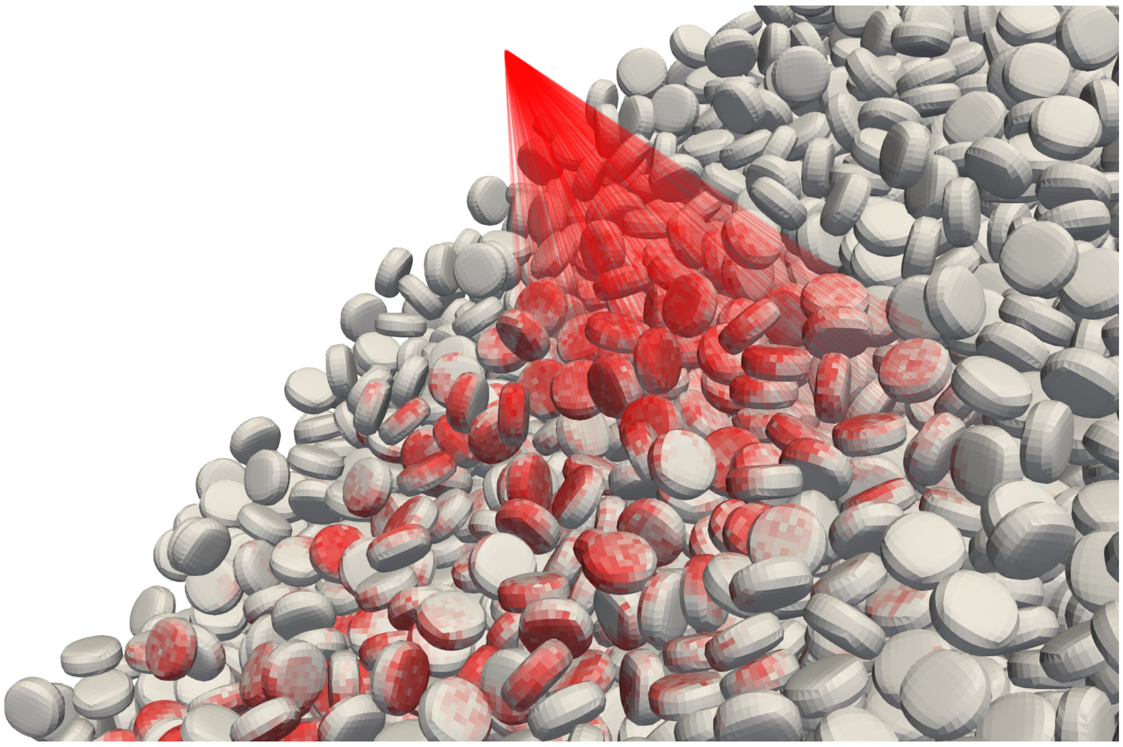
CHALMERS
UNIVERSITY OF TECHNOLOGY



Ray Casting DEM Simulation Data to Characterize the Spray Coating of Superquadric Particles

Master's thesis in Engineering Mathematics and Computational Science

PONTUS ANDERSSON

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:59115

Ray Casting DEM Simulation Data to Characterize the Spray Coating of Superquadric Particles

PONTUS ANDERSSON

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2020

Ray Casting DEM Simulation Data to Characterize the Spray Coating of Superquadric Particles
PONTUS ANDERSSON

Supervisors: Luis Martin de Juan & Johan Remmelglas, Modelling and Simulation, Global Medicines Development, Pharmaceutical Technology & Development, Operations, AstraZeneca, Gothenburg, Sweden
Supervisor and Examiner: Alexey Geynts, Department of Mathematical Sciences, Chalmers University of Technology, Gothenburg, Sweden

Master's Thesis 2020:59115
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Modelling and Simulation, Global Medicines Development, Pharmaceutical Technology & Development, Operations, AstraZeneca, Gothenburg, Sweden

Cover: Rays simulating the spray in a coating process.

Typeset in $\LaTeX$
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Ray Casting DEM Simulation Data to Characterize the Spray Coating of Superquadric Particles
PONTUS ANDERSSON
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Drum spray coating of tablets is a common unit operation in the pharmaceutical production process. Increasingly, the Discrete Element Method (DEM) is being used to study and optimize this process in order to improve the resulting tablet coating uniformity. Multi-spheres are a popular approach to shape approximation in DEM, but have been shown to be insufficient at modeling the bulk kinematics of certain tablet shapes. This work develops a method that enables the use of superquadrics in DEM to study inter- and intra-tablet coating variability. Specifically, a standalone C++ program is developed that models the spray by casting rays on data exported from DEM simulations. Acceleration techniques like spatial subdivision and bounding volumes are used to reduce the runtime, and an efficient intra-particle representation is developed and used to compute the intra-tablet coating variability. The final implementation is capable of processing a 180 second simulation containing 2 million tablets in just a few hours on a conventional desktop computer. Key parameters of the implementation are studied through benchmarks and suggestions of suitable values are made. The tool is finally used to show that the blockiness of the DEM shape representation has a significant impact on intra-tablet coating variability, but further studies are needed to better understand this effect.

Keywords: Coating variability, DEM, Superquadric, Shape approximation, Ray casting, Acceleration techniques

Acknowledgements

This work would not have been possible without the support of some key individuals. I'd like to thank my supervisors Alexey, Luis and Johan for their guidance and many insightful discussions over the course of the project. Many thanks also go out to members of my family who served as enxhaustible sources of encouragement.

This project has been funded by AstraZeneca.

Pontus Andersson, Gothenburg, 2020

Contents

List of Figures

List of Tables

1 Introduction

Drum spray coating of tablets is a critical part of the pharmaceutical production process, particularly for functional coatings whose uniformity impacts stability and drug release. In recent years, improvements in computer performance have enabled successful use of the Discrete Element Method (DEM) to simulate the interactions between all individual tablets in the coating process [2, 3]. DEM simulations provide rich information that can be used to tune process parameters.

Spheres are commonly used to represent particle shapes in DEM simulations because of their rotational invariance and simple collision detection. However, few tablets are spherical and several different methods have been developed to model non-spherical shapes. Below are four popular approaches listed roughly in order of increasing accuracy and computational complexity [4]:

1. **Rolling friction models**: Typically uses spheres to represent the particles but adjusts the coefficient of rolling friction to approximate non-spherical behavior.
2. **Multi-sphere**: Approximates shapes by gluing multiple spheres together into a single rigid shape. Retains the computational advantages of spheres, but may require a large number of spheres to get sufficiently good approximations, which increases computational cost.
3. **Superquadrics**: A compact parametric model that extends ellipsoids and is capable of modeling certain symmetrical shapes like cylinders and cuboids. The disadvantage is that the contact models become complicated and typically require Newton iterations to solve.
4. **Polyhedra**: Uses connected vertices to form polygonal faces and can thereby approximate any shape arbitrarily well. However, a lot of vertices are required to approximate smooth curves which increases computational cost.

The hypothesis underlying this work is that accurate shape representation in DEM plays a major role in making accurate coating predictions, particularly for the intra-tablet coating variability. This is supported by findings that shape representation has a significant impact on particle packing [5], as well as on the asymptotic intra-tablet coating behavior [6]. Superquadrics can approximate certain symmetrical tablet shapes very well and have been shown to keep computational costs down compared to equivalent multi-sphere approximations [7]. Recently, an implementation of superquadrics was added to the open source DEM software LIGGGHTS [8, 9].

These factors make superquadrics an interesting option for studying the impact of tablet shape approximation on coating outcome.

There are two fundamentally different ways of extracting spray coating information from DEM simulations. The first is to do it during runtime, that is to simulate the spray simultaneously with everything else. This could for instance be done by simulating the spray droplets in a coupled CFD-DEM fashion, which enables modeling effects like capillary action and friction changes between sprayed tablets [10]. The second approach is post-processing, which means to simulate the tablets without any spraying, export the tablet positions and orientations and then apply the spray on the exported data. One such post-processing approach is to model the spray as a cone of rays, or half lines in space, and to intersect those rays with the tablets to compute which particles were hit by the spray at each time step. This approach makes a lot of simplifying assumptions, but it has the benefit of letting the user try multiple different spray patterns and positions with the data from a single simulation, which can save a lot of time for large simulations.

1.1 Previous Work

Toschkoff developed three different methods of extracting inter-tablet coating information from DEM simulations of spherical particles [11]. The first approach defined the spray zone as a bounding volume. The top layer of tablets was identified and coating mass was applied proportional to the time spent in the spray zone. The second approach simulated the spray droplets as spheres alongside the tablets, applying coating mass whenever a droplet hit a tablet. Both of these methods ran during the simulation. The third approach instead post-processed the simulation data by modeling the spray droplets as rays and intersecting them with the tablets. They found that all methods gave agreeing results.

Freireich studied intra-tablet variability both theoretically and using DEM simulations of multi-sphere particles [6, 12]. In one approach they tessellated the particle surfaces and used a GPU to render images of the simulation data. Each triangle, or panel, was assigned a unique color, and coating mass was added to each in panel in proportion to the number of corresponding pixels. They found that the shadowing effects from neighboring tablets had a significant impact on the intra-tablet coating outcome, something that Toschkoff's spray zone approach does not capture. This indicates the importance of modeling spray droplet trajectories when estimating intra-tablet coating variability.

Pei [13] followed up on the work of Freireich by studying the cap-to-band coating ratio for a number of different multi-sphere shapes using both the render method and a ray caster. They found that the cap-to-band coating ratio decreased with a smaller horizontal spray angle, as well as a significant difference in coating outcome between the cylindrical spray model of the render method and the conical spray model of the ray caster.

1.2 Aims and Goals

Previous work has been mostly focused on multi-sphere shape representations and small lab coaters. Moving to superquadrics and increasing the simulation size will increase computational complexity and necessitate an efficient computational approach.

The aim of this work is to develop a method to characterize inter- and intra-tablet coating varability when the tablets are modelled using superquadrics. Software should be developed that can process large amounts of DEM simulation data to generate coating predictions. It is critical that the developed implementation has a reasonably short runtime, as this enables researchers to more quickly iterate on ideas. In line with this aim, the goals of this thesis project are to:

1. Develop a ray caster for superquadric particles that performs well when applied to drum coating simulations
2. Develop a method to compute and visualize intra-tablet coating variability when tablets are modeled using superquadrics
3. Validate and benchmark the ray caster, and provide a description of what can be expected in different scenarios using different settings
4. Use the developed methods to assess the impact of shape representation on the coating outcome

2 Theory

2.1 Superquadrics

A superellipse can be defined either implicitly as

$$\left(\frac{x}{a}\right)^{2/\epsilon} + \left(\frac{y}{b}\right)^{2/\epsilon} = 1 \tag{2.1}$$

or explicitly as

$$\mathbf{x}(\theta) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a\cos^{\epsilon}(\theta) \\ b\sin^{\epsilon}(\theta) \end{bmatrix}, \ -\pi \leq \theta \leq \pi. \tag{2.2}$$

The scale parameters a and b determine the width and height of the ellipse, and the shape parameter ϵ alters its shape. A value of $\epsilon = 1$ gives an ellipse, $\epsilon \to 0$ gives a more square shape and $\epsilon \to \infty$ a more concave shape. Figure 2.1 shows examples of superellipses for different values of ϵ. Note that the powers in (2.2) actually represent signed power functions

$$x^p = \operatorname{sgn}(x)\,|x|^p = \begin{cases} |x|^p & x \geq 0 \\ -|x|^p & x < 0. \end{cases} \tag{2.3}$$

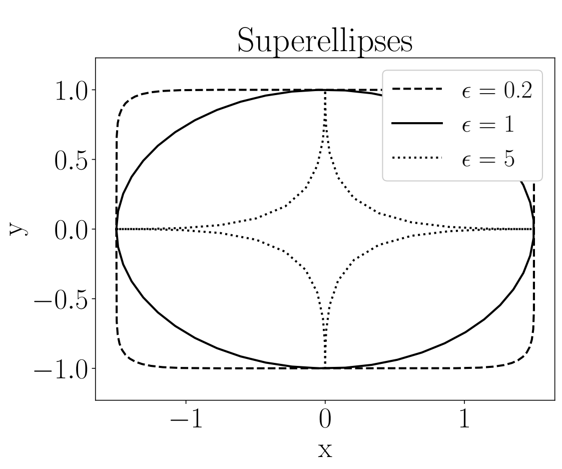


Figure 2.1: Three superellipses for different values of ϵ with $a = 1.5$ and $b = 1$.

The signed power function is necessary to avoid potentially complex results that come from exponentiation with a negative base. All powers in superellipse expressions that follow should be read as signed power functions.

Barr [14] extended the geometric family of quadric surfaces, which includes shapes like ellipsoids and hyperbolic sheets, by constructing superquadrics as spherical products of superellipses. The spherical product of two parametric curves $\mathbf{m}(\eta)$ and $\mathbf{h}(\omega)$ in $\mathbb{R}^2$ is given by

$$\mathbf{r}(\eta,\omega) = \mathbf{m}(\eta) \otimes \mathbf{h}(\omega) = \begin{bmatrix} m_1(\eta)h_1(\omega) \\ m_1(\eta)h_2(\omega) \\ m_2(\eta) \end{bmatrix}, \quad \begin{matrix} \omega_0 \leq \omega \leq \omega_1 \\ \eta_0 \leq \eta \leq \eta_1 \end{matrix} . \tag{2.4}$$

When the two parametric curves are superellipses, i.e.

$$\mathbf{m}(\eta) = \begin{bmatrix} \cos^{\epsilon_1}(\eta) \\ c\sin^{\epsilon_1}(\eta) \end{bmatrix}, \quad \mathbf{h}(\omega) = \begin{bmatrix} a\cos^{\epsilon_2}(\omega) \\ b\sin^{\epsilon_2}(\omega) \end{bmatrix} \tag{2.5}$$

their spherical product gives the explicit form of the superquadric

$$\mathbf{x}(\eta,\omega) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{m}(\eta) \otimes \mathbf{h}(\omega) = \begin{bmatrix} a\cos^{\epsilon_1}(\eta)\cos^{\epsilon_2}(\omega) \\ b\cos^{\epsilon_1}(\eta)\sin^{\epsilon_2}(\omega) \\ c\sin^{\epsilon_1}(\eta) \end{bmatrix}. \tag{2.6}$$

This explicit form can be transformed into a corresponding implicit form [15]. First, rearrange the three equations in (2.6) and square them to get

$$\left(\frac{x}{a}\right)^2 = \cos^{2\epsilon_1}(\eta)\cos^{2\epsilon_2}(\omega) \tag{2.7}$$

$$\left(\frac{y}{b}\right)^2 = \cos^{2\epsilon_1}(\eta)\sin^{2\epsilon_2}(\omega) \tag{2.8}$$

$$\left(\frac{z}{c}\right)^2 = c\sin^{2\epsilon_1}(\eta) \tag{2.9}$$

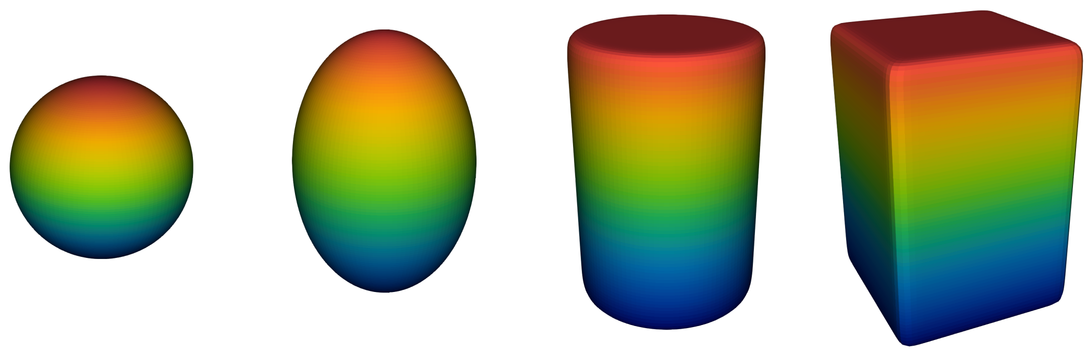
Adding (2.7) and (2.8) together, raising both sides to $1/\epsilon_2$ and cancelling the resulting $\cos^2(\omega) + \sin^2(\omega) = 1$ gives

$$\left(\frac{x}{a}\right)^{2/\epsilon_2} + \left(\frac{x}{a}\right)^{2/\epsilon_2} = \cos^{2\epsilon_1/\epsilon_2}(\eta). \tag{2.10}$$

In a similar fashion, raising (2.10) to ϵ_2/ϵ_1 and (2.9) to $1/\epsilon_1$ respectively and adding them together gives

$$f(x,y,z) := \left(\left|\frac{x}{a}\right|^{n_2} + \left|\frac{y}{b}\right|^{n_2}\right)^{n_1/n_2} + \left|\frac{z}{c}\right|^{n_1} - 1 = 0, \;\; n_1, n_2 \in (0,\infty]. \tag{2.11}$$

where we have introduced the blockiness parameters $n_1 = 2/\epsilon_1$ and $n_2 = 2/\epsilon_2$. The equation in (2.11) is the implicit surface equation of the superquadric. The function



(a) Sphere. **(b)** Ellipsoid. **(c)** Cylinder. **(d)** Cuboid.

