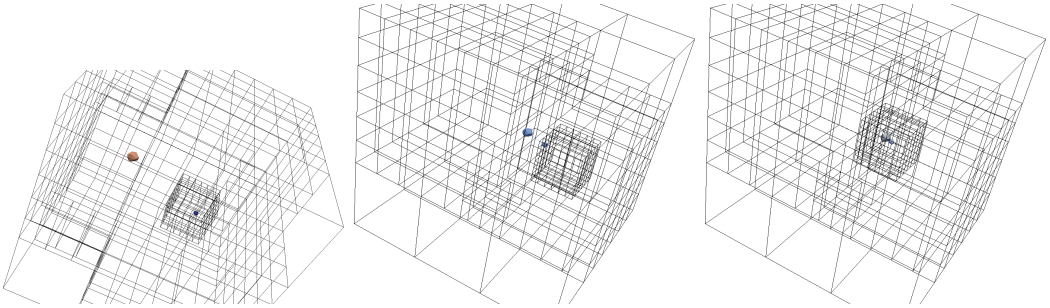
A.3.2 Two Particles Crash

The present experiments study two particles that bump into each other and then move away because of the spring dash-pot forces. The two particles are represented by 126 triangles. We simulate 15,000 time steps. A contact is detected first in iteration 9,541 and introduces a force making the two particles to move away from each other. The code however still needs another 5 iterations to make the particles be away from each other more than $\epsilon = 10^{-4}$. If the two particles are compared, this results in 3,944 triangle-triangle comparisons.

Regular grid. We obtain a grid with 1,032 vertices. The first 6,334 iterations, we do not perform any contact detection. The subsequent grid traversals perform all 3,944 comparisons till iteration 12,503. Starting from time step 12,504, the particles are far away from each other again and we do not compare anything any more.

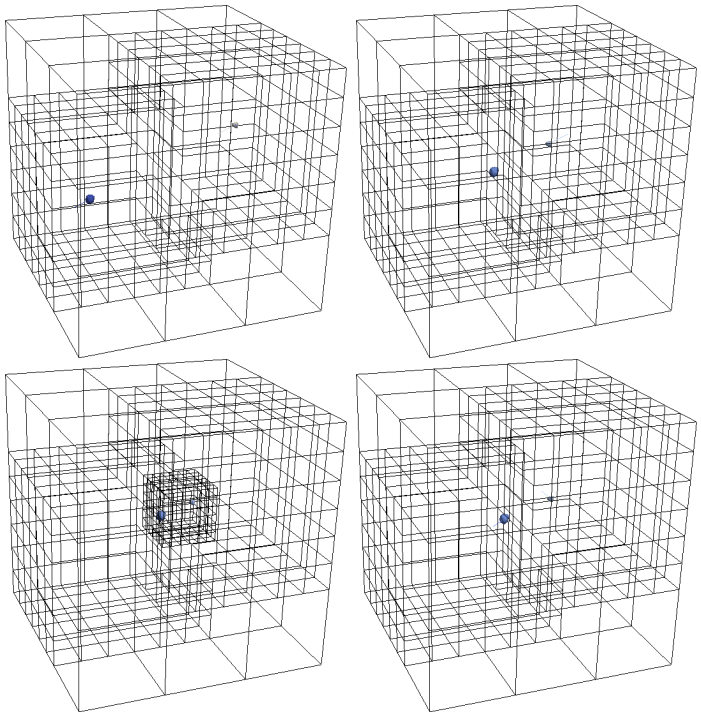


Adaptive grid. We obtain a grid with 747 vertices. Only few iterations (when one particle enters a neighbouring region) make the number increase to 980, but these grids are immediately after reduced to 747 again. The first 8,556 iterations, we do not perform any contact detection. The subsequent grid traversals perform all 3.944 comparisons. In iteration 9,539 we detect contact and the particles start to move away from each other, but we continue to check for further contacts till iteration 10,415. From here on, the particles are far away from each other again and we do not compare anything anymore.



Reluctant adaptive grid. The reluctant adaptive grid behaves qualitatively similar to the plain adaptive grid though yields quantitatively a smaller number of vertices. We start from a grid with 498 vertices, i.e. we do not refine the grid down to the same fine level as the adaptive approach. In iteration 6,334, both particles enter the same coarse grid cell and we run 3,944 triangle-triangle comparisons. They yield no contact yet. The reluctant criterion refines the grid that has now 980 vertices starting from iteration 6,338 on. The code continues to run without any further comparisons till iteration 8,556 when the particles again are close to each other (close this time w.r.t. finest grid level allowed by the particle diameters). Between iteration 8,557 and 9,533 we continue to run with 980 vertices and 3,944 comparisons. The code decides to reduce the grid in iteration 9,536 to 747 vertices when the particles approach each other within a single cube in a fine grid. We detect contact in iteration 9,541 after which the particles move away from each other. The

747 vertices are preserved and the code continues to run 3,944 comparisons. In iteration 10,452, the particles are far away from each other. No more comparisons are performed from here on. In iteration 14,556 the particles are far away from each other, leaving the fine tessellation around the last contact point and we continue with a coarser grid of 498 vertices.



