Parallel local and implicit time stepping for triangulated rigid body dynamics

PETER JOHN NOBLE

Parallel local and implicit time stepping for triangulated rigid body dynamics

Peter Noble

A Thesis presented for the degree of
Doctor of Philosophy

Department of Computer Science
Durham University
United Kingdom
December 2023

Abstract

Discrete Element Methods (DEM), i.e. the simulation of many rigid particles, suffer from very stiff differential equations plus multiscale challenges in space and time. The particles move smoothly through space until they interact almost instantaneously due to collisions. Dense and dynamic particle packings require tiny time step sizes, while free and stationary particles can advance with large time steps. The ratio of different admissible time step sizes starts to span multiple orders of magnitude once we encounter quickly moving objects and particles that settle into dense packings before they distribute again. We propose an adaptive local time stepping algorithm which identifies clusters of particles that can advance independently of each other on-the-fly, advances them optimistically in time, determines collision time stamps in space-time such that we maximise the time step sizes used, resolves the momentum exchange in the collisions implicitly, and rolls back particles upon demand.

We also propose a family of novel multiscale collision detection and resolution algorithms that can be applied to triangulated objects within implicit time stepping methods. Inspired by multigrid methods and adaptive mesh refinement, we determine collision points iteratively over a resolution hierarchy. Coarse surrogate geometry representations identify an educated guess which triangle subsets of the next finer level might yield collisions. They prune the search tree and furthermore feed conservative contact force estimates into the iterative solve behind an implicit time stepping. Implicit time stepping and non-analytical shapes often yield prohibitive high compute cost for rigid body simulations. Our approach reduces the object-object comparison cost algorithmically by one to two orders of magnitude. It also exhibits high vectorisation efficiency due to its iterative nature.

The implicit solve of the actual collision equations avoids particles getting locked into tiny time step sizes, the clustering yields a high concurrency level, and the acceleration techniques in combination with the local time stepping avoid unnecessary computations. This brings a scaling, implicit adaptive time stepping for DEM for real-world challenges into reach.

Declaration

The work in this thesis is based on research carried out at the 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.

Acknowledgements

This PhD research was funded by the Engineering and Physical Sciences Research Council (EPSRC). I am grateful to the Durham University Department of Computer Science for access to the supercomputing resources used throughout this work and the support needed to attend conferences in Seattle and Amsterdam, which both personally and academically have been highlights of my studies.

I would like to thank Dr. Tobias Weinzierl for his constant conviction in the value of my work, encouragement exploring new ideas wherever our work took us and especially for the many hours on video calls he spent fostering the friendly and engaged atmosphere in the research group I have been a part of, even when we all had to be working remotely. Ideas discussed with my peers in our group catch-up calls regularly became the key insights to unlocking the next stage of my work.

I would also like to thank the mentors and friends I have had the great pleasure of meeting from my first day arriving at Collingwood College all the way through my 8 years studying here.

Finally, thank you to Emma and my parents for the examples they are to me and for their tireless support.

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

DEM	Discrete Element Method
BVH	Bounding Volume Hierarchy
SPH	Smooth Particle Hydrodynamics
OBB	Oriented Bounding Box
CCD	Continuous Collision Detection
GJK	Gilbert–Johnson–Keerthi. Convex shape intersection algorithm
FEM	Finite Element Method
SDF	Signed Distance Field
LCP	Linear Complementary Problems
SIMD	Single Instruction, Multiple Data
SSE	Streaming SIMD Extensions
AVX	Advanced Vector eXtensions
$\mathbb{T}(p,t)$	Triangle set of particle p at time t
$\mathbb{P}$	The set of all particles
ϵ	Region of length ϵ surrounding a particle
$B(p,t)$	A bounding geometry around particle p at time t
$n(c)/n$	The normal direction of contact c
K_s	Spring constant

τ	Torque
F	Force
ω	Angular velocity
L	Angular momentum
R	Rotation
P	Impulse
I	Inertia matrix
μ	Coefficient of friction
t	Time/time stamp
$\mathbb{T}$	Triangle set
ρ	Density
V	Volume
m	Mass
v	Velocity
r	A point (usually a contact) relative to the centre of mass
x	Position
a	Acceleration
$\mathcal{T}$	A surrogate hierarchy of triangles

CHAPTER 1

Introduction

In many industrial and scientific fields the simulation of rigid or incompressible bodies is required. The Discrete Element Method (DEM) was proposed [1] as a means of simulating millions of these objects and so study the properties of granular materials. The realism of such simulations relies on the ability to handle vast numbers of particles with varying shapes and sizes [2–14]. Complex natural physical properties, such as the separation of different scales of particles or blockages when particles lock together in narrow gaps, may not arise when using simple shapes or uniform sizes. This requires us to simulate complex shapes with significantly varying shapes, sizes and masses. Furthermore, to study the formation of such natural phenomena we are required to simulate systems over a relatively long time span. This requires simulations to be both efficient and stable over the desired span and through very dynamic configurations.

Two categories of scenario are typically studied using DEM codes and extensive literature exists comparing DEM methods to physical experiments [15].

The first is the more static packing scenarios, where the effect of properties such as size distribution, external pressure and shape of particles on the density or strength of a material. A number of different packing configurations are studied

including depositing particles in a container [16, 17], vibrating particles [18], piling particles in a loose pile [19–22], packing under compression [23] and a variety of experimental data confirms the effectiveness of these methods [24–26]. For particle sizes under $100\mu m$ non-contact forces such as Van der Waals can play an important part [27] but this has also been successfully modelled using DEM variants [28, 29].

The second is the more dynamic flow scenarios, where the effect of particle properties as well as properties of a container are studied for material flowing through pipes, hoppers or drums. A number of different flow configurations are studied in literature including flow within an enclosed volume [30–32], flow in an unconstrained container [33–35], flow due to vibration [36–38], flows through hoppers [4, 39–42], flows inside rotating drums [43–46] and again these types of method can be confirmed against experimental date [47–52]. Our work focuses exclusively on the category of flows over unconstrained containers (for example, only a floor plane to constrain the motion of particles) and flows through hoppers.

The challenges This work addresses two main challenges faced by DEM algorithms attempting to provide long running, stable and highly dynamic simulations using complex particle shapes.

1. Collision detection between complex particles can become prohibitively expensive

2. We would like to take large time steps but may be forced to take small time steps to remain stable

First, the collision detection between two complex objects can easily become prohibitively expensive. This is especially true when a fixed point method is used to resolve interactions between particles that requires repeated evaluation of the contacts between complex geometries. Second, in a highly dynamic scenario the time spans over which objects interact is very small compared to the time spans between interactions, which presents a problem for adaptive time stepping. At any given time in a simulation only a small fraction of objects might be interacting in a meaningful way so this is the only computation we wish to perform. Simultaneously

if we reduce the computation to only include the small number of interacting objects in a tiny window of time then we significantly reduce the potential for parallel work. In an age of many-core architectures the ability to utilise huge core counts is required for a method to scale competitively.

The first challenge of collision detection times dominating run time is reported by a variety of codes [3,5,6,53,54] and is made even more challenging when continuous collision detection is introduced [55–58] to identify (exactly or approximately) the time stamp of interactions between objects. We tighten the challenge and study a DEM prototype over particles where

1. rigid particles may have massively differing size
2. rigid particles are discretised by many triangles
3. rigid particles have complicated, non-convex shapes.

In addition to collision checks between individual pairs, DEM codes must also be able to deal with large variations in the sizes of objects. Efficient data structures for managing particles with orders of magnitude difference in size are essential for being able to simulate and understand scenarios of real world relevance [59].

The second challenge requires a DEM algorithm to be able to select a time step size that is as large as possible. However, this is prohibited, for global adaptive time stepping, by any individual interaction that requires a tiny time step size currently. When interactions between objects happen almost instantaneously the difference in desired time step size between an interacting pair of objects and 'free' objects can have orders of magnitude difference. Furthermore, by implementing implicit time stepping we increase the cost of resolving in-contact objects. While this cost may be, at least in part, mitigated by increased time step sizes it still leads to potentially very unbalanced workloads during individual time steps and so a further challenge in load balancing parallel work.

The idea of advancing a particle based simulation to the time of the next interaction of interest in an event-driven manor isn't a new one [60]. For a long time the idea of updating only the regions of the simulation domain where interactions occur has been explored [61] but for almost as long the issue of how to implement

an efficient parallel strategy for such simulations has been wrestled with [62]. This becomes particularly problematic if a strictly incompressible formulation is used for the particles and when a dense configurations of particles causes the time step size to 'collapse' towards zero.

Contribution This thesis presents contributions in three areas for the narrowed definition of the problem described above

Challenge	State of the art	Our contribution
Triangle-triangle distance checks are ill-suited to CPU architecture	1) Iterative distance minimisation with fallback [5] 2) Manually vectorised comparison based distance check [63]	A less computationally expensive iterative distance minimisation algorithm using the manually vectorised comparison based variant as the fallback Further described in Chapter 5
Collision detection (particularly for implicit methods) is prohibitively expensive	1) Guaranteed intersection free implicit rigid body simulation [55, 57] 2) On the fly construction and intersection tests for BVH-like data structure [64]	A multi-resolution iterative algorithm for merging and accelerating contact detection and resolution for complex meshes Further described in Chapter 6
Global adaptive time stepping is insufficient for reducing computational load for dynamic scenarios	1) Local time stepping SPH by discretising time step sizes over a grid [65] 2) Discretised grid based local time stepping for DEM [66]	A local time stepping algorithm capable of allowing individual particles to advance with the most suitable time step size, regardless of the time step sizes chosen by particles in other regions of the simulation Further described in Chapter 4

and includes results published here

P. J. Noble and T. Weinzierl, "A multiresolution discrete element method

for triangulated objects with implicit time stepping," *SIAM Journal on Scientific Computing*, vol. 44, no. 4, pp. A2121–A2149, 2022.

describing our work on a novel geometric multi-resolution method for accelerating fixed point implicit contact detection and resolution and being prepared to be published here

P. J. Noble and T. Weinzierl, "Parallel local time stepping for rigid bodies represented by triangulated meshes"

describing our work on a novel 'anarchic' local time stepping scheme to concentrate computational resources only in areas that require them.

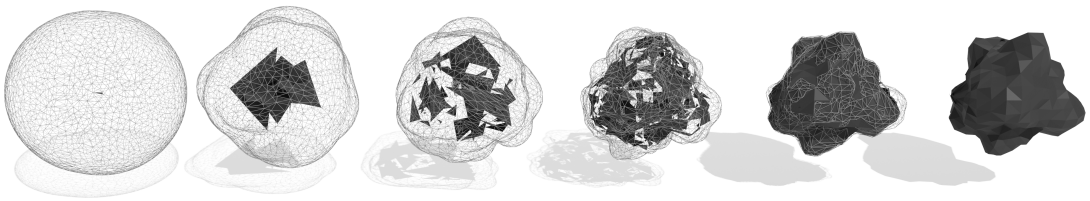


Figure 1.1: Our proposed surrogate triangle hierarchy, demonstrated on a 'bumped sphere', used for accelerating iterative collision resolution. Using each level of the geometric hierarchy yields meaningful approximations to contacts. This is useful for computing early cheap approximate solutions to speed up convergence. The actual sphere geometry $\mathbb{T}_h$ is shown on the right. From left to right: Finer and finer surrogate representations incl. their ϵ-environments. The coarser a representation, the lower the triangle count and the larger the corresponding ϵ. The surrogate triangles are weakly connected. The right most geometry is the actual particle with a closed surface.

Model We model objects as particles using triangles to represent the surface and supplement this surface with ϵ-area [2,9–11]. Contacts are defined as overlaps in this boundary regions and are represented as normal equipped points that are unique up to an ϵ-displacement. The interaction of two objects can produce more than one contact point. In the case of complex non-convex particles there may be many contact points. Per time step many contact points can exist making a large network of stiff interactions. Between time steps the arrangement and topology of this interaction graph can change significantly.

Using this model we propose two complementary novel methods (and a refinement of the previously proposed triangle-to-triangle penalty distance check method) that brings the large scale simulation of complex non-analytical rigid bodies with local implicit time stepping within reach.

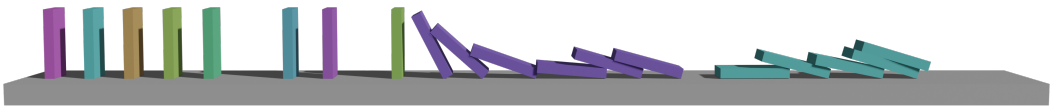


Figure 1.2: Dominoes where multiple pieces (particles) tumble and push each other over. As long as particles remain stationary, they can advance in time with large time steps, while the actual collisions happen on short time scales and change the particle topology (who interacts with whom). Different particle clusters which can use different time step sizes and advance in time independently are shown in different colours (This figure is illustrative and all clusters used a uniform time step size of 0.01s).

Core novel contribution 1: Novel local time stepping The first novel method we propose is a local time stepping scheme. Each particle has its own time stamps and may be able to advance in time independently of other particles.

Collisions are identified over a span of time using less detailed surrogate models to accelerate the identification of potential collision pairs. These cheap early checks are used to aggressively prune the graph of possible collisions. Next, particles are sorted into groups, called a cluster, where interactions occur such that each cluster is independent and can be processed in parallel. The particles in each cluster have their time stamps synchronised and, for this time step, will advance in-sync. Particles can belong to different clusters during different time step sizes and will be synchronised whenever required. Per cluster we compute a maximum admissible time step size such that contacts are generated but no physically invalid state is reached. A user defined maximum time step size and a requirement that time step sizes are positive are the only other requirements on the selected size. Therefore, clusters can advance through time 'anarchically'. In other words, the time step size of a cluster isn't required to be a multiple of the time step size of any other cluster. Clusters are marked for advancing if and only if we are able to roll back all time stamps to synchronise with any other particle in case new contacts are identified during the

current time span as part of the next time step iteration. Finally, once collisions and clusters have been identified the stiff series of contacts in each cluster that is marked for advancing is handled with an implicit contact resolution scheme.

To the best of our knowledge, our 'anarchic' local time stepping scheme, which greatly increases the efficiency of simulating highly dynamic scenarios, is unique among DEM codes. Chapter 4 presents the details of our algorithm.

Core novel contribution 2: Novel multi-scale implicit contact resolution The second novel method we propose is a multi-scale contact detection and resolution algorithm for accelerating a fixed point implicit scheme. Coarse surrogate representations of the particles are used to refine early approximations to the solution and therefore aggressively reduce the compute time.

Contact detection and resolution is phrased as an iterative algorithm that works through progressively less coarse surrogates of the particle mesh. We identify the following optimisations that take advantage of our method:

1. Distance checks between surrogates are used to prune 'no contact' configurations as early and as cheaply as possible. This is the same general method used by classic Bounding Volume Hierarchy (BVH) data structures [67–71] for accelerating spatial operations. If two parent volumes do not overlap then we know for certain that the child elements of these parents do not overlap. A description of our surrogate hierarchy data structure can be found in Chapter 5.

2. Another advantage associated with traversal of Bounding Volume Hierarchies, compared to plain level of detail approaches [72], is that only the areas of interest are expanded down to the fine resolution. When potential collisions are identified by overlapping parent volumes only the children of those nodes are required to be expanded. In the case where a single contact exists at the fine level and a single collision pair exists at each level of the surrogate tree then the cost of each iteration is constant compared to exponential when an N-to-N comparison is done instead.

3. The surrogate representations of the particle mesh yields contacts that result in conservative interactions compared to the corresponding contacts generated at the fine mesh level. We use these conservative estimates to inform early iterations in a fixed point implicit scheme. In other words, we permute the fixed point iterations and traversal of the mesh hierarchy compared to a traditional algorithm. Cheaper early approximate iterations result it fewer iterations at the fine resolution and so lower time to solution overall.

4. Comparison of triangles within nodes of the surrogate tree (including the fine resolution mesh) are phrased as a distance minimisation problem [4, 5]. A refined iterative algorithm is used, which approximates the Jacobian with a diagonal matrix and applies a fixed number of Newton iterations. This algorithm is efficiently vectorised. Our refined triangle-to-triangle distance algorithm is presented in Chapter 5.

5. The iterative distance minimisation algorithm may fail to converge in the small number of allocated iterations. For surrogate distance checks we can choose to skip these rare ill-posed configurations and continue as if there was a collision. Where we require a solution these failed cases are collected together and a manually vectorised version of the 'text book' comparison based distance check is used to guarantee a correct solution.

DEM simulations naturally lend themselves to algorithms that scale well due to the large number of elements and extremely short ranged interactions. However, this can become of limited importance when individual or small groups of interactions become so expensive that the rest of the simulation is held up by these. This is especially true when non-convex and densely triangulated particles with massively varying scales are interacting. Our hierarchical and iterative approach, with excellent vectorisation characteristics, brings implicit DEM simulations within practical reach and with iterative stages well suited to tuning a solution to a required accuracy (by early stopping as soon as the desired result is achieved) lays the foundations for algorithms that could handle effectively unlimited geometric detail when a bound on the required accuracy is given.

To the best of our knowledge, our loop permutation and fusion algorithm that combines ideas from traditional scientific and graphics fields, such as the Bounding Volume Hierarchy, approximate contact estimation and iterative triangle distance checks is unprecedented. Our novel algorithm is presented in Chapter 6.

Summary of our algorithm We phrase the traditional DEM algorithm in more challenging algorithmic language in three ways:

1. Particles are grouped together into clusters. However, the membership of a cluster can change within the span of time originally selected to be a time step. While one particle in free fall takes a single large time step other particles may take several small time steps, joining and leaving clusters, before being set on a new trajectory. Within a single time step and one cluster the movement and interaction of the particles are handled as if in total isolation to the rest of the simulation. This method avoids many globally small time steps [73] causes by as few as a single interaction requiring a small time step.

2. Each cluster is free to choose a step size freely. Since the difference between the time span of an interaction and the time span between interactions can be orders of magnitude methods that discretise the available time step sizes [65,66] fail to take full advantage of potential computational savings. An initial test is performed per cluster, using continuous collision detection, to optimistically estimate an admissible time step size. Where we later discover new possible contacts within the current time span or where a step size that was too large is selected, leading to an invalid physical state, the state of the particles in the cluster are rolled back to an earlier time stamp. This combination of techniques for handling particle clusters allows us to use tiny time step sizes where required without severely increasing the number of steps required for all other particles. We attempt to keep the number of clusters that can be advanced in parallel high by 'optimistically' advancing as close in time to the first invalid state we know of. A small number of roll backs are an acceptable cost in exchange for the high number of successful and parallel time steps for other clusters.

3. Contact detection and resolution is phrased as a multi-resolution problem. 'No contact' configurations are quickly identified and discarded. 'Potential contact' configurations have approximate contact resolution solutions computed, even before the fine resolution contact detection is completed. 'Fine resolution contacts' are used to refine an already close solution to the required accuracy. Furthermore, our multi-resolution surrogate mesh representation can be searched like a graph to accelerate continuous collision detection like a standard hierarchical spatial data structure. The contacts generated by the surrogate representations should provide conservative but accurate estimates of the reaction forces/impulses. The trade off for a more complex data structure is the ability to approximate solutions to contact detection and resolution - particularly valuable for fixed point implicit methods.

Summary of research questions and contributions We identify a number of challenges faced when using the DEM algorithm in highly dynamic scenarios using complex particles.

1. The time spans in which interactions between particles vary substantially and are typically orders of magnitude smaller than the time step sizes we would like to take for particles with no interactions.

2. It would be beneficial to use implicit time stepping to assist with the stability of these stiff simulations using larger time step sizes. However, contact detection within an iterative algorithm that refines an approximate solution but requires recalculating contacts each iteration quickly become prohibitively expensive for complex particle geometries.

The main contributions of this work can be summarised as:

1. A demonstration of the effectiveness of 'anarchic' local time stepping, where particles can independently advance in time without discretising the time intervals, when applied to dynamic physical scenarios.

2. A demonstration of the effectiveness of multiresolution techniques when applied to contact detection and resolution between triangulated particles by

decreasing the number of triangle-triangle distance comparisons using surrogate geometry inside a fixed point iterative scheme.

Shortcomings Beyond the field of DEM the algorithmic mindset of our method might have potential for other Lagrangian techniques involving stiff problems. However, the methodology of aggressively culling potential work to reduce the total computational load does yield a very hard dynamic load balancing challenge. This work focuses solely on single node shared memory systems when a work stealing scheduler can often cope but load balancing remains an ever present challenge as the number of cores grows. In a distributed memory system this challenge would be greatly amplified. Furthermore, for the purposes of developing novel time stepping schemes we implement only simple physical models. We see no fundamental obstacles to implementing our method in another code base, with more sophisticated physical models, but by its nature our algorithm touches many aspects of the classic DEM algorithm and so would require wide ranging changes to existing code bases.

Paradigm *Wherever possible avoid doing work (spatially and temporally) using cheap heuristics but where we need to do work create as much simultaneous work as possible by separating work units and starting with approximations and only refining to the required precision.*

We observe this pattern in each of the contributions of our work. First, when detecting contacts we use hierarchical data structures to aggressively prune (or avoid doing work) the search tree between two potentially colliding particles. Where regions with possible collisions are identified we bundle together as much simultaneous work as possible to do in rushes. The surrogate trees are constructed to contain 'buckets' of triangles at each node rather than individual triangles in order to increase the work done in these units of work. The number of triangles per bucket is experimentally determined to tune the balance between avoiding work where possible but doing as much at once in the cases where work must be done. Furthermore, triangle-triangle distance checks are phrased as a distance minimisation rather than a simple comparison based check. Per triangle pair this increases the amount of required computation but when processed in batches vectorisation allows us to achieve

a higher overall throughput. The criteria for determining convergence is adjusted to the required precision (determined by the user defined ϵ-region).

Second, our local time stepping scheme, and all the associated mechanics, was conceived in order to avoid taking more time steps per particle than necessary. In simulation time the less frequently we are required to update a particle the more computationally efficient the simulation is. However, at specific points computations are needed to produce the desired state change caused by interacting particles (without which our simulations cannot yield any relevant results). Therefore, we select any particle that might reach a valid state during this time step to advance in order to create as much simultaneous work as possible. The small risk of reaching an invalid state and having to be rolled back is outweighed by the advantage of keeping as much of the available processing power occupied with potentially useful work. Again the ϵ-region around each particle gives us a window of valid time for a contact to occur. To increase the temporal accuracy of the rigid contacts the ϵ-region can be tuned to be smaller.

Finally, our fixed point implicit contact detection and resolution algorithm can be thought of as avoiding unnecessary work by using surrogate representations of the particles for as many iterations as possible. At the same time we could view the whole algorithm as a single search down the surrogate tree and at each stage doing as much work as possible. By permuting the normal order of operations (where each iterations would invoke a full descent through the surrogate hierarchy) we only search through the surrogate hierarchy once. For each level of the surrogate hierarchy that is visited we perform many iterations of the contact resolution algorithm. This will involve repeated evaluations of the contacts between pairs of triangle buckets but doesn't explore other nodes of the surrogate tree at this point. These dense units of work are well suited to vectorisation and act on the same memory locations repeatedly, making better use of available fast cache memory. The convergence criteria for these inner iterations can be tuned to a user define threshold.

Delta is the project that this work is conceptually built on. The iterative triangle-triangle distance check described in our work is a refinement of the original method

proposed in Delta [4,74]. In addition to this Delta organises particles on an adaptive multiscale grid to efficiently compute potential particle pairs [75]. The source code for the original Delta project is available online [76].

Outline This thesis is organised as follows: We first outline the DEM algorithm and relevant technical background in Chapter 2 and identify where within the phases of the algorithm the challenges that we discuss are faced. Next, in Chapter 3, we review historical and state of the art methods for modelling particle geometry and contact behaviour. The modelling methods used throughout this work are highlighted.. Chapter 4 details the first of the two core contributions of this work. We first discuss the model used to represent a particle state over a short window of time, which enables our time stepping model. Next, a we describe a compatible broad phase collision detection algorithm, which identifies the particle clusters, the configuration and setup of the cluster and the calculation of an admissible time step size per cluster. To keep the simulation data consistent in time, we decide in which clusters we allow to advance in time, before we actually compute the particle interactions and allow them to progress in time. We also present results studying various scenarios that demonstrate the characteristics of the arising algorithm. In Chapter 5 an efficient contact detection between triangulated surfaces of two particles via a minimisation problem is introduced, before we rewrite the underlying geometric problem as a multiresolution challenge and introduce our notion of a surrogate data structure. In Chapter 6, we bring both the efficient triangle comparisons and the tree idea together, in the second of the core contributions, as we plug them into an implicit time stepping code, before we allow the non-linear equation system solvers' iterations to move up and down within the surrogate tree. We present some further numerical results to show the effectiveness of our multi-resolution approach. Finally, in Chapter 7 we close the discussion with a brief conclusion and an outlook.

CHAPTER 2

DEM algorithms

Many elements of the DEM algorithm are common across most implementations and have remained broadly the same since the method was proposed [1]. The stages of our algorithm fit the pattern of a standard DEM code but we make changes throughout all stages of the algorithm to accommodate the requirements of implicit and/or local time stepping. Therefore, this chapter first gives an overview of the wider DEM algorithm structure so we can identify the challenges faced as well as justifying changes we make to sections of the algorithm to support novel methods. Next we provide an overview of some background concepts for DEM and state-of-the-art methods that support or give context for the rest of our work.

The DEM algorithm is typically broken down into the following phases:

1. Broad phase collision detection
2. Time step selection
3. Narrow phase collision detection
4. Collision resolution
5. Integrate new state

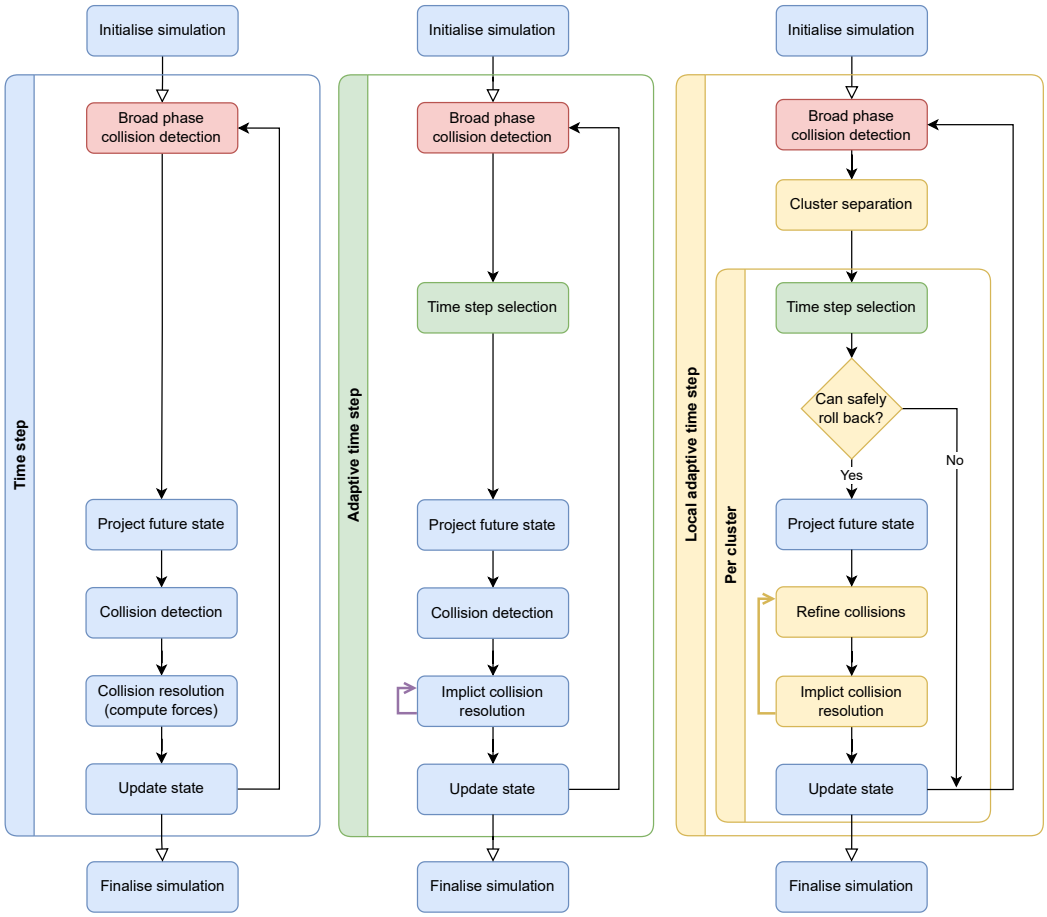


Figure 2.1: Phases of three variants of the DEM algorithm. Left - most simple, centre - adaptive time stepping with an iterative implicit scheme for resolving contacts, right - our method for local adaptive time stepping with fully implicit contact detection and resolution. Blue phases mark fundamental aspects of the algorithm. Red phases mark those required to implement even basic codes in practice. Green phases mark added steps to global adaptive time stepping. Purple indicates where an iterative algorithm or matrix solve can be added to solve contacts with an implicit scheme. Yellow indicates our contributions. Derivatives of this figure will be reintroduced at the beginning of future chapters to illustrate which areas of the DEM algorithm our contributions are made to.

While the broad phase collision detection isn't strictly required for a basic definition of the DEM algorithm (Figure 2.1 left) this optimisation is so ubiquitous and without it even 'toy examples' become intractable (for example, adopting a compare-all-to-all strategy results in $O(n^2)$ contacts for any scenario). Therefore, we include the broad phase collision detection phase in our outline discussion in this chapter. The elements and variants ((Figure 2.1) of the DEM algorithm presented in this work are again shown at the start of chapters 3-6 with the relevant areas

discussed in those chapters highlighted each time.

The broad phase collision detection phase is a way of rapidly identifying the potential collision pairs from a possible $O(n^2)$ pairs. This dramatically reduces the time spend doing collision detection using costly exact methods. The broad phase operates on a coarse simplification of the particles and involves a spatial partitioning data structure or an ordering of particles to organise the objects in space. A spatial partitioning approach divides up space into regions, which particles can then be sorted into. Perhaps the most basic example of a spatial partitioning data structure is a simple grid. Along each axis the 'world' is divided into equally sided buckets. Each object is sorted into one or more of these buckets. To gather a list of all possible collisions we inspect each cell for multiple occupants but also compare against the members of neighbouring cells. Any objects that don't occupy the same or adjacent cells cannot be interacting (as long as the size of particles is less than the size of grid cells). A more practical example of a spatial partitioning data structure when objects have varying size or are spread unevenly across a large volume of space is the Octree [77]. To construct an Octree all particles start grouped into a single cell, which is then repeatedly divided in two along each axis, and the particles contained in this cell are resorted into the child cells if they are small enough. To detect potential collisions only particles stored in the same cell, cells on the path back to the root or in cells neighbouring the cells on the path back to the root need to be compared against.

A popular alternative to the Octree is the Bounding Volume Hierarchy (BVH) [78, 79]. Usually the bounding volumes used are Axis Aligned Bounding Boxes, although compelling reasons exist of other options like Oriented Bounding Boxes (OBB) [80] in applications such as ray-tracing (to the best of our knowledge there is no proven benefit to using OBBs for collision detection) or simpler objects like spheres to reduce the cost of comparisons. To construct the hierarchy the total set of objects are recursively split in two (for example, by splitting the particles evenly across a plane positioned along a chosen axis) and the bounding volume of the resulting groups become the next two nodes in the tree. A similar traversal of

neighbours, ancestors and neighbours of ancestors to that used for Octrees can be used to detect potential collisions in a BVH. Alternatively, two separate particles, each equipped with a BVH, can be compared by recursively expanding branches of the hierarchy wherever potential collisions are found between the coarse bounding representations.

For triangulated or assemblies of primative objects the individual components (eg. triangles) could be embedded in the spatial data structure directly. This effectively rolls the broad and narrow phases into one.

Despite the drastic reduction in the number of collision checks needed it can still be very expensive to build and traverse these data structures. A popular optimisation is to maintain a neighbour list of each particle [81, 82]. At each step this list is used as a preliminary source of potential contacts. A criteria is chosen and if exceeded triggers a more expensive update to the list. An alternative optimisation is to build the BVH on the fly [64] based on the areas of geometry that potentially interact with other objects. This avoids the cost of building the data structure around particles that, for example, are very far from their nearest neighbours.

Time step selection is required for ensuring the stability, accuracy and computational efficiency of the simulation [65, 66, 73, 83]. If time step sizes that are too large are chosen then we would quickly observe missed collisions including intersections between particles. Stiff interactions between particles occur over very short spans of time compared to the time traversed between interactions. Even if no contacts were missed we might still see instabilities emerge from being unable to resolve the contacts. The most simple time step selection strategy would be to select a constant step size that is sufficiently small. To dynamically select a time step size (adaptive time stepping) continuous collision detection (CCD) is typically used to approximate a time step size [56, 84]. In the case of a DEM simulation this might including taking into account factors such as the first time of contact between two objects, the relative velocity with which the two objects are closing at the contact point, the relative tangential velocity at the point of contact and the length/depth of the contact.

Adaptive time stepping comes in two 'flavours'. Global adaptive time stepping selects a single time step size, which is used to advance every object in the simulation in lock step. Local adaptive time stepping selects multiple time step sizes or multiple sub steps, which are applied to subsets of the objects in the scene. Sub stepping seems to be the more common option where local adaptive time stepping is used, likely due to it being easier to implement.

Narrow phase collision detection is performed once a time step has been selected and the broad phase collision detection has yielded a 'short list' for potential collisions. Next, the exact contacts at the desired time stamp are computed. Depending on the contact model used these contacts are either equipped with a penetration depth or a minimum distance between the objects. Perhaps the best known option is the GKJ algorithm (see section 2.1.2), which results in contacts generated with a depth wherever two objects interpenetrate. Another option is to compute the overlap of each object's Minkowski sum with a sphere. The Minkowski sum can be defined as $\{a + b | a \in A, b \in B\}$, where A and B are the set of all points contained in the volume of the two objects. Or in other words, the total volume covered by sliding the centre of the sphere over every point in the object's volume. This boils down to a series of minimum distance checks between primitive objects and then comparing those distances to a radius.

At this point we are able to throw away the triangulated representation of the particles for this time step and instead work purely with the contact points and point representations of the particles.

Collision resolution or collision response is the phase where the force or momentum transfer between particles in contact is computed. In the case where contacts are represented as points equipped with a direction we can consider this phase to be solving a constraint problem over a graph. The nodes are particles with corresponding state. The edges are the contacts between objects and each is equipped with a normal direction. The relative velocities can be computed at the contact points. The solution is a series of non-negative magnitudes attached to the edges representing the magnitude of the response along that edge.

For a simple explicit scheme this phase might simply look like a single pass of the contacts computing a spring force for each. More sophisticated schemes like a fixed-point implicit solver might require many passes over the contacts or alternatively assembling a constraint matrix to numerically invert. A fixed point method requires an update, or residual, function to improve the approximate on each contact individually, on bundles of contacts or on all contacts simultaneously. A constraint matrix approach requires a method for encoding all the contacts - often encoded as Linear Complementary Problems (LCP) [85]. Friction proves particularly challenging in that case as the direction is dependant on the relative velocity at the point, which is in turn dependant on the state of the particles as it is being updated.

Integrate position is the final phase where the updating velocities are applied to the particles to advance them in time. The most simple example would be Euler integration but other options such as Verlet or Runge-Kutta integration are also available if higher order accuracy is required (see section 2.1.5).

2.1 Background

The following sections outline some core concepts and algorithms that can appear in DEM codes. Where we use, draw inspiration from or make observations about these methods in our work but where these details don't directly impact the core contributions of our work we comment on these in this section.

2.1.1 Inertia tensors for non-convex triangulated objects

To model the motion of an object subject to forces we require an inertia tensor to describe how these forces translate to changes in rotational momentum. This is analogous to how the mass of a point-particle is required to determine how a force translates to a change in momentum. Inertia tensors are also often called Mass Moment Of Inertia tensors (MMOI) or inertia matrices.

For an object with density ρ at a given point in its volume V the mass is given

as

$$m = \int \rho \, dV. \tag{2.1}$$

The centre of mass is given as

$$\vec{c} = \frac{1}{m} \int \vec{r} \rho \, dV. \tag{2.2}$$

The inertia tensor is give as

$$\boldsymbol{I}_0 = \int \begin{bmatrix} y^2 + z^2 & -xy & -xz \\ -xy & x^2 + z^2 & -yz \\ -xz & -yz & x^2 + y^2 \end{bmatrix} \rho \, dV. \tag{2.3}$$

An inertia tensor can be translated by $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ by adding

$$m \begin{bmatrix} y^2 + z^2 & -xy & -xz \\ -xy & x^2 + z^2 & -yz \\ -xz & -yz & x^2 + y^2 \end{bmatrix}, \tag{2.4}$$

and therefore, the inertia matrix about the centre of mass c is given by

$$\boldsymbol{I}_C = \boldsymbol{I}_0 - m \begin{bmatrix} c_y^2 + c_z^2 & -c_x c_y & -c_x c_z \\ -c_x c_y & c_x^2 + c_z^2 & -c_y c_z \\ -c_x c_z & -c_y c_z & c_x^2 + c_y^2 \end{bmatrix}. \tag{2.5}$$

Alternatively, in practice the coordinate system used to calculate $\boldsymbol{I}_0$ could be translated by $-\vec{c}$ and then calculate the inertia tensor with respect to the new coordinate systems origin. This results in the same solution but in our case having the geometry translated to be about the centre of mass is helpful when it comes to computing the results of iterations at contact points.

As the inertia tensor is simply the integral over the volume of an object, the inertia tensor of two connected objects can simply be summed to get the inertia

tensor of the whole assembly (as long as the same coordinate system was used to calculate each of the component inertia tensors). For any body there exists a set of orthogonal axis such that the inertia tensor computed relative to these axis will be a diagonal matrix. While this may provide a small memory saving we consider this to be negligible and the other advantages related to analysing physical systems don't apply to our iterative method so we store the inertia tensors of meshes as 3x3 matrices.

Another property to note is that for uniform densities we can simply factor out the density and multiply the result by this constant afterwards. This allows us to compute a unit density inertia tensor for each each mesh and then for each particle that uses that mesh we equip them with a density, which is then used to compute its own inertia tensor from the mesh's unit one. This allows us to vary the density and/or scale per particle that shares the same mesh without recomputing the inertia tensor.

For simple analytical shapes the inertia tensor can be directly computed. For example, for an solid sphere of radius r and mass m

$$\boldsymbol{I} = \begin{bmatrix} \frac{2}{5}mr^2 & 0 & 0 \\ 0 & \frac{2}{5}mr^2 & 0 \\ 0 & 0 & \frac{2}{5}mr^2 \end{bmatrix}, \tag{2.6}$$

or for a cuboid with width w, height h, depth d and mass m:

$$\boldsymbol{I} = \begin{bmatrix} \frac{1}{12}m(h^2+d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2+d^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2+h^2) \end{bmatrix}, \tag{2.7}$$

or a simple point with mass m and position $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$:

$$\boldsymbol{I} = m \begin{bmatrix} y^2+z^2 & -xy & -xz \\ -xy & x^2+z^2 & -yz \\ -xz & -yz & x^2+y^2 \end{bmatrix}. \tag{2.8}$$

It is also possible to derive the inertia tensor for a tetrahedron defined by the coordinates of its vertices [86].

Using those components we consider two approaches to computing the inertia tensor for a given mesh. First, since the inertia tensor of component elements can be summed and the inertia tensor of a sphere or cube is known and can be translated to any position we could pack the volume of a mesh with spheres or cubes and sum their inertia tensors. This would require a means of determining if a point is inside or outside the volume of the mesh. The result would be an approximation of the exact inertia tensor but increasing the number of sampled points would improve the estimate. Second, if we could decompose the mesh in tetrahedrons and sum the inertia tensors of each tetrahedron. This would require a means of decomposing a non-convex mesh. For our code both options would require calling out to external code to determine if a point is inside or outside the mesh or to decompose it to tetrahedron. Therefore, we choose to use the second method to obtain an exact result.

We can also calculate the inertia tensor of a very thin triangle defined by the points A, B and C with thickness ϵ and density ρ

$$\boldsymbol{I} = \frac{2 * \rho * area * \epsilon}{3}(\boldsymbol{I}_A + \boldsymbol{I}_B + \boldsymbol{I}_C), \tag{2.9}$$

where $\boldsymbol{I}_X$ is the inertia tensor of a point a position X with unit mass and

$$area = \frac{1}{2}|A \times B + B \times C + C \times A|. \tag{2.10}$$

Therefore, we can also calculate the inertia tensor of the mesh as if it is a thin shell by computing the inertia tensor of each triangle in the mesh individually and summing them. In addition to enabling thin shell objects this also allows us to (combined with our no-mesh-inter-penetrations method) model thin objects with just a single layer. For example, a playing card could be constructed with just a pair of triangles rather than a very thin cuboid with 12 triangles.

2.1.2 GJK algorithm

The GJK algorithm [87] can be used to efficiently determine if two convex shapes are intersecting. The algorithm works exclusively with convex shapes.

Our work presented in later chapters focuses exclusively on methods that require guaranteed intersection free states at all stages. Therefore, GJK can't be used for our work but it forms a very important basis for many codes doing collision detection between triangulated meshes and so we briefly outline the algorithm here for context.

A function called a Support Function is required for both objects. This function takes as input a direction and returns the point (Support Point) in the volume of the object that is furthest along the axis defined by the input direction. This point will always be on the surface of the object. While triangles are a popular choice for using with this algorithm they are in no way required. Any convex shape that can have a support function defined can be used, even if the two objects being tested have different underlying representations (eg. a triangle mesh and an analytical sphere).

Some important observations made by GJK are:

1. If $O \in \{a - b | a \in A, b \in B\}$ (called the Minkowski difference), where O is the origin, A contains all points in one object and B contains all points in the other object, then the objects are intersecting. This is equivalent to saying that an intersection exists if $\exists p$ st. $p \in A \wedge p \in B$.

2. The Minkowski difference between two sets of points that are convex is also itself convex.

3. For a given direction the support point of the Minkowski difference is the support point for that direction of the first object subtracted by the support

point of the opposite direction on the second object.

Therefore, the problem boils down to determining if the origin is within a convex shape given a support function. Using this information the GJK algorithm follows the steps outlined (for a 2D example):

1. Select an arbitrary starting direction and compute the support point of that direction for the first object and the support point of the opposite direction for the second object. Take the difference of the support points to get our first search point A.

2. Using the direction from A to the origin as our next direction, search for another Minkowski difference support point B along that direction. If this point passes the origin in the given direction then continue. Otherwise there is no intersection.

3. Using the direction perpendicular to AB that faces towards the origin as the next search direction find the next Minkowski difference support point C. Again, test if this passes the origin.

4. Test if the origin falls within the newly created triangle.

5. If the origin does fall within the triangle then select either AC or BC, selecting the side where the origin falls, and repeat from step 2 using these points as A and B.

2.1.3 Sequential impulses

Given a set of particles with contacts the Sequential Impulses algorithm [88, 89] is an efficient iterative method for determining the impulses that should be applied along the normals of the contacts (particularly in a real time environment).

The general structure of the algorithm follows four steps:

1. Using a prediction of the future state compute contact points

2. Apply external forces (for example, gravity)

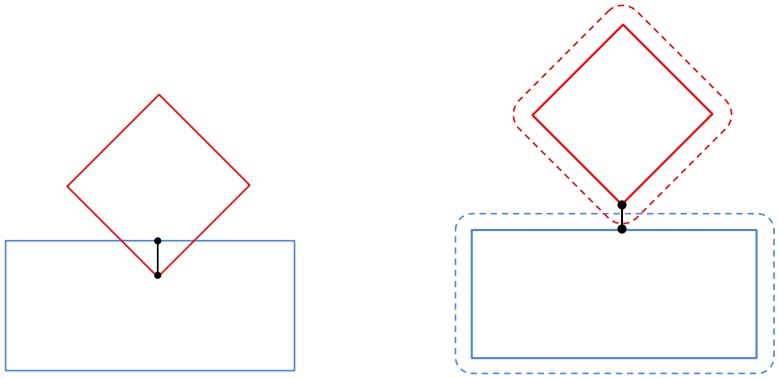


Figure 2.2: A single contact generated using a GJK-like method (left) and a Minkowski sum based method (right).

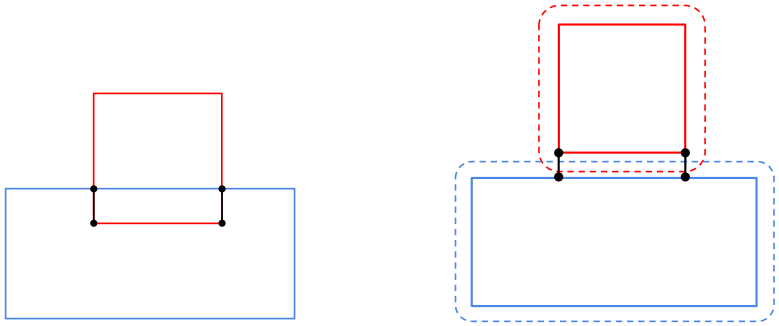


Figure 2.3: Multiple contact generated using a GJK-like method (left) and a Minkowski sum based method (right).

3. Compute contact impulses

4. Apply impulses and update positions

For each contact we can consider two constraints we may wish to enforce: Firstly, if c is the depth of each contact (such that $c < 0$ represents a penetration either of the methods or epsilon boundary depending on the contact generation methodology) we could enforce $c \geq 0$ for every contact. Secondly, if v is the relative velocity at the contact normal (such that $c \cdot n < 0$ represents a closing velocity while $c \cdot n > 0$ represents a separating velocity) we could enforce $c \cdot n \geq 0$ for every contact.

Using just the first constraint is known to lead to stability issues and complications arise when directly computing the updates positions of particles without computing the velocity. Instead the sequential impulses algorithm operates directly

on the velocity ($m\Delta v = P$ where P is impulse) to enforce the second constraint. On it's own this would require us to allow some penetration but we will later apply a corrective term. Our epsilon formulation of the contact detection allows for a small inter-penetration regardless.

If we consider the integral over time of Newton's second law of motion to update our velocity estimate value we get

$$v = \bar{v} - m^{-1} \int F \, dt, \tag{2.11}$$

and since impulse is the time integral of force we can rewrite this as

$$v = \bar{v} - m^{-1} P, \tag{2.12}$$

similarly we can apply the equivalent update to the rotational velocity

$$w = \bar{w} - I^{-1} r \times P. \tag{2.13}$$

We know that $\frac{P}{|P|} \cdot n = 1$ for any $|P| > 0$. We just need to determine the magnitude of P.

To iteratively apply the non-negative separation velocity constraint we compute the relative velocity at the contact point, the relative velocity along the normal and the tangential velocity:

$$v_r = v_2 + w_2 \times r_2 - v_1 - w_1 \times r_1, \tag{2.14}$$

$$v_n = (v_r \cdot n)n, \tag{2.15}$$

$$v_t = v_r - v_n. \tag{2.16}$$

Observation 1 *While it seems plausible that v_r could be calculated by a finite difference, between the contact point and the same point transformed to the previous time step, to reduce the required operations (three vector subtractions verses two cross products and three vector subtractions) we found this introduced a serious jitter.*

We apply the following update to the impulse applied by this contact

$$\Delta P_n = \frac{-v_n}{m_1^{-1} + m_2^{-1} + (I_1^{-1}(r_1 \times n) \times r_1 + I_2^{-1}(r_2 \times n) \times r_2) \cdot n}, \tag{2.17}$$

where the denominator is the effective total mass of the contact. In other words, the mass to use to solve $\Delta v = v_n = \frac{P}{m}$ as if this contact were a simple point-on-point contact (*'What impulse do I need to apply to this contact to result in zero closing velocity?'*).

This formulation will lead to a scheme that converges at zero relative velocity along the normal and so result in a perfectly damping collision. To introduce bouncing you would instead target $+v_n \cdot \alpha$, where $0 \leq \alpha \leq 1$ (to range from perfectly damping to perfectly bouncy).