Figure 2.2: Examples of superquadrics. The sphere in (a) has parameters $a = b = c = 1$ and $n_1 = n_2 = 2$. The ellipsoid in (b) has the same settings as the sphere but adds $c = 1.5$. The cylinder in (c) adds $n_1 = 20$ and the cuboid in (d) adds $n_2 = 20$.

f is the so-called inside-outside function, because all points $\mathbf{p}$ satisfying $f(\mathbf{p}) < 0$ are inside the superquadric, and those satisfying $f(\mathbf{p}) > 0$ are outside. Examples of superquadrics can be seen in Figure 2.2. The gradient of f will be of later use, and it is

$$\nabla f = \begin{bmatrix} \frac{n_1}{a}\left(\left|\frac{x}{a}\right|^{n_2} + \left|\frac{y}{b}\right|^{n_2}\right)^{n_1/n_2-1} \operatorname{sgn}(x)\left|\frac{x}{a}\right|^{n_2-1} \\ \frac{n_1}{b}\left(\left|\frac{x}{a}\right|^{n_2} + \left|\frac{y}{b}\right|^{n_2}\right)^{n_1/n_2-1} \operatorname{sgn}(y)\left|\frac{y}{b}\right|^{n_2-1} \\ \frac{n_1}{c}\operatorname{sgn}(z)\left|\frac{z}{c}\right|^{n_1-1} \end{bmatrix}. \tag{2.12}$$

2.2 Ray Casting

The fundamental ray casting problem is that of intersecting a ray with a collection of geometrical objects distributed in 3D space. The collection of objects is commonly referred to as a scene, and examples of common objects are spheres, planes, triangles and cuboids. A ray is defined as the half-line

$$\mathbf{r}(t) = \mathbf{O} + \mathbf{D}t,\ t > 0 \tag{2.13}$$

where $\mathbf{O}$ is the origin and $\mathbf{D}$ the normalized direction vector. A lot of the early work on ray casting was done in the computer graphics field of rendering [16]. There the idea is to let rays represent photons and to trace them as they bounce around the scene hitting the objects and reflecting, refracting and getting absorbed. It's a simple model that can yield very realistic results, but it also requires tracing a lot of rays which quickly drives up the computational cost. Acceleration techniques involving bounding volumes and spatial subdivision have been developed to speed up ray tracing. Two of the main ways are to either reduce the cost of individual ray-object intersection checks or to reduce the total amount of such checks performed. The acceleration techniques considered in this work will be presented in the implementation section.

2.2.1 Ray-superquadric intersection

Ray casting can be done on many different kinds of geometric objects, but implicit surfaces like the one in (2.1) make for a particularly clean problem formulation. Given the inside-outside equation in (2.1), where the points $\mathbf{p}$ on the surface are those satisfying $f(\mathbf{p}) = 0$, the ray casting problem becomes that of finding the roots to the univariate function

$$F(t) = f(\mathbf{r}(t)). \tag{2.14}$$

This transforms the geometric problem of ray-superquadric intersection into the analytical problem of finding the roots of $F(t)$. In the case of a convex superquadric with $\epsilon_1, \epsilon_2 <= 2$ this function turns out to be convex [17]. This can be shown by setting $a = b = c = 1$, $x, y, z >= 0$ in (2.11) which gives

$$F(t) = f(\mathbf{r}(t)) = \left((O_1 + D_1 t)^{2/\epsilon_2} + (O_2 + D_2 t)^{2/\epsilon_2}\right)^{\epsilon_2/\epsilon_1} + (O_3 + D_3 t)^{2/\epsilon_1} - 1.$$

Differentiating F with respect to t gives

$$\frac{dF}{dt} = \frac{2}{\epsilon_1}\left(x^{2/\epsilon_2} + y^{2/\epsilon_2}\right)^{\epsilon_2/\epsilon_1 - 1}\left(D_1 x^{2/\epsilon_2 - 1} + D_2 y^{2/\epsilon_2 - 1}\right) + \frac{2D_3}{\epsilon_1} z^{2/\epsilon_1 - 1} \tag{2.15}$$

and

$$\begin{aligned}\frac{d^2F}{dt^2} =& \frac{2(2-\epsilon_1)}{\epsilon_1^2}\left(x^{2/\epsilon_2} + y^{2/\epsilon_2}\right)^{\epsilon_2/\epsilon_1 - 1}\left(D_1 x^{2/\epsilon_2 - 1} + D_2 y^{2/\epsilon_2 - 1}\right)^2 + \\ & \frac{2D_3^2(2-\epsilon_1)}{\epsilon_1^2} z^{2(1/\epsilon_1 - 1)} + \\ & \frac{2(2-\epsilon_2)}{\epsilon_1 \epsilon_2}\left(x^{2/\epsilon_2} + y^{2/\epsilon_2}\right)^{\epsilon_2/\epsilon_1 - 2}\left(D_1 x^{1/\epsilon_2 - 1} y^{1/\epsilon_2} - D_2 x^{1/\epsilon_2} y^{1/\epsilon_2 - 1}\right)^2 \geq 0.\end{aligned} \tag{2.16}$$

All terms in (2.16) are greater than or equal to zero whenever $0 < \epsilon_1, \epsilon_2 < 2$, which means $F(t)$ is a convex in those cases. This convexity will be of later use when computing ray-superquadric intersections.

2.3 Spray Coating

Drum spray coating involves a rotating drum, typically with baffles, that is filled with tablets. The rotation causes the tablets to flow around the drum and mix. Inside the drum, sprays are pointed down toward the top layer of tablets at different positions to coat the tablets as they pass by. After the tablets have been sprayed and left these spray zones, hot air causes the solvent to evaporate and leave a solidified coating behind. This process runs until the desired outcome is achieved, which for instance could be measured by the overall mass gain or some prediction of coating uniformity. Certain statistics are of particular interest as indicators of the quality of the mixing, coating and subsequent end product. These statistics are the single visit residence time distribution, inter-tablet coating variability and intra-tablet coating variability. All these will be defined and described below.

2.3.1 Single Visit Residence Time

The single visit residence time distribution describes how much time tablets spend during and between single visits to the spray zone. This information can be used to asses mixing quality or build models that can be extrapolated into the future [1]. Such extrapolations are typically necessary for large coaters where it's infeasible to simulate the entire coating processes which might last for several hours.

At any given point in time, if a tablet has been recently hit by the spray it is in the spray zone, and otherwise in the drum zone. Ketterhagen defined a visit to the spray zone as an interval of time in which the tablet is hit by the spray at least every 0.1 seconds [18].

2.3.2 Inter-tablet Variability

Inter-tablet coating mass variability is defined as the relative standard deviation of accumulated coating mass between all tablets in the mixing process. It is one of the most important factors when tuning a coating process [1].

Let m_i denote the accumulated coating mass on tablet i and $\overline{m}$ the average accumulated coating mass across all N tablets. The coefficient of inter-tablet variation, is then defined as

$$\mathrm{CoV}_{\text{inter}} = \frac{1}{\overline{m}} \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (m_i - \overline{m})^2}. \tag{2.17}$$

2.3.3 Intra-tablet Variability

Intra-tablet coating mass variability is defined as the area-weighted relative standard deviation of coating density across the surface of a tablet. This uniformity is especially important for functional coatings, meaning those that either contain the active pharmaceutical ingredient or regulate the delivery of it.

To compute the intra-tablet variation the surface of each of the N tablets in the coating simulation is divided into N_p panels. Let A be the total surface area of a tablet, A_j the area of the j:th panel and h_{ij} the coating thickness on the j:th panel of the i:th tablet. There are many different ways to define this measure of variation, and two will be presented below [12]. The first takes each individual mean

$$\mu_i = \frac{1}{A} \sum_{j=1}^{N_p} h_{ij} A_j \tag{2.18}$$

and variance

$$\sigma_i^2 = \frac{1}{A} \sum_{j=1}^{N_p} (h_{ij} - \mu_i)^2 A_j \tag{2.19}$$

into account and defines the bulk intra-tablet coefficient of variation as

$$\mathrm{CoV}_{\mathrm{intra,bulk}} = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N} \sigma_i^2}}{\frac{1}{N}\sum_{i=1}^{N} \mu_i}. \tag{2.20}$$

The second measure combines all the tablets into a single representative tablet by first summing up the coating thickness across all panels

$$h_j = \sum_{i=1}^{N} h_{ij} \tag{2.21}$$

and then computing the mean μ_t and variance σ_t of these combined thicknesses as in (2.18) and (2.19) to finally arrive at

$$\mathrm{CoV}_{\mathrm{intra,agg}} = \frac{\sqrt{\sigma_t^2}}{\mu_t}. \tag{2.22}$$

These variability measures can also be symmetrically averaged across the particle surface if the particle shape and panels allow it, which can drastically cut down on the number of samples required to get a stable estimate. However, this is probably most relevant for $\mathrm{CoV}_{\mathrm{intra,agg}}$ as $\mathrm{CoV}_{\mathrm{intra,bulk}}$ typically is used to measure the actual variation which is not symmetrically averaged. Applying (2.22) to a symmetrically averaged representative tablet would give the measure $\mathrm{CoV}_{\mathrm{intra,agg}}$. Figure 2.3 shows a simple example that highlights the difference between $\mathrm{CoV}_{\mathrm{intra,bulk}}$ and $\mathrm{CoV}_{\mathrm{intra,agg}}$.

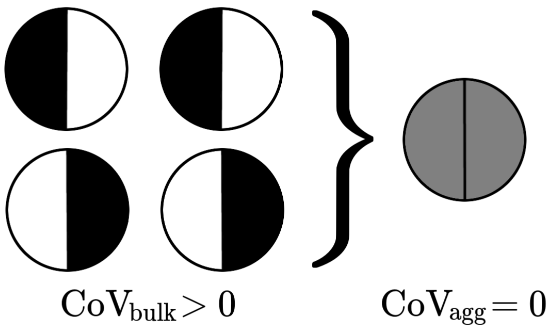

$\mathrm{CoV}_{\mathrm{bulk}} > 0$
$\mathrm{CoV}_{\mathrm{agg}} = 0$

Figure 2.3: An example that highlights the difference between $\mathrm{CoV}_{\mathrm{intra,bulk}}$ and $\mathrm{CoV}_{\mathrm{intra,agg}}$. Assume a population of 4 tablets, each divided into 2 panels. Two of the tablets have only received coating on side, and the other two on the other side. For each individual tablet, $\sigma_i > 0$ and so $\mathrm{CoV}_{\mathrm{intra,bulk}} > 0$. However, when the individidual coatings are aggregated the coating becomes uniform and so $\mathrm{CoV}_{\mathrm{intra,agg}} = 0$.

3
Implementation

This section explains the steps taken to ray cast and extract relevant coating statistics from the DEM simulation data. First, the main acceleration structure used to speed up the ray casting is motivated and described. Then, the individual steps taken by the ray caster and the complete ray casting algorithm are presented. Finally, the intra-particle representations used to summarize coating information are motivated and described.

3.1 Spatial Subdivision

Ray casting can be sped up by reducing the amount of ray-particle intersection checks performed. The naive approach of performing intersection checks between all pairs of rays and particles has a computational complexity of $\mathcal{O}(PR)$, where P is the number of particles and R the number of rays. By grouping objects into hierarchies or subdividing space and then only intersecting each ray with a subset of the particles, the complexity can be dramatically reduced to $\mathcal{O}(N + R)$ or $\mathcal{O}(N \log(N) + R)$ [19], which is much faster for large N and R.

Objects can be grouped in two fundamentally different ways. The first involves clustering objects that are close together and creating a hierarchy of bounding volumes. This procedure takes the positions and extents of all objects into account which makes it adaptive, meaning it will typically be finer in regions of space that are more densely packed with objects, and coarser in sparser regions. This is an advantage when the scene is non-homogeneous, meaning it contains objects of varying scales spread out non-uniformly in space. The other approach ignores the objects and instead just partitions space directly. This is obviously not adaptive which becomes a problem if the scene is not homogeneous, but it has other advantages. Fujimoto [20] found that spatial subdivision was an order of magnitude faster than octrees, a hierarchical structure, in terms of time taken to build and traverse the structure. The difference became even greater for homogeneous scenes.

The scene of a drum coater is very homogeneous because the tablets are all equal-sized and tightly packed near the bottom of the drum. This is close to ideal for spatial subdivision, which is why it was chosen as the main acceleration structure in the final implementation. The construction and traversal of this structure will be described in the following sections.

3.1.1 Grid Construction

Before particles can be indexed, the grid cells must be defined and particles added to all nearby grid cells. Before this can happen, the maximum extents of the grid must be computed, and this requires the component-wise minimum and maximum coordinates $\mathbf{p}_{\min}$ and $\mathbf{p}_{\max}$ across all tablets. The grid boundaries $\mathbf{g}_{\min}$ and $\mathbf{g}_{\max}$ are then computed as

$$\begin{aligned} \mathbf{g}_{\min} &= g_s \left\lfloor \frac{\mathbf{p}_{\min} - r_{\mathrm{B}}}{g_s} \right\rfloor \\ \mathbf{g}_{\max} &= g_s \left\lceil \frac{\mathbf{p}_{\max} + r_{\mathrm{B}}}{g_s} \right\rceil \end{aligned} \tag{3.1}$$

where g_s is the grid spacing. This ensures that all particles are completely contained within the grid. Next, particles are added to each grid cell that their bounding sphere overlaps. The distance d_{box} from a particle centroid $\mathbf{p}$ to a grid cell, which is an axis-aligned box with corners $\mathbf{b}_{\min}$ and $\mathbf{b}_{\max}$, is computed as

$$\begin{cases} dx = \max(\mathbf{b}_{\min,x} - \mathbf{p}_x, 0, \mathbf{p}_x - \mathbf{b}_{\max,x}) \\ dy = \max(\mathbf{b}_{\min,y} - \mathbf{p}_y, 0, \mathbf{p}_y - \mathbf{b}_{\max,y}) \\ dz = \max(\mathbf{b}_{\min,z} - \mathbf{p}_z, 0, \mathbf{p}_z - \mathbf{b}_{\max,z}) \\ d_{\mathrm{box}}^2 = dx^2 + dy^2 + dz^2. \end{cases} \tag{3.2}$$

and so the overlap criterion becomes $d_{\mathrm{box}} < r_{\mathrm{B}}$. This approximation is used because it is considerably cheaper than computing the exact intersection between the superquadric and the cell.

