Clinker Simulation

Master Thesis

Author(s):
Nikolakopoulos, Ilias

Publication date:
2014

Permanent link:
https://doi.org/10.3929/ethz-a-010294753

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

IfB
Institut für Baustoffe
Institute for Building Materials

Ilias Nikolakopoulos

Clinker Simulation

Master's Thesis
Computational Science and Engineering (CSE)

Group of Computational Physics for Engineering Materials (comphys)
Institute for Building Materials (IfB)
Swiss Federal Institute of Technology (ETH Zürich)

Supervisor

Prof. Dr. Hans J. Herrmann

Advisors

Dr. Falk Wittel
Dr. Miller Mendoza Jimenez

April 3, 2013

Abstract

A geometrical method for simulating Clinker is developed. The Clinker is treated as a granular system composed of single crystals (grains) of its major phase, Alite. The focus is on reproducing the primary constituent's volume-fraction and, roughly, the grain size distribution. The grains are considered to be convex polyhedra. Relevant statistical microstructural information is extracted from 2D micrographs and predefined grain shapes are used, estimated by directional cleavage energies. Two stages are involved: (1) Alite grains are placed without overlaps within a spherical volume and packed to a sufficiently high volume fraction and (2) the volume fraction is "boosted" by either compaction or expansion of the grains. Simulations show that this procedure results in a realistic volume fraction for Alite and seems promising in terms of easily incorporating other convex phases. The resulting clinker sample is evaluated by visual inspection of 2D slices of the clinker sample and a final size distribution check.

Contents

List of Figures

Chapter 1

Introduction

Concrete is one of the most common materials of everyday life, although rarely noticed due to its abundancy in the modern world. It is composed of sand and aggregates "glued" together by an appropriate hydraulic binder, cement. Cement (or Portland cement, the name coined for the most typical variant) is manufactured via an energetically-intensive procedure, during which 80% minerals and 20% clays are co-grinded, calcined and burnt in a kiln, delivering a quenched (mainly) four-phase material, known as Clinker. Its four major phases are known as Alite, Belite, Aluminate and Ferrite [1], the first two of which are crystalline with the rest being amorphous. The Clinker is grinded down to a powder of certain specific surface to yield the final product that constitutes Portland cement. Several improvements have been conceived [2] in order to boost the efficiency of the aforementioned manufacturing process. In particular, fragmentation studies have indicated, that a potential refinement of the comminution stage could substantially reduce the energy consumption of the overall cement production.

In past research, statistical models for brittle fragmentation and corresponding simulation schemes have been developed, as presented in [3] and references therein. As far as the models are concerned, micro-mechanisms related to propagating crack merging, crack instability & branching, fragment size distribution, damage-fragmentation transition, and criticality probing have been systematically investigated. Simulations based on Lennard-Jones (MD) systems, continuum-, elastic element-, beam- and lattice models have, on their part, reproduced quite nicely the observed behaviour. Most materials usually include more than one phases of single crystals (grains), however for the sake of simplicity and to increase computational efficiency, many simulations only address single phase material fragmentation.

This project addresses the issue of modelling a more realistic multi-phase Clinker, meant to assist ongoing fragmentation studies. Although several models for simulating microstructures are available, spanning from microscale spin models to purely geometrical ones (e.g. based on Voronoi tesselations), this work deems expedient to model Clinker by directly incorporating microstructural information from micrographic data and atomistic simulations [4]. The work is arranged as follows: Chapter 2 discusses briefly the Clinker's main phases, their, in this context, most relevant statistical properties and how these can be derived from 2D micrographs; Chapter 3 provides a short overview of some of the possible modelling approaches before describing in detail the one implemented in this project; Chapter 4 reports on the simulation results and compares to available data and, finally, Chapter 5 draws conclusions, accompanied by an outlook to potential improvements and extensions of the work.

Chapter 2

The microstructure of Clinker

Simulations or even statistical modelling of fragmentation, this being instantaneous or continuous, blast- or impact-induced, yield meaningful macroscopic results in the high energy regime, even when the particular shape of the grains composing the system under loading is not taken into account in all detail. However, in the low energy regime the microstructure is expected to have a significant impact on the evolution of the fragmentation process. There is, therefore, room for possible improvements on these computational models, namely the incorporation of more microstructural information. In this Chapter the microstructure of Clinker is briefly presented, with the attention being focused on the four main phases and in more detail on the two of particular interest for this work: Alite and Belite.

In the first Section the general properties of the phases are described as well as their role in cement, in the second Section the estimation of Miller indices of a possible equilibrium shape for Alite [4] from directional cleavage energies is shortly described and in the final section a grain-size distribution for Alite is estimated based on a single 2D micrograph. Everything done for Alite can be adopted for incorporating Belite (or any other phase composed of convex grains, for that matter) in the model, and is, for this reason, omitted.

2.1 The four main phases, properties and volume fractions

By heating limestone, clay and potentially other admixtures to temperatures up to 1450°, reached in a cement kiln, partial fusion results in a nodule known as Clinker, which upon grinding yields the Portland cement [1]. The Clinker's four major phases are Alite, Belite, Aluminate and Ferrite and their corresponding chemical notation and volume fractions are summarized in Table 2.1.

The most important of those phases, in terms of cement hardening, are the first two. Moreover, they are also the only two crystalline ones to be incorporated in a geometric treatment of the

Phase	Cement Chemist Notation	Volume fraction
Alite	C_3S	50-70%
Belite	C_2S	15-30%
Aluminate	C_3A	5-10%
Ferrite	C_4AF	5-15%

Table 2.1: The four major phases in Clinker.

Clinker's microstructure simulation. Results from their molecular modelling, as done in [4], are shown in Fig. 2.1, along with respective crystal structure information. A microstructural

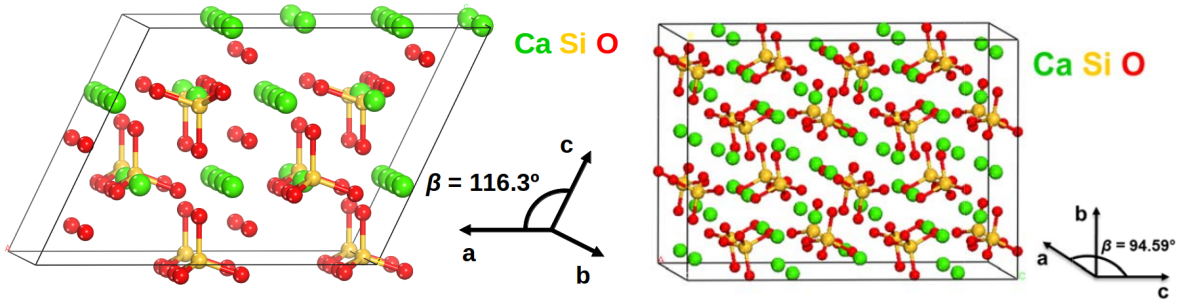


Figure 2.1: C_3S and C_2S molecular models (super cell dimension: 1×2×1 and 2×2×2 respectively) [4]. The cell parameters of C_3S, a = 1.2235, $b = 0.7073$, $c = 0.9298$ nm and $\beta = 116.31°$ can be used [5] to obtain the shape of Fig. 2.2 in Cartesian coordinates.

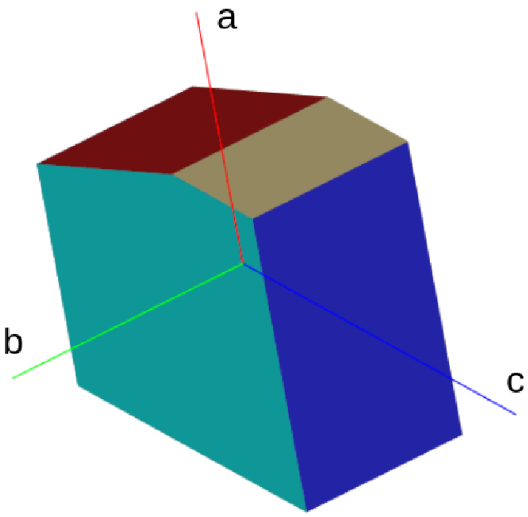
picture is given in Fig. 2.5 and 2.6. According to that, the Alite grains exhibit sharp edges and angles and appear to be in touch at distinct grain boundaries. In addition, their 2D profiles display a not particularly complex shape, thus making its approximation by a simple convex polyhedron a plausible approach. The Belite grain profiles are also clearly formed and well separated from the Alite. In contrast to Alite, though, they possess no sharp angles and in some cases they appear to be non-convex. Their roundish shape makes them more complex to geometrically approximate. The Aluminate is observed as an amorphous phase of rod-like domains. Its very complex and irregular shape, alike Ferrite, shall be excluded from this first approach to Clinker simulation.

2.2 Estimation of Alite's Miller indices from directional cleavage energies

An equilibrium shape of Alite is one that exhibits minimal surface energy. A face of this shape contributes to minimizing the energy, if cleaving parallel to it requires minimal energy w.r.t. other cleavage plane orientations. The energy required for cleaving a material along some oriented plane is directly related to the energy of the atomic bonds this plane intersects. Therefore, atomistic simulations, which attempt to cleave the material by means of several differently oriented planes, can determine possible face orientations that minimize the surface energy. Such simulations have been performed [4] and the results were used to estimate a possible equilibrium shape for Alite. In Fig. 2.2 the Miller indices of such faces are shown, accompanied by the corresponding cleavage energies. A subset of those indices, which give minimal cleavage energy, was chosen to yield the shape depicted in the same figure. However, this Alite shape - actually its transformation to Cartesian coordinates - does not seem to be in agreement with the micrographic data available and only some of its main features (e.g. number of faces) will be taken into account in this work.

2.3 Recovering Alite's grain size distribution from a 2d micrograph

The grain size distribution of a crystalline phase (at least a plausible estimation of it) is a prerequisite for any microstructural simulation. It can be used either to initialize a representative



Miller Indices	Cleavage Energy (mJ/m2)
(1 0 0), (2 0 0)	1534 ± 56
(3 0 0), (-3 0 0)	1590 ± 31
(8 0 0), (-8 0 0)	1401 ± 35
(0 1 0), (0 2 0)	2157 ± 120
(0 4 0), (0 -4 0)	1324 ± 25
(0 0 1) ~ (0 0 3) ~ (0 0 -3)	1335 ± 25
(0 0 2)	1350 ± 31
(0 0 8) ~ (0 0 -8)	1436 ± 30
(10 0 -3)	1385 ± 38

Figure 2.2: C_3S equilibrium shape based on the depicted directional cleavage energies [4]. The seven appearing faces correspond to a set of indices of the lowest cleavage energies.

population of grains, or to perform a verification test, i.e. have a realistic quantity to compare the resulting microstructure with. Usually, such statistical microstructural information is contained in two-dimensional micrographs, obtained by means of some tomographic technique. It is assumed, that the spatial resolution of a micrograph allows for a reliable measurement of relevant sizes of the particles appearing in it. However, one faces the challenge of drawing conclusions for 3D quantities, based on 2D cross-sections. Such problems are addressed via stereological methods.

In this work the grain size distribution of Alite is measured from a single micrograph. To proceed, a valid assumption on the shape of a typical Alite grain is required. This is, however, an irregular convex polyhedron, which renders most of the relatively simple stereological techniques impractical. For this reason, the assumption is made, that the grains are spherical and the problem reduces to determining the grain size distribution of a polydisperse system (same shape, different size) of spherical particles. By size, the max caliper diameter is meant, which for spheres reduces to the diameter.

The approach presented here was adopted from [6] and a more general discussion can be found in [7, 8]. A test case is first examined (Fig. 2.3). The sphere-sizes and the 2d profile-sizes are considered distributed into 4 bins. As shown in the first equation of Fig. 2.3, the probability of a sphere (green) in the 4-th bin of sphere-sizes, $N_A(4)$, to give a circular profile with size within the 4-th bin of profile-sizes (blue), is given by K_{44} (red). That is, a large profile size can only stem from the intersection of a large sphere. On the other hand, as depicted in the 2nd equation of Fig. 2.3, a profile of very small size could be coming from spheres of all sizes; most likely from one of the same size, less likely from larger ones and least likely from the largest (since this would mean that an $N_V(4)$ sphere was intersected close to a pole). Applying this consideration to all possible binned profiles, the system of the last row of Fig. 2.3 is obtained. Note that all columns sum up to a fully red sphere, i.e. they are normalized. For spheres, the

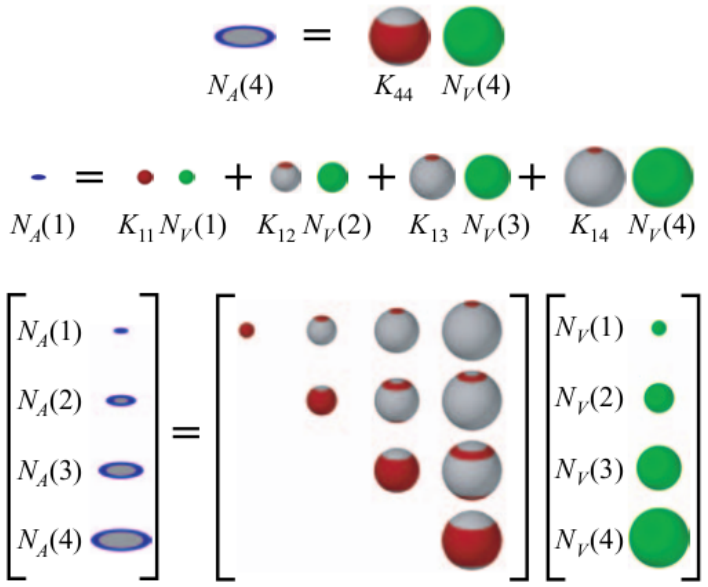


Figure 2.3: Equations relating the 2D circular profile densities (i.e. number of profiles per area, with size within each bin) with the 3D spherical densities (i.e. number of spheres per volume, with size within each bin). The circular profiles are the ones observed in a micrograph, while the green ones are unknown. The coefficients K relate those two densities and are known for special cases, such as spherical systems. In the first row a large profile is associated with the largest sphere, in the 2nd, a small profile could result from various weighted scenarios and in the 3d row, all equations are gathered in a system. Fig. from [6].

matrix K is known to be upper triangular with entries

$$\mathrm{K}_{ij} = \frac{1}{n}\left(\sqrt{j^2-(i-1)^2} - \sqrt{j^2-i^2}\right), \quad j \geq i$$

n being the number of bins. With K known and the profile distribution obtained from the micrograph, the 3D size distribution can be computed by

$$\mathrm{N}_V = \frac{1}{d_{max}}\mathrm{K}^{-1}\mathrm{N}_A$$

where d_{max} is the maximum size for profiles and spheres, assumed to be the same for both. It can be set to one by scaling N_A. Applying this technique to the micrograph of Fig. 2.7, the distribution of Fig. 2.4 is obtained. It should be noted, that the entries of the matrix K can be readily computed only for special cases. However, as previously described, these entries represent the probabilities that a known 3D shape has given a certain 2D profile after it was intersected with a random plane. Therefore, if the shape of the system is known and constant, the corresponding matrix K (aka the kernel) can be estimated by simulation [9, 8]: one can create random planes, intersect them with a shape of unitary size and then measure the size of the profile-sizes. This is not completely trivial, since the whole idea is based on the randomness of the plane selection; the planes should be able to produce all possible intersections, that could occur in practice, with the same probability.

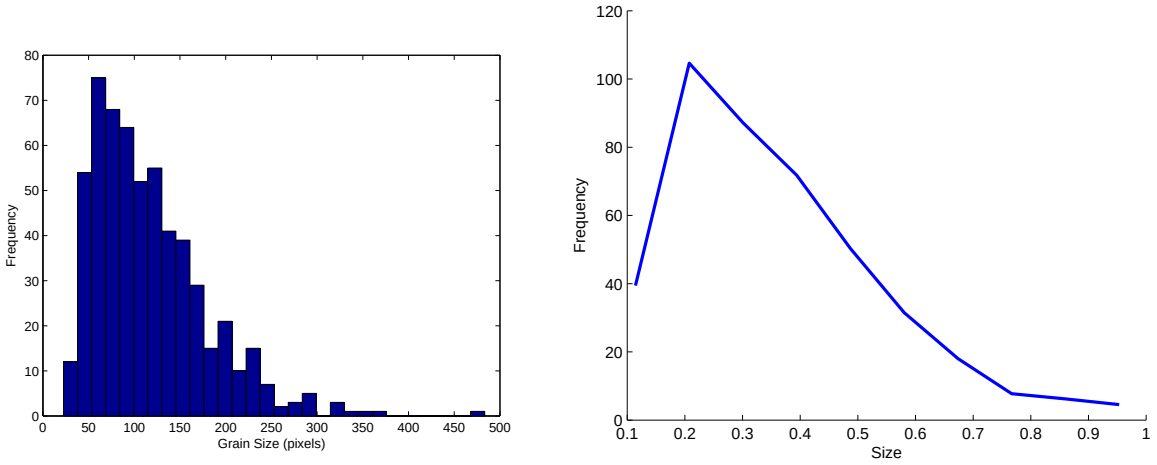


Figure 2.4: *Left*: The 2D grain size distribution of the Alite grains for Fig. 2.7 and *Right*: The 3D distribution obtained by means of the stereological equation under the assumption that the Alite phase is a polydisperse spherical system. Only the shapes of the two distributions are relevant.