While updating the impulse values we must allow negative impulse updates to be accounted for (for example, in a first iteration a too large impulse is applied so in the second iteration the value needs reducing to converge). However, it's important the total impulse applied along the normal never drops below zero so the total impulse is accumulated with a clamp

$$P_n^{i+1} = max(P_n^i + \Delta P_n^i, 0). \tag{2.18}$$

With this scheme we achieve the desired results only if the time step size is selected perfectly for every collision. Otherwise collision will result in cancelled velocity but some overlap. Over multiple interactions this overlap may grow and result in intersecting geometry. To correct for this we introduce a tiny position update for any contact where $|n| > \delta_{slop}$ where δ_{slop} is a small constant used to prevent one particles on top of another being pushed apart and then gaining a small downward velocity in the next step only to have it immediately cancelled (results in jittering).

Using the computed impulse along the normal direction we compute the tangential friction impulse using a similar method to the normal impulse but using the tangential velocity v_t, an effective mass calculated from the normalised tangent t and updated clamping $|P_t| \leq \mu P_n$. μ represents the coefficient of friction for this

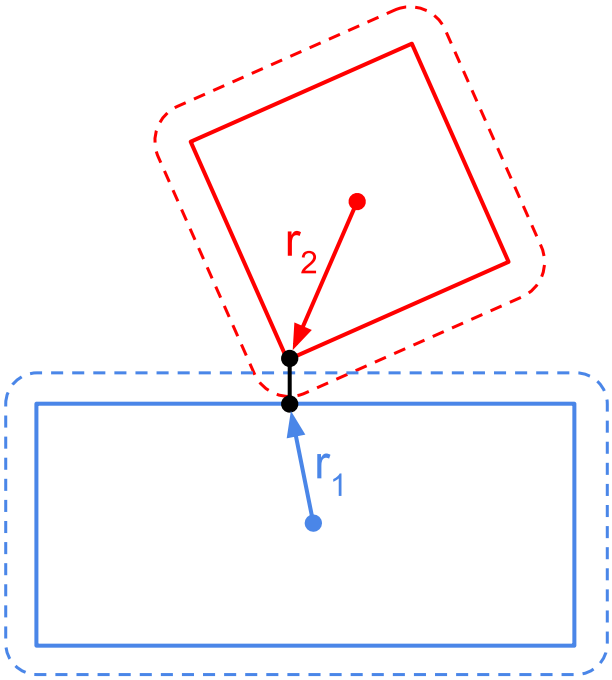


Figure 2.4: A contact and the associated contact direction vectors (r_1 and r_2).

contact. Computing μ is a complex task (and open question) and depends on the material properties. Accurately modelling this is beyond the scope of our work and we simply assign a μ value to each object and take the minimum of each to be the value used by the contact.

$$P_u^i = P_t^i - \frac{v_t^i}{m_1^{-1} + m_2^{-1} + (I_1^{-1}(r_1 \times n) \times r_1 + I_2^{-1}(r_2 \times n) \times r_2) \cdot t}, \tag{2.19}$$

$$P_t^{i+1} = clamp\_magnitude(P_u^i, \mu |P_n^i|). \tag{2.20}$$

In our work we ignore twisting frictional impulses. For example, the resistance experienced at the tip of a spinning top against a table surface so in our code a spinning top initialised in a sufficiently stable state so as to only ever produce one contact point with the table will continue to spin forever [90]. Further terms such as the twisting frictional force could be added at this point to improve the physical accuracy of the system but as this is beyond the scope of our work we

simply add a tiny damping factor to the overall energy possessed by each particle (maybe justified as air resistance), which doesn't noticeably affect most scenarios but prevents perpetual motion.

2.1.4 Graph colouring

Many of the algorithms utilised by DEM involve an interaction graph where each element requires some update but is dependant on neighbouring elements.

One method for being able to complete this work in parallel would be to equip each node in the graph with a mutex and proceed through the computation with a simple parallel for loops while locking nodes as their values are needed. In some early testing we observe order of magnitudes decrease in speed for single core execution (therefore guaranteeing no contention of resources).

Alternatively, the nodes of the graph could be coloured such that no neighbouring pair of nodes share the same colour. As such, work on any set of nodes with the same colour can be performed concurrently [91].

Perhaps the most simple and best know algorithm for graph colouring is the greedy algorithm. The algorithm is guaranteed to colour the graph with $max_i(D_i)+1$ colours (where $d(i)$ is the degree of node i) and does so in $O(n * (max_i(d(i)) + 1))$ time. Reduced numbers of colours result in a larger average number of nodes per colour

A simple variation of the algorithm [92] sorts the nodes by degree. This variation reduces the maximum number of colours to $max_i(min(d(i) + 1, i))$ with one additional sort.

This algorithm colours the nodes of the graph. However, we often want to perform operations on the edges of the graph where the resources for the computation are held on the nodes. In this case we take the dual of the graph and apply the same colouring algorithm. This way tasks operating on edges of the same colour can access both connecting nodes without conflicts.

Later when we come to implement graph colouring in our work where edges represent contacts between particles it's common to find multiple edges between nodes (for example, a cube resting on the ground might have a contact point in each cor-

Algorithm 1 Welsh-Powell greedy graph colouring algorithm

Input
G = (V, E)
Output
Colours over vertices C

$G_s(V_s, E_s) \leftarrow sortByDegree(G)$
$avail \leftarrow \{True_1, True_2, ..., True_{d(0)}\}$
$C \leftarrow \{-1_1, -1_2, ..., -1_{|G|}\}$
$C(0) \leftarrow 0$
for $i \leftarrow 1, ..., |G_s|$ **do**
    for $e \in E(i)$ **do**
        $avail(e) \leftarrow False$
    end for
    for $a \leftarrow 1, ..., |avail|$ **do**
        if $avail(a)$ **then**
            $C(i) \leftarrow a$
            break
        end if
    end for
    for $e \in E(i)$ **do**
        $avail(e) \leftarrow True$
    end for
end for
return C

ner). In this case every edge would end up with a different colour assigned to it and result in large numbers of small groups of colours (bad for efficient parallelisation). Edges that share common endpoints can be collected together into bundles and treated as a single edge for colouring. We also observe that this slightly reduces the number of required iterations in stable states for the sequential impulse solver by processing all the contacts in a bundle before applying. This is likely because stable states result from multiple balanced contacts. If the contacts are applied sequentially then the stable state is upset to begin with and then converges back towards a stable state. If the contacts are applied simultaneously then they remain balanced and while the magnitude may over or undershoot to begin with it will converge to the stable state while avoiding introducing unnecessary rotations.

2.1.5 Integration in time

To compute the motion of particles over time and under various forces we must numerically solve ordinary differential equations (ODEs). For simplicity, here we just consider systems with particles represented by point masses. Therefore, we use

Newton's second law of motion to govern how a particle moves under a given force

$$m\frac{dv}{dt} = F, \quad \frac{dx}{dt} = \mathbf{v}, \tag{2.21}$$

where m, v, and x are the mass, velocity, and position of particle respectively, and F is the net force acting on it. The choice of time integration method must balance several factors:

Accuracy: The degree to which the numerical solution approximates the exact solution.

Stability: The ability of the numerical method to produce bounded solutions over long integration times.

Computational Efficiency: The computational cost per time step.

Euler Method is the most simple explicit method. Explicit methods compute the next state of the system based only on the state at the current or previous time steps. The Euler method is generally simple to implement and computationally inexpensive per time step. New velocity and position states are computed using

$$v^{n+1} = v^n + \frac{\Delta t}{m}F^n, \quad x^{n+1} = x^n + \Delta t v^n, \tag{2.22}$$

where Δt is the time step size and the superscript n denotes the current time step.

The Euler method is a first-order method, meaning the error is proportional to the time step size $O(\Delta t)$.

Verlet Integration , like the Euler method, is an explicit method but uses the positions from the two previous time steps and doesn't explicitly compute velocity but instead uses a previous position

$$x^{n+1} = 2x^n - x^{n-1} + \frac{\Delta t^2}{m}F^n. \tag{2.23}$$

The Verlet method preserves total energy over long simulations, allows for relatively larger time steps compared to explicit Euler while still only requiring one force evaluation per time step. Unlike the Euler method, this method is second-order and

so the error is proportional to $O(\Delta t^2)$. The position from the last two time steps much be stored, potentially increasing memory usage.

Explicit Runge-Kutta uses multiple intermediate steps for each time step to achieve higher order accuracy. Runge-Kutta requires evaluating the force multiple times at different times and with different particle positions. We denote this as $F(t, x)$ for the force on a particle at time t with position x. For a fourth-order scheme with intermediate positions k_x^i and velocities k_v^i (for step i) the intermediate steps are:

Intermediate step 1:

$$k_x^1 = \Delta t v^n \quad \text{and} \tag{2.24}$$

$$k_v^1 = \Delta t \frac{F(t^n, x^n)}{m}. \tag{2.25}$$

Intermediate step 2:

$$k_x^2 = \Delta t\big(v^n + \frac{1}{2}k_v^1\big) \quad \text{and} \tag{2.26}$$

$$k_x^2 = \Delta t \frac{F(t^n + \frac{\Delta t}{2}, x^n + \frac{1}{2}k_x^1)}{m}. \tag{2.27}$$

Intermediate step 3:

$$k_x^3 = \Delta t\big(v^n + \frac{1}{2}k_v^2\big) \quad \text{and} \tag{2.28}$$

$$k_x^3 = \Delta t \frac{F(t^n + \frac{\Delta t}{2}, x^n + \frac{1}{2}k_x^2)}{m}. \tag{2.29}$$

Intermediate step 4:

$$k_x^4 = \Delta t\big(v^n + k_v^3\big) \quad \text{and} \tag{2.30}$$

$$k_x^4 = \Delta t \frac{F(t^n + \Delta t, x^n + k_x^3)}{m}. \tag{2.31}$$

And finally the new position and velocity are calculated:

$$x^{n+1} = x^n + \frac{1}{6}(k_x^1 + 2k_x^2 + 2k_x^3 + k_x^4) \quad \text{and} \tag{2.32}$$

$$v^{n+1} = v^n + \frac{1}{6}(k_v^1 + 2k_v^2 + 2k_v^3 + k_v^4). \tag{2.33}$$

Intuitively, step 1 uses the Euler method an approximation of the final state and

then steps 2 and 3 use the current estimate to reevaluate the force at the centre point of the time step. The final step takes the current estimate to the end of the range before all a weighted average of all the estimates are used to compute the final position.

While Runge-Kutta achieved higher order accuracy it requires the force on particles to be evaluated multiple times, which is potentially very costly. Furthermore, for the simulation to remain stable (when using stiff forces) the time step may still need to be very small.

Backward Euler Method is an implicit method. Implicit methods compute the system's future state by solving equations or iteratively approximate both current and future information. They are generally unconditionally stable for linear problems but require solving algebraic (often nonlinear) equations at each time step. The implicit Euler method updates the velocities and positions using

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \frac{\Delta t}{m}\mathbf{F}^{n+1} \quad \text{and} \tag{2.34}$$

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{v}^{n+1}. \tag{2.35}$$

This requires solving for $\mathbf{v}_i^{n+1}$ and $\mathbf{x}^{n+1}$ simultaneously.

Solving this system of equations is a lot more computationally intensive but the greater stability allows for larger time steps.

Newton-Raphson method is an iterative technique for solving nonlinear algebraic equations arising in implicit time integration methods. By rearranging the implicit equations into a new function f such that we have a root finding problem then this method can be applied by repeatedly applying the following update:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \tag{2.36}$$

where x_i is the current estimate of the solution at iteration i. The function f may be a truncated version of the taylor expansion of the function. This iterative step is repeated until the solution converges, which can be very computationally

expensive.

CHAPTER 3

Physical Model

In order to simulate physical phenomena a level of abstraction is required to model physical reality. Decisions about this representation impact the algorithms used to simulate a system over time. Therefore, this chapter outlines the method used to represent the shape of particles (Section 3.1), how contacts between particles are represented (Section 3.2), how we compute interactions as forces using a spring model (Section 3.3.1), how we compute interactions as impulses (Section 3.3.2) and how friction forces/impulses are modelled (Section 3.4). Throughout this chapter we draw on existing literature to attempt to contextualise the representations and methods chosen within the state of the art and highlight where our work advances the field.

While we use a relatively simple set of physical modelling ideas it should be possible to apply the rest of our work to contexts where the following basic physics elements have been replaced by more sophisticated ones.

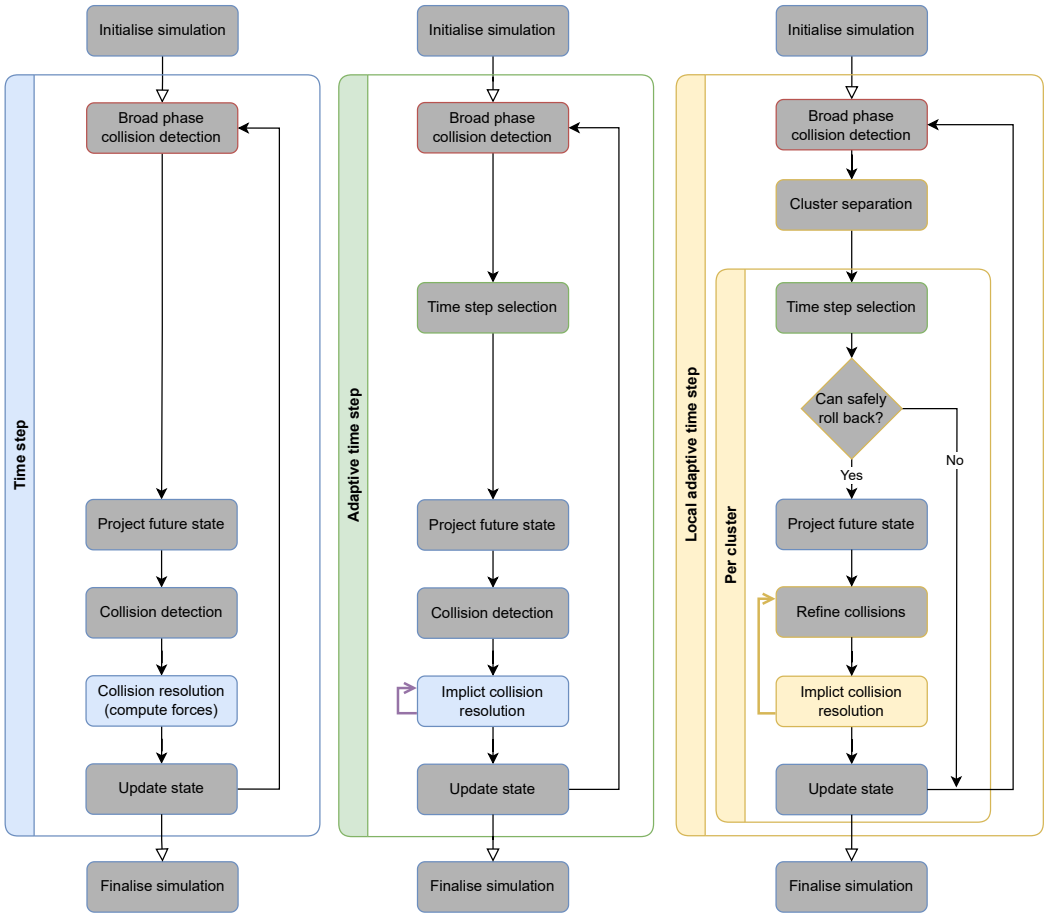


Figure 3.1: The phases of the algorithm that are discussed in this chapter are shown in colour.

3.1 Geometric Representation

In order to simulate the motion of particles we require a method to model their surfaces. How we choose to represent these surfaces can have a significant impact on the accuracy of the simulation [93, 94] as well as the run time cost of performing operations (such as collision detection).

Definition 1 (Geometric representation) *The physical shape and size of each particle is described such that, combined with the current state, geometric checks can be performed, such as collision detection.*

A repeated paradigm here is the idea of combining simple shapes into assemblies. To avoid an explosion in possible combinations of primitive objects requiring collision checks these simple shapes are embedded into a spatial data structure. Already we

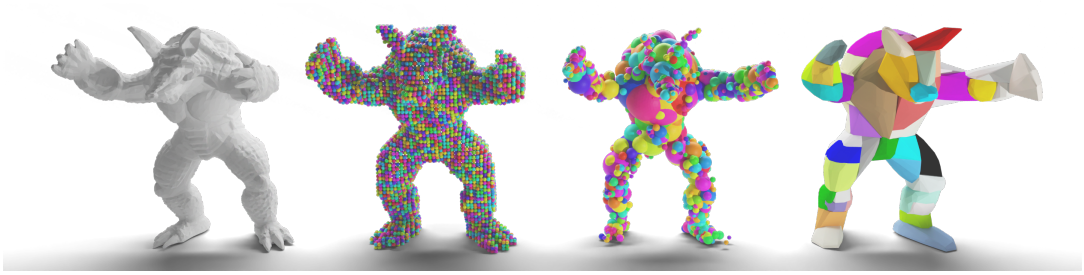


Figure 3.2: A triangulated armadillo mesh (left) decomposed into assemblies of 10,500 uniform small spheres (centre left), 900 scaled spheres (centre right) and an 68 convex hulls (right).

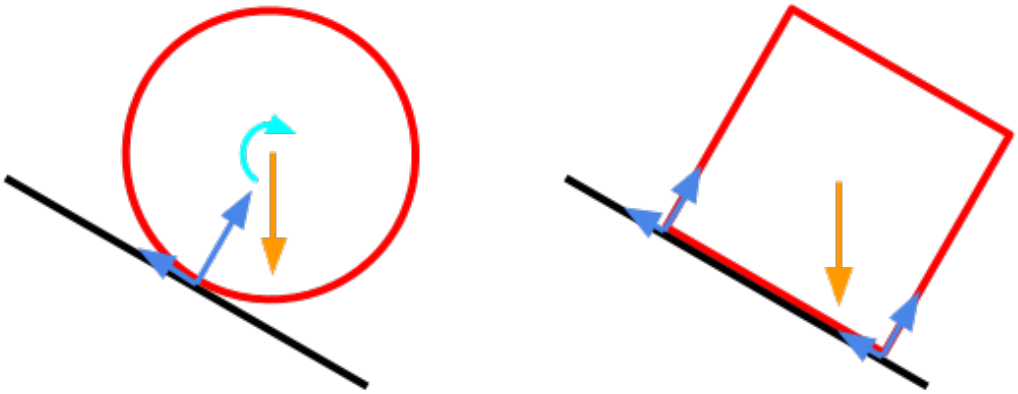


Figure 3.3: A demonstration of the impact of the geometric representation chosen. With a simple model of forces, a spherical particle (left) on a slope will begin to roll while a rectangular particle (right) may stay stationary. Orange indicates the force due to gravity. Blue arrows indicate the normal and friction forces from contacts with the surface. Cyan shows the total torque applied to the object.

see hierarchical data structures for algorithmically improving spatial queries but also the opportunity for multi-resolution data structures used to accelerate iterative algorithms. We take advantage of a similar acceleration data structure with multi-resolution properties to implement our implicit time stepping algorithm in chapter 6.

A wide variety of methods exist and all, to some extent, trade off between accuracy and computational efficiency [13, 78]. We consider a few of the most popular. Broadly working from the most simple and computationally cheap (but geometrically the least accurate) to towards the more complex and computationally expensive (but geometrically more accurate):

Analytical representations are perhaps the most commonly used. The most simple being a sphere. Along with the transformation of the object each one simply needs a radius to fully describe the geometry. Collision checks become a simple distance calculation between points and a comparison to the sum of the radii computation. Sphere based particles offer a simple solution and provides not just cheap collision detection but a deterministic cost for collision detection (good for load balancing) and no branching (good for CPU pipeline usage and vectorisation). However, there are limitation to what a sphere based particle can represent. Consider a simple scenario with a grain of sand on a gently sloped surface (Figure 3.3). In reality the friction between the grain and the surface will cause the grain to remain stationary and the irregular shape of the grain will prevent it from beginning to roll. When we try to simulate this using a simple sphere the friction will hold the grain in place to begin with but the particle will begin to roll and so move along the surface. Additional effects, such as modelling rolling friction [44, 95], can be used to emulate the same outcome but requires explicit modelling and calibration to be effective.

More complex analytical shapes can be used (ellipsoids, superquadratics etc) to counteract some of the shortcomings of the sphere. While these methods offer greater geometric accuracy for the particles they are still heavily limited in what they are able to model. While some of the benefits of the cheap analytical collision detection remain there is always a trade off. More accurate representations lead to more complex systems that must be solved to analytically determine contacts.

Superquadratics use 5 parameters to describe the shape of an object (length along each axis and two blockiness parameters) [96] and have been shown to be an effective balance between simplicity and the range of shapes that can be described [97–99]. This model can be used to describe spheres, disks, ellipses, cylinder-like and box-like objects. Analytical solutions exist for calculating the inertia matrix, bounding sphere and oriented bounding box of a superquadratic surface [100]. However, to calculate contacts an iterative algorithm is required. This method demonstrates increasingly slow convergence for high blockiness factors. Open source implementations of these algorithms are readily available [100]

A spherical harmonic approach [101, 102] is another a specific example of a more

complex analytical method. A spherical harmonic function is defined over a sphere and could be considered analogous to a Fourier transform, where instead of coefficients weighting a sum of cosines instead coefficients weight the sum of smooth functions over a sphere. The surface of an object is defined by offsetting each point on the surface of a sphere by the value of the spherical harmonic at that point. This limits the shapes that can be represented to "star like" objects, which is defined as a shape where any half line between the centre of the object and any point on the surface passes through the surface exactly once.

The level of detail represented can be tuned by adjusting to degree of the spherical harmonic to include greater or fewer numbers of coefficients. Editing these coefficients presents the opportunity to model behaviour such as the surface being eroded away [103]. Furthermore, these coefficients are inherently linked to rotations making rotations

Collision detection can be handled by looking for the overlap between the bounding spheres of two objects and then sampling the spherical harmonic functions along the surfaces of the overlapping regions of the bounding sphere. The smooth nature of the functions allows the volume of the overlap to be efficiently numerically calculated.

Level Sets (LS), Signed Distance Functions and Signed Distance Fields (SDF) represent a very different approach to modelling particle shapes. In literature Level Sets can be continuous functions defined over each point in space and take a positive value when outside a given shape and a negative value when inside that shape [104] (or Potential Particle [105]). Alternatively, a Level Set can refer to a sampled set of points where the value of each point is positive if outside the volume of the particle, negative for inside and zero if the point is exactly on the surface [106]. In addition to the signed property of the function, the gradient at any point always points towards a surface of the particle. The Signed Distance variants of Level sets are the Signed Distance Function and Signed Distance Field. A signed distance function is defined over all points in space as the signed orthogonal (or shortest) distance from that point to the closest point on the boundary of a set

ω. ω contains all points in space within the volume of an object. The function is 'signed' because points inside ω result in positive values, points on the boundary of ω have a zero value and point on the outside of ω have negative values. The gradient of the function at a point is the direction of the nearest boundary point. A Signed Distance Field samples a series (normally a grid) of points and for each stores the orthogonal distance to the closest boundary. Usually a triangle (or polygonal) mesh is used as the input to generate the signed distance field.

Like the other analytical methods, Signed Distance Functions suffer from the limits of assembling complex geometries from more simple analytical shapes and the growing cost of evaluating them. Signed distance fields [107] have a different trade off. In its most basic form the signed distance field is a grid of sampled points. Therefore, the accuracy of this representation is determined by the density of sampled points and the density of queries made. If the SDF is generated from a triangle mesh then for a given density of sampled points it doesn't make a difference to the final size/resolution of the model if the input had 10 or 1,000,000 triangles. Alternatively, the SDF might come from another source such as a CT scan [108]. This makes this technique popular in real time applications [109] where a strict time allocation for collision detection must be adhered to. The density of sampled points correlates to the size of features that can be resolved. With an insufficient number of points sharp features will appear rounded and thin features may disappear altogether. One option for accurately representing fine details using SDFs without the number of required sampling points becoming unreasonable is to hierarchically subdivide the grid points are sampled on [110].

While point-SDF look-ups are simple (to the point of being trivial) it isn't immediately clear how this should be extended to surface-SDF distance checks. For example, for a sphere intersecting a box we can sample the vertex positions of the sphere and observe that they are within the volume of the box and so generate the corresponding contacts. However, in the case where two long cylinders made of a pair of connected circles are overlapped in a cross shaped configuration there are no vertices on either mesh that fall within the volume of the other. To be able to handle these cases we need an additional step. One solution is to uniformly scatter points

over the surface of each particle and then sample those points in the SDF of the other particle [111]. Alternatively, employ a root finding method over the surface of interest to find a point with minimum value in the SDF of the other particle [112] or take the two fields together and find the root representing the mid point between the surfaces [105].

Another modification of this method allows for point-SDF continuous collision detection and so could be used to accelerate the time step selection phase [113].

Polygon meshes are the method that this work utilises. When handling geometric operations polygons are typically decomposed into triangles. We therefore focus solely on triangle meshes.

While polygon meshes could be considered similar to analytical assemblies (in other words, an assembly of individual polygons) they are typically categorised and compared as different classes of method [114, 115]. An important difference is that an analytical assembly represents a set of overlapping volumes - each element has its own volume and overlaps neighbours. By comparison a polygon mesh encloses a volume with a set of surface elements - each element represents a patch of surface and is connected to neighbours.

An alternative collision detection method for triangulated meshes is to take the Minkowski sum of the mesh and a sphere (sphere swept volume) and then intersect the resulting volumes with each other. In other words, compute the minimum distance between two meshes and then compare this distance to the sum of the radii. The advantages of this method are that contacts are all handled without the underlying meshes intersecting and that the collision detection problem can be phrased as a minimum distance problem, which is perhaps simpler than the problem of calculating the penetration depth and direction of intersecting meshes. The major disadvantage is that contacts cease having a well defined direction as soon as there is any intersection of the meshes. Unlike the signed distance or analytical approaches we also have no concept of the inside or outside of a volume. A triangle mesh only encodes the boundary of the volume. Further computation can be done (casting rays out from a point) to determine if the point is inside or outside the mesh but

this isn't explicitly represented by this model. This means that any step in which a particle moves entirely through a surface into the interior of an object could be missed by this technique whereas it could be corrected using another method.

Polygon meshes represent an opportunity for editing the particle shape in response to interactions with other particles. For example, sharp edges could be smoothed to simulate abrasion between sharp grains or sections could be removed/separated to simulate chipping from sharp grains.

Assemblies can be constructed using simpler forms combined together to represent more complex objects [116]. For an analytical assemblies spheres are commonly used (Figure 3.2 centre right). One method for constructing these assemblies is to pack the interior of a mesh with spheres. The accuracy of this method depends on the limit for how small the spheres are. Sharp edges still cannot be exactly modelled by this method but it may be possible to reduce the geometric errors of such features to the point where the geometric representation is no longer the most significant factor in the simulations accuracy. As the accuracy of this method is increased the number of spheres increases and so the computational cost increases too. However, it has been shown that not only sphericity (how close an object is to being a sphere overall) but also the roundness (the curvature of each individual corner) can have an effect on the mechanical properties of a granulate material, potentially making simulations using sphere-assemblies pack much closer without as much external pressure.

Algorithms for efficiently computing sphere assemblies are known and well understood [117–120] and open source implementations are available [121]. However, it may not be adequate to simply divide up an object into a set of sub-objects and apply the same contact model when multiple contacts exist between objects since the response can be over-stiff or over-damped [122]. Furthermore, it has been shown that dividing more complex shapes, such as an ellipse, into spheres can significantly increase the computational cost of contact detection [123] and that to match experimental results it is important to use a sufficiently large number of sphere to represent objects such as cylinders [124].

A popular method of performing collision detection using triangulated meshes is the GJK algorithm (Appendix 2.1.2). However, this has two drawbacks. First, the algorithm only works with convex geometry. This is typically overcome by creating assemblies of convex shapes (Figure 3.2 right). Second, the algorithm detects penetrations between geometries. Collision resolution techniques based on GJK often focus on minimising this penetration depth rather than guaranteeing all states being intersection free.

An additional use for particle assemblies is for modelling fracturing particles. Two main methods exist in literature. The first is to model particles as a single shape but when a sufficiently large force (or shear force, for example) is applied to the particle it is replaced with a cluster of smaller particles [125–127]. When using simple shapes, such as spheres, an issue can arise with volume preservation. Replacing one sphere with, for example, 2 spheres, where each of the new spheres is fully contained within the volume of the original sphere will result in a volume loss. Using polygonal cells would prevent this issue. Additionally, computing the fracture pattern caused by a given force requires further computation. An alternative method is to represent one larger breakable particle with an assembly of smaller particles (or 'Cells') that are bonded together with additional attraction forces called the Bonded Cell Method (BCM) [43, 128–131]. These internal forces are equipped with an additional breaking threshold and once this threshold is reached the bond is removed.

Tetrahedral meshes Similarly to the polygon meshes, tetrahedral meshes use connected vertices but by utilising a tetrahedron, which has volume, a more explicit representation of the interior of objects can be achieved. The first step in constructing a tetrahedral mesh might be to create the surface polygon mesh.

This representation might be required as an intermediate step to compute the inertia matrix (Appendix 2.1.1) by first calculating the inertia matrix for each individual tetrahedron even when a polygon mesh is used for contact detection. In this case the intermediate mesh can be discarded once the inertia matrix is computed.

A possible use case for this representation is when the finite-element-method

(FEM) is used to simulate deformable objects [132]. It has been shown that popular contact models used in DEM and molecular dynamics can miss important features when the deformation of the underlying object plays an important role in the interaction [133]. Specifically the energy loss as two particles rebound off each other. However, using FEM models it can also be shown that (for spheres at least) the accurate energy loss can be calculated using parameters such as the relative velocity, Young's modulus and yield stress of the material and applied to a rigid body simulation [134].

Summary given in approximately ascending order of complexity (and therefore approximately inverse ordering for execution speed) and in relation to the method we employ:

	Primitive	**Assembly**
Analytical	**Spheres** Collision checks (including continuous) are simple and inexpensive. However, the accuracy is poor for representing any objects with features that might prevent rolling. We utilise this representation for filtering potential contacts. **Ellipses and more complex shapes** Some of the short comings of using spheres can be countered by using more complex analytical shapes. This comes with the disadvantage of more complex and more expensive routines for collision detection. We don't utilise this representation in our work.	**Analytical assemblies** approximate the volume of an object using many simpler shapes. We use a similar data structure to accelerate geometric queries (Section 4.3.

Signed Distance	**Signed Distance Fields** enable simple and cheap geometric distance checks. Like analytical methods this method can be made robust to interpenetrations. We don't use any methods such as this in our work.	**Hierarchical Signed Distance Fields** present an option for combining the benefits of SDF query speed with the benefits of a hierarchical data structure compared to a densely sampled grid. Furthermore, a hierarchical representation storing progressively more coefficients of a polynomial representation of the surface may be able to be used alongside our iterative contact resolution algorithm (Chapter 6).
Polygon volume	**Convex polygon** Somewhat efficient algorithms exist to compute the penetration depth between convex hulls in a similar way to the analytical options. This computation is significantly more complex and expensive but can encode may more characteristics of a mesh. We do not use this method at all in our work.	**Convex polygon assembly** is the natural extension of this representation for modelling non-convex shapes. The complexity grows with the shape of the mesh.
Polygon surface	**Minkowki sum triangle volume** Sphere swept triangles can have well vectorised distance checks performed (Section 5.1.2). However, on their own the complexity of the method far outweigh the benefits due to the limited modelling capacity of a single triangle.	**Triangle soup surface** is made up of a collection of sphere swept triangles. Relatively fast geometric checks using distance minimisation and a hierarchical data structure (Section 5.3) while having the potential to be very geometrically accurate.
Tetrahadral volume	**Tetrahedron** The inertia matrix of a tetrahedron is simple to compute and the internal volume is explicitly represented.	**Tetraheron assembly** has similar characteristics to a polygon surface for contact detection. A series of connected tetrahedra model an internal structure.

3.2 Contact Point Model

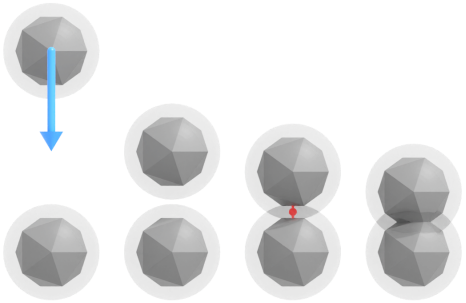


Figure 3.4: A pair of particles, with diameter 3.5m, where one particle has an initial position of $+10m$ and a negative velocity of $-10ms^{-1}$. Left, the initial condition. Left mid, with a step size $\Delta t = 0.5$ no overlap in the epsilon region is observed. This remains a valid state. Right mid, with a step size $\Delta t = 0.6$ an overlap in the epsilon region is observed. This remains a valid state. Right, with a step size $\Delta t = 0.7$ results in an intersection of the meshes. When using a contact point model where intersections are not allowed, this particle has now jumped directly from a non-contact state to an invalid state.

It is impossible to simulate exact incompressibility with explicit time stepping schemes when no interpenetration is allowed: Every time we update a particle, we run the risk that it slightly penetrates another one due to the finite time step size Δt. At the same time, particles exchange no momentum as long as they are not in direct contact yet. The momentum exchange remains "trivial" until we have violated the rigid body constraint. For these two reasons, we switch to a weak incompressibility model where each particle is surrounded by an $\epsilon > 0$ area [2, 9–11]. The area is spanned by the Minkovski sum of the triangles from $\mathbb{T}(p, t)$ and a sphere of radius ϵ minus the actual rigid object. Without loss of generality, we assume a uniform ϵ per particle. Our formalism is equivalent to a soft boundary formulation with an extrusive surface.

Definition 2 (Contact point) *Each particle is surrounded by an ϵ-environment. Two particles are in contact, if their ϵ-environment overlaps. Overlaps yield contact points which in turn yield forces.*

We parameterise each contact point with its penetration depth: Our contact detection algorithm identifies the closest path between two triangles. A potential contact

point c is located at the centre of this line. It is equipped with a normal $n(c)$, which points from the contact point towards either of the closest triangles (Figure 3.5). There is an overlap between the two ϵ-augmented triangles if and only if $|n(c)| \leq \epsilon$, that is if the ϵ-environments penetrate. In this case c is added to the set of collision points.

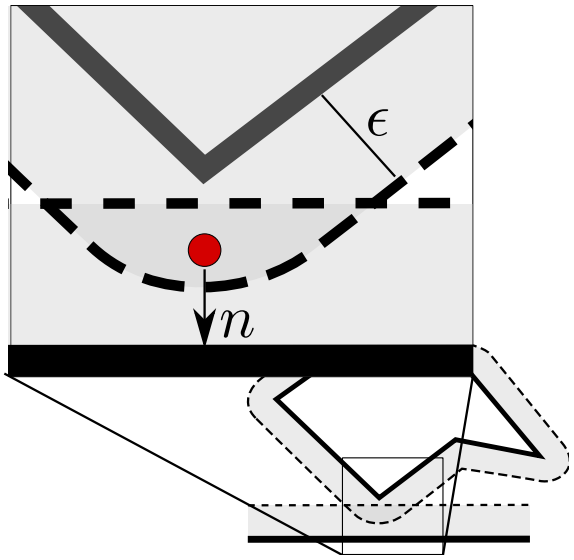


Figure 3.5: A pair of objects (black, solid) with their ϵ-environment (black, dotted) collide. The zoom-in shows the contact point which is located in the middle of the overlap region. The contact normal is directed from the middle point towards the closest point on one of the objects involved.

We note that two triangulated particles can yield redundant contact points if any contact lies on a shared edge or vertex and thus appears in multiple triangle pairs. We also observe that contact points are not unique for parallel triangles—they can be located anywhere within the ϵ-overlap. Therefore, contact points are filtered before being used by any other algorithmic steps. Contact points that are close together and oriented in the same direction are fused together.

3.3 Momentum exchange

Contacts between strictly rigid bodies happen instantaneously, and their arising forces have to be modelled by a Dirac distribution (a function with a value of zero at all points except zero and an integral of one over the whole range) such that

the momentum equation mirrors the laws of physics. However, when working in a force domain this would require an 'infinite' force so a numerical approximation is required.

To compute forces and the arising accelerations, i.e. the momentum exchange, from such a numerical approximation, there are two major thought schools: One option is to replace the Dirac distribution with a smooth force function. Its support is bounded by ϵ, i.e. it is zero if the distance between two objects or triangles exceeds 2ϵ.

The smooth approximation of the force function (Section 3.3.1 Contact Forces) means that particles become weakly compressible, and their interaction resembles the compression of a spring. Time step sizes have to be dramatically reduced as long as two objects continue to approach each other despite the actions of the force. As a consequence, global assemblies of objects with multiple contact points will rarely reach an exact equilibrium. They will always oscillate around the steady state solution as long as an external force such as gravity holds them together.

Another option is boundary extrusion (Section 3.3.2 Contact Impulses). Any overlap of the ϵ-layers is considered an immediate contact, which implies that the objects cannot approach each other any further; despite the fact that $0 < |n(c)| < 2\epsilon$, which means that we are not exact in a geometric sense. Once two ϵ-areas overlap in one time step, any subsequent time step may not compress this overlap further.

Consequently, the momentum exchange, i.e. force amplitude, is given implicitly: All relative approach velocity along the contact normal is cancelled. Instead of using spring forces to model contacts between particles we use the integral in time (impulses) directly and omit forces all together. This allows us to model an instantaneous change in velocity explicitly.

We might compare the differences between the force and impulse models to the difference between continuous time and event driven simulation models. The force model requires us to select the correct time step sizes in order to accurately integrate forces that change smoothly spatially and therefore the resulting momentum exchange. The impulse model requires us to select the correct time step sizes in order to accurately estimate the time of contact between particles and therefore the

resulting momentum exchange.

3.3.1 Contact Forces

A wide variety of contact force models exist in literature [135]. In general each model captures the behaviour of one or more of these forces that act between particles when they're in contact:

- Normal forces - When considered from a physical perspective this is the force that result from the material of the particle slightly deforming under the pressure of the contact. This deflection causes a force to act in the normal direction to return the particle to its original shape.

- Tangential forces - In other words: Friction. Friction is challenging to model for arbitrary objects because it depends so heavily on not just the material of an object but also the material of the object it's interacting with. To be able to model friction a coefficient of friction can be measured experimentally and this is then used to bound the frictional force to a factor of the normal force. Furthermore, to model the stick-slip phenomenon, where two surfaces will judder across each other (eg chalk on chalkboard), we can equip each contact with a pair of friction coefficients [136]. One for when the objects are static relative to each other and the second, and lower, for then the objects are sliding against each other.

- Adhesive forces - Are attractive forces that could be the result of chemical bonding, capillary force, electrostatic charge or other similar short-range attractive forces. Sometimes referred to as 'stiction', and an overlapping idea with having a higher static friction coefficient, this defines a force required to separate two options in static contact [137].

With a set of contact points plus their normals and dimensionless masses $M(p_i)$ and $M(p_j)$, we can derive a spring based force acting on a particle.

$$F(c) = \frac{n(c)}{|n(c)|} K_s (1 - \frac{|n(c)|}{\epsilon}) \sqrt{\frac{1}{\frac{1}{M(p_i)} + \frac{1}{M(p_j)}}} \tag{3.1}$$

computes the force arising from one contact point by mapping the ϵ-area onto a simple spring with a perpendicular friction force yet without any empirical damping [138]. The force depends on the contact normal $n(c)$ and applies to both colliding particles p_i and p_j subject to a minus sign for one of them. It is calibrated by a spring constant $K_s = 1,000\ Nm^{-1}$. A particle's total force is then the sum over the individual contact forces. Taking the centre of mass, the total mass and the mass distribution of a particle into account, the total force and total torque determine its acceleration and angular acceleration [139, 140].

This is a simplistic presentation—we ignore, for example, sophisticated interaction functions which distinguish contact points from contact faces— where the discontinuous force that arises at a contact is approximated by a "fade-in" force as two particles approach each other: Over an interval of size ϵ, the force smoothly approximates the target force, as the penetration depth $|n(c)|$ increases. The simple physics allow us to focus on the core challenge how to find contact points efficiently.

Additional or alternative damping terms or attractive terms (for surfaces that grip or stick to each other) can be added here but for our purposes we employ this simple spring model.

The behaviour of bouncing is inherently included in the notion of a spring based force. As objects approach each other the force acts against the incoming velocity and slowly reduces it to zero. At this point all the potential energy (subject to the accuracy of the numerical scheme) has been transferred to the imaginary spring. In the following time steps the energy is transferred back to the velocity of the particles.

In reality all objects deform a small amount and as they spring back to their original shape we get the effect of a collision and bouncing. In this sense spring based models give us a true-to-life representation of the underlying physics.

The force resulting from a contact is directly applied to the linear velocity of the particle,

$$\Delta v = \frac{F \Delta t}{m}, \tag{3.2}$$

and we aren't required to store the linear momentum.

Each particle is equipped with an inertia matrix, which describes the distribution of mass. The force is also applied to the angular momentum. The torque applied is

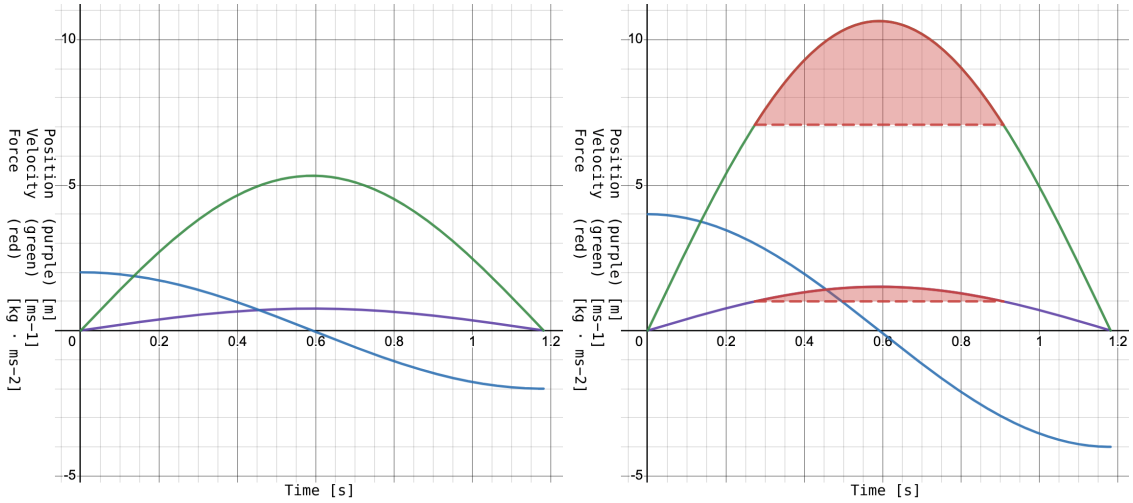


Figure 3.6: Purple - Position over time $[m]$. Blue - Velocity over time $[ms^{-1}]$. Green - Force over time $[kg \cdot ms^{-2}]$. Red - Value exceeds physically plausible limits. Left: The analytical solution for a simple particle impacting a stationary object with an initial approaching velocity of $2ms^{-1}$, a combined ϵ-region of $1m$, spring constant K_s of $10kg \cdot s^{-2}$ and mass $1kg$. Right: The same scenario but with an initial approaching velocity of $4ms^{-1}$. The linear spring is unable to counter the incoming velocity before an intersection occurs.

given as

$$\tau = F \times (x - c), \tag{3.3}$$

where x is the point of contact and c is the centre of mass. Once the torque is applied to the rotational momentum it is used to update the rotation,

$$\omega = RI^{-1}R^T L, \tag{3.4}$$

$$R = \exp\left(\frac{\omega t}{2}\right) R, \tag{3.5}$$

where R is the rotation, I^{-1} is the inverse inertia matrix, L is the angular momentum and exp is the exponential map. Unlike linear velocity, angular velocity isn't constant for the corresponding angular momentum. Therefore, angular velocity is recalculated from angular momentum at each time step.

Bouncing ball example Consider the scenario in Figure 3.7, where a rigid sphere of radius $1m$ approaches a fixed plane at $2ms^{-1}$. The sphere has a mass of $1kg$ and

we use a combined ϵ-region of $1m$ and spring constant K_s of $10kg \cdot s^{-2}$. We ignore the effects of gravity. Once the surface of the ball is within $1m$ of the ground a single contact is generated with the normal perpendicular to the plane. The analytical solution to this interaction (Figure 3.6 left) shows us the ball should stop with a $0.248m$ separation at $t = 0.591$. The position over time follows a sinusoidal curve. The velocity of the ball starts at $2ms^{-1}$, decreases to zero at the point of closest approach and then as the potential energy in the virtual spring is put back into the velocity of the ball the velocity goes to $-2ms^{-1}$ (separating).

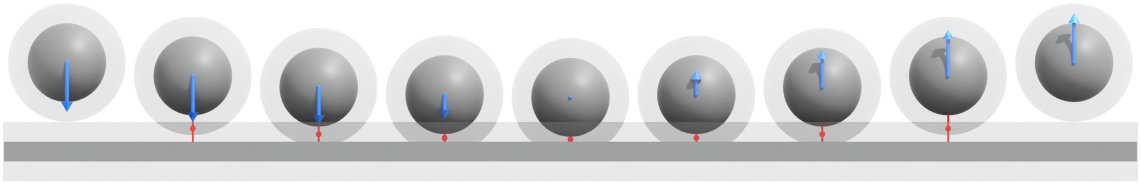


Figure 3.7: A simple bouncing ball example. Left to right shows individual time steps. A solid ball approaches a static object. The velocity (blue) decreases and then reverses as the contact (red) gets shorter and then longer again.

This example also demonstrates a case where this model fails (Figure 3.6 right). If the initial velocity is too high then the initial kinetic energy of the ball exceeds the maximum energy that the linear spring is able to counteract over the distance defined by the ϵ-region. If this scenario and model appeared in a time stepping simulation then the result would be intersecting geometry.

Instead of a linear spring model a smoothed version of the Dirac function (a function with a value of zero everywhere except at zero and an integral across the whole range of one) could be used, such that $\lim_{|n(c)|\mapsto 0} f(|n(c)|) = \infty$. However, this isn't sufficient to ensure intersection free simulations. The energy absorbed by a ϵ-region is given $\int_0^\epsilon f(x)dx$ and the kinetic energy of a particle can be arbitrarily large with increasing velocity. Therefore, a force function capable of simulating any velocity must satisfy $\int_0^\epsilon f(x)dx = \infty$. However, a function that rapidly grows introduces the issue of overshooting the velocity equilibrium point and introducing non physically large forces.

3.3.2 Contact Impulses

One limitation of using forces to simulate the interaction between rigid bodies while maintaining a strict separation of meshes is that forces take time to act on a body and change its velocity. A simple spherical particle landing on a plane might take a handful of time steps to get a sufficiently accurate numerical integration of the force over time required to make the particle bounce. However, in this case it is simple to calculate the desired response analytically. In addition to this, if we are strictly modelling particles as incompressible then the response to the collision would occur instantaneously. It could be argued that if the time step size is large (which we're often aiming to do) compared to the duration of the contact then in the frame of reference of our simulation the contact should happen instantaneously.

Furthermore, the existence of valid solutions to a system of contacts, when using contact forces and a simple model of friction (see Section 3.4), is called into question by the so called 'Painlevé paradox'. Simple scenarios can be constructed such that assuming the direction friction acts in to obtain a solution results in a relative velocity at the contact in an opposite direction to the assumed one [141]. The proofs later found to demonstrate the existence of solutions to the Painlevé paradox involve using 'impulsive forces' to introduce discontinuities into the velocity of objects [142]. The 'Moreau-Jean' scheme was among the first to separate impulse and force based responses from each other to provide solution guarantees [143, 144].

Therefore, for our work on local time stepping we choose to work with impulses, the time integral of force (Figure 3.8), to enable the largest possible time step sizes.

To resolve a series of contacts we phrase each contact as a constraint. The relative velocity along the normal of each contact must be zero or negative (where negative implies two objects are moving away from each other). A valid solution looks like a minimum set of impulses applied to each normal such that all constraints are satisfied.