Given the superquadric parameters a, b, c, n_1 and n_2, Podlozhnyuk [8] showed that the bounding sphere radius r_{B} can be computed using

$$\begin{cases} \alpha = (b/a)^{2/(n_2-2)} \\ \gamma = (1+\alpha^{n_2})^{n_1/n_2-1} \\ \beta = (\gamma c^2/a^2)^{1/(n_1-2)} \\ \tilde{x} = 1/\left((1+\alpha^{n_2})^{n_1/n_2} + \beta^{n_1}\right)^{1/n_1} \\ r_{\mathrm{B}}^2 = (a\tilde{x})^2 + (\alpha b\tilde{x})^2 + (\beta c\tilde{x})^2. \end{cases} \tag{3.3}$$

If the grid spacing g_s is chosen such that $\geq\geq r_{\mathrm{B}}$, the particle cannot extend across more than one grid cell and can thus only reach the 27 cells in a 3x3x3 grid surrounding the cell containing the centroid. Further, if $g_s \geq 2r_{\mathrm{B}}$ the particle bounding sphere can only reach the 8 cells in the 2x2x2 cell grid of the octant containing the centroid. The equivalent 2D procedure is shown in Figure 3.1, where the equivalent quadrant division is shown as a dotted line. In this example, the centroid lies in the top-right octant, and so only the top-right 2x2 grid of cells need to be considered,

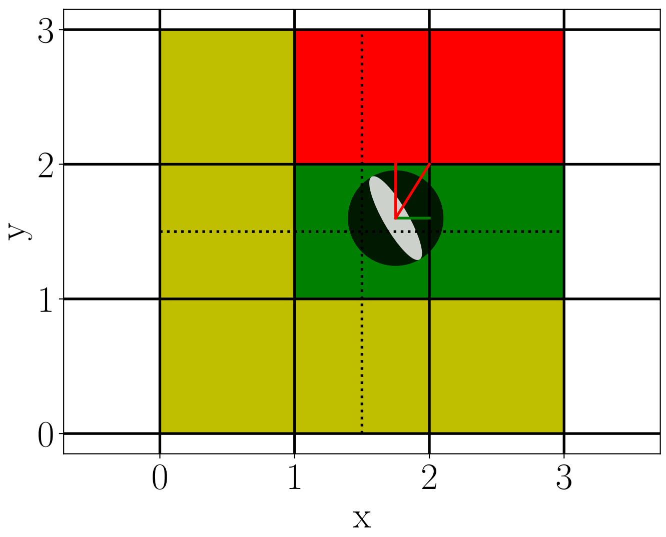


Figure 3.1: A 2D example of adding a single particle to the grid cells. The quadrant, shown as a dotted line, enables immediate exclusion of five grid cells (yellow). Then, for each remaining cell, the particles is added if its bounding sphere overlaps the cell (green) or not if it doesn't (red).

and the others, shown in yellow, can be ignored. The distance d_{box} to all remaining cells is computed and compared against r_{B}. In this case, two cells, shown in green, satisfied $d_{\text{box}} < r_{\text{B}}$ and so the particle is added to the candidate list of those two cells.

There is a space-time trade-off in how cells store the particles they contain. Benchmarks showed that storing a contiguous array of copies of all the particles in each cell speeds up the ray casting, and was found to be significantly faster than storing pointers to a unique set of the particle data. The ratio of the total number of copies stored in the index to the total number of particles is called the indexing factor, and it can significantly impact the runtime for large simulations.

3.1.2 Grid Traversal

Fujimoto [20] described how to efficiently traverse a uniform spatial grid using a technique called 3D Digital Differential Analyzer, or 3DDDA. This approach exploits the regularity of the grid to traverse it using only incremental calculations, meaning no multiplications or divisions which are generally slower. The grid traversal procedure begins by intersecting the ray $\mathbf{r}(t)$ with the axis-aligned bounding box that contains the entire grid, a procedure to be described later. If the ray origin is already inside the grid, that will be the intersection point. Once the intersection point t_{grid} is found it is converted to grid cell cordinates C via

$$\mathbf{c} = \frac{\mathbf{O} + \mathbf{D}t_{\text{grid}} - \mathbf{g}_{\text{min}}}{g_s} \tag{3.4}$$

and the step deltas are computed

$$\Delta\mathbf{t} = g_s / |\mathbf{D}| \tag{3.5}$$
$$\Delta\mathbf{c} = \operatorname{sgn}(\mathbf{D}) \tag{3.6}$$

where $\mathbf{t} = (t_x, t_y, t_z)$ is a vector containing the ray t-values of the next grid cell intersections along each respective axis, and $\mathbf{c}$ is the current grid cell coordinate. After initializing $\mathbf{t} = P + \Delta\mathbf{t}$, each subsequent step is taken by computing $\psi = \arg\min \mathbf{t}$ and incrementing the corresponding components

$$\mathbf{t}_\psi = \mathbf{t}_\psi + \Delta\mathbf{t}_\psi \tag{3.7}$$
$$\mathbf{c}_\psi = \mathbf{c}_\psi + \Delta\mathbf{c}_\psi. \tag{3.8}$$

This traversal continues until the ray leaves the grid or until an intersection is found.

3.2 Ray-Particle Intersection

Computing the intersection between a ray and a superquadric can in general be quite expensive, mainly due to the cost of computing exponentials. In order to reduce the number of superquadric intersection checks, the ray is first intersected with a couple of bounding volumes, namely the bounding sphere and the bounding box of the superquadric. The ray is first intersected with the sphere, and if it hits the sphere it is intersected with the bounding box, and only when it hits the bounding box is it intersected with the superquadric surface. These three steps and the surrounding considerations are detailed below.

3.2.1 Ray-Sphere Intersection

The points at which a ray intersects a sphere with radius r_B and centroid $\mathbf{c}$ can be expressed mathematically as

$$\|\mathbf{r}(t) - \mathbf{c}\|^2 = \|\mathbf{O} + \mathbf{D}t - \mathbf{c}\|^2 = r_\mathrm{B}^2. \tag{3.9}$$

Expanding the left-hand side gives

$$\|\mathbf{D}\|^2 t^2 - 2\langle\mathbf{D}, \mathbf{O} - \mathbf{c}\rangle t + \|\mathbf{O} - \mathbf{c}\|^2 - r_\mathrm{B}^2 =: t^2 + pt + q = 0 \tag{3.10}$$

where we have defined $p = -2\langle\mathbf{D}, \mathbf{O} - \mathbf{c}\rangle$ and $q = \|\mathbf{O} - \mathbf{c}\|^2 - r_\mathrm{B}^2$. Note that $\|\mathbf{D}\|^2 = 1$ since $\mathbf{D}$ is a normalized vector. The ray hits the sphere when this second order polynomial has real solutions, which happens when the discriminant D is positive

$$D = \left(\frac{p}{2}\right)^2 - q = \langle \mathbf{D}, \mathbf{O} - \mathbf{c} \rangle^2 + r_{\mathrm{B}}^2 - \|\mathbf{O} - \mathbf{c}\|^2 > 0. \tag{3.11}$$

Given $D > 0$, the distance to the nearest intersection is $t^* = -p/2 + \sqrt{D}$.

3.2.2 Ray-Box Intersection

Algorithm 1 shows how a ray is intersected with an axis-aligned box [16], which is a box whose normal vectors are parallel to the coordinate axes. The algorithm is based on intersecting the ray with slabs which are volumes enclosed by two parallel planes [21]. In this case the planes are the box faces, and so the slabs end up parallel to the coordinate axes which has computational advantages. Given a ray and the bounds of the axis-aligned box, the t values at which the ray intersects each slab are computed. This happens on line 6 of Algorithm 1 for the x-, y- and z-axis faces respectively. The last entry t_{near} and first exit t_{far} from each of the slabs is recorded. If after processing all three slabs $t_{\text{near}} < t_{\text{far}}$ it means that the ray was inside all three slabs simultaneously, or that it intersected the box. In that case the corresponding entry and exit points are returned, otherwise a miss indicator is returned instead.

This algorithm forms the basis of the ray-box intersection, but the particles and their bounding boxes typically have some orientation. To intersect such oriented bounding boxes the ray is rotated into the local coordinate system of the particle before Algorithm 1 is called. The ray is rotated rather than the particle because it is less computationally expensive than doing the opposite [16]. Let R be the 3x3 rotation matrix and $\mathbf{P}$ the position vector that respectively rotate and translate the particle from local coordinates to global coordinates. To intersect the ray (2.13) with this particle, the ray origin and direction vectors, which are in global coordinates, are transformed to the particle local coordinates by applying the corresponding inverse transformations

$$\mathbf{O}' = R^{-1}(\mathbf{O} - \mathbf{P}) \tag{3.12}$$

$$\mathbf{D}' = R^{-1}\mathbf{D} \tag{3.13}$$

where $\mathbf{O}'$ and $\mathbf{D}'$ are the origin and direction vectors of the transformed ray. These transformed vectors are then passed into Algorithm 1 and the intersection points are computed.

3.2.3 Ray-Superquadric Intersection

The difficulty of intersecting a ray with an implicit surface depends heavily on the surface. For completely general surfaces Kalra [22] developed a method guaranteed to give the correct results while using early termination criteria based on Lipschitz constants to keep down computational costs. Mitchell [23] introduced interval arithmetic to solve the same problem. Since only convex superquadrics are of interest in this work, a simpler approach can be used, but the principles remain the same.

Algorithm 1: Intersecting a ray with an axis-aligned box.

```
Input:
  $\mathbf{O}, \mathbf{D}$: Ray origin and direction
  $\mathbf{b}_{\text{min}}, \mathbf{b}_{\text{max}}$: Bounds of an axis-aligned box
Output:
  $(t_{\text{near}}, t_{\text{far}})$: The two intersection points, or void if no intersection
1  $t_{\text{near}} \leftarrow -\infty$
2  $t_{\text{far}} \leftarrow \infty$
3  for $\psi$ in $\{x, y, z\}$ do
4      if $\mathbf{D}_\psi = 0$ and $(\mathbf{O}_\psi < \mathbf{b}_{\text{min},\psi}$ or $\mathbf{O}_\psi > \mathbf{b}_{\text{max},\psi})$ then
5          return void
6      $\mathbf{t}_\psi \leftarrow ((\mathbf{b}_{\text{min},\psi} - \mathbf{O}_\psi)/\mathbf{D}_\psi, (\mathbf{b}_{\text{max},\psi} - \mathbf{O}_\psi)/\mathbf{D}_\psi)$
7      $t_{\text{enter}} \leftarrow \min(\mathbf{t}_\psi)$
8      $t_{\text{exit}} \leftarrow \max(\mathbf{t}_\psi)$
9      if $t_{\text{enter}} > t_{\text{near}}$ then
10         $t_{\text{near}} \leftarrow t_{\text{enter}}$
11     if $t_{\text{exit}} < t_{\text{far}}$ then
12         $t_{\text{far}} \leftarrow t_{\text{exit}}$
13 if $t_{\text{near}} <= t_{\text{far}}$ and $t_{\text{far}} >= 0$ then
14     return $(\max(t_{\text{near}}, 0), t_{\text{far}})$
15 else
16     return void
```

The problem of finding the roots of $F(t)$ from (2.14) and the corresponding intersection points, can be divided into two steps. The first is to isolate intervals of t where only a single root of F can occur, and for this an interval search method is used. The second is to refine the isolated roots to within a desired tolerance, and here Newton's method is used.

3.2.3.1 Bisection Search

Bisection search, shown in Algorithm 2, is used to find a point inside the superquadric between the bounding box intersection points. Given a starting interval $[l_0, r_0]$, the midpoint $m_0 = (l + r)/2$ and corresponding function values $F(m_0)$ and derivative $F'(m_0)$ are computed. If $F(m_0) > 0$, the sign of $F'(m_0)$ determines which of the two sub-intervals $[l_0, m_0]$ and $[r_0, m_0]$ contains the root, and the same procedure is then recursively applied to that subinterval. Because the function is convex the curve always lies above its tangent lines, so if the tangent line at the point m_k intersects the x-axis outside $[l_k, r_k]$, then f can't have any roots in $[l_k, r_k]$ and the algorithm can be safely terminated before reaching the tolerance.

Algorithm 2: Bisection search for root isolation.

```
Input:
  $f$: Continuous convex function
  $[l, r]$: Interval to be searched
  $tol$: Tolerance
Output: An interval $[l_{\text{out}}, r_{\text{out}}]$ containing a single root
1 while $(r - l) > tol$ do
2     $m \leftarrow (l + r)/2$
3     if $f(m) < 0$ then return $[l, m]$
4     $p \leftarrow m - f(m)/f'(m)$
5     if $p < l$ or $p > r$ then return void
6     if $f'(m) < 0$ then $l \leftarrow m$
7     else $r \leftarrow m$
8 return m
```

3.2.3.2 Newton's Method

Taking the dot product between ∇f from (2.12) and the ray direction vector D gives the derivative $F'(t)$ of $F(t)$ from (2.2.1), which can then be used to compute the Newton iterations

$$t_{n+1} = t_n - \frac{F(t_n)}{F'(t_n)}. \tag{3.14}$$

The gradient ∇f can be computed without computing any extra exponentials by making use of the already computed values of f in (2.11). This saves considerable time, especially for non-integer exponents. When benchmarked this was found to be almost as fast as only computing the function values, which is why Newton's method was favoured over derivative-free alternatives that have worse convergence rates.

3.2.3.3 Intersection Algorithm

The intersection algorithm takes as starting interval the two points from the bounding box intersection. It then uses bisection search to find a point inside the superquadric. Finally, if an interior point is found, Newton's method is used to refine the root inside the corresponding interval to get an accurate estimate of the intersection point.

Figure 3.2 shows the graph of $F(t)$ alongside a render of the corresponding intersection showing the ray, bounding box and superquadric. In this case, given the starting interval $[l_0.r_0]$, the algorithm starts by evaluating $F(m_0) < 0$ and finds that m_0 is inside the superquadric. Finding a point inside on the first iteration happens around 80% of the time in practice. Once an interior point has been found, Newton iterations start from the left-most endpoint of the current interval, which in this case

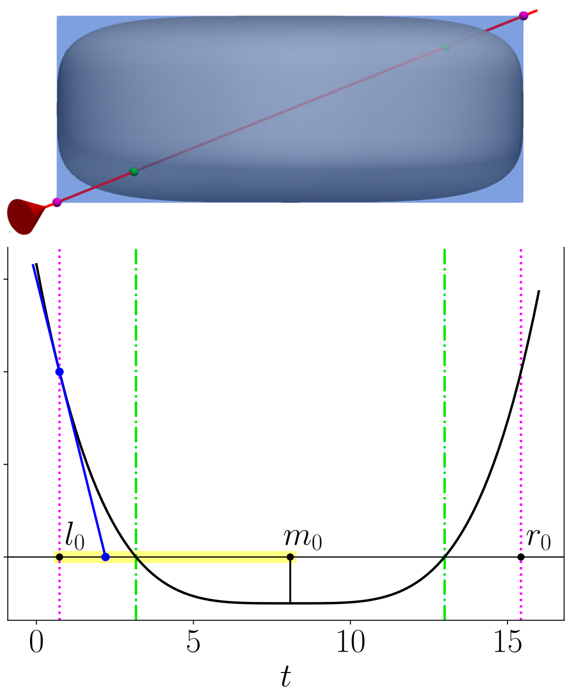


Figure 3.2: Example of a ray-superquadric intersection, showing both a render of the ray passing through the superquadric particle and the corresponding intersection function $F(t)$. The sought roots of $F(t)$ are shown as green dots and dashed lines in the render and graph respectively. Similarly, the box intersection points are shown in magenta. The box intersection points determine the starting interval $[l_0, r_0]$ and the initial midpoint m_0. In this case $F(m_0) < 0$, and so the first root must be in the interval $[l_0, m_0]$, highlighted in yellow. Newton iterations then start from l_0. The first Newton iterate and corresponding tangent line are shown in blue.

is l_0. The Newton iterations continue until the step length is less than the desired tolerance. The convexity of $F(t)$ guarantees that the Newton iterations will always converge.

Figure 3.3 shows an example of an intersection check where the ray misses. The box intersection points determine the starting interval $[l_0, r_0]$ and the initial midpoint $m_0 = a$. Since $F(m_0) > 0$ and $F'(m_0) < 0$ the next interval $[l_1, r_1]$ becomes $[m_0, r_0]$. Further, because the tangent intersection point corresponding to m_0 falls inside $[l_0, r_0]$, the procedure does not terminate. Next, the midpoint $m_1 = b$ of $[l_1, r_1]$ is computed and a similar argument gives the next interval $[l_2, r_2] = [l_1, m_1]$, where once again the tangent intersection point falls inside the corresponding interval. Finally, the midpoint $m_2 = c$ of $[l_2, r_2]$ is still not inside the superquadric, but the tangent intersection point now falls outside of the corresponding interval and the procedure correctly terminates since there can be no root in $[l_2, r_2]$.

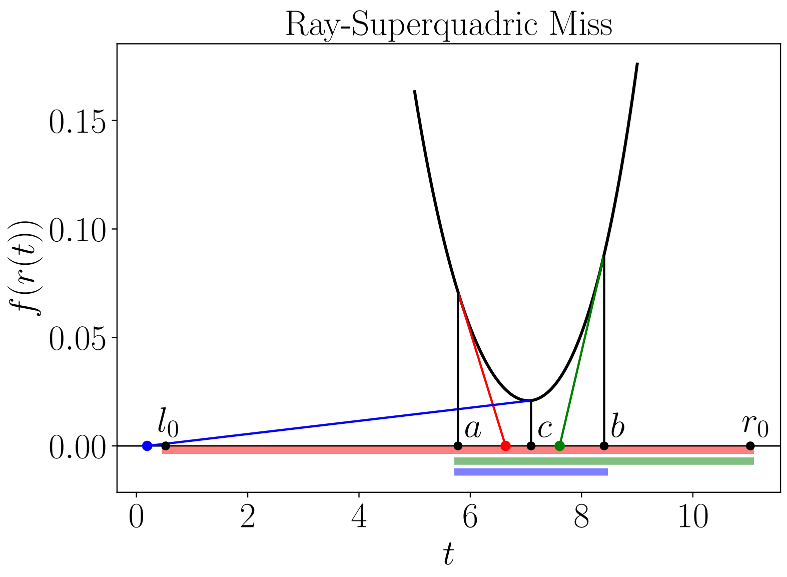


Figure 3.3: Example of a ray-superquadric intersection where the ray misses. The initial search interval $[l_0, r_0]$ and corresponding tangent line at a are shown in red. Likewise, the two subsequent intervals and tangents at b and c are shown in green and blue respectively.

