

UNIVERSITY OF BIRMINGHAM

Development of a microscale numerical modelling method for the simulation of 3D powder bed printing of Bioceramics

by

MOHAMMAD ALI AUSAF QURESHI

A thesis submitted to the
University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Metallurgy and Materials
College of Engineering and Physical Sciences
The University of Birmingham
Birmingham B15 2TT
United Kingdom
March 2023

UNIVERSITY OF
BIRMINGHAM

University of Birmingham Research Archive

e-theses repository

This unpublished thesis/dissertation is copyright of the author and/or third parties. The intellectual property rights of the author or third parties in respect of this work are as defined by The Copyright Designs and Patents Act 1988 or as modified by any successor legislation.

Any use made of information contained in this thesis/dissertation must be in accordance with that legislation and must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the permission of the copyright holder.

ABSTRACT

Significant progress has been made in powder bed selective laser processing (PBSLP) of 3D printing of metals and polymers. Researchers have developed numerical models to simulate these processes to develop a better understanding, both at macro and microscales. However, due to their high thermal resistance, laser processing of ceramic materials presents unique challenges. Therefore, this resea rch focused on modelling and simulating the PBSLP of bioceramics to develop a closer insight into its processing at the microscale. Hydroxyapatite (HAP) and alumina powders were particularly addressed, due to their broad applications in the medical industry, including orthopaedic extremities.

The printing process consists of a pre-processing phase involving the powder bed deposition and layer formation, followed by a laser processing phase that selectively scans the powder region to generate heat for sintering. Therefore, we adopted a discrete element method (DEM) tool, which could model the dynamics of discrete spherical particles virtually at any scale (as low as nano).

In the preprocessing phase, the impact of the circular profile blade coater was investigated on packing density and surface roughness for 5 different size distributions. For each sample, 4 layers were deposited with varying layer thicknesses of 100, 150, and 200 µm. The 100 and 150 µm thicknesses exhibited considerable voids in the initial layers, indicating loose packing densities—however, the effect of layer size diminished by the fourth layer, resulting in comparable final packing densities. As the DEM methodology enabled a closer examination of powders' temporal and dynamic behaviour during the deposition process, it was observed that voids in the smaller layer thicknesses were caused by the smaller ground clearance of

the blade, which swept away the larger particles. The visualisation of the deposition process simulations also demonstrated how those voids closed as more layers were added. Furthermore, a ray tracing code was also developed to assess each sample's surface quality, revealing that distributions with smaller average particle sizes yielded finer surfaces.

In the laser processing phase, ray tracing laser heating and particles sintering codes were developed and coupled with the DEM solver to implement a limited-scale PBSLP model. The impact of various fibre laser parameters, such as power (30-54 W) and scan speed (25-125 mm s^{-1}), on the sintering of HAP powder was investigated. Simulations correctly predicted the sintering responses for 72% of the parameters compared to the experiments. 12% of the laser parameters in simulations overestimated the results by predicting melting temperatures instead of sintering, while 16% underestimated the results by predicting no sintering while they occurred in the experiments. Combining DEM models with laser ray tracing and sintering codes provided valuable insights into the PBSLP. The heat distribution field resulting from the laser scan could be precisely modelled and investigated within the random-sized powders. The sintering processes against different laser parameters were better understood. The slower laser scan speeds for the same power improved the densification of powders. Higher power and lower speeds resulted in powder melting, while lower powers and speeds produced no sintering response. This methodology could also identify the combination of laser parameters that produced a solid-state sintering phase.

This thesis is dedicated to my late mother (Azra Jabeen Qureshi)

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Allah Almighty for this blessed opportunity.

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska-Curie grant agreement No 764935 (www.doc-3d-printing.eu). I would like to thank and acknowledge this funding source from European Commission that made this project possible.

I would like to express my deepest gratitude and respect to my supervisors Prof Artemis Stamboulis and Dr William David Griffiths, for their continuous support, understanding of my circumstances and kind supervision throughout my PhD studies. I would also like to thank Dr Kit Windows-Yule (Chemical Engineering) for his time-to-time advise on developing the DEM simulations, Dr Qirong Chen, Dr Asif ur Rehman and Dr Pedro Jesus Navarrete-Segado for providing experimental data.

I am forever grateful to my beloved parents (Attiq-ur-Rehman Qureshi and Shahida Mehmood Qureshi) and sister (Noor Fatima Qureshi), who unconditionally supported me through the times of doubt in my personal and academic life leading to this PhD. The care and support of my wife (Tabinda Masood) and the cheeky presence of my beloved 2-year-old daughter (Fatima Ali Qureshi) made this journey possible. In memory of my late uncle (Zafar Iqbal Qureshi), who has always been a source of inspiration and guidance.

I would like to thank all my colleagues and friends from the Biomaterials Research Group, who were always friendly and polite and made our office a great place to work. Nevertheless, my special thanks to my close friends (Dr.-Ing. Qaisar Nawaz and Ahmad Taha) for keeping me motivated in my scientific and personal life.

THIS PAGE WAS INTENTIONALLY LEFT BLANK

ABBREVIATIONS

AM	Additive manufacturing
AOR	Angle of repose
API	Application programming interface
BCRC	Belgian Ceramic Research Centre
CFD	Computational fluid dynamics
CIRIMAT	Centre Inter-Universitaire de Recherche et d'Ingénierie des Matériaux
DEM	Discrete element method
DMLS	Direct metal laser sintering
DOC-3D-Printing	Horizon 2020 Marie Curie ITN on the Development of ceramics 3D printing
FEA	Finite element analysis
FEM	Finite element method
GUI	Graphical user interface
HAP	Hydroxyapatite
HM	Hertz-Mindlin
ID	Identity
LAMMPS	Large-scale atomic/molecular massively parallel simulator
LIGGGHTS	LAMMPS improved for general granular and granular heat transfer simulations
MOOSE	Multiphysics object-oriented simulation environment
PB	Powder bed

PBSLP	Powder bed selective laser processing
PD	Packing density
PEEK	Polyether ether ketone
RTP	Ray to particle
SLM	Selective laser melting
SLS	Selective laser sintering
UoB	University of Birmingham
UYS	Unconfined yield strength
VOF	Volume of fluid

TABLE OF CONTENTS

1. INTRODUCTION 1

1.1. BACKGROUND OF THE PROJECT 2
1.2. AIMS AND OBJECTIVES 3
1.2.1 PREPROCESSING PHASE: POWDER DELIVERY 3
1.2.2 PROCESSING PHASE: SELECTIVE LASER PROCESSING 3
1.3. INTRODUCTION TO THE THESIS 4

2. LITERATURE REVIEW 7

2.1. LASER PROCESSING OF POWDERS USING PBSLP 8
2.2. FINITE ELEMENT METHOD (FEM) 10
2.2.1 BACKGROUND 10
2.3. DISCRETE ELEMENT METHOD (DEM) 11
2.3.1 BACKGROUND 11
2.3.2 CONTACT THEORY MODELS 13
2.3.3 LINEAR SPRING-DASHPOT MODEL 14
2.3.4 HERTZ MODEL 15
2.3.5 HERTZ-MINDLIN MODEL 16
2.3.6 HERTZ-MINDLIN MODEL WITH JKR COHESION 18
2.3.7 COMPUTATIONAL TIME-STEP FOR DEM SIMULATIONS 20
2.3.8 CALIBRATION OF POWDERS IN DEM 22
2.4. MODELLING OF SOLID-STATE SINTERING MECHANISM 26
2.4.1 FOR FINITE ELEMENT METHOD: CONTINUUM VISCOELASTIC MODEL 29
2.4.2 FOR DISCRETE ELEMENT METHOD 31
2.4.3 NUMERICAL STABILITY AND MASS SCALING OF PARTICLES IN DEM 33
2.5. MODELLING OF HEAT CONDUCTION IN POWDER BEDS 34
2.5.1 FOR FINITE ELEMENT METHOD 34
2.5.2 FOR DISCRETE ELEMENT METHOD 38
2.6. STATE-OF-THE-ART IN NUMERICAL SIMULATIONS OF SINTERING AND PBSLP 42

2.6.1 FINITE ELEMENT METHOD (FEM) 42
2.6.2 DISCRETE ELEMENT METHOD (DEM) FOR LASER HEATING & SINTERING 47
2.6.3 DISCRETE ELEMENT METHOD (DEM) FOR POWDER DELIVERY SYSTEMS 51
2.7. SUMMARY 57

3. CERAMIC POWDER GENERATION AND CALIBRATION USING DEM 59

3.1. INTRODUCTION 60
3.2. ALUMINA POWDERS 62
3.2.1 DEM POWDER GENERATION AND VOLUMETRIC DISTRIBUTION 62
3.2.2 POWDER CALIBRATION FOR SIMULATIONS 72
3.2.3 SIMULATION SETUP, PARAMETERS, AND PROCESS 76
3.2.4 POWDER DENSITY EVALUATION METHOD AND RESULTS....82
3.2.5 SURFACE ROUGHNESS EVALUATION USING THE RAY TRACING METHOD....89
3.2.6 RESULTS AND DISCUSSION 96
3.3. HYDROXYAPATITE POWDERS....100
3.3.1 DEM POWDER GENERATION AND VOLUMETRIC DISTRIBUTION 100
3.3.2 POWDER CALIBRATION FOR SIMULATIONS 103
3.3.3 POWDER BED COMPRESSION 104

4. LASER HEAT SOURCE AND ITS THERMAL INTERACTION..... 105

4.1. INTRODUCTION 106
4.2. GENERAL MODELLING OF LASER HEAT FLUX....108
4.3. RAY-TRACING AND DEM COUPLING 110
4.3.1 BEAM DISCRETIZATION AND ENERGY ALLOCATION....110
4.3.2 MINIMUM CELL SIZE FOR PARTICLE DETECTION....113
4.3.3 THE RAY TRACING METHOD AND ALGORITHM 115
4.4. INTERCEPTION, REFLECTIONS, AND TRANSMISSIONS....120
4.4.1 RAY-PARTICLE INTERCEPTION....120
4.4.2 RAY REFLECTION 122
4.4.3 RAY TRANSMISSION (REFRACTION) AND TOTAL EXTERNAL REFLECTION...125
4.4.4 ALGORITHM PITFALLS AND TROUBLESHOOTING 130

4.5. LASER AND POWDER-BED THERMAL INTERACTION 133
4.5.1 MINIMUM PARTICLE SIZE FOR LASER INTERACTION 133
4.5.2 LASER ABSORPTION 134
4.5.3 REFRACTIVE INDEX 135
4.5.4 MODELLING OF CONVECTIVE AND RADIATIVE HEAT LOSSES 136
4.5.5 BIOT NUMBER 143
4.6. TESTING AND ANALYTICAL VALIDATION OF SIMPLIFIED CASE 145
4.7. SIMULATIONS SETUP AND PARAMETERS 147
4.7.1 SIMULATION SETUP IN FEM DOMAIN 147
4.7.2 SIMULATION SETUP IN DEM DOMAIN 149
4.8. LASER AND MATERIAL PARAMETERS 151
4.9. RESULTS AND DISCUSSION 153
4.9.1 CO_2 LASER VS Nd:YAG LASER 154
4.9.2 FEM VS DEM DOMAIN 160

5. SELECTIVE LASER SINTERING 163

5.1. INTRODUCTION 164
5.2. SINTERING CONTACT USING DEM 165
5.2.1 STOPPING FORCE AND TORQUE 165
5.2.2 STRESS RELAXATION 169
5.2.3 SINTERING, REPULSIVE AND BONDING FORCES 170
5.2.4 MASS SCALING OF PARTICLES 174
5.2.5 SUMMARY 177
5.3. THERMAL CONDUCTIVITY MODEL AND OPTIMISATION 178
5.4. COMBINATION OF SUB-MODELS 181
5.4.1 CONDITIONS AND SEQUENCE FOR APPLYING RELEVANT FORCES 181
5.4.2 CONDITIONS AND SEQUENCE FOR APPLYING THERMAL MODELS 183
5.5. SIMULATIONS SETUP, PARAMETERS, AND OPTIMIZATION 184
5.5.1 SETUP, EXECUTION, AND RESULTS VISUALIZATION 184
5.5.2 SIMULATION PARAMETERS 187
5.5.3 SIMULATION OPTIMIZATION PARAMETER 191

5.6. RESULTS AND DISCUSSION 193
5.6.1 HYDROXYAPATITE WITHOUT ADDITIVES 193
5.6.2 HYDROXYAPATITE WITH 10% GRAPHITE ADDITIVE 195

6. CONCLUSIONS AND OUTLOOK 201

6.1. CONCLUSIONS 202
6.2. LIMITATIONS, FUTURE WORK, AND IDEAS 205

REFERENCES 207

APPENDIX-A: TABLES 225

APPENDIX-B: FORTRAN SUBROUTINES FOR ABAQUS CAE 227

APPENDIX-C: RAY-TRACING CODE FOR SURFACE ROUGHNESS 229

APPENDIX-D: RAY-TRACING CODE FOR LASER HEATING OF POWDERS 235

APPENDIX-E: THERMAL CONDUCTIVITY CODE 250

APPENDIX-F: THERMAL LOSS CODE 253

APPENDIX-G: STOPPING FORCE AND TORQUE CODE 254

APPENDIX-H: SOLID STATE SINTERING CODE 255

APPENDIX-I: GRAPHICAL RESULTS FOR ALL LASER SETTINGS 271

LIST OF ILLUSTRATIONS

Figure 2.1. The schematic of the selective laser sintering (SLS) process, illustrating the overall 3D printing process. The roller or blade delivers the powder to the fabrication powder bed from the powder supply container/bed after each layer has been laser processed. The excess powder is accumulated in the separate containers to avoid material wastage [10]. Reuse license: 5517331371643 (Elsevier – Comprehensive Materials Processing). 9

Figure 2.2. Classification of particle interaction force models by contact force and non-contact force [30]. Reuse license: Open Access / Creative Commons. 13

Figure 2.3. (a) General relationship between force and displacement in Hertz-Mindlin model: No contact force exists when displacement or overlap is zero. The overlap induces this HM force [30]. **(b)** Schematic of Hertz-Mindlin contact model between two particles consisting of parallel configuration of spring and damping components [35]. Reuse license: Open Access / Creative Commons.....16

Figure 2.4. (a) Contact between two spherical bodies as defined by the JKR **(b)** Stress distribution curves for the JKR [32]. Reuse license: 1338159-1 (Copyright Clearance Centre, Inc. on behalf of HARRISON AND SON).....18

Figure 2.5. General relationship between force and displacement in Hertz-Mindlin (HM) + JKR model. In contrast to simple HM, HM + JKR force has attraction component that induces force even when the particles are not in contact. [30]. Reuse license: Open Access / Creative Commons. 19

Figure 2.6. Cylinder method: **(a)** Particles generation within the cylindrical volume; **(b)** Settlement of particles; **(c)** Heap formation. Boxed compartments method: **(d)** Particles generation in the upper compartment, with a trap door, to prevent the flow of particles

initially; **(e)** Once the particles are settled, the trap door is opened, and they are allowed to flow through the rectangular opening; **(f)** Heap formation in the lower compartment. 23

Figure 2.7. Measurement of dynamic angle of repose (or avalanche angle) in DEM. 23

Figure 2.8. Schematics of the formation of the dihedral angle between two identical touching cylinders, **(a)** particles in touch, **(b)** neck formation, $0 < \emptyset < \emptyset_e$. Reuse license: 5516180873657 (Springer Nature – Journal of Materials Science). 26

Figure 2.9. Three-sphere model showing the six transport paths from **Table 2.3**. The paths 1,2, and 3 represent surface, lattice, and vapour diffusion of matter, respectively, with matter source as surface. While the path 4 represent boundary diffusion with grain boundary as matter source, while the paths 5, and 6 represent lattice diffusion, with grain boundary and dislocation as matter source respectively. In all case, the matter sinks through the neck of the interacting particles, but only path 4 to 6 lead to densification [48]. 28

Figure 2.10. Rheological scheme of thermo-viscoelastic model, consisting of sintering force in parallel to tangential and viscous force components, with compressive forces exerted at both ends [49]. Reuse license: 5516190219018 (Elsevier – Computational Material Science). 29

Figure 2.11. The thermal conductivity of alumina powder (porosity = 0.71) by measurements (Landolt-Bornstein), compared with the values predicted by Equation (25), Zehner-Bauer's equation, and Yagi-Kunii's equation [64]. Reuse license: upst/03037406 (Taylor & Francis Group). 36

Figure 2.12. Normalized data of the thermal conductivity of powder beds (k/k_g) compared to the normalized predicted thermal conductivity values according to Equation [64]. Reuse license: upst/03037406 (Taylor & Francis Group). 36

Figure 2.13. Thermal conductivity data and correlations for the 0.01 and 1.0 µm Al_2O_3 powders, with smaller powders exhibiting higher conductivities, while both indicate

temperature dependence [66]. Reuse license: 5516200122116 (Elsevier - Energy Conversion and Management). 37

Figure 2.14. Contact radius (aij) and normal overlap (δN) of the interacting particles i and j. 38

Figure 2.15. Contact of radius a between two spherical particles with the same radius R and temperatures T_1 and T_2 illustrating their immediate heat affected zone during thermal conduction [70]. Reuse license: 5516200464214 (Elsevier - International Journal of Heat and Mass Transfer). 40

Figure 2.16. Effective thermal conductivity of Gusarov et al. model powder bed versus contact size, compared against the face centre cubic (FCC), body centred cubic (BCC), and simple cubic (SC) structure, displaying a good agreement. Additionally the normalized conductivity of diamond structure has also been compared with the rest of the models [70]. Reuse license: 5516200464214 (Elsevier - International Journal of Heat and Mass Transfer).....41

Figure 2.17. The 3D finite element model, illustrating the cuboidal DCD8 mesh of the laser scanned body [71]. Reuse license: nmcm/03037405 (Taylor & Francis Group) 43

Figure 2.18. SEM images under different conditions: **(a)** the initial nano-HAP, **(b-d)** the sintered nano-HAPs when the laser scanning speeds are 600, 200 and 120 mm/min, respectively [71]. Reuse license: nmcm/03037405 (Taylor & Francis Group) 43

Figure 2.19. (a) 3D view of Construction of the 3D multi-layer powder stacking model: 3D schematic for the simulation + Laser irradiation region. **(b)** Front view [73]. Reuse license: 5516210671050 (Elsevier - Journal of Materials Processing Technology). 44

Figure 2.20. Comparison between the experimental and simulated results in the **(a)** length of the sintered piece and **(b)** shrinkage depth [73]. Reuse license: 5516210671050 (Elsevier - Journal of Materials Processing Technology). 44

Figure 2.21. The meshed powder-substrate structure in the FVM model: Mesh is finer near the surfaces to capture the drastic change in t:emperature due to laser scan. [75]. Reuse license: 5516210860084 (Elsevier - Ceramics International). 46

Figure 2.22. (a). The numerical model geometry used for the study of past publications, with dark gray top slack representing the powder layer, while light grey base representing the substrate **(b).** Comparison between numerical and experimental results of melt track width in different studies, illustrating their absolute errors [77]. Reuse license: Open Access / Creative Commons.....46

Figure 2.23. Temperature distributions in the DMLS process simulations. **(a)** Vertical deposition of particles, **(b)** recoating process, **(c)** laser heating of particles, and **(d)** holding period. The particles are labelled using colours to present their temperature (unit: K) [11]. Reuse license: 5516220269197 (Elsevier - Powder Technology). 47

Figure 2.24. (a) Schematics for distribution of the volumetric laser energy deposition. **(b)** Relative temperature field induced as a result of laser scan (with different processing parameters) and resulting particle sintering/melting, after laser beam has travelled over half of powder bed length [80]. Reuse license: Open Access / Creative Commons. 49

Figure 2.25. Representation of the roller based powder delivery simulation setup by Haeri et al., representing the roller geometry, differently aspected powder bed layering, periodic and neutral wall boundaries [83]. Reuse license: Open Access / Creative Commons. 52

Figure 2.26. The effects of particle shape, $V_{spreader}$, $\delta_{spreader}$ and the spreader-type on the bed quality. The quality is quantified by a volume fraction ϕ, which is shown to be increasing up to 1.5 aspect ratio (A_r) and then decreasing, and roughness parameter ϵ, which continues to increase with increasing aspect ratio [83]. Reuse license: Open Access / Creative Commons. 52

Figure 2.27. (a) Different spreader profiles of a modified blade profile proposed by Haier, that was used for the optimization **(b)** Representation of the device simulation set-up, modified blade geometry, initial preparations, periodic and neutral boundaries [85]. Reuse license: Open Access / Creative Commons. 53

Figure 2.28. Comparison between a roller and the optimised blade type spreader (the legend applies to both figures). Clearly, the optimised blade-type spreader produces a powder bed with volume fractions at least as good as the roller and surface roughness values significantly better [85]. Reuse license: Open Access / Creative Commons. 54

Figure 2.29. Representation of a DEM simulation setup of roller-spreading process, feeding cylinder, and build platform (for laser processing) by Zhang et al. [18]. Reuse license: Open Access / Creative Commons..... 55

Figure 2.30. The powder-bed characteristics under different layer thickness values: **(a)** H = 50 µm representing signifant voids, **(b)** H = 75 µm representing reduced voids compared to 50 µm but still visible, and **(c)** H = 150 µm with no visible voids [18]. Reuse license: Open Access / Creative Commons. 55

Figure 3.1. Powder SP-1: **(a)** Experimental and simulation generated data comparison of volume distribution, **(b)** optical laser image visualization in contrast to **(c)** DEM. 67

Figure 3.2. Powder SP-2: **(a)** Experimental and simulation generated data comparison of volume distribution, **(b)** optical laser image visualization in contrast to **(c)** DEM. 68

Figure 3.3. Powder SP-3: **(a)** Experimental and simulation generated data comparison of volume distribution, **(b)** optical laser image visualization in contrast to **(c)** DEM. 69

Figure 3.4. Powder SP-4: **(a)** Experimental and simulation generated data comparison of volume distribution, **(b)** optical laser image visualization in contrast to **(c)** DEM. 70

Figure 3.5. Powder SP-5: **(a)** Experimental and simulation generated data comparison of volume distribution, **(b)** optical laser image visualization in contrast to **(c)** DEM. 71

Figure 3.6. Temporal evolution of bulk angle of repose test for alumina SP-3 powders (as an example for general flow) in DEM domain: **(a)** Initial domain (height cropped from 1.5 mm to 1.0 mm for illustration); **(b)** Powders generation in the prescribed volume of the domain; **(c)** Powders settling and adapting to the shape of slanted slits; **(d)** Powders start to escape (flow out) through the narrow slit; **(e)** Powders first hitting the base, i.e., lower boundary of the domain; **(f)** Formation of initial heap on the base, while remaining powders continue to fall on top; **(g)** Final heap formation, used for measurement of angle of repose. 74

Figure 3.7. Comparison of simulated and experimented static angle of repose used for calibration of alumina powders **(a)** SP-1, **(b)** SP-2, **(c)** SP-3, **(d)** SP-4, and **(e)** SP-5. Images from the AOR experiments on the right side are courtesy of Dr Qirong Chen (Belgian Ceramic Research Centre, Belgium).....75

Figure 3.8. Illustration of the simulation domain within EDEM graphical user interface: **(a)** Dimensions of the simulation domain and particles generator volume (light grey) inside; **(b)** Particles being generated inside the virtual box/volume.....76

Figure 3.9. Illustration of bulk powder being pushed forward, and generic forces are coming into play. 77

Figure 3.10. Unit cell/domain for a periodic boundary condition: **(a)** Particle travelling in a downward direction towards the base, **(b)** exiting the cell, and then **(c)** appearing from the opposite side with the same travel direction and velocity. 78

Figure 3.11. General illustration (not to scale) for the path followed by the blade. 79

Figure 3.12. Temporal evolution of layer formation process for alumina SP-1 powders (as a general example) in DEM domain: **(a)** At t = 0.001, powder for the first layer is generated **(b)**

At t = 0.0125, the generated powder settles down and the blade starts to move; **(c)** Layer smoothing in process; **(d)** Completion of first layer formation process. The same steps are followed for rest of the layers, for instance: **(e)** Blade is now raised to the thickness of fourth layer, and at the same time powder for the fourth layer is deposited on top of third layer; **(f)** Fourth layer smoothing in process; **(g)** Completion of fourth layer formation process; **(h)** Blade reset to its original horizontal position. 80

Figure 3.13. Illustration of the final powder bed (i.e., post blade run): **(a)** Complete powder bed with heaps near the walls of the domain; **(b)** Clipped powder bed region considered for calculating the density. The clipping planes appear as 'lines' because the figure projects the side view. 82

Figure 3.14. Change in packing density of **alumina powder SP-1** w.r.t. the addition of layers for different layer thicknesses of 100 µm, 150 µm, and 200 µm. 84

Figure 3.15. Change in packing density of **alumina powder SP-2** w.r.t. the addition of layers for different layer thicknesses of 100 µm, 150 µm, and 200 µm. 85

Figure 3.16. Change in packing density of **alumina powder SP-3** w.r.t. the addition of layers for different layer thicknesses of 100 µm, 150 µm, and 200 µm. 86

Figure 3.17. Change in packing density of **alumina powder SP-4** w.r.t. the addition of layers for different layer thicknesses of 100 µm, 150 µm, and 200 µm. 87

Figure 3.18. Change in packing density of **alumina powder SP-5** w.r.t. the addition of layers for different layer thicknesses of 100 µm, 150 µm, and 200 µm. 88

Figure 3.19. (a). Discretization of the square domain into smaller square cells (denoted by integer R); **(b).** Setting up of rays' centre positions (xR, yR). 89

Figure 3.20. (a) Depiction of rays falling on the powder bed, up to the point where they intersect the first powder particle; **(b)** Remaining rays with varying lengths illustrating the effect of powder bed surface profile. 90

Figure 3.21. Possible ray-particle or line-sphere interactions; **(a).** No interception occurs; **(b).** Tangential contact at a single point; **(c).** Ray path interfering with the particle, with two possible contact points, where only the entry contact point is considered. 91

Figure 3.22. Calculation and determination process for ray-particle interception, analogous to line-sphere intersection. This flow chart also represents the expanded/sub process given in **Figure 4.5-a** in the later **Section 4.3.3**. 94

Figure 3.23. Variation in **flowability** of powders in terms of angle of repose (AOR), w.r.t. composition of SP-1 and SP-5 powders. 96

Figure 3.24. Change in **packing density** of alumina powder w.r.t. percentage composition of SP-1 and SP-5 powders for initial and final layers and different layer thicknesses of 100 µm, 150 µm, and 200 µm. 97

Figure 3.25. Illustration of the first layer of 100 µm size producing visually significant voids in all cases, growing with increasing the powder size. 98

Figure 3.26. Variation in **surface roughness** of alumina powder w.r.t. percentage composition of SP-1 and SP-5 powders for the final layer of 150 µm. 99

Figure 3.27. (a) SEM micrograph for commercial Urodelia® 300-08-2 powder [88] and **(b)** DEM generation of the powders with the same size distribution. 101

Figure 3.28. Comparison of simulated and experimented static angle of repose used for calibration of hydroxyapatite powders: Urodelia® 300-08-2. 103

Figure 3.29. (a) Pre-compressed powder bed: ~300 µm. **(b)** Post-compressed powder bed: 100 µm, showing unrealistic overlaps. 104

Figure 4.1. (a) Schematic diagram of laser beam irradiating on the powder bed surface [92]; **(b)** Illustration of ion of light intensity attenuation via the Beer-Lamber law [93].....106

Figure 4.2. Step-by-step discretization of laser beam; **(a).** Defining the circular bound; **(b).** Introducing square of sides equal to diameter of the circular domain, and discretization into smaller square cells (denoted by R); **(c).** Setting up rays' centre positions (xR, yR); **(d).** Elimination of rays formed outside the circular domain.....110

Figure 4.3. Generic representation of discretized energy distribution in the cells.....112