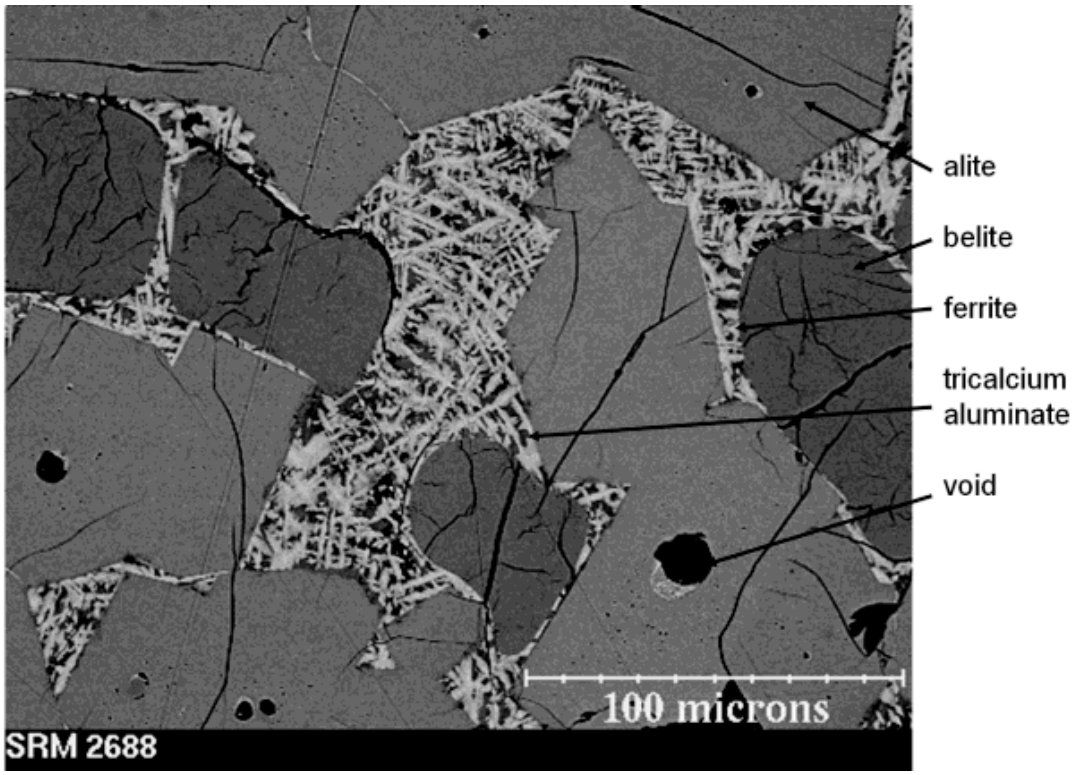


Figure 2.5: Planar section of a Clinker sample. 2D profiles of the four main phases can be readily identified as well as their characteristics. Alite is a sharp convex grain, Belite a roundish, partly non-convex one and the two amorphous phases, Aluminate & Ferrite almost fill the inserstitial space with irregular formations. Image from [10].

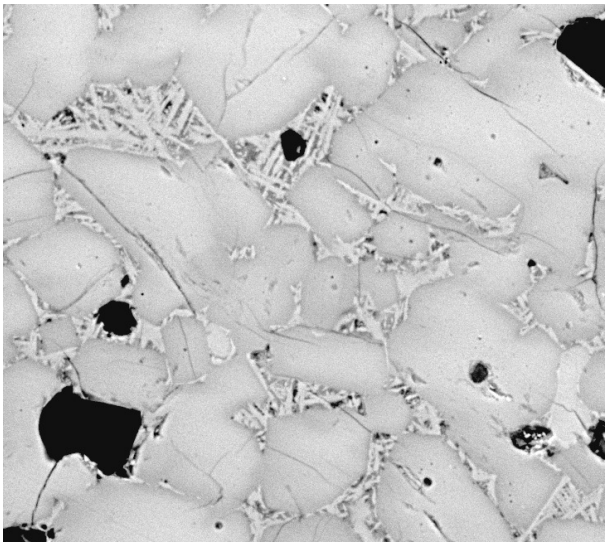


Figure 2.6: Micrograph of 2D Portland Clinker. As in Fig. 2.5, the major phases can be identified and, additionaly, a better idea about their relative arrangement and their volume fraction can be obtained. In particular, Alite clearly covers a quite high fraction of the space, although in this figure Belite can not be reliably individualized. Image courtesy of Jan Bisschop.

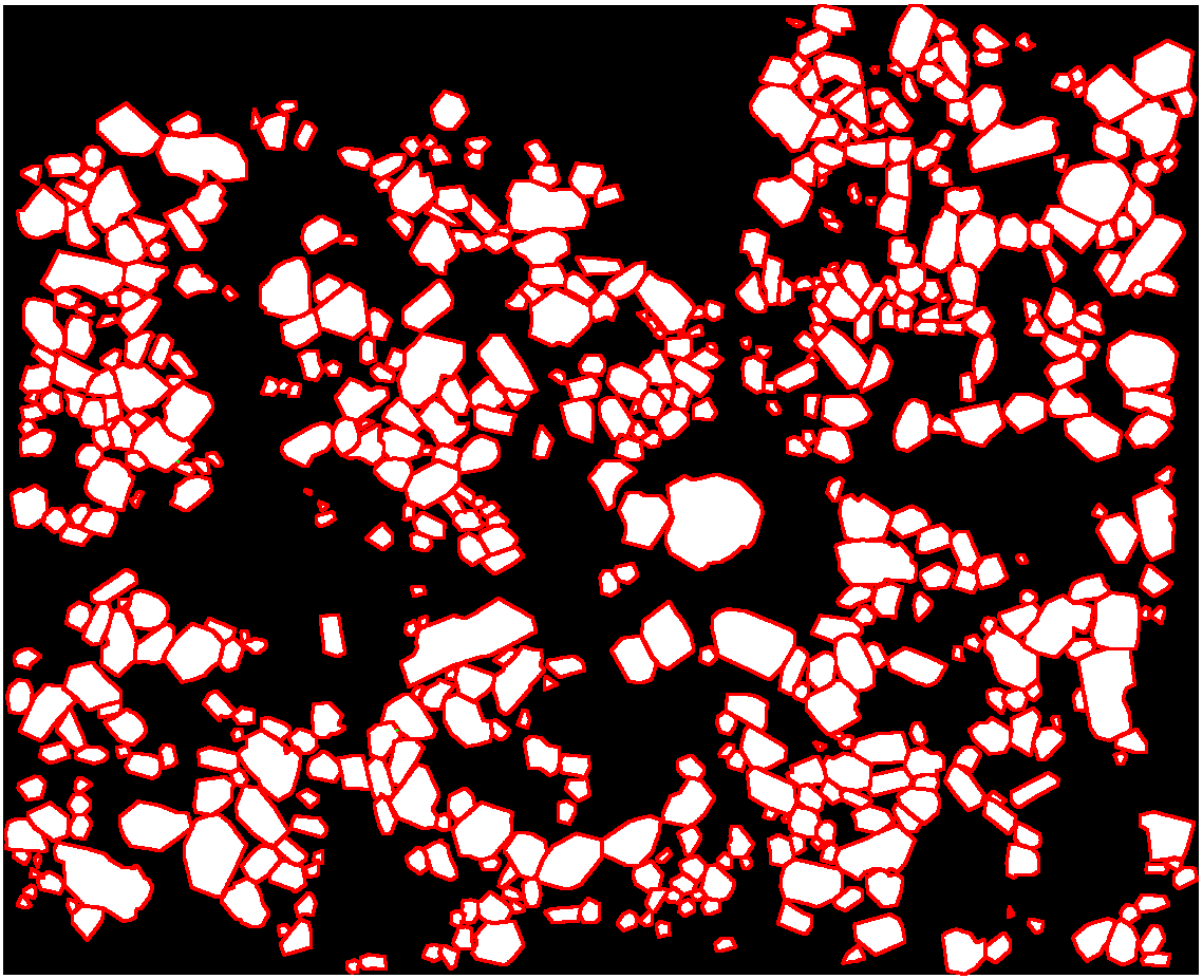


Figure 2.7: The boundaries of the planar sections of Alite grains are tracked on a 2D micrograph. A Clinker micrograph is fist converted into a black-and-white image and subsequently a boundary tracking method [11] is used to detect the boundary pixels (red). It should be pointed out that the density of planar sections of grains is not indicative of the Alite's volume fraction in Clinker; some (few) objects were considered artifacts of the overall image processing and removed (i.e. it was unclear whether they were, in fact, two or more adjacent grains and if so, how those should be separated). Evidently, even after the preprocessing, not all shapes are convex and some of them could again be several, incorrectly grouped together. Further refinement of the image processing procedure, however, is not deemed important, since the objective of this work is to qualitatively investigate a method not expected to be particularly sensitive to small changes in the grain size distribution. Original image courtesy of Jan Bisschop.

Chapter 3

Simulating Clinker as a multiphase granular system

Most engineering materials feature, in general, multiphase structures, each composed of distinct domains of single phase material, e.g. single crystals or grains. For the following discussion, grain will mean a single constituent of a phase, i.e. a single crystal, in case of crystalline phases or a spatially distinct domain of single phase, in case of amorphous phases. The characteristics of these phases as well as those of each of their constituents largely determine the macroscopic properties a material exhibits. Such characteristics could be: the volume fraction of each phase, the relative arrangement of the phases, the grain size distribution of the grains comprising a crystalline phase, their shape and orientation. In this respect, any model attempting to simulate such a material is evaluated by its capability to realistically reproduce as many of the aforementioned properties as possible.

This Chapter is dedicated to the modelling possibilities for the simulation of Clinker. In the first Section, various models are described, that could potentially serve as a viable solution to the problem at hand. After clarifying why those models where not opted for, the second Section discusses in detail the method used in this work for modelling Clinker or other multiphase materials. The two basic stages of the procedure are demonstrated through experimentation on small test systems, which are easier to visually inspect. Finally, some of the algorithms involved are briefly described.

3.1 Computer models for mesoscale microstructure simulations

Possible approaches to the problem of mesoscale microstructure modelling adhere, naturally, to one of the following perspectives: one that focuses on the physical process behind the growth or, more generally, formation of the material under investigation, and one that focuses on its final state (geometry and statistical characteristics). In the former case, one is interested in the physical laws, which govern the evolution of an original configuration towards the final microstructure. In the latter, the physics behind a material's formation and the formation itself don't matter as far as one can persuasively reconstruct the finally appearing microstructure. For instance, studying the driving micro-mechanisms of grain-growth, e.g. the curvature-driven boundary migration, requires that one allows for an initial configuration to evolve under certain physical rules, which translate accordingly within the model's framework, and observes several snapshots of this evolution. Data corresponding to what would be the final snapshot of such an evolutionary scheme might be, however, directly available from X-ray tomography (Fig. 2.7),

thus calling for a more immediate approach. Indeed, in cases where one is interested only in the final microstructure of material confined in a certain volume, as it happens in fragmentation simulations, it seems much more straightforward to mimic the structure by putting together geometrical objects of certain statistical characteristics (e.g. grain size distribution). A final note to make the distinction between the two views more clear is that in the first case, every snapshot of the evolving system should be a possible, naturally occurring situation, while in the second, one tries to "reverse engineer" the last snapshot alone. The reason why a growth view is also relevant to the simulation of Clinker is because, in principle, Clinker is formed after grain growth following sintering.

With these introductory comments in mind, the following paragraphs describe computational models related to *microstructure evolution* as well as *geometrical polycrystalline modelling* corresponding to the two discussed approaches, respectively. Some of the models, mostly used in grain growth studies, are conventionally classified as *sharp-interface* or *diffuse-interface*. This distinction indicates how grain boundaries (interfaces) are represented in the model: discrete sites on the two sides of a *sharp* boundary belong to different phases, whereas a *diffuse* boundary separates two phases, as one of them continuously "fades" to the other. Corresponding tags are shown next to each model title. Finally, a few models which don't fall into any category are also discussed. The list should by no means considered complete.

Potts model (Sharp-interface)

Since its appearance in 1952 as an extension to the ferromagnetic Ising model, the Potts model has been often employed for the study of several systems that allow for a lattice representation. As thoroughly reviewed in [12], the model was introduced in the field of grain growth in [13] due to the evident resemblance between grains and spin (or, generally, state) clusters, usually occuring in the context of magnetic system simulations. Except for the model's success in studies on grain growth kinetics, it has been also proven suitable for conveniently generating microstructures, an otherwise intricate endeavor.

The basic MC algorithm suffered, however, certain physical and numerical deficiencies due to its purely stochastic nature. Notably, it undervalued the theoretically expected grain growth exponent in the small grain size regime, it allowed for artificial nucleation events and was computationally very expensive. Several improvements were proposed, spanning from computational ones (n-fold method) to ones gradually incorporating grain growth physics in the Monte Carlo algorithm [14, 15, 16]. In [17], more recent understanding of the grain growth mechanics led to further modifications, to wit, the dependence of the grain boundary mobility on *both* grain boundary misorientation and grain boundary inclination was effectively taken into account.

Phase-field model (Diffuse-interface)

A rather different, though quite popular, approach to the simulation of grain growth is the phase-field model with early work by [18, 19], more recently in [20] and reviewed in [21]. Within its framework, a microstructure is described by a set of spatially dependent continuous field variables, conventionally denoted by $\eta_i(\mathrm{r})$, i indicating classification with respect to some (discretized) microstructural property. A field variable has also a pre-defined range, say $[-1, 1]$. For example, if i labels the crystallographic orientation, i.e. a discretized version of it, then $\eta_1(\mathrm{r}) = 1$ means that the material at position r has the orientation corresponding to $i = 1$. It is understood that, ideally, one needs an infinite number of field variables. In the above

example, for instance, there should be one for each orientation. However, it is known that even a small number of them, in practice > 36, is usually enough to capture most microstructural characteristics [20].

The field variables may be *conserved* (e.g. a composition field) or *non-conserved* (e.g. the crystallographic orientations field described earlier). For conserved fields, the temporal evolution satisfies the Cahn-Hilliard equation

$$\frac{\partial \eta}{\partial t} = -\nabla \cdot \mathbf{J} \quad ,$$

a diffusion equation, where the current $\mathbf{J}$ is proportional to $\nabla\mu$, μ being the chemical potential, in turn related to the free energy (F) of the system. In the case of a non-conserved field, the evolution is dictated by the Ginzburg-Landau equation

$$\frac{\partial \eta_i(r,t)}{\mathrm{dt}} = -\mathrm{L}_i \frac{\delta F}{\delta \eta_i(r,t)} \quad \forall i \quad ,$$

a relaxation equation, where L_i are relaxation constants (coefficients describing the grain boundary mobilities). The key part of the model is, consequently, to mindfully construct/assume the free energy density functional $f(\eta_1, \eta_2, \ldots)$, so that the total free energy of the system

$$F = F_0 + \int \left[f(\eta_1, \eta_2, \ldots) + \sum_i \frac{\kappa_i}{2} \left(\nabla \eta_i(r)\right)^2 \right] \mathrm{d}^3 r \quad ,$$

where κ_i are the so-called energy gradient coefficients, has an energy minimum for each field variable.

In short, the free energy minimization accounts for the temporal evolution of the field variables, while the boundary topology is being only *implicitly* modified. This way, there is no need for any modification rules for the boundaries as could be the case in Potts-, Vertex- a.o. models. On the downside, the indirect incorporation of boundaries in the model leads to them having finite thickness; a phase turns into another one fast but continuously nevertheless. This limited boundary resolution as well as its considerable computational requirements might render the model unattractive, depending on the case study. The elegance, however, of several important features following from the minimization of a single meaningful quantity, cannot be overlooked.