3.3 Ray Casting Algorithm

The entire ray casting procedure can now be assembled from all the parts described above. A diagram of the procedure carried out for a single ray is in Figure 3.4, a more detailed description in Algorithm 3, and a visualization in Figure 3.5. Following the diagram in Figure 3.4, as the ray enters a cell, the corresponding list of particles is retrieved. Each candidate in the list is intersected following the procedure inside the dashed box, which begins with a sphere intersection check, followed by a bounding box intersection check in case of a sphere hit, and finally a superquadric intersection check in case of a box hit. These intersection checks get progressively more expensive, so it is important to abort and go to the next candidate at the first miss. The procedure terminates when the nearest particle has been hit.

Algorithm 3 shows more details than the diagram, particularly when it comes to mailboxing and the termination criteria. Firstly, the mailbox is simply a set of the IDs of all the particles whose bounding boxes and superquadrics have been checked for intersection with the current ray. This set is checked before doing any box or superquadric intersection checks to avoid performing the same expensive checks multiple times. Secondly, the termination criteria makes use of the distances to the bounding spheres. All sphere intersection checks are done and the distances to each hit sphere are stored. Then, these distances are sorted in ascending order and the remaining candidates are intersected in the sorted order. As soon as the distance to the bounding sphere is greater than the currently closest superquadric hit, a closer hit is impossible since that distance bounds the distance to the superquadric from below, and the algorithm terminates.

Each particle has its own set of three counters that keep track of

- The total number of times the particle has been hit by a ray
- The last time the particle was hit
- The last time the particle entered the spray zone

These counters are updated whenever the particle is hit by a ray, as shown on line 8 of algorithm 4. More specifically, when the particle is it, if the time since last hit exceeds the spray zone threshold of 0.1 seconds the particle has now re-entered the spray zone after leaving it. The duration of spray and drum zone visits can then be added to the corresponding distributions. The sequential nature of the residence times slightly hinders a fully concurrent implementation, which would otherwise be quite straightforward. Still, because the grid is rebuilt for each time step, the concurrent implementation of having the threads work on separate files and pooling the computed results still gives a very good speed-up for both the grid construction and the ray tracing compared to a single-threaded implementation.

The spray is modelled as an elliptical cone [11]. Random samples $\rho \in [0, 1]$ and $\alpha \in [0, 2\pi]$ are used to angle samples (θ, φ) from an ellipse with width and height (a, b)

$$\begin{aligned} x &= \rho^u a \cos(\alpha) \\ y &= \rho^u b \sin(\alpha) \end{aligned} \tag{3.15}$$

where u is a uniformity factor. Using $u = 0.5$ gives uniform samples from the ellipse, whereas $u > 0.5$ and $u < 0.5$ concentrates the samples more toward the center and edges respectively. Once the samples have been drawn, the final ray directions are computed by rotating the spray axis using x and y as Euler angles in the horizontal and vertical direction respectively. Once the cone of rays has been generated, they are sorted by the $\mathbf{D}_z$. This is done to improve cache-coherency since rays closer together will take more similar paths through the grid and perform the same intersection checks.

3.4 Intra-particle Sampling

In order to quantify intra-particle variability the particle surface was divided into panels. This discretization of the surface may not be strictly necessary, but it provides a way to summarize the potentially huge amounts of data generated by the ray casting. Desirable qualities of a panel parametrization include

- Uniform distribution of panel sizes: If some panels are very small the corresponding density estimates will have a large variance.
- Computationally efficient inversion: The intersection points returned by the ray caster need to be mapped to the correct panels, and this must be done for every ray, which means it should be fast.

Algorithm 3: Single ray casting.

```
Input: A ray and a grid index of particles
Output: A ray-particle intersection
1  if ray misses grid then return null
2  Initialize coordinate inside the grid
3  Initialize hit to INF
4  while true do
5      Get candidates at coordinate
6      for each candidate do
7          Intersect ray with bounding sphere
8          if ray hits then store candidate and distance to sphere
9      Sort remaining candidates by ascending sphere distance
10     for each candidate do
11         if candidate sphere distance is further than hit then break
12         if candidate in mailbox then continue
13         Store candidate in mailbox
14         Intersect ray with bounding box
15         if ray misses then continue
16         Intersect ray with superquadric
17         if ray hits and the hit is closer then store new hit
18     if a closer hit is impossible then break
19     Step into next grid cell and update coordinate
20     if coordinate is outside the grid then break
21 return hit
```

Algorithm 4: Batch file ray casting.

```
Input: Ray casting settings and input data files
1  Set up rays and output data structures
2  for each input file do
3      Compute grid index for the particles in file
4      for each ray do
5          Intersect the ray with all the particles in grid
6          if the ray hits a particle then
7              Store information about the hit
8              Update counters for the hit particle
9      Compute and export all relevant information
10 return
```

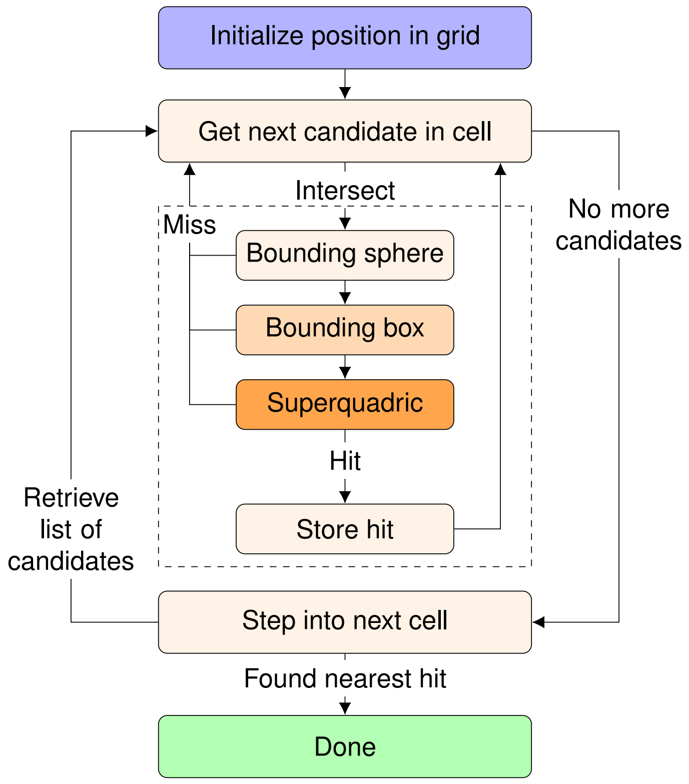
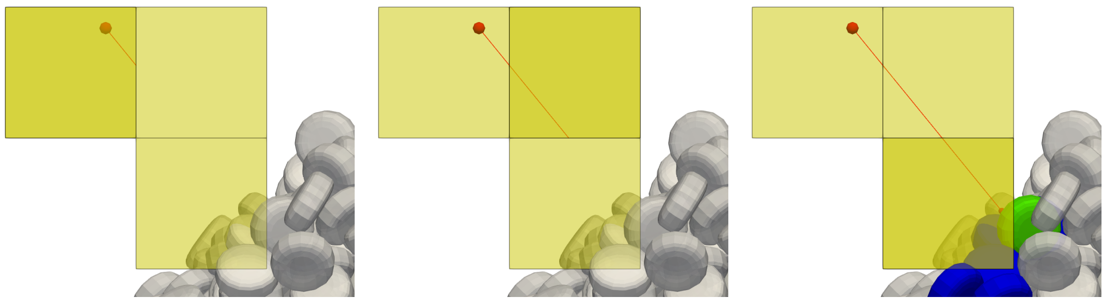


Figure 3.4: Diagram of grid traversal and particle intersection for a single ray. The cost of the checks inside the intersection loop are highlighted in orange, with more color indicating a more expensive operation.



(a) Step 1. **(b)** Step 2. **(c)** Step 3.

Figure 3.5: A single ray cast showing the traversal and intersection checks. The grid position is initialized given the ray origin in (a). The traversal then starts, and the ray first passes through the two empty in (a) and (b) before finding some particles in the third cell in (c). The candidates in the final cell are highlighted in blue, and the hit particle in green.

- Eightfold symmetry across octants: So that $\text{CoV}_{\text{intra,agg,sym}}$ can be computed, which can reduce the required number of samples.
- Smooth function of the shape parameters: Small changes to shape parameters should produce small changes to the panel distribution.

Below follow descriptions of some different parametrizations.

3.4.1 Uniform Parametrization

A straightforward way to create panels is to uniformly sample η and ω in the superquadric parametrization (2.6), that is to set

$$\begin{aligned}\eta_i &= -\frac{\pi}{2} + \frac{\pi i}{N_\eta - 1}\ ,\ i = 0, \dots, N_\eta - 1 \\ \omega_j &= -\pi + \frac{2\pi j}{N_\omega - 1}\ ,\ j = 0, \dots, N_\omega - 1.\end{aligned} \tag{3.16}$$

This creates a uniform mesh in parameter space, shown in Figure 3.6a, where each rectangle with corners (ω_j, η_i), (ω_{j+1}, η_i), $(\omega_{j+1}, \eta_{i+1})$ and (ω_j, η_{i+1}) corresponds to a panel on the superquadric surface. Given an estimate of the ray-superquadric intersection point $\mathbf{h}$, the panel it belongs to can be estimated by inverting (2.6)

$$\begin{aligned}\eta^* &= \arcsin\left(\operatorname{sgn}(z)\left(\left|c^{-1}z\right|\right)^{1/\epsilon_1}\right) \\ \omega^* &= \operatorname{arctan2}\left(\operatorname{sgn}(y)\left|ay\right|^{1/\epsilon_2},\ \operatorname{sgn}(x)\left|bx\right|^{1/\epsilon_2}\right).\end{aligned} \tag{3.17}$$

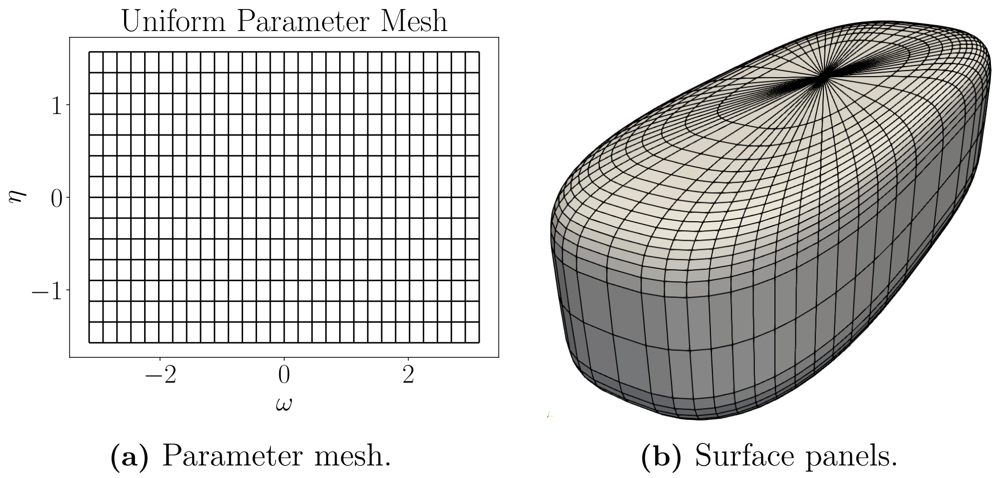
Here, arctan2 is the function that returns the inverse tangent in the range $[-\pi, \pi]$ using the signs of its two arguments. The signed power functions have been written out for clarity. Note however that due to the approximate intersection and finite precision floating point calculations, $\mathbf{h}$ may be far enough from the superquadric surface to cause significant distortions in the coordinates returned by (3.17). This can be remedied by rescaling $\mathbf{h}$ so that it lies on the surface. Specifically, let $\beta >= 0$ be the scale factor such that $\mathbf{h}_s = \beta\mathbf{h}$ is on the superquadric surface, or equivalently $f(\mathbf{h}_s) = 0$ where f is the inside-outside function from (2.11). Plugging $\mathbf{h}_s$ into f

$$f(\mathbf{h}_s) = \left(\left|\frac{\beta h_x}{a}\right|^{n_2} + \left|\frac{\beta h_y}{b}\right|^{n_2}\right)^{n_1/n_2} + \left|\frac{\beta h_z}{c}\right|^{n_1} - 1 = 0 \tag{3.18}$$

and solving for β gives

$$\beta = (f(\mathbf{h}) + 1)^{-1/n_1}. \tag{3.19}$$

Now, given the approximate intersection point $\mathbf{h}$, it is scaled by β to get $\mathbf{h}_s$ which in turn is used to compute η^* and ω^* using (3.17). Finally, the corresponding panel is



(a) Parameter mesh. **(b)** Surface panels.

Figure 3.6: Uniform parametrization mesh and the resulting surface panels.

be retrieved by rounding η^* and ω^* to the sampled values η_i and ω_j which uniquely identify the panel.

With this approach the parametrization points on the surface tend to concentrate around poles and areas of high curvature as can be seen in Figure 3.6b. This non-uniformity gets worse as the values of the blockiness parameters increase, and so this parametrization was deemed unfit for the task of summarizing intra-particle coating variability.

3.4.2 Uniform Sampling

A more uniform set of panels can be created by sampling superellipses at roughly equal arc length. Pilu [24] proposed such a sampling approach based on first order Taylor approximations. Given the superellipse parametrization $\mathbf{x}(\theta)$ from (2.2), the arc length between the two points $\mathbf{x}(\theta)$ and $\mathbf{x}(\theta + \Delta(\theta))$ can for small values of the function $\Delta(\theta)$ be approximated by the distance between the two points

$$D^2 = \left\|\mathbf{x}(\theta + \Delta(\theta)) - \mathbf{x}(\theta)\right\|^2. \tag{3.20}$$

Applying a first order Taylor approximation

$$D^2 \approx \left\|\frac{\partial \mathbf{x}}{\partial \theta}\Delta(\theta)\right\|^2 = \left\|\begin{bmatrix} a\epsilon \cos^{\epsilon-1}(\theta)(-\sin(\theta))\Delta(\theta) \\ b\epsilon \sin^{\epsilon-1}(\theta)\cos(\theta)\Delta(\theta)\end{bmatrix}\right\|^2 \tag{3.21}$$

and solving for $\Delta(\theta)$ gives

$$\Delta(\theta) = \frac{D}{\epsilon}\sqrt{\frac{\cos^2(\theta)\sin^2(\theta)}{a^2\cos^{2\epsilon}(\theta)\sin^4(\theta) + b^2\sin^{2\epsilon}(\theta)\cos^4(\theta)}}. \tag{3.22}$$

This $\Delta(\theta)$ does not work for θ close to 0 and $\pi/2$. In the prior case we have that $\sin(\theta) \approx \theta$ and $\cos(\theta) \approx 1$ and so the superellipse parametrization (2.6) can be simplified to

$$\mathbf{x}_0(\theta) = \begin{bmatrix} a \\ b\theta^{\epsilon} \end{bmatrix} . \tag{3.23}$$

Using the corresponding formulas in the latter case and a Taylor approximation to solve for $\Delta(\theta)$ as before in (3.21) and (3.22) gives, respectively

$$\Delta_0(\theta) = \left(\frac{D}{b} + \theta^{\epsilon}\right)^{1/\epsilon} - \theta \tag{3.24}$$

$$\Delta_\pi(\theta) = \left(\frac{D}{a} + (\frac{\pi}{2} - \theta^{\epsilon})\right)^{1/\epsilon} - (\frac{\pi}{2} - \theta) \tag{3.25}$$

A threshold τ is used to switch between (3.22), (3.24) and (3.25) depending on the value of θ. The iterative sampling is done starting from both ends of the part of the superellipse inside the first quadrant

$$\begin{aligned} \theta_i &= \theta_{i-1} + \Delta(\theta_{i-1}) \ , \ \theta_0 = 0 \\ \theta_j &= \theta_{j-1} - \Delta(\theta_{j-1}) \ , \ \theta_0 = \pi/2. \end{aligned} \tag{3.26}$$

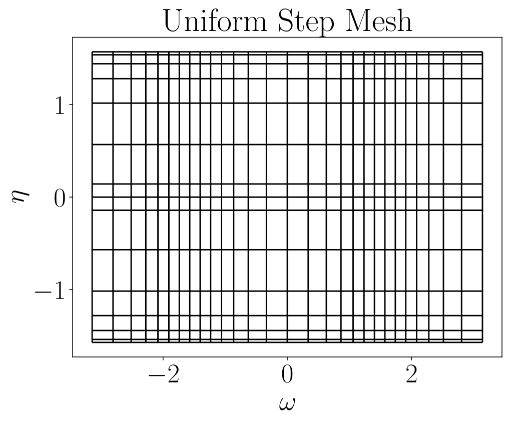
The sampling scheme in (3.26) continues until $\theta_i > \theta_j$. The final samples in the middle will typically not be a distance D apart, but this can be corrected by first sampling with the desired distance D, adjusting it depending on the resulting middle distance to get $\widetilde{D}$, and then resampling with the adjusted value.

