# PARTICLE PACKINGS AND MICROSTRUCTURE MODELING OF ENERGETIC MATERIALS

BY

GUILHERME AMADIO

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Aerospace Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2014

Urbana, Illinois

## Abstract

This dissertation explores the use of packings of frictionless hard particles as models of the microstructure of particulate heterogeneous materials.

In the first part of this dissertation, we present the current mathematical framework used for understanding the properties of particle packings, as well as the methods and algorithms we have developed to generate packings of frictionless hard particles with a computer. We develop two algorithms to model hard-particle systems: a collision-driven molecular dynamics algorithm for the simulation of packings of spheres, and a novel hybrid algorithm employing both molecular dynamics and Monte Carlo techniques for the simulation of packings of particles with general convex shapes, such as spheres, cylinders, ellipsoids, polyhedra, etc. We focus heavily on performance in order to enable the simulation of large systems containing $10^6$–$10^7$ particles, previously too computationally expensive to simulate. We use performance benchmarks to demonstrate that our implementations of these algorithms scale roughly linearly with the number $N$ of particles in the system, and show the impact that polydispersivity has on performance.

In the second part of this dissertation we explore the properties of disordered and ordered hard-particle packings. We reproduce key results found in the literature for packings of spheres and polyhedra, and discuss some of their statistical properties. We then follow the discussion with applications of particle packings as models of the microstructure of particulate materials obtained via computed tomography. We find that the shape of the particles and their size distribution both play a crucial role in the determination of the statistical properties of heterogeneous materials.

## Acknowledgements

This research project has been a long journey, and I am indebted to a number of people that supported me along its way. I would like to thank my family for their immense support and love, without which I could not have finished this work. Being away from home for such a long period of time has been difficult at times, but my family's support and understanding made it possible.

I would like to thank my adviser, Dr. Tom Jackson, for his guidance and support throughout my stay in Illinois. I am also thankful to the other members of our research group, Yuya Matsumura and Yang Zhang. Doing research with all of you has been a very pleasant and positive experience, as you helped build a great and collaborative environment.

Finally, I would like to thank prof. Steven Son at Purdue University for the preparation of the material samples that were used in this research project.

Work presented in this dissertation was supported in part by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), under grant BEX-1911-07-4; by Fulbright and the Institute of International Education, under ID number 15087693; and by the Defense Threat Reduction Agency, Basic Research Award № HDTRA1-13-0010, to University of Illinois at Urbana-Champaign; Dr. Suhithi Peiris, program manager.

*To my parents.*

# Contents

# Chapter 1

# Introduction

This dissertation focuses on the generation of realistic computer models of the microstructure of energetic heterogeneous materials relevant to the space industry, such as solid rocket motor propellants and explosives. Accurate prediction of the macroscopic properties of these materials is crucial for the development of the next generation of space exploration missions that can improve our knowledge of the universe. Nonetheless, the methods presented here have a wide range of applications in materials science, engineering, classical geometry, and other areas of research.

## 1.1 Characteristics of Energetic Heterogeneous Materials

Heterogeneous materials have inhomogeneities that are in general much larger than the molecular scale of its constituents, albeit much smaller than typical macroscopic length scales. These inhomogeneities are commonly referred to as the microscopic structure or *microstructure* of the material. The length scale of the microstructure varies greatly depending on the material, ranging from a few nanometers for some gels and fine powders to several meters in geological formations. In solid propellants and explosives, this length scale lies somewhere in between—in the range of a few to a few hundred micrometers.

The solid propellants and explosives we intend to model are particulate composites—they typically comprise a mix of oxidizer particles embedded in a polymerized binding matrix. Oxidizer particles are often small crystals, hence their shapes most closely resemble polyhedra. Some common oxidizer materials found in solid propellants are ammonium perchlorate (AP), ammonium dinitramide (ADN), and ammonium nitrate (AN). In explosives, crystalline materials such as HMX, RDX, PETN, and CL–20 are used more often. In some cases, energetic crystals usually found in explosives are utilized in solid propellants to enhance their burning rate. Among binding materials, hydroxyl-terminated polybutadiene (HTPB), dicyclopentadiene (DCPD), and polybutadiene acrylonitrile (PBAN) are all very common. Many solid propellants also contain additives to tune burning characteristics and reduce smoke—and, consequently, environmental impact—, among other things. An extensive discussion of solid rocket propellants can be found in [1].

Although the determination of the macroscopic properties of heterogeneous materials has received a lot of attention in recent years [2–8], most studies have employed rudimentary models that represent their microstructure using disks and spheres (e.g. [5–7]). The microstructural information can play a crucial role in determining the behavior of a material under certain physical processes. To illustrate this fact, consider the propagation of mechanical waves inside a heterogeneous explosive material. This physical process is highly dependent on the geometric shapes of the oxidizer particles in the material, since the interface between fuel and oxidizer is a source of reflected shock waves that can drive the local temperature up until detonation is triggered. Micropores within crystalline particles of the oxidizer can also become hot spots when hit by shock waves. The transition from shock waves to detonation is therefore an important phenomenon for designers of explosives, making realistic models of materials containing crystalline particles an immediate need. Particle shape is also important in modeling the combustion of solid rocket propellants [9–11], as well as in the determination of effective properties of heterogeneous materials, such as permeability [12] and stress [13].

On the other hand, fine aluminum powder is a common additive incorporated into many solid rocket motor propellants—e.g., in the solid boosters of the now retired space shuttle—to increase specific impulse [1]. A typical composition of such a propellant might be, by weight, 70% AP, 18% aluminum, and 12% binder. The increase in specific impulse provided by the addition of aluminum comes from the exothermic reactions with water and carbon dioxide—both of them products of combustion—, and can reach about 10%. The addition of aluminum has other desirable effects as well, such as damping some combustion chamber instabilities. However, the introduction of small aluminum particles also makes modeling propellants more complicated [10], since near the burning surface the binding matrix melts, enabling aluminum particles to migrate and agglomerate into larger particles that contribute significantly to nozzle erosion. The key

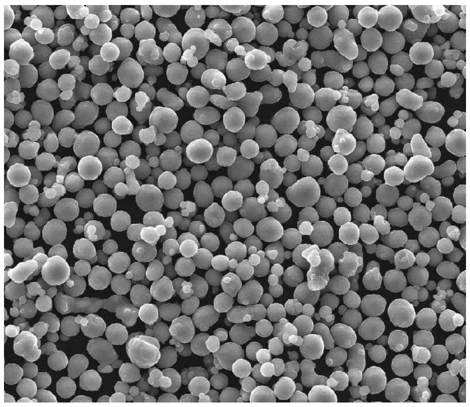


Figure 1.1: Fine aluminum powder.

parameters affecting agglomeration are the size of the aluminum particles and the average size of the fuel rich pockets between the larger AP particles. Due to nozzle erosion, models of aluminum agglomeration [11, 14, 15] are essential to rocket designers concerned with safety. A failed launch can cost millions of dollars or even worse, lives. Although spheres can be used to model aluminum powder, the challenge in this case is the large number of particles needed—from $10^5$ to $10^6$—and the high polydispersivity of aluminized propellant materials, since aluminum particles tend to be much smaller than oxidizer particles.

In some cases, physical quantities of interest are measured on a variety of propellants and curve-fitting methods are then employed to obtain empirical formulas to describe how they vary with the composition of the material. However, not only are these methods susceptible to large potential errors when extrapolated to untested regions, but the experiments can be both time-consuming and very expensive. Therefore, it would be helpful to be able to substitute the experiments with a physics-based computer model to predict the macroscopic properties of heterogeneous materials from the properties of their constituents and their microstructure. The first step in the development of such a model is to produce a realistic representation of the microstructure of the materials, and that is the motivation for our research.

In the next section, we provide some brief historical remarks before we proceed with a discussion of the methods and algorithms we use to simulate the microstructure of heterogeneous materials with packings of convex hard particles.

## 1.2 Historical Remarks

Heterogeneous materials are an essential element of modern technology. Even so, their utility is known since ancient times. Perhaps the best example of an ancient heterogeneous material is Roman concrete [17]. In ancient Rome, hot lime was mixed with a volcanic sand—called pozzolana—and pomice to produce a primitive version of concrete. The Romans used this mixture extensively to build their cities, roads, bridges and aqueducts. The high compressive strength of concrete, combined with architectural novelties of the time such as arches, vaults, and domes, allowed the Romans to revolutionize the ancient world [18]. Masterpieces such as the Colosseum, Pont du Gard, and the Pantheon—which remained as the largest dome in the world for over a thousand years—were all built with concrete. These ancient buildings became icons of the durability of Roman concrete, the secrets of which are still the topic of modern research [16, 19]. The Romans even used additives to enhance their concrete. Volcanic ash, for example, was used to produce hydraulic cements that could set underwater, while horse hair prevented shrinkage, making concrete less susceptible to cracks while hardening. Unfortunately, the secrets of Roman concrete were lost with the fall of the Roman empire. Concrete's resurgence would only happen 18 centuries later, when it allowed for a second revolution in architecture that led to the skyscrapers of today.

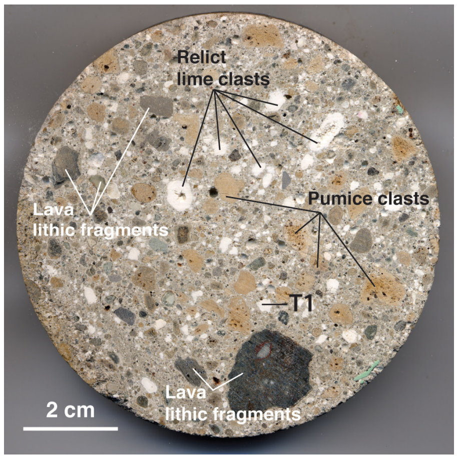


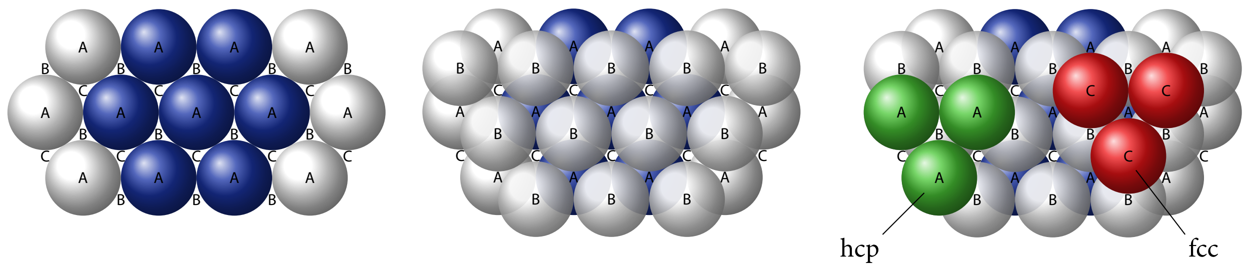
Figure 1.2: Microstructure of Roman concrete, from [16].

Although heterogeneous materials were already known to the Romans and even earlier civilizations [17], the theoretical framework that allowed us to begin understanding the connection between their microscopic structure and their macroscopic properties came only much later, in the early 19th century. In 1821, Cauchy and Navier laid down the foundations of continuum mechanics using a Newtonian picture of matter as an assemblage of "material molecules" interacting via central forces. The continuous model arising from this discrete description of matter is commonly referred to as a *homogenization* of the problem, in the sense that the macroscopic behavior of matter can be derived from its underlying "molecular" behavior. The idea of homogenization problems would later reappear in other fields of research as well. In 1824, Poisson presented his first *Mémoire* [20] on the theory of magnetism to the French Academy. In it, Poisson had a model for induced magnetism in which he considered an insulating material embedded with an isotropic distribution of small conducting spheres. In his second *Mémoire* [21], Poisson generalized his idea, substituting the spheres with aligned conducting ellipsoids to treat anisotropic magnetic materials. Later, in 1838, Faraday applied Poisson's ideas to dielectrics [22], using a similar model of conducting spheres embedded in an insulating material. Maxwell, in his famous *Treatise on Electricity and Magnetism* [23], summarized the work of Faraday [22, 24] and others. He unified the theories of electricity and magnetism, and derived an exact expression for the electric conductivity of a dilute emulsion of spheres. Mosotti and Clausius—for which the formula would later come to be known—derived the formula independently, although the explicit formula appeared only in Clausius' work. Later, in 1905, Einstein determined the effective viscosity of a suspension of spheres in a liquid as part of his PhD work [25]. These and other problems were important precursors that helped us better understand heterogeneous materials—which now go much beyond Roman concrete and have a vast range of applications in diverse fields of science and engineering.

Many heterogeneous materials are synthetic: gels, foams, fiber composites, particulate composites, powders, emulsions, phase separated metallic alloys, etc; others are natural: wood, bone, soils, minerals, polycrystals, etc. The remarkably broad range of rich and complex microstructures exhibited by these materials means that the basic ideas introduced in years past are simply not enough for one to study them in detail. Neither is it possible to design new, even more complex materials without a more powerful engineering framework to guide us.

Random packings of particles are a natural step forward from the single inclusion models they supplanted. Packings of particles can provide many insights into the structure and bulk properties of many materials, including glasses, crystals, granular media, etc. For this and other reasons, packing problems have a history as long as that of heterogeneous materials. As remarked by Bernal in [26], "heaps (close-packed arrangements of particles) were the first things that were ever measured in the form of basketfuls of grain for the purpose of trading or the collection of taxes." The study of the mathematical properties of sphere packings—in particular, the face-centered cubic packing—can be traced as far back as 499CE, to a work composed in Sanskrit [27].

In more modern times, late in the 16th century, Hariot—who at the time was Raleigh's mathematical assistant—became interested in close packings of spheres after Raleigh assigned him the task of determining formulas for the number of cannonballs in regularly stacked piles. Hariot was the first to distinguish between the face-centered cubic (fcc) and hexagonal close packing (hcp) configurations (shown in figure 1.3). He also connected sphere packings to Pascal's triangle—which was only introduced by Pascal himself much later.



(a) single layer (b) two layers (c) three layers

Figure 1.3: Difference between face-centered cubic (fcc) and hexagonal close packing (hcp).

At the turn of the century, Kepler also became involved with packings of spheres through his correspondence with Hariot. In 1611, Kepler described the face-centered cubic packing in an essay exploring a theory of matter composed by small spherical particles. He conjectured that in this configuration (fcc) "the packing will be the tightest possible, so that in no other arrangement could more pellets be stuffed into the same container." Later, in 1831, Gauss proved [28] that this was the densest possible Bravais lattice packing of spheres. However, the formal proof by Hales [29] that no configuration could exceed the density of face-centered cubic packing ($\phi = {}^{\pi}/_{3\sqrt{2}} \approx 0.74048\ldots$) would only emerge another two centuries later, in 2005. In fact, Hales published an entire series on sphere packings [27], beginning with an overview of the problem and culminating with a rigorous proof of Kepler's conjecture. For the interested reader, Hales's series has a much more detailed account of the history of sphere packings.

More recently, the attention has shifted towards packings of more complex objects, such as polyhedra. There has also been a dramatic increase in the number of studies that explore applications of disordered packings in many fields of research. In the next section, we discuss some of these applications and provide a brief overview of the methods available to generate particle packings with a computer.

## 1.3 Introduction to Particle Packings and Applications

Packings of disks and spheres have played an important role in the modeling of liquids [26, 30–32], glasses [33, 34], colloids [35, 36], granular media [37, 38], heterogeneous materials [39–42], as well as in the simulation of physical processes such as fluid flow through packed beds [43–45], among other things. However, packings of disks and spheres have hitherto been used mostly due to their simplicity. In reality, most materials have particles with irregular shapes and continuous size distributions, as is the case of the solid propellants and explosives we would like to model. Recently, with the advent of powerful computers and nondestructive X–ray imaging techniques, the level of sophistication with which we can study complex heterogeneous materials has improved significantly. The need to refine rudimentary sphere models has also led to an increasing number of studies exploring the properties of packings of nonspherical hard particles such as ellipsoids [46, 47], superellipsoids [48], superballs [49], and polyhedra [50–57]. A few studies also explore packings of polyhedra experimentally with plastic and ceramic dice [58, 59], as well as synthetic nanocrystals [60]. In [60], Henzie *et al.* report that the packing structure of silver nanocrystals depends on the packing method—their results differ if they either dry a droplet containing a suspension of particles, or let particles suspended in a liquid settle to the bottom due to gravity. Packing densities and order metrics of packings generated experimentally are not necessarily the same as those obtained with a computer. Nevertheless, computer models can provide invaluable insights in the design of new materials. For example, another recent study [57] investigates computationally the self-assembly of polyhedra into crystals, glasses, and disordered structures, depending on their shape. However, despite the growing number of studies in this area, there are still many open questions concerning packings of nonspherical particles. The determination of the densest packing of tetrahedra, for example, is a notoriously difficult problem to solve. It is part of the $18^{\text{th}}$ problem in Hilbert's famous set of problems. For some time, the densest known packings of tetrahedra reported in the literature were quasi-crystals with density $\phi > 0.8$. Torquato and Jiao [51, 52], and Haji-Akbari *et al.* [50] took turns for who held the record density until Kallus [61] identified dense packings of tetrahedra analytically. The dense packings identified by Kallus were based on a two-particle fundamental cell. Torquato and Jiao [62, 63], and Chen [64] later improved the results of Kallus by eliminating some unnecessary constraints in his work, obtaining slightly denser packings. The current densest known packing of tetrahedra obtained after the improvements has a density of $\phi \approx 0.856347$. Is it possible to pack tetrahedra even further? That remains an open question. We need to build new methods and tools that will allow us to tackle this and other particle packing problems, that still remain relatively unexplored.

## 1.4 Survey of Packing Algorithms

Before we describe our own approach to generating disordered packings of convex particles, we thought it would be useful to provide a brief discussion of other methods found in the literature, most of them for generating packings of disks and spheres.

Perhaps the simplest packing protocol is the random sequential addition method [65–67]. In this method, particles are added to the simulation region one by one, while ensuring that they do not intersect any of the previously added particles. Although this method is quite limited, it is extensively used to provide initial configurations for more sophisticated methods. Packings of monodisperse spheres generated by this method can only reach packing fractions up to about $\phi \leq 0.4$ [66]. This algorithm has also been used to find the saturation densities of packings of hyperspheres in higher dimensions [65, 68]. An example of how this algorithm could proceed is shown in figure 1.4.

Another class of sequential packing algorithms consists of sedimentation methods [69–72], in which particles are placed in a linear gravitational field and fall onto an initially disordered substrate. Each subsequent sphere then rolls on top of the others until it forms three contacts and its configuration becomes stable. This kind of algorithm is thought to generate "random loose packings" representative of granular materials when gravity dominates interparticle forces—i.e., when there is no mechanical shaking. However, packings generated with this method form layers, hence they show strong vertical anisotropy. In order to cope with this problem, some variations of sedimentation algorithms employ a centrally symmetric gravitational field and let particles fall from

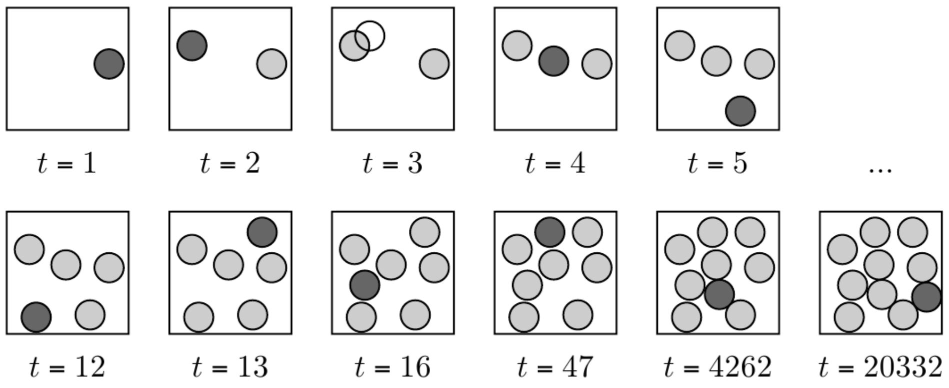


Figure 1.4: Random sequential addition algorithm.

random directions on top of an initial seed [69]. Some other groups combine sedimentation methods with particle rearrangements to mitigate anisotropy and obtain more closely packed structures at the same time.

In the Zinchenko packing algorithm [73], sticky spheres are placed randomly in space and form a contact network as they grow with a uniform rate. When contacts are formed, a self-stress is computed to resist further particle growth. When all contact forces are negative, meaning that no contact can be broken, the simulation stops. This algorithm produces isostatic packings with uniform packing density around $\phi \approx 0.637$. It is a quite unique algorithm, but it is difficult to implement efficiently, so it has not been used in many studies.

Force-based algorithms, such as the one by Jodrey and Tory (JT) [74] are more common. In this algorithm, spheres have an impenetrable inner diameter $d_{in}$ and a soft outer diameter $d_{out}$. At each step of the simulation, outer overlaps are eliminated by displacing the spheres according to a central repulsive force given by

$$F_{ij} = d^i_{out} d^j_{out} \left[ \frac{4r^2_{ij}}{(d^i_{out} + d^j_{out})^2} - 1 \right].$$

The outer diameter is then gradually reduced during the simulation until it reaches the inner diameter. This algorithm can be seen as an energy minimization procedure in which any soft overlap acts to increase the energy. It has been generalized to packings of spherocylinders in [75]. Other groups have also used force-based and overlap elimination algorithms with different potentials to study packings of hard particles [76–78] and random porous materials [79].

The Torquato–Jiao (TJ) algorithm [80, 81] is a linear programming method to efficiently produce strictly jammed sphere and hypersphere packings. This method can create packings of spheres with densities varying continuously from the so called "random close packing" density of $\phi \approx 0.64$ to the maximum packing density of spheres $\phi = \pi/3\sqrt{2}$. In this method, the packing problem is cast as an optimization problem where the design variables are the position of the spheres and the function to be minimized is the negative of the packing fraction. The adaptive shrinking cell algorithm we will describe later is based on the same principles as this method, but is used to produce packings of polyhedra instead.

Another class of algorithms we would like to discuss are those based on molecular dynamics. Lubachevsky and Stillinger (LS) were the first to introduce the idea of applying molecular dynamics techniques to packing problems [82–84]. The basic idea of the LS packing algorithm is to start the simulation with a set of infinitesimally small disks or spheres, and allow them to grow over time while they undergo elastic collisions that act to prevent intersections. Since collisions are the only form of interaction between the particles, instead of using a fixed time step, the simulation can evolve from event to event, where an event is anything that changes the motion state of a particle. For this reason, this algorithm is said to be *collision-driven* or *event-driven*. Event-driven molecular dynamics (EDMD) packing algorithms have a strong appeal, since they have a direct connection with the underlying physical processes of liquids, glasses and crystals. EDMD algorithms are in general very computationally efficient for packing spheres and other smooth convex shapes, such as ellipsoids, and appear in numerous studies [47, 85–88].

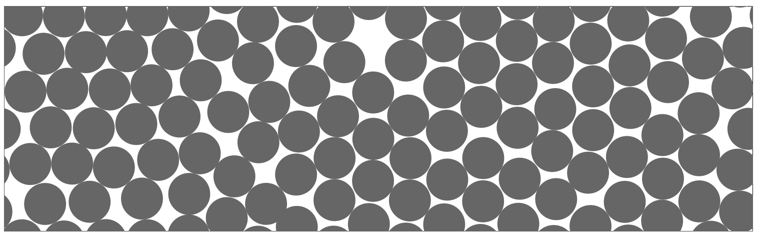


Figure 1.5: Packing of disks generated with the Lubachevsky-Stillinger packing algorithm.

The algorithms we described so far have only been used for packing spheres and other simple smooth shapes. These shapes are less difficult to pack than polyhedra because there are readily available algebraic methods to determine particle intersections. Algebraic methods, if implemented correctly, can be very numerically robust. For polyhedral particles, no such methods exist. Algorithms for computing intersections of polyhedra can suffer from numerical errors that negatively affect the performance of the packing algorithm. For that reason, when packing particles with sharp edges such as polyhedra, collision-driven algorithms can be less efficient than the alternatives. The collision response between two expanding polyhedra may fail to prevent future intersections due to numerical errors, getting the simulation locked into low density states. In order to overcome this limitation, algorithms for packing polyhedra usually abandon the model of particle growth used in many of the methods for packing smooth shapes in favor of Monte Carlo techniques. That is the case of the adaptive shrinking cell (ASC) scheme by Torquato and Jiao [51, 52]. In the ASC method, particles are placed randomly inside a periodic, deformable fundamental cell at the beginning of the simulation. Then, random displacements and rotations are performed. Trial deformations are also performed on the fundamental cell. If the fundamental cell can shrink after a trial move, it is accepted, otherwise it is rejected. This is an optimization procedure similar to the one described in the TJ algorithm, but the design variables must now also include particle orientations. As the size of the fundamental cell decreases during the simulation, the packing density of the particles inside increases. When no particle can be moved while keeping the others fixed and the fundamental cell cannot be shrunk any further, the packing is considered jammed and the simulation stops.

Whilst Monte Carlo methods such as the ASC are the ones most commonly found in the literature [50–52, 57, 60], they are by no means the only ones. An interesting algorithm based on the erosion of tetrahedral elements in a tessellation of space has been proposed in [56]. They seem most interested in simulating the structure of granular media. In [89], a force-based algorithm is used to simulate a packing of irregular polyhedra. We see the development of novel algorithms as being very important for the field, since each one can access different configurations and let us explore different aspects of particle packings.

Our own method to generate packings of convex hard particles is a hybrid between molecular dynamics and Monte Carlo methods. For this research project, we first implemented an extension of the molecular dynamics packing algorithm of Lubachevsky and Stillinger [82] for packing spheres into several boundary geometries. This code was designed to scale well when packing the large number of spheres ($N \geq 10^6$) necessary in aluminum agglomeration models. It was meant to pack both spheres and polyhedra, but since it did not perform well in the latter case, we decided to implement a new method specific for polyhedra, which we will describe in the first part of this thesis, after introducing some basic concepts about particle packings and related topics.

# Part I

# Theory and Algorithms

# Chapter 2

# Basic Concepts

In this chapter we summarize some of the basic concepts and definitions concerning particle packings. A *packing* is simply a collection of nonintersecting solid objects or particles in $d$-dimensional Euclidean space $\mathbb{R}^d$ or in a subset of it, such as the interior of a box with solid walls or the interior of a sphere. Packings can also be defined in compact and curved spaces, such as on the surface of a sphere. However, our primary focus is the generation of packings of convex hard particles in $\mathbb{R}^3$. The *packing fraction* or *packing density*—here denoted by the Greek letter $\phi$—is simply the fraction of space covered by the particles.

A *saturated packing* is one in which there is no available space to add another particle of the same kind. Most interesting monodisperse packings of particles are at least nearly saturated. However, the notion of saturation in a polydisperse packing becomes more vague, since it may always be possible to add smaller and smaller particles to it. If no particle in a saturated packing is able to move while keeping all of the other particles fixed, then the packing is considered to be *jammed*. The different classes of jamming will be discussed in more detail later.

A lattice $\Lambda$ in $\mathbb{R}^d$ is a subgroup of the integer linear combinations of vectors that constitute a basis for $\mathbb{R}^d$. It is also generally referred to as a *Bravais* lattice in the physical sciences and engineering. A *lattice packing* is one in which all particles have a common orientation and their centroids are located at the points of $\Lambda$. Lattice packings are a subset of the possible packings for a given particle shape in $\mathbb{R}^d$. Lattice systems are classified according to the axial distances and angles of the lattice vectors. There are five different Bravais lattice systems in two dimensions: square, rectangular, rhombic (centered rectangular), oblique, and hexagonal. In three dimensions, there seven lattice systems: cubic, hexagonal, tetragonal, orthorhombic, rhombohedral, monoclinic, and triclinic. These lattice systems are shown in figures 2.1 and 2.2, respectively.

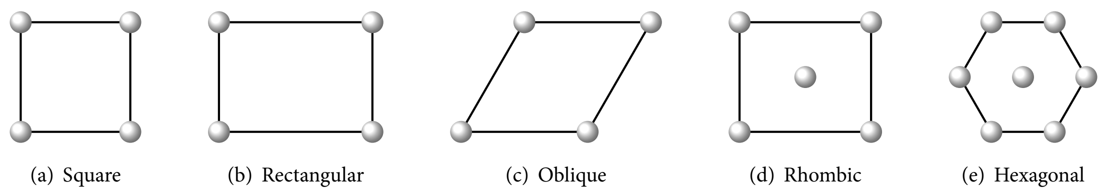


Figure 2.1: Bravais lattice systems in two dimensions.

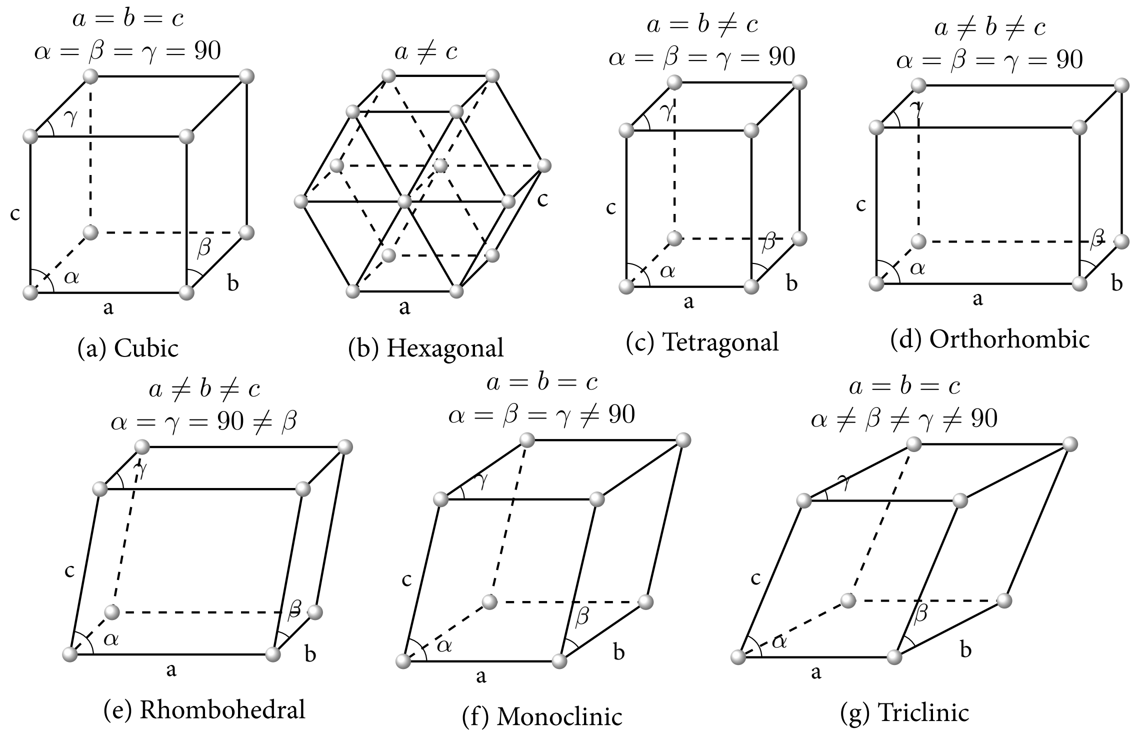

$a = b = c$
$\alpha = \beta = \gamma = 90$
$\gamma$ c $\beta$ $\alpha$ a b
(a) Cubic
$a \neq c$
c a
(b) Hexagonal
$a = b \neq c$
$\alpha = \beta = \gamma = 90$
(c) Tetragonal
$a \neq b \neq c$
$\alpha = \beta = \gamma = 90$
(d) Orthorhombic
$a \neq b \neq c$
$\alpha = \gamma = 90 \neq \beta$
(e) Rhombohedral
$a = b = c$
$\alpha = \beta = \gamma \neq 90$
(f) Monoclinic
$a = b = c$
$\alpha \neq \beta \neq \gamma \neq 90$
(g) Triclinic

Figure 2.2: Bravais lattice systems in three dimensions.

Torquato and Jiao [51, 52] recently conjectured that, for centrally symmetric particles, Bravais lattice packings are the densest possible packings. However, that is not true for particles that are not centrally symmetric, such as the tetrahedron, as its densest Bravais lattice packing has a relatively low packing density $\phi = \frac{18}{49} \approx 0.367$, while dense disordered packings with $\phi > 0.8$ have been reported in the literature [50–52]. No proof exists that this conjecture is true, although both computational [51, 52] and experimental [60] data indicate that it holds. If the four centuries it took between Kepler's conjecture and its formal proof are any indication, we might still be a long way away from coming to a rigorous conclusion on the validity of this conjecture.

The concept of a lattice packing can be generalized to that of a *periodic packing*, by allowing the particles to have arbitrary orientations and filling the lattice cell with not one but $N \geq 1$ nonoverlapping particles. The packing is still periodic under translations by $\Lambda$, but particles can now occur anywhere within the periodic cell. Lattice packings have a high degree of order, while periodic (and nonperiodic) packings are in general highly disordered. We are much more interested in disordered packings, since most of the heterogeneous materials we intend to model consist of random packings of small crystalline particles.

## 2.1 Jamming in Hard-Particle Packings

Jamming is a subject of considerable interest in packings of hard particles of various shapes. We have an intuitive notion for what jamming means, but a precise mathematical definition is necessary to describe the several types of packings that can be idealized or generated in simulations. Early studies found in the literature had only a vague notion of jamming based on the idea that when it was no longer possible to increase the volume fraction of the particles in a packing simulation, the packing was considered to be jammed. However, there are important distinctions that need to be made between different types of jammed states. Torquato and Stillinger were the first to make such distinctions [90]; they classified jammed states into three hierarchical categories that we describe later in this chapter.

### 2.1.1 Mathematical Definition of Jamming

In the mathematics literature [91], jammed packings are commonly referred to as *stable* or *rigid* packings, as they are mechanically rigid against applied external loads. The definition of jamming in this context is in terms of the available configuration space of the particles. Consider, for instance, a hard-particle packing $\mathcal{P}$ in a finite region of Euclidean space $\mathbb{R}^3$. Its configuration is characterized by the positions and orientations of all particles, that is, $\mathcal{P} = \mathcal{P}(\vec{r}_1, \mathbf{q}_1, \ldots, \vec{r}_n, \mathbf{q}_n)$ where $\vec{r}_1, \ldots, \vec{r}_n$ are the positions of each particle and $\mathbf{q}_1, \ldots, \mathbf{q}_n$ are their orientations, represented here as *quaternions* (see e.g., [92, Chapter 10] for more information).

Naturally, not all configurations are accessible to the system of particles—configurations where any pair of particles overlap are discarded by requiring the packing to satisfy nonpenetration constraints. Additionally, there may be boundary conditions that need to be satisfied—e.g., the particles may be confined to a box with either solid or periodic boundaries, or into a finite region of a different shape, such as a cylinder or a sphere, for example. Therefore, while at low density any two configurations can be reached via continuous transformations (displacements and rotations of the particles) between them, as the density increases, the volume of the available configuration space decreases, reflecting the increasing lack of mobility of the particles. When the packing is near saturation, this leads to isolated regions in the configuration space—it is no longer possible to continuously transform a given state into another, as the interstices between particles is too small, preventing them from moving freely to another part of the packing. This isolated region is called a *jammed basin* in the configuration space. The packing becomes jammed when the volume in the configuration space near the jammed basin approaches zero—i.e., no particle can move without violating the nonpenetration constraints. To avoid ambiguity, when two configurations differ only by a global rigid-body motion of the particles—i.e., when all interparticle distances are the same between both configurations—they are considered to be equivalent to each other.

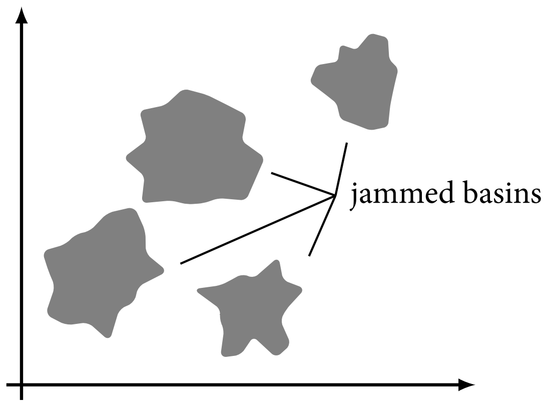
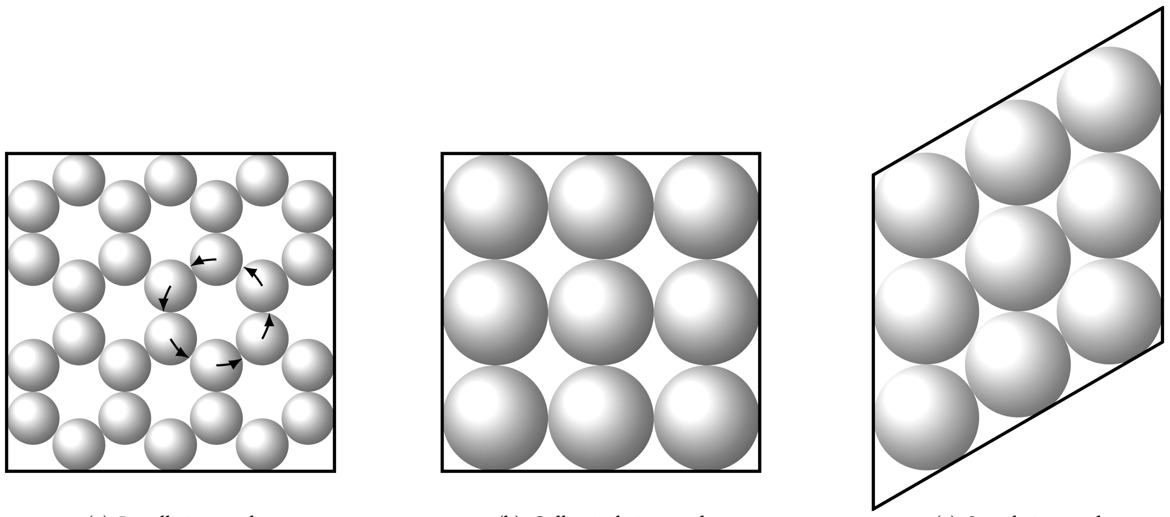


Figure 2.3: Jamming as isolation in configuration space.

## 2.1.2 Jamming Categories

Consider a packing of N particles in $d$-dimensional Euclidean space. Additionally, consider that the particles are confined to a convex region whose boundary can be considered smooth on the scale of the particle diameters. The boundary can either have solid walls or be periodic, and it may also be deformable. An individual particle in the system is considered to be jammed if it cannot be translated or rotated while maintaining all other N-1 particles fixed in place. For the entire system to be jammed, it is necessary that each of the N particles be individually jammed. This means that there can be no "rattlers" in the system. In computer generated packings, however, that can be guaranteed only to a certain degree, as the configuration of the system is limited to the finite machine precision. Nevertheless, the concentration of rattler particles is usually small, and they can be simply removed from the system if jamming is the main subject to be studied. Let us then move on to describe each of the jamming categories as proposed by Torquato and Stillinger [90].



(a) Locally jammed (b) Collectively jammed (c) Strictly jammed

Figure 2.4: Jamming categories.

A packing is considered to be *locally jammed* if the system boundary is not deformable and the conditions we described so far are met, that is, if each particle in the system is individually jammed. If a packing is locally jammed, it may be possible to unjam it via a collective motion of the particles, as shown in figure 2.4(a).

A packing is considered to be *collectively jammed* if the system boundary is not deformable and it is a locally jammed configuration in which there is no collective motion of any contacting subset of particles that would cause the packing to become unjammed. Figure 2.4(b) shows an example of a collectively jammed packing of disks on a non-deformable square boundary.

Finally, a packing is considered to be *strictly jammed* if, in addition to being collectively jammed, its configuration remains fixed under infinitesimal global boundary deformations. In other words, any global boundary deformation, accompanied or not by an individual or collective motion of the particles, would cause particles to intersect each other. Figure 2.4(c) shows a strictly jammed packing obtained by shearing the boundary of the packing in figure 2.4(b).

It is important to note that these definitions can and do depend on the type of boundary conditions imposed. For example, the Kagomé lattice in two dimensions is not even locally jammed on a system with a rectangular solid wall boundary, but can be strictly jammed within a suitable hexagonal boundary. Similarly, a packing may fall under a different jamming category if its boundary has solid or periodic walls.

## 2.2 The Maximally Random Jammed State

Geometric and statistical properties, and jamming characteristics of hard-sphere packings have been a persistent theme of studies found in the literature. Early on, beginning with the pioneering works of Bernal [26, 30] and others [93, 94], emerged the concept of a uniquely defined state—namely, the random close packed (RCP) state—at which the packing density of random congruent spheres was believed to be at its *maximum*. This traditional view dominated the literature until the turn of the millennium, when Torquato *et al.* [95] argued that the mathematical definition of this state was not sufficiently precise to be useful and should therefore be abandoned. The problem with the RCP state, as pointed out by Torquato [95], is that the terms *random* and *close* are at odds with each other. As a matter of fact, packings of spheres at the so called RCP state (with $\phi \approx 0.64$) are not at the highest possible packing fraction that spheres can attain ($\phi_{\text{max}} = {}^{\pi}/_{3\sqrt{2}} \approx 0.74048$). Therefore, it is always possible to find a configuration with a marginally larger packing fraction at the expense of some of the randomness in the system, which means that the proportion between densification and randomness is arbitrary in the RCP state, invalidating the assumption that it is well-defined. Evidence of this abounds in the literature, since different packing protocols produce packings with variable final densities [70, 73, 74, 77, 80, 95]. In particular, experiments in which ball bearings were poured into large vibrating containers have demonstrated that the final density can vary with the amplitude, frequency, and duration of the vibrations, as well as with the pouring rate and the smoothness and shape of the containers [96]. To address the shortcomings of the RCP state, Truskett *et al.* introduced the notion of an *order metric* [97] to quantify the degree of order of a given packing configuration. The introduction of the order metrics by Truskett [97] allowed for the definition of a state with more precise properties than the RCP state: the maximally random jammed (MRJ) state. The MRJ state is the strictly jammed state that minimizes one or more of the order metrics. The diagram in figure 2.5 shows a possible set of jammed states according to some metric, and the MRJ state according to the same metric. Nevertheless, several studies in the literature still use the old terminology [98–101].

The concept of the "random close packing" state became popular because many algorithms for producing disordered packings of spheres in three dimensions could not proceed any further after reaching $\phi \approx 0.64$. However, Torquato *et al.* later demonstrated [95] that it was possible to produce packings with final densities varying continuously in the range $0.64 \lesssim \phi \lesssim 0.74$ by changing the growth rate of the particles in their packing algorithm. Packings with faster growth rates became jammed more quickly and had a higher degree of disorder than packings in which the particles grew

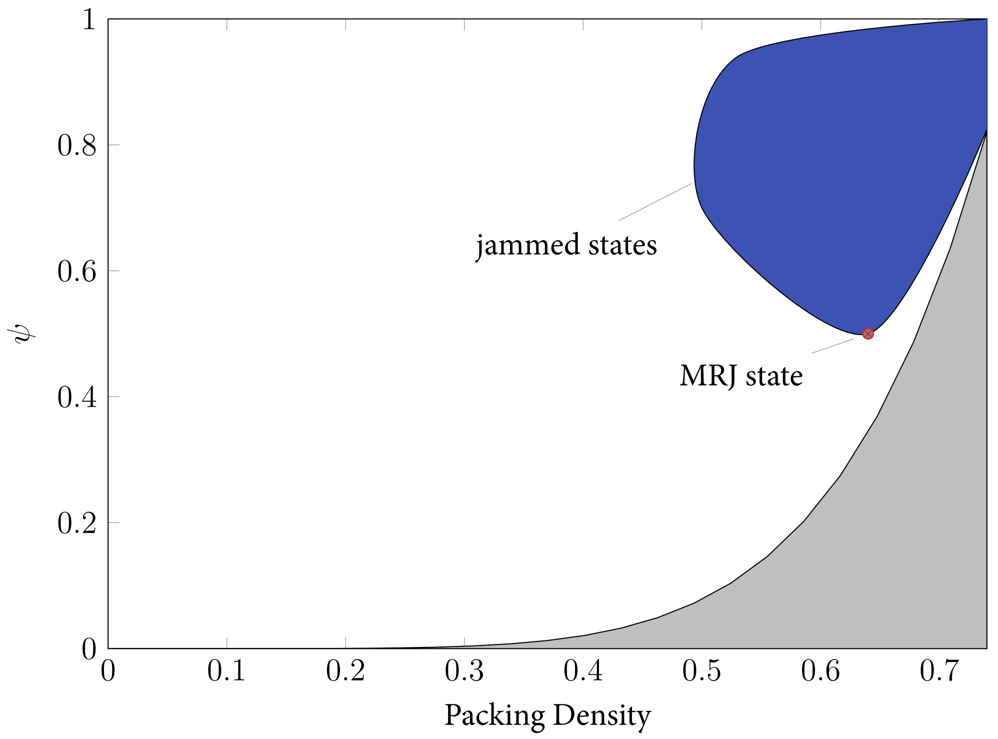


Figure 2.5: Maximally random jammed (MRJ) state, defined as the jammed state that minimizes the order metrics $\psi$. The area shaded in gray represents unreachable states.

more slowly. Torquato and Jiao [80] later described an improved, deterministic algorithm based on linear programming that could consistently produce states with a high degree of order with packing fractions in the range $0.64 \lesssim \phi \lesssim 0.74$. From the point of view of thermodynamics, congruent spheres undergo a first-order disordered–ordered phase transition near the packing fraction of the so called RCP state. Molecular dynamics simulations can reproduce this phase transition, although the freezing transition into a highly ordered state only occurs with a small probability. Decreasing the growth rate of the spheres can increase the probability of highly ordered final states, but several runs may need to be attempted before a lattice configuration is achieved. The most common outcome is that the simulation will proceed from the freezing point into a strictly jammed disordered state. A phase diagram for congruent spheres is sketched in figure 2.6, along with the possible paths that a simulation might take during densification. In molecular dynamics packing algorithms the collision frequency between particles diverges as the packing approaches a jammed state, so although theoretically possible, it would take an infinite time to generate a perfect lattice packing of spheres with hcp or fcc structure. It is also possible that the two types of lattice coexist in a packing, preventing the simulation from reaching $\phi = \pi/3\sqrt{2}$, but coming close to it. The case in which a single crystal with either hcp or fcc structure naturally forms during the simulation is indeed quite rare.

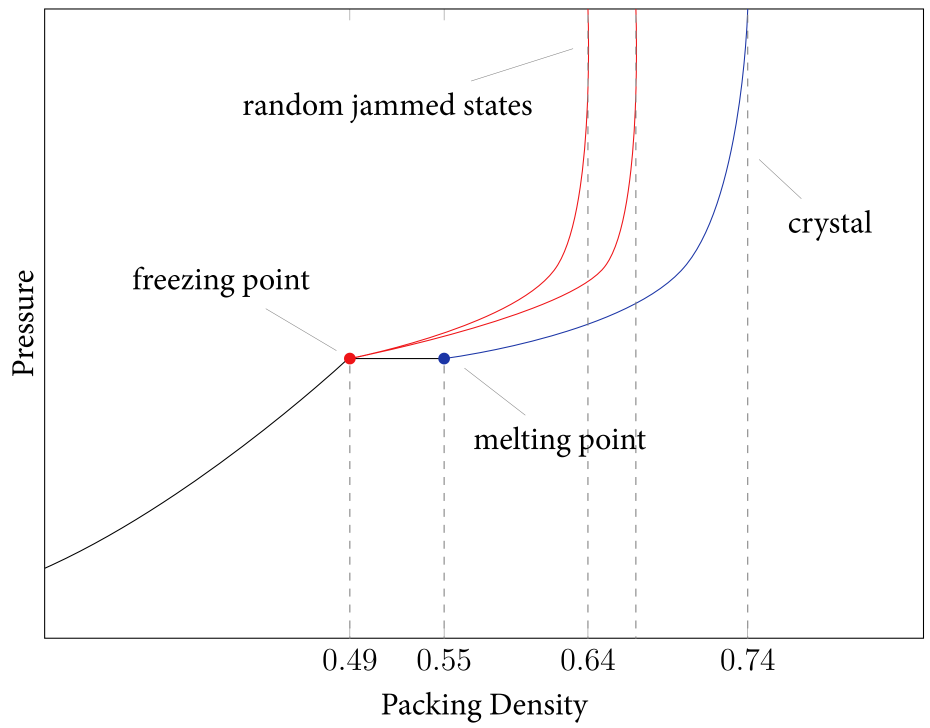


Figure 2.6: Phase diagram for a system of monodisperse identical spheres.

### 2.2.1 Order Metrics

Characterizing large particle packings according to the jamming categories introduced by Torquato and Stillinger [90] can be a challenge. The intention of the order metrics proposed by Truskett *et al.* [97] was to provide an intensive parameter by which to characterize particle packings, much like the temperature and pressure can be used to characterize gases. We have seen in the preceding sections that one of the problems with the RCP state is its ambiguity, since there is no quantification of randomness in its definition. The order metric solves this problem. Ideally, the order metric should provide a local measurement of randomness, such that it can be used to quantify randomness on individual jammed configurations. The reason to prefer a local metric rather than a global entropic metric is that the latter faces several practical hurdles to its implementation, since it requires the generation of all possible jammed states or, at least, a representative subset of them. Moreover, even if all states were given, the task of assigning weights to each of them is not a trivial one.

For a packing $\mathcal{P}$ with a configuration that can be fully described by the positions and orientations of its particles, i.e., $\mathcal{P} = \mathcal{P}(\vec{r}_1, \mathbf{q}_1, \ldots, \vec{r}_n, \mathbf{q}_n)$, the order metric function is constructed to possess the following three basic properties: (i) It is a well-defined scalar function of a packing configuration; (ii) It is normalized such that $0 \leq \psi \leq 1$; (iii) It provides a measure of order, such that for two packings $\mathcal{P}_A(\vec{r}_{A1}, \mathbf{q}_{A1}, \ldots, \vec{r}_{An}, \mathbf{q}_{An})$, and $\mathcal{P}_B(\vec{r}_{B1}, \mathbf{q}_{B1}, \ldots, \vec{r}_{Bn}, \mathbf{q}_{Bn})$, $\psi(\mathcal{P}_A) > \psi(\mathcal{P}_B)$ implies that the packing $\mathcal{P}_A$ is to be considered more ordered than $\mathcal{P}_B$.

There are many possible order metrics, according to which specific parameters one wishes to choose to characterize a particle packing. For packings of congruent spheres, two order metrics are quite common: the bond-orientational, and the translational order metrics.

### Bond-Orientational Order Metrics

The bond-orientational order $Q_\ell$ [102] is defined in terms of the spherical harmonics $Y_{\ell m}(\theta_i, \phi_i)$ in three dimensions, where $\theta_i$ and $\phi_i$ are the polar and azimuthal angles of the bond between a particle centered at the origin and its $i$-th near-neighbor particle. The coordinate system for the angles $\theta_i$ and $\phi_i$ can be either fixed by the boundary of the packing, or it can be carefully chosen to try to follow preferred directions in particle clusters (more information in section II of [102]). A particle is usually considered to be a near-neighbor if it falls under some pre-determined distance from the central particle, or if their Voronoi polyhedra share a common face. The value of $\ell$ to be used depends on the particular type of structure formed by the particles. For spheres, $\ell = 6$ is of particular interest, since $Q_6$ is maximum for hcp and fcc structures, as each layer of spheres in those structures is packed on a hexagonal lattice. The expression for $Q_6$ is then

$$Q_6 = \left( \frac{4\pi}{13} \sum_{m=-6}^{6} \left| \frac{1}{N_b} \sum_{i=1}^{N_b} Y_{6m}(\theta_i, \phi_i) \right|^2 \right)^{1/2},$$

where $N_b$ is the number of (near-neighbor) bonds for each particle. Averaging $Q_6$ over all particles in a packing provides a global measure of order for that packing. Since this average is maximum for spheres packed into a face-centered cubic configuration, the order metric $\psi_B$ can be defined as $\psi_B \equiv Q_6/Q_6^{\text{fcc}}$.

### Translational Order Metrics

The translational order metric [95, 97] is defined using the radial coordination structure of the fcc structure for spheres, by computing the mean occupation number of thin concentric shells around a central particle. Comparing the mean occupation numbers of the particles with both the fcc structure and an ideal gas at the same packing fraction, it is possible to define the function

$$T = \left| \frac{\sum_{i=1}^{N_{\text{shells}}} \left( n_i - n_i^{\text{ideal}} \right)}{\sum_{i=1}^{N_{\text{shells}}} \left( n_i^{\text{fcc}} - n_i^{\text{ideal}} \right)} \right| ,$$

where $n_i$ are the occupation numbers of each thin shell around the central particle, and $N_{\text{shells}}$ is the number of shells. Averaging this function over all particles then gives a different—but equally reasonable—global measure of order than the bond-orientational metrics.

# Chapter 3

# Characterization of Heterogeneous Materials

Each heterogeneous material has its distinctive microscopic structure with a characteristic length scale that is much larger than the molecular dimensions of the constituent materials, albeit much smaller than the typical length scale of a macroscopic sample. In the case of gels and fine powders, the characteristic length scale is on the order of a few nanometers, while in geological rock formations it can reach several meters. For solid propellants—the class of materials that we are interested in studying—the length scale of the microstructure is usually within a few hundred micrometers.

Heterogeneous materials can often be designed to have desirable physical and chemical properties. For example, a rocket designer can tune the proportions of fuel and oxidizer in a propellant to optimize the amount of thrust a rocket can produce. The mechanical properties of concrete can also be modified by the size distribution of rocks and sands in its composition. However, finding the right proportions of each component that optimizes a given physical or mechanical property of a heterogeneous material can be difficult. Often the only method available is to perform a series of experiments and to derive empirical formulas from the results. Alternatively, it may be possible to substitute experiments for simulations of the same physical and chemical processes on a virtual sample of material. That, in turn, leads to a different challenge—how to decide if a given model is a realistic representation of a certain material? The answer naturally depends on what material properties need to be predicted from the simulations. Several statistical descriptors are available [39] to quantitatively study different aspects of the microstructures of heterogeneous materials. In aluminum agglomeration in solid propellants, the size of fuel-rich pockets in between oxidizer particles is thought to play a major role in determining the final size of agglomerated particles [11, 14, 15].

Therefore, functions that can estimate pore-size between particles are crucial in order to study this phenomenon. On the other hand, in other processes the important characteristic of the material could be the shape and size distribution of the particles, the surface area between the phases, etc. In the following sections, we describe a small selection of the statistical descriptors used to characterize heterogeneous materials. A more detailed treatment of the subject can be found in [39], and [41].

## 3.1 *n*-Point Probability Functions

In our work, we make extensive use of *$n$-point probability functions* to quantify the statistics of microstructures. The $n$-point probability functions give the probability of simultaneously locating $n$ randomly chosen points each in a given material phase of the sample. Consider, for instance, the phase indicator function $\mathcal{I}_i(\vec{x}, n)$ defined as

$$\mathcal{I}_i(\vec{x}, n) = \begin{cases} 1 & \text{if } \vec{x} \in \mathcal{V}_i(n) \\ 0 & \text{otherwise,} \end{cases}$$

where $\vec{x}$ is a random point in the medium, $i$ is the label of one of the phases, and $\mathcal{V}_i(n)$ is the domain occupied by that phase in a given ensemble member $n$. The phases can be void space ($i = 0$), or a solid material ($i > 0$). Multiple particle sizes of a same material can also be subdivided and considered as different phases. This definition of $\mathcal{I}_i(\vec{x}, n)$ allows for the formulation of the $n$-point probability function $S_{i_1,i_2,\ldots,i_n}(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n})$, which gives the probability that the points $\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}$ be found respectively in phases $i_1, i_2, \ldots, i_n$ as

$$S_{i_1,i_2,\ldots,i_n}(\vec{x_1}, \vec{x_2}, \ldots, \vec{x_n}) = \overline{\mathcal{I}_{i_1}(\vec{x_1})\mathcal{I}_{i_2}(\vec{x_2})\ldots\mathcal{I}_{i_n}(\vec{x_n})},$$

where the bar denotes the ensemble average:

$$\overline{\mathcal{I}_i(\vec{x})} = \int_S \mathcal{I}_i(\vec{x}, n)\, p(n)\, \mathrm{d}n.$$

Here, $p(n)$ is the probability density of $n$ in the ensemble space $S$. The ensemble space represents a collection of a large number of tomographic scans and/or computer generated samples. In our work, we use one-, two- and three-point probability functions to describe material samples. The probability functions for a heterogeneous medium are spatially complex, but if the medium satisfies ergodicity, statistical isotropy, and statistical homogeneity, then the ensemble average can be replaced by the volume average and the determination of the statistics is significantly simplified. For example, the first-order or one-point probability function reduces to the volume fraction, while the higher-order probability functions depend only on the distance between the points and their angles.

### 3.1.1 Representative Volume Element (RVE)

Examining the two-point probability function is a convenient way for determining the size of a representative volume element (RVE) of a given material. When considering a random heterogeneous material, one would like to generate a model of the medium that is statistically equivalent to the physical medium, but with suitable dimensions as not to make simulations overly costly computationally. For instance, simulating a 1 cm$^3$ sample that contains $10^6$ particles will certainly require a lot of computational resources, while using only $10^2$ particles to represent the medium is clearly insufficient.

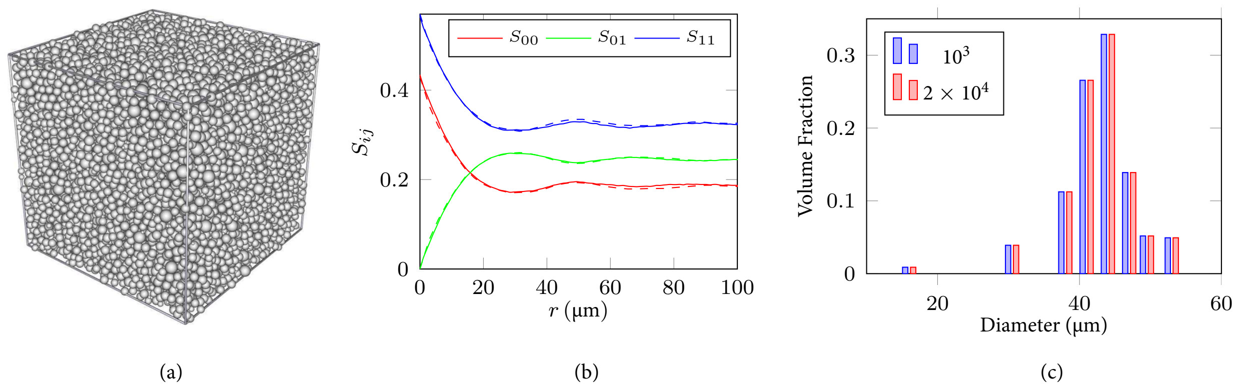


Figure 3.1: Representative volume element of a material sample: (a) polydisperse packing containing $2 \times 10^4$ spheres; (b) two-point probability functions $S_{ij}$ for two packings: $10^3$ particles (solid), and $2 \times 10^4$ particles (dash). Index 0 in $S_{ij}$ corresponds to the void region and index 1 corresponds to the solid region; (c) histogram of particle size distributions for both packings.

As an example, consider a polydisperse packing of spheres with diameters $D_i = 63.4$, 53.0, 49.5, 47.0, 44.0, 41.0, 38.0, 30.5, and 16.0 µm. This packing has a narrow distribution with a size ratio between the largest and the smallest particles that is close to 4. We consider two different packings, one with $2 \times 10^4$, and one with $10^3$ particles. Figure 3.1 shows: (a) the $2 \times 10^4$ particle packing; (b) the two-point probability functions for both packings; and (c) the size distribution of the particles. From 3.1(b) and 3.1(c) it can be seen that the two packings are statistically equivalent, at least for the one- and two-point probability functions. If one were to consider, say, a packing with $10^2$ particles, both the histogram and the two-point probability functions would be different.

Thus one can conclude that, in order to calculate the macroscopic properties of a material with this particle distribution, a packing with $10^3$ particles is sufficient. This way the computational burden can be reduced.

### 3.1.2 Ensemble Averaging

Ensemble averaging is necessary to account for the effect of orientations, spacing, etc., on the macroscopic properties. The ensemble average of a physical property $K$ and its standard deviation $\sigma_K$ are defined as

$$\langle K \rangle = \frac{1}{N}\sum_{n=1}^{N} K_n, \qquad \sigma_K = \left(\frac{1}{N}\sum_{n=1}^{N}(K_n - \langle K \rangle)^2\right)^{1/2} = \sqrt{\langle K^2 \rangle - \langle K \rangle^2},$$

where $N$ is the ensemble size. As pointed out in [103], when the RVE is sufficiently large, or—under the assumption of ergodicity—when a sufficiently large ensemble size is used for relatively smaller RVEs, the mean for the full ensemble should converge. Thus, in addition to matching higher order statistics between a much larger pack and a smaller RVE, matching the ensemble averages should also be a requirement.

## 3.2 Two-Point Cluster Function

The two-point cluster function $C_2^{(i)}(\vec{x_1}, \vec{x_2})$ is somewhat similar to the two-point probability function $S_2^{(i)}(\vec{x_1}, \vec{x_2})$, but it contains topological information about the connectedness of the material. It was introduced by Torquato, Beasley and Chiew [104], and it is defined as the probability that two points $\vec{x}_1$ and $\vec{x}_2$ both lie in the same cluster of phase $i$ of the material. A cluster is defined as any contiguous region of a given phase, i.e., a region in which any given point can be reached without passing through other phases. It can also be defined as a set of contacting particles in a particulate material.