Figure 4.4. Illustration of a particle located within differently sized discretized beam cells. The particle goes potentially undetected in **(a)** and **(b)** due to the coarse discretization of the beam. In case **(c)** the discretization is fine enough to ensure that one or more rays will detect the particle.....113

Figure 4.5. A complete rays' identifier and tracer algorithm with integrated single ray-particle interaction algorithm (**Figure 4.6**); Sub-algorithms: **(a).** Checks ray-particle interception; **(b).** Calculates reflected ray's direction and energy; **(c).** Calculates refracted ray's direction after entering, and then emerging from the particle and energy.....117

Figure 4.6. Extraction from **Figure 4.5** to represent simplified single ray-particle interaction. 118

Figure 4.7. Possible ray-particle or line-sphere interactions; **(a).** No interception occurs; **(b).** Tangential contact at a single point; **(c).** Ray path interferes with the particle, with two possible contact points, where only the entry contact point is considered; The direction of the rays/lines is assumed for illustrative purposes only.....120

Figure 4.8. Splitting of incident ray; **(a).** Surface reflection; **(b).** Proxy refraction, transmitted through the particle; **(c).** Emergent or final refraction.....122

Figure 4.9. Step-by-step calculation of ray reflection. This flow chart represents the expanded/sub-process given in **Figure 4.5-b**. 123

Figure 4.10. Step-by-step process for calculation and determination of refraction or total-internal-reflection and/or total-external-reflection. This flow chart represents the expanded/sub process given in **Figure 4.5-c**. 127

Figure 4.11. Procedure for selection of correct particle intercepted by the ray. The red particles represent potential but false detection, while the green represent the correct detection; **(a).** Ray intersecting multiple number of particles; **(b).** Non-intersected particles eliminated from the count; **(c).** The particle with the shortest ray-to-particle (at intersection point) displacement considered. 130

Figure 4.12. Illustration of reflected ray interacting with particles, with potential pitfall: **(a).** False detection: The same particle (red) as for the incident ray being redetected by the reflected ray; **(b).** The intersection point was evaluated slightly below the surface. 131

Figure 4.13. An extract from the ray-tracing code for laser heating of powders is given in **Appendix D**. 131

Figure 4.14. Illustration of reflected ray interacting with particles, with potential pitfall: **(a).** False detection: The particle (red) in the backward direction to the ray detected; **(b).** Correct detection (green). 132

Figure 4.15. The green particles on the top layer only were subjected to convection and radiative heat losses. 136

Figure 4.16. The green particles are selected and considered to lose heat. They have their centroids located within the limit of their respective radii from the top surface. While the red particles are off limit and are considered below the top surface. 137

Figure 4.17. Plot of heat transfer coefficient against the temperature. 141

Figure 4.18. Biot number of 500 µm particle (HAP and alumina), up to 1400 °C. 144

Figure 4.19. Single ray's interaction with the powder particles: **(a) Without** the scattering effect (due to reflections and/or transmissions), where the particles intercepted by the ray travel path gain energy only; **(b) With** the scattering effect, where the particles other than being intercepted by the ray travel path gained energy as well. 146

Figure 4.20. Illustration of **(a)** powder bed mesh in FEM, **(b)** a close-up of mesh, and **(c)** the cubical mesh element. 147

Figure 4.21. Illustration of laser scan and temperature record region on top surface of the powder bed. 148

Figure 4.22. Illustration of the **(a)** uniform sized (50 µm) powder bed generated in DEM, within an enclosed shell/domain represented by red wireframe, **(b)** volumetric distribution plot and powder bed visualization for non-uniform sized particles with SD 0.1, and **(c)** volumetric distribution plot and powder bed visualization for non-uniform sized particles with SD 0.2. 150

Figure 4.23. Temperature dependent specific heat capacity (constant pressure) of alumina [102]. 151

Figure 4.24. Temperature dependent thermal conductivity of alumina [102]. 152

Figure 4.25. Temperature dependent thermal Conductivity of porous Alumina [102]. 152

Figure 4.26. Graphical plots for average temperature change in laser scan area (illustrated in **Figure 4.21**, **Page 147**), representing the effect of interaction with **CO_2 laser**, for **(a)** 48% packing density, **(b)** 54% packing density, and **(c)** 75% packing density. 155

Figure 4.27. Comparison of surface temperature fields resulting from CO_2 laser scans, in FEM and DEM domains, with **(a)** temperature legend, for **(b)** 48% packing density, **(c)** 54% packing density, and **(d)** 75% packing density. 156

Figure 4.28. Graphical plots for average temperature change in laser scan area (as illustrated in **Figure 4.21**, **Page 147**)**,** representing the effect of interaction with **Nd:YAG laser**, for **(a)** 48% packing density, **(b)** 54% packing density, and **(c)** 75% packing density. 158

Figure 4.29. Comparison of surface temperature fields resulting from Nd:YAG laser scans, in FEM and DEM domains, with separately defined temperature legends, for **(b)** 48% packing density, **(c)** 54% packing density, and **(d)** 75% packing density. 159

Figure 5.1. Illustration of **translational velocity** distribution in the powder bed after they have settled. **(a)** Presence of residual velocities in a number of particles; **(b)** Velocities of all the particles reduced to zero after applying the Stopping Force algorithm. 167

Figure 5.2. Illustration of **angular velocity** distribution in the powder bed after they have settled. **(a)** Presence of residual angular velocities in a number of particles; **(b)** Angular velocities of all the particles reduced to zero after applying the Stopping Force algorithm. 168

Figure 5.3. Particle overlaps: **(a)** Point or no contact between particles, numerically represented as $\delta = 0$ or $\delta < 0$ respectively, **(b)** particles having actual overlap, considered as in compressive stress state, and **(c)** particle are still under compression, i.e., have an existing physical overlap, but their overlap is numerically overridden to 0 and passed to Hertz-Mindilin part of the algorithm while keeping their original positions unaffected. 169

Figure 5.4. Flowchart for determination and calculation of sintering and bonding force between the given two interacting particles. 172

Figure 5.5. The timestep files in the h5 format opened in the HDFView for modification: **(a)** Actual raw inertia and mass in the timestep 4; **(b)** Modified (scaled up by the factor 10^{14}) raw inertia and mass in the timestep 5. 176

Figure 5.6. (a) Temperature legend in Kelvin; **(b)** The simplified setup of 5 particles. The inner particle is initially set at 500 K, while the outer particles at 1000K; **(c)** Final state, where all the particles have reached 900 K, and no further heat exchange exists. 179

Figure 5.7. Graphical plots comparison temperature evolution of centre particle, for different values of RF against the EDEM's standard model (i.e., Chaudhuri et al. [69]). 179

Figure 5.8. Sequence of implementation of three main phases of simulation and associated conditions for application of various forces. 182

Figure 5.9. Phase diagram of the system $CaO–P_2O_5$ (C = CaO, P = P_2O_5) at elevated temperatures. Each line represents a phase boundary. Here C7P5 represents $7CaO·5P_2O_5$; the other abbreviations can be written out in the same manner [105].....188

Figure 5.10. PBSLP process window comparison of the HAP without (graphite) additives. Navarrete-Segado et al. [89].....193

Figure 5.11. Visualization of temperature field on the scan region of pure HAP at 54 W power and 25 mm s^{-1} speed. The minimum temperature on the legend was set at 298 K (25 °C), which was the standard room temperature. In contrast, the maximum temperature was set at 1373 K (1100 °C) to help visually identify the (potential) particles above the sintering temperature. 194

Figure 5.12. Comparison of simulation and experimental indication of sintering response. **(a)** The response in the simulations is indicated by the average temperature recorded in the scan region, **(b)** while visually in the experiment, from **Figure 5.13**. The **blue** cells represent the **negative sintering** response. The **green** and **red** represent the **positive sintering** and **melting/consolidation** response, respectively.....195

Figure 5.13. Processed SEM images of single track experimental results for the given laser powers and scan speeds. The images highlighted in **blue** represent no sintering, the **green**

hightlight indicate sintering took place, while the **red** indicate consolidation (after melting). Images courtesy of Dr Asif ur Rehman (ERMAKSAN, Turkey) and Dr Pedro Jesus Navarrete-Segado (CIRIMAT - INPT, France). 197

Figure 5.14. Temperature distributions after laser scan with 42 W power, at 50 and 75. The legends of figures **(a)** and **(b)** represent the maximum temperature of 1373 K (sintering), while **(c)** and **(d)** represent the maximum 1943 (melting). 199

Figure 5.15. Cropped and magnified views of laser scanned region for **(a)** 75 mm s^{-1} and **(b)** 50 mm s^{-1}. Sintering was triggered at both speeds, but lower speed allowed denser formation. 200

Figure 6.1. Sintering prediction in our simulations, compared to the experiments shown in **Figure 5.13** (**Page 195**). 204

Figure 6.2. Illustration of the particle replacement model (PRM) [115]. 206

Figure 0.1. Simulations show orientation of particles before and after laser scan, and experiments (optical images) demonstrate effect of laser scan on the powder surface, for **different scan speeds** at **30 W power**. 272

Figure 0.2. Simulations show orientation of particles before and after laser scan, and experiments (optical images) demonstrate effect of laser scan on the powder surface, for **different scan speeds** at **36 W power**. 273

Figure 0.3. Simulations show orientation of particles before and after laser scan, and experiments (optical images) demonstrate effect of laser scan on the powder surface, for **different scan speeds** at **42 W power**. 274

Figure 0.4. Simulations show orientation of particles before and after laser scan, and experiments (optical images) demonstrate effect of laser scan on the powder surface, for **different scan speeds** at **48 W power**. 275

Figure 0.5. Simulations show orientation of particles before and after laser scan, and experiments (optical images) demonstrate effect of laser scan on the powder surface, for **different scan speeds** at **54 W power**. 276

LIST OF TABLES

Table 2.1. Examples of DEM solvers. 12

Table 2.2. The relationship between the angle of repose and flowability (Riley et al. 1978) [45], [46] 24

Table 2.3. Mechanisms of mass transportation during sintering [48]. 27

Table 2.4. Summary of radiant exchange factor correlations for the radiative conductivity [65]. Reuse license: 5516180873657 (Elsevier - International Journal of Heat and Mass Transfer). 35

Table 2.5. The value of roller-spreading parameters [18]. Reuse license: Open Access / Creative Commons. 54

Table 2.6. Summary of literature with its methodologies and applications. 58

Table 3.1. Composition of different powders used in the powder delivery simulations and experiments. 62

Table 3.2. Parameters used in EDEM to generate the respective size distributions for the powders, for alumina powders. 63

Table 3.3. Summary of the particle sizes for the alumina powders used in this study, with respect to their cumulative percentages of 10%, 50% and 90%. 66

Table 3.4. Material properties and interaction coefficients for alumina powders [18], [90]. 72

Table 3.5. Simulation parameters for spreading of alumina powders. This data was provided by BCRC as adapted for their experiments. 79

Table 3.6. Simulation parameters for spreading of alumina powders. This data was provided by BCRC as adapted for their experiments. 81

Table 3.7. Parameters used in EDEM to generate the respective size distributions for the powders, for HAP powders. 100

Table 3.8. Summary of the particle sizes for the HAP (Urodelia® 300-08-2) powders used in this study, with respect to their cumulative percentages of 10%, 50% and 90%. 102

Table 3.9. Material properties and interaction coefficients for hydroxyapatite powders. .. 103

Table 4.1. List of devised pseudo algorithms used in developing the ray tracing code for laser heating of powder bed. 116

Table 4.2. Minimum particle size to be considered against the given laser wavelength, calculated from equation (4.30). 133

Table 4.3. Laser absorption of alumina and hydroxyapatite powders [6], [89]. 134

Table 4.4. The refractive indices for α-alumina and hydroxyapaptite [95]. 135

Table 4.5. An excerpt from **Appx-A Table 1**, for empirical correlations for the average Nusselt number during natural convection over horizontal surfaces [100]. 139

Table 4.6. Calculation of heat transfer coefficient over the range of temperatures. 140

Table 4.7. Laser parameters for simulations (both FEM and DEM). 151

Table 5.1. Modification (scaling up) of actual values in the simulations. 175

Table 5.2. Summary of conditions to calculate the relevant contact force. 177

Table 5.3. Summary of conditions to select the thermal contact model. 183

Table 5.4. Laser processing parameters, adapted from Navarrete-Segado et al. [89]. 189

Table 5.5. Summary of material, optical and sintering parameters for SLS of hydroxyapatite. 190

Table 5.6. Constant values of simulation optimization factor (SOF) for different modes of heat transfer: Input via laser ray tracing (RT), heat losses (HL), and heat conduction (HC)..... 192

1. INTRODUCTION

1.1. BACKGROUND OF THE PROJECT

Technical or high-value ceramic materials have a wide range of applications in the development of aerospace and medical devices due to their outstanding properties: Lower density, high mechanical strength as well as excellent wear, corrosion, and thermal resistance. The ceramic parts developed from conventional manufacturing techniques are time-consuming and induce shrinkage in the final geometry during the firing process. They also incur high costs due to increased wear in machining tools and manual labour. As a result, their industrial utilization and growth often remain restricted. To tackle the conventional limitations, the alternative techniques for 3D printing of ceramics were studied under the consortium of EU's Innovative Training Network: **DOC-3D-Printing (Development of Ceramics 3D-Printing)**. 6 academic, 7 industrial, and 1 non-profit association partners are participating. A total of 14 Ph.D. projects were dedicated to it, undertaken by the respective ESRs (Early Stage Researchers) based across the project partners.

1.2. AIMS AND OBJECTIVES

This research project aimed to develop novel methodologies to simulate the powder bed selective laser processing (PBSLP) of ceramic materials. The study was intended for two phases mainly, i.e., preprocessing and processing of powder bed, and strived to achieve the following objectives:

1.2.1 PREPROCESSING PHASE: POWDER DELIVERY

- To simulate the powder deposition process of alumina powders using a blade coater and evaluate the resulting packing densities.
- To analyse the effect of layer thickness on the packing densities of alumina powders.
- To develop a methodology to assess the surface quality of the simulated powder bed.
- To investigate the effect of the particle size distribution both on packing density and surface quality.

1.2.2 PROCESSING PHASE: SELECTIVE LASER PROCESSING

- To assess and choose an appropriate numerical methodology to simulate the PBSLP process of HAP.
- To investigate the effect of laser parameters (power and scan speed) on sintering process of pure HAP powders and with addition of graphite additives.
- To optimise and validate the model against the experiments of similar parameters.
- To develop algorithms and codes for laser-powder thermal interaction, particle-particle sintering, particle-particle heat conduction, and particle heat losses, which would enable the aforementioned laser parametric investigation.

1.3. INTRODUCTION TO THE THESIS

This project was part of Work Package 5 for DOC-3D-Printing, devoted to the overall process modelling of selective laser sintering, including powder delivery, laser irradiation, sintering, and developing relevant algorithms and codes. Bioceramic materials such as alumina (Al_2O_3) and hydroxyapatite (HAP) were of particular interest in this case for their applications in orthopaedic extremities. The overview of tasks addressed in the individual chapters has been discussed as follows:

Chapter 1 (current) introduces the background and structure of the thesis.

Chapter 2 offers a literature survey covering the main topics for this research work, which includes background on general laser processing of powder beds, the significance of powder delivery for powder bed selective laser processing (PBSLP) methods, and most importantly, the state-of-the-art in numerical modelling and simulations of selective laser sintering/melting (SLS/M) processes. The key findings have been presented in a tabulated summary, which helps identify the current gaps and those addressed in this thesis.

Chapter 3 presents the DEM-based modelling and simulation work on the delivery and layer formation of alumina (Al_2O_3) and hydroxyapatite (HAP) ceramic powders. For alumina powders, different size distributions of particles were considered to observe the impact on the final packing density of the powder bed. A detailed approach to generating the precise particle size distributions and flowability calibration in DEM is also given, that well aligns with the experimental powders. Furthermore, a code for ray-tracing algorithm was developed to evaluate the surface quality, which has also been discussed here.

Chapter 4 discusses the modelling of laser heating of powder beds, initially developed using FEM solver Abaqus CAE and Fortran subroutines. Furthermore, it proposes another methodology, along with FEM, to simulate the laser heating of ceramic powder beds. It is based on a DEM-coupled ray tracing approach. A complex code for the ray tracing method was developed in C++ and coupled with the commercial DEM solver Altair EDEM. It accounted for the interaction of a single (laser) ray with a single powder particle (in DEM) and combined the accumulated effect of all the numerically discretized rays within the beam with all particles to map out a heat distribution field during laser scan. The code could identify partial and total external reflections, refractions (transmissions), on and through each particle, as well as energy absorption of individual particles.

Chapter 5 proposes a methodology to simulate the sintering behaviour of powder particles This model was then combined with the ray tracing model (from **Chapter 4**) using the processed HAP powders (from **Chapter 3**) to simulate the overall selective laser sintering process.

Chapter 6 provides the conclusion and critical analysis of the research work carried out in this thesis, with insight into potential improvements and future work.

THIS PAGE WAS INTENTIONALLY LEFT BLANK

2. LITERATURE REVIEW

Highlights:

1. Literature survey on the background of PBSLP

2. Background of finite and discrete element method (FEM/DEM)

3. Modelling of sintering and heat conduction within the powder beds

4. State-of-the-art of PBSLP modelling techniques using FEM and DEM

5. Summary of gaps being addressed in this thesis

2.1. LASER PROCESSING OF POWDERS USING PBSLP

In recent decades, 3D printing, or additive manufacturing (AM) processes have caught special attention in industries and academia. This is due to their ability to produce complex shapes, which would otherwise be difficult with conventional methods. One of the categories of AM is powder bed selective laser processing (PBSLP), commonly known as selective laser sintering (SLS) or melting (SLM) [1]. Sintering is a manufacturing process that fuses a mixture of powdered metal or ceramic without melting it. The weakly bonded powders are brought close to 50 – 90% of the melting point and consolidated, sometimes with additional pressure [2], [3]. SLS is a powder bed fusion AM technology that uses thermal energy from lasers to fuse the powder particles within the selected region without completely melting them or even under the melting point.

This process has the potential to produce complex parts with intricate internal and external morphology [4]. PBSLP has the main advantage of the powder bed acting as an integrated support structure. Moreover, parts of virtually any material type can be produced if they thermally respond to the laser [5], [6]. It allows the parts to be developed in one go, eliminating the use of moulds or dies entirely. So, additional machining steps are often not required in its operation [7]. This method was invented and patented by Dr Carl Deckard and his academic advisor Dr Joe Beaman at the University of Texas at Austin in 1989 [8].

The standard setup for the SLS based 3D-printer contains a supply container, a build chamber, and a collection chamber. The platform or base of each container is controlled by pistons, which enables it to be raised or lowered. A laser is installed on top that emits the beam horizontally. This beam is guided to the build chamber using the mirror deflection system that

utilizes a Galvo Motor System or XY-Tables [9]. **Figure 2.1** below illustrates a typical SLS machine [10]:

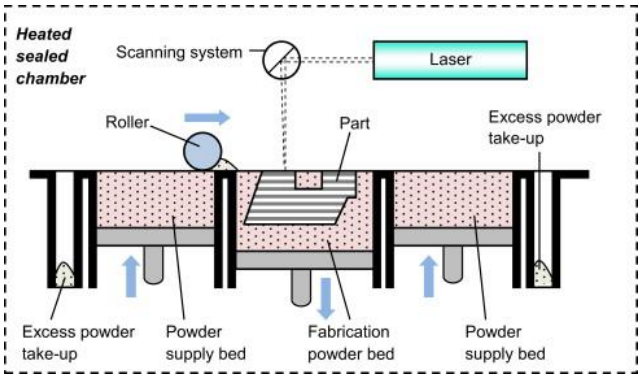


Figure 2.1. The schematic of the selective laser sintering (SLS) process, illustrating the overall 3D printing process. The roller or blade delivers the powder to the fabrication powder bed from the powder supply container/bed after each layer has been laser processed. The excess powder is accumulated in the separate containers to avoid material wastage [10]. Reuse license: 5517331371643 (Elsevier – Comprehensive Materials Processing).

The powder is loaded in the feed container initially. A roller or a recoating blade guides the first layer into the 'Build-Chamber' on the substrate. The layer's cross-section is defined from the geometry of the CAD model, which is then scanned by the laser beam. The particles in the volume exposed to the laser are sintered together due to heat. The build chamber is then lowered to accommodate the subsequent layer. This process is repeated layer-by-layer until the solid part is completely developed [11].

2.2. FINITE ELEMENT METHOD (FEM)

2.2.1 BACKGROUND

FEM is a numerical method that uses domain discretization, i.e., splitting the domain into a finite number of divisions or elements, also called ‘meshing’. This method addresses complex problems, specifically those defined by partial differential equations. Each element shares a geometric boundary with neighbouring elements and is connected by nodes. The problem is always solved at the nodes of the elements. It means exact information is available only at the nodes, and the rest is interpolated. When the number of elements or sub-divisions in a domain is increased, the mesh becomes finer and leads to a solution with better accuracy. However, it comes at the cost of higher computational time and power [12].

2.3. DISCRETE ELEMENT METHOD (DEM)

2.3.1 BACKGROUND

The general dynamic interaction of discrete particles has been discussed in this section. DEM is based on molecular dynamic methods, representing the interaction between the two types of particles: Hard-sphere (rigid) and soft-sphere (deformable) particles. In the former method, the collisions are instantaneous, while in the case of a soft-sphere, there is a finite duration contact due to deformability [13]. Lundig states the hard-sphere method as 'event-driven' and considers that the DEM be distinguished from it, as it deals with soft-sphere methods in the mainstream [14].

The particles in the granular media deform under stress. The modelling of actual deformation can be complex. Therefore, even in the case of soft spheres, the particles are treated as rigid bodies. However, their geometries are allowed to overlap. These overlaps represent deformation and are used to calculate the interacting forces between them [14]. Generally, less than 5% of the particle radius overlaps are allowed [15]. Each particle carries specific properties such as shape, size, material properties, interaction coefficients and velocity. Initially, the particles are subject to motion based on Newton's laws, and they come into contact with each other's grain boundaries. The contact models result in reaction forces in addition to external factors (e.g., fluid drag, buoyancy, acceleration etc.) [16]. The soft sphere (i.e., deformable particle) DEM was developed by Cundal and Strack, published in 1980 [17]. The motion of particles in DEM models, both translational and rotary, are governed by Newton's second law, as follows [18]:

$$m_i \frac{dv_i}{dt} = \sum_j \left(F_{ij}^n + F_{ij}^t\right) + F_i^g \tag{2.1}$$

$$I_i \frac{d\omega_i}{dt} = \sum_j \left(T_{ij}^s + T_{ij}^r\right) \tag{2.2}$$

where m_i and I_i are the mass and moment of inertia of the particle i, respectively; v_i and ω_i are the translational and rotational velocities, respectively, F_{ij}^n and F_{ij}^t are the normal and tangential contact forces on particle i by particle j or wall; F_i^g is the gravitational force acting on particle i; T_{ij}^s and T_{ij}^r are the torques on particles caused by tangential force and rolling friction, and dt is the time step. They use explicit timestep method (discussed later under **Section 2.3.7**).

There are numerous closed (commercial) and open-source (public access) DEM solvers, including the following for example, but not limited to:

Table 2.1. Examples of DEM solvers.

Closed-source (commercial) solvers	**Open-source solvers**
1. EDEM (Altair Engineering Inc.) [19]	1. LAMMPS [20]
2. Aspherix (DCS Computing GmbH) [21]	2. LIGGGHTS (DCS Computing GmbH) [22]
3. ThreeParticle CAE (Becker 3D GmbH) [23]	3. MercuryDPM [24]
4. Rocky DEM [25]	4. YADE [26]
5. PFC (Particle Flow Code) [27]	5. Blaze-DEMGPU [28]
	6. MUSEN [29]

2.3.2 CONTACT THEORY MODELS

During the layering and spreading process, the powder particles come into contact with each other. The contact theories mathematically define their interaction. They play an essential role in the modelling of particle behaviour using DEM. The particle interactions can generally be categorized in **Figure 2.2** [30], with some commonly used contact models discussed in the following (**Sections 2.3.3 – 2.3.6**).

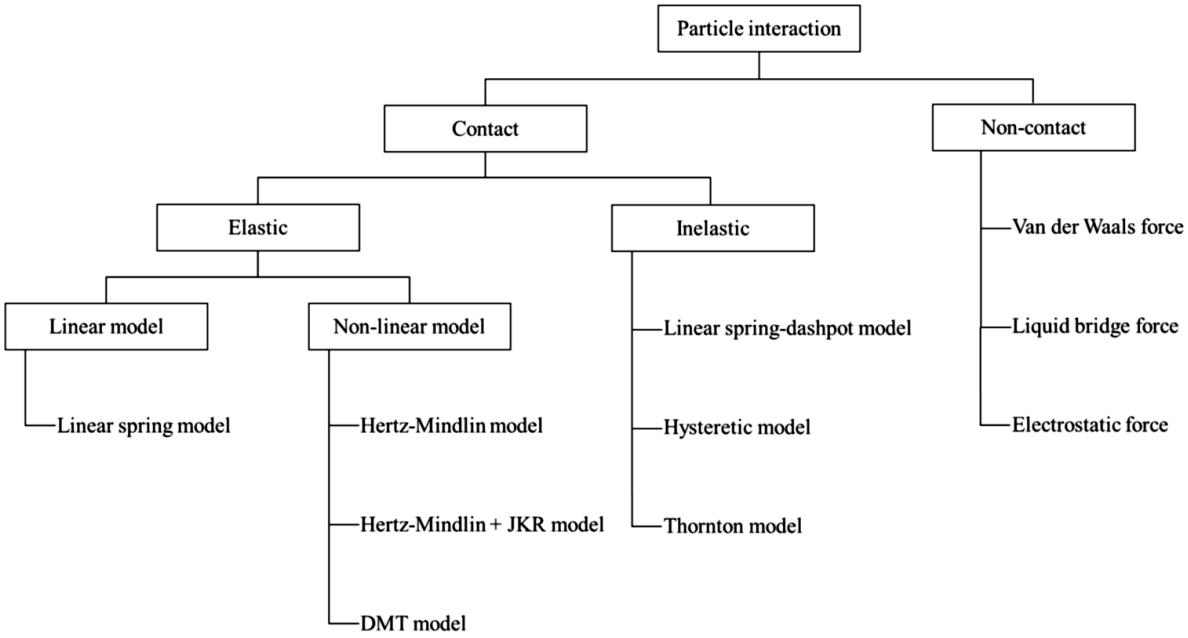


Figure 2.2. Classification of particle interaction force models by contact force and non-contact force [30]. Reuse license: Open Access / Creative Commons.

2.3.3 LINEAR SPRING-DASHPOT MODEL

This model is based on Hooke's law of first-order approximation to a mass-spring system. It is generally accurate for minor deformations and presents a smaller computational overhead due to its simplicity. For a simple linear spring model, only a completely elastic contact is possible [30]. Therefore, the dashpot components, parallel to the spring, represent kinetic energy dissipation. Both normal and tangential (friction) components of forces are used, as follows [31]:

$$F = \left(K_n \delta_{n_{ij}} - \gamma_n V_{n_{ij}}\right) + \left(K_t \delta_{t_{ij}} - \gamma_t V_{t_{ij}}\right) \tag{2.3}$$

where K_n and K_t are normal and tangential stiffness, $\delta_{n_{ij}}$ and $\delta_{t_{ij}}$ are normal and tangential overlaps, $V_{n_{ij}}$ and $V_{t_{ij}}$ are normal and tangential velocities, γ_n and γ_t are viscoelastic damping constants for normal and tangential contacts, respectively [31].

2.3.4 HERTZ MODEL

In 1896, Hertz examined the contact between two smooth elastic bodies with the following main conditions [32]:

1. assumption of negligible strain and contact area;

2. continuous and frictionless surfaces;

3. the contact area is significantly smaller than the radius of the body.