To uniformly sample a superquadric with scale parameters a, b and c and shape parameters ϵ_1 and ϵ_2, the two superellipses $(0.5(a+b), c, \epsilon_1)$ and (a, b, ϵ_2) are uniformly sampled using the method described above. Taking their spherical product (2.4) then gives the desired surface parametrization.

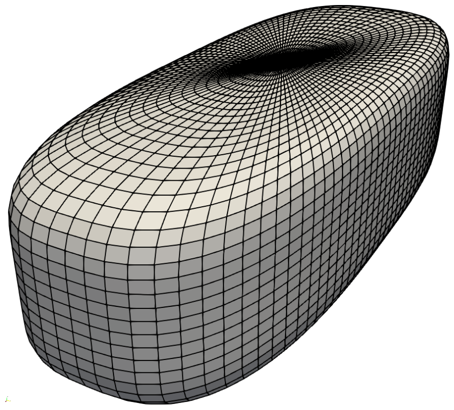
This parametrization a significant improvement over the uniform sampling and retains the rotational symmetry. An example of the parameter mesh and the resulting panels can be seen in Figure 3.7. However, the panel size uniformity is still far from perfect since sampled points still concentrate around the poles. On top of this, the threshold parameter τ generally needs to be tuned after the superquadric parameters to get consistent results.

3.4.3 Cube Sampling

There are many different methods to generate quasi-uniform samples on a sphere, and some of those methods turn to work quite well also for superquadrics. One of those methods is the cube-sphere [25], where the idea is to create a mesh on the unit



(a) Parameter mesh.



(b) Surface panels.

Figure 3.7: Uniform step parameter mesh and the resulting surface panels.

cube and then project that mesh onto the circumscribed sphere. More specifically, given a parametrization along each coordinate axis

$$x_i, y_j, z_k \in [-1, 1], \quad \begin{array}{l} i = 0, \dots, N_x \\ j = 0, \dots, N_y \\ k = 0, \dots, N_z \end{array} \tag{3.27}$$

and a transformation that improves the area-preservation of the projection

$$g(x) = \tan\left(\frac{\pi}{4}x\right) \tag{3.28}$$

vectors that correspond to mesh points on each of the 6 faces of the cube can be defined. For instance, the vector families of the faces orthogonal to the x-axis, call them the $+X$ and $-X$ faces, are given by

$$\mathbf{v}_1(y_j, z_k) = \begin{bmatrix} 1 \\ g(y_j) \\ g(z_k) \end{bmatrix}, \; \mathbf{v_2}(y_j, z_k) = \begin{bmatrix} -1 \\ g(y_j) \\ g(z_k) \end{bmatrix}. \tag{3.29}$$

In a similar way, the vector families $\mathbf{v}_3$, $\mathbf{v}_4$, $\mathbf{v}_5$ and $\mathbf{v}_6$ are set up for the $+Y$, $-Y$, $+Z$ and $-Z$ faces respectively. To generate points on the superquadric, these vectors are first scaled onto the normalized superquadric surface, that is with $\widetilde{a} = \widetilde{b} = \widetilde{c} = 1$, using (3.18). Then they are scaled to the actual values of a, b and c, which gives the surface panels seen in Figure 3.8. Because the number of samples along each axis can be changed independently in (3.27), uniform grids can be achieved even on particles with very large aspect ratios.

Inverting this parametrization starts by rescaling the vector $\mathbf{h}$

$$v = \begin{bmatrix} h_x/a \\ h_y/b \\ h_z/c \end{bmatrix} \tag{3.30}$$

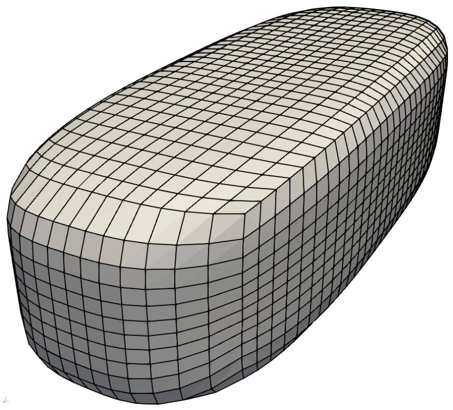


Figure 3.8: Surface panels of the cube parametrization.

and then finding the dominant axis

$$\psi = \arg\max\{|v_x|, |v_y|, |v_z|\}. \tag{3.31}$$

Together, ψ and $\text{sgn}(v_\psi)$ determine which face $\mathbf{h}$ belongs to. If for example $\psi = 1$ and $\text{sgn}(v_\psi) = +1$, the face is $+X$ and the cube vector is recovered using

$$w = \begin{bmatrix} 1 \\ v_y/|v_x| \\ v_z/|v_x| \end{bmatrix} \tag{3.32}$$

and the panel index in that case is found by binary searching for w_y and w_z in y_j and z_k respectively.

The cube parametrization can create much more uniform sets of panels than the previous approaches. This is partly because it doesn't have any singularities at the poles and partly because the number of panels along each axis can be independently changed to fit particles of varying aspect ratio. The downside is that it lacks rotational symmetry which sometimes causes panels to wrap around sharp corners in unappealing ways. This downside was however outweighed by the other desirable qualities, and so the cube parametrization was selected and used for the intra-particle analysis.

4

Methods

DEM simulations were carried out to generate data for validating and benchmarking the rays caster, as well as to study the impact of particle shape on the resulting coating outcome. The software, simulations and methodologies used to implement, validate and test the ray caster are motivated and described below.

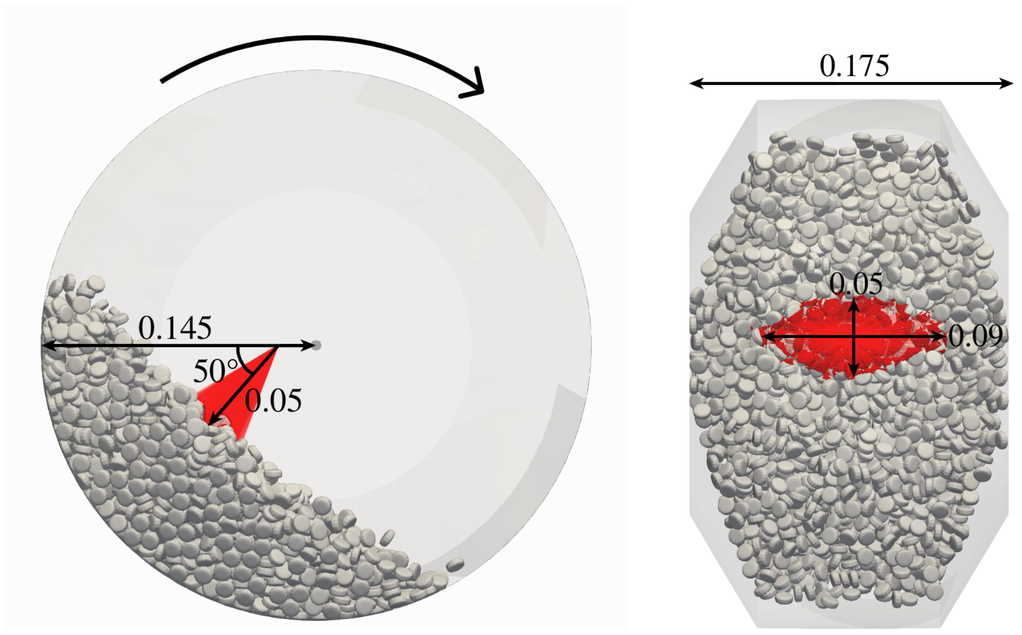
4.1 Software

ParaView [26] was used to visualize the coater, tablets, rays and surface coating densities, and LIGGGHTS [9] was used to run the DEM simulations, the details of which are given in the next section.

The ray caster was implemented as a standalone program in C++ using only standard library components. There were two main reasons for choosing C++. The first is performance. An initial prototype of the ray caster was written in Python, but early benchmarks found that C++ was orders of magnitude faster. C++ is a compiled low-level language that can provide very fast execution times, and the speed is critical to making the ray caster a useful tool in practice. The second reason for choosing C++ is that LIGGGHTS is implemented in C++, and writing the ray caster in the same language with minimal dependencies would simplify potential future integration with LIGGGHTS.

4.2 DEM Simulation

The simulations were carried out using a model of a lab-scale rotating drum coater. Figure 4.1 shows a side-view of the drum and a top-view that shows the spray zone. The coater has four baffles, a radial diameter of 145 mm and an axial depth of 175 mm. The spray was positioned at an offset of 20 mm from the center of the drum and angled downward 50 degrees from the horizontal plane so as to be roughly perpendicular to the tablet bed. The ray direction vectors representing the spray were uniformly sampled from an elliptical cone with angles 40 and 20 degrees for the major and minor axes respectively, that is with uniformity factor $u = 0.5$ in (3.15). This resulted in a spray-to-bed distance of around 50 mm and an elliptical spray zone of size 90 mm by 50 mm in the spanwise and streamwise directions respectively.



(a) Side view. **(b)** Top view.

Figure 4.1: Setup used in the DEM simulations. Distances are shown in meters.

The simulation procedure starts by dropping all the tablets into the drum over the course of 2 seconds. Then, a pre-run period of 5 seconds enables the system to reach a state of dynamic equilibrium before the main part of the simulation starts. The main part then runs for 180 seconds, during which data containing particle positions and orientations are written to text files on disk every 0.02 seconds using the LIGGGHTS command *dump*.

4.2.1 Pharmaceutical Shapes

A set of typical pharmaceutical particle shapes was simulated and studied in order to validate and benchmark the ray caster and to compare the results to previous work. The shapes and the intra-particle panels are shown in Figure 4.2, and the parameters and properties are given Table 4.2. Two larger simulations of the Round shape were also used to assess the ray casting performance at scale. These simulations will be referred to as Round$_{\text{medium}}$ and Round$_{\text{large}}$, and they contained 650 thousand and 2 million tablets respectively.

4.2.2 Blockiness Variation

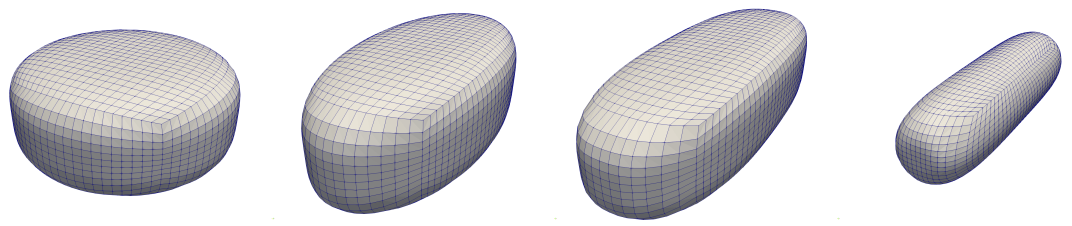
The other set of shapes are variations of the Round shape. The purpose of these variations was to study the impact of blockiness on the coating outcome. Their parameters are shown in Table 4.3 and Figure 4.3 shows a side-view highlighting the different levels of blockiness. The shapes were made increasingly blocky by raising the value of n_1. Round4 served as the baseline with $n_1 = 4$. Round6 and Round10

Table 4.1: Simulation settings used in LIGGGHTS.

Parameter	**Value**
Pan rotational speed	22 RPM
Pan loading	1.5 kg
Number of particles	3K-15K
Particle mass	93-505 mg
Contact force model	Hertz-Mindlin
Particle Poisson's ratio	0.25
Wall Poisson's ratio	0.30
Particle density	1500 kg/m^3
Wall density	7500 kg/m^3
Coef. of static friction (particle-particle)	0.5
Coef. of static friction (particle-wall)	0.5
Coef. of rolling friction	0.0
Coef. of restitution	0.5
Total simulation time	180 s
Simulation time step	1.7-3.5 10^{-5}
Sampling interval	0.02 s

Table 4.2: Parameters and properties of the pharmaceutical particle shapes. The aspect ratio is computed as AR $= 0.5(a+b)/c$ (inverted for the Capsule) and the maximum aspect ratio as AR$_{\text{max}} = \max(a,b,c)/\min(a,b,c)$.

Particle Shape	**a** (mm)	**b** (mm)	**c** (mm)	$\mathbf{n_1}$	$\mathbf{n_2}$	**Volume** (mm^3)	N_{panels}	AR	AR$_{\text{max}}$
Round	5.0	5.0	2.25	4.0	2.0	309	2024	2.22	2.22
Oval	7.0	3.5	2.5	4.0	2.0	336	2064	2.10	2.80
Caplet	8.0	3.0	2.25	4.0	2.5	319	2000	2.44	3.56
Capsule	1.5	1.5	5.0	4.0	2.0	62	2072	3.33	3.33



(a) Round. **(b)** Oval. **(c)** Caplet. **(d)** Capsule.

Figure 4.2: The four shapes in 4.2, showing the panels used for intra-tablet variability estimation.

Table 4.3: Parameters and properties of Round blockiness variation shapes.

Particle Shape	**a** (mm)	**b** (mm)	**c** (mm)	**n$_1$**	**n$_2$**	**Volume** (mm^3)
Round4	5.0	5.0	2.25	4.0	2.0	309
Round6	5.0	5.0	2.25	6.0	2.0	330
Round6EV	5.0	5.0	2.1	6.0	2.0	308
Round10	5.0	5.0	2.25	10.0	2.0	343
Round10EV	5.0	5.0	2.0	10.0	2.0	306

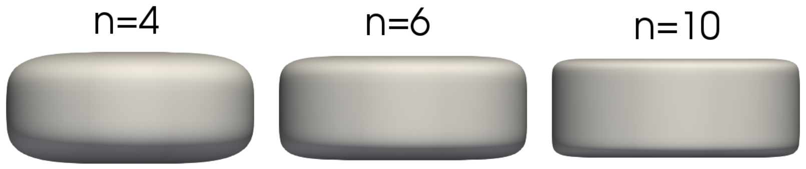


Figure 4.3: Profiles of Round4, Round6EV and Round10EV showing the variation in blockiness for different values of n_1.

used $n_1 = 6$ and $n_1 = 10$ respectively and the same thickness $c = 2.25$ as Round4, making their volumes slightly larger. Round6EV and Round10EV also used $n_1 = 6$ and $n_1 = 10$ respectively but with slightly smaller tablet thicknesses to keep the volume constant. The same panel dimensions were used across all Round shapes.

4.3 Benchmarks

Relevant parts of the code were timed using the C++ class *chrono::high_resolution_clock*. The runtime benchmark that used all 180 seconds of simulation data showed no significant variation between multiple runs, so only single benchmarks were performed in those cases. For all other benchmarks the minimum runtimes of 3 runs was recorded. All benchmarks were run on a conventional desktop computer using an Intel Core i5-2500K 3.30 GHz CPU.

5

Results and Discussion

Two main sets of results will be presented below. The first set gives performance measures of the implemented algorithm, which provide insight into the process of implementing and using a ray caster in a DEM context. The second set compares the intra-tablet coating variability to previous studies and shows the effects of using superquadrics for DEM shape approximation in the context of drum coating.

5.1 Ray Casting

The ray caster has a number of parameters that need to be set, like the intersection tolerance, the number of rays and the grid spacing. The results below will shed some light on reasonable values, and what kind of runtimes can be expected for some different coating simulations.