Unlike the two-point probability function, which is usually short-ranged, the two-point cluster function becomes long-ranged when the percolation threshold is reached. Therefore, the two-point cluster function provides better information about the microstructural information of heterogeneous media. In general, for statistically inhomogeneous media, the standard two-point probability function can be decomposed into its connected and disconnected parts:

$$S_2^{(i)}(x_1, x_2) = C_2^{(i)}(\vec{x_1}, \vec{x_2}) + E_2^{(i)}(\vec{x_1}, \vec{x_2}),$$

where $E_2^{(i)}(\vec{x_1}, \vec{x_2})$ is defined as the probability that the two points $\vec{x}_1$ and $\vec{x}_2$ lie in different clusters of phase $i$ and is called the two-point blocking function.

The two-point cluster function provides important information about percolation effects in a material. When clusters that span the entire material sample first appear, the percolation threshold is reached and can cause interesting phenomena to happen. For example, the breakdown voltage of a heterogeneous material can strongly depend on the percolation threshold of the most conductive phase of the material [105].

## 3.3 Lineal-Path Function

The lineal-path function $L^{(i)}(z)$ [106] is defined as the probability that a line segment of length $z$ lies entirely within the material phase $i$ when placed at random into the sample. Since the probability that the segment will intersect a different phase increases with the length of the line segment, the lineal-path function is a monotonically decreasing function of $z$. For statistically homogeneous and isotropic media, the lineal-path function depends solely on $z$; for statistically homogeneous but anisotropic media, it is also a function of the orientation of the line segment $\vec{z}$; and for inhomogeneous media it depends on the absolute positions $\vec{x_1}$ and $\vec{x_2}$ of the end points of the segment as well, i.e. $L^{(i)}(z) = L^{(i)}(\vec{x_1}, \vec{x_2})$, where $\vec{z} = \vec{x_2} - \vec{x_1}$.

If we define $L^{(12)}(z)$ in a two-phase material as being the probability that the line segment intersects the interface between the two phases at any point, then

$$L^{(1)}(z) + L^{(2)}(z) + L^{(12)}(z) = 1,$$

since the line segment will either lie entirely on phase $1$, entirely on phase $2$, or will intersect the interface between the two phases. Moreover, if we shrink the segment $z$ to zero length, we can conclude that $L^{(i)}(0) = \varphi_i$, where $\varphi_i$ is the volume fraction of phase $i$. Using a similar argument, we can also conclude that if we stretch the line segment into infinity, the probability that it will lie entirely on a single phase will become very small, so $L^{(i)}(\infty) = 0$. Nevertheless, this second argument is only true for random materials that are also statistically homogeneous and isotropic. For a layered material, for example, $L^{(1)}(z) = \varphi_1$ and $L^{(2)}(z) = \varphi_2$ when $z$ is parallel to the layers, while when $z$ is perpendicular to the layers they will be proportional to the average layer thicknesses of each phase. Thus, lineal-path function provides a rough measure of the connectedness of the phases in a material. Analytic expressions for the lineal-path function for hard spheres are given in [106], but for non-trivial microstructures exact expressions for it become nearly intractable. The best approach in this case is to use Monte Carlo simulations.

## 3.4 Pair Correlation Function

The *pair correlation function* $g_2(\vec{r}_{12}, \mathbf{q}_1, \mathbf{q}_2)$ is proportional to the probability of finding the center of a particle with orientation $\mathbf{q}_2$ at position $\vec{r}_{12}$ relative to a reference particle with orientation $\mathbf{q}_1$. For simplicity, consider spherical particles in a statistically isotropic medium. In this case, the pair correlation function depends only on the magnitude of $\vec{r}_{12}$, that is,

$$g_2(\vec{r}_{12}, \mathbf{q}_1, \mathbf{q}_2) = g_2(r_{12}),$$

and it is usually referred to as the *radial distribution function*.

More generally, for a system of $N$ spheres confined to a finite region of volume $V$ in $\mathbb{R}^3$, it is possible to define an $n$-particle correlation function as

$$g_n(\vec{r}^n) = \frac{\rho_n(\vec{r}^n)}{\rho^n},$$

where $\rho_n(\vec{r}^n)$ is the *generic $n$-particle probability density function* given by

$$\rho_n(\vec{r}^n) = \frac{N!}{(N-n)!} \int P_N(\vec{r}^n) \mathrm{d}\vec{r}^{N-n},$$

and $\rho$ is simply the particle number density

$$\rho \equiv \lim_{N,V\to\infty} \frac{N}{V}.$$

The function $P_N(\vec{r}^n)$ above is the *specific particle probability density function* and is defined such that $P_N(\vec{r}^N)\mathrm{d}\vec{r}^N$ gives the probability of finding the center of particle 1 inside the volume element $\mathrm{d}\vec{r}^1$ about $\vec{r}_1$, the center of particle 2 inside the volume element $\mathrm{d}\vec{r}^2$ about $\vec{r}_2$, and so on. The volume element $\mathrm{d}\vec{r}^N$ is $\mathrm{d}\vec{r}^N = \mathrm{d}\vec{r}_1\mathrm{d}\vec{r}_2 \dots \mathrm{d}\vec{r}_N$. Being a probability density function, $P_N(\vec{r}^N)$ is normalized to unity, that is

$$\int P_N(\vec{r}^N)\mathrm{d}\vec{r}^N = 1.$$

If the system is not in thermal equilibrium, $P_N(\vec{r}^N)$ may also depend on time. The generic probability density function, despite its name, is not normalized to unity,

$$\int \rho_n(\vec{r}^n)\,\mathrm{d}\vec{r}^n = \frac{N!}{(N-n)!}.$$

It keeps this name because it can be trivially renormalized.

In disordered systems, $\rho_n(\vec{r}^n) \to \rho^n$ at large distances, thus $g_n(\vec{r}^n) \to 1$ as well. Therefore, $g_n(\vec{r}^n) - 1$ provides a measure of spatial correlation between the particles, with zero corresponding to no correlation.

## 3.5 Pore Size Distribution Function

The pore-size distribution function $P(\delta)$ was first defined in order to characterize the void or "pore" space in porous media [107]. Usually, $P_1(\delta)\,\mathrm{d}\delta$ gives the probability that a randomly chosen point inside phase 1 of a two-phase material lies at a distance in between $\delta$ and $\delta + \mathrm{d}\delta$ from the nearest point in the interface between the two phases. Similarly, $P_2(\delta)\,\mathrm{d}\delta$ would give the probability that randomly chosen points inside phase 2 lie at a distance in between $\delta$ and $\delta + \mathrm{d}\delta$ from the nearest point in the interface. However, since we are interested in using this function to characterize solid rocket propellants, we shall provide a definition of $P(\delta)$ that is more suitable for multi-phase materials. Consider, for example, a solid propellant that comprises three phases: ammonium perchlorate (AP), aluminum, and a polymer binder. We would like to know the size distribution of the fuel-rich pockets in between the large AP particles, since that is a key ingredient in models of aluminum agglomeration. Therefore, we can define $P_{AP}(\delta)\,\mathrm{d}\delta$ as the probability that a randomly chosen point that lies *outside* the AP particles lies at a distance in between $\delta$ and $\delta + \mathrm{d}\delta$ of the nearest AP particle.

The pore-size distribution function has dimensions of inverse length and satisfies

$$\int_0^\infty P(\delta)\,\mathrm{d}\delta = 1,$$

since it is a probability density function. It is always positive and at its extremes, $P(\delta)$ takes the values

$$P(0) = \frac{s}{\varphi_1} \quad \text{and} \quad P(\infty) = 0,$$

where $\frac{s}{\varphi_1}$ is the interfacial area per unit pore volume.

In three dimensions, one can think about $P(\delta)$ intuitively as the probability that a sphere of radius $\delta$ fits entirely within the pore space without overlapping with the material phase when placed at random. The pore-size distribution function also contains vague information about the connectedness of the material phase, since the pore size is also an estimate of the maximum distance that a random point can be from the material phase.

## 3.6 Computer Codes for Calculating Statistical Properties

The ability to analyze statistical properties of materials we simulate is crucial to understand the connection between their microstructure and their macroscopic properties. Therefore, as part of this research project, we developed a few auxiliary tools that currently cover a subset of the statistical descriptors discussed in the previous sections. The file format we have adopted for statistical analysis is the format produced by the tomography machines on which we have scanned samples of solid propellants. Samples produced by our packing code are converted to a binary mesh in this file format to allow a single statistical analysis tool to be used to process data from tomographic scans and simulations in the same way, regardless of their origin. This same tool can be used to create two-dimensional image slices of packings generated with the packing code.

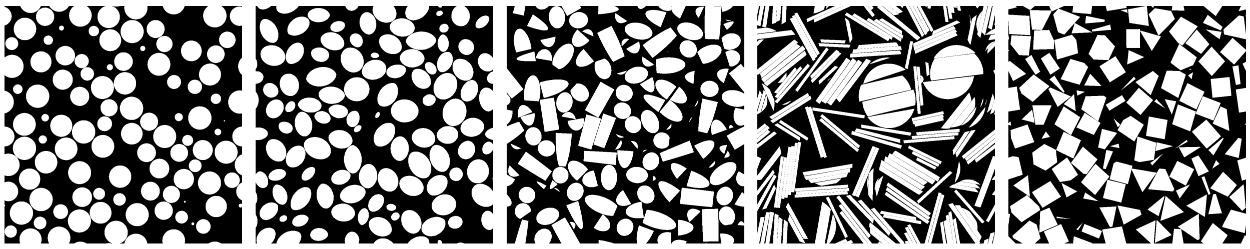
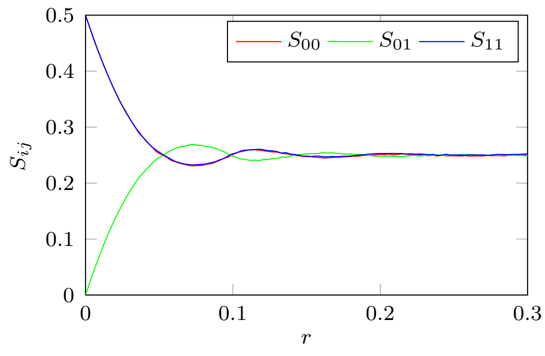

(a) Spheres (b) Ellipsoids (c) Cylinders (d) Disks (e) Cubes

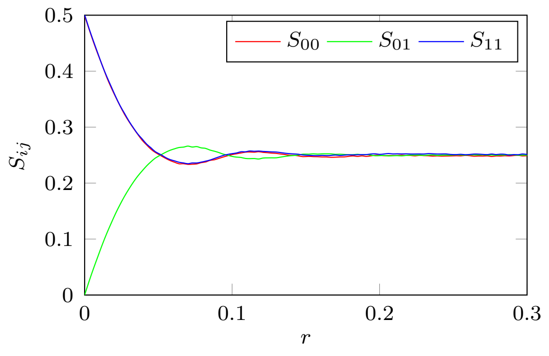
Figure 3.2: Cross-section images of packings of different convex objects, at $\phi = 0.5$.

Figure 3.2 shows a few examples of two-dimensional slices through packings of particles with different shapes, at a packing fraction of $\phi = 0.5$. Each shape can be easily recognized through its cross-sectional images. The slices allow a qualitative comparison with similar cuts through samples of real particulate materials; they roughly indicate how well the particle shapes used in the simulation reflect those of the particles in the real material.

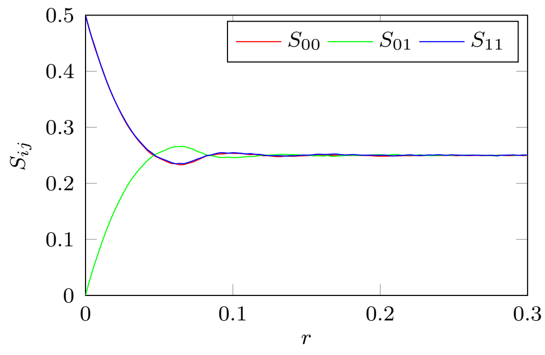
Once packings have been converted to the binary mesh format, statistical descriptors can be computed using other auxiliary tools. Figure 3.3 shows the two-point probability function for each of the examples in figure 3.2. Note that, at the same volume fraction, the scale of the particles does not quite match. The domain size in each case is a cubic periodic box of side 1. The distance $r$ between points is shown in absolute terms, and spans about a third of the domain. At this scale, the packings become statistically homogeneous—i.e., the 2-point probability becomes constant. Since packing fraction is about half of the total volume in all cases, all functions approach 0.25 when they become constant.



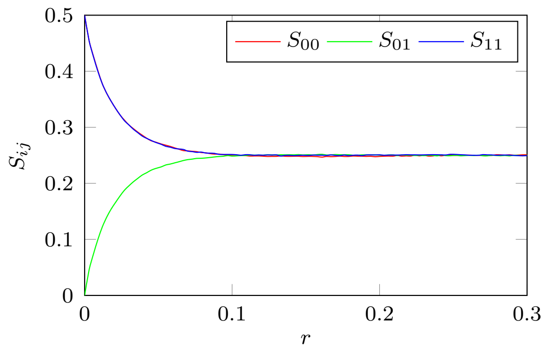
(a) Spheres



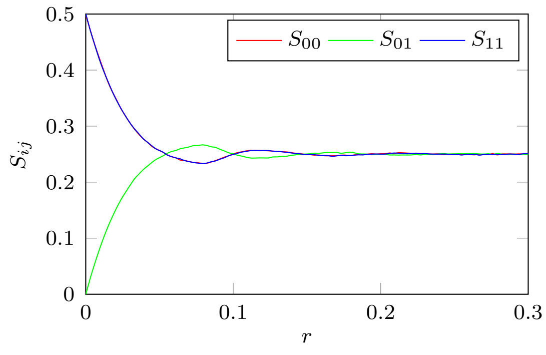
(b) Ellipsoids



(c) Cylinders



(d) Disks



(e) Cubes

Figure 3.3: Two-point probability functions for packings of different convex objects at $\phi = 0.5$.

We also have tools to compute other statistical functions from simulated samples, such as the radial distribution function, etc. More examples of statistical descriptors will be shown in later chapters.

# Chapter 4

# Rigid-Body Dynamics

## 4.1 Introduction

In this chapter we review some of the basic concepts of physics that are relevant in the analysis of the motion and interaction of rigid bodies. For the interested reader, the classical treatment of rigid body dynamics is the one given by Goldstein [108]. However, that text is somewhat on the heavy side in mathematics. Here, we will present a rather light discussion of the subject.

Newton's laws of motion are the foundation of classical mechanics. The three famous laws were first compiled by him in his *Philosophiæ Naturalis Principia Mathematica* (Mathematical Principles of Natural Philosophy), published in 1687 [109]. They are

**First Law** A body in an inertial reference frame is either at rest or moves at constant velocity, unless acted upon by an external force.

**Second Law** The acceleration of a body is directly proportional to, and in the same direction as the sum of the external forces acting upon it, and inversely proportional to its mass—that is, $\vec{F} = m\vec{a}$.

**Third Law** For every force exerted by a body upon a second body, there is a force, equal in magnitude and opposite in direction, that is felt by the first body as being exerted by the second body. This law is commonly known as the law of action and reaction.

In slightly more formal terms, the second law can be stated as

$$\vec{F}(t) = \frac{\partial \vec{p}}{\partial t} = \frac{\partial (m\vec{v})}{\partial t}.$$

We kept the time dependence of $\vec{F}$ intentionally as a reminder that the force can change dynamically, but we will drop the time dependence of $\vec{F}$ and other physical quantities to avoid cluttering the notation. When a force $\vec{F}$ acts upon a body over a finite time interval $\Delta t$, the change in momentum due to its action is called an *impulse*, and it is given by

$$\vec{J} = \Delta\vec{p} = \int_{\Delta t} \vec{F}\, \mathrm{d}t.$$

Impulses are used extensively in dynamic simulations of rigid bodies to prevent objects from interpenetrating.

Newton's laws of motion only describe the movement of *point masses*. In order to describe the movement of finite rigid bodies, we need to also take into account the orientation and rotational motion of the objects. One way of representing the orientation of an object is through an *orientation matrix* or *rotation matrix* $\mathbf{R}(t)$. The relationship between the rotation matrix and the angular velocity $\vec{\omega}(t)$ of the particle is the equation

$$\frac{\partial \mathbf{R}(t)}{\partial t} = \text{Skew}(\vec{\omega}(t))\, \mathbf{R}(t),$$

where

$$\text{Skew}(\vec{\omega}) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

is a skew-symmetric matrix derived from the angular velocity vector $\vec{\omega}$. The relationship between the angular velocity and the angular momentum $\vec{L}(t)$ is

$$\vec{L}(t) = \mathbf{I}(t)\, \vec{\omega}(t),$$

where $\mathbf{I}(t)$ is a tensor describing the moment of inertia of the particle, which is the angular analogue of mass. The inertia tensor is always constructed relative to a certain coordinate system. The angular analogue to Newton's second law of motion is the equation

$$\frac{\partial \vec{L}(t)}{\partial t} = \vec{\tau}(t),$$

where $\vec{\tau}(t)$ is the *torque* applied to the particle—it is the angular analogue of force. Since the particles can be moving and rotating, the inertia tensor does vary with time. However, instead of computing the inertia tensor and its inverse at each time step in a simulation, which could become very expensive depending on the shape of the object, it is usually constructed on the local coordinate system of the particle and a coordinate transformation is used to convert it into world coordinates. The transformation to convert the local inertia tensor into world coordinates is

$$\mathbf{I}(t) = \mathbf{R}(t)\, \mathbf{I}_{\text{body}}\, \mathbf{R}(t)^T,$$

where $\mathbf{I}(t)$ is the time-dependent inertia tensor in world coordinates and $\mathbf{I}_{\text{body}}$ is the local inertia tensor—which is constant, since the body is rigid.

## 4.2 Equations of Motion

The equation of motion for a particle—that is, a point mass—of mass $m$, with world position $\vec{x}$, world velocity $\vec{v} = \dot{\vec{x}}$, and world acceleration $\vec{a} = \dot{\vec{v}} = \ddot{\vec{x}}$ is simply given by Newton's second law:

$$\vec{F}(t) = m\vec{a} = m\dot{\vec{v}} = m\ddot{\vec{x}}.$$

If no external forces are present, i.e. $\vec{F} = 0$, then $\vec{a} = 0$ and this equation integrates to $\vec{v}(t) = \vec{v}_0$, a constant. The position of the particle then evolves as $\vec{r}(t) = \vec{r}_0 + t\vec{v}_0$, where $\vec{r}_0$ is the initial position of the particle. For a system of perfectly rigid bodies that interact only via penetration constraints, this equation is sufficient to describe the motion of their center of mass. The external forces are different than zero only when two objects collide. However, the constraint forces are not known *a priori* and must be calculated by the collision detection system at the time of each collision. The angular equations of motion for a rigid body without any external torques can be integrated in a similar manner. When implementing a system to simulate the dynamics of rigid bodies, all variables are usually grouped into a state vector

$$\vec{S}(t) = \begin{bmatrix} \vec{x}(t) \\ \vec{q}(t) \\ \vec{p}(t) \\ \vec{L}(t) \end{bmatrix},$$

and the total equations of motion become

$$\frac{\partial \vec{S}(t)}{\partial t} = \frac{\partial}{\partial t}\begin{bmatrix} \vec{x} \\ \vec{q} \\ \vec{p} \\ \vec{L} \end{bmatrix} = \begin{bmatrix} \dot{\vec{x}} \\ \dot{\vec{q}} \\ \dot{\vec{p}} \\ \dot{\vec{L}} \end{bmatrix} = \begin{bmatrix} \frac{\vec{p}}{m} \\ \frac{\vec{\omega}\vec{q}}{2} \\ \vec{F} \\ \vec{\tau} \end{bmatrix}.$$

The integration of the angular equations of motion should be carried out carefully, since over many time steps the numerical errors can accumulate and make the orientation matrix $\mathbf{R}(t)$ no longer orthonormal. One solution is to renormalize the matrix in regular intervals to avoid numerical problems, but the most common way to cope with errors is to represent rotations in a more numerically robust way, using *quaternions*. The angular equations of motion in terms of the unit-length quaternion $\vec{q}(t)$ corresponding to the orientation matrix $\mathbf{R}(t)$ is

$$\frac{\partial \vec{q}(t)}{\partial t} = \frac{1}{2}\vec{\omega}(t)\vec{q}(t),$$

where $\vec{\omega}(t)$ is now also a (not necessarily unit-length) quaternion that corresponds to the angular velocity.

## 4.3 Impulse-Based Collision Response

The constraints due to contact forces between perfectly rigid bodies are very simple. They are used to ensure that rigid bodies never interpenetrate. For a long time, the standard way of dealing with contact forces was to cast the nonpenetration constraints as a linear complementarity problem:

$$\begin{aligned} \vec{a} = \mathbf{A}\vec{f} - \vec{b} &\geq 0 \\ \vec{f} &\geq 0 \\ \vec{f} \cdot \vec{a} &= 0 \end{aligned}$$

In the equations above, $\vec{a}$ represents the relative normal acceleration at the contact point, and $\vec{f}$ is the normal component of the force at the point of contact. The matrix $\mathbf{A}$ and the vector $\vec{b}$ are constants which can be determined from the configuration of the system at the time of each collision. The constraint $\vec{a} \geq 0$ means that the two objects must either be moving apart or not moving, for if $\vec{a} < 0$ the objects are getting closer to each other and will penetrate if they are already in contact with each other. The constraint $\vec{f} \geq 0$ represents the fact that contact forces can only push objects apart, not pull them together. Finally, $\vec{f} \cdot \vec{a} = 0$ means that either the two objects are in contact (in which case $\vec{a} = 0$) and there is a force acting upon them to prevent penetration, or that they are moving apart with positive acceleration and the contact force between them is equal to zero. (It does not mean that the force and acceleration are perpendicular, since only one of them is nonzero at any given time.)

For frictionless systems, the linear complementarity problem always possesses a unique solution, given that certain nondegeneracy constraints are satisfied. When friction is added, the solution is not unique anymore, or may not exist altogether. A better model to satisfy the nonpenetration constraints is to apply instantaneous impulses to the particles at the time of collision. This approach became popular after the PhD works of David Baraff [110] and Brian Mirtich [111].

Let us then formulate how to resolve a colliding contact between two rigid bodies $A$ and $B$ by applying an impulse as done by Baraff and Mirtich. Let $t_0$ be the time of first impact between the rigid bodies $A$ and $B$ and $\mathcal{P}_0$ be the point of contact. For concave bodies it may not be possible to reduce the contact to a single point, but in our simulations only convex particles are involved in collisions so this assumption is reasonable. We will treat two types of contacts: vertex–face and edge–edge. Vertex–vertex contacts are very rare and can be ignored, and face–face contacts can be treated as one of the above. Whenever we have a vertex-face contact, we will adopt the convention that a vertex of body $A$ collides with a face of body $B$. Let $\vec{N}_0$ be the outward pointing unit normal perpendicular to the face of body $B$. If the contact type is edge–edge, let $\vec{N}_0$ be the normalized cross product between the colliding edges of $A$ and $B$ instead. In this case, we also choose $\vec{N}_0$ to point outside of body $B$. For a brief period of time that foregoes the collision, we can consider the trajectories of the points $\mathcal{P}_A(t)$ belonging to body $A$ and $\mathcal{P}_B(t)$ belonging to body $B$ for $t \leq t_0$, such that $\mathcal{P}_A(t_0) = \mathcal{P}_B(t_0) = \mathcal{P}_0$ at the time of collision. The normal vector of body $B$ at $\mathcal{P}_B(t)$ during this time interval is then $\vec{N}(t)$ and $\vec{N}(t_0) = \vec{N}_0$ according to our convention. The signed distance between the bodies, measured along the normal direction, is

$$D(t) = \vec{N}(t) \cdot (\mathcal{P}_A(t) - \mathcal{P}_B(t)\,)\,.$$

The time derivative of this equation gives the relative velocity between the bodies in the normal direction

$$\dot{D}(t) = \dot{\vec{N}}(t) \cdot (\mathcal{P}_A(t) - \mathcal{P}_B(t)\,) + \vec{N}(t) \cdot \left(\dot{\mathcal{P}}_A(t) - \dot{\mathcal{P}}_B(t)\right).$$

This equation is often (but not always) used to compute the time of impact between the two bodies.

At the time of impact $t_0$, we have

$$D(t_0) = 0, \qquad \dot{D}(t_0) = \vec{N}_0 \cdot \Big(\dot{\mathcal{P}}_A(t_0) - \dot{\mathcal{P}}_B(t_0)\Big),$$

where

$$\dot{\mathcal{P}}_A = \vec{v}_A + \vec{w}_A \times \vec{r}_A \quad \text{and} \quad \dot{\mathcal{P}}_B = \vec{v}_B + \vec{w}_B \times \vec{r}_B,$$

since $A$ and $B$ are both rigid bodies. In the equation above, $\vec{v}_A$ and $\vec{v}_B$ are the center of mass velocities of bodies $A$ and $B$, and $\vec{w}_A$ and $\vec{w}_B$ are their angular velocities with respect to their center of mass. The vectors $\vec{r}_A$ and $\vec{r}_B$ are simply $\vec{r}_A = \mathcal{P}_A - \mathcal{C}_A$ and $\vec{r}_B = \mathcal{P}_B - \mathcal{C}_B$, where $\mathcal{C}_A$ and $\mathcal{C}_B$ are the positions of the center of mass of each body. Therefore, at the time of collision, we have

$$\dot{D}(t_0) = \vec{N}_0 \cdot \left( (\vec{v}_A(t_0) + \vec{w}_A(t_0) \times \vec{r}_A(t_0)) - (\vec{v}_B(t_0) + \vec{w}_B(t_0) \times \vec{r}_B(t_0)) \right).$$

To prevent interpenetration of the pair of solid bodies for $t \geq t_0$ when $\dot{D}(t_0) < 0$, we must then apply a discontinuous change of velocity to them—in other words, an instantaneous impulse—that ensures that $\dot{D}(t_0) > 0$ afterwards. If $\vec{v}^-$ is the relative velocity before the impulse, then

$$\vec{v}^- = \vec{v}^\perp + (\vec{N} \cdot \vec{v}^-)\vec{N},$$

where we have dropped the index for the outer pointing normal $\vec{N}_0$, and $\vec{v}^\perp$ is the component of $\vec{v}^-$ that is perpendicular to $\vec{N}$. The post-impulse velocity $\vec{v}^+$ is then selected to be

$$\vec{v}^+ = \vec{v}^\perp - (\vec{N} \cdot \vec{v}^-)\vec{N},$$

which is simply a reflection of the normal component of the relative velocity, meaning that there is no loss of kinetic energy during the collision. A coefficient of restitution $\varepsilon$ such that

$$\vec{v}^+ = \vec{v}^\perp - \varepsilon(\vec{N} \cdot \vec{v}^-)\vec{N}$$

can be introduced to generate collision responses in which part of the kinetic energy is lost. That can be useful to keep the temperature of the particle system down during a molecular dynamics simulation. Let then $\dot{\mathcal{P}}_A^\pm$ and $\dot{\mathcal{P}}_B^\pm$ be the preimpulse and postimpulse local velocities of the points in $A$ and $B$ that come into contact at $t_0$, respectively. Then we have

$$\dot{\mathcal{P}}_A^\pm = \vec{v}_A^\pm + \vec{w}_A^\pm \times \vec{r}_A \quad \text{and} \quad \dot{\mathcal{P}}_B^\pm = \vec{v}_B^\pm + \vec{w}_B^\pm \times \vec{r}_B.$$

Now we need to find out the impulse magnitude such that $\vec{v}^{+} = \vec{v}^{\perp} - \varepsilon(\vec{N} \cdot \vec{v}^{-})\vec{N}$ after the impulse is applied. Since the impulsive force is only in the normal direction in our case (i.e. no friction), $\vec{F} = f\vec{N}$, where $f$ is the magnitude we need to determine. The changes in linear and angular momentum are then given by

$$
\begin{aligned}
\vec{p}_A^{+} &= \vec{p}_A^{-} + f\vec{N} & \qquad \vec{p}_B^{+} &= \vec{p}_B^{-} - f\vec{N} \\
\vec{L}_A^{+} &= \vec{L}_A^{-} + \vec{r}_A \times f\vec{N} & \qquad \vec{L}_B^{+} &= \vec{L}_B^{-} - \vec{r}_B \times f\vec{N}
\end{aligned}
$$

and the respective changes in the linear and angular velocities are

$$
\begin{aligned}
\vec{v}_A^{+} &= \vec{v}_A^{-} + \frac{f\vec{N}}{m_A} & \qquad \vec{v}_B^{+} &= \vec{v}_B^{-} - \frac{f\vec{N}}{m_B} \\
\vec{\omega}_A^{+} &= \vec{\omega}_A^{-} + \mathbf{I}_A^{-1}(\vec{r}_A \times f\vec{N}) & \qquad \vec{\omega}_B^{+} &= \vec{\omega}_B^{-} - \mathbf{I}_B^{-1}(\vec{r}_B \times f\vec{N})
\end{aligned}
$$

where $\mathbf{I}_A$ and $\mathbf{I}_B$ are the moments of inertia of body $A$ and $B$, respectively.

The post impulse local velocities of body $A$ and $B$ at the point of contact are then

$$
\begin{aligned}
\dot{\mathcal{P}}_A^{+} &= \dot{\mathcal{P}}_A^{-} + f\left(\frac{\vec{N}}{m_A} + \mathbf{I}_A^{-1}(\vec{r}_A \times \vec{N}) \times \vec{r}_A\right) \\
\dot{\mathcal{P}}_B^{+} &= \dot{\mathcal{P}}_B^{-} - f\left(\frac{\vec{N}}{m_B} + \mathbf{I}_B^{-1}(\vec{r}_B \times \vec{N}) \times \vec{r}_B\right)
\end{aligned}
$$

Note that the sign is different for body $B$ because the impulse on $B$ is $-\vec{F}$. After some algebra, we obtain the following expression for the impulse magnitude:

$$
f = \frac{-(1+\varepsilon)\left(\vec{N} \cdot (\vec{v}_A^{-} - \vec{v}_B^{-}) + \vec{w}_A^{-} \cdot (\vec{r}_A \times \vec{N}) - \vec{w}_B^{-} \cdot (\vec{r}_B \times \vec{N})\right)}{m_A^{-1} + m_B^{-1} + (\vec{r}_A \times \vec{N})^T \mathbf{I}_A^{-1}(\vec{r}_A \times \vec{N}) + (\vec{r}_B \times \vec{N})^T \mathbf{I}_B^{-1}(\vec{r}_B \times \vec{N})}.
$$

The right-hand side of the equation depends only on the configuration of the bodies $A$ and $B$ at the time of collision, their moments of inertia, and the restitution coefficient $\varepsilon$—all known quantities. The impulse magnitude $f$ is always positive: the denominator is positive because the mass of each body is positive and the inverse of the moments of inertia are positive definite; the numerator is positive because the negative term $-(1+\varepsilon)$ is multiplying the local relative velocity at the point of contact, which is also negative since the objects are moving towards each other.

## 4.4 Moments of Inertia of Solid Polyhedra

The moment of inertia of a solid body is a measure of the rotational inertia of the body about an axis. When two bodies collide, the moment of inertia is necessary to calculate the magnitude of the impulse needed in the collision response. As described earlier, the moment of inertia depends on the coordinate system adopted. Nevertheless, it is possible to compute the moment of inertia once in the local coordinates of the solid body and transform it into global coordinates when needed. For a continuum of mass in three dimensions, the inertia tensor is a matrix given by

$$\mathbf{I} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix},$$

where

$$I_{xx} = \int_V y^2 + z^2 \, \mathrm{d}m, \qquad I_{yy} = \int_V x^2 + z^2 \, \mathrm{d}m, \qquad I_{zz} = \int_V x^2 + y^2 \, \mathrm{d}m$$

and

$$I_{xy} = \int_V xy \, \mathrm{d}m, \qquad I_{xz} = \int_V xz \, \mathrm{d}m, \qquad I_{yz} = \int_V yz \, \mathrm{d}m,$$

where $V$ denotes the volume of the solid body.

In our code, we need to compute the moments of inertia of general polyhedra defined by the user in the input and configuration files. If the density is constant inside the polyhedron—an assumption we make in our code—it is possible to reduce the volume integrals to surface integrals using the divergence theorem and then further reduce those integrals to line integrals using Green's theorem. The result is that the moments of inertia of the polyhedron become much easier to evaluate. This method was first described by Mirtich [111] in his PhD thesis, but can also be found on textbooks [92]. In order to compute the moment of inertia of the polyhedron, many integrals of the form

$$\int_V p(x,y,z) \, \mathrm{d}V$$

need to be evaluated, where $p(x,y,z)$ is one of the following polynomials: $1$, $x$, $y$, $z$, $x^2$, $y^2$, $z^2$, $xy$, $xz$, and $yz$. For each of these polynomials, we then choose a function $\vec{F}$ from table 4.1 such that $\nabla \cdot \vec{F} = p(x,y,z)$ so that

$$\int_V p(x,y,z) \, \mathrm{d}V = \int_V \nabla \cdot \vec{F} \, \mathrm{d}V = \int_S \vec{N} \cdot \vec{F} \, \mathrm{d}S,$$

Table 4.1: Vector fields $\vec{F}$ such that $p = \nabla \cdot \vec{F}$.

| $p(x, y, z)$ | $\vec{F}(x, y, z)$ |
|---|---|
| $1$ | $(x, 0, 0)$ |
| $x$ | $(x^2/2, 0, 0)$ |
| $y$ | $(0, y^2/2, 0)$ |
| $z$ | $(0, 0, z^2/2)$ |
| $x^2$ | $(x^3/3, 0, 0)$ |
| $y^2$ | $(0, y^3/3, 0)$ |
| $z^2$ | $(0, 0, z^3/3)$ |
| $xy$ | $(x^2y/2, 0, 0)$ |
| $xz$ | $(0, 0, z^2x/2)$ |
| $yz$ | $(0, y^2z/2, 0)$ |

where $S$ is the surface of the polyhedron, and $\vec{N}$ is the outward unit-length normal. Considering that the surface of the polyhedron can be decomposed into $n$ faces $\mathcal{F}_1, \ldots, \mathcal{F}_n$, the integral of $\vec{F}(x, y, z)$ over $S$ becomes

$$\int_S \vec{N} \cdot \vec{F} \, \mathrm{d}S = \sum_{i=1}^{n} \int_{\mathcal{F}_i} \vec{N}_{\mathcal{F}_i} \cdot \vec{F} \, \mathrm{d}S,$$

where $\vec{N}_{\mathcal{F}_i} = (\eta_x, \eta_y, \eta_z)$ is the constant outward normal of face $\mathcal{F}_i$.

We are then left with the following integrals to compute:

$$\begin{aligned}
\int_V 1 \, \mathrm{d}V &= \sum_{i=1}^{n} \eta_x \int_{\mathcal{F}_i} x \, \mathrm{d}S & \int_V xy \, \mathrm{d}V &= \frac{1}{2} \sum_{i=1}^{n} \eta_x \int_{\mathcal{F}_i} x^2 y \, \mathrm{d}S \\
\int_V xz \, \mathrm{d}V &= \frac{1}{2} \sum_{i=1}^{n} \eta_z \int_{\mathcal{F}_i} z^2 x \, \mathrm{d}S & \int_V yz \, \mathrm{d}V &= \frac{1}{2} \sum_{i=1}^{n} \eta_y \int_{\mathcal{F}_i} y^2 z \, \mathrm{d}S \\
\int_V x \, \mathrm{d}V &= \frac{1}{2} \sum_{i=1}^{n} \eta_x \int_{\mathcal{F}_i} x^2 \, \mathrm{d}S & \int_V x^2 \, \mathrm{d}V &= \frac{1}{3} \sum_{i=1}^{n} \eta_x \int_{\mathcal{F}_i} x^3 \, \mathrm{d}S \\
\int_V y \, \mathrm{d}V &= \frac{1}{2} \sum_{i=1}^{n} \eta_y \int_{\mathcal{F}_i} y^2 \, \mathrm{d}S & \int_V y^2 \, \mathrm{d}V &= \frac{1}{3} \sum_{i=1}^{n} \eta_y \int_{\mathcal{F}_i} y^3 \, \mathrm{d}S \\
\int_V z \, \mathrm{d}V &= \frac{1}{2} \sum_{i=1}^{n} \eta_z \int_{\mathcal{F}_i} z^2 \, \mathrm{d}S & \int_V z^2 \, \mathrm{d}V &= \frac{1}{3} \sum_{i=1}^{n} \eta_z \int_{\mathcal{F}_i} z^3 \, \mathrm{d}S.
\end{aligned}$$

These integrals all assume the form

$$\eta_\ell \int_{\mathcal{F}_i} f(x, y, z) \, \mathrm{d}S,$$

where $\ell$ is either $x$, $y$, or $z$, and $f(x, y, z)$ is a polynomial, as given above. It is possible to convert these integrals into line integrals on the boundary of each face of the polyhedron, as is done by Mirtich [111], but the most convenient way of calculating these integrals is to break up each polygonal face into triangles. The integrals can then be evaluated by direct parametrization of the triangles. For instance, consider a triangular face with vertices $\mathbf{V}_i = (x_i, y_i, z_i)$, $0 \leq i \leq 2$, ordered counterclockwise. Also, consider the edges $E_1$ and $E_2$ given by

$$\vec{E}_i = \mathbf{V}_i - \mathbf{V}_0 = (x_i - x_0, y_i - y_0, z_i - z_0) \equiv (\alpha_i, \beta_i, \gamma_i).$$

That gives the following parametrization for the triangular face

$$\begin{aligned}\mathbf{P}(u, v) &= \mathbf{V}_0 + u\vec{E}_1 + v\vec{E}_2 \\ &= (x_0 + \alpha_1 u + \alpha_2 v, y_0 + \beta_1 u + \beta_2 v, z_0 + \gamma_1 u + \gamma_2 v) \\ &= (x(u, v), y(u, v), z(u, v))\end{aligned}$$

where $u$ and $v$ satisfy $0 \leq u, v$ and $u + v \leq 1$.

The outward normal of the triangular face can be simply given by the cross product of the two edges

$$\vec{N}_{\mathcal{F}} = \frac{\vec{E}_1 \times \vec{E}_2}{|\vec{E}_1 \times \vec{E}_2|} = \frac{(\beta_1\gamma_2 - \beta_2\gamma_1, \alpha_2\gamma_1 - \alpha_1\gamma_2, \alpha_1\beta_2 - \alpha_2\beta_1)}{|\vec{E}_1 \times \vec{E}_2|} \equiv \frac{(\delta_0, \delta_1, \delta_2)}{|\vec{E}_1 \times \vec{E}_2|}.$$

Now, the integrals are reduced to the following form

$$\left(\vec{N}_{\mathcal{F}} \cdot \vec{\ell}\right) \int_{\mathcal{F}} f(x, y, z)\, \mathrm{d}S = \left(\left(\vec{E}_1 \times \vec{E}_2\right) \cdot \vec{\ell}\right) \int_0^1 \int_0^{1-v} f\left(x(u, v), y(u, v), z(u, v)\right) \mathrm{d}u\mathrm{d}v,$$

where $\mathrm{d}S = |\vec{E}_1 \times \vec{E}_2|\, \mathrm{d}u\mathrm{d}v$.

The final results are then

$$
\begin{aligned}
\left(\vec{N}_{\mathcal{F}} \cdot \vec{i}\right) \int_{\mathcal{F}} x \,\mathrm{d}S &= \frac{\delta_0}{6}\, f_1(x) \\
\left(\vec{N}_{\mathcal{F}} \cdot \vec{i}\right) \int_{\mathcal{F}} x^2 \,\mathrm{d}S &= \frac{\delta_0}{12}\, f_2(x) \\
\left(\vec{N}_{\mathcal{F}} \cdot \vec{i}\right) \int_{\mathcal{F}} x^3 \,\mathrm{d}S &= \frac{\delta_0}{20}\, f_3(x) \\
\left(\vec{N}_{\mathcal{F}} \cdot \vec{j}\right) \int_{\mathcal{F}} y^2 \,\mathrm{d}S &= \frac{\delta_1}{12}\, f_2(y) \\
\left(\vec{N}_{\mathcal{F}} \cdot \vec{j}\right) \int_{\mathcal{F}} y^3 \,\mathrm{d}S &= \frac{\delta_1}{20}\, f_3(y) \\
\left(\vec{N}_{\mathcal{F}} \cdot \vec{k}\right) \int_{\mathcal{F}} z^2 \,\mathrm{d}S &= \frac{\delta_2}{12}\, f_2(z) \\
\left(\vec{N}_{\mathcal{F}} \cdot \vec{k}\right) \int_{\mathcal{F}} z^3 \,\mathrm{d}S &= \frac{\delta_2}{20}\, f_3(z) \\
\left(\vec{N}_{\mathcal{F}} \cdot \vec{i}\right) \int_{\mathcal{F}} x^2 y \,\mathrm{d}S &= \frac{\delta_0}{60}\, \left(y_0 g_0(x) + y_1 g_1(x) + y_2 g_2(x)\right) \\
\left(\vec{N}_{\mathcal{F}} \cdot \vec{j}\right) \int_{\mathcal{F}} y^2 z \,\mathrm{d}S &= \frac{\delta_1}{60}\, \left(z_0 g_0(y) + z_1 g_1(y) + z_2 g_2(y)\right) \\
\left(\vec{N}_{\mathcal{F}} \cdot \vec{k}\right) \int_{\mathcal{F}} z^2 x \,\mathrm{d}S &= \frac{\delta_2}{60}\, \left(x_0 g_0(z) + x_1 g_1(z) + x_2 g_2(z)\right)
\end{aligned}
$$

where $f_i(\lambda)$ and $g_i(\lambda)$ are

$$
\begin{aligned}
f_1(\lambda) &= \lambda_0 + \lambda_1 + \lambda_2 \\
f_2(\lambda) &= \lambda_0^2 + \lambda_0\lambda_1 + \lambda_1^2 + \lambda_2 f_1(\lambda) \\
f_3(\lambda) &= \lambda_0^3 + \lambda_0^2\lambda_1 + \lambda_0\lambda_1^2 + \lambda_1^3 + \lambda_2 f_2(\lambda) \\
g_i(\lambda) &= f_2(\lambda) + \lambda_i f_1(\lambda) + \lambda_i.
\end{aligned}
$$

In $f_i(\lambda)$ and $g_i(\lambda)$, $\lambda$ is just an alias for either $x$, $y$, or $z$ and the indexed quantities correspond to the vertex coordinates of the triangular faces.

# Chapter 5

# Collision Detection

Collision detection is a vast subject with many applications in various areas, such as robotics, virtual reality, computer games and animation [92, 112–114], and physics-based simulations in engineering [115], to name a few. In computer games, collision detection is used, for example, to determine which objects need to be drawn on screen, to perform line-of-sight and intersection queries between virtual world objects and to prevent players from walking through walls, other players, etc. In robotics, collision detection helps robots steer away from obstacles that lie along their trajectories. In engineering, collision detection is used in simulations of car crashes to aid in the development of better structural components without the need to destroy real (and expensive) cars in the process. In movies, clothing is simulated as a large mesh with many triangles and collision detection is used to ensure that there are no self-intersections and that the clothes behave in a life-like manner.

The performance requirements of a collision detection system depends on the type of application for which it is built. In computer animation (movies), the performance of the collision detection system is not critical, since the time spent in collision detection is but a fraction of the time spent rendering each frame. In games, however, the need to be interactive and run in real-time imposes rather tight restrictions on the time that can be spent detecting collisions. To put this into perspective, we can consider that, to run smoothly, a game must run at least at 60 fps (frames per second). At this frame rate, each frame must be ready for display in 16.66 ms—not that much time. Collision detection can usually only account for about 20% to 30% of each frame in order to leave enough time for game logic, input, network communication, rendering, etc. That leaves roughly 3–5 ms for collision detection. Moreover, each frame can contain hundreds of objects, of which at least a dozen will need to be tested for collisions after a collision culling phase. This means that in the end there may be only 30 to 200 μs available for collision detection per object pair—very little time indeed. The collision detection system in a packing code does not have such tight restrictions on time, since it does not need to run interactively. Nevertheless, performance is still critical to keep overall packing time reasonably low, since packings may contain many thousands of particles.

Robustness requirements also vary, and there is naturally a trade off between robustness and performance in every collision detection system. In computer games, the severe restrictions on time make it necessary to favor performance over everything else. However, there are no major drawbacks in doing this—in a game, it is more than enough for things to just look right, so small intersections between world objects can be tolerated. Most games employ simplified geometrical models for collision detection—e.g. by substituting a player with a cylinder or a box—and reserve the more sophisticated models for rendering only. In a packing code, correctness is very important—penetration between particles is often unacceptable. Therefore, in this context, it is preferable to use continuous rather than discrete algorithms for collision detection, even though it means that some performance must be sacrificed in the process.

Collision detection queries can be broadly categorized into three different types, concerning the problems of *if*, *when*, and *where* two objects come into contact. The first type of query is the simplest and cheapest in terms of computational cost—it consists in determining a Boolean result indicating whether or not two objects intersect. This is a static query, since both objects are immobile. Determining *when* two objects collide based on their movement is a dynamic query. It can be determined by interval bisection if the two objects intersect at the end of the interval but not at the beginning, but in general more sophisticated algorithms are necessary in order to ensure that no collisions are missed. Establishing *where* two objects come into contact is the most complex query, and also the hardest to make numerically robust. A related problem to finding where two objects intersect is the problem of finding their penetration depth once it is established that they intersect by a finite amount—that is, finding the smallest translation vector that would eliminate the intersection. In our packing code, however, penetration between particles is not allowed.

Our choice of collision detection algorithm is based on the characteristics of the particulate materials that we intend to model with our packing code. In general, the particles are convex, and their shape is either spherical or polyhedral (for the explosive crystals). Most explosives used in solid rocket propellants are ground and milled, so elongated particles are uncommon. Hence, oxidizer particles are represented in our code with either spheres, ellipsoids, or convex polyhedra defined in external files. Many algorithms exist for detecting collisions between convex solid objects. In summary, we need to use algorithms that work in general for any convex shape, and are both robust and as fast as possible. We have not found the algorithms in the particle packing literature to be particularly appealing in terms of performance. Torquato *et al.* [51, 52] use an algorithm based on the separating axis theorem. They state that "the [separating] axis is either perpendicular to one of the faces or perpendicular to a pair of edges, one from each polyhedron. This reduces the number of axes that need to be checked from infinity to $[E(E-1)/2 + 2F]$, where $E$ and $F$ are the number of edges and faces of the polyhedron, respectively." However, testing all of these possibilities can become very expensive as the number of vertices of the polyhedra increases. On the other hand, in a previous code developed for our own group by D. S. Stafford [53, 116], the algorithm for computing particle intersections was based on the minimization of a level set function representing the particle shapes. In addition to the difficulties in minimizing functions with sharp features, that method ran very slowly due to the iterative nature of the algorithm and the large number of iterations needed to converge to the minimum with confidence. We wanted to address the shortcomings of the level set algorithm in our new code in order to enable us to simulate larger systems, so we turned our attention to algorithms used in computer games, where performance is one of the most important aspects. We decided to use the Gilbert–Johnson–Keerthi (GJK) algorithm for polyhedra and specialized algorithms for other shapes. These algorithms will be described in the next sections.

## 5.1 Bounding Volumes

The computational cost of collision queries of any type is naturally proportional to the complexity of the objects being tested. Therefore, bounding volumes are used to speed up the process. The simplest and most convenient type of bounding volume is a sphere. Since oxidizer particles in most solid propellants are roughly spherical, this is a suitable choice of bounding volume for our code. Elongated particles are still supported, albeit with a performance penalty if compared to particles that are close to being spherical. Other obvious choices for bounding volumes would be axis-aligned bounding boxes (AABBs) and object-aligned or oriented bounding boxes (OBBs). We provide an implementation of AABBs with our code, even though only spherical bounding volumes are currently used. Spheres have the added convenience of being rotationally invariant while AABBs must be recomputed at every collision check. The implementation of AABBs may be used at a later time for other applications if necessary. Figure 5.1 shows a list of bounding volumes from the simplest (left) to most complex (right). An in-depth discussion of bounding volumes is out of the scope of this work. For the interested reader, we recommend the texts by Ericson [114] and Eberly [92, 113].

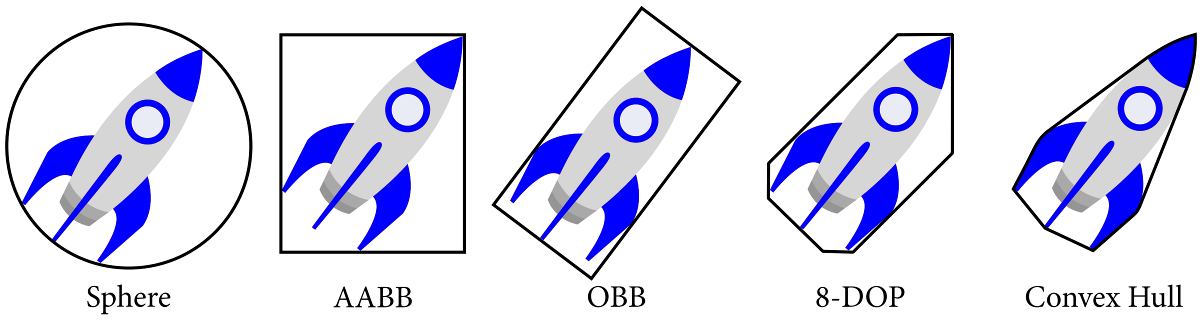


Figure 5.1: Types of bounding volumes: sphere, axis-aligned bounding box (AABB), oriented bounding box (OBB), eight-direction discrete orientation polytope (8-DOP), and convex hull.

### 5.1.1 Computing a Bounding Sphere

The bounding sphere of a given set of points can be calculated in various ways. One of the simplest methods is to use the average of all points as the center of the sphere and use the furthest point from this center to compute the radius of the bounding sphere. This is a very fast method, but it can create spheres that are larger than needed for particles with asymmetrical shapes, like a diamond or a pyramid. In our code, we use the center of mass of the polyhedron instead of the average of the vertices. The bounding radius is then computed from the vertex that is furthest away from the center of mass. This is better than using a simple average, but it is by no means the best method of computing the bounding sphere. Nevertheless, our method has important advantages over methods that provide tighter bounding spheres: the center of mass is shared with the polyhedron, so it saves some memory that would be needed to store the center of the bounding sphere, and the bounding sphere does not need to be adjusted as the particle rotates, which also helps to simplify bounding volume intersection tests. A possible alternative would be to use the OBB of the particle shape and take its midpoint in each direction as the center of the sphere and calculate the radius from the furthest point. Methods also exist for computing a tight bounding sphere of a set of points using the direction of maximum spread, and for computing an optimal bounding sphere using a brute force method [114]. However, the improvements offered by these methods if applied to our code are only marginal, since in the vast majority of cases the particles we use are either symmetric or roughly spherical, and our method produces tight bounding spheres in those cases.

### 5.1.2 Collision Detection for Growing Spheres

Whenever a particle in the system undergoes a collision, it is necessary to update its state and schedule a new event for it in the event queue. After the collision impulse is applied, the particle is then tested against all of its neighbors—as determined by the spatial partitioning scheme—to decide which of them has the earliest collision time. However, in order to save time, the full collision test is only performed if their bounding spheres are predicted to intersect in the future. Since the particles grow linearly with time, we need a method for testing whether two moving and growing spheres will intersect.

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be the centers of spheres 1 and 2 at $t = 0$, and $\vec{v_1}$ and $\vec{v_2}$ be their velocities, respectively. Additionally, let $r_1(t) = g_1 t$ and $r_2(t) = g_2 t$ be the time-dependent radii of the two spheres, where $g_1$ and $g_2$ are their growth rates. Then, at a given time $t$, the spheres intersect if

$$|(\mathcal{C}_1 + \vec{v_1}t) - (\mathcal{C}_2 + \vec{v_2}t)| < r_1(t) + r_2(t).$$

Therefore, if two moving particles whose bounding spheres are spheres 1 and 2 will collide in the future, the collision must happen within the time interval given by the roots of the quadratic equation

$$|(\mathcal{C}_1 + \vec{v_1}t) - (\mathcal{C}_2 + \vec{v_2}t)|^2 = (r_1(t) + r_2(t))^2.$$

If no roots are found, the particle pair can be safely discarded as not colliding. Otherwise, a full collision test must be performed within the interval to determine if there is indeed a collision.

The method above may seem simple, but a lot of care is needed in order to avoid catastrophic errors caused by floating point arithmetic, among other things. The first thing we should notice is that when the two particles are actually spheres, the test above is the only test we perform to determine if they will collide (and when) or not. If there are large numerical errors in the calculation of their collision time, the code may get stuck on an infinite loop by predicting that the same collision will happen over and over again. Therefore, we cannot simply apply the quadratic formula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ to solve the equation $ax^2 + bx + c = 0$ with the appropriate coefficients. The reason is simple—when two spheres are close, $c$ is small, which leads to large numerical cancellation errors when subtracting $b$ from the other term in the numerator, since $\sqrt{b^2 - 4ac} \approx b$. In addition, if the sum of the growth rates of the spheres is close to their separating velocity, i.e. $g_1 + g_2 \approx |v_1 - v_2|$, then $a \approx 0$ and the division by $2a$ can significantly amplify any numerical errors in the numerator as well as errors in $a$ itself. Therefore, it is important to treat the case when $a$ is small in a special way, and to use alternative formulas to solve the quadratic equation for $t$ to prevent these problems.

Another important consideration we have to make is that if the bounding spheres grow faster than they are moving apart, then they will start overlapping at some time $t$, but will never break apart, which means that one of the roots to the equation above should be infinity. However, if

$a = 0$, both roots given by the quadratic formula will be infinite, and if $a \neq 0$ then both roots will be finite. How can we solve this problem? The answer is to look at $a$ again. In our quadratic equation, $a = |\vec{v_1} - \vec{v_2}|^2 - (g_1 + g_2)^2$. When $a \leq 0$ it means that the particles are not moving away with enough speed to increase the distance between them. In that case, only one of the roots found with the quadratic formula will be valid, and we should manually assign infinity to the other root.

A few last things are worth our consideration as well. When two spheres have just collided, they are very close to each other—down to machine precision. The update from their previous position (from a past event) to the current position (where they are colliding) can introduce small numerical errors such that if the collision prediction is applied again, it will return an immediate collision for the two particles. Therefore, in addition to the tests above and all special cases, it is necessary to check the relative velocity of the particles at the time of impact to make sure a collision will happen before returning two particles as colliding. However, this test must be performed only when the particles are themselves spheres, since particles with other shapes can have their bounding spheres intersecting at any time.

When the tests above are performed for *bounding spheres* of other objects, the full interval must be always returned, with care to return the infinite roots correctly. Also, contrary to when testing for solid spheres, intersection intervals for bounding spheres of other particles starting in the past relative to current simulation time should be allowed, since the bounding spheres may already be intersecting at the current simulation time. Only the collision time for the actual particle shapes must be in the future relative to current simulation time. The collision time between the two actual particle shapes can then be determined through interval bisection if they intersect at the end of the interval. Otherwise, the whole interval must be searched for intersections. Therefore, in order to reduce the amount of expensive intersection tests between the actual particle shapes, the bounding sphere overlapping interval is confined to the simulation time. That is, if the bounding spheres began to overlap in the past relative to simulation time, the beginning of the testing interval can be safely modified to be the current simulation time, and if the bounding spheres never stop overlapping, the test interval can end at the end of simulation time (when the desired packing fraction would be reached or when packing density would be equal to 1).

## 5.2 The Gilbert–Johnson–Keerthi Algorithm

The Gilbert-Johnson-Keerthi (GJK) algorithm is an iterative method for computing the distance between two convex objects [117, 118]. In this algorithm, the distance between two convex sets

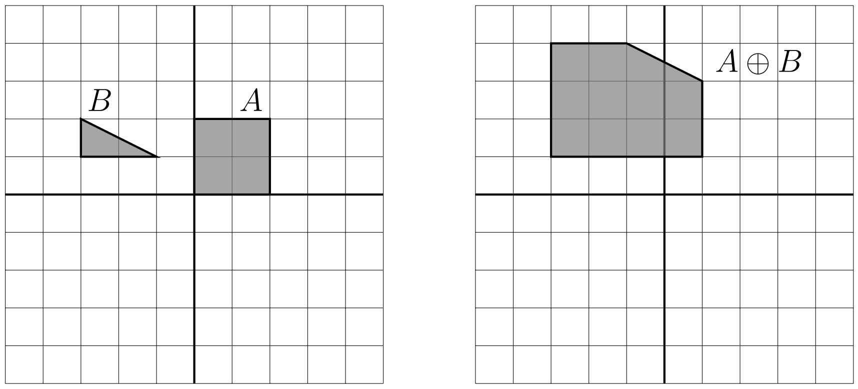


Figure 5.2: The Minkowski sum of a square ($A$) and a triangle ($B$).

of points is obtained from the distance between their Minkowski difference and the origin. This reduces the problem of finding the distance between two convex set of points to finding the distance between a single convex set of points and the origin. The GJK algorithm can be applied to any convex set that possesses a support mapping function. The major advantage of using this algorithm over level sets is that it performs very well for polyhedra, for which it always converges in a small number of steps.

### 5.2.1 Minkowski Sum and Difference

Let us now describe the concepts of Minkowski sum and difference, and support mapping functions before proceeding to the description of the algorithm itself. Let $A$ and $B$ be two point sets and let $\mathcal{A}$ and $\mathcal{B}$ be elements of $A$ and $B$, respectively. The Minkowski sum $A \oplus B$ is then defined as

$$A \oplus B = \{\mathcal{A} + \mathcal{B} \mid \mathcal{A} \in A, \mathcal{B} \in B\}$$

where $\mathcal{A} + \mathcal{B}$ is the vector sum of the two points $\mathcal{A}$ and $\mathcal{B}$. Visually, the Minkowski sum can be seen as the region swept by $A$ translated to every point in $B$. An illustration of the Minkowski sum of two sets is given in Figure 5.2.1.

The Minkowski difference of two point sets is defined analogously as

$$A \ominus B = \{\mathcal{A} - \mathcal{B} \mid \mathcal{A} \in A, \mathcal{B} \in B\}.$$

Geometrically, this is equivalent to define $A \ominus B$ as $A \ominus B \equiv A \oplus (-B)$, as illustrated in Figure 5.2.1.

The Minkowski difference of two convex polyhedra is also a convex polyhedron, and contains the origin if and only if the polyhedra intersect. In fact, if $C$ is the Minkowski difference between $A$

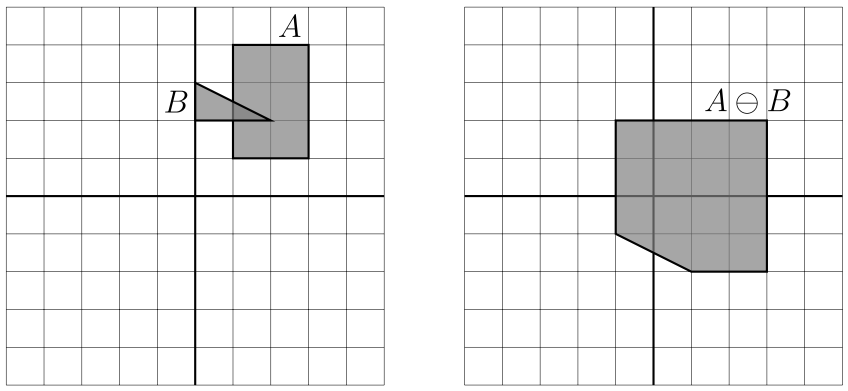


Figure 5.3: The Minkowski difference of a square ($A$) and a triangle ($B$). Since they intersect, their Minkowski difference contains the origin.

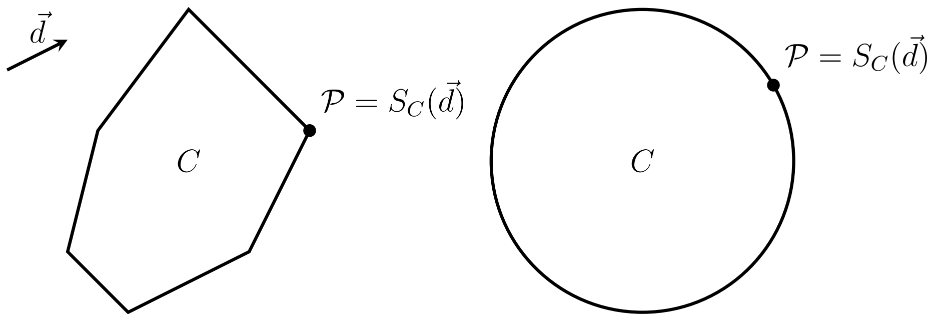


Figure 5.4: A supporting vertex on a polygon with respect to direction $\vec{d}$ (left), and a supporting point on a circle with respect to the same direction (right).

and $B$, we can establish the following result

$$\text{distance}(A, B) = \min\{\|\mathcal{A} - \mathcal{B}\| \mid \mathcal{A} \in A, \mathcal{B} \in B\} = \min\{\|\mathcal{C}\| \mid \mathcal{C} \in C = A \ominus B\}.$$

Therefore, to calculate the minimum distance between the two sets $A$ and $B$ is the same as to calculate the minimum distance between $C$ and the origin.

### 5.2.2 Supporting Points and Support Mappings

For a general convex set of points $C$, not necessarily a convex polyhedron, a point from this set most distant along a certain direction $\vec{d}$ is called a *supporting point* of $C$. In other words, it holds that $\vec{d} \cdot \mathcal{P} = \max\{\vec{d} \cdot \mathcal{V}, \forall \mathcal{V} \in C\}$, i.e., $\mathcal{P}$ is a point for which $\vec{d} \cdot \mathcal{P}$ is maximal. Figure 5.2.2 illustrates supporting points for two different convex sets. For a polytope, one of its vertices can always be selected as a supporting point for a given direction.