Vertex & Front-tracking models (Sharp-interface)

Vertex and/or front-tracking models, sometimes considered the same model under two different names [22], were analytically introduced in 2D grain growth in [23]. The basic idea behind them is conveyed in [24], where an early extension to 3D is also reported. A microstructure is represented as a discretized topological network, as the two-dimensional one shown in Fig. 3.1 (a). In its simplified form, the discretization accommodates only vertices of the network, implying that only straight segments connect these. Keeping track of the positions of the vertices and considering only their motion is, therfore, enough (it then is a pure Vertex model). In its full extent, additionally to the vertices, the whole interfacial network is discretized by points, allowing for curved boundaries too. It then becomes a front-tracking, since the interfacial points track the boundary motion. The motion of the points is dictated by two factors: (i) equations stemming from curvature-driven surface energy reduction and (ii) topological effects such as vertex collisions. Additionaly, other topological constraints, markedly, enforcement of certain angles

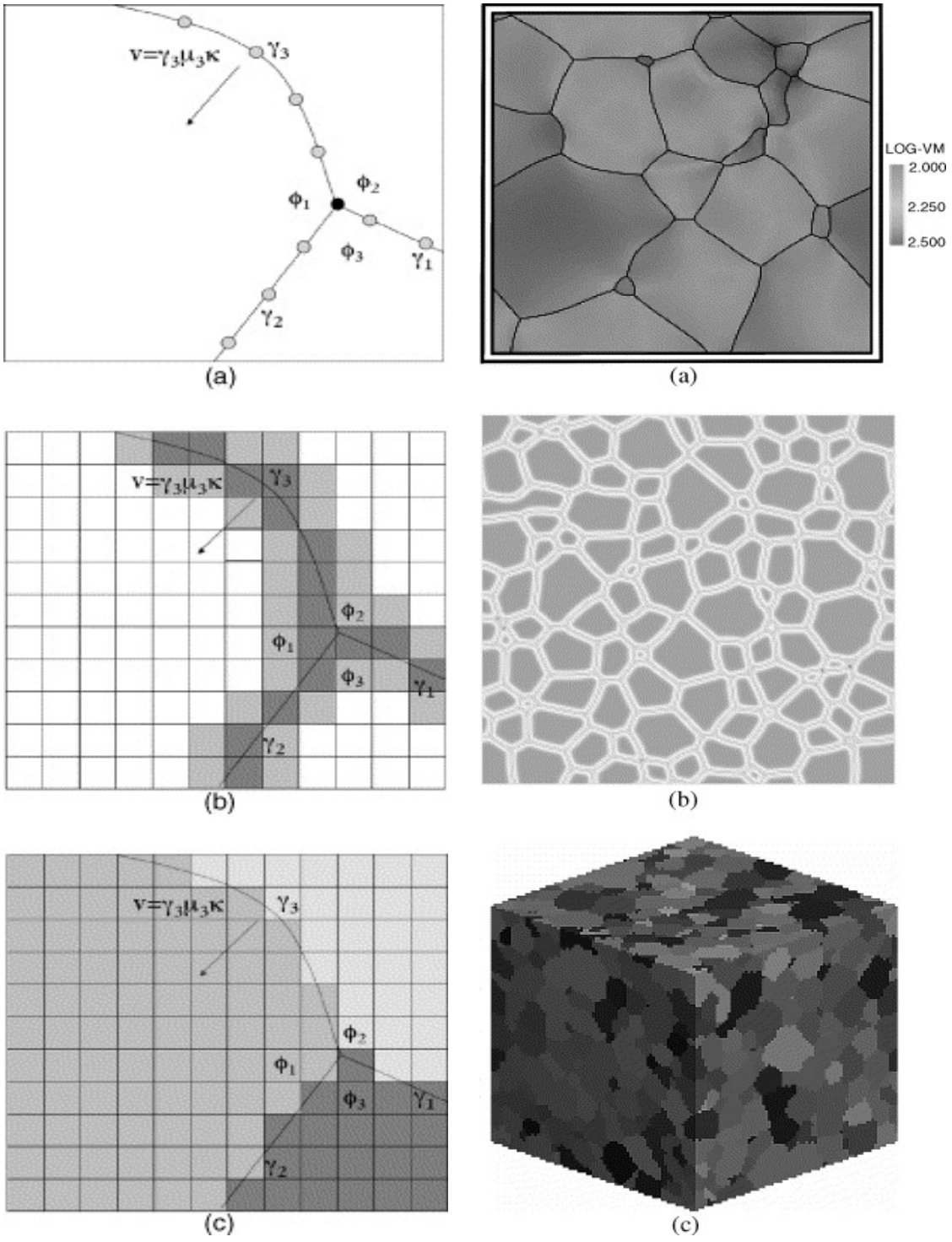


Figure 3.1: Comparison of three grain growth models: Vertex, Phase-field and Potts, one per row, respectively. *Left*: The differences in the modeling approach, (a) Explicit (sharp) boundary motion by means of the vertex/front-tracking model: points of discretized boundaries move with velocity perpendicular to the boundary, depending on its mobility μ, surface tension γ and curvature κ. (b) Implicit (diffuse) boundary motion via the phase-field model: the boundary is not clearly defined. (c) Stochastic boundary migration under the Potts model. *Right*: Final snapshots of microstructures generated by the three methods, to give an idea of how those could look like. Images from [22].

between lines tangent on edges that meet on vertices (2D), introduce further complications. These can be resolved by carefully setting up deterministic sets of rules, a not trivial undertaking, especially in 3D. Apart from its relatively complicated nature, the model is very precise, in particular in studying problems focused on the exact grain boundary topology. Time is also naturally defined, since equations of motion are solved, in contrast to the Potts model, for instance.

Cellular Automaton models (Sharp-interface)

A microstructural evolution model based on a Cellular Automaton (CA), in a basic form in [25, 26] and after more recent refinement in [27, 28], suggests a discretization in space, time and state of a microstructure. A CA is a mathematical representation of a system, under which space is considered a large grid of cells being in a number of finite states (cf. Fig. 3.2). The larger

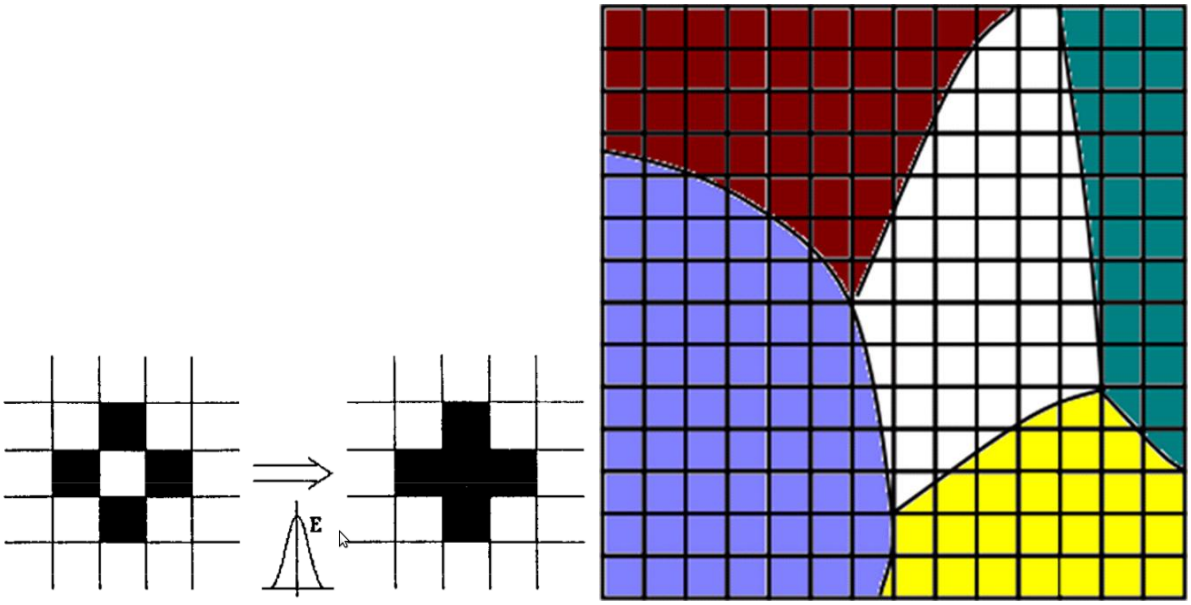


Figure 3.2: *Left*: A simple CA showing how a simple function can be applied to the neighbourhood of a cell to generate the next state (Fig. from [25]). *Right*: A more recent approach to CA, where fractional states are considered, which allow to explicitly identify grain boundaries in cells of fractional states (Fig. from[27]).

the grid, the higher the spatial resolution. The evolution of the system is fully determined by an initial configuration and a predefined set of transition rules. The transition rules change the state of a cell according to the states of the other cells in its neighbourhood. The neighbourhood itself is, in turn, something that needs to be defined. In a basic approach, the neighbours of a cell are the cell itself as well as its next nearest neighbours. Then, a transition rule can be thought of, as a mathematical function F of the form

$$x(i,j,t+dt) = F\left(x(i,j,t), x(i-1,j,t), x(i+1,j,t), x(i,j-1,t), x(i,j+1,t)\right)$$

where x denotes the state of cell (i,j) at time t. It is understood, that the physics behind the evolution is introduced to the model through the transition rules. As a result, the quality of the final "evolved" microstructure depends on how carefully those have been defined.

DEM-oriented models

Discrete Element Models (DEM), dating back to 1979, have attracted considerable attention in the last decade, in particular in fragmentation simulations. Unlike the previously described grain growth models, they do not attempt to simulate the microstructure evolution; they focus on generating a single version of a microsctructure. In this view, they could potentially be used in combination with one of the previously presented models, i.e. use the final simulated microstructure from a grain growth moded and convert it to one for DEM simulation.

In DEM-oriented models, agreggates are modelled as clusters of particles held together by cohesive forces, which keep the relative dispacement between pairs of particles within a certain range. *Beams* are used to express these particle-particle interactions and breaking thresholds are introduced to simulate fracture occurrence. DEMs provide a versatile simulation scheme, particularly in terms of fragmentation, due to the fact that they bridge the simulation to the actual phenomenon at all stages: crack initiation, propagation realistic & customizable bonding parameterization. Samples used by DEMs have been modelled both in 2D and 3D, with agreggates of spherical shape being the most commonplace choice. Such shapes are, clearly, easier to deal with, though not completely realistic. In 3D few attempts to construct realistic DEM samples of polyhedral particles exist [29, 30, 31], with most recent being [32], for which some shape assumption or spherical approximation is employed in order to facilitate contact/collision detection and efficient space filling, while maintaining shape information and grain size distribution. Finally, X-Ray tomography combined with pattern recognition has also been exploited to build realistic DEM samples [33].

Other models

Other models, which are not discussed here in detail, include models based on **Voronoi** tesselations (often employed to generate an initial configuration for some other model but also used on their own), **Mosaic** models (they tessellate 2D (mostly) space by appropriately intersecting straight lines; specialized in cement simulations) and **Packing** models (attempting to sufficiently pack a set of - usually - simplified shapes). The model used in this work is a packing model.

3.2 Clinker simulation by multi-phase grain packing & clipping

As already mentioned, the aim is set at designing a method able to randomly generate a multiphase material, the Clinker. This work focuses on the main constituent of the Clinker, the Alite, which is the phase occuppying over 75% of its volume. The reason behind this is that the techniques involved should provide the means for dealing with other phases as well, as long as they are composed by convex grains of some known shape and grain size distribution. Throughout this Section, the words *grain, shape* and *single crystal* are used interchangeably and are thought of as convex polyhedra. *Growth* will be always refering to *shape-evolution*.

Since the Clinker sample is meant to be used in fragmentation simulations and since in this project the interest does not lie in studying the grain-formation during sintering, the way Alite is originally formed, it seems reasonable to consider grains at the mesoscale, instead of considering detailed grain-growth physical models. This does by no means imply that those models couldn't produce a representative Clinker microstructure. They were not chosen simply because

the available data provide solely shape-related information, while deep understanding of the sintering mechanisms is missing. Moreover, most of those models, start and evolve microstructures which fill the space. However, observing the micrograph in Fig. 2.5, one immediately recognises the complex formations of the Aluminate and Ferrite phases. It seems unlikely that rules could be set, which would lead to precise formation of such a microstructure, starting from a filled space. For these reasons, it is considered more practical to start from the convex phases (i.e. not a filled volume) and generate a microstructure w.r.t. those first.

Alite being a crystal, its shape contains information that should be exploited, namely planes that correspond to certain minimum cleavage energies. To incorporate this information into the model, we use previously computed Miller indices [4], which determine a possible equilibrium shape for Alite. The size of a grain is defined as its maximum caliper diameter. The grain-size distribution is the last piece of information to be considered known throughout this Section needed to create an Alite-grained Clinker.

Shape generation

In a preliminary step, a number of grains, which the Clinker sample will be initialized with, are generated and stored in a shape-pool (Table 3.1). Some shape-evolution considerations follow the ideas in [34]. The simplest way to initialize the pool is to simply scale an initial equilibrium shape in a number of instances. This corresponds to shape-evolution with equal perpendicular velocities assigned to each crystal face, which seems to be a reasonable assumption. As a first approach, this is followed throughout the project. Alternatively, one could also generate shapes - corresponding to a certain grain-size - after assigning different velocities to each crystal shape, as long as one has reason to do so. This would correspond to an Alite grain growing in an anisotropic (local) environment, in which case further information on certain sintering conditions, e.g. average temperature gradient around an Alite grain, should be available. The latter choice for shape evolution should yield a much greater shape-diversity, thus making a random Clinker sample more realistic, although affecting the behaviour of the system under packing, e.g. the time for reaching a certain packing density, if reaching it at all.

In any case, one must make sure that the range of grains-sizes in the pool is broad enough to effectively reproduce the desired grain-size distribution. Additionally, it is helpful, as it is natural according to the previously described procedure, to store the shapes in order of increasing size. This reduces the problem of sampling grain-sizes from a certain statistical distribution to sampling the corresponding indices according to the corresponding discrete distribution.

Packing

After having to our disposal the shape-pool, a number of grains can be sampled and placed within a pre-specified Clinker volume. A spherical volume is preferred, within which the Clinker is built. The initial configuration consists on randomly placed Alite grains in an arrangement, under which the corresponding circumscribed spheres (w.r.t. each grain-radius) do not overlap. After the desired number of grains is in place, the size-distribution constraint is already satisfied and the shapes encode all the information regarding the crystal structure of the phase. It then boils down to achieving a realistic volume-fraction for this phase while distorting the aforementioned characteristics as less as possible. To this end, a simple packing algorithm is applied: for a sufficiently large number of steps, a grain is chosen at random and the following transformations are applied to it

1. A random translation within some range $[0, \mathrm{ds}]$.

Grain Structure	
Vertices	Vertex coordinates
Edges	Edge connectivity table (optional & changeable)
Faces	Face connectivity table (necessary & changeable)
Face2Vtx	Face2Vtx(i): indices of faces intersecting on vertex i
Normals	Face normals (optional)
GState	Geometric crystal state (the **h** vector)
Centroid	Centroid of the grain polyhedron
Size	The max caliper diameter of the grain
Radius	The max distance between the centroid and a vertex (segment of Size)
K	A triangulation of the grain's surface
Volume	The grain volume
MaxSize	The maximum size possible to find in the shape pool
MinSize	The minimumm size possible to find in the shape pool

Table 3.1: The fields of the grain structure. Not all of them need to be calculated and stored in the structure upfront but this improves efficiency during the intersection computations. Furthermore, some might need to be altered on the course of the simulation, namely every time a grain is scaled or clipped.

2. A random rotation around *each* axis, within some range $[0, \mathrm{d}\theta]$.
3. With probability $\mathrm{P} = 0.5$, a random translation in the range $[0, \mathrm{dr}]$ towards the center of the sphere (radial translation).

The move is only accepted if the new position does not cause an overlap, otherwise it is rejected and a new random grain is chosen.

The translation towards the center guarantees that the granular system will eventually move to higher packing density (depending on the initial configuration, of course, which in our case is one of very low density). However, it only occurs with a certain probability in order to allow for sufficient freedom for the grains to re-arrange, even when the density has increased, and to avoid blockings. For example, it makes possible for a grain to step back and change its orientation before attempting to move towards the center again. The parameters ds, $\mathrm{d}\theta$, dr are chosen arbitrarily, although potential tuning of those could probably yield faster and tighter packing.

Note: Each grain posesses a local coordinate system which can either be rotated and translated with it at each step, or once at some later point. In the latter case and if one uses the face orientation to recover this system, it is crucial that this happens *before* clipping, which changes the face connectivity table.

Clipping

After the grains have been sufficiently packed, which can be checked either by visual inspection or by keeping track of the density change over time and consider the system packed when no significant change occurs anymore, one observes that the volume-fraction remains way lower than one wishes. This is to be expected, since it is virtually impossible to fill space efficiently with randomly shaped polyhedra. Further action needs, therefore, to be taken for higher density to be reached. Two different approaches have been tested and are described in the following.

Clipping approach I: Clipping after collective central translation

Under this approach, a subsequent collective translation is applied, with all grains being translated by some radial displacement dr towards the center of the spherical Clinker volume. At this step *only*, all overlaps are allowed. This will, almost surely, lead to several undesired grain intersections, which must, in turn, be resolved. By keeping in mind that this last move was meant to reduce the empty volume within the spherical boundary defining the Clinker, it becomes clear that the grains should be modified in a way that would distort their shape as little as possible, while allowing them to retain as much of their volume as possible. Thinking in this direction, we do the following for each (distinct) pair of grains:

1. The convex intersection of the two grains is explicitely computed as well as its centroid.
2. The plane passing from the intersection's centroid and whose normal is parallel to the vector connecting the two grain-centroids is determined.
3. The two grains are clipped w.r.t. this plane at a pre-defined separating distance (e.g. the machine precission).

This treatment of the grain overlaps is plausible for two reasons: (1) the displacement that causes the overlaps is chosen to be considerably smaller that the typical grain-size and, consequently, the overlaps are not expected to be large and (2) the larger overlaps should be the ones between subsequent radial layers of grains and, intuitively, most of these grains would be in contact at their lowest area faces (it simply makes sense for a tighter packing relative to the initial one).

The overall compacting strategy is itself justified by considering how the volume-fraction behaves under the imposed change: while the collective central translation explicitly reduces the spherical volume containing the Clinker by reducing its radius by dr ($V_{sph} \propto R^3$), the small grain-overlaps, which, presumably, lead to small clipped-out volumes off of the involved grains, make the original total Alite volume decrease by a much smaller amount. Consequently, the ratio of occuppied to available space is expected to increase.

It should be stressed out that the random Clinker nodule generated by the described procedure is expected to exhibit significant porosity, since the interstitial volume between the grains is only stochastically reduced.

The just-mentioned clipping approach was applied and proved to only slightly increase the volume fraction of the Alite phase. Therefore, it will be no further discussed.

Clipping approach II: Clipping after grain expansion

The second clipping approach consists in a grain expansion step preceding the grain clipping. It results in a configuration with all grain centroids retaining their previous positions, while arbitrary overlaps between grains occur. The purpose of this step is, simply, to occupy as much volume as possible at the cost of arbitrary shape deformation. However, this does not necessarily suggest a physically impossible situation: if one considers a pair of grains and their centroids as growth points, it is to be expcted that, at some point during shape-evolution, the two grain surfaces will come in contact and the growth along a certain direction (the one between their centroids) will be disrupted, with a new face, most probably, appearing on the grain surface. The evolution of the remaining faces goes on naturally. Therefore, clipping the pair of expanded grains is justifiable, since it only implies that all faces of the first grain, which are involved in the

clipping, reached the faces of the second grain, which are also involved in the clipping, exactly at the centroid of their intersection (Fig. 3.3). This has, in turn, certain implications on the corresponding face velocities or the textrure of the material. It could mean, for instance, that one of the faces grows faster or that they grow at the same speed but have certain angles w.r.t the plane.

The subsequent clipping follows the three-step procedure described in the preceding paragraph, Clipping approach I. In contrast to the strategy based on compaction though, the one based on expansion fills space *from the inside* instead of shrinking the Clinker's spherical boundary and the final product is, thus, expected to exhibit lower porosity.