5.1.1 Intersection tolerance

The ray-superquadric intersection method used in this work is exact, meaning it will never misclassify ray hits and misses, given infinite machine precision. However, because computers use finite precision this will never be achieved in practice. Two validation tests were used to assess the impact of finite precision. In the first, a single superquadric particle was placed at the origin, and 100 million rays were cast from uniformly randomly sampled points on a large sphere surrounding the particle towards points on the particle surface, meaning all rays were hitting. The proportion of rays that hit was recorded, and for all superquadric shapes and a sufficiently small tolerances τ, a perfect hit rate of 1 was achieved. In the second test, in order to estimate a tolerance that would be sufficient in practice, 100 million rays modeling the spray were cast on data from the different simulations. The tolerance value was initially set to $\tau = \min(a, b, c)/10$ and then lowered in small steps until the number of superquadric intersection checks stopped changing. No differences in intersection outcome was observed for tolerances below $\tau = \min(a, b, c)/1000$, and so this tolerance was used in all subsequent analyses.

5.1.2 Grid Spacing

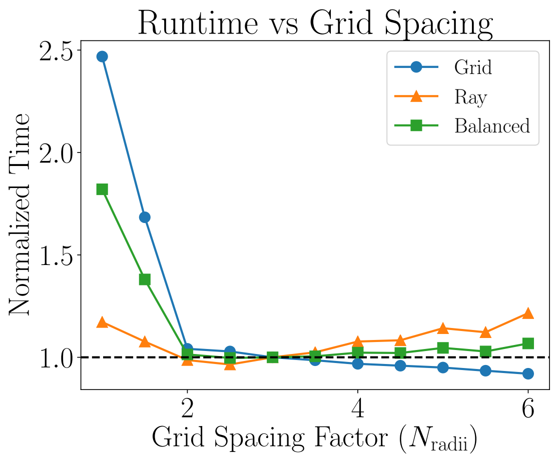
The impact of grid spacing on the runtime was studied by running benchmarks for different grid spacings. Figure 5.1 shows how the runtime, broken down into grid construction and ray casting, depends on the grid spacing factor in the Round, Capsule and Round$_{\text{medium}}$ cases. Given the grid spacing factor α, the grid spacing is computed as $g_s = \alpha r_{\text{B}}$, where r_{B} is the radius of the minimum bounding sphere of the superquadric particle.

Figures 5.1a and 5.1b show that the ray casting runtime is minimized for grid spacings factors $\alpha = 2.5$ and $\alpha = 2$ respectively. The ray casting slows down for smaller grid spacings because the ray has to traverse more grid cells and access more memory locations, as well as for larger grid spacings where the cells contain more particles which mean more sphere intersection checks have to be performed. In the latter case, the increase in runtime is substantial despite the relatively cheap cost per ray of sphere intersection checks because the amount of checks increases with the volume of the grid cells. This effect is greater for the longer Capsule shape since its bounding spheres are packed more tightly than in the Round case.

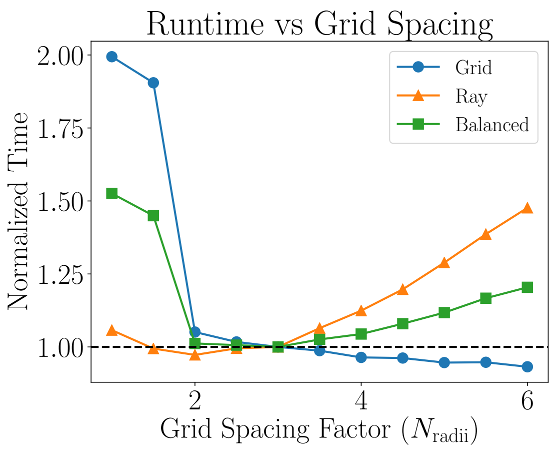
Whereas the ray casting runtime increases with increasing grid spacing, the opposite is true for the grid construction runtime, and this is mainly due to the indexing factor shown in Figure 5.2. The indexing factor is the ratio of the number of copies of particles stored in the index to the total number of particles. In the extreme case of using a grid spacing factor of $\alpha = 1$, an average of 20 copies of each particle are stored in the index, which increases memory usage and runtime proportionally. Another important factor is the octant method which is used for grid spacings $g_s \geq 2r_{\text{B}}$. It significantly reduces the runtime by decreasing the amount of sphere-box distance checks in the grid construction by a factor of almost 4, since only 8 of the 27 surrounding grid cells need to be checked.

The total runtime under a balanced workload in the Round case is basically the same for grid spacings $\alpha \geq 2$, because the increase in ray casting runtime is cancelled out by the decrease in grid construction runtime. In the Capsule case, an equal workload balance attains a clearer minimum runtime around $\alpha = 3$ because of the much greater slow-down in ray casting for larger grid spacings. Still, even using a the worst grid spacing factor of $\alpha = 6$ in that case is only about 25% slower than the optimal grid spacing.

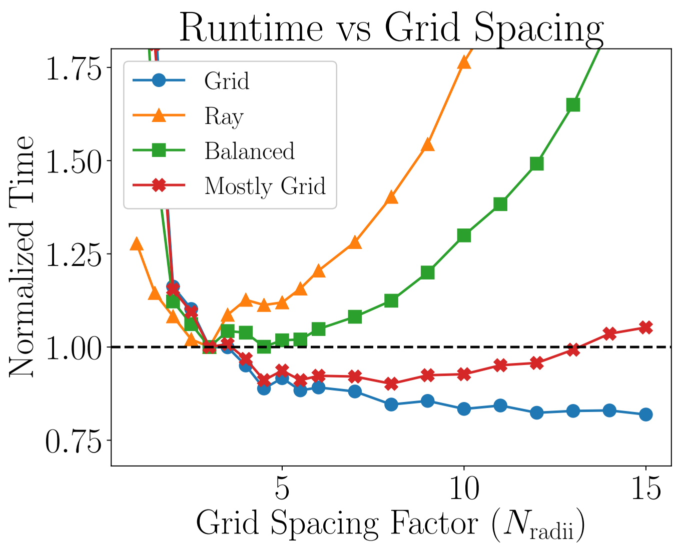
Figure 5.1c shows the impact of grid spacing on runtime for a larger simulation, namely the Round$_{\text{medium}}$ case with 650K tablets of Round shape. Its balanced workload graph attains a couple of minima in the range $\alpha \in [3, 5]$. An additional graph corresponding to a heavy grid construction workload is shown as well. As simulations are scaled up, the workload balance will shift in the direction of more grid construction. As the amount of data increases, so does the time to construct the index. In the Round$_{\text{medium}}$ case, the runtime under the heavy grid construction workload is minimized for a grid spacing factor around $\alpha = 8$. The curve is however quite flat, and the relative difference in runtime for $\alpha \in [2, 15]$ stays within about 25%, showing that the choice of grid spacing factor in that case has a modest impact on the final runtime.



(a) Round, 3K tablets.



(b) Capsule, 16K tablets.



(c) Round$_{\text{medium}}$, 650K tablets.

Figure 5.1: Runtime as a function of the grid spacing factor α for the Round$_{\text{medium}}$ simulation. The runtimes are broken down into grid construction and ray casting, both of which have been normalized using the values at $\alpha = 3$. The green curve shows the runtime given a balanced workload, meaning that equal time is spent on grid construction and ray casting, and the red curve in (c) shows a workload where 90% of the time is spent on grid construction, and 10% on ray casting.

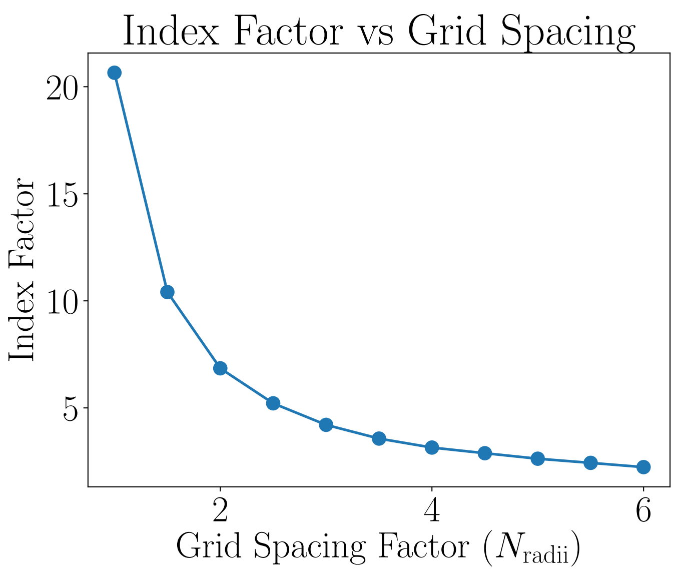


Figure 5.2: Indexing factor vs the grid spacing factor α.

The grid spacing that minimizes the overall runtime depends on tablet shape as well as workload balance, which in turn depends on the number of tablets and rays. However, grid spacing factors in the range $\alpha \in [2, 5]$ worked well for all cases shown here, and changes within this interval had a modest impact on the total runtime, with relative differences staying well within 25%. For this reason, and because it eliminates a variable, $\alpha = 3$ will be used for all runs in the following sections.

5.1.3 Sampling Convergence

Each ray provides a single sample for the coating variability estimate, which means the accurary and variance of the estimate will depend on the number of rays cast. Figure 5.3 shows how the convergence of $\text{CoV}_{\text{inter}}$, $\text{CoV}_{\text{intra,bulk}}$, $\text{CoV}_{\text{intra,agg}}$ and $\text{CoV}_{\text{intra,agg,sym}}$ depend on the number of rays cast per time step in the Round case, which has about 150 tablets in the spray zone and 3000 tablets in total, each divided into 2000 panels. The graphs show that both $\text{CoV}_{\text{inter}}$ and $\text{CoV}_{\text{intra,agg,sym}}$ do not benefit much from more than 1K rays per time step. This result for $\text{CoV}_{\text{inter}}$ is similar to that found by Toschkoff for a similar spray zone area and tablet size [11]. The non-symmetrical $\text{CoV}_{\text{intra,agg}}$ behaves just like its symmetrical counterpart, except that it lags behind by a factor of 8, requiring around 10K rays per time step to converge in this case. Compared to the others, $\text{CoV}_{\text{intra,bulk}}$ requires significantly more samples to converge, since its final value changes even when going from 100K to 1M rays per time step.

The reason the sample requirements differ by several orders of magnitude for the different coefficients is that they are estimating fundamentally different quantities. $\text{CoV}_{\text{inter}}$ only counts hits per tablet while ignoring the intra-tablet panels, so its ray requirement only depends on the amount of tablets in the spray zone. Similarly, the representative tablet approach of $\text{CoV}_{\text{intra,agg,sym}}$ means that its ray requirement only

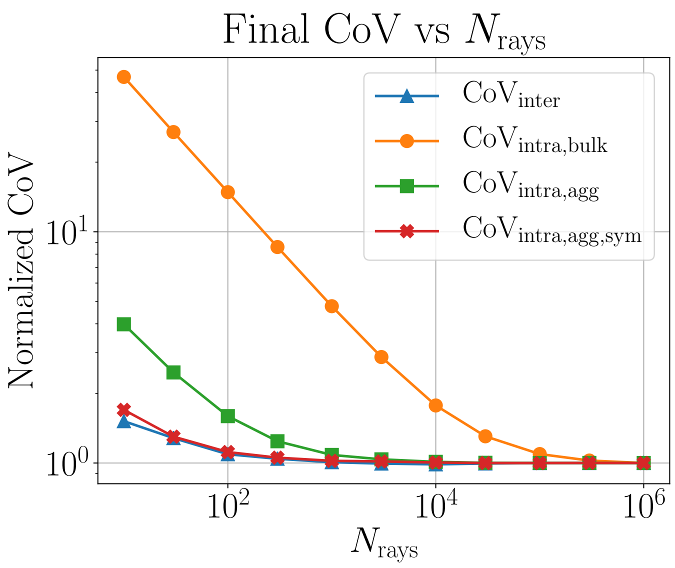
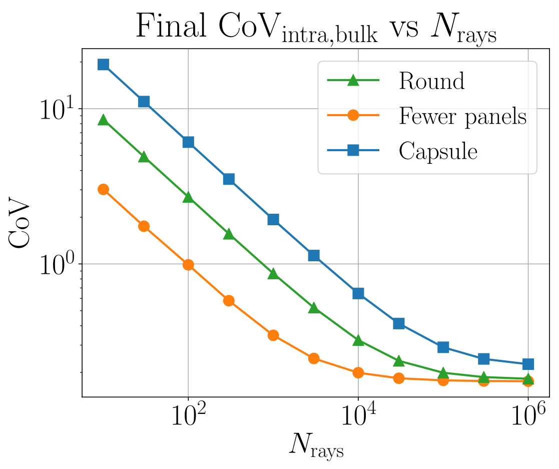


Figure 5.3: CoV at the end of the 180 second Round simulation as a function of the number of rays cast per time step, showing how many rays are required for the different coefficients to converge. The values have been normalized by the final value when 1M rays are cast.

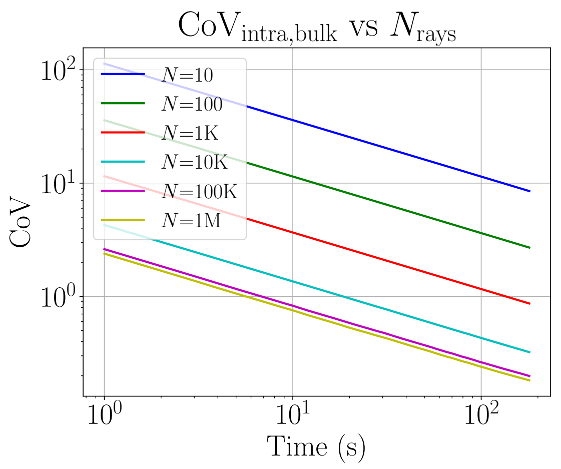
depends on the number of panels, and not on the number of tablets. $\text{CoV}_{\text{intra,bulk}}$ depends on both, or more specifically on the average number of unique panels in the spray zone. Figure 5.4a helps explain this dependency. It shows the ray requirement of $\text{CoV}_{\text{intra,bulk}}$ in three different cases, namely Round with 2000 panels, Round with 256 panels, and Capsule. Compared to the Round case with 2000 panels, in the Round case with 256 panels, the number of tablets remains the same while the number of panels is reduced, so the ray requirement drops because there are fewer unique panels in the spray zone. Conversely, in the Capsule case the number of panels remains the same while the amount of tablets in the spray increases, so the ray requirement goes up. Increasing the spray zone area would have a similar effect.

As shown, $\text{CoV}_{\text{intra,bulk}}$ requires significantly more samples than the other estimates. However, the linear nature of the graphs in Figure 5.4b hint that an extrapolation procedure, similar to inverse power law used to extrapolate $\text{CoV}_{\text{inter}}$ [1], could be applicable. In that case, it would be enough to cast a large number of rays only for a few seconds of simulation time, and thereby drastically reduce the total amount of rays that need to be cast. Note however that the same power law does not actually apply to $\text{CoV}_{\text{intra,bulk}}$ over long periods of time, because in general it will reach a non-zero asymptote [6]. However, the extrapolation could perhaps provide a rough estimate of how quickly the tablets in the coating process will reach a certain uniformity.

Sampling time is another factor to consider. Figure 5.5 shows how the convergence of $\text{CoV}_{\text{intra,agg}}$ and $\text{CoV}_{\text{intra,agg,sym}}$ depend on the length of the simulation when a large, fixed number of rays are cast. This gives an idea of how many independent coating trials are needed to get a stable estimate of the asymptotic intra-tablet coating variability. After only 10 seconds of simulation data, the symmetrical $\text{CoV}_{\text{intra,agg,sym}}$



(a) $\mathrm{CoV}_{\mathrm{intra,bulk}}$.



(b) $\mathrm{CoV}_{\mathrm{intra,bulk}}$.

Figure 5.4: Final values and convergence of $\mathrm{CoV}_{\mathrm{intra,bulk}}$ as a function of the number of rays cast per time step in the Round case.

has converged to within 10% of the final value, showing that most of the information about asymptotical intra-tablet coating variability can be extracted from simulations as short as 10 seconds. The non-symmetrical $\mathrm{CoV}_{\mathrm{intra,agg}}$ lags behind, but after about 80 seconds it becomes indistinguishable from $\mathrm{CoV}_{\mathrm{intra,agg,sym}}$.