A *support mapping* is a function $S_C(\vec{d})$ that, given a direction $\vec{d}$, maps it into a supporting point of

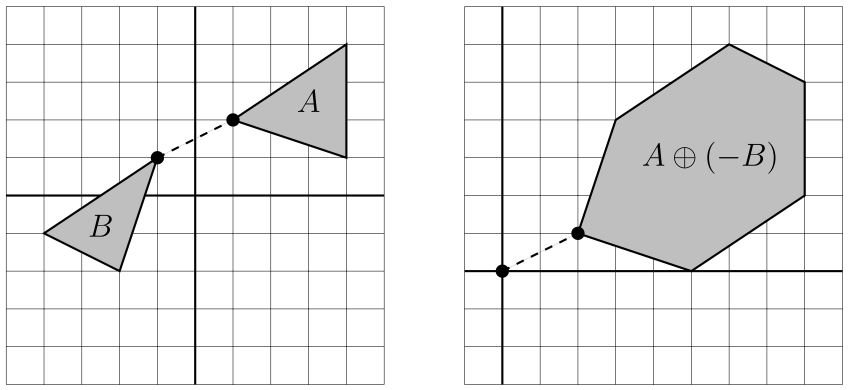


Figure 5.5: The distance between $A$ and $B$ is equivalent to the distance between their Minkowski difference and the origin.

$C$. Support mappings exist in closed form for simple convex shapes such as spheres, cylinders, boxes and cones. For a sphere, for example, the support mapping function is simply given by $S_C(\vec{d}) = O + r\vec{d}/\|\vec{d}\|$, where $O$ is the center of the sphere and $r$ its radius. For a polytope of $n$ vertices, a supporting vertex can be calculated in $O(n)$ time by checking all vertices. If vertices are kept in a data structure with a list of neighbors, and with the application of a few enhancements, this can be reduced to $O(\log(n))$ time. Therefore, even for complicated polyhedral shapes, this algorithm should be very fast.

### 5.2.3 Algorithm for Computing the Distance

At a first glance, using the Minkowski difference of two objects does not seem to simplify the problem of computing the distance between them, since the Minkowski difference is not trivial to compute. However, the cleverness of the GJK algorithm lies in not having to explicitly compute the Minkowski difference, but instead sample it through the support mappings for each of the convex shapes. The key concept is that the support mapping for $C = A \ominus B$ can be expressed in terms of the support mappings of $A$ and $B$. In fact, $S_C(\vec{d}) = S_{A\ominus B}(\vec{d})$ can be written as $S_{A\ominus B}(\vec{d}) = S_A(\vec{d}) - S_B(-\vec{d})$. Therefore, points of the Minkowski difference $A \ominus B$ can be computed from supporting points of the individual sets $A$ and $B$.

To search for the minimum distance point in $C$, the GJK algorithm uses a result from convex analysis known as *Carathéodory's Theorem*. This theorem states that for a convex body $H$ of $\mathbb{R}^d$, each point of $H$ can be expressed by a convex combination of no more than $d+1$ points from $H$. This means that the algorithm can search the volume of the Minkowski difference $C = A \ominus B$ by keeping a set of points $Q$ containing up to $d+1$ points from $C$ at each iteration. The convex hull of

$Q$ forms a simplex inside $C$, and if the origin is contained in this simplex, the polyhedra intersect and the algorithm stops. If not, $Q$ is updated to a simplex guaranteed to contain points closer to the origin, by choosing a new supporting point of $C$. Eventually, the algorithm terminates when the simplex $Q$ contains the closest point to the origin. In the case that $A$ and $B$ do not intersect, the smallest distance between $A$ and $B$ is given by the point of minimum norm in the convex hull of $Q$. The step-by-step algorithm is as follows:

1. Initialize the simplex $Q$ to contain up to $d+1$ points from $A \ominus B$.

2. Compute the point of minimum norm in the convex hull of $Q$.

3. If $\mathcal{P}$ is the origin, return the polyhedra as intersecting.

4. Reduce $Q$ to the smallest subset $Q'$ that still contains $\mathcal{P}$, i.e. remove all points from $Q$ not determining the subsimplex that contains $\mathcal{P}$.

5. Let $\mathcal{V} = S_{A\ominus B}(-\mathcal{P}) = S_A(-\mathcal{P}) - S_B(\mathcal{P})$ be a supporting point in the direction $-\mathcal{P}$.

6. If $\mathcal{V}$ is no more extremal than $\mathcal{P}$ in the direction $-\mathcal{P}$, then return $A$ and $B$ as non-intersecting, the distance between them being $\|\mathcal{P}\|$.

7. Add $\mathcal{V}$ to $Q$ and go back to 2.

Figure 5.2.3 shows an example of how the GJK algorithm works in two dimensions. Here, $Q$ is initialized to $Q = \{\mathcal{A}\}$. For a single-vertex simplex, the vertex itself is the point of minimum norm. The supporting point in the direction $-\mathcal{A}$ is $\mathcal{B}$. Therefore, now $Q = \{\mathcal{A}, \mathcal{B}\}$. The point of minimum norm in $Q$ is $\mathcal{C}$, a convex combination of $\mathcal{A}$ and $\mathcal{B}$. No points can be eliminated from $Q$ at this time. Then, the supporting point in direction $-\mathcal{C}$ is $\mathcal{D}$, leading to $Q = \{\mathcal{A}, \mathcal{B}, \mathcal{D}\}$. The point of minimum norm in $Q$ is now $\mathcal{E}$. Since only $B$ and $\mathcal{D}$ are needed to express $\mathcal{E}$ as a convex combination of vertices, $Q$ is updated to $Q = \{\mathcal{B}, \mathcal{D}\}$. The next supporting vertex in the direction $-\mathcal{E}$ is $\mathcal{F}$, so it is added to $Q$. The closest point to the origin in the convex hull of $Q$ is now $\mathcal{G}$. At this point, since no point is more extremal than $\mathcal{G}$ in the $-\mathcal{G}$ direction, $\mathcal{G}$ itself must be the closest point to the origin, and the algorithm terminates.

Although presented here as a method for polyhedra, this algorithm works for any kind of convex set of points, given that it has a support mapping function. The algorithm always terminates in a finite number of steps for polyhedra. However, it only converges asymptotically to the separation

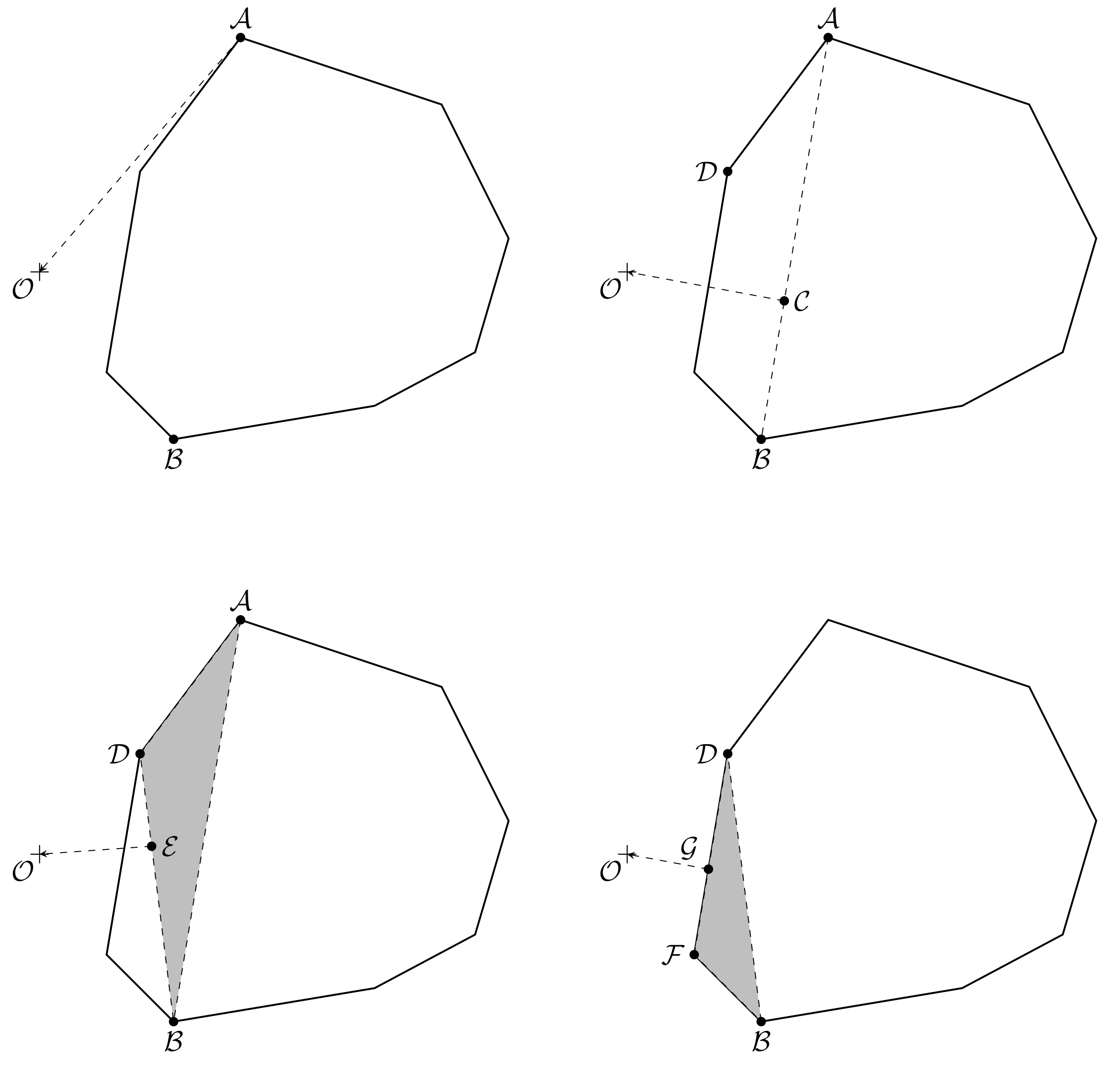


Figure 5.6: The Gilbert–Johnson–Keerthi algorithm finding closest point to the origin in a polygon.

distance of arbitrary shapes. It is therefore necessary to have stop criteria that allow the algorithm to terminate correctly in such cases without taking too many steps.

It is possible for this algorithm to additionally calculate the closest points in $A$ and $B$ in the non-intersecting case, and by caching $Q$ between calls, the algorithm can terminate in $O(1)$ time when predicting collisions of moving objects, by exploiting the fact that the closest feature will not change between small time steps. For concave polyhedra, this same algorithm can be applied by calculating convex decompositions. Alternatively, there are other algorithms that can be used [119–121] to support concave shapes.

# Chapter 6

# Spatial Partitioning

In the previous chapter, we described the algorithms we use to perform intersection tests between a pair of particles. In a large system containing up to hundreds of thousands of particles, however, pairwise tests for all particle pairs is impractical. The idea is then to have some broad-phase processing to restrict the number of pairwise tests to the particles that are already close enough that they could possibly intersect. This is usually accomplished through *spatial partitioning*—the simulation domain is divided into several regions and only objects located in the same region are tested against each other. Spatial partitioning techniques drastically reduce the number of pairwise tests needed, allowing the code to scale linearly with the number of particles, instead of quadratically. Due to their importance to the industry of 3D games, collision detection and spatial partitioning are both topics of extensive discussion in textbooks [92, 112–114].

## 6.1 Uniform Grids

A simple—yet effective—partitioning scheme is to overlay space with a regular grid of cells, dividing space into a number of regions of equal size. Each particle can then be assigned to a cell according to where its centroid is located. Since only particles that overlap a common cell can be in contact with each other, pairwise tests need only be performed for particles that belong to the same cell as the particle being tested or to one of its adjacent cells. Uniform grids are quite convenient, since the transformation between world coordinates and cell coordinates is very simple, and obtaining a list of the neighboring cells is trivial. Therefore, uniform grids are a popular choice for spatial partitioning.

The key parameter that affects the performance of a uniform grid is the size of its cells. As a general rule, the cells should be adjusted to be large enough to fit the largest particle at any orientation. This guarantees that a particle will overlap no more than eight cells for a 3D grid, and that only particles belonging to the same cell or adjacent cells need to be tested for intersection. Having particles overlap a minimum amount of cells is also important to limit the amount of work necessary to insert and update particles in the grid. A common method to determine the dimensions of a cell is the $n^{1/3}$ rule: given $n$ particles, divide space into a $k \times k \times k$ grid, with $k = n^{1/3}$. The ratio between the number of particles and cells, i.e., the average number of particles per cell, is called the *grid density*. If the grid density is far from its optimal value, it can have great impact in terms of performance, or—for statically allocated structures—make the grid prohibitively expensive in terms of memory. The optimal grid density will balance the amount of particle transfers needed to keep the grid updated and the number of pairwise intersections that need to be performed for each particle. If the grid density is too high, then many particles share each cell, making the number of necessary pairwise computations too high. On the other hand, if the grid density is too low, the number of particle transfer events relative to the number of pairwise intersection tests will become too high, partially destroying the performance gains provided by the grid.

The most natural way to implement a uniform grid is to allocate an $n$-dimensional array with the same dimensions as the number of cells. That way, there is a one-to-one mapping between grid cells and array elements. In order to handle the case in which multiple particles belong to the same cell, each array element becomes a pointer to the head of a linked list of particles. Using double-linked lists, it is possible to make insertions and deletions both $O(1)$. However, a drawback of this approach is that the number of linked lists for large grids can require a large amount of memory. Increasing the cell size of the grid may not be the best solution, since that will disrupt the balance between transfers and intersection tests as mentioned earlier, merely degrading the performance in a different way. We will discuss our solution to this problem in the next section, in which we describe the spatial partitioning scheme we use in our codes.

As the reader can probably conclude, the main disadvantage in using a uniform grid is that it becomes problematic when particles are either too large or too small compared to the cell size. Therefore, if all particles in a given simulation are equally sized—arguably the most common case—then a uniform grid is a perfectly adequate choice, offering both efficiency and simplicity. However, in material modeling, particles usually have a distribution of sizes, and particles of different materials can greatly vary in size. This means that using a uniform grid will come at a great cost in terms of performance. For example, in highly polydisperse packings it is possible that a grid becomes both too coarse and too fine at the same time, as shown in figure 6.1. Moreover, uniform grids are a

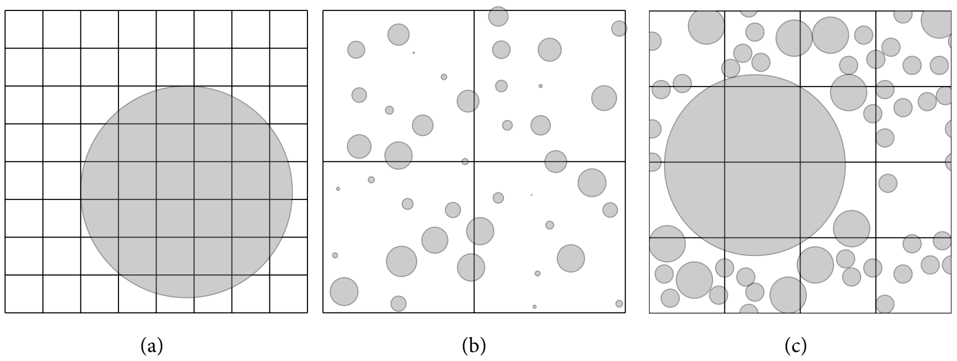


Figure 6.1: Issues related to cell size: (a) grid is too fine, (b) grid is too coarse, (c) grid is both too coarse and too fine.

suboptimal solution for packings of very elongated particles.

One possible approach to cope with this problem is to use neighbor lists to avoid dividing space into regions of a given size. However, pure neighbor list-based methods have been demonstrated to perform much worse than cell-based methods [122]. What we need is a grid that can adapt to particles of varying sizes—that is, a hierarchical grid of cells with an appropriate number of levels to keep particles of different sizes in optimally sized cells, as shown in figure 6.2.

## 6.2 Hierarchical Grids

A hierarchical grid is very much like a uniform grid in the sense that in it space is divided into a collection of regions and each particle belongs to only a single region at a time. However, in a hierarchical grid there are many levels, each of which is a uniform grid. Particles are then distributed according to their size as well as their position in space. Everything we discussed so far about uniform grids applies equally well to hierarchical grids. Since each level is a uniform grid, it is important to ensure that no level is either too coarse or too fine to hold particles at that level. The strategy is to make the cell size of finer levels be an integer fraction of the coarser level above it so that when a particle grows and becomes too big for a given level it is not too small for the next level. Another point is to not create extra levels that are much larger than the largest particles or smaller than the smallest particles, since every level must be checked when a particle is tested for collisions. At the highest level, the size of the cells is automatically adjusted to fit the largest particles at the end of the simulation (particles move up in the hierarchy of levels as they grow during the simulation). The number of levels needed is also automatically determined from the ratio between the largest and smallest particles such that cells at the lowest level can still contain the smallest particles through-

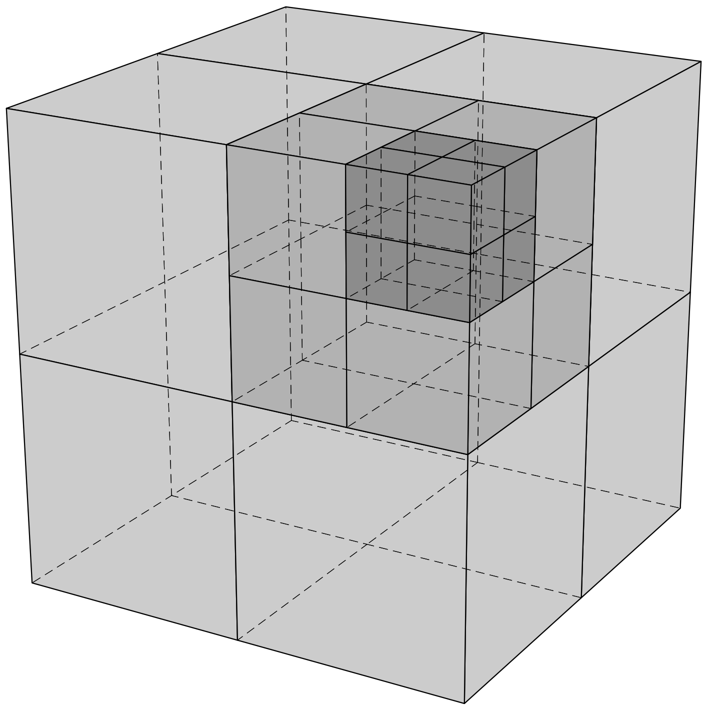


Figure 6.2: Hierarchical cell grid. Larger particles are kept in larger grid cells (level 0), medium sized particles are kept on medium sized cells (level 1), and so on.

out the simulation. That way, no empty levels remain at the bottom at any time. Intermediate levels are created as needed—if a whole level is empty, it is skipped entirely.

## 6.3 Hierarchical Grid Using Hash Tables and Linked Lists

The spatial partitioning scheme in our code went through several phases of development before being ready for large simulations. In the initial stages, we tested our code with only a few particles without using any spatial partitioning scheme. When collisions were implemented properly for spheres, a partitioning scheme was needed to run larger simulations. At this time, the code was being tested with up to a thousand spheres at a time. A run without the partitioning scheme would take several minutes for a thousand particles and a packing fraction of 0.5. Our early implementations of a partitioning scheme dropped it to about 10 seconds of running time, while in the latest code the same test finishes in about 0.2 seconds. The performance gains are enormous.

The first partitioning schemes we used were based on trees. Trees were appealing because they are well structured and the implementation of several functions, such as cell assignment, could be simplified with the use of recursion. However, trees also have many drawbacks. Trees are usually

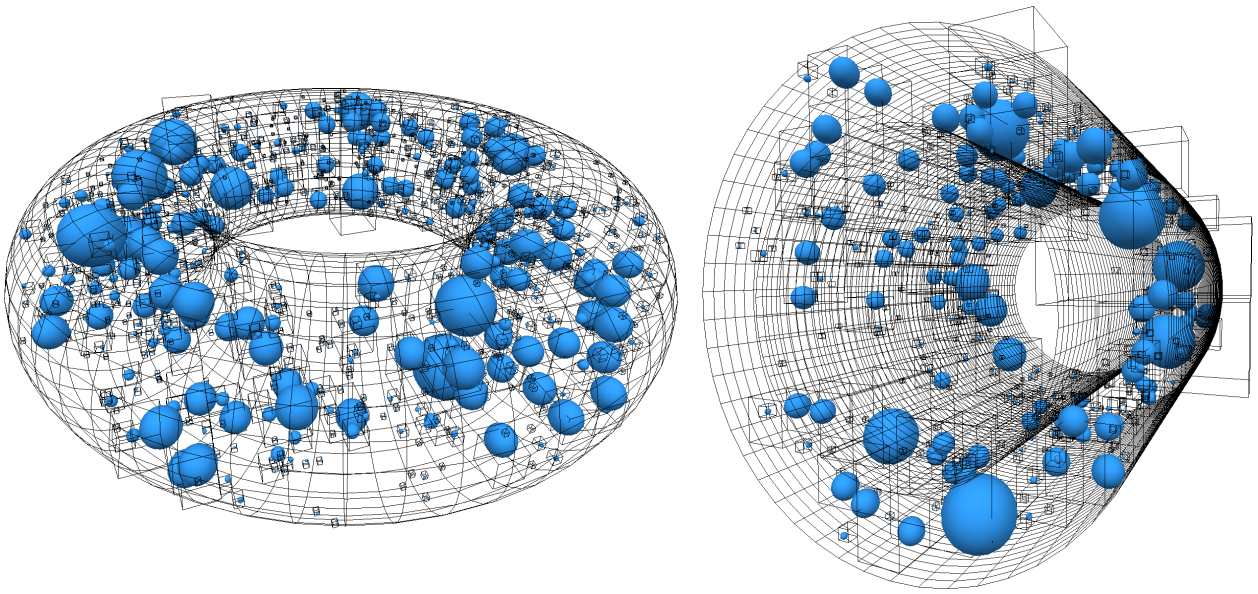


Figure 6.3: Demonstration of the hierarchical cell grid, as implemented in our packing code. Each particle is kept in a cell of the appropriate size. Only active cells are shown.

built to conform to the geometry of the container (in our case, a box), so the cells become elongated along with the box, which is not something particularly desirable. It was also difficult to adapt a tree to boundaries of complex geometries, such as a torus. The reason for this is that the top level of the tree must be large enough to contain the whole boundary, so many levels at the top would remain unused. This problem was exacerbated in simulations with many particles, since even at the end of the simulation the particles would still be quite small compared with the top level of the tree. Particles much smaller than the top level cell make traversing the tree hierarchy to find neighbors very inefficient, since no levels can be skipped. Moreover, we found that both dynamically allocated and statically allocated trees did not perform well. A dynamically allocated tree depends on many slow system calls to create and destroy cells by allocating and freeing memory. Since particles are constantly moving, the frequency of system calls becomes high enough to significantly degrade performance. On the other hand, a statically allocated tree, although free from slow system calls, simply occupies too much memory for the lower levels since each cell must have a pointer to its parent and a pointer to each child cell. The high amount of memory needed then causes many cache misses, also leading to weak performance. We tested various ratios between cells of consecutive levels trying to mitigate these problems, but, dissatisfied with the performance, decided to abandon the idea altogether.

Since a hierarchical grid is only a collection of uniform grids, the largest cells do not need to enclose the whole boundary, but only need to be large enough to accommodate the largest particles

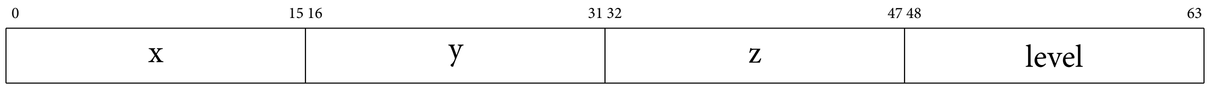


Figure 6.4: Bitfield description of the hash values assigned to particles.

at the end of the simulation. This means that even when there are many particles, there is no extra levels at the top to traverse in the search of neighbor particles for collision tests. Another benefit of this approach is that the complexity of the simulation boundary does not matter if the cells are dynamically allocated—the smaller the cells at the top level, the better the grid will conform to the boundary. Since each level is a uniform grid, transformations between simulation coordinates and cells is simple. Traversing the hierarchy to search for neighbors is also much simpler and more efficient, since only occupied levels need to be checked. However, some caution is needed in order to maximize performance. Similarly to what happens with trees, if cells are maintained as separate structures that are dynamically allocated, the same slow system calls will be necessary and performance will suffer. The first hierarchical grid we implemented that was not based on a tree still used these dynamic memory allocation system calls. Even if the performance was significantly improved compared with hierarchical grids based on trees, there was still a feeling of insatisfaction after we discovered that a large portion of running time (up to 80% in some cases) was spent on the hierarchical grid during profiling tests. The insight that illuminated the path to a reasonable solution was to realize that each particle can only occupy one cell at a time. Then, instead of allocating and destroying cells as needed, the particles themselves were adapted to contain part of the structure for the hierarchical grid. In the final implementation, each particle is assigned an integer hash value corresponding to its cell that describes both its position and at which level it is in the hierarchy. Each particle also contains two pointers that are used by the hierarchical grid to build the doubly linked lists for each cell. If a particle is the first to appear inside a cell, it becomes the head pointer for the linked list of particles in that cell. A pointer to the particle is then stored on a hash table that contains all cells in the hierarchy. When a second particle enters the same cell, the head of the list is fetched from the hash table, and the second particle is inserted into the list. When no particles belong to a cell anymore, the entry on the main hash table is cleared. Since each particle can only occupy a single cell at a time, the main hash table contains at most the same number of entries as there are particles, so it can be allocated in advance. Also, since all remaining pointers for the grid are kept inside the particle structure, no further dynamic allocation is needed, despite the fact that cells are created and destroyed as needed. Therefore, the performance in this implementation is a few orders of magnitude higher than previous codes developed at University of Illinois [53, 116, 123, 124].

After we were satisfied with the performance of our hierarchical grid, we moved on to automatically tune the cell sizes and number of levels for the particular size distribution in each run. To simplify level transfers, at each level the size of the cells is divided by two, effectively creating a hierarchy similar to an octree. The hierarchical grid provides a function call for the prediction of the next cell transfer for a moving particle and another function call to construct a list of neighbors that need to be checked when a particle collides with another. A bit mask of occupied levels is kept in order to skip intermediate unoccupied levels during collision checks. Each particle is stored in the grid according to its centroid position and bounding radius.

This hierarchical grid implementation is shared with our implementation of the LS packing algorithm, so it needed to retain the ability to treat moving particles. See [114] for more details on how to implement a similar structure. Yao *et al.* [122] reported some performance improvements by sorting particles in memory by a hash value based on Morton code, but we do not perform such optimizations in our code yet.

# Chapter 7

# Packing Algorithms for Convex Hard Particles

## 7.1 Introduction

In this chapter we present a detailed description of two collision-driven molecular dynamics packing algorithms for systems of nonspherical convex hard particles. One of the algorithms is an extension of previous event-driven molecular dynamics algorithms based on the original work of Lubachevsky and Stillinger [82]. This algorithm is most efficient for spheres and other smooth convex particle shapes, but it also supports polyhedral shapes. Since the performance of this algorithm for polyhedral shapes was not satisfactory, we developed a modification of it in which elastic collisions are replaced with a stochastic procedure for trial particle displacements to avoid particle intersections. Even though this novel algorithm partially eliminates the connection to thermodynamic systems that packing algorithms based on molecular dynamics usually have, it leads to several simplifications in the collision detection between particles, hence it is much more efficient. Nevertheless, due to the high computational cost of polyhedral intersections, packings of polyhedra take longer to generate if compared to packings of spheres.

The earliest studies employing molecular dynamics simulations used a simple model consisting of hard spheres [93]. Later, other methods were implemented for systems of spherical particles subject to a central, short range interaction potential. There is a fundamental difference between these two approaches: while soft particles interacting via a short range potential may overlap each other depending on the shape of the potential, the potential for hard particles is singular, thus implying that no particle intersections may occur. This requirement of no overlaps has a strong influence on the methods used to simulate each type of system. Systems of soft particles are relatively straightforward to simulate, since one can simply integrate a system of differential equations representing Newton's laws of motion. Many techniques exist to efficiently compute the force generated on each particle by their interaction with all the others, or with neighbors inside some cutoff distance for short range potentials, but these techniques do not alter the core of the algorithm, which is to proceed with the simulation by integrating the equations of motion on fixed time steps. In contrast, systems of hard particles cannot tolerate particle intersections, so instead of integrating a system of differential equations, one needs to predict a sequence of collision events and process them accordingly, by applying instantaneous impulse forces to prevent intersections between particles. For this reason, algorithms for hard particles are referred to as *event-driven* or *collision-driven* molecular dynamics (EDMD), as opposed to the *time-driven* molecular dynamics (TDMD) algorithms for soft particles.

Event-driven algorithms depend on continuous collision detection to predict and schedule a sequence of events that will happen in the future. Then, the simulation is advanced to the time of the earliest scheduled event and that event is processed. The list of scheduled events is updated if necessary and this process is repeated. This approach was used in the first molecular dynamics simulation of a system of hard disks [93]. Lubachevsky and Stillinger turned it into a packing algorithm [82] by starting with small disks and letting them grow over time. Later, these algorithms were improved and extended in a variety of ways. Perhaps the two most substantial improvements were the introduction of event priority queues and spatial partitioning methods. Early algorithms stored collision times for all particles on a triangular matrix. Besides the large amount of memory that this approach demanded, the triangular matrix needed to be searched in order to determine the earliest event, wasting a lot of resources. In modern implementations, only one event is kept per particle, and events are stored in a priority queue structure usually based on a heap. When a collision happened, early algorithms predicted new collisions against all of the other particles, which scaled as $O(n^2)$ with the number of particles $n$. However, in a simulation with many particles, only a few of them will be actually close enough to a given particle to be able to collide with it. Spatial partitioning methods, such as uniform grids and other cell-based methods, as well as neighbor list methods, allow collision checks to be performed only against particles that are sufficiently close, thereby greatly increasing the performance of the algorithm—i.e. making it scale as $O(n)$, as we will demonstrate later.

Most event-driven packing algorithms have been developed only for spheres, the primary reason being that collision detection for other particle shapes can be quite difficult to implement correctly, and the reason for that is the finite precision of floating point arithmetic and all of the problems that it can cause. As David Goldberg puts it well [125], despite being ubiquitous, "floating point arithmetic is considered an esoteric subject by many people." In fact, floating point representation errors, rounding errors, numerical cancellation, overflow and underflow, etc, can often catch us by surprise and make seemingly simple and naive algorithms behave in unexpected ways. This is especially true in computational geometry and computer graphics algorithms, which are in general very hard to implement in a numerically robust way. Nevertheless, molecular dynamics algorithms have been used for packing nonspherical particles such as ellipsoids, superellipsoids [86, 87], spherocylinders and polyhedra [53]. Molecular dynamics can be the only solution that will yield correct results in some cases, such as in the simulation of packings of needles, for example [126]. Our own packing algorithms support any convex shape that has a support mapping, and can be extended to support other shapes by individually providing intersection algorithms for each one.

One of the main objectives of our algorithms is to improve on the performance of previous codes to allow the simulation of packings with a larger number of particles in the same running time. In order to accomplish this goal, we utilize the Gilbert–Johnson–Keerthi (GJK) algorithm [117] in a packing algorithm for the first time. The GJK algorithm is used extensively in the 3D games industry, where performance is critical, but has apparently not been noticed by scientists working to develop particle packing codes. We also borrowed some ideas from techniques used in 3D games to develop our spatial partitioning scheme that we described in the previous chapter. Next, we discuss how to combine all these algorithms together to build our particle packing codes. First, however, we give a high-level overview of both time-driven and event-driven molecular dynamics algorithms.

### 7.1.1 Time-Driven Molecular Dynamics

In time-driven molecular dynamics algorithms, the equations of motion for all particles are integrated directly. All particles thus move *synchronously* in small time steps $\Delta t$. Particles are tested for overlap at the end of each time step. If any intersections are found, the simulation must be rolled back to the initial time of overlap to process the collisions between particles. Depending on the size of the time step, the whole time step can be rolled back, or an approximate collision time can be computed that lies in between the previous and current time steps. After all collisions are processed, the simulation proceeds as usual. This approach has some advantages: it is easier to parallelize, collision detection can often be implemented using only static tests (i.e. for particles that are not moving), the integration of equations of motion with external forces is easy to implement, etc. However, there are big disadvantages with this approach too. Since the integration of the equations of motion proceeds in fixed time steps, sometimes the collisions between particles may be missed, or the correct order in which collisions happen may be wrongly predicted. Small overlaps between particles are almost inevitable too, so this approach does not provide a *rigorous* solution when packing hard particles. Small overlaps can be mitigated by making the time step smaller, but in most cases the time step needs to be excessively small, making the code become very inefficient. This situation may change in the future if time-driven algorithms are ported to use the graphics processing unit (GPU). GPUs are highly parallel and can make even very small time steps feasible. Nonetheless, porting any algorithm to run on a modern GPU is nontrivial and requires a lot of effort and technical expertise. Even though molecular dynamics algorithms used to simulate protein folding, for example, have used GPUs for some time [127], we are not aware of any packing algorithms that use GPUs.

### 7.1.2 Event-Driven Molecular Dynamics

In event-driven molecular dynamics, the simulation proceeds in variable time steps that correspond to the times when particle collisions and other events occur. An *event* is defined as anything that changes a particle's state. Binary collisions are the most common type of event, hence event-driven algorithms are also referred to as being *collision-driven*. Other events can be collisions against hard walls, transfer between grid cells, neighbor list updates, collision checks, etc. Since each particle is only updated when it is part of an event, the simulation is *asynchronous*. Particles only need to be synchronized at the end of the simulation. After each event is processed, the new *impending event* for each of the particles involved is predicted. An *event priority queue* is maintained to keep track of predicted events in the order that they will happen. Occasionally, events may be mispredicted, or they can become invalid if the collision partner suffers an earlier collision against another (different) particle. In those cases, instead of applying an impulse to the particle, a new prediction of its next event is made (i.e., the event becomes a *check* event), and the simulation continues as usual. However, a mechanism must be in place to detect when an event becomes invalid. The two most common solutions are to keep a record of either the latest timestamp of the particles in the event, or to keep the number of collisions that each particle has gone through. If there is a mismatch, one of the particles must have been part of an event and the current event should be considered invalid. The sequential processing of events means that parallelizing this algorithm, although not impossible [128], is quite difficult. Most implementations of event-driven molecular dynamics algorithms are serial, including ours. Nevertheless, event-driven algorithms significantly outperform time-driven algorithms for packing spheres and other simple smooth shapes, and have the major advantage of being *rigorous* (to machine precision, at least).

At this point, one might be wondering about events in which more than two particles are involved in a collision at the same time, since these types of events are not treated in event-driven molecular dynamics simulations. Although in principle three-way or $n$-way collisions are possible, the probability for that to happen is small, so these events are safe to ignore in event-driven algorithms. What will happen is that a three-way or $n$-way collision will just be processed sequentially instead, with time steps of $\Delta t = 0$, since $\Delta t$ is variable. In time-driven algorithms, however, impulse-based methods for collision resolution may fail spectacularly, leading to large intersections between particles. Therefore, in time-driven algorithms, it is often necessary to consider collisions with more than two particles explicitly. This is especially true in simulations in which there are resting contacts between particles due to gravity, for example. Even in event-driven algorithms impulse-based methods of collision response can become unreliable. Using the GJK algorithm to compute intersections between polyhedra, we found out that the collision response becomes unreliable when particles are very close to each other due to floating point arithmetic problems, the reason being that the sub-simplex of the Minkowski difference between the polyhedra becomes degenerate—i.e., it might be a very flat tetrahedron or a triangle with nearly collinear points—and the resulting normals to the collision are not accurate enough to solve for an impulse that will move both particles apart. That is why we needed to modify our packing algorithm for polyhedra to use a stochastic procedure to move particles apart instead of applying impulses to resolve collisions.

## 7.2 The Lubachevsky–Stillinger Packing Algorithm

The first algorithm we have developed, for packing spheres, is based on the Lubachevsky–Stillinger packing algorithm [82]. In this algorithm, particles begin as infinitesimal points and grow over time until the packing becomes jammed or the desired packing fraction is achieved. Our implementation of this algorithm can generate packings of spheres inside a periodic box, or inside solid containers of various shapes, including a box, a sphere, a cylinder, an annulus (including wedges with $0 < \theta < \pi$), and a torus, as shown in figure 7.1. New container types can be easily added to the code since their implementation is decoupled from the rest of the system. The interface that a boundary needs to implement is shown in listing 1. The function `operator()(const Vector& x)` gives the distance to the boundary; it is negative inside the boundary and positive outside. The `gradient()(const Vector& x)` function is the gradient of the distance. The other functions are self-explanatory. For periodic boundaries, an additional function called `reposition()` is used to fix the position of a particle that currently lies outside the boundary (usually an image that was

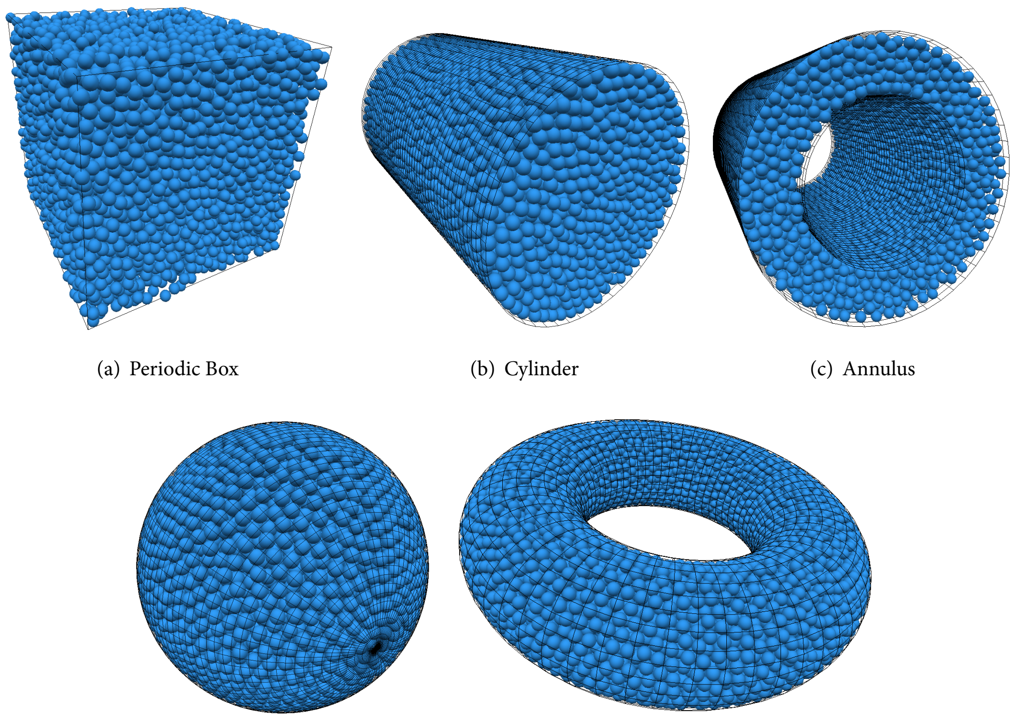


Figure 7.1: Packings of spheres confined to different boundaries.

part of a collision). The code has been implemented in C++ in order to make use of object orientation techniques to improve readability, since the concepts of boundaries, particles, shapes, etc,

```cpp
class Boundary {
  public:
    virtual ~Boundary() {}

    virtual float operator()(const Vector& x) const = 0;

    virtual Vector gradient(const Vector& x) const;

    virtual float predict_collision(const Particle& p) const;
    virtual void collision_response(Particle& p) const;
    virtual Point get_random_position() const;

    virtual bool is_periodic() const { return false; }
    virtual Vector dimensions() const = 0;

    virtual float volume() const = 0;
};
```

Listing 1: Boundary class definition.

make perfect sense to be abstracted away as classes. Using polymorphism, different boundaries and particle shapes can be implemented separately, while the parts of the code that make use of them remain generic and need not change when new boundary or particle shapes are introduced. The code also offers the user the possibility to change the growth rate of the particles, thus enabling some limited control on the type of final state that will be generated. In general, slow growth rates have a higher chance of producing packings with higher degrees of order, while fast growth rates will almost inevitably lead to jammed packings near the MRJ packing fraction of 64%.

```
while (t_curr < t_stop) {
  event_queue->next_event()->process();

  event_queue->update();

  t_next = event_queue->next_event()->time();
  t_prev = t_curr;
  t_curr = t_next < t_stop ? t_next : t_stop;
}
```

Listing 2: Main loop.

The main loop of the code is shown in listing 2—code for printing progress information and drawing particles on the screen was removed for clarity. The logic is very simple: at each step, we check for the ending condition, then proceed by processing one event and updating the event queue. The stopping time `t_stop` is computed ahead of time at the beginning of the simulation, using the sum of the volumes of all particles as a function of time. After the event is processed, the event queue is updated by computing new events for each particle involved in the current event, The current time `t_curr` and stopping time `t_stop` are also used to bracket the time interval that needs to be checked when predicting particle collisions. which is a single particle in the case of a collision with the boundary or a cell transfer, or two particles in the case of a binary collision. Most of the complexity of the code is in predicting and processing the events. The biggest improvement of the current implementation relative to old codes is in performance. Choosing the right algorithms for collision detection and spatial partitioning has allowed the new code to perform large simulations in a reasonable amount of time. Simulations that used to take several days to run with the old code [53] now finish in less than an hour with the new code. Even large simulations of polydisperse packings containing up to a million particles—previously unfeasible—now take only a few hours to run.

## 7.3 Event Processing

In the original LS algorithm, there was no structure to keep track of which particles would be first to collide, i.e. which event was going to happen next. Instead, a triangular matrix containing the collision times of all particle pairs was used. More recent algorithms use priority queues to store events. Priority queues allow the earliest event to be determined in constant time, while also only requiring a single event to be kept for each particle, thus eliminating the need for a triangular matrix of collision times. The event priority queue used in our code is based on a complete binary tree [129]. In this implementation, each leaf of the tree is associated with a particle event, and each internal node of the tree has the label of the earliest event between its two children. No deletions need to be performed on the tree since when an event happens, a new prediction is made for that particle and the tree is updated accordingly. This avoids using slow system calls for memory allocation, which would make the priority queue perform much worse, as the survey in [129] demonstrates. More sophisticated algorithms for priority queues that scale better with the number of particles do exist [130], but the complexity to implement them is much higher. We believe that the complete binary tree algorithm provides a good trade off between performance and complexity. Since standard implementations of heaps can be found in textbooks [131], we do not show our implementation here.

The structure of an event is shown in listing 3. We use four types of events: collisions, boundary collisions, transfers and invalid events. An event holds its own type, which is determined and set at each time `predict()` is called. It also holds the time at which it will happen, which is always larger than the current simulation time; the numbers of collisions that particles `A` and `B` have undergone before its prediction; a displacement vector `m_shift` to allow particles to collide with images across periodic boundaries; and a pointer to one or two particles, the first of which is permanently bound to the event when it is created, hence the lack of arguments in the functions `predict()` and `process()`.

One could separate each collision type into a different class via polymorphism, but we have found that this approach is vastly inferior in performance since the type of the next event of a particle changes very frequently, and each type change requires a new memory allocation (as well as freeing the memory of the previous object, of course, to avoid memory leaks). Therefore, although not as elegant, the current implementation is much faster.

The function to process an event is also fairly simple. It is shown in listing 4. Events are processed according to their validity and their type. The function `collision_response(*A, *B)`

```cpp
class Particle;

enum EventType { INVALID, COLLISION, BOUNDARY_COLLISION, TRANSFER };

class Event {
  public:
  Event(unsigned int id, Particle* A);

  int type() const { return m_type; }
  float time() const { return m_time; }
  unsigned int id() const { return m_id; }
  unsigned int secondary_id() const;

  void print();
  void process();
  void predict();

  private:
  float m_time;
  unsigned int m_id;
  Particle *A, *B;
  unsigned int cA, cB;
  Vector m_shift;
  EventType m_type;
};
```

Listing 3: Event class definition.

is responsible for computing and applying an impulse that will ensure that particles A and B move away from each other afterwards. The function sync() integrates the trajectory of a particle to synchronize it with the current event time. In chapters 4 and 5 we have discussed how to integrate the equations of motion of a particle and the topics of collision detection and collision response. Therefore, we do not show the implementations of the functions collision_response(*A, *B) and boundary->collision_response(*A) here. These functions also depend on particle shapes and boundary types, so listing them here would be impractical.

The function to predict an event, shown in listing 5, is somewhat more complicated. Since a particle always will escape the boundary if left alone (it will either move out or grow enough to touch it), we predict this event type first. Then, if a transfer or a binary collision happen earlier than the boundary collision, the event is changed accordingly. We do not show here the implementation of the function time_of_impact(), where collision predictions are actually performed. As is the case with collision detection, collision prediction also varies with particle shapes (and with packing algorithm, as will be discussed later).

```
void Event::process()
{
    bool validA = A->collisions() == cA;
    bool validB = B && B->collisions() == cB;

    switch (m_type) {
        case COLLISION: {
            A->sync(m_time);
            B->sync(m_time);

            if (validA && validB) {
                if (boundary->is_periodic())
                    A->translate(m_shift);

                A->collided();
                B->collided();

                collision_response(*A, *B);

                if (boundary->is_periodic()) {
                    ((PeriodicBoundary*)(boundary))->reposition(*A);
                }

            }
            break;
        }

        case BOUNDARY_COLLISION: {
            A->sync(m_time);
            if (validA) {
                boundary->collision_response(*A);
                A->collided();
            }
            break;
        }

        case TRANSFER: {
            A->sync(m_time);
            hgrid->rehash(A);
            break;
        }

        default:
            /* nothing is done for an invalid event,
             * event queue will update it accordingly */
            return;
    }
}
```

Listing 4: Function to process an event.

```
void Event::predict()
{
    unsigned int i, j, k, n;
    static std::vector<Particle*> neighbors;

    /* particle A and id never change */
    cA = A->collisions(); B = NULL; cB = 0; m_shift = Vector(0.0, 0.0, 0.0);

    /* check collision with boundary */
    m_type = BOUNDARY_COLLISION;
    m_time = boundary->predict_collision(*A);

    /* check cell transfer time */
    float t_transfer = hgrid->next_rehash_time(A);

    if (t_transfer < m_time) {
        m_type = TRANSFER;
        m_time = t_transfer;
    }

    /* check collision for particle and its images if periodic */
    if (!boundary->is_periodic()) {
        neighbors.clear();
        hgrid->find_neighbors(A, neighbors);

        for (n = 0; n < neighbors.size(); i++) {
            float t_collision = time_of_impact(*A, *neighbors[n], t_curr, m_time);

            if (t_collision < m_time) {
                m_time = t_collision; B = neighbors[n];
            }
        }
    } else {
        Point x = A->position();
        Vector period = boundary->dimensions();

        for (int i = -1; i <= 1; i++) {
            for (int j = -1; j <= 1; j++) {
                for (int k = -1; k <= 1; k++) {
                    Vector shift(i*period[0], j*period[1], k*period[2]);

                    neighbors.clear();
                    A->set_position(x + shift);
                    hgrid->find_neighbors(A, neighbors);

                    for (n = 0; n < neighbors.size(); n++) {
                        float t_collision =
                            time_of_impact(*A, *neighbors[n], t_curr, m_time);

                        if (t_collision < m_time) {
                            m_time = t_collision; B = neighbors[n]; m_shift = shift;
                        }
                    }
                }
            }
        }

        A->set_position(x); /* return particle to initial position */
    }

    if (B != NULL) { m_type = COLLISION; cB = B->collisions(); }
}
```

Listing 5: Function to predict the next event of a particle.

## 7.4 Jamming Criterion

In chapter 2, we discussed some jamming properties of packings of spheres. In computer models, it is impossible to reach an idealized state in which all of the particles are exactly touching each other. There will always be a small finite gap between particles that is limited by the machine precision. A packing can then be considered jammed when the volume of the configuration space becomes sufficiently small. However, measuring this volume is not easy. Often, computer codes use heuristic criteria to determine when a packing is jammed. In packings of hard-spheres generated with the Lubachevsky–Stillinger algorithm, for example, the collision frequency between the particles relative to simulation time diverges as the packing approaches jamming. In Monte Carlo simulations of hard-particle packings, the packing is considered jammed if a high number of trial moves fail to increase the density of the packing. In our code, there are two possibilities to determine when to stop the simulation. One is to simply tell the code to stop after a certain amount of time that is usually larger than necessary for the packing to reach jamming. Another alternative is to stop the simulation when progress becomes very slow, i.e., when the increase in packing density per unit time (wall clock time) falls below a given threshold. However, these stopping criteria are optional. When they are not in use, the code will run indefinitely until the user decides to stop it manually, which is the only other alternative to stop the simulation.

It is important to note, however, that none of the artificial criteria to stop the simulation imply that any packings are indeed jammed. Donev, in his PhD thesis [132], demonstrated that if the simulation is allowed to continue, the interparticle distances gradually decrease until machine precision is reached. It is for this reason that we provide the means to leave that decision to the user instead. Therefore, there is no strict confirmation that any packing is in a jammed state at termination—that is not the focus of our research. However, our algorithm should lead to at least collectively jammed configurations if the simulation is run for a long enough time. Rigorous tests performed on packings of hard spheres generated with Lubackevsky–Stillinger's algorithm and other methods [132] show that the resulting packings are virtually strictly jammed, although packings of disks generated with the same methods are not in general even collectively jammed. That is due to a fundamental difference in the disordered–ordered phase transition in two and three dimensions. While there is a clear phase transition in three dimensions, it is much less pronounced in two dimensions (if a transition exists at all).

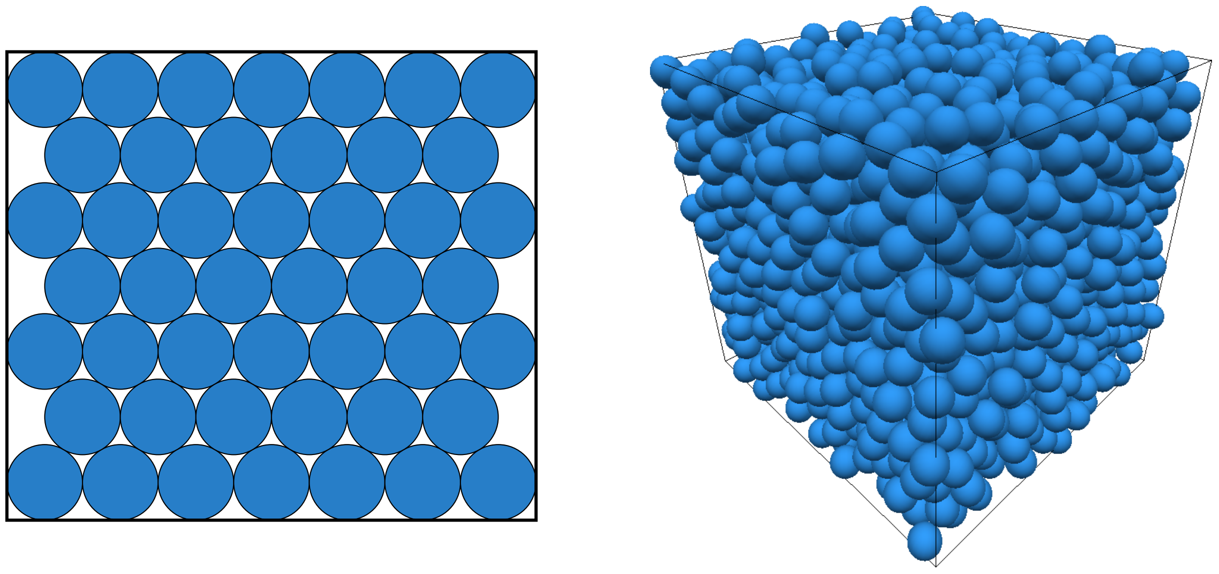

(a) Strictly jammed disks (b) Strictly jammed spheres

Figure 7.2: Strictly jammed packings of disks are highly ordered while strictly jammed packings of spheres can be disordered.

## 7.5 Hybrid Monte Carlo Packing Algorithm for Polyhedra

The Lubachevsky–Stillinger packing algorithm is very reliable for packing spheres. It leads to virtually strictly jammed states in almost every run with packing densities around $\varphi \approx 0.64$. Packing polyhedra, however, is much more difficult. The misleading belief that frictionless particles should "nudge" themselves into place via collisions in a similar fashion to spheres made us struggle for a long time to make the same algorithm work for polyhedra, to no avail. The packings always seemed to get locked into low density states, since the collision response system was unable to find a suitable impulse to prevent particles from growing into each other. We have investigated the source of the problem and found it to be related to numerical errors in the computation of the normal vector between colliding bodies. More specifically, during collision detection and collision response, if particles are too close to each other, it is difficult to determine the normal vector of the collision. Our first implementation of the GJK algorithm, which is the algorithm responsible for these computations, was based on the so called Johnson's distance algorithm to compute the closest point to the origin in the Minkowski difference between the two bodies. When the bodies are too close, the simplex—usually a triangle or tetrahedron—becomes degenerate or very near degenerate. That causes a division by zero in the Johnson's distance algorithm, which in turn breaks the normal computation. We then reimplemented that part of the algorithm using a purely geometrical method that did not rely on inverting any matrices to find the closest point to the origin, leading to some stability in the algorithm. Nevertheless, since the two bodies are always very close at the time of collision, the flattened simplices still caused many errors when computing normal vectors and collision times. Therefore, we needed to abandon collisions altogether and use a different method for packing polyhedra that did not make use of collisions. The most common algorithms in the literature for packing polyhedra are based on Monte Carlo techniques. Instead of using growing particles with collisions, in these algorithms trial displacements are randomly applied to particles and they are accepted whenever the packing fraction can grow further with the new configuration. This brute force method has been used to find the highest packing fractions of tetrahedra in the past—although it was later surpassed by analytical methods.

A drawback of Monte Carlo methods is their performance. The purely random rearrangements often displace particles that are already far apart, wasting processing power. For that reason, some runs reported in the literature took up to a month of running time. That is a hefty time investment to make when trying to model the microstructure of a material. The ASC algorithm presented by Torquato and Jiao [51] is more sophisticated, but its internals are not described in detail and it seems to be needlessly complex for the kind of application that we aim for. Therefore, in our packing algorithm for polyhedra, we merely replace the functions for collision prediction and collision response with functions that resolve collisions by moving particles apart by a finite distance. This makes our algorithm a hybrid between Monte Carlo methods and the Lubachevsky–Stillinger algorithm, as it is still an event-driven algorithm. All of the code discussed in the previous section for the Lubachevsky–Stillinger packing algorithm is shared between both implementations. However, particles are not allowed to move and undergo elastic collisions, thereby greatly simplifying collision prediction and response. Since the particles do not move, but only grow in place, the only tests needed between them to determine collisions are Boolean intersection tests between static particles. This kind of test can be performed without problems with the GJK algorithm almost to machine precision without any problems. The modified collision response function is shown in listing 6. When particles are about to collide, they are randomly moved away from each other. If no move succeeds for either particle (because they may be locally jammed), all of their neighbors are moved as well. The number of trial moves and the size of the displacements both become smaller as packing density increases, to compensate for the fact that particles are getting close together and large displacements would invariably fail. They are also decreased in each successive try, to increase the probability of success. Optionally, it is possible to shake all particles in the packing when no progress is made after several iterations. However, this usually degrades performance of runs with a large number

```cpp
void collision_response(Particle& A, Particle& B)
{
    unsigned int i;
    static std::vector<Particle*> neighbors;

    if(!move(A) && !move(B)) {
            neighbors.clear();
            hgrid->find_neighbors(&A, neighbors);
            for (i = 0; i < neighbors.size(); i++) {
                move(*neighbors[i]);
                event_queue->update_id(neighbors[i]->get_event_id());
            }

            neighbors.clear();
            hgrid->find_neighbors(&B, neighbors);
            for (i = 0; i < neighbors.size(); i++) {
                move(*neighbors[i]);
                event_queue->update_id(neighbors[i]->get_event_id());
            }

            move(A); move(B);
    }
}
```

Listing 6: Collision response for polyhedra.

of particles, since shaking the entire packing becomes very computationally expensive. Therefore, this procedure is usually only used in packings containing less than $10^4$ particles. In the future, we plan to speed up this algorithm by performing trial moves in parallel, or on the GPU, to allow simulations with large numbers of polyhedra. This will improve the calculation of statistical properties of packings of polyhedra, such as the radial distribution function, for example.

# Chapter 8

# Performance Benchmarks

In this chapter we present performance benchmarks of our codes using packings of spheres and Platonic solids. We have devoted special attention to the choice and implementation of the algorithms we use in our codes to ensure that the improved performance would allow us to simulate large, complex systems containing many particles. Solid propellant materials often comprise mixtures of particles of various sizes and shapes which can be a challenge to simulate if polydispersivity and the performance of intersection tests are neglected during code design. Performance is also important because algorithms based on molecular dynamics are inherently serial, and our codes are not an exception to the rule. Therefore, we cannot rely on hardware improvements alone to lower running time.

The new codes developed for this dissertation offer significant improvements over previous implementations, in addition to several new features, some of which were described in previous chapters. We have successfully used our codes to generate packings containing up to a few million spheres and tens of thousands of polyhedra, although larger systems can be simulated if time is not an issue. Most runs we are interested in, however, typically finish in a few hours of runtime.

## 8.1 Packings of Spheres

The Lubachevsky–Stillinger packing algorithm has a direct link with the actual dynamics of a system of hard particles. In the case of spheres, the behavior of the system resembles that of an ideal gas. The collision frequency between particles, for example, is somewhat analogous to the pressure in the system. As the packing density increases, the collision frequency also increases. Therefore, the collision frequency of the particles in the simulation is expected to diverge as the packing approaches a jammed state. This limits the maximum packing density that can be obtained with this packing algorithm. In some studies, the collision frequency is used to estimate the "true" jamming density of simulated packings using the equations of state for an ideal gas and extrapolating data into higher densities. Here, due to the focus on performance, we will try a different approach using the running time of the code for different numbers of particles and packing densities. Let us assume that, for a fixed number of particles, the packing time is a function of the form

$$T(\phi) = \frac{A}{(\phi - \phi^*)^2} + B,$$

where $A$ and $B$ are constants proportional to the number of particles in the packing and represent the packing cost and initialization cost of the algorithm, respectively.

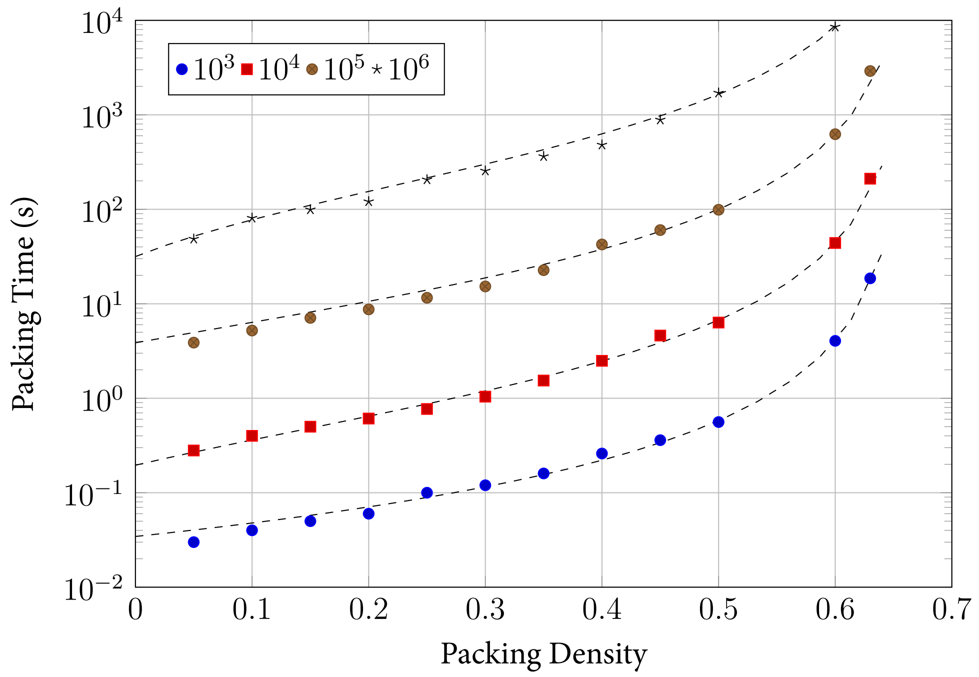


Figure 8.1: Packing time of monodisperse spheres as a function of final density for different numbers of spheres.

The initialization cost is the time it takes to read input files, generate the particles, randomly place them inside the simulation domain, predict the earliest event for each particle, and initialize the event priority queue. The initialization cost is usually negligible compared with the total cost of packing, but can become large in absolute terms (several seconds) for packings containing more than a few hundred thousand particles. This function diverges at $\phi = \phi^*$, which is the point at which the packing becomes jammed and the simulation cannot proceed any further. The exponent in the denominator was determined *ad hoc*, so while it may represent a deeper concept that we do not capture in this analysis, it is also possible that for a different implementation of the same packing algorithm this dependency could be different due to a number of factors.