Apart from the limiting conditions, this model did not address particle adhesion. The contact radius a_0 was derived to be:

$$a_0^3 = \frac{3}{4}\pi(k_1 + k_2)\frac{R_1 R_2}{R_1 + R_2}F_0 \tag{2.4}$$

where R_1 and R_2 are the radii of each sphere, F_0 is the load pressing the spheres together, and k is the elastic constant, defined by:

$$k_n = \frac{1 - \nu_n^2}{\pi E_n} \tag{2.5}$$

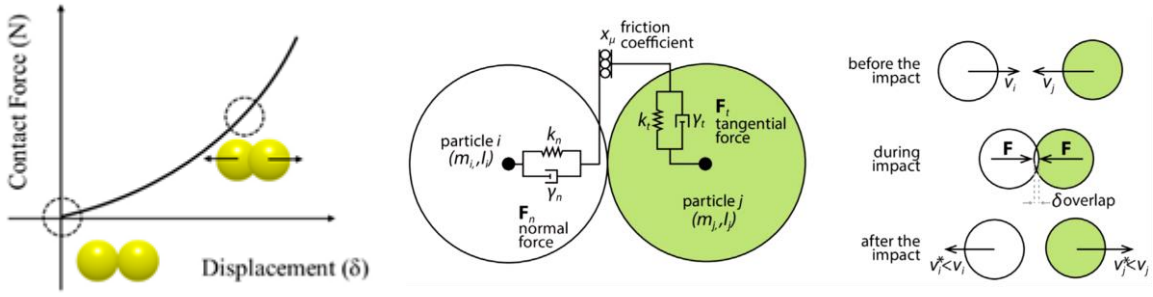
where ν is the Poisson's ratio, E_n is Young's modulus. The resulting displacement (δ) due to compression is given by:

$$\delta^3 = \frac{9}{16}\pi^2(k_1 + k_2)^2\frac{R_1 + R_2}{R_1 R_2}P_0^2 \tag{2.6}$$

where P_0 is the normal load. Hertz verified his theory using an optical microscope to measure the contact between the glass spheres.

2.3.5 HERTZ-MINDLIN MODEL

This model is based on Hertz' (1949) contact theory [32] representing the normal force, and Mindlin-Dereseiwicz' (1953) theory for elastic frictional contact, representing the tangential component [32], [33]. Both components hold damping coefficients, and the tangential force is limited by the Coulomb (static) friction ($\mu_s F_n$). The complete form of this model is computationally expensive [34]. Therefore, it was simplified to 'no slip'. It has been used in numerous process simulations due to its accuracy [30]. **Figure 2.3** represents its force-displacement graphical relationship and schematic:



(a) **(b)**

Figure 2.3. (a) General relationship between force and displacement in Hertz-Mindlin model: No contact force exists when displacement or overlap is zero. The overlap induces this HM force [30]. **(b)** Schematic of Hertz-Mindlin contact model between two particles consisting of parallel configuration of spring and damping components [35]. Reuse license: Open Access / Creative Commons.

The normal force (F_n) as a function of normal overlap or displacement (δ_n), is given by [19], [36]:

$$F_n = -\frac{4}{3} E^* \sqrt{R^*} \delta_n^{\frac{3}{2}} \tag{2.7}$$

where $E^* = \frac{E_i E_j}{(1-\nu_i^2)E_j + (1-\nu_j^2)E_i}$ and $R^* = \frac{R_i R_j}{R_i + R_j}$ are equivalent Young's Modulus and equivalent radius, respectively, and ν is the Poisson's ratio. The normal damping force is given by [19]:

$$F_n^d = -2\sqrt{\frac{5}{6}}\,\beta\,\sqrt{S_n m^*}\,v_n^{\overrightarrow{rel}} \tag{2.8}$$

where $m^* = \frac{m_i m_j}{m_i + m_j}$ and $v_n^{\overrightarrow{rel}}$, are the equivalent mass and normal component of the relative velocity, respectively, with $\beta = \frac{\ln e}{\sqrt{\ln^2 e + \pi^2}}$, and normal stiffness $S_n = 2E^*\sqrt{R^*\delta_n}$, with e being the coefficient of restitution. While the tangential force (F_t) is given by [36]:

$$F_t = 8G^*\sqrt{R^*\delta_n}\,\delta_t \tag{2.9}$$

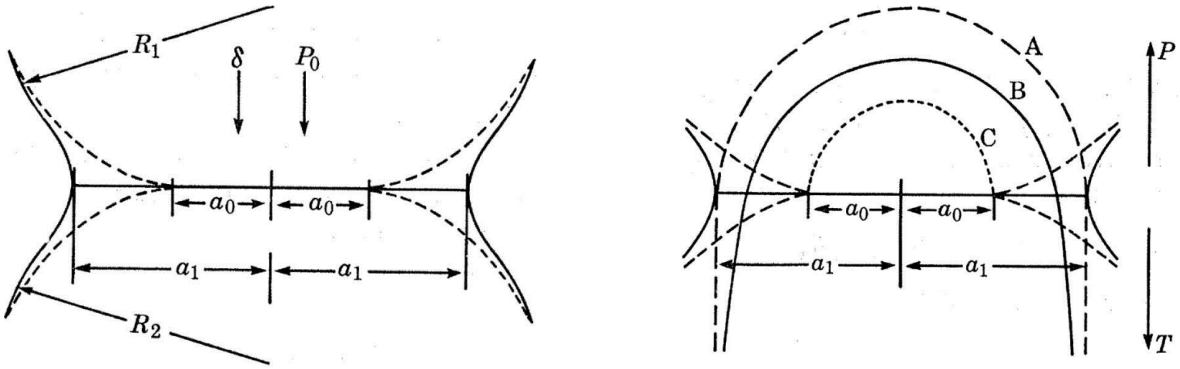
where G^* is the equivalent shear modulus and δ_t is the tangential displacement. The tangential damping force is given by [19]:

$$F_t^d = -2\sqrt{\frac{5}{6}}\,\beta\,\sqrt{S_t m^*}\,v_t^{\overrightarrow{rel}} \tag{2.10}$$

where S_t $(= 8G^*\sqrt{R^*\delta_n})$ is the tangential stiffness, and $v_t^{\overrightarrow{rel}}$ is the relative tangential velocity.

2.3.6 HERTZ-MINDLIN MODEL WITH JKR COHESION

In 1968, Roberts experimented and observed that under smaller loads, the contact area between the spheres was significantly greater than predicted by Hertz. Kendall also had similar observation in 1969. It resulted from strong adhesive forces between the contact surfaces [32]. Johnson further observed the tensile stresses at the edge of contact and compressive stresses at the centre. In his approximate theory, a_1 is calculated using compression only, **Figure 2.4 (a)**. The full JKR theory considers both tension and compression for a_1, **Figure 2.4 (b)**. While in contrast, Hertz theory calculated a_0 only, since it does not consider adhesion, and uses compression only.



(a) **(b)**

Figure 2.4. (a) Contact between two spherical bodies as defined by the JKR **(b)** Stress distribution curves for the JKR [32]. Reuse license: 1338159-1 (Copyright Clearance Centre, Inc. on behalf of HARRISON AND SON)

In the above **Figure 2.4**, R_1 and R_2 are the radii of the interacting particles, P_0 the normal load, δ the elastic displacement, a_0 and a_1 are contact radii without and with adhesion, respectively. The nonlinear Hertz-Mindlin JKR model accounts for JKR's adhesion force, along with an additional normal cohesive force that depends on overlap (δ) and surface energy (γ), as following [37]:

$$F_{JKR} = \frac{4E^*}{3R^*}a^3 - \sqrt{8\pi a^3 \delta \gamma E^*} \tag{2.11}$$

where E^* is equivalent Young's modulus, R^* is the equivalent radius, a is the contact radius, and γ is the surface energy. The corresponding deformation (δ) is given as [37]:

$$\delta_{JKR} = \frac{a^2}{R^*} - \frac{2}{3}\sqrt{\frac{\pi a \Delta\gamma}{E^*}} \tag{2.12}$$

This model can also evaluate the attractive cohesion forces, even if the particles are not in physical contact. The critical gap between the particles, so that there is non-zero force, is given by [37]:

$$\delta_c = \frac{1}{3R}\left(\frac{27}{8}\frac{\pi R^{*2}\Delta\gamma}{E^*}\right)^{\frac{2}{3}} \tag{2.13}$$

The maximum value of the cohesion force, also called the pull-out force, occurs when the particles do not have physical contact, but the separation is less than δ_c, given by [19]:

$$F_{pullout} = \sqrt{\frac{4\pi\gamma a_c}{E^*}} + \frac{9\pi\gamma R^{*2}}{R^*} \tag{2.14}$$

The following **Figure 2.5** represents the graphical relationship between the contact force and displacement for Hertz-Mindlin with the JKR contact model [30]:

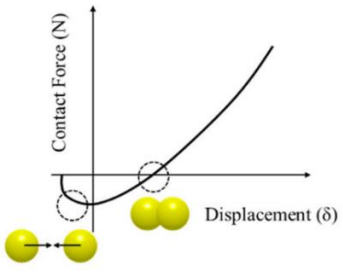


Figure 2.5. General relationship between force and displacement in Hertz-Mindlin (HM) + JKR model. In contrast to simple HM, HM + JKR force has attraction component that induces force even when the particles are not in contact. [30]. Reuse license: Open Access / Creative Commons.

2.3.7 COMPUTATIONAL TIME-STEP FOR DEM SIMULATIONS

An explicit numerical integration scheme is generally used to solve equations of motion in DEM. This is where information for the current timestep is taken from the previous. The timestep for the simulations must be lower than the critical timestep – a fraction of the natural frequency of an equivalent mass-spring system. This depends on several factors such as material properties (specifically Young's or Shear-Modulus) and particle size - the smaller the particle, the smaller the time-step. In addition, the overall computational time is further dependent on particle shape, total simulation time, and the total number of particles in the system – which means a larger number of particles require more computations [38]. Moreover, small time increments are necessary for DEM simulations for several reasons, including avoiding the numerical instability in explicit solutions and capturing the times of contact formation. Additionally, in quasi-static simulations, the strain rate must also be low to prevent the development of dynamic effects (such as wave propagation through the assembly) and preserve equilibrium conditions [39].

Typically, two different methodologies are used to evaluate the maximum or critical time step for stable DEM simulations. They depend on the contact model being employed. For the first case where models such as spring-dashpot are used, stability is the function of mass and stiffness. The maximum time step (Δt_c) calculated as a function of mass and stiffness, which is proportional to $\sqrt{\frac{m}{k}}$, and given by [15]:

$$\Delta t_c = \frac{2}{\omega_{max}} \tag{2.15}$$

where $\omega_{max} = \sqrt{\lambda_{max}}$, and λ_{max} is the maximum eigenvalue of the matrix $M^{-1}K$. Here M is the mass matrix, and K is the stiffness matrix. In the second case, for models such as Hertz-Mindlin, the stability is evaluated from the Rayleigh time step method. It is also one of the most commonly employed methods. Here the critical time step is calculated from the Rayleigh wave velocity as follows [15]:

$$\Delta t_c = \frac{\pi r}{\beta}\sqrt{\frac{\rho}{G}}, \tag{2.16}$$

where r is the particle radius, ρ is the density, G is the shear modulus, and $\beta = 0.8766 + 0.163\nu$, where ν is the Poisson's ratio. Solving the DEM model can be time-consuming due to the small timesteps required. The computational time can be reduced by various strategies, such as scaling up the particles sizes to reduce the total number of particles in the system [40]. Another common approach is to reduce the shear or Young's modulus to an acceptable degree such that the overall dynamic of the particles remains unaffected [41]. In some cases, mass scaling of particles is also used (discussed under **Section 2.4.3**).

2.3.8 CALIBRATION OF POWDERS IN DEM

The flowing powder particles in DEM must be correctly characterized so that their dynamic behaviour is as close to the actual powders (in experiments) as possible for precise and accurate simulation results. Some of the common methods used in DEM calibration discussed are the static angle of repose (AOR), dynamic angle of repose, and unconfined yield strength (UYS).

In the static AOR method, the particles are generated inside a cylinder or hopper with an open base through which the particles can flow (**Figure 2.6-a**). The size distribution of particles can be as desired, such as uniform, normal, random, or user-defined. After the powder has settled (**Figure 2.6-b**), the cylinder is lifted to allow the particles to flow out to form a cone-shaped heap. The angle formed by its slope, with respect to the base, is called the angle of repose (**Figure 2.6-c**) [42]. One of the alternatives is the draw-down test (**Figure 2.6-d**), which contains two volume segments or compartments. In the upper compartment, the particles are generated, which connects with the lower compartment using a rectangular opening. The powder flows out through this opening (**Figure 2.6-e**) and drops down to the base of the lower compartment in a similar heap formation (**Figure 2.6-f**) [43].

In contrast to static AOR, the dynamic AOR method considers a transparent rotating drum partially filled with powder. The drum is rotated several times, which leads to the heap formation of powders. The angle formed by the heap against the base is known as dynamic AOR or avalanche angle [44]. A similar approach is adopted in the DEM simulations, where particles are generated in the circular geometry, which is then rotated for heap formation (**Figure 2.7**).

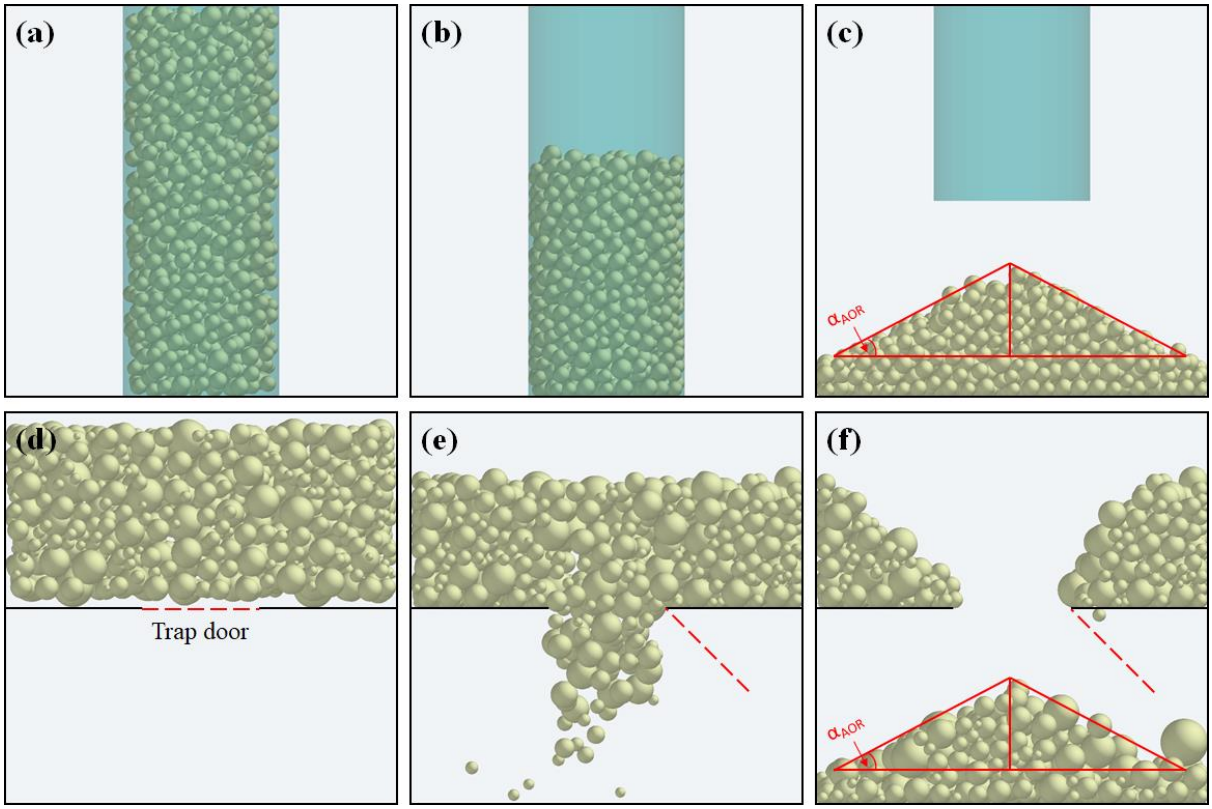


Figure 2.6. Cylinder method: **(a)** Particles generation within the cylindrical volume; **(b)** Settlement of particles; **(c)** Heap formation. Boxed compartments method: **(d)** Particles generation in the upper compartment, with a trap door, to prevent the flow of particles initially; **(e)** Once the particles are settled, the trap door is opened, and they are allowed to flow through the rectangular opening; **(f)** Heap formation in the lower compartment.

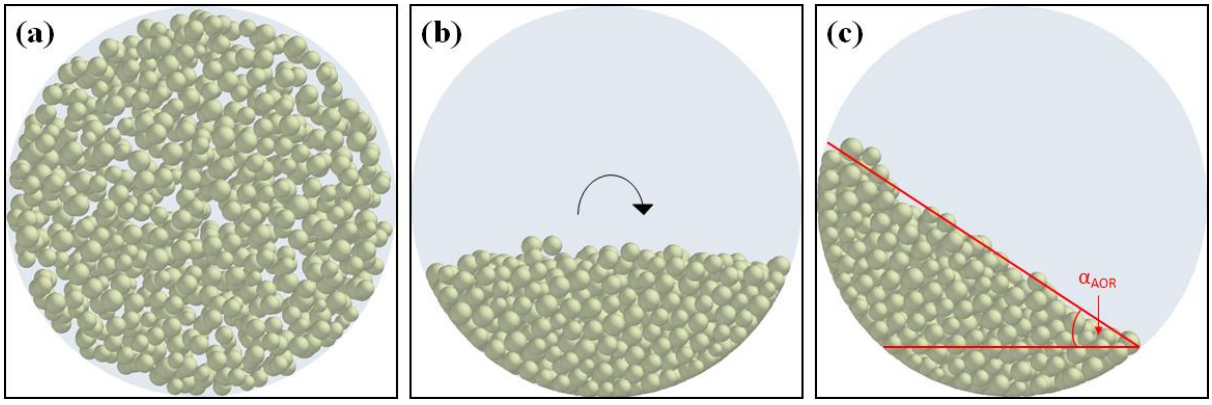


Figure 2.7. Measurement of dynamic angle of repose (or avalanche angle) in DEM.

Riley et al. graded the flowability of powders in their work, as following [45], [46]:

Table 2.2. The relationship between the angle of repose and flowability (Riley et al. 1978) [45], [46].

Flow property	Angle of repose (θ)
Excelent	25-30
Good	31-35
Fair-aid not needed	36-40
Passable-may hang up	41-45
Poor-must agitate, vibrate	46-55
Very poor	56-65
Extremely poor	>66

The UYS is the maximum amount of compressive force or stress needed to cause buckling deformation or breakage to the material when it is not confined or supported by an external structure such as a container. When cohesive powders need to be simulated in DEM, they are calibrated with respect to their UYS. Lupo et al. suggested two methodologies. In the first method, uniaxial testing is used, in which the powder is generated within the cylindrical mould, compressed using a piston, and then the mould is removed. For sufficient compressive force, the powder gets agglomerated and keeps its cylindrical shape. Further force is then applied until the cylindrical agglomerate is broken. The resulting ratio between this force and the agglomerate cross-section calculates UYS. However, the unavoidable wall shear stresses reduce the effect of compression near the bottom of the sample [42].

The alternative method indirectly measures the UYS from shear testing experiments. It is based upon the assumption that the Mohr circle obtained for the major principal stress during shear testing consolidation coincides with the consolidation stress in the uniaxial test. The yield condition for the uniaxial tester is represented by the point passing through the strain-stress plane origin and tangent to the static yield locus that is derived from shear testing at the same consolidation. UYS is calculated as a function of major principal stress at

consolidation. Due to the complexity of this method, the DEM replica is intricate to implement. Therefore, a simplified uniaxial test is performed with unknown DEM parameters, such as particle-to-particle static and rolling frictions. These parameters are calibrated by using comparing the resulting compressive force in DEM against the one obtained from shear testing experiments. In this way, the effect of wall friction is eliminated [42].

2.4. MODELLING OF SOLID-STATE SINTERING MECHANISM

Considering a two-sphere model, initially, the interface is formed, and neck growth occurs (**Figure 2.8-a**), between the interacting particles, without any grain growth. Shi (1999), in his study on solid-state sintering of ceramics, discussed the interaction between surface tension and interfacial tension, where he states surface tension as the driving force for densification and interface as resistance [47]:

$$\Delta G = \Delta G_s + \Delta G_i = \gamma_s dA_s + \gamma_i dA_i \tag{2.17}$$

where ΔG, ΔG_s and ΔG_i are changes in total free energy, surface energy, and interface energy, respectively. γ_s, A_s, γ_i, and A_i are surface tension, surface area, interfacial tension, and interfacial area, respectively.

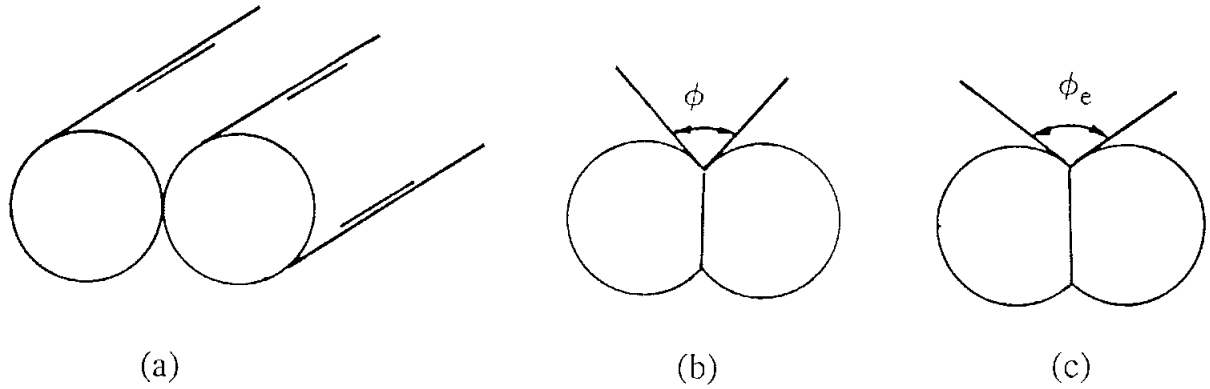


Figure 2.8. Schematics of the formation of the dihedral angle between two identical touching cylinders, **(a)** particles in touch, **(b)** neck formation, 0 <Ø<Øe. Reuse license: 5516180873657 (Springer Nature – Journal of Materials Science).

Furthermore, the chemical potential (or the driving force) for neck formation and mass transfer through the interface is considered. Based on the Gibbs-Kelvin relation, the neck formation driving force ($\Delta\mu_n$) is given by [47]:

$$\Delta\mu_n = \gamma_s \Omega \left(\frac{1}{\rho_n} - \frac{1}{r}\right) \tag{2.18}$$

where Ω is the atomic volume, ρ_n is the neck radius, and r is the particle radius. Additionally, the chemical potential for mass transfer within the particles is [47]:

$$\Delta\mu_c = 2\gamma_s \Omega \left(\frac{1}{r_1} - \frac{1}{r_2}\right) \tag{2.19}$$

where r_1 and r_2 are the radii of the interacting particles. The neck continues to grow until an equilibrium is reached, defined by a so-called dihedral contact angle ($\emptyset_e$), where $\emptyset$ is a contact angle between the two adjacent surfaces (**Figure 2.8-b,c**) [47]:

$$\cos\frac{\emptyset_e}{2} = \frac{\gamma_i}{2\gamma_s} \tag{2.20}$$

Next, grain growth occurs, and continuous channels of pores along the edges start to form as well. The mass transportation/transfer occurs through the neck of the particles (**Figure 2.9**) by one or more of the following mechanisms [48]:

Table 2.3. Mechanisms of mass transportation during sintering [48].

#	Transport path	Source of matter	Sink of matter
1	Surface diffusion	Surface	Neck
2	Lattice diffusion	Surface	Neck
3	Vapour diffusion	Surface	Neck
4	Boundary diffusion	Grain boundary	Neck
5	Lattice diffusion	Grain boundary	Neck
6	Lattice diffusion	Dislocations	Neck

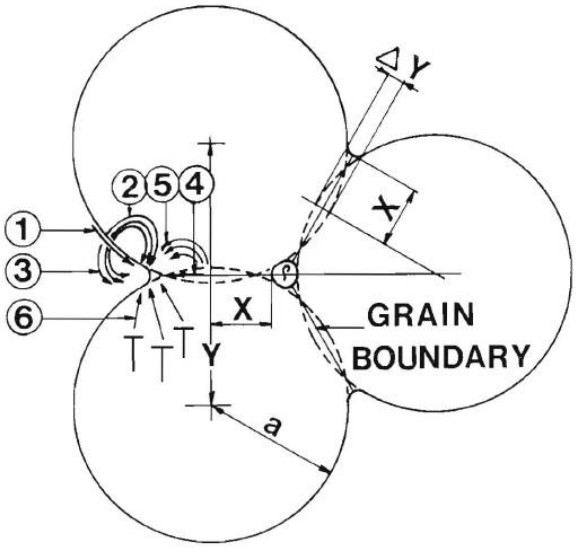


Figure 2.9. Three-sphere model showing the six transport paths from **Table 2.3**. The paths 1,2, and 3 represent surface, lattice, and vapour diffusion of matter, respectively, with matter source as surface. While the path 4 represent boundary diffusion with grain boundary as matter source, while the paths 5, and 6 represent lattice diffusion, with grain boundary and dislocation as matter source respectively. In all case, the matter sinks through the neck of the interacting particles, but only path 4 to 6 lead to densification [48].

2.4.1 FOR FINITE ELEMENT METHOD: CONTINUUM VISCOELASTIC MODEL

Nosewicz et al. (2017) developed a generalized viscoelastic model of sintering that took GB diffusion and thermal expansion into account. Based on it, Wawrzyk et al. (2018) modelled the sintering process in the micro and macroscopic domains. Their rheological representation of sintering was as follows [49]:

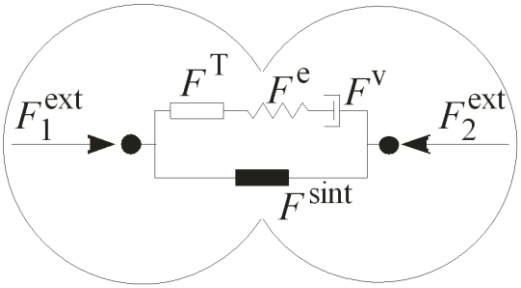


Figure 2.10. Rheological scheme of thermo-viscoelastic model, consisting of sintering force in parallel to tangential and viscous force components, with compressive forces exerted at both ends [49]. Reuse license: 5516190219018 (Elsevier – Computational Material Science).

The interaction (forces) between the sintered particles, based on mass transport and stresses at the grain boundary, was given as [49]:

$$F_n = \frac{\pi a^4}{8D_{eff}} v_{rn} + \pi\gamma_s \left[4r\left(1 - \cos\frac{\emptyset_e}{2}\right) + a\sin\frac{\emptyset_e}{2}\right] \tag{2.21}$$

where F_n is the normal force, v_{rn} is the normal relative velocity, r is the particle radius, a is the interparticle boundary radius, and D_{eff} represents the effective grain boundary diffusion is given by [49]:

$$D_{eff} = \frac{D_{GB}\delta\Omega}{k_B T} \tag{2.22}$$

and the model considered the thermal expansion as [49]:

$$r = r_0[1 + \alpha(T - T_0)] \tag{2.23}$$

where r_0 is the radius at the initial reference temperature T_0. Their macroscopic model considered combined strain rates of thermal ($\dot{\varepsilon}^{th}$), elastic ($\dot{\varepsilon}^{e}$) and viscosity ($\dot{\varepsilon}^{visc}$) [49]:

$$\dot{\sigma} = \mathbf{D}\left(\dot{\varepsilon} - \dot{\varepsilon}^{th} - \dot{\varepsilon}^{visc}\right) + \dot{\sigma}^{S} \tag{2.24}$$

where $\dot{\varepsilon} = \dot{\varepsilon}^{th} + \dot{\varepsilon}^{e} + \dot{\varepsilon}^{visc}$ and $\dot{\sigma}^{S}$ is the sintering stress. Their study focused on compression tests on nickel alumina (NiAl) powder. They were implemented both in FEM (macroscale) and DEM (microscale) domains for comparison. Neither experimental validation for the results nor details regarding DEM methodology were presented. Their results, however, demonstrated a good agreement between FEM and DEM-based models [49].