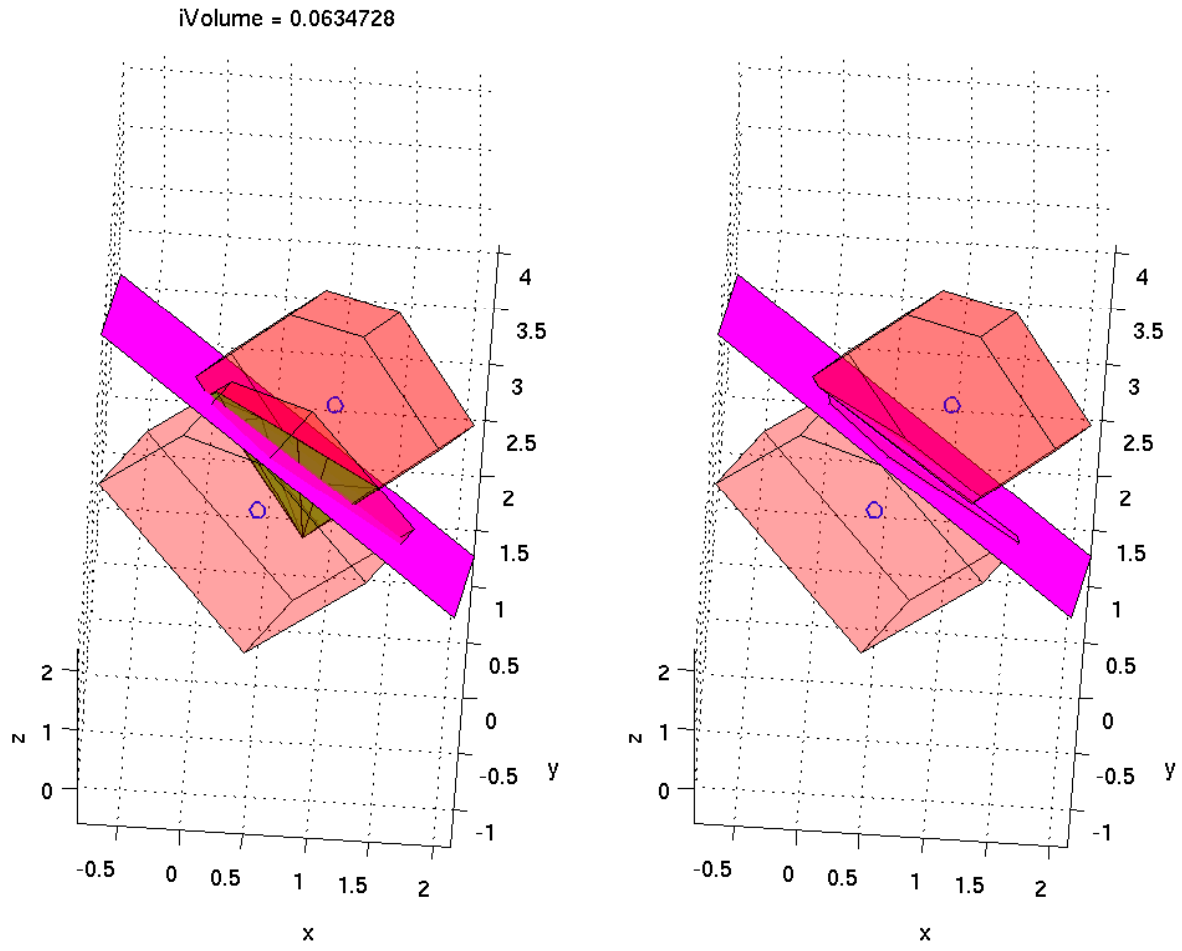


Figure 3.3: Two expanded and overlaping Alite grains are being clipped. Their centroids are also depicted as well as the clipping plane with normal parallel to the line connecting the two centroids and passing through the intersection's centroid.

Related Algorithms

This paragraph offers a brief description of the main algorithms that have been used for implementing the packing and clipping of the granular system. Several geometric computations, which were available as part of the Matlab libraries [35, 36] have been extensively used. Moreover, all figures of Matlab plots have been converted to *.eps* format from Matlab's *.fig* by means of the tools in [37]. The essential function, that enables packing is one that performs *checks* for

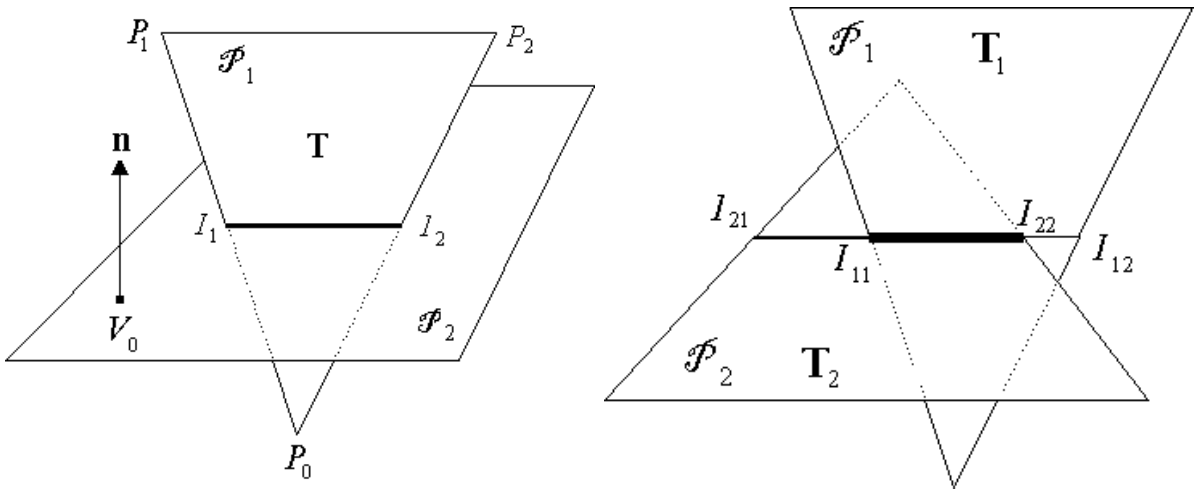


Figure 3.4: 3D triangle-plane and triangle-triangle intersection. *Left*: I_1I_2 is determined by intersecting the lines P_0P_1 and P_0P_2 with the plane $\mathcal{P}_2$. *Right*: Triangle-plane intersections between the pairs $(T_1, \mathcal{P}_2)$, $(T_2, \mathcal{P}_1)$ result to two respective overlaping line-segments (in case there *is*, in fact, an intersection), which by projection to a (non-perpendicular to the segments) axis and sorting yield the endpoints of $I_{11}I_{22}$. The routines that implement the basic triangle-plane intersection, the intersection of two colinear 3D segments and find an appropriate projection axis are listed in Appendix A. Fig. from [38].

intersection of two convex polyhedra. For the clipping part, we wish to compute the intersection explicitly, i.e. obtain all vertices of the polyhedron that defines it. Therefore, a slightly modified version of the second algorithm can be used for the intersection checks as well - although not efficiently - so we only need describe this.

Given the vertices and faces of two arbitrary convex polyhedra, a straightforward convex intersection computation is implemented (cf. Appendix A), based on triangle-triangle intersections. The vertices of the shape comprising the intersection are collected in two steps: in the first step all vertices lying on one of the two surfaces are determined and in the second step all intruding vertices of each shape into the other are determined. For the first step, the faces of both shapes are triangulated, if not provided as such, and all triangle-triangle intersections are specified (for a non-triangular face, the intersection of adjacent triangles with a plane will give a recurring vertex). For the second step a point-in-polyhedron check [36] is performed for the vertices of the first shape w.r.t. the second and vice-versa. The set of (unique) collected vertices is the vertex representation of the convex polyhedral intersection (Fig. 3.6, 3.7).

Concerning the clipping part, Fig. 3.5, 3.3, the mesh of shape is simply clipped to the clipping plane, by means of a corresponding function in [35].

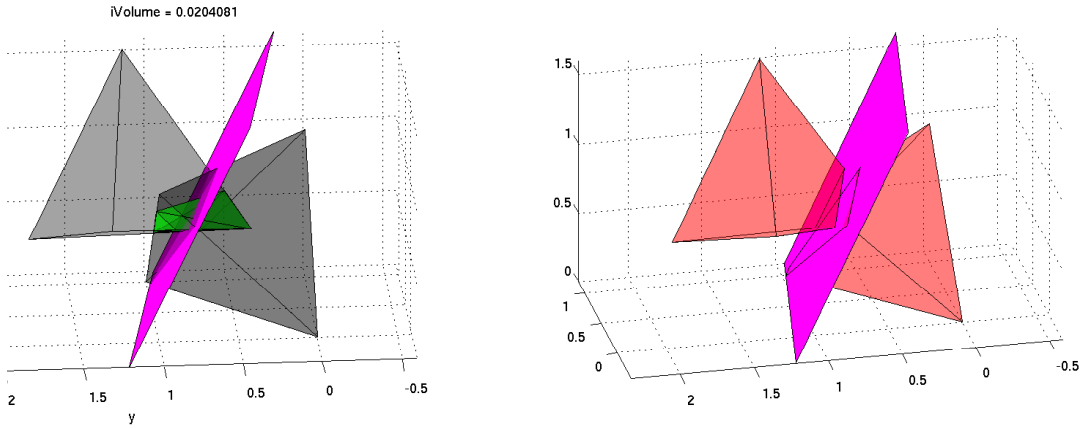


Figure 3.5: Clipping method for separating two overlaping tetrahedral grains. *Left*: A pair of grains in their initial overlaping configuration, the green colored intersection volume and the computed clipping plane. *Right*: The pair of grains after having been clipped w.r.t. the depicted clipping plane. The two grains are tetrahedral, for clear access to visual inspection.

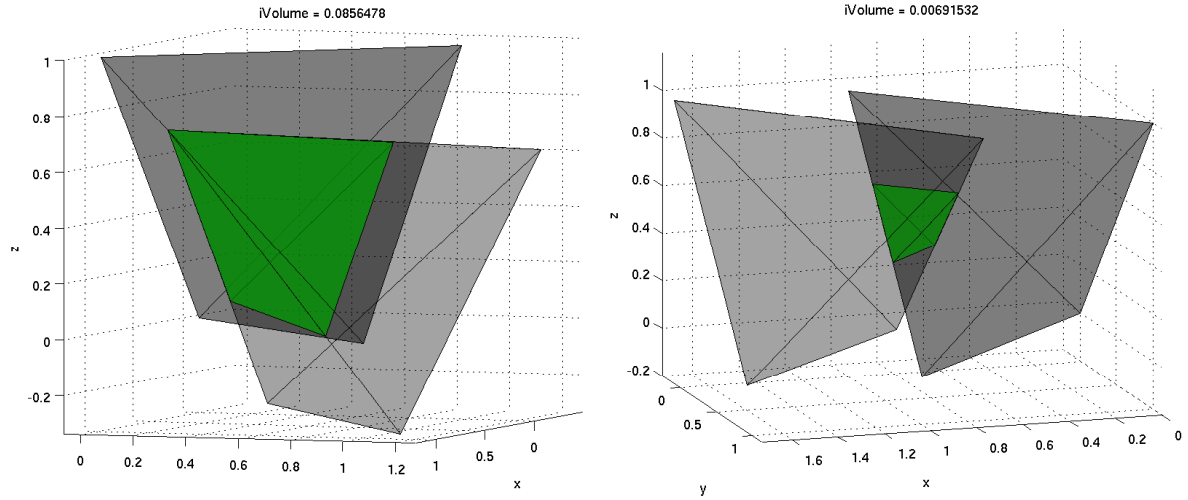


Figure 3.6: Explicit intersection computation fot two special cases of overlaping tetrahedra. *Left*: A vertex of the rightmost shape lies within the first. *Right*: No vertices of one shape are included by the other one. *Green*: The convex hull of each intersection with the corresponding volumes depicted on top of the plots. The intersection is computed as a triangulation, although this is not obvious in the above cases, since both intersections happen to be tetrahedral. The two grains are tetrahedral themselves, for clear access to visual inspection.

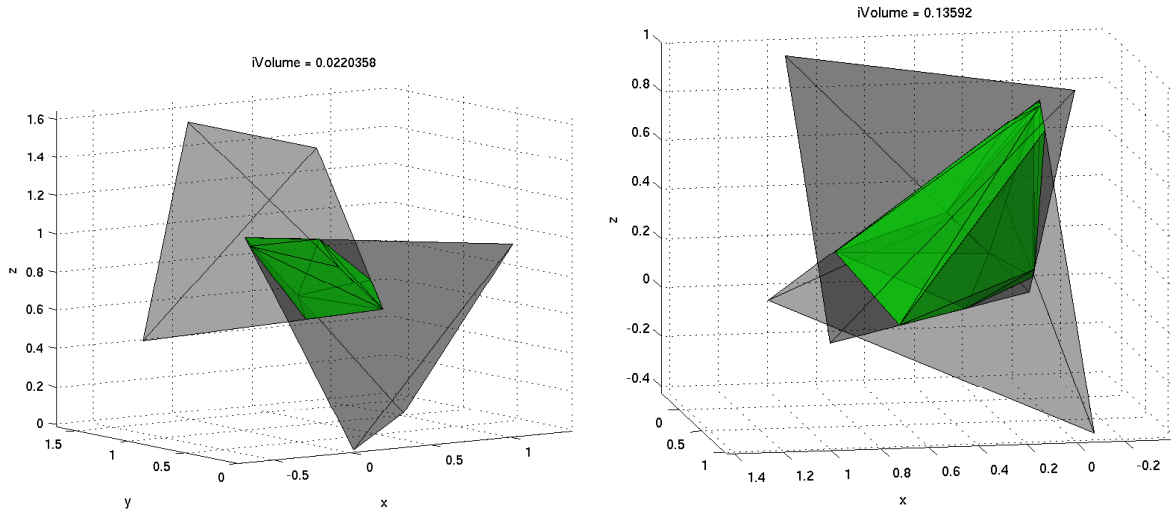


Figure 3.7: Explicit intersection computation for two (more complicated) special cases of overlaping tetrahedra. In both cases, the polyhedral intersection needs to be computed by face-face intersections as well as vertex inclusion checks. *Green*: The convex hull of each intersection with the corresponding volumes depicted on top of the plots. The triangulated intersection surfaces show clearly in the above two plots. The two grains are tetrahedral, for clear access to visual inspection.

Chapter 4

Results

This Chapter reports on the results obtained after simulating Clinker by means of the method proposed in this work. The setup of the simulation is explained and the resulting Clinker sample is examined. Moreover, the efficiency of the used algorithm is discussed, particularly w.r.t. its weak points. Finally, the results are tested against available data.

4.1 Simulating multigrained Clinker of Alite

Simulation of the Clinker as a polydispersed (same shape, different size) granular system of Alite was performed. A shape-pool was generated by scaling a simple Alite shape to thirty instances. For simplicity, the original shape was obtained by truncating a rectangular cuboid of side length $0.5\times1\times1$ and was afterwards scaled by an arbitrarily chosen scaling factor, $\mathrm{S} = 1.05$ (cf. Fig. 4.1). Although this gave an arbitrary shape-size range, polydispersity alone was considered good enough for a first experiment. The shape chosen is not completely irrelevant, since it has the same number of faces as an expected Alite equilibrium shape (Fig. 2.2) and a set of its parallel faces.

In the initial configuration, 100 grains were picked, whose size followed a Gamma-like distribution. In practice, this was done by sampling the indices of the shapes in the shape-pool, using a Poissonian distribution. The grains were randomly placed within a sphere of radius $\mathrm{R} = 12$ without overlaps and with random orientation. The result was an initial configuration of very low density. This was chosen to be low, for the grains to be rapidly placed. The initial configuration of the Alite phase is depicted in Fig. 4.2 and its grain size distribution in Fig. 4.6.

The system was then packed by the algorithm described in Chapter 3. The translational step was set to vary in ds $\in [0, 0.02]$, the rotational in dtheta $\in [0, \pi/100]$ for all three axes and the probability for the extra central displacement was set to $P = 0.6$. The number of iterations was initialized to fourty thousand, though progressively raised to fifty- and, finally, sixty thousand, since potential for higher packing density was evident. The result of the packing process is shown in Fig. 4.3.

It is evident, even by mere visual inspection, that the packing density remained quite low w.r.t. the desired one, i.e. $\approx 80\%$. The packing was, therefore, followed by expansion of all grains and clipping of the sample, as described in Chapter 3, in order to recover the no-overlap constraint. The scaling factor used in the grain expansion was varied within $\{1.5, 1.7, 1.9\}$, with $\mathrm{S} = 1.9$ being the first to yield an acceptable volume fraction of $\approx 78\%$. The Clinker sample

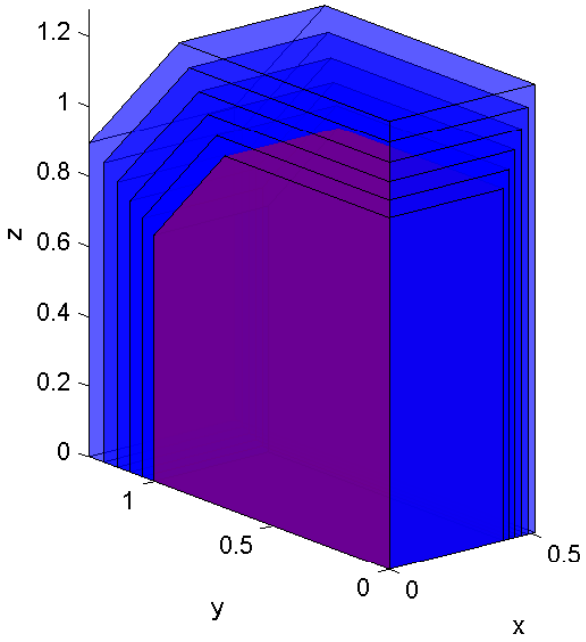


Figure 4.1: Five scaled instances (blue) of an original Alite grain (red) are generated in order to be placed into the shape-pool.

after clipping is plotted in Fig. 4.4 and its new grain size distribution in Fig. 4.6.