In other words, let $\mathbb{C}$ denote all contact points within a cluster, and let $c_{p_1}, c_{p_1} \in \mathbb{C}$ identify the two particles associated with this contact point. We end up with a constrained equation system,

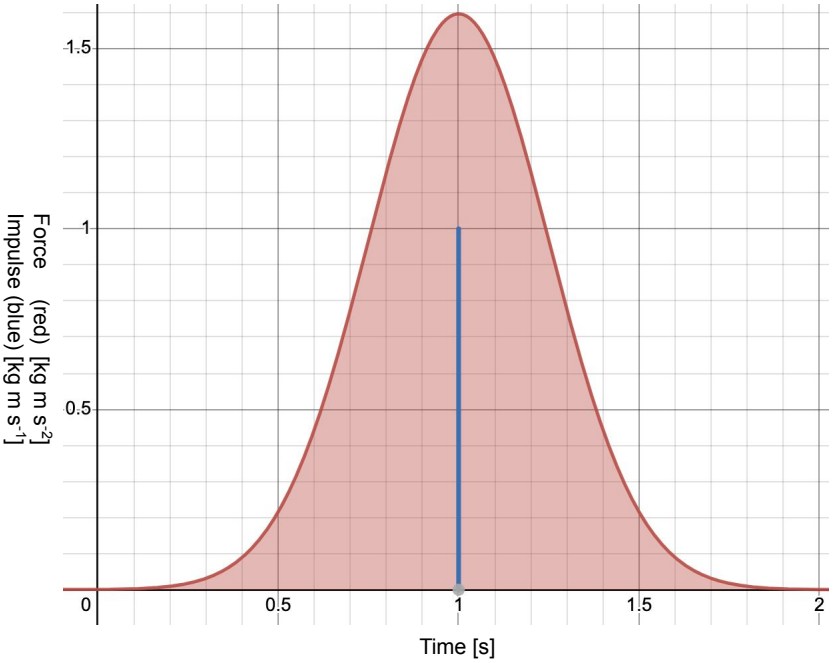


Figure 3.8: The different approaches we use to modelling momentum exchange between objects (ie. a collision). Using a spring based force model the force increases as the penetration depth increases (red). The profile of this curve would vary with the function used to describe the force. However, there will always be a peak where the relative velocity reaches zero and the particles begin to separate. If instead we use an impulse model all the momentum exchange is applied at a single point in time (blue). The peak of the impulse must be equal to the force integrated over the full range of time.

$$\forall c_{p_1}, c_{p_2} \in \mathbb{C}_i : m(c_{p_1})a(c_{p_1}) = -m(c_{p_2})a(c_{p_2}) \quad s.t. \quad v_{\text{rel}}(c) \geq 0, \tag{3.6}$$

where $v_{\text{rel}}(c)$ is the relative velocity of the closest points between c_{p_1} and c_{p_2} that arises from the computed accelerations $a(c_{p_1})$ and $a(c_{p_2})$. A $v_{\text{rel}} > 0$ indicates that the two particles involved have a velocity that will lead them to separate at this point.

As this is a non-trivial non-linear system due to the constraints yet phrases an equilibrium equation, we apply a Picard iteration, where we update the individual contributions per contact point one by one. The constraints are weakly enforced via a penalty term, and we aggressively damp the iterations. In line with other codes, we assume that the Picard iterations converge [145], which is notably a valid assumption as we minimise $t^{(\text{collision})}$ such that only few contact points remain.

Since the constraints are phrased over relative velocities and not on the lengths

of the contact normals, we can end up in a situation where particles gradually drift towards each other over several time steps, even though their relative velocity is cancelled each time they come into contact. Eventually, they might start to penetrate. Therefore, we add an additional force term to slowly separate the particles. This artificial term does not contribute to the resulting velocities.

We utilise the Sequential Impulses algorithm to solve this system of constraints (Appendix 2.1.3). The update to the impulse applied by a given contact can be written

$$\Delta P_n = \frac{-v_n}{m_1^{-1} + m_2^{-1} + (I_1^{-1}(r_1 \times n) \times r_1 + I_2^{-1}(r_2 \times n) \times r_2) \cdot n}, \tag{3.7}$$

where v_n is the relative velocity along the normal of the contact, I_i is the inertia matrix of particle i, which describes the distribution of mass of an object, m_i the total mass and r_i is the relative position of the contact compared to the centre of mass.

Unlike the spring force model, bouncing isn't necessarily a side effect of this method. Instead a small adjustment to the constraints can reintroduce bouncing. Instead of setting the target for convergence to less than or equal to zero relative velocity we can set it to the inverse of the relative velocity at the start of the time step. This would give an energy conserving bounce. A damping factor can then be introduced to match real world materials.

Bouncing ball example Considering the same example but this time using impulses instead of spring forces. During the first time step where the epsilon regions overlap the velocity is adjusted to the final (separating) value.

3.4 Friction

We model friction using a simple coulomb model where the maximum force or impulse at a given point is given by

$$f = \mu N, \tag{3.8}$$

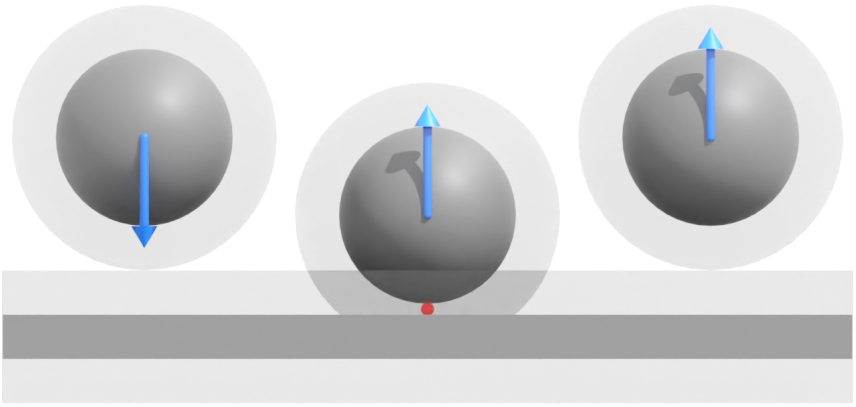


Figure 3.9: Our example illustrated for impulses. Left to right shows individual time steps. A solid ball approaches a static object. The velocity (blue) remains constant until the first time step where the epsilon regions overlap and then reverses instantaneously by the impulse applied by the contact (red).

where μ is a material parameter and N the force or impulse along the normal direction. Frictional forces always act in tangent direction to the normal of the contact. The friction force at each contact will be large enough to either reach the maximum allowed by the normal force or else large enough to completely cancel and relative velocity on the tangent plane defined by the contact normal.

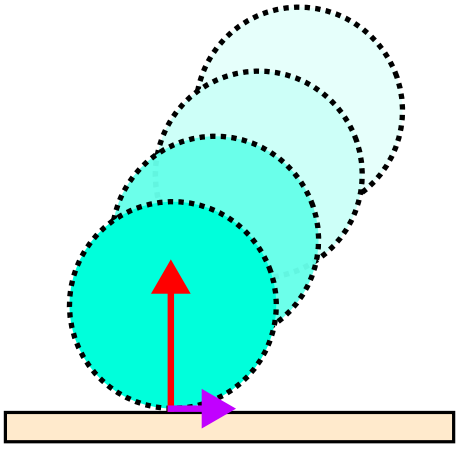


Figure 3.10: The impulses acting on a particle colliding with the ground. Red: The impulse along the contact normal to prevent the objects from interpenetrating. Purple: The friction impulse, bounded by the magnitude of the normal impulse scaled by a coefficient of friction.

Many further effects can be added to this simple friction model to enhance the accuracy to real world behaviour. For example, when a spherical objects rolls across

a perfectly flat surface the sphere as well as the ground will very slightly deform at the contact and will lose some energy when it springs back. 'Rolling friction' can be explicitly modelled to account for the energy lost to this imperceptible deformation [44]. When modelling contacts as points rather than surfaces the problem of friction about a spinning point is also encountered. No real world contact is actually an infinitely small point and so there will always be surface area that is in contact with the other object and so there will be relative motion as the contact surface rotates about the centre of rotation. This relative sliding motion results in friction and so energy loss. In a contact point model this behaviour is only encoded when the surface representation captures multiple contact points between objects. For example, a cube made up of 12 triangles spinning on top of a plane on one corner will generate just one contact point. This contact point has no relative motion along the surface of the plane so no friction force is added. By modelling contact points with additional properties, such as an estimated surface area and relative rotational velocity on the plane tangential to the contact normal, the energy dissipated (as heat and sound) can be explicitly modelled [146]. Furthermore, we use a single coefficient of friction per object rather than two per pair of interacting materials. This allows us to focus on the contributions of this work but prevents us modelling more complex physical behaviour, such as stick-slip.

This work focuses on algorithmic advancements to the collision detection and time stepping aspects of the DEM algorithm. Therefore, we implement both simple contact and simple friction models. However, we see no reason that this would restrict our algorithmic contributions to codes with more complex physical models.

CHAPTER 4

Local time stepping algorithm

In this chapter a novel local time stepping algorithm is presented. This algorithm targets the wasted computation caused by an uneven distribution of computational load across all the particles in a simulation over time. By only using computational resources for interactions that define the progression of a physical system a lower time to solution might be achieved. Globally adapting the time step size to the minimum step size required by all potential interactions begins to address this. However, if only a few interactions are taking place during any one time step then the available level of parallelism that can be exploited is potentially much lower. In order to combat this our algorithm allows particles to advance with potential different time step sizes to each other. We simultaneously computer particle interactions across a window of time and therefore increase the available (and useful) work to occupy parallel hardware. We demonstrate results for a selection of artificial scenarios showing the benefits of our algorithm for highly dynamic configurations.

Local time stepping can be characterised in several ways: First, we have to make a choice between optimistic and conservative time stepping. In optimistic time stepping, we follow a trial-and-error approach. We start with a time step that is as large as possible, but, in the case of failure, i.e. penetration of physically

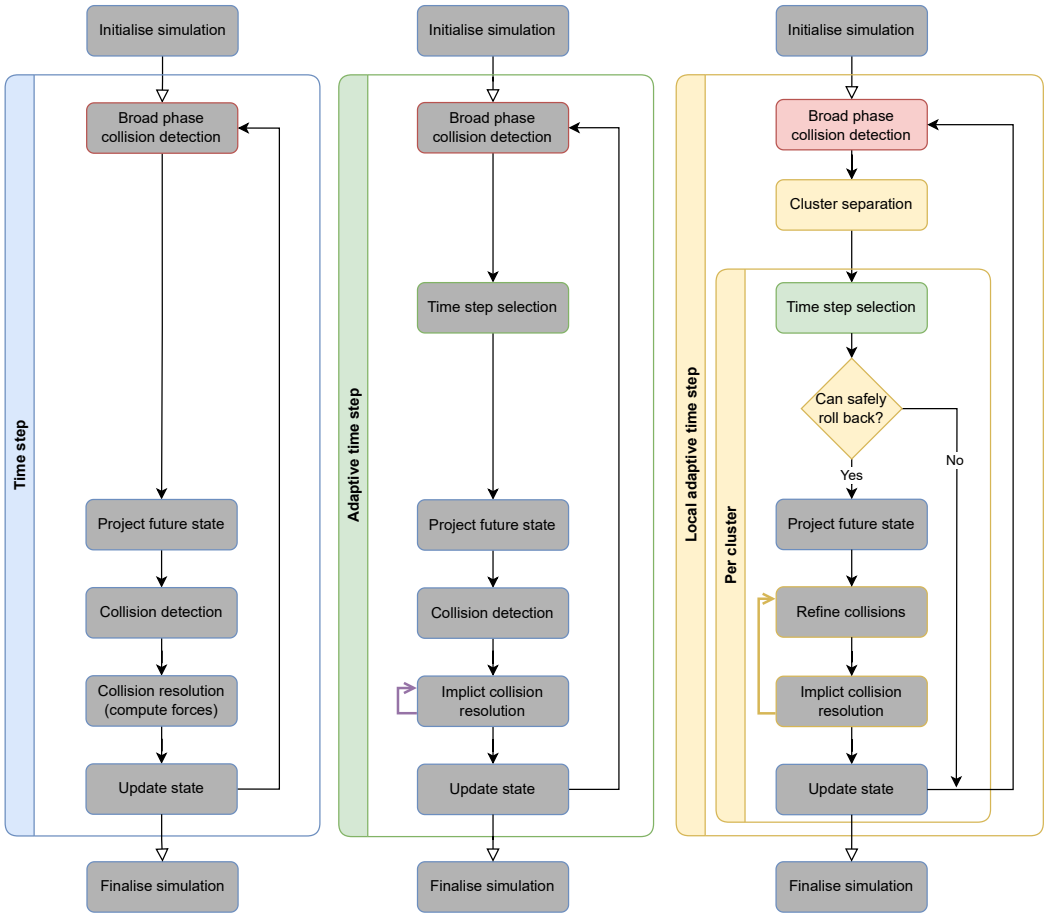


Figure 4.1: The phases of the algorithm that are discussed in this chapter are shown in colour.

incompressible objects, roll back, reduce the time step size and recompute until we have found an admissible step size. In conservative time stepping, we underestimate the admissible time step size all the time such that we can guarantee an admissible step size without any penetration.

Second, local time stepping can be characterised through its granularity. Individual particles could progress with their own time step size, or we could cluster particles and make these clusters advance in time with the same time step size, i.e. in-sync. Global time stepping is equivalent to a degenerated cluster approach, where one cluster comprises all rigid bodies within the system. A global time stepping scheme typically requires time step sizes orders of magnitude smaller than the local time step required for particles away from collisions [73]. The granularity determines how many particles can advance in time before they "unlock" further clusters

to progress as well. A radical solution allows only one rigid body pair in the whole system to be updated at any point and hence translates the continuum model into a sequence of events. The efficiency gain induced by small clusters—we only update what is necessary—is to be balanced against the loss of concurrency.

The "flavour" of local time stepping that most commonly appears in literature involves selecting a target global time step size and then locally selecting a number of substeps to take to achieve the desired final time stamp [65]. This is the technique used by the only work, to the best of our knowledge, that utilises local time stepping for DEM code [66], which demonstrates it's use for simulation of analytical spheres in close to static states and dense interactions. However, due to the potentially orders of magnitude different in required time step size for handling contacts between particles compared to particles away from collisions [73] this method of substepping seems likely to not fully exploit potential speedups from avoiding unnecessary work in situations where iterations are more sparse. Therefore, we consider a different "flavour" where each particle is free to advance with it's own time step size.

4.1 Model

The ruling physical model is simple: Each particle $p \in \mathbb{P}$ is rigid. However, since exact collisions are a singularity in time and therefore difficult to handle numerically, we augment each particle by a small halo layer of width $\epsilon(p) > 0$. This cushion around the particle helps us to weaken the notion of a contact: Two particles are in full contact if their ϵ-layers overlap. Full contact means that, once the halo layer overlaps, the two particles completely exchange momentum and then will separate again or stick to each other. But they will not approach each other further.

Further to that, we hold multiple snapshots per particle, and we assume that we can interpolate: If we have a snapshot of the particle position $x(p,t)$, velocity $v(p,t)$, rotation $r(p,t)$ and the angular velocity $w(p,t)$ for two time stamps $t_1 \leq t_2$, we can reconstruct all four quantities of interest at any point in-between by linear interpolation or spherical linear interpolation for quaternion types respectively. This scheme neglects higher order effects affecting the trajectory, and it neglects long-

range global forces such as gravity.

Each particle is studied at three snapshots $t^{(\text{old})}(p) \leq t^{(\text{current})}(p) \leq t^{(\text{new})}(p)$, i.e. our intention is to develop $t^{(\text{new})}(p)$ from the two previous snapshots. The time span between two snapshots is not constrained. As particles may therefore reside on different time stamps, we abandon the notion of a time step. Instead, we consider a time step to be an operation which moves a nonempty subset of $\mathbb{P}$ by at most a time span of Δt which we prescribe a priori. To simulate the system's behaviour over time, we have to invoke this time step multiple times, until

$$t^{(\text{min})} = \min_{p \in \mathbb{P}} t^{(\text{current})}(p) \tag{4.1}$$

fulfills $t^{(\text{min})} \geq t^{(\text{final})}$, where $t^{(\text{final})}$ is the given final simulation time.

4.2 Particle state

One simple but significant difference between the original implementation of Delta Chapter 1 and our implementation Delta2 is how we store the state of the particles. Instead of storing the position and rotation of the particle by updating the vertex positions Delta2 prefers to store vertices with local coordinates and a separate transformation.

There are two main motivations for this choice: First, we aim to use particles with thousands or even millions of vertices but where only a few interact with neighbouring particles at any time. In this case updating a particle's state would require visiting every vertex where instead we wish to access as few as possible. Second, for our local time stepping algorithm we need to represent the state of a particle over a span of time. This would require us to store every vertex multiple times if position and rotation were encoded directly. However, the trade off is that whenever we require the global position of a vertex, edge or triangle a matrix multiplication is needed.

Five key properties are stored per state:

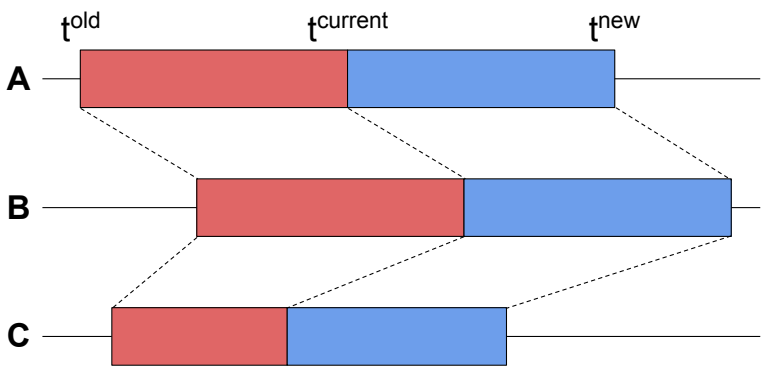


Figure 4.2: Particles A, B and C are each equipped with three states at independent time stamps. The time between these timestamps is assumed to contain linear motion. Therefore the three states of each particle describes the state of the particle across a window of time that can be different to other particles.

```
class State {
    public:
        ...
    private:
        double _time;

        Eigen::Quaterniond _rotation;
        Eigen::Vector3d _angular_momentum;
        Eigen::Vector3d _translation;
        Eigen::Vector3d _velocity;
};
```

4.3 Broad phase collision detection

The goal of a broad phase is to narrow the set of all possible pairs of particles to just the ones that might induce a contact. We denote this with the function $c : \mathbb{P} \times \mathbb{P} \mapsto \{\top, \bot\}$, interacting and not interaction, and serves as guard for follow-up checks. Without this the collision detection phase runs with quadratic complexity $\mathcal{O}(|\mathbb{P}|^2)$. Our realisation relies on existing code offered through Intel Embree's raytracing kernels [147]. As a traditional raytracing toolbox, the software is tuned to handle

bounding box checks and variants thereof efficiently.

This problem is made even more challenging when we introduce the idea of 'anarchic' local time stepping, where each particle stores state spanning an independent window of time. Our work uses a combination of two models to rapidly identify $\perp$, i.e. to narrow down the number of potential collision pairs.

Throughout this first phase, we extrapolate from the current state assuming that each particle moves in space and time without any interaction. Due to our physical model, each particle spans a space-time tube: Let $B(p, t^{(\text{current})})$ be a bounding shape around a particle at time $t^{(\text{current})}$. It covers all of the particle including its ϵ-halo. As we know the particle properties $x(p, t^{(\text{current})}), v(p, t^{(\text{current})}),\ r(p, t^{(\text{current})})$ and $w(p, t^{(\text{current})})$ at this point, we can assume $v(p, t^{(\text{current})})$ and $w(p, t^{(\text{current})})$ to be const and therefore linearly or spherical linear extrapolate $v(p, t^{(\text{current})})$ and $r(p, t^{(\text{current})})$.

We use this extrapolated state to construct a bounding geometry $B(p, t^{(\text{new})})$. The two bounding shapes can now be connected in space-time, i.e. each object's projected trajectory is now covered by a space-time bounding geometry $B^{\text{space-time}}(p)$, and we can intersect these space-time bounding objects of any particle pair p_1, p_2 to determine their image $c(p_1, p_2)$.

Given any two particles they might reside at different time stamps, i.e. the algorithm might decide to bucket two particles with two different time stamps into one cluster, to roll the one that is further ahead in time back to a common time stamp, and then to evolve them in time. To accommodate this possibility, it is important to extend the space-time formalism to span $t^{(\text{old})}$, too: We study two space-time bounding geometries from $t \in (t^{(\text{old})}, t^{(\text{current})}) \cup (t^{(\text{current})}, t^{(\text{new})})$.

Particle-specific time step sizes Larger estimated time step sizes result in a higher probability of two objects being identified as potentially overlapping by the initial test. Therefore, we assign each particle its specific time step size with the aim of minimising the time span covered per particle, which obeys

$$\Delta t(p_i) = \min(\alpha(t^{(\text{current})}(p_i) - t^{(\text{old})}(p_i)), \Delta t). \tag{4.2}$$

Due to the constant $\alpha > 1$, the time step size that feeds into the trajectory prediction

is a creeping average between the maximum global time step size Δt and previous time steps. After a contact reduces the time step size for a particle the time step size increases again, approaching Δt, unless further reduced by new contacts.

Check 1: Space-time bounding boxes With a well-defined time step size per particle at hand, we use axis-aligned bounding boxes for $B(p)$ and construct a space-time bounding box for $B^{\text{space-time}}(p)$. These space-time bounding boxes between any two particles are compared.

As long as two particles' timespans $(t^{(\text{old})}, t^{(\text{new})})$ overlap, the algorithm can be read as a purely spatial approach: We take the particle positions at the three points in time and surround all three states of the particle, including their ϵ-environment, with a bounding box. If these super-bounding boxes of two particles do not overlap, $c(p_1, p_2) = \perp$ and we terminate. Otherwise, we continue with the next step of the checks.

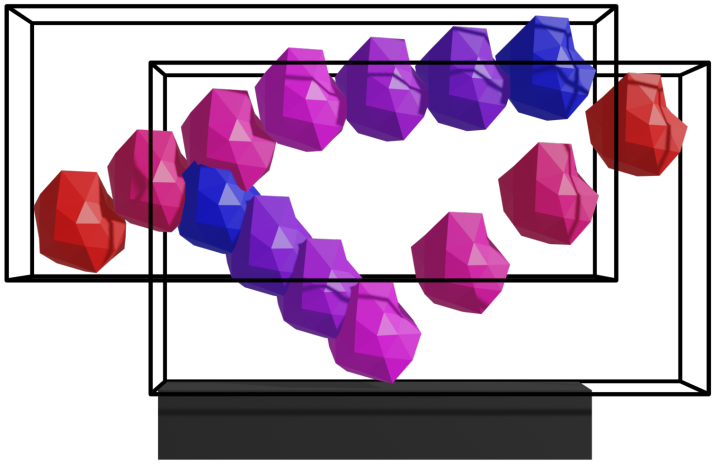


Figure 4.3: The last state (red), current state (pink) and future state (blue) and the overall bounding box for two particles in motion. One particle moves left to right and follows an arc due to gravity. The other particle moves right to left and comes into a contact a solid ground object causing it to bounce. Comparing the overall bounding boxes of both objects suggests that a collision might occur. However, when we consider the space-time volume spanned by the particles it is clear that these particles will pass by each other without interacting.

For this first check we reduce the space-time problem to a spatial problem only. We collapse the time dimension (Figure 4.5).

Check 2: Space-time tubes. Having two space-time bounding boxes overlapping can result in a lot of false-positives, i.e. potential collisions which are not really confirmed later on. This happens, for example, if two fast moving particles cross paths at different time stamps within the same step. To eliminate such false-positives, we add a second test.

We embed each particle into a bounding sphere, i.e. let B be a sphere. The space-time object $B^{\text{space-time}}(p)$ now resembles a hose with two linear segments, as we model each particle as a sphere moving linearly along it's path from $t^{(\text{old})}$ to $t^{(\text{current})}$ and then continuing from $t^{(\text{current})}$ to $t^{(\text{new})}$. If two $B^{\text{space-time}}(p_1), B^{\text{space-time}}(p_2)$ hoses do not intersect, we reset $c(p_1, p_2) = \bot$. Otherwise, this second test confirms $c(p_1, p_2) = \top$.

Static geometric objects. The treatment of static geometry parts requires special attention throughout the clustering: Walls for example are often modelled with few huge triangles. Without special rules, such a wall "couples" all particle assemblies hitting this wall globally, even though the assemblies of particles could be handled as separate clusters. They do not interact with each other. They do not have to advance in time with the same time step.

Definition 3 *Static geometric objects are parts of all the clusters which host at least one particle which directly interacts with them. However, they do not induce connections between particles.*

We eliminate static objects from the actual cluster identification.

Efficiency The first pre-check with cubes stands in the tradition of pessimistic algorithms, and yields information of limited interest if particles move very fast or over a long time span, while the second step actually takes rapid changes of the trajectory and fast velocities into account. It prevents too many collision pairs to be added to c, while the first step handles quasi-stationary setups quickly and eliminates them from further checks.

Stationary objects such as walls are explicitly eliminated from the initial clustering. They are invisible to the extrapolation. Once the clustering algorithm has

terminated, we replicate all static objects, such that each cluster sees "its own version" of the static object such as a wall. We assume that particles that bump into a wall typically use small time step sizes (unless it is an initial collision with a wall). Many particles that hit the same wall in some distance thus tend to end up in different clusters whereas the wall as proper geometric particle in the computation of c would merge all of these objects into one big cluster and remove concurrency.

For the actual clustering, i.e. the identification of unconnected subgraphs, a very simple, but parallel, iterative method. Nodes are repeatedly coloured with the lowest colour id of the neighbouring particles until a complete iteration produces no changes (Appendix 2.1.4). The problem is in $\mathcal{O}(|\mathbb{P}|^2)$, yet on average.

The expressiveness of c but also the efficiency of the first step hinge upon the choice of the used time step sizes. On the one hand, the chosen time step size depends on the last time step size used. Clusters which previously have used a very small time step size will continue to use small time steps, while others might use a time step size close to Δt. This models the fact that particles involved in complex interactions during the previous time step are likely to still be involved in complex interactions in the next time step. Through α, we can control how quickly a previously small time stepping particle approaches the maximum time step Δt. We use $\alpha = 2$.

On the other hand, the Δt is in itself a global tuning parameter, which we can set to control the optimism of our time stepping and to restrict the maximum time step size. By picking small Δt, we run risk of constraining particles overly pessimistically which could travel free of any collision over longer time spans. Large Δt however make the c checks overly pessimistic and lead to large clusters. Yet, we note that the historic data used to make the time step creep eliminates the fact that we are too pessimistic after the very first time step. It is hence reasonable to artificially set $\Delta t(p)$ to a small value initially—no historic data does exist here, so the formula has to rely on some starting condition anyway—and to pick a large global Δt afterwards. It will be approached by those particles, where the physics allow for it.

Cluster decomposition. As we reduce the time step size of individual particles, we potentially sparsify c further. This can lead to situations where a cluster decomposes into several subclusters which in turn increases the concurrency level. We did not find any performance gain from a re-clustering and further decomposition of cluster tasks, and therefore do not exploit this additional increase of concurrency.

4.4 Time step selection

The time step calculation identifies the largest time step size with which all particles of that cluster can safely advance without missing out on further collisions within the cluster. We search for this time step size within the time span $\Delta t(\mathbb{P}_i)$.

Let $t^{(\text{collision})}(p_1, p_2)$ denote the time when two particles' halos intersect for the first time. It is subject to

$$\begin{aligned} t^{(\text{min})}(\mathbb{P}_i) \ll t^{(\text{collision})}(p_1, p_2) &\leq t^{(\text{max})}(\mathbb{P}_i) \quad \forall p_1, p_2 \in \mathbb{P}_i, \\ \text{where } t^{(\text{max})}(\mathbb{P}_i) &= t^{(\text{min})}(\mathbb{P}_i) + \Delta t(\mathbb{P}_i). \end{aligned} \tag{4.3}$$

The following case distinctions feed into the calculation of $t^{(\text{collision})}(p_1, p_2)$ within (4.3):

1. If two particles move away from each other, $t^{(\text{collision})}(p_1, p_2) = t^{(\text{max})}(\mathbb{P}_i)$. Particles that separate at $t^{(\text{current})}(\mathbb{P}_i)$ do not constrain our time step size choice further.

2. If two particles' ϵ-area overlaps at $t^{(\text{current})}(\mathbb{P}_i)$, their $t^{(\text{collision})}(p_1, p_2) = t^{(\text{max})}(\mathbb{P}_i)$. Particles that stick together at the begin of a time stamp (they rest upon each other or move parallel, e.g.) do not constrain the time step size further.

3. For all other particles, we compute the collision time stamp.

With (4.3), we can update each cluster's time step or new time stamp respectively:

$$t^{(\text{new})}(\mathbb{P}_i) = \min_{p_1,p_2 \in \mathbb{P}_i} t^{(\text{collision})}(p_1, p_2) < \Delta t, \qquad \text{and therefore} \tag{4.4}$$

$$\Delta t(\mathbb{P}_i) \leftarrow t^{(\text{collision})}(\mathbb{P}_i) - t^{(\text{current})}(\mathbb{P}_i). \tag{4.5}$$

The search for $t^{(\text{collision})}$ over all particle pairs within a cluster is computationally demanding:

Geometric model. Let $\mathbb{T}_h(p)$ describe the set of triangles describing a particle p. τ^ϵ denotes the volumetric extension of its corresponding triangle $\tau \in \mathbb{T}_h(p,t)$, i.e. the triangle plus its ϵ-environment.

$$\mathbb{T}_h^\epsilon(p,t) = \bigcup_{\tau \in \mathbb{T}_h(p)} \tau^\epsilon \tag{4.6}$$

yields a shell object, i.e. a volumetric geometric object into which the surface of the particle is embedded.

Definition 4 *A* contact point *is a tuple of a position in space x, a time stamp t, a normal vector n and two triangles $\tau_1 \in p_1$ and $\tau_1 \in p_2$ from two different particles $p_1 \neq p_2$. The following properties hold:*

1. *The distance between x and τ_1 or τ_2 is at most ϵ.*
2. *If two triangles are closer than 2ϵ, they have exactly one contact point per time stamp.*
3. *n is the shortest distance vector to one of the nearest triangles.*

Two particles are in contact at a certain time, if there is at least one contact point between their triangles. We note that two triangulated particles can yield redundant contact points if any contact lies on a shared edge or vertex and thus appears in multiple triangle pairs. We also observe that contact points are not unique for parallel triangles—they can be located anywhere on a submanifold within the ϵ-overlap. Our discussion from hereon assumes that contact points are filtered, i.e. contact points

that are close in space are fused into one contact point to avoid spatial replicas. n's orientation finally is not unique. It can point to either of the associated triangles.

Collision time detection. Contacts between two triangles are either placed on the shortest distance between a vertex of one triangle and the other triangle's surface, or are located in-between two triangle edges. Efficient code blocks to compute them are known [5, 53]. We have to transfer such spatial algorithms into the space-time domain over $(t^{(\text{current})}, t^{(\max)})(\mathbb{P}_i)$ with a new (minimal) $\Delta t(\mathbb{P}_i)$ in-between to be determined.

Per triangle pair we compute a time of contact by considering the space time contact between each vertex-face and edge-edge pair. The maximum admissible time step size is therefore the minimum of all the individually computed contact times. Furthermore, at each space-time contact we can compute the relative velocities and so compute an approximation for both the earliest time of contact and the time the two meshes will intersect. Our selection of final cluster step size is taken as a weighted average of these two values. The aim is to select the largest step size possible while acknowledging that our space-time contacts are an element-wise linear approximation and fail to fully capture possible contacts caused by rotations alone.

For each vertex-face pair, we compute the position of the vertex relative to the triangle at the beginning and end of the maximum admissible time step: We use a relative coordinate system which moves with the triangle. After that, the algorithm takes the space-time line segment connecting the two relative positions. We can now analytically compute the minimum distance, as the minimum distance is either observed at the start or the end point, or arises from the time stamp when the line intersects the (relative) plane in which triangle lives—if it exists. For each of these two or three, respectively, situations we can use Barycentric coordinates to construct the actual contact point, which is an approximation as we neglect rotation in this ansatz.

For each edge-edge pair, we realise the minimum distance computation iteratively by repeatedly bisecting the time span in which we search for collisions. If the minimal distance does not reduce anymore, we stop the bisection. Other more sophisticated

methods exist [56] with additional efficiency and correctness guarantees, but this simple search serves well as a cheap heuristic for the time of contact. Again, the search is an approximation.

After $t^{(\text{collision})}$ $(t^{(\text{current})} + \Delta t')$ is determined, we evaluate the geometric setup at this moment in time: The time stamp might still be too far in the future and hence yield penetrating objects caused, for example, by rotational effects that are missed out. In the case of penetrating objects we roll back and rerun the time step selection using $\Delta t'$ as initial cluster time step size.

Accelerated multiscale algorithm. To accelerate this algorithm, we use an iterative multi-resolution scheme exploiting the notion of surrogate geometries [53]:

Definition 5 *Let a* surrogate *representation of a triangle set* $\mathbb{T}^\epsilon_h(p)$ *be another triangle set* $\mathbb{T}^\epsilon_{2h}(p)$ *with*

$$\mathbb{T}^\epsilon_h(p) \subseteq \mathbb{T}^\epsilon_{2h}(p) \quad \textit{and} \tag{4.7}$$

$$|\mathbb{T}^\epsilon_h(p)| > |\mathbb{T}^\epsilon_{2h}(p)|. \tag{4.8}$$

Per particle, we can construct a sequence of surrogates $\mathbb{T}^\epsilon_h(p)$, $\mathbb{T}^\epsilon_{2h}(p)$, $\mathbb{T}^\epsilon_{4h}(p)$, $\mathbb{T}^\epsilon_{8h}(p)$, ... recursively. We refer to this sequence as levels. Each surrogate is conservative with respective to the next finer surrogate due to (4.7): If we decide that two surrogates of level k do not collide, then their corresponding surrogates of level $k/2$ do not collide either. Each surrogate is also effective with respect to the next finer surrogate due to (4.8): It is cheaper to handle a surrogate of level k compared to the surrogate of level $k/2$, as the former features fewer triangles.

Definition 6 *Let the parent of a triangle* $\tau \in \mathbb{T}^\epsilon_{kh}(p)$ *be given as*

$$\tau_{parent} \in \mathbb{T}^\epsilon_{\frac{k}{2}h}(p) \quad \textit{and} \quad \tau^\epsilon \subseteq t^\epsilon_{parent}.$$

With the notion of a surrogate at hand, we can introduce an iterative variant expanding upon our vanilla code version:

1. For two particles p_1, p_2 of interest, we start with their coarsest surrogate representations. They define the search sets $\mathbb{S}(p_1)$ and $\mathbb{S}(p_2)$.

2. Per pair $(\tau_1, \tau_2) \in \mathbb{S}(p_1) \times \mathbb{S}(p_2)$, we approximate the minimum time of contact. τ_1 and τ_2 are removed from their search sets once we have evaluated all the pairs they are involved in.

3. If a new contact times exists and it falls into our search span defined by $t^{(\text{current})}(\mathbb{P}_i)$ and $t^{(\text{collision})}(\mathbb{P}_i)$,

   - we update $t^{(\text{collision})}(\mathbb{P}_i)$ if they reduce this quantity further and if τ_1 and τ_2 are both taken from the fine mesh triangulations, i.e. $(\tau_1, \tau_2) \in \mathbb{T}^{\epsilon}_{kh}(p_1) \times \mathbb{T}^{\epsilon}_{kh}(p_2)$;
   - otherwise, we throw away these contacts and the contact times and instead insert all children of τ_1 and τ_2 into $\mathbb{S}(p_1)$ or $\mathbb{S}(p_2)$, respectively.

4. If $\mathbb{S}(p_1) \neq \emptyset \wedge \mathbb{S}(p_2) \neq \emptyset$, we continue with Step 2.

Efficiency, contextualisation and implementation. Our particle-to-particle comparison only studies particle combinations with $c(p_1, p_2) = \top$. Even though two particles reside within one cluster, we might have come to the conclusion earlier that these particles cannot collide. This information is used here.

$t^{(\text{collision})}(\mathbb{P}_i)$ is monotonously decreasing due to our iterative scheme. It iteratively narrows down the particles and the branches of the surrogate tree that can collide. As we run the tests over the space-time geometries and reduce the time span of interest, we disable more and more entries within c, as we find out that particles do not collide. Within each particle-particle comparison, the multiscale variant terminates the exploration of branch pairs of the surrogate trees early. Even branch pairs that would result in a collision later in the original time span are dropped, once we know that another particle pair yields a collision earlier on. Per iteration of the iterative scheme, fewer particle pairs and a narrower time span are to be studied. Therefore, it would be reasonable not to run the multiscale algorithm above on a particle-pair by particle-pair basis, but to examine all particle pairs in one go, i.e. to run through their surrogate geometry levels concurrently, as a further optimisation.

Our intention is that, where geometrically possible, only the first iterations deal with large particle counts. In these cases, our surrogate formalism ensures that large particle counts do not translate into excessive triangle counts, as we use rather coarse geometric representations. The overall triangle count per iteration per particle is likely to increase monotonously, but our hope is that the total triangle count per cluster per iteration does not grow (too fast).

With $|\mathbb{T}_{kh}(p_1)|$ triangles for particle p_1 and $|\mathbb{T}_{kh}(p_2)|$ for p_2 at one step of the algorithm, we have to perform a maximum of $|\mathbb{T}_{kh}(p_1)| \cdot |\mathbb{T}_{kh}(p_2)|$ triangle-triangle continuous comparisons. In an ideal case where there exists a single contact between two particles and only a single branch of each particle has to be explored then for one step of the algorithm we only need to perform K^2 triangle-triangle continuous comparisons, where K is the maximum number of children each surrogate triangle possess. This gives us an upper and lower bound on the number of triangle-triangle collision checks required. When no contacts exist or contacts only exist after the current $t^{(\text{collision})}(\mathbb{P}_i)$ (ie. another particle pair in the cluster has already been evaluated and an early contact was found) then we are able to beat this lower bound by early stopping the exploration.

We assume that triangles move linearly through space. This is not valid once $r(p) \neq 0$. Our current implementation ignores the possibility that a section of an object rotates fully through another one. For small admissible time step sizes and slow rotations, this is reasonable. In an application with finer interacting geometry (for example, fine teeth on interlocking gears) it may be necessary to add extra checks here. An alternative option is it to increase the ϵ region of each triangle and surrogate to ensure the real volume swept by a rotation triangle is covered. Notably, we can make ϵ the larger the coarser the surrogate. While this would introduce more false-positive collisions early throughout the multiscale algorithm, it would ensure there are no false-negatives.

Observation 2 *Even though we search for a minimum contact time per cluster, multiple contacts can arise at this time span, as we approximate the contacts spatially due to the ϵ-layer and temporarily, as we slightly "overestimate" the time to the first contact.*

The algorithm's underlying sweeps over particle pairs and triangle pairs translates directly into data parallelism. However, a realisation with parallel for loops is disadvantageous as it introduces synchronisation points after each fork-join and starts to suffer from the inherent imbalances due to non-constant compute cost per compute step. We hence advocate for a strict dynamic task tree formalism, where we initially spawn one task per particle-particle comparison triggered through c. Each task kicks off the algorithm in Step 1. The respective tasks unfold into child tasks in Step 3. However, this unfolding is constrained by the continuously decreasing search time span. Through such a task formalism, the iterations over particle- and triangle-pairs are intrinsically interwoven. It is clear that the arising dynamic task graph is challenging to load balance.

4.5 Time stamp rendezvous

The current state of a particle covers a window of time represented by three momentary states each with a time stamp. Only the oldest state of these three is assumed to be totally correct. Furthermore, for any particle pair the span of time from the $t^{(\text{old})}$ to $t^{(\text{current})}$ states must overlap but doesn't have to be identical. To be able to compute the interaction between them $t^{(\text{current})}$ time stamp must be identical. Therefore, we require a mechanism for performing a rendezvous in time between particles that have been identified by the broad phase as potentially interacting.

For any pair of particles if $t_A^{(\text{current})} \neq t_B^{(\text{current})}$ then either $t_A^{(\text{current})}$ lies within $(t_B^{(\text{old})}, t_B^{(\text{current})})$ or else $t_B^{(\text{current})}$ lies within $(t_A^{(\text{old})}, t_A^{(\text{current})})$. We take the $t^{(\text{current})}$ with the highest current timestamp and roll it back to the $t^{(\text{current})}$ with the lowest timestamp.

Since each span of time is a linear approximation of the state of the particle, to roll back a state we simply take the linear interpolation of $t^{(\text{old})}$ and $t^{(\text{current})}$ to get the desired time stamp. The exception to this is rotation, which requires spherical linear interpolation (Figure 4.4) to maintain a valid state.

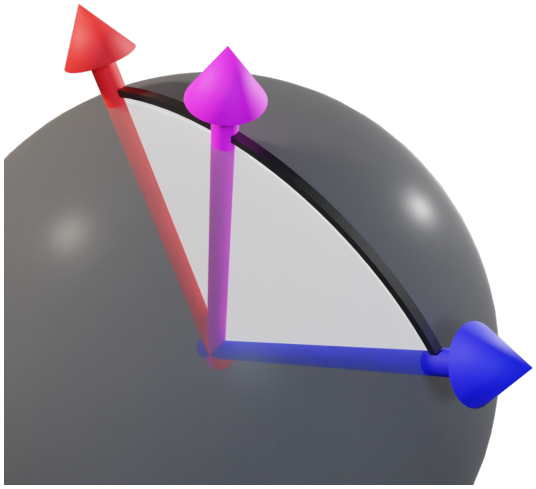


Figure 4.4: Rotations are interpolated using spherical linear interpolation (SLERP). $t^{(\text{old})}$ blue and $t^{(\text{current})}$ red are slerped by a factor of 0.65 to calculate a new rotation in pink.

4.5.1 Cluster setup and consolidation

As we want to advance all the particles within a cluster in-sync, we first of all have to ensure that they all start from a state with the same time stamp. Our clusters are recomputed in each and every sweep. Consequently, a common, joint time stamp $t^{(\text{current})}$ for all particles of a cluster is not guaranteed.

Once we have determined $t^{(\text{min})}(\mathbb{P}_i)$, we interpolate between the particles' snapshots to reconstruct a joint, synchronised state. Besides the physical simplification, an interpolation relies on the assumption that a particle does not experience any collisions between $t^{(\text{old})}$ and $t^{(\text{min})}(\mathbb{P}_i)$.

Further to the consolidation of the particle snapshot, we also assign each cluster $\mathbb{P}_i$ a unique

$$\Delta t(\mathbb{P}_i) = \min_{p \in \mathbb{P}_i} \Delta t(p). \tag{4.9}$$

This is a quantity that is used in follow-up steps. It consolidates the individual time step sizes fed by historic data over the whole cluster.

By optimistically selecting a time step size, we introduce the possibility of advancing too far and creating an invalid state, as we miss out on contacts.

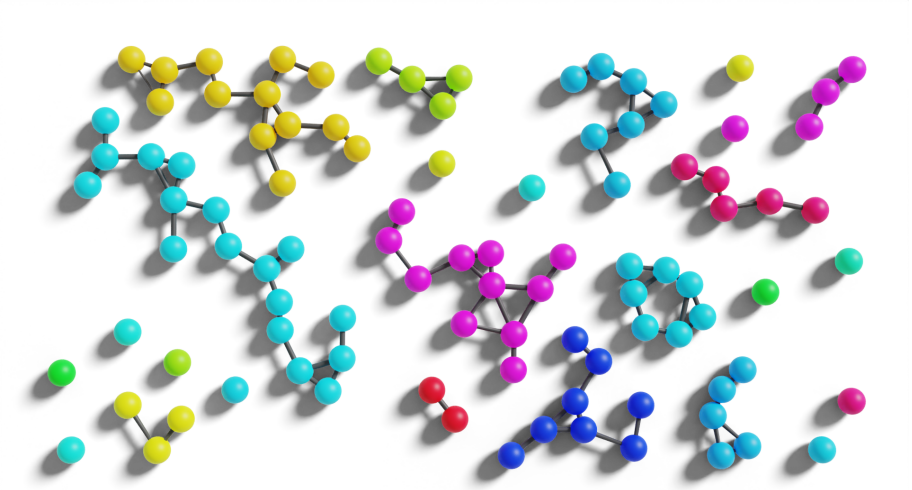


Figure 4.5: A set of simple particles resting on a surface with the potential interactions identified by the broad phase shown as connecting lines. The colours indicate the clusters that each particle is sorted into.

Observation 3 *If particles change cluster membership, they typically induce a roll back (Section 4.5).*

Our cluster consolidation substep thus is the technique to implicitly handle overly optimistic time step choices.

4.6 Multi threading strategy

Our multi threading strategy is guided by a few main principles.

1. The workloads involved with DEM are inherently imbalanced when considering the broad phases of the algorithm. Therefore, whenever possible separate the work into units, avoid processing units where work doesn't need to be done (even if this requires additional administration later) and spin off all other units as tasks. A tasking runtime is then left to handle the distribution and balancing of the workloads.

2. Where a workload is over an interaction graph we look to use graph colouring to allow parallel operations on the graph. Particularly where this operation is on the critical path we optimise purely for the time to solution even if extra work is being done as a result.

3. At the lower levels of the algorithm where vectorisation might be possible we want to create and pack together as much work as possible. Unused vector lanes cannot be utilised by other processes so we prioritise time to solution for large batches rather than the time to solution for individual instances.

The final principle is primarily applied to contact detection as discussed in Chapter 5 so we won't address this further here.

The first principle gets applied in our work through the use of OneTBB work groups. Work groups are able to be nested, which is important for our approach to load balancing the DEM algorithm due to the variable nature of the workloads. For our use case tasks can branch further sub-tasks but all sub-tasks always rejoin the parent task before the parent task ends. This is by no means a fundamental constraint of task based parallelism and could be a potential avenue for future optimisations but this task fork-join pattern makes the algorithm easier to reason about. In our application we never have the sub-task of one task be dependant on the sub-task of another task. Tasks and all their descendant tasks are always independent of every other task that is currently in the pool of available tasks. While this strategy of nested tasks provides an effective way to load balance (across a single node) it can also lead to excessive overhead and so requires some tuning of the grain size of tasks (Appendix B.1).

A common pattern that we see appear throughout the DEM algorithm (often when we use an iterative matrix free alternative to what would otherwise be a matrix assembly and inversion) is the requirement to perform an operation on each vertex (particle) or edge (contact) in the interaction graph where data on vertices or edges needs to be accessed by the task being performed on neighbouring elements. Furthermore, if these operations are being performed on the global interaction graph then there won't be any other parallel work happening simultaneously to mask this work and so these operations become bottlenecks with poor resource utilisation. Therefore, we employ a simple implementation of the Welsh-Powell greedy graph colouring algorithm (Appendix 2.1.4) and the process all elements with the same colour in parallel.

4.6.1 Parallel cluster separation

Identifying clusters of particles that have the potential to interact within the time span of interest forms a core part of our algorithm and this operation is performed at least once per time step. For larger numbers of particles and interactions this step can begin to become very expensive. Initially we utilised the *connected_components* function from the libIGL library [148]. However, this requires an adjacency matrix to be constructed and then the results need copying back into the internal cluster format used by Delta2. Furthermore, in our target applications we expect to see large numbers of clusters with relatively small numbers of particles in each.

At this stage of the algorithm we are unable to progress with other work until the clustering has been completed so the full processing resources available could be used for this stage. The libIGL function only utilises a single thread.

Due to the single threaded nature of the library function and the added overhead of building the adjacency matrix and decoding the result we instead decide to implement our own solution.