2.4.2 FOR DISCRETE ELEMENT METHOD

To model the sintering behaviour of discrete particles, the free or pressure-less sintering contact model at the particulate level (i.e., DEM) is often considered. This is based on the works of Bouvard and McMeeking [50], Parhami and McMeeking [51], Parhami et al. [52] and Coble [53]. Under its numerical scheme, the force consists of normal and tangential components. The normal force is further divided into attractive (or driving) and viscous (or dissipative) forces. The attractive force is given by [51], [54]:

$$F_{sinter} = \pi\gamma_s \left[\frac{\alpha}{\beta}\hat{R}\left(1 - cos\frac{\varphi}{2}\right) + a_s\, sin\frac{\varphi}{2}\right] \tag{2.25}$$

where γ_s is the particles surface energy, α and β are model parameters representing the dominant mass transport mechanism, $\hat{R}$ is the equivalent radius of interacting particles, φ is the dihedral angle, and a_s is the contact (or neck) radius of the sintering particles. The second force during interaction offers resistance to the overall sintering mechanism and is given by [54], [55]:

$$F_{viscous} = \frac{\pi a_s^4}{\beta \Delta_b} v \tag{2.26}$$

where v is the normal relative velocity (or approach velocity between the particles), and Δ_b is the effective diffusion coefficient, given by [54], [55]:

$$\Delta_b = \frac{D_g \delta_g \Omega}{k_B \bar{T}} \exp\left(-\frac{Q_b}{R_g \bar{T}}\right) \tag{2.27}$$

where D_g is the diffusion coefficient, δ_g is the thickness of the grain boundary, Ω is the atomic volume, k_B is the Boltzmann constant, Q_b is the activation energy, R_g is the universal gas constant, and $\bar{T}$ is the average temperature of the interacting particles, given by:

$$\bar{T} = \frac{T_1 + T_2}{2} \tag{2.28}$$

Therefore, the overall normal force can be expressed as:

$$F_N = \pi\gamma_s \left[\frac{\alpha}{\beta}\hat{R}\left(1 - cos\frac{\varphi}{2}\right) + a_s\, sin\frac{\varphi}{2}\right] - \frac{\pi a_s^4}{8\Delta_b} v \tag{2.29}$$

The other component of the sintering force acted in the tangential direction, and it was also viscous, as given by [54], [55]:

$$F_T = -\eta \frac{\pi a_s^2 \hat{R}^2}{\beta\Delta_b} \tag{2.30}$$

where η is the dimensionless viscous parameter, which signifies viscosity or slip resistance between the particles, typically set between 0 and 0.1 [54], [55]. Martin et al. stated that there is no established agreement about its value, but it must be set below 0.5 [56].

2.4.3 NUMERICAL STABILITY AND MASS SCALING OF PARTICLES IN DEM

As discussed in **Section 2.3.7** (**Page 20**), the timestep size must be set below a critical value for a numerically stable simulation. However, this holds true for linear spring or Hertz-Mindlin methods only. In contrast, it is crucial to consider that currently, our contact model has changed from Hertz-Mindilin to Sintering, which works on an entirely different set of equations, as given in **Section 2.4.2** (**Page 31**). Therefore, the required timestep size also needed to be redefined. Henrich et al. studied the timestep size issue for DEM-based sintering simulations, where they deduced the critical timestep would be [57]:

$$\Delta t = 10^{-20} \frac{k_B T R^4}{\gamma_s \Omega \delta \Delta_b} \tag{2.31}$$

However, this led to unrealistic computation times, requiring calculations for trillions of time steps. The scale-up of particle mass by several orders of magnitude is applied to counter this issue, as this strategy would not affect the particles' equilibrium, providing no significant inertial effect. In other words, as long as the particles stay in the quasi-static state, this strategy may be applied [39], [57]. This technique has been applied in few recent studies [55], [58], [59].

2.5. MODELLING OF HEAT CONDUCTION IN POWDER BEDS

2.5.1 FOR FINITE ELEMENT METHOD

Over the decades, several studies have attempted to model the effective thermal conductivity of powder beds. One of the earliest is Yagi-Kunii (1957) model, defined as the following [60]:

$$\frac{k_e^{\circ}}{k_g} = \delta\left(\frac{k_s}{k_g}\right) + \frac{(1-\epsilon-\delta)\beta}{\gamma\left(\frac{k_g}{k_s}\right) + \frac{1}{\frac{1}{\varphi} + \frac{D_p h_{rs}}{k_g}}} + \epsilon\beta\frac{D_p h_{rs}}{k_g} \tag{2.32}$$

where k_e° is the effective thermal conductivity, k_g – thermal conductivity of gas / surrounding fluid, δ is the contact area, γ is the length of solid affected by thermal conductivity, φ is the effective thickness of the fluid film in the void, ϵ is the void fraction, β is the ratio of average length between the centres of particles (l_p) to particle diameter (D_p), and h_{rs} is the heat transfer coefficient. The following simplified form was stated by Bugeda and Lombera (1999) [61] and Sofia et al. (2005) [62] in their work:

$$k_e = \frac{\mu k}{1 + \emptyset\frac{k_s}{k_g}} \tag{2.33}$$

where k_s is the thermal conductivity of the solid material, k_g is the thermal conductivity of the air or surrounding fluid, μ is the solid fraction, and $\emptyset$ represents the empirical coefficient, which is assumed as $0.02 \cdot 10^{2(0.7-\mu)}$. Sih and Barlow (2004) developed a predictive model for thermal conductivities, which was derived from Zehner–Schlünder (1970) model [63], stated as follows [64]:

$$\frac{k_{eff}}{k_f} = \left(1 - \sqrt{1-\varphi}\right)\left(1 + \frac{\varphi k_r}{k_f}\right) + \sqrt{1-\varphi}\left[\frac{2}{1-\frac{k_f}{k_s}}\left(\frac{1}{1-\frac{k_f}{k_s}} ln\left(\frac{k_s}{k_f}\right) - 1\right) + \frac{k_r}{k_f}\right] \quad (2.34)$$

where k_{eff} is the effective thermal conductivity, k_f the thermal conductivity of the atmosphere at ambient conditions and k_r is the thermal conductivity due to radiation among particles, which is defined as [64]:

$$k_r = 4F\sigma T_p^3 D_p \quad (2.35)$$

Where F is the radiation exchange or view factor (sphere to sphere in the present case), σ is the Stefan-Boltzmann constant, T_p the particle temperature and D_p is the average diameter of the particles. Qian et al. (2018) summarized the radiant exchange factor correlations (**Table 2.4**) by different authors [65]:

Table 2.4. Summary of radiant exchange factor correlations for the radiative conductivity [65]. Reuse license: 5516180873657 (Elsevier - International Journal of Heat and Mass Transfer).

Correlations	Radiation exchange factor, F_E
Argo and Smith (A-S)	$\frac{1}{(2/\varepsilon_r - 1)}$
Yagi and Kunii (Y-K)	$\frac{\varepsilon_g}{1+\frac{\varepsilon_g}{2(1-\varepsilon_g)}\frac{1-\varepsilon_r}{\varepsilon_r}} + \frac{1-\varepsilon_g}{(2/\varepsilon_r - 1)}$
Laubitz	$\frac{\varepsilon_r[1-(1-\varepsilon_g)^{2/3}+(1-\varepsilon_g)^{4/3}]}{(1-\varepsilon_g)}$
Wakao and Kato (W-K)	$\frac{2}{(2/\varepsilon_r - 0.264)}$
Schotte	ε_r
Zehner and Schlünder (Z-S)	$\left\{\frac{(1-\sqrt{1-\varepsilon_g})\varepsilon_g}{2/\varepsilon_r - 1} + \frac{\sqrt{1-\varepsilon_g}}{2/\varepsilon_r - 1} \times \frac{B+1}{B} \cdot \frac{1}{1+\frac{1}{(2/\varepsilon_r - 1)\Lambda_f}}\right\}$
Breitbach and Barthels (B-B)	$\left\{(1-\sqrt{1-\varepsilon_g})\varepsilon_g + \frac{\sqrt{1-\varepsilon_g}}{2/\varepsilon_r - 1} \times \frac{B+1}{B} \cdot \frac{1}{1+\frac{1}{(2/\varepsilon_r - 1)\Lambda_f}}\right\}$
Rosseland	$\frac{4}{3(a+\sigma_s)d_p}$
Chen and Churchill (C-C)	$\frac{2}{(a+2\sigma_s)d_p}$
Vortmeyer	$\frac{2B_r+\varepsilon_r(1-B_r)}{2(1-B_r)-\varepsilon_r(1-B_r)}$

Sih's and Barlow's (2004) model proved closer to the measured values of alumina powder compared to Yagi-Kunii's and a modified Zehner–Schlünder's (Zehner-Bauer) model, as plotted in **Figure 2.11**. However, comparing the calculations from their equations against data presented by numerous authors resulted in a ±30% error (**Figure 2.12**) [64].

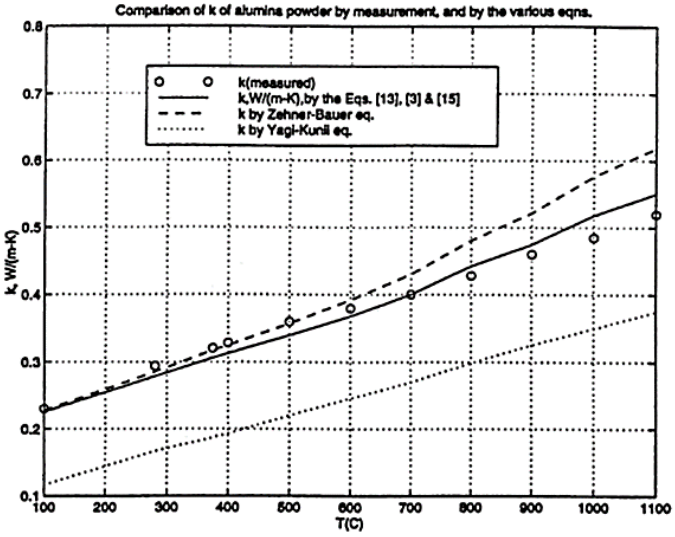


Figure 2.11. The thermal conductivity of alumina powder (porosity = 0.71) by measurements (Landolt-Bornstein), compared with the values predicted by Equation (25), Zehner-Bauer's equation, and Yagi-Kunii's equation [64]. Reuse license: upst/03037406 (Taylor & Francis Group).

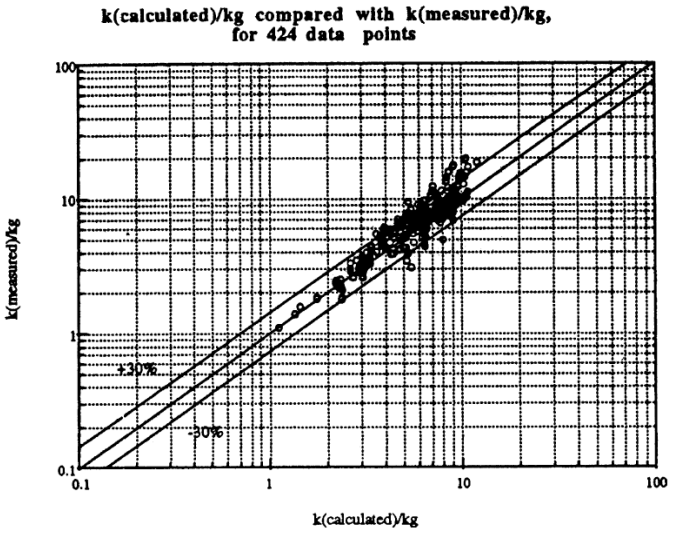


Figure 2.12. Normalized data of the thermal conductivity of powder beds (k/k_g) compared to the normalized predicted thermal conductivity values according to Equation [64]. Reuse license: upst/03037406 (Taylor & Francis Group).

In the case of considering fine powders in FEM simulations, empirical thermal conductivities can also be adapted alternative to previously discussed methods. For instance, Huang and El-Genk (2000) measured the thermal conductivities of fine alumina powders: 0.01 µm and 0.1 µm. Their results indicated strong dependability on temperature (**Figure 2.13**) [66]:

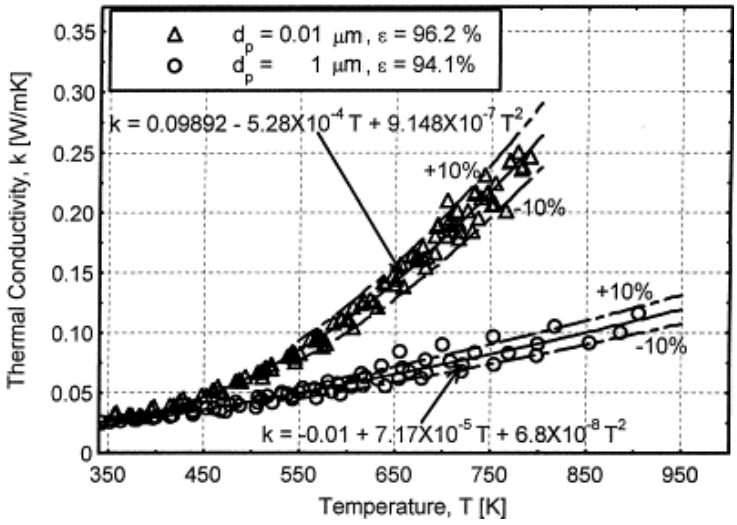


Figure 2.13. Thermal conductivity data and correlations for the 0.01 and 1.0 µm Al_2O_3 powders, with smaller powders exhibiting higher conductivities, while both indicate temperature dependence [66]. Reuse license: 5516200122116 (Elsevier - Energy Conversion and Management).

2.5.2 FOR DISCRETE ELEMENT METHOD

Determining the temperature field in the powder bed (or granular assemblies of large particles) can be time-consuming and computationally expensive. Therefore, particles are assumed to be isothermal to save computation time [67]. The heat flow between the two particles is formulated as follows:

$$Q_{i,j} = h_c \Delta T_{i,j} \tag{2.36}$$

where $\Delta T_{i,j}$ is the temperature difference, and h_c is the heat transfer coefficient or thermal contact conductance. Batchelor and O'Brien formulated the thermal contact conductance between the two particles as follows [68]:

$$h_c = 2\lambda a_{ij} \tag{2.37}$$

where λ is the thermal conductivity of the particle and a_{ij} is the contact radius between the interacting particles, as illustrated in **Figure 2.14** below:

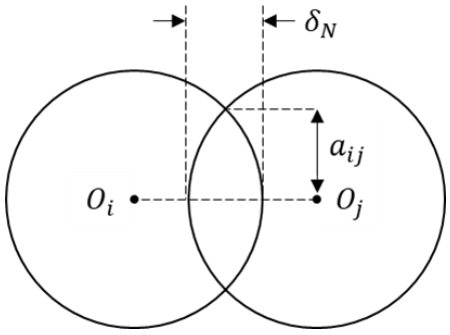


Figure 2.14. Contact radius (a_{ij}) and normal overlap (δ_N) of the interacting particles i and j.

The contact radius can be derived from the Coble geometric model as follows [53], [55]:

$$a_{ij} = \sqrt{2R^* \delta_N} \tag{2.38}$$

where R^* is their equivalent radius given by [55]:

$$\frac{1}{R^*} = \frac{1}{R_i} + \frac{1}{R_j} \tag{2.39}$$

and δ_N is their overlap, which can be calculated as follows [55]:

$$\delta_N = R_i + R_j - \left|\Delta X_{ij}\right| \tag{2.40}$$

where $\left|\Delta X_{ij}\right|$ is the distance between the centroids of the particles.

In contrast, Chaudhuri et al. (2006) proposed another heat conduction model for DEM. However, it could only be combined with the Hertz-Mindlin contact model. The heat coefficient (h_c) in their model was given as [69]:

$$h_c(T) = \frac{4k_i(T)k_i(T)}{k_i(T) + k_j(T)}\left(\frac{3F_N R^*}{4E^*}\right)^{\frac{1}{3}} \tag{2.41}$$

where $k_i(T)$ and $k_j(T)$ are temperature-dependent (T) thermal conductivities of the interacting particles, F_N is the normal force between them, calculated from the Hertz-Mindlin contact model (**Section 2.3.5, Page 16**), and E^* is their effective Young's modulus.

The temperature change as a result of this heat conduction for each particle is calculated as follows:

$$m_P c_p \frac{\Delta T}{\Delta t} = \sum Q_{heat} \tag{2.42}$$

where $\sum Q_{heat}$ is the sum of conductive and convective heat fluxes.

Furthermore, Gusarov et al. (2003) estimated the effective thermal conductivity of packed powder beds based on packing density, average coordination number (number of contact points between any particle and its neighbours) and contact size, as follows [70]:

$$\frac{\lambda_e}{\lambda_s} = \frac{pn}{\pi} x \tag{2.43}$$

where λ_e is the effective thermal conductivity, λ_s is the thermal conductivity of the bulk material, *p* is the packing density, n is the average coordination number, and $x = \frac{a}{R}$ is the contact size ratio, i.e., contact spot radius a to the sphere radius R (**Figure 2.15**). While their model considered an evolution in the contact area, which would be expected during sintering, but it was restricted to equal-sized particles. They compared thermal conductivities against the contact size ratio with two other models, displaying a satisfactory agreement in the range $0 < x < 0.3$ (**Figure 2.16**) [70].

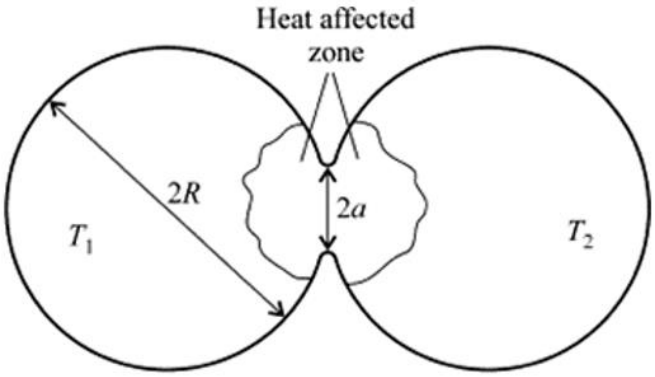


Figure 2.15. Contact of radius a between two spherical particles with the same radius R and temperatures T_1 and T_2 illustrating their immediate heat affected zone during thermal conduction [70]. Reuse license: 5516200464214 (Elsevier - International Journal of Heat and Mass Transfer).

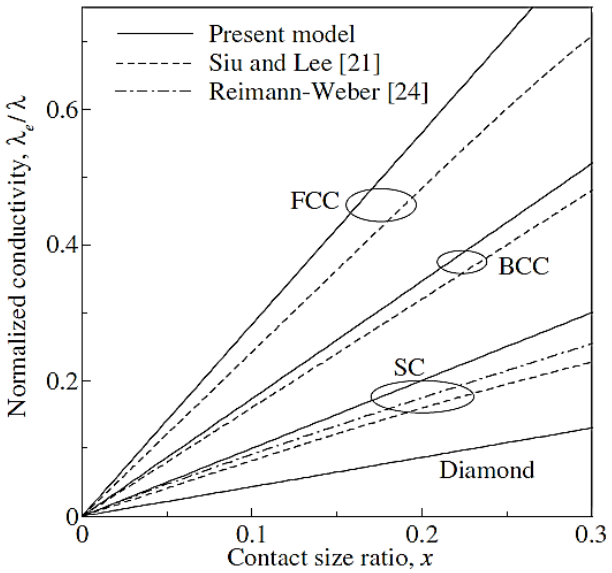


Figure 2.16. Effective thermal conductivity of Gusarov et al. model powder bed versus contact size, compared against the face centre cubic (FCC), body centred cubic (BCC), and simple cubic (SC) structure, displaying a good agreement. Additionally the normalized conductivity of diamond structure has also been compared with the rest of the models [70]. Reuse license: 5516200464214 (Elsevier - International Journal of Heat and Mass Transfer).

2.6. STATE-OF-THE-ART IN NUMERICAL SIMULATIONS OF SINTERING AND PBSLP

2.6.1 FINITE ELEMENT METHOD (FEM)

Shuai et al. [71] simulated the dynamic temperature field during the SLS of ceramic powder: Nano-HAP. Their ANSYS-based model used a solid cuboid (**Figure 2.17**) with the thermal properties of HAP powder and unidirectional laser movement on the surface. The simulated results demonstrated the effect of the transient temperature field on sintering, using the different speeds of laser: 600, 200, and 120 mm/min, which generated the peak surface temperatures of 1064 °C, 1289 °C, and 1466 °C, respectively. Furthermore, they carried out SEM and XRD experiments on sintered nano-HAP. Only at 200 mm/min closely bonded powders were observed, while at the higher speeds, powders did not sufficiently sinter, and the lower speed caused melting and decomposition (**Figure 2.18**). This study aimed to provide a guideline for choosing optimum laser parameters for better sintering performance using simulation methods. In another study, Fu et al. [72] developed a model with a layer build-up approach. The first layer would be activated and laser-scanned, followed by subsequent layers similarly. A more comprehensive study of the SLS process was carried out by Liu et al. [73], which investigated the macro-scale thermal evolution within the porous structure of Pure Iron powder. They developed 3D FEM models of multi-layer powder stacking with uniformly distributed cubic-shaped powders (**Figure 2.19**). The gap between each powdered particle represented porosity. They used a laser power of 5-20 W, a scanning speed of 5-30 mm/s and a beam diameter of 50 μm. Their results drew experimental, and simulation comparisons of (1) the length of the sintered piece and (2) shrinkage depth, with good agreement, observed between the two (**Figure 2.20**).

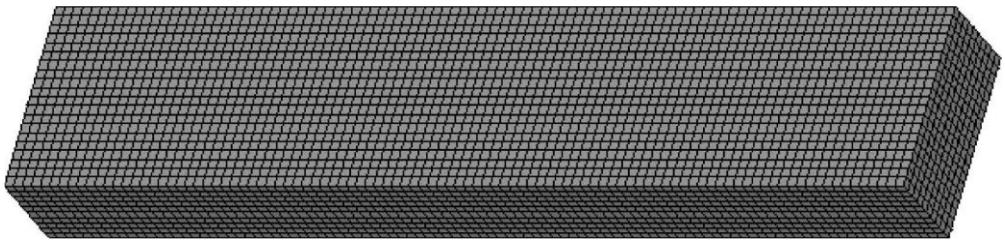


Figure 2.17. The 3D finite element model, illustrating the cuboidal DCD8 mesh of the laser scanned body [71]. Reuse license: nmcm/03037405 (Taylor & Francis Group)

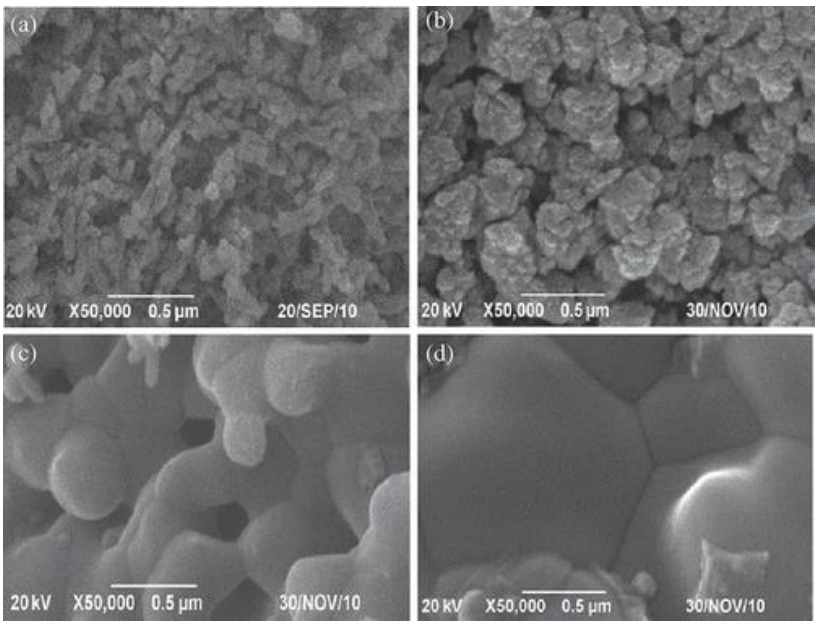


Figure 2.18. SEM images under different conditions: **(a)** the initial nano-HAP, **(b-d)** the sintered nano-HAPs when the laser scanning speeds are 600, 200 and 120 mm/min, respectively [71]. Reuse license: nmcm/03037405 (Taylor & Francis Group)

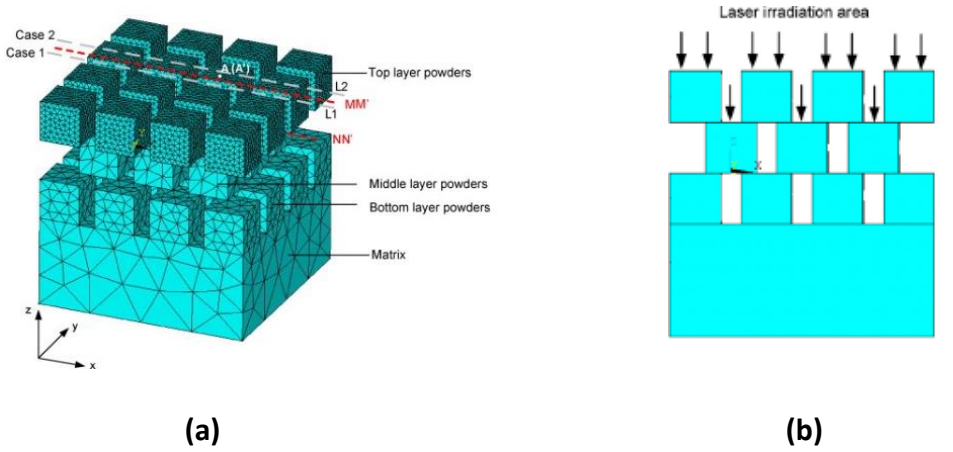


Figure 2.19. (a) 3D view of Construction of the 3D multi-layer powder stacking model: 3D schematic for the simulation + Laser irradiation region. **(b)** Front view [73]. Reuse license: 5516210671050 (Elsevier - Journal of Materials Processing Technology).

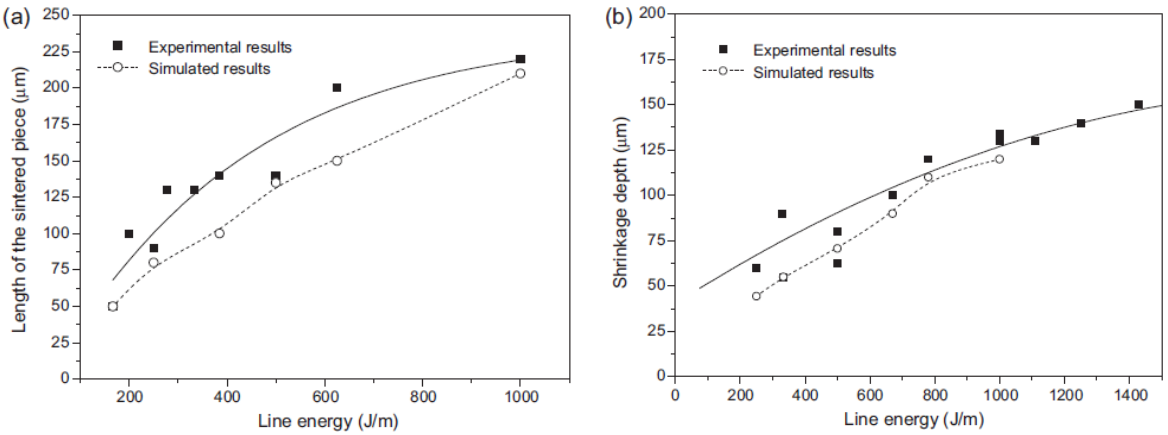


Figure 2.20. Comparison between the experimental and simulated results in the **(a)** length of the sintered piece and **(b)** shrinkage depth [73]. Reuse license: 5516210671050 (Elsevier - Journal of Materials Processing Technology).