4.2 Performance of the simulation algorithm

It is clear, that the most computationally intensive part of the Clinker simulation is the packing of convex polyhedral grains. This is due to the fact that at each iteration a grain is randomly moved and then checked for overlap against all other grains in its neighbourhood. Although the space surrounding each grain is mostly empty in the initial configuration, this soon changes, as the grains tend to gather towards the center of a sphere. As a result, the simulation becomes quite slow after a few thousand iterations. The major reason for that is the naive implementation of the convex intersection checks, which intersects two convex polyhedra as two triangulated surfaces, i.e. triangle-triangle. Considering that the grains used have a surface triangulation of 14 triangles, even a single check becomes quite expensive. This also imposes a constraint on the shapes one might employ and, this way, phases composed of more complex polyhedra, are highly disfavoured; e.g. the Belite. Actually, even the shapes employed in this work, are complex enough to effectively hinder any attempt for serious experimentation. Namely, the Clinker sample presented in this Chapter took more than three days to pack[1]. Of course, such computational complexity is to be expected, when dealing with arbitrary convex polyhedra but far more efficient approaches exist and are discussed in Chapter 5.

Due to the above-mentioned restrictions, the packing rules could not be optimized. The parameters defining the random movements of the grains were intuitively chosen and could probably be configured to speed-up the packing procedure. The rejection ratio (ratio of rejected moves to

[1]On an Intel(R) Core(TM)2 Quad CPU Q9650 @ 3.00GHz, under standard Matlab configuration.

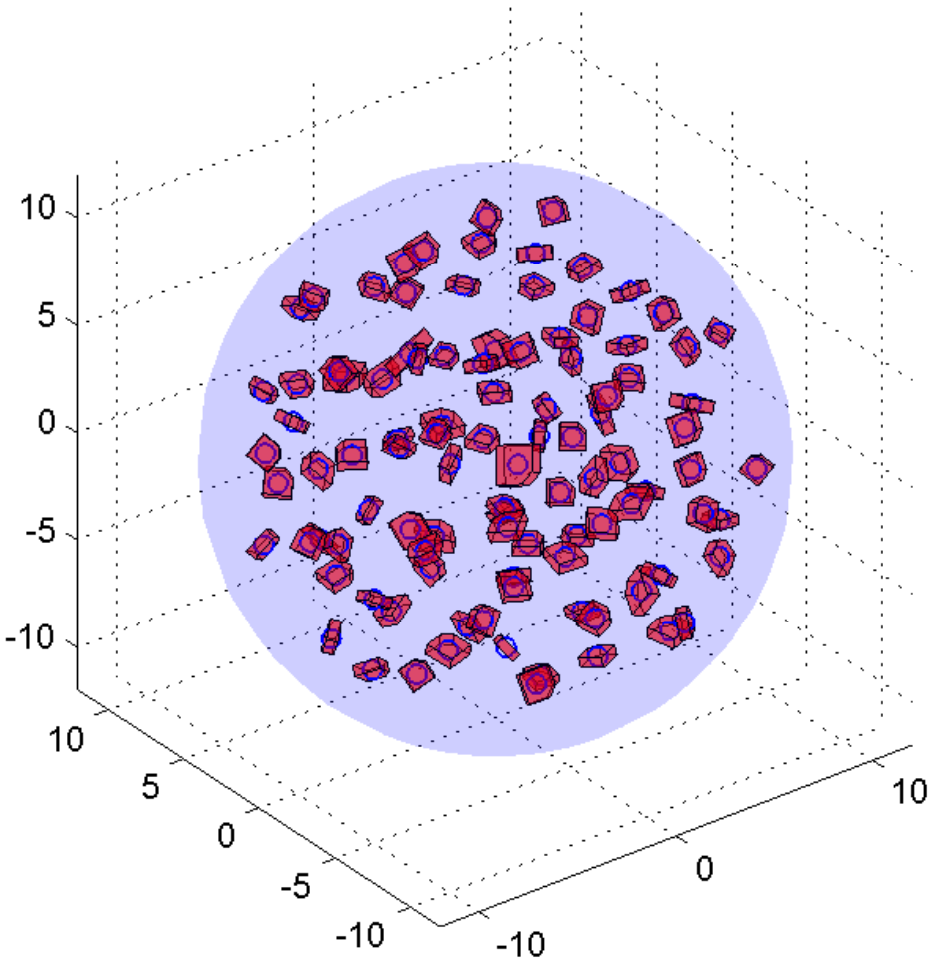


Figure 4.2: A polydisperse system of 100 Alite grains is initialized inside a spherical volume by randomly placing the grains without overlaps and randomly oriented. Their centroids are also plotted to emphasize the difference in sizes. For the grains to be placed fast, a high initial sparsity was allowed. The grain size distribution is the original one plotted in 4.6. The system is centered around the origin.

total number of moves) during packing was measured and found to rise sharply after the grains started to approach each other, to finally lie within $55 - 60\%$.

Clipping also makes use of the intersection computation and is also considered expensive. However, since it is applied only once for each pair of grains, the time-cost is limited within some acceptable range. In the simulation reported in this Chapter, it took 30-40 minutes for the packed & expanded sample to be clipped, the time increasing with the expansion factor.

4.3 Clinker sample volume fraction, grain size distribution & 2D profile

The volume fraction of a phase is the volume of Clinker, occuppied by this phase. It was the main objective of the simulation to obtain a realistic volume fraction for the Alite phase. This

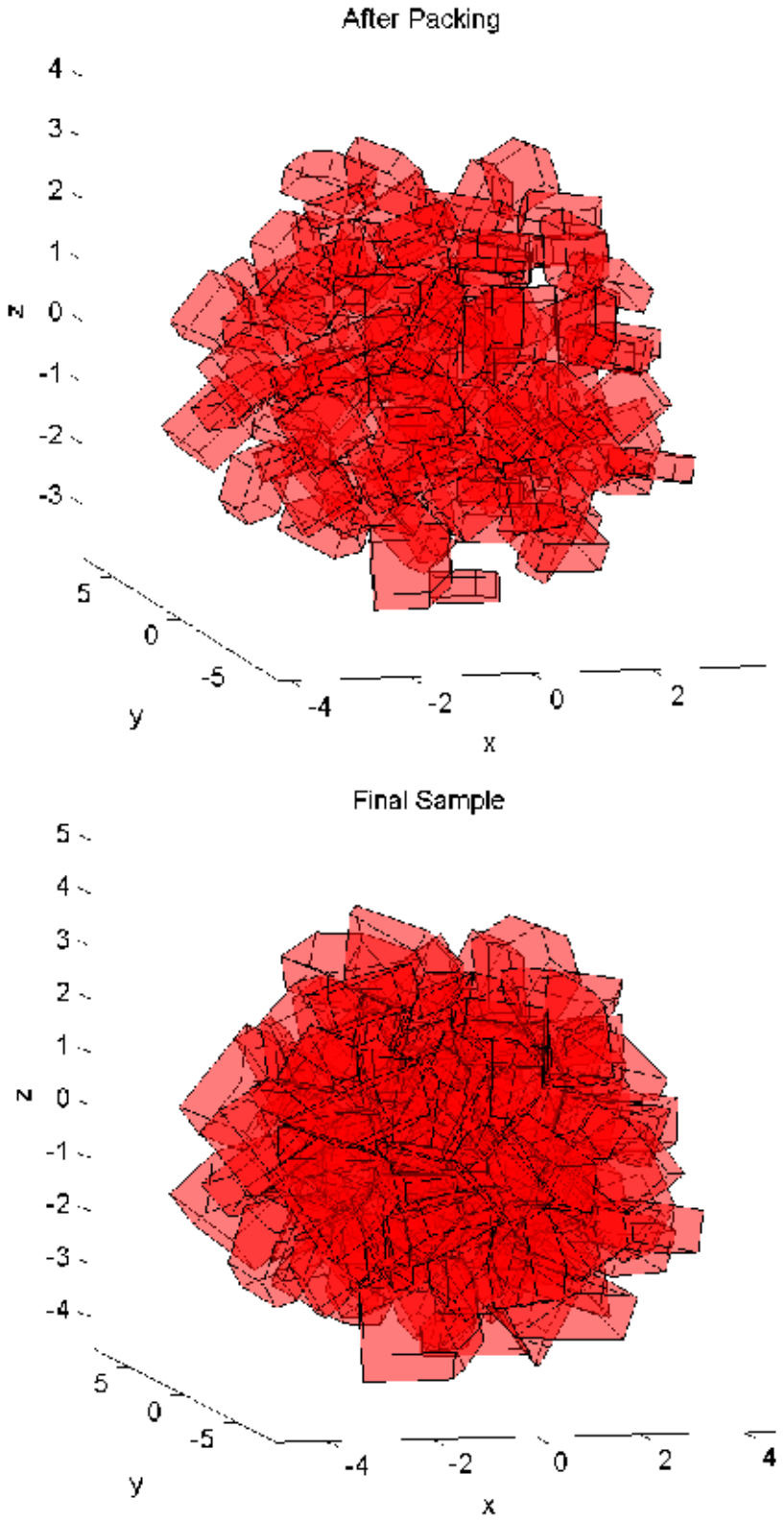


Figure 4.3: Close-up on the Clinker sample right after the packing and after the clipping (final). The packing artifacts of the merely packed sample are evident; there are even grains, which barely touch the bulk of the sample. However, after the expansion & clipping, the sample appears to be solid and dense, especially within an evident spherical interior bulk.

was achieved by raising the expansion factor to $S = 1.9$. However, determining the Clinker's volume is not trivial, since the outermost layer of the sample is highly inhomogeneous (cf. Fig. 4.4). One could, for instance, define the Clinker's volume to be that of a sphere centered at the system's center of mass, with radius being the average distance between the center of mass and some of the centroids of the outermost grains. This would, though, still be problematic, due to several prodruding grains on the surface (a packing artifact). Their centroids would cause the sphere's radius to be overestimated and the volume fraction to be substantially underestimated (the volume of a sphere is $\propto R^3$, R being the radius).

After this observation, the volume of the Clinker is approximately (visually) determined, as indicated by the spherical surface depicted in Fig. 4.4. It is assumed, that the volume of the grains protruding this sphere, could actually fit within. Under this approximation, the volume fraction is reported to be $\approx 78\%$.

Next, the grain size distribution of the sample is computed for three different values of the expansion factor, S. The results are shown in Fig. 4.6. As expected, due to the expansion of the grains, the distribution is shifted towards higher grain sizes, depending on S. It could also be observed, that the distribution tends to look like a Gaussian, although a safe comment on that would require far better statistics. What is expected, is that grains lying on the spherical Clinker surface are much less affected by the clipping, since they have very few neighbours, in comparison to the interior grains. As a consequence, those are the ones that tend to keep the distribution at its original form, while all interior ones are irregularly clipped. This might be the reason, why for $S = 1.5$ and 1.7 the distribution seems to be maintaining its original positive skewness, whereas at $S = 1.9$ the interior grains are more extensively clipped, leading to a distribution of negligible skewness.

A 2D profile of the Clinker sample is useful to compare against a micrograph of Chapter 2 (cf. Fig. 2.7). To this end, the sample was intersected near its center with a plane of arbitrary orientation (in this case, parallel to an axis) and the grain profiles are depicted in Fig. 4.5. The comparison with the available micrograph can only be done in an abstract level, since (i) there is no solid evidence that the original Alite shape closely resembles the ones giving the profiles in the micrograph 2.7, (ii) even if it did, the use of same shapes for all grains is not realistic and (iii) the clipping procedure anyway distorts the grain-shapes in an unpredictable manner.

4.4 Quantifying the microstructure distortion

It is assumed that the original microstructure has the statistical characteristics of the Clinker sample just after it has been packed, i.e. original size distribution, random orientation and no shape distortion. Clipping does not affect the orientation of a grain but it could change its shape considerably. With this in mind, the quantification of the microstructure distortion is done by computing the average number of faces for the Alite grains in the final sample. This is found to be 12.19, indicating an important increase in the number of faces for each grain but no relevant data is available to compare with.

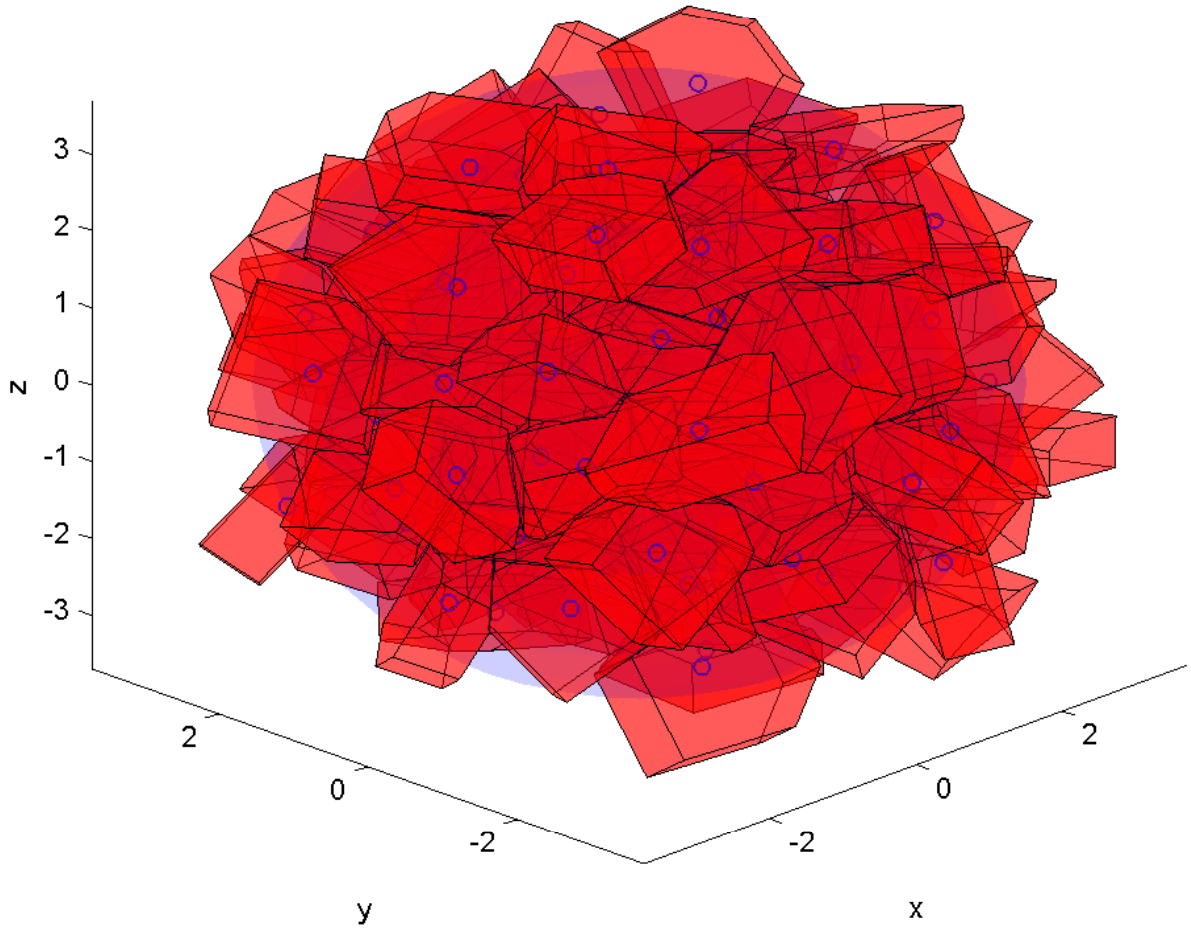


Figure 4.4: The final sample of Alite-composed Clinker. Depicted are the final shapes of the grains, their centroids and the sphere (light blue) considered to confine the Clinker volume. The grains have been packed, expanded with scaling factor $S = 1.9$ and finally clipped. Protruding grains, which are a packing artifact, are still obvious but the bulk of the material under the outermost layer is considered to be homogeneous. The average number of faces for a grain in this sample is raised from 7 (before clipping) to 12.19 (after).