Algorithm 2 Parallel graph clustering algorithm

Input
G = (V, E)
Output
Cluster id over vertices C

$C \leftarrow \{1, ..., |V|\}$
$\mathbb{E} \leftarrow$ colour_graph_edges(E)
halt $\leftarrow$ False
while not halt **do**
    halt $\leftarrow$ True
    for $c \leftarrow 1, ..., |E|$ **do**
        for parallel $i \leftarrow 1, ..., |\mathbb{E}_c|$ **do**
            if $C(\mathbb{E}_c(i)(0)) < C(\mathbb{E}_c(i)(1))$ **then**
                $C(\mathbb{E}_c(i)(1)) \leftarrow C(\mathbb{E}_c(i)(0))$
                halt $\leftarrow$ False
            else if $C(\mathbb{E}_c(i)(1)) < C(\mathbb{E}_c(i)(0))$ **then**
                $C(\mathbb{E}_c(i)(0)) \leftarrow C(\mathbb{E}_c(i)(1))$
                halt $\leftarrow$ False
            end if
        end for
    end for
end while
return C

With a large number of cores available and clusters that we assume are likely densely connected in small clusters we use a simple iterative algorithm (Algorithm

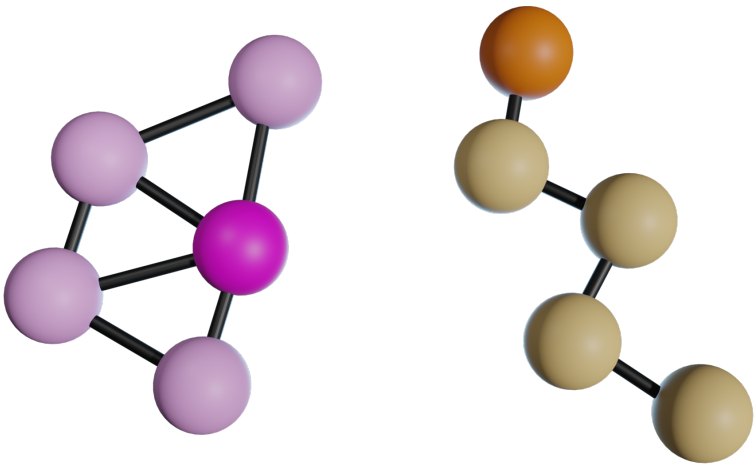


Figure 4.6: The best case (left) and worst case (right) arrangement of five particles. The darker coloured node on each side indicates the node with the lowest index. The left configuration takes a single iteration to propagate the lowest index to all nodes. The right configuration takes four iterations.

2).

We iterate over every contact and 'colour' both particles involved with the id of the particle with the lower id. We repeat this process until there were no changes in 'colour' for a complete iteration of all the contacts. Possible race conditions are mitigated by colouring the contacts (Appendix 2.1.4) such that no two contacts with a shared particle have the same colour.

This method has the advantage of being very simple to implement and, for the expected workloads, takes advantage of the available hardware. In the best case scenario every particle shares a contact with the particle in its cluster with the lowest id (Figure 4.6 left). This case requires just two iterations of the contacts (one to set all the ids and one to check that nothing more changes). In the worst case scenario (Figure 4.6 right) the particle with the lowest id appears at the end of a chain. This case will require N iterations (where N is the number of particles in the chain). The first iteration will correctly colour the second particle in the chain, the second iteration the third particle etc. The $N-1$ iteration will correctly colour the last particle in the chain and the N^{th} iteration will confirm no more changes.

The worst case scenario for the number of iterations required by this algorithm is equal to the number of particles. The work within a single iteration can be split

across a large number of cores. The work units (inspect a contact and assign the minimum 'colour' to both of them) are extremely cheap so we require a vast number of contacts to utilise hundreds of available cores. For scenarios small enough to fall short of this scale criteria the time to solution is insignificant so we don't concentrate on the performance or efficiency here. For large scenarios the cluster sizes we expect to see are small and densely connected compared to the total number of contacts so the work is effectively parallelised. Furthermore, the contact resolution stage will likely involve some process of iteratively transferring the effects of interactions along chains of interacting particles (even if this isn't specifically part of the method this behaviour will emerge from the requirement to have smaller time steps later) that will similarly increase in the number of required iterations. An iteration of contact resolution is substantially more expensive than an iteration of colouring so in the case of long chains the tiny added cost of a few more iterations of colouring is lost compared to other phases and in the case of more densely connected clusters the time to solution is close to constant for all sizes of problem that we study.

4.6.2 Parallel collision detection

Our aim is to be able to handle extremely dense meshes and so the collision detection stage can become a bottleneck. This is particularly true when there are a small number of very complex meshes that only occasionally come into contact. This results in highly imbalanced workloads and many threads sitting idle. To take advantage of these unused cores we introduce tasking into the traversal of the mesh hierarchies. When traversing and comparing a pair of mesh hierarchies we maintain a list of nodes that need comparing in the next batch. To begin with this list contains just the pair of root nodes. As we explore the tree this list may grow to include many pairs of nodes - one from the first mesh hierarchy and one from the second. Instead of immediately exploring the children of any node pair that results in a potential contact we add the potential pairs to the batch to be processed next. Once our current batch has run out we move the next batch list into the current batch list and repeat. By doing this we defer the work that needs doing until larger batches can be processed together. We then split up this work over a number of tasks and each task

combines all the triangle-triangle checks it must do so that it's all computed in one rush and vector efficiency is maximised. If there are a large number of simultaneous particle-particle collision checks happening then it is possible all these tasks will be performed on the same core. The deferred triangle-triangle checks still pay off in this case because the vector lanes are used to maximum efficiency. Where there are a small number of complex particles interacting and causing a bottleneck then the added parallelism here can help alleviate this.

4.6.3 Parallel time stepping

At each 'time step' we identify a number of clusters to advance. To maintain the property that all particles must have overlapping time spans to be able to roll back the $t^{(\text{current})}$ to a matching time it is required that the cluster that defined the new minimum $t^{(\text{old})}$ advances. All the other clusters that are valid candidates for advancing can but do not have to advance. If they do advance then they are (for this time step at least) entirely independent of the other clusters so can be processed concurrently.

For each cluster that can advance we spawn a new task and then wait for all tasks to complete. We observed that scenarios with a similar number of clusters trying to advance each time step as the number of core resulted in a large amount of idle processor time. For example, imagine the scenario where 101 identical clusters are ready to process on a 100 core machine. 101 tasks are created and each of the 100 threads takes one task. They all finish at approximately the same time and one of the threads grabs the last task. While this one core processes the final cluster the other 99 threads remain idle.

We investigated a solution where we limited the number of clusters that could be processed at each step to αt where t is the number of threads and α is a tuning parameter (set to 2 in our case). By imposing this limit we hoped to balance the requirement for enough work at each step to both initially occupy all the threads but also provide some load balancing capability for imbalanced loads while smoothing out the number of clusters that can be processed at each step by deferring some of the work until the next step.

In practice we observed that there was little difference to performance. While the number of time steps where large imbalances occurred appeared to be reduced the overall time to solution was largely unaffected. This has lead us to believe that for our workloads the imbalance in cost per unit of work is so great and the total number of clusters advancing per step relatively limited that by further limiting the number of clusters that can be processed at once we are harming the existing load balancing mechanic for this step to an extent that can sometimes be significant compared to the improved load balancing in a future step. Furthermore, by delaying a cluster advancing, which may have enabled a different cluster to advance in the next step, we can sometimes increase the total number of steps required. Each step recomputes the broad phase, clustering and time step selection (although there are optimisations for each that avoid expensive recomputations where there has been no change) so adds additional overhead.

This approach to parallelism is effective for large numbers of small to medium sized homogeneous clusters. It is insufficient on it's own to occupy many cores when the clusters vary widely in cost to process and/or when there are a small number of clusters. However, in both these cases this top level separation of work units becomes the foundation for further separation of work.

4.6.4 Parallel sequential impulses contact resolution

One of the benefits to the traditional sequential impulses algorithm is that reactions introduced at one point in the system can quickly propagate to all parts of the solution. For example, a tower of stable objects. Before any impulse computations are done it may appear as if all particles are in free fall since there is no relative velocity between the particles but there is gravity acting equally upon all of them. However, the particle at the base of the stack will receive an upwards 'push' from the ground it is resting on. The tower will eventually converge to a stable state where the upward impulses counteracts the influence of gravity. In the best case where the lowest contact is computed first and then progressing up the chain then some amount of the impulse from the ground will be propagated to the very top particle in the stack after just one complete iteration. If instead all the impulse responses

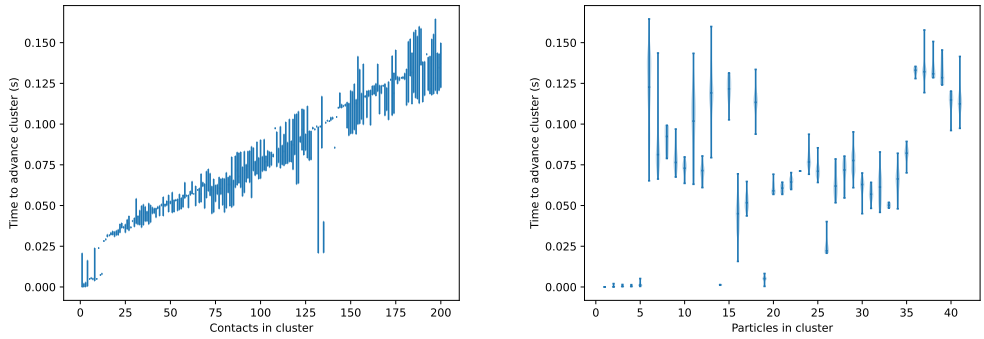


Figure 4.7: The cost (seconds) per cluster based on number of contacts (left) and particles (right). The strong correlation between number of contacts and the time to solution makes it a better heuristic for the grain size of work compared to the number of particles.

were calculated before any of them are applied (perhaps referred to as "Impulses" rather than "Sequential Impulses") then it will take N iterations (where N is the number of particle in the stack) for any of the influence from the ground reaches the top particle.

The idea of processing a cluster (where we know each particle is reachable by any other particle by taking a series of hops along contacts and so the solution state for any particle in the cluster could affect the solution of any other particle) in parallel appears to counteract the benefits of this fast propagation of impulses. However, the wasted hardware resources resulting from this expensive serial section of the algorithm is too significant to ignore. Instead we choose to process the contacts in parallel using a small number of threads compared to the number of contacts. This allows close to correct solutions to propagate quickly through the cluster while better taking advantage of the available hardware. A small increase in the number of iterations required to converge is acceptable if each iteration is substantially faster due to more threads doing the work. We observe that the time to solution is correlated to the number of contacts per cluster and not the number of particles (Figure 4.7). Therefore we experimentally determine a ratio of contacts to threads for optimal time to solution. Due to this limit in number of threads used we are unable to scale well with just a single cluster. However, when many clusters are all producing tasks we are likely to be able to saturate the threads with useful work a

large portion of the time.

Since contacts can share particles with other contacts it becomes possible to introduce a data race condition. To avoid this we colour the edges such that no two edges are the same colour if they share a particle and then process all contacts with the same colour in parallel. The effectiveness of this pair of algorithms together is demonstrated by the OpenCL implementation of the Bullet physics engine [91], popular in the games industry for its extreme speed and stability (even with very low accuracy caused by using a very small constant number of iterations). However, for objects more complicated than spheres this regularly leads to more colours than expected due to multiple contacts between individual pairs of particles. A cube on a plane, for example, could have four contacts (one in each corner) and so using the basic graph edge colouring we would get four colours and serial evaluation of the contacts. However, this is an even worse problem because in this example case the cube will simply rest on the surface but when we compute a single iteration of Sequential Impulses the first corner will be given a large 'kick' upwards to counter the gravitational acceleration for the whole object. This will cause the cube to spin. As more iterations proceed the solution will converge back to a steady state but it may take many more iterations than should be necessary. We employ a common solution to this problem and weave the solution into our parallel implementation. All contacts that share the same particle pair are combined into a bundle. The updated impulses of each contact in a bundle are computed simultaneously and applied in one rush.

4.7 Efficacy and correctness

The proof that $t^{(\text{old})}$ is always valid is technical yet brief with a combination of an induction over the number of particles plus a contradiction. As we terminate our algorithm as soon as the global time stamp $t^{(\text{old})}$ exceeds the terminal time, the theorem addresses the correctness of the simulation outcome.

Proof 1 *The correctness is trivially true for one particle.*

Let all state transitions be globally enumerated. We use the counter n for the

enumeration. Assume that the tuple $(t_n^{(old)}, t_n^{(current)})(p_i)$ *produced for particle* p_i *in step* n *is invalid: it should not have been made, as we dropped an old solution in this step to which we should have rolled back. This formalism implies that step* n *triggered the state transition*

$$(t_{n-1}^{(old)}, t_{n-1}^{(current)})(p_i) \mapsto (t_n^{(old)}, t_n^{(current)})(p_i) \quad \text{with } t_{n-1}^{(current)}(p_i) = t_n^{(old)}(p_i). \tag{4.10}$$

By induction, we know that $1 \leq i \leq |\mathbb{P}| - 1$ *and that it has been particle* $p_{|\mathbb{P}|}$ *which would have crashed into* p_i *at a time* $t^{(collision)} < t_n^{(old)}$*. This is the collision we missed.*

p_i *and* $p_{|\mathbb{P}|}$ *have been in different clusters in step* n*. Otherwise, they would have advanced in sync. Therefore, there is a step* $m < n$ *in which* $p_{|\mathbb{P}|}$ *has updated its position the last time.*

$$t_m^{(current)}(p_{|\mathbb{P}|}) = t_{m+1}^{(current)}(p_{|\mathbb{P}|}) = \ldots = t_{n-1}^{(current)}(p_{|\mathbb{P}|}) = t_n^{(current)}(p_{|\mathbb{P}|}) \tag{4.11}$$

from thereon, i.e. we haven't updated this particle anymore since then. According to our assumption, we learn about this missing collision after step n*, but the actual collision happened at* $t_m^{(current)}(p_{|\mathbb{P}|}) < t^{(collision)} < t_n^{(old)}(p_i)$*. Now we cannot roll back anymore. This can happens if and only if particle* $p_{|\mathbb{P}|}$ *lacks behind.*

p_i*'s transition in step* n *is allowed if and only if the other particles in their clusters have not been left behind. This is a contradiction to the time constraints on* $t^{(collision)}$ *above.*

□

With a detailed description of the individual algorithmic steps, we can indeed show that our algorithm terminates.

Lemma 1 *The time step sizes of active particles are significantly bigger than zero, i.e. they do not deteriorate.*

The lemma can only be proven once we assume that the kinetic energy of a system is not growing, i.e. as long as the system is closed and no energy is injected.

Proof 2 *Let a particle that is not subject to changes of its kinetic energy follow a trajectory of straight line segments $l_0, l_1, l_2, \ldots$. This is the analytic trajectory, i.e. no two line segments point into the same direction. To contradict the lemma, $|l_n| < |l_{n-1}|$. Such a situation can arise if a particle, for example, falls into a funnel or hopper and bounces in-between the walls coming closer and closer, until the particle finally gets stuck in-between walls.*

We first recognise that the reduction above means $\lim_n |l_n| = 0$, as our particle interaction model is subject to friction. A particle notably cannot bounce forth and back between two walls without slowing down.

However, to construct such a trajectory is not possible for our algorithm, as the particles are surrounded by an ϵ-layer. Hence, the segment length is bounded by $|l_n \geq \epsilon$ or two particles are in a "permanent" contact, i.e. stick to each other. In the latter case, the interaction is explicitly removed from the time step size calculation. □

The lemma formalises the $\gg$ in (4.3). It becomes clear that friction works in our favour in this context: As particles slow down due to friction after each collision, even constant path segments would lead to larger time intervals in-between collisions. Even without friction, our ϵ-formalism effectively truncates analytical trajectories of line segments which would approach zero. One can still construct arbitrary long sequences of time step sizes approaching zero if a system is permanently stimulated by injecting additional kinetic energy. We neglect such setups.

Observation 4 *The semi-implicit collision model, where the contacts are prescribed and only impulses are implicitly determined avoids time step size degeneration and bouncing particles when we encounter compact, settled particle geometries.*

With a smooth spring force, time step sizes have to be dramatically reduced as long as two objects continue to approach each other despite the actions of the force. As a consequence, global assemblies of objects with multiple contact points will rarely reach an exact equilibrium. They will always oscillate around the steady state solution as long as an external force such as gravity holds them together or otherwise disassemble. While a force-based formalism may have its advantages,

such vibrating assemblies of objects are problematic for local time stepping, as they lead to an effective time step size degradation. It is hence not only an algorithmic decision to determine the contact points and impulses implicitly each—though the overall schemem remains explicit as these to ingredients are not coupled—but it is essential to make the local time stepping work.

Lemma 2 *The particle with the smallest $t^{(current)}$ is always a member of an active cluster.*

Proof 3 *We assume the lemma were wrong an construct a contradiction: Let $t^{(current)}(p_i) = t^{(min)}$, i.e. let p_i determine the global minimum time stamp. Another particle p_j carries $t^{(current)}(p_j) > t^{(current)}(p_i)$. Let p_i and p_j end up in different clusters, were they both determine the minimal cluster time stamp $t^{(min)}(\mathbb{P}_i)$ or $t^{(min)}(\mathbb{P}_j)$ respectively.*

If p_j vetos the advance of the $\mathbb{P}_i$,

$$t^{(current)}(\mathbb{P}_i) > t^{(collision)}. \tag{4.12}$$

However, $t^{(collision)}(p_j) > t^{(current)}(p_j)$ by definition.

□

What could happen however is that cluster $\mathbb{P}_i$ takes a large time step and overtakes the particles within $\mathbb{P}_j$, missing out on collisions at $t > t^{(\text{collision})}(p_j)$. In this case, it will roll back later to $t^{(\text{collision})}(p_j)$ as particles from $\mathbb{P}_i$ and $\mathbb{P}_j$ are joined into one cluster.

Lemma 3 *Every particle is updated after a finite number of steps, i.e. every particle becomes a member of an active cluster.*

Proof 4 *The lemma results directly from the previous lemmas: A particle p_i is not strictly advancing in time. However, let $T = \sum_{p_j}(t^{(current)}(p_i) - t^{(current)}(p_j))$ be a sum over all particles p_j with $t^{(current)}(p_i) > t^{(current)}(p_j)$. This sum is strictly monotonously decreasing with a decrease that bounded away from zero. After a finite number of steps, particle p_i is therefore updated.*

□

4.8 Runtime results

All experiments were run on AMD EPYC 7702 chips in a two socket configuration with 2×64 cores. They run at 2.0 GHz, while TurboBoost can increase this up to 3.35 GHz. Yet, AVX instructions make a core fall back to a reduced frequency of at most 1.8 GHz to stay within the TDP limits [149]. One EYPC node has access to 256 GB of main memory. Each socket is divided into 4 NUMA regions with separate memory channels. Each NUMA domain hosts 4 groups, each with 16MB L3 cache. Each group hosts 4 cores, each with 512KB L2 and 32KB L1 cache.

Our code was translated with the Intel oneAPI DPC++/C++ Compiler 2021.4.0 using the flags `-std=c++17 -O3 -march=native`, i.e. we tailored it to the native instruction set. Our realisation relies on CGAL 5.5.1 [150], Eigen 3.4 [151], Embree 3.13 [147], libigl 2.3 [148], SIMDe 0.7.2 [152], TetGen 1.6.0 [153], Catch2 3.1.0 [154] and GLFW 3.3.6 [155].

Definition 7 *In addition to simple triangulated spheres, cubes and planes our scenarios use* sphere-like particle shapes*, which result from a randomised parameterisation: We decompose the sphere with radius 1 into $|\mathbb{T}|$ triangles. The vertices on the sphere which span the triangles are subject to a Perlin noise function, which offsets the vertex along the normal direction of the surface. $\eta_r = 1$ adds no noise and thus yields a perfect, triangulated sphere with radius 1 where all vertices are exactly r unit away from the sphere's origin, where r is the desired base radius. Otherwise, the per-vertex radius is from $[r, r \cdot \eta_r]$. As we use a hierarchical noise model, a high η_r yields a degenerated shape which contains both low and high frequency detail.*

We consider the result converged when the update to the impulse and torque applied to every contact underruns a threshold of $1e^{-6}$. $\epsilon = 10^{-2}$ is uniformly used on the finest mesh level. This is a relative quantity, i.e. chosen relative to the particle diameter. For the plane, we uniformly use $\epsilon = 10^{-2}$.

4.8.1 Particle pairs

In a first experiment, we create two sets of the same number of particles which are arranged vertically yet spaced out, i.e. they do not touch each other (Figure 4.8).

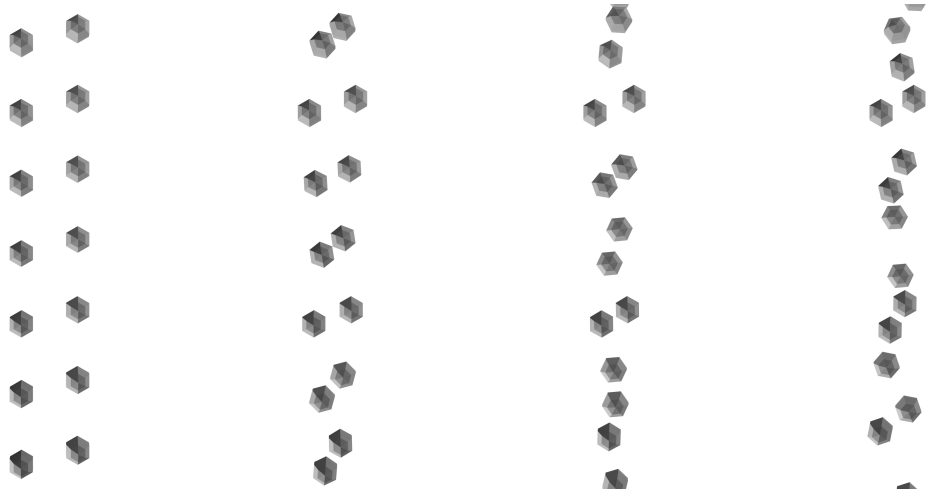


Figure 4.8: The *particle-pair* scenario from left to right: Starting from a column-wise arrangement, pairs of particles approach each other with randomised velocity. The pairs collide at different points in time and separate again.

These two columns of particles are set on a horizontal collision trajectory, such that always two particles bump into each other. As we randomly vary the initial horizontal velocities between the different particle pairs, the collision time stamps differ. After the collision of a particle pair, the particles separate again and may come into contact with other particles to form new pairs or small groups. We configure the initial geometric arrangement of this *particle-pair* scenario such that each particle collides exactly once with its counterpart, while each particle's geometry results from a triangulated sphere.

Each particle pair runs through three computational phases: While the particles approach, there is no collision and no forces act on the particles, as we omit gravity. When they collide, the system becomes very stiff suddenly and the particles exchange forces, before the objects repulse each other again and separate. As we use random initial horizontal velocities, the phases of three particles are not in-sync in any way.

Observation 5 *On both a single core and on multiple cores, our local time stepping outperforms global time stepping for the particle-particle setup with sufficiently many particles.*

Global adaptive time stepping for a large number of particle pairs on a multicore machine scales close to linearly while there remains unused CPU resources.

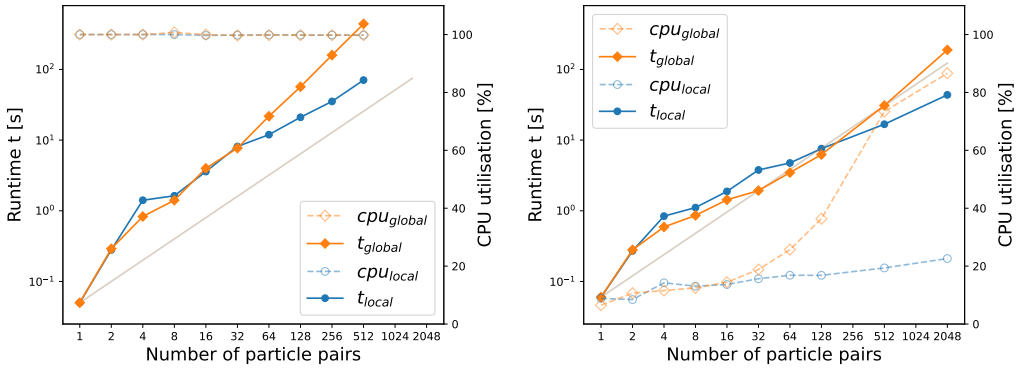
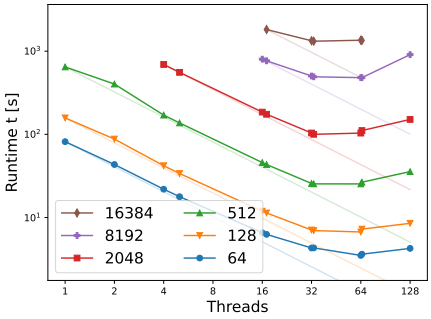


Figure 4.9: Runtime and CPU utilisation for the collision of particle pairs on a single core (left) and the whole node (right). The CPU utilisation is normalised against the number of cores available.



Phase	Run time 64 pairs	Run time 512 pairs	Run time 8192 pairs
Broad collision	0.9%	0.5%	1.0%
Clustering	0.2%	0.2%	0.7%
Time step size	82.6%	83.0%	82.7%
Narrow collision	16.2%	16.0%	15.0%
Collision	0.2%	0.2%	0.6%
Snapshot roll-over	0.0%	0.1%	0.0%

Figure 4.10: Left: Various strong scaling curves for the particle-pairs setup. Different numbers of particles are used with the local time stepping scheme. Right: Break down of the different algorithm phases.

Cluster topology, active clusters and concurrency level With $|\mathbb{P}|$ particles in total, the algorithm identifies between $|\mathbb{P}|$ or $|\mathbb{P}|/8$ clusters, i.e. the space-time checks merges at most four particle-pairs, which is by a factor of four worse than the theoretical optimum. Most of the time, the cluster topology remains invariant. As the particle pairs are totally out-of-sync, the clusters "diverge" in time quickly. Due to the global constraint on clusters advancing too far, this leads to a situation where only a few clusters can be updated per time step and materialises in a low CPU usage compared to global time stepping (Figure 4.9). The finding is not counterintuitve: our algorithm is unaware of which particles collide at most once and which might interact with other particles in the future and hence has to prohibit particles from "shooting off" after their first contact. The concurrent work packages (clusters) are evenly balanced, but only few of them may update: The total CPU utilisation

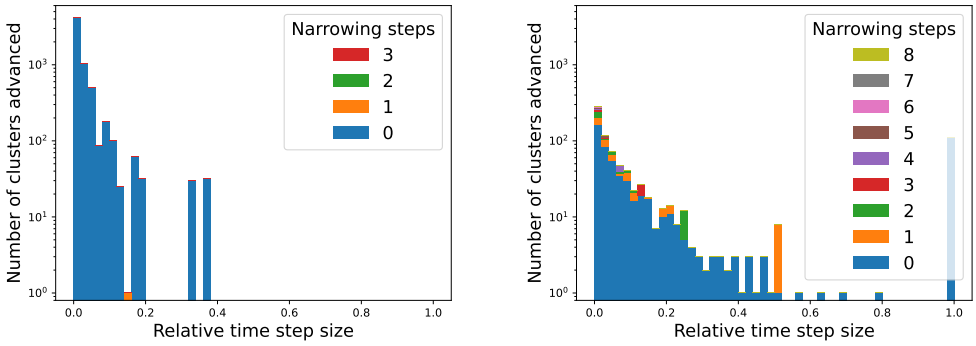


Figure 4.11: The number of clusters advanced in time using different sized time steps for a 50 pair run of the *particle-pair* scenario. A globally selected time step (left) results in a consistently small step size while local time stepping (right) allows single particle clusters to advance with the maximum allowed step size. Local time stepping exhibits a larger number of rollbacks and five additional layers of recursive rollbacks in the worst case. However, the total number of times clusters are advances is significantly reduced.

grows slowly in $|\mathbb{P}|$ and never exceeds 25% in our experiments. We hence struggle to exploit the parallel machine for small total particle counts. However, massive particle counts mitigate this problem to some degree and we get reasonable weak scaling (Figure 4.10).

Roll backs and time step size distribution The time step size chosen depends on potential collisions and hence correlates directly with the initial velocity distribution. Due to the simple cluster topology, few rollbacks are required. The distribution of the time step sizes mirrors the three computational phases, i.e. during the approach the time step size remains at the full size, there is then a sudden drop to a small time step sizes to resolve the collision and this is followed by a rapid increase of step size back to the full interval. The peak at 0.5 suggest to us that the time step size approximation is less effective for large time steps. This results in an invalid state, which will trigger an automatic halving of the step size before reattempting to approximate a more specific time for the contact.

Contact points and particle interactions The particles used are of identical size and mesh density: Each sphere has a radius of 1, uses $|T| = 48$ and $\epsilon = 0.01$.

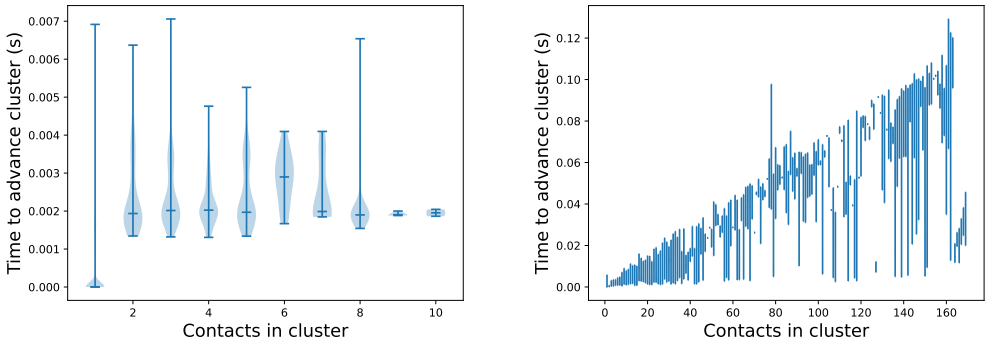


Figure 4.12: For the *particle-pair* scenario where clusters never contain more than 10 contacts there is no clear relationship and a amount of noise between the number of contacts and the time it takes to advance that cluster (left). For a different scenario (*particle-stack*) where there are significantly greater numbers of contacts we observe a clear but noisy linear trend for the maximum time a cluster with a given number of contacts will take to advance (right). For a cluster of any number of particles it is always possible that no contacts result in a transfer of momentum so requires just a single cheap iteration through the contacts before completing.

As we use triangulated objects it is possible to have more than one contact resulting from a sphere-sphere interaction. We observe a maximum cluster size of 5 particles and 12 contacts per cluster.

Discussion For small particle counts, we obtain a superlinear cost curve relative to the particle cardinality. It is not clear if this is due to caching effects—few particles apparently fit into close-core caches—a significant management overhead or a combination of both. As the particle count increases, any overhead is amortised, and the local time stepping cost curve starts to follow a linear trend, whereas the global time stepping cost continues to follow a superlinear trend. Parallelisation subsequently helps to reduce both the curves' growth. The global time stepping to a linear trend and our local time stepping to sub linear (while CPU resources remain available).

Our setup is of artificial character. However, we note that many subsequent experiments lead to small particle groups "spinning off" from the main bulk of the objects. As they fly away, our cluster analysis manages to identify arising concurrency. However, such objects only add a very limited amount of extra parallelism,

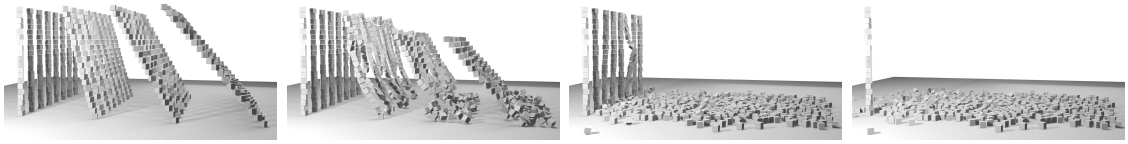


Figure 4.13: The *particle-stack* scenario from left to right: The initial state before the towers begin to topple. The stacks on the far left settle in a stable tower, the stacks towards the left (with the shallow angles) begin to topple over while the stacks on the right are close to free fall. The particles from the collapsed towers disperse and some hit the stable towers causing them to topple. A final stationary state is reached with all particles either resting on the floor or part of a stable stack.

as the local time stepping constrains their movement. We expect forking off mini-clusters not to affect the performance severely, but we cannot guarantee that they help us to exploit otherwise idle compute resources.

4.8.2 Particle stacks and the tower setup

In the *particle-stack* setup, we study a series of stacks of cubes, which are initialised with a variety of slanting angles. All but the left-most stacks are unstable, each stack topples over and interacts with the other stacks. Eventually, most stacks have collapsed, and the cubes end up scattered over the floor (Figure 4.13).

Again, our simulation progresses through three distinct phases: Initially, the particles aka cubes within their cube stacks are almost stable. They stiffly interact with their adjacent cubes and the cube in itself is almost at rest. It is the whole stack which initially tilts. In the second phase, which is more chaotic, cubes switch from free fall into situations where they crash into cubes from other stacks or cubes from their own stacks before they distribute over the floor. In the third and final phase, the cubes "roll" over the floor before they finally come to rest. Fewer stiff cube-cube interactions arise from this final phase.

In our simulation setup, the ground plane is modelled by two $|\mathbb{P}| = 2$ triangles, and the cubes use $|\mathbb{P}| = 12$ with $\epsilon = 0.01$. We scale this experiment by increasing the number of rows of stacks (with 20 cubes per stack and 4 stacks per row). The total length of the simulation is 3 seconds.

Observation 6 *Except for small setups with very few towers, local time stepping*

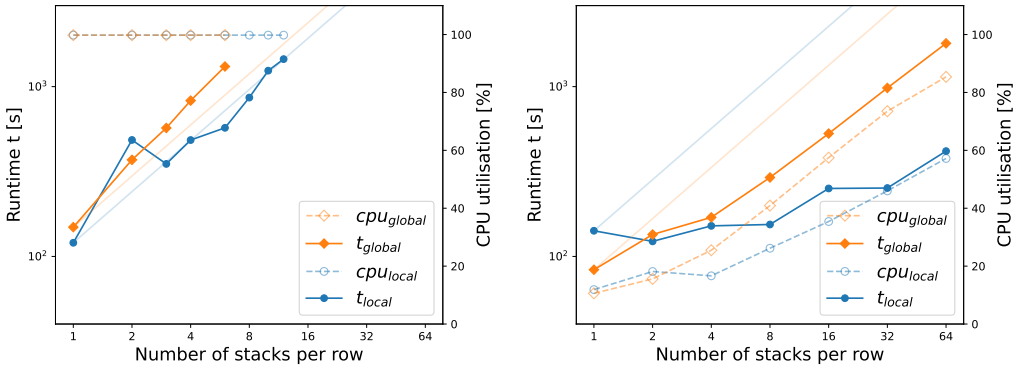


Figure 4.14: Runtime and CPU utilisation for the collision of the particle/cube stacks on a single core (left) and the whole node (right).

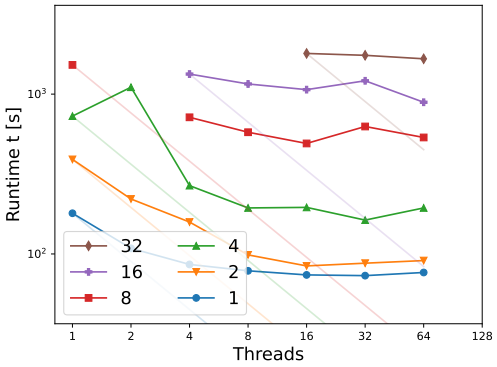


Figure 4.15: Various strong scaling curves for the particle stacks with increasing number of rows of stack.

outperforms the global time stepping approach on a parallel computer. On a single core, the effect is less pronounced.

Cluster topology, active clusters and concurrency level As long as the particle stacks remain more or less "intact", i.e. while they tilt, we obtain around one cluster per stack. This is due to the fact that the stationary floor is explicitly excluded from the cluster construction. For the final stationary setup, we get a number of clusters consisting of many single particle cluster where the particles end up scattered over the floor and a small number of large clusters where the falling particles have ended up piled together. In-between these two phases, the cluster topology is quite dynamic. It is this in-between phase, where the overall system is rather stiff and asks for globally small time step sizes. However, individual

subclusters of particle constellations might nevertheless advance aggressively in time. Therefore, we see the local time stepping outperform a global time stepping once the particle count is sufficiently high.

Furthermore, compared to a single core run where local time stepping scales close to linearly with the number of stacks the multicore local time stepping run scales sublinearly (Figure 4.14) by taking advantage of an increasing number of cores as the available work increases. Global time stepping scales superlinearly with just a single thread and so we would expect to see a return to this trend once the CPU utilisation reaches 100% for the multicore experiment from the approximately linear trend before that point.

Roll backs and time step size distribution At the same time, this frequent change of cluster topologies implies that the scalability is not particularly high, independent of the total particle count (Figure 4.15). At any point in the intermediate phase, many particles either are associated with inactive clusters, may have to roll back in time, or may only advance with tiny time steps.

Contact points and particle interactions In the first and third simulation phase, particles on average experience either one or two contact points. The solution of the arising non-linear equation systems is rather straightforward. In the middle phase, some particles may face many stiff contacts within a single time span. The fact that the arising equation systems become difficult to solve and (potentially) require many iterations (Figure 4.12) contributes further to the fact that the setup's scaling is limited, as massive iteration counts coincide with phases of low concurrency.

The tower setup We confirm our obsergations through the *particle-tower* setup, where we arrange rings of 32 cubes into layers on top of each other. Similar to bricks, they yield a tower-like construction. A small, heavy and very fast moving object crashes into the bottom of the tower and makes it collapse (Figure 4.16).

Initially, the tower is in a steady state. Its bricks (particles) do not move. As the projectile hits the tower, a shock travels through the structure as the particles

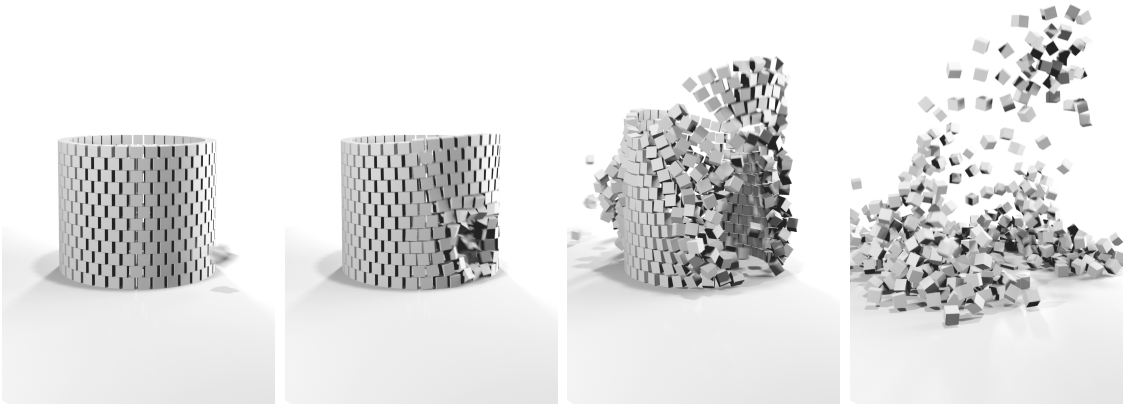


Figure 4.16: The tower scenario from left to right: Rings of cubes form a steady tower, when a single sphere-like particle approaches its base with a high speed. The fast moving particle destroys the tower.

are incompressibe. Cubes in the tower use $|\mathbb{T}| = 12$, the ground plane uses $|\mathbb{T}| = 2$ and the projectile uses $|\mathbb{T}| = 80$. We scale this experiment by increasing the height of the towers.

Observation 7 *The tower yields a computational character that is an extreme case of the stacks: Two clusters "suddenly" interact and the simple two-cluster topology is completely destroyed, before the simulation settles and yields scattered particles which come to rest.*

With a dense packing of the tower bricks, the setup is either at rest or extremely stiff, or the shock has separated off particles already. Compared to the particle stack setup, we have a simpler topology, and this topology remains intact over a longer time span. Therefore, the global time stepping always outperforms its local time stepping counterpart by a fixed proportion (Figure 4.17). We effectively skip the middle phase from a time stepping point of view. As we only enrich one cluster with more and more particles, we see a linear increase of the compute cost for both local and global time stepping. We do not observe an additional speed-up at any problem size. Due to the dense interactions in large portions of the simulation the majority of the particles are in contact with each other (directly or through others) resulting in a time stepping scheme which is effectively globally selection. In this case the

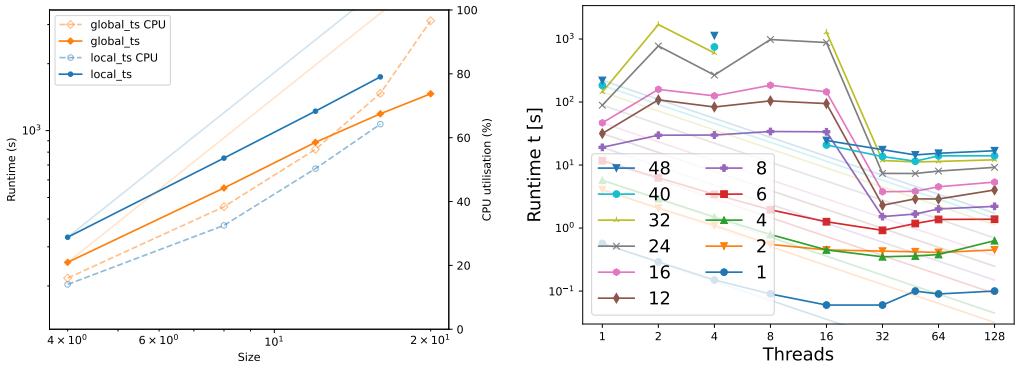


Figure 4.17: Left: Benchmarking of local time stepping vs. global time stepping on a whole node. Right: Various strong scaling measurements for the tower setup, where additional particles result from additional layers (rings) on top of the tower.

majority of the simulation time is spend detecting the large number of contacts and then resolving the resulting contact constraints for large clusters with small time step sizes. Any particles thrown out of contact with the others are temporarily in free fall (allowing a large time step and no contact detection) and make up a negligible portion of the compute time. If a particle like this comes back in contact with the larger clusters it will trigger a rendezvous and potentially undo work done on the larger cluster. Overall, the system struggles to scale. For reasonably large setups, we only gain performance once we add a second memory controller; which is stereotypical for bandwidth-bound codes.

Discussion Our algorithmic design is guided by the geometric identification of clusters. If such a cluster topology ceases to exist, we can not exploit a spatial decomposition anymore to obtain a scaling code. Contrary, such a transition phases from a topologically stable particle arrangement into an anarchic collision pattern helps the local time stepping to shine. In this context, we conclude that cluster decomposition and local time stepping show their strenghts in different constellations.

4.8.3 The hopper

In our hopper scenario, a collection of particles of randomised distorted spherical shape are dropped through a hopper and come to rest in a pile on a plane underneath.

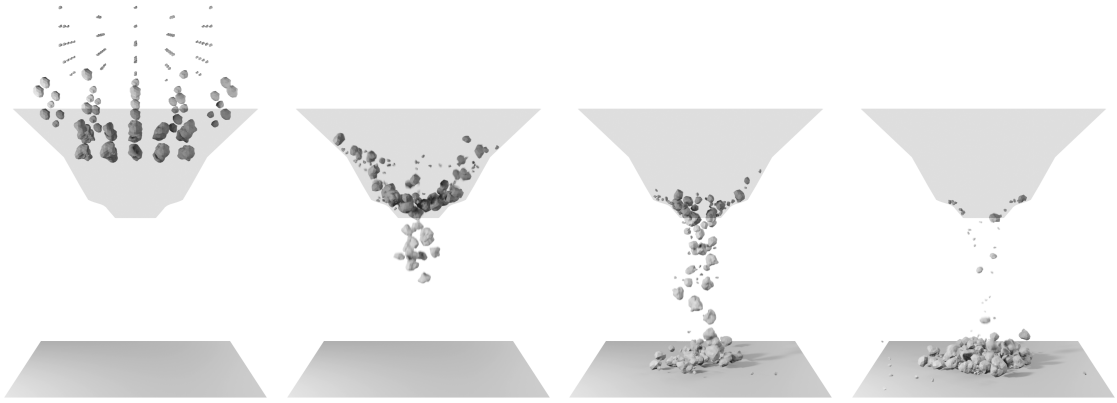


Figure 4.18: Hopper setup from left to right: The initial state with particles are released above a hopper. They drop into the hopper where they block each other. Eventually, the hopper empties to create a pile directly underneath.

Dropped particles begin in free fall with no interaction with surrounding particles before they enter the hopper. As they bounce from the hopper walls, and as more and more particles fall into the mouth of the hopper over time, they interact both with each other and the hopper's walls. This results in a slowed-down flow or even some kind of blockage. This effect is more pronounced whenever large particles arrange around the hopper outflow. We "encourage" such complex interaction patterns by arranging particles of decreasing size and complexity above the hopper. That is, large, complex shaped particles are dropped in first.

The quasi-spherical particles range from small grains ($|\mathbb{P}| = 80$ and a radius of $r = 0.025$) to larger rocks ($|\mathbb{P}| = 320$ with a radius of $r = 1$). The hopper has an opening at the top of a edge length of 24 and narrows down at the bottom to 4.25. It is modelled via $|\mathbb{P}| = 1,856$ triangles. All particles and the hopper use $\epsilon = 0.01$. We run the simulation only for the time until the last particle has hit the hopper.

Observation 8 *For the hopper setup, the local timestepping struggles to keep pace with global time stepping unless the available processor resources become limited. For example, with a large number of particles. The strong and weak scalability however are very good.*

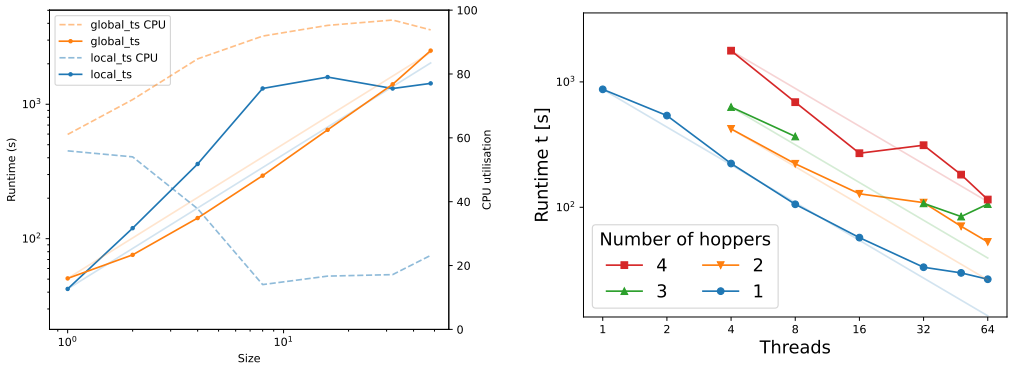


Figure 4.19: Runtime and CPU utilisation for an increasing number of copies of the hopper setup on the whole node (left) and strong scaling curves for multiple copies of the hopper setup (right).

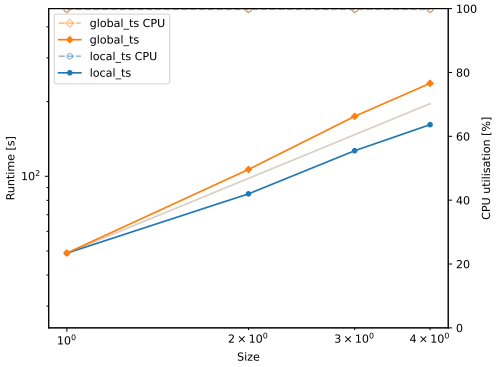


Figure 4.20: Runtime and CPU utilisation for an increasing numebr of copies of the hopper setup on a single core.

Cluster topology, active clusters and concurrency level The hopper simulation starts of with around $|\mathbb{P}|$ clusters, but as the particles assemble within the hopper, we obtain larger clusters. This is counteracted by small step sizes from complex interactions reducing the possible interaction range of any single particle in a given step and so still allowing more than one cluster to form, advance independently and then rendezvous with a larger cluster (Figure 4.23). The mix of large tightly interacting clusters and single pair clusters caused by short time span complex interactions can result in a workload imbalance. Consequently, as the number of hoppers is increased the global time stepping scheme scales close to the expected trend (Figure 4.19) as all particles are advanced at the smaller step size anyway. The local time stepping scheme struggles to generate enough parallel work while either waiting for the clusters with the smallest step sizes to advance and blocking other clusters from advancing. As the number of hoppers increases we end up with a larger number of clusters that can usefully advance in a given step so the available processor cores are more effectively utilised. Local time stepping struggles to help and actually imposes some overhead, unless the particle count becomes extremely high.