Hussein et al. [74] used a nonlinear transient thermomechanical model to study the stress and temperature field in 316L Steel produced by LPBF. The study utilised the finite element approach for high-temperature gradients and thermal stresses. To research the various LPBF parameter values, they created a non-linear transient finite element model using the ANSYS software. In contrast, Fan et al. [75] studied the effect of laser energy density (LED) on alumina's geometry, microstructure, and micromechanical properties during the SLM process. The simulation was based on FVM, with a single-track laser movement on the material surface.

However, they used an empirical relation to approximate the effect of porosity and morphology of powder particles on the thermal conductivity given in **Equation (2.34)**. The geometry of their model was also developed in ANSYS FLUENT and represented by a uniformly structured meshed continuous block (**Figure 2.21**). In addition to their original study, they also concluded that the HPDL laser (0.98 μm) performed better than CO_2 and Nd:YAG counterparts, especially in uniform heating and smooth surface finishing. Generally, the incident light near-infrared (NIR) spectra are weakly absorbed and highly reflected by the bulk Alumina. Contrarily, alumina powder's absorption is much higher than bulk. With the use of a finite element, Conti et al. [76] created a thermomechanical model of the growth process that examined how the behaviour of the part varies as the material transitions between the powder, liquid, and solid phases, in combination with element birth-and-death technique. As mentioned earlier, this technique is known as activation-deactivation [72].

The machine-specific processing standards could not be established due to the novelty of the SLS/M process. Therefore, their calibration heavily depends on trial and error, leading to significant wastage of raw material. Waste management directly affects manufacturing sustainability. Consequently, Olivier et al. [77] examined the possibility of replacing inefficient R&D methods with numerical modelling. Their study attempted to establish the reliability of numerical thermal models to predict material behaviour. They used the data from the previous publications' single-track laser experiments on tungsten carbide-cobalt (WC-Co) to develop the numerical simulations in Abaqus (**Figure 2.22-a**). Those experimental results contained the melt-track width, approximated in their simulations by observing the thermal field where the temperatures exceeded the melting point. However, this study led to a mixed set of accuracies in results (**Figure 2.22-b**).

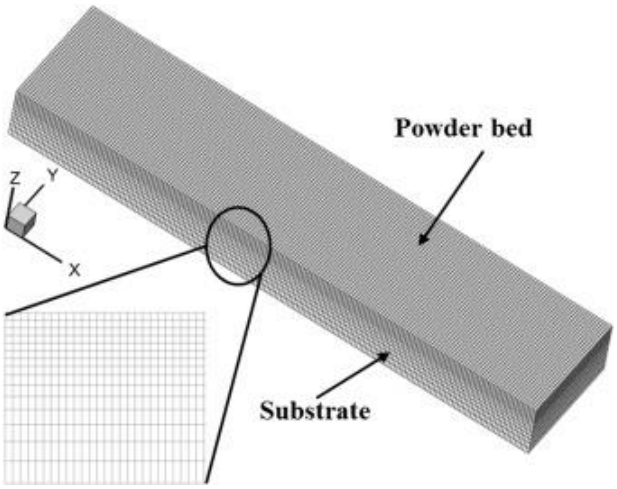


Figure 2.21. The meshed powder-substrate structure in the FVM model: Mesh is finer near the surfaces to capture the drastic change in t:emperature due to laser scan. [75]. Reuse license: 5516210860084 (Elsevier - Ceramics International).

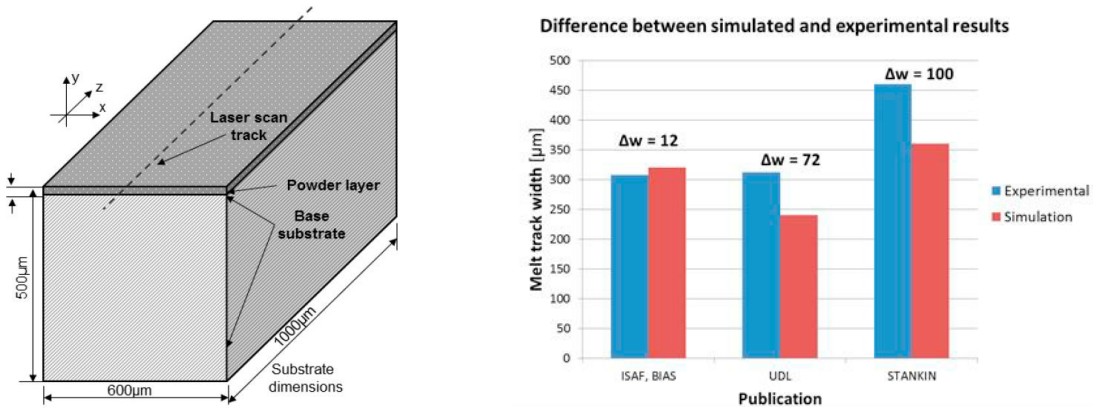


Figure 2.22. (a). The numerical model geometry used for the study of past publications, with dark gray top slack representing the powder layer, while light grey base representing the substrate **(b).** Comparison between numerical and experimental results of melt track width in different studies, illustrating their absolute errors [77]. Reuse license: Open Access / Creative Commons.

2.6.2 DISCRETE ELEMENT METHOD (DEM) FOR LASER HEATING & SINTERING

The SLS is a complex multi-physical process. It greatly depends on powder particles' morphology, size distribution, microstructure, melt pool formation, etc. In contrast, the FEM models are restricted to mainly continuum bodies. Thus, they rely heavily on the empirical data for SLS simulations, potentially leading to higher inaccuracies. As the DEM simulations are carried out by considering the discrete set of particles, they can answer the limitations presented by the FEM since the morphology of individual particles and their interaction with others can be considered.

Lee et al. (2017) developed the DEM model in an open-source software LIGGGHTS, for the powder flow and laser heating of stainless steel (15-5 PH1) powder during SLS. Their simulation demonstrated powder deposition, recoating, laser heating, and holding (**Figure 2.23**) [11].

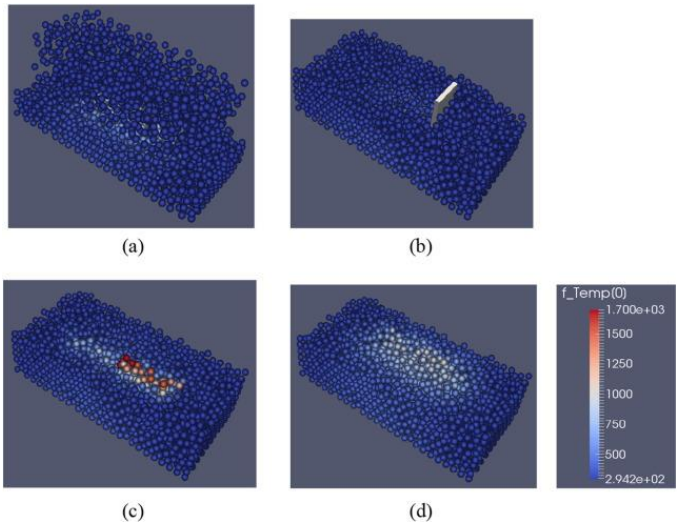


Figure 2.23. Temperature distributions in the DMLS process simulations. **(a)** Vertical deposition of particles, **(b)** recoating process, **(c)** laser heating of particles, and **(d)** holding period. The particles are labelled using colours to present their temperature (unit: K) [11]. Reuse license: 5516220269197 (Elsevier - Powder Technology).

However, there were some critical limitations, such as equal-sized particles and uniform laser heating (i.e., no gaussian distribution considered) due to software restrictions. Moreover, they did not consider the effect of the coating blade's dimensions [11]. Li et al. (2016) adapted a ray tracing method to calculate the energy absorption by the sphere particles in their DEM-based study on laser absorption by metal powder bed in SLS processes. They generated a randomly packed powder bed (PB) in a cuboid container using the Probabilistic Analytic Consecutive Kinetic Simulation (PACKS). The model worked by (laser) rays shot vertically on the PB, with a specific size, direction, and power. The particle absorbs a portion of the incident ray, and the remainder is reflected. They concluded that top-hat laser was better than Gaussian, and powder bed structure absorbs better than flat structure [78]. In comparison, Xin et al. (2017) simulated laser heat transfer in the powder bed based and developed their model based on a customized Monte Carlo – raytracing method and the Mie scattering theory. They investigated the effect of radiative transfer and heat diffusion in translucent polymer materials, both in DEM and FEM. They observed higher temperature gradients with the ray-tracing DEM approach than FEM volumetric heat distribution, due to higher absorption by the region closer to the surface under scattering. Furthermore, they demonstrated that photons were concentrated in a cylindrical shape with volumetric heat flux definition, while distributed widely with scattering effect but with greater concentration near the surface [79].

Yang et al. (2019) employed both DEM and FEM to simulate the SLS of stainless steel 316L. Their study was based on non-isothermal phase-field modelling of microstructure. Initially, they generated a powder bed of non-uniform size distribution in the DEM-based simulator YADE. Furthermore, they implemented the SLS model in FEM based MOOSE framework. Their model employed different laser powers and speeds. They also considered particles'

morphological changes and microstructure densification. The results observed that the parameters 20 W, 100 mm s^{-1} induced partial melting, while 20 W, 80 mm s^{-1} created a full melt pool. Only 20 W, 150 mm s^{-1} proved effective in maintaining the solid-state sintering (**Figure 2.24**) [80]. Quintana-Ruiz and Campello (2023) proposed a comprehensive methodolgy to simulate the SLS process, purely using DEM. The used a lumped heat transfer approach to describe the heat dissipation within the powder bed. They introduced a adhesive bonded particle approach to model the sintering phenomena, considered the phase transformation on elevated temperatures, and provided a few examples to illustrate the implementation of the methodolgy [81].

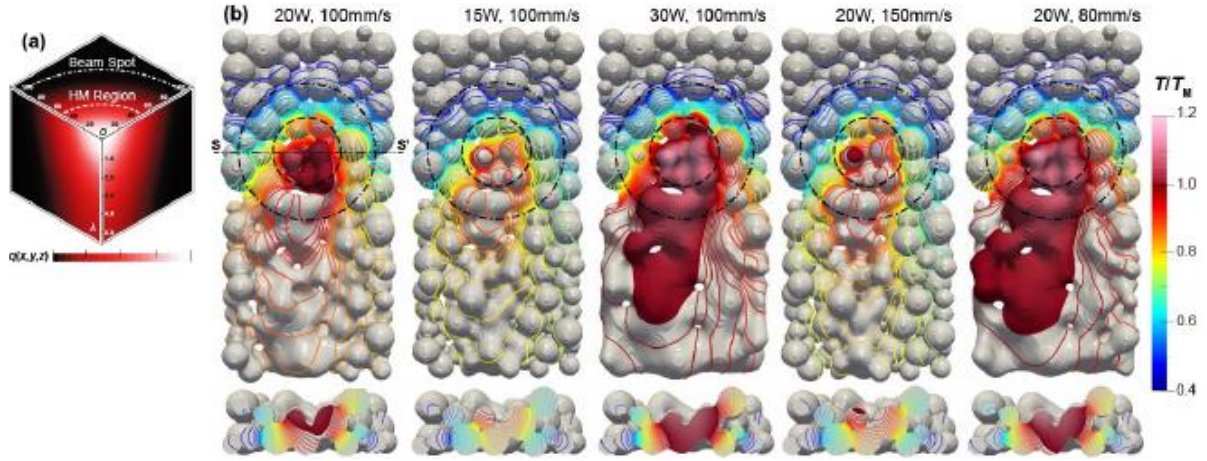


Figure 2.24. (a) Schematics for distribution of the volumetric laser energy deposition. **(b)** Relative temperature field induced as a result of laser scan (with different processing parameters) and resulting particle sintering/melting, after laser beam has travelled over half of powder bed length [80]. Reuse license: Open Access / Creative Commons.

Yan et al. (2014) used discrete element simulations to model the co-sintering process of Ni/BaTiO3 multilayer ceramic capacitors (MLCCs). They defined model parameters such as particle size distribution, layer thickness, and sintering temperature, and developed a 3D model of the MLCCs. They simulated the co-sintering process and analyzed the microstructure evolution of the MLCCs using various techniques, such as scanning electron microscopy and

X-ray diffraction. The concluded that the formation of discontinuities during the sintering process can be reduced by employing a rapid heating method, which helps to minimize the sintering mismatch between the electrode and dielectric [82]. In contrast, Teixeira et al. (2022) developed a thermomechanical for high heat rate sintering of alumina, using DEM framework MUSEN. They formulated the sintering, heat conduction and heat loss models. The models were combined to produce the simulation for densification process. The processes were analysed at 1250 °C and 1350 °C. Higher temperature demonstrated increased initial and final shrinkage rates [55].

2.6.3 DISCRETE ELEMENT METHOD (DEM) FOR POWDER DELIVERY SYSTEMS

The parameters for the contact model and material properties play an important role in correctly implementing the DEM model. The contact model considers static friction, dynamic (rolling) friction, and coefficient of restitution as base parameters. Material property parameters include density, shear or Young's modulus, particle size and Poisson's ratio. It is a standard practice to use a smaller than the actual value for Young's modulus to represent a soft-sphere model that leads to reduced computational times. Chen et al. (2017) demonstrated the effect of smaller than actual Young's modulus (where E_0 is the actual value) during the transverse mixing of particles within a rotating drum in their DEM study. The contact forces were calculated using the Hertz-Mindlin model. They determined that reducing the moduli up to $0.001E_0$ did not affect the particle mixing. Going smaller reduced the velocity gradient in the active layer and significantly decreased the mixing rate [41].

Powder spreading has a substantial role in powder bed fusion processes. A smooth surface and a greater number of contacts (i.e., high solid volume) among the particles signify a good quality powder bed that reduces the porosity in the final processed part. Several studies in the DEM domain have considered the effect of a roller [18], [83] and blade type [84] spreaders on the quality of powder beds. For instance, Haeri et al. (2017) performed simulations in EDEM to demonstrate the effect of particle shape and operating conditions of roller and blade on the powder-bed quality of PEEK, depicted by surface roughness and solid volume fraction. They considered different aspect ratios (A_r) of particles and modelled the elliptical-shaped particles by joining multiple spheres (**Figure 2.25**). Their study concluded that the roller outperformed the blade, while larger A_r and higher translational velocities decreased the bed quality (**Figure 2.26**) [83].

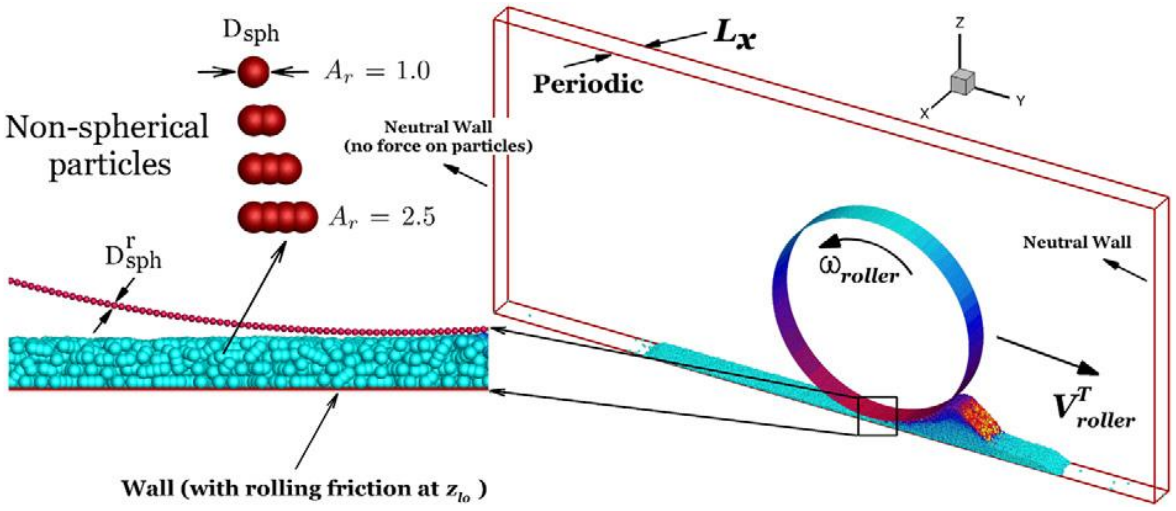


Figure 2.25. Representation of the roller based powder delivery simulation setup by Haeri et al., representing the roller geometry, differently aspected powder bed layering, periodic and neutral wall boundaries [83]. Reuse license: Open Access / Creative Commons.

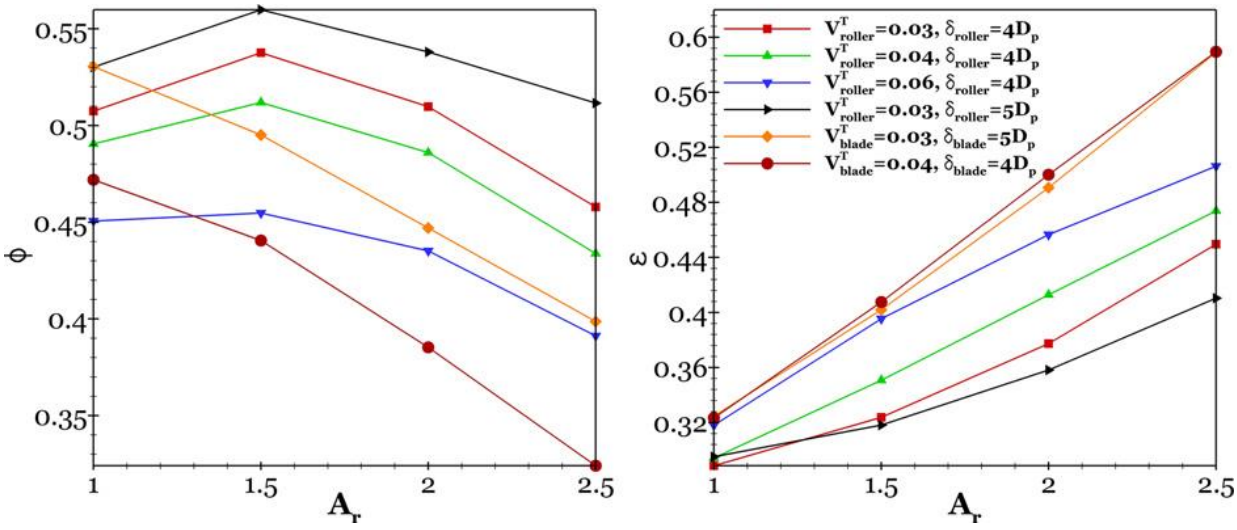
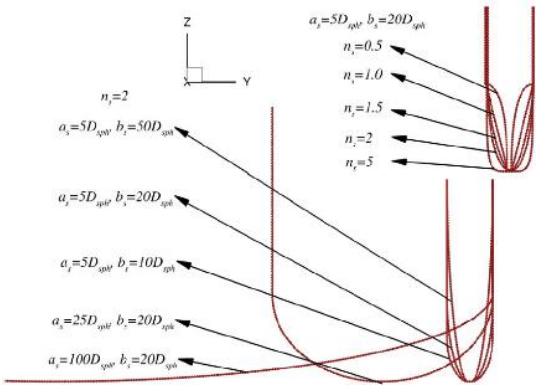


Figure 2.26. The effects of particle shape, $V_{spreader}$, $\delta_{spreader}$ and the spreader-type on the bed quality. The quality is quantified by a volume fraction ϕ, which is shown to be increasing up to 1.5 aspect ratio (A_r) and then decreasing, and roughness parameter ϵ, which continues to increase with increasing aspect ratio [83]. Reuse license: Open Access / Creative Commons.

In contrast to his previous study, Haeri proposed another method to improve the powder-spreading process. It involved innovative optimization of the blade spreader that used a super-elliptic edge profile (**Figure 2.27-a**), again considering different aspect ratios (A_r) for the PEEK

particles with EDEM's multi-sphere approach that represented non-spherical bodies (**Figure 2.27-b**). He used the width (a_s), height (b_s) and overall profile shape (n_s) as the three main controlling parameters. The results illustrated the comparisons for the volume fraction and surface roughness against the type of spreader. The optimized blade performed marginally better in terms of producing the solid volume fraction (or packing density) but outperformed the roller in reducing the surface roughness (**Figure 2.28**) [85].

(a)



(b)

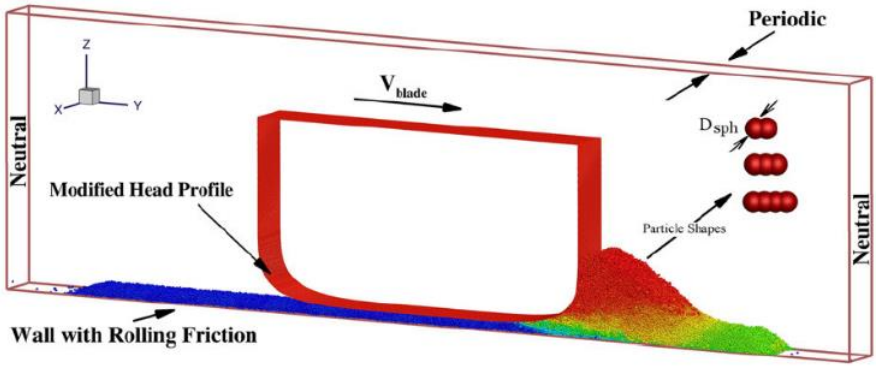


Figure 2.27. (a) Different spreader profiles of a modified blade profile proposed by Haier, that was used for the optimization **(b)** Representation of the device simulation set-up, modified blade geometry, initial preparations, periodic and neutral boundaries [85]. Reuse license: Open Access / Creative Commons.

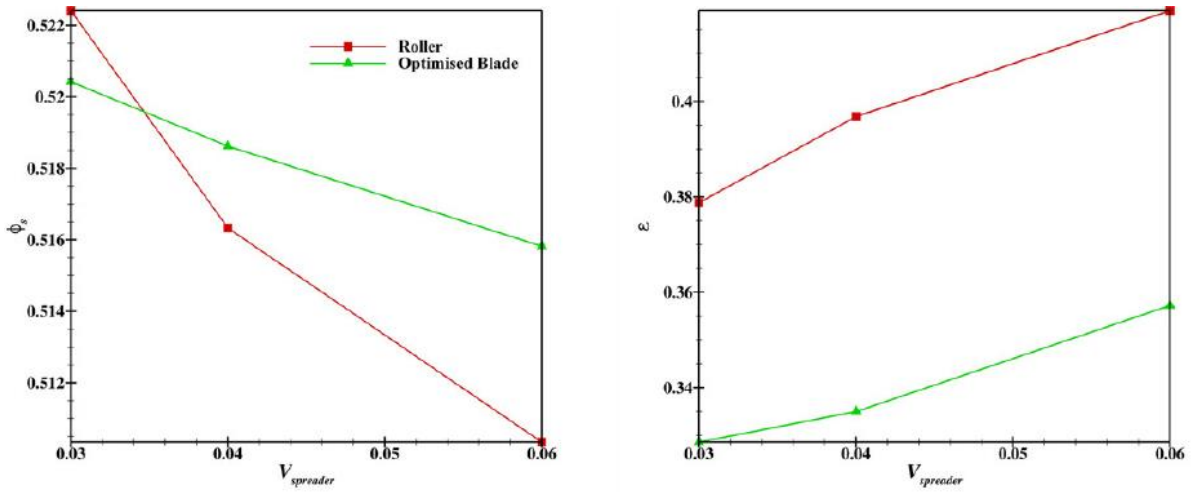


Figure 2.28. Comparison between a roller and the optimised blade type spreader (the legend applies to both figures). Clearly, the optimised blade-type spreader produces a powder bed with volume fractions at least as good as the roller and surface roughness values significantly better [85]. Reuse license: Open Access / Creative Commons.

Zhang et al. studies focused entirely on the roller-spreading process for alumina ceramic powder (**Figure 2.29**). They used the Hertz-Mindlin JKR model in their simulations and demonstrated the effect of powder layer thickness (H) and three parameters for roller, i.e., translation velocity (Vs), rotational speed (ω), and diameter (D), on the powder bed density, within the tabulated range (**Table 2.5**) [18].

Table 2.5. The value of roller-spreading parameters [18]. Reuse license: Open Access / Creative Commons.

Parameters	Value
Roller's translational velocity V_s (mm/s)	40/60/80/100/120/140/160
Roller's rotational velocity ω (rpm)	40/80/120/160/200/240/280/320
Roller's diameter D (mm)	3/4/5/6/7/8
Powder layer thickness H (μm)	50/75/100/125/150/175

However, they did not offer insight into parametric optimization. Their study generally determined that higher translational velocity of roller led to poor powder-bed density, while angular velocity had an insignificant effect. This conclusion also backed up the study of Haeri

et al. [83]. Furthermore, increased diameter caused compression to a greater number of particles, leading to improved densification. The layer thickness had especially the most significant effect, with a lower limit of 50 µm tended to produce voids. The upper limit of 150 µm thickness produced higher density and uniformity (**Figure 2.30**) [18].

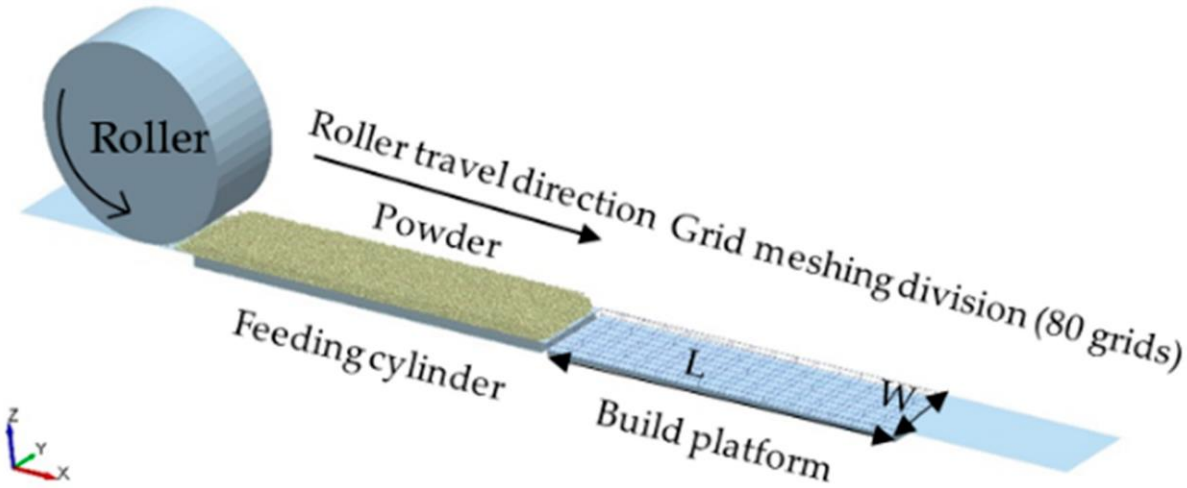


Figure 2.29. Representation of a DEM simulation setup of roller-spreading process, feeding cylinder, and build platform (for laser processing) by Zhang et al. [18]. Reuse license: Open Access / Creative Commons.