A.3.3 Hopper-Particle Flow

Hopper structure. A more complex scenario is the hopper particle flow. This is an interesting simulation as it is widely studied topic in engineering. The granular flow through hopper structures is found in several industrial processes (manufacturing, rock mechanics, particle processing, e.t.c). We therefore study a basic mock-up scenario to test our methods and provide the ground for large simulations where we can benchmark the proposed DEM algorithm. In this project we do not contribute to the study of granular kinematics but instead we focus on the algorithmic behaviour of the flow.

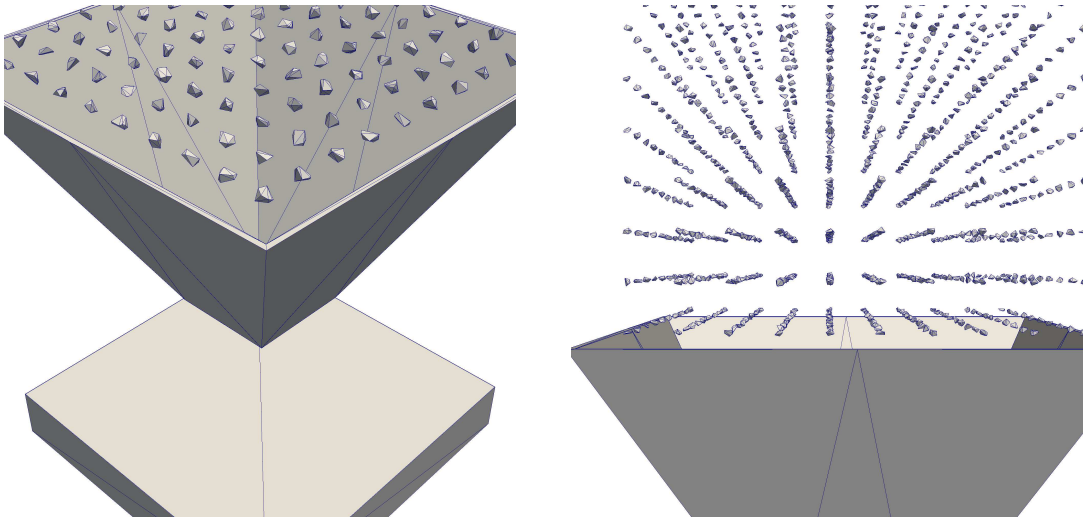


Figure A.6: The hopper structure is composed of four main sides (left). The stack of particles is placed above the structure (right).

The hopper structure (Figure A.6) is composed out of four planes that form an elongated toroid shaped structure. The shape is hollow, which means particles are allowed to move through the gap that intersect the structure. For visualisation purposes we make the triangulated structure water-tight on both sides. Additional triangles are added to the top and bottom of the walls to fill the gaps between the internal and external wall faces. The dimensions of the hopper as well as upper and lower hatches is fixed for our simulation scenarios.

Particles are positioned on the upper hatch of the hopper in a stack formation. The stack size and dimensions is defined based on the number of particles used in the system. At the preprocessing stage we enforce that no particle is positioned above the hopper hatch boundary, we set a distance offset between the hopper hatch wall and the boundary of the particle stack. The stack in formed in layers of hundreds or thousands of particles depending on the given scenario. The shape and sizes of the particles (granulate or spheres, uniformly sized or not) affect the formation layout of the particle stack. The particle stack is always uniform as the positions of the particles is equidistant. In order to have as equidistant formations as possible the stack follows a grid structure where each stack cell element determines the maximum length of a particle regardless of shape. All uniform sized spheres in the stack are perfectly equidistant by default. Non-spherical particles due the random irregularity on their surface are equidistant to each other from the centre of geometry. Nevertheless, with our maximum particle size constraints in-place, the initial setup yields a uniform positioned cluster of particles.

The initial conditions of our hopper-based scenarios vary based on the case study. During the preliminary studies of shapes we vary particle size and shape as well as

number of particles. In the performance studies we focus on the triangulation density and total number of particles. In all scenarios gravity is the only initiating force for the dynamics. Spring forces, damping, materials, epsilon parameters are set and adjusted accordingly a priori. Time step size Δt is chosen reasonably small such the bodies do not inter-penetrate (along with ε) or move over one grid cell at a time. Nevertheless, high velocities require more sophisticated collision models and a more sophisticated vertex association [95].