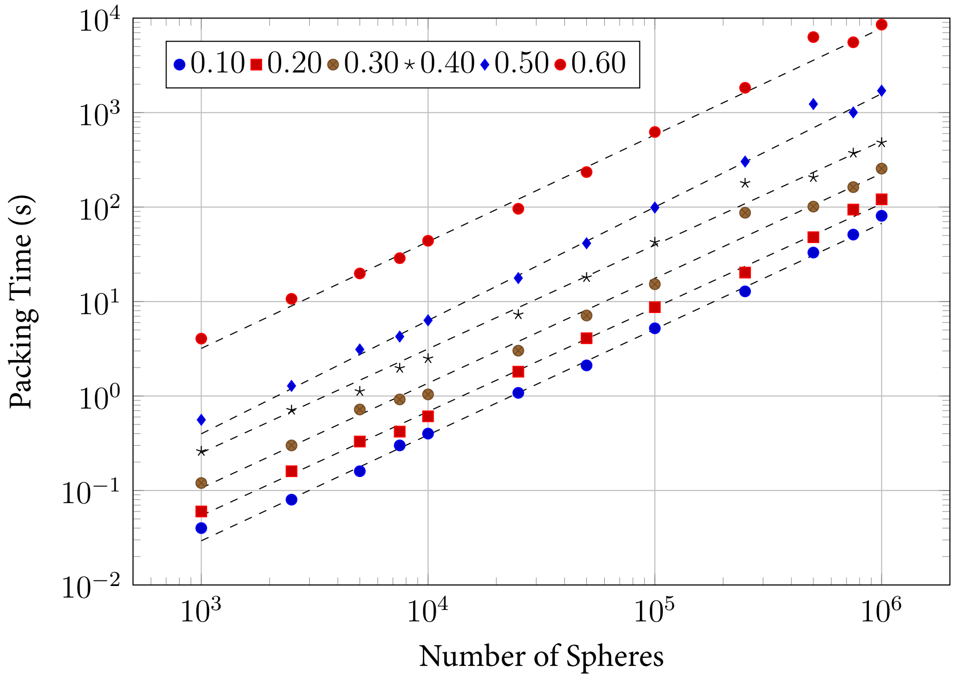


Figure 8.2: Packing time of monodisperse spheres as a function of the number of particles.

Figure 8.1 shows packing times for monodisperse packings of spheres as a function of the final density for different numbers of spheres. Dashed lines show fits to the data using the expression above for packing time $T(N, \phi)$, where $N$ is the number of particles and its dependency is hidden in the constants $A$ and $B$, which change according with the number of particles. All runs were performed on a desktop computer with an Intel®Quad Core i7 2.8 GHz processor with 8MB of shared L3 cache and 4 G of DDR3 1067 MHz memory. In table 8.1 we list values of the jamming packing density $\varphi^*$ obtained from the fits to the data. Ideally, if the code could run for a very long time, the final packing density would slowly approach these values.

Table 8.1: Jamming Packing Density $\varphi^*$

| $N$ | $10^3$ | $10^4$ | $10^5$ | $10^6$ |
|---|---|---|---|---|
| $\varphi^*$ | 0.6610 | 0.6657 | 0.6680 | 0.6809 |

In figure 8.2, we can see that our modified Lubachevsky–Stillinger packing algorithm scales roughly linearly with the number of particles. The dashed lines are fits to the data of the form

$T(N) = AN^b$, where $A$ and $b$ are constants and $N$ is the number of spheres. The exponent $b$ assumes values in the range $1.1 \leq b \leq 1.2$ in the fits shown.

In order to see how polydispersivity affects the performance of the code, we generated packings of $10^3$, $10^4$ and $10^5$ particles with lognormal size distributions with varying polydispersivity $\sigma$. The results are shown in figure 8.3. Except for one of the runs for $10^5$ particles at $\sigma = 0.3$, that ran atypically fast, all runs take longer as the polydispersivity increases. This is because as the number of occupied levels increases, more cells need to be searched for neighbors during collision checks, hence the simulation becomes slower. Larger particles also have many neighbors against which they must be checked every time one of them happens to be involved in an event. The higher number of occupied levels also means that particles need to be transferred from lower to upper levels more frequently as they grow. However, since particles with a lognormal size distribution do not vary drastically in size, an additional test using uniform size distributions was performed. In this worst case scenario, in which all levels are evenly occupied with spheres of varying sizes, the impact on performance is much more clearly visible. The number of levels necessary to cover the full range of sizes has a large impact on performance due to the larger number of cells with neighbors that need to be checked for future collisions.

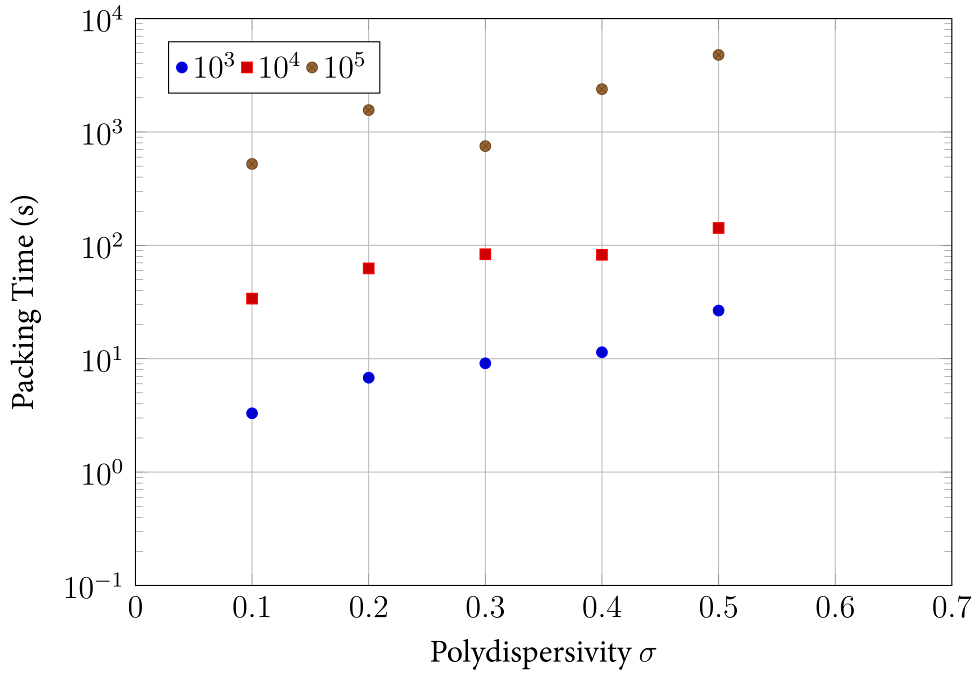


Figure 8.3: Packing time of polydisperse spheres as a function of the polydispersivity $\sigma$.

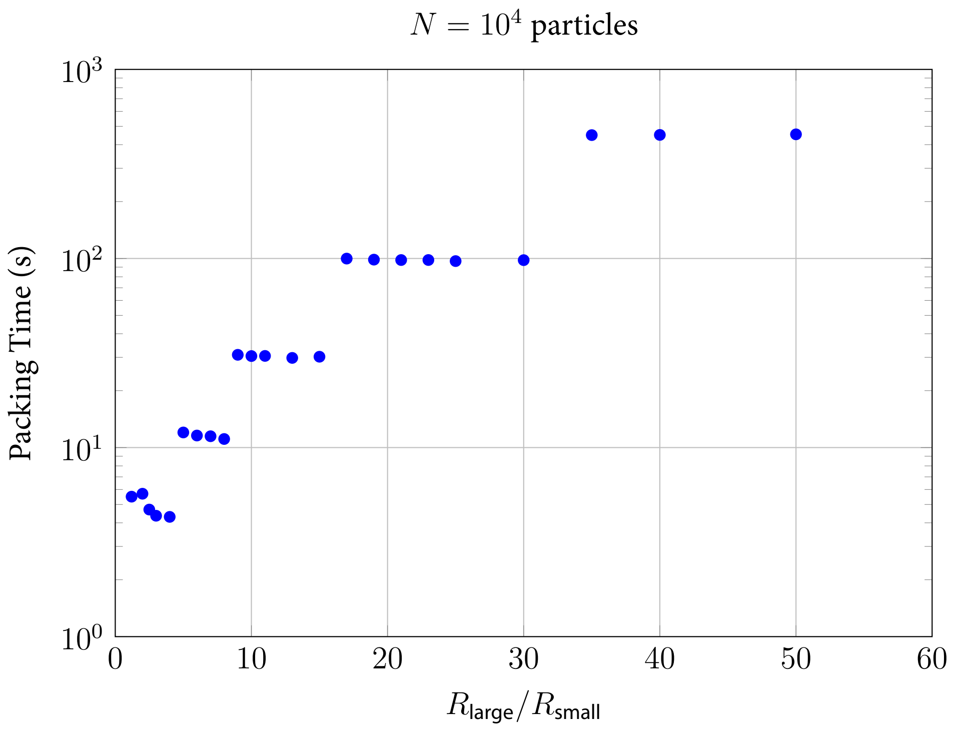


Figure 8.4: Packing time of polydisperse spheres with uniform distribution as a function of the ratio between the largest and smallest spheres.

## 8.2 Packings of Convex Polyhedra

When packing polyhedra, particle intersections become much more expensive to compute, so packing times are many times higher for the same number of particles. The impulse-based collision response mechanism from the Lubachevsky–Stillinger packing algorithm is also ineffective in preventing particles from intersecting each other, which leads to dead locks in the event queue. Therefore, the new algorithm for polyhedra presented here using random displacements becomes more efficient when generating packings of polyhedra. This algorithm also scales linearly with the number of particles, as shown in figures 8.6 and 8.7. Figure 8.6 shows packing times for $10^4$ icosahedra and figure 8.7 shows the same data for cubes. In order to determine how our code scales as the number of particles grows, we fitted a function of the form $T = A * N^b$ to the data. For polyhedra, the exponent $b$ is usually in the range $0.95 \lesssim b \lesssim 1.1$, which means that the code for packing polyhedra also scales roughly linearly with the number of particles. Deviations from perfect linearity are due to the random nature of the algorithm. However, the constant $A$ is certainly much higher for packings of polyhedra, and depend not only on the number of particles, but also on particle shape. In figure 8.5 we compare how hard it is to pack each of the Platonic solids and show that tetrahedra, although being the simplest polyhedron in number of features, is the hardest to pack overall. Moreover, despite intersection calculations being cheaper for cubes, and the fact that cubes can tile Euclidean space, cubes are harder to pack than icosahedra, since icosahedra are closer in shape to a sphere. In general, the higher the asphericity of a polyhedron, the more difficult it is to produce dense packings of those polyhedra. Packing tetrahedra tightly together, for example, requires specially arranged initial conditions and very long runs [50].

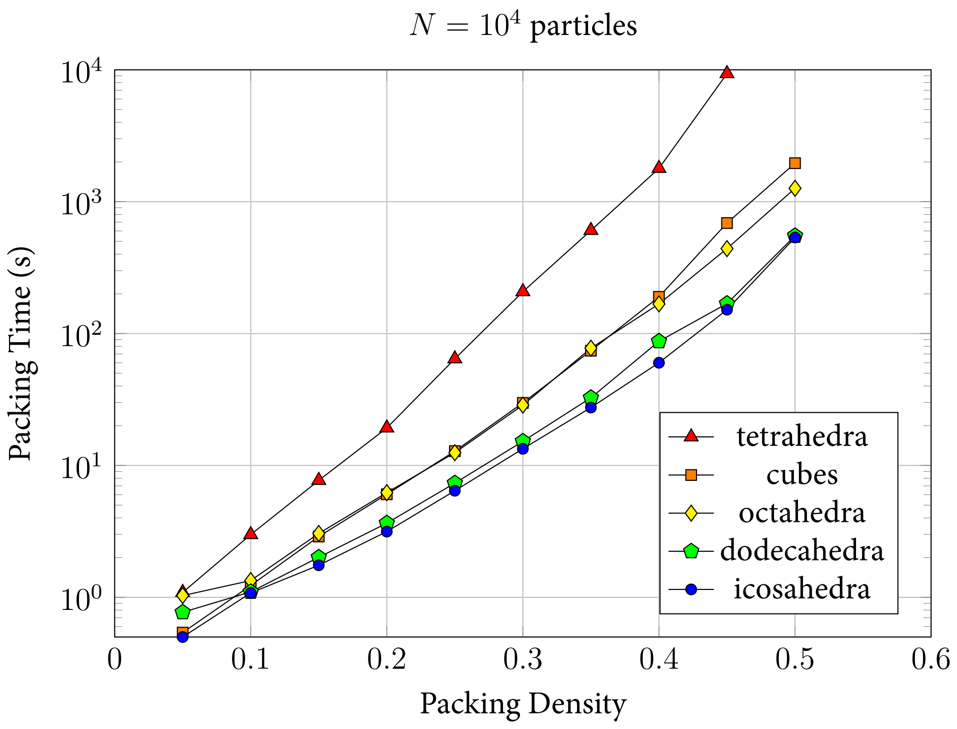


Figure 8.5: Packing times as a function of packing density for $10^4$ particles of each Platonic solid.

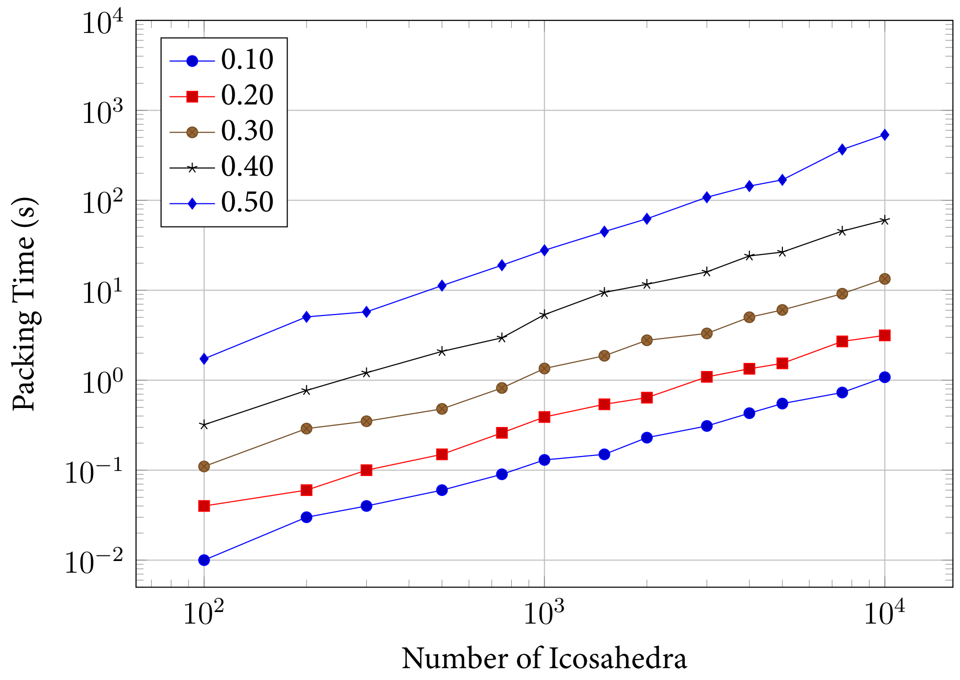


Figure 8.6: Packing times of monodisperse icosahedra.

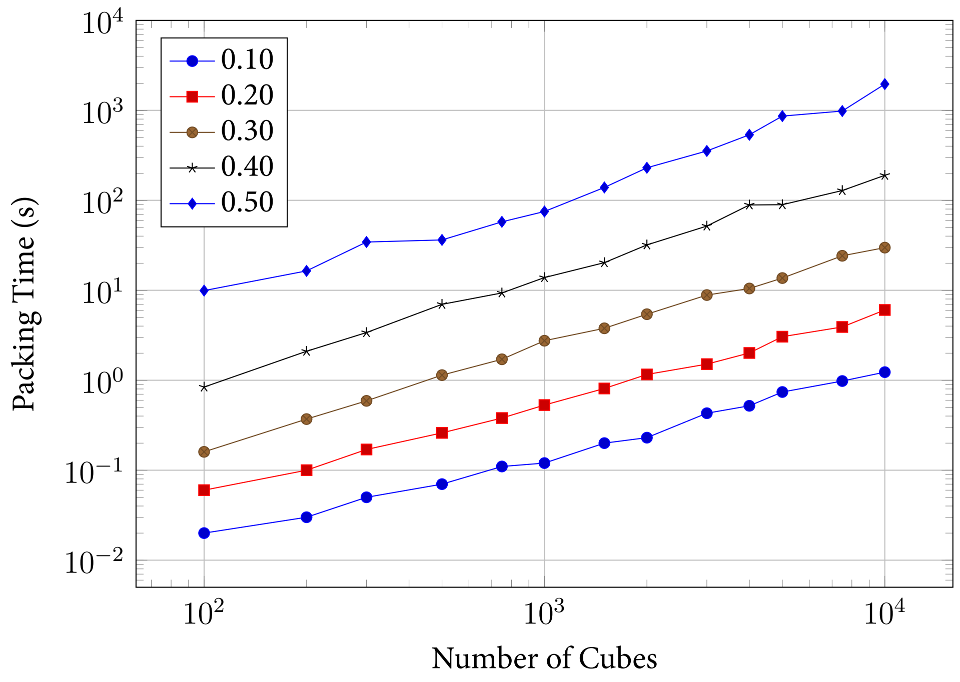


Figure 8.7: Packing time of monodisperse cubes.

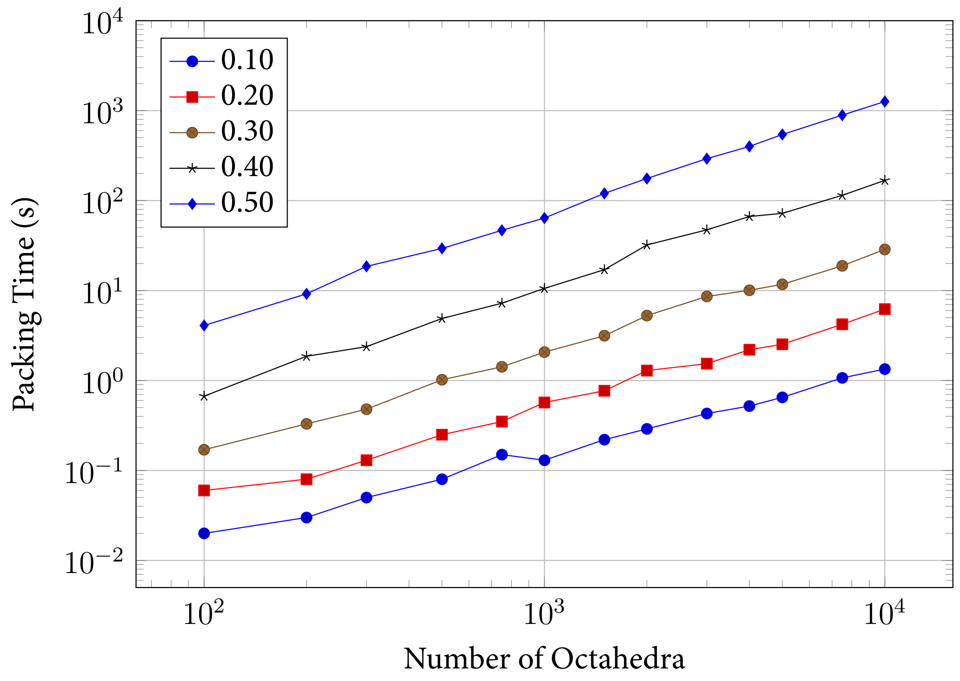


Figure 8.8: Packing time of monodisperse octahedra.

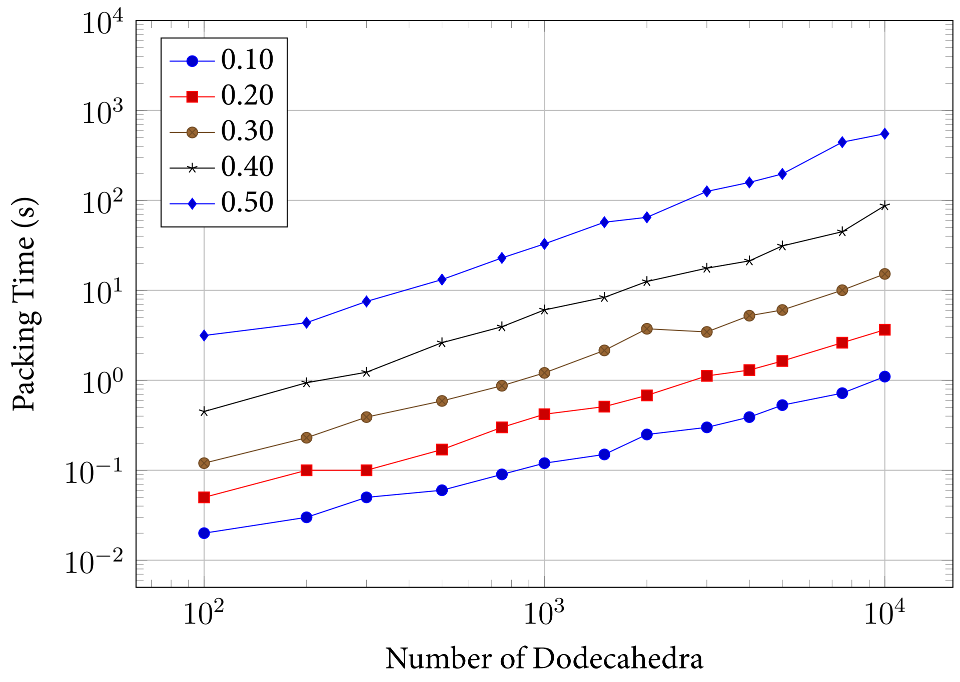


Figure 8.9: Packing time of monodisperse dodecahedra.

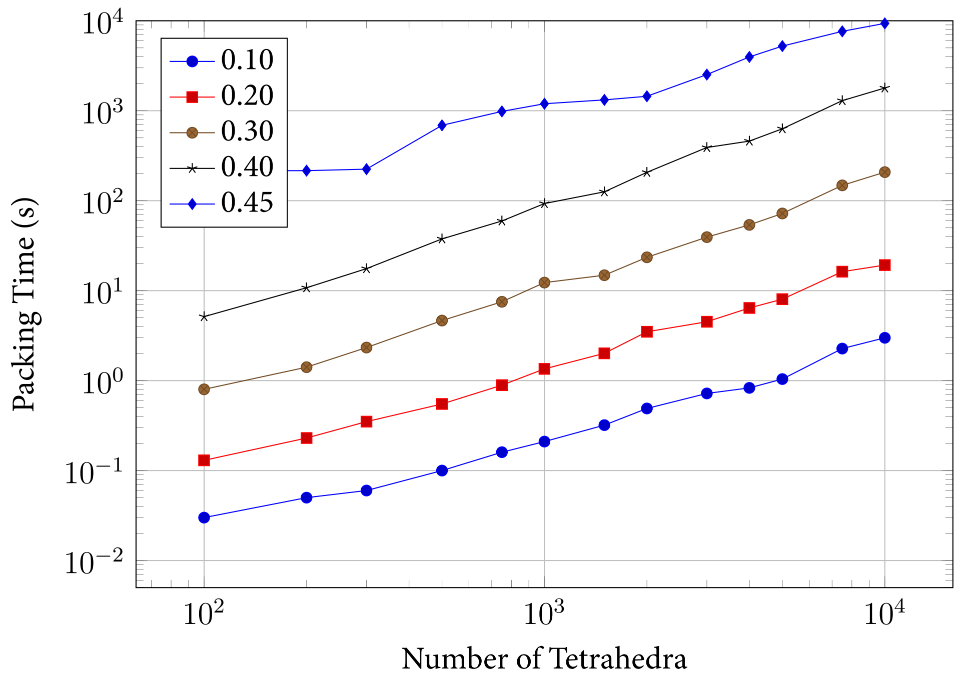


Figure 8.10: Packing time of monodisperse tetrahedra.

# Part II

# Particle Packings: Applications

# Chapter 9

# Packings of Hard Spheres

## 9.1 Introduction

In this chapter we apply the Lubachevsky–Stillinger packing algorithm to study packings of hard spheres. Despite their simplicity, spheres can be used in a wide variety of models of physical systems such as liquids [26, 30, 93], glasses [34], colloids [35, 36], and granular materials [5, 37, 69, 133, 134]. They also find applications in models of physical phenomena such as fluid flow through packed beds [31, 135], as well as in seemingly unrelated areas of research, such as in communication theory [136]. Our interest in packings of spheres advenes from their use as a model of highly polydisperse aluminized solid rocket propellants. The agglomeration of fine aluminum particles in the liquefied thermal layer near the burning surface plays an important role in nozzle erosion. Apart from the direct applications of hard-sphere packings mentioned above, the scientific community has also demonstrated a continuous interest in the statistical and geometric properties of hard-sphere packings [71, 76, 78, 83, 88, 94, 137–141] and related phenomena, such as thermal equilibrium and jamming properties [90, 98, 99, 142–145]. Although these topics are not the main goal of our research, we deem them to be of sufficient importance to justify a brief discussion of some of them here. Unless otherwise stated, the results we present henceforth were obtained with our own implementation of the Lubachevsky–Stillinger molecular dynamics packing algorithm. We would like to alert the reader, however, that we perform no rigorous tests that guarantee strict jamming of the packings we present next. The implementation of rigorous tests for jamming requires significant additional effort without much in return in our case, since enforcing jamming is not critical for modeling propellants. Moreover, our packing code can easily reach packing densities around 64% that are typically associated with strictly jammed packings—namely, the RCP and MRJ states—, thus verification of strict jamming is not one of our major concerns. Rigorous tests that demonstrate that the Lubachevsky–Stillinger packing algorithm generates virtually strictly jammed packings of spheres have been performed recently by Donev as part of his PhD work [132].

## 9.2 Monodisperse Packings of Hard Spheres

The first systems we consider in our discussion are packings of monodisperse spheres. The most efficient arrangement of a packing of identical spheres—as proposed by Kepler and recently proved by Hales [29]—is the face-centered cubic configuration, with packing density $\varphi = \frac{\pi}{3\sqrt{2}} \approx 74.048\%$. At a first glance, one may conclude that packing experiments and simulations can easily reach this ideal state. Perhaps surprisingly, however, both mechanical packing methods and packing simulations employing algorithms such as the Lubachevsky–Stillinger packing algorithm usually stall much earlier, after reaching a random disordered configuration with a packing density close to $64\%$. This interesting phenomenon became the subject of much debate in the literature [39, 95, 98–101, 146, 147]. Nevertheless, the underlying physical processes behind the natural preference for disordered states remain elusive. Over the years, solid evidence has accumulated that a disordered–ordered phase transition happens near the jamming packing fraction of disordered spheres. Traditionally, the jammed state of spheres at this density ($\phi \approx 0.64$) has been referred to in the literature as the "random close packing" (RCP) state. However, the concept of the RCP state has been plagued by lack of mathematical rigor in its definition; different packing protocols lead to different values for the jamming packing fraction. In fact, using the Lubachevsky–Stillinger packing algorithm, Torquato *et al.* demonstrated in [95] that the final packing fraction could be influenced by the growth rate of the particles during the simulation. The fastest compression rates led to states with a packing fraction around 64%, while slower compression rates allowed the packings to proceed further to progressively more ordered states. They argued against the notion of the RCP state due to its ambiguity, and advocated the use of a more precisely defined state based on the jamming categories proposed in [95] and local measures of randomness [97]—i.e., the maximally random jammed state (MRJ). The MRJ state is the strictly jammed state that minimizes one of the order metrics (preferably many). Despite the introduction of the MRJ state, however, some recent publications still use the more traditional concept of the RCP state [99–101]. These studies provide alternative methods for improving the definition of the RCP state as the point at which the phase transition into ordered states takes place. In these studies, the definition of jamming is based on the minimum requirements for mechanical stability. Mechanical equilibrium imposes a minimum number of constraints that is at least equal to the number of equations balancing the forces and torques in the system. This so-called *isostatic* condition [148–150] is widely believed to be a necessary condition for jamming. Jin and Makse [99] provide an in-depth analysis of the transition. Using the average coordination number of spheres in jammed packings ranging from $\phi = 0.56$ to 0.74 packing density, they found that the jamming packing density $\phi_j$ and the average coordination number $Z_j$ are related to each other by

$$\phi_j = \frac{Z_j}{Z_j + 2\sqrt{3}}$$

in the disordered branch of the phase diagram (see figure 1 in [99]). The isostatic condition for frictionless spheres is then $Z = 2d = 6$, where $d$ is the number of dimensions. Therefore, to satisfy the isostatic condition, the density of random close packed states should be $\phi_{\text{rcp}} = \frac{6}{6+2\sqrt{3}} \approx 0.634$.

To provide an estimate for $\phi_{\text{rcp}}$ using our implementation of the Lubachevsky–Stillinger packing algorithm, we performed 1000 runs of a packing of 1000 spheres inside a periodic box. Each run is stopped when the rate of increase of packing density per unit time falls under $3 \times 10^{-4}$/min. This threshold is chosen to ensure that the system is close enough to jamming, since about 90% of running time is spent on increasing the packing density by only a few times $10^{-4}$ at the end. The choice of a small system is necessary to allow for a large number of runs in a reasonable amount of time with this condition. A histogram of the packing densities is shown in figure 9.1. The average packing fraction was found to be $\phi = 0.6410$, with standard deviation $\sigma = 0.0054$. This is only slightly higher than the computed density for the RCP state, $\phi_{\text{rcp}} \approx 0.634$. Nevertheless, our results show the ambiguity argued by Torquato in the definition of the RCP state, since we obtain a different mean packing density in our runs. Using the order metrics described in chapter 2, one could assign different values for the amount of ordering in each packing, and choose the packing for which the value or the order metric is minimum as the MRJ state. On the other hand, to get sufficiently close to the true minimum of the order metric among jammed packings, it is necessary to use a large ensemble of jammed packings. The growth rate of the particles can influence the final jamming density of the packings [95, 99], but we do not reproduce these results here. As it can be seen in the histogram, the great majority of the runs stop near $\phi \approx 0.64$. Although some packings do reach higher densities, packings at $\phi > 0.66$ are progressively more difficult to obtain. This is usually the case for molecular dynamics algorithms in general.

Using a larger packing containing $10^5$ particles, we compute the number of contacts per particle after the packing becomes jammed. It should be noted that packings generated on a computer inevitably contain finite (but small) interparticle gaps. This means that the particles do not form perfect contacts, and hence contacting neighbors must be determined via a gap tolerance. A common

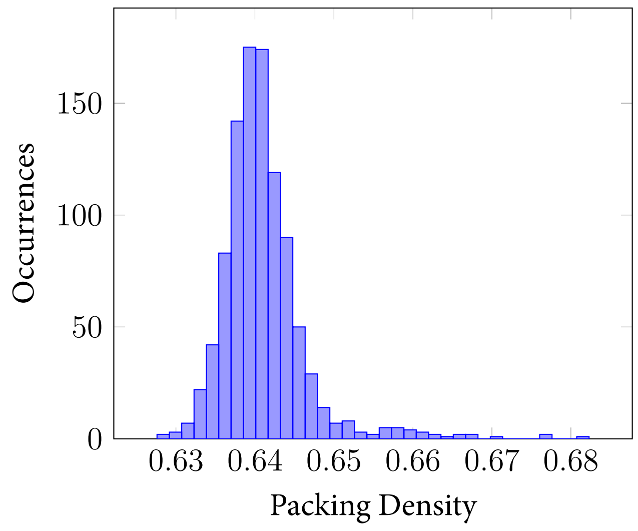


Figure 9.1: Histogram of final packing densities for a system of 100 spheres.

choice for the gap tolerance is the average gap size across the entire packing. The mean gap size in our packing of spheres is about $2 \times 10^{-4}$, which is slightly less than $10^{-2}$ times the particle diameter. Gaps smaller than the gap tolerance are associated with a contact between the particles. The average coordination number obtained with this choice of gap tolerance is $Z_j = 6.026$, which is very close to the isostatic condition. The resulting histogram showing the probability for each number of contacts is shown in figure 9.2. The interparticle distances typically decrease monotonically until they reach machine precision if the packing is allowed to run for a very long time, although the corresponding increase in packing fraction is rather small. Therefore, the packing can be considered

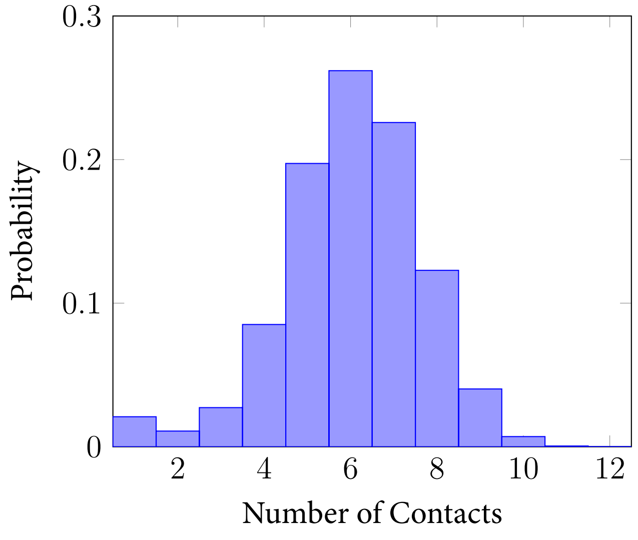


Figure 9.2: Histogram of the coordination number for a packing of $10^5$ spheres.

to be virtually jammed despite the finite gaps that exist between particles. We can also conclude that this jammed packing generated with our code is nearly isostatic, implying that it is likely to be *strictly* jammed as well, or, in other words, the packing is mechanically stable.

### 9.2.1 Statistical Properties

The statistical properties of packings of congruent spheres are fairly well known. In some cases, it is possible to compute the $n$-point probability functions exactly, if the pair correlation functions are known. The $n$-point probability functions for monodisperse hard spheres can be expressed as a summation of a series of integrals that depend on the pair correlation functions (see chapter 5 in [39]). The expression for the $n$-point probability functions of impenetrable-sphere systems was first derived by Torquato and Stell (1982). Although calculating the $n$-point probability functions and radial distribution functions exactly is out of the scope of our work, we compute these functions directly, using large systems of spheres simulated with our packing code.

#### Radial Distribution Function

The radial distribution function $g(r)$ gives the probability of finding a particle center at a distance $r$ from a certain particle fixed at the origin. In figure 9.3 we provide the radial distribution functions of four different packings of $10^5$ spheres at different packing fractions. The radial distribution function can be used to illustrate how this system transitions from random configurations into more ordered ones, as the packing fraction increases. Ordering occurs wherever $g(r)$ deviates from 1, as $g(r)$ is normalized against the expected value for randomly distributed points in space. For small packing fractions, the deviation from a random distribution is small, while at higher densities there are clear peaks that grow as spheres come closer into contact. The two clearly visible peaks at $r = 2\sqrt{3}$ and $r = 4$ for the packing at $\phi = 0.64$ are very characteristic of the MRJ state. The peak at $r = 2\sqrt{3}$ represents nearly planar clusters of 4 spheres, arranged in a 1–2–1 configuration, while the peak at $r = 4$ represents linear clusters of three spheres. Packings with crystalline structure exhibit infinite peaks at discrete values of $r$. Peaks at $r = 2\sqrt{2}$ and $r = 2\sqrt{5}$ are characteristic of the face-centered cubic configuration. Although at long range MRJ packings become uniform, in short- and mid-range distances there are density fluctuations that decay with a power-law in $r$.

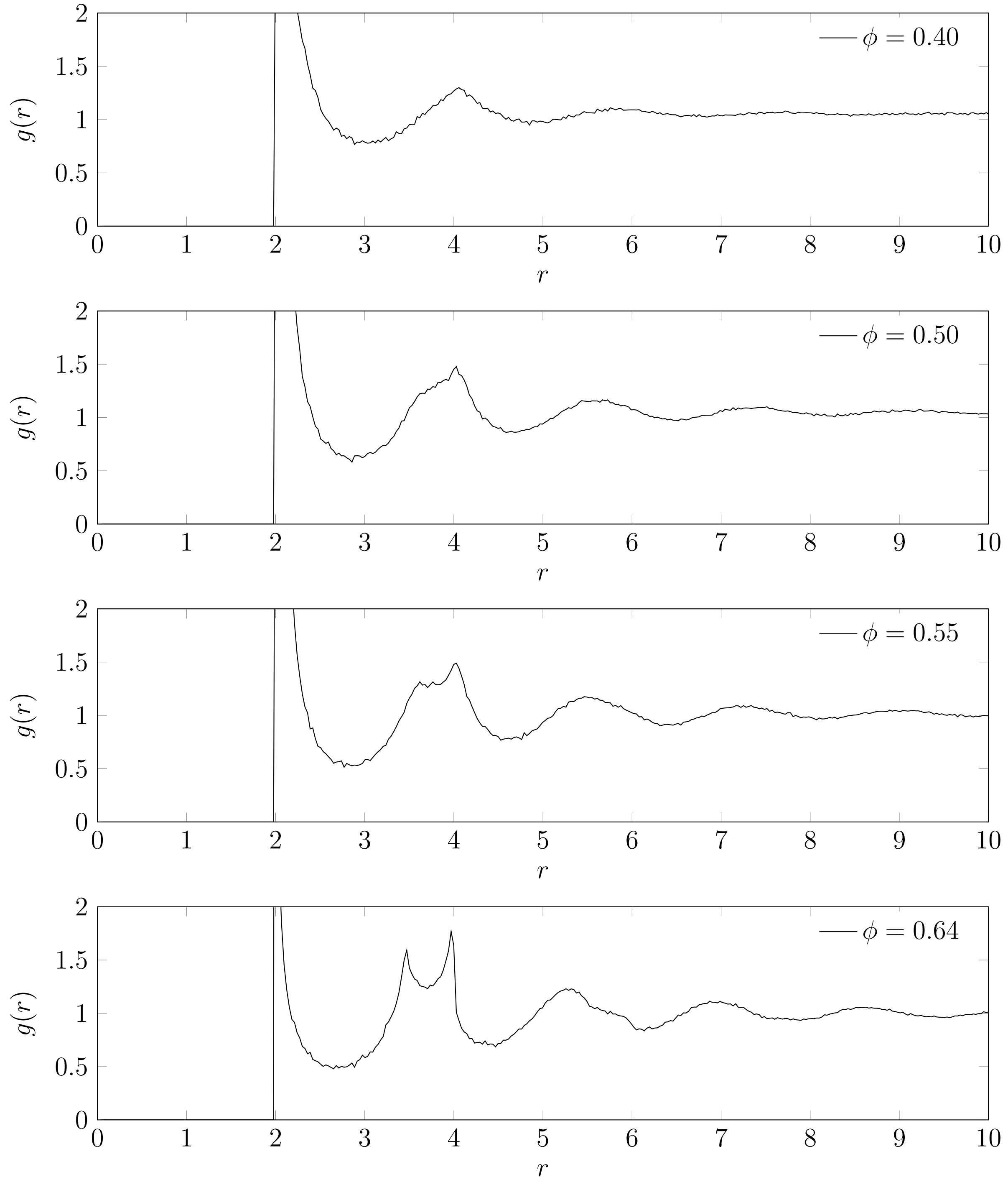


Figure 9.3: Radial distribution functions for a system of $10^5$ spheres.

### *n*-Point Probability Functions

The $n$-point probability functions, as discussed in chapter 3, give the probability of finding randomly placed points inside different phases of a given material. They are particularly useful to determine the minimum size for a system to be statistically homogeneous, but they can also provide information about the shape and polydispersivity of the particles in a packing as well. The $n$-point probability function of monodisperse packings shows more structure corresponding to the shape of the particles. In polydisperse packings, this structure is somewhat smoothed out by the varying sizes, but the exact form of the function contains information about how much of each particle size the packing contains. Figure 9.4 shows the two-point probability function for a packing of $10^5$ spheres, at $\phi = 0.64$. Figure 9.5 shows the three-point probability function at a fixed angle of $\theta = 60°$ between the segments formed by joining the points $x_1$, $x_2$ and $x_3$ and varying distances $r_1 = |x_1 - x_2|$ and $r_2 = |x_3 - x_2|$. At either $r_1 = 0$ or $r_2 = 0$, the three point probability function is equal to the two-point probability function if the extremities of the collapsed segment correspond to the same phase; as one might imagine, it is equal to zero, otherwise. At larger scales, there is no more spatial correlation between the points, so each function converges to the product of the volume fractions of the extremal points, i.e., $S_{111}$ converges to $\phi_1^2$, $S_{000}$ converges to $\phi_0^2$, etc. The distinctive features in the two- and three-point probability functions of monodisperse spheres are the decaying oscillations that match the scale of the particles themselves.

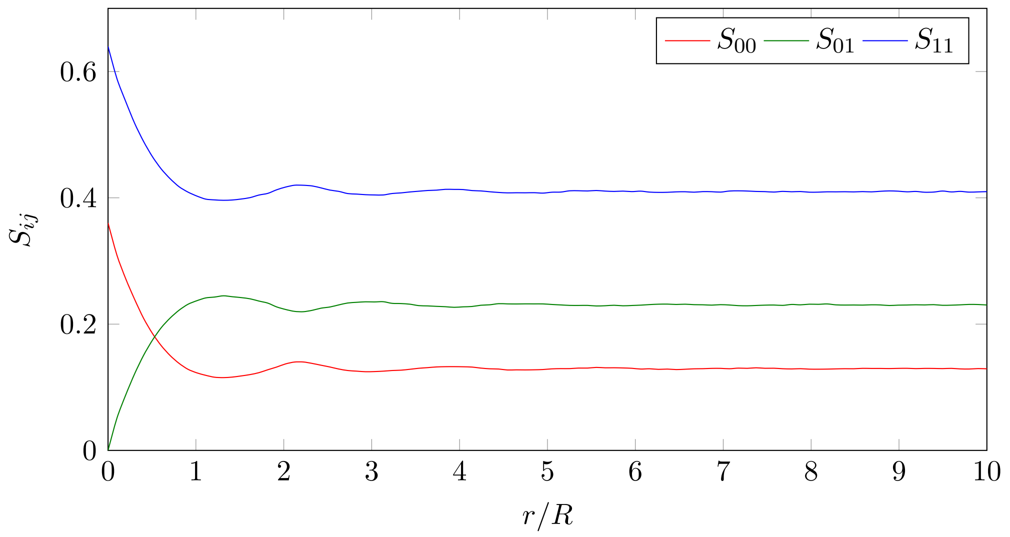


Figure 9.4: Two-point probability functions for a packing of $10^5$ spheres.

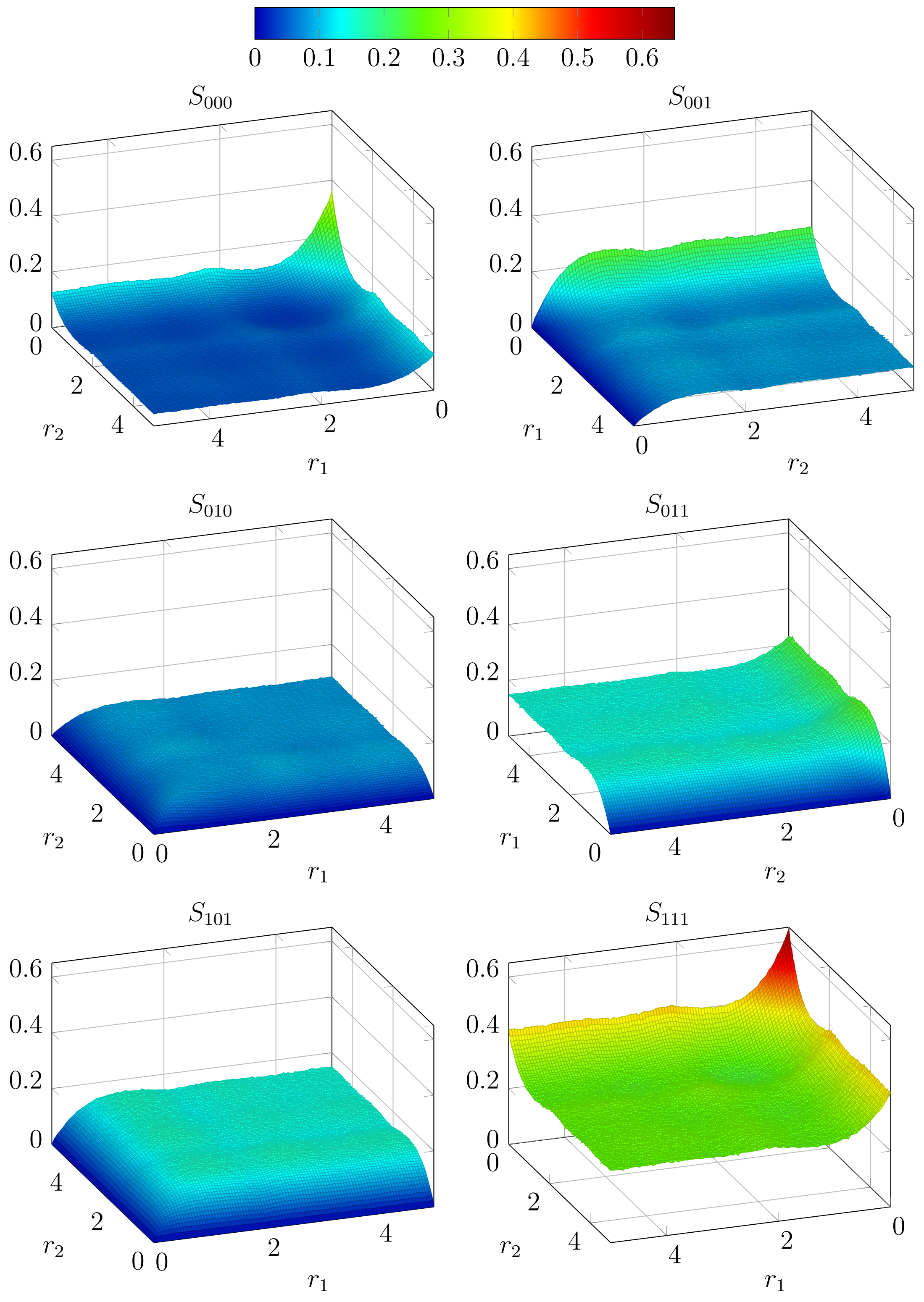


Figure 9.5: Three-point probability function for a packing of $10^5$ spheres ($\theta = 60°$).

## 9.3 Polydisperse Packings of Hard Spheres

Monodisperse packings of spheres constitute a very idealized model for materials. In practice, synthetic “monodisperse” particles have a continuous size distribution. For this reason, polydisperse packings of spheres offer a better model for many systems of technological significance such as colloids [151], powders, foams, packed beds for fluid flow analysis, and solid rocket propellants. However, despite the small step in complexity from monodisperse to polydisperse packings of spheres, the proportion of studies devoted to latter is rather small. The determination of structural and statistical properties of polydisperse packings of spheres hence remains relatively unexplored. An example of this is the optimal packing fraction for bidisperse packings of spheres, for arbitrary values of the radii $r_1$ and $r_2$ of the spheres and their partial concentrations, which was only explored in depth very recently [152, 153]. Another interesting question is the determination of the RCP limit for polydisperse packings of spheres with continuous distributions, as well as their statistical properties. For instance, it has been suggested that even a small amount of polydispersivity can suppress the disordered to ordered phase transition observed for monodisperse packings [154]. The structural stability and jamming properties can be similarly affected by polydispersivity. The first comprehensive study of bidisperse packings of spheres—cited by Torquato in his review of hard-particle packings and applications [147]—has been reported by Hudson and Harrowell [152]. They determine optimal packing fractions of bidisperse systems composed of small spheres packed into the interstices of crystalline lattices of large spheres for a variety of tiling lattices generated from regular polyhedra. Hopkins, Stillinger, and Torquato later explored binary packings of spheres further. Using the Torquato–Jiao packing algorithm, they found the densest binary packings with minimal bases of 12 spheres or fewer for a wide range of size ratios and partial concentration of small spheres [153]. They then utilized the same algorithm to study disordered strictly jammed binary packings [155]. In a more recent study, Baranau and Tallarek [156] determined the random-close packing limits of polydisperse packings of spheres with lognormal size distributions.

Since polydispersivity is indispensable for simulating the microstructure of heterogeneous materials, our packing code supports both particles with fixed sizes, as well as sets of particles with continuous size distributions. In the next sections, we provide some brief remarks about how polydispersivity affects the packing density and the radial distribution function of the RCP states of bidisperse and polydisperse packings by reproducing some of the results of the most recent studies on disordered packings by Hopkins *et al.* [155], and Baranau and Tallarek [156].

### 9.3.1 Polydisperse Packings with Continuous Size Distributions

We would like to know the effect that a continuous distribution of particle sizes can have on the characteristics of hard-sphere packings. We use packings containing from $10^3$ to $10^5$ spheres with a lognormal size distribution to determine their random-close packing limits and radial distribution functions. The simulations are run until packings become virtually jammed—at which point the collision rate exceeds $10^{13}$ collisions per unit time in the simulation. The random-close packing limits are averaged over 10 runs with the same conditions. Rattler particles are not excluded from the packings when computing packing densities. All sphere packings were generated with our Lubachevsky–Stillinger packing code keeping the growth rate of the particles constant. The lognormal distribution has the form

$$p(x|\mu,\sigma) = \frac{1}{\sigma x\sqrt{2\pi}} \exp[-(\ln x - \mu)^2/2\sigma^2].$$

We used a constant $\mu = 1.0$ and varied the standard deviation of the lognormal distribution from $\sigma = 0.01$ to $\sigma = 0.38$. Our code also supports other particle size distributions, such as uniform, Gaussian, Weibull, etc, but we will limit the current discussion to particles with a lognormal distribution.

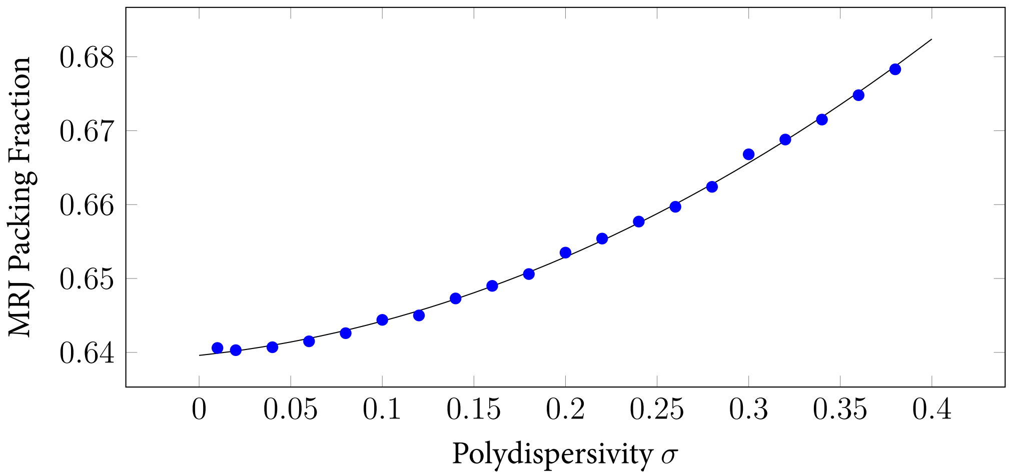


Figure 9.6: Maximum packing fraction for polydisperse packings of spheres as a function of the polydispersivity.

The random-close packing fractions are shown in figure 9.6. We observe that the maximum packing density increases as the standard deviation of the size distribution increases. The solid line in figure 9.6 was obtained with a simple polynomial fit to the data. Farr and Groot have developed a theoretical expression for the maximum packing density of particles with a lognormal distribution

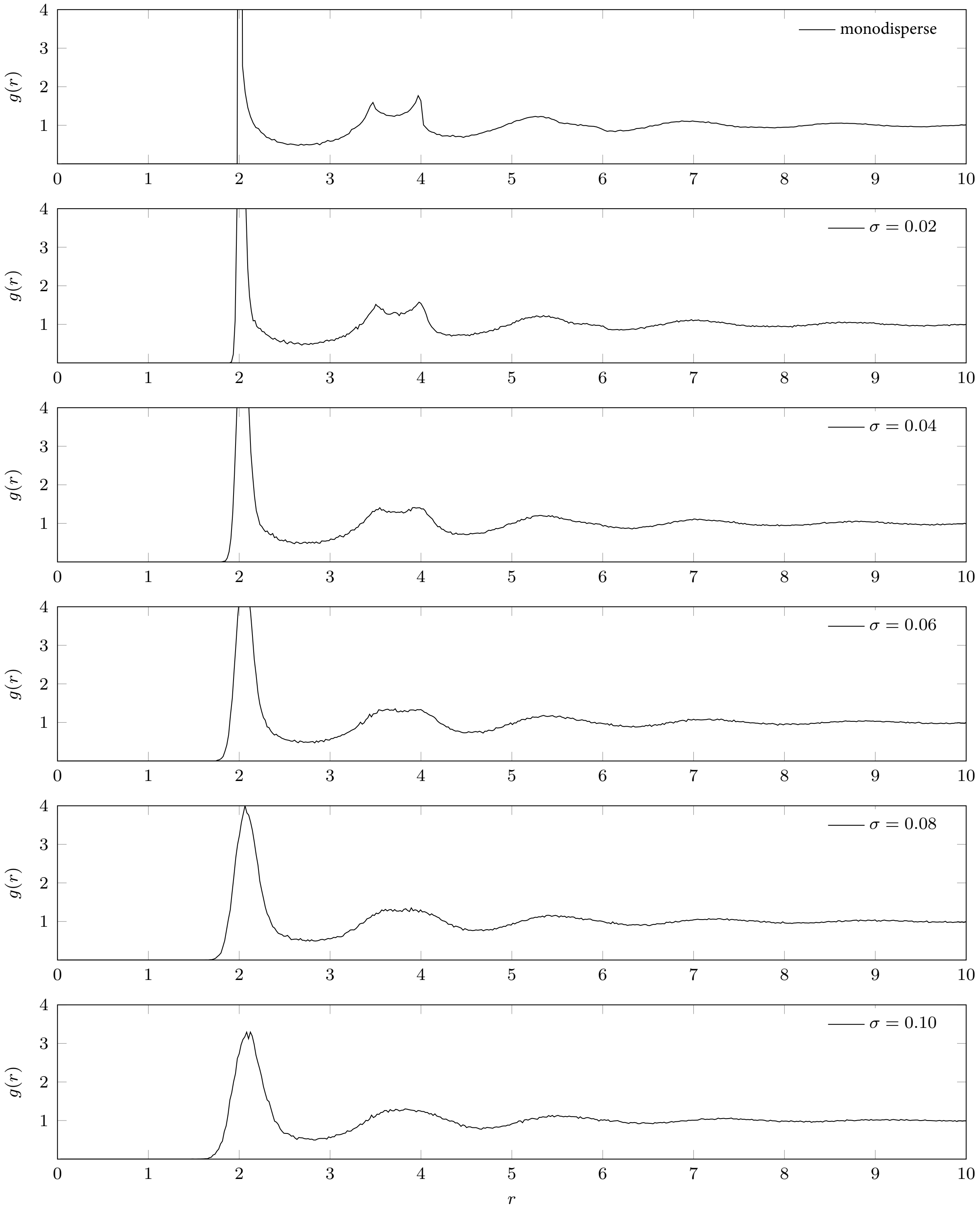


Figure 9.7: Variation in the radial distribution function due to polydispersivity.

in [157]. Our results are compatible with both the results of Farr and Groot [157], and those by Baranau and Tallarek [156] for similar packings generated with both a force-bias method and the Lubachevsky–Stillinger algorithm. While the difference in packing density due to polydispersivity is fairly small, the differences in the radial distribution functions are much more pronounced. The sharp peaks observed in jammed packings of monodisperse spheres are quickly smoothed out as the polydispersivity increases, as shown in figure 9.7. The broadening of the first peak at $r = 2$ is a consequence of the varying radii of the particles. Since the works cited above only use a few thousand particles in their simulations, however, no data exists in the literature for comparing the radial distribution functions.

### 9.3.2 Binary Packings

Binary packings of spheres are particularly interesting because the densest packings can correspond to the arrangement of atoms in binary solids with steep mutual repulsion and for matter in high temperature and pressure conditions. In [152], Hudson and Harrowell performed the first extensive survey of the maximum packing fractions of binary packings of spheres on several lattices based on polyhedral tilings of space using analytical methods. The highest packing fraction they report in their work is $\phi = 0.84849$, for a tiling lattice based on octahedra and truncated cubes. The highest possible density for binary packings, in the limit that the size ratio approaches infinity, is $\phi = 1 - (1 - \phi_{\text{fcc}})^2 = 1 - (1 - \frac{\pi}{3\sqrt{2}})^2 \approx 0.93265$. This is the density for an hcp packing of large spheres in which interstices are filled with an hcp packing of infinitesimally small spheres. A more thorough investigation of binary packings using the Torquato–Jiao packing protocol, by Hopkins *et al.* [153], led to the discovery of new alloy structures that could correspond to yet unidentified stable configurations of binary atomic and molecular materials. They provide a detailed picture of the phase diagram of binary matter and the maximal packing surface for size ratios in the range $0.2 \leq \alpha \leq 0.66$ and partial concentrations $x < 11/12$. It is important to note that for $\alpha > 0.66$ the densest arrangements are always phase-separated Barlow packings (such as hcp and fcc) of each sphere type. In a later work, Hopkins *et al.* explore the properties of disordered binary packings. They report that disordered binary packings of spheres can be produced in a wide range of average packing densities $0.634 \leq \phi \leq 0.829$ for small to large sphere ratios $\alpha \geq 0.100$.

Here, we show the average packing density of binary packings of various size ratios for equal *volume* fractions of large and small spheres and compute the radial distribution function of a large binary packing containing $10^6$ spheres. To compute the average packing density as function of size ratio, we generated packings containing between $2.1 \times 10^3$ to $1.26 \times 10^5$ spheres for size ratios

$0.30 \leq \alpha \leq 0.95$, in steps of $0.05$. The total number of spheres was chosen such that the minimum number of large spheres was always equal to or larger than $10^3$. The average packing fraction as a function of size ratio is shown in figure 9.8.

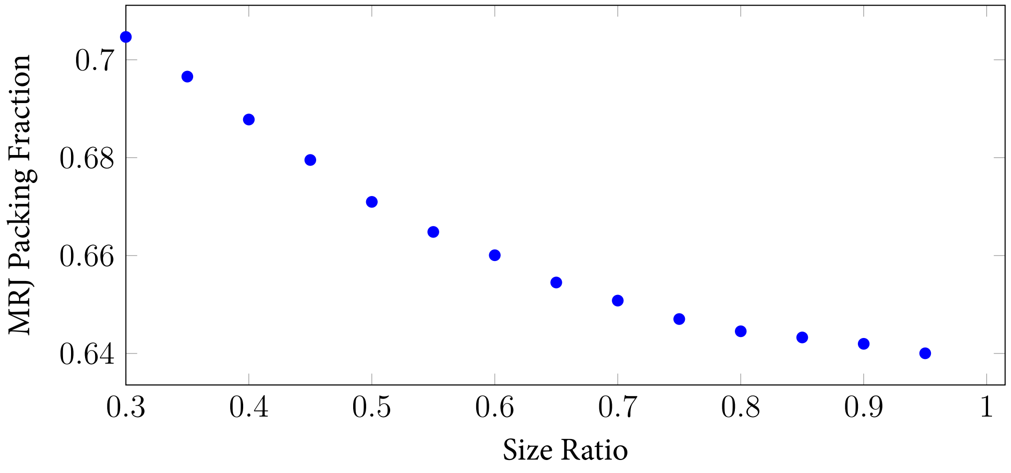
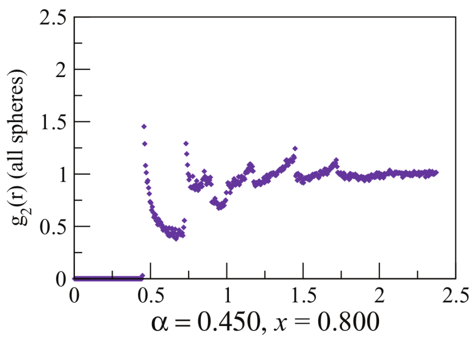
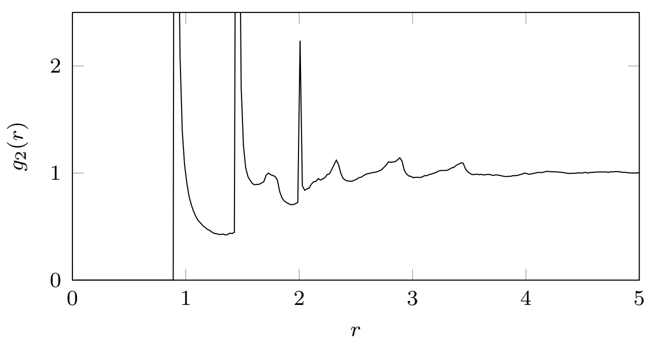


Figure 9.8: Maximum packing fraction for binary packings of spheres as a function of the size ratio.

For the radial distribution function, we use a single packing containing $10^6$ spheres with a size ratio between large and small spheres $\alpha = 0.45$, and small sphere partial concentration $x = 0.8$. The packing is shown in figure 9.11, and its packing density is $\phi = 0.6772$. The radial distribution function calculated for our packing is compared with the same function reported in [155] in figure 9.9. In addition, we provide separate radial distribution functions for small and large spheres. The radial distribution function reported in [155] has been computed from the ensemble averaging of ten packings of $10^3$ spheres. The high-performance of our code enables us to use many more particles to calculate the radial distribution function with high resolution without the need for any ensemble averaging. The $x$-axis is different because our plots are in units of the radius of the larger sphere, rather than the diameter. In their work, Hopkins *et al.* [155] argue that the split peaks commonly found in monodisperse packings of spheres are not present in binary packings, indicating a fundamental difference in the structure of binary and monodisperse packings. However, we do observe peaks that correspond to similar structures as those represented by the split peak of monodisperse packings in our high-resolution calculation of the radial distribution function. As remarked by Hopkins, nevertheless, there is a preference for collinearity between spheres that is clearly evident by prominent peaks at distances corresponding to aligned spheres.



(a) RDF, from [155]



(b) RDF, our work

Figure 9.9: Comparison of radial distribution function for all spheres of binary packing of spheres with $\alpha = 0.45$ and $x = 0.8$.