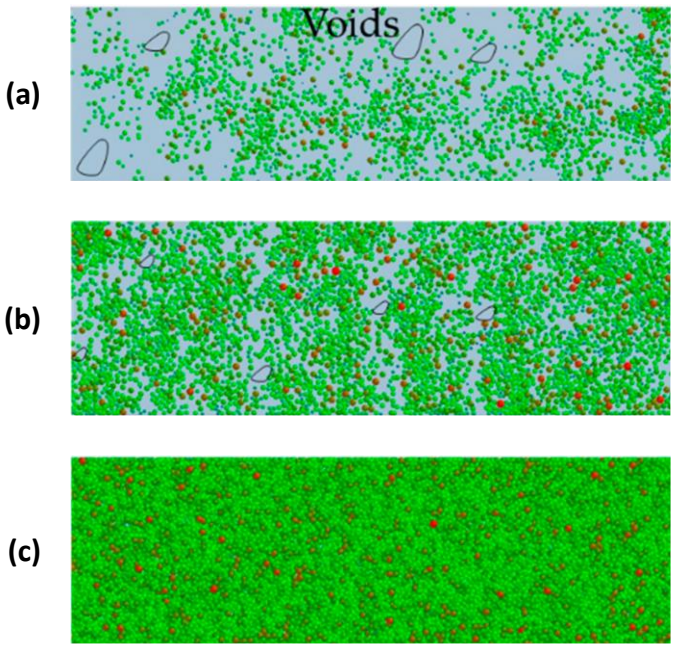


Figure 2.30. The powder-bed characteristics under different layer thickness values: **(a)** H = 50 µm representing signifant voids, **(b)** H = 75 µm representing reduced voids compared to 50 µm but still visible, and **(c)** H = 150 µm with no visible voids [18]. Reuse license: Open Access / Creative Commons.

Conclusively, the general aim of the DEM simulations is to assist in evaluating the effect of different parameters on the powder bed spreading process. A smaller value for Young's or shear modulus is used in DEM simulations for the 'soft sphere model' that accounts for any deformations within the interacting particles. This can be reduced to 1/1000th of the original value before affecting the dynamics of the results. Such consideration also helps widen the critical Rayleigh time step, which helps decrease the overall simulation's solver time. Furthermore, it can be concluded from the discussed studies that the roller usually has better powder-spreading abilities than the blade in terms of final powder quality. However, Haeri's (2017) proposed optimized blade would be an exception to this. Besides, the layer thickness plays the most crucial role, while rotational velocity was insignificant out of the discussed parameters. It can generally be established that larger roller sizes with lower translational velocities deliver better powder bed quality.

2.7. SUMMARY

The discussed work in the literature is not limited but only used as examples of ongoing development in the field of numerical simulations of powder bed 3D printing. The aim was to gain core subject knowledge for our applications and identify the potential gaps that could be addressed. The summary is given in **Table 2.6** (next page).

As mentioned previously, the modelling work focused on alumina and hydroxyapatite (HAP) ceramics in this thesis. Although alumina has been a subject of a few studies related to the modelling of sintering as well as powder delivery systems, very limited work was identified for HAP in the literature. Therefore, we specifically addressed HAP to develop the simulations of single-track laser powder-bed interaction in line with similar experiments.

Table 2.6. Summary of literature with its methodologies and applications.

Research		Methodologies				Applications				
Year	Authors	FEM/FVM	DEM	Other	Powder delivery	Vol	RT	Sinter/Melt	Other	Material
2023	Quintana-Ruiz and Campello [80]		×			×		×		SS 316
2022	Wakai et al. [86]		×					×		-
2022	Teixeira et al. [55]		×					×		Al_2O_3
2019	Olivier et al. [77]	×				×				WC-Co
2019	Yang et al. [80]	×	×	×		×		×		SS 316L
2018	Fan et al. [75]	×				×		×		Al_2O_3
2018	Conti et al. [76]	×				×				SS 316L
2017	Xin et al. [79]		×				×			PEEK
2017	Lee et al. [11]		×		×	×				15-5 PH1
2017	Chen et al. [41]		×						×	SiO_2
2016	Li et al. [78]		×				×			SS, Cu, Fe
2014	Fu et al. [72]	×				×				Fe
2013	Shuai et al. [71]	×				×				HAP
2013	Hussein et al. [74]	×				×				SS 316L
2012	Liu et al. [73]	×				×		×		SS 316L, Fe
2020	Zhang et al. [18]		×		×					Al_2O_3
2017	Haeri et al. [83]		×		×					PEEK
2017	Haeri [85]		×		×					PEEK

3. CERAMIC POWDER GENERATION AND CALIBRATION USING DEM

Highlights:

1. DEM powders calibration using the angle of repose method for alumina and hydroxyapatite

2. Modelling of powder delivery and layer formation process

3. Evaluation of powder bed densities

4. Development of a ray tracing algorithm to evaluate powder bed surface roughness

3.1. INTRODUCTION

The DEM methodologies developed in this chapter covered the generation of the correct volumetric size distribution of powders, the process for the powder delivery system using the blade, and the evaluation of the packing density of the processed powder bed. Furthermore, a ray tracing method was developed to evaluate the surface roughness of the powder bed.

The studies were carried out on two different powders, i.e., alumina and hydroxyapatite. Both powders in DEM were experimentally calibrated for their flowability, and their volumetric size distributions were generated corresponding to the powders used in experiments. It was done to ensure that their dynamic behaviour in the simulation environment would be as close as possible to their actual (or physical) counterparts.

The alumina powders considered in this study were fine alpha-alumina: P172LSB, Alteo, France. Due to their non-free-flowability, they were post-processed and spray-dried, with their final form referred to as Al-SP [87]. The data (including images) for experimentally measured volumetric distributions of the powders, as mentioned earlier, and their delivery parameters (discussed later in **Section 3.2.3**) were provided by Dr Qirong Chen (previously affiliated with the Belgian Ceramic Research Centre – BCRC).

The hydroxyapatite (HAP) powders: Commercial Urodelia® 300-08-2 HAP powders, used in the studies by Navarrete-Segado et al. [88], [89] at the Centre Inter-Universitaire de Recherche et d'Ingénierie des Matériaux (CIRIMAT), were considered as the benchmark for our DEM simulations, for their size distribution and flowability characteristics. The experimental data for these powders (including corresponding images) was provided by Dr Pedro Jesus Navarrete-Segado (previously affiliated with the CIRIMAT).

Furthermore, each powder served two different purposes for our DEM model development. We used alumina as benchmark powder, specifically for generating correct powder size distribution, modelling of blade-based powder delivery system, calculating resulting packing densities and estimating powder bed surface roughness. In contrast, the hydroxyapatite powders were only considered for generating the correct powder size and calibration. They were specifically used as the base for the later chapter for studying the selective laser sintering process.

3.2. ALUMINA POWDERS

3.2.1 DEM POWDER GENERATION AND VOLUMETRIC DISTRIBUTION

A total of 5 alumina powder samples of different volumetric size distributions were considered, labelled as SP-1, SP-2, SP-3, SP-4, and SP-5. The distributions of powders SP-2 to SP-4 were formed by mixing SP-1 and SP-5 by varying their composition. Their percentages are given in the following **Table 3.1**:

Table 3.1. Composition of different powders used in the powder delivery simulations and experiments.

Composition w.r.t.	Powder				
	SP-1	**SP-2**	**SP-3**	**SP-4**	**SP-5**
SP-1	100%	70%	50%	30%	0%
SP-5	0%	30%	50%	70%	100%

As close as possible to BCRC's data, we generated the particle sizes for our simulations in the DEM solver called Altair EDEM (or simply EDEM). An exact match is not possible since the solver randomly generates the particles by considering the given parameters: **(a)** Physical radius, **(b)** mean/average scale, **(c)** standard deviation, **(d)** upper and **(e)** lower limit/cap, scaled with respect to either radius or volume. In the simulations, the physical radius for all the powders was always set equal to the average obtained from experimental distributions. The rest of the parameters **(b)** to **(e)** were varied until a size distribution similar to experiments was obtained, given in **Table 3.2**. The data was regenerated at least 5 times for each set of powders in DEM to check for and avoid any anomalies. Furthermore, this was done to ensure that the average simulated distribution was maintained within 10% of the given range of the corresponding experiment.

Table 3.2. Parameters used in EDEM to generate the respective size distributions for the powders, for alumina powders.

Classification		Parameters				
Powder type	**Diameter size range [µm]**	**Radius [m]**	**Scaling**	**Standard deviation**	**Limit/cap***	
					Lower	**Upper**
SP-1	D = 20	$1.0 \cdot 10^{-5}$	1	0	0	0
	D ≥ 25	$2.2 \cdot 10^{-5}$	0.4	0.5	0.55	1.9
SP-2	D < 25	$1.25 \cdot 10^{-5}$	1	0.5	0.8	0.99
	25 ≤ D < 150	$1.5 \cdot 10^{-5}$	0.75	1.5	0.84	5
	D ≥ 150	$7.5 \cdot 10^{-5}$	1	0.5	0.85	1.2
SP-3	D ≤ 25	$3.9 \cdot 10^{-5}$	1	0.5	0.25	0.31
	25 ≤ D < 117	$3.9 \cdot 10^{-5}$	1	0.85	0.32	1.5
	D ≥ 117	$3.9 \cdot 10^{-5}$	1	0.75	1.5	3
SP-4	D ≤ 19.5	$3.9 \cdot 10^{-5}$	1	0.5	0.25	0.31
	19.5 < D ≤ 172	$4.9 \cdot 10^{-5}$	1	0.7	0.2	1.75
	D > 172	$4.9 \cdot 10^{-5}$	1	1	1.75	2.25
SP-5	20 ≤ D < 25	$1.125 \cdot 10^{-5}$	1	0.5	0.89	1.1
	25 ≤ D < 50	$1.875 \cdot 10^{-5}$	1	0.5	0.67	1.3
	50 ≤ D < 125	$4.375 \cdot 10^{-5}$	1	0.75	0.6	1.4
	125 ≤ D < 175	$7.500 \cdot 10^{-5}$	1	0.35	0.8	1.2
	175 ≤ D ≤ 200	$9.375 \cdot 10^{-5}$	1	0.1	0.975	1.025

* w.r.t. particle radius (or diameter)

The SP-1 (**Figure 3.**) was the finer of the two main powders, the other being SP-5 (**Table 3.1**). The average size of particles in this powder was 44.46 µm. Their optical laser image and DEM solver generation can be visually compared in **Figure 3.-b,c**. 23.17% (by volume) of its particles were less than 25 µm in size, while the DEM solver generated 24.90 (± 0.77) %, using the parameters in **Table 3.2**. 42.53% of the particles were observed to have a size range of 25 to 50 µm, against which DEM generated 41.08 (± 2.62) %. This range carried the largest volume of particles. 21.87% of the powders were 50 to 75 µm in size and 29.99 (± 2.26) % in the DEM-

domain. 8.56% of the particles were sized 75 to 100 µm, and DEM generated 4.03 (± 1.81) %. Only 3.87% of the particles were larger than 100 µm. However, this range was ignored in the DEM solver.

The SP-2 (**Figure 3.2**) was composed of 70% SP-1 and 30% SP-5 (**Table 3.1**), with the resulting mean size of 58.40 µm. In this sample, 21.18% of the particles were under 25 µm, while DEM created 16.85 (± 0.63) %. The largest volume of particles, i.e., 33.25%, were in the range 25 to 50 µm, against 27.79 (± 3.01) % in DEM. It was followed by 50 to 75 µm having 19.09% of the particles, compared to 23.35 (± 3.25) %. 75 to 100 µm contained 10.78%, while the DEM counterpart carried 16.13 (± 1.89) %. 14.31% volume of the particles was sized over 100 µm, and they were 15.88 (± 4.89) % in DEM.

Even quantities of 50% each of SP-1 and SP-5 (**Table 3.1**) were mixed to form SP-3 powders (**Figure 3.3**). It led to the overall average particle size of 77.97 µm. In this mixture, 13.28% of the particles were less than 25 µm in size and 10.93 (± 0.36) % in DEM simulations. 24.88% were 25 to 50 µm, against 21.04 (± 1.88) % in DEM. 50 to 75 µm sized particles formed 18.54% of the volume, while 18.30 (± 3.43) % in DEM. Particles of 75 to 100 µm carried 13.28% of the volume, in contrast to 21.53 (± 0.95) % in DEM. 29.22% volume of the particles was of a diameter larger than 100 µm, compared to 28.19 (± 6.24) %.

The SP-4 (**Figure 3.4**) dominantly contained 70% SP-5 against 30% SP-1 (**Table 3.1**) and an average size of 97.74 µm. Its mixture consisted of only 7.86% of the powder under 25 µm, in contrast to 6.40 (± 0.22) % in DEM. 25 to 50 µm formed 12.55% of the volume, versus 16.52 (± 0.85) % from DEM. 15.99% of particles by volume consisted of 50 to 75 µm size, in comparison to 18.49 (± 3.03) % in DEM, while 75 to 100 µm were 19.12%, compared to 13.13 (± 3.41) % in DEM. As SP-5 was the coarse powder, a significant volume of the particles was

larger than 100 µm. 19.12% were situated in the range 100 to 125 µm, in contrast to 14.13 (± 1.42) % in the DEM domain. 8.71% of the particles were in the range 125 to 150 µm, against 12.41 (± 4.43) % in DEM, whereas 18.80% were still greater than 150 µm, and 18.91 (± 5.37) % generated by the DEM solver.

SP-5 powder (**Figure 3.5**) particles had a mean size of 97.11 µm, being the coarser option amongst the two main categories of powders, i.e., SP-1 and SP-5, given in **Table 3.1**. Here only a very small volumetric percentage of 6.40% particles were less than 25 µm, and DEM generated marginally at 7.88 (± 0.14) %. 25 to 50 µm range consisted of 9.07 % of the particles, in contrast to 13.03 (± 0.70) % generated by the DEM. Here, 39.02% of the particles were in the 50 - 100 µm, while DEM generated 43.56 (± 2.93) %. Nearly half of the particles by volume (i.e., 45.51%) were larger than 100 µm, and 35.53 (± 4.99) %.

Furthermore, it can be observed that the mean size of the particles in the SP-5 category was comparable to SP-4. It might seem counterintuitive initially, as a 30% mixture of the (smaller) SP-1 powders in the SP-4 would be expected to bring the mean down compared to SP-5. However, it would be necessary to reiterate at this point that both SP-1 and SP-5 powders have a ranged distribution of sizes, in contrast to being fixed. In addition, the 30% SP-1 powders in SP-4 also contained the particles on the higher end of the spectrum (i.e., > 100 µm), so their larger size could have made its contribution negligible in reducing the mean size for SP-4 powders against the dominant SP-5 powders.

DEM generation of the radically larger particles (usually ≥ 200 µm) was ignored, even when they were in a smaller quantity. It was done because they could inhibit the local flow of the powder, which could potentially have a more significant impact inside the smaller or limited domain. While within the larger simulation domain or in the actual powder bed, their global

effect would be minimized. On the other hand, smaller-sized particles require a smaller time-step size for a computationally stable solution, as discussed in **Section 2.3.7** (**Page 20**). The solver evaluates the critical time step using the smallest particle in the domain, leading to a higher required time to solve the model. Therefore, generally, the particles below 20 µm were also ignored because they were in a very small percentage in the powder bed. The average particle sizes, both physical and in simulations, at D_{10}, D_{50}, and D_{90} have been listed in **Table 3.3**.

Table 3.3. Summary of the particle sizes for the alumina powders used in this study, with respect to their cumulative percentages of 10%, 50% and 90%.

Powder type	Particle size [µm]					
	D_{10}		D_{50}		D_{90}	
	Physical	**Simulation**	**Physical**	**Simulation**	**Physical**	**Simulation**
SP-1	12	20	38	42	76	75
SP-2	13	20	45	49	110	107
SP-3	18	27	63	79	145	155
SP-4	27	36	88	96	159	133
SP-5	37	20	87	82	151	144

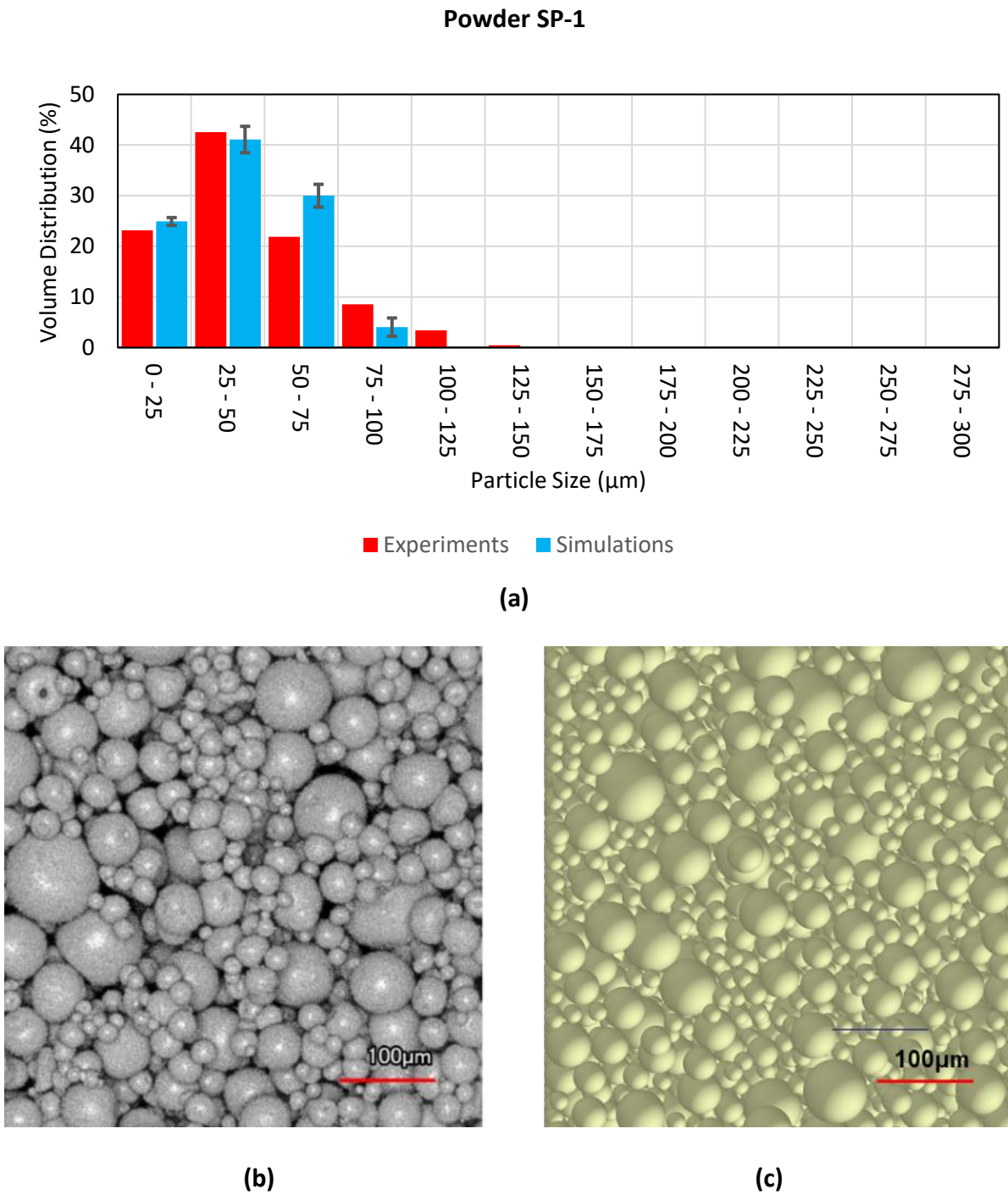


Figure 3.1. Powder SP-1: **(a)** Experimental and simulation generated data comparison of volume distribution, **(b)** optical laser image visualization in contrast to **(c)** DEM.

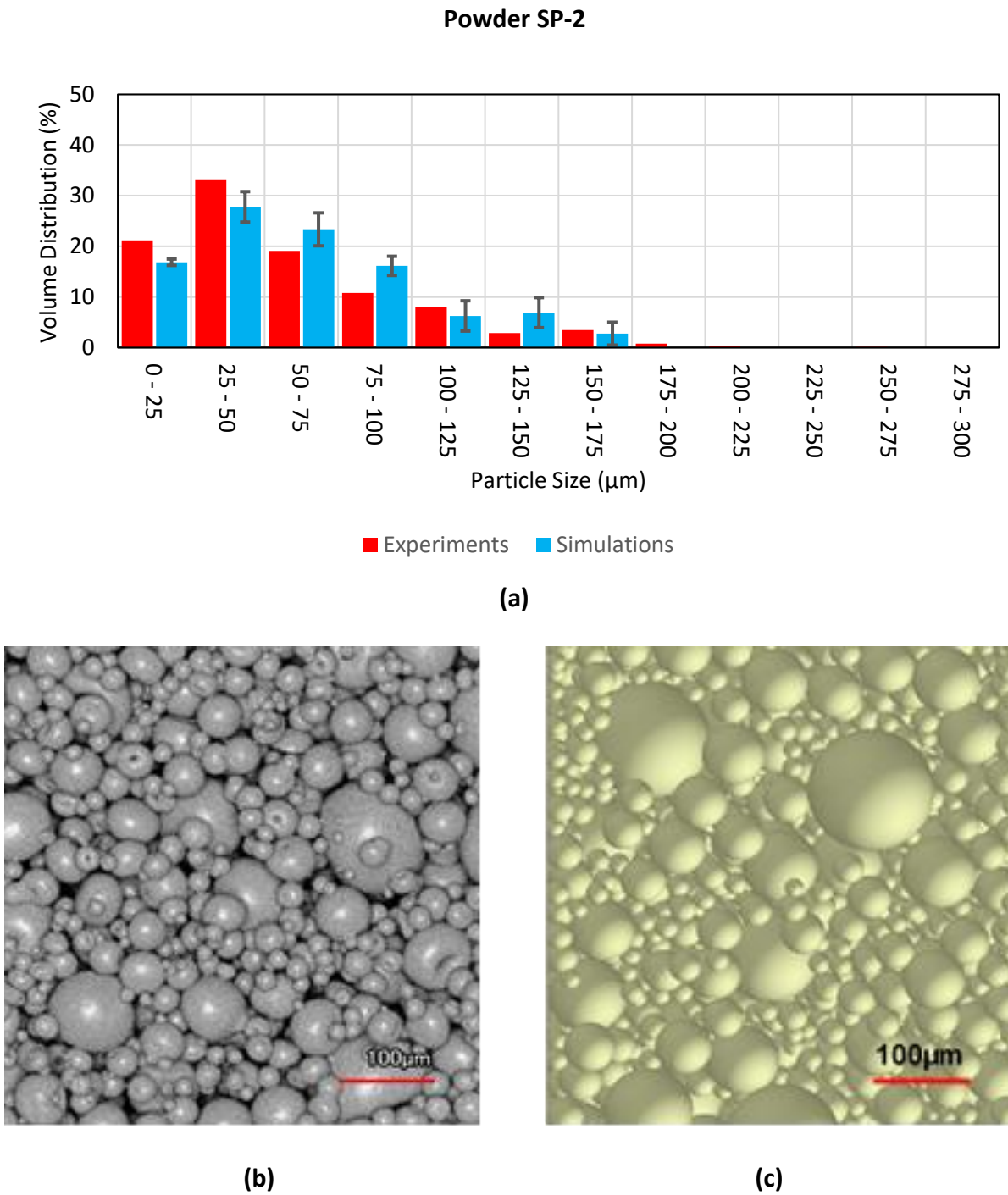


Figure 3.2. Powder SP-2: **(a)** Experimental and simulation generated data comparison of volume distribution, **(b)** optical laser image visualization in contrast to **(c)** DEM.

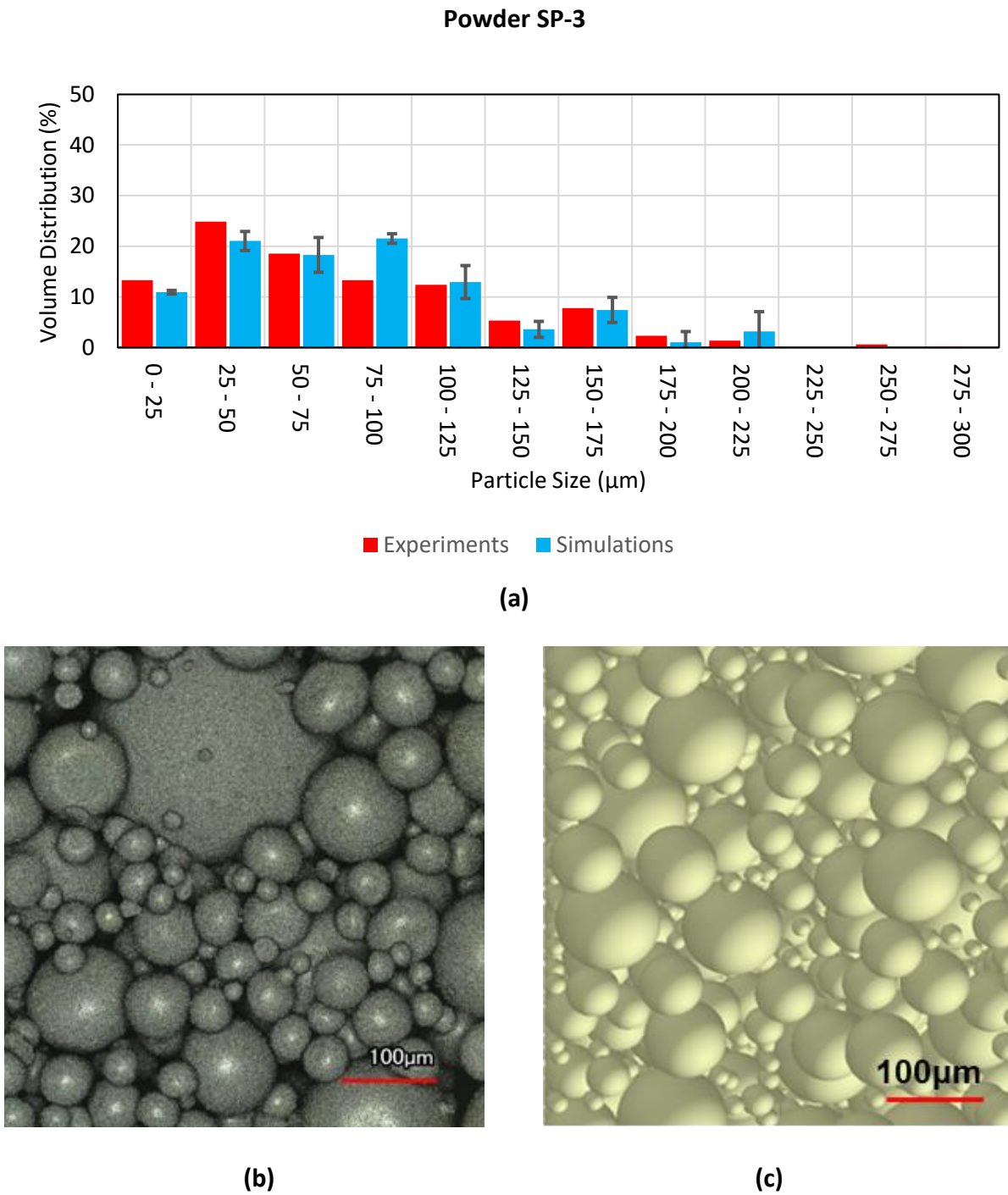
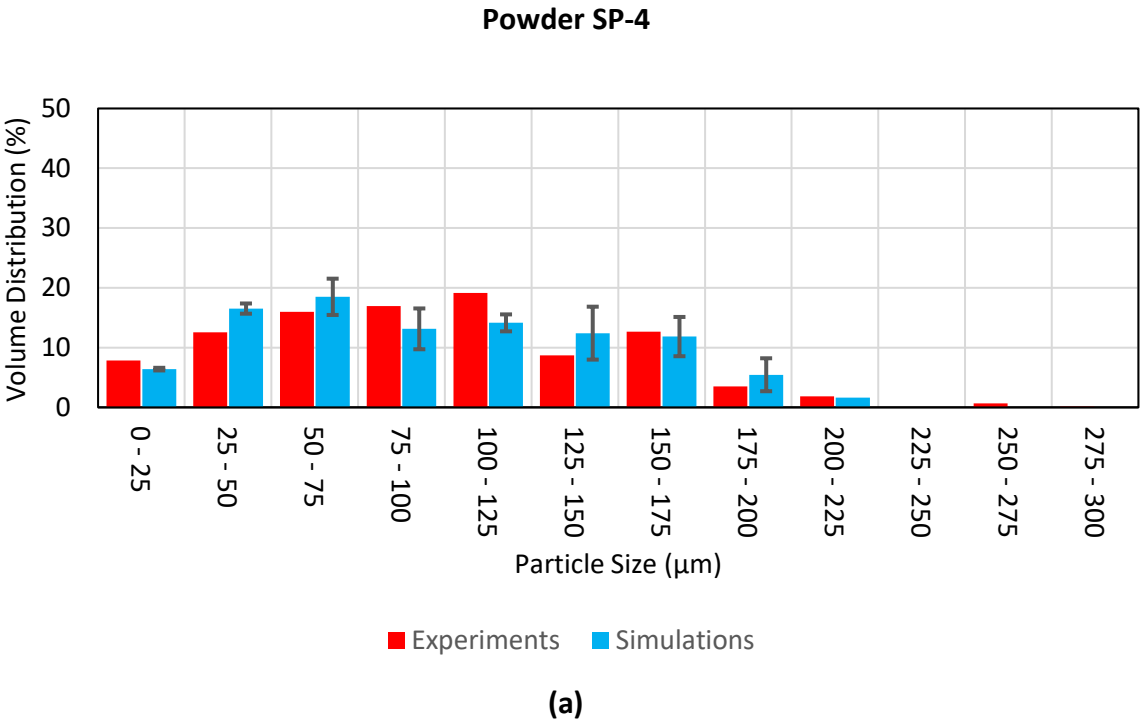
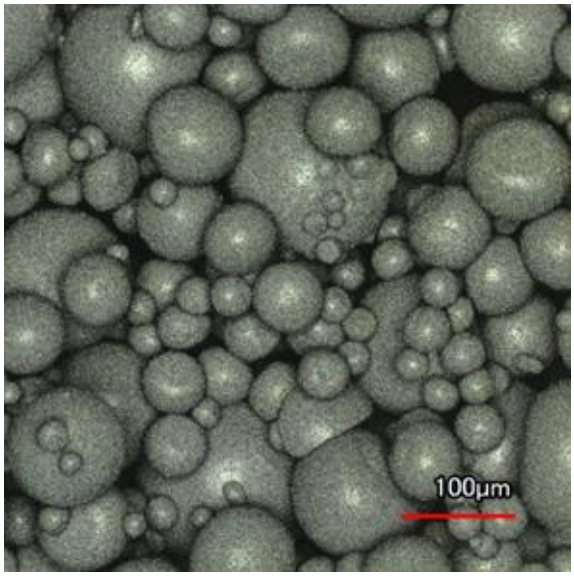


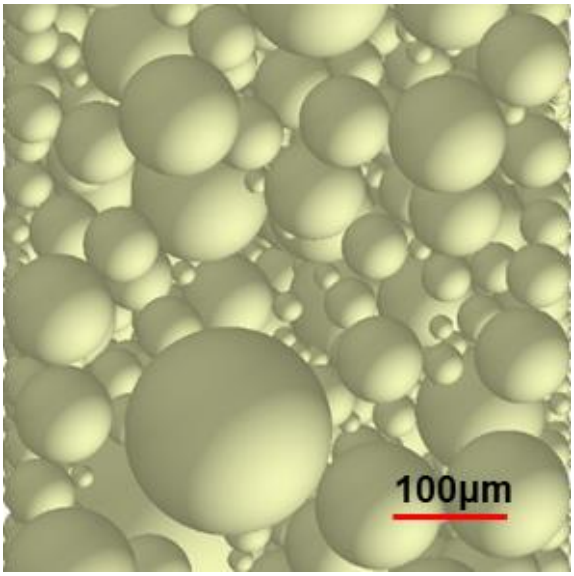
Figure 3.3. Powder SP-3: **(a)** Experimental and simulation generated data comparison of volume distribution, **(b)** optical laser image visualization in contrast to **(c)** DEM.