Roll backs and time step size distribution Once the majority of particles are moving through the mouth of the hopper there will be only one bulk of particles, many of the particles within the hopper advance in time with a similar time step size. The local and global time stepping yield the same update pattern. Unfortunately, the few particles that have already fallen through the hopper, still have not clashed into the main assembly and particles pairs with a sufficiently small time step size to be separated from the main bulk of particles yield a very imbalanced overall cluster workload. When the smaller 'break off' clusters are able to advance they then trigger rendezvous roll backs when reintroduced to the bigger cluster(s) (Figure 4.23). The bulk of particles resides in a small number of big clusters, but very few others can advance independently. Due to such an imbalanced decomposition, our task-based parallelism yields a very low CPU utilisation.

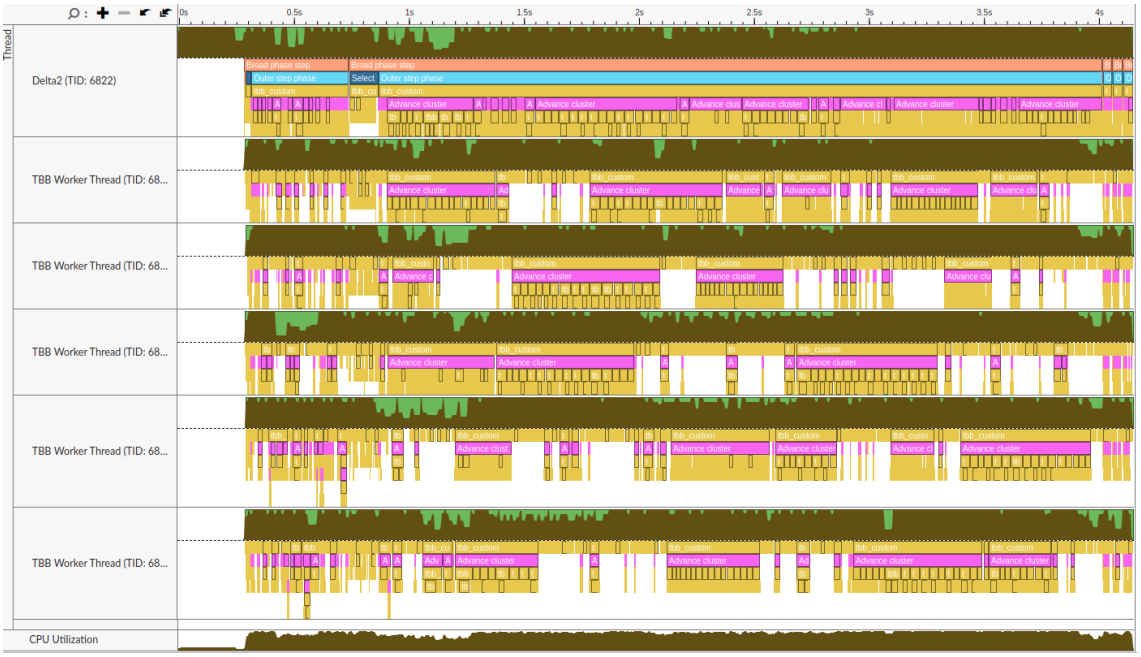


Figure 4.21: VTune analysis of the hopper's first five compute steps using six threads. The steps are of extremely varying length. The long step has an elongated time step selection phase (dark blue), followed by the individual cluster updates.

Contact points and particle interactions Tracking the algorithmic work distribution over time highlights that some very compute-heavy time steps take turns with cheap time steps, in which the few free-falling particles are updated (Figure 4.21).

Particles on a staircase We confirm our statements with additional runs where we let many particles hop down a staircase. The staircase is made up of 20 planes, each consisting of two triangles. For the particles, we use exactly the same configuration as for the hopper setup (Figure 4.24). Whenever particles hit the staircase, they either come to rest at their step or continue bouncing downwards. On this trip, they can hit further particles and knock them off their respective steps where they might have come to rest already. Eventually, all particles are either settled on the flat steps or have fallen over the side or bottom.

Discussion While the staircase scenario exhibits many of the same traits as the hopper the greater freedom of the slope compared to the neck of the hopper leads

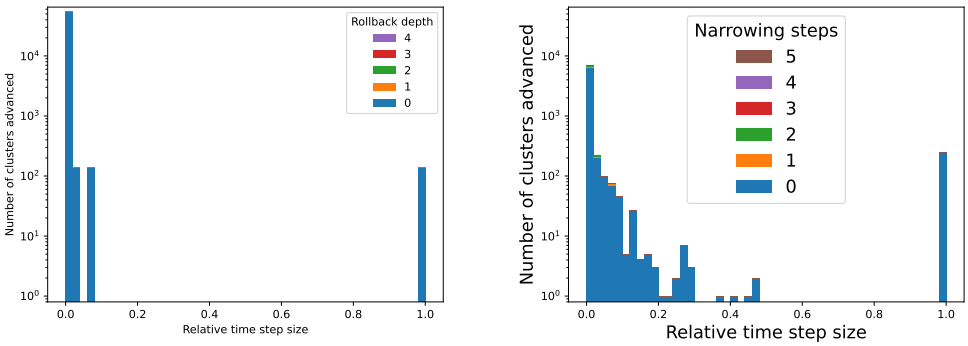


Figure 4.22: The number of clusters advanced in time using different sized time steps for the first second of a single hopper run. A globally selected time step (left) is dominated by the smallest time step sizes. During the period of free fall at the beginning of the simulation results in the spike at 1.0 but as soon as any particles are interacting with the hopper or each other then the global time step size gets forced down to a very small value. This free fall results in an only slightly larger number of full steps when using local time stepping (right). However, the number of time steps taken using the very smallest step sizes is drastically reduced and instead a variety of larger step sizes are used.

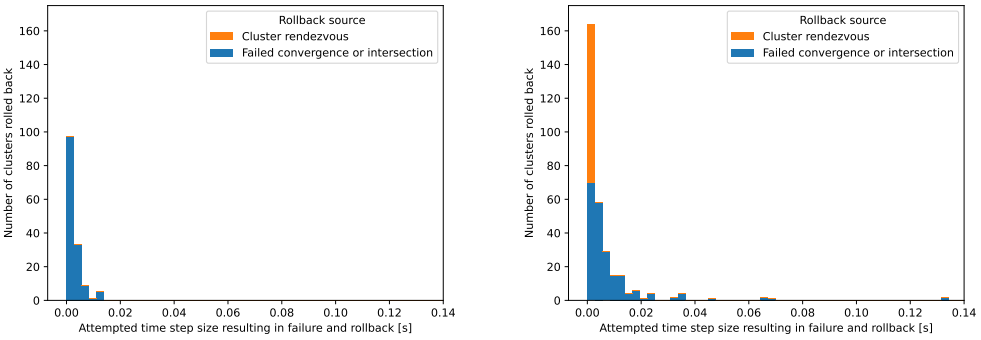


Figure 4.23: The number of clusters rolled back while using global adaptive time stepping (left) and local adaptive time stepping (right) for the first second of a single hopper run. A globally selected time step results in very few roll backs and since all particles advance in lock step there is never a need for roll backs to rendezvous the current time stamp. A local time step exhibits a larger number of rollbacks and requires a number of rendezvous roll backs. Compared to the total number of time steps taken (Table 4.26) there are a very small number of time steps that are "wasted" by rolling back.

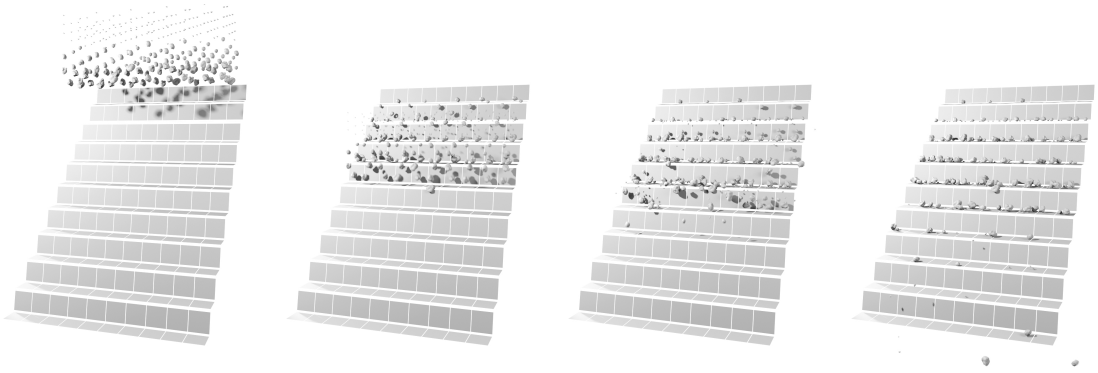


Figure 4.24: Particles tumble down a staircase.

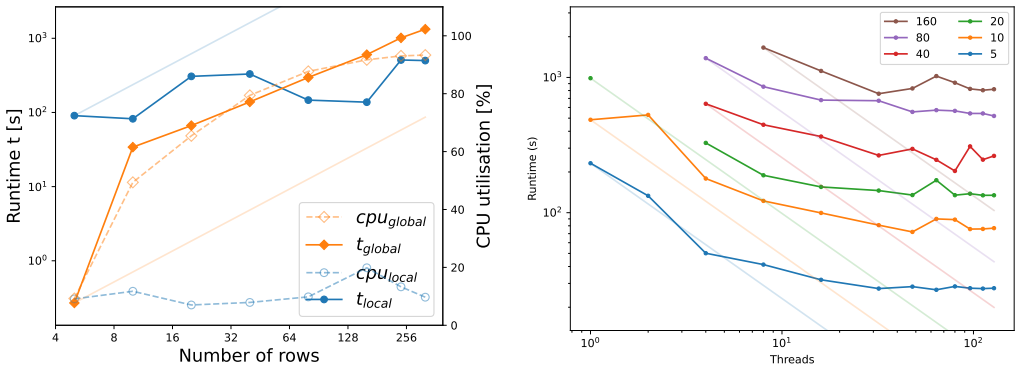


Figure 4.25: Left: Comparison of the global time stepping to local stepping on a whole node for various particle counts of the staircase scenario. Right: Scalability over cores.

to a wider distribution of time step sizes and a larger number of smaller clusters (Figure 4.26). This results in a significantly improved runtime (Figure 4.25). In our experiments the local time stepping variant fails to use a significant portion of the available cores but scales excellently. While CPU resources remain available the global time stepping scheme scales linearly, as we would expect. We predict that this trend would deteriorate once there was no longer available cores to allocate clusters to in any given time step.

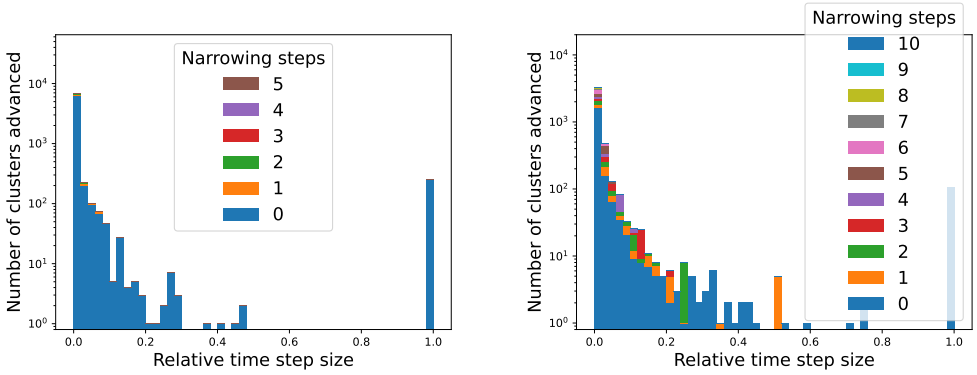


Figure 4.26: Time step size distribution for the hopper (left) and the staircase (right) setup using local time stepping.

Wrap-up Throughout this chapter we have introduced novel methods for 'anarchic' local time stepping that are guided by perhaps the most pervasive conceptual motif in this work: To accelerate difficult computations we attempt to cheaply approximate solutions but allow the possibility of failures by including a fall back mechanism to correct these cases. Our algorithm helps to alleviate high computational load in highly dynamic scenarios where small numbers of highly stiff interactions can cause the whole simulation to require tiny time step sizes. Each particle is equipped with multiple states over a time span and a simple method for synchronising particles in time when they need to interact. Within each synchronised cluster we effectively run an isolated simulation for a single step, which mean a variety of integration methods could also be applied alongside our algorithm. This also enables the wealth of work and software related to contact detection and contact force modelling to be applied alongside our time stepping method.

Even with our time stepping algorithm, particularly dense areas of a simulation with many contacts over many time steps can result in unacceptably slow simulations. This is more likely to be a challenge when there are a variety of particle complexities present. If a local area has a large number of interactions between a small number of complex particles this will become a computational bottleneck. Therefore, in Chapter 5 we discuss an incremental improvement to a state of the art triangle-to-triangle distance check method followed by a novel implicit multiresolution contact algorithm in Chapter 6. The first improves contact detection times by approximately a constant factor, with respect to particle complexity, while the multiresolution algorithm is shown to significantly reduce the time required to compute contacts involving very complex particles.

CHAPTER 5

Closest points calculations

At the heart of any DEM code is the process for detecting contacts between objects. Since we choose to represent a impenetrable inner mesh surrounded by an additional radius we therefore need to be able to detect when these epsilon regions overlap. To do this we calculated the minimum distance between elements and then compare this to the sum of the radii.

This chapter describes the process we use for determining minimum distances between objects. First, we discuss the primitive distance checks for triangle-triangle minimum distance comparisons and propose iterative algorithms to accelerate that process (Section 5.1). Second, we briefly outline an extension to our iterative distance algorithm to approximate a collision time between two moving triangles (Section 5.2). Third, we propose a hierarchical data structure that provides us with the algorithmic benefits of a hierarchical data structure while also allowing us to equip each level of the hierarchy with meaningful data about the underlying shape (Section 5.3). This extra data becomes relevant for our multi-resolution contact resolution algorithm in Chapter 6. Next, we discuss the method used for storing triangle mesh data to optimise the efficiency of access for different requirements (Section 5.4). Finally, we present some preliminary results (Section 5.5).

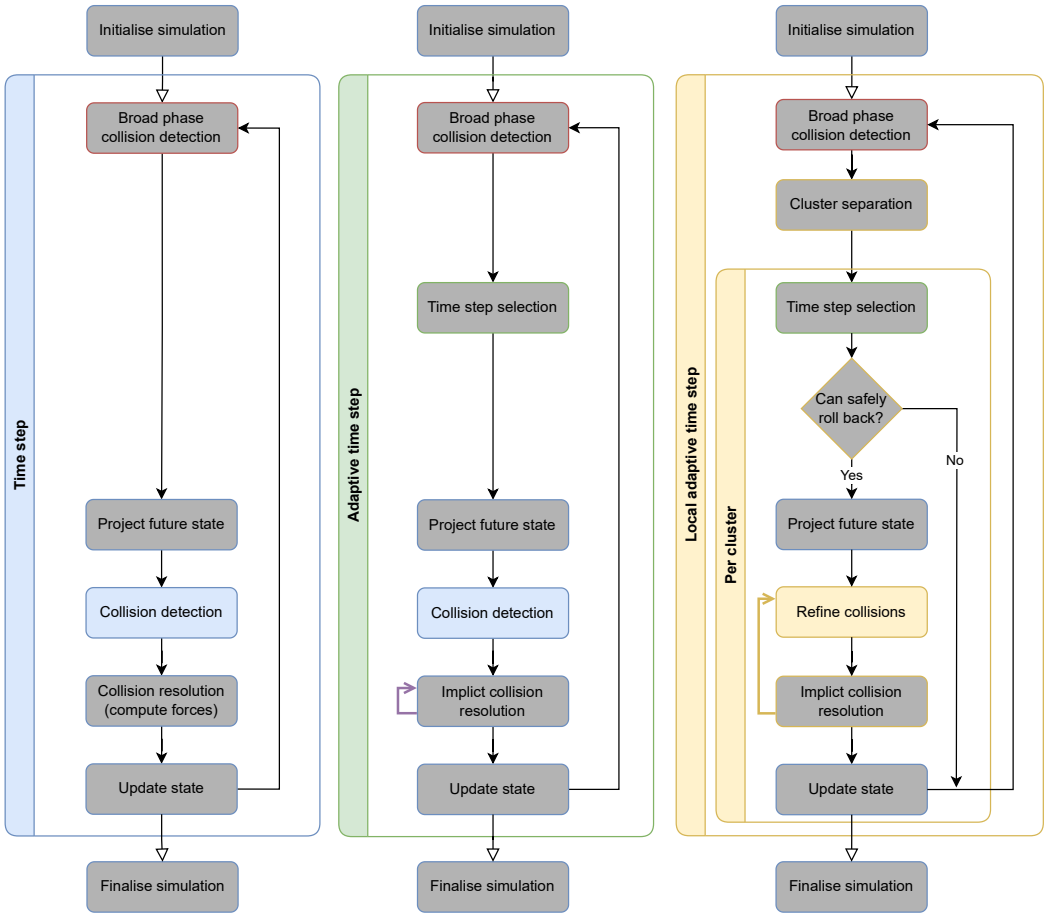


Figure 5.1: The phases of the algorithm that are discussed in this chapter are shown in colour.

While Chapter 4 focuses on a global level optimisation, this chapter focuses on the interaction between individual complex particle pairs. In cases where small numbers of complex particle-particle interactions dominate the computational load the benefits of our algorithm presented in Chapter 4 can be lost or even become detrimental to the total runtime. Therefore, the primary aim of the algorithms proposed in this chapter and Chapter 6 are to reduce the time to solution for individual particles on individual or small numbers of threads. The intended synergy is for the 'anarchic' local time stepping to be providing only relevant particle clusters but a sufficiently large number of particle pairs that can then be resolved efficiently using all available vector lanes and cores.

5.1 Triangle closest point calculation

To find the closest point between two triangles is a classic computational geometry problem [78]. We rely on four different algorithms to solve it:

5.1.1 Direct distance calculation (comparison-based)

A comparison-based identification of contact points consists of taking the minimum of a number of individual primitive distance calculations. We compute the distance from each vertex of the two triangles to the closest point on the other triangle face as well as the distance between each pair of edges between the two triangles [78]. This yields six point-to-triangle distance tests and nine edge-to-edge distance tests. In a second step, we select the minimum distance out of the 15 combinations. This brute force calculation yields an exact solution—agnostic of truncation errors—yet requires up to 30+14 comparisons (if statements) per triangle pair. Dealing with the cases where intersections between triangles is possible requires an additional six edge-to-plane distance tests, where intersections outside the area of the triangle are discounted.

5.1.2 Iterative distance calculation

As an alternative to a comparison-based approach, we replace the geometric checks with a function where we minimise the distance between the two planes spanned by the triangles but add the admissibility conditions over the Barycentric coordinates as Lagrangian parameters [4, 5]. Let $a, b \in [0, 1]$ describe any point on their respective triangle:

$$\begin{aligned}\underset{a_1,b_1,a_2,b_2}{\arg\min}\, J(a_1,b_1,a_2,b_2) := \quad & \underset{a_1,b_1,a_2,b_2}{\arg\min} \underbrace{\frac{1}{2}\|t_i(a_1,b_1) - t_j(a_2,b_2)\|^2}_{=:\hat{J}(a_1,b_1,a_2,b_2)} + \alpha_{\text{iterative}}\Big(\\ & \max(0, a_1 - 1) + \min(-a_1, 0) + \max(0, b_1 - 1) + \\ & \max(-b_1, 0) + \max(0, a_1 + b_1 - 1) + \\ & \max(0, a_2 - 1) + \min(-a_2, 0) + \max(0, b_2 - 1) + \\ & \max(-b_2, 0) + \max(0, a_2 + b_2 - 1)\Big). \end{aligned} \tag{5.1}$$

This is a weak formulation of the challenge, since any $\alpha_{\text{iterative}} < \infty$ allows the closest distance line between two triangles to be rooted slightly outside the very triangles.

The minimisation problem can be solved via Newton iterations. However, the arising Hessian becomes difficult to invert or non-invertible for close-to-parallel or actually parallel triangles. We therefore regularise it by adding a diagonal matrix. Unlike the original formulation of this method [4] we only approximate the regularised Hessian, using the terms on the diagonal, and update the minimisation and constraints alternatingly:

$$\begin{aligned}
(a_1, b_1, a_2, b_2)^{(n+0.5)} &= (a_1, b_1, a_2, b_2)^{(n)} - \text{diag}^{-1}\Big(\hat{H}(a_1, b_1, a_2, b_2)^{(n)} \\
&\quad + \alpha_{\text{regulariser}}\ id\Big) \nabla_{a_1,b_1,a_2,b_2} \hat{J}(a_1, b_1, a_2, b_2)^{(n)} \\
(a_1, b_1, a_2, b_2)^{(n+1)} &= (a_1, b_1, a_2, b_2)^{(n+0.5)} - \text{diag}^{-1}\Big(\tilde{H}(a_1, b_1, a_2, b_2)^{(n+0.5)} \\
&\quad + \alpha_{\text{regulariser}}\ id\Big) \nabla_{a_1,b_1,a_2,b_2} \tilde{J}(a_1, b_1, a_2, b_2)^{(n+0.5)}
\end{aligned} \tag{5.2}$$

$\alpha_{\text{regulariser}} > 0$ is small, while $^{(n)}$ is the iteration index. The $\hat{J}$ and its Hessian correspond to the quadratic term in (5.1). $\tilde{J}$ and its Hessian cover the remaining penalty terms, i.e. $J - \hat{J}$. Our solver iterates back and forth between the $\hat{J}$-minimisation and a fullfillment of the constraints. This modified Newton becomes a gradient descent, where the step size is adaptively chosen by analysing an approximation of the inverse to the Hessian.

We derive the Newton-Raphson update steps using a simple Python script using the SymPy library [156].

```
import sympy

# Define tx1, ty1, tz1, tx2, ty2, tz2 as the x, y & z coordinates for a
#   point define by barycentric coordinates on triangle 1 and 2 respectively

distanceX = tx1(a, b) - tx2(c, d)
distanceY = ty1(a, b) - ty2(c, d)
distanceZ = tz1(a, b) - tz2(c, d)

penalty = sympy.Max(0, -a) + sympy.Max(0, -b) +
            sympy.Max(0, -c) + sympy.Max(0, -d) +
            sympy.Max(0, a - 1) + sympy.Max(0, b - 1) +
```

```
            sympy.Max(0, c - 1) + sympy.Max(0, d - 1) +
            sympy.Max(0, a + b - 1) + sympy.Max(0, c + d - 1)

# Alpha can be used to tune the 'strictness' of the penalty terms
# In practice we apply the penalty updates separately to unsure they're enforced
J = 0.5 * (distanceX**2 + distanceY**2 + distanceZ**2 + alpha * penalty**2)

dJ_a = sympy.diff(J, a)
dJ_b = sympy.diff(J, b)
dJ_c = sympy.diff(J, c)
dJ_d = sympy.diff(J, d)

# sympy.simplify can be used on each of these to give a smaller output
ddJ_aa = sympy.diff(dJ_a, a)
ddJ_bb = sympy.diff(dJ_b, b)
ddJ_cc = sympy.diff(dJ_c, c)
ddJ_dd = sympy.diff(dJ_d, d)

update_a = a - 1.0 / (C_dontdividebyzero + ddJ_aa) * dJ_a
update_b = b - 1.0 / (C_dontdividebyzero + ddJ_bb) * dJ_b
update_c = c - 1.0 / (C_dontdividebyzero + ddJ_cc) * dJ_c
update_d = d - 1.0 / (C_dontdividebyzero + ddJ_dd) * dJ_d

# Use C11CodePrinter to output code for each of the updates
```

5.1.3 Hybrid distance calculation

If two subsequent iterates $|J(a_1, b_1, a_2, b_2)^{(n+1)} - J(a_1, b_1, a_2, b_2)^{(n)}| \leq C\epsilon$, we have found a solution, (5.1) and can terminate the minimisation. In this case, we assume that a_1, b_1, a_2, b_2 identify the minimum distance. Without further analysis, it is impossible to make a statement on the upper bound on n. Our hybrid algorithm therefore eliminates the termination criterion and imposes $n \leq N_{\texttt{iterative}}$. Consequently, it labels a distance calculation as invalid if $|J(a_1, b_1, a_2, b_2)^{(N_{\texttt{iterative}})} - J(a_1, b_1, a_2, b_2)^{(N_{\texttt{iterative}}-1)}| > C\epsilon$. The iterative code's result realises a three-valued logic: "found a contact point", "there is no contact", or "has not terminated" (Algorithm 3).

Our hybrid algorithm invokes the modified iterative algorithm. If the result equals "not terminated" ($\odot$), the hybrid algorithm falls back to the comparison-based distance calculation. It is thus not really a third algorithm to find a contact

point, but a combination of the iterative scheme with the comparison-based approach serving as a posteriori limiter.

Algorithm 3 A hybrid, batched reformulation of the iterative distance calculation. The iterative sweep over a whole batch of triangles is stripped of a dynamic stopping criterion (top) and therefore yields three types of results per triangle pair: x holds a coordinate if the Barycentric coordinates yield a contact point, or $\perp$ if they yield no contact point, or $\odot$ if the triangle combination has to be postprocessed with the comparison-based algorithm (bottom).

for $t_i \in \mathbb{T}(p_i),\ t_j \in \mathbb{T}(p_j),\ t_i \neq t_j$ **do**
  $(a_1, b_1, a_2, b_2, n)(t_i, t_j) \leftarrow 0$ ▷ Variables per triangle pair allow us to vectorise
  $J_{\text{old}}(t_i, t_j) \leftarrow \infty$ ▷ over triangles in $\mathbb{T}$
  $J(t_i, t_j) = J(a_1(t_i, t_j), b_1(t_i, t_j), a_2(t_i, t_j), b_2(t_i, t_j))$ ▷ Functional from (5.1)
  while $n \leq N_{\text{iterative}}$ **do** ▷ Fixed iteration count instead of $|J - J_{\text{old}}| > C\epsilon$
    $J_{\text{old}}(t_i, t_j) \leftarrow J(t_i, t_j)$
    update $a_1(t), b_1(t), a_2(t), b_2(t)$ ▷ Modified gradient descent from (5.2)
    $J(t_i, t_j) = J(a_1(t_i, t_j), b_1(t_i, t_j), a_2(t_i, t_j), b_2(t_i, t_j))$
    $n \leftarrow n + 1$
  end while
  $$x(t_i, t_j) \leftarrow \begin{cases} \frac{1}{2}(t_i(a_1, b_1) - t_j(a_2, b_2))(t_i, t_j) & \text{if } |J(t_i, t_j) - J_{\text{old}}(t_i, t_j)| \leq C\epsilon \wedge \hat{J}(t_i, t_j) \leq 2\epsilon^2 \\ \perp & \text{if } |J(t_i, t_j) - J_{\text{old}}(t_i, t_j)| \leq C\epsilon \wedge \hat{J}(t_i, t_j) > 2\epsilon^2 \\ \odot & \text{otherwise} \end{cases}$$
end for

for $t_i \in \mathbb{T}(p_i),\ t_j \in \mathbb{T}(p_j),\ t_i \neq t_j$ **do**
  if $x(t_i, t_j) = \odot$ **then**
    $\hat{n}(t_i, t_j) \leftarrow$ shortest distance vector between t_i and t_j ▷ Use comparison-based algorithm
    $\hat{x}(t_i, t_j) \leftarrow$ is central point on line identified by $\hat{n}(t_i, t_j)$
    $$x(t_i, t_j) \leftarrow \begin{cases} \hat{x}(t_i, t_j) & \text{if } |\hat{n}(t_i, t_j)| \leq 2\epsilon \\ \perp & \text{otherwise} \end{cases}$$ ▷ Eliminate $\odot$ entries in result
  end if
end for

5.1.4 Intrinsics method

While the aim is always to have all the distances between triangle pairs computed by the iterative method we find the fallback to the comparison based method is still required a non-trivial portion of the time. Therefore, we implement a manually vectorised version of the comparison based method [63] using compiler intrinsics.

The intuition behind this method is to compute the series of individual edge-edge, vertex-face and edge-face comparisons in batches and order the operations such that we avoid branching where possible. For example, determining which side of a line a point is can be computed without branch and can be used to sort the

triangle pairs into different buckets for the next required operation. If all the points on a neighbouring triangle are on the other side of line AB compared to vertex C then we know there is no need to check lines BC and AC against any of the lines on the other triangle, for example. The next operation that is required per particle pair is determined by the result of the previous operation. This could be visualised as a decision tree. To begin with all triangle pairs are in the root node of the tree. A manually vectorised version of the comparison represented by this node in the tree is performed and the result used to sort the triangle pairs into the child nodes. We then proceed to the child nodes and repeat the process. Because we process the decision nodes in batches the vector efficiency remains high.

SSE instructions are used rather than AVX. No comparison has been done between these instruction sets for this method to the best of our knowledge. While AVX would increase the number of elements processed in parallel, it would also require a larger amount of work to keep the lanes busy and tends to result in a decrease in processor frequency while in use. Failing to keep the vector lanes fully occupied while at a lower frequency may result in an AVX version having a slower time to solution.

Speedups compared to non-vectorised versions are reported between 3-7x depending on the configurations of triangle pairs used [63].

For all further discussion and measurements when the comparison method is referred to we instead use this implementation.

5.2 Continuous triangle closest point

Estimating the time of contact between two triangles is required for ensuring triangles don't pass through each other by selecting a time step size that is too large. Due to our roll back mechanic we only require this time of contact to be an estimate.

In principle this problem could be formulated with time-distance minimisation in a similar way to our iterative distance check algorithm. The function to minimise is equipped with the vertex positions, start and end transforms and a time variable. However, in practice this method fails to converge for a significant proportion of

triangle-pair configurations. We believe further work on this method could lead to similar benefits to those seen when comparing static in time comparison based to hybrid triangle distance checks.

A different method that could take advantage of our iterative method would be to create new 3D meshes for each moving triangle by sweeping a volume with the positions of the triangles at the beginning and end of the time step (Appendix A.1). This reduces a space-time problem to a purely spatial one where we can use our existing algorithms. We consider the movement of each vertex to be linear within this time span. Therefore, each starting triangle results in eight new triangles. Once nearest distances have been established between these hulls we then use the barycentric coordinates of the hit points to estimate a time of contact (ie. hits closer to the triangle position at the beginning of the step are likely to mean a contact happens early in the time span and vice versa). However, despite being computationally cheap and algorithmically simple, this method suffers from two major draw backs. First, any particle that is significantly smaller than individual triangles of another particle can easily miss contacts and pass right through the surface of the larger object if the larger object is moving. This is because the space-time hull of the smaller object can be entirely enclosed within the space time hull of a single triangle in the larger object. Second, the combination of approximating the motion of vertices as linear and approximating the time of contact from barycentric coordinates can introduce significant error. This can result in very inaccurate estimates, especially for rotating objects, and therefore causes a greater number of roll backs.

Our final method was simply to consider moving faces, edges and vertices separately. This requires to make two types of check. Vertex-face and edge-edge. For each vertex-face pair, we compute the position of the vertex relative to the triangle at the begin and end of the maximum admissible time step: We use a relative coordinate system which moves with the triangle. After that, the algorithm takes the space-time line segment connecting the two relative positions. We can now analytically compute the minimum distance, as the minimum distance is either observed at the start or the end point, or, if it exists, it arises from the time stamp when the line intersects the (relative) plane in which triangle lives. For each of these two or

three, respectively, situations we can use Barycentric coordinates to construct the actual contact point, which is an approximation as we neglect rotation.

For each edge-edge pair, we realise the minimum distance computation iteratively by repeatedly bisecting the time span in which we search for collisions. If the minimal distance does not reduce anymore, we stop the bisection. Other more sophisticated methods exist [56] with additional efficiency and correctness guarantees, but this simple search serves well as a cheap heuristic for the time of contact. Again, the search is an approximation.

5.3 Surrogate hierarchies

Algorithm 4 Contact identification between two particles p_i and p_j.

```
1: function findContacts(𝕋(p_i), 𝕋(p_j))
2:     ℂ = ∅
3:     for t_i ∈ 𝕋(p_i), t_j ∈ 𝕋(p_j), t_i ≠ t_j do          ▷ Run over all triangles pairs
4:         c ← contact(t_i, t_j)    ▷ Find closest point in-between t_i and t_j and compare normal
5:         if c ≠ ⊥ then                                    ▷ |n| against ε; return ⊥ if |n| > ε
6:             ℂ ← ℂ ∪ {c}
7:         end if
8:     end for
9:     return ℂ
10: end function
```

The cost to compare two particles p_i and p_j is determined by their triangle counts $|\mathbb{T}(p_j)|$ and $|\mathbb{T}(p_j)|$. To reduce this cardinality, we construct geometric cascades of triangle models per particle—the *surrogate models* (Fig. 1.1)—and plug representations from this cascade into Algorithm 4.

Definition 8 *A surrogate model* $\mathbb{T}_k(p),\ k \geq 1$ *is a triangle-based geometric approximation of a particle described by* $\mathbb{T}(p)$*. A sequence of surrogate models* $\{\mathbb{T}_1(p), \mathbb{T}_2(p), \mathbb{T}_3(p), \ldots\}$ *for p with its volumetric extensions* $\mathbb{T}_k^\epsilon(p)$ *hosts efficient (Definition 9),* conservative *(Definition 10), and* weakly connected *(Definition 11) abstractions of* $\mathbb{T}(p)$*.*

Surrogate models are different representations of a particle. The term "cascade" highlights that each particle is assigned a whole sequence of representations. These representations can step in for our real geometry and are a special class of bounding volume techniques [78]. To simplify our notation, $\mathbb{T}_0(p) := \mathbb{T}(p)$, i.e. the $k = 0$

surrogate model is the geometric object itself. The bigger k, the coarser, i.e. more abstract the surrogate. We emphasise that the ϵ is a generic symbol, i.e. each surrogate model can host its own, bespoke ϵ-environment.

Definition 9 *A surrogate model $\mathbb{T}^\epsilon_{k_i}(p_i)$ is* efficient *if, for any other model $\mathbb{T}^\epsilon_{k_j}(p_j)$, finding all contact points between $\mathbb{T}^\epsilon_{k_i}(p_i)$ and $\mathbb{T}^\epsilon_{k_j}(p_j)$ is cheaper than finding all contact points between $\mathbb{T}^\epsilon_{\hat{k}_i}(p_i)$ and $\mathbb{T}^\epsilon_{k_j}(p_j)$ for all $0 \leq \hat{k}_i < k_i$.*

We assume an almost homogeneous cost per triangle-to-triangle comparison—an assumption that is shaky for the hybrid comparison and subject to vectorisation efficiency and memory management effects. Hence, the triangle count of surrogate models decreases with increasing k, i.e. $|\mathbb{T}_k| \ll |\mathbb{T}_{k+1}|$.

Definition 10 *A surrogate model $\mathbb{T}_{k_i}(p_i)$ inducing $\mathbb{T}^\epsilon_{k_i}(p_i)$ is* conservative *if*

$$\forall p_i, p_j, k_i, k_j : \ \mathbb{T}^\epsilon_{k_i}(p_i) \cap \mathbb{T}^\epsilon_{k_j}(p_j) = \emptyset \ \Rightarrow \ \mathbb{T}^\epsilon(p_i) \cap \mathbb{T}^\epsilon(p_j) = \emptyset. \tag{5.3}$$

Conservative means that any two surrogates of two particles are in contact (overlap) if the two particles are in contact. Yet, this does not have to hold the other way round: If their surrogates are in contact, there might still be gaps between the particles, i.e. there might be no contact point.

Corollary 1 *Let a surrogate model hierarchy $\mathbb{T}_k(p),\ k \geq 1$ be* monotonous *if*

$$\forall 1 \leq \hat{k} < k : \quad \forall t^\epsilon_{\hat{k}} \in \mathbb{T}^\epsilon_{\hat{k}}(p) : \quad t^{\epsilon_{\hat{k}}}_{\hat{k}} \subseteq \bigcup_{t \in \mathbb{T}^\epsilon_k(p)} t. \tag{5.4}$$

Our surrogate hierarchies do not have to be monotonous.

A monotonous cascade of triangles plus ϵ-environments would grow in space as we move up the hierarchy of models. Therefore, we do not impose it, even though monotonicity would imply conservativeness "for free". Empirical evidence suggests that abandoning monotonicity allows us to work with significantly tighter ϵ-choices per level. Yet, it also implies that we generally cannot show algorithm correctness through plain induction.

Classic level-of-detail algorithms require a coarsened representation of a triangulated model to preserve certain properties such as connected triangle surfaces or the preservation of certain features such as sharp edges. Our surrogate models however are to be used as temporary replacements within our calculations. We thus can ask for weak representations of the geometries:

Definition 11 *The triangles of a particle have to span a connected surface. For surrogate models, there is no such constraint. Therefore, the surrogate models can be* weakly connected*: Their triangles can be disjoint with gaps in-between or they can intersect each other. A surrogate's ϵ-environment is connected however, and it covers (overlaps) all connectivity of the original model.*

The connectivity addendum in Definition 11 motivates the term weakly connected. It clarifies that we—despite the disjoint configuration of a surrogate triangle set (Figure 1.1)— cannot miss out on some geometry extrema such as sharp edges due to tests with surrogates: If there is no contact between two surrogates, there is also no contact between their real discretisations.

Definition 10 implies that we can use surrogate models as guards and run through them for coarse to fine: If there are no contact points between two surrogate models, there can be no contact points for the more detailed models. We can stop searching for contact points immediately. It does however not hold the other way round. Definition 9 implies immediately that the number of triangles that we examine in such an iterative approach is monotonously growing. Definition 11 gives us the freedom to construct such triangle hierarchies, as it strips us from many geometric constraints.

Definition 12 *Let $\mathcal{T}$ be a directed acycling graph where each node represents a set of triangles. The level ℓ of a node is its distance (edge count) from the root node in $\mathcal{T}$. The resulting graph is a* surrogate tree *if and only if*

1. *the root node hosts the coarsest surrogate model $\mathbb{T}_{k_{max}}$*
2. *the union over all leaf sets yields the particle triangulation $\mathbb{T}_0$*
3. *any triangle is a surrogate for the union over its children's triangle sets.*

With Definition 12, the union over all sets with the same level yields the surrogate model $\mathbb{T}_{k_{\max}-\ell}$. We use $N_{\text{surrogate}}$ to denote the number of children of a surrogate triangle. $N_{\text{surrogate}}$ does not have to be uniform over the tree, i.e. is a generic symbol. If we have a surrogate model, take the nodes within the tree which hold its triangles, and replace the triangles with those triangles stored within children nodes, we obtain the next finer surrogate. A surrogate tree is a generalisation of the concept of a cascade of surrogates: The tree formalism allows us also to construct different, hybrid-level surrogate models if we only replace some triangles of a model with their children.

Implementation remark 1. There are multiple ways to construct surrogate trees. We construct our trees through a recursive algorithm. It starts from the triangle set $\mathbb{T} = \mathbb{T}_0$ of the particle and splits this set into $N_{\text{surrogate}}$ subsets of roughly the same size hosting close-by triangles. Per subset, we construct one surrogate and thus obtain a tree of depth one where the root node hosts $|N_{\text{surrogate}}|$ triangles. As long as a node within the tree hosts more triangles than a prescribed threshold, we apply the splitting recursively and thus disentangle the triangle sets further and further: Existing tree levels are pushed down or sieved through the tree hierarchy (Section 5.3.2). A follow-up bottom-up traversal of the tree constructs well-suited surrogate triangles with appropriate ϵ choices. As we only require weakly connected surrogates, the steps within this bottom-up traversal are independent of operations on sibling nodes within the tree and can be mapped onto local minimisation problems (Appendix 5.3.1). □

5.3.1 Surrogate triangles

The construction of good surrogate models is a challenge of its own, as there are infinitely many surrogate models for a given particle p. We rely a functional minimisation with a fixed coarsening factor to construct the surrogate hierarchy bottom-up. Let one surrogate triangle for a set of triangles $\mathbb{T}_{k-1}$ be spanned by its three vertices $x_1, x_2, x_3 \in \mathbb{R}^3$ which follow

$$\begin{aligned}\underset{x_1,x_2,x_3}{\arg\min} \quad & \frac{1}{\beta_{\text{size}}} \sum_{x\in\{x_1,x_2,x_3\},t\in\mathbb{T}_{k-1}} \|x-t\|^{\beta_{\text{size}}} + \frac{\alpha_{\text{area}}}{2}|(x_2-x_1)\times(x_3-x_1)|^{-2} \\ + \quad & \frac{\alpha_{\text{inside}}}{\beta_{\text{normal}}} \sum_{x\in\{x_1,x_2,x_3\},x_t\in t \ \forall t\in\mathbb{T}_{k-1}} (\max(0,(x-x_t)\cdot N(x_1,x_2,x_3)))^{\beta_{\text{normal}}} \end{aligned} \tag{5.5}$$

$\beta_{\text{size}} \geq 2$ is a fixed integer parameter which we usually pick very high, and the term $\|x-t\|$ denotes the distance between a point x and a triangle t. The sum thus minimises the maximum distance between the vertices of the surrogate triangle and the triangles from $\mathbb{T}_{k-1}$. We try to make the surrogate triangle as small as possible. The second term acts as a regulariser that avoids that the surrogate triangle degenerates and becomes a single point or a line. Without it, the first term would yield a single point, i.e. $x_1 = x_2 = x_3$.

The third term exploits the fact that each triangle t of p has a unique outer normal $N(t)$. Even though our surrogate models can be weakly connected, it is thus possible to assign each surrogate triangle an outer normal, too. The penalty term over the scalar product drops out due to the max function if the surrogate's normal points into the same direction as the triangles' normals. Effectively, this term ensures that the surrogate triangle nestles closely around a particle and that spikes do not induce a blown-up surrogate (Fig. 5.2). Once (5.5) yields a surrogate triangle, ϵ is chosen such that the triangle is conservative for $\mathbb{T}_{k-1}$.

5.3.2 Surrogate trees

Let $N_{\text{surrogate}} > 1$ be the surrogate coarsening factor. We construct a surrogate tree top-down (Algorithm 5):

- The first triangle that we insert into $\mathcal{T}$ is the coarsest surrogate model, i.e. a degenerated object description consisting of one triangle. In line with Definition 12, this is the root node of our surrogate tree.
- Recursively dividing creates a tree over sets where all non-leaves have cardinality one. The leaf sets have a cardinality of roughly $N_{\text{surrogate}}$. The number of children per tree node is bounded and typically around $N_{\text{surrogate}}$.

Algorithm 5 Top-down algorithm to construct a cascade of surrogate models for a given triangulation $\mathbb{T}$. The algorithm yields a tree defined through $\sqsubseteq_{\text{child}}$ and hence allows us to derive a vast set of different, locally adaptive surrogate models.

function constructSurrogate($\mathbb{T}$)
  Construct surrogate triangle t for $\mathbb{T}$ ▷ Solve (5.5)
  Assign t smallest ϵ such that t^ϵ is conservative surrogate
  Create trivial graph $\mathcal{T}$ with single node $\{t^\epsilon\}$ and no edges
  constructSurrogateRecursively($t^\epsilon, \mathbb{T}$)
end function
function constructSurrogateRecursively($t^\epsilon_{\text{local}}$, $\mathbb{T}_{\text{local}}$)
  if $\mathbb{T}_{\text{local}} \leq N_{\text{surrogate}}$ **then**
    Add node $\mathbb{T}_{\text{local}}$ to $\mathcal{T}$
    Add edge $\mathbb{T}_{\text{local}} \sqsubseteq_{\text{child}} \{t^\epsilon_{\text{local}}\}$ to $\mathcal{T}$
  else
    Split $\mathbb{T}_{\text{local}}$ into $N_{\text{surrogate}}$ sets $\mathbb{T}_{\text{local},0}, \mathbb{T}_{\text{local},1}, \mathbb{T}_{\text{local},2}, \ldots$ of roughly same size
    for i **do**
      Construct surrogate triangle t_{new} for $\mathbb{T}_{\text{local},i}$ ▷ Solve (5.5)
      Assign t_{new} smallest ϵ such that t^ϵ_{new} is conservative surrogate over $\mathbb{T}_{\text{local},i}$
      Add node $\{t^\epsilon_{\text{new}}\}$ to $\mathcal{T}$
      Add edge $\{t^\epsilon_{\text{new}}\} \sqsubseteq_{\text{child}} \{t^\epsilon_{\text{local}}\}$ to $\mathcal{T}$
      constructSurrogateRecursively($t^\epsilon_{\text{new}}, \mathbb{T}_{\text{local},i}$)
    end for
  end if
end function

- We construct the surrogate triangles by copying one triangle out of the underlying triangle set. Then, we iteratively minimise the functional (5.5).
- To obtain surrogates with reasonably small ϵ, we cluster the triangle sets through a tailored k-means algorithm [157]. The subsets $\mathbb{T}_{\text{local},i}$ thus are reasonable compact.

There are many alternative paradigms to construct surrogate trees: Bounding sphere hierarchies would be an alternative. Our approach is relatively slow, yet is exclusively used as pre-processing.

Implementation remark 2. A surrogate hierarchy is constructed for a specific arrangement of triangles. Therefore, if anything causes the arrangement of those triangles to change (for example, editing a particle's mesh to simulate abrasion) the correctness guarantees of the surrogate hierarchy may no longer hold. If a child triangle moves out of the region covered by its parent then the surrogate hierarchy must be recomputed or the parents of the adjusted triangles must be recursively adjusted to ensure all children are always covered by the parents' volume. □

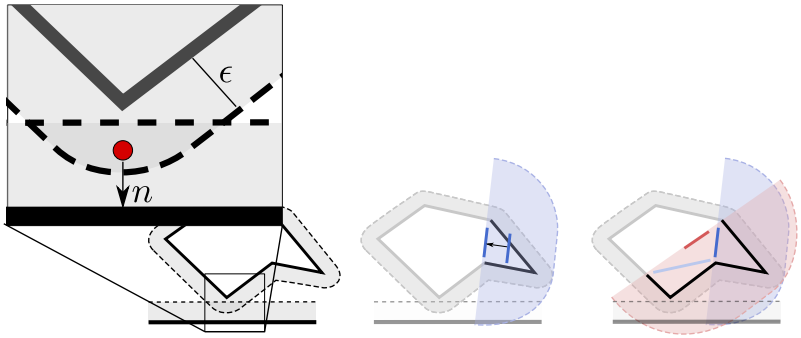


Figure 5.2: A pair of objects (black, solid) with their ϵ-environment (black, dotted) collide. In the present sketch, one object is a "spherical" particle spanned by six edges, while the other object is a plane at the bottom. Left: Two-dimensional sketch of the contact point concept. The zoom-in shows the contact point which is located in the middle of the overlap region. The contact normal is directed from the middle point towards the closest point on one of the objects involved. Middle: When an object hosts very extruded features, we slightly shrink the surrogate such that the surrogate without ϵ becomes a closer fit around the "un-bumped" real geometry. We trade such a surrogate for a bigger ϵ. Empirical evidence suggests that this yields slightly advantageous forces within a multiscale iterative solve. Right: The conservative property of the surrogate triangles states that all fine level geometry (including its epsilon boundary) must be encompassed by the surrogate's epsilon. This doesn't suggest that all surrogate children are included.

Accelerated multiscale algorithm. We refer to the sequence of surrogates that form a surrogate tree as $\mathbb{T}^\epsilon_h(p), \mathbb{T}^\epsilon_{2h}(p), \mathbb{T}^\epsilon_{4h}(p), \mathbb{T}^\epsilon_{8h}(p), \ldots$ recursively. We also refer to elements within this sequence as levels. Each surrogate is conservative with respective to the next finer surrogate. That is, if we decide that two surrogates of level k do not collide, then their corresponding surrogates of level $k/2$ do not collide either. Each surrogate is also effective with respect to the next finer surrogate. That is, it is cheaper to handle a surrogate of level k compared to the surrogate of level $k/2$, as the former features fewer triangles.