A.4 The Gilbert Johnson Keerthi Method

Convex non-spherical particles with sharp features, simulation are commonly found to employ the GJK (Gilbert Johnson Keerthi) method [27, 55, 88] to implement a particle contact model. As there is no contribution in this method so we do not discuss multiple variants and models of contact point generation but instead we focus on the generic form. Contact models that are based on the GJK method implement contact point generation in two stages. At the first stage the method determines whether two particles overlap, this is performed using an iterative algorithm. When two particles are found to be in contact, the overlapped region is extracted and often re-triangulated into a convex hull geometry. The extracted region then passes through a secondary algorithm that defines unique contact points and normals. Often a postprocessing filtering phase reduces all redundant contacts [28, 49, 60]. The GJK algorithm operates only on convex bodies and it assumes that both interacting particles' are convex during contact detection. Contact detection of unstructured non-watertight triangles is not possible [38] with this method. Otherwise convexity has to be pseudo-temporally-constructed [27] on run the fly a priori to contact detection.

The GJK method relies on particle interpenetration to detect possible pair-wise collisions. An overlap between two convex shapes is used to indicate a contact state. Although an overlap is non-physical the nonphysical gap can be minimised or resolved explicitly. However in some contact configurations, interpenetration raise the issue of contact divergence as shown in Figure A.7 [38]. Divergence in shapes arise (Figure A.7) when the minimum penetration depths d_1, d_2 of two bodies A, B give rise to one vertical and one horizontal contact. When this interaction is simulated, this particular contact formation leads to a rightward rotation for object A. This state subsequently cause an increased penetration on the next time step. Thus creating an unpredictable outcome during the simulation. This interpenetration problem that lead to undefined behaviour is found in simulation codes that make

use of GJK-based variants [27, 55, 88].

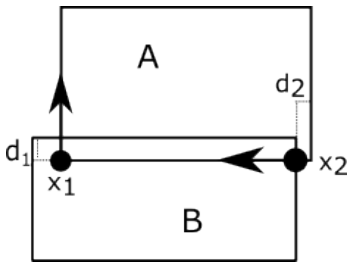


Figure A.7: Contact divergence situation due to penetration. Where A,B are convex bodies, x_1, x_2 contact points with normals, d_1, d_2 is the penetration depth. In the next time step object A will move upwards to the right and object B downwards to the left.

The GJK method primarily relies on the Minkowski sum operation to detect when two particles overlap. For two shapes A and B, the Minkowski sum is all the points in shape A added to all the points in shape B: $A + B = a + b \mid a \in A,\ b \in B$. If both shapes are convex, the resulting shape is also convex. The significance for collision detection is not in the Minkowski summation but in the Minkowski subtraction: $A - B = a - b. \mid a \in A,\ b \in B.$ because if two shapes are intersecting the Minkowski difference will contain the origin. The origin inclusion into the produced shape is used as an indicator for collision.

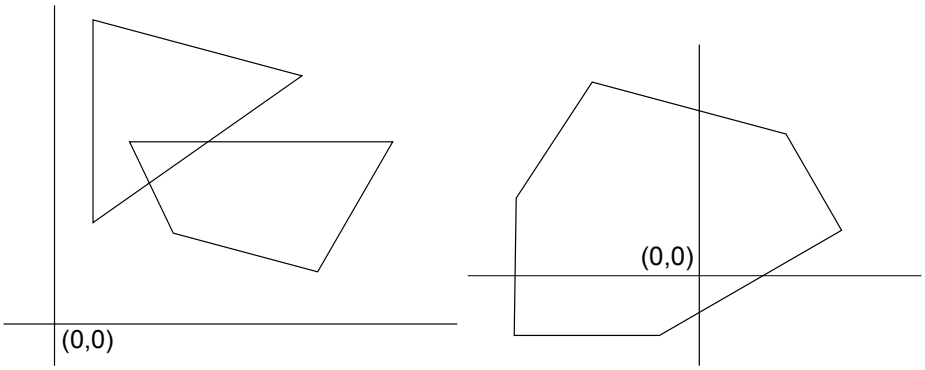


Figure A.8: Left: Two convex shapes are in a state of intersection because their geometries overlap each other. Right: The Minkowski difference of the two bodies enclose the origin which indicates that the bodies intersect and thus are in contact.

The computation of the Minkowski difference on two shapes as shown in Figure A.8 is the fundamental concept behind the algorithm. In Figure A.8 the geometry and position of two particles in 2D space. On the right depiction of Figure A.8 the computed Minkowski difference shape contains the origin. The enclosure of the origin indicates that the two shapes are in contact or intersect. Performing the Minkowski sum requires subtractions along the vertices but it is sufficient to perform the subtraction on selected vertices instead of computing the whole Minkowski

difference. As long as it is possible to ensure enclosure or disclosure of the origin within the Minkowski Difference, the algorithm can determine whether the shapes/-particles are in contact. Executing the GJK algorithm iteratively, a polygon is build in the Minkowski difference.

In order to build the polygon a support function is used. The support function returns a point in the Minkowski difference given two shapes. Take a point from shape A and shape B, take their difference, and obtain a corresponding point in the Minkowski difference and progressively build the polygon. To reduce the steps of the process of constructing the polygon, an origin direction support function can be used so that the simplex is build around the origin faster.

It is important to take the farthest point towards a direction at each iteration using the support function because the simplex creation should contain a maximum area therefore increase the chance of quick convergence). All the points returned by the support function should lie on the edge of the Minkowski Difference, therefore any point that skip the origin along some direction, cannot contain the origin. Using the shapes in Figure A.8 and performing the support function three times, the points of the new simplex results to a triangle (Figure A.9.