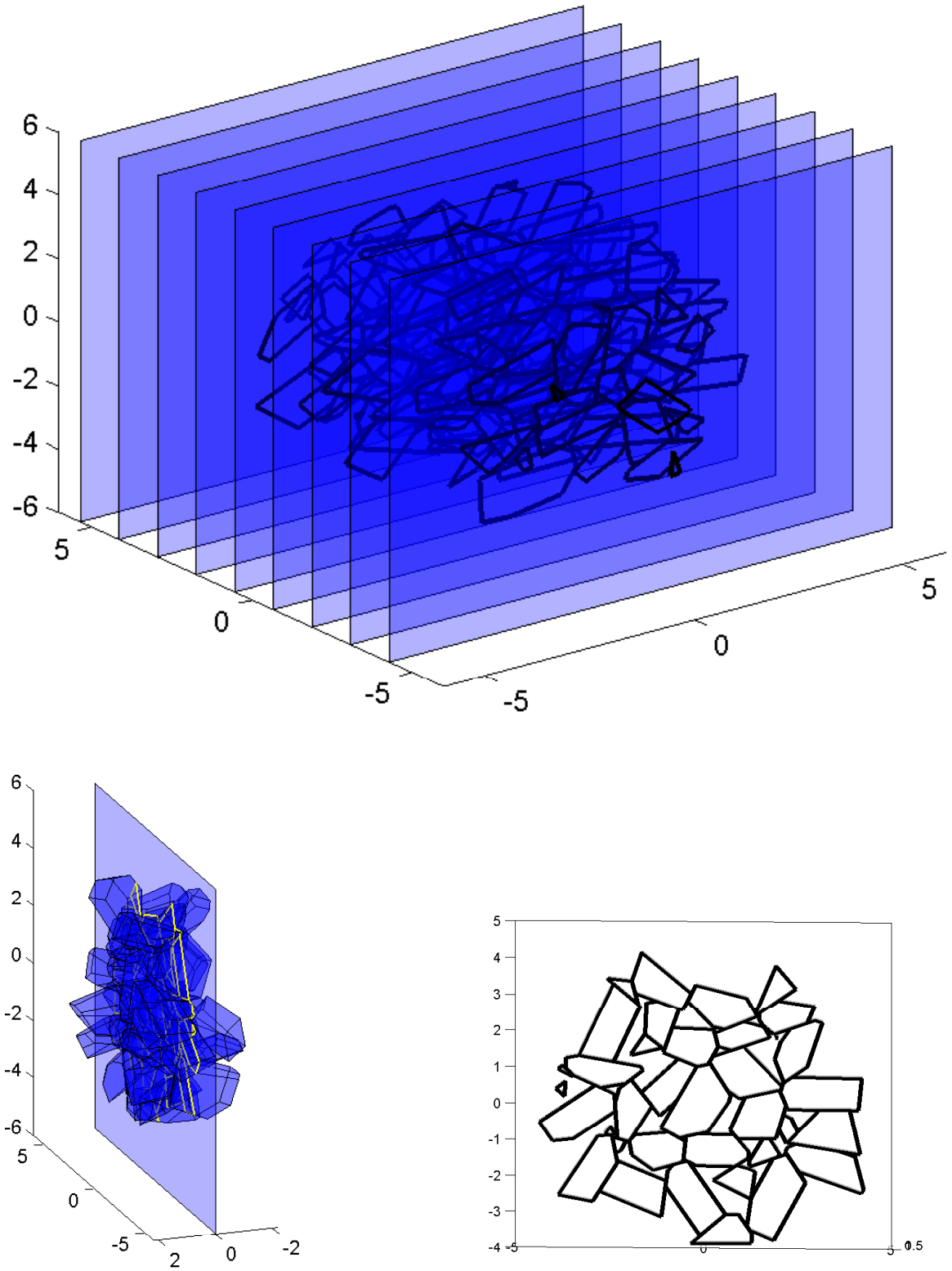


Figure 4.5: *Top*: Nine parallel planes intersect the simulated Clinker sample of 4.4 giving 2D slices to qualitatively compare with micrographs such as 2.5, 2.7. *Bottom Left*: a single 2D slice of the sample near the center. Plotted are the plane of intersection, all grains which intersect with it as well as their 2D profiles. *Bottom Right*: The singled out individual 2D grain profiles as seen from a frontal view.

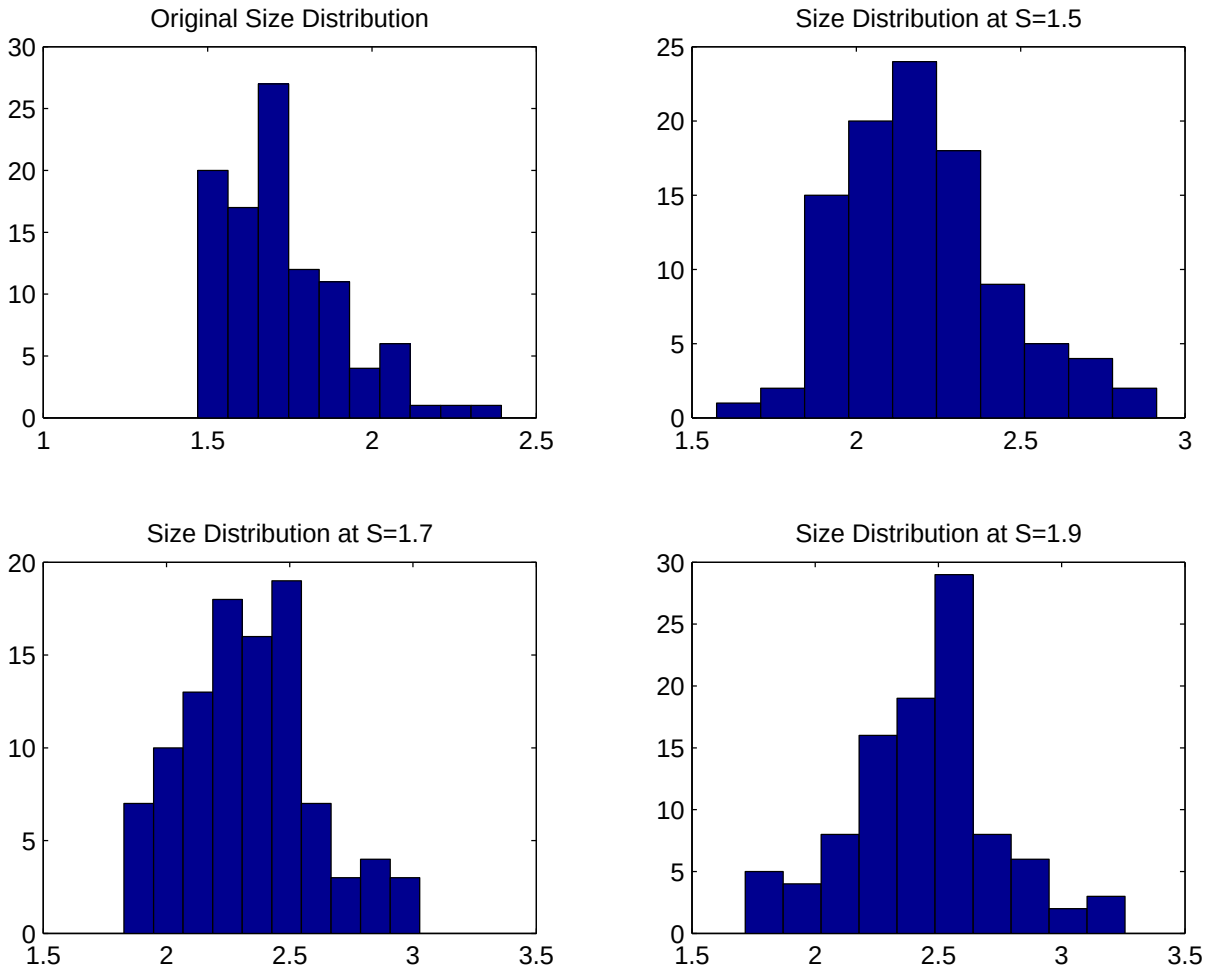


Figure 4.6: The original grain size distribution followed by the computed one after expanding all grains by a scaling factor S = 1.5, S = 1.7 and S = 1.9, respectively. Except for the expected shifting to larger sizes, the positive skewness of the original distribution appears to be subsiding with S.

Chapter 5

Conclusions & Outlook

Simulation of multigrained Clinker composed by Alite, its major phase, was performed by means of a conceptually simple method, based on packing and clipping. A polydispersed system of 100 convex Alite grains of simplified shape was initialized inside a spherical volume. The system was then packed, partly by random and partly by collective motion towards the center of the sphere, the grains were expanded by a certain scaling factor and finally clipped to eliminate possible overlaps.

The resulting Clinker sample (Fig. 4.4) yields an reasonable volume fraction of $\approx 78\%$, which could be modified by the choice for the scaling factor. Due to the grain expansion, the distortion of the grain shapes was anavoidable, with the sharp increase of the number of faces per grain being the predominant effect. It is also understood that the shape of outermost grains was far less affected by the clipping. Some packing artifacts were also detected, namely grains being barely in contact with the bulk of the sample and, thus, destroying the homogeneity of the outermost layer of the Clinker. Two-dimensional slices of the sample (Fig. 4.5) showed reasonable grain profiles. The grain size distribution of the grains (Fig. 4.6), initially a Gamma-like distribution, was found be deformed from positive to zero skewness, depending on the scaling factor S. Based on the above observations, only qualitative agreement between the 2D Alite grain profiles, Fig. (Fig. 2.7 & 4.5), can be concluded. For the microstructural statistical properties of the sample to be consistently incorporated into the method, several changes/variations should be further considered and experimented with.

In view of the geometrical approach employed in this work, the goal is set to simulate Clinker as a microstructure composed by two crystalline and two amorphous phases. I assume that the concept of starting with shapes consistent with the ones estimated by 2D profiles, found in tomographic micrographs, remains in effect. One would then need to re-arrange these shapes, in order to obtain the desired packing density, orientation distribution etc. Existing methods to exhaustively exploit the available micrographic input [8] are known and could serve as a better start. In particular, even if there is no information about the grain shape of a phase, one could estimate certain of its characteristics from 2D profiles alone. If the shape is known, its characteristics can be estimated more precisely by simulating its 2D profiles as random planar intersections with its original form.

As far as the packing part is concerned, there seem to be several attractive options. Except for the shape itself, there are numerous parameters that could affect the effectiveness of the packing. For instance, the range of sizes in the initial size distribution, the shape diversity (different shapes, e.g., with different number of faces, are valid and correspond to a single

size), the shape itself (if only one is used), the initial positions of the grains and, of course, the packing rules. The rejection method of random displacement with additional collective motion towards the center of the confining volume applied in this work, is a set of packing rules that can certainly be improved. One can think of more relaxed rejection criteria, such as allowing low-volume overlaps that might be resolved in one of the next steps. Extensive experimentation with variations such as: applying either a translation or a rotation on a grain but not both in one move, optimizing the translational and/or rotational steps (ds,dtheta), optimizing the probability for the central displacement, changing the central nature of the overall motion and, perhaps, introducing Molecular Dynamics in the system (e.g. repulsive contact forces), could considerably improve the packing (w.r.t. both speed-up and/or packing density).

Since packing arbitrary polyhedra is unlikely to yield a sufficiently high packing density, a clipping step (or some other sort of "final re-arrangement") should be, in any case, needed. In contrast to the generic rule used in this work, namely clipping of two overlaping grains based on, roughly, minimizing the discarded volume, while convexity is retained, more careful analysis could give better insight as to how to optimize this step. For instance, the ratio of the part of the intersection volume that belongs to each grain to that grain's volume could be another criterion driving the clipping plane selection. Finally, considering the grains' centroids as growth points (seeds) and asignment of face velocities to grains, could give rise to much more sophisticated clipping rules, which could, potentially, reverse engineer the grain growth.

For testing the above ideas, a much more efficient algorithm needs to be designed. More importantly, the algorithm that tests for (convex) grain overlaps should be separated from the one that explicitly computes the overlap. In the former case, an implementation of a check based on the Separating Axis Theorem (SAT) [39, 40] seems to be the best choice, if the grain shapes don't have many edges. In the latter case, algorithms used in games for real-time collision detection [40], prominently, the Gilbert-Johnson-Keerthi (GJK) algorithm, are expected to substantially speed-up the convex intersection computations. Further suggestions to this direction would point to a hierarchical representation of all convex grains (Dobkin Kirkpatrick hierarchy), which can also be combined with the GJK algorithm. This would boost the efficiency even more, especially for convex shapes of high complexity and this way, additional phases featuring grains with more complicated surfaces could be added to the simulation. For instance, Belite was not simulated along with Alite for this very reason; it is shown to exhibit a roundish 2D profile (Fig. 2.5) and would, therefore, require a grain shape of rather complex surface, something that would render the simulation (under inefficiently implemented intersection computations) practically stagnant. Under this perspective, the propositions in these paragraph should be considered with the highest priority, since they would make possible to test much larger systems, much faster and under various configurations, thus allowing for optimization of the method.

Listings

Appendix A

Matlab Code

Listing A.1: makeShapePool.m

```
% makeShapePool.m
clear all;
% Script that generates a shape-pool and saves it in 'Pool.mat'.
%
% An initial shape is defined and it is scaled by a defined scaling factor
% and for a defined number of times.
%
% Shapes (or Crystals or Grains) in the pool are convex and represented
% by structures with the following fields:
%
%   Vertices    The vertices of the grain.
%   Edges       The edges of the grain.
%   Faces       The faces of the grain.
%   Normals     The face-normals.
%   Centroid    The centroid.
%   Size        The max-calliper diameter.
%   Radius      The radius of the smallest sphere containing the grain.
%   K           The surface triangulation of the grain.
%   Volume      The Volume.
%   maxSize     The maximum size of a shape in the pool.
%   minSize     The minimum size of a shape in the pool.

% Plotting flag.
pflag = 1;

% Initialize a shape that looks like alite in equilibrium.
[v e f] = createCube;

% Create clipping planes to truncate the cube.
Plane1 = createPlane([0.4 0 0],[1 0 0]);
Plane2 = createPlane([0.3 0.8 0.9],[0 1 1]);
[v,f] = clipConvexPolyhedronHP(v,f,Plane1);
[v,f] = clipConvexPolyhedronHP(v,f,Plane2);

if(pflag)
    figure, axis equal, view(3);
    drawMesh(v,f,'facealpha',0.4);
end

N = normalize(faceNormal(v,f));

```

```
% Initialize vars.
maxSize = 0;
minSize = 1e5;

% Define number of shapes in the pool.
nShapes = 5;

% Create the structure holding the shapes.
for k = 1:nShapes

    D = pdist(v); % pairwise distance between  vertices.
    Z = squareform(D); % put it into square form.

    Crystal.Vertices = v;
    %Crystal.Edges = e;
    Crystal.Faces = f;
    Crystal.Normals = N;
    Crystal.Centroid = polyhedronCentroid(v,f);
    Crystal.Size = max(D);

    % vidx (vertex indices) corresponding to distance max(D).
    [I,J] = find(Z==Crystal.Size);
    Crystal.Radius = Z(I(1),J(1)); % radius of circumscribed sphere.

    [Crystal.K, Crystal.Volume] = convhull(v);

    if(Crystal.Size>maxSize), maxSize=Crystal.Size; end;
    if(Crystal.Size<minSize), minSize=Crystal.Size; end;

    C{k} = Crystal;

    ST = createScaling3d(1.05);
    v = transformPoint3d(v,ST);

    if(pflag)
        drawPolyhedron(v,f,'facealpha',0.4,'facecolor','b');
    end

end

if(pflag)
    axis equal, xlabel('x'), ylabel('y'), zlabel('z'), view(3);
end

% Save maxSize and minSize of the Pool in all grains.
for j = 1:k
    C{j}.MaxSize = maxSize;
    C{j}.MinSize = minSize;
end

% Save the shape-pool.
% save('Pool.mat','C');
```

Listing A.2: initConfig.m

```
% initConfig.m
clear all;
% Places polyhedral grains at random points within a sphere.
% The grains are placed in a way that their circumscribed spheres don't
```

```
% overlap.
%
% The grains are loaded by the file 'Pool.mat', which is assumed to have
% been generated by the script makeShapePool.m.

% Load Shape-Pool.
load('Pool.mat','C');

nObj = 10;      % Desired number of objects to be placed.
nObjPlaced = 0; % Number of objects, which have been placed.

% Initialize the cell array of grains.
Grains = cell(nObj,1);

R0 = 0;      % Sphere center.
R = 12;      % Sphere radius.

while nObjPlaced < nObj

    % Random direction.
    [phi, theta] = randomAngle3d;

    % Pick a shape from the Pool,
    % sample the indices with a Poisson distribution.
    sidx = poissrnd(5) + 1;
    Shape = C{sidx};

    % Pick a radius *contained* in the volume.
    rho = R0 + R*rand - Shape.Radius;

    % New centroid.
    newCentroid = sph2cart2(phi,theta,rho);

    % Check if newCentroid is a good position (no overlaps).
    if( ¬posGood(Grains,Shape.Radius,newCentroid) )
        continue;
    end

    % Translation vector.
    tVector = newCentroid - Shape.Centroid;

    % Create translation transform.
    TSL = createTranslation3d(tVector);

    % Create random rotation transforms.
    Rx = createRotationOx(newCentroid,rand*pi);
    Ry = createRotationOy(newCentroid,rand*pi);
    Rz = createRotationOz(newCentroid,rand*pi);

    % Create final transform.
    TF = composeTransforms3d(TSL,Rx,Ry,Rz);

    % Apply transform.
    v = transformPoint3d(Shape.Vertices,TF);

    Shape.Vertices = v;              % Updated Vertices.
    Shape.Centroid = newCentroid;    % Updated Centroid.

    nObjPlaced = nObjPlaced + 1
    Grains{nObjPlaced} = Shape;      % Updated Configuration.
```

```
65
66 end
67
68 % Save the initial configuration.
69 save('SystemInit.mat','Grains');
```