Reasoning about the amount of tablets, panels and unique panels in the spray zone can help to roughly estimate the amount of rays required to get convergence in $\mathrm{CoV}_{\mathrm{inter}}$, $\mathrm{CoV}_{\mathrm{intra,agg,sym}}$ and $\mathrm{CoV}_{\mathrm{intra,bulk}}$ respectively. But it also depends on spray zone area, sample rate and average single visit residence time in the spray zone. Given a number of rays that is known to give convergence, scaling can be quite straightforward. For instance, if the total spray zone area doubles with all else equal, both $\mathrm{CoV}_{\mathrm{inter}}$ and $\mathrm{CoV}_{\mathrm{intra,bulk}}$ will require twice as many rays as they did previously, whereas the requirement of $\mathrm{CoV}_{\mathrm{intra,agg,sym}}$ will be largely unaffected. Similar rules of thumb can also be used to arrive at initial estimates of ray requirements. In the Round case above, there were around 150 tablets in the spray zone and the average single visit residence time was 5 time steps. The estimate of $\mathrm{CoV}_{\mathrm{inter}}$ converged with 1K rays per time step, which indicates that an average of about 30 rays per tablet and spray zone visit may be sufficient for this shape. Ultimately however, as shall be seen in the next section, it is not very costly to run a quick test on a small subset of the data to see how many rays are required to estimate a certain coefficient.

5.1.4 Runtimes

Benchmarks were run by ray casting the DEM data from the different simulations. To better explore the performance of the ray caster, the simulations were split into two different use cases. A grid spacing factor of 3 was used in both, as discussed in the grid spacing section, and the number of rays cast will be motivated below.

The first use case was to asses inter-tablet and asymptotical intra-tablet variability

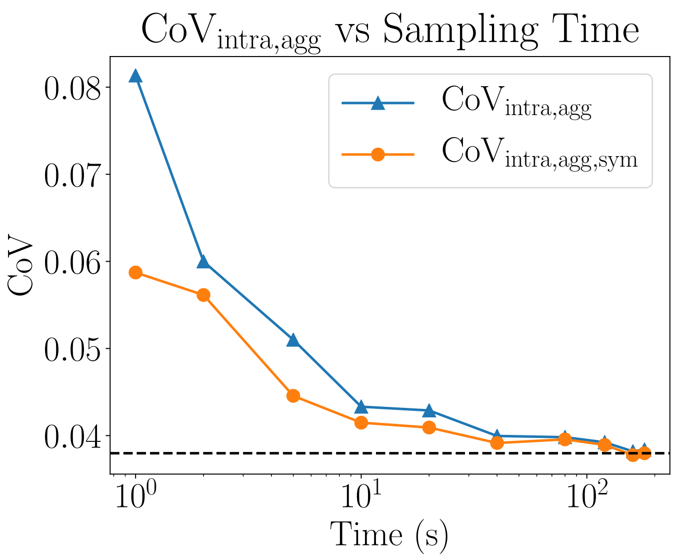


Figure 5.5: Estimate of the asymptotic $\mathrm{CoV}_{\mathrm{intra,agg}}$ and $\mathrm{CoV}_{\mathrm{intra,agg,sym}}$ for the Round shape as a function of the simulation length when a total of 100 million rays are cast. The dashed line shows the average of the final values.

for the small simulations of pharmaceutical shapes. This has a much lower ray requirement than estimating $\mathrm{CoV}_{\mathrm{intra,bulk}}$. In the Round case, $\mathrm{CoV}_{\mathrm{inter}}$ required 1K rays per time step to converge, but Capsule has about 5 times more tablets, so the requirement should be around 5K in that case. However, even with 5K rays, the symmetrical surface densities were still not quite as smooth as desired, so the amount of rays was doubled to 10K.

The second use case was to estimate $\mathrm{CoV}_{\mathrm{intra,bulk}}$ for the two larger simulations, $\mathrm{Round}_{\mathrm{medium}}$ and $\mathrm{Round}_{\mathrm{large}}$. Due to memory constraints, the number of panels was reduced to 256. For the Round case with 256 panels, shown in Figure 5.4a, 100K rays per time step was enough to get decent convergence. Since the number of panels is the same, the only difference that impacts the ray requirement of $\mathrm{CoV}_{\mathrm{intra,bulk}}$ is the spray zone area, which is roughly 5 and 10 times larger in $\mathrm{Round}_{\mathrm{medium}}$ and $\mathrm{Round}_{\mathrm{large}}$ respectively. Therefore, 500K and 1M rays were cast per time step in $\mathrm{Round}_{\mathrm{medium}}$ and $\mathrm{Round}_{\mathrm{large}}$ respectively.

Table 5.1 shows runtimes for processing the data from the different simulations. Using 4 threads on a conventional desktop computer, the wall-clock runtimes range from around 1 to 2 minutes for the smaller coaters, up to around 180 minutes for the largest one. The workload balance shifts as the size of the simulation increases: The ray casting runtime dominates for the smaller simulations, but a more even workload is seen for the larger ones. It is interesting to note however that the time spent per ray does not increase much from the small Round simulations to the larger ones, the biggest difference being from 0.37 µs for Round to 0.45 µs for $\mathrm{Round}_{\mathrm{medium}}$. The fact that the time spent per ray does not increase much with the size of the simulations shows the effectiveness of the acceleration structures in reducing the amount intersection checks.

The grid construction is a significant bottleneck for the largest simulation $\mathrm{Round}_{\mathrm{large}}$.

Table 5.1: Ray casting runtimes for 180 seconds of data from different simulations. All runs used a grid spacing factor of $\alpha = 3$. T_{wall} is the elapsed wall-clock runtime and T_{grid} and T_{ray} are single-thread estimates for grid construction and ray casting respectively based on averaging across the 4 threads. The 180 second runtimes of Round$_{\text{medium}}$ and Round$_{\text{large}}$ have been extrapolated from runtimes of processing 12 simulation data files, and the number of panels in those cases were reduced due to memory constraints.

Simulation	N_{tablet}	N_{panel}	N_{ray}	T_{wall}	T_{grid}	T_{ray}	$T_{\text{ray}}/N_{\text{ray}}$
Round*	3237	2024	90M	157.5 s	16.0 s	121.2 s	1.35 µs
Round	3237	2024	90M	43.8 s	4.4 s	33.6 s	0.37 µs
Round10EV	3270	2024	90M	49.4 s	4.5 s	39.2 s	0.44 µs
Oval	2973	2064	90M	53.6 s	4.0 s	44.2 s	0.49 µs
Caplet	4834	2000	90M	103.0 s	11.2 s	85.1 s	0.95 µs
Capsule	16186	2072	90M	121.5 s	39.4 s	64.5 s	0.72 µs
Round$_{\text{medium}}^*$	650K	256	4.5B	190.6 m	58.9 m	122.8 m	1.64 µs
Round$_{\text{medium}}$	650K	256	4.5B	64.5 m	17.6 m	33.6 m	0.45 µs
Round$_{\text{large}}^*$	2M	256	9.0B	509.4 m	264.4 m	220.7 m	1.47 µs
Round$_{\text{large}}$	2M	256	9.0B	179.4 m	96.6 m	61.8 m	0.41 µs

* Single-threaded.

The speed-up in grid construction when going from 1 to 4 threads in that case is not almost 4 like it is for Round and Round$_{\text{medium}}$, but is instead closer to 2.5. This is because the simulation data files are so large that the program hits the data read limit of the hard drive. This could however be remedied to some extent. The time to process files could be reduced by storing them in a binary format rather than text, and the file sizes could be significantly reduced by subsampling to only dump information about tablets near a priori known spray zone locations.

The increase in ray casting runtime in the Oval case compared to the Round case, despite the number of tablets decreasing, is partly due to the larger maximum aspect ratio of the Oval tablets. This lowers the superquadric to bounding sphere volume ratio and means that bounding spheres on average extend further than the superquadric in the ray direction. This in turn means that more intersection checks have to be performed before the ray casting algorithm can be safely terminated based on the sphere distance criterion. Ultimately this increases the average amount of intersection checks per ray as shown in Table 5.2. The table shows that from the Round to the Oval case, the average number of sphere intersection checks per ray doubles, and the average number of superquadric intersection checks goes from 1.14 to 1.42. Caplet and Capsule have even larger maximum aspect ratios than Oval and perform an average of 1.77 and 1.71 superquadric intersection checks per ray respectively. Without mailboxing these numbers grow to 2.58 and 2.59, which shows the importance of mailboxing.

The difference in ray casting runtimes between Caplet and Capsule is however not explained by maximum aspect ratio, but by shape parameters. The Caplet is the only shape with $n_2 = 2.5$, and this drives up the ray casting runtime considerably,

Table 5.2: Number of intersection checks per ray performed while processing the data from the different simulations. Round$_{\text{medium}}$ and Round$_{\text{large}}$ used the Round shape.

Simulation	I_{sphere}	I_{box}	I_{sq}
Round	14.21	1.22	1.14
Round10EV	14.91	1.24	1.11
Oval	30.62	2.08	1.42
Caplet	55.99	3.19	1.77
Capsule	54.20	3.42	1.71
Round$_{\text{medium}}$	58.99	1.20	1.13
Round$_{\text{large}}$	62.57	1.23	1.14

as the power functions can no longer be computed using multiplication. If possible, blockiness parameters should be rounded to integer values to reduce this cost. The time spent intersecting each bounding volume was measured, and it was found that sphere intersections on average take 30-60 clock cycles, box intersections 150-300 cycles and superquadrics 1200-3000 cycles. The longest average intersection cost for superquadrics was for the Caplet shape. This shows the value in minimizing the number of ray-superquadric intersection checks performed.

In conclusion, the runtime depends mostly on these key factors:

- **Intersection tolerance**: Smaller tolerance requires more Newton iterations and takes more time. The tolerance can be increased, or the intersection refinement step bypassed entirely if only inter-tablet variation is of interest, to dramatically speed up the ray casting.

- **Number of rays and tablets**: The time spent constructing the grid and ray casting grows roughly linearly with the number of tablets and rays respectively. However, the time spent per ray increases very little with the number of tablets, showing that the ray casting is scaling well to larger simulations.

- **Grid spacing**: The grid spacing can be tuned to trade off between grid construction and ray casting runtimes. A larger grid spacing makes for fewer grid cells and a quicker grid construction at the cost of increased ray casting runtime. However, the grid spacing was found to have a relatively modest impact on the final runtime, and a grid spacing factor of $\alpha = 3$ was found to work okay in all cases studied.

- **Shape parameters**: Shapes with larger maximum aspect ratios and thereby larger bounding spheres to particle volume ratios require more sphere intersection checks which increases the runtime. Additionally, computing large or non-integer exponents is more costly than small integers which can be computed using multiplication. Therefore, if the workload is heavy in ray casting, the blockiness parameters could if possible be rounded to nearby integer values to reduce this cost.

5.2 Coating Variability

In the following two sections, the coating variability statistics extracted from the two simulations sets are analyzed.

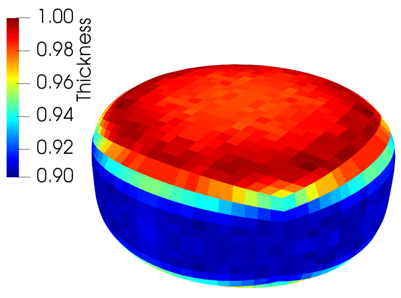
5.2.1 Pharmaceutical Shapes

Figure 5.6 shows surface densities of the four pharmaceutical tablet shapes that were considered. Shapes similar to the Round and Oval seen here were also studied by Freireich [12], and overall they report similar results. However note that the shapes are not equal, only similar, and they also used multi-sphere shape approximation which might affect the simulated tablet flow and thereby the results. Similarities between those results and the ones seen here are, more specifically, for the Round shape, they report a higher coating density on the cap than on the band, and a slightly thicker coating towards the edge of the cap, just as in Figure 5.6a. The Oval shape also shows a similar overall coating distribution, including a lower coating density in areas of low radius of curvature at the ends of the tablet. However, their Oval shape shows a varying density along the cap, with the highest density in the center. This difference is likely due to their Oval shape having a rounded cap whereas the cap in Figure 5.6b is basically flat which results in a more uniform coating. This difference in cap geometry means this comparison should be taken with a grain of salt, but the overall agreement still lends credibility to the approach used in this work.

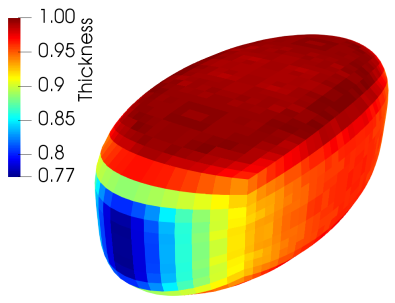
The predicted time for $\text{CoV}_{\text{inter}}$ of the four pharmaceutical tablets to hit 5% was around 60 minutes. The prediction is based on the theoretical $t^{-0.5}$ power law trend [1]. There is variation between the different shapes as seen in Figure 5.7a. The prediction for the Oval is 44 minutes, whereas that of the Caplet is 64 minutes. In terms of intra-tablet variability, Figure 5.7b shows that $\text{CoV}_{\text{intra,agg,sym}}$ is lowest for the Round shape and largest for the Caplet and Capsule shapes, which correlates well with the sphericity of the tablets as observed in previous work [18]. The graph also shows that it takes a longer time for $\text{CoV}_{\text{intra,agg}}$ to converge when the asymptotic value is lower, which should be taken into consideration when choosing the amount of rays to cast.

5.2.2 Blockiness Variations

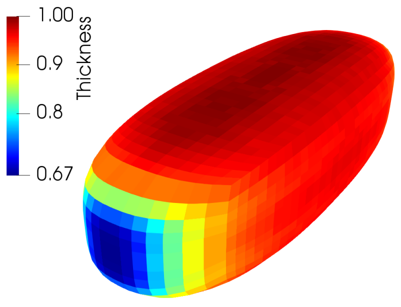
Figure 5.8 shows the effect of varying n_1 on the intra-tablet coating density. As the blockiness increases, the coating distribution shifts from the cap to the band, which in this case increases the coating uniformity since Round4 has a higher coating density on the cap than on the band. The shift is most notable for Round10 in Figure 5.8d where the ratio flips to the band having a higher coating density than the cap. There is little to no difference in inter-tablet variability between the Round variations, as can be seen in Figure 5.9a. However, as indicated by the densities, the intra-tablet coating uniformity increases with blockiness, and this is



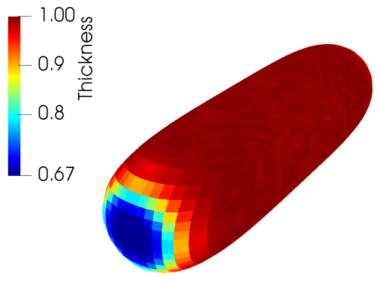
(a) Round.



(b) Oval.

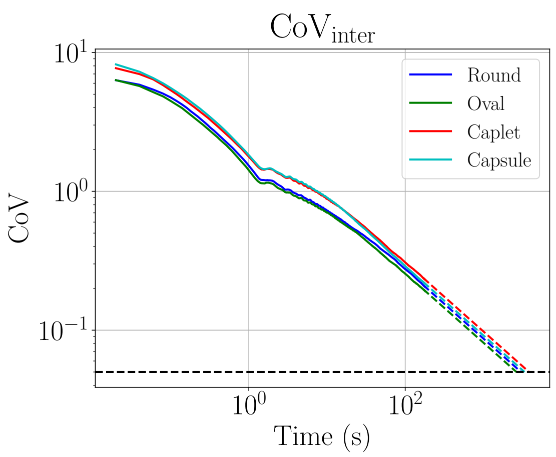


(c) Caplet.

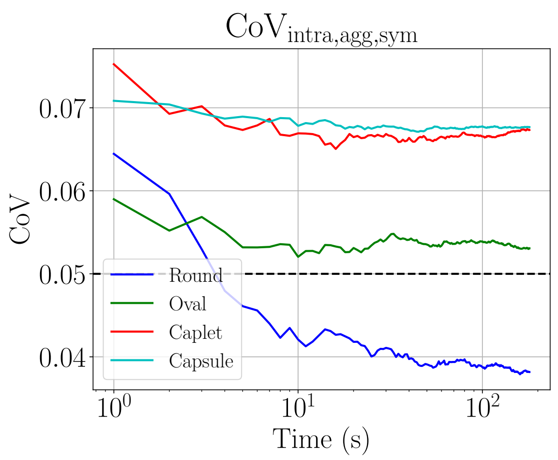


(d) Capsule.

Figure 5.6: Surface densities of four pharmaceutical tablet shapes. The densities have been symmetrically averaged and normalized by dividing by the maximum density.