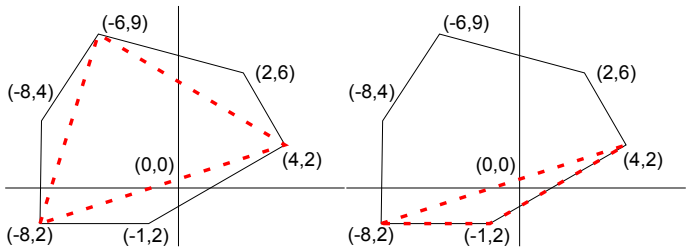


Figure A.9: left: Simplex where origin is not enclosed. right: Simplex including origin. The best-case scenario is when only a triangle is created (three iterations, three simplex points) allowing to determine whether there is an intersection between two convex shapes.

If the third point in the simplex does not contain the origin, the algorithm has to recalculate another point and use that next. It is not possible to guarantee that the first formed triangle will contain the origin nor it is guaranteed that the Minkowski difference contains the origin at all until the algorithm ends. Variants of the GJK algorithm modify the iterative scheme accordingly to optimise the search direction scheme to enclose points towards the origin from within the Minkowski difference space.

There are two conditions that the algorithm needs to meet to converge:

i. Simplex is containing the origin

ii. There is a direction d that encloses the origin by modifying the polygon.

Algorithm 14 Iterative GJK Pseudocode Algorithm.

```
1: function GJK(A, B, C, D, E, F, rho, tol)
2:     d ← initialguess
3:     while true do
4:         simplex.append(support(d))
5:         if simplex.lastVertex · d ≤ 0 then
6:             return false
7:         else
8:             if simplex contains origin then
9:                 return true
10:            else
11:                d ← newdirection(simplex)
12:            end if
13:        end if
14:    end while
15: end function
```

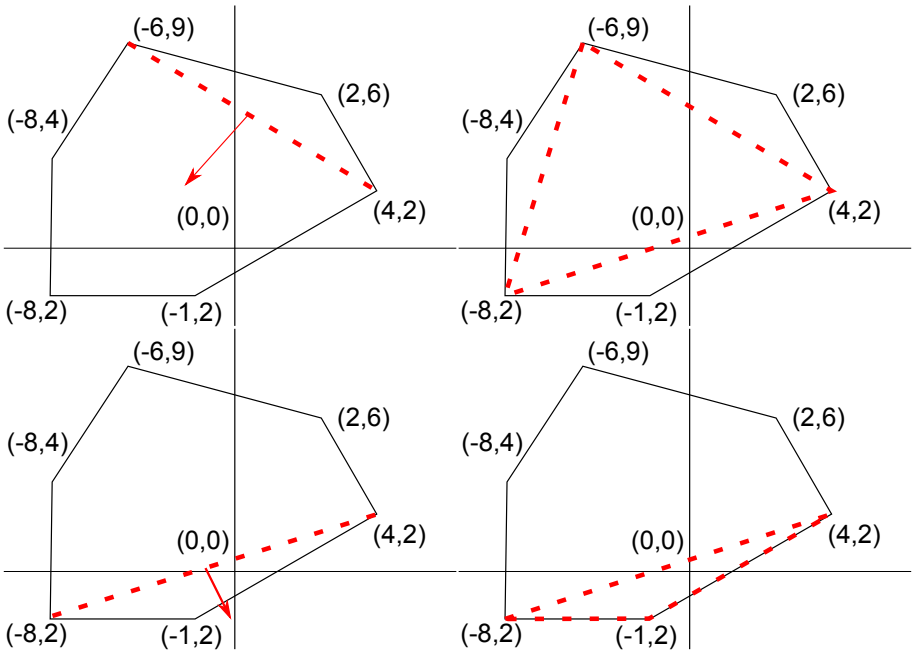


Figure A.10: Top left: First iteration. Top right: Second iteration. Bottom left: Backtrack iteration, set simplex, direction. Bottom right: Third iteration; determined contact.

The process of iteration starts off (see Algorithm 14) from a user defined initial search direction guess d (line 1). The algorithm then progressively forms a simplex geometry. To enhance the iterative process, the search for direction towards the origin is performed by a series of plane checks. When the line segment is defined from point A to point B where A is the last point added to the polygon, then both A and B are on the edge of Minkowski difference and the origin cannot lie in either normal direction (Algorithm 14). The origin can only lie in the rest uncovered area. Since a line segment cannot contain the origin, a third point is required. The perpendicular p of AB line segment towards direction of origin is $p = ((B-A)\times(O-A))\times(B-A)$ where O is the origin. If the origin is on the segment then the perpendicular will be a zero vector. This occurs on two locations:

i. inside the Minkowski Sum

ii. on the edge of the Minkowski Sum.