Listing A.3: packSystem.m

```
1  % packSystem.m
2  clear all;
3  % Packs a granular system of non-overlapping convex grains.
4  %
5  % The configuration is loaded from the file 'SystemInit.mat', which is
6  % assumed to have been generated by the script initConfig.m .
7
8  % Load initial configuration.
9  load('SystemInit.mat','Grains');
10
11 % Reset the Random Number Generator.
12 rng('shuffle');
13
14 % Plotting flag.
15 pflag = 0;
16
17 nGrains = size(Grains,1);
18
19 % Radius of the confining spherical volume.
20 R = 12;
21
22 if(pflag)
23     % Plot initial configuration.
24     figure; hold on; view(3);
25     axis([-R R -R R -R R]);
26     drawSphere([0 0 0],R,'facecolor','blue','facealpha',0.1);
27     for k = 1:nGrains
28         drawMesh(Grains{k}.Vertices,Grains{k}.Faces,...
29             'facecolor','r','facealpha',0.4);
30         drawPoint3d(Grains{k}.Centroid);
31     end
32     pause(0.1);
33 end
34
35 nRep = 10000; % Number of iterations.
36 rratio = 0;   % Rejection ratio.
37
38 % Define infinitesimal translation/rotation bounds.
39 ds = 0.02;
40 dtheta = pi/100;
41
42 % Pick random grain indices for the whole process.
43 gidx = ceil(nGrains*rand(nRep,1));
44
45 % Uncomment for movie generation.
46 % frameId = 1;
47
48 % For a number of repetitions do.
49 for irep = 1:nRep
50
51     % Every 100 repetitions print info.
```

```
52      if(mod(irep,10)==0)
53          irep
54          rratio*100/irep
55      end
56
57      % Pick a particle at random.
58      iGrain = Grains{gidx(irep)};
59
60      % Pick translational/rotational params randomly.
61      ids = ds*randc(1,3);
62      idtheta = dtheta*randc(1,3);
63
64      % Translate; With P=0.6 add a translation part towards the center.
65      if(rand<0.6), iTSL = createTranslation3d(ids);
66      else iTSL = createTranslation3d(ids+([0 0 0]-iGrain.Centroid)*0.01);
67      end
68
69      % Rotate.
70      iRx = createRotationOx(iGrain.Centroid,idtheta(1));
71      iRy = createRotationOy(iGrain.Centroid,idtheta(2));
72      iRz = createRotationOz(iGrain.Centroid,idtheta(3));
73
74      % Compose transforms.
75      iTF = composeTransforms3d(iRx,iRy,iRz,iTSL);
76
77      % Apply transforms.
78      iGrain.Vertices = transformPoint3d(iGrain.Vertices,iTF);
79      iGrain.Centroid = transformPoint3d(iGrain.Centroid,iTSL);
80
81      % If the moved grain overlaps with another, reject.
82      if(grainOverlaps(Grains,iGrain,gidx(irep)))
83          rratio = rratio + 1;
84          continue;
85      end;
86
87      % The grain has been moved, update vertex/centroid positions.
88      Grains{gidx(irep)}.Vertices = iGrain.Vertices;
89      Grains{gidx(irep)}.Centroid = iGrain.Centroid;
90
91      % Plotting update.
92      if(pflag)
93          if(mod(irep,100)==0) % plot
94              clf; hold on; view(3);
95              axis([-R R -R R -R R]);
96              drawSphere([0 0 0],R,'facecolor','blue','facealpha',0.1);
97              for k = 1:nGrains
98                  drawMesh(Grains{k}.Vertices,Grains{k}.Faces,...
99                      'facecolor','r','facealpha',0.4);
100                 drawPoint3d(Grains{k}.Centroid);
101             end
102             % Uncomment to generate movie.
103             % M(frameId) = getframe;
104             % frameId = frameId + 1;
105
106             % pause(0.1);
107         end
108     end
109
110 end
111
```

```
% Print rejection ratio.
rratio = rratio*100 / nRep

% Save the packed configuration.
save('AlitePacked.mat','Grains');

% Uncomment to save movie.
% save('AlitePacked_Movie.mat','M');
```

Listing A.4: expandClip.m

```
% expandClip.m
clear all;
% Blows up (expands) all grains in the system & then clips them.
% Plots the system before and after.
%
% NOTE: Right after clipping is done, the centroids should be updated by
% means of the script updateCentroids.m .
%
% The grains are loaded from the file 'AlitePacked.mat', which is assumed
% to have been created by the script packSystem.m .

% Load the packed configuration.
load('AlitePacked.mat','Grains');
nGrains = size(Grains,1);

% Plotting flag.
pflag = 1;

if(pflag)
    figure;
    subplot(1,2,1), hold on, axis equal, view(3);
    title('Initial');
    for k = 1:nGrains
        drawMesh(Grains{k}.Vertices,Grains{k}.Faces,...
            'facecolor','r','facealpha',0.3);
        drawPoint3d(Grains{k}.Centroid);
    end
end

% Expansion (scaling) factor.
S = 1.7;

for k = 1:nGrains

    % Translation vector.
    tVector = [0 0 0] - Grains{k}.Centroid;

    % Translate to the origin.
    Grains{k}.Vertices = ...
        Grains{k}.Vertices + repmat(tVector,size(Grains{k}.Vertices,1),1);

    % Scale (blow up).
    Grains{k}.Vertices = transformPoint3d(Grains{k}.Vertices,...
        createScaling3d(S));

    % Move back to original position.
    Grains{k}.Vertices = ...
        Grains{k}.Vertices + repmat(-tVector,size(Grains{k}.Vertices,1),1);
```

```
49
50 end
51
52 % CLIPPING
53
54 % Get the neighbor-lists for all grains.
55 [Lists, N] = getNeighbors(Grains);
56
57 counter = 1;
58
59 % For each grain...
60 for k = 1:nGrains
61
62     Grain1 = Grains{k};
63
64     % For each 'neighbor' of k-th grain..
65     for q = 1:size(Lists{k},2)
66
67         % Print progress.
68         sprintf('%d/%d', counter, N)
69         counter = counter+1;
70
71         Grain2 = Grains{Lists{k}(q)};
72
73         % Explicitly compute convex intersection.
74         [iPoints, ips, K, Vol, flag] = ...
75             fastConvexIntersection(Grain1.Vertices, [], Grain1.Faces,...
76             Grain2.Vertices, [], Grain2.Faces);
77
78         % If no intersection, continue.
79         if(¬flag), continue, end;
80
81         % Compute clipping plane.
82         iCentroid = centroid(iPoints);
83         pNormal = normalizeVector3d(createVector(Grain1.Centroid,...
84             Grain2.Centroid));
85         cliPlane = createPlane(iCentroid,pNormal);
86
87         % Clip the pair of grains w.r.t. cliPlane.
88         [Grain1.Vertices, Grain1.Faces, Grain2.Vertices, Grain2.Faces]=...
89             createGrainBnd( Grain1.Vertices, Grain1.Faces,...
90             Grain1.Centroid,Grain2.Vertices,Grain2.Faces,cliPlane,2*eps );
91
92         % Update Vertices & Faces.
93         Grains{k}.Vertices = Grain1.Vertices;
94         Grains{k}.Faces = Grain1.Faces;
95         Grains{Lists{k}(q)}.Vertices = Grain2.Vertices;
96         Grains{Lists{k}(q)}.Faces = Grain2.Faces;
97
98         % Centroids updated after clipping by 'updateCentroids.m'.
99     end
100 end
101
102 if(pflag)
103     subplot(1,2,2), hold on, axis equal, view(3);
104     title('Final');
105     for k = 1:nGrains
106         drawMesh(Grains{k}.Vertices,Grains{k}.Faces,...
107             'facecolor','r','facealpha',0.3);
108         drawPoint3d(Grains{k}.Centroid);
```

```
109     end
110 end
111
112 % Save the final expanded & clipped sample.
113 % save('Alite100S1.9PckBlownUpClip.mat','Grains');
```

Listing A.5: createGrainBnd.m

```
1  function [v1,f1,v2,f2] = createGrainBnd(V1,F1,C1, V2,F2, cPlane,sep)
2  % Clips two convex grains to a plane.
3  %
4  % INPUT
5  %       V1,F1,V2,F2  Vertices & Faces of the two convex grains.
6  %       C1           Grain centroid.
7  %       cPlane       Clipping plane.
8  %       sep          Separation distance (not lower than eps).
9  %
10 % OUTPUT
11 %       v1,f1,v2,f2:  Vertices & Faces of the clipped grains.
12 %
13 if(sep<2*eps)
14     error('Separation distance too small (sep≥2*eps).');
15 end
16
17 % Distance from cPlane to push away each grain.
18 d = sep/2;
19
20 % It should be certain that both centroids cannot be on one side of cPlane.
21 if(isBelowPlane(C1,cPlane))
22
23     cPlane1 = parallelPlane(cPlane,-sep);
24     [v1,f1] = clipConvexPolyhedronHP(V1,F1,cPlane1);
25
26     % Reverse clipping plane orientation.
27     cPlane = cPlane([1 2 3 7 8 9 4 5 6]);
28
29     % Parallel translate it by distance d away from the other grain.
30     cPlane2 = parallelPlane(cPlane,-d);
31
32     [v2,f2] = clipConvexPolyhedronHP(V2,F2,cPlane2);
33
34 else
35
36     cPlane2 = parallelPlane(cPlane,-d);
37     [v2,f2]=clipConvexPolyhedronHP(V2,F2,cPlane2);
38
39     % Reverse clipping plane orientation.
40     cPlane = cPlane([1 2 3 7 8 9 4 5 6]);
41
42     % Parallel translate it by distance d away from the other grain.
43     cPlane1 = parallelPlane(cPlane,-d);
44
45     [v1,f1] = clipConvexPolyhedronHP(V1,F1,cPlane1);
46
47 end
48
49 return
```

Listing A.6: slice2d.m

```
% slice2d.m
clear all;
% Script that intersects the clinker sample with a plane and computes the
% 2D profiles of the grains.

% Load Clinker sample.
load('Alite100S1.9PckBlownUpClip.mat','Grains');

pflag = 1;

% Slice plane normal.
pNormal = [0 1 0];

% Point on the slice plane.
% pPoint = [0 -5.5 0];
pPoint = [0 0 0];

figure; hold on;
axis([-5 5 -6 6 -4 5]);
view(3);

for iter = 1:1

    % pPoint(2)= pPoint(2)+1.2;

    % Create slicing plane.
    sPlane = createPlane(pPoint,pNormal);

    % Intersect with all grains.
    for k = 1:size(Grains,1)

        v = Grains{k}.Vertices;

        % If they all lie on one side of sPlane, continue.
        if( all(isBelowPlane(v,sPlane)) || all(¬isBelowPlane(v,sPlane)) )
            continue;
        end

        f = Grains{k}.Faces;

        % Intersect the plane with the shape's mesh.
        iPoly = polyhedronSlice(v,f,sPlane);

        % Sample the characteristic you're interested in.
        if(isempty(iPoly)), continue; end;

        % Plot
        if(pflag)
            drawMesh(v,f,'facecolor','b');
            drawPolygon3d(iPoly,'color','black','LineWidth',2);
            axis equal, view(3);
        end

    end

    drawPlane3d(sPlane,'b');
    alpha(0.7);
    axis equal;
end
```

Listing A.7: analyzeInitConfig.m

```
% analyzeInitConfig.m
clear all;
% Script intended to visualize the initial configuration of grains.
%
% The initial configuration is loaded from the file 'SystemInit.mat', which
% is assumed to have been generated by the script initConfig.m .

% Load the initial configuration.
load('SystemInit.mat','Grains');
nGrains = size(Grains,1);

Volume = 0;                         % Volume fraction.
grainSize = zeros(nGrains,1);       % Grain size histogram.

R = 12; % Radius of the spherical confining volume).

% Initialize plot.
hold on;
axis([-R R -R R -R R]);
drawSphere([0 0 0],R,'facecolor','blue','facealpha',0.1);
view(3); axis equal; grid on;

for k = 1:nGrains

    Grain = Grains{k};

    Volume = Volume + Grain.Volume;
    grainSize(k) = Grain.Size;

    drawMesh(Grain.Vertices,Grain.Faces,'facecolor','r','facealpha',0.4);
    drawPoint3d(Grain.Centroid);

end

figure;
hist(grainSize);

% Rough estimate of the volume fraction.
(Volume*100) / ( (4/3)*pi*(R^3) )
```

Listing A.8: analyzePacking.m

```
% analyzePacking.m
clear all;
% Script intended to visualize the packed system.
%
% It plot the grains and a confining spherical volume, whose radius has to
% be manually inserted (R).
%
% The packed configuration is loaded from the file 'AlitePacked.mat', which
% is assumed to have been generated by the script packSystem.m .

% Load packed configuration.
load('AlitePacked.mat','Grains');

% Radius of the confining spherical volume.
R = 4;

```

```
% Total volume.
Vol = 0;

figure; hold on, axis([-R R -R R -R R]), view(3);
drawSphere([0 0 0],R,'facecolor','b','facealpha',0.1);

for k = 1:size(Grains,1)
    drawMesh(Grains{k}.Vertices,Grains{k}.Faces,...
        'facecolor','r','facealpha',0.4);
    drawPoint3d(Grains{k}.Centroid);

    % Compute grain volume.
    [~,kVol] = convhull(Grains{k}.Vertices);
    Vol = Vol + kVol;

end

% Estimate volume fraction, assuming the sphere
% of radius R represents the approximate Clinker's volume.
Vol*100/((4/3)*pi*(R^3))
```

Listing A.9: computeHState.m

```
% computeHState.m
clear all;
% Script that computes the h-vector (i.e. vectors from the centroid,
% perpendicularly pointing to each face with magnitude d(centroid,face)).
%
% NOTE: The centroid must be updated after clipping by the script
%       updateCentroids.m .

% Load Clinker sample.
load('Alite100S1.5PckBlownUpClip.mat','Grains');

% Plotting flag
pflag = 0;

% Sphere radius.
R = 5;

if(pflag)
    figure, hold on, axis([-R R -R R -R R]), view(3);
    drawSphere([0 0 0],R,'facecolor','b','facealpha',0.1);
end


for k = 1:size(Grains,1)

    if(pflag)
        drawMesh(Grains{k}.Vertices,Grains{k}.Faces,...
            'facecolor','r','facealpha',0.6);
    end

    Normals = faceNormal(Grains{k}.Vertices, Grains{k}.Faces);

    nFaces = size(Grains{k}.Faces,2);

    P = zeros(nFaces,3);

```

```
37      for j = 1:size(Grains{k}.Faces,2)
38
39          Line=createLine3d(Grains{k}.Centroid,...
40              Normals(j,1),Normals(j,2),Normals(j,3));
41
42          Plane=createPlane(Grains{k}.Vertices(Grains{k}.Faces{j}(1:3),:));
43
44          P(j,:) = intersectLinePlane(Line,Plane); % intersection point.
45
46      end
47
48      plot3( P(:,1),P(:,2),P(:,3), '.','color','blue' );
49
50      h = createVector(repmat(Grains{k}.Centroid,nFaces,1), P);
51      drawVector3d(repmat(Grains{k}.Centroid,nFaces,1),h,'b');
52  end
```

Listing A.10: getNeighbors.m

```
1   function [Lists, N] = getNeighbors(Grains)
2   % Computes the neighbor lists for all grains in cell array Grains.
3   %
4   % INPUT
5   %   Grains  A grain configuration.
6   % OUTPUT
7   %   Lists   The neighbour lists.
8   %   N       The total number of neighbours for all grains.
9
10  N = 0;
11  nGrains = size(Grains,1);
12
13  % All unique pairs.
14  [J I] = find(tril(squareform(ones(nGrains*(nGrains-1)/2,1))));
15
16  nPairs = length(J);
17
18  % Initialize neighbour lists.
19  Lists = cell(nGrains,1);
20
21  for k = 1:nPairs
22
23      if( Grains{I(k)}.Radius + Grains{J(k)}.Radius > ...
24              norm(Grains{I(k)}.Centroid - Grains{J(k)}.Centroid) )
25
26          % Update lists.
27          Lists{I(k)} = [Lists{I(k)} J(k)];
28
29          N = N+1;
30      end
31
32  end
```

Listing A.11: grainOverlaps.m

```
1   function flag = grainOverlaps(Grains,iGrain,gidx)
2   % True if Grain{gidx} overlaps with another from the cell-array Grains.
3   flag = 0;
```

```
4
5  for k = 1:(gidx-1)
6
7      % Make sure the computation of Grain.Radius gives always the
8      % expected result.
9
10     if( norm(Grains{k}.Centroid-iGrain.Centroid)  <= ...
11             Grains{k}.Radius + iGrain.Radius )
12
13         % check for overlap.
14         flag = convexIntersectionCheck(Grains{k}.Vertices,...
15             [],Grains{k}.Faces, iGrain.Vertices,[],iGrain.Faces);
16
17         if(flag), return; end
18
19     end
20
21 end
22
23 % skipping the iGrain itself...
24
25 for k = (gidx+1):size(Grains,1)
26
27     if( norm(Grains{k}.Centroid-iGrain.Centroid) <= ...
28             Grains{k}.Radius + iGrain.Radius )
29
30         flag = convexIntersectionCheck(Grains{k}.Vertices,...
31             [],Grains{k}.Faces, iGrain.Vertices,[],iGrain.Faces);
32
33         if(flag), return; end;
34
35     end
36
37 end
38
39
40 return;
```

Listing A.12: posGood.m

```
1  function flag = posGood(Grains,iGrainRadius,pos)
2  % POSGOOD True if pos guarantees no overlaps with other Grains.
3  flag = 0;
4  for k = 1:size(Grains,1)
5      if(isempty(Grains{k})), break; end;
6      if( norm(Grains{k}.Centroid - pos) <=  Grains{k}.Radius+iGrainRadius )
7          return;
8      end
9  end
10 flag=1;
11 return;
```

Listing A.13: updateCentroids.m

```
1  % updateCentroids.m
2  clear all;
3  % Script that updates the centroids after Clipping.
```

```
4
5  pflag = 0;
6
7  % Load clipped configuration.
8  load('Alite100S1.7PckBlownUpClip.mat','Grains');
9
10 nGrains = size(Grains,1);
11
12 % Update centroids.
13 for k = 1:nGrains
14     Grains{k}.Centroid = ...
15         polyhedronCentroid(Grains{k}.Vertices, Grains{k}.Faces);
16 end
17
18 % Plot
19 if(pflag)
20     figure;
21     for k = 1:nGrains
22         drawMesh(Grains{k}.Vertices,Grains{k}.Faces,...
23             'facecolor','r','facealpha',0.3);
24         drawPoint3d(Grains{k}.Centroid);
25     end
26     axis equal, view(3);
27 end
28
29 % Save final configuration.
```