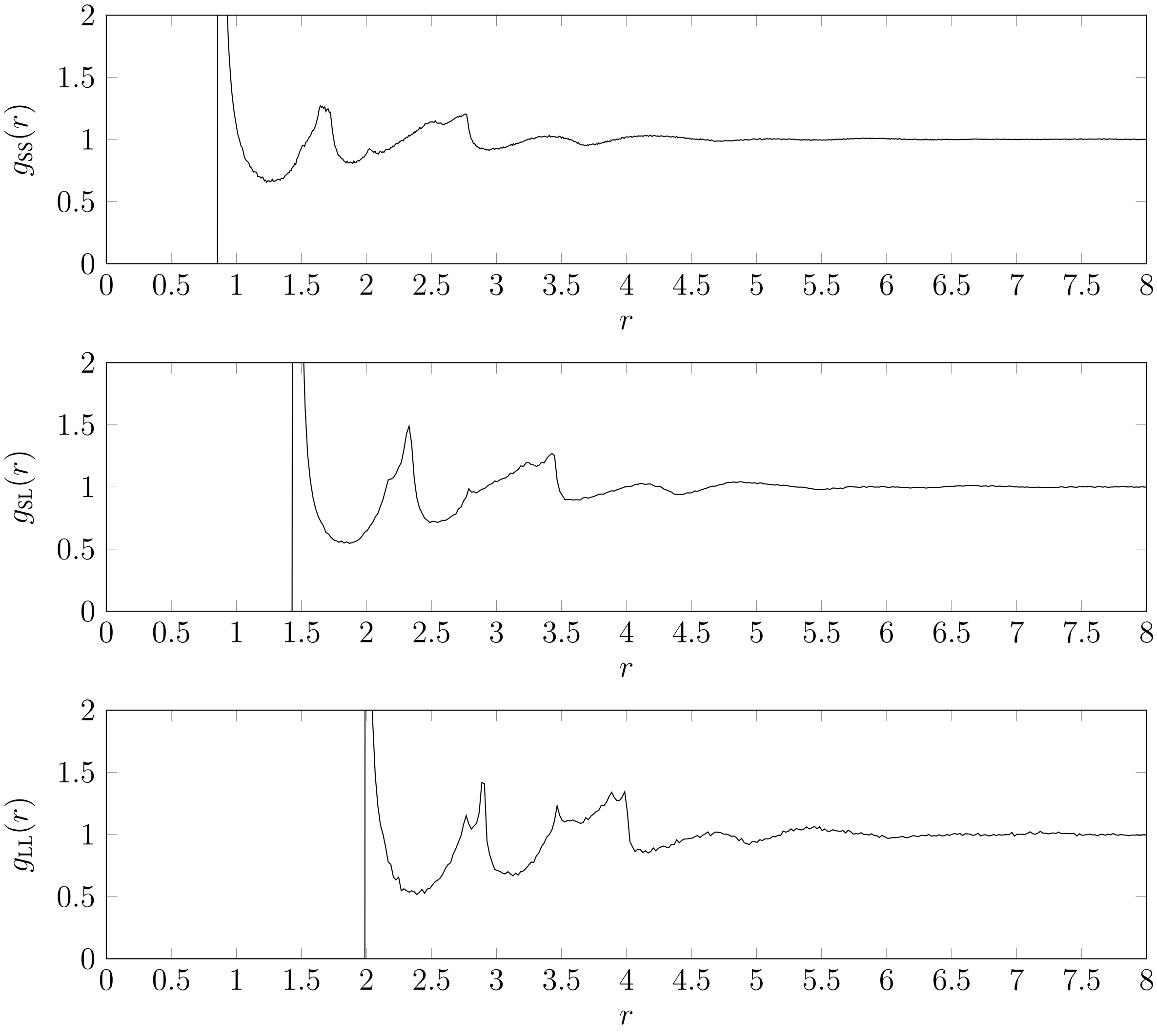


Figure 9.10: Radial distribution functions of a binary packing of $10^6$ spheres with size ratio $\alpha = 0.45$ and small sphere partial concentration $x = 0.8$.

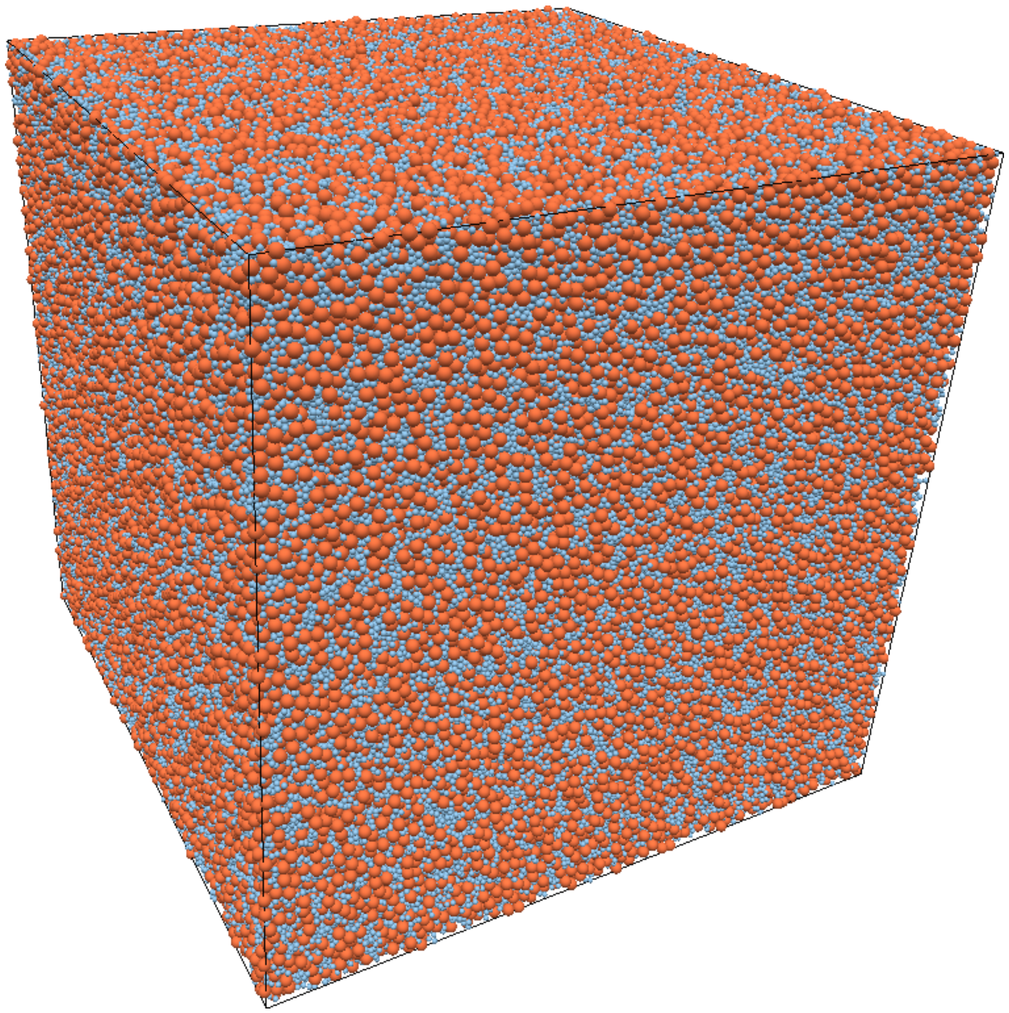


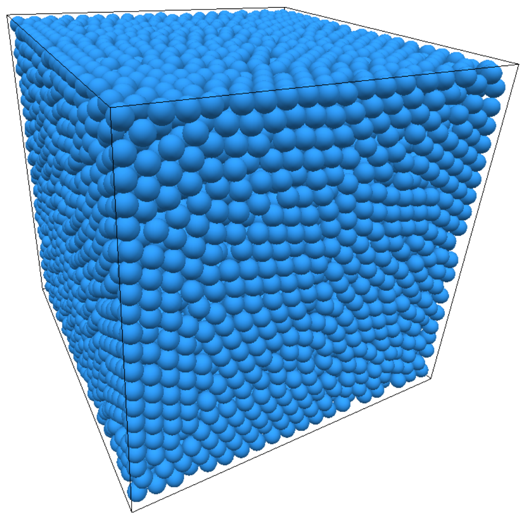
Figure 9.11: Binary packing of $10^6$ spheres at packing density $\phi = 0.6772$ ($\alpha = 0.45$ and $x = 0.8$).

## 9.4 Packings of Spheres in Finite Containers

In packings we have considered heretofore, the boundary conditions have always been periodic. Nevertheless, there are interesting problems that require packings simulated inside a finite region—as is the case in fluid flow through packed beds, for example. In order to simulate how water would flow through a filter consisting of a narrow channel packed with particles, the boundary of the channel needs to be properly taken into account. The local packing density in the vicinity of the solid walls of the channel, if very different from the average density, may affect the overall porosity of the channel. A study of these effects is presented by Khirevich in his PhD work [135] using rectangular, circular, and trapezoidal channels filled with monodisperse spheres. Since we have not yet developed computer codes for the simulation of fluid flow through packed beds in three dimensions, we compare here some of the characteristics of packings with solid boundary with periodic packings. More specifically, we look at the local packing densities and the volume distribution of the Voronoi cells defined by the particles in the packings.

An interesting question—and perhaps also one of the most important—is how thick is the layer of particles under the effects of the solid boundary. To answer this question, we use several packings of $10^4$ spheres inside boxes with solid and periodic boundaries, in a cylinder, and an annulus. Figure 9.12 shows two packings of $10^4$ monodisperse spheres in boxes with solid and periodic boundary conditions. Figure 9.13 shows packings of $10^4$ monodisperse spheres in a cylinder and an annulus. The local packing density, integrated along the $z$-axis, is shown in figure 9.14. Along with the four packings from figures 9.12 and 9.13, it shows two polydisperse packings of spheres with a lognormal particle size distribution. As we can see from these plots, the monodisperse packings inside solid boundaries have a clearly visible boundary layer of ordered particles that change the local density. From figures 9.15, that shows the local packing fraction as a function of the position inside the packing along the $x$-axis, we can see that the boundary layer extends into the packing by a distance roughly equivalent to 4 particle diameters. In the case of the annulus, since there are boundaries on both sides, the whole packing is ordered in radial layers of particles, as the packing itself is less than 8 particle layers thick. This is also visible in figure 9.16, where the local packing fraction is plotted along the center line of the cylindrical and annular packings. From figures 9.14 and 9.15, we can also conclude that the thickness of the boundary layer is reduced with polydispersivity.

A distinct measurement of the heterogeneity of the packings can be obtained from the distribution of the Voronoi cell volumes of each particle in each packing. Figure 9.17 shows this volume distribution for the two monodisperse packings inside solid and periodic boxes. The long tail of large Voronoi cell volumes in the solid box packing corresponds to the loosely packed cells of the particles near the boundary. On the other hand, the Voronoi cell volume distribution for the periodic packing is much narrower, indicating that it is packed more homogeneously.


(a) Solid Boundary

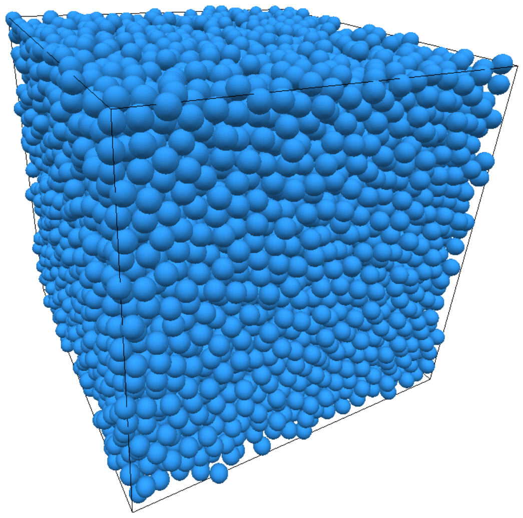
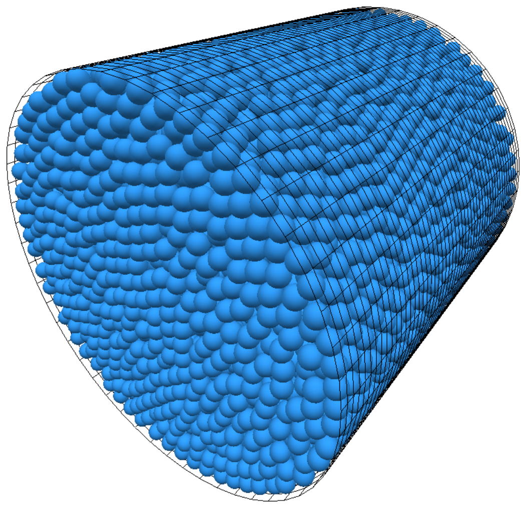

(b) Periodic Boundary

Figure 9.12: Packings of $10^4$ monodisperse spheres in: (a) a solid box; and (b) a periodic box.


(a) Cylinder

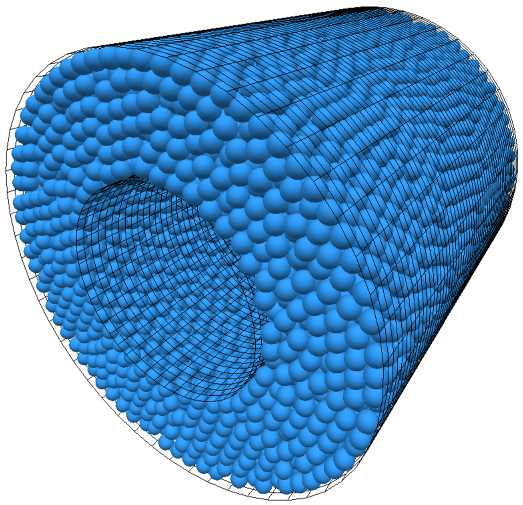

(b) Annulus

Figure 9.13: Packings of $10^4$ spheres inside cylinder and annulus.

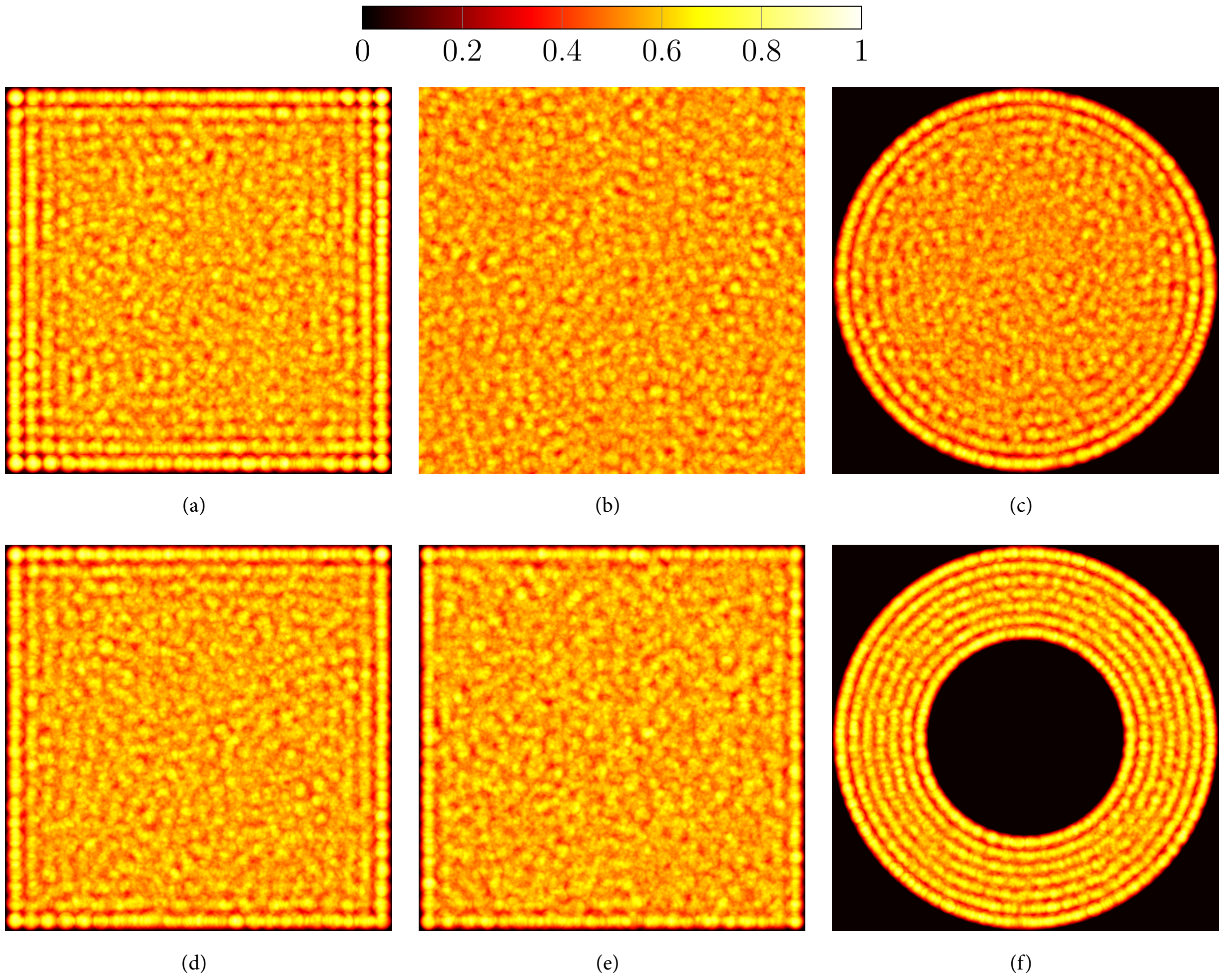


Figure 9.14: Local packing density, integrated along $z$-axis for $10^4$ monodisperse spheres in: (a) a solid box, (b) a periodic box, (c) a cylinder; $10^4$ polydisperse spheres with a lognormal size distribution of (d) $\sigma = 0.1$ and (e) $\sigma = 0.2$ in a solid box; and (f) $10^4$ monodisperse spheres in an annulus.

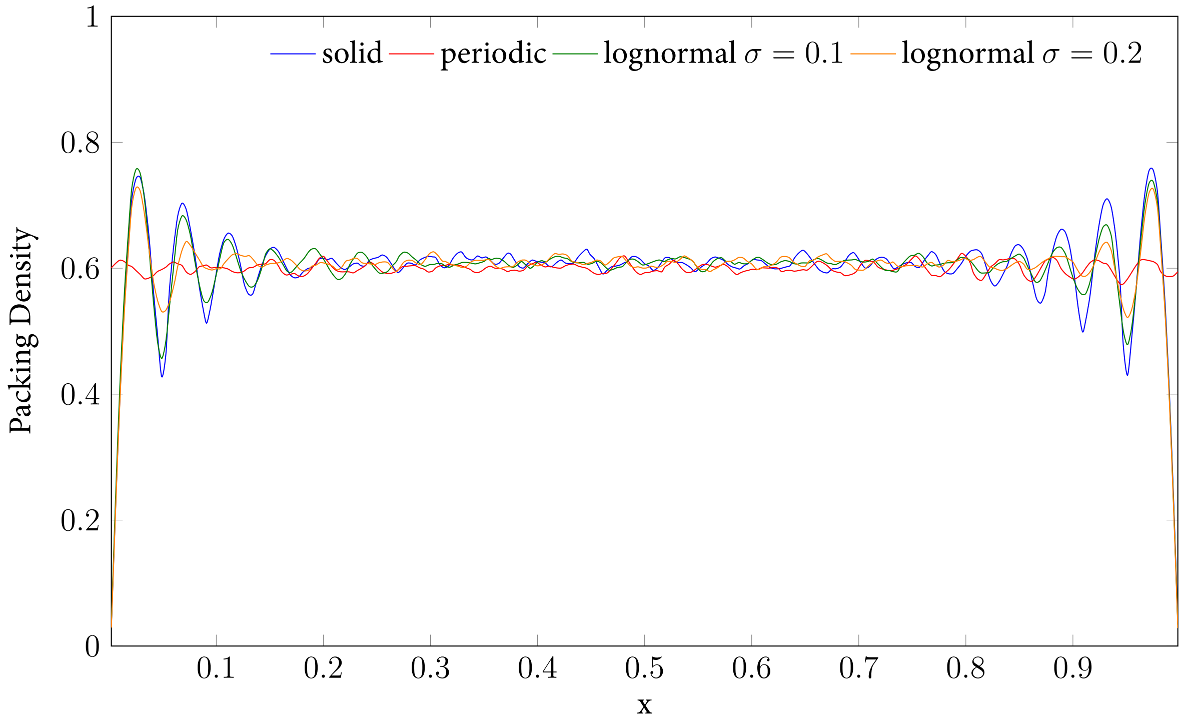


Figure 9.15: Local packing density integrated in two dimensions.

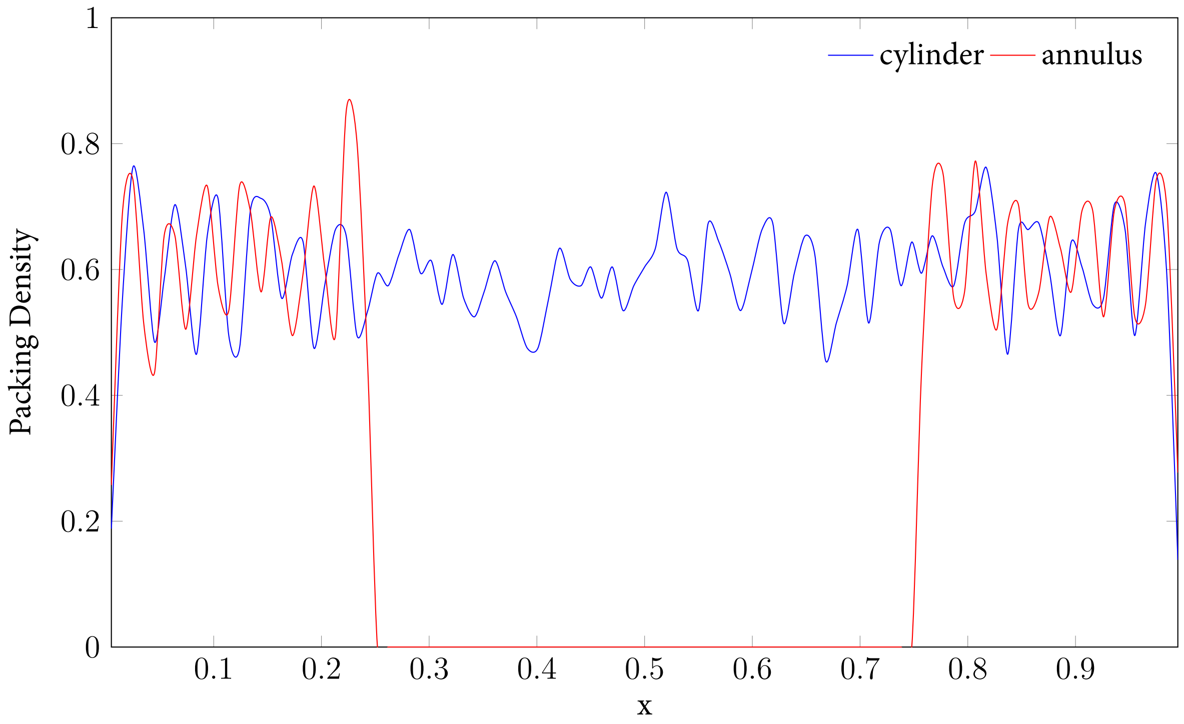


Figure 9.16: Local packing density across center line of the cylinder and annulus, integrated only in the axial direction.

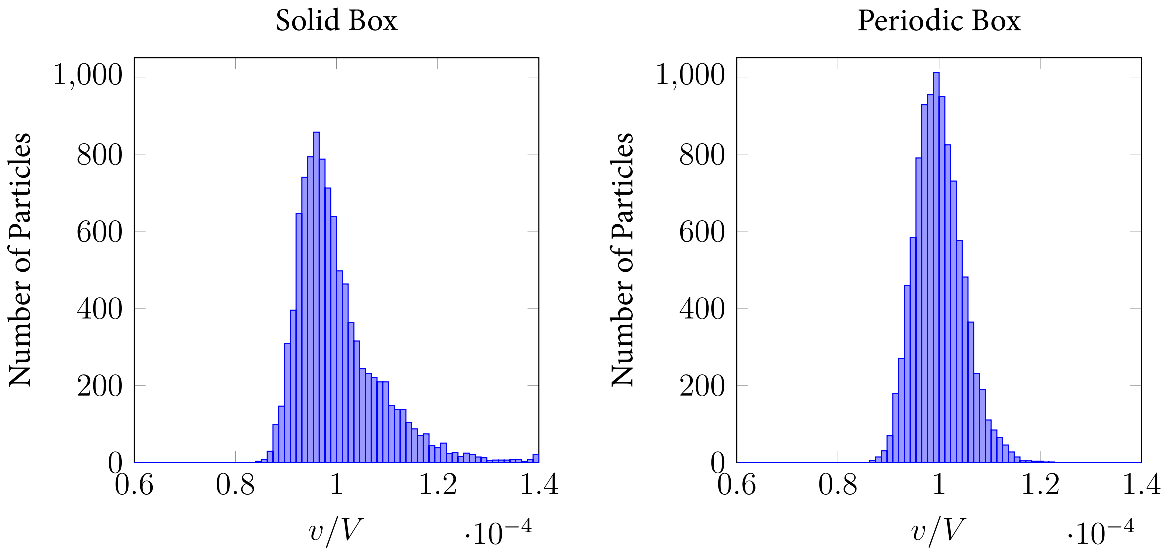


Figure 9.17: Voronoi cell volume distribution for packings of monodisperse spheres inside solid and periodic boxes.

# Chapter 10

# Packings of Polyhedra

## 10.1 Introduction

Whilst packings of spheres are relatively straightforward to generate, packing nonspherical particles involves considerably more effort due to the extra complexity introduced by the new rotational degrees of freedom. Even particles that could be considered small deviations from spheres, such as nearly spherical ellipsoids, make it necessary to abandon the simple algebraic formulas in favor of complex algorithms when computing intersections and predicting collisions between particles. Moreover, if particles are significantly elongated, efficient algorithms for minimizing the number of collision checks against neighboring particles also become more complicated compared to algorithms for monodisperse—or even polydisperse—systems of spheres. Nevertheless, even if these obstacles seem discouraging, they have not prevented researchers from searching for answers for the many interesting mathematical questions that arise when packing particles with shapes other than spheres. In this chapter, we will discuss packings of convex polyhedra. It is easy to find studies on smooth non-spherical particles in the literature [46–49, 75, 86, 87, 147, 158, 159], and our code does currently support at least ellipsoids and cylinders. Polyhedra, however, are particularly interesting for us because many of the explosive materials used as oxidizers in solid rocket propellants are powders whose particles have crystalline polyhedral shapes.

Packings of polyhedra have only begun to be explored recently, in the last couple of decades [51, 53–62, 89, 147, 160–165]. Polyhedra themselves, however, have been known and studied since ancient times. The Platonic solids, for example, were extensively studied by the ancient Greek—they were described by Plato in his *Timaeus*, around 350 BC. Contrary to what one might expect, however, they were not discovered by Plato, but predate him by a whole millennium! The late neolithic people of Scotland are believed to be the first to produce the Platonic solids. They carved these solids and similar shapes in ornamental stones, some of which are now kept in the Ashmolean Museum in Oxford [166].

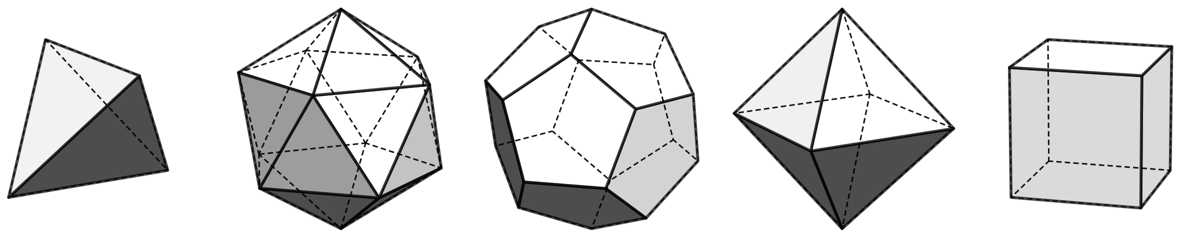

(a) Tetrahedron (b) Icosahedron (c) Dodecahedron (d) Octahedron (e) Cube

Figure 10.1: The five Platonic solids.

The Platonic solids are also commonly referred to as regular solids or regular polyhedra. They are the set of polyhedra whose faces are congruent regular polygons. Euclid, in his book *Elements*, proved that there are exactly five Platonic solids in three dimensions: the tetrahedron, the icosahedron, the dodecahedron, the octahedron, and the cube. Schläfli [167] later proved that there are six regular solids in four dimensions, three in five dimensions, and three in all higher dimensions.

In ancient times, Plato associated the Platonic solids with the classical elements of fire (tetrahedron), water (icosahedron), air (octahedron) and earth (cube). He even provided some intuitive reasoning for his associations—e.g., earth's solidity means that it is made of cubes, since this is the only regular solid that tiles space, while tetrahedra are sharp and stabbing, like fire. Kepler was also enchanted by the regularity of the Platonic solids, and attempted to relate them to the orbits of the known planets at the time, before the real laws that govern the orbits of the planets were discovered.

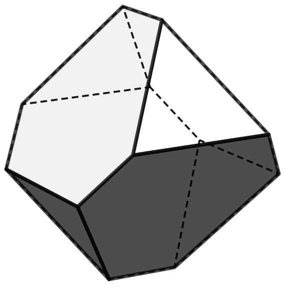
A basic geometric characteristic of a Platonic solid is its dihedral angle, i.e., the interior angle between any two faces, which is equal to

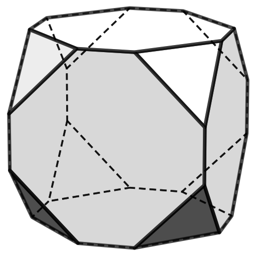
$$\sin(\tfrac{\theta}{2}) = \frac{\cos(\pi/q)}{\sin(\pi/p)},$$

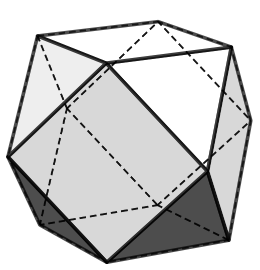
where $p$ and $q$ are the number of sides of each face and the number of faces meeting at a vertex, respectively. Since the dihedral angle of the cube is the only submultiple of $2\pi$, the cube is the only Platonic solid that can tile space. In fact, there are an infinite number of irregular tessellations of space by cubes (where the cubes are not aligned, but each layer is shifted with respect to the previous).

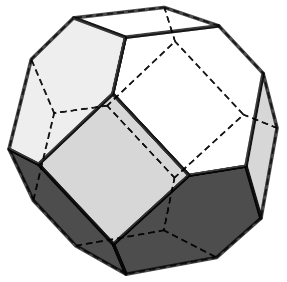
The Platonic solids are composed of only one type of regular polygon meeting at identical vertices. If we relax this requirement to allow different types of regular polygons for each face, we obtain a different family of highly symmetric semi-regular polyhedra named after their discoverer, the Greek mathematician Archimedes. Archimedes is generally considered to be one of the greatest mathematicians of all time. He was the first to calculate an accurate approximation of $\pi$, and after the discovery in 1906 of the Archimedes Palimpsest [168], it became clear that he had already begun to develop the basic ideas of calculus almost two millennia ahead of Newton! Unfortunately, unlike his inventions, his written mathematical work was little known in his time, and few copies of it survived through the Middle Ages. The Archimedean solids (shown in figure 10.2) are 13 in total: truncated tetrahedron, truncated icosahedron, snub cube, snub dodecahedron, rhombicosidodecahdron, truncated icosidodecahdron, truncated cuboctahedron, icosidodecahedron, rhombicuboctahedron, truncated dodecahedron, cuboctahedron, truncated cube, and truncated octahedron.

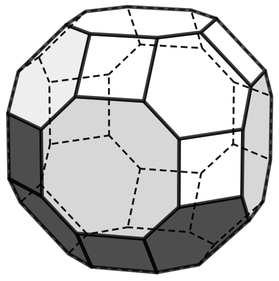
In geometry, all polyhedra are associated into pairs called *duals*, in which all of the vertices and faces are interchanged. Every polyhedron has a dual, and the dual of its dual is the original polyhedron. The dual of a Platonic solid is another Platonic solid. The icosahedron and dodecahedron form a dual pair; the cube and octahedron form another dual pair; and the tetrahedron is self-dual. The dual polyhedra of the Archimedean solids are called Catalan solids. Archimedean and Catalan solids can both be obtained from regular polyhedron *seeds* (Platonic solids) via geometric operators such as the dual operator, and the truncation operator defined by Kepler, among others. The idea of using these operators was extended by Conway [169], and the resulting notation to describe polyhedra generated from successive operators is thus called Conway polyhedron notation. In this notation, each Platonic solid is described by its uppercase initial, and operators are denoted by lowercase letters, such as 't' for the truncation operator and 'd' for the dual operator. Using Conway polyhedron notation is a very convenient way of creating new shapes for use with our packing code. Many of the shapes we include with it were generated using Conway notation with external tools and later converted to our own input format. Some shapes, however, such as HMX, ADN, and CL-20, were created in the past [162] from crystals of these explosive materials for use in packings of particles to model the microstructure of solid propellants [40].

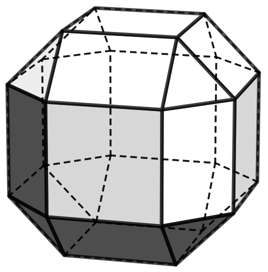

(a) Truncated Tetrahedron

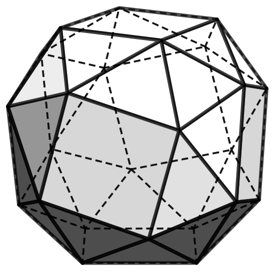

(b) Truncated Cube

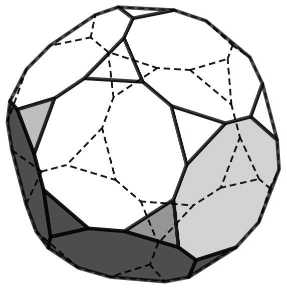

(c) Cuboctahedron

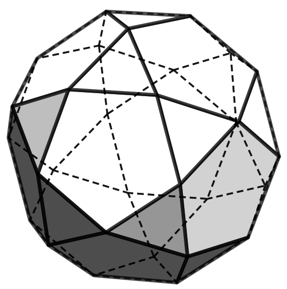

(d) Truncated Octahedron

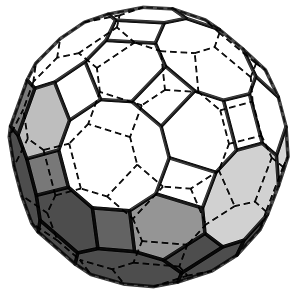

(e) Truncated Cuboctahedron

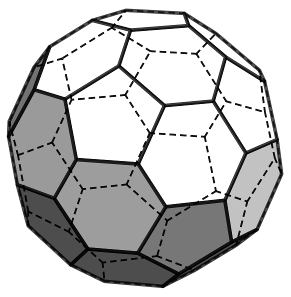

(f) Rhombicuboctahedron

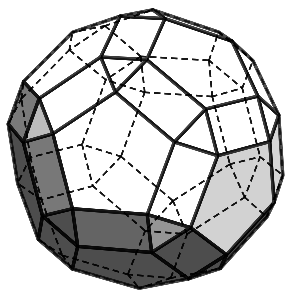

(g) Snub Cube


(h) Truncated Dodecahedron


(i) Icosidodecahedron


(j) Truncated Icosidodecahedron


(k) Truncated Icosahedron


(l) Rhombicosidodecahedron

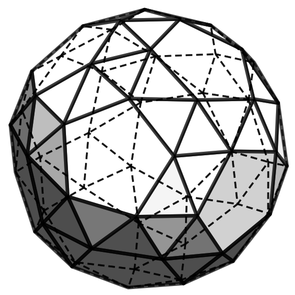

(m) Snub Dodecahedron

Figure 10.2: The Archimedean solids.

## 10.2 Packings of Regular Polyhedra

Until some recent developments in the algorithms to generate packings of polyhedra, little was known about the densest packings of polyhedral particles. In one of the early works, Betke and Henk [160] found the densest *lattice* packings of the Platonic and Archimedean solids analytically. It is conjectured by Torquato *et al.* [51, 52], that Bravais lattice packings are the densest possible arrangement for centrally symmetric convex polyhedra. However, for polyhedra without central symmetry, Bravais lattices are usually not the optimal configuration for maximum packing density. That is the case of the tetrahedron and truncated tetrahedron, for example. Few studies provide systematic investigations of packings of polyhedra. Damasceno *et al.* [57] seem to be the first to study computationally how polyhedra self-assemble into complex structures such as glasses, liquids, and crystals, depending on their shape. Chen *et al.* [165] recently combined analytic calculations and Monte Carlo simulations of three families of two-parameter polyhedra to study their optimal packing-density surfaces. On the experimental side, Henzie *et al.* [60] used silver nanocrystals to provide additional insights into self-assembly. Many questions regarding packings of polyhedra, however, remain wide open.

The main application of the particle packing code that we have developed is in the generation of realistic computer models of energetic materials. Nevertheless, it can produce packings of any mixture of convex particles with various shapes and sizes, so it is useful to explore many of the unanswered problems concerning packings of polyhedra and much more. However, since our code does not use a deformable boundary, it is not possible to use it to determine the crystal structure of nonorthogonal Bravais lattices, although it can be used to create packings of any Bravais lattice in general. Packings of polyhedra generated with our code use a periodic box boundary (other boundary types are not yet supported), and the initial condition is a dilute packing of infinitesimal particles. In the following sections, we discuss the most relevant results concerning packings of the Platonic and Archimedean Solids and present packings generated with our code. It is possible to mix different particle types and different particle sizes, but in this chapter we will discuss only packings of identical particles.

### 10.2.1 Platonic Solids

#### Tetrahedron

The tetrahedron is without doubt the most interesting among the Platonic solids when packing is concerned. Since the tetrahedron is not centrally symmetric, its densest lattice packing is only

$\phi \approx 0.36$, while much denser ($\phi > 0.85$) disordered packings have been reported in the literature. Tetrahedra are notoriously difficult to pack, so it is not surprising that determining the highest packing density of tetrahedra in three dimensions is still an open question. Nevertheless, a lot of progress has been made to answer this question in the last few years. Torquato and Jiao [62] provide an interesting account of the latest developments in this area. The first dense non-Bravais packings of tetrahedra to be reported were the so-called "Welsh" packings, with a packing density of $\phi = 0.708333$ and 34 particles in the fundamental cell. At a slightly higher density ($\phi = 0.716559$) are icosahedral packings, in which tetrahedra are roughly arranged into imaginary icosahedra and packed in groups of 20. After experiments had shown that tetrahedral packings could form dense jammed configurations with a packing density of about $\phi = 0.75$ [59], a dense periodic lattice packing of tetrahedra with $\phi = 0.7786$ density was discovered via computer algebra [170]. Using their adaptive-shrinking-cell (ASC) optimization scheme, Torquato and Jiao improved the density until they finally reached about $\phi = 0.823$. To their surprise, the resulting packing did not present any long-range order. Haji-Akbari *et al.* [50] broke that record soon later using Monte Carlo simulations to construct a quasi-crystalline packing of tetrahedra with a density of $\phi = 0.8503$. Despite all effort employed to generate these disordered dense packings, subsequent improvements to the packing density of tetrahedra by Kallus *et al.* [61] were based on a simple uniform packing with high symmetry with only four particles in the fundamental cell. The packing density of this configuration is $\phi = \frac{100}{117} \approx 0.854700$.

Further parametrizations by Torquato and Jiao [63], and Chen [64] led to packings with densities of $\phi = 0.855506$ and $0.856347$, respectively. This seems to be the best packing fraction of tetrahedra so far. Monte Carlo methods to generate packings of polyhedra, such as the one used in our code, are of course no match to the specialized analytical methods used for the latest improvements in packing tetrahedra. Haji-Akbari mentions in his work [50] that some runs took close to a month of running time, often with carefully crafted initial conditions. Similarly to what happens with packings of spheres, that usually reach a jammed state with lower than optimal density, we observe that packings of tetrahedra generated with our code jam much earlier than

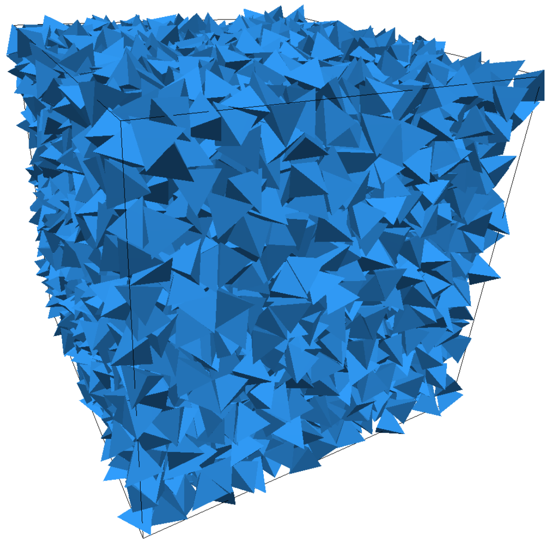


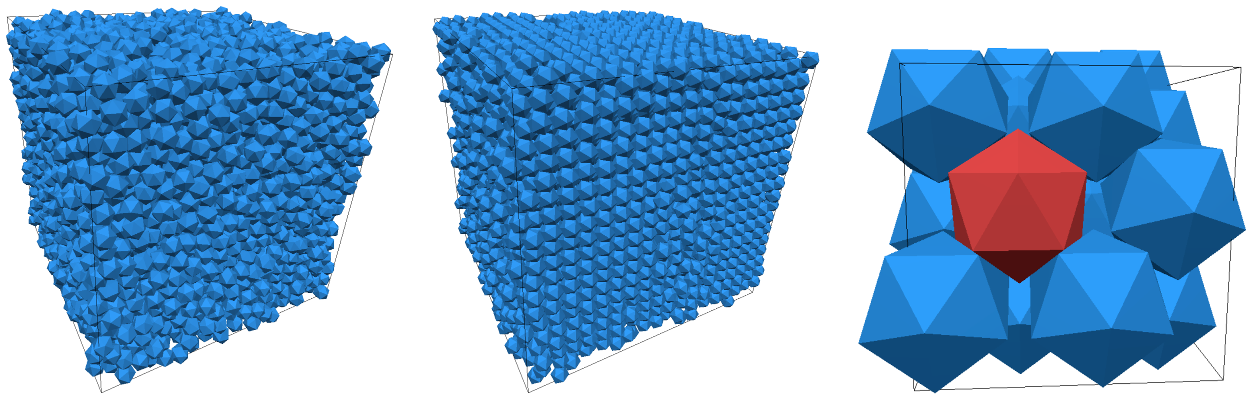
Figure 10.3: Tetrahedra at $\phi = 0.5583$.

even "Welsh" packings. Figure 10.3 shows a packing of tetrahedra generated with our code with a packing density of $\phi = 0.5583$. We do not have sophisticated tools to prepare the necessary initial conditions for denser packings of tetrahedra.

With a smaller number of particles, it is possible to build fundamental cells with higher packing fraction with our code, but reaching densities higher than $\phi = 0.7$ with our code is still fairly difficult (although not impossible). This is certainly an area where we could improve later with more work to determine the best recipe for the trial displacements and their acceptance rules. During development of our code, we noticed that packing performance can be significantly affected by the several parameters involved, such as the number of trial moves per particle, how large the displacements should be, how to rotate particles, how displacements become smaller as density increases, etc. We have used several test cases with different particle shapes and different number of particles to tune these parameters for the best performance overall. However, a more systematic approach could lead to further improvements.

### Icosahedron

The icosahedron is centrally symmetric, so its highest packing density, according to Torquato's conjecture, is $\phi \approx 0.836$. Since the choice of initial conditions is not as crucial in getting dense packings of icosahedra if compared with tetrahedra, we are able to create small packings of icosahedra with a fundamental cell containing 20 particles that are very close to the optimal lattice packing with our packing code. The example shown in figure 10.4 is slightly lower than optimal because one of the icosahedra (highlighted in red) is rotated relative to the others—in other words, there is a lattice defect in the packing.



(a) Disordered ($\phi = 0.6403$) (b) Partially Ordered ($\phi = 0.7006$) (c) Nearly optimal ($\phi = 0.8239$)

Figure 10.4: Packings of icosahedra.

The final state of a packing is highly dependent on the initial conditions. If started completely at random, we observe that icosahedra usually reach a disordered jammed state in the range of $0.64 \leq \phi \leq 0.70$. However, it is possible to obtain denser disordered packings of icosahedra by using lattices as initial conditions. The packing at the center in figure 10.4 was generated with such an initial condition. Torquato and Jiao [54] claim that the MRJ packing density of icosahedra is slightly above $\phi = 0.70$, while our code can only reliably reach up to $\phi = 0.68$ depending on the number of particles. Regardless of the number of particles, the success rate in generating denser packings gets progressively smaller for densities above $\phi = 0.64$. On the other hand, packings with densities up to $\phi = 0.64$ can be generated very reliably, with a success rate of virtually 100%. Similar to spheres, icosahedra can become jammed in a wide range of densities.

### Dodecahedron

Packings of dodecahedra are similar in characteristics to packings of icosahedra. The optimal density for a Bravais lattice packing of dodecahedra is $\phi = 0.904508$, and the MRJ packing fraction as reported by Jiao [54] is $\phi = 0.716$. Figure 10.5 shows packings of dodecahedra generated with our code. Dodecahedra have a higher tendency to get stuck in local-density minima than icosahedra,

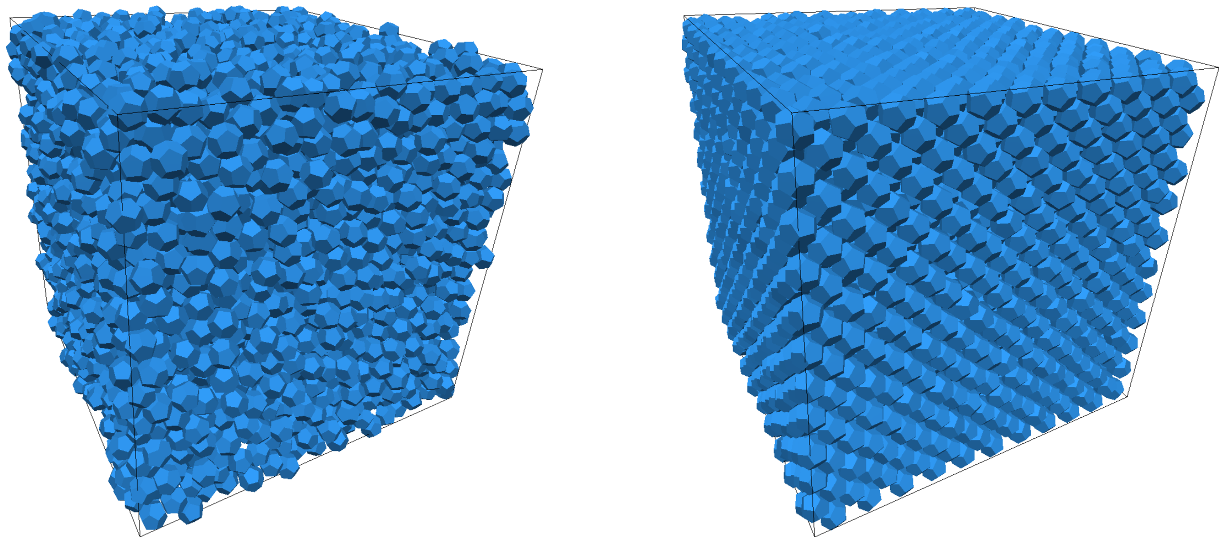
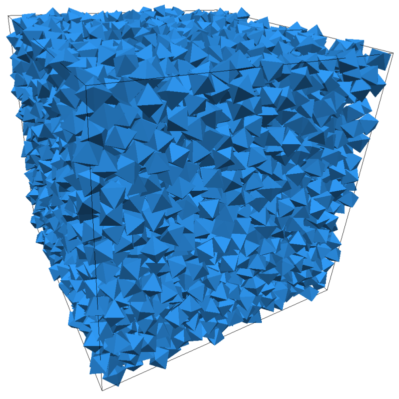

(a) Disordered ($\phi = 0.6070$) (b) Highly Ordered ($\phi = 0.7244$)

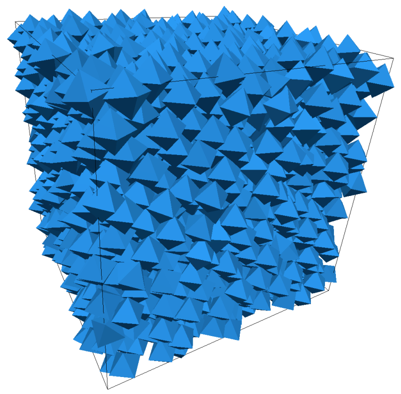
Figure 10.5: Packings of dodecahedra.

so it is difficult to generate near-optimal packings with our code in its current state. In contrast with icosahedra and octahedra, orientation alignment of particles is less likely in packings of dodecahedra. Another point worth mentioning is that packings of icosahedra essentially behave like a hard-sphere fluid at low densities—the asphericity of icosahedra only plays a role when the packing is already near its jamming point. For packings of dodecahedra, the asphericity begins to influence the structure of the packing somewhat earlier. Dodecahedra tend to form face-to-face contacts that force jamming at lower densities than for icosahedra and octahedra when particles are started at random positions and random orientations. When we use lattices as initial conditions, we can easily achieve the reported MRJ packing density for dodecahedra, as seen in the packing on the right in figure 10.5, which has a packing density of $\phi = 0.7244$. We believe that with especially crafted routines for the Monte Carlo trial displacements our code could be able to overcome local minima and produce packings at higher densities in the future without needing special initial conditions.

### Octahedron

For octahedra, the optimal lattice packing has a density of $\phi = 0.947368$. The packing on the right of figure 10.6 shows such a lattice packing, although the packing density is lower because the lack of a deformable boundary means that the periodic walls are not perfectly adjusted to the lattice itself. Packings of octahedra with random initial conditions generated with our packing code usually get stuck in states of lower density, around $\phi \approx 0.6$. Although octahedra are in principle easier to pack than dodecahedra, when random dilutions are used as initial conditions, the ability to form ordered layers of particles (that allow packing to proceed further) is impaired. To generate high-density packings, it is necessary to use lattice configurations as initial conditions. Partially ordered states can also be generated by perturbing the lattice, as can be seen in the center packing of figure 10.6. Notice that the orientations of the particles in this packing are much more aligned than in the completely disordered packing on the left.


(a) Disordered ($\phi = 0.6044$)


(b) Partially ordered ($\phi = 0.6466$)

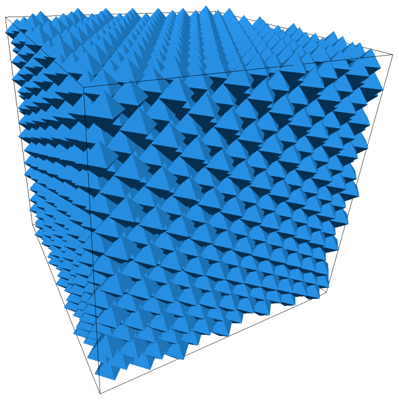

(c) Highly ordered ($\phi = 0.8553$)

Figure 10.6: Packings of octahedra.

### Hexahedron (Cube)

Cubes are rather uninteresting to pack, since they tile space, but we present packings of cubes here merely for completion. Many materials form crystals with cubic shape, so the random packing characteristics of cubes are just as important to us as the other shapes. It is interesting to note that although cubes can be readily arranged into a tiling lattice, a computer packing code using random displacements does not necessarily find the optimal packing configuration every time. Figure 10.7 shows a relatively large packing of cubes containing $18^3$ particles (left) and a small packing demonstrating the type of sub-optimal configuration that makes the packing become jammed earlier than expected.

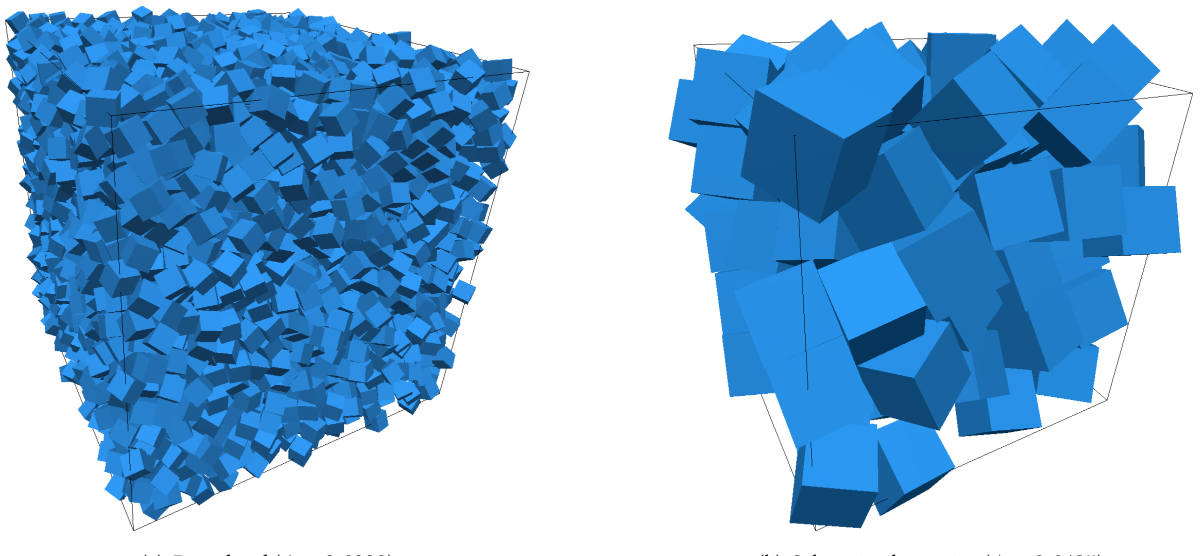
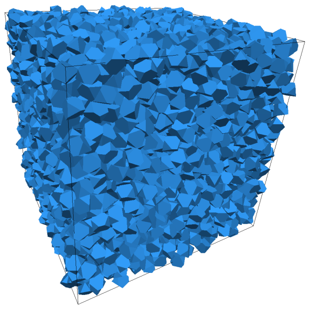

(a) Disordered ($\phi = 0.6023$) (b) Sub-optimal jamming ($\phi = 0.6435$)

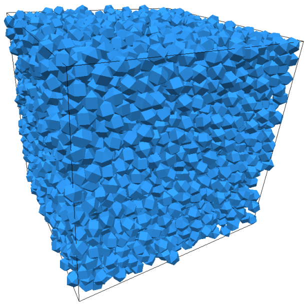
Figure 10.7: Packings of cubes.

## 10.2.2 Archimedean Solids

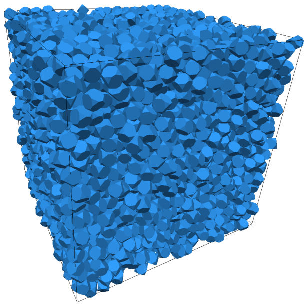
We have also generated disordered packings of the Archimedean solids containing $18^3 = 5832$ particles each to determine their jamming densities. Since several Archimedean solids are roughly spherical, it is not surprising that disordered packings of the Archimedean solids become jammed at densities close to that of monodisperse packings of spheres. The packing densities obtained here are lower bounds for the MRJ packing density of Archimedean solids. It is possible to generate denser packings of these solids with our code with lower numbers of particles. However, since at the moment we lack the tools to quantify ordering in packings of polyhedra, we can not yet estimate the true MRJ density of these packings. Jiao and Torquato [54] recently investigated maximally random



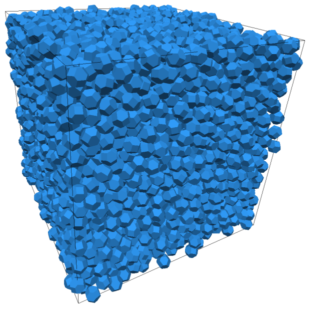
(a) Truncated Tetrahedra



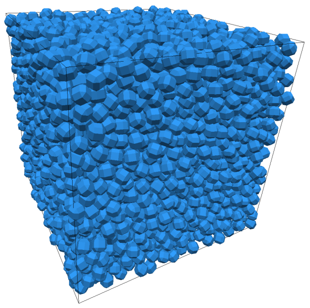
(b) Cuboctahedra



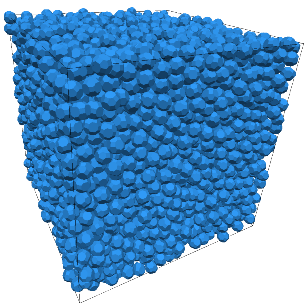
(c) Truncated Cubes



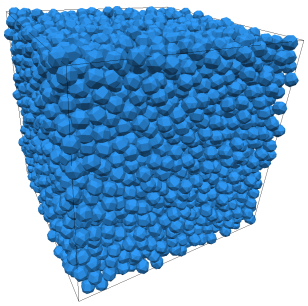
(d) Truncated Octahedra



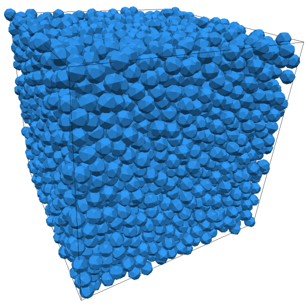
(e) Rhombicuboctahedra



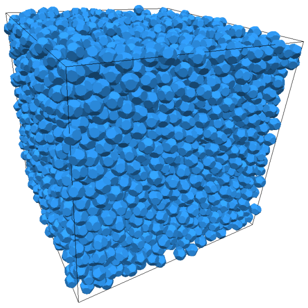
(f) Truncated Cuboctahedra



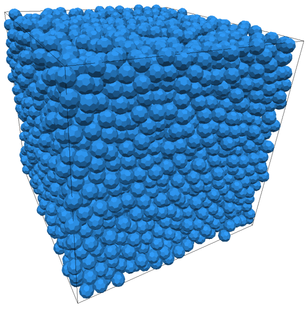
(g) Snub Cubes



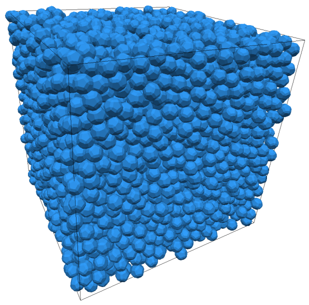
(h) Icosidodecahedra



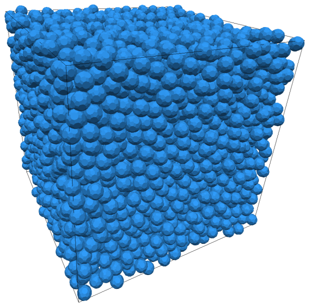
(i) Truncated Dodecahedra



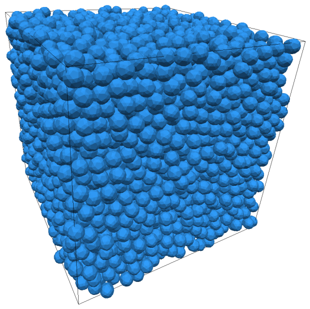
(j) Truncated Icosahedra



(k) Rhombicosidodecahedra



(l) Truncated Icosidodecahedra



(m) Snub Dodecahedra

Figure 10.8: Disordered packings of the Archimedean solids.

Table 10.1: Jamming density of Disordered Packings of Archimedean Solids.

| Polyhedron | Density | Polyhedron | Density |
|---|---|---|---|
| Truncated Tetrahedron | 0.6411 | Cuboctahedron | 0.6080 |
| Truncated Cube | 0.6578 | Truncated Octahedron | 0.6325 |
| Rhombicuboctahedron | 0.6026 | Truncated Cuboctahedron | 0.6270 |
| Snub Cube | 0.6380 | Icosidodecahedron | 0.6327 |
| Truncated Dodecahedron | 0.6056 | Truncated Icosahedron | 0.6457 |
| Rhombicosidodecahedron | 0.6398 | Truncated Icosidodecahedron | 0.6268 |
| Snub Dodecahedron | 0.6380 | | |

jammed packings of Platonic solids (results for packings of Archimedean solids are yet unpublished). Their reported MRJ densities of these polyhedra are somewhat higher than densities of jammed packings generated with our packing code. They use a two-step process to generate MRJ packings: first, a pre-jammed configuration is found using the positions of disordered jammed spheres as initial condition; then, a slow compression algorithm is used to allow a contact network between particles to be established which induces the jamming of the particles. Our packing algorithm, on the other hand, currently uses a single-step packing procedure starting with a dilute packing of infinitesimal particles with random positions and orientations. In the future, we hope to improve our packing code for polyhedra and develop the necessary tools to better investigate the properties of packings of Platonic and Archimedean solids, as well as other families of polyhedra, such as the Catalan and Johnson solids, whose packing properties still remain virtually unknown.

### 10.2.3 Comparison with Experimental Data of Packings of Polyhedral Dice

In this section, we compare the packing densities of packings of the Platonic solids generated with our packing code with experimental data found in the literature. Experiments with plastic dice were performed by Jaoshvilli [59] (tetrahedra), and Baker and Kudrolli (Platonic solids) [58]. Baker and Kudrolli [58] use four different packing methods to produce packings of plastic dice with regular polyhedral shapes. In the first method, particles are simply poured into the container from the height of a few particle sizes. The average height of a lid is then used to determine the packing density for an ensemble of ten packings. In the second method, the container is shaken by hand after each few layers is inserted, in order to let particles settle and rearrange. In the third method, the container is shaken with an electrical shaker rather than by hand. Finally, the last packing method they used was to fill the container (in which particles were already present) with water from the bottom with various fluid flow rates to suspend the particles in the liquid. Then, the container is slowly drained to allow the particles to settle into position. The jamming densities of the packings of dice of each shape and for each packing method are shown in table 10.2. Results for simulations of large polyhedral packings using our code are shown in table 10.3. Except for tetrahedra, our packing code is able to generate packings at least as dense as mechanically shaken packings of the Platonic solids produced experimentally. With lower numbers of particles, however, it is possible to produce denser packings with our code. Therefore, we can conclude that the performance of our packing code for the simulation of particulate materials is satisfactory, since in these simulations particles are usually packed at lower densities.