(a)



(b)



(c)

Figure 3.4. Powder SP-4: **(a)** Experimental and simulation generated data comparison of volume distribution, **(b)** optical laser image visualization in contrast to **(c)** DEM.

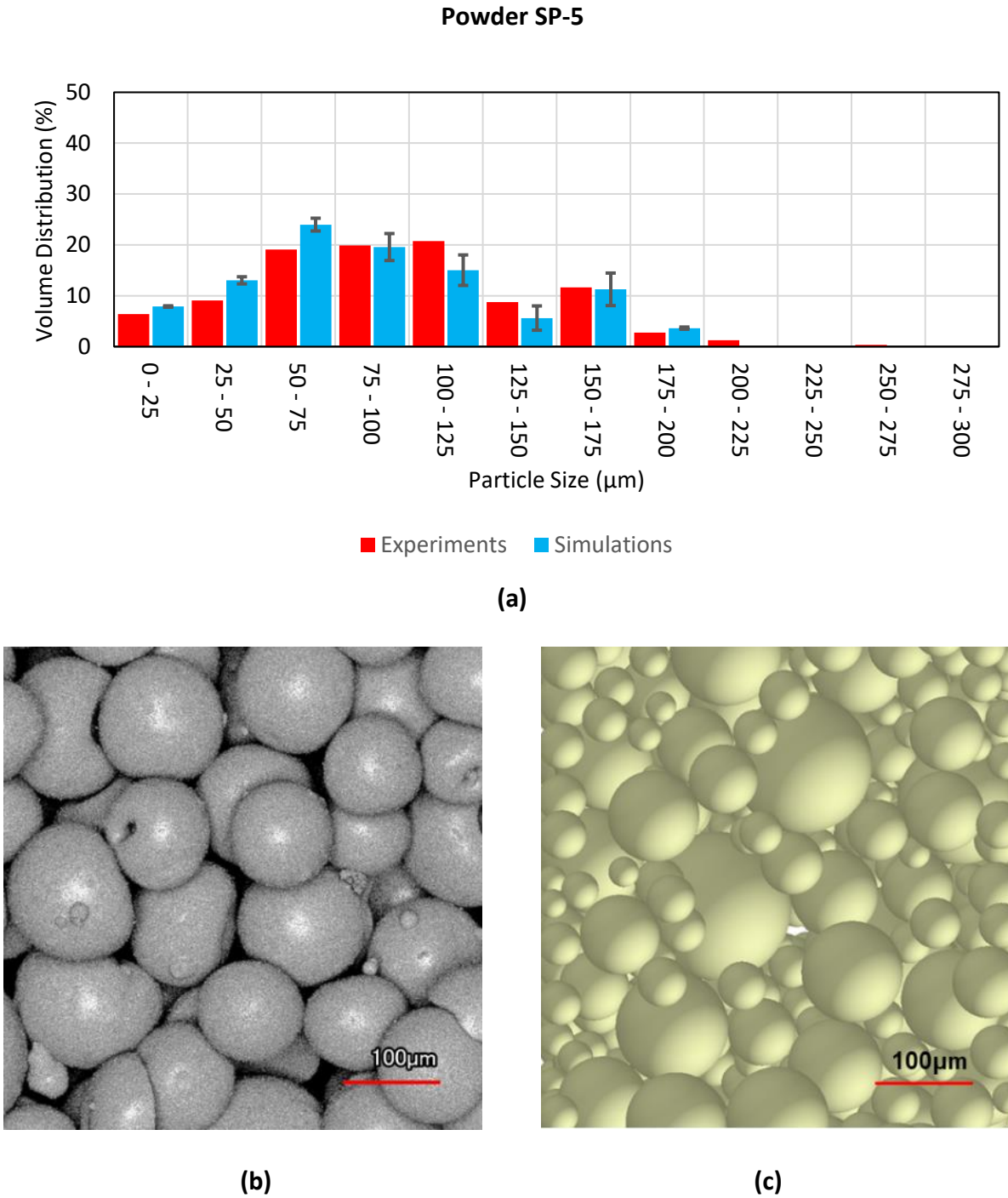


Figure 3.5. Powder SP-5: **(a)** Experimental and simulation generated data comparison of volume distribution, **(b)** optical laser image visualization in contrast to **(c)** DEM.

3.2.2 POWDER CALIBRATION FOR SIMULATIONS

The powders generated in the DEM simulations were calibrated for their flowability against the actual powders used in experiments or pre-production within the LPBF machines. It was done to ensure that during the powder delivery and layer-formation simulations, the DEM particles dynamically behave as close as possible to the real powders, providing precise and accurate insight into the process. The dynamic behaviour of the DEM particles depends on these material properties and interaction coefficients: **(a)** Solid density (ρ), **(b)** Poisson's ratio (ν), **(c)** shear or Young's modulus (E), **(d)** coefficient of restitution (e), **(e)** coefficient of static friction (μ), **(f)** coefficient of rolling friction (μ_R), and **(g)** JKR surface energy density for cohesion (C). The values for **(a)** to **(f)** were adapted from the literature, while **(g)** was iteratively varied until a good match with the experiments for the AOR was obtained. The following values, given in **Table 3.4**, for these parameters were used:

Table 3.4. Material properties and interaction coefficients for alumina powders [18], [90].

Powder type	ρ [kg m^{-3}]	ν -	G [Pa]	e -	μ -	μ_R -	C [J m^{-2}]
SP-1				0.5	0.4	0.1	$5.0 \cdot 10^{-4}$
SP-2				0.5	0.4	0.05	$2.5 \cdot 10^{-4}$
SP-3	3820	0.3	$8.8 \cdot 10^{7}$	0.5	0.34	0.05	$2.5 \cdot 10^{-4}$
SP-4				0.5	0.4	0.1	$3.0 \cdot 10^{-4}$
SP-5				0.5	0.4	0.1	$5.0 \cdot 10^{-4}$

The method adopted to calibrate the powders was the static angle of repose (AOR). A 3D domain was set using EDEM GUI. The domain platform was set to 1 mm x 0.5 mm (square) and the height 1.0 mm. It has been illustrated in **Figure 3.6** (**Page 74**), within which a virtual cuboid volume on the top half of the domain was prescribed for bulk generation of powder

particles. Below this volume, two slanted surface barriers were attached opposite to each other, with a small gap between them acting as a slit. This slit acted as a funnel that guided the bulk powders' free-fall flow (due to gravity) towards the base. After this, all the powder flowed through the slit and formed a heap on the base. The angle formed by this heap is known as AOR. This routine was followed for all the powders, i.e., SP-1 to SP-5. **Figure 3.7** (**Page 75**) compares the AORs obtained in the EDEM simulation software at the University of Birmingham (UoB) against the experimental measurements (images courtesy of Dr Qirong Chen) from the Belgian Ceramic Research Centre (BCRC).

Furthermore, it would be important to consider that we were working on a smaller scale simulations (both for powder calibration and powder delivery process) compared to experiments, due to EDEM's processing limitation that could only utilize the local computational resources (i.e., personal computer), rather than high performance computing (HPC) facilities. Therefore, under a realistic timeframe it was not possible to simulate hundreds of thousand particles to calibrate the powder. The primary objective of the DEM simulations is to capture the overall representative behaviour of the particles. The proper calibration of powder particles even in a small scale against the experimental data establishes the confidence in simulations to replicate the comparable dynamic response of the particles. Nevertheless, similar observations can be made in the works of Zhang et al. [18], Lupo et al. [42], and Geer et al [91].

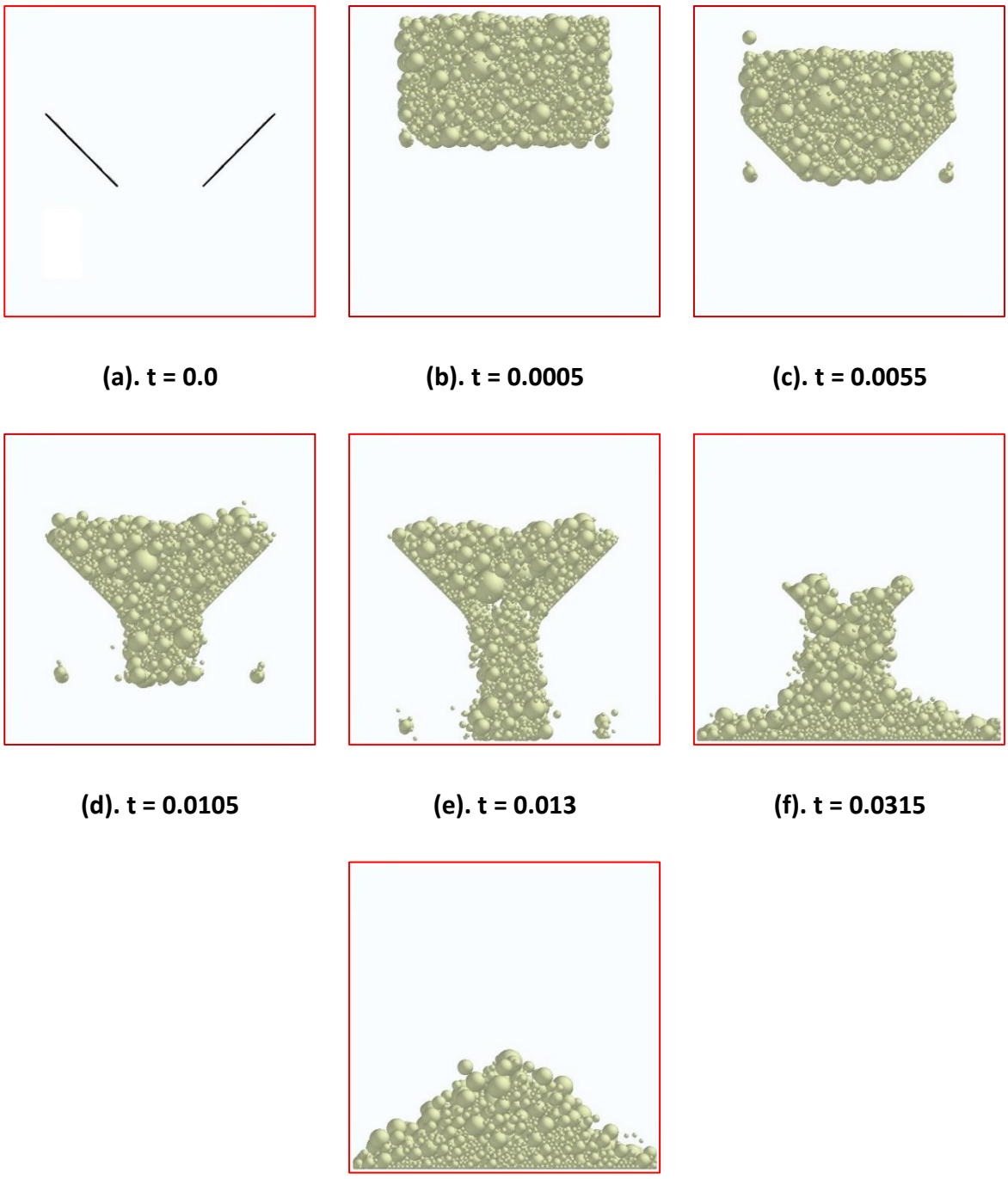


Figure 3.6. Temporal evolution of bulk angle of repose test for alumina SP-3 powders (as an example for general flow) in DEM domain: **(a)** Initial domain (height cropped from 1.5 mm to 1.0 mm for illustration); **(b)** Powders generation in the prescribed volume of the domain; **(c)** Powders settling and adapting to the shape of slanted slits; **(d)** Powders start to escape (flow out) through the narrow slit; **(e)** Powders first hitting the base, i.e., lower boundary of the domain; **(f)** Formation of initial heap on the base, while remaining powders continue to fall on top; **(g)** Final heap formation, used for measurement of angle of repose.

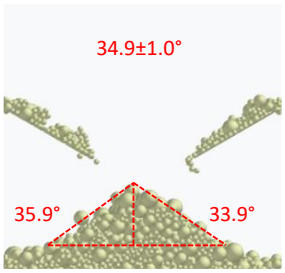
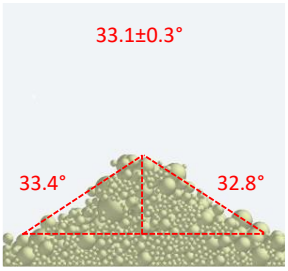
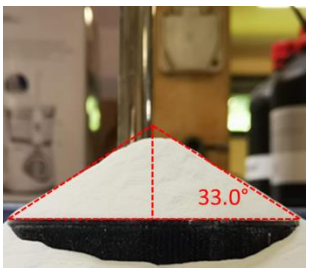
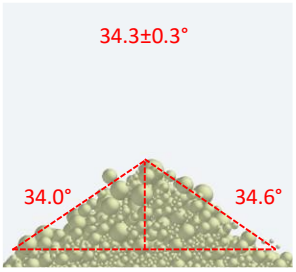
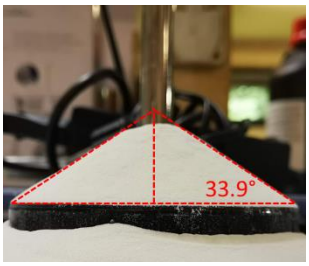
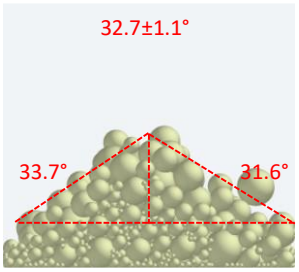
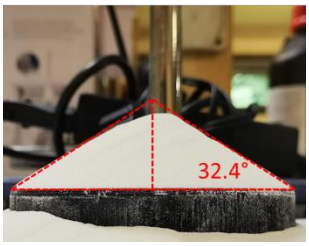
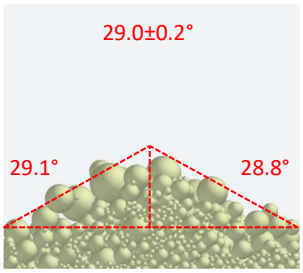
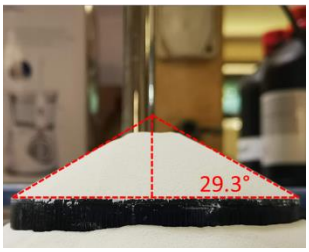
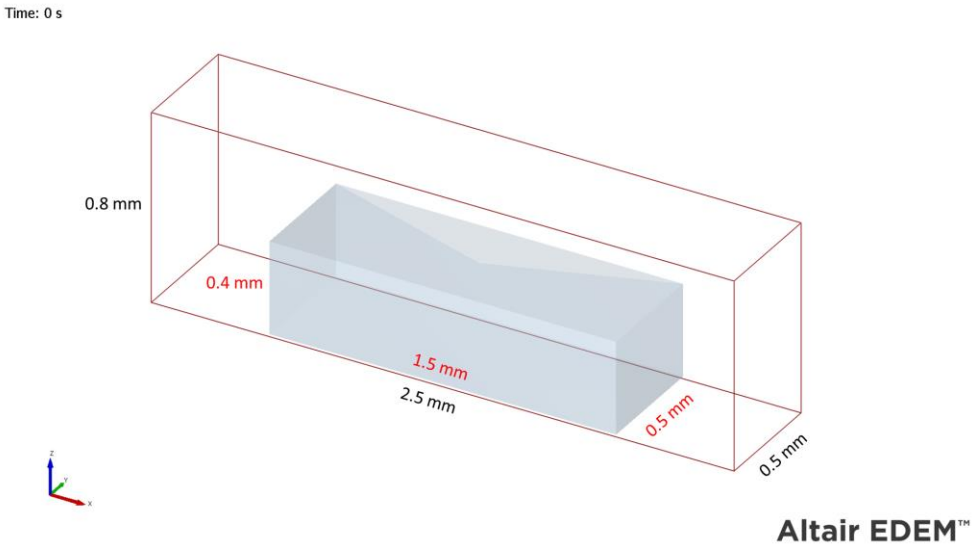
Powder	Simulations (University of Birmingham)	Experiments (Belgian Ceramic Research Centre)
(a). SP-1		
(b). SP-2		
(c). SP-3		
(d). SP-4		
(e). SP-5		

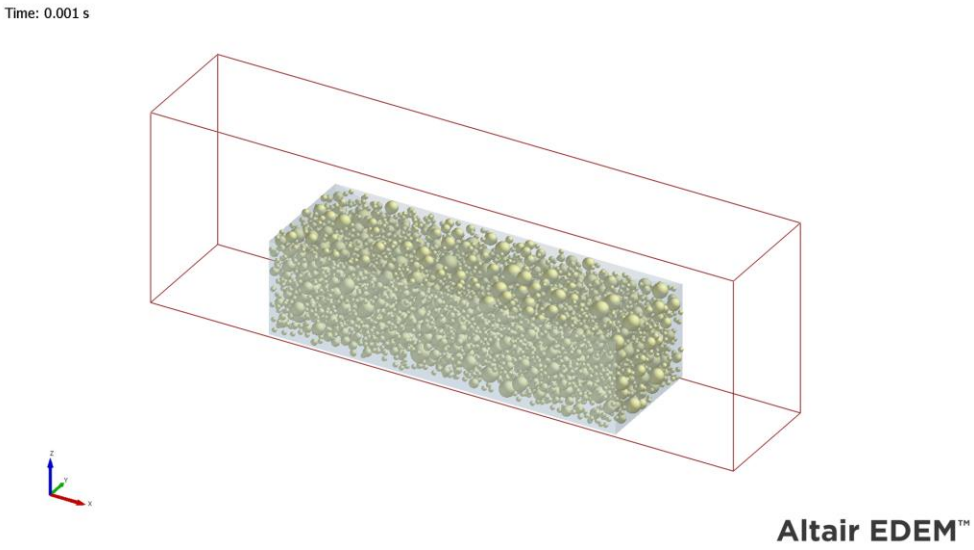
Figure 3.7. Comparison of simulated and experimented static angle of repose used for calibration of alumina powders **(a)** SP-1, **(b)** SP-2, **(c)** SP-3, **(d)** SP-4, and **(e)** SP-5. Images from the AOR experiments on the right side are courtesy of Dr Qirong Chen (Belgian Ceramic Research Centre, Belgium).

3.2.3 SIMULATION SETUP, PARAMETERS, AND PROCESS

The simulation setup consisted of a volumetric domain of 2.5 mm × 0.5 mm × 0.8 mm (**Figure 3.8-a**). The dynamic interaction of the particles could only take place within it. Any particle that left the domain was removed. A volumetric region for particles generation (**Figure 3.8**) was placed inside it, of dimensions 1.5 mm × 0.5 mm × 0.4 mm (**Figure 3.8-a**).



(a)



(b)

Figure 3.8. Illustration of the simulation domain within EDEM graphical user interface: **(a)** Dimensions of the simulation domain and particles generator volume (light grey) inside; **(b)** Particles being generated inside the virtual box/volume.

Millions of particles are present in the actual powder bed, and the boundary walls of the container could only have a minimal effect on the flowability of the powders, especially near the central region (**Figure 3.9**). However, due to the limitation of available computational power, a maximum of 40000 were considered within a very limited domain (**Figure 3.8-a**). As the blade pushes the particles forward, the shear and compressive effect of the boundary walls (perpendicular to the direction of motion) can inhibit the flow (**Figure 3.9**).

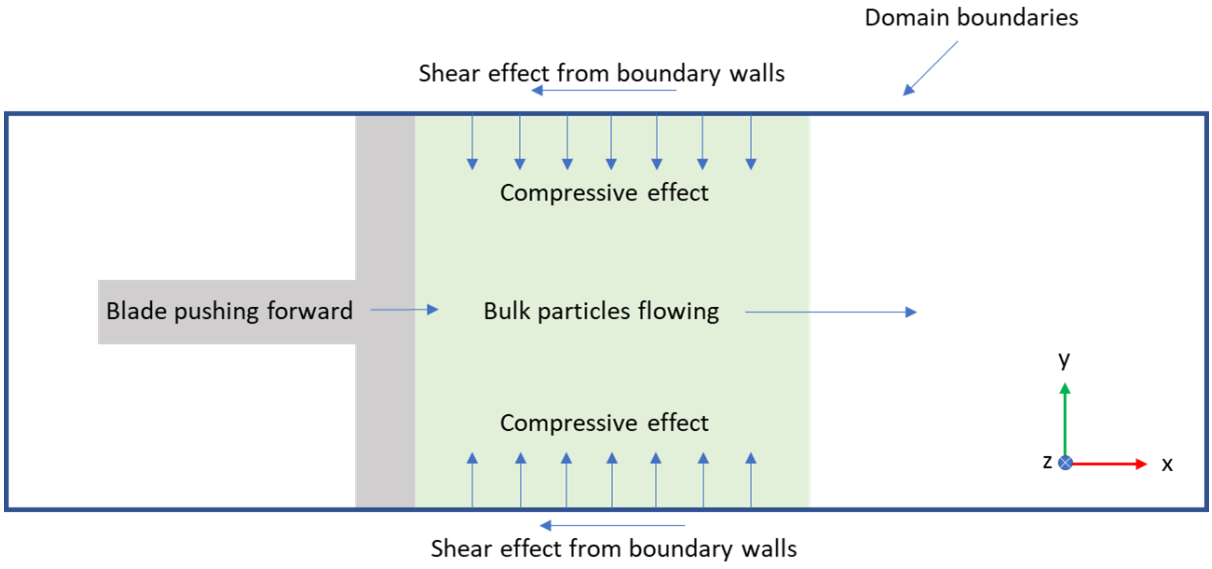


Figure 3.9. Illustration of bulk powder being pushed forward, and generic forces are coming into play.

Therefore, the periodic boundary conditions were applied in the y-axis (i.e., XZ plane). It implied that any particle that left one of the XZ planar boundaries would appear on the other boundary with the same velocity and movement direction, as illustrated in **Figure 3.10** (next page). These boundary conditions were chosen to approximate an infinitely large system using a limited volume or area called a unit cell. In our case, the domain walls on the YZ plane (perpendicular to the X direction) were treated as open. Any particle leaving them was eliminated. The periodic boundary here would have caused the bulk to be taken away to

appear on the other side, which would be an unrealistic approach. In contrast, the close boundary would cause heap formation on top of the smoothed layers towards the edge.

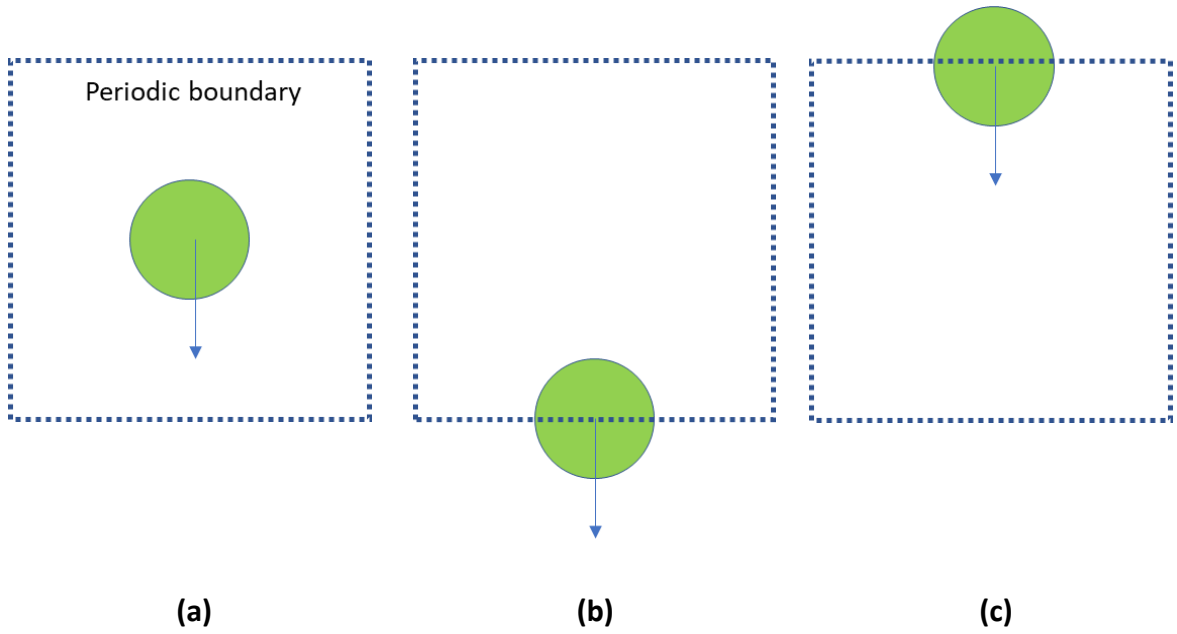


Figure 3.10. Unit cell/domain for a periodic boundary condition: **(a)** Particle travelling in a downward direction towards the base, **(b)** exiting the cell, and then **(c)** appearing from the opposite side with the same travel direction and velocity.