Listing A.14: fastConvexIntersection.m

```
1  function [iPoints, ips, K, Vol, flag] = ...
       fastConvexIntersection(V1,¬,F1,V2,¬,F2)
2  % Computes the convex intersection of two convex polyhedra.
3  %
4  % INPUT
5  %   V1, F1  Vertices and Faces of a convex polyhedron.
6  %   V2, F2  Vertices and Faces of a second convex polyhedron.
7  %
8  % OUTPUT
9  %       iPoints  The points whose convex hull is the intersection.
10 %       ips      iPoints(ips,:) is the last point on a face of P1,P2, the
11 %                rest are interior.
12 %       K        Triangulation of the intersection.
13 %       Vol      Intersection Volume.
14 %       flag     1 if there is intersection, 0 otherwise.
15
16 iPoints=[];
17
18 [tri1, ¬] = triangulateFaces(F1);
19 [tri2, ¬] = triangulateFaces(F2);
20
21 % Intersect all triangles of tri1 with all of tri2.
22 for k1 = 1:size(tri1,1)
23
24     for k2 = 1:size(tri2,1)
25
26         Points1 = V1(tri1(k1,:),:); % Red.
27         Points2 = V2(tri2(k2,:),:); % White.
28
29         Plane1 = createPlane(Points1);
```

```
30          Plane2 = createPlane(Points2);
31
32          [IP12, flag12] = intersectTrianglePlane(Points1,Plane2);
33          [IP21, flag21] = intersectTrianglePlane(Points2,Plane1);
34
35          % If one of the triangle-plane intersections doesn't exist,
36          % there is no triangle-trianle intersection at all.
37          if(¬(flag12 & flag21)) %#ok<AND2>
38              continue;
39          end
40
41          % Intersection of the colinear segments IP12,IP21.
42          [Segm, sflag] = colinearSegmentIntersection(IP12,IP21);
43
44          if(sflag)
45              iPoints = [iPoints; Segm]; %#ok<*AGROW>
46          end
47      end
48  end
49
50  iPoints = unique(iPoints,'rows');
51
52  % ips is the index of the last point on a face, interior points follow.
53  ips = size(iPoints,1);
54
55  % vidx of V1 that lie within P2.
56  inIdx12 = inhull(V1,V2,tri2);
57  % vidx of V2 that lie within P1.
58  inIdx21 = inhull(V2,V1,tri1);
59
60  iPoints = [iPoints; V1(inIdx12,:)];
61  iPoints = [iPoints; V2(inIdx21,:)];
62
63  if(ips>0)
64      [K, Vol] = convhull(iPoints(:,1),iPoints(:,2),iPoints(:,3));
65      flag = 1;
66  else
67      K = [];
68      Vol = 0;
69      flag = 0;
70  end
71
72  return;
```

Listing A.15: convexIntersectionCheck.m

```
1  function iflag = convexIntersectionCheck(V1,¬,F1,V2,¬,F2)
2  % Checks for intersection of two convex polyhedra.
3  iflag = 0;
4
5  [tri1, ¬] = triangulateFaces(F1);  % This can be precomputed.
6  [tri2, ¬] = triangulateFaces(F2);  % This can be precomputed.
7
8  % If a vertex ov V1 lies within the convhull of V2, return 1.
9  if(sum(inhull(V1,V2,tri2))>0)
10     iflag=1;
11     return;
12 end
13
```

```
14 % If a vertex ov V2 lies within the convhull of V1, return 1.
15 if(sum(inhull(V2,V1,tri1)>0))
16     iflag=1;
17     return;
18 end
19
20 % Intersect all triangles of tri1 with all of tri2.
21 for k1 = 1:size(tri1,1)
22
23     for k2 = 1:size(tri2,1)
24
25         Points1 = V1(tri1(k1,:),:); % Red.
26         Points2 = V2(tri2(k2,:),:); % White.
27
28         Plane1 = createPlane(Points1);
29         Plane2 = createPlane(Points2);
30
31         [IP12, flag12] = intersectTrianglePlane(Points1,Plane2);
32         [IP21, flag21] = intersectTrianglePlane(Points2,Plane1);
33
34         % If one of the triangle-plane intersections doesn't exist,
35         % there is no triangle-trianle intersection at all.
36         if(¬(flag12 & flag21)) %#ok<AND2>
37             continue;
38         end
39
40         % Intersection of the colinear segments IP12,IP21.
41         [¬, iflag] = colinearSegmentIntersection(IP12,IP21);
42
43         if(iflag)
44             return;
45         end
46
47     end
48 end
49
50 return;
```

Listing A.16: intersectTrianglePlane.m

```
1  function [IP, flag] = intersectTrianglePlane(Points,Plane)
2  % Intersects a triangle with a plane.
3  %
4  % INPUT
5  %   Points  The vertices of the triangle.
6  %   Plane   The plane in the form returned by 'createPlane'/geom3d.
7  % OUTPUT
8  %   IP      2x3 matrix with rows the intersection points.
9  %   flag    1 if there is intersection, 0 otherwise.
10
11 % Do the triangle and the plane intersect?
12 Below = isBelowPlane(Points, Plane);
13
14 % If all vertices lie on one side of the plane, no intersection.
15 if(all(Below==Below(1)))
16     IP = [NaN NaN NaN];
17     flag = 0;
18     return;
19 end
```

```
20
21 idx1 = find(Below);          % Points below.
22 idx2 = setdiff(1:3,idx1);    % Points above.
23
24 if( length(idx1) > length(idx2) )
25     % (i.e. point idx2 is on one side and points idx1 on the other.)
26
27     % Segment 1.
28     seg1_start = Points(idx2,:);
29     seg1_end = Points(idx1(1),:);
30     Line1 = createLine3d(seg1_start,seg1_end);
31
32     % Segment 2.
33     seg2_start = Points(idx2,:);
34     seg2_end = Points(idx1(2),:);
35     Line2 = createLine3d(seg2_start,seg2_end);
36 else
37     % (i.e. point idx1 is on one side and points idx2 on the other.)
38
39     % Segment 1.
40     seg1_start = Points(idx1,:);
41     seg1_end = Points(idx2(1),:);
42     Line1 = createLine3d(seg1_start,seg1_end);
43
44     % Segment 2.
45     seg2_start = Points(idx1,:);
46     seg2_end = Points(idx2(2),:);
47     Line2 = createLine3d(seg2_start,seg2_end);
48 end
49
50 % Find the intersections of the two segments with the plane.
51
52 IP1 = intersectLinePlane(Line1,Plane);
53 IP2 = intersectLinePlane(Line2,Plane);
54
55 % Output
56 IP = [IP1; IP2];
57 flag = 1;
58
59 return;
```

Listing A.17: colinearSegmentIntersection.m

```
1  function [SegIP, flag] = colinearSegmentIntersection(seg1,seg2)
2  % Returns the intersection of two colinear segments.
3  %
4  % INPUT
5  %   seg1, seg2    2x3 matrices with an endpoint on each row.
6  %
7  % OUTPUT
8  %   SegIP   2x3 matrix with rows the endpoints of the intersection segment.
9  %   flag    Intersection flag: 0 if no intersection, 1 otherwise.
10
11 SegIP = [];
12 flag = 0;
13
14 if(size(seg1,1)>2 | size(seg2,1)>2) %#ok<*OR2>
15     error('Segment has not exactly two endpoints.');
16 end
```

```
17
18 % Project on appropriate axis.
19 axisId = appropriateProjectionAxis(seg1);
20 switch axisId
21     case 1
22         proj_axis = createLine3d([0 0 0],[1 0 0]); % x-axis
23     case 2
24         proj_axis = createLine3d([0 0 0],[0 1 0]); % y-axis
25     case 3
26         proj_axis = createLine3d([0 0 0],[0 0 1]); % z-axis
27 end
28
29 P = [seg1; seg2];
30
31 % Project the four endpoints on proj_axis.
32 proj_points = projPointOnLine3d(P, proj_axis);
33
34 % Sort points w.r.t. the appropriate coord.
35 [~, I] = sort(proj_points(:,axisId));
36
37 % If the segments are disjoint, return.
38 if( isempty(setdiff(1:2,I(1:2))) | isempty(setdiff(3:4,I(1:2))) )
39     return;
40 end
41
42 % Otherwise the intersection-segment's endpoints are the two middle ones.
43 SegIP = P(I(2:3),:);
44 flag = 1;
45
46 return;
```

Listing A.18: appropriateProjectionAxis.m

```
1 function axis = appropriateProjectionAxis(Points)
2 % Returns an axis index in [1 2 3] - for [x y z] - which corresponds to a
3 % valid projection axis, in terms of sorting colinear points. It simply
4 % avoids the axes perpendicular to the line defined by the points, that is.
5
6 % NOTE: Points are assumed colinear.
7
8 % Get indices of columns (i.e. coords) without recurring values.
9 A = find( any(Points(1:2,:)-repmat(Points(1,:),2,1)) );
10 axis = A(1); % Pick the first of the non-perpendicular axes.
11
12 return
```

Listing A.19: trackBoundary.m

```
1 % trackBoundary.m
2 clear all;
3 clf;
4 % Script that tracks the boundaries of Alite grains on a 2d micrograph.
5
6 % Load Black-and-White micrograph.
7 X=imread('clinker02.jpg');
8
9 % Convert to binary image.
```

```
10 I=im2bw(X,graythresh(X));
11
12 % Track boundaries.
13 [B,L,N] = bwboundaries(I,4); % watch out the connectivity
14
15 % Show image.
16 imshow(I);
17
18 hold on;
19
20 grainsize=zeros(length(B),1);
21
22 % Draw boundaries and compute grain sizes (max-calliper diameter).
23 for k = 1:length(B),
24     boundary = B{k};
25     if(k>N)
26         plot(boundary(:,2), boundary(:,1), 'g', 'LineWidth',2);
27     else
28         plot(boundary(:,2), boundary(:,1), 'r', 'LineWidth',2);
29     end
30     % Save the max diameter of all grains
31     grainsize(k)=max(pdist(boundary));
32 end
33
34 %figure;
35 %hist(grainsize(grainsize>5), 30)
```

Acknowledgements

I would hereby like to sincerely thank Dr. Falk Wittel and Dr. Miller Mendoza Jimenez for the numerous discussions we had regarding this thesis. Apart from assisting with technical advice, Dr. Wittel also gave me very useful advice with respect to the writing of the report. Dr. Mendoza's lasting support was essential for completing this work. Moreover, Dr. Ratan Mishra was kind enough to share with me his insights into the clinker's chemistry in several occasions. Last, but not least, I thank Prof. Dr. Hans Herrmann, who pointed to the right direction at the right moment.

Funding by The Alexander S. Onassis Public Benefit Foundation in Greece during the whole period of this Master's thesis is gratefully acknowledged.

References

[1] F.W. Taylor, Harry. *Cement Chemistry*. Thomas Telford, 2nd edition, 1997.

[2] Robert J. Flatt, Nicolas Roussel, and Christopher R. Cheeseman. Concrete: An eco material that needs to be improved. *Journal of the European Ceramic Society*, 32(11):2787 – 2798, 2012. Special Issue: ECerS XII, 12th Conference of the European Ceramic Society.

[3] J. A. Åström. Statistical models of brittle fragmentation. *Advances in Physics*, 55(3-4):247–278, 2006.

[4] Ratan K. Mishra. *Simulation of Interfaces in Construction Materials: Tricalcium Silicate, Gypsum, and Organic Modifiers*. PhD thesis, University of Akron, Polymer Engineering, 2012.

[5] Yongchun Zhang, Jacob P. Sizemore, and Michael F. Doherty. Shape evolution of 3-dimensional faceted crystals. *AIChE Journal*, 52(5):1906–1915, 2006.

[6] Robert Jagnow, Julie Dorsey, and Holly Rushmeier. Stereological techniques for solid textures. *ACM Trans. Graph.*, 23(3):329–335, August 2004.

[7] G. Mertens and J. Elsen. Use of computer assisted image analysis for the determination of the grain-size distribution of sands used in mortars. *Cement and Concrete Research*, 36(8):1453 – 1459, 2006. 10th EUROSEMINAR on microscopy applied to building materials, University of Paisley, June 21-25, 2005.

[8] Joachim Ohser and Frank Mücklich. *Statistical Analysis of Microstructures*. John Wiley & Sons, Ltd, 2000.

[9] Russ C. John. *Practical Stereology*. Plenum Press, 1986.

[10] Paul Stutzman. Scanning electron microscopy imaging of hydraulic cement microstructure. *Cement and Concrete Composites*, 26(8):957 – 966, 2004. Scanning electron microscopy of cements and concretes.

[11] Rafael C. Gonzalez, Richard E. Woods, and Steven L. Eddins. *Digital Image Processing Using MATLAB*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003.

[12] Elizabeth A. Holm and Corbett C. Battaile. The computer simulation of microstructural evolution. *JOM*, 53(9):20–23, 2001.

[13] M.P Anderson, D.J Srolovitz, G.S Grest, and P.S Sahni. Computer simulation of grain growth - I. kinetics. *Acta Metallurgica*, 32(5):783 – 791, 1984.

[14] B. Radhakrishnan and T. Zacharia. Simulation of curvature-driven grain growth by using a modified Monte Carlo algorithm. *Metallurgical and Materials Transactions A*, 26:167–180, 1995.

[15] Qiang Yu and Sven K. Esche. A Monte Carlo algorithm for single phase normal grain growth with improved accuracy and efficiency. *Computational Materials Science*, 27(3):259 – 270, 2003.

[16] C. Ming Huang, C.L. Joanne, B.S.V. Patnaik, and R Jayaganthan. Monte Carlo simulation of grain growth in polycrystalline materials. *Applied Surface Science*, 252(11):3997 – 4002, 2006. ICMAT 2005: Symposium L.

[17] O.M. Ivasishin, S.V. Shevchenko, and S.L. Semiatin. Implementation of exact grain-boundary geometry into a 3-d Monte-Carlo (Potts) model for microstructure evolution. *Acta Materialia*, 57(9):2834 – 2844, 2009.

[18] L.-Q. Chen. Novel computer simulation technique for modeling grain growth. *Scripta Metallurgica et Materialia*, 32:115–120, January 1995.

[19] Long-Qing Chen and Yunzhi Wang. The continuum field approach to modeling microstructural evolution. *JOM*, 48(12):13–18, 1996.

[20] Liangliang Liu, Feng Gao, Guoxin Hu, and Jiangnan Liu. Phase field simulation for the evolution of textured ceramics microstructure. *Ceramics International*, 38(7):5425 – 5432, 2012.

[21] Long-Qing Chen. Phase field model for microstructure evolution. *Annual Review of Materials Research*, 32(1):113–140, 2002.

[22] Mark A. Miodownik. A review of microstructural computer models used to simulate grain growth and recrystallisation in aluminium alloys. *Journal of Light Metals*, 2(3):125 – 135, 2002. Modelling of Light Metals.

[23] Kyozi Kawasaki, Tatsuzo Nagai, and Katsuya Nakashima. Vertex models for two-dimensional grain growth. *Philosophical Magazine Part B*, 60(3):399–421, 1989.

[24] D. Weygand, Y. Brechet, and J. Lepinoux. A Vertex simulation of grain growth in 2d and 3d. *Advanced Engineering Materials*, 3(1-2):67–71, 2001.

[25] Y. Liu, T. Baudin, and R. Penelle. Simulation of normal grain growth by Cellular Automata. *Scripta Materialia*, 34(11):1679 – 1683, 1996.

[26] J. Geiger, A. Rosz, and P. Barkczy. Simulation of grain coarsening in two dimensions by Cellular Automaton. *Acta Materialia*, 49(4):623 – 629, 2001.

[27] S. Raghavan and Satyam S. Sahay. Modeling the grain growth kinetics by Cellular Automaton. *Materials Science and Engineering: A*, 445446(0):203 – 209, 2007.

[28] S. Raghavan and Satyam S. Sahay. Modeling the topological features during grain growth by Cellular Automaton. *Computational Materials Science*, 46(1):92 – 99, 2009.

[29] S.A. Galindo-Torres, D.M. Pedroso, D.J. Williams, and L. Li. Breaking processes in three-dimensional bonded granular materials with general shapes. *Computer Physics Communications*, 183(2):266 – 277, 2012.

[30] M. Lu and G.R. McDowell. The importance of modelling ballast particle shape in the Discrete Element Method. *Granular Matter*, 9:69–80, 2007.

[31] Y. Liu and Z. You. Visualization and simulation of asphalt concrete with randomly generated three-dimensional models. *Journal of Computing in Civil Engineering*, 23(6):340–347, 2009.

[32] D. Zhang, X. Huang, and Y. Zhao. Algorithms for generating three-dimensional aggregates and asphalt mixture samples by the Discrete-Element Method. *Journal of Computing in Civil Engineering*, 27(2):111–117, 2013.

[33] Linbing Wang, Jin-Young Park, and Yanrong Fu. Representation of real particles for DEM simulation using X-ray tomography. *Construction and Building Materials*, 21(2):338 – 346, 2007.

[34] Christian Borchert and Kai Sundmacher. Efficient formulation of crystal shape evolution equations. *Chemical Engineering Science*, 84(0):85 – 99, 2012.

[35] David Legland. geom3d. http://www.mathworks.com/matlabcentral/fileexchange/24484, June 2009.

[36] John D'Errico. inhull. http://www.mathworks.com/matlabcentral/fileexchange/10226-inhull, September 2012.

[37] Oliver Woodford. Exportfig. http://www.mathworks.com/matlabcentral/fileexchange/23629-exportfig, December 2012.

[38] http://geomalgorithms.com/a06-_intersect-2.html.

[39] S. Torquato and Y. Jiao. Dense packings of the Platonic and Archimedean solids. *Nature*, 460:876–879, August 2009.

[40] C. Ericson. *Real-Time Collision Detection*. CRC Press, har/cdr edition, 2004.

[41] H. A. Carmona, F. K. Wittel, F. Kun, and H. J. Herrmann. Fragmentation processes in impact of spheres. *Phys. Rev. E*, 77:051302, May 2008.

[42] G. Timár, F. Kun, H. A. Carmona, and H. J. Herrmann. Scaling laws for impact fragmentation of spherical solids. *Phys. Rev. E*, 86:016113, Jul 2012.