Definition 13 *Let the parent of a surrogate of a fine mesh triangle or surrogate triangle $\tau \in \mathbb{T}^\epsilon_{kh}(p)$ be given as*

$$\tau_{parent} \in \mathbb{T}^\epsilon_{\frac{k}{2}h}(p) \quad \textit{and} \tag{5.6}$$

$$\tau^\epsilon \subseteq t^\epsilon_{parent}. \tag{5.7}$$

With the notion of a surrogate at hand, we can introduce an iterative variant ex-

panding upon our vanilla code version. We start with the coarsest surrogate models of all particles and for each particle with a corresponding $c(p_1, p_2) = \top$ entry we add the pair of top level surrogate triangles to the search set $\mathbb{S}$. While $\mathbb{S} \neq \emptyset$ for each pair $(\tau_1, \tau_2) \in \mathbb{S}$ we approximate the time of contact and the minimum time of contact using the expanded ϵ region and (τ_1, τ_2) is removed from $\mathbb{S}$. If the smallest contact distance within the allowed time span is less than the sum of the ϵ regions then every combination of the children of t_1 and t_2 is added to $\mathbb{S}$ unless either are fine mesh triangles. If t_1 and t_2 are fine mesh triangles and the estimated time of collision is less than $t^{(\text{collision})}(\mathbb{P}_i)$ is updated.

Efficiency. $t^{(\text{collision})}(\mathbb{P}_i)$ is monotonously decreasing in our iterative scheme. Therefore, the algorithm iteratively narrows down the particles and the branches of the surrogate tree that can collide. Initially, we use the information stored within c. As we run the tests over the space-time geometries and reduce the time span of interest, we disable more and more entries within c and can terminate the exploration of branch pairs of the surrogate trees early. Even branch pairs that would result in a collision later in the original time span. Per iteration, fewer particle pairs and a narrower time span are to be studied.

Our intention is that, where geometrically possible, only the first iterations deal with large particle counts. In these cases, our surrogate formalism ensures that large particle counts do not translate into excessive triangle counts, as we use rather coarse geometric representations. The overall triangle count per iteration per particle increases monotonously, but our hope is that the total triangle count per cluster per iteration does not grow (too fast).

With $|\mathbb{T}_{kh}(p_1)|$ triangles for particle p_1 and $|\mathbb{T}_{kh}(p_2)|$ for p_2 at one step of the algorithm, we have to perform a maximum of $|\mathbb{T}_{kh}(p_1)| \cdot |\mathbb{T}_{kh}(p_2)|$ triangle-triangle continuous comparisons. In an ideal case where there exists a single contact between two particles and only a single branch of each particle has to be explored then for one step of the algorithm we only need to perform K^2 triangle-triangle continuous comparisons, where K is the maximum number of children each surrogate triangle possess. This gives us an upper and lower bound on the number of continuous

triangle-triangle collision checks we need to do. When no contacts exist or contacts only exist after the current $t^{(\text{collision})}(\mathbb{P}_i)$ (ie. another particle pair in the cluster has already been evaluated and an early contact was found) then we are able to beat this lower bound by early stopping the exploration.

Robustness. There are two simplications within our algorithm which require special attention. First, we assume that triangles move linearly through space. This is not valid once $r(p) \neq 0$. For our use case we simply ignore the possibility of this causing a section of an object to rotate fully through another by instead bounding the maximum time step size for the geometry we use. In an application with finer interacting geometry (for example, fine teeth on interlocking gears) it may be necessary to add extra checks here. One such option would be to increase the ϵ region of each triangle and surrogate to ensure the real volume swept by a rotation triangle is covered. While this would introduce more false-positive collisions it would ensure there are no false-negatives.

The second assumption results directly from the fact that we work with floating point numbers and work towards a geometric model where the particles are weakly incompressible. Hence, we might run into situations where the algorithm yields no contact point at all, as we just slightly underestimate a valid $\Delta t(\mathbb{P}_i)$. Further to that, we will need a certain non-zero overlap of the particle ϵs in the subsequent step. Therefore, we make our algorithm artificially increase the time step size after each iteration: In the realisation, we compute the minimum and the maximum collision time stamp. Rather than the minimum, the average value is used as estimate for the collision time from hereon.

5.4 Triangle storage

The method used to store the triangles of a particle can have a significant impact on the efficiency of the collision detection and continuous collision detection algorithms. Since all mesh operations in our algorithm are embedded in a tree traversal we choose to store triangles in 'buckets' that each has a maximum number of triangles. Within each bucket there are two different methods we use to represent triangles:

First, store a list of vertex positions, a list of index pairs representing edges and a list of index triples representing faces. We name this a 'Connected' bucket. Second, store a list of 3-tuples, which each represent a complete triangle. We name this a 'Triangle Soup' bucket.

Connected buckets can be advantageous when we wish to operate on edges or vertices. For a densely connected mesh with many shared edges this method removed redundant copies. However, accessing a triangle requires several operations. For a fixed number of triangles the number of vertices and edges isn't constant requiring a dynamically sized data structure. This is ill suited to vectorisation. We use this type of bucket for leaf nodes in our surrogate hierarchies to store densely connected patches of the fine mesh.

Triangle soup buckets as well suited when there are no shared edges between triangles. Since the number of triangles in the bucket is constant and the storage required per triangle is fixed the data structure is fixed size. Furthermore, the triangle coordinates can be arranged in a way to aid vectorisation. We use this type of bucket for non-leaf nodes in our surrogate hierarchies. Early contact detection stages are likely to yield many potential contacts (and so result in lots of work that benefits from vectorisation) and never share edges.

5.5 Results

We observe a constant factor speed up of 2 when doing N-to-N triangle comparisons (Figure 5.3) using our hybrid method. This drops to around 1.5 when used in conjunction with the surrogate tree. The surrogate tree demonstrate an algorithmic improvement over the naive N-to-N in line with other hierarchical data structures.

As we would expect from a hierarchical method the runtime scales very well with number of elements. When simulating a sphere with a varying number of triangles colliding with a sphere of 80 triangles we see the following runtimes

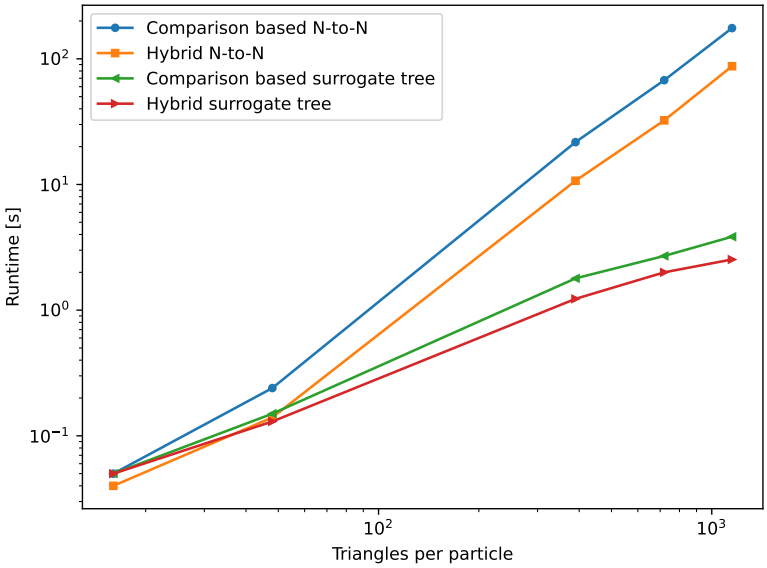


Figure 5.3: A comparison of triangle-to-triangle and particle-to-particle distance check methods for 500 steps of a simple simulation.

Triangles	Runtime [s]
80	0.087
320	0.23
1,280	0.26
5,120	0.28

The number of triangles interacting with the triangles in the other sphere doesn't increase significantly as the area of contact is so small. Therefore, when using a hierarchical method, the runtime is only loosely correlated to the number of triangles in the mesh.

Wrap-up In this chapter see a different form of the iterative-but-fallback-to-robust concept. We propose a refined version of an iterative algorithm to accelerate computations that would usually be too complex to solve purely with a simple iterative algorithm by phrasing the solution as both a simple iterative scheme and a robust fall back method. As long as the iterative scheme converges for a sufficient proportion of inputs and is significantly faster than the exact robust solution then we see an overall speed improvement. By further simplifying an existing iterative triangle-triangle distance minimisation algorithm, with little impact on the convergence, while also

implementing an improved version of the robust fallback method, we achieve a very high throughput.

The idea of a hierarchical multi-resolution mesh is also proposed In this chapter we show the construction of a hierarchical data structure similar in many ways to conventional spatial partitioning data structures (eg. Octrees). However, we emphasis the multi-resolution nature of our method. Each level of the hierarchy has a geometric representation that matches the underlying geometry. Therefore, while our mesh representation can be used to accelerate spatial queries in the traditional sense the key idea is that lower resolutions from the mesh can be used for geometric queries with a tunable trade off between accuracy and computational cost. The following chapter will describe how we exploit this idea as part of a multiresolution implicit contact resolution algorithm.

CHAPTER 6

Multi-resolution Implicit Contact Resolution

6.1 Multiresolution contact detection

With our surrogate tree definition from Chapter 5, we are in the position to propose a multiscale algorithm for an explicit Euler, which utilises the tree as early stopping criterion in the search for contacts. While the refined triangle-triangle distance check algorithm from the previous chapter is important for reducing particle-particle distance checks, by approximately a constant factor, the problem still remains that very complex particles quickly become prohibitively expensive to check for collisions when using naive algorithms. After outlining a simple explicit scheme using the tree data structure to accelerate spatial lookups, we derive two implicit time stepping algorithms that exploit the multiscale nature of the geometry to terminate searches early and to supplement the underlying iterative algorithm with educated guesses.

We show that a novel inverted order of operations for classic spatial data structures used in an iterative implicit scheme results in a method with a faster time to solution. When combined with the local time stepping algorithm from Chapter 4 to produce sufficiently large numbers of particle-particle contact candidates to occupy many cores and the triangle-triangle distance check algorithm from Chapter 5 for

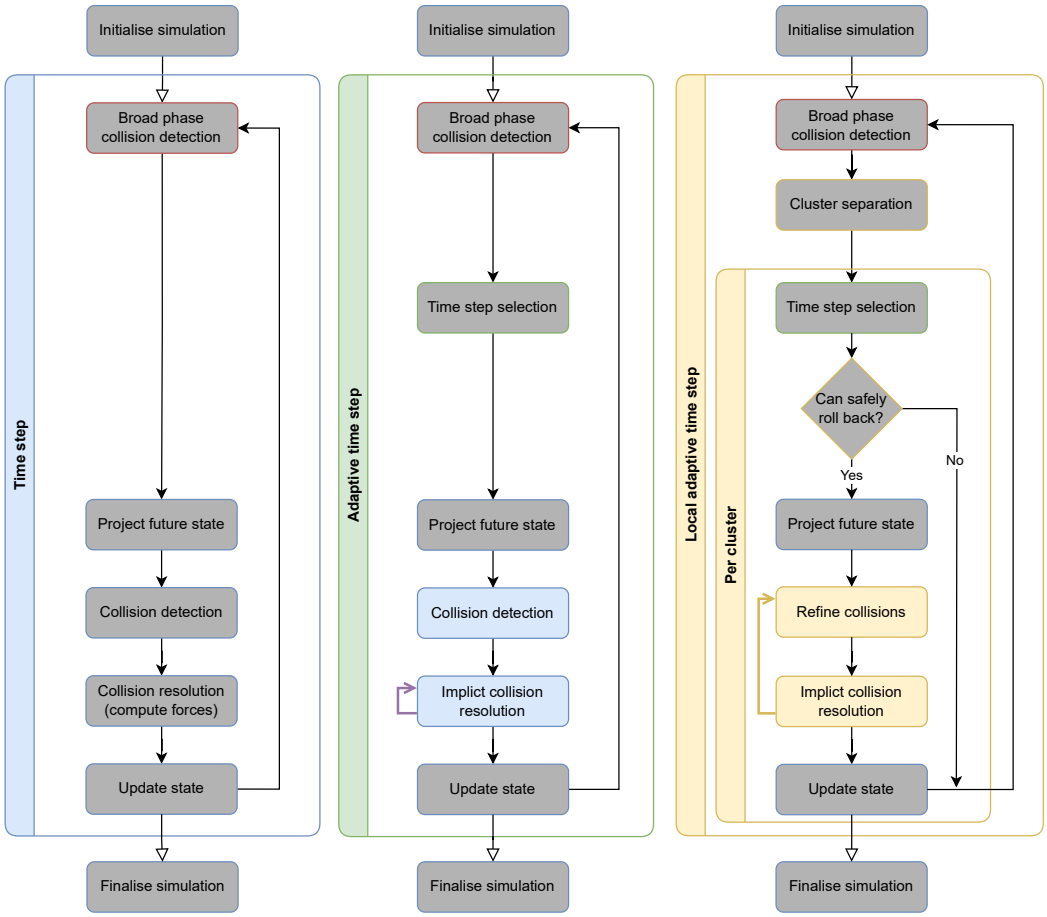


Figure 6.1: The phases of the algorithm that are discussed in this chapter are shown in colour.

efficiently utilising all available vector lanes, our algorithm is able to utilise a very high proportion of the available CPU hardware to efficiently compute solutions to large and highly dynamic scenarios with very stiff interactions using an iterative implicit method.

6.1.1 Explicit Euler

Our explicit Euler exploits the multiscale hierarchy by looping over the the resolutions held in $\mathcal{T}$ top down. The tree is unfolded depth-first, and we implement an early stopping criterion: If a surrogate triangle and a triangle set from another particle do not collide, the children of the surrogate triangle in our triangle tree cannot collide either. The depth-first traversal along this branch of the tree thus can terminate early. The surrogate model unfolds adaptively.

Algorithm 6 High-level pseudo code for an explicit Euler for rigid particles. The continues properties $v(p,t), \omega(p,t)$ and $\mathbb{T}(p,t)$ are discretised in time and thus become $v(p), \omega(p)$ and $\mathbb{T}(p)$. t is the (discretised) time, Δt the time step size.

```
1: while t < T_terminal do                                   ▷ We simulate over a time span
2:     ∀p_i ∈ ℙ : ℂ(p_i) ← ∅                                 ▷ Clear set of collisions for particle p_i
3:     for p_i, p_j ∈ ℙ, p_i ≠ p_j do                        ▷ Run over all particle pairs
4:         ℂ(p_i) ← ℂ(p_i) ∪ findContacts(𝕋(p_i), 𝕋(p_j))
5:         ℂ(p_j) ← ℂ(p_j) ∪ findContacts(𝕋(p_i), 𝕋(p_j))
6:     end for
7:     for p_i ∈ ℙ do
8:         𝕋(p_i) ← update(𝕋(p_i), v(p_i), ω(p_i), Δt)      ▷ Update geometry
9:     end for                                               ▷ using velocity, rotation and time step size
10:    for p_i ∈ ℙ do
11:        (dv, dω) ← calcForces(ℂ(p_i))
12:        (v, ω)(p_i) ← (v, ω)(p_i) + Δt · (dv, dω)         ▷ Update velocity and rotation
13:    end for
14:    t ← t + Δt
15: end while
```

The concept defines a marker (Algorithm 9): Let $\mathbb{A}$ identify a set of active nodes from $\mathcal{T}$. The union of all sets labelled by $\mathbb{A}$ yields all triangles from a particle that participate in collision checks. At the begin of a particle-to-particle comparison, only the particles' roots are active. From there, we work our way down into finer and finer geometric representations as long as the surrogate models suggest that there might be some contacts, until we eventually identify real contact points stemming from the finest mesh.

Lemma 4 *The hierarchical algorithm yields exactly the same outcome as our baseline code over sets $\mathbb{T}^\epsilon(p_i)$ and $\mathbb{T}^\epsilon(p_j)$. Algorithm 9 is correct.*

Proof 5 *The argument relies on three properties:*

1. *If a contact point is identified for a surrogate triangle, it is not added to the set of contact points. Therefore, a given active set never identifies artificial/too many contact points.*

2. *A contact point is added if it stems from the comparison of two triangles from the fine grid tessellations which are in the active sets.*

3. *Let two triangles t_i^ϵ and t_j^ϵ yield a contact point. As surrogates are conservative, they belong to nodes (triangle sets) $\mathbb{T}(p_i)$ and $\mathbb{T}(p_j)$ with $\mathbb{T}(p_i) \sqsubseteq_{child} \hat{t}_i$ and*

Algorithm 7 Multiresolution contact detection within explicit time stepping. It compares two particles p_i and p_j given by their surrogate trees $\mathcal{T}(p_i)$ and $\mathcal{T}(p_j)$ with each other.

$\mathbb{A}_i \leftarrow root(\mathcal{T}(p_i))$, $\mathbb{A}_j \leftarrow root(\mathcal{T}(p_j))$ ▷ Set of active triangles to check
while $\mathbb{A}_i \neq \emptyset \vee \mathbb{A}_j \neq \emptyset$ **do**
  $\mathbb{A}_{i,\text{new}} \leftarrow \emptyset$, $\mathbb{A}_{j,\text{new}} \leftarrow \emptyset$
  for $t_i \in \mathbb{A}_i, t_j \in \mathbb{A}_j$ **do**
    $c \leftarrow$ CONTACT_ITERATIVE(t_i, t_j) ▷ Use context-specific ϵ depending on t_i, t_j
    if $c = \odot \wedge t_i \in \mathbb{T}_0^\epsilon(p_i) \wedge t_j \in \mathbb{T}_0^\epsilon(p_j)$ **then** ▷ Not converged on non-surrogate triangles
      $c \leftarrow$ CONTACT_COMPARISON(t_i, t_j) ▷ Use comparison-based algorithm this time
    end if
    if $c \neq \perp$ **then**
      if $t_i \in \mathbb{T}_0^\epsilon(p_i) \wedge t_j \in \mathbb{T}_0^\epsilon(p_j)$ **then** ▷ No surrogate triangles,
        $\mathbb{C}(p_i) \leftarrow \mathbb{C}(p_i) \cup \{c\}$, $\mathbb{C}(p_j) \leftarrow \mathbb{C}(p_j) \cup \{c\}$ ▷ i.e. proper contact point
      else ▷ Unfold
        if $t_i \in \mathbb{T}_0^\epsilon(p_i)$ **then**
          $\mathbb{A}_{i,\text{new}} \leftarrow \mathbb{A}_{i,\text{new}} \cup \{t_i\}$
        else
          $\mathbb{A}_{i,\text{new}} \leftarrow \mathbb{A}_{i,\text{new}} \cup \{\hat{t} : \hat{t} \sqsubseteq_{\text{child}} t_i\}$
        end if
        if $t_j \in \mathbb{T}_0^\epsilon(p_j)$ **then**
          $\mathbb{A}_{j,\text{new}} \leftarrow \mathbb{A}_{j,\text{new}} \cup \{t_j\}$
        else
          $\mathbb{A}_{j,\text{new}} \leftarrow \mathbb{A}_{j,\text{new}} \cup \{\hat{t} : \hat{t} \sqsubseteq_{\text{child}} t_j\}$
        end if
      end if
    end if
  end for
  $\mathbb{A}_i \leftarrow \mathbb{A}_{i,\text{new}}$, $\mathbb{A}_j \leftarrow \mathbb{A}_{j,\text{new}}$
end while

$\mathbb{T}(p_j) \sqsubseteq_{child} \hat{t}_j$ *in* $\mathcal{T}(p_i)$ *or* $\mathcal{T}(p_j)$*, respectively. These surrogates fulfil*

$$t_i^\epsilon \cap t_j^\epsilon \neq \emptyset \Rightarrow \hat{t}_i^\epsilon \cap \hat{t}_j^\epsilon \neq \emptyset. \tag{6.1}$$

and therefore are replaced in the active set by their children in Algorithm 6 before the respective algorithm terminates.

The correctness of the algorithm follows from bottom-up induction over the levels of $\mathcal{T}$*: The property holds directly for the finest surrogate levels* $\mathbb{T}_1$ *of the tree. Any violation thus has to arise from* $\mathbb{T}_k, k \geq 2$ *in* p_i *or* p_j*. We apply the arguments recursively.*

We have two triangle-to-triangle comparison strategies on the table (hybrid and comparison-based) which are robust, i.e. always yield the correct solution. If we employ the comparison-based approach only, the $c = \odot$ condition never holds and

the corresponding branch is never executed. Otherwise, our algorithmic blueprint implements the hybrid's fall-back as it automatically re-evaluates the contact search for $c = \odot$. However, it is indeed sufficient to rerun this a posteriori contact search if and only if both triangles stem from the finest triangle discretisation:

Corollary 2 *On the surrogate levels within the tree, it is sufficient to use the (efficient) iterative collision detection algorithm (Algorithm 3, bottom), without falling back to the comparison-based variant.*

Proof 6 *Let $\mathbb{T}^\epsilon(p_i) \cap \mathbb{T}^\epsilon(p_j) \neq \emptyset$, i.e. two particles collide. We assume the lemma is wrong, i.e. the tree unfolding terminates prematurely. This assumption formally means*

$$\exists t_i \in \mathcal{T}(p_i), t_j \in \mathcal{T}(p_j) : r(p_i, p_j) = \bot, \tag{6.2}$$

with

$$\exists t_{0,i} \in \mathbb{T}(p_i), t_{0,j} \in \mathbb{T}(p_j) : \; t_{0,i} \sqsubseteq_{child} \ldots \sqsubseteq_{child} t_i \wedge t_{0,j} \sqsubseteq_{child} \ldots \sqsubseteq_{child} t_j \wedge t_i^\epsilon \cap t_j^\epsilon \neq \emptyset. \tag{6.3}$$

This assumption is a direct violation of the definition of a surrogate model which has to be conservative.

6.1.2 Implicit Euler with multiresolution acceleration

Picard iterations can exploit the multiscale hierarchy by looping over the hierarchy levels top down: Per iteration of Algorithm 8, we have to identify all contact points for the current particle configuration. This search for contact points is the same search as we use it in an explicit Euler. If we replace the contact detection within the inner loop with our multiscale contact detection from Section 6.1.1, we obtain an implicit Euler where the surrogate concept is used *within the Picard loop* as multiresolution acceleration. The surrogate concept enters the algorithm's implementation as a black-box.

Assumption 1 *Our implicit time stepping problem exhibits a contraction property, i.e. Picard iterations can solve the underlying non-linear equation system.*

Algorithm 8 High-level pseudo code for an implicit Euler.

```
1: while t < T_terminal do                                   ▷ We simulate over a time span
2:     ∀p_i ∈ ℙ : 𝕋^guess(p_i) ← 𝕋(p_i), v^guess(p_i) ← v(p_i), ω^guess(p_i) ← ω(p_i)
3:     while 𝕋^guess, v^guess or ω^guess change significantly for any p do
4:         ∀p_i ∈ ℙ : ℂ(p_i) ← ∅                          ▷ Clear set of collisions for particle p_i
5:         for p_i, p_j ∈ ℙ, p_i ≠ p_j do                 ▷ Run over all particle pairs
6:             ℂ(p_i) ← ℂ(p_i) ∪ findContacts(𝕋^guess(p_i), 𝕋^guess(p_j))
7:             ℂ(p_j) ← ℂ(p_j) ∪ findContacts(𝕋^guess(p_i), 𝕋^guess(p_j))
8:         end for
9:         for p_i ∈ ℙ do
10:            (dv, dω) ← calcForces(ℂ(p_i))
11:            (v^guess, ω^guess)(p_i) ← (v, ω)(p_i) + Δt · (dv, dω)
12:            𝕋^guess(p_i) ← update(𝕋(p_i), v^guess(p_i), ω^guess(p_i), Δt)
13:        end for
14:    end while
15:    ∀p_i ∈ ℙ : 𝕋(p_i) ← 𝕋^guess(p_i), v(p_i) ← v^guess(p_i), ω(p_i) ← ω^guess(p_i)
16:    t ← t + Δt
17: end while
```

Corollary 3 *An implicit Euler using surrogates within the Picard loop body to speed up the search for contact points yields the same output as a flat implicit code with the same number of Picard iterations.*

Proof 7 *This is a direct consequence of Lemma 4 and implies the algorithm's correctness.*

Though we end up with exactly the same number of Picard iterations, the individual iterations are accelerated by the multiresolution technique: For a localised contact between two particles, the surrogate tree is unfolded along a single or few branches of the tree. If the nodes within the tree hold roughly the same number of triangles, the number of triangles to be compared grows linearly with the number of Picard steps. We benefit both from a zapping through the resolution levels and the localisation of contacts, i.e. the fact that two particles usually collide only in a small area compared to the overall geometric object.

6.1.3 Implicit multi-resolution Euler

A more bespoke implicit multiresolution algorithm arises from ideas inspired by multilevel non-linear equation system solvers. The multiscale Algorithm 6.1.2 consists of two nested while loops—the outer loop stems from the Picard iterations, the inner loop realises the tree unfolding—which we can permute. We obtain an algorithm

that runs top-down via the active sets through the surrogate hierarchies and unfolds the trees step by step. Per unfolding step, it uses the Picard loop to converge on the selected hierarchy level. The rationale behind such a permutation is the observation that the efficiency of a nonlinear equation system solver hinges on the availability of a good initial guess. Surrogate resolution levels might be well-suited to deliver a good initial guess of what $\mathbb{T}$ looks like in the next time step. This train of thought is similar to the extension of multigrid into full multigrid. On the other hand, the same multigrid analogy suggests that we do not have to converge on a surrogate level, as the level supplements only a guess anyway. In the extreme case, it is thus sufficient to run one Picard iteration per unfolding step only.

Algorithm 9 Multiresolution contact detection within explicit time stepping. It compares two particles p_i and p_j given by their surrogate trees $\mathcal{T}(p_i)$ and $\mathcal{T}(p_j)$ with each other.

$\mathbb{A}_i \leftarrow root(\mathcal{T}(p_i))$, $\mathbb{A}_j \leftarrow root(\mathcal{T}(p_j))$ ▷ Set of active triangles to check
while $\mathbb{A}_i \neq \emptyset \vee \mathbb{A}_j \neq \emptyset$ **do**
  $\mathbb{A}_{i,\text{new}} \leftarrow \emptyset$, $\mathbb{A}_{j,\text{new}} \leftarrow \emptyset$
  for $t_i \in \mathbb{A}_i, t_j \in \mathbb{A}_j$ **do**
    $c \leftarrow$ CONTACT_ITERATIVE(t_i, t_j) ▷ Use context-specific ϵ depending on t_i, t_j
    if $c = \odot \wedge t_i \in \mathbb{T}_0^\epsilon(p_i) \wedge t_j \in \mathbb{T}_0^\epsilon(p_j)$ **then** ▷ Not converged on non-surrogate triangles
      $c \leftarrow$ CONTACT_COMPARISON(t_i, t_j) ▷ Use comparison-based algorithm this time
    end if
    if $c \neq \perp$ **then**
      if $t_i \in \mathbb{T}_0^\epsilon(p_i) \wedge t_j \in \mathbb{T}_0^\epsilon(p_j)$ **then** ▷ No surrogate triangles,
        $\mathbb{C}(p_i) \leftarrow \mathbb{C}(p_i) \cup \{c\}$, $\mathbb{C}(p_j) \leftarrow \mathbb{C}(p_j) \cup \{c\}$ ▷ i.e. proper contact point
      else ▷ Unfold
        if $t_i \in \mathbb{T}_0^\epsilon(p_i)$ **then**
          $\mathbb{A}_{i,\text{new}} \leftarrow \mathbb{A}_{i,\text{new}} \cup \{t_i\}$
        else
          $\mathbb{A}_{i,\text{new}} \leftarrow \mathbb{A}_{i,\text{new}} \cup \{\hat{t} : \hat{t} \sqsubseteq_{\text{child}} t_i\}$
        end if
        if $t_j \in \mathbb{T}_0^\epsilon(p_j)$ **then**
          $\mathbb{A}_{j,\text{new}} \leftarrow \mathbb{A}_{j,\text{new}} \cup \{t_j\}$
        else
          $\mathbb{A}_{j,\text{new}} \leftarrow \mathbb{A}_{j,\text{new}} \cup \{\hat{t} : \hat{t} \sqsubseteq_{\text{child}} t_j\}$
        end if
      end if
    end if
  end for
  $\mathbb{A}_i \leftarrow \mathbb{A}_{i,\text{new}}$, $\mathbb{A}_j \leftarrow \mathbb{A}_{j,\text{new}}$
end while

Our advanced variant of the implicit Euler is an *outer-loop multi-resolution Picard scheme*. Let the Picard loop start from the coarsest surrogate representation per particle (Algorithm 10). These representations form our initial active sets. Af-

ter the Picard step, any surrogate triangle for which the hybrid algorithm has not terminated or for which we identified a contact point is replaced by its next finer representation. In the tradition of value-range analysis, we widen the active set [158]. The Picard loop terminates if the plain algorithm's termination criteria hold, i.e. the outcome of two subsequent iterations does not change dramatically anymore, and no surrogate tree node has unfolded anymore throughout the previous iterate.

The algorithm is completed by a clean up which removes "obsolete" triangles from the active set: If all children of a surrogate triangle do certainly not contribute a contact point anymore, they are replaced with their parent surrogate triangle. We narrow the active set.

Implementation remark 3. Different to the explicit scheme, we maintain an active set $\mathbb{A}(p_i, p_j)$ per particle-particle combination p_i, p_j: A particle p_i can exhibit a very coarse surrogate representation against one particle, while using a very detailed mesh when we compare it to another one. While the number of particle-particle combinations is potentially huge, it is small in practice, as particles are rigid and thus cannot cluster arbitrarily dense. □

Our genuine multiscale formulation stresses the convergence assumptions: While Assumption 1 guarantees the convergence of the Picard iterations on the finest level, our multi-resolution approach may push the solution into the wrong direction via the surrogate levels and thus make the initial guess on the next finer level leave the single level's convergence domain.

Assumption 2 *We assume that a Picard iteration on any level of the surrogate trees yields a new solution on the same or a finer resolution which preserves the Picard iteration's contraction property.*

Lemma 5 *If Assumption 2 holds and if Algorithm 10 terminates, it delivers the correct solution.*

Proof 8 *We have to study two cases over the active sets $\mathbb{A}(p_i, p_j)$ and $\mathbb{A}(p_j, p_i)$. First, assume that $(t_{\mathbb{A}(p_i,p_j)}, t_{\mathbb{A}(p_j,p_i)}) \in \mathbb{A}(p_i, p_j) \times \mathbb{A}(p_j, p_i)$ yields an invalid contact point, i.e. a contact point that does not exist in $\mathbb{T}_0(p_i)$ compared to $\mathbb{T}_0(p_j)$. One of*

the triangles has to be a surrogate triangle. They are replaced by their children and the algorithm has not terminated. Instead, we approach the solution further.

In the other case, assume that the algorithm has terminated yet misses a triangle pair $(t(p_i), t(p_j)) \notin \mathbb{T}(p_i) \times \mathbb{T}(p_j)$ *which contributes a contact point in the plain model. Due to Definition 10 over conservative surrogates,*

$$\forall t(p_i) \in \mathbb{T}_0(p_j), \exists \hat{t}_{\mathbb{A}(p_i,p_j)} \in \mathbb{A}(p_i): \quad t(p_i) \sqsubseteq_{child} \ldots \sqsubseteq_{child} \hat{t}_{\mathbb{A}(p_i,p_j)} \tag{6.4}$$

such that $\hat{t}_{\mathbb{A}(p_i,p_j)}$ *yields a contact point. This point is eventually removed as the corresponding* $\hat{t}_{\mathbb{A}(p_i,p_j)}$ *is replaced by its children. The algorithm has not terminated yet.*

Corollary 4 *The removal of triangles from the active set can cause Algorithm 10 to violate Assumption 2.*

Proof 9 *If no triangles are ever removed from the active set, the proof of Lemma 5 trivially demonstrates that the algorithm terminates always, as the surrogate tree is of finite depth and width. Even if we overshoot with the Picard iterations, i.e. if we violate the contraction property, we will, in the worst case, get* $\mathbb{A}(p_i, p_j) = \mathbb{T}(p_i)$. *From hereon, the algorithm converges.*

If we however remove triangles, it is easy to see that we cannot guarantee that we do not introduce cycles or even amplify oscillations. The contraction property is violated.

Implementation remark 4. In practice, Corollary 4 implies that we, on the one hand, have to damp the Picard iterations. We artificially reduce the force contributions from coarse surrogate levels to avoid oscillations. On the other hand, we work with a memory set $\mathbb{U}(p_i, p_j)$ in which we hold references to triangles which have been removed from the active set. Once they are re-added, we veto any subsequent removal from hereon and, hence, the activation of such triangles' surrogates. □

6.1.4 Implementation

There are two reasons why our multi-resolution algorithms are expected to yield better performance than a straightforward textbook implementation: First and foremost, we expect the number of triangle-to-triangle comparisons to go down compared to a flat, single-level approach. The multiscale algorithm iteratively narrows down the region of a particle where contacts may arise from. These savings on the finer geometric resolutions compensate for additional checks with surrogate triangles. However, any cost amortisation has to be studied carefully—in particular for the implicit, non-linear case where trees unfold and collapse again—and it hinges upon an efficient realisation: In this context, we expect the streaming, comparison-free variants of our algorithm to benefit from vector architectures.

The multiresolution representation of an object can be computed at simulation startup as a preprocessing step. Though we keep the multiresolution hierarchy when particles move and rotate, the flattening of the active sets of triangles from $\mathcal{T}(p)$ into a sequence of coordinates is done on-the-fly and the flattened data is not held persistently.

On the one hand, this ensures that the memory overhead remains under control. On the other hand, it pays tribute to the fact that the active set changes permanently. To remain fast despite permanently changing active sets, we pick $N_{\text{surrogate}}$ such that the finest nodes within $|\mathbb{T}|$ hold triangle sequences for which streaming instructions such as AVX already pay off. The tree clusters $\mathbb{T}$ into segments that fit to the architecture, and Algorithm 3 hence does not process all triangles from $\mathbb{T}$ in one batch. Instead it runs over subchunks of batches.

We apply this argument recursively and make each non-leaf node within $\mathcal{T}$ hold a set of triangles, too. We make the nodes in the surrogate tree host many triangles and the tree overall shallow, such that the per-node data cardinality again ensures that we benefit from vector units. This "tweak $N_{\text{surrogate}}$" idea however does not fit perfectly to multiscale algorithms where the coarser tree levels typically do not occupy a complete vector length. It would be a coincidence if $|\mathbb{T}|$ and $N_{\text{surrogate}}$ yielded only nodes that fill a vector unit completely on each and every surrogate level. Therefore, we do not run triangle-to-triangle comparisions within the tree directly.

Instead, we make the tree/triangle traversal collect all comparisons to be made with a buffer. Once we have identified all triangle collisions to be computed, we stream the whole buffer through the vector units. We merge the triangle representations on-the-fly.

Implementation remark 5. If our surrogate tree hosts more than one triangle per non-leaf node, the algorithm has to be completed by a further clean-up step which ensures that the active set remains consistent with the tree: It runs through the active set of a particle-particle combination once again. If any of a surrogate triangle's children is part of the active set, the surrogate is removed from the set, but all of its children become active. Without such an additional sweep over the active set, all triangles of a node could be active plus the children of one triangle which implies that we test against the children triangle set plus their surrogate triangle. Restricting the node triangle cardinality for surrogate levels to one renders the additional clean-up unnecessary. □

6.2 Runtime results

Our algorithms yield correct results (Lemma 4, Corollary 3 and Lemma 5), but they do not provide an efficiency guarantee. We hence collect runtime results, i.e. gather empirical evidence. Real-world experiments benchmarked against measurements remain out-of-scope for the present paper. We furthermore continue to focus on the actual collision detection and neglect the impact of different time step sizes—in particular comparisions between implicit and explicit schemes facilitating different stable time step choices—the cost of a coarse-grain neighbour search via a grid, e.g., and notably the construction cost for the surrogates which are done offline prior to the simulation run. All experiments are run on Intel Xeon E5-2650V4 (Broadwell) chips in a two socket configuration with 2×12 cores. They run at 2.4 GHz, though TurboBoost can increase this up to 2.9 GHz. However, a core executing AVX(2) instructions will fall back to a reduced frequency (minimal 1.8 GHz) to stay within the TDP limits [149].

Our node has access to 64 GB TruDDR4 memory, which is connected via a

hierarchy of three inclusive caches. They host $12 \times (32 + 32)$ KiB, 12×256 KiB or 12×2.5 MiB, respectively. We obtain around 109 GB/s in the Stream TRIAD [159] benchmark on the node which translates into 4,556 MB/s per core. The node has a theoretical single precision peak performance between 2.4 (non-AVX mode and baseline speed) and 46.4 Gflop/s per core (`AVX 2.0 FMA3` with full turbo boost). All of our calculations are ran in single precision. They are translated with the Intel 19 update 2 compiler and use the flags `-std=c++17 -O3 -qopenmp -march=native -fp-model fast=2`, i.e. we tailor them to the particular instruction set.

All presented performance counter data are read out through LIKWID [160]. DEM codes are relatively straightforward to parallelise as their particle-particle interaction is strongly localised: We can combine grid-based parallelism (neighbour cells) with an additional parallelisation over the particle pairs [5]. The load balancing of these concurrency dimensions however remains challenging. As our ideas reduce the comparison cost algorithmically yet do not alter the concurrency character, we stick (logically) to single core experiments to avoid biased measurements due to parallelisation or load balancing overheads. Yet, we artificially scale up the setup by replicating the computations per node over multiple OpenMP threads whenever we present real runtime data or machine characteristics, and then break down the data again into cost per replica per core. This avoids that simple problems fit into a particular cache or that memory-bound applications have exclusive access to two memory controllers.

6.2.1 Experimental setup

We work with two simple benchmark setups, before we validate our results for large numbers of particles. In the *particle-particle* setup, we study two spherical objects which are set on direct collision trajectory. They bump into each other, and then separate again. The setup yields three computational phases: While the particles approach, there is no collision and no forces act on the particles as we neglect gravity. When they are close enough, the particles exchange forces and the system becomes very stiff suddenly, before the objects repulse each other again and separate. We focus exlusively on the middle phase. Throughout this approach-and-

contact situation, the algorithmic complexity of the contact detection is in $\mathcal{O}(|\mathbb{T}|^2)$, as we assume that both particles have the same triangle count.

In the *particle-on-plane scenario*, we drop a spherical object onto a tilted plane. The particle hits the plane, bumps back in a slightly tilted angle, i.e. with a rotation, and thus hops down the plane. This problem yields free-fall phases which take turns with stiff in-contact situations. Furthermore, the area of the free particles which is subject to potential contacts changes all the time as the particle starts to rotate, and the contacts result from a complex geometry consisting of many triangles compared to a simplistic geometry with very few triangles. The underlying computational complexity is in $\mathcal{O}(|\mathbb{T}|)$.

In the *grid scenario*, we finally arrange 24,576 spheres in a Cartesian grid. Each particle slightly overlaps the epsilon region of it's neighbours. As there is no ground plane or gravity, the particles "float" in space. Due to the regular particle layout, the interaction pattern yields a Cartesian topology, i.e. each particle collides with four other particles initially.

Our codes work exclusively with *sphere-like particle shapes*, which result from a randomised parameterisation: We decompose the sphere with radius 1 into $|\mathbb{T}|$ triangles. If not stated otherwise, $|\mathbb{T}| = 1,280$. In the first two scenarios the vertices on the sphere which span the triangles are subject to a Perlin noise function, which offsets the vertex along the normal direction of the surface. $\eta_r = 1$ adds no noise and thus yields a perfect, triangulated sphere with radius 1 where all vertices are exactly 1 unit away from the sphere's origin. Otherwise, the per-vertex radius is from $[1, \eta_r]$. As we use a hierarchical noise model, a high η_r yields a degenerated shape which retains a relatively smooth surface.

For the implicit schemes, we consider the result converged when the update to the force and torque applied to every particle underruns a relative threshold of 1%. With this accuracy, single precision is sufficient. The Picard iterations are subject to damping and acceleration: Any update $(dv, d\omega)$ relative to the start configuration of a time step results from the weighted average between the currently computed forces and the forces of the previous step. If forces "pull" into one direction over multiple iterations, the updates behind trials become successively bigger. If the

forces oscillate, these oscillations are diminishing. Empirical data suggest that this choice helps us to meet Assumptions 1 and 2. $\epsilon = 10^{-2}$ is uniformly used on the finest mesh level. This is a relative quantity, i.e. chosen relative to the particle diameter. For the plane, we uniformly use $\epsilon = 10^{-2}$.

6.2.2 Surrogate properties

We first assess our surrogate geometry's properties. Our coarsest surrogate model consists of a single triangle. We compare this triangle's longest edge (diameter) $d_{k_{\max}}$ plus its corresponding $\epsilon_{k_{\max}}$ value to the radius $r_{\text{sphere}} = \frac{\eta_r}{2}$ of the bounding sphere of the fine grid object (Table 6.1). For the surrogate hierarchy, we use $N_{\text{surrogate}} = 8$ as coarsening factor; a choice we employ throughout the experiments. This implies that an object with $|\mathbb{T}| = 64$ triangles spans three surrogate organised as a tree: They host $|\mathbb{T}| = 64$ (original model), $|\mathbb{T}_{\lightning}| = 64/N_{\text{surrogate}} = 8$ and $|\mathbb{T}_{\lightning}| = 1$ triangles. In this first test, we approximate low frequency noise by scaling along one axis, i.e. we elongate the sphere along one direction yet do not introduce bumps or extrusions. With growing η_r, we obtain increasingly non-spherical objects resembling an ellipsoid. The rationale behind this simplified noise is that we eliminate non-deterministic effects and study the dominant sphere distortion effects.

Table 6.1: Different triangle counts $|\mathbb{T}|$ per spherish object scaled along one axis by a factor of μ. Per setup, we study the top level surrogate which contains one triangle and compare the maximum triangle diameter plus its halo size against the bounding sphere radius. Here we only report on the increase in ϵ for the coarsest surrogate ie. at the finest level $\epsilon = 0$.

	$\lvert\mathbb{T}\rvert = 80$		$\lvert\mathbb{T}\rvert = 320$		$\lvert\mathbb{T}\rvert = 1,280$		
η_r	$d_{k_{\max}}$	$\epsilon_{k_{\max}}$	$d_{k_{\max}}$	$\epsilon_{k_{\max}}$	$d_{k_{\max}}$	$\epsilon_{k_{\max}}$	r_{sphere}
1.0	0.09	0.49	0.09	0.50	0.08	0.52	0.50
1.2	0.10	0.55	0.10	0.56	0.10	0.56	0.60
1.4	0.34	0.54	0.12	0.65	0.11	0.66	0.70
1.8	0.14	0.84	0.89	0.53	1.35	0.51	0.90
2.6	1.54	0.58	2.24	0.51	2.36	0.52	1.30

The combination of $d_{k_{\max}}$ and $\epsilon_{k_{\max}}$ characterises the shape of our coarsest surrogate model. A large diameter relative to a small halo size describes a disc-like object. A small diameter relative to a large halo size describes a sphere-like object.

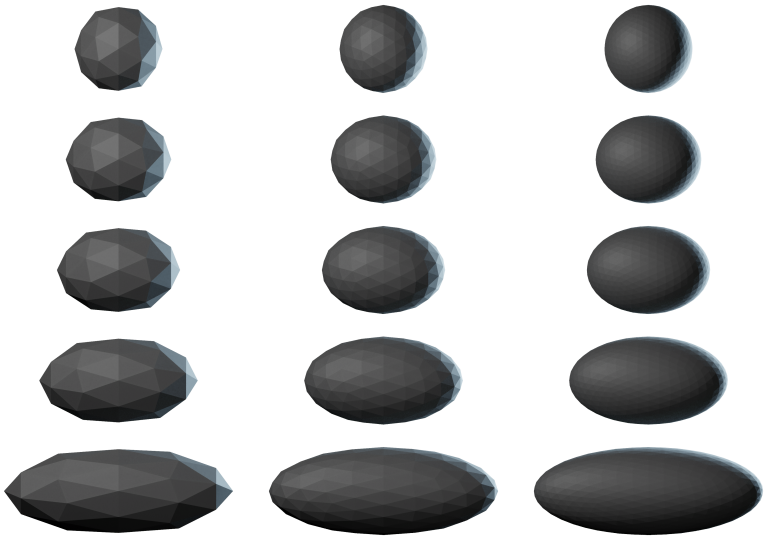


Figure 6.2: A series of objects created from unit spheres with increasing level of detail ($80 \leq |\mathbb{T}| \leq 1,280$ from left to right) and a scale factor ($1.0 \leq \eta_r \leq 2.6$ from top to bottom).

Different triangle counts for the fine grid model allow us to assess the impact of the level of detail of the fine grid mesh onto the resulting coarsest surrogate geometry.

Our surrogate model of choice on the coarsest level (cmp. the penalty term in (5.5) of the Appendix) almost degenerates to a point if the underlying triangulated geometry approximates a sphere. It approaches a bounding sphere. The triangle count approximating a spherical object does not have a significant qualitative or quantitative impact on this characterisation of the coarsest surrogate triangle. Once the triangulated mesh becomes less spherical, the surrogate triangle starts to align with the maximum extension of the fine mesh. It spreads out within the geometry along the geometry's longest diameter; an effect that is the more distinct the higher the fine geometry's triangle count. The halo layer $\epsilon_{k_{\max}}$ around the surrogate triangle, which is analogous to a sphere's radius if the surrogate triangle approaches a point, remains in the order of $r = 0.5$. This is the radius of the original unit sphere ($\eta_r = 1$).

For a close-to-spherical geometry, our volumetric surrogate model never exceeds 135% of the bounding sphere volume ($|\mathbb{T}| = 1,280$). For the highly non-spherical cases ($\eta_r = 2.6$) our surrogate volume can be as little as 37% ($|\mathbb{T}| = 80$) of the simple bounding sphere volume. This advantageous property results from the observation

that a growth of $d_{k_{\max}}$ anticipates any extension of the geometry, while the $\epsilon_{k_{\max}}$ ensures that the minimal geometry diameter, which is at least as large as the original sphere, remains covered by the surrogate triangle plus its halo environment.

Observation 9 *For highly non-spherical sets of triangles, our surrogate formalism yields advantageous representations. For spherical observations, it resembles the bounding sphere. This holds for all levels of the surrogate cascade.*

In a surrogate tree, fine resolution tree nodes (surrogate triangles) are characterised by the low triangle count measurements in Table 6.1 where localised patches are highly non-spherical (large η_r). Surrogate triangles belonging to coarser levels inherit characteristics corresponding to larger $|\mathbb{T}|$. We conclude that our triangle-based multiresolution approach is particularly advantageous as an early termination criterion ("there is certainly no collision") on the rather fine surrogate resolution levels within the surrogate tree, or is overall tighter fitting than bounding sphere formalisms for non-spherical geometries.

6.2.3 Hybrid single level contact detection

Even though our multiscale approach intends to reduce the number of distance calculations, a high throughput of the overall algorithm continues to hinge on the efficiency of the core distance calculation. We hence continue with studies around the explicit Euler where we omit the multiscale hierarchy. We work with the finest particle mesh representation only.

The assessment of the core comparison efficiency relies on our sphere-on-plane and the particle-particle setup. They represent two extreme cases of geometric comparisons: With the plane, a complex particle with many triangles hits very few triangles. As long as the triangles spanning the plane are large relative to the particle diameter, it is irrelevant how the plane is modelled, i.e. if it consists of solely one or two triangles or an arrangement of multiple triangles. The multiscale algorithms asymptotically approach a $|\mathbb{T}| : 1$ comparison with $\mathbb{T}$ given by the particle. When we collide two particles—we use the same triangle count for both—we can, in the worst case, run into a $|\mathbb{T}| : |\mathbb{T}|$ constellation. For both setups, we use strictly spherical

particles ($\eta_r = 1$).

For the triangle-triangle comparisons, two code variants are on the table: We can run the comparison-based (baseline) algorithm, or we can use the hybrid code variant which runs four steps of the iterative scheme before it checks if the two last iterates of the contact point differ by more than $C\epsilon_h$. We use $C \approx 1$. If the difference exceeds the threshold, our algorithm assumes that the code has not converged, and hence reruns the comparison-based code to obtain a valid contact assessment. The comparison-based code variant is 4-way vectorised and relies on Intel intrinsics. The hybrid variant is vectorised over batches of eight packed triangle pairs using an OpenMP `simd` annotation.