Table 10.2: Jamming density of polyhedral dice, reproduced from [58].

| | Sequential Addition | Hand Shaken | Mechanically Shaken | Fluidization |
|---|---|---|---|---|
| Tetrahedron (plastic) | $0.54 \pm 0.01$ | $0.62 \pm 0.02$ | $0.64 \pm 0.01$ | $0.51 \pm 0.01$ |
| Cube (plastic) | $0.57 \pm 0.01$ | $0.66 \pm 0.02$ | $0.67 \pm 0.02$ | $0.54 \pm 0.01$ |
| Octahedron (plastic) | $0.57 \pm 0.01$ | $0.62 \pm 0.01$ | $0.64 \pm 0.01$ | $0.52 \pm 0.01$ |
| Dodecahedron (plastic) | $0.56 \pm 0.01$ | $0.60 \pm 0.01$ | $0.63 \pm 0.01$ | $0.51 \pm 0.01$ |
| Icosahedron (plastic) | $0.53 \pm 0.01$ | $0.57 \pm 0.01$ | $0.59 \pm 0.01$ | $0.50 \pm 0.01$ |
| Tetrahedron (ceramic) | $0.48 \pm 0.02$ | $0.59 \pm 0.01$ | | |

Table 10.3: Typical jamming density range of packings of Platonic solids simulated with our code.

| Polyhedron | Jamming density |
|---|---|
| Tetrahedron | 0.50–0.60 |
| Cube | 0.60–0.80 |
| Octahedron | 0.60–0.65 |
| Dodecahedron | 0.60–0.62 |
| Icosahedron | 0.62–0.68 |

# Chapter 11

# Modeling the Microstructure of Energetic Materials

## 11.1 Introduction

In the preceding chapters we presented packings of particles of various shapes to demonstrate the capabilities of the packing code that constitutes the bulk of our work. In this chapter, we in turn discuss how packings generated with our code can be applied to model the microstructure of real materials. The microstructural information and the phase properties of a heterogeneous material play a crucial role in the determination its macroscopic—or *effective*—properties. Therefore, understanding the relationship between the structure and the effective properties of materials is one of the main goals of materials science. However, while the connection between structure and properties is well developed for single-phase media such as metallic alloys, polymers, and ceramics, it is much less well understood for heterogeneous materials composed of several single-phase materials. This is because the dependency of the effective properties on the microstructure and phase properties is quite complex [171]. Simple mixture relations based on the volume fractions of the material phases and their properties are insufficient to describe the interactions that arise between material phases. Solid rocket propellants are a clear example of this fact. The burning rate of solid propellants depends strongly on the surface area between the fuel and oxidizer phases. Therefore, a propellant that is composed of alternating flat layers of fuel and oxidizer will burn with different characteristics than a propellant in which the oxidizer is made of particles that are embedded in a fuel matrix, even if the quantity of each material is the same in each case. Similarly, these two examples of propellants would respond very differently to shock waves. In the layered material, the direction of propagation of the shock wave relative to the interface between the phases can significantly affect the behavior of the material; in the particulate material, on the other hand, the direction of the shock wave is much less important, since the material is most likely statistically homogeneous and isotropic. Nevertheless, the sensitivity to ignition is also affected by features of the microstructure at much smaller scales. For example, tiny voids inside explosive particles can lead to the formation of local hotspots that in turn can trigger detonation. This makes the relationship between microscopic and macroscopic properties of energetic heterogeneous materials even more difficult to simulate in multi-scale models. To further illustrate the complex relation between structure and effective properties, consider now the effective electric conductivity of two heterogeneous materials, each composed of two phases: one that is highly electrically conductive; the other, highly insulating. Both materials are

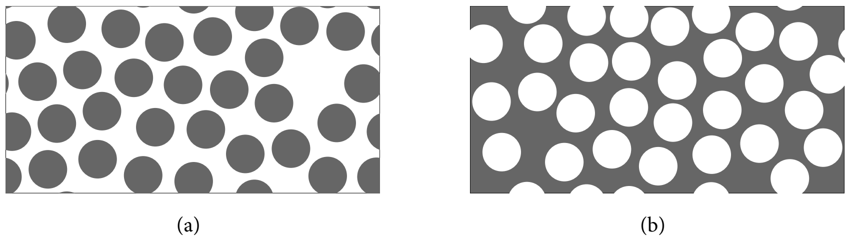


Figure 11.1: Materials with disconnected (a) and connected (b) conducting phases.

shown in figure 11.1. In the first material the conductive phase is disconnected across the sample, while in the second material there is a continuous path that can be followed from one side of the sample to the other through the conductive phase. Even if both of these samples have the same quantities of each material, their electric conductivity will be clearly different. The connectedness of the conductive phase in the second sample makes its conductivity much higher. Hence, simple formulas like the average or harmonic mean of the conductivities for each phase are not adequate to describe either material. These formulas either grossly underestimate or overestimate the results, not to mention that they yield the same results for two materials that obviously have distinct values for the conductivity.

For a systematic theory to be able to predict how changes in the microstructure quantitatively affect the effective properties of composite materials, it is necessary to develop the means to describe the microstructural information. The most basic information is the volume fraction of each phase. For random heterogeneous materials, it is also necessary to obtain information about the connectivity of each phase, the surface areas between the interface, etc, and for particulate materials the orientations, sizes, shapes, and spatial distributions of the particles are just as important. Quantitative information about the materials can also be obtained via statistical descriptors, such as the ones described in chapter 3. In particular, the $n$-point probability functions appear frequently in the averaging process of effective properties, since in the resulting integrals over the spatial domain of the material, the local fields of the relevant properties are weighed by the $n$-point probability functions. Since these functions appear naturally in the description of the microstructural information of heterogeneous materials, they can be used as a benchmark to test how realistic simulated materials are when compared with data obtained for real materials.

The microstructural information of a material can be obtained through various techniques, such as scanning electron microscopy (SEM), transmission electron microscopy (TEM), and X–ray computed tomography (XCT), among others. X–ray computed tomography, for example, can be used to produce volumetric images of material samples *nondestructively*. This means that not only can the information be obtained from the material samples, but it is also possible to later test models against the same microstructure directly. Figure 11.2 shows a volumetric image of salt particles obtained via X–ray computed tomography. From an image like this, it is possible to obtain the size distribution of the particles, volume fraction, $n$-point probability functions, etc. Combined with knowledge about the shape of salt crystals, it is possible to use our packing code to produce virtual packings of particles with very similar characteristics as the original sample of the material. In the next sections we use our packing codes to reconstruct the microstructure of a few materials and compare the statistical properties obtained via tomography with that obtained from simulated packings.

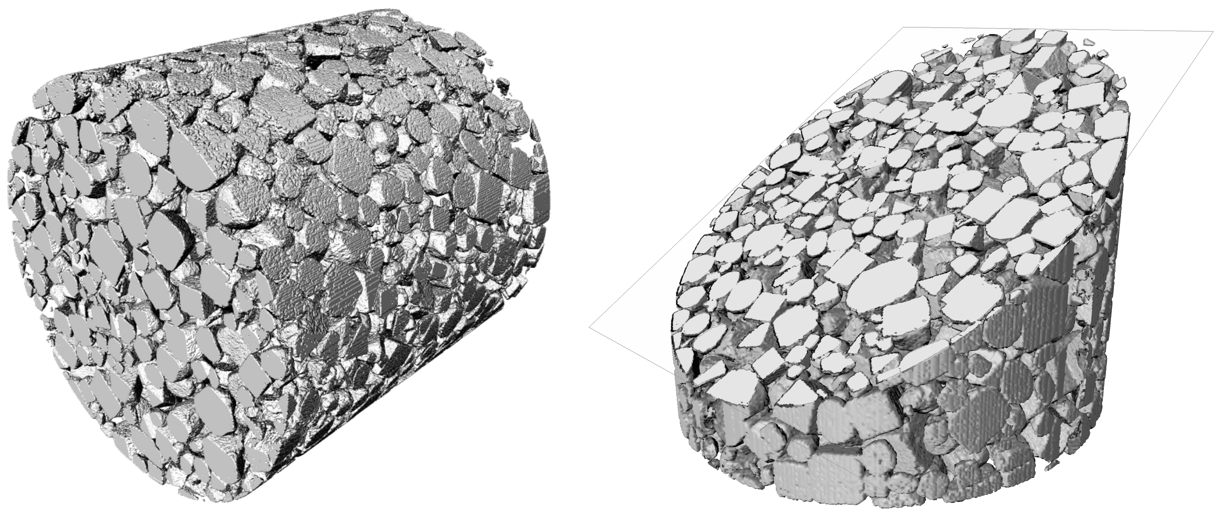


Figure 11.2: Volumetric rendering of salt particles obtained via X–ray computed tomography.

## 11.2 Microstructure Reconstruction from Tomography

In order to compare how packings generated with our code reproduce real materials, we performed tomographic scans of a small set of samples of heterogeneous materials, including some solid rocket propellants. All material samples shown henceforth have been scanned at University of Illinois' Beckman Institute for Advanced Science and Technology. The scans were performed on an Xradia MicroXCT-200 high-resolution 3D X–ray computed tomography machine. The Xradia machine is capable of scanning samples up to 100 mm in diameter and weighting up to 1 kg at resolutions as high as 1 μm per pixel. In practice, however, samples containing very small particles ($r < 10$ μm) are difficult to be resolved. To be able to resolve them, some samples have been cut to a smaller size to fit entirely within the scanning volume of the X–ray machine. Higher resolution scans have higher amounts of noise, which compromises later processing steps to obtain statistical data. However, using smaller samples to resolve finer particles also has its own drawbacks—it is not possible to compute the radial distribution function for those samples, since there are not enough particles in the scanning volume. Nevertheless, it is still possible to compute the $n$-point probability functions.

Samples embedded in polymer binders were hand-mixed, hence they inevitably contain a certain amount of voids. In those cases, the voids have been incorporated into the binder phase in the statistical analysis. Vacuum-packed samples would have been ideal, but we did not have access to them. The shape of the samples was either a cylinder or a square rod. Samples were about 4–5 mm in diameter and 1–2 cm in length, as shown in figure 11.10.

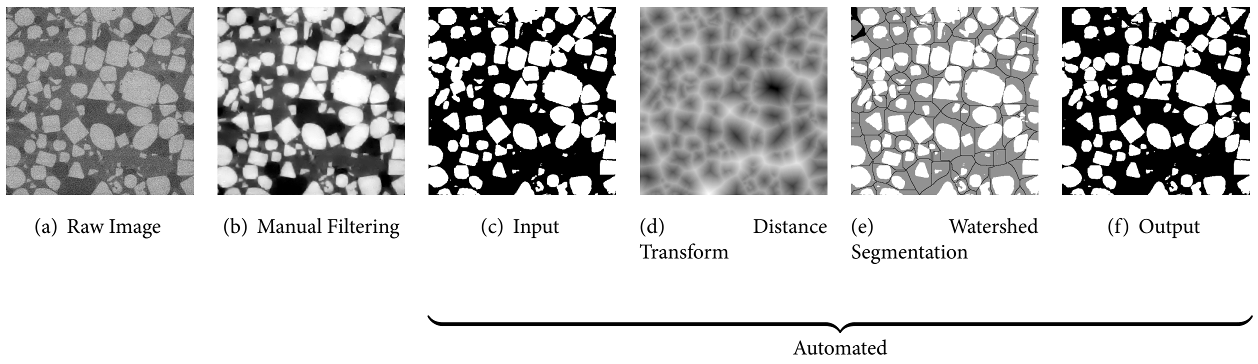


Figure 11.3: Image processing workflow.

Once we acquire data for each sample, we process the raw images to segment them into the separate phases. We use the software Amira [172] for all volumetric image processing work. First, we apply filters to reduce the noise in the raw image, until it becomes possible to binarize it into the particle and binder/void phases. Then we use a watershed segmentation algorithm to separate particles that may have clumped together if the gap between them was too small to be resolved. These steps are depicted in figure 11.3. Although we show each step of the watershed segmentation algorithm, this process is automated in Amira. Particle separation is very important to obtain good data on the size distribution and shape information of the particles. Clumped particles may skew the size distribution into larger particles and generate errors in the calculation of particle flatness and elongation distributions. After particle separation, we use Amira's shape analysis module to obtain information about the shape and size distribution of the particles.

### 11.2.1 Spherical Glass Particles

The first model we discuss is a packing of small glass spheres with a diameter of about 45 μm into a cylindrical plastic container. The spheres were packed via mechanical deposition into the container, without any binder material to hold them together. The scanned volume and a cross section of the sample are shown in figure 11.4. The length and diameter of the scanning volume are each about 2 mm; resolution is about 2.8 μm per pixel. Roughly 60500 particles fill the volume of the scan.

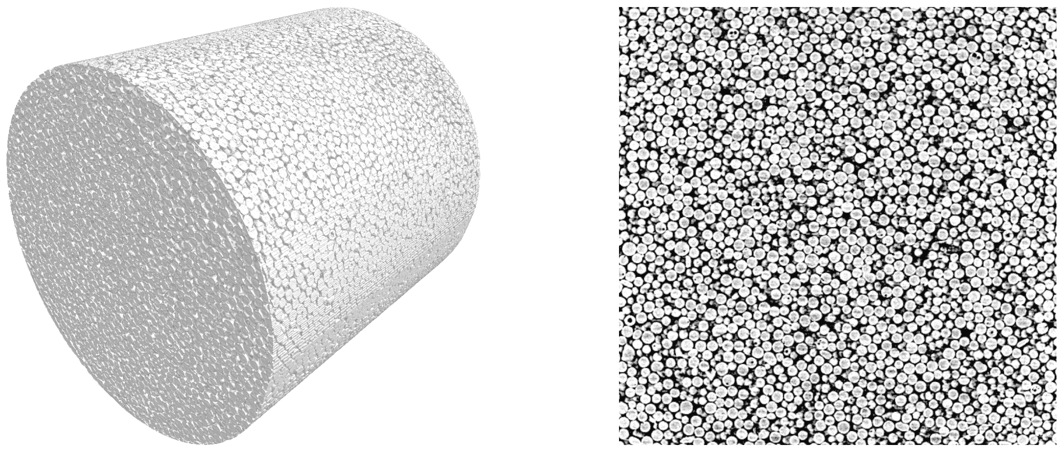

(a) Volume rendering (b) Cross section

Figure 11.4: Glass spheres with diameter $D \approx 45$ μm.

After processing the raw image files to reduce noise, we segmented the sample into its void and particle phases. We then separated the particles using the same process shown in figure 11.3 and used Amira's shape analysis module to obtain the size distribution. Particle sizes are calculated from their volume, assuming they are perfectly spherical. From figure 11.4 we can see that this is a reasonable assumption. However, since the glass particles are not perfectly spherical, we needed to apply a small correction to the mean radius (to be about 1% larger) to ensure that the scale of the radial distribution functions matched. The size distribution is shown in figure 11.5. The solid curve fit is a lognormal distribution with $\mu = 3.802 \pm 0.0014$ and $\sigma = 0.0691 \pm 0.0011$. Particles that intersect the boundary of the scanning volume were excluded from the calculation of size distribution. Figure 11.6 shows the distribution of flatness and elongation of the particles—most particles are indeed almost spherical. The broader distribution of elongated particles is due to clusters of two particles that have not been separated properly by the watershed algorithm in Amira. After obtaining the particle size distribution, we reproduced the packing with our packing code. We compare the statistical properties with two systems: a monodisperse packing of spheres, and a packing with a lognormal size distribution. The packing density $\phi \approx 0.57$ was determined by counting the voxels

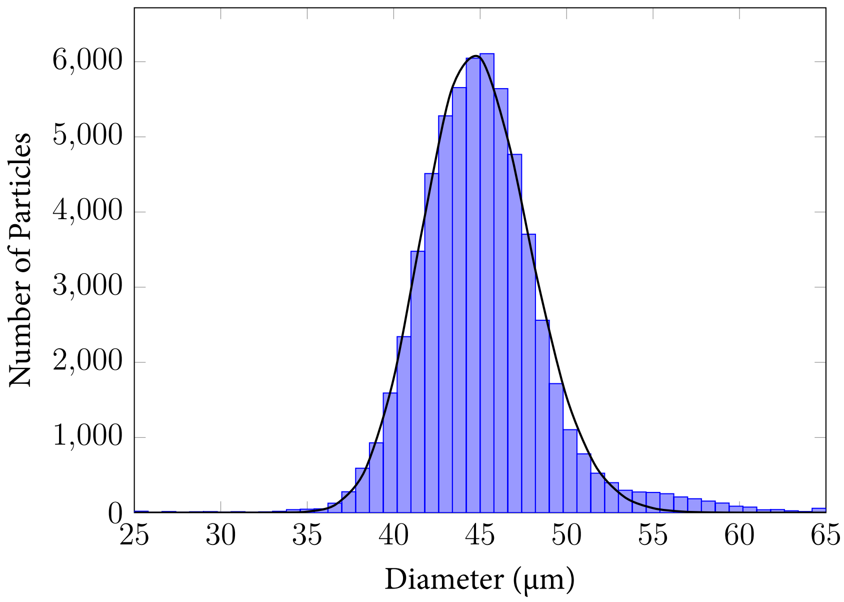


Figure 11.5: Particle size distribution obtained from tomography scan.

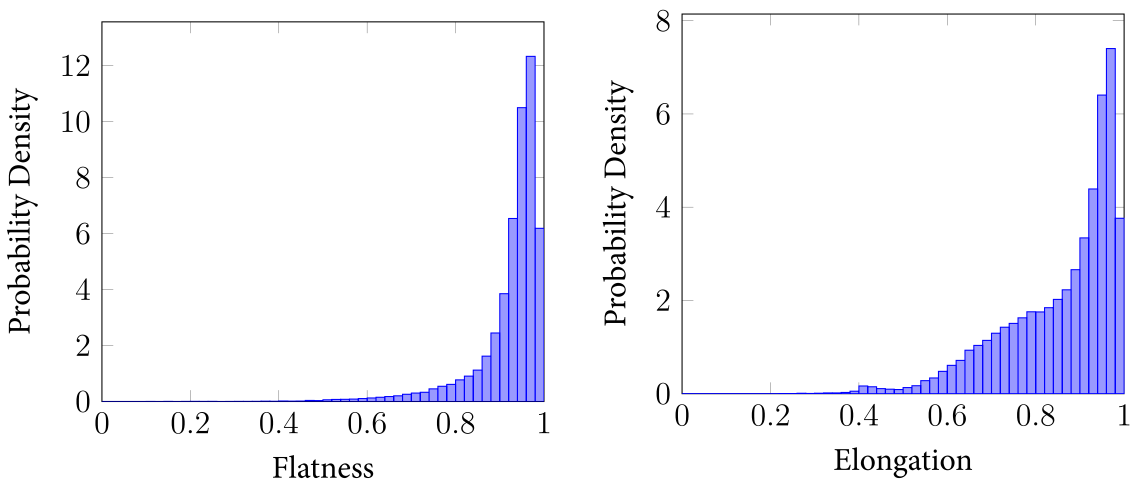


Figure 11.6: Distributions of flatness and elongation of the glass particles.

belonging to particles in the scanning volume. The radial distribution function of the particles is

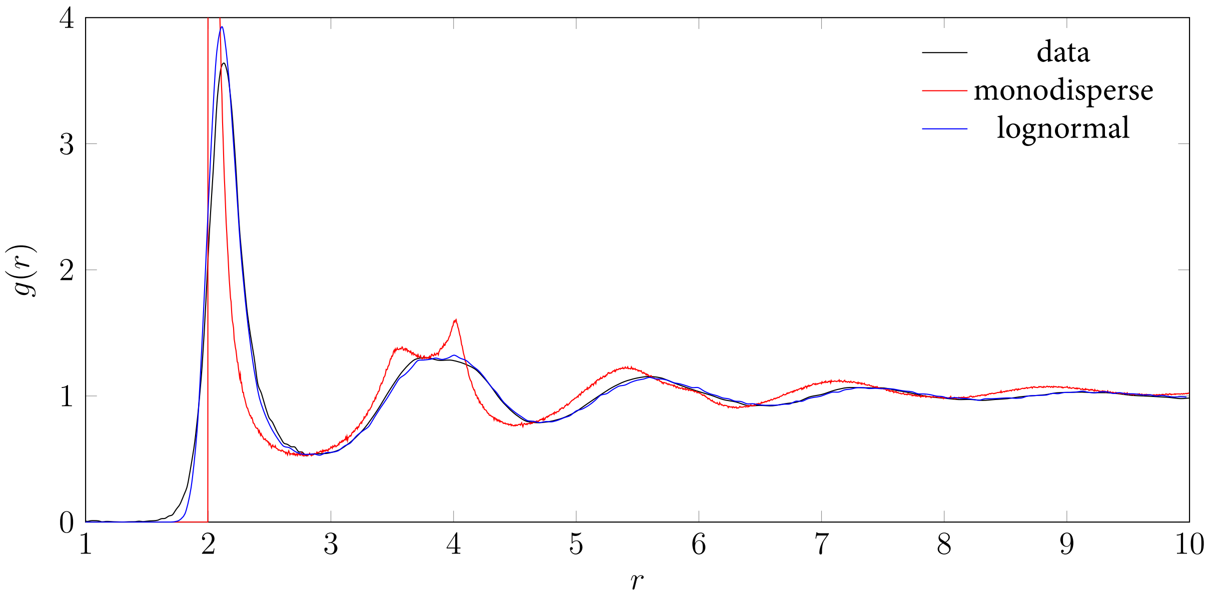


Figure 11.7: Radial distribution function obtained from tomography (black), compared with those obtained from packings of $10^6$ monodisperse (red) and $10^5$ polydisperse (blue) spheres with a log-normal size distribution.

shown in figure 11.7 in comparison with the two different simulated packings. For the polydisperse packings, we scaled the radii of the particles by their median value to be able to compare the results with a monodisperse packing of spheres with $r = 1$. Monodisperse spheres have a very sharp peak in the radial distribution function at $r = 2$, so the lognormal distribution of spheres reproduces the data much better. The peaks corresponding to small clusters of spheres, typical in monodisperse packings, are smoothed out by the polydispersivity of the particles. This is well reproduced by the simulated packing with a size distribution similar to that of the glass sample. It is also important to note that using monodisperse spheres with radii based on the median radius of the size distribution slightly underestimates the scale of the radial distribution function. This is probably due to a small error in the curve fit of the lognormal distribution. We still observe a small discrepancy with the data near $r \approx 2$, which we believe is either due to a small error in the standard deviation used in the simulation, or a consequence of using perfect spheres to simulate the particles rather than ellipsoids, but overall agreement with the data is very good. Figure 11.8 shows the two-point probability function for the glass particles (solid lines) and the corresponding function for the polydisperse packing generated with our code. In this case, the monodisperse and polydisperse packings yielded almost identical results, hence we only show a comparison of the data with the latter in figure 11.8.

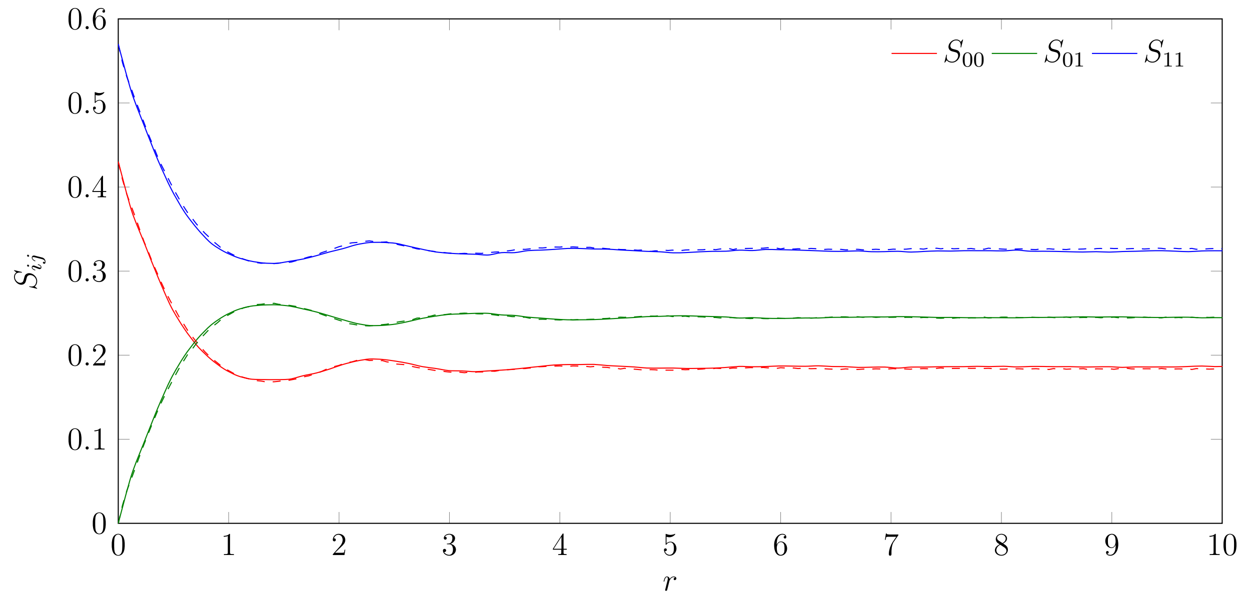


Figure 11.8: Two-point probability function for glass spheres (solid), compared with polydisperse computer generated packing (dashed).

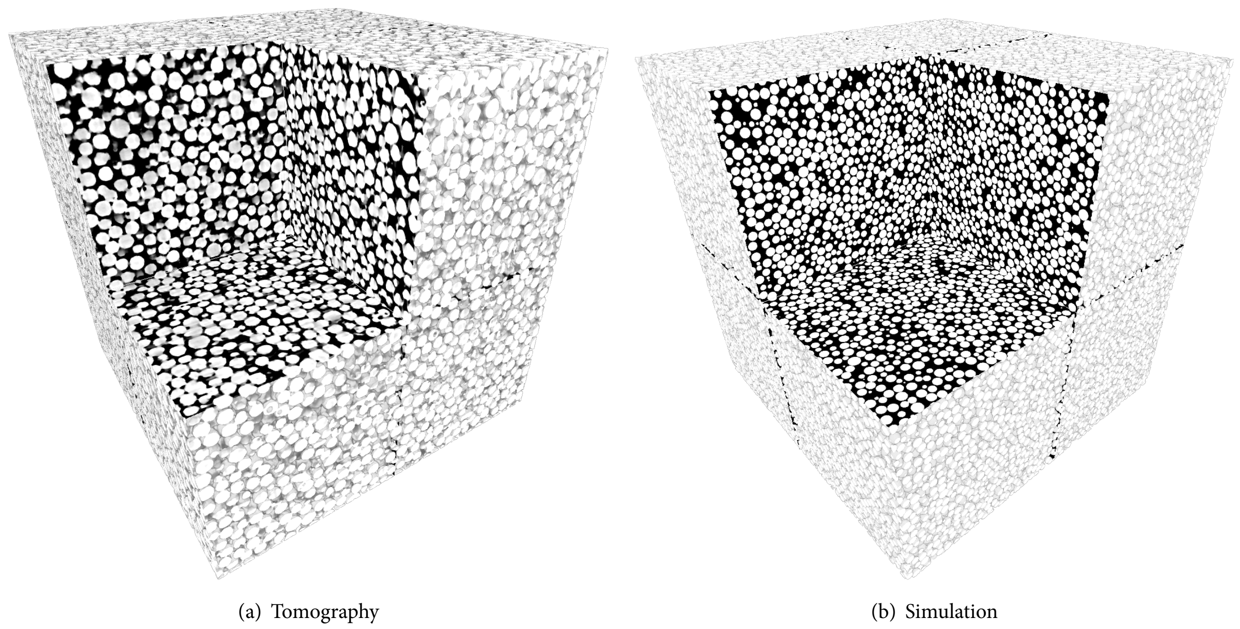


Figure 11.9: Volume renderings of the glass sample and simulated polydisperse packing.

In conclusion, we can say that the statistical characteristics of packings of particles with a small polydispersivity are very well reproduced with our code. Figure 11.9 shows a visual comparison of the glass sample and the computer-generated packing.

### 11.2.2 Salt Particles Embedded in a Polymer Matrix

The second material we discuss is a surrogate material for rocket propellants. We reconstruct two samples of table salt particles embedded in binder materials commonly used in solid propellants. One sample is packed in dicyclopentadiene (DCPD), and the other in hydroxyl-terminated polybutadiene (HTPB). Pictures of one of the samples before and after the tomography scan are shown in figure 11.10. The yellow coloration due to the excitation of sodium atoms gives an idea of the size of the total scanned volume. Both samples are scanned at a resolution of about 5 μm per pixel, and the scanned volume is about 5 mm in each dimension. Figure 11.11 shows volume renderings and a cross section of each sample. Although these samples are larger than the glass sample discussed in the previous section, they contain less particles because salt particles are much larger. They contain 2–3 thousand particles that are on average 220 μm across. The packing density is slightly different for each of the two samples: $\phi_{\mathrm{DCPD}} \approx 0.49$, and $\phi_{\mathrm{HTPB}} \approx 0.55$.

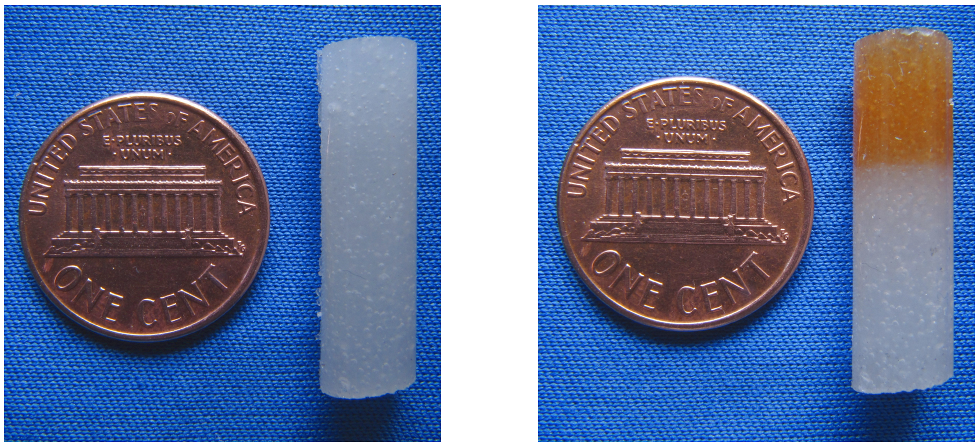
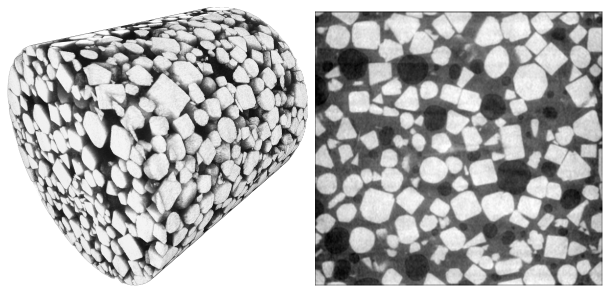

(a) Before scan (b) After scan

Figure 11.10: Photo of a sample of salt embedded in HTPB binder.

Instead of using a lognormal size distribution, this time we use the size distribution directly from the data. Although many particles have rounded shapes due to rubbing against each other, salt crystals are in general approximately cubic. Therefore, we use cubic particles in our simulations, as well as spheres for comparison. Again, particle sizes are calculated from their volume. For cubic particles, the size is calculated from their volume assuming a perfect cubic shape, i.e., we choose the edge size $a = \sqrt[3]{V}$. The scale of the spherical particles has been chosen such that they have the same volume as the cubes, using the same distribution of sizes. This time we could not exclude particles that intersect the boundary from the particle distribution. Even though the number of small particles is large, they occupy only a small fraction of the total volume of the particles in


(a) Salt in DCPD binder

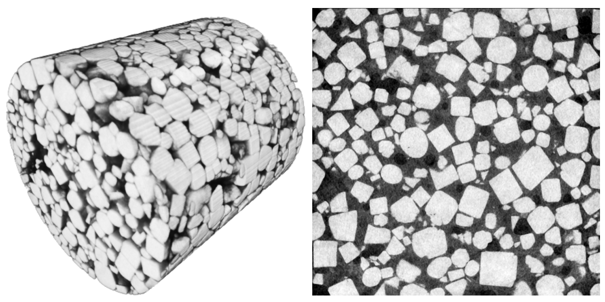
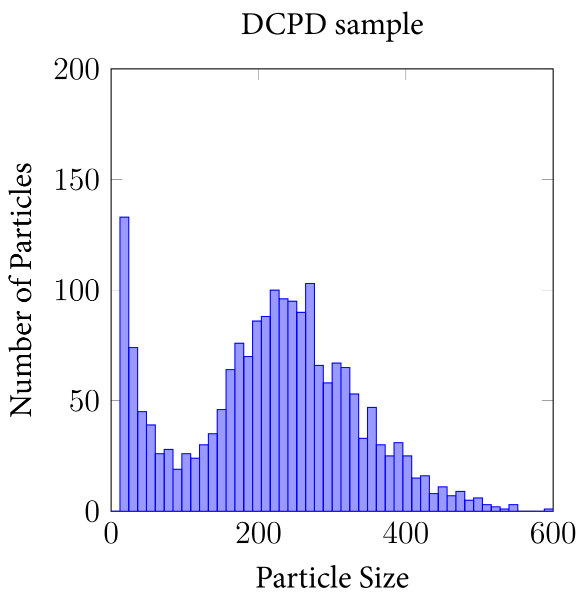

(b) Salt in HTPB binder

Figure 11.11: Salt samples embedded in DCPD and HTPB.

the packings. Figure 11.12 shows the histogram of size distributions for each sample. Using this information, we generate packings with particles of similar sizes and shapes as the salt particles. We use about 45 different sizes and a total number of 19790 particles in our polydisperse simulations. The packing of monodisperse cubes contains only 10000 particles.



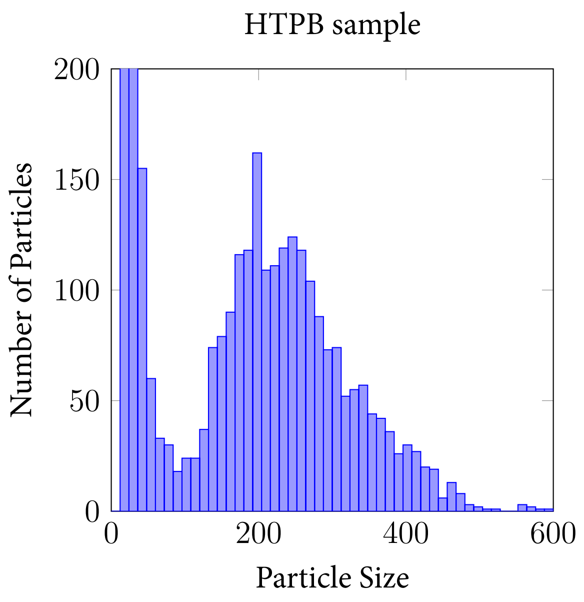
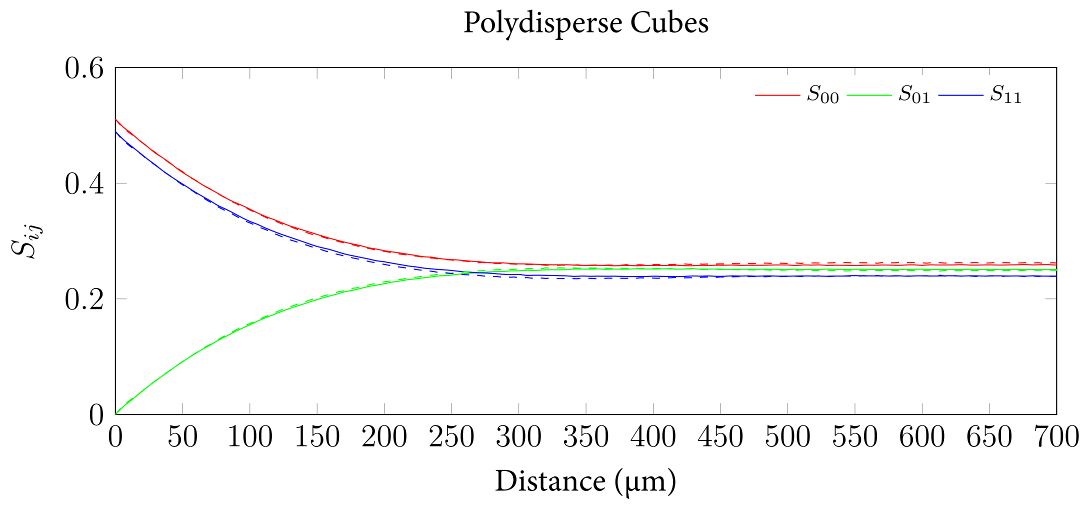
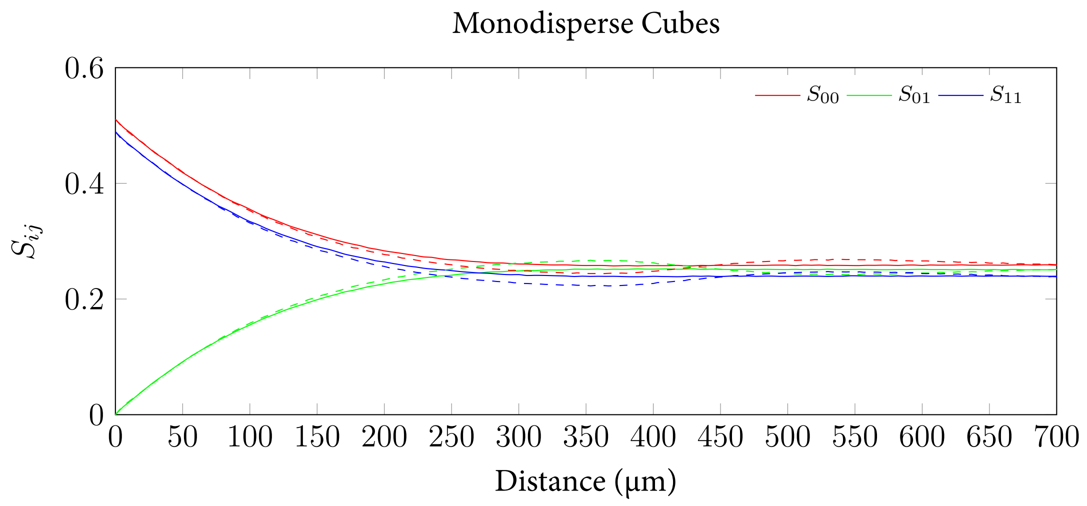


Figure 11.12: Particle size distributions.

A comparison of the two-point probability functions of the DCPD sample with simulated packings is shown in figure 11.13. To illustrate the importance of particle shape in reproducing the statistical properties, we also compare the three-point statistical functions of each sample with those of packings of polydisperse spheres, monodisperse cubes, and polydisperse cubes. Figure 11.14 shows the three-point probability function calculated from tomography, and figures 11.15, 11.16, and 11.17 show the same function for simulated packings using polydisperse cubes, monodisperse





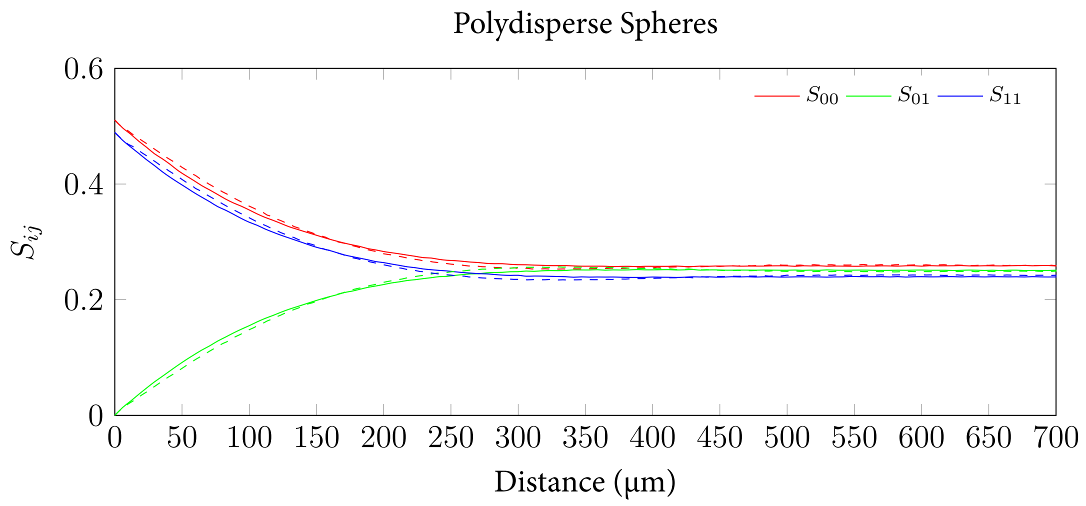


Figure 11.13: Comparison of two-point probability functions computed from tomographic data (solid) against various computer generated packings (dashed).

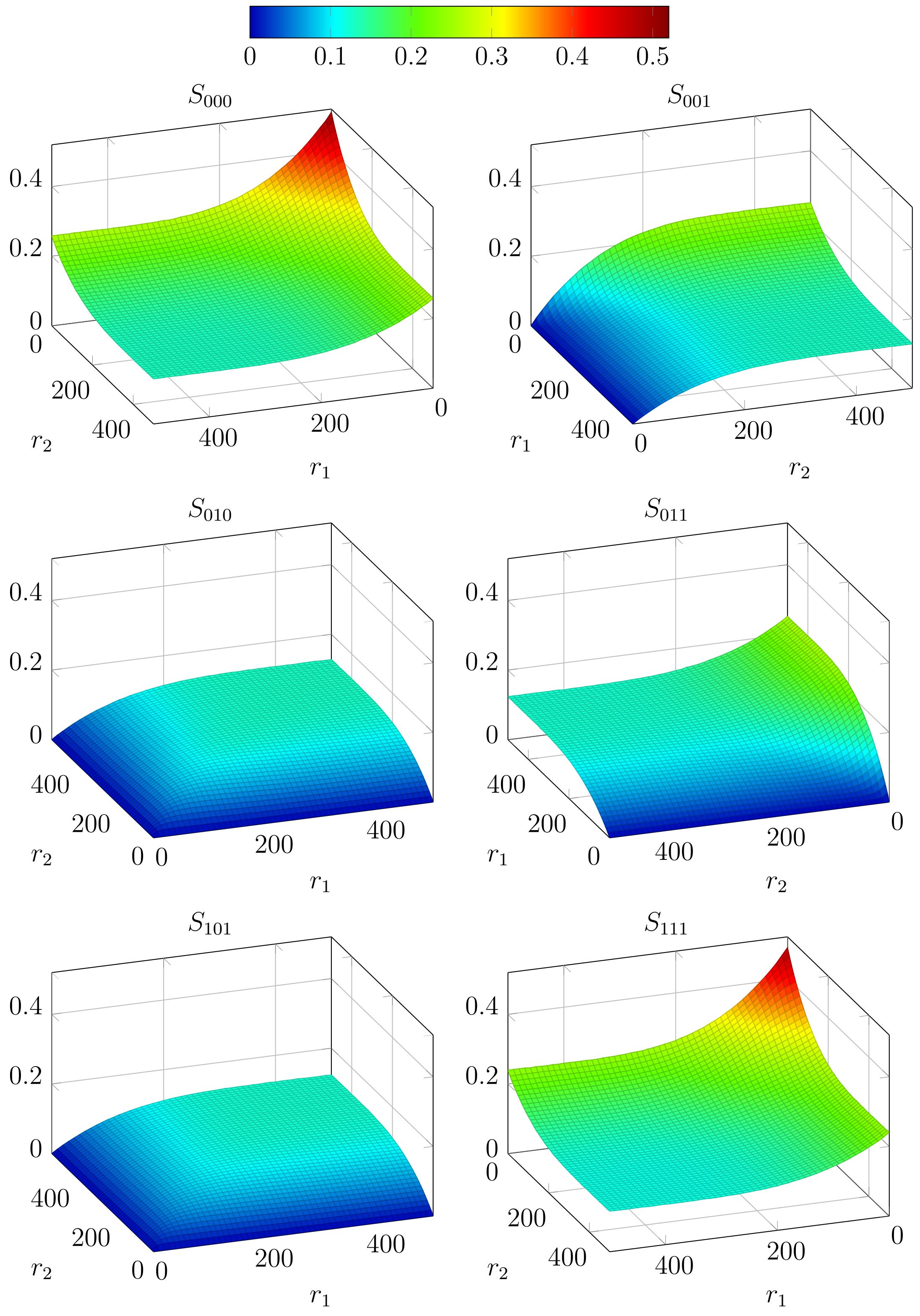


Figure 11.14: Three-point probability function for the DCPD-packed sample of salt, calculated from tomographic data. (Note the different orientations of each plot.)

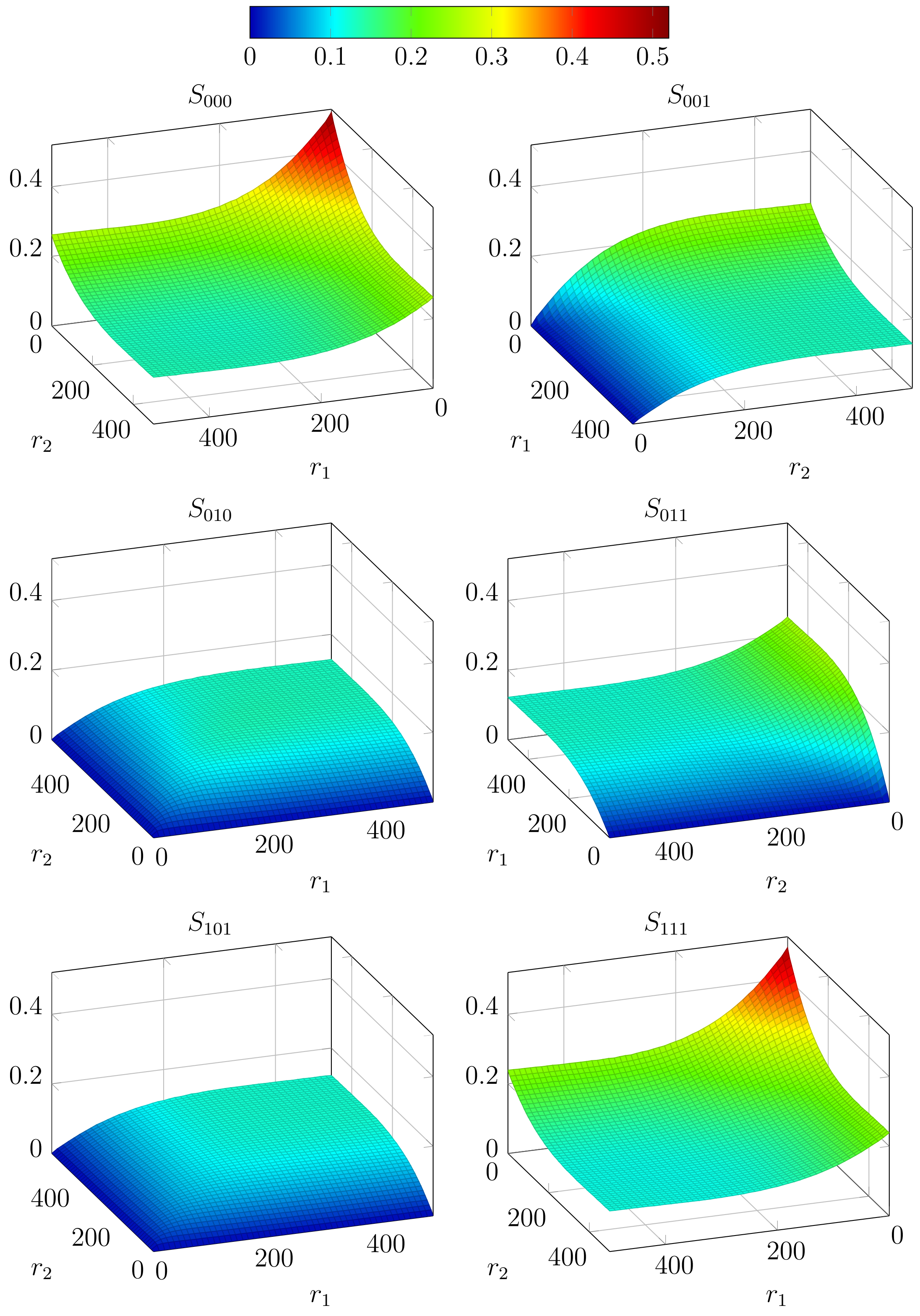


Figure 11.15: Three-point probability function for the simulated packing using polydisperse cubes.

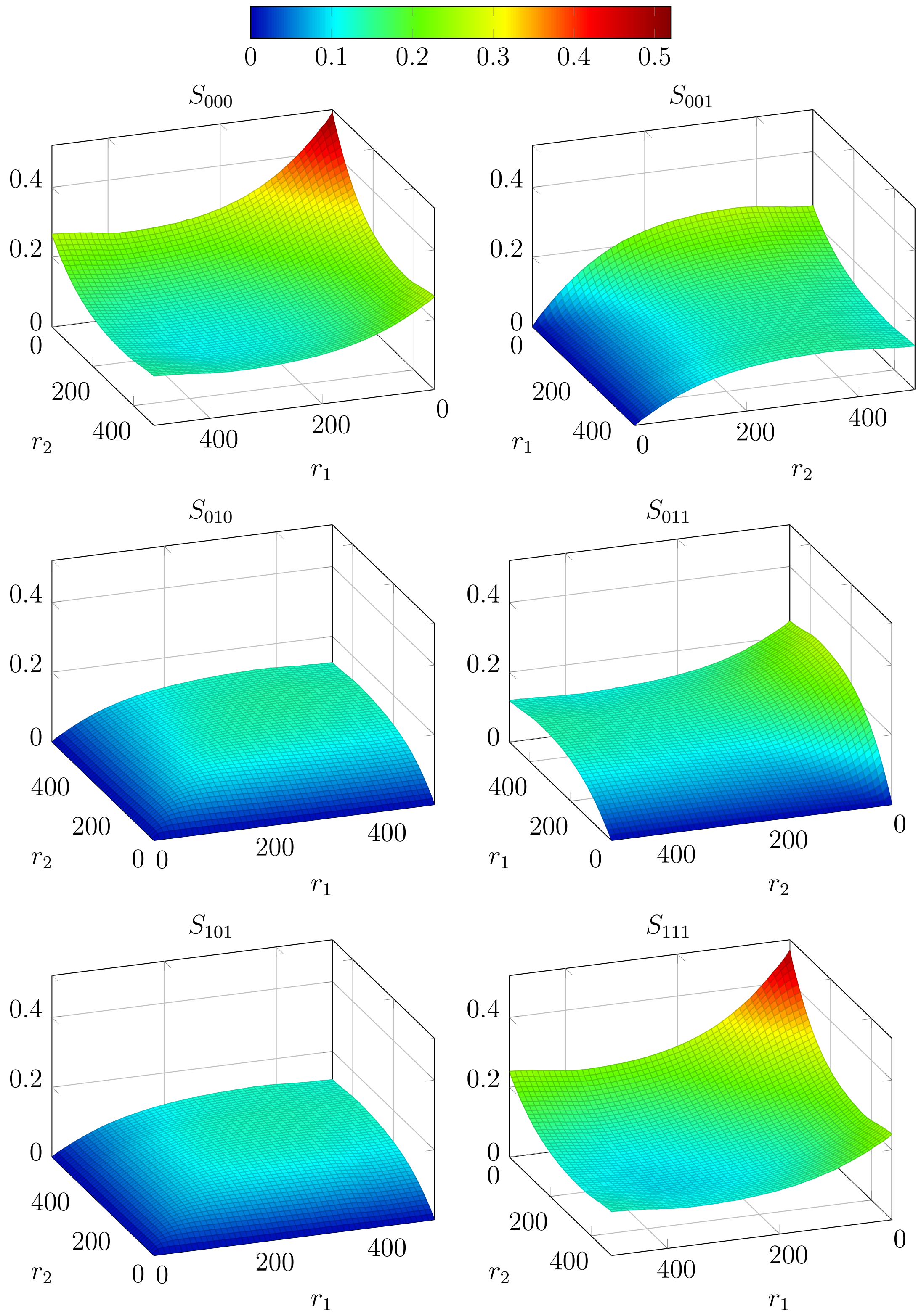


Figure 11.16: Three-point probability function for the simulated packing using monodisperse cubes.

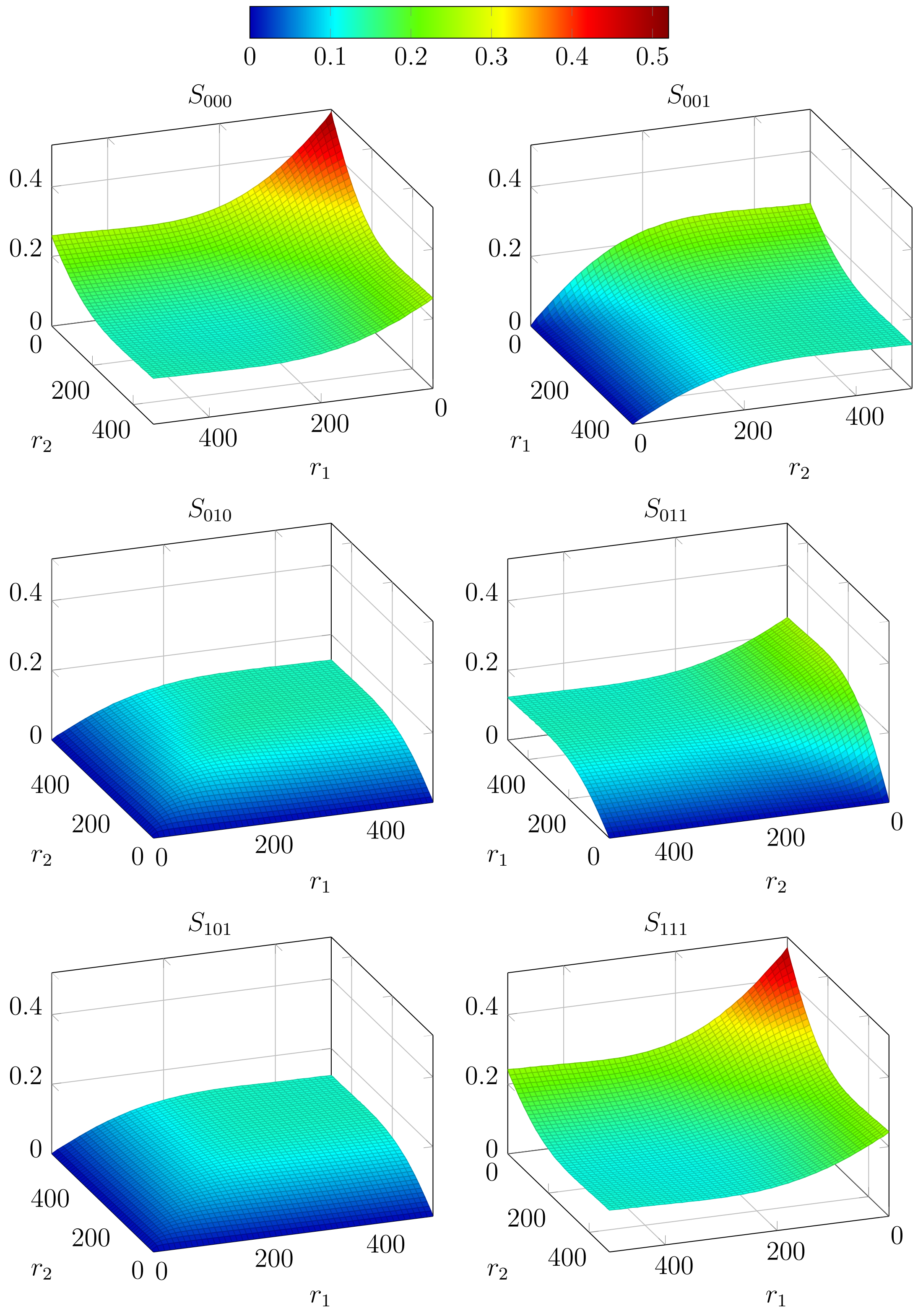


Figure 11.17: Three-point probability function for the simulated packing using polydisperse spheres.

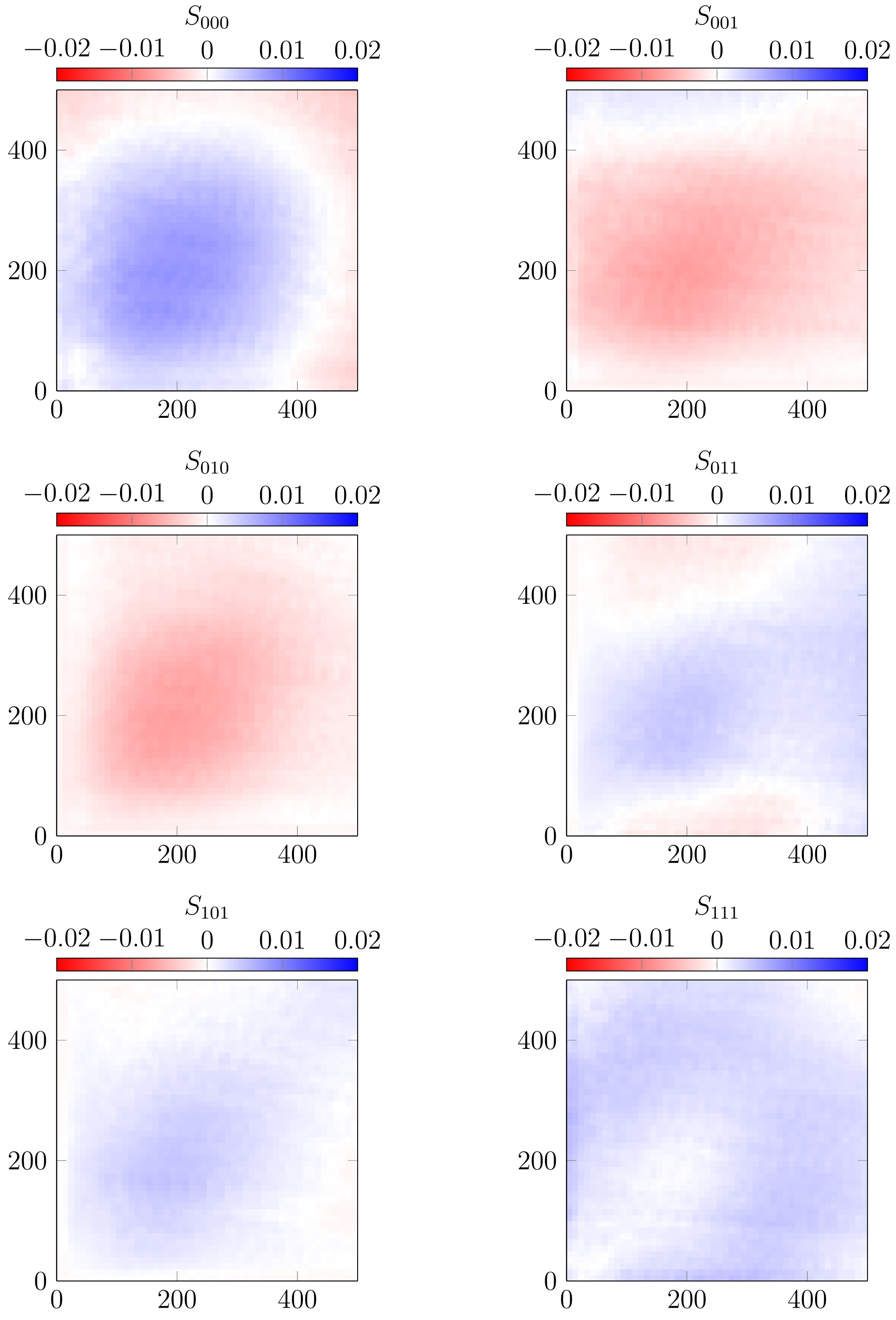


Figure 11.18: Error in the three-point probability function of the DCPD sample calculated using polydisperse cubes.