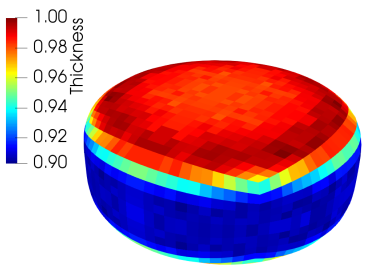


(a) $\mathrm{CoV}_{\mathrm{inter}}$.

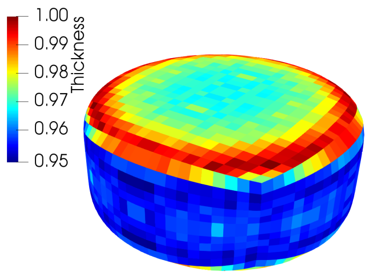


(b) $\mathrm{CoV}_{\mathrm{intra,agg,sym}}$.

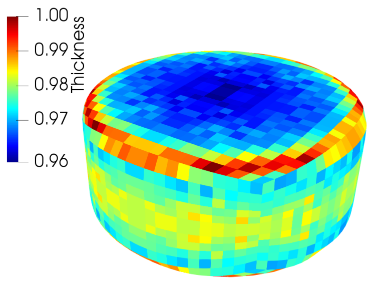
Figure 5.7: $\mathrm{CoV}_{\mathrm{inter}}$ and $\mathrm{CoV}_{\mathrm{intra,agg,sym}}$ for the four pharmaceutical tablet shapes. The projections of the $\mathrm{CoV}_{\mathrm{inter}}$ are based on a theoretical $t^{-0.5}$ fall-off [1].



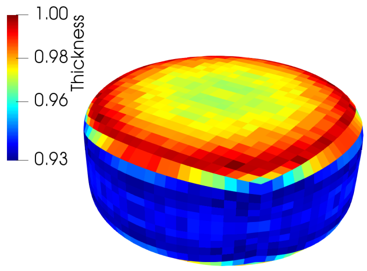
(a) Round4.



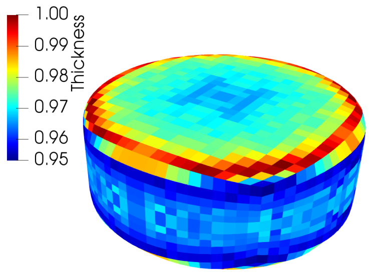
(b) Round6.



(d) Round10.



(c) Round6EV.



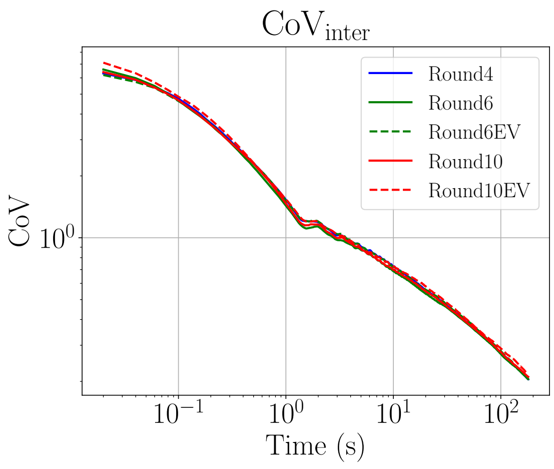
(e) Round10EV.

Figure 5.8: Surface coating densities of the Round variations. The densities have been symmetrically averaged and normalized by dividing by the maximum density.

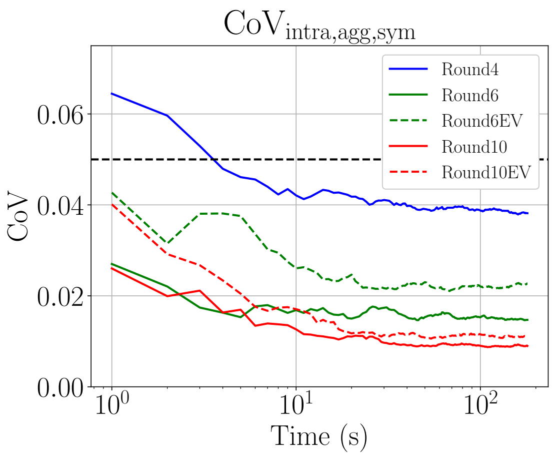
further confirmed by the asymptotic $\text{CoV}_{\text{intra,agg,sym}}$ predictions in Figure 5.9b. The $\text{CoV}_{\text{intra,agg,sym}}$ of Round4 and Round10 are predicted to reach 3.82% and 0.90% respectively.

The shift in coating distribution indicates that the preferred orientation of tablets in the spray zone is changing, and this is backed up by the orientation distribution in Figure 5.10a. The orientation of a tablet was measured by first detecting it in the spray zone through ray casting and then measuring the angle between the vector pointing from the tablet toward the spray and the vector orthogonal to the tablet cap. An angle of 0° therefore means that the cap is pointing straight towards the spray. The shift in orientation from Round4 to Round10 shows clearly that the Round10 tablets have a higher tendency to face their caps away from the spray. This shift also correlated with an increase in top rotational speed, as shown in Figure 5.10b, which indicates that tablets are flowing a bit more erratically. One possible explanation for this change in bulk flow behavior could be that the blockier particles in some sense have smaller effective aspect ratio since their bands are slightly flatter and taller. This would then also be true for both Round6EV and Round10EV despite them having a lower thickness c than Round4 and thereby a lower aspect ratio. Further work will be necessary to understand the connection between increased blockiness, tablet flow and resulting coating variability.

One downside of the cube parametrization is that it creates cubical artifacts on the surface, as can been seen in the densities of Figure 5.8. The boundary between the cap and the band on the Round tablet is axisymmetrical, but the cube parametriza-

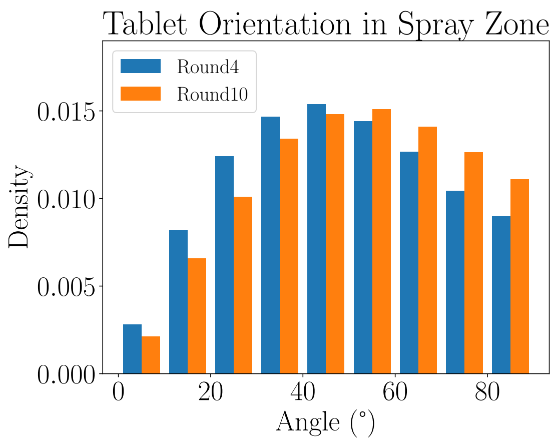


(a) $\mathrm{CoV}_{\mathrm{inter}}$.

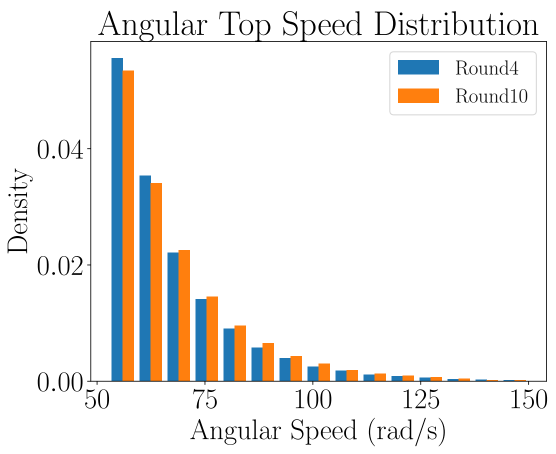


(b) $\mathrm{CoV}_{\mathrm{intra,agg,sym}}$.

Figure 5.9: $\mathrm{CoV}_{\mathrm{inter}}$ and $\mathrm{CoV}_{\mathrm{intra,agg,sym}}$ for Round blockiness variation shapes. The projections of the $\mathrm{CoV}_{\mathrm{inter}}$ are based on a theoretical $t^{-0.5}$ fall-off [1]. The intra-tablet coating uniformity increases with the blockiness. The asymptotic prediction for Round4 is 3.82% while that of Round10 is 0.90%.



(a) Orientation.



(b) Top angular speed.

Figure 5.10: Distributions of tablet orientation and rotational speeds from the Round4 and Round10 simulations. The orientations are sampled from the spray zone, and the top angular speeds are those greater than the corresponding 95th percentile in the Round4 simulation. The shift in both histograms show that the Round10 tablets are facing their cap towards the spray less and rotating faster in the free surface than the Round4 tablets.

tion obscures this symmetry. It also creates the illusion of a smoother transition in density between cap and band, like in Figure 5.8e where a stripe of orange panels can be seen between the red ring of the cap and the blue band. On other parts of the same tablet there are neighboring red and blue panels, indicating that this density transition is likely much sharper than the orange panels make it seem. This effect not only makes it more difficult to interpret the surface density, but it will also result in a smaller estimate of $\mathrm{CoV}_{\mathrm{intra,agg}}$, especially when using fewer panels. This needs to be kept in mind when comparing variability between shape representations that use different numbers of panels, both in terms of $\mathrm{CoV}_{\mathrm{intra,agg}}$ and in terms of the appearance of the surface density.

6
Conclusion

An efficient ray casting algorithm to model the spray in drum coating processes and a corresponding C++ implementation were developed and described in detail. The method was able to generate inter- and intra-tablet coating predictions by post-processing data from DEM simulations where tablet shapes were modeled using superquadrics. To achieve good performance, a spatial subdivision index and particle bounding volumes were used to reduce the number of ray intersection checks performed, and the convexity of the superquadrics was exploited to efficiently compute the intersection points. The intersection accuracy was shown to only be limited by the tolerance, and a value of a thousandth of the smallest tablet dimension was found to give negligable errors practice.

Simulations of varying size and tablet shape were benchmarked, and the largest one, containing 2 million tablets, was processed in less than 3 hours on a conventional desktop computer. The multi-threaded implementation scaled well and significantly reduced the runtime for all simulations, but when processing the largest one, the size and format of the simulation data files were found to be a bottleneck for the index construction. Switching to a binary file format and subsampling the exported data could alleviate this problem.

An invertible cube projection was used to divide the superquadric surface into a uniform set of panels that were used to compute and visualize intra-tablet coating variability. While it had many advantages over the other parametrizations considered, the lack of axisymmetry caused cubical artifacts on the surface. Further, for certain tablet shapes, the visualizations revealed that some panels straddled the boundary between cap and band. This will reduce the estimated coefficient of intra-tablet variability, especially when using a small number of panels.

Finally, the developed spray method was applied to two sets of tablet shapes. In the first set of common pharmaceutical shapes, the coating variability results showed overall good agreement with previous work based on multi-spheres. The second set contained round tablets of varying blockiness and was meant to study the impact of blockiness on coating outcome, an analysis that would have been more difficult to do using multi-spheres. The results showed that blockiness had a significant impact on the intra-tablet coating, even when tablet thickness was reduced to keep an equal volume. The coating density was shown to shift from cap to band, and an analysis of tablet orientation in the spray zone confirmed that the blockier tablets had a higher tendency to face their caps away from the spray. However, further work will be necessary to explore the cause of this change in bulk flow behavior.

Bibliography

[1] G. Toschkoff and J. G. Khinast, “Mathematical modeling of the coating process,” *International Journal of Pharmaceutics*, vol. 457, no. 2, pp. 407–422, 2013.

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

[3] W. R. Ketterhagen, M. T. am Ende, and B. C. Hancock, “Process modeling in the pharmaceutical industry using the discrete element method,” *Journal of Pharmaceutical Sciences*, vol. 98, no. 2, pp. 442–470, 2009.

[4] M. Alizadeh Behjani, A. Hassanpour, M. Martín, and m. pasha, *Introduction to Software for Chemical Engineers, Second Edition.* USA: CRC Press, Inc., 08 2019.

[5] R. Cabiscol, J. H. Finke, and A. Kwade, “Calibration and interpretation of dem parameters for simulations of cylindrical tablets with multi-sphere approach,” *Powder Technology*, vol. 327, pp. 232 – 245, 2018.

[6] B. Freireich, W. R. Ketterhagen, and C. Wassgren, “Intra-tablet coating variability for several pharmaceutical tablet shapes,” *Chemical Engineering Science*, vol. 66, no. 12, pp. 2535–2544, 2011.

[7] B. Soltanbeigi, A. Podlozhnyuk, S.-A. Papanicolopulos, C. Kloss, S. Pirker, and J. Y. Ooi, “Dem study of mechanical characteristics of multi-spherical and superquadric particles at micro and macro scales,” *Powder Technology*, vol. 329, no. nil, pp. 288–303, 2018.

[8] A. Podlozhnyuk, S. Pirker, and C. Kloss, “Efficient implementation of superquadric particles in discrete element method within an open-source framework,” *Computational Particle Mechanics*, vol. 4, pp. 101–118, Jan 2017.

[9] C. Kloss, C. Goniva, A. Hager, S. Amberger, and S. Pirker, “Models, algorithms and validation for opensource dem and cfd-dem,” *Progress in Computational Fluid Dynamics*, vol. 12, pp. 140 – 152, 06 2012.

[10] L. Fries, S. Antonyuk, S. Heinrich, and S. Palzer, “Dem–cfd modeling of a fluidized bed spray granulator,” *Chemical Engineering Science*, vol. 66, no. 11, pp. 2340 – 2355, 2011.

[11] G. Toschkoff, S. Just, A. Funke, D. Djuric, K. Knop, P. Kleinebudde, G. Scharrer, and J. G. Khinast, “Spray models for discrete element simulations of parti-

cle coating processes," *Chemical Engineering Science*, vol. 101, no. nil, pp. 603–614, 2013.

[12] B. Freireich, R. Kumar, W. Ketterhagen, K. Su, C. Wassgren, and J. A. Zeitler, "Comparisons of intra-tablet coating variability using dem simulations, asymptotic limit models, and experiments," *Chemical Engineering Science*, vol. 131, no. nil, pp. 197–212, 2015.

[13] C. Pei and J. A. Elliott, "Asymptotic limits on tablet coating variability based on cap-to-band thickness distributions: A discrete element model (dem) study," *Chemical Engineering Science*, vol. 172, pp. 286 – 296, 2017.

[14] A. H. Barr, "Superquadrics and angle-preserving transformations," *IEEE Computer Graphics and Applications*, vol. 1, p. 11–23, Jan. 1981.

[15] A. Jaklic, A. Leonardis, and F. Solina, *Segmentation and Recovery of Superquadrics*, vol. 20 of *Computational imaging and vision*. Dordrecth: Kluwer, 2000. ISBN 0-7923-6601-8.

[16] A. S. Glassner, ed., *An Introduction to Ray Tracing*. GBR: Academic Press Ltd., 1989.

[17] J. Zhu, S. Zhao, Y. Ye, and G. Wang, "Computed tomography simulation with superquadrics," *Medical Physics*, vol. 32, no. 10, pp. 3136–3143, 2005.

[18] W. R. Ketterhagen, "Modeling the motion and orientation of various pharmaceutical tablet shapes in a film coating pan using dem," *International Journal of Pharmaceutics*, vol. 409, no. 1-2, pp. 137–149, 2011.

[19] I. Wald and V. Havran, "On building fast kd-trees for ray tracing, and on doing that in o(n log n)," in *2006 IEEE Symposium on Interactive Ray Tracing*, pp. 61–69, 2006.

[20] A. Fujimoto, T. Tanaka, and K. Iwata, "Arts: Accelerated ray-tracing system," *IEEE Computer Graphics and Applications*, vol. 6, no. 4, pp. 16–26, 1986.

[21] T. L. Kay and J. T. Kajiya, "Ray tracing complex scenes," *SIGGRAPH Comput. Graph.*, vol. 20, p. 269–278, Aug. 1986.

[22] D. Kalra and A. H. Barr, "Guaranteed ray intersections with implicit surfaces," *SIGGRAPH Comput. Graph.*, vol. 23, p. 297–306, July 1989.

[23] D. P. Mitchell, "Robust ray intersection with interval arithmetic," in *Proceedings on Graphics Interface '90*, (CAN), p. 68–74, Canadian Information Processing Society, 1990.

[24] M. Pilu and R. B. Fisher, "Equal-distance sampling of superellipse models," in *BMVC*, 08 1995.

[25] C. Ronchi, R. Iacono, and P. Paolucci, "The "cubed sphere": A new method for the solution of partial differential equations in spherical geometry," *Journal of Computational Physics*, vol. 124, no. 1, pp. 93 – 114, 1996.

[26] J. P. Ahrens, B. Geveci, and C. C. W. Law, "Paraview: An end-user tool for large-data visualization," in *The Visualization Handbook*, 2005.