Table 6.2: Particle-particle scenario. We compare a comparison-based realisation (top) against a hybrid realisation (bottom). Per setup, we present the time-to-solution ([t]=s) per Euler step, i.e. one run through all possible triangle combinations, and we augment these data with MFlop/s rates split up into scalar and vectorised contributions. Vector calculations are categorised as 128 bit packed (SSE) or 256 bit packed (AVX) for four and eight simultaneous 32 bit floating point operations respectively. For the hybrid setup, we finally quantify how many triangle pairs had to be checked a posterio, i.e. as fallback, by the comparison-based algorithm. This runtime is included in the data. All measurements are given as the average per core.

	$\lvert\mathbb{T}\rvert$	Runtime	Scalar	Packed 128B	Packed 256B	Fallback
Comp.	12	$6.59 \cdot 10^{-2}$	$1.52 \cdot 10^{-2}$	$3.22 \cdot 10^{3}$		
	36	$5.17 \cdot 10^{-2}$	$1.90 \cdot 10^{-3}$	$3.46 \cdot 10^{3}$		
	140	$3.80 \cdot 10^{-1}$	$3.00 \cdot 10^{-4}$	$3.18 \cdot 10^{3}$		
	1,224	$4.35 \cdot 10^{1}$	$0.00 \cdot 10^{0}$	$3.04 \cdot 10^{3}$		
Hybrid	12	$4.99 \cdot 10^{-2}$	$6.27 \cdot 10^{1}$	$1.11 \cdot 10^{3}$	$1.07 \cdot 10^{4}$	7.7%
	36	$2.76 \cdot 10^{-2}$	$1.27 \cdot 10^{2}$	$9.69 \cdot 10^{2}$	$1.45 \cdot 10^{4}$	4.5%
	140	$1.83 \cdot 10^{-1}$	$1.97 \cdot 10^{2}$	$3.25 \cdot 10^{2}$	$1.84 \cdot 10^{4}$	1.2%
	1,224	$1.99 \cdot 10^{1}$	$2.42 \cdot 10^{2}$	$5.30 \cdot 10^{1}$	$2.10 \cdot 10^{4}$	0.028%

Our hybrid approach outperforms a sole comparison-based approach robustly for the $|\mathbb{T}| : |\mathbb{T}|$ setups. For strongly ill-balanced triangle counts, the insulated comparison-based approach is superior (Table 6.2). The comparison-based code variant is not able to benefit from AVX at all (not shown), while the hybrid AVX usage increases with increasing triangle counts. We end up with up to 40–45% "turbo-mode" peak performance which we have to calibrate with the AVX frequency

Table 6.3: Experiments from Table 6.2 for the particle-on-plane setup.

	$\lvert\mathbb{T}\rvert$	Runtime	Scalar	Packed 128B	Packed 256B	Fallback
Comp.	12	$2.60 \cdot 10^{-2}$	$3.87 \cdot 10^{-2}$	$3.33 \cdot 10^{3}$		
	36	$6.50 \cdot 10^{-2}$	$1.96 \cdot 10^{-2}$	$3.85 \cdot 10^{3}$		
	140	$1.84 \cdot 10^{-1}$	$1.28 \cdot 10^{-2}$	$4.09 \cdot 10^{3}$		
	1,224	$1.98 \cdot 10^{0}$	$2.90 \cdot 10^{-3}$	$4.27 \cdot 10^{3}$		
Hybrid	12	$2.80 \cdot 10^{-2}$	$1.06 \cdot 10^{1}$	$1.25 \cdot 10^{3}$	$9.34 \cdot 10^{3}$	6.3%
	36	$6.70 \cdot 10^{-2}$	$1.56 \cdot 10^{1}$	$1.34 \cdot 10^{3}$	$1.19 \cdot 10^{4}$	5.1%
	140	$1.79 \cdot 10^{-1}$	$2.15 \cdot 10^{1}$	$1.34 \cdot 10^{3}$	$1.35 \cdot 10^{4}$	4.8%
	1,224	$1.85 \cdot 10^{0}$	$3.03 \cdot 10^{1}$	$9.32 \cdot 10^{2}$	$1.48 \cdot 10^{4}$	3.6%

reduction [149]. The relative number of fallbacks, i.e. situations where the iterative scheme does not converge within four iterations, decreases with growing geometry detail, while the same effect is not as predominant for the particle-on-plane scenario.

Our data confirm the superiority of the hybrid approach for the particle-particle comparisons [4, 5]. They confirm that the approximation of the Hessian does not significantly harm the robustness, even though the number of fallbacks becomes non-negligible. Our arithmetic intensity determines how much improvement results from the hybrid strategy. It is significantly higher for the particle-particle setup as opposed to the particle-on-plane. Therefore, only the former prospers through vectorisation. We see an increased fallback for decreased geometric detail due to the larger relative epsilon, which is used to identify fallback conditions.

Observation 10 *As long as we do not compare extreme cases (single triangle vs. a lot of triangles), the hybrid approach is faster. It is thus reasonable to employ it on all levels of the surrogate tree, even though it might be reasonable to skip iterative comparisons a priori if the coarsest surrogate level is involved. The latter observation does not result from a mathematical "non-robustness" but is a sole machine effect.*

6.2.4 Multiresolution comparisons for explicit time stepping

Within an explicit time stepping code, our multiresolution approach promises to eliminate unnecessary comparisons since it identifies "no collission" constellations quickly through the surrogates: Whenever it compares two geometries, the algorithm runs through the resolution levels top-down (from coarse to fine). The monotonicity

of the surrogate definition implies that we can stop immediately if there is no overlap between two surrogates. The code either employs the pure geometry-based approach or the hybrid strategy on all levels. We focus on spherical particles (η_r=1) discretised by 1,224 triangles, and average over 100 time steps such that we run into no-collision phases for the particle-particle setup and see the particle roll down the plane for particle-on-plane.

Table 6.4: Time-to-solution ([t]=s) and number of triangle distance checks for an explicit Euler step for the particle-particle collision (top) and the particle-on-plane setup (bottom). We compare a comparision-based setup to a hybrid approach on a single level vs. a surrogate hierarchy which is traversed from coarse to fine. For the hybrid configuration, we present the number of comparison-based fallbacks vs. the number of iterative comparisons. Each iterative comparison of two triangles consists of four Newton steps. Only if these four steps fail to converge, the algorithm issues the comparison-based postprocessing. Both the iterative comparisons plus the (fewer) comparison-based postprocessing steps determine the runtime (right column).

	Comparison-based		Hybrid		
Method	#tri. comp.	Runtime	#tri. comp.	#iterative	Runtime
Single level	$1.50 \cdot 10^6$	$3.96 \cdot 10^0$	$1.66 \cdot 10^3$	$1.50 \cdot 10^6$	$1.87 \cdot 10^0$
Surrogate hierarchy	$8.20 \cdot 10^3$	$2.60 \cdot 10^{-2}$	$1.44 \cdot 10^2$	$7.64 \cdot 10^3$	$1.80 \cdot 10^{-2}$
Single level	$6.27 \cdot 10^5$	$1.68 \cdot 10^0$	$1.57 \cdot 10^4$	$6.27 \cdot 10^5$	$8.00 \cdot 10^0$
Surrogate hierarchy	$5.27 \cdot 10^3$	$7.68 \cdot 10^{-2}$	$4.96 \cdot 10^2$	$5.03 \cdot 10^3$	$4.00 \cdot 10^{-2}$

Our measurements confirm the superiority of the hybrid scheme in the surrogate context (Table 6.4): In line with Section 6.2.3, no multiresolution setup with comparison-based contact detections on surrogate levels is able to outperform the configurations where all levels are tackled through the hybrid approach. Further studies where different variants are used on different (surrogate) levels are beyond scope.

In our two-particles scenario, the particles yield $1,224^2$ comparisons per time step if no surrogate helper data structure is used. Our data reflects the quadratic (particle-particle) or linear (particle-on-plane) computational complexity. In both cases, the number of triangle comparisons is reduced by more than an order of magnitude through the surrogate hierarchy, and the surrogate version outperforms its single-level baseline robustly. The hierarchical scheme's additional computational cost (overhead) is negligible, though it does not significantly alter the ratio of iter-

ative checks to fallback comparisons in the hybrid scheme.

Observation 11 *Our surrogate technique efficiently reduces the number of comparisons between two geometries, as "no-collision" setups are identified with low computational cost.*

6.2.5 Multiresolution comparisons for implicit time stepping

Implicit methods are significantly more stable then their explicit counterpart. The price to pay for this is an increased computational complexity: The Picard iterations that we use imply that we have to run the core contact point detection more often per time step. The Picard iterations' update of collision point detections imply that the surrogate tree does not unfold linearly anymore. While the explicit time stepping algorithm runs through the tree from coarse to fine, the implicit scheme descends into finer levels yet might, through the iterative updates of the rotation and position, find alternative tree parts that have to be taken into account too or instead. The increased stability of implicit time stepping allows users to pick larger time step sizes which, in practice, compensate to some degree of the increased compute load. This problem-specific cost amortisation is beyond scope here, i.e. we compare implicit and explicit schemes with the same time step size. All data results from spherical particles and the default triangle counts. We average all measurements over 100 time steps and include the "roll down the plane" situation for the particle-on-plane setup.

Table 6.5: Average number of Picard iterations per time step for our first two scenarios.

	Particle-particle		Particle-plane	
Method	Comparison-based	Iterative	Comparison-based	Iterative
Single level or surrogate within Picard	4.6	4.9	7.1	7.1
Multiscale Picard	6.2	6.2	13.0	13.1

Averaged over 100 time steps, we observe that the number of Picard iterations is small and bounded (Table 6.5). We study the impact of a switch to the iterative

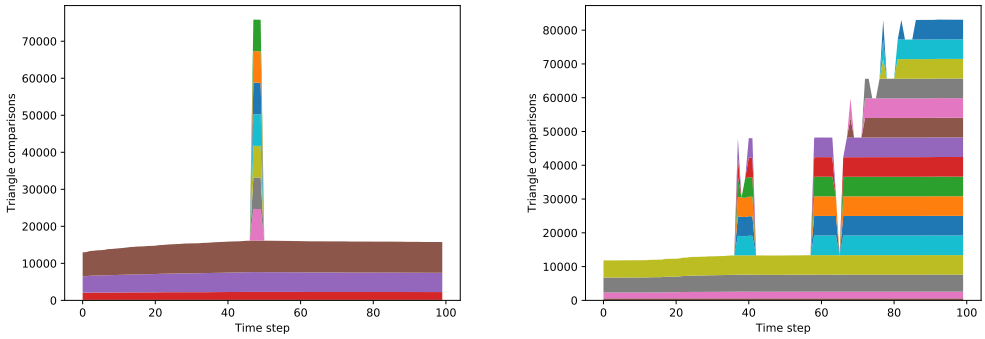


Figure 6.3: Number of triangle-to-triangle comparisons over time per surrogate representation level. The data stems from the particle-particle (left) and the particle-on-plane setup (right) subject to the implicit time stepping. The lowest layer illustrates the triangle comparison count for the first iteration, the next layer for the second iteration and so forth. The number of layers corresponds to the total number of iterations.

scheme, with the hybrid fallback on the finest level, and observe that it slightly increases the Picard iteration count. The usage of a multiscale method merged into the Picard iterations increases the iteration count, too. Both modifications yield flawed contact point guesses and thus require us to run more Picard iteration steps overall. The wrong guesses have to be compensated later on.

Observation 12 *Both the iterative approximation of contact points and the "one Picard step before we widen the active set" strategy increase the total number of required Picard iterations.*

The surrogate hierarchy yields an efficient early termination criterion for our collision detection. If there is no collision, the code does not step down into the fine grid resolutions. This property carries over from the explicit to the implicit algorithm (Figure 6.3). An increase of the computational cost by a factor of 4.6 is acceptable in return for an implicit scheme. We however observe that this increase holds for brief point contacts only. It raises to a factor of 13.1 if contacts persist. In our example, this happens once the spherical object starts to roll and slide down the tilted plane.

Within our multi-resolution framework, the cost per Picard iteration is not uniform and constant but depends heavily on the surrogate tree fragments that are

Table 6.6: Time per time step ([t]=s]) and triangle distance comparisons for our implicit schemes for the particle-particle collision (top) and the particle-on-plane setup (bottom).

	Comparison-based		Hybrid		
Method	#tri. comp.	Runtime	#tri. comp.	#iterative	Runtime
Single level	$6.89 \cdot 10^8$	$1.69 \cdot 10^2$	$8.78 \cdot 10^5$	$7.34 \cdot 10^8$	$7.44 \cdot 10^1$
Surrogate within Picard	$3.97 \cdot 10^6$	$1.08 \cdot 10^0$	$7.14 \cdot 10^4$	$3.70 \cdot 10^6$	$7.20 \cdot 10^{-1}$
Multiscale Picard	$1.82 \cdot 10^6$	$7.70 \cdot 10^{-1}$	$2.25 \cdot 10^4$	$1.70 \cdot 10^6$	$4.70 \cdot 10^{-1}$
Single level	$4.43 \cdot 10^8$	$9.61 \cdot 10^1$	$1.11 \cdot 10^7$	$4.43 \cdot 10^8$	$4.41 \cdot 10^1$
Surrogate hierarchy	$3.59 \cdot 10^6$	$4.80 \cdot 10^{-1}$	$3.44 \cdot 10^5$	$3.42 \cdot 10^6$	$2.20 \cdot 10^{-1}$
Multiscale Picard	$3.50 \cdot 10^6$	$4.90 \cdot 10^{-1}$	$3.11 \cdot 10^5$	$3.35 \cdot 10^6$	$1.90 \cdot 10^{-1}$

used. The cost are in particular non-uniform for non-simplistic setups. The growth in Picard iterations (on average) per time step (Table 6.5) increases the number of triangle-to-triangle checks, compared to the explicit schemes (Table 6.4), by exactly this factor if we stick to a plain geometry model. Yet, it does not manifest in an explosion of the runtime (Table 6.6) if we employ the surrogate trees. They help to reduce the compute cost dramatically, as we study only those parts of the surrogate tree which might induce a collision. We prune the tree per Picard iteration.

Observation 13 *Our multiscale Picard approach is particularly beneficial for strongly instationary setups where the topology of particle interactions changes quickly.*

Permuting and fusing the Picard iteration loops and the traversal over the surrogate tree reduces the number of triangle-to-triangle comparisons further. This observation holds for the particle-particle setup. It does not hold for the particle-on-plane. The advanced version benefits from the fact that we memorise the active set in-between two Picard iterations: While the implicit version from Section 6.1.2 runs through the whole tree starting from the root in every iteration, our advanced version starts from a certain resolution and unfolds at most one level per Picard step. We save the progress through coarser resolutions, and we do not step all the way down in early iterations. This state-based approach works as our narrowing is effective: if we step down into a "wrong" part of the tree and find out that these fine resolutions do not contribute towards the final force, we successively remove

these fine resolutions from the (active) comparison sets again. For a sphere rolling or hopping down a plane, the active set remains almost invariant throughout the Picard iterations, and we do not benefit from the narrowing or an early termination. We do however benefit here from the adaptive localisation of the contact detection within the tree.

Observation 14 *The multiscale Picard approach in combination with a hybrid contact detection keeps the cost of the implicit time stepping bounded by a factor of four compared to an explicit scheme.*

6.2.6 Multiresolution comparisons for implicit time stepping with large scale differences

To assess collisions between objects of different scales, we first instantiate the particle-particle scenario with both particles being spheres approximated by $|\mathbb{T}| = 80$ triangles ($\eta_r = 1$). We fix one of the spheres with a unit diameter, and scale the other one up. All data studies situations where the two particles are close and yield contact points.

Table 6.7: Time ([t]=s) and triangle distance per time step for two colliding particles of different size with comparable triangle sizes.

Scale	Triangles	Iterations	#tri. comp	#iterative	Runtime
1	80	13	$1.59 \cdot 10^1$	$1.45 \cdot 10^2$	$8.72 \cdot 10^{-2}$
2	320	19	$4.18 \cdot 10^1$	$5.05 \cdot 10^2$	$2.25 \cdot 10^{-1}$
4	1,280	20	$4.66 \cdot 10^1$	$5.88 \cdot 10^2$	$2.60 \cdot 10^{-1}$
8	5,120	17	$3.80 \cdot 10^1$	$5.44 \cdot 10^2$	$2.40 \cdot 10^{-1}$

There is no clear trend relating the number of Picard iterations to the scale of the involved particles (Table 6.8). As the scale difference grows, the number of triangle-triangle comparisons reduces slightly (cmp. how often the iterative scheme is invoked), while more iterative updates run into the fallback, i.e. require a posteriori, if-based triangle comparisons. The cost to compare a large with a small particle thus remains in the same order as the comparison of two equally sized particles. There is no clear runtime trend.

The bigger the size difference between two triangles the more likely it is that the hybrid scheme runs into the fallback (cmp. data in Table 6.3 where the slope

triangles are large compared to the particle triangles). The frequent fallbacks team up with the effect that coarse (surrogate) triangles within the larger particle overlap large numbers of surrogate triangles on the smaller collision partner. Relatively large fractions of the small particle's surrogate tree have to be unfolded.

Table 6.8: Time ([t]=s) plus triangle distance comparisons per implicit time step for a particle-particle setup (80 triangles) where one particle is scaled relative to the other one.

Scale	Triangles	Iterations	#tri. comp.	#iterative	Time
1	80	13	$1.59 \cdot 10^1$	$1.45 \cdot 10^2$	$8.72 \cdot 10^{-2}$
2	80	13	$1.93 \cdot 10^1$	$1.27 \cdot 10^2$	$6.80 \cdot 10^{-2}$
4	80	18	$4.31 \cdot 10^1$	$1.78 \cdot 10^2$	$1.00 \cdot 10^{-1}$
8	80	14	$4.10 \cdot 10^1$	$9.82 \cdot 10^1$	$6.82 \cdot 10^{-2}$

To distinguish the impact of the particle sizes from the impact of the triangle, we next subdivide the upscaled colliding particle such that the triangle sizes of both particles remain almost invariant and comparable. Doubling the particle size thus corresponds to four times more triangles. Within the surrogate tree, we keep the number of triangles per leaf roughly constant.

Despite the increase in the number of triangles per upscaling, the number of comparisons and the runtime quickly plateau (Table 6.7). The additional cost due to the increase in the triangle count is bounded by a factor of four, even if we continue to increase the geometric detail beyond a factor of four. The cost per triangle reduces as one particle grows relative to the other.

As the size of the larger object grows, its surrogate tree becomes deeper. The algorithm unfolds the larger tree, but this unfolding is a localised process. We end up with roughly scale-invariant compute cost which is dominated by the number of triangles stored within one node of the finest tree level.

Observation 15 *The multiscale Picard approach is particularly effective if we compare objects of massively different size yet with the same level of detail. In this case, the cost per triangle comparison remains almost invariant.*

The observation is particular encouraging for simulations with objects of massively different size where the large objects exhibit a high level of detail. Such situations are found in fluid-structure interaction, where a collision induces a force on a bending

object. Such objects often are modelled via adaptive meshes which have to refine around the contact point.

6.2.7 Many particle experiments

Our multiscale Picard method shows an improvement in time-to-solution for individual particles which collide with other particles of different size or simple geometric object such as a plane. To be useful in real DEM codes, the algorithms must remain performant when we run simulations with many particles.

As long as the neighbourhood search is efficient, our achievements for individual particle-particle interactions carry over to dynamic simulations with large particle counts. If particles move around and bounce into each other, yet each particle only interacts with few other objects per time step, our multiscale concepts simply are to be generalised from two objects to a small object count. The situation changes if particles enter a stationary regime, i.e. if the particles are densely clustered and almost at rest.

We therefore finally simulate 24,576 particles that barely overlap and are initially at rest. This imitates a basin of granulates, e.g. Each particle is made up of 1,224 triangles, i.e. we handle a total memory footprint of approximately 3.87GiB which is substantially larger than the 30MiB total L3 cache.

Table 6.9: Measurements for 24,576 particles which are densely clustered yet almost at rest. The runtime is the runtime per time step per core. We present the total time and the time per particle-particle comparison.

	Comparison-based			Hybrid			
		Time				Time	
Method	#tri. comp.	total	per comp	#tri. comp.	#iterative	total	per comp
Single level	$6.50 \cdot 10^9$	$1.92 \cdot 10^6$	$9.67 \cdot 10^2$	$1.29 \cdot 10^8$	$6.50 \cdot 10^9$	$8.36 \cdot 10^5$	$4.20 \cdot 10^2$
Surrogate within Picard	$2.49 \cdot 10^6$	$4.70 \cdot 10^2$	$2.37 \cdot 10^{-1}$	$2.92 \cdot 10^2$	$2.49 \cdot 10^6$	$3.77 \cdot 10^2$	$1.90 \cdot 10^{-1}$
Multiscale Picard	$9.08 \cdot 10^5$	$2.30 \cdot 10^2$	$1.16 \cdot 10^{-1}$	$1.09 \cdot 10^2$	$1.06 \cdot 10^6$	$2.13 \cdot 10^2$	$1.07 \cdot 10^{-1}$

The hierarchical scheme yields a massive reduction of (iterative) triangle-triangle comparisions, and it notably succeeds to avoid too many comparisons which subsequently have to be postprocessed (Table 6.9). The runtime per particle-particle

comparison is significantly lower than in previous dynamic setups. Consequently, the cost per particle is lower, too, even though we have around six collision partners per particle.

Since the particles are almost at rest and only have tiny overlaps, the arising forces are very small. Consequently, few Picard iterations are sufficient to converge for the overall setup. The surrogate trees unfold only around the localised contacts, and this unfolding is very efficient, i.e. we barely remove triangles from the active set while we run the Picard iterations.

Observation 16 *The multiscale approach remains advantageous for larger particle assemblies where the particles are densely clustered.*

Wrap-up In this chapter we again saw repeated instances of some core ideas. In this chapter we presented a novel algorithm, which by breaking down meshes into a hierarchy of resolutions accelerates iterative contact resolution algorithms by using progressively higher resolutions to make early iterations significantly cheaper, when the current solution is potentially very far from the actual solution. Using approximate geometry in early iterations may push the solution too far from the correct solution and result in a failure to converge but since we have a mechanism for rolling back the overall cost is significantly reduced.

The same computational motif is applied to the actual triangle-triangle distance checks (Chapter 5) and results in a combined method that achieves an exceptionally high portion of peak performance on a small number of cores per interaction. Furthermore, when combined with our local time stepping algorithm (Chapter 4), which produces large numbers of candidate particle pairs, our method achieves an exceptionally high portion of peak performance across a whole node.

Algorithm 10 Implicit time stepping algorithm where the Picard and multiresolution loop are intermingled.

$\forall p_j \neq p_i \in \mathbb{P}:\ \mathbb{A}(p_i,p_j) \leftarrow root(\mathcal{T}(p_i))$ ▷ Active sets are now parameterised over interactions
while $\mathbb{T}^{\text{guess}}(p_i), v^{\text{guess}}(p_i), \omega^{\text{guess}}(p_i)$ or any $\mathbb{A}$ change significantly for any p_i **do**
  $\forall p_j \neq p_i \in \mathbb{P}:\ \mathbb{A}_{\text{new}}(p_i,p_j) \leftarrow \emptyset,\ \mathbb{A}_{\text{new}}(p_j,p_i) \leftarrow \emptyset,\ \mathbb{C}(p_i,p_j) = \emptyset,\ \mathbb{C}(p_j,p_i) = \emptyset$
  for $p_j \neq p_i \in \mathbb{P}$ **do**
    for $t_i \in \mathbb{A}(p_i,p_j), t_j \in \mathbb{A}(p_j,p_i)$ **do**
      $c \leftarrow$ contactIterative(t_i,t_j) ▷ Use context-specific ϵ depending on t_i, t_j
      if $c = \odot \wedge t_i \in \mathbb{T}_0^\epsilon(p_i) \wedge t_j \in \mathbb{T}_h^\epsilon(p_j)$ **then** ▷ Not converged on non-surrogate triangles
        $c \leftarrow$ contactComparisonBased(t_i,t_j) ▷ Use comparison-based algorithm this time
      end if
      if $c \neq \perp \wedge c \neq \odot$ **then** ▷ Implicit guess
        $\mathbb{C}(p_i) \leftarrow \mathbb{C}(p_i) \cup \{c\},\ \mathbb{C}(p_j) \leftarrow \mathbb{C}(p_j) \cup \{c\}$
      end if
      if $c = \perp$ **then** ▷ Add only parents
        $\mathbb{A}_{\text{new}}(p_i,p_j) \leftarrow \mathbb{A}_{\text{new}}(p_i,p_j) \cup \{\hat{t} : t_i \sqsubseteq_{\text{child}} \hat{t}\}$
        $\mathbb{A}_{\text{new}}(p_j,p_i) \leftarrow \mathbb{A}_{\text{new}}(p_j,p_i) \cup \{\hat{t} : t_j \sqsubseteq_{\text{child}} \hat{t}\}$
      else ▷ Widen active sets
        ... ▷ Compare to Algorithm 9
      end if
    end for
  end for
  $\forall p_j \neq p_i \in \mathbb{P}:\ \mathbb{A}(p_i,p_j) \leftarrow \mathbb{A}_{\text{new}}(p_i,p_j),\ \mathbb{A}(p_j,p_i) \leftarrow \mathbb{A}_{\text{new}}(p_j,p_i)$
  for $p_i \in \mathbb{P}$ **do**
    $(dv, d\omega) \leftarrow$ calcForces$(\mathbb{C}(p_i))$ ▷ Remove redundant contact points first
    $(v^{\text{guess}}, \omega^{\text{guess}})(p_i) \leftarrow (v,\omega)(p_i) + \Delta t \cdot (dv, d\omega)$ ▷ Additional damping might be required
    $\mathbb{T}^{\text{guess}}(p_i) \leftarrow$ update$(\mathbb{T}(p_i), v^{\text{guess}}(p_i), \omega^{\text{guess}}(p_i), \Delta t)$
  end for
  for $p_j \neq p_i \in \mathbb{P}$ **do** ▷ Clean-up, i.e. add siblings
    ▷ Obsolete if surrogate nodes host only one triangle
    $\forall t, \hat{t} \in \mathbb{A}(p_i,p_j)$ with $t \sqsubseteq_{\text{child}} \hat{t} : \mathbb{A}(p_i,p_j) \leftarrow \mathbb{A}(p_i,p_j) \cup \{t' \in \mathcal{T}(p_i) : t' \sqsubseteq_{\text{child}} \hat{t}\}$
    $\forall t, \hat{t} \in \mathbb{A}(p_j,p_i)$ with $t \sqsubseteq_{\text{child}} \hat{t} : \mathbb{A}(p_j,p_i) \leftarrow \mathbb{A}(p_j,p_i) \cup \{t' \in \mathcal{T}(p_j) : t' \sqsubseteq_{\text{child}} \hat{t}\}$
    ▷ and remove "redundant" parents
    $\forall t \in \mathbb{A}(p_i,p_j):\ \mathbb{A}(p_i,p_j) \leftarrow \mathbb{A}(p_i,p_j) \setminus \{\hat{t} \in \mathcal{T}(p_i) : t \sqsubseteq_{\text{child}} \hat{t}\}$
    $\forall t \in \mathbb{A}(p_j,p_i):\ \mathbb{A}(p_j,p_i) \leftarrow \mathbb{A}(p_j,p_i) \setminus \{\hat{t} \in \mathcal{T}(p_j) : t \sqsubseteq_{\text{child}} \hat{t}\}$
  end for
end while
$\forall p_i \in \mathbb{P}:\ \mathbb{T}(p_i) \leftarrow \mathbb{T}^{\text{guess}}(p_i),\ v(p_i) \leftarrow v^{\text{guess}}(p_i),\ \omega(p_i) \leftarrow \omega^{\text{guess}}(p_i)$
$t \leftarrow t + \Delta t$

CHAPTER 7

Conclusions

We present a family of novel local time stepping and implicit multi-resolution contact detection algorithms that reduce the required computations while increasing the vectorisation potential in areas where work is unavoidable. Our local time stepping scheme primarily achieves this by each particle maintaining its own states, time stamps and time step size and thereby avoiding excessive and unnecessary state updates. As far as we're aware, our 'anarchic' time stepping scheme is unique in DEM. Our implicit multi-resolution contact detection scheme reduces the required work by phrasing the object interaction phase as an iterative process, where coarse surrogate representations of geometry is used to approximate contacts. While this work borrows heavily from multiresolution concepts used by AMR solvers, we believe it's application to DEM is unique. Furthermore, contact detection between two meshes is phrased as a distance minimisation algorithm with high vector efficiency. Our contribution to this element is an incremental improvement to a method that is able to compute non-intersecting triangle-to-triangle distances by utilising an extremely high portion of peak performance, even when used in tandem with our multiresolution concept.

Our local time stepping algorithm focusses on reducing the required state up-

dates at each time step while creating as much parallel work with the remaining required updates. Our implicit multi-resolution contact detection algorithm focusses on reducing the time to solution primarily at the single node level.

Using a fixed time step size with explicit time stepping, while requiring interpenetration free steps for rigid bodies with tiny boundaries, would require every time step to be as small as the smallest required time step size. The size of this smallest allowable time step is hard to know ahead of run time. Global adaptive time stepping partially mitigates these challenges by selecting the most appropriate time step size on the fly. For individual particle pairs the step size can vary by multiple orders of magnitude. However, when large numbers of interactions exist then at any one time there is always likely to be an interaction that requires a close to zero time step size. Our algorithm enables local time stepping so excess updates to particles that can be advanced in time with large steps can be avoided. For large scale and highly dynamic scenarios local time stepping bring efficient adaptive time stepping within reach.

While avoiding unnecessary updates to state leads to significant performance improvements it also leads to a challenge in load balancing and utilising available CPU resources. We effectively employ work stealing task queues to balance this workload on a single node. However, scenarios where interactions form a dense network, therefore preventing independent selection of time step sizes, make creating parallel work, even on a single node, challenging.

In order to maintain large time step sizes while remaining stable implicit time stepping is required. This is made particularly important as we use a thin boundary and enforce non-intersection. Iterative contact detection and resolution algorithms are expensive. Therefore, we phrase the problem in multi-resolution language where inexpensive approximate geometry provides good initial guesses and is then followed by more expensive, but more accurate, steps. Loops permutation in iterative algorithms becomes an effective tool for accelerating these algorithms. The traversal of triangle pairs with Newton iterations to minimise distance is flipped to enable aggressive vectorisation. The Picard iterations of implicit contact detection and resolution and the unrolling of surrogate representations of geometry are reversed

such that expensive computations are reserved only for when the required accuracy is high.

Limitations Our 'anarchic' local time stepping scheme can be very effective for highly dynamic (especially when also sparse) scenarios but is severely limited for static or dense scenarios. Since the whole domain of particles is divided up into clusters and each cluster has a time step size selected, if there are very few clusters able to be separated from one another or the range of required time steps is very low then the benefits of our local time stepping are mitigated. For example, for packing experiments, where every particle is almost stationary and interacting with many neighbours, our scheme would be no better than a global adaptive time stepping scheme but would suffer from potentially significant computational and memory overheads due to tracking the extra particle states required for rollbacks.

When considering individual particle-particle interactions, our multiresolution method is highly effective at reducing the time to solution when there are a large number of triangles per particle. However, when there are only particles made up of small numbers of triangles there is little benefit to the multiresolution method. For our inverted loop algorithm to be effective several surrogate layers are required. Opposed to this requirement is the need to compute a sufficiently high number of triangle-to-triangle distance checks in parallel. This leads to us storing batches of triangles together and having a high branching factor in our surrogate trees, which all leads to shallow trees. As our method is currently implemented, only batches of triangles from a single particle-particle interaction are batched together for distance checks. Therefore, the batches have to be sufficiently large (by using complex particles with a high surrogate tree branching factor) to occupy all the vector lanes of a CPU core.

Furthermore, constructing the surrogate trees adds an additional setup stage to any simulation and traversing the surrogate hierarchy with our multiresolution algorithm adds additional code complexity to an otherwise simple algorithm.

Future study While our contributions show the potential for increased vector efficiency there remains opportunities to refine the core methods that form the building

blocks of our algorithms. Highlighted first are two specific areas where future work could refine our implementation. Next we briefly discuss the challenges and potential in making algorithmic adjustments to our method before finally suggesting four larger specific areas where our algorithmic ideas could be applied. First, each time we advance in time we discard all information about contacts in previous time steps. The effectiveness of an iterative algorithm can be greatly improved by a good initial estimate of a solution (perhaps called a preconditioner). Particularly during the span of a contact the state of particles changes only a small amount between time steps. Therefore, exact contacts computed in the previous iteration are likely to be good estimates for the current iteration. This property could be exploited at two different levels.

1. Instead of always starting the exploration of surrogate trees at the root nodes we could instead start part way down the tree, focussing the search for medium resolution contacts to local areas that produced contacts previously.

2. Our iterative triangle-triangle distance algorithm initialises potential closest points to the centre of the triangles. Instead the previous contact position could be used. The penalty terms would quickly restrict this point back onto the triangles area while providing a starting point that is likely to be closer to the solution.

Second, the idea of different precisions is baked into our algorithm in the form of the surrogate hierarchies. Using reduced precision representations for geometry and particle state is a potentially attractive way of reducing memory usage and therefore maximising cache utilisation. Furthermore, the presence of massively parallel low precision accelerators, in the form of GPUs, in may systems raises the potential for further work on batch processing elements of our algorithm. As it is currently implemented triangle-triangle distance checks are only batched together in the scope of a single particle-particle interaction. This provides a sufficient number of triangle pairs to be effectively vectorised on the CPU. However, if these triangle pairs could be collected from across all interacting particles in might become beneficial to offload these large batches of optimisation problems to a GPU. Furthermore, many aspects

of our algorithm could benefit from being entirely executed on the GPU if sufficient scale is present. For example, continuous collision detection, iteratively computing the interactions between particles and updating the state of all particles based on accumulated forces are all examples of parallel process in our algorithm that, with sufficient scale, may be candidates for GPU acceleration with lower precisions. Finally, we observe that generally increasing the number of parallel tasks created increases the tasking runtimes ability to load balance but for sparse scenarios (many individual pairs of interacting particles, for example) the cost of each unit of work is small compared to the task runtime overhead for so many tasks. In such a case it may be beneficial to artificially rejoin clusters together even though they do not interact to create larger units of work to decrease the tasking overhead.

At a broader level, the contributions of our work remain largely algorithmic in nature and, while offering not just incremental but algorithmic potential improvements, have remaining challenges for adopting in massively parallel real-world situations. Primarily, and ever present throughout our work, is the need to create sufficient computational work to keep available resources occupied. The requirement for scientific codes to perform well across large clusters significantly increases this challenge. Second, the physical model used in this work is comparatively simple. While we see no reason these method should be any more complex to implement for a real physics code an example application using our methods along with 'real' physics would verify this. Finally, we don't provide guarantees on the number of iterations required for convergence for any of our iterative algorithms. While in practice we don't encounter problems with convergence behaviour, it would be beneficial to better understand the relationship between the accuracy of the used surrogate model and the rate of convergence. With such knowledge we could improve the behaviour of surrogate unrolling to involve skipping early layers if they are unlikely to improve the intial guess sufficiently, skipping of surrogate layers where the added accuracy isn't worth additional Picard iterations and early stopping where reaching the fine resolution mesh isn't required for a given target accuracy.

Adding a single stored past state enabled us to implement 'anarchic' time stepping and massively reduce the required computation in some cases. Storing multi-

ple past time states per particle may further increase this effect. In this case roll backs can potentially undo several time steps worth of work for a group of particles. However, if the cost of this is significantly outweighed by the increased resource utilisation enabled by always having a large number of particles that are able to have a new state computed then the cost of rare roll backs might be worth the increased memory footprint. Furthermore, if by inspecting properties such as the total energy (kinetic and potential) we are able to reason about the maximum possible velocities of particles in the near future then it might be possible to further isolate clusters from each other and allow updated states to be computed where there is no possibility of clusters interacting within that time span (even if this removes the ability to roll back to a common time stamp between these clusters).

With an even wider view of the DEM and scientific computing landscape we consider the following ideas and how our algorithmic ideas to be applied more widely:

- **Grid based semi-'anarchic' local time stepping** An alternative option for local time stepping is to equip the grid in which particles are embedded with a time step size per cell. Currently our method is limited to dynamic and sparse scenarios but changing to a scheme where cells on a grid are used for determining time step size rather than cluster membership may be an option for bringing our 'anarchic' time stepping concepts to densely packed scenarios. A structure would need to be imposed such that the difference in time step size between neighbouring cells is limited and our iterative implicit solver would need adjusting to accommodate for particles with different time step sizes interacting together. Combining some ideas from our 'anarchic' time step sizes with this structured step size scheme would enable us to handle more realistic scenarios. For example, a pile of sand with a flow of new grains falling on to one side of the pile. Our method allows the grains in free fall to step independently and with large step sizes, unless bumping another falling particle. However, once part of the large pile the whole thing must step with tiny step sizes because of the continuous stream of new grains impacting the pile. With time step sizes also computed spatially within a single cluster we could assign a large step size to the far side of the pile (in a static state except

the occasional shift when the weight of a large number of new grains causes the pile to shift), a medium step size to grains at the bottom of the pile (which don't move much but must remain stable under higher pressures) and tiny step sizes directly around the areas of impact. These particle-local (as opposed to cluster-local) time step sizes could be applied using a grid data structure as in other works [65, 66] or, sticking closer to our paradigm, could be 'diffused' through our interaction graph (ie. particles requiring a small time step as well as those up to N hops away take additional sub steps while those further than N interactions away take fewer sub steps).

- **Local time stepping without global loop** Currently our algorithm still follows the pattern of having a global time step loop. However, each particle is updating independently of any globally defined limit. Therefore, we should continuously add cluster steps to a pool of tasks rather than proceeding in rounds. The eligibility of a cluster to advance in time would be implicitly managed by task dependencies. The merging and splitting of clusters as well as selection of new time step sizes should all also be handled locally by tasks that depend on the results of the local clusters. Successfully implemented, this would help to mitigate the effects of bottle neck clusters (the ones almost every other cluster is waiting to advance for) by continuously adding tasks to a work stealing pool as soon as they are available (rather than waiting until the next global loop) new work is always being made available to idle cores. Furthermore, when considering a distributed compute environment this method could reduce the synchronisation between nodes and instead synchronisation is implicitly defined by dependencies that cross a boundary between the spatial regions 'owned' by different ranks opening up the possibility of significantly larger simulations..

- **Multi-resolution sequential impulses** While the Sequential Impulses algorithm remains a favourite for entertainment applications, such as video games and movie visual effects, the worlds of offline scientific simulation and real time interactive simulations continue to collide in new ways. It is therefore

important to consider the case where effectively arbitrary complexity geometry is used for simulations that are required to run in or close to real time. In the field of rendering recent developments have allowed this kind of nearly unlimited geometric complexity by a similar mechanism of surrogates, which are selected to give a result that is 'good enough' [161]. We have demonstrated the benefits of using multi-resolution surrogate representations when it comes to physics simulations and we believe there is further potential here when considering real time applications. Like with the rendering example, having a multi-resolution method would allow the simulation to dynamically stop when a 'good enough' solution is reached. Perhaps defined by regions or objects of interest or by proximity to a user agent. The stopping condition could also be defined by a time budget, required in a strict environment where frame rates must be maintained.

- **Fluid-structure interaction** Simulations involving fluid structure interaction represent some of the most cutting edge research currently happening. In a simulation where rigid body objects are suspended in or otherwise interacting with fluid the density of the simulation mesh for the fluid must be significantly greater than the resolution of the mesh. Therefore, the cost of particle-particle interactions is likely to be small in comparison with the cost of solving the fluid-particle (fluid-structure) interaction. Where the fluid simulation is being computed using a multi-resolution technique it seems likely that some speed improvement could be obtained by using multi-resolution geometry. Whenever the CFD computation called for a distance-to-nearest-surface check, for example, a surrogate model could be chosen to give a level of detail proportional to the fluid grid size at that level. As the results from that level are projected down and the solution further refined then distance look-ups could be further refined to finer and finer surrogate mesh resolutions.

Bibliography

[1] P. A. Cundall and O. D. L. Strack, "A discrete numerical model for granular assemblies," *Géotechnique*, vol. 29, no. 1, pp. 47–65, 1979. 1, 2

[2] F. Alonso-Marroquín and Y. Wang, "An efficient algorithm for granular dynamics simulations with complex-shaped objects," *Granular Matter*, vol. 11, pp. 317–329, 2009. 1, 1, 3.2

[3] K. Iglberger and U. Rüde, "Massively parallel granular flow simulations with non-spherical particles," *Computer Science - Research and Development*, vol. 25, no. 1-2, pp. 105–113, 2010. 1, 1

[4] K. Krestenitis and T. Koziara, "Calculating the minimum distance between two triangles on simd hardware," 4 2015. 1, 4, 1, 5.1.2, 5.1.2, 6.2.3

[5] K. Krestenitis, T. Weinzierl, and T. Koziara, "Fast dem collision checks on multicore nodes.," in *Parallel processing and applied mathematics : 12th International conference, PPAM 2017, Lublin, Poland, September 10-13; revised selected papers. Part 1.* (R. Wyrzykowski, J. J. Dongarra, E. Deelman, and K. Karczewski, eds.), no. 10777 in Lecture Notes in Computer Science, pp. 123–132, 2018. 1, 1, 1, 4, 4.4, 5.1.2, 6.2, 6.2.3

[6] T. Y. Li and J. S. Chen, "Incremental 3D collision detection with hierarchical data structures," *Proceedings of the ACM symposium on Virtual reality software and technology 1998 - VRST '98*, vol. 1998, pp. 139–144, 1998. 1, 1

[7] 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*, vol. 302, pp. 108–152, 2016. 1

[8] T. Weinhart, C. Labra, S. Luding, and J. Y. Ooi, "Influence of coarse-graining parameters on the analysis of dem simulations of silo flow," *Powder Technology*, vol. 293, pp. 138–148, 2016. 1

[9] F. Alonso-Marroquin and Y. Wang, "An efficient algorithm for granular dynamics simulation with complex-shaped objects," 2008. 1, 1, 3.2

[10] 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*, vol. 224, p. 374–389, 07 2012. 1, 1, 3.2

[11] S. Zhao and J. Zhao, "A poly-superellipsoid-based approach on particle morphology for dem modeling of granular media," *International Journal for Numerical and Analytical Methods in Geomechanics*, 2019. 1, 1, 3.2

[12] H. Kruggel-Emden, S. Rickelt, S. Wirtz, and V. Scherer, "A study on the validity of the multi-sphere discrete element method," *Powder Technology*, vol. 188, no. 2, pp. 153–165, 2008. 1

[13] E. Rougier, A. Munjiza, and J.-P. Latham, "Shape selection menu for grand scale discontinua systems," *Engineering Computations*, vol. 21, pp. 343–359, 03 2004. 1, 3.1

[14] 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*, vol. 26, no. 2, pp. 89–111, 2002. 1

[15] H. Zhu, Z. Zhou, R. Yang, and A. Yu, "Discrete particle simulation of particulate systems: A review of major applications and findings," *Chemical engineering science*, vol. 63, no. 23, pp. 5728–5770, 2008. 1

[16] Z. Zhang, L. Liu, Y. Yuan, and A. Yu, "A simulation study of the effects of dynamic variables on the packing of spheres," *Powder Technology*, vol. 116, no. 1, pp. 23–32, 2001. 1

[17] J.-P. Latham, Y. Lu, and A. Munjiza, "A random method for simulating loose packs of angular particles using tetrahedra," *Geotechnique*, vol. 51, pp. 871–879, 01 2001. 1

[18] X. An, R. Yang, K. Dong, R. Zou, and A. Yu, "Micromechanical simulation and analysis of one-dimensional vibratory sphere packing," *Physical review letters*, vol. 95, p. 205502, 12 2005. 1

[19] J. Lee and H. J. Herrmann, "Angle of repose and angle of marginal stability: molecular dynamics of granular particles," *Journal of Physics A Mathematical General*, vol. 26, pp. 373–383, Jan. 1993. 1

[20] Y. Li, Y. Xu, and C. Thornton, "A comparison of discrete element simulations and experiments for 'sandpiles' composed of spherical particles," *Powder Technology*, vol. 160, no. 3, pp. 219–228, 2005. 1

[21] L. Smith and U. Tüzün, "Stress, voidage and velocity coupling in an avalanching granular heap," *Chemical Engineering Science*, vol. 57, no. 18, pp. 3795–3807, 2002. 1

[22] H. Matuttis, S. Luding, and H. Herrmann, "Discrete element simulations of dense packings and heaps made of spherical and non-spherical particles," *Powder Technology*, vol. 109, no. 1, pp. 278–292, 2000. 1

[23] R. P. Behringer, "Jamming in granular materials," *Comptes Rendus Physique*, vol. 16, no. 1, pp. 10–25, 2015. Granular physics / Physique des milieux granulaires. 1

[24] M. Kong and J. Lannutti, "Effect of agglomerate size distribution on loose packing fraction," *Journal of the American Ceramic Society*, vol. 83, pp. 2183 – 2188, 12 2004. 1

[25] G. Fu and W. Dekelbab, "3-d random packing of polydisperse particles and concrete aggregate grading," *Powder Technology*, vol. 133, no. 1, pp. 147–155, 2003. 1

[26] A. D. Rosato and D. Yacoub, "Microstructure evolution in compacted granular beds," *Powder Technology*, vol. 109, no. 1, pp. 255–261, 2000. 1

[27] J. Duran and R. Behringer, "Sands, powders, and grains: An introduction to the physics of granular materials," *Physics Today - PHYS TODAY*, vol. 54, pp. 63–64, 04 2001. 1

[28] K. Z. Y. Yen and T. K. Chaki, "A dynamic simulation of particle rearrangement in powder packings with realistic interactions," *Journal of Applied Physics*, vol. 71, pp. 3164–3173, 04 1992. 1

[29] R. Y. Yang, R. P. Zou, and A. B. Yu, "Computer simulation of the packing of fine particles," *Phys. Rev. E*, vol. 62, pp. 3900–3908, Sep 2000. 1

[30] R. Nedderman and C. Laohakul, "The thickness of the shear zone of flowing granular materials," *Powder Technology*, vol. 25, no. 1, pp. 91–100, 1980. 1

[31] J. Härtl and J. Ooi, "Experiments and simulations of direct shear tests: Porosity, contact friction and bulk friction," *Granular Matter*, vol. 10, pp. 263–271, 06 2008. 1

[32] C. David, R. Garcia Rojo, H. Herrmann, and S. Luding, "Powder flow testing with 2d and 3d biaxial and triaxial simulations," *Particle Particle Systems Characterization*, vol. 24, pp. 29 – 33, 06 2007. 1

[33] R. Brewster, G. Grest, J. Landry, and A. Levine, "Plug flow and the breakdown of bagnold scaling in cohesive granular flows," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 72, p. 061301, 01 2006. 1

[34] K. Saitoh and H. Hayakawa, "Rheology of a granular gas under a plane shear," *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 75, p. 021302, 03 2007. 1

[35] L. Silbert, J. Landry, and G. Grest, "Granular flow down a rough inclined plane: Transition between thin and thick piles," *Physics of Fluids*, vol. 15, 06 2002. 1

[36] C. Zeilstra, J. Collignon, M. van der Hoef, N. Deen, and J. Kuipers, "Experimental and numerical study of wall-induced granular convection," *Powder Technology*, vol. 184, no. 2, pp. 166–176, 2008. DISCRETE ELEMENT MODELLING OF FLUIDISED BEDS (Special Issue: Dedicated to Professor Yutaka Tsuji). 1

[37] A. D. Rosato, Y. Lan, and M. Richman, "On the calculation of self-diffusion in vertically agitated granular beds," *Powder Technology*, vol. 182, no. 2, pp. 228–231, 2008. Granular Temperature. 1