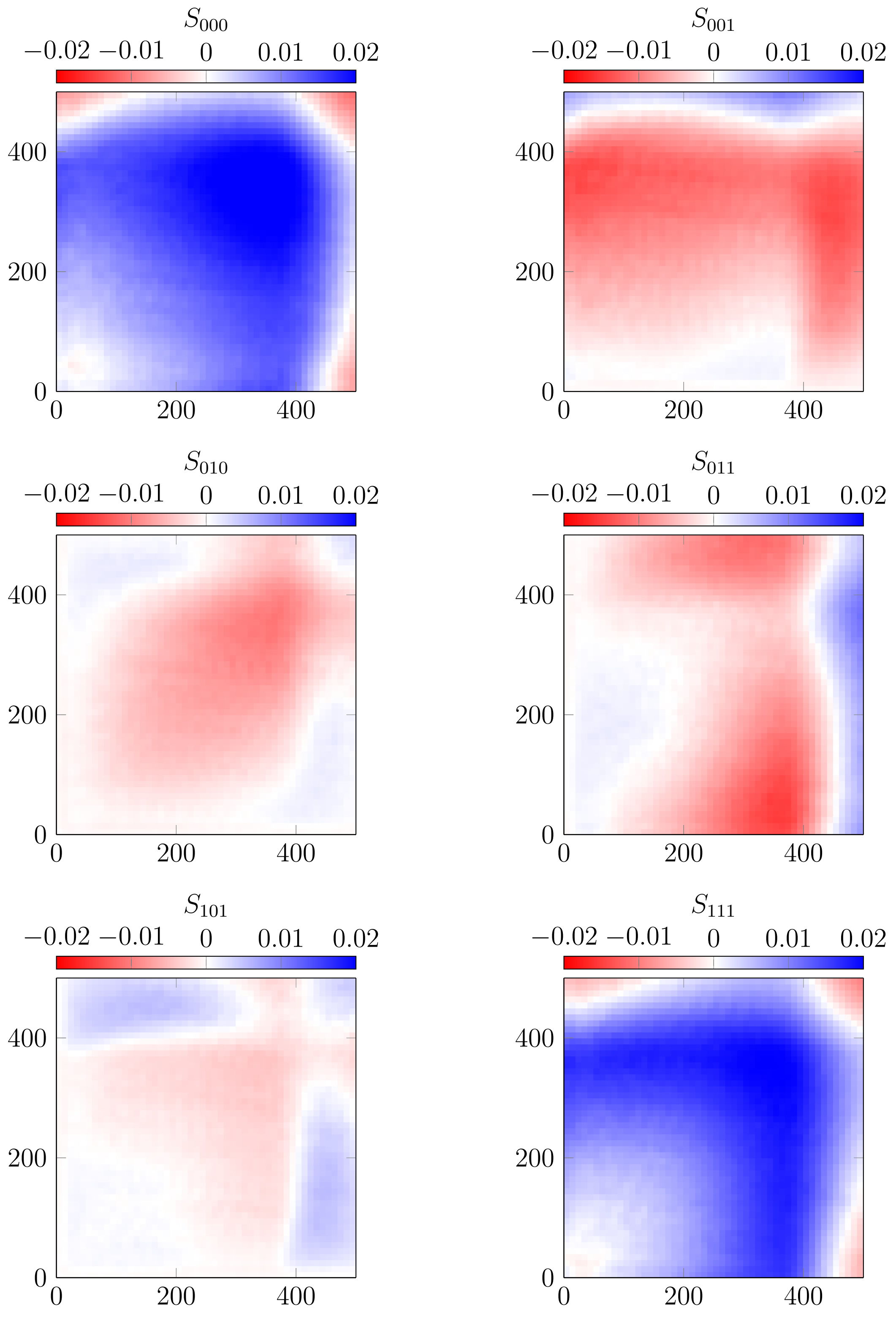


Figure 11.19: Error in the three-point probability function of the DCPD sample calculated using monodisperse cubes.

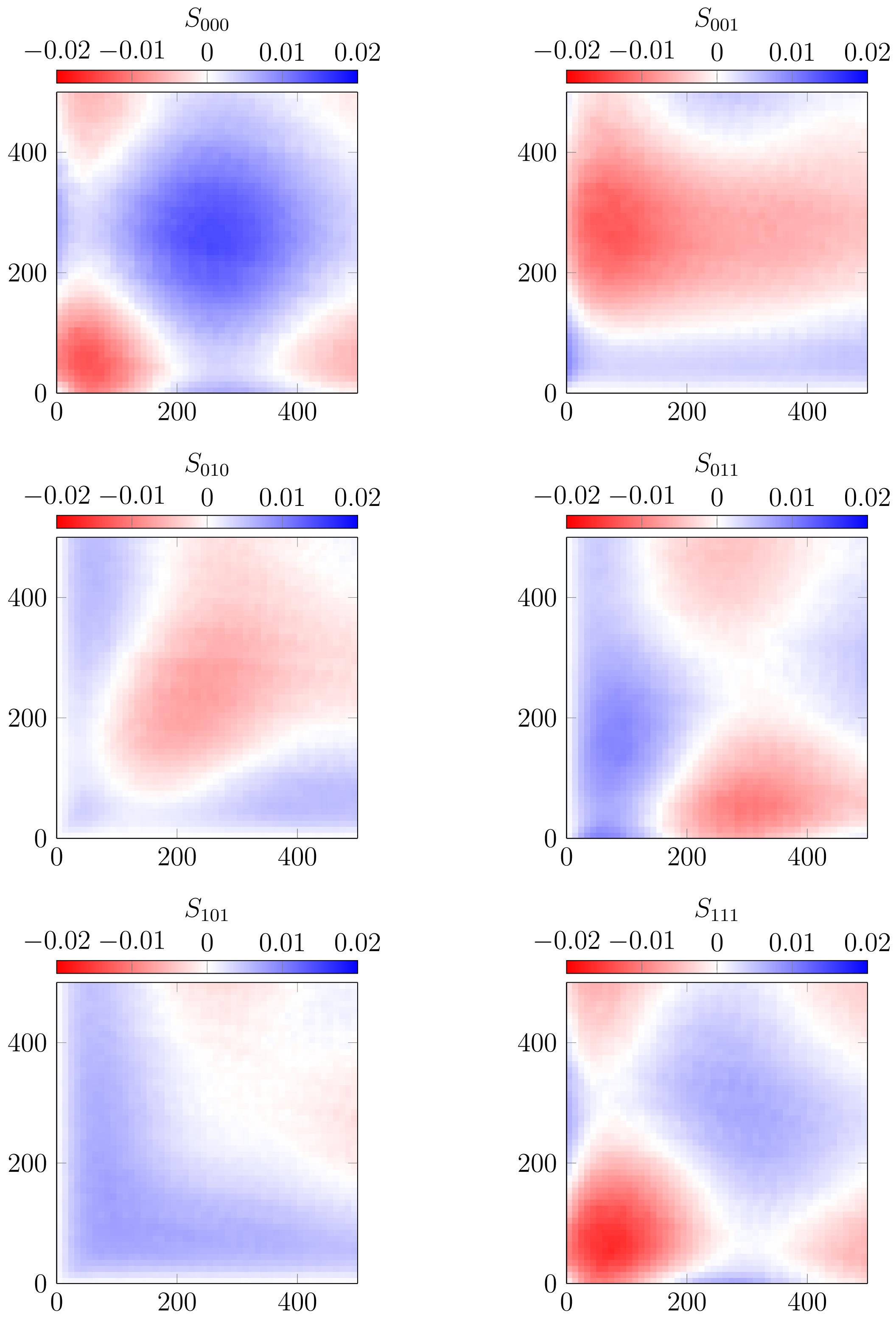
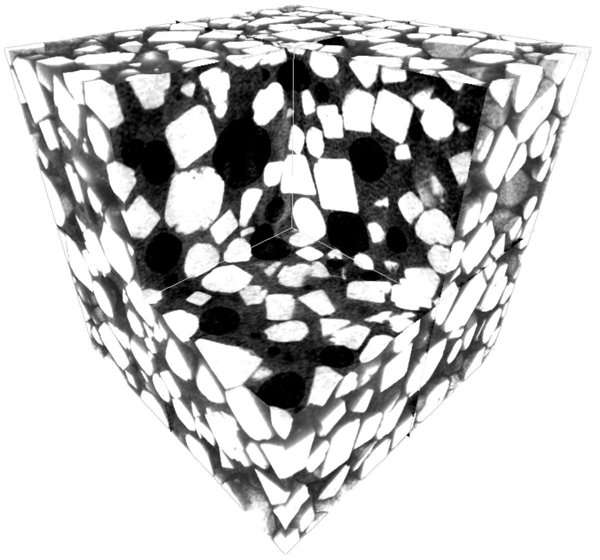
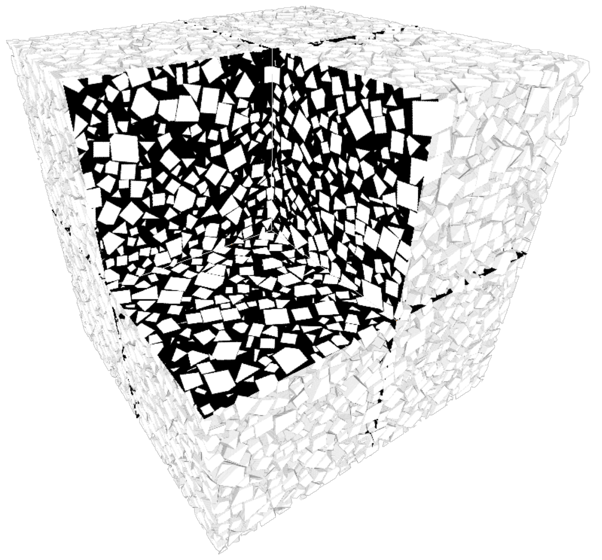
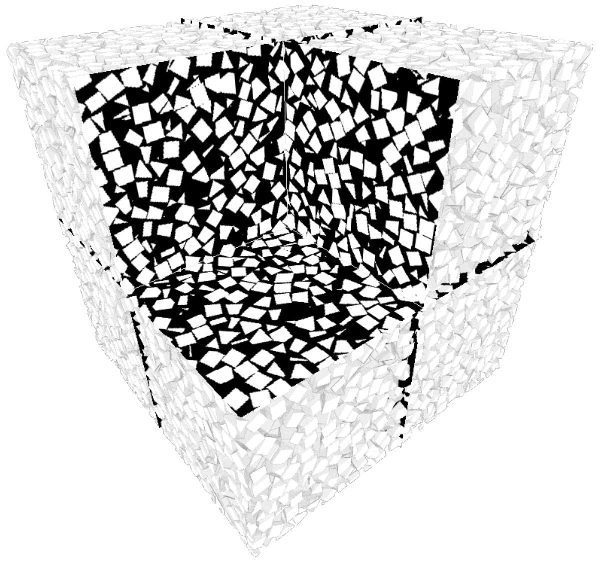


Figure 11.20: Error in the three-point probability function of the DCPD sample calculated using polydisperse spheres.

cubes, and polydisperse spheres, respectively. The difference between the three-point probability functions for the tomography scan and those for each of the simulated samples is shown in figures 11.18, 11.19, and 11.20, respectively. Polydisperse cubes perform the best, as expected. The $L^2$ error norm for polydisperse cubes is 0.441187, compared with 1.13473 for monodisperse cubes and 0.701959 for polydisperse spheres. Divided by the number of grid points, we get about $1.7 \times 10^{-4}$ for polydisperse cubes, $2.7 \times 10^{-4}$ for polydisperse spheres, and $4.4 \times 10^{-4}$ for monodisperse cubes. Although using particles of the right shape is important in reproducing the statistical properties of


(a) Tomography


(b) Polydisperse Cubes


(c) Monodisperse Cubes

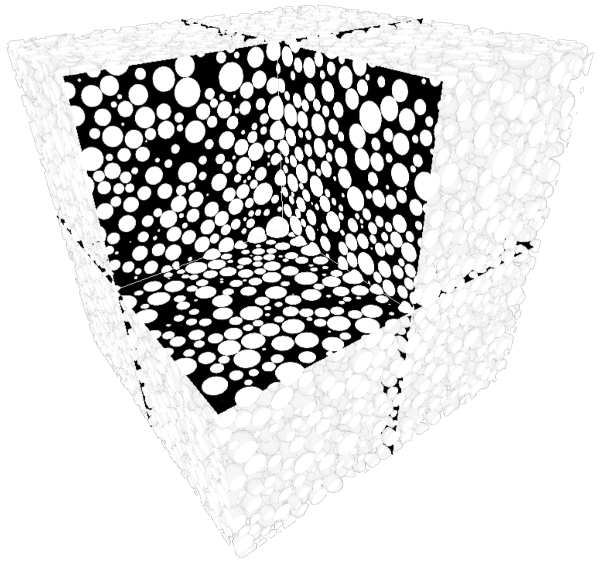
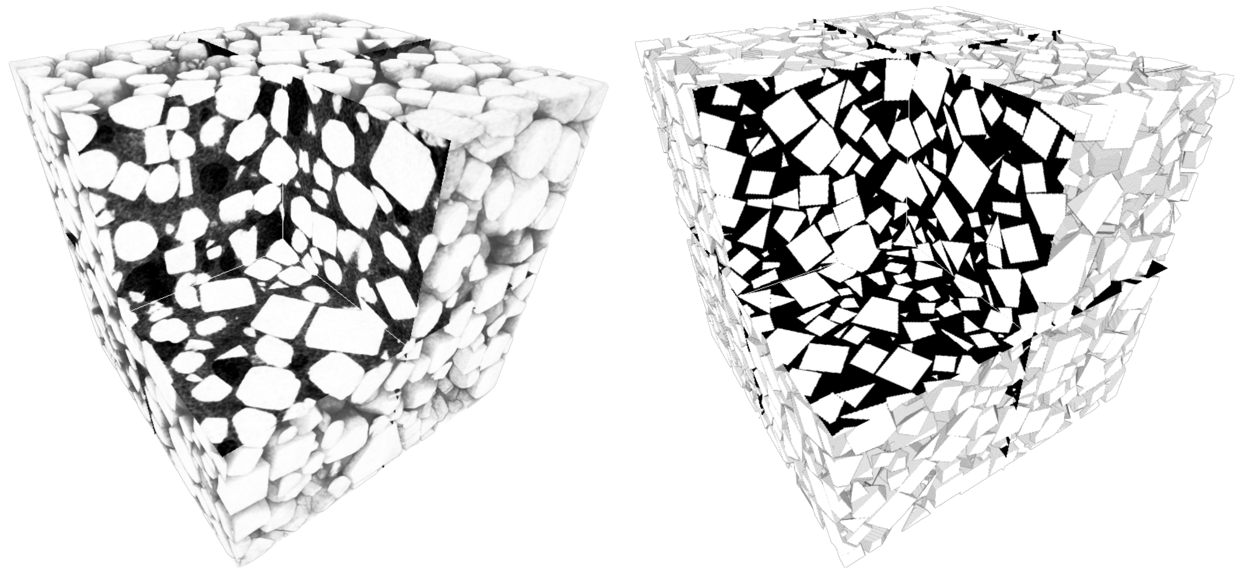

(d) Polydisperse Spheres

Figure 11.21: Volume renderings of the DCPD salt sample and computer-generated packings.

the sample more accurately, we can see from the error plots that the size distribution is clearly more important, since monodisperse cubes perform the worst among the simulated samples. At higher packing fractions, we expect these differences to become even larger. Overall, using perfect cubes as a model for salt is a good approximation, although spheres do not perform too bad either, most likely because many particles have broken corners from rubbing against other particles. However, the right scale of sizes is essential for properly reproducing the statistical properties, as well as the correct value of packing fraction. A difference of even a few times $10^{-3}$ in the packing fraction can significantly increase the errors. For the packing with monodisperse cubes, we scaled the particle sizes to best fit the data, which means that we used monodisperse cubes about 275 μm across instead of the mean size of about 220 μm. Nevertheless, the total error was almost three times higher than that for polydisperse cubes. Unfortunately, the tomographic scans do not contain enough particles to let us compute and compare radial distribution functions. Figure 11.21 shows a visual comparison of the scanned sample and the various simulated packings. Since the virtual packings contain more particles, the scale of the particles in the volume appears smaller. The calculation of statistical functions does not depend on sample size, however, since resolution is taken into account through the dimensions of each voxel in microns (which differs across samples).

Results were similar for the reconstruction of the sample of salt packed in HTPB binder. For brevity, we show here only a visual comparison of the sample with the simulation and a comparison of the three-point probability functions.



(a) Tomography

(b) Polydisperse Cubes

Figure 11.22: Volume renderings of the HTPB salt sample and computer-generated packing.

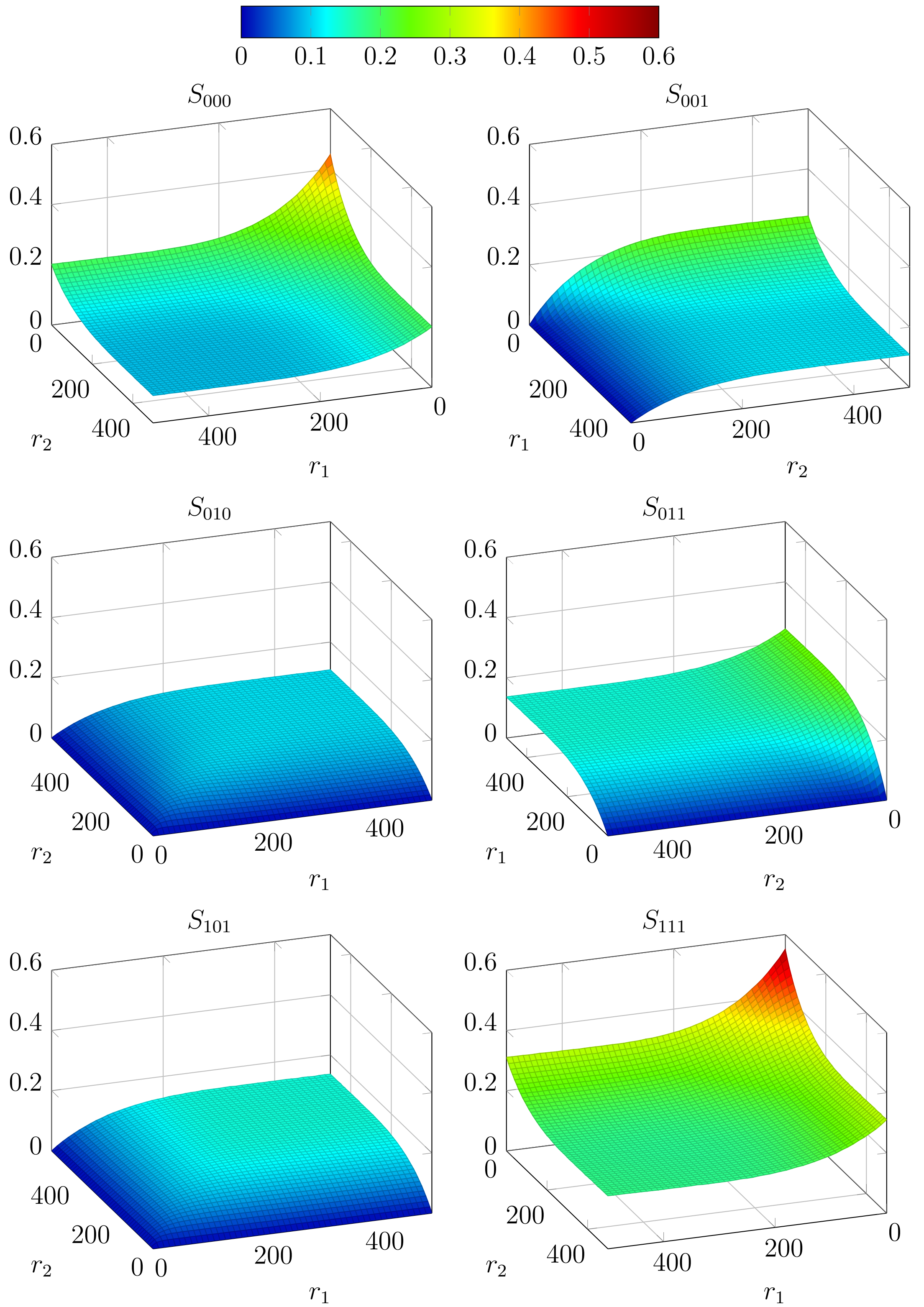


Figure 11.23: Three-point probability function for the HTPB-packed sample of salt, calculated from tomographic data.

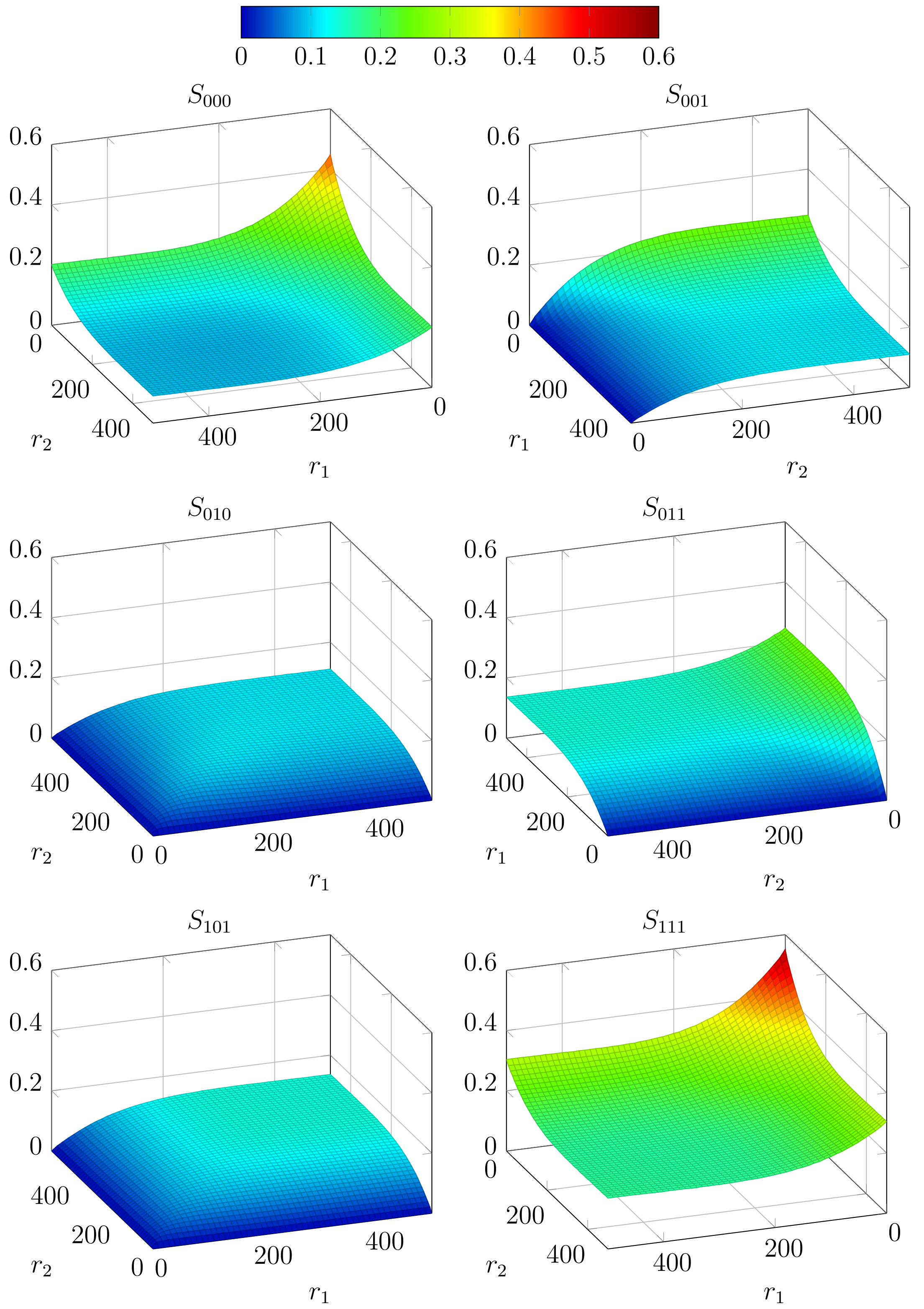


Figure 11.24: Three-point probability function for the simulated packing using polydisperse cubes.

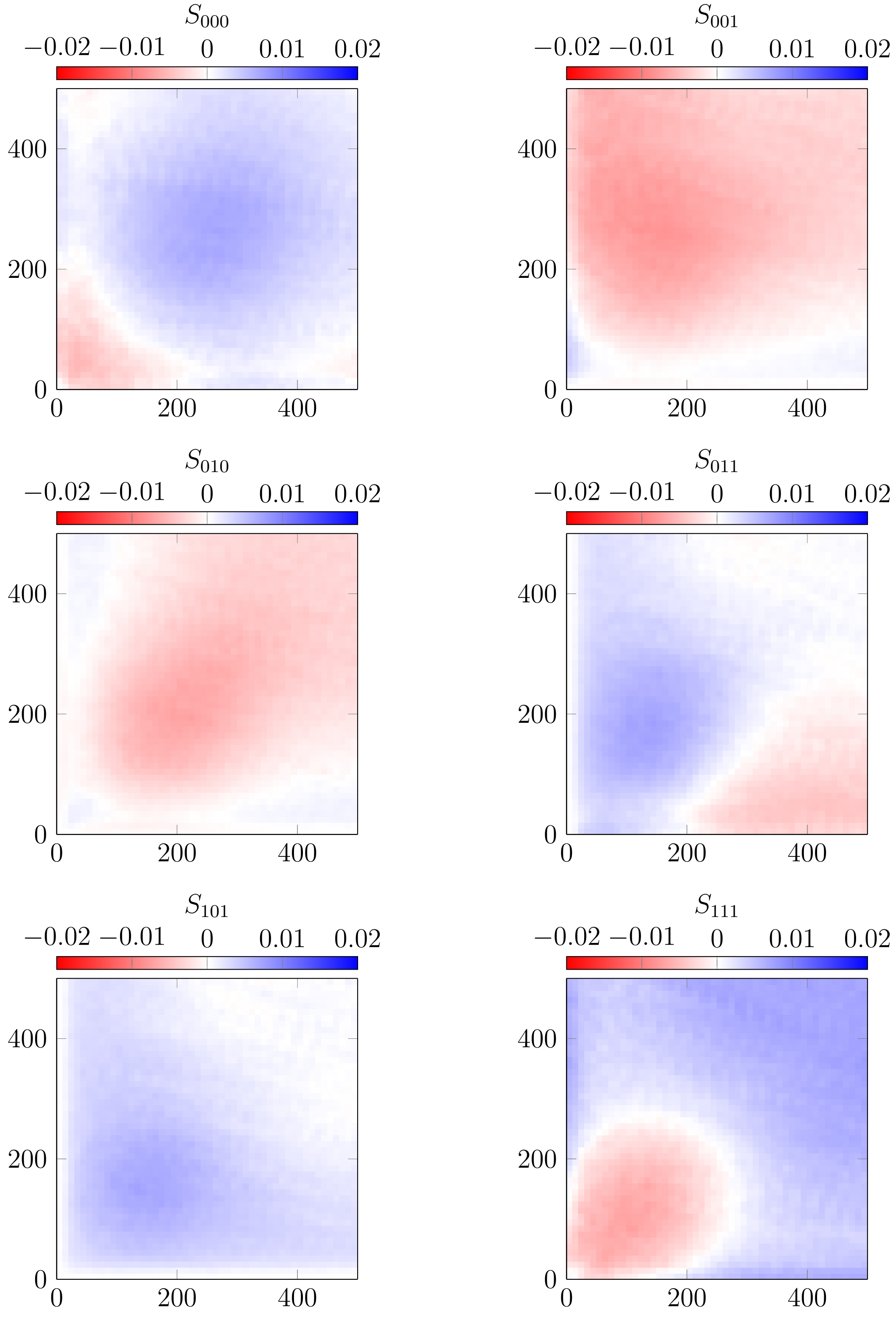


Figure 11.25: Error in the three-point probability function of the HTPB sample calculated using polydisperse cubes.

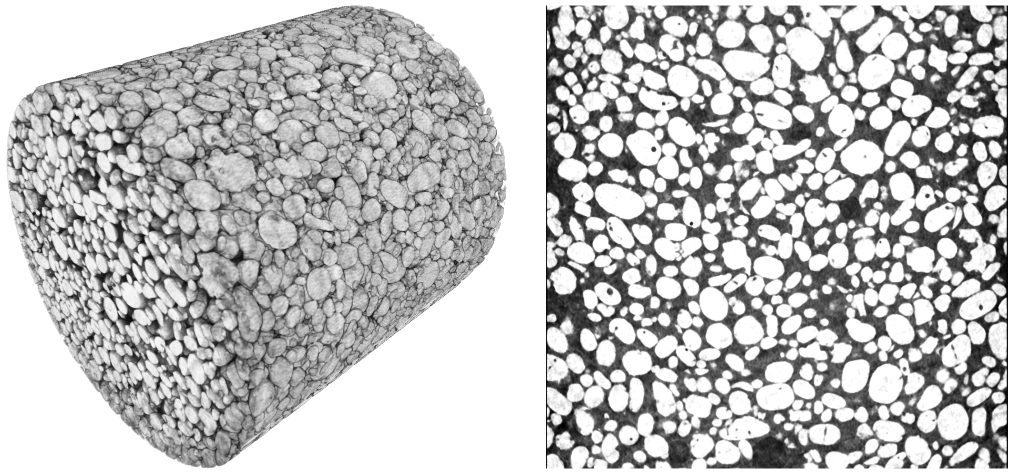
## 11.2.3 Solid Rocket Propellants and Explosive Materials

The final set of samples we reconstruct using our packing code are two common solid propellant materials and an explosive. The three samples are packed in HTPB binder. The two solid propellant samples contain ammonium perchlorate (AP) oxidizer particles, while the explosive material contains particles of the explosive 2,4,6,8,10,12-hexanitro-2,4,6,8,10,12-hexaazaisowurtzitane ($C_6H_6N_{12}O_{12}$), known as CL-20. CL-20, along with HMX, is one of the organic compounds with highest known energy density. Unlike HMX, however, CL-20 is very sensitive to impact, so the two are often mixed in high-grade explosives to increase energy content while maintaining impact sensitivity close to that of pure HMX. Both AP and CL-20 being organic materials, their contrast with HTPB in the tomography scans is not as good as that of inorganic materials like salt and glass, which contain elements with heavier nuclei (chlorine and silicon). This means that processing the volumetric images to obtain particle size and shape distributions is much harder. Only one of the three samples—containing only coarse AP particles—yielded usable results. The second solid propellant sample contained too many small particles that were difficult to resolve using the tomography machine available to us, and the CL-20 particles in the explosive sample have very irregular shapes and sizes that are difficult to characterize using simple polyhedra or ellipsoids. Nevertheless, we can still try to use visual information alone to see how well we can reconstruct such samples. Figure 11.26 shows volume renderings of the scanned volumes and cross sections of each sample.

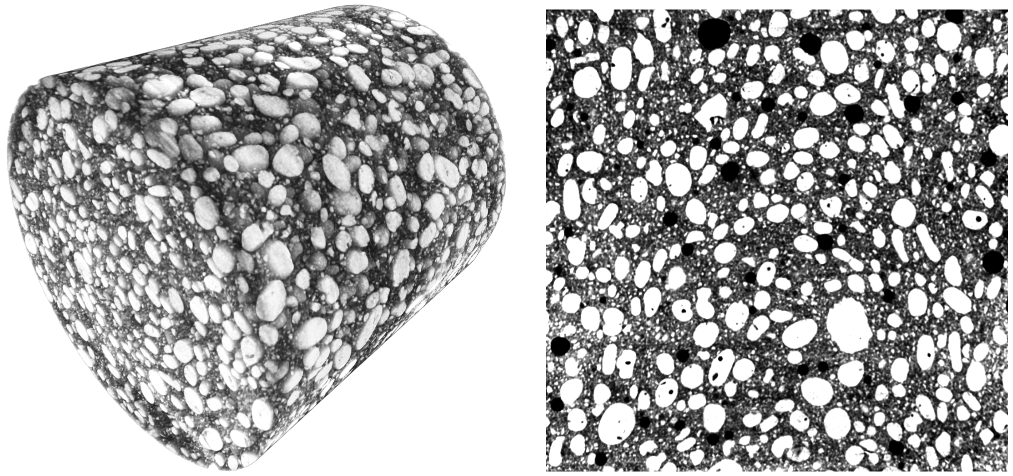
### Reconstruction of Coarse AP Sample

Since this sample is the only one for which we could obtain reliable data on the size and shape distributions of the particles, we will try to reconstruct it in more detail. First, we find the distribution of particle sizes from their volumes. We use a simple formula to obtain the size distribution using the cubic root of the volume, and later scale the particles appropriately so that their scale matches that of the sample. The size and shape distributions obtained are shown in figure 11.27. The solid curve in the left of figure 11.27 is a lognormal distribution with parameters $\mu = 4.529 \pm 0.004$ and $\sigma = 0.307 \pm 0.004$. The shape distribution was obtained using Amira's shape analysis module, which fits an ellipsoid to each particle. We divide the $[0, 1]$ interval of values of flatness and elongation into 20 bins and calculate how many particles fall in each bin. We then create a multiple of the number of particles in each bin with the appropriate ellipsoidal shape and the size distribution shown at the right of figure 11.27. The simulated packing contains 8753 particles in total.

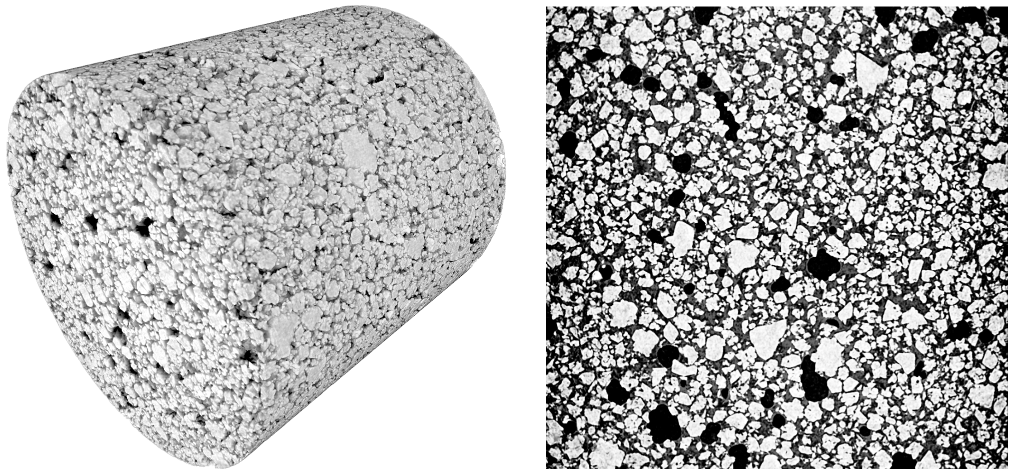
Renderings of the actual sample and the simulated packing are shown in figure 11.28. In total,



(a) Coarse AP



(b) Mixture AP



(c) CL-20 Explosive

Figure 11.26: Volume renderings and cross sections of solid propellants and explosive samples.

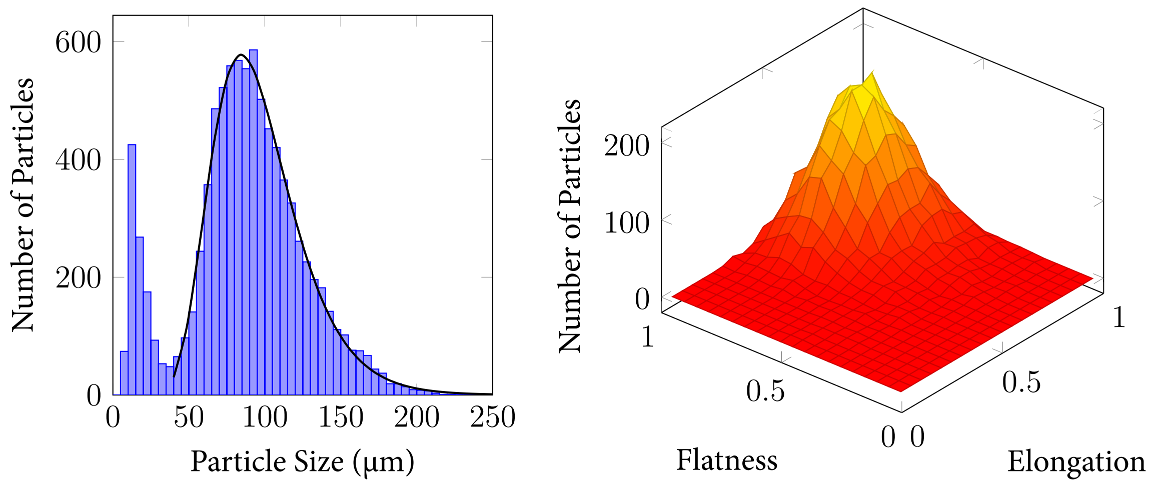


Figure 11.27: Size and shape distributions of coarse AP particles

105 different shapes of ellipsoids were used in the packing. Particles too far into the tail of the distribution of flatness and elongation were discarded. Using a distribution of particle sizes and different shapes is enough to reproduce the statistical characteristics of this sample quite well. Figures 11.29 and 11.30 show the three-point probability functions of the sample and the simulated packing, and figure 11.31 shows the difference between the functions of the simulated packing and those of the sample itself. The minimum and maximum differences across all six plots is -0.0078 and 0.0068; the $L^2$ error norm is 0.367558, which, divided by the number of grid points yields $1.41 \times 10^{-4}$. This is the smallest difference across all samples we have reconstructed—with the exception of the glass sample, for which we compare the more sensitive radial distribution function instead.

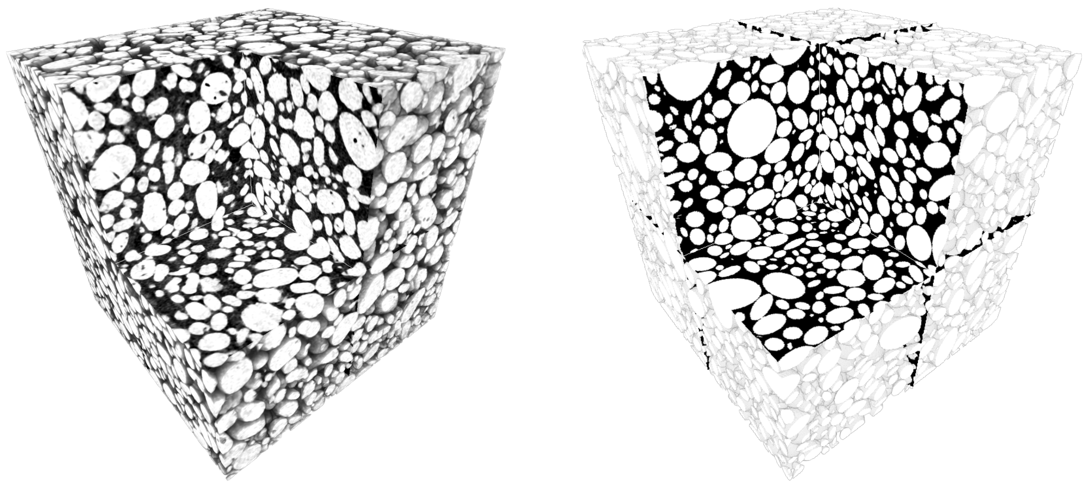

(a) Tomography (b) Simulation

Figure 11.28: Volume renderings of the coarse AP sample and computer-generated packing.

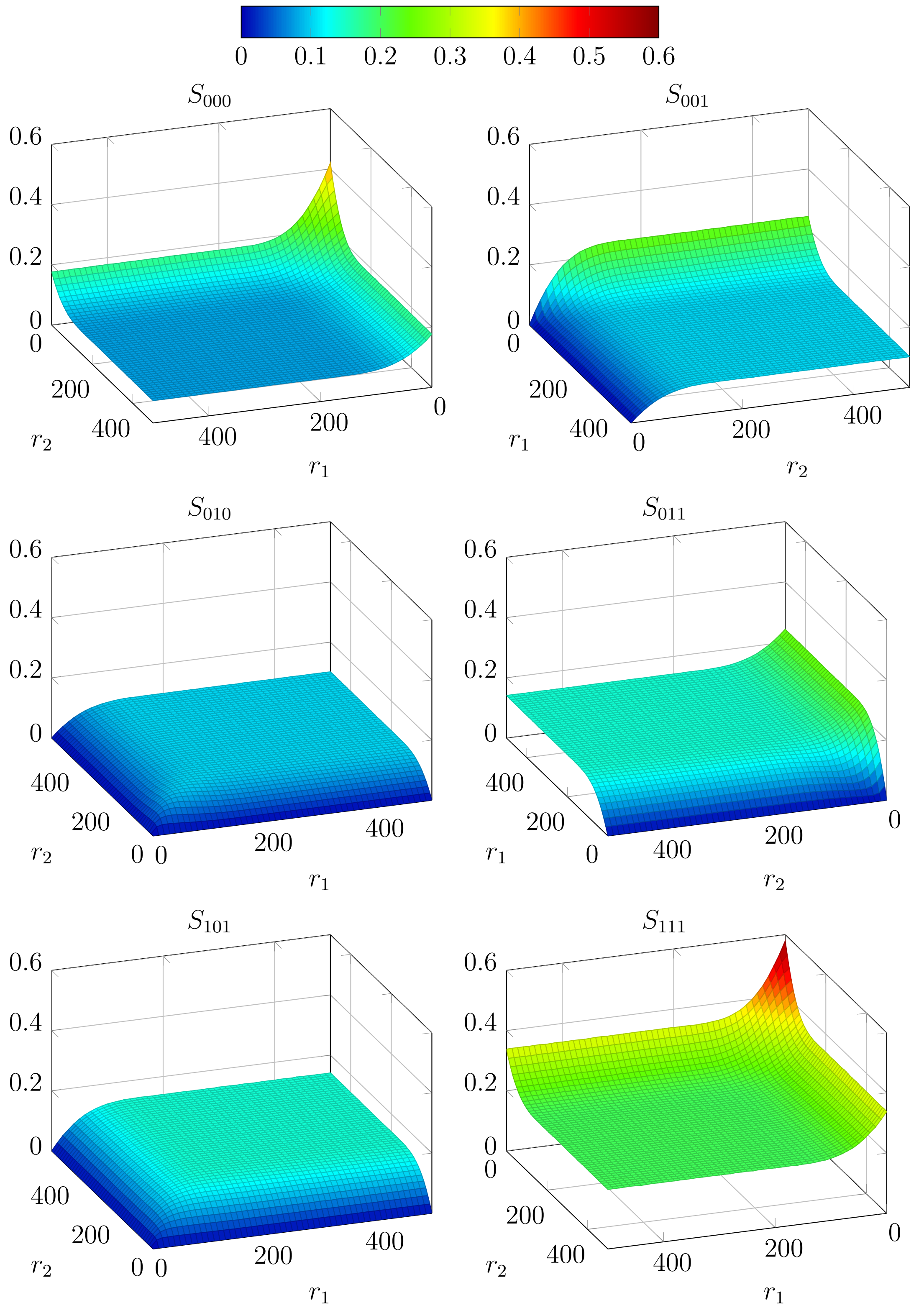


Figure 11.29: Three-point probability function for the coarse ap sample, calculated from tomographic data.

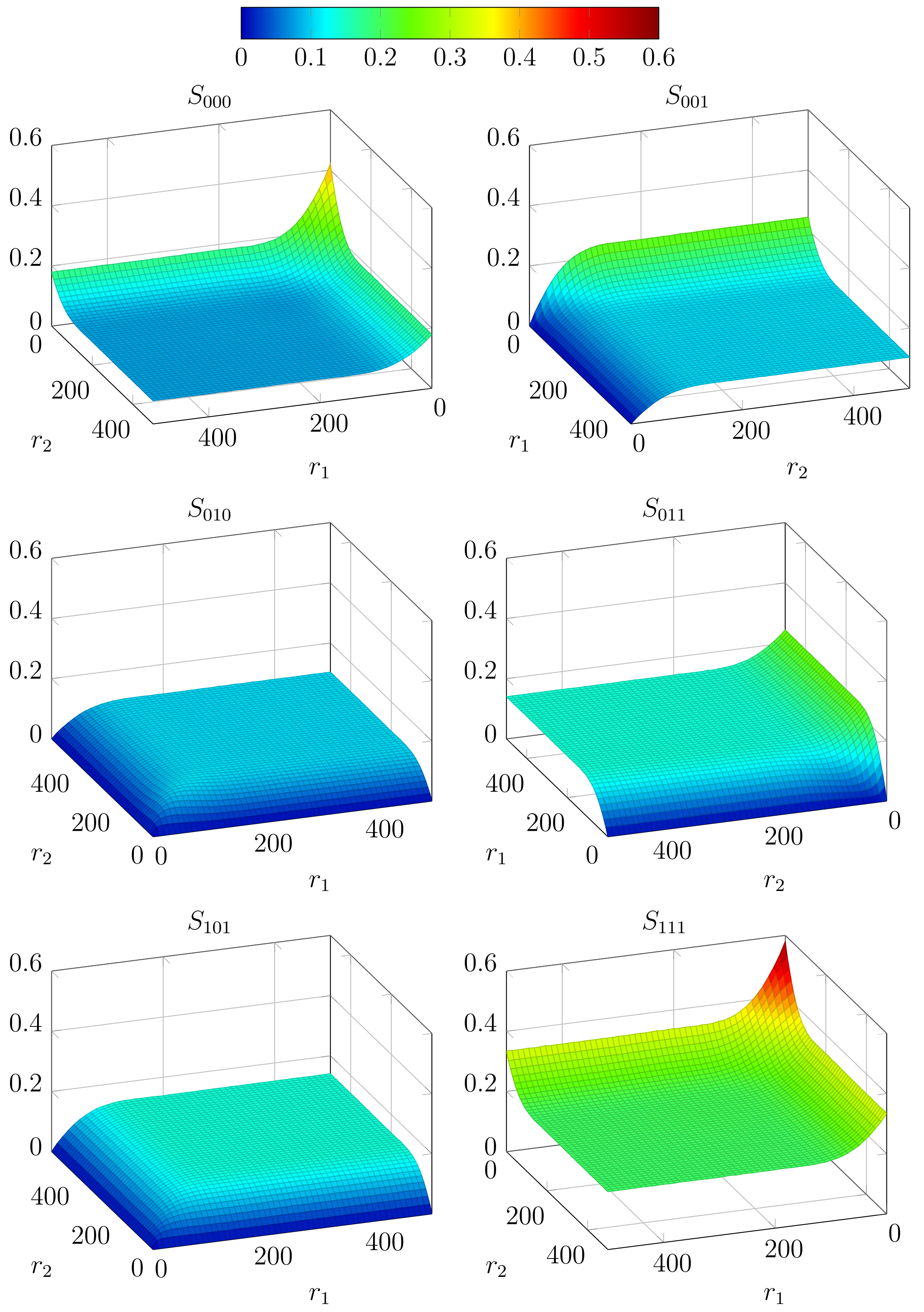


Figure 11.30: Three-point probability function for the simulated packing using polydisperse ellipsoids.

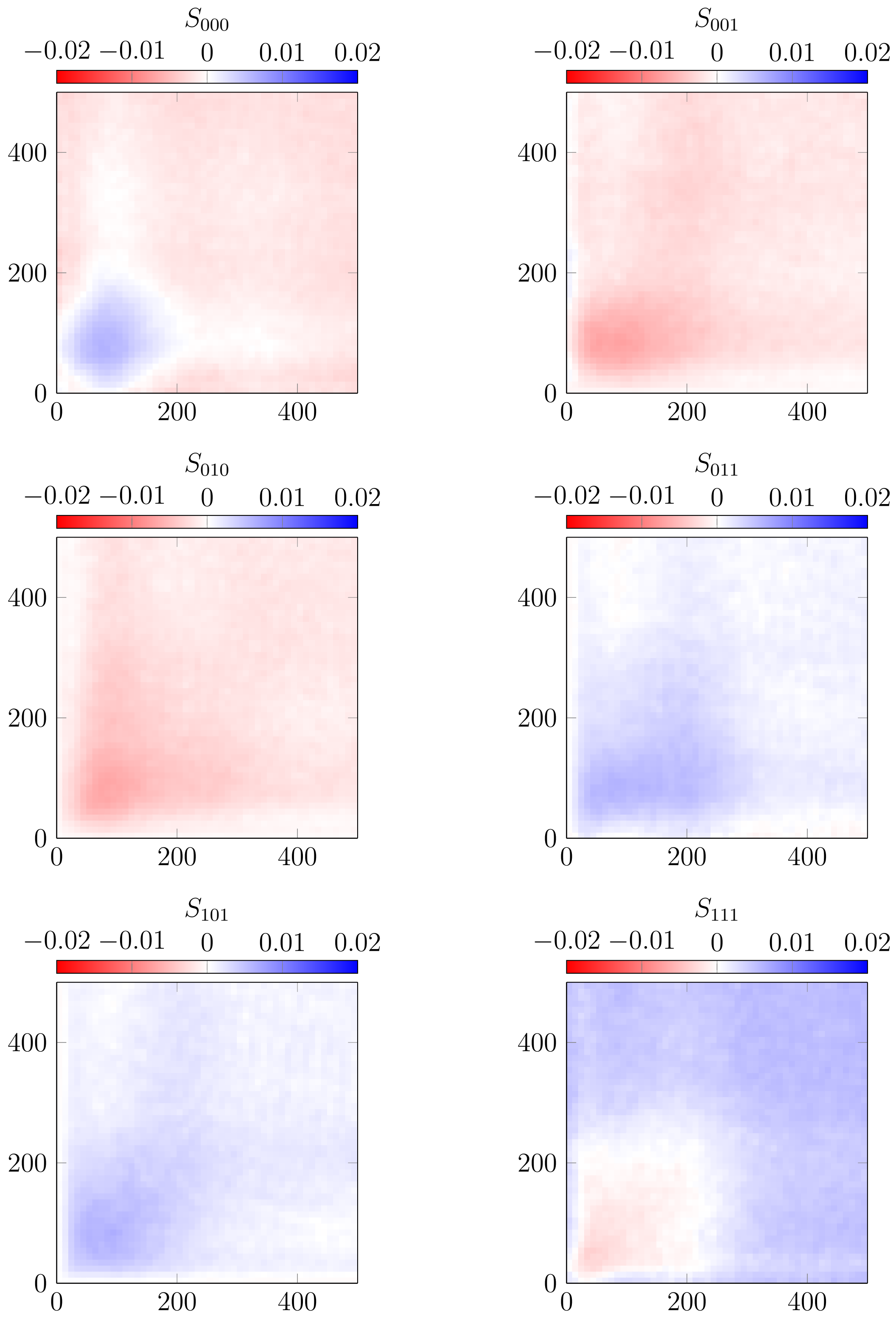


Figure 11.31: Error in the three-point probability function of the simulated coarse ap packing.

### Reconstruction of AP Sample with a Mixture of Coarse and Fine Particles

The propellant sample containing only coarse AP particles yielded reliable data on the distribution of particle sizes and shapes. The scan of the polydisperse sample, however, was difficult to process due to the large number of small particles. Many large large particles remained clustered together even after the particle separation step of the image analysis as well. Perhaps with more reliable image processing software it could be possible to properly separate the particles and reproduce the packing accordingly. Since Amira is the only software available to us, we need to attempt to reproduce the sample using only the image of the sample and the specifications we received with the sample. The sample was prepared at Purdue University, with a mixture of coarse and fine powders of AP, with roughly the same weight proportions. The coarse particles are in principle roughly 200 μm across and the fine particles, about 20 μm. However, these are only rough estimates. We would like to know how well such a specification can be used to reconstruct the sample. Therefore, we use two different particle sizes in our simulation: 130 μm for large particles, and 25 μm for small particles. Although the coarse particles are rated at 200 μm, we decided to use smaller particles based on the distribution obtained for the previous sample. However, since there are many particles larger than 100 μm in the sample, we decided to choose a size somewhat higher than the median value in the distribution in figure 11.27, so that all particles could be approximately represented by a single size. In this particular case, our intention is not to reproduce the sample as exactly as possible as we have done in the preceding sections, but to use a suboptimal case to try to understand what can be learned from the differences that arise in the statistical properties. The proportion between large and small particles, for example, was adjusted only visually. Nevertheless, we do try to keep the shape of the particles well represented by using the same distribution of flatness and elongation found for the previous sample. We used the same 105 ellipsoidal shapes, but this time only in two different fixed sizes rather than using a lognormal distribution. The simulated packing contains 45476 particles in total (1711 large particles, and 43765 small particles). A volume rendering of the result is shown in 11.32, along with the actual sample. The simulation looks similar to the material, but can the statistical functions tell us if they really are similar? The three-point probability functions for the tomographic data and the simulation are shown in figures 11.33 and 11.34, respectively. Although the plots do look quite similar, there is some information we can extract from the differences between the two, shown in figure 11.35. For short distances, up to about 150 μm, the difference is small across all plots. This is the same scale of the particles themselves, so at this scale, the small amount of error indicates that the shape of the particles is accurate. However, if we look at the plot for $S_{111}$ we see that it is underestimated at the scale of about 100 μm. Similarly, in the plot of $S_{010}$, the region around 100 μm is underestimated, while there are two regions around 300 μm that have a slight overestimation. The most visible of all differences, however, is in $S_{000}$, which seems correct around 100 μm, but is overestimated at larger scales. These are all signs of the same thing: in choosing only two sizes of particles, we have neglected the mid-sized particles. Looking closer at the renderings of the sample and the simulation, it is easy to notice that the voids between particles seem larger in the simulation than the actual sample, since in the actual sample there are mid-sized particles that bridge this gap. Although the overall packing fraction is the same, a fact that can be confirmed by the small error in $S_{111}$, when we have three points that are more distant apart than the scale of a large particle in the simulation, the probability that they all fall in the void region is slightly larger because there are no mid-sized particles to fill those gaps. Indeed, the differences in the statistical functions do tell us what is wrong with our simulation. This is a good thing, since this information can be used to improve later simulations by including the right amounts of each particle type. Could we know when we are using the wrong particle shapes in a simulation using the same statistical functions? We investigate that in the next section.

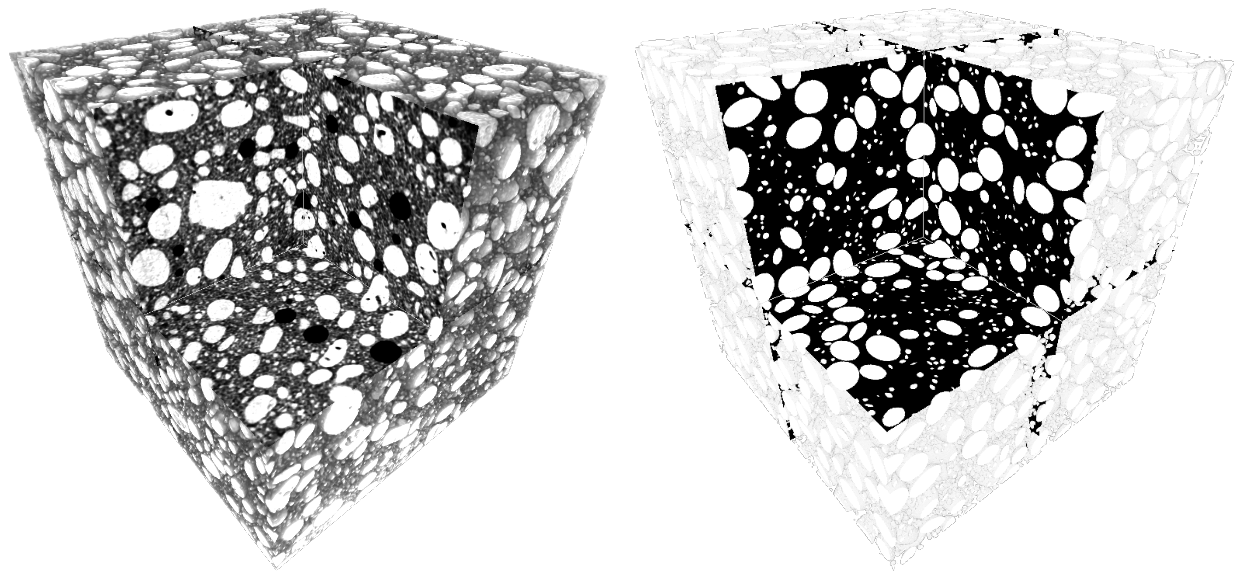

(a) Tomography (b) Simulation

Figure 11.32: Volume renderings of the AP sample with coarse and fine particles and computer-generated packing.

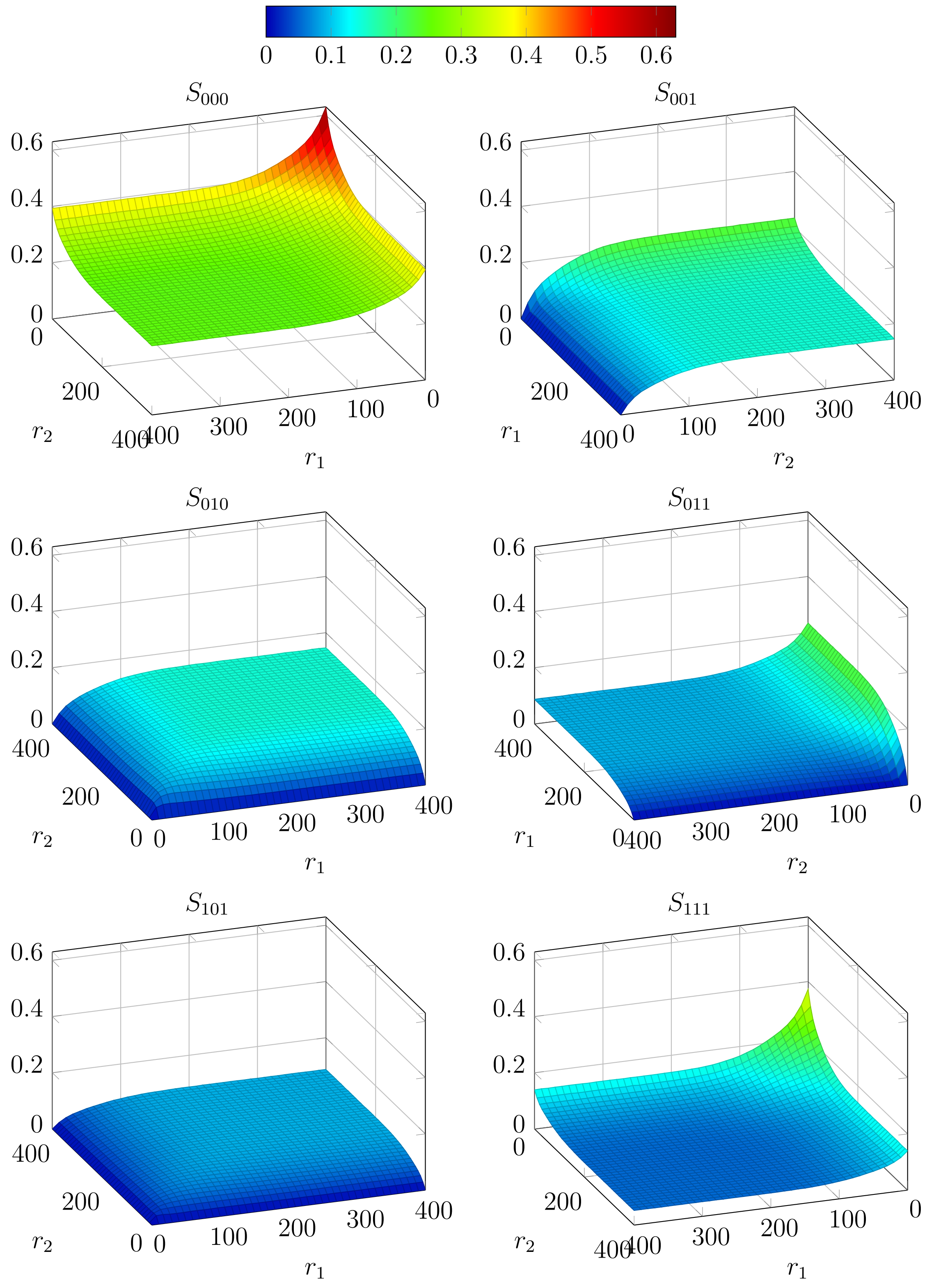


Figure 11.33: Three-point probability function for the polydisperse AP sample, calculated from tomographic data.

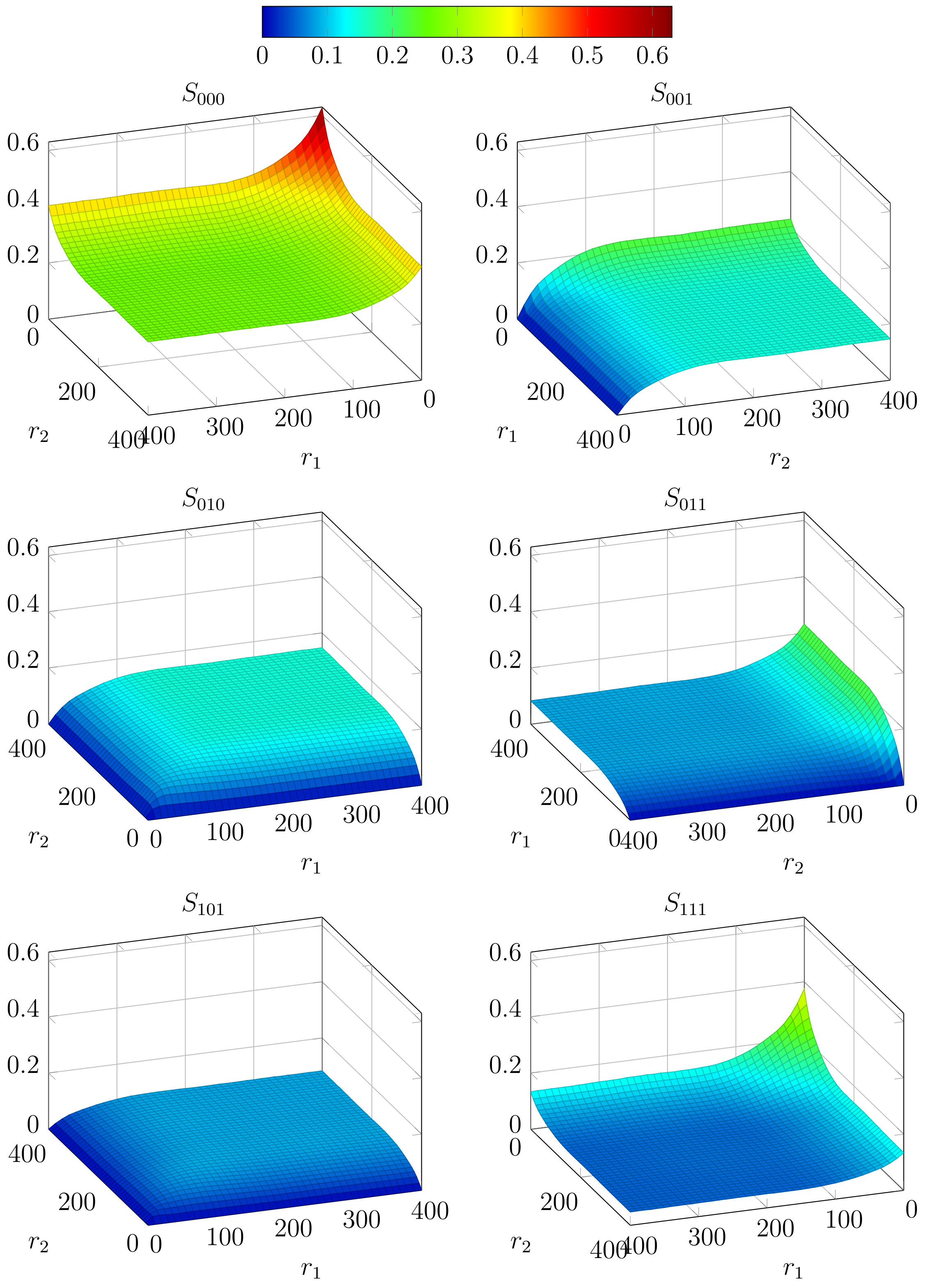


Figure 11.34: Three-point probability function for the simulated polydisperse AP packing using ellipsoids.