We compute $(AC \times AB) \times AB$ to yield the perpendicular vector of AB. The second iteration turns the simplex into a triangle shape (Figure A.10). By computing perpendicular $p(AB)\dot{A}O)$ it is determined that the origin is in covered region.

When contact is determined then the next step is to define contact points. The overlappped region is extracted by constructing a new polygon. The overlap polygon is created by determining and extracting the overlapping sides/triangles (line segments, faces, vertices) of the two non-spherical particle pairs that intersect. In three dimensions triangulation of the overlap polyhedron is usually performed using a Delauney or convex hull triangulation algorithm [27]. The contact area of two polyhedra is defined by the intersection line along the surfaces of the two particles and the centre of mass of the overlap polyhedron. The normal direction of the contact point is then defined as the weighted average of the normals to the triangles in the triangulated surface [27].

The GJK algorithm for contact point generation is not favourable for contact simulations for several reasons. Firstly it is only suitable for geometries composed of convex polyhedrons that allow inter-penetration [27, 38]. In addition, contact detection using GJK requires penetration which results in contact normal divergence issues [38]. Contact divergence can be mitigated with additional checking mechanisms in place (contact clustering) [38]. Lastly, the GJK method that requires additional algorithmic steps to contact detection (intersection extraction, triangulation, overlap region extraction, contact clustering) to determine contact points and normals.

A.5 Distributed Memory Parallelisation

Unstructured Grid Parallelisation. To scale the performance of contact detection simulation, the computational workload has to be processed in parallel by splitting the domain into sub-domains, computational deployment is based upon the sub-domain splitting, making the decomposition an important stage. Decomposition affects load balancing and the communication patterns of the simulation. Communication is essential when sub-domains are inter-dependent since new boundaries are introduced with decomposition. It is important to decompose evenly whilst also reducing sub-domain communication. There are three types of decomposition that are widely used in different contexts [21, 69] that apply in DEM. Decomposition by particle, decomposition by force and spatial decomposition.

In domain decomposition by particle [49], the domain is divided such that all sub-domains hold equal number of particles. For N particles we split the domain to N parts and we assign them to P processors (N/P). The sub-domain boundary is always located at the space between particles. In this method an all-to-all communication is required to detect contact points because they are always located at the sub-domain boundary. The global communication is a major disadvantage for small particles as each processor communicates with all other processors, staggering the overall parallel processing. Another disadvantage is the non-equal splitting (N/P) as some particles may require more computation than others (i.e. more triangles). The main advantage of particle-based decomposition is implementation simplicity [69].

An alternative decomposition approach found in literature is force-based decomposition that is often applied in molecular dynamics (MD) and smoother particle hydrodynamic (SPH) simulations [9, 34, 69, 80]. The method formulates an interaction force matrix that solves the contact detection and the forces as a group of linear equations. The matrix is decomposed into small blocks and shared on each process to solve in parallel. Force based decomposition assume short-range interactions and thus a dense force matrix to solve. In force based methods, the advantage is that computation is decomposed without any spatial information from the particles. The major disadvantage in DEM applications is the assumption of dense and uniform sparse force matrices, the assumption can be wrong leading to imbalanced domains on run-time.

According to N-body simulation literature [21, 29, 88] the state-of-the-art method for supercomputing applications is the spatial decomposition. Our decomposition is based on the spatial position of vertices using Recursive Coordinate Bisection (RCB) [6]. In RCB, the computational domain is first divided into two regions by a cutting plane orthogonal to one of the coordinate axes so that half the work load is in

each of the sub-regions. The splitting direction is determined by computing in which coordinate direction the set of objects is most elongated, based upon the geometric locations of the objects. Each triangle vertex thus is owned and persistently stored exclusively on one process/sub-domain for the whole duration of the timestep.