[38] X. Fang and J. Tang, "A numerical study of the segregation phenomenon in granular motion," *Journal of Vibration and Control*, vol. 13, pp. 711–729, 05 2007. 1

[39] D. R. Parisi, S. Masson, and J. Martinez, "Partitioned distinct element method simulation of granular flow within industrial silos," *Journal of Engineering Mechanics*, vol. 130, no. 7, pp. 771–779, 2004. 1

[40] F. Fraige and P. Langston, "Integration schemes and damping algorithms in distinct element models," *Advanced Powder Technology*, vol. 15, no. 2, pp. 227–245, 2004. 1

[41] S. P. D. Amlan Datta, B. K. Mishra and A. Sahu, "A dem analysis of flow characteristics of noncohesive particles in hopper," *Materials and Manufacturing Processes*, vol. 23, no. 2, pp. 195–202, 2008. 1

[42] W. R. Ketterhagen, J. S. Curtis, C. R. Wassgren, A. Kong, P. J. Narayan, and B. C. Hancock, "Granular segregation in discharging cylindrical hoppers: A discrete element and experimental study," *Chemical Engineering Science*, vol. 62, no. 22, pp. 6423–6439, 2007. 1

[43] L. F. Orozco, D.-H. Nguyen, J.-Y. Delenne, P. Sornay, and F. Radjai, "Discrete-element simulations of comminution in rotating drums: Effects of grinding media," *Powder Technology*, vol. 362, pp. 157–167, 2020. 1, 3.1

[44] C. Xie, H. Ma, and Y. Zhao, "Investigation of modeling non-spherical particles by using spherical discrete element model with rolling friction," *Engineering Analysis with Boundary Elements*, vol. 105, pp. 207–220, 2019. 1, 3.1, 3.4

[45] V. Buchholtz, T. Pöschel, and H.-J. Tillemans, "Simulation of rotating drum experiments using non-circular particles," *Physica A: Statistical Mechanics and its Applications*, vol. 216, no. 3, pp. 199–212, 1995. 1

[46] M. Sakai, K. Shibata, and S. Koshizuka, "Development of a criticality evaluation method involving the granular flow of the nuclear fuel in a rotating drum," *Nuclear Science and Engineering*, vol. 154, pp. 63 – 73, 2006. 1

[47] "Chapter 14 - non-linear soil models and numerical solutions of problems," in *Rheological Fundamentals of Soil Mechanics* (S. S. VYALOV, ed.), vol. 36 of *Developments in Geotechnical Engineering*, pp. 481–556, Elsevier, 1986. 1

[48] Y. P. Cheng, Y. Nakata, and M. Bolton, "Discrete element simulation of crushable soil," *Cheng, Y.P. and Nakata, Y. and Bolton, M.D. (2003) Discrete element simulation of crushable soil. Géotechnique, 53 (7). pp. 633-641. ISSN 00168505*, vol. 53, 01 2003. 1

[49] X. Zhang and L. Vu-Quoc, "Simulation of chute flow of soybeans using an improved tangential force–displacement model," *Mechanics of Materials*, vol. 32, pp. 115–129, 02 2000. 1

[50] M. Saeki, "Analytical study of multi-particle damping," *Journal of Sound and Vibration*, vol. 281, no. 3, pp. 1133–1144, 2005. 1

[51] T. J. Goda and F. Ebert, "Three-dimensional discrete element simulations in hoppers and silos," *Powder Technology*, vol. 158, no. 1, pp. 58–68, 2005. Prof. Dr.-Ing. Otto Molerus 70th birthday. 1

[52] M. Sakai, K. Shibata, and S. Koshizuka, "Development of a criticality evaluation method involving the granular flow of the nuclear fuel in a rotating drum," *Nuclear Science and Engineering*, vol. 154, pp. 63 – 73, 2006. 1

[53] P. J. Noble and T. Weinzierl, "A multiresolution discrete element method for triangulated objects with implicit time stepping," *SIAM Journal on Scientific Computing*, vol. 44, no. 4, pp. A2121–A2149, 2022. 1, 4.4, 4.4

[54] 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*, vol. 324, pp. 18–35, 2018. 1

[55] Z. Ferguson, M. Li, T. Schneider, F. Gil-Ureta, T. Langlois, C. Jiang, D. Zorin, D. M. Kaufman, and D. Panozzo, "Intersection-free rigid body dynamics," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 40, no. 4, 2021. 1, 1

[56] B. Wang, Z. Ferguson, T. Schneider, X. Jiang, M. Attene, and D. Panozzo, "A large-scale benchmark and an inclusion-based algorithm for continuous collision detection," *ACM Trans. Graph.*, vol. 40, sep 2021. 1, 2, 4.4, 5.2

[57] M. Li, D. M. Kaufman, and C. Jiang, "Codimensional incremental potential contact," *ACM Trans. Graph. (SIGGRAPH)*, vol. 40, no. 4, 2021. 1, 1

[58] L. Lan, D. M. Kaufman, M. Li, C. Jiang, and Y. Yang, "Affine body dynamics: Fast, stable and intersection-free simulation of stiff materials," *ACM Trans. Graph.*, vol. 41, jul 2022. 1

[59] T. Weinhart, L. Orefice, M. Post, M. P. van Schrojenstein Lantman, I. F. Denissen, D. R. Tunuguntla, J. Tsang, H. Cheng, M. Y. Shaheen, H. Shi, P. Rapino, E. Grannonio, N. Losacco, J. Barbosa, L. Jing, J. E. Alvarez Naranjo, S. Roy, W. K. den Otter, and A. R. Thornton, "Fast, flexible particle simulations — an introduction to mercurydpm," *Computer Physics Communications*, vol. 249, p. 107129, 2020. 1

[60] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*. Oxford University Press, 06 2017. 1

[61] B. D. Lubachevsky, "How to simulate billiards and similar systems," *Journal of Computational Physics*, vol. 94, pp. 255–283, jun 1991. 1

[62] S. Miller and S. Luding, "Event-driven molecular dynamics in parallel," *Journal of Computational Physics*, vol. 193, pp. 306–316, 01 2004. 1

[63] E. Shellshear and R. Ytterlid, "Fast distance queries for triangles, lines, and points using sse instructions," *Journal of Computer Graphic Techniques*, vol. 3, p. 86–110, 12 2014. 1, 5.1.4

[64] L. Lan, D. M. Kaufman, M. Li, C. Jiang, and Y. Yang, "Affine body dynamics: Fast, stable and intersection-free simulation of stiff materials," *ACM Trans. Graph.*, vol. 41, jul 2022. 1, 2

[65] L. He, X. Ban, X. Liu, and X. Wang, "Individual Time Stepping for SPH Fluids," in *EG 2015 - Short Papers* (B. Bickel and T. Ritschel, eds.), The Eurographics Association, 2015. 1, 2, 2, 4, 7

[66] H. Sitaraman and R. Grout, "An adaptive timestepping methodology for particle advance in coupled cfd-dem simulations," 02 2018. 1, 2, 2, 4, 7

[67] G. Barequet, B. Chazelle, L. J. Guibas, J. S. Mitchell, and A. Tal, "Boxtree: A hierarchical representation for surfaces in 3d," *Computer Graphics Forum*, vol. 15, no. 3, pp. 387–396, 1996. 1

[68] H. Dammertz, J. Hanika, and A. Keller, "Shallow bounding volume hierarchies for fast simd ray tracing of incoherent rays," *Comput. Graph. Forum*, vol. 27, pp. 1225–1233, 06 2008. 1

[69] C. Eisenacher, G. Nichols, A. Selle, and B. Burley, "Sorted deferred shading for production path tracing," *Computer Graphics Forum*, vol. 32, 07 2013. 1

[70] S. Gottschalk, M. Lin, and D. Manocha, "Obbtree: A hierarchical structure for rapid interference detection," *Computer Graphics*, vol. 30, 10 1997. 1

[71] M. Held, J. Klosowski, and J. Mitchell, "Real-time collision detection for motion simulation within complex environments," in *SIGGRAPH '96*, 1996. 1

[72] T. Akenine-Mller, E. Haines, and N. Hoffman, *Real-Time Rendering, Fourth Edition*. USA: A. K. Peters, Ltd., 4th ed., 2018. 2

[73] H. Sitaraman and R. Grout, "An adaptive timestepping methodology for particle advance in coupled cfd-dem simulations," 02 2018. 1, 2, 4

[74] K. Krestenitis, T. Weinzierl, and T. Koziara, *Fast DEM Collision Checks on Multicore Nodes*, pp. 123–132. 03 2018. 1

[75] K. Krestenitis and T. Weinzierl, "A multi-core ready discrete element method with triangles using dynamically adaptive multiscale grids," *Concurr Comput*, vol. 31, p. e4935, Aug. 2018. 1

[76] "Delta repository." `https://github.com/KonstantinosKr/delta`. Accessed: 2024-10-20. 1

[77] D. Meagher, "Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer," 10 1980. 2

[78] C. Ericson, *Real-Time Collision Detection*. 2, 3.1, 5.1, 5.1.1, 5.3

[79] G. van den Bergen, "Efficient collision detection of complex deformable models using aabb trees," *J. Graph. Tools*, vol. 2, p. 1–13, jan 1998. 2

[80] S. Woop, C. Benthin, I. Wald, G. S. Johnson, and E. Tabellion, "Exploiting Local Orientation Similarity for Efficient Ray Traversal of Hair and Fur," in *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics* (I. Wald and J. Ragan-Kelley, eds.), The Eurographics Association, 2014. 2

[81] A. Donev, S. Torquato, and F. H. Stillinger, "Neighbor list collision-driven molecular dynamics simulation for nonspherical hard particles. i. algorithmic details," *Journal of Computational Physics*, vol. 202, no. 2, pp. 737–764, 2005. 2

[82] B. Muth, M.-K. Müller, P. Eberhard, and S. Luding, "Collision detection and administration methods for many particles with different sizes," *Particle & Particle Systems Characterization*, 01 2007. 2

[83] D. Peng, S. J. Burns, and K. J. Hanley, "Critical time step for discrete element method simulations of convex particles with central symmetry," *International Journal for Numerical Methods in Engineering*, vol. 122, no. 4, pp. 919–933, 2021. 2

[84] S. Redon, A. Kheddar, and S. Coquillart, "Fast continuous collision detection between rigid bodies," *Computer Graphics Forum*, vol. 21, 05 2002. 2

[85] D. E. Stewart, "Rigid-body dynamics with friction and impact," *SIAM Review*, vol. 42, no. 1, pp. 3–39, 2000. 2

[86] F. Tonon, "Explicit exact formulas for the 3-d tetrahedron inertia tensor in terms of its vertex coordinates," *Journal of Mathematics and Statistics*, vol. 1, pp. 8–11, Mar 2005. 2.1.1

[87] E. Gilbert, D. Johnson, and S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988. 2.1.2

[88] Erin Catto, "Fast and simple physics using sequential impulses." URL: `https://box2d.org/files/ErinCatto_SequentialImpulses_GDC2006.pdf`, 2006. 2.1.3

[89] Raphael Priatama, "Sequential impulses explained from the perspective of a game physics beginner." URL: https://raphaelpriatama.medium.com/sequential-impulses-explained-from-the-perspective-of-a-game-physics-beginner-72a37f6fea05, 2020. 2.1.3

[90] *Inception*. Warner Bros., 2010. 2.1.3

[91] Erwin Coumans, "Opencl game physics: Bullet: A case study in optimizing physics middleware for the gpu." URL: https://www.nvidia.com/content/gtc/documents/1077_gtc09.pdf, 2009. 2.1.4, 4.6.4

[92] D. J. A. Welsh and M. B. Powell, "An upper bound for the chromatic number of a graph and its application to timetabling problems," *The Computer Journal*, vol. 10, pp. 85–86, 01 1967. 2.1.4

[93] Y. H. Xie, Z. X. Yang, D. Barreto, and M. D. Jiang, "The influence of particle geometry and the intermediate stress ratio on the shear behavior of granular materials," *Granular Matter*, vol. 19, apr 2017. 3.1

[94] Y. Li, M. Otsubo, R. Kuwano, S. Nadimi, and V. Angelidakis, "Dem modelling of granular materials with real shape morphology," vol. 73, pp. 355–358, 12 2021. 3.1

[95] J. Ai, J.-F. Chen, J. M. Rotter, and J. Y. Ooi, "Assessment of rolling resistance models in discrete element simulations," *Powder Technology*, vol. 206, no. 3, pp. 269–282, 2011. 3.1

[96] Barr, "Superquadrics and angle-preserving transformations," *IEEE Computer Graphics and Applications*, vol. 1, no. 1, pp. 11–23, 1981. 3.1

[97] J. R. WILLIAMS and A. P. PENTLAND, "Superquadrics and modal dynamics for discrete elements in interactive design," *Engineering computations*, vol. 9, no. 2, pp. 115–127, 1992. 3.1

[98] P. W. Cleary, "Large scale industrial dem modelling," *Engineering computations*, vol. 21, no. 2/3/4, pp. 169–204, 2004. 3.1

[99] P. Cleary, N. Stokes, and J. Hurley, "Efficient collision detection for three dimensional super-ellipsoidal particles," 01 1997. 3.1

[100] A. Podlozhnyuk, S. Pirker, and C. Kloss, "Efficient implementation of superquadric particles in discrete element method within an open-source framework," *Computational Particle Mechanics*, vol. 4, pp. 101 – 118, 2016. 3.1

[101] R. Capozza and K. Hanley, "A hierarchical, spherical harmonic-based approach to simulate abradable, irregularly shaped particles in dem," *Powder Technology*, vol. 378, pp. 528–537, 2021. 3.1

[102] M. Imaran, J. Young, R. Capozza, K. Stratford, and K. J. Hanley, "Spherical harmonic–based dem in lammps: Implementation, verification and performance assessment," *Computer Physics Communications*, vol. 304, p. 109290, 2024. 3.1

[103] M. Imaran, J. Young, R. Capozza, K. Stratford, and K. J. Hanley, "Spherical harmonic–based dem in lammps: Implementation, verification and performance assessment," *Computer Physics Communications*, vol. 304, p. 109290, 2024. 3.1

[104] J. Gomes and O. Faugeras, "Reconciling distance functions and level sets," *Journal of Visual Communication and Image Representation*, vol. 11, no. 2, pp. 209–223, 2000. 3.1

[105] G. Houlsby, "Potential particles: a method for modelling non-circular particles in dem," *Computers and Geotechnics*, vol. 36, no. 6, pp. 953–959, 2009. 3.1

[106] Z. Lai, Y. Feng, J. Zhao, and L. Huang, "Unifying the contact in signed distance field-based and conventional discrete element methods," *Computers and Geotechnics*, vol. 176, p. 106764, 2024. 3.1

[107] Hart, John C, "Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces," *The Visual Computer*, vol. 12, pp. 527–545, 1996. 3.1

[108] I. Vlahinić, E. Andò, G. Viggiani, and J. Andrade, "Towards a more accurate characterization of granular media: Extracting quantitative descriptors from tomographic images," *Granular Matter*, vol. 16, 02 2014. 3.1

[109] M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and Z. Corse, "Local optimization for robust signed distance field collision," *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 3, may 2020. 3.1

[110] D. Koschier, C. Deul, and J. Bender, "Hierarchical hp-adaptive signed distance fields," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '16, (Goslar, DEU), p. 189–198, Eurographics Association, 2016. 3.1

[111] R. Kawamoto, E. Andò, G. Viggiani, and J. E. Andrade, "Level set discrete element method for three-dimensional computations with triaxial case study," *Journal of the Mechanics and Physics of Solids*, vol. 91, pp. 1–13, 2016. 3.1

[112] M. Macklin, K. Erleben, M. Müller, N. Chentanez, S. Jeschke, and Z. Corse, "Local optimization for robust signed distance field collision," *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 3, may 2020. 3.1

[113] H. Xu and J. Barbič, "Continuous collision detection between points and signed distance fields," *Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS 2014*, 2014. 3.1

[114] M. Tolomeo and G. R. McDowell, "Modelling real particle shape in dem: a comparison of two methods with application to railway ballast," *International Journal of Rock Mechanics and Mining Sciences*, vol. 159, p. 105221, 2022. 3.1

[115] D. Höhner, S. Wirtz, and V. Scherer, "A numerical study on the influence of particle shape on hopper discharge within the polyhedral and multi-sphere discrete element method," *Powder Technology*, vol. 226, pp. 16–28, 2012. 3.1

[116] H. Kruggel-Emden, S. Rickelt, S. Wirtz, and V. Scherer, "A study on the validity of the multi-sphere discrete element method," *Powder Technology*, vol. 188, no. 2, pp. 153–165, 2008. 3.1

[117] J. Favier, M. Abbaspour-Fard, M. Kremmer, and A. Raji, "Shape representation of axi-symmetrical, non-spherical particles in discrete element simulation using multi-element model particles," *Engineering Computations*, vol. 16, pp. 467–480, 06 1999. 3.1

[118] T. Matsushima and H. Saomoto, "Discrete element modeling for irregularly-shaped sand grains la modÉlisation aux ÉlÉments discrets d'un sable a grains de forme irrÉguliÈre," *NUMGE2002: Numerical Methods in Geotechnical Engineering*, 01 2002. 3.1

[119] M. Price, V. Murariu, and G. Morrison, "Sphere clump generation and trajectory comparison for real particles," 2007. 3.1

[120] R. Taghavi, "Automatic clump generation based on mid-surface," 02 2011. 3.1

[121] V. Angelidakis, S. Nadimi, M. Otsubo, and S. Utili, "Clump: A code library to generate universal multi-sphere particles," *SoftwareX*, vol. 15, p. 100735, 2021. 3.1

[122] M. Kodam, R. Bharadwaj, J. Curtis, B. Hancock, and C. Wassgren, "Force model considerations for glued-sphere discrete element method simulations," *Chemical Engineering Science*, vol. 64, no. 15, pp. 3466–3475, 2009. 3.1

[123] D. Markauskas, R. Kačianauskas, A. Džiugys, and R. Navakas, "Investigation of adequacy of multi-sphere approximation of elliptical particles for dem simulations," *Granular Matter*, vol. 12, pp. 107–123, Feb 2010. 3.1

[124] M. Marigo and E. Stitt, "Discrete element method (dem) for industrial applications: Comments on calibration and validation for the modelling of cylindrical pellets," *KONA Powder and Particle Journal*, vol. 32, pp. 236–252, 02 2015. 3.1

[125] O. Ben-Nun, I. Einav, and A. Tordesillas, "Force attractor in confined comminution of granular materials," *Phys. Rev. Lett.*, vol. 104, p. 108001, Mar 2010. 3.1

[126] L. Elghezal, M. Jamei, and I.-O. Georgopoulos, "Dem simulations of stiff and soft materials with crushable particles: an application of expanded perlite as a soft granular material," *Granular Matter*, vol. 15, pp. 685–704, Oct 2013. 3.1

[127] V. Esnault and J.-N. Roux, "3d numerical simulation study of quasistatic grinding process on a model granular material," *Mechanics of Materials*, vol. 66, pp. 88–109, 2013. 3.1

[128] D.-H. Nguyen, E. Azéma, P. Sornay, and F. Radjai, "Bonded-cell model for particle fracture," *Phys. Rev. E*, vol. 91, p. 022203, Feb 2015. 3.1

[129] J. Wang and H. Yan, "Dem analysis of energy dissipation in crushable soils," *Soils and Foundations*, vol. 52, no. 4, pp. 644–657, 2012. 3.1

[130] Y. P. Cheng, Y. Nakata, and M. Bolton, "Micro- and macro-mechanical behavior of dem crushable materials," *Geotechnique*, vol. 58, pp. 471–480, 01 2008. 3.1

[131] F. Kun and H. J. Herrmann, "A study of fragmentation processes using a discrete element method," *Computer Methods in Applied Mechanics and Engineering*, vol. 138, no. 1, pp. 3–18, 1996. 3.1

[132] S. Nadimi and J. Fonseca, "A micro finite-element model for soil behaviour," *Geotechnique*, vol. 68, no. 4, pp. 290–302, 2017. 3.1

[133] Q. Zheng, H. Zhu, and A. Yu, "Finite element analysis of the contact forces between a viscoelastic sphere and rigid plane," *Powder Technology*, vol. 226, p. 130–142, 08 2012. 3.1

[134] C. yu Wu, L. yuan Li, and C. Thornton, "Rebound behaviour of spheres for plastic impacts," *International Journal of Impact Engineering*, vol. 28, no. 9, pp. 929–946, 2003. 3.1

[135] H. Zhu, Z. Zhou, R. Yang, and A. Yu, "Discrete particle simulation of particulate systems: Theoretical developments," *Chemical Engineering Science*, vol. 62, pp. 3378–3396, 07 2007. 3.3.1

[136] R. KOVAR, B. GUPTA, and Z. KUS, "4 - stick-slip phenomena in textiles," in *Friction in Textile Materials* (B. Gupta, ed.), Woodhead Publishing Series in Textiles, pp. 95–173, Woodhead Publishing, 2008. 3.3.1

[137] K. Nakano and V. L. Popov, "Dynamic stiction without static friction: The role of friction vector rotation," *Phys. Rev. E*, vol. 102, p. 063001, Dec 2020. 3.3.1

[138] P. A. Cundall and O. D. L. Strack, "A discrete numerical model for granular assemblies," *Geotechnique*, vol. 29, no. 1, pp. 47–65, 1979. 3.3.1

[139] D. Baraff, "An introduction to physically based modeling: Rigid body simulation i - unconstrained rigid body dynamics," in *In An Introduction to Physically Based Modelling, SIGGRAPH '97 Course Notes*, 1997. 3.3.1

[140] S. Luding, "Introduction to discrete element methods: Basic of contact force models and how to perform the micro-macro transition to continuum theory," *European Journal of Environmental and Civil Engineering - EUR J ENVIRON CIV ENG*, vol. 12, pp. 785–826, 10 2008. 3.3.1

[141] Michael Nosonovsky, "Who was painlevé and why his paradoxes are so important for the study of friction." https://sites.uwm.edu/nosonovs/2016/10/12/who-was-painleve-and-why-his-paradoxes-are-so-important-for-the-study-of-friction/, 2016. 3.3.2

[142] D. E. Stewart, "Existence of solutions to rigid body dynamics and the painlevé paradoxes," *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, vol. 325, no. 6, pp. 689–693, 1997. 3.3.2

[143] J. Moreau, "Numerical aspects of the sweeping process," *Computer Methods in Applied Mechanics and Engineering*, vol. 177, no. 3, pp. 329–349, 1999. 3.3.2

[144] J. J. Moreau, "Standard inelastic shocks and the dynamics of unilateral constraints," in *Unilateral Problems in Structural Analysis* (G. Del Piero and F. Maceri, eds.), (Vienna), pp. 173–221, Springer Vienna, 1985. 3.3.2

[145] A. Schmitt, J. Bender, and H. Prautzsch, "On the convergence and correctness of impulse-based dynamic simulation," Technical Report 17, University of Karlsruhe, 2005. 3.3.2

[146] C. Bouchard, M. Nesme, M. Tournier, B. Wang, F. Faure, and P. G. Kry, "6d frictional contact for rigid bodies," in *Proceedings of the 41st Graphics Interface Conference*, GI '15, (CAN), p. 105–114, Canadian Information Processing Society, 2015. 3.4

[147] "Embree." https://github.com/embree/embree/releases/tag/v3.13.5. Accessed: 2023-07-07. 4.3, 4.8

[148] "libigl." https://github.com/libigl/libigl/releases/tag/v2.3.0. Accessed: 2023-07-07. 4.6.1, 4.8

[149] D. Charrier, B. Hazelwood, E. Tutlyaeva, M. Bader, M. Dumbser, A. Kudryavtsev, A. Moskovsky, and T. Weinzierl, "Studies on the energy and deep memory behaviour of a cache-oblivious, task-based hyperbolic pde solver," *The International Journal of High Performance Computing Applications*, vol. 33, no. 5, pp. 973–986, 2019. 4.8, 6.2, 6.2.3

[150] "The computational geometry algorithms library." https://www.cgal.org/2022/10/12/cgal551/. Accessed: 2023-07-07. 4.8

[151] "Eigen." https://eigen.tuxfamily.org/index.php. Accessed: 2023-07-07. 4.8

[152] "Simde." https://github.com/simd-everywhere/simde/releases/tag/v0.7.2. Accessed: 2023-07-07. 4.8

[153] "Tetgen." https://wias-berlin.de/software/index.jsp?id=TetGen&lang=1. Accessed: 2023-07-07. 4.8

[154] "Catch2." https://github.com/catchorg/Catch2/releases/tag/v3.1.1. Accessed: 2023-07-07. 4.8

[155] "Glfw." https://github.com/glfw/glfw/releases/tag/3.3.6. Accessed: 2023-07-07. 4.8

[156] "Sympy." https://www.sympy.org/. Accessed: 2023-10-12. 5.1.2

[157] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," 1967. 5.3.2

[158] K. Apinis, H. Seidl, and V. Vojdani, *Enhancing Top-Down Solving with Widening and Narrowing*, pp. 272–288. Cham: Springer International Publishing, 2016. 6.1.3

[159] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, 1995. 6.2

[160] 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, pp. 207–216, IEEE Computer Society, 2010. 6.2

[161] B. Karis, R. Stubbe, and G. Wihlidal, "A deep dive into nanite virtualized geometry." https://www.youtube.com/watch?v=eviSykqSUUw, 2021. Accessed: 2024-10-20. 7

APPENDIX A

Algorithms

A.1 Swept volume closest point in time

We propose modelling a moving triangle as a set of static triangles and using any contact points on this surface to estimate the time of contact. These triangles represent an approximation of the surface of the volume swept by the triangle over the time step. For each of the two moving triangles we create the following static triangles:

1) The original triangle transformed by the state at the start of the time step

3) For each edge (A, B) add the triangles (A_t, A_{t+1}, B_{t+1}) and (A_t, B_t, B_{t+1}) where A_t is the first vertex of an edge at time step t.

2) The original triangle transformed by the state at the end of the time step

If there is a closest point between the triangles in category (1) then the toc is t.

We then compare the triangle in category (1) from triangle A to the triangle from category (1) from triangle B. If this results in the shortest contact then the estimated toc is t.

Next, we compare each of the triangles in category (2) from triangle A to each of the other triangles in category (2) from triangle B. If there is a closest point between

these then the barycentric coordinates of the contact points are used to estimate a toc. Each of the triangles in this category have one vertex from time step t, one from $t+1$ and one that could be either. We use the closeness of the contact point to the vertices "from" each time step to estimate the toc.

Finally, we compare the triangle in category (3) from triangle A to the triangle from category (3) from triangle B. if this results in the shortest contact then the estimated toc is $t+1$.

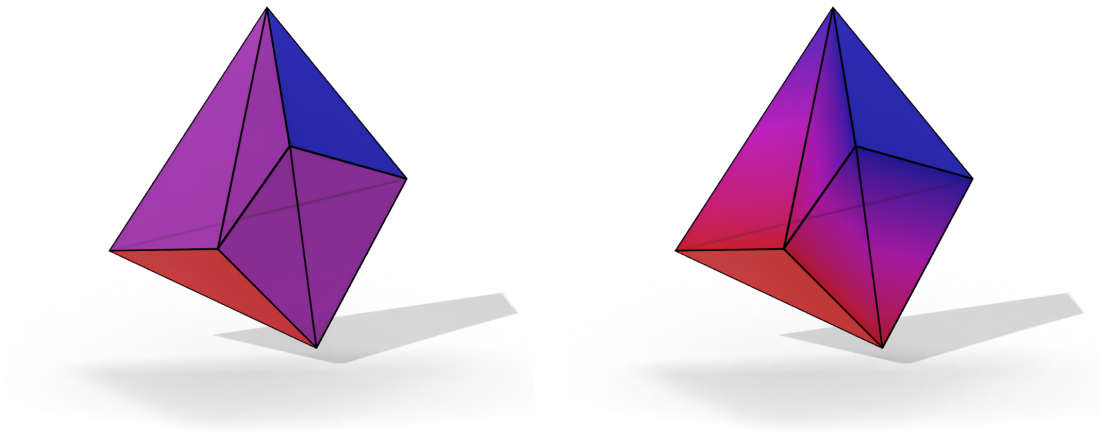


Figure A.1: Left: A triangle at time step t (red), at time step t and the triangles we add to bound the approximated volume swept by the moving triangle (pink). Right: A contact point interpolates the time of contact from the contact location.

This method closely fits our existing triangle comparison model and so automatically see some of the performance characteristics of our earlier algorithm. However, there are some severe limitations:

1. Neighbouring swept triangles sharing edges result in redundant triangles

2. High vector usage is good... not doing the work in the first place is better. We don't need accurate solution, just an estimate of the time, in the same way the contacts have to be accurate.

The alternative method (and the method used for all measurements) considers edge-edge and vertex-face pairs separately and using different methods.

A.2 Iterative triangle-triangle closest point in time

We have shown the potential effectiveness of iteratively computing the closest points between triangles (Chapter 5). A similar scheme could be imagined for the continuous in time version of the problem. Instead of minimising the distance over two pairs of barycentric coordinates (a, b, c and d) for a given pair of triangles we could add a time component and a pair of states for each triangle. The minimisation would be over 5 parameters (a, b, c, d, t).

Like with the barycentric coordinates we also include penalty terms to keep the t parameter within a 0-1 bound. These terms follow the same min/max form as the barycentric coordinates.

The function to optimise:

$$P_1 = A_t + a(B_t - A_t) + b(C_t - A_t)$$

$$P_2 = D_t + c(E_t - D_t) + d(F_t - D_t)$$

$$min_{a,b,c,d,t}(|P_1 - P_2| + penalty)$$

The vertices of the triangles are simply interpolated linearly in time.

$$A_t = A_0(1 - t) + A_1 t$$

Alternatively, the vertices could be interpolated using a spherical linear interpolation (slerp).

$$T_t = slerp(T_0, T_1, t)$$

$$A_t = AT_t$$

where T_0 and T_1 are the transformations of the triangle at the start and end of the time window and $slerp(...)$ is the linear interpolation of translation and spherical linear interpolation of the rotational component of two transformations.

We differentiate the linear interpolation variant using SymPy to create a Newton-Raphson style solver.

$$\begin{aligned}\underset{a_1,b_1,a_2,b_2,t}{\arg\min}\ J(a_1,b_1,a_2,b_2,t) := \quad & \underset{a_1,b_1,a_2,b_2,t}{\arg\min} \underbrace{\frac{1}{2}\|t_i(a_1,b_1,t) - t_j(a_2,b_2,t)\|^2}_{=:\hat{J}(a_1,b_1,a_2,b_2,t)} + \alpha_{\text{iterative}}\Big(\\ & \max(0, a_1 - 1) + \min(-a_1, 0) + \max(0, b_1 - 1) + \\ & \max(-b_1, 0) + \max(0, a_1 + b_1 - 1) + \\ & \max(0, a_2 - 1) + \min(-a_2, 0) + \max(0, b_2 - 1) + \\ & \max(-b_2, 0) + \max(0, a_2 + b_2 - 1) + \\ & \max(0, t - 1) + \max(-t, 0)\Big). \end{aligned} \tag{A.1}$$

While the stationary in time variant of this method converges in most situations, and we have a robust fallback method when it doesn't, in our experiments the continuous in time variant fails to robustly converge in many cases. Further experimentation with this would be a worthy future direction of investigation as the continuous collision detection can quickly come to dominate the run time and we've already shown the effectiveness of this methodology when used for static triangle-triangle distance checks.

The outline of the SymPy code used to derive the Newton-Raphson iterations:

```
import sympy

# Define tx1, ty1, tz1, tx2, ty2, tz2 as the x, y & z coordinates for a
#   point at a moment in time defined by barycentric coordinates and a
#   timestamp on triangle 1 and 2 respectively

distanceX = tx1(a, b, t) - tx2(c, d, t)
distanceY = ty1(a, b, t) - ty2(c, d, t)
distanceZ = tz1(a, b, t) - tz2(c, d, t)

penalty = sympy.Max(0, -a) + sympy.Max(0, -b) +
            sympy.Max(0, -c) + sympy.Max(0, -d) +
            sympy.Max(0, a - 1) + sympy.Max(0, b - 1) +
            sympy.Max(0, c - 1) + sympy.Max(0, d - 1) +
            sympy.Max(0, a + b - 1) + sympy.Max(0, c + d - 1) +
            sympy.Max(0, t - 1) + sympy.Max(0, -t)

# Alpha can be used to tune the 'strictness' of the penalty terms
```

```
# In practice we apply the penalty updates separately to unsure they're enforced
J = 0.5 * (distanceX**2 + distanceY**2 + distanceZ**2 + alpha * penalty**2)

dJ_a = sympy.diff(J, a)
dJ_b = sympy.diff(J, b)
dJ_c = sympy.diff(J, c)
dJ_d = sympy.diff(J, d)
dJ_t = sympy.diff(J, t)

# sympy.simplify can be used on each of these to give a smaller output
ddJ_aa = sympy.diff(dJ_a, a)
ddJ_bb = sympy.diff(dJ_b, b)
ddJ_cc = sympy.diff(dJ_c, c)
ddJ_dd = sympy.diff(dJ_d, d)
ddJ_dt = sympy.diff(dJ_t, t)

update_a = a - 1.0 / (C_dontdividebyzero + ddJ_aa) * dJ_a
update_b = b - 1.0 / (C_dontdividebyzero + ddJ_bb) * dJ_b
update_c = c - 1.0 / (C_dontdividebyzero + ddJ_cc) * dJ_c
update_d = d - 1.0 / (C_dontdividebyzero + ddJ_dd) * dJ_d
update_t = t - 1.0 / (C_dontdividebyzero + ddJ_dt) * dJ_t

# Use C11CodePrinter to output code for each of the updates
```

Furthermore, linearly interpolating the points of each triangle between the start and state is a very inaccurate representation for any triangles undergoing a fast rotation. This may cause contacts to be missed or the incorrect position to be reported.

A.3 Position vs force vs impulse based dynamics

There are a variety of philosophies when it comes to how these are formulated.

- **Position based dynamics** - Contacts are phrased as distance constraints. The positions and rotations of each particle are directly manipulated to satisfy the constraints and then the velocity is computed retrospectively by taking the difference between the last state and the new current state. This method is typically very efficient and stable but lacks accuracy, making this more suitable to real-time applications.

- **Spring based forces** - Contacts are phrased as springs under compression. The shorter the contact the greater the force. Forces between particles are explicitly modelled over time. As two objects approach the force increases over several time steps until the relative velocity of the contact becomes zero and then reverses. The time integration of spring based forces can easily result in numerical instabilities if time step sizes aren't carefully controlled. This is the method typically used where a high degree of accuracy or the ability to customise the behaviour of specific materials are the primary concerns.

- **Impulses** - Contacts are phrased as velocity constraints. Applying an impulse can instantaneously change the velocity of an object without resulting in forces that approach infinite as the time step size approaches zero. Using impulses is typically more efficient that springs and has some of the advantages of explicitly computing a momentum exchange at each contact.

APPENDIX B

Implementation details

B.1 Thread creation

In a code with multiple distinct phases with different performance characteristics and scenarios involving particles with a wide variety of properties (scale, mass, speed, mesh complexity, interaction with other particles) the performance characteristics displayed can be wildly different. For any property it's usually possible to engineer a scenario that will cause the simulation to perform poorly in that regard. For example, to cause the time step selection phase to perform poorly while every other stage performs well the scenario could look like a sparse selection of (concave sphere-like) objects with very complex meshes with similar masses. The sparse nature makes the broad phase and cluster separation cheap and the concave sphere-like nature suggests contacts between particles will be very minimal (likely one per contact pair) and have similar mass ratios so the sequential solver will finish with few iterations over small numbers of contacts. Our solution to this particular bottleneck is to compute the contacts at each level of the surrogate model in parallel. Unfortunately this optimisation comes apart when considering a similar scenario but using simple cubes instead of complex sphere-like objects. As before most phases

of the simulation run extremely quickly but when selecting time step sizes a large number of parallel tasks are created but due to the simple nature of the meshes each task contains very little work. In this case a huge portion of the runtime is spent on thread creation and scheduling and we see a significant slow down (where total available threads is small compared to the work units) or significant decreases in efficiency without significantly improved time to solution (where the total available threads is high compared to the work units). A simple, if inelegant, solution is to provide two code paths - one parallel and one sequential. Then the following addition is to limit the number of threads available for the given amount of work. This allows smaller units of work to be done in parallel without making such large efficiency sacrifices. For very small scenarios this limits the total parallelism but for suitably large scenarios the full number of available cores can still be utilised by having multiple cluster tasks producing these batches of work in parallel.

```
if (work.size() > parallel_threshold) {
    int target_threads = work.size() / parallel_grain_size;
    int default_concurrency = tbb::info::default_concurrency();
    int max_concurrency = std::min(default_concurrency, target_threads);
    tbb::task_arena arena(max_concurrency);
    arena.execute([&] {
        tbb::parallel_for(
            tbb::blocked_range<int>(0, work.size(), parallel_grain_size),
            [&](tbb::blocked_range<int> r) {
            for (int d = r.begin(); d < r.end(); d++) {
                WorkItem& w = work[d];
                w.doWork();
                }
            }
        }
    });
} else {
    for (WorkItems& w : work) {
        w.doWork();
    }
}
```

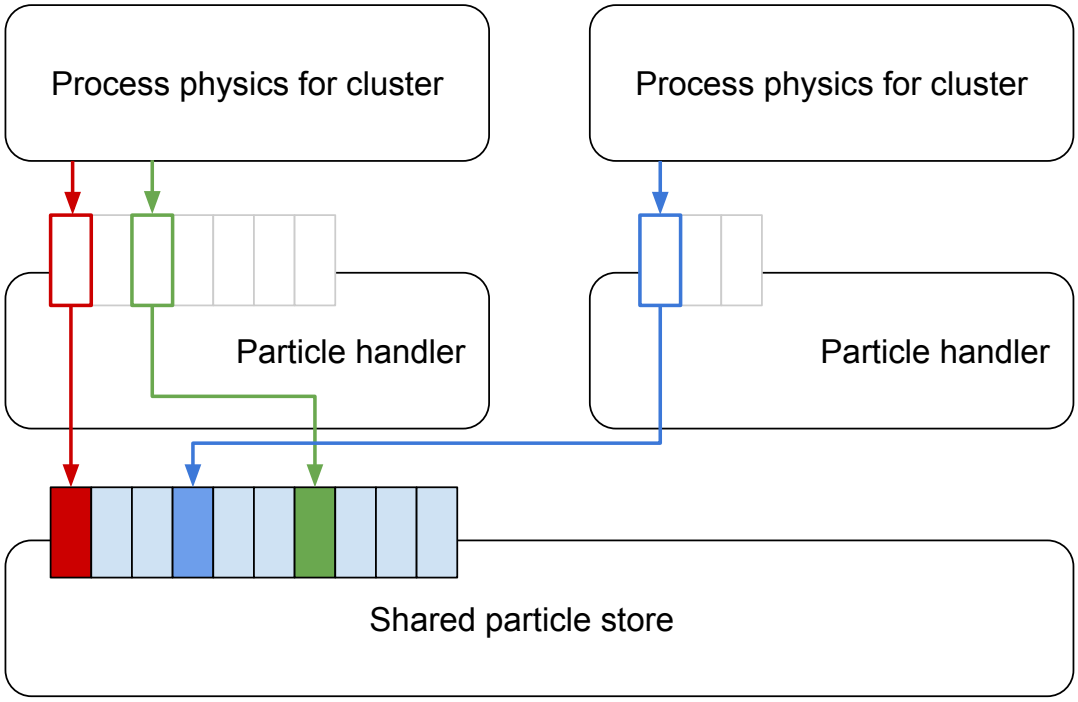


Figure B.1: Two cores independently processing separate clusters. Each is only aware of particles within that cluster while the data for each particle doesn't have to be copied to or from local storage.

B.2 Particle Handler

With a view to future work much of code is build on a component called a Particle Handler, which abstracts away the details of how particles and their meshes are stored and handled. The other reason for implementing the Particle Handler is each stage of our algorithm only has to be implemented as if it is dealing with all the particles in the simulation at once. When we separate off clusters to be processed concurrently we are able to simply create a new particle handler that is only aware of the particles in that cluster. For a shared memory system all the different Particle Handlers have a shared set of particles and then each one simply holds a list of indices to the particles belonging to that cluster (Figure B.1). Particle Handlers can be treated as iterators so individual phases of the algorithm iterate over what appears to be all particles but are only fed the particles in the relevant cluster. To extend our algorithm to a shared memory system a new Particle Handler could be introduced, which keeps its own store of particles that are owned by that node. If particles are required to move to a different node then the Particle Handler objects could coordinate the transfer in a way that is invisible to a developer working purely on the physics code. By doing this the upgrade to a shared memory system would require a mechanism to deciding how to distribute/load balance particles and the Particle Handler logic to move the memory associated with each particle to the relevant node but all the physics code would remain untouched.

Load balancing is the ever present challenge with the DEM algorithm and a distributed memory system would present an even greater challenge for our algorithm. The solution would be by no means trivial but using the Particle Handler mechanism should provide a reasonably painless path for the code that handles processing clusters (Figure B.2).

B.3 Intel Embree

Intel Embree forms a key aspect of the Delta2 application. While originally designed to provide highly optimised raytracing kernels the Embree library gives developers access to an extremely efficient spacial datastructure. This data structure is able to

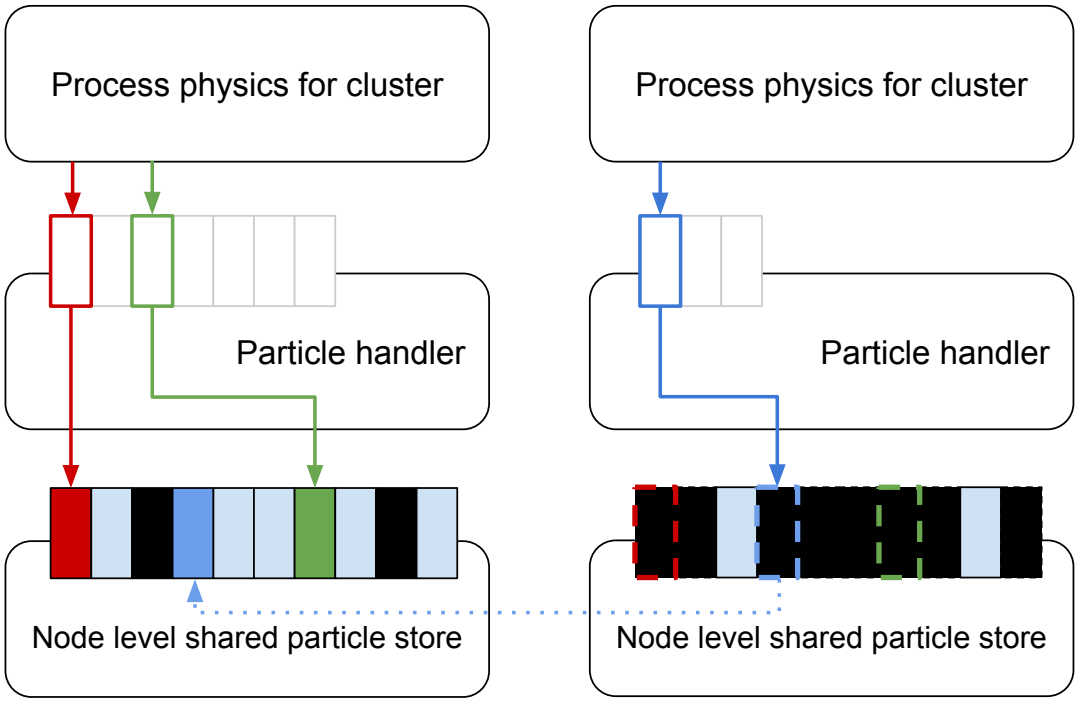


Figure B.2: A conceptual view of two nodes independently processing separate clusters. Each is only aware of particles within that cluster while the data for each particle doesn't have to be copied to or from local storage. The transfer of particle data could be hidden to make writing 'user' code simplier.

be used for more than raytracing and so we utilise it to perform the broad phase collision checks. The Embree repository includes an example for collision detection, however, this use case falls outside the normal use case and so isn't mentioned often in the documentation. We therefore include a brief description of the key step required to make Embree work for our broad phase collision detection.

The library function 'rtcCollide' accepts a pair of 'RTCScene' objects, a collision function and a vector to store the results in. An 'RTCScene' object contains the bounding volume hierarchy data structure at the heart of Embree. The two scene objects that are passed to 'rtcCollide' are compared and for any objects with overlapping bounding boxes are included in an array passed to the collision function when it is called. If this collision function confirms that the objects are in fact colliding then this pair of objects and the id of the geometric primitive is added to the results vector. In our case we don't include primitives (these would be triangles in our case) but instead just look at the bounding box of the whole geometry.

To find collisions within our current setup we first add the bounding boxes (see the discussion in Section 4.3 about time spanning bounding boxes) of all the particles to a scene. Next we pass this scene object as both the scene arguments of 'rtcCollide'. This will inevitably include self contacts in the list of possible collisions passed to the collision function. Therefore, we prefilter the potential collisions by removing any where the ID of both objects in a potential collision pair are identical.

APPENDIX C

Sofware

C.1 Delta2

Everything needed to build and run Delta2 is provided publicly in this repository: `https://github.com/Peter-Noble/Delta2`

A list of the libraries that Delta2 depends on can be found in the repository readme.

C.1.1 Building and running Delta2

The 'src' directory contains the build script for the project. A number of library paths are required to be updated for your system before this makefile will run.

Once this is complete a typical build command might look like: 'make release -j 32'

The 'is_hamilton' (the name of the Durham supercomputer where all measurements were conducted) variable can be used to build the project in a headless mode. With this selected all calls to the OpenGL viewer will be ignored and none of the OpenGL headers will be included or libraries linked.

Using the 'test' build target will build the Catch2 executable for running unit

tests.

Once built the arguments for the executable can be displayed by entering './Delta2 –help'. Perhaps the most important arguments to start with being:

-s	Integer	The scenario to run
-t	Float	The maximum time step size
-f	Float	The final time stamp of the simulation

C.1.2 Source code layout

Within the 'src' directory the source files are separated into the following folders by purpose:

collision_detection contains all the code relating to the triangle-triangle distance checks as well as for traversing mesh hierarchies during contact detection. Both momentary and continuous varients are present here. The two other small but important components here are the definition for the Contact object as well as the function used to separate an interaction graph into clusters.

common contains a wide variety of basic types (such as edges and triangles), basic utility functions used throughout Delta2 (transforms, linear interpolation, angle representation conversions) and debug tools such as the OpenGL viewer and logger.

embree_common is a set of utilities provided by Intel that wraps the Embree library to make certain functionality easier to use.

model contains everything to do with particles, how the geometry is represented and how the state of the particles is stored.

quick_hull is a header only library with some wrapper code in this folder. This is left in for convenience as it is integrated with the mesh objects for maintaining contact witnesses to accelerate repeated contact detection of similar configurations of the same particles. This feature isn't utilised in any of our presented results.

runner is the home for two separate main functions. The make script for the project allows for building a regular executable for running simulation or for building a Catch2 executable that can then be directly run or passed to VSCode to run unit tests.

strategies contains a series of interchangeable components (selected in '../runner/main.cpp') that can be used to customise the behaviour of Delta2. For example, 'contact_detection_hybrid.h' could be swapped out for 'contact_detection_comparison.h' to use the non-iterative version of contact detection throughout the application.

scenarios contains the definitions for all the different scenes that were used for testing and measurement. Each file contains a separate scenario, which are then all included in 'scenario.cpp'.

sympy contains a series of Python scripts that were used to derive (usually iterative) code from formula. Not every experiment represented here made it into the code.

testing houses a series of testing utilities to make unit testing an application that relies heavily on non-primitive types (eg. vectors, matrices, triangles etc).

timestepping is slightly separate from the rest of the application. These classes represent a basic implementation of a selection of time stepping schemes that were used for testing and measuring the multi-resolution code. These schemes only allow for individually interacting particle pairs (ie. clusters always have a fixed size of 2). Included for reference as to how implicit time stepping could be implemented in the local time stepping code or how the core multi-resolution code might be reused in another application.

meshes exists in a directory next to 'src' and contains all the files with geometry needed to run all the scenarios.