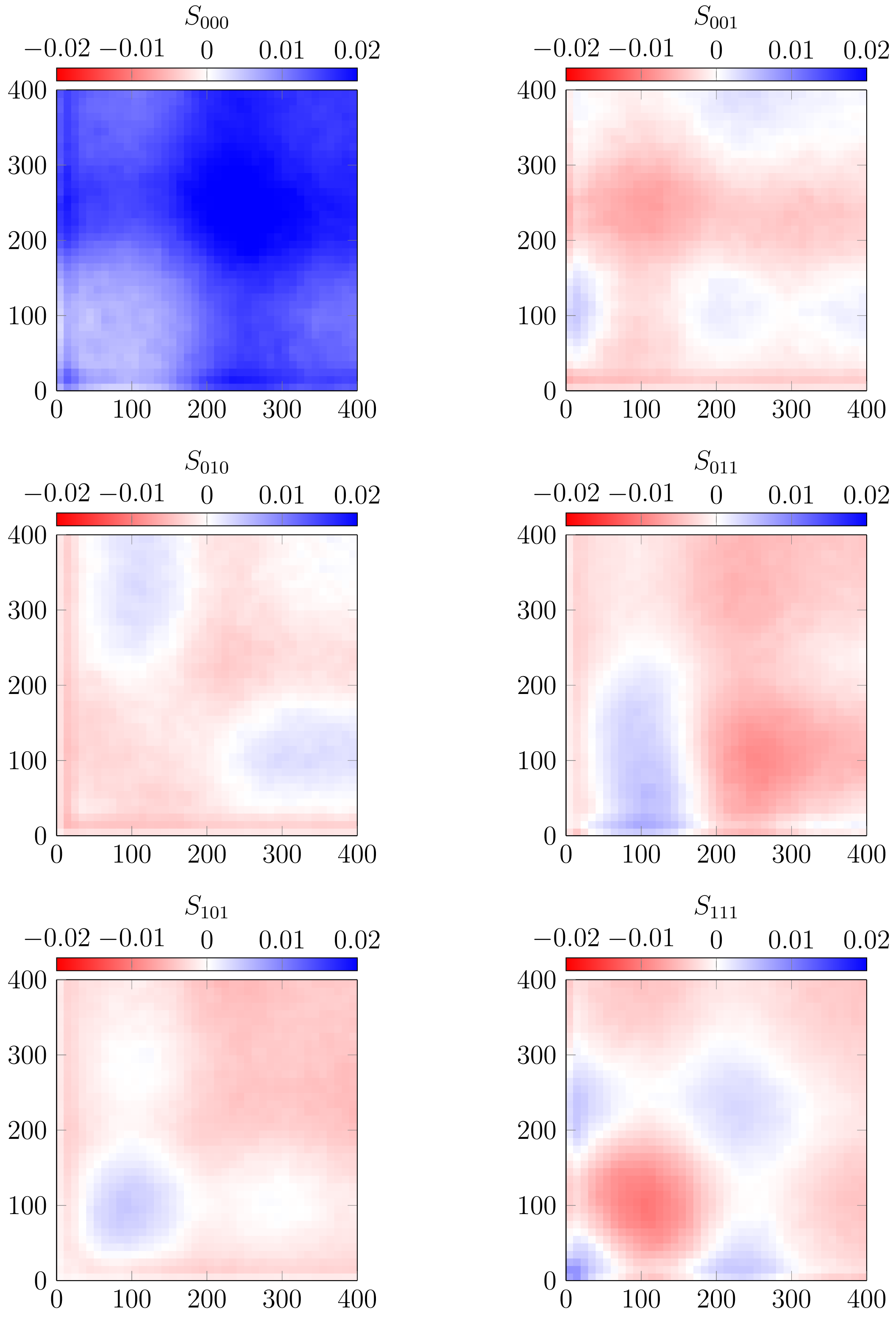
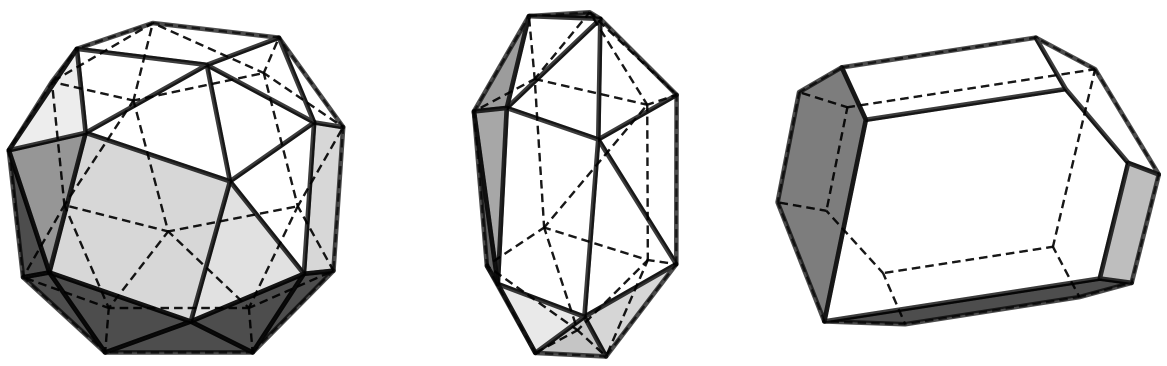


Figure 11.35: Error in the three-point probability function of the simulated polydisperse AP packing.

## Reconstruction of CL-20 Explosive Sample

The last sample we reconstruct is a sample of the explosive material CL-20. This sample is very challenging to reconstruct, since the broken crystals of CL-20 have irregular shapes. Resolving individual particles in the sample is difficult as well, so we cannot rely on any information provided by Amira. Looking at a cross section of the sample, we can see that there are both large and medium size particles. Since size and shape information is limited, we attempt to reconstruct the sample with a mix of three different types of polyhedra and two different sizes, at a ratio of 3.5:1. To make the size distribution somewhat more realistic, we use a narrow Gaussian distribution of sizes for each particle type with a standard deviation of about 12%. The particle shapes used in the simulation were: a roughly spherical polyhedron, the snub cube; a somewhat elongated polyhedron, the snub triangular prism; and, finally, a polyhedron derived from crystals of HMX. The polyhedra are shown in figure 11.36.



(a) Snub Cube (b) Snub Triangular Prism (c) HMX

Figure 11.36: Shapes used in the simulation of CL-20 particle packing.

The simulated packing contains 31500 particles, of which 30000 are small and 1500 are large. The packing density by weight is roughly 70%, which translates to about 50% of the volume filled with particles, since CL-20 is denser than the HTPB binder. The end result is shown in figure 11.37.

The three-point statistical function for both the sample and the simulated packing are shown in figures 11.38 and 11.39. As with the other samples, the differences are difficult to be visualized just by visual inspection. Since the packing density is the same, several characteristics of the function are reproduced automatically, such as the value for $r_1 = r_2 = 0$, which is just the value of the volume fraction for the given phase for $S_{000}$ and $S_{111}$, or zero otherwise, since a point cannot fall into two different phases. Also, in the absence of large voids between particles, the function converges faster to the square of the volume fractions at large distances. From the plot of the difference be-

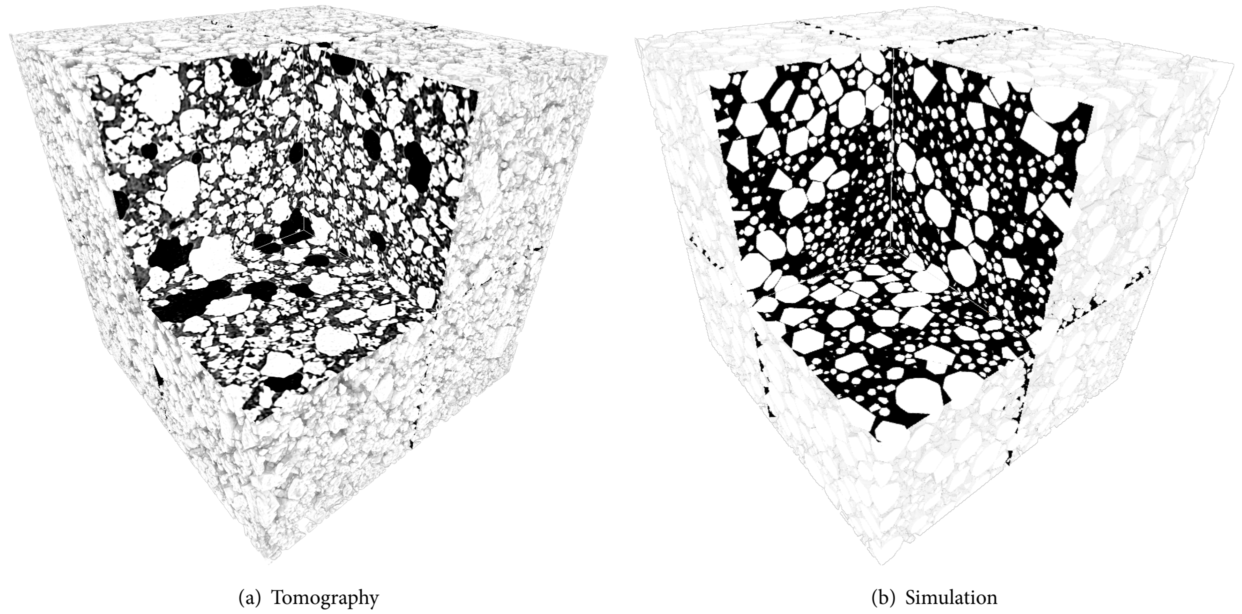

(a) Tomography (b) Simulation

Figure 11.37: Volume renderings of the CL-20 sample and computer-generated packing.

tween the three-point statistical functions of the simulated packing and the actual sample, shown in figure 11.40, we can see that the discrepancies are largest at the scale of the particle sizes, around 100 µm. On the other hand, at large scales, the values to which the functions converge have a much smaller error. This is a clear indication that the shape we have used for the particles is not similar to the shapes of the particles in the sample. The function $S_{000}$, for example, is lower in the simulation than in the sample throughout the entire region we calculated. Indeed, since many of the particles of CL-20 are concave, it is more likely that points will fall in the binder/void phase more frequently, especially at small scales. That is also why $S_{111}$ is lower at lower scales. When the distance between random points in the material is roughly the same as the scale of a particle, there is a higher probability for one of them to fall into the binder/void region, since particles are concave and have ragged boundaries. When we decide to use convex particles only, the probability that points fall all in the same phase at small scales is higher. This is also clear in the function $S_{011}$. When the mid point falls inside a particle, the probability that one of the others will also fall inside a particle is overestimated, while the probability that they will fall outside is underestimated. In the function $S_{001}$, this trend is reversed, which means that when the mid point falls in the binder region, the probability that one of the other points will also fall in the binder region is underestimated, while the probability that one of the other points falls inside a particle is overestimated. Not surprisingly, the minimum and maximum differences are the largest when compared with other reconstructed samples, at -0.028 and 0.021, respectively. The $L^2$ error norm is 0.8725, or $5.2 \times 10^{-4}$, when divided by the number of grid points. Even the monodisperse cubes used to describe salt performed better. It is certainly possible to improve the reconstruction of this sample with better choices for the particle shapes and more accurate particle size distributions. The differences in the statistical functions can serve as a guide towards those better choices. The size distribution could be measured independently via laser diffraction, for example. However, a better reconstruction should probably be accomplished with the addition of concave particles—something our code is not yet capable of packing. For the moment, convex particles are still capable of reproducing a wide variety of materials—not only glass, salt, and propellants, but also concrete, foams, emulsions, colloids, among many other materials.

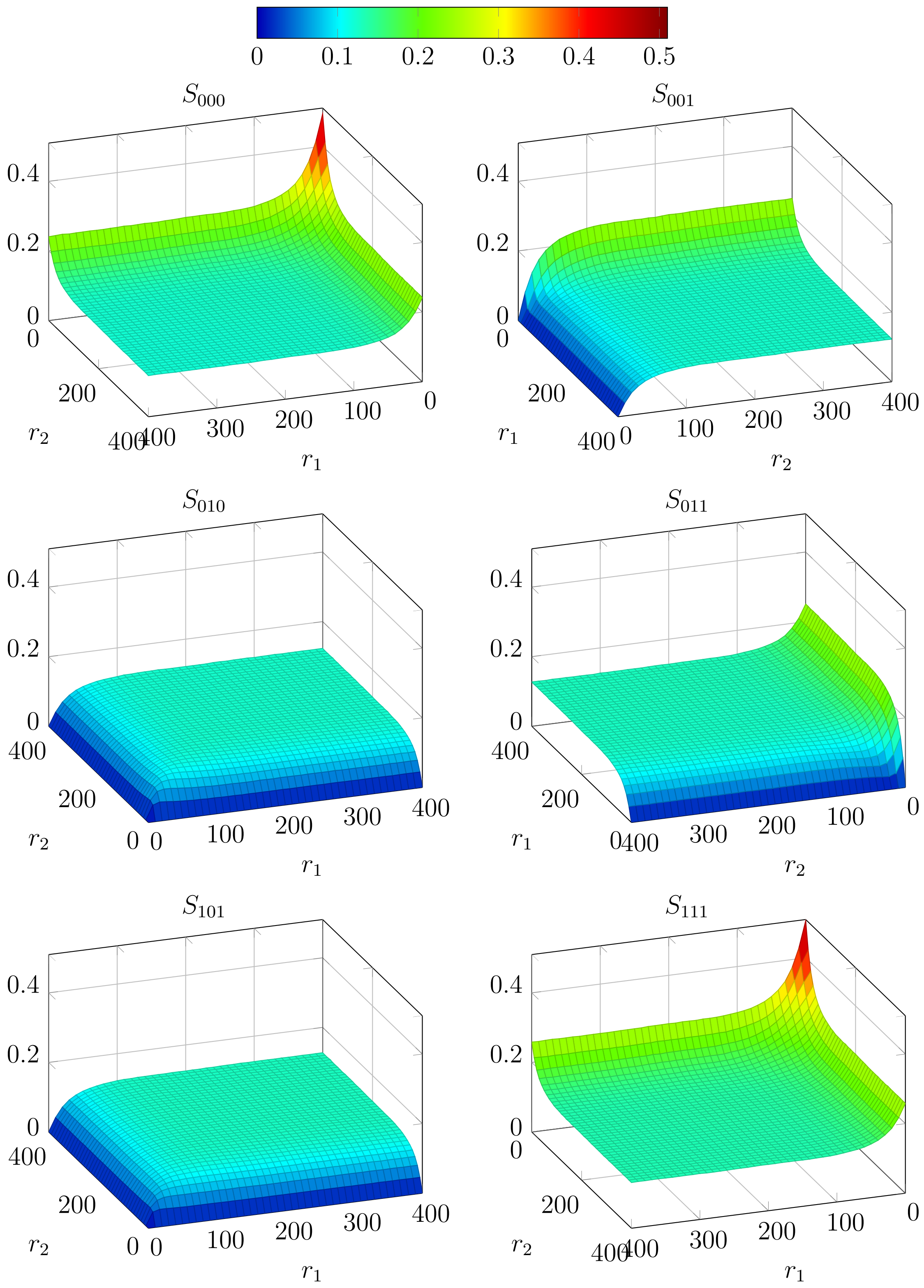


Figure 11.38: Three-point probability function for the CL-20 sample, calculated from tomographic data.

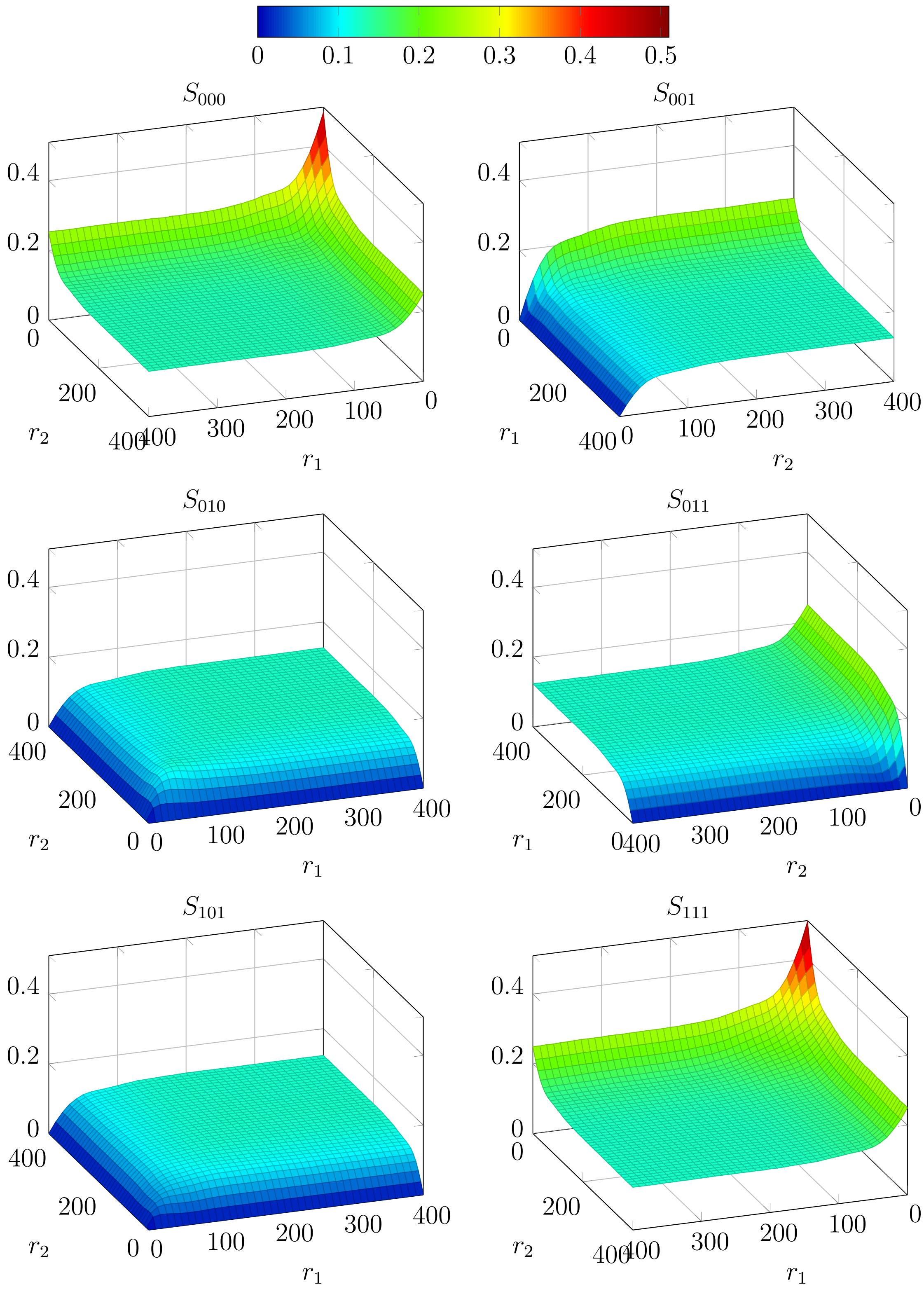


Figure 11.39: Three-point probability function for the simulated CL-20 packing using polydisperse polyhedra.

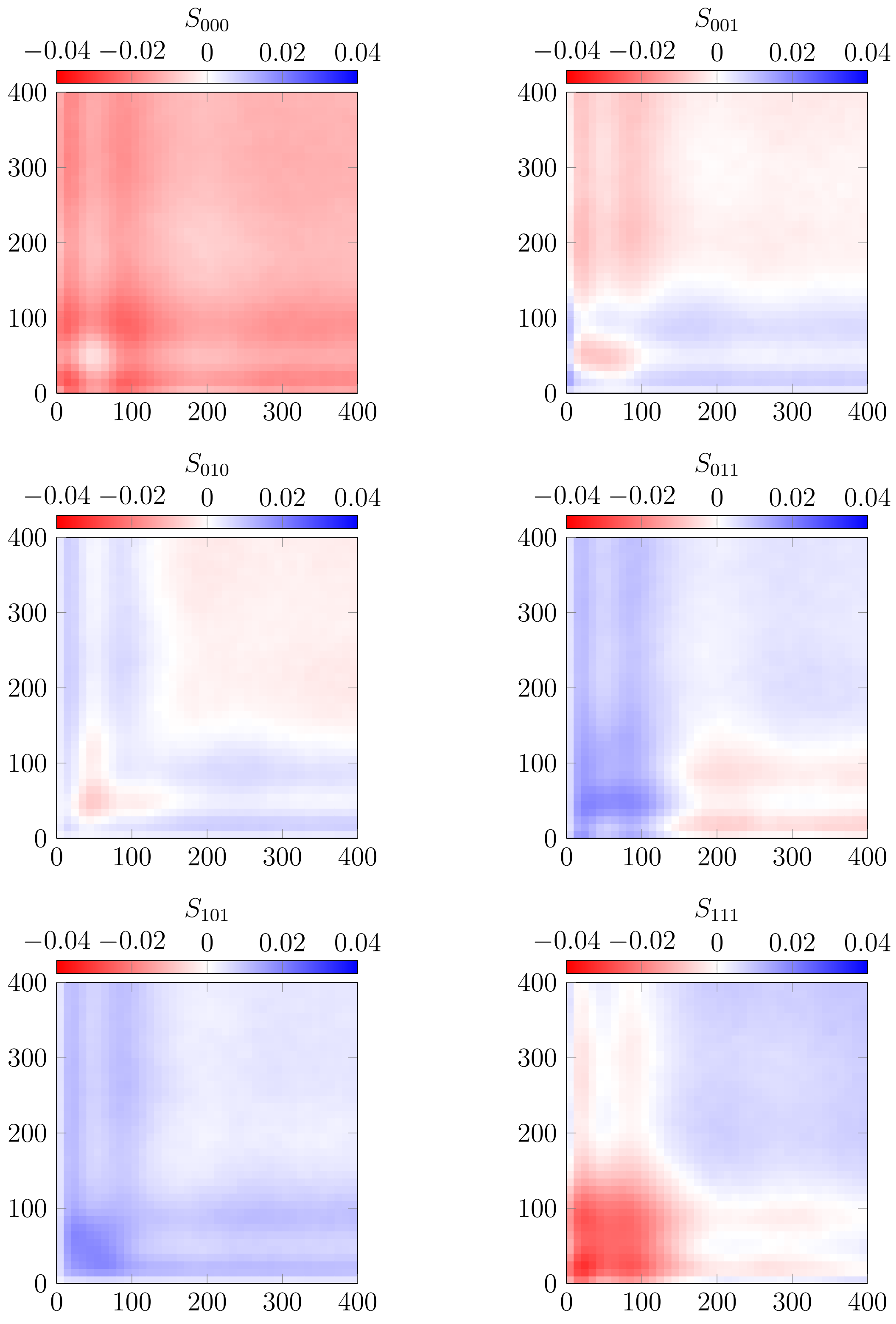


Figure 11.40: Error in the three-point probability function of the simulated CL-20 packing.

# Chapter 12

# Future Work

In the preceding chapters we have presented a set of tools developed to simulate and statistically analyze microstructures of particulate materials. This set of tools uses new and efficient algorithms to provide a significant performance leap over previous implementations, enabling the simulation of much larger systems of particles while also taking considerably less time to do so. The new codes also introduced several new features not present in other software, such as the possibility to mix all of the particle types together in the same simulation, continuous particle size distributions, and new boundary types for packing spheres. Nevertheless, in a project like this, in which software development is the main product of research, there will always be interesting features left behind due to time constraints. This project is no exception. While our main goal of enabling the simulation of more complex structures has been accomplished, there is still much that could be done to improve the codes we have developed. We leave suggestions for some of these improvements in the following sections as a reminder to ourselves and as guidelines to others that might also want to improve the code.

## 12.1 Code Design

The main objective of our implementation of a particle packing code was to provide a robust, high-performance code capable of simulating large systems that were previously impossible to simulate. We have put much effort into the design of the basic elements of the code and in the choice of algorithms that we used, keeping performance, clarity, and simplicity as top priorities. Unfortunately, the design process leading to our current implementation is not as well documented in this work as we would like. It could have been useful to save others from incurring into the same mistakes we had to go through before we reached a reasonable implementation. Since it was not possible to refine the design in all areas, there is inevitably some technical debt left to be repaid in future implementations. In most cases the technical debt is due to lack of time for further development, but there are also instances where simplicity was favored over performance. This is the case of the algorithm for collision prediction, for example, that could have used more sophisticated methods for find the time of impact, such as Brent's method, but uses a simple bisection method instead.

The most obvious improvements that could be easily implemented are in the user interface of the code. We have a simple text input file format and visualization window for the simulation in the code. The input and output file formats are supported by a simple implementation of a recursive descent parser. It allows the user to set variables that control the behavior of the code, create new shapes, and declare how many particles of each type must be created in the simulation. However, many variables controlling the code are still not exposed through this interface, so they must be changed inside the code and require a recompilation. Among these variables are the floating point tolerances used in several algorithms to compute particle intersections between themselves and other particles and boundaries, variables to control the camera view of the visualization window, frequency between progress updates, choice of collision response mechanism, etc. Exposing these variables would give the user more control over the simulations, although at the same time it would make the code more prone to failure due to poorly chosen parameters. (Admittedly, several of these parameters required some fine tuning to prevent catastrophic failures due to floating point rounding and truncation errors.) Improvements could also be done to the OpenGL interface of the code. The current interface is fairly simplistic, with mostly hard-coded parameters for the camera view, lighting conditions, etc, and a small list of commands to control the visualization. Our plan was to have a more sophisticated interface that would automatically center and zoom the camera to fit the packing simulation, record videos, and allow the user to create packings without the need to write input files. However, numerical problems in some algorithms, that prevented the code from working, created unexpected delays during the development of the code. Therefore, noncritical areas such as the graphical and input/output interfaces received less attention than deserved. We intend to remedy this situation by continuing to develop the code in the future.

The first thing we would like to address in the near future are glaringly missing features, such as the ability to pack polyhedra in all of the same new boundary types introduced for spheres. The solid box and solid sphere boundaries are not too complicated to implement, but the same cannot be said for boundaries with more complex geometry such as the wedge of an annulus, or a torus.

## 12.2 Proposed Improvements for the Packing Algorithm

We currently have two different codes: one for packing spheres, and another for packing general convex shapes (including spheres). Initially, we planned to use the Lubachevsky–Stillinger packing algorithm to simulate packings of both spheres and polyhedra. After much chagrin over numerical issues in the Lubachevsky–Stillinger code for polyhedra, we felt compelled to abandon it entirely and develop a different packing algorithm to pack polyhedra. The numerical problems arise when trying to compute the collision normal of two particles that are almost touching—floating point errors yield normal vectors slightly off the correct direction, which in turn means that the applied collision impulses are not effective to make particles move away from each other. If particles don't move away from each other, the code then gets stuck on an infinite loop at low densities.

A thorough research in the literature then led us to the conclusion that the adaptive shrinking cell scheme by Torquato and Jiao [51, 52] or something similar was the only real option going forward. However, we had spent a considerable amount of time developing the current code and starting from scratch one more time was not a viable option anymore. In the end, we decided to reuse the existing code for the Lubachevsky–Stillinger algorithm to implement a new hybrid algorithm based on molecular dynamics and Monte Carlo trial displacements as described in chapter 7. The resulting code turned out to be quite reliable and outperformed its predecessor by a few orders of magnitude, although the adaptation and late development have left their marks. There are a few areas in particular that we believe deserve more attention going forward, since there is an enormous potential for further performance gains over the current implementation.

The Lubachevsky–Stillinger packing algorithm is inherently serial, since at each step the next event is computed, and events must be processed linearly in time. When a collision happens, the collision response via an impulse is computed and applied once; it is also a serial process. However, in the new algorithm for packing polyhedra, collision impulses had to be abandoned because of numerical issues, so the colliding particle pair is instantly translated and rotated away using trial Monte Carlo displacements. In the current version of the code, displacements are tried sequentially until one of them can be accepted that effectively prevents the particles from growing onto each other. With some modifications to the code, these displacements could be tried in parallel using threads, resulting in a significant speedup of the code near jamming, when most trial moves are rejected due to intersections.

The Monte Carlo trial displacements also depend on several parameters that greatly affect performance, such as the number of trial moves for each collision, and how the size of trial displacements should change according to several parameters such as the number of past attempts, the current volume fraction, the size of the particles, etc. In the current implementation we have utilized an exponentially decaying displacement size that varies with these parameters, and used a few test cases to roughly optimize their values. However, the optimal parameters also depend on the shapes of the particles being packed, such as the asphericity, for example. Tetrahedra cannot be packed much further than 0.55 with the current parameters that were determined for other particles, but it is possible to change these parameters to allow packing tetrahedra up to about 0.7 packing fraction. Such a systematic study of how to tune trial displacements depending on particle shape, including when to perform a full shaking of the packing to allow unjamming, is something that we have painfully left behind due to lack of time.

Nevertheless, paralellizing and tuning Monte Carlo displacements are not the only means to improve the performance of the code. Another option is to develop a deterministic algorithm that can reconfigure the system to avoid intersections without relying on randomness. Torquato and Jiao have shown this to be possible for spheres [80] using a linear programming algorithm to optimally dislocate a sphere relative to its neighbors at each step. This allowed them to produce packings of spheres much more efficiently compared to packings generated with their Lubachevsky–Stillinger code. However, they have not implemented an equivalent algorithm for polyhedra, choosing instead to rely on Monte Carlo trial moves as well. One idea we have for such an algorithm is to use a combination of attraction and repulsion forces between features of the polyhedra to move them into a new minimum located in the interstice between all neighbors. That is, given a list of all neighbors of a given particle that is about to collide with another, compute mutual forces between the particle and its neighbors such that iterating over small consecutive displacements given by these forces, the position of the particle will converge to an optimal minimum that will maximize the amount of time until the next expansion. This could be accomplished, for example, with repulsive forces between particle centroids and rotational forces that promote face–face contacts, as they tend to optimize packing fraction. A danger of such approach is that the interactions between particles during the packing process could lead to unrealistic configurations. On the other hand, this might be the only option to generate materials that are random, but also not isotropic, such as a particulate material with particles roughly aligned in a given direction. While our code can be used to produce anisotropic materials, the possibilities are fairly limited and currently require directly changing the source code.

## 12.3 Proposed New Features

Despite all of the possible improvements to the current code that certainly should take precedence over the implementation of new features, this chapter would not be complete if we did not present what our vision is for what our code could become in the long term.

The packing code and auxiliary tools presented in this work have been implemented with material modeling in mind. The design of the code and the decisions about whether or not to support certain features were thus strongly influenced by how they could impact typical microstructure modeling simulations. For example, our code does not support a deformable boundary that ensures strict jamming of the particles, since that is not an important feature of the microstructures of solid rocket propellants. On the other hand, the shape and size distribution of the particles are quite important, and our code does support user-defined polyhedral particle shapes created in external files, as well as discrete and continuous particle size distributions. The design of the hierarchical grid of particles, for instance, needed to take into account the high polydispersivity of propellants, and the choice of the GJK algorithm for particle intersections was made based on its excellent performance for polyhedral particles, which are found in solid propellants and explosive materials, that commonly contain crystalline particles. Nevertheless, the choice of the GJK algorithm does limit the particle types to convex shapes. Moreover, the lack of support for deformable boundaries—and solid boundaries for polyhedra—also restrict the classes of problems the code can be applied to. Therefore, new boundary types and more complex particle shapes are significant improvements that should be implemented in the future. New boundary types would be useful to study the crystal structure of materials, to study boundary effects in the flow through porous media, etc, while support for concave shapes and more complex shapes based on triangle meshes or constructive solid geometry would enable the simulation of more types of materials.

In addition to features that could be implemented into the packing code itself, there are several new auxiliary tools that could be developed to provide further insights into the simulated materials. The statistical tools we currently provide cover only a fraction of the functions discussed in chapter 3. We have tools for computing $n$-point probability functions, radial distribution functions, etc, and it is also possible to use the packing code itself to calculate the contact network, some benchmarks of the packing algorithms, and to generate simple Cartesian meshes. Tools for computing pore size distributions could be very useful in the development of aluminum agglomeration models, for example. A complete set of statistical analysis tools would greatly increase our ability to understand how the effective properties of heterogeneous materials are affected by their microstructure.

# Bibliography

[1] G. P. Sutton and O. Biblarz. *Rocket Propulsion Elements*. Wiley, 2010.

[2] J. C. Michel, H. Moulinec, and P. Suquet. Effective properties of composite materials with periodic microstructure: a computational approach. *Computer Methods in Applied Mechanics and Engineering*, 172(1–4):109–143, 1999.

[3] S. Torquato. Modeling of physical properties of composite materials. *International Journal of Solids and Structures*, 37(1-–2):411–422, 2000.

[4] V. Kouznetsova, W. A. M. Brekelmans, and F. P. T. Baaijens. An approach to micro-macro modeling of heterogeneous materials. *Computational Mechanics*, 27(1):37–48, 2001.

[5] F. Ballani, D. J. Daley, and D. Stoyan. Modelling the microstructure of concrete with spherical grains. *Computational Materials Science*, 35(4):399–407, 2006.

[6] J. Segurado and J. LLorca. Computational micromechanics of composites: The effect of particle spatial distribution. *Mechanics of Materials*, 38(8–10):873–883, 2006.

[7] S. Ravi Annapragada, D. Sun, and S. V. Garimella. Prediction of the effective thermo-mechanical properties of particulate composites. *Computational Materials Science*, 40(2):255–266, 2007.

[8] S. Torquato. Optimal design of heterogeneous materials. *Annual Review of Materials Research*, 40(1):101–129, 2010.

[9] L. Massa, T. L. Jackson, J. Buckmaster, and M. Campbell. Three-dimensional heterogeneous propellant combustion. *Proceedings of the Combustion Institute*, 29(2):2975–2983, 2002.

[10] L. Massa, T. L. Jackson, and J. Buckmaster. New kinetics for a model of heterogeneous propellant combustion. *Journal of Propulsion and Power*, 21(5):914–924, 2005.

[11] T. L. Jackson. Modeling of heterogeneous propellant combustion: A survey. *AIAA Journal*, 50(5):993–1006, May 2012.

[12] X. Garcia, L. T. Akanji, M. J. Blunt, S. K. Matthai, and J. P. Latham. Numerical study of the effects of particle shape and polydispersity on permeability. *Physical Review E*, 80:021304, August 2009.

[13] R. C. Hidalgo, I. Zuriguel, D. Maza, and I. Pagonabarraga. Role of particle shape on the stress propagation in granular packings. *Physical Review Letters*, 103:118001, Sep 2009.

[14] T. L. Jackson, F. Najjar, and J. Buckmaster. New aluminum agglomeration models and their use in solid-propellant-rocket simularions. *Journal of Propulsion and Power*, 21(5):925–936, 2005.

[15] Y. Yavor, A. Gany, and M. W. Beckstead. Modeling of the agglomeration phenomena in combustion of aluminized composite solid propellant. *Propellants, Explosives, Pyrotechnics*, 2013.

[16] M. D. Jackson, S. Ri. Chae, S. R. Mulcahy, C. Meral, R. Taylor, P. Li, A. M. Emwas, J. Moon, S. Yoon, G. Vola, et al. Unlocking the secrets of al-tobermorite in roman seawater concrete. *American Mineralogist*, 98(10):1669–1687, 2013.

[17] N. J Delatte. Lessons from roman cement and concrete. *Journal of Professional Issues in Engineering Education and Practice*, 127(3):109–115, 2001.

[18] H. N Lechtman and L. W. Hobbs. Roman concrete and the roman architectural revolution. In *High-Technology Ceramics: Past, Present, and Future-The Nature of Innovation and Change in Ceramic Technology*, volume 3, pages 81–128. The American Ceramic Society, Inc., 1986.

[19] D. Hill. Ancient roman concrete could hold secrets to improving modern construction. *Civil Engineering—ASCE*, 83(9):44–45, 2013.

[20] S.-D. Poisson. Mémoire sur la théorie du magnétisme. *Mémoires de l'Academie Royale de France*, V:247–338, 1824.

[21] S.-D. Poisson. Second mémoire sur la théorie du magnétisme. *Mémoires de l'Academie Royale de France*, V:488–553, 1824.

[22] M. Faraday. Experimental researches in electricity.–eleventh series. *Philosophical Transactions of the Royal Society of London*, 128:1–40, 1838.

[23] J. C. Maxwell. *A treatise on electricity and magnetism*, volume 1. Clarendon press, 1881.

[24] M. Faraday. Historical sketch of electro-magnetism. *Annals of Philosophy*, 2:195–200, 1821.

[25] A. Einstein. Eine neue bestimmung der moleküldimensionen. *Ann. Physik*, 19:289–306, 1905.

[26] J. D. Bernal. Liquids: Structure, properties, solid interactions. *Elsevier, Amsterdam*, page 25, 1965.

[27] T. C. Hales. An overview of the kepler conjecture. *arXiv:math/9811071 [math.MG]*, 1998.

[28] C. F. Gauss. Besprechung des buchs von l. a. seeber: Untersuchungen über die eigenschaften der positiven ternären quadratischen formen. *Göttingsche Gelehrte Anzeigen*, page 1065, July 1831. see also J. Reine Angew. Math. 20, 312–320 1840.

[29] T. C. Hales. A proof of the kepler conjecture. *Annals of Mathematics*, 162(3):1065–1185, 2005.

[30] J. D. Bernal. Geometry of the structure of monatomic liquids. *Nature*, 185:68–70, 1960.

[31] L. V. Woodcock and C. A. Angell. Diffusivity of the hard-sphere model in the region of fluid metastability. *Physical Review Letters*, 47:1129–1132, October 1981.

[32] N. Xu, J. Blawzdziewicz, and C. S. O'Hern. Random close packing revisited: Ways to pack frictionless disks. *Physical Review E*, 71:061306, Jun 2005.

[33] F. H. Stillinger, E. A. DiMarzio, and R. L. Kornegay. Systematic approach to explanation of the rigid disk phase transition. *Journal of Chemical Physics*, 40(6):1564–1576, 1964.

[34] R. J. Speedy. On the reproducibility of glasses. *Journal of Chemical Physics*, 100(9):6684–6691, 1994.

[35] W. B. Russel, D. A. Saville, and W. R. Schowalter. *Colloidal dispersions*. Cambridge University Press, 1992.

[36] C. W. Hong. New concept for simulating particle packing in colloidal forming processes. *Journal of the American Ceramic Society*, 80(10):2517–2524, 1997.

[37] T. Pöschel and T. Schwager. *Computational granular dynamics: models and algorithms*. Springer, 2005.

[38] S. Luding. *The Physics of Granular Media*. Wiley-VCH Verlag GmbH & Co. KGaA, 2005.

[39] S. Torquato. *Random Heterogeneous Materials*. Springer-Verlag, 2001.

[40] F. Maggi, S. Stafford, T. L. Jackson, and J. Buckmaster. Nature of packs used in propellant modeling. *Physical Review E*, 77:046107, Apr 2008.

[41] S. Torquato. Statistical description of microstructures. *Annual Reviews of Material Research*, 32:77–111, 2002.

[42] Azeddine Benabbou, Houman Borouchaki, Patrick Laug, and Jian Lu. Numerical modeling of nanostructured materials. *Finite Elements in Analysis and Design*, 46(1-2):165–180, 2010.

[43] D. Grolimund, M. Elimelech, M. Borkovec, K. Barmettler, R. Kretzschmar, and H. Sticher. Transport of in situ mobilized colloidal particles in packed soil columns. *Environmental Science and Technology*, 32(22):3562–3569, 1998.

[44] H. S. Choi, H. C. Park, C. Huh, and S. Kang. Numerical simulation of fluid flow and heat transfer of supercritical $CO_2$ in micro-porous media. *Energy Procedia*, 4:3786–3793, 2011.

[45] Y. Matsumura and T. L. Jackson. Numerical simulation of fluid flow through random packs of cylinders using immersed boundary method. *Physics of Fluids*, 26(4):043602, 2014.

[46] A. Donev, I. Cisse, D. Sachs, E. A. Variano, F. H. Stillinger, R. Connelly, S. Torquato, and P. M. Chaikin. Improving the density of jammed disordered packings using ellipsoids. *Science*, 303:990–993, February 2004.

[47] A. Donev, R. Connelly, F. H. Stillinger, and S. Torquato. Underconstrained jammed packings of nonspherical hard particles: Ellipses and ellipsoids. *Physical Review E*, 75:051304, May 2007.

[48] G. W. Delaney and P. W. Cleary. The packing properties of superellipsoids. *Europhysics Letters*, 89:34002, 2010.

[49] M. Mailman, C. F. Schreck, C. S. O'Hern, and B. Chakraborty. Jamming in systems composed of frictionless ellipse-shaped particles. *Physical Review Letters*, 102:255501, Jun 2009.

[50] A. Haji-Akbari, M. Engel, A. S. Keys, X. Zheng, R. G. Petschek, P. Palffy-Muhoray, and S. C. Glotzer. Disordered, quasicrystalline and crystalline phases of densely packed tetrahedra. *Nature*, 462:773–777, August 2009.

[51] S. Torquato and Y. Jiao. Dense packings of polyhedra: Platonic and archimedean solids. *Physical Review E*, 80:041104, Oct 2009.

[52] S. Torquato and Y. Jiao. Dense packings of the platonic and archimedean solids. *Nature*, 460:876–879, August 2009.

[53] D. S. Stafford and T. L. Jackson. Using level sets for creating virtual random packs of non-spherical convex shapes. *Journal of Computational Physics*, 229(9):3295–3315, 2010.

[54] Y. Jiao and S. Torquato. Maximally random jammed packings of platonic solids: Hyperuniform long-range correlations and isostaticity. *Physical Review E*, 84:041309, October 2011.

[55] K. C. Smith, M. Alam, and T. S. Fisher. Athermal jamming of soft frictionless platonic solids. *Physical Review E*, 82:051304, Nov 2010.

[56] S. R. Buechler and S. M. Johnson. Efficient generation of densely packed convex polyhedra for 3d discrete and finite-discrete element methods. *International Journal for Numerical Methods in Engineering*, 94(1):1–19, 2013.

[57] P. F. Damasceno, M. Engel, and S. C. Glotzer. Predictive self-assembly of polyhedra into complex structures. *Science*, 337(6093):453–457, 2012.

[58] J. Baker and A. Kudrolli. Maximum and minimum stable random packings of platonic solids. *Physical Review E*, 82:061304, Dec 2010.

[59] A. Jaoshvili, A. Esakia, M. Porrati, and P. M. Chaikin. Experiments on the random packing of tetrahedral dice. *Physical Review Letters*, 104:185501, May 2010.

[60] J. Henzie, M. Grünwald, A. Widmer-Cooper, Phillip L. Geissler, and Peidong Yang. Self-assembly of uniform polyhedral silver nanocrystals into densest packings and exotic superlattices. *Nature Materials*, 11:131–137, 2012.

[61] Y. Kallus, V. Elser, and S. Gravel. Dense periodic packings of tetrahedra with small repeating units. *Discrete & Computational Geometry*, 44(2):245–252, 2010.

[62] S. Torquato and Y. Jiao. Exact constructions of a family of dense periodic packings of tetrahedra. *Physical Review E*, 81:041310, Apr 2010.

[63] S. Torquato and Y. Jiao. Analytical constructions of a family of dense tetrahedron packings and the role of symmetry. *arXiv:0912.4210 [cond-mat.stat-mech]*, January 2010.

[64] E. R. Chen, M. Engel, and S. C. Glotzer. Dense crystalline dimer packings of regular tetrahedra. *arXiv:1001.0586 [cond-mat.stat-mech]*, July 2010.

[65] W. S. Jodrey and E. M. Tory. Random sequential packing in $R^n$. *Journal of Statistical Computation and Simulation*, 10:87–93, 1980.

[66] D. W. Cooper. Random-sequential-packing simulations in three dimensions for spheres. *Physical Review A*, 38:522–524, Jul 1988.

[67] J. D. Sherwood. Packing of spheroids in three-dimensional space by random sequential addition. *Journal of Physics A: Mathematical and General*, 30(24):L839, 1997.

[68] G. Zhang and S. Torquato. Precise algorithm to generate random sequential addition of hard hyperspheres at saturation. *Physical Review E*, 88:053312, Nov 2013.

[69] C. H. Bennet. Serially deposited amorphous aggregates of hard spheres. *Journal of Applied Physics*, 43:2727, 1972.

[70] G. T. Nolan and P. E. Kavanagh. Computer simulation of random packing of hard spheres. *Powder Technology*, 72:149–155, 1992.

[71] D. Coelho, J.-F. Thovert, and P. M. Adler. Geometrical and transport properties of random packings of spheres and aspherical particles. *Physical Review E*, 55:1959–1978, Feb 1997.

[72] S. Heitkam, W. Drenckhan, and J. Fröhlich. Packing spheres tightly: Influence of mechanical stability on close-packed sphere structures. *Physical Review Letters*, 108:148302, April 2012.

[73] A. Z. Zinchenko. Algorithm for random close packing of spheres with periodic boundary conditions. *Journal of Computational Physics*, 114:298–307, 1994.

[74] W. S. Jodrey and E. M. Tory. Computer simulation of close random packing of equal spheres. *Physical Review A*, 32:2347–2351, 1985.

[75] S. R. Williams and A. P. Philipse. Random packings of spheres and spherocylinders simulated by mechanical contraction. *Physical Review E*, 67:051301, May 2003.

[76] A. S. Clarke and J. D. Wiley. Numerical simulation of a dense random packing of a binary mixture of hard spheres. *Physical Review B*, 35:7350–7356, 1987.

[77] M. Bargiel and J. Moscinski. C-language program for the irregular close packing of hard spheres. *Computer Physics Communications*, 64:183–192, 1991.

[78] A. Bezrukov, M. Bargiel, and D. Stoyan. Statistical analysis of simulated random packings of spheres. *Particle & Particle Systems Characterization*, 19:111–118, 2002.

[79] H. Hermann, A. Elsner, and D. Stoyan. Surface area and volume fraction of random open-pore systems. *Modelling and Simulation in Materials Science and Engineering*, 21:085005, 2013.

[80] S. Torquato and Y. Jiao. Robust algorithm to generate a diverse class of dense disordered and ordered sphere packings via linear programming. *Physical Review E*, 82:061302, Dec 2010.

[81] É. Marcotte and S. Torquato. Efficient linear programming algorithm to generate the densest lattice sphere packings. *Physical Review E*, 87:063303, Jun 2013.

[82] B. D. Lubachevsky and F. H. Stillinger. Geometric properties of random disk packings. *Journal of Statistical Physics*, 60(5-6):561–583, 1990.

[83] B. D. Lubachevsky, F. H. Stillinger, and E. N. Pinson. Disks vs. spheres: Contrasting properties of random packings. *Journal of Statistical Physics*, 64:501–525, 1991.

[84] B. D. Lubachevsky. How to simulate billiards and similar systems. *Journal of Computational Physics*, 94(2):255–283, 1991.

[85] A. R. Kansal, S. Torquato, and F. H. Stillinger. Computer generation of dense polydisperse sphere packings. *Journal of Chemical Physics*, 117(18):8212–8218, November 2002.

[86] 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*, 202(2):737–764, 2005.

[87] A. Donev, S. Torquato, and F. H. Stillinger. Neighbor list collision-driven molecular dynamics simulation for nonspherical hard particles. II. Applications to ellipses and ellipsoids. *Journal of Computational Physics*, 202(2):765–793, 2005.

[88] A. Donev, S. Torquato, and F. H. Stillinger. Pair correlation function characteristics of nearly jammed disordered and ordered hard-sphere packings. *Physical Review E*, 71:011105, Jan 2005.

[89] E. Azéma, F. Radjai, and G. Saussine. Quasistatic rheology, force transmission and fabric properties of a packing of irregular polyhedral particles. *Mechanics of Materials*, 41(6):729–741, 2009.

[90] S. Torquato and F. H. Stillinger. Multiplicity of generation, selection, and classification procedures for jammed hard-particle packings. *The Journal of Physical Chemistry B*, 105(47):11849–11853, 2001.

[91] R. Connelly and W. Whiteley. Second-order rigidity and prestress stability for tensegrity frameworks. *SIAM Journal on Discrete Mathematics*, 9(3):453–491, 1996.

[92] D. Eberly. *Game Physics*. Morgan Kaufmann Publishers, Elsevier, 2003.

[93] B. J. Alder and T. E. Wainwright. Studies in molecular dynamics. I. general method. *Journal of Chemical Physics*, 31:459, 1959.

[94] Jr. Stillinger, F. H. and Z. W. Salsburg. Limiting polytope geometry for rigid rods, disks, and spheres. *Journal of Statistical Physics*, 1(1):179–225, 1969.

[95] S. Torquato, T. M. Truskett, and P. G. Debenedetti. Is random close packing of spheres well defined? *Physical Review E*, 84:2064–2067, Mar 2000.

[96] G. D. Scott and D. M. Kilgour. The density of random close packing of spheres. *Journal of Physics D: Applied Physics*, 2(6):863, 1969.

[97] T. M. Truskett, S. Torquato, and P. G. Debenedetti. Towards a quantification of disorder in materials: Distinguishing equilibrium and glassy sphere packings. *Physical Review E*, 62:993–1001, Jul 2000.

[98] C. Song, P. Wang, and H. A. Makse. A phase diagram for jammed matter. *Nature*, 453:629–632, 2008.

[99] Y. Jin and H. A. Makse. A first-order phase transition defines the random close packing of hard spheres. *Physica A*, 389:5362–5379, August 2010.

[100] C. Briscoe, C. Song, P. Wang, and H. A. Makse. Jamming iii: Characterizing randomness via the entropy of jammed matter. *Physica A: Statistical Mechanics and its Applications*, 389(19):3978–3999, October 2010.

[101] P. Wang, C. Song, Y. Jin, and H. A. Makse. Jamming ii: Edwards' statistical mechanics of random packings of hard spheres. *Physica A: Statistical Mechanics and its Applications*, 390(3):427–455, February 2011.

[102] P. J. Steinhardt, D. R. Nelson, and M. Ronchetti. Bond-orientational order in liquids and glasses. *Physical Review B*, 28:784–805, July 1983.

[103] M. Chen, J. Buckmaster, T. L. Jackson, and L. Massa. Homogenization issues and the combustion of heterogeneous solid propellants. *Proceedings of the Combustion Institute*, 29(2):2923 – 2929, 2002.

[104] S. Torquato, J. D. Beasley, and Y. C. Chiew. Two-point cluster function for continuum percolation. *Journal of Chemical Physics*, 88:6540, 1988.

[105] S. Gallier. Numerical modeling of dielectric breakdown in solid propellant microstructures. *International Journal of Multiscale Computational Engineering*, 8(5), 2010.

[106] B. Lu and S. Torquato. Lineal-path function for random heterogeneous materials. *Physical Review A*, 45:922–929, Jan 1992.

[107] S. Prager. Interphase transfer in stationary two-phase media. *Chemical Engineering Science*, 18(4):227–231, 1963.

[108] H. Goldstein, C. P. Poole Jr., and J. L. Safko. *Classical Mechanics*. Addison-Wesley, 3 edition, 2001.

[109] I. Newton. *PhilosophiæNaturalis Principia Mathematica*. Trinity College, London, 1687.

[110] D. Baraff. *Dynamic Simulation of Non-penetrating Rigid Bodies*. PhD thesis, Cornell University, USA, 1992.

[111] B. V. Mirtich. *Impulse-Based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California at Berkeley, USA, 1996.

[112] G. van den Bergen. *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann Publishers, Elsevier, 2003.

[113] D. Eberly. *3D Game Engine Architecture: Engineering Real-Time Applications with Wild Magic*. Morgan Kaufmann Publishers, Elsevier, 2005.

[114] C. Ericson. *Real-Time Collision Detection*. Morgan Kaufmann Publishers, Elsevier, 2005.

[115] M. G. Coutinho. *Guide to Dynamic Simulations of Rigid Bodies and Particle Systems*. Springer, 2013.

[116] D. S. Stafford. *Random Packs and Their Use in Modeling High-Speed Porous Flow*. PhD thesis, University of Illinois at Urbana-Champaign, USA, 2008.

[117] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *Robotics and Automation, IEEE Journal of*, 4(2):193–203, 1988.

[118] E. G. Gilbert and C.-P. Foo. Computing the distance between general convex objects in three-dimensional space. *Robotics and Automation, IEEE Transactions on*, 6:53–61, February 1990.

[119] X. Zhang, M. Lee, and Y. J. Kim. Interactive continuous collision detection for non-convex polyhedra. *available at: http://graphics.ewha.ac.kr/FAST/*, 2004.

[120] J. A. Carretero and M. A. Nahon. Solving minimum distance problems with convex or concave bodies using combinatorial global optimization algorithms. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 35:1144–1155, December 2005.

[121] X. Chen, J. Yong, G. Zheng, J. C. Paul, and J. Sun. Computing minimum distance between two implicit algebraic surfaces. *Computer-Aided Design*, 38(10):1053–1061, 2006.

[122] Z. Yao, J. S. Wang, G. R. Liu, and M. Cheng. Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method. *Computer Physics Communications*, 161:27–35, April 2004.

[123] G. M. Knott, T. L. Jackson, and J. Buckmaster. Random packing of heterogeneous propellants. *American Institute of Aeronautics and AstronauticsJournal*, 39(4):678–686, 2001.

[124] S. Kochevets, J. Buckmaster, T. L. Jackson, and J. A. Hegab. Random packs and their use in modeling heterogeneous solid propellant combustion. *Journal of Propulsion and Power*, 17(4):883–891, 2001.

[125] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991.

[126] D. Frenkel and J. F. Maguire. Molecular dynamics study of the dynamical properties of an assembly of infinitely thin hard rods. *Molecular Physics*, 49(3):503–541, 1983.

[127] J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, and K. Schulten. Accelerating molecular modeling applications with graphics processors. *Journal of Computational Chemistry*, 28(16):2618–2640, 2007.

[128] S. Miller and S. Luding. Event-driven molecular dynamics in parallel. *Journal of Computational Physics*, 193:306–316, August 2003.

[129] M. Marín and P. Cordero. An empirical assessment of priority queues in event-driven molecular dynamics simulation. *Computer Physics Communications*, 92(2–3):214–224, 1995.

[130] G. Paul. A complexity o(1) priority queue for event driven molecular dynamics simulations. *Journal of Computational Physics*, 221:615–625, 2007.

[131] D. E. Knuth. *The Art of Computer Programming, Volume 3, Sorting and Searching*. Addison-Wesley, 1973.

[132] A. Donev. *Jammed Packings of Hard Particles*. PhD thesis, Princeton University, USA, 2006.

[133] A. Mehta, editor. *Granular Matter*. Springer-Verlag, New York, 1994.

[134] H. A. Makse, D. L. Johnson, and L. M. Schwartz. Packing of compressible granular materials. *Physical Review Letters*, 84:4160–4163, May 2000.

[135] S. Khirevich. *High-Performance Computing of Flow, Diffusion, and Hydrodynamic Dispersion in Random Sphere Packings*. PhD thesis, Philipps-Universität Marburg, Germany, 2010.

[136] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423; 623–656, 1948.

[137] J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices, and Groups*. Springer-Verlag, New York, 1999.

[138] C. Zong. *Sphere Packings*. Springer-Verlag, New York, 1999.

[139] T. Aste and D. Weaire. *The Pursuit of Perfect Packing*. IOP Publishing, 2000.

[140] L. E. Silbert, D. Ertaş, G. S. Grest, T. C. Halsey, and D. Levine. Geometry of frictionless and frictional sphere packings. *Physical Review E*, 65:031304, Feb 2002.

[141] T. Aste, M. Saadatfar, and T. J. Senden. Geometrical structure of disordered sphere packings. *Physical Review E*, 71:061302, Jun 2005.

[142] L. E. Silbert, D. Ertaş, G. S. Grest, T. C. Halsey, and D. Levine. Analogies between granular jamming and the liquid-glass transition. *Physical Review E*, 65:051307, May 2002.

[143] F. H. Stillinger, H. Sakai, and S. Torquato. Lattice-based random jammed configurations for hard particles. *Physical Review E*, 67:031107, Mar 2003.

[144] K. W. Desmond and E. R. Weeks. Random close packing of disks and spheres in confined geometries. *Physical Review E*, 80:051305, Nov 2009.

[145] P. Chaudhuri, L. Berthier, and S. Sastry. Jamming transitions in amorphous packings of frictionless spheres occur over a continuous range of volume fractions. *Physical Review Letters*, 104:165701, Apr 2010.

[146] M. Jerkins, M. Schröter, H. L. Swinney, T. J. Senden, M. Saadatfar, and T. Aste. Onset of mechanical stability in random packings of frictional spheres. *Physical Review Letters*, 101:018301, Jul 2008.

[147] S. Torquato and F. H. Stillinger. Jammed hard-particle packings: From kepler to bernal and beyond. *Review of Modern Physics*, 82:2633–2672, September 2010.

[148] S. Alexander. Amorphous solids: their structure, lattice dynamics and elasticity. *Physics Reports*, 296(2–4):65–236, 1998.

[149] C. F. Moukarzel. Isostatic phase transition and instability in stiff granular materials. *Physical Review Letters*, 81:1634–1637, Aug 1998.

[150] S. F. Edwards and D. V. Grinev. Statistical mechanics of stress transmission in disordered granular arrays. *Physical Review Letters*, 82:5397–5400, Jun 1999.

[151] K. A. Dawson. The glass paradigm for colloidal glasses, gels, and other arrested states driven by attractive interactions. *Current Opinion in Colloid & Interface Science*, 7(3–4):218 – 227, 2002.

[152] T. S. Hudson and P. Harrowell. Dense packings of hard spheres of different sizes based on filling interstices in uniform three-dimensional tilings. *The Journal of Physical Chemistry B*, 112(27):8139–8143, 2008.

[153] A. B. Hopkins, F. H. Stillinger, and S. Torquato. Densest binary sphere packings. *Physical Review E*, 85:021130, 2012.

[154] S. I. Henderson, T. C. Mortensen, S. M. Underwood, and W. van Megen. Effect of particle size distribution on crystallisation and the glass transition of hard sphere colloids. *Physica A: Statistical Mechanics and its Applications*, 233(1–2):102 – 116, 1996.

[155] A. B. Hopkins, F. H. Stillinger, and S. Torquato. Disordered strictly jammed binary sphere packings attain an anomalously large range of densities. *Physical Review E*, 88:022205, Aug 2013.

[156] V. Baranau and U. Tallarek. Random-close packing limits for monodisperse and polydisperse hard spheres. *Soft Matter*, 10:3826, February 2014.

[157] R. S. Farr and R. D. Groot. Close packing density of polydisperse hard spheres. *arXiv:0912.0852 [cond-mat.soft]*, December 2009.

[158] C. De Michele. Simulating hard rigid bodies. *Journal of Computational Physics*, 229(9):3276–3294, 2010.

[159] S. Mukhopadhyay and J. Peixinho. Packings of deformable spheres. *Physical Review E*, 84:011302, Jul 2011.

[160] U. Betke and M. Henk. Densest lattice packings of 3-polytopes. *Computational Geometry*, 16(3):157 – 186, 2000.

[161] J. H. Conway and S. Torquato. Packing, tiling, and covering with tetrahedra. *Proceedings of the National Academy of Sciences*, 103(28):10612–10617, 2006.

[162] T. L. Jackson, D. E. Hooks, and J. Buckmaster. Modeling the microstructure of energetic materials with realistic constituent morphology. *Propellants, Explosives, Pyrotechnics*, 36:252–258, 2011.

[163] J. H. Conway, Y. Jiao, and S. Torquato. New family of tilings of three-dimensional euclidean space by tetrahedra and octahedra. *Proceedings of the National Academy of Sciences*, 108(27):11009–11012, 2011.

[164] S. Torquato and Y. Jiao. Organizing principles for dense packings of nonspherical hard particles: Not all shapes are created equal. *Physical Review E*, 86:011102, Jul 2012.

[165] E. R. Chen, D. Klotsa, M. Engel, P. F. Damasceno, and S. C. Glotzer. Complexity in surfaces of densest packings for families of polyhedra. *Phys. Rev. X*, 4:011024, Feb 2014.

[166] M. Atiyah and P. Sutcliffe. Polyhedra in physics, chemistry and geometry. *Milan Journal of Mathematics*, 71(1):33–58, 2003.

[167] L. Schläfli. "Theorie der vielfachen Kontinuität". Unpublished manuscript. 1852. *Published in Denkschriften der Schweizerischen naturforschenden Gessel*, 38:1–237, 1901.

[168] The archimedes palimpsest project. `http://archimedespalimpsest.org`.

[169] J. H. Conway, H. Burgiel, and C. Goodman-Strauss. *The Symmetries of Things*. A K Peters, 2008.

[170] E. R. Chen. A dense packing of regular tetrahedra. *Discrete Comput Geom*, 40(2):214–240, 2008.

[171] S. Torquato. Random heterogeneous media: Microstructure and improved bounds on effective properties. *Applied Mechanics Reviews*, 44:37–76, February 1991.

[172] Visualization Sciences Group. Amira 3d analysis software for life sciences. `http://www.vsg3d.com/amira`, 2014.

[173] H. Lee, M. Brandyberry, A. Tudor, and K. Matouš. Three-dimensional reconstruction of statistically optimal unit cells of polydisperse particulate composites from microtomography. *Physical Review E*, 80:061301, Dec 2009.