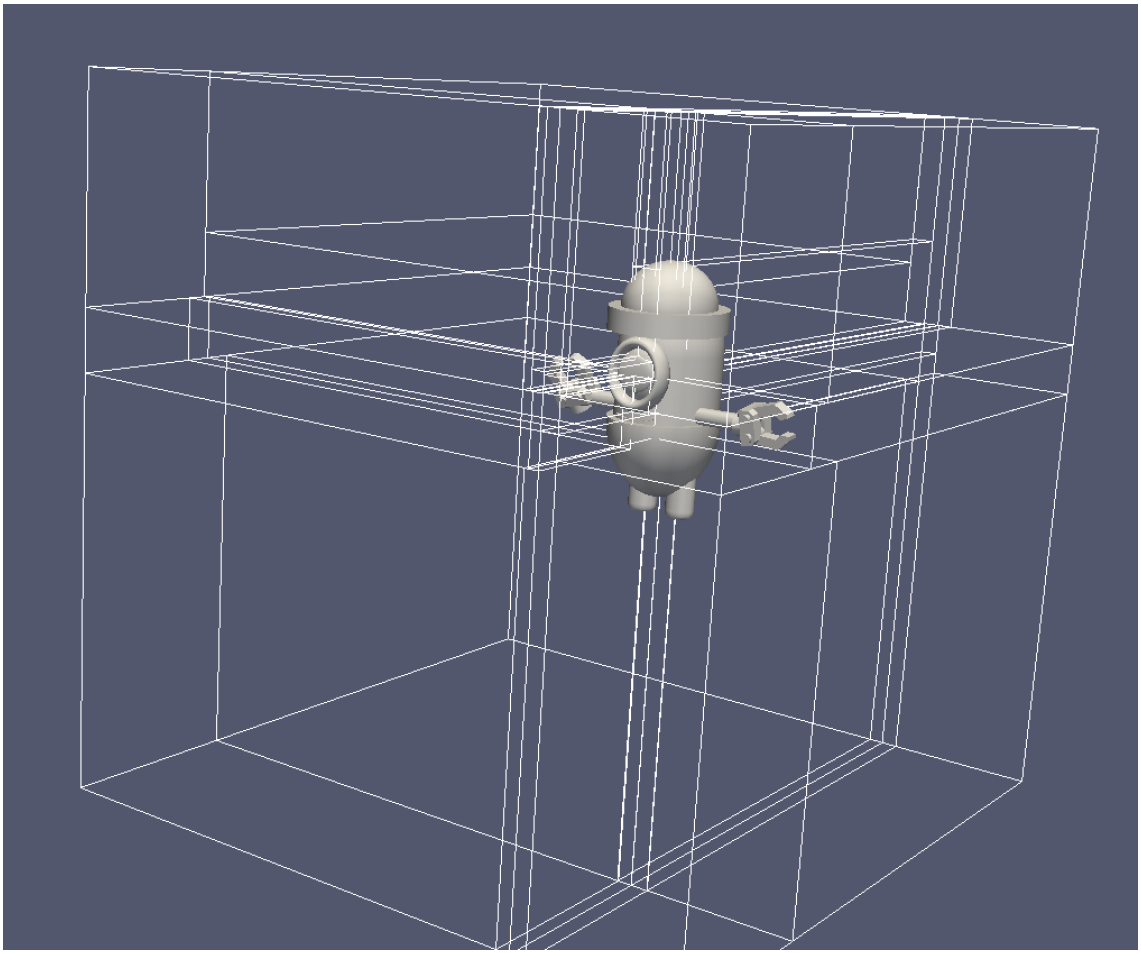


Figure A.11: Spatial domain decomposition using triangular elements on 32 MPI processes, visible boundaries cut space/domain of each process.

Spatial decomposition creates boundaries that split space and computation into sub-domains A.11. The contact detection complexity of N triangles that belong to n_i particles without bounding boxes is $\sum_{i=0}^{n} n_i$. With structured bounding boxes splitting the domain then the total collision detection complexity is reduced. If C is the maximum number of particles then there are 26 neighbouring cells and $max(n_i)$ is the maximum number of triangles per particle i which leads to $n_1 \times 26 \times C \times max(n_i)$ triangles per particle i in the local cell neighbourhood. For all N triangles in the domain there are $N \times 26 \times C \times max(n_i)$ per neighbourhood. Boundary boxes due to spatial decomposition reduce the overall computational complexity to $O(n)$. Similarly, in MD simulations a cut-off range set the range of interaction fields per

particle to remove redundant computation. Equally spaced boundary boxes rely on octree-based variants data structures that are great for recursive eight cell space subdivision. Our decomposition relies on non-uniform recursive subdivisions that rely on kd-tree-based data structure decomposition (i.e. RCB) and the number of neighbours can be arbitrarily many as they are not equally spaced, but they are equally vertex-sized.

The number of neighbours for both methods may have an implication on communication patterns between processes. It is an open question what are the performance implications on uniform spaced octree-based decomposition versus the non-uniform kd-tree based decomposition. It is an area of investigation since on an octree-based approach information about the level of refinement is known a priori by the sub-domain boundary size, which is an interesting application for multiscale simulations.

When spatial decomposition finishes we migrate the data to the processors (Algorithm 15 line 2) with blocking synchronous communication. At each time step the triangles migrate according to the DEM kinematics. In addition to migration, a local area data exchange is required to communicate the boundaries of the sub-domains that cut triangles at the boundary.

Unstructured Sub-domain Halo Data Exchange. The triangles that overlap into a remote sub-domain due to the decomposition cuts are copied to one or more sub-domains in order to perform a complete contact detection. In this section, we study MPI communication characteristics when we resolve data dependencies between the sub-domains caused by ghost triangles in order to maximise performance of distributed contact detection. We explore the implication of inter-process communication and local computational performance using two communication strategies that use asynchronous non-blocking communication.

Algorithm 15 Naive Asynchronous Data Exchange Pseudo-code.

1. Load balance triangles
2. Migrate triangles to MPI network using blocking communication
3. Initiate neighbourhood all-to-all asynchronous MPI send/receive
4. Wait for neighbourhood asynchronous communication to terminate
5. Contact detection
6. Derive contact forces from contact points generated
7. Explicit time integration

The first strategy exchanges local data to all neighbours (Algorithm 15 line 3). The goal is to utilise the communication bandwidth while minimising communication administration overhead. If the exchange does not reach the upper bandwidth limit then exchange of all data is faster than filtering out the ghost triangles, i.e. doing any preprocessing that finds out which triangle from a sub-domain might be

required from a neighbour. Alternatively, we can send out only triangles that overlap from one sub-domain into another sub-domain. This filtering of triangles is an $O(N)$ operation. As soon as MPI communication finishes, the algorithm invokes the existing contact detection routines exploiting vectorised floating point operations with a single contact detection routine. Exchange of all local data to all neighbours significantly increases the number of triangles to be processed from T_{local} to $(T_{local} \cdot N_{ranks} \cdot T_{remote})$ where N_{ranks} is the number of neighbours of neighbours. The increase of triangles to be checked increases the total computation performed locally because of the redundant triangles.

The disadvantage of such a naive method is that total MPI wait (figure A.12) for an all-to-all neighbour data exchange increases with the number of processes. The asynchronous communication wait time results to time wasted with idle processors. The method is potentially useful with decomposition schemes where all data exchange can be processed in the background, i.e. enough bandwidth is available.

Figure A.12: Waiting time (s) per MPI rank/node for all to all neighbour data exchange over 1000 time steps (25 mil triangles, 10k non-spherical particles).

Algorithm 16 Asynchronous Data Exchange Pseudo-code.

```
1 Load balance triangles
2 Migrate triangles to MPI network using blocking communication
3 Search overlapping ghost triangles to send
4 Initiate neighbours asynchronous MPI send/receive
5       Local contact detection
6       Retrieve required ghost triangles from neighbours
7       Local against to remote ghost triangle contact detection
8 Wait for neighbourhood asynchronous communication to terminate (No Real Wait)
9 Derive contact forces from contact points generated
10 Explicit time integration
```

The second strategy filters out local ghosts from the data structure and sends them to the overlapping neighbouring processes. Using the spatial decomposition information, we find the specific processes/boundary cells that the triangle bounding box overlaps. In this strategy we minimise data exchange at the cost of a filtering overhead and the allocation of buffers in memory. The method aims to minimise the waiting time of MPI processes by overlapping concurrent computation over communication. As shown in Algorithm 16 line 5 local contact detection is executed as soon as asynchronous MPI communication is initiated, leading to overlapped communication. A second contact detection is initiated to determine contact between local and remote ghost triangles at line 7. The strategy is advantageous as neighbour waiting is always zero as long as local contact detection takes longer than the transmission of the data. As shown in Figure A.12 the waiting time is increased proportionally to the number of MPI ranks. The waiting time can be overlapped with computation.

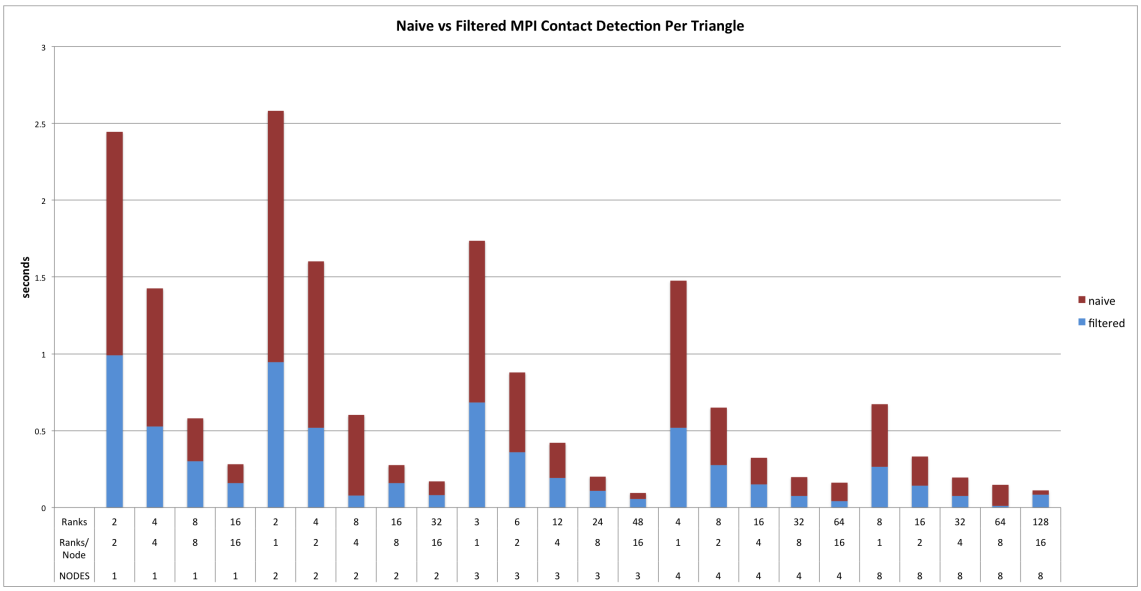


Figure A.13: Normalised naive vs filtered contact detection performance per triangle pair running on multiple nodes.

We measure both strategies to determine the performance of normalised computation per triangle. The result in Figure A.13 shows for each rank and compute node on the x axis, the time required to compute distance of a pair of triangles. The blue bars show the average normalised time for the second strategy to process a triangle (filtering approach). The red bars show the difference between the overall naive communication time and the filtered method. The time is reduced linearly as the number of ranks increase.

For the measurement set-up in A.13 we use Durham University Hamilton supercomputer that has per node 2 x Intel Xeon E5-2650 v2 (Ivy Bridge) 8 cores, 2.6 GHz

processors, 64 GB DDR3 memory, 1 x TrueScale 4 x QDR single-port InfiniBand interconnect.

A.6 The Newton Method

Barrier Method. The barrier method [76] exploits a logarithmic function to incorporate the constraints into an extended objective function to solve:

$$B(x) = f(x) + r \sum_{i=1...6} -log(c(x_i)) \tag{A.1}$$

cf. barrier figure, where r is the barrier parameter. The barrier method is characterised by its requirement to arrive at the solution from within the feasible region as $B(x)$ is undefined outside of the feasible region.

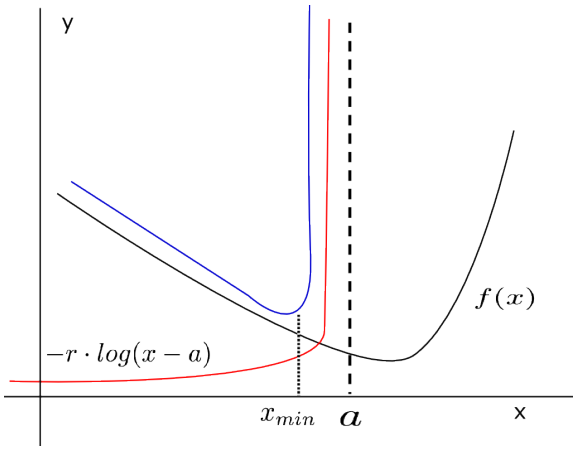


Figure A.14: Illustration of a 2D problem where the logarithmic barrier function (red line) is added to the objective function (black line) f(x) to define feasible region (red line) under the constraint a (dash line).

The barrier method inherits the property to arrive to the solution from within the feasible region. It is known [76] that there are cases that the use of barrier method is not preferred because it is not always possible to have a initial guess value that is within the feasible region. In the case of the distance problem an ideal initial guess value for the Newton method is the triangle barycentric centre. The method is prototyped on MATLAB, and then ported for C/C++ for performance testing.

In practice the use of Barrier method for solving constrained problems faces several computational difficulties [76]. The main performance downfall is due to the structure of the logarithmic barrier function and for the small values of the barrier parameter r that are required. The searching process is facing ill-conditioning effects

and round-off errors [76] when the search is approaching the boundary of the feasible region the logarithmic function becomes infinite. As the boundary of the feasible region is approached and because the search is in discrete Newton steps, a step may lead to a region outside the feasible region. An explicit check at every Newton step has to ensure that the value of each of the constraints has to be applied such that a re-positioning is guaranteed and the search always remains in the feasible region. The extra explicit check and correction on each search step doesn't help in the creation of an efficient solver. By explicitly checking the search position an increase of Newton iterations is unavoidable. Line searching algorithms [76] can be deployed to minimise the step size but that is another overhead to the Newton search.

Augmented Lagrange Method. The augmented Lagrange method is investigated due to its similarity to the penalty method. The Lagrange method in addition to a penalty-like function it also uses the Lagrange multipliers lambda. Augmented Lagrange multipliers augments the objectives function f and creates a new function that minimises L such that

$$L(x, \lambda) = f(x) + r \cdot \sum_{i=1...6} \lambda_i \cdot max(c(x_i))^2 \tag{A.2}$$

where *lambda* is a vector of size six which is the number of constraints. This implies that in order to minimise the problem it is required to solve $\nabla L(x, \lambda) = 0$. The multipliers *lambda* are updated using an additional iteration method at every set of Newton iterations to make the Newton search arrive to the solution exactly.

On the hand it Lagrange multipliers method is a widely used solver for many inequality constraint minimisation problems, but for the triangle-to-triangle minimum distance computation multipliers are not necessary required. Such a solver variant is not preferable because there is no need to have lambda multipliers as the problem is convex and simple to solve just by using penalty. The addition of extra unknown values to solve (a lambda for each constraint) per iteration requires unnecessary additional computation. On the other hand, the advantage of augmented Lagrange when compared to the barrier and penalty methods is that it converges to a solution always at the boundary of the constraints. Contradictory to the barrier method, the barrier method may only arrive to a solution from within the feasible region and constrained by a line search method to the solution. When the method is compared to penalty, the penalty arrives to the solution externally to the constrains due to the fact it uses a penalising function. The extra lambda solution required by augmented Lagrange do not pay off. This approach is inevitably the slowest method because of

the extra numerical operations.

List of Figures

List of Tables