The simulation setup also consisted of a blade, as illustrated in **Figure 3.11**. The blade consisted of a circular profile at the bottom (not to be mistaken with the roller), which interacted with the particles and pushed them forward. The starting point of the blade was offset from the left edge of the substrate to allow clearance for powder generation and deposition. The blade was then set to move with the translational motion, horizontally - left to right, to compress and even out the surface of the powder, which would form the 'process layer'. At the end of each run, the blade was lifted equivalent to the desired layer thickness and returned to its original (start) position in the x-direction.

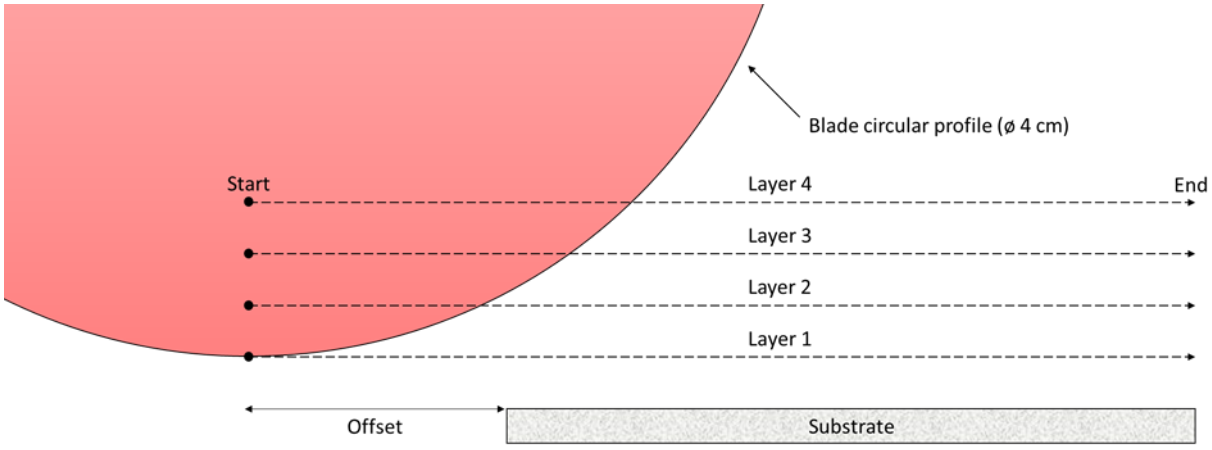


Figure 3.11. General illustration (not to scale) for the path followed by the blade.

The particle generator box (**Figure 3.8-b, Page 76**) was lifted by the same height as the layer thickness after each blade run. This step allowed the powders to be deposited on top of the previous layer and prevented the generation of particles within the gaps or pores of existing layers. This process was repeated for up to 4 layers, where consistent packing densities started to be observed (discussed in **Section 3.2.4**). The simulation parameters are listed in **Table 3.5** below. The general simulation output of the layer formation process (SP-1 powders in this case) can be visualized in **Figure 3.12**. The images show the temporal evolution of only the first and last (fourth) layer formation, as the second and third followed the same pattern. The rest of the powders (SP-2 to SP-5) were similarly simulated.

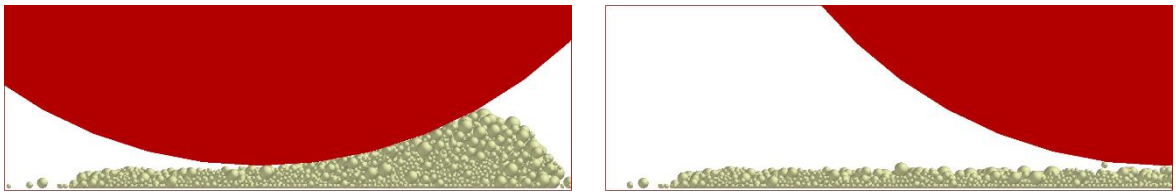
Table 3.5. Simulation parameters for spreading of alumina powders. This data was provided by BCRC as adapted for their experiments.

Recoater / Spreader	Profile	Size [mm]	Speed [mm s^{-1}]	Layer thickness [µm]	Total layers
Blade	Circular	4	50	100, 150, 200	4

Layer 1 deposition and smoothing process



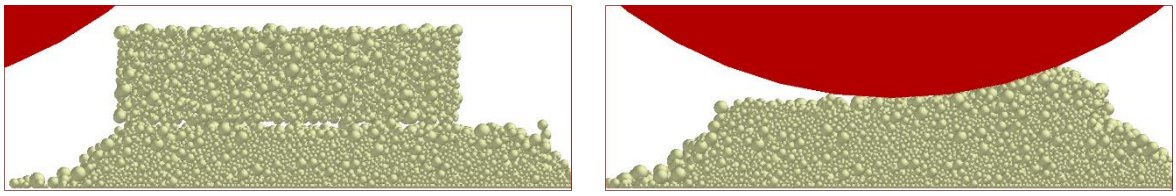
(a). t = 0.0010 s **(b).** t = 0.0125 s



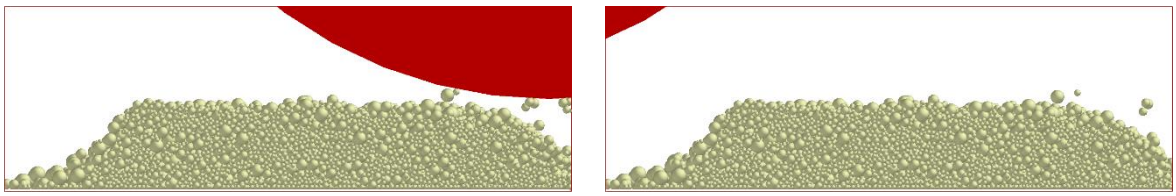
(c). t = 0.0475 s **(d).** t = 0.0745 s

⋮

Layer 4 deposition and smoothing process



(e). t = 0.2085 s **(f).** t = 0.2485 s



(g). t = 0.271 s **(h).** t = 0.273 s

Figure 3.12. Temporal evolution of layer formation process for alumina SP-1 powders (as a general example) in DEM domain: **(a)** At t = 0.001, powder for the first layer is generated **(b)** At t = 0.0125, the generated powder settles down and the blade starts to move; **(c)** Layer smoothing in process; **(d)** Completion of first layer formation process. The same steps are followed for rest of the layers, for instance: **(e)** Blade is now raised to the thickness of fourth layer, and at the same time powder for the fourth layer is deposited on top of third layer; **(f)** Fourth layer smoothing in process; **(g)** Completion of fourth layer formation process; **(h)** Blade reset to its original horizontal position.

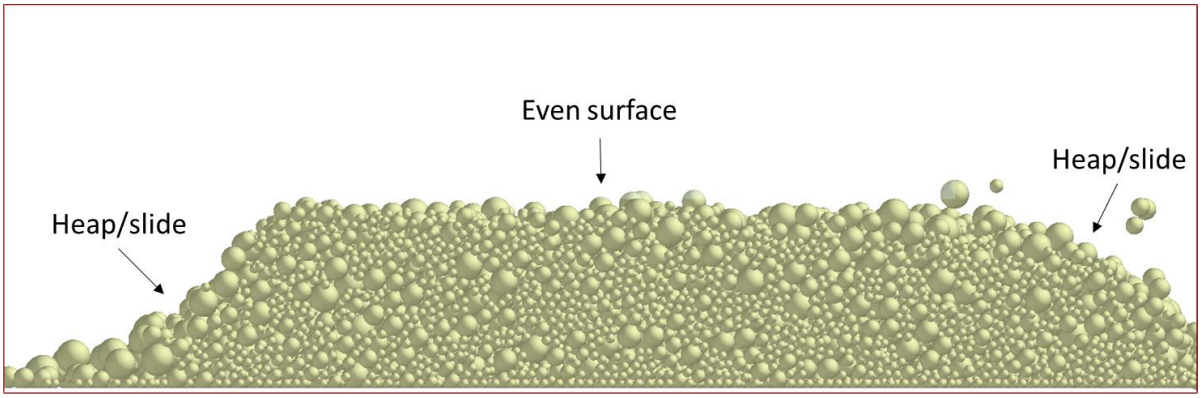
Furthermore, the critical Rayleigh timestep size (**Section 2.3.7, Page 20)** determined by the EDEM solver and our defined timestep have been listed in **Table 3.7**. Our chosen timestep was kept within 25% of the critical timestep.

Table 3.6. Simulation parameters for spreading of alumina powders. This data was provided by BCRC as adapted for their experiments.

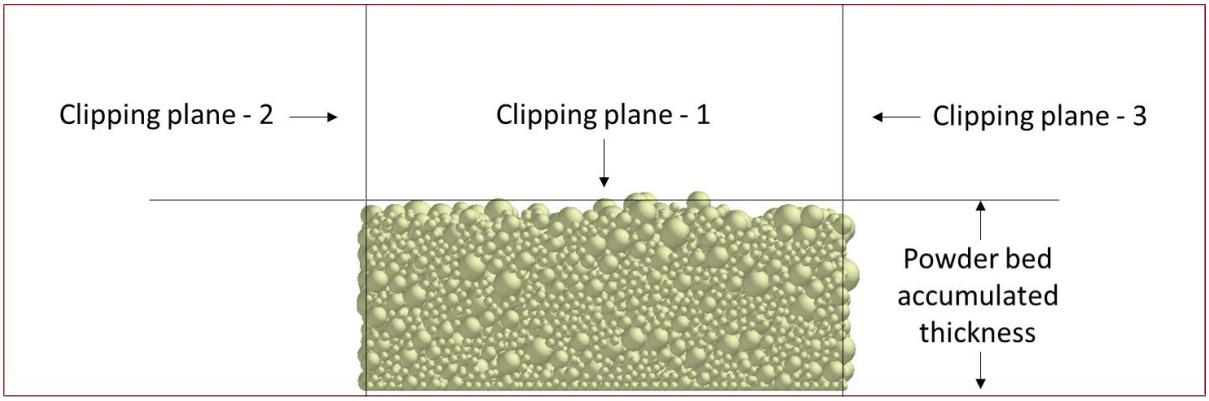
Powder	Critical Rayleigh timestep [s]	Defined timestep [s]	Percentage of Rayleigh Time Step
SP-1	$2.2364 \cdot 10^{-7}$	$5 \cdot 10^{-8}$	22.4%
SP-2	$8.6580 \cdot 10^{-8}$	$2 \cdot 10^{-8}$	23.1%
SP-3	$2.1805 \cdot 10^{-7}$	$5 \cdot 10^{-8}$	22.9%
SP-4	$2.1805 \cdot 10^{-7}$	$5 \cdot 10^{-8}$	22.9%
SP-5	$3.2204 \cdot 10^{-7}$	$5 \cdot 10^{-8}$	15.5%

3.2.4 POWDER DENSITY EVALUATION METHOD AND RESULTS

The region containing the heap (or slides) (**Figure 3.13-a**) was clipped out to evaluate the final packing density (PD) of the powder bed. It left the unclipped region containing powder particles with even surfaces from all sides (**Figure 3.13-b**).



(a)



(b)

Figure 3.13. Illustration of the final powder bed (i.e., post blade run): **(a)** Complete powder bed with heaps near the walls of the domain; **(b)** Clipped powder bed region considered for calculating the density. The clipping planes appear as 'lines' because the figure projects the side view.

After clipping, the data for the number of particles and their corresponding diameters were extracted. In **Figure 3.13-b**, the 'powder bed accumulated thickness' referred to the combined thickness of the current and previous layers. The clipping plane 1 was initially set perpendicularly above the base, with the displacement of one layer thickness. After post-

processing the first layer, this plane was raised by the thickness size for each subsequent layer. In this way, the change in powder bed density as a result of further layer addition could be analysed. The powder bed density (PD), in this case refers to the solid fraction of the unclipped region, and this was calculated as follows:

$$PD = \frac{\sum_{i=0}^{N} \frac{4}{3}\pi R_i^3}{V_{UCR}} \tag{3.1}$$

where N is the total number of particles in the clipped region, R_i is the radius of the i-th particle, and V_{UCR} is the volume of the unclipped region, i.e., bounded by the clipping planes (**Figure 3.13-b**). It would be important to note that the particles situated closer to the clipping plane, with their centres inside the unclipped region but part of their volume protruding outside, were still fully included. It led to a slight unaccounted overestimation of the exact PD.

On the other hand, all the particles have a 1 – 2% overlap, which is used to calculate the Hertz-Mindlin force between them (as discussed in **Section 2.3.5**). These overlaps apparently make the DEM powder bed slightly denser. It was also ignored in the PD calculation. However, to balance this assumption, the real particles would also be compressed to a certain degree due to their weight and the compressive effect of neighbouring particles.

As stated in **Table 3.5** (**Page 79**), the blade was used as a recoater and powder spreader, with a 4 mm diameter circular profile interacting with particles. It was set to move at a fixed speed of 50 mm s^{-1}. For each powder, i.e., SP-1 to SP-5, 3 different layer thicknesses of 100 µm, 150 µm, and 200 µm were considered for up to a total number of 4 layers. The selected layer thickness was set constant for each simulation. The obtained results are described from the next page.

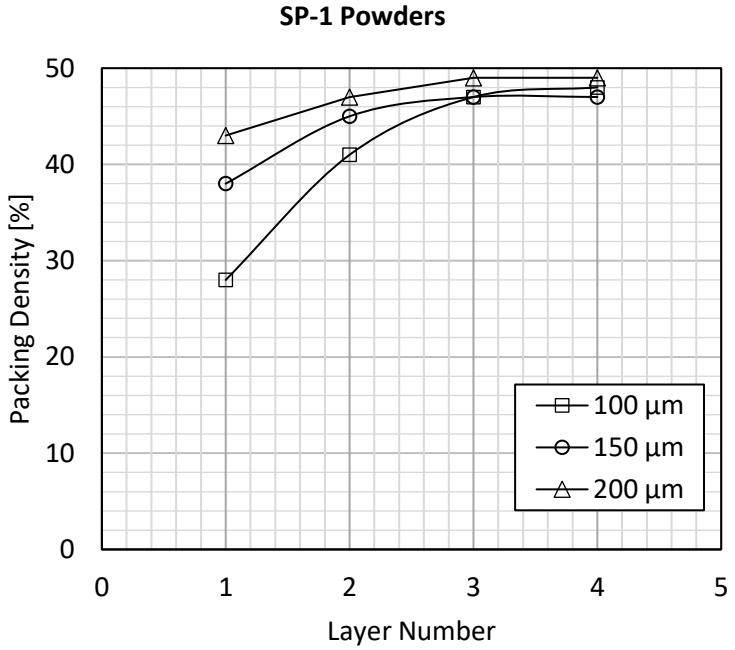


Figure 3.14. Change in packing density of **alumina powder SP-1** w.r.t. the addition of layers for different layer thicknesses of 100 µm, 150 µm, and 200 µm.

For the powder SP-1, it can be observed in **Figure 3.14** above that when we spread a 100 µm size layer, the first layer formed a sparse 28% packing (or solid) density, followed by an increase to 41% after the second layer, 47% after the third layer, and a final density of 48% with the fourth layer. In contrast, when the layer size was increased to 150 µm, the PD after the first layer improved to 38%, 45% after the second layer, and increased up to 47% by the third layer. However, no further change was observed with the fourth layer. Comparatively, the 200 µm layer size presented a better 43% PD from the very first layer. It was followed by a minor increment to 47% by the end of the second layer processing and settled at 49% by the third layer. Again, there was no further change with the addition of the fourth layer. Overall, the final PD that could be achieved for SP-1 powders was 48 (±1)%, with the given blade parameters, regardless of layer size.

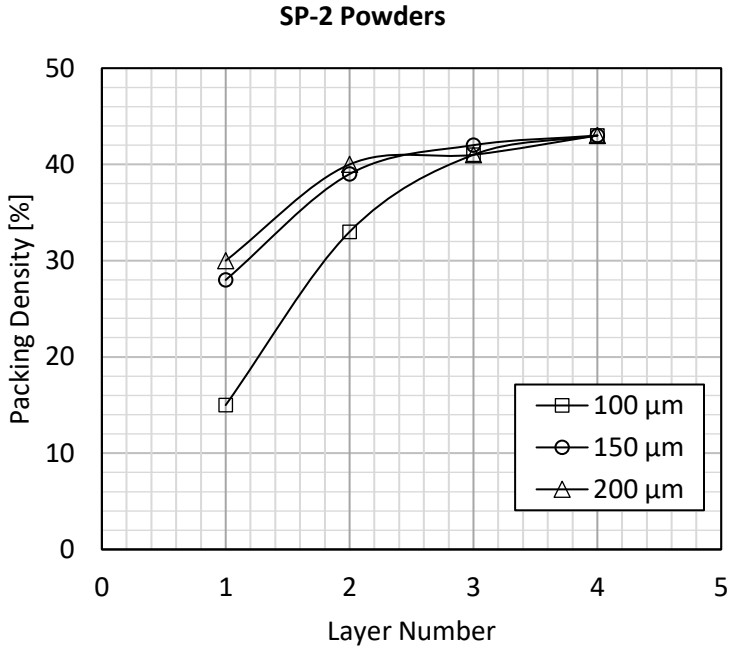


Figure 3.15. Change in packing density of **alumina powder SP-2** w.r.t. the addition of layers for different layer thicknesses of 100 µm, 150 µm, and 200 µm.

We followed the same pattern for the SP-2 (**Figure 3.15** above) and all the subsequent powders. The first layer of 100 µm size for SP-2 turned out to be sparser when compared to SP-1, forming 15% PD only. However, this improved to 33% with the addition of the second layer, followed by 41% with the third layer. The final PD reached 43% after the processing of the last layer. When the layer size was increased to 150 µm, the first layer formed 28% PD, followed by the second layer at 39%, and then the third layer at 42%. The final PD was 43% after the smoothing of the fourth layer. With a 200 µm layer size, the first layer started with 30% PD, which jumped to 40% and 41% after the second and third layers, respectively. The final PD was again observed to be 43% after the fourth layer.

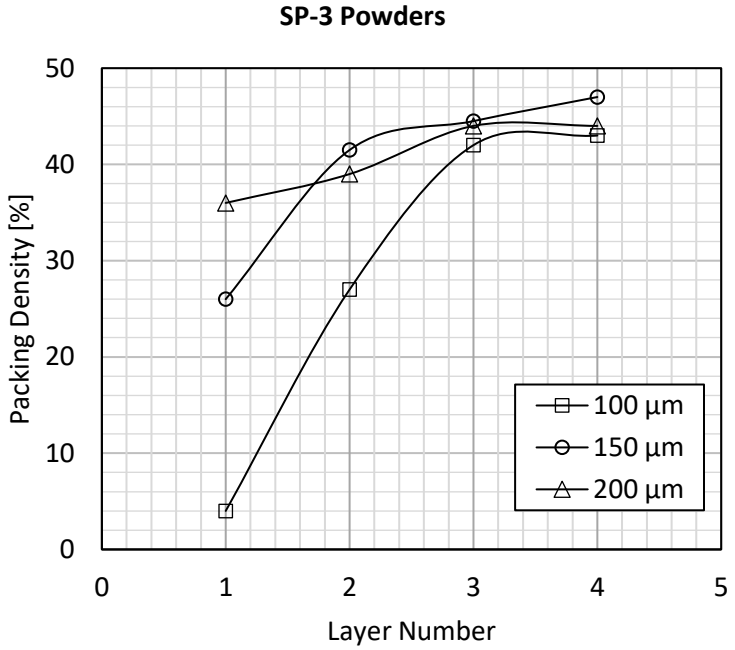


Figure 3.16. Change in packing density of **alumina powder SP-3** w.r.t. the addition of layers for different layer thicknesses of 100 µm, 150 µm, and 200 µm.

In the case of SP-3 powders (**Figure 3.16** above), the first layer of 100 µm size formed only 4% PD. Most of the particles were brushed off. The PD increased to 27% after processing the second layer, followed by another jump to 42% with the third layer. The final PD was 43% after the accumulation of fourth-layer particles. After the layer size was increased to 150 µm, the first layer started with 26% PD, followed by a climb to 42% PD after the second layer, 45% PD after the third layer, and finally attaining 47% by the end of the fourth layer's processing. The first layer of 200 µm thickness formed 36% PD, followed by 39% at the end of the second layer. The final PD of 44% was attained in this case with the addition of a third layer, and further addition of the fourth layer did not demonstrate any change, staying at 44%.

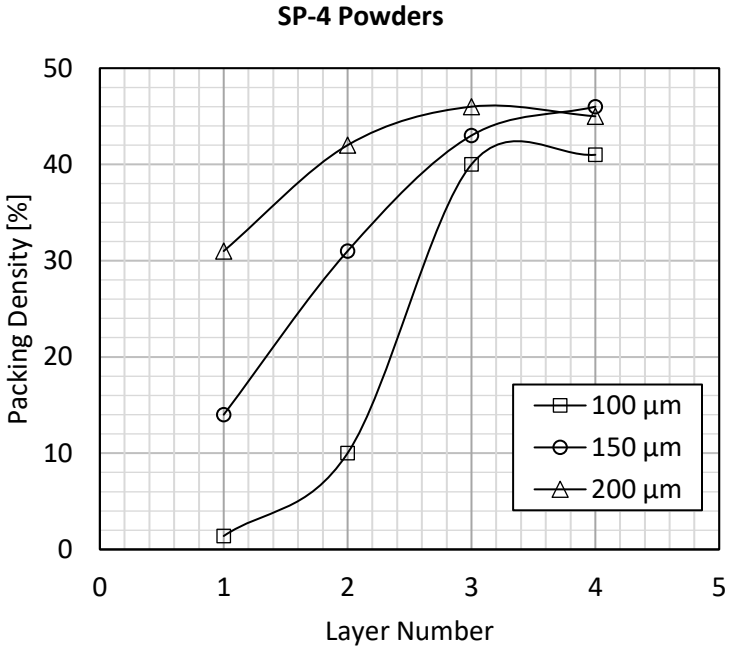


Figure 3.17. Change in packing density of **alumina powder SP-4** w.r.t. the addition of layers for different layer thicknesses of 100 µm, 150 µm, and 200 µm.

The first layer of 100 µm size for the SP-4 powders (**Figure 3.17** above) was almost swept away by the blade (discussed later), leaving only 1% PD. It was followed by only a modest increase to 10% PD after adding and processing the second layer. It was not unexpected, as the base layer had practically disappeared. A sharp increase to 40% PD followed for the third layer and attained 41% after the final layer. The 150 µm size produced 14% PD after the first layer, which jumped to 31% after the second layer, followed by another jump to 43% PD after the third layer. After processing the fourth layer, the final PD was 46% in this case. For the 200 µm layer size, the first layer started by producing 31% PD. The PD increased to 42% with the second layer and peaked at 46% after the third layer. A slight drop to 45% was observed after processing the final (fourth) layer.

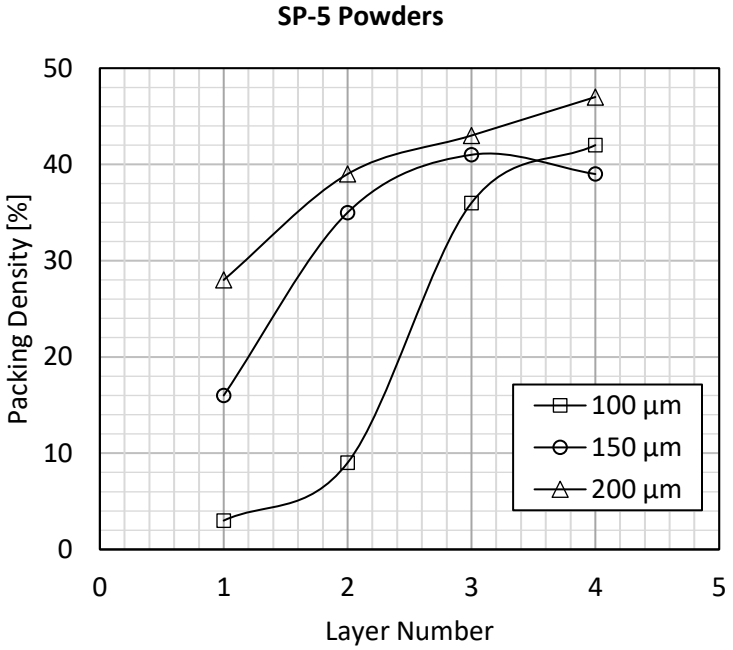


Figure 3.18. Change in packing density of **alumina powder SP-5** w.r.t. the addition of layers for different layer thicknesses of 100 µm, 150 µm, and 200 µm.

Similar to the (previous) SP-4 case for the first layer of 100 µm size, only 3% of the powder was left by the blade, followed by an insignificant increase to 9% after the second layer. The PD steeply jumped to 36% with the third layer, and the final layer achieved 42% PD. The first layer of 150 µm size started at 16%, which was better than 100 µm, but still very sparse. After the second layer, PD jumped to 35%, followed by 41% PD after the third layer. A slight decrease in PD to 39% was observed with the final layer. The first layer of 200 µm produced 28% PD, followed by 39% after the second layer. PD went up to 43% after processing the third layer and finally reached 47% after the final fourth layer.

3.2.5 SURFACE ROUGHNESS EVALUATION USING THE RAY TRACING METHOD

We developed a ray tracing method to evaluate the surface roughness of the powder beds in DEM. A square domain was generated with its side equal to the width of the DEM domain, and it was further discretized into the finite number of square cells, with their location identified by row (i) and column (j) number (**Figure 3.19-a**).

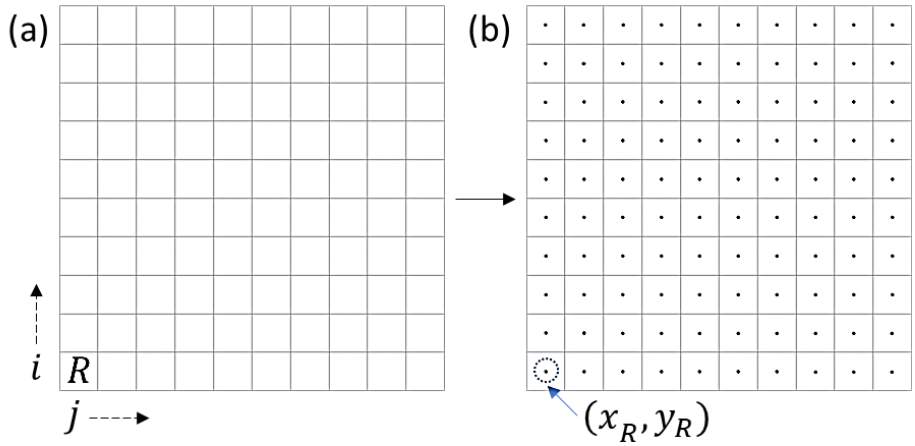


Figure 3.19. (a). Discretization of the square domain into smaller square cells (denoted by integer R); **(b).** Setting up of rays' centre positions (x_R, y_R).

Each of the cells denoted a ray, with a unique integer identifier R (or ray number), and its corresponding centre representing the ray's origin coordinates (x_R, y_R) (**Figure 3.19-b**), were calculated as:

$$x_R = \left[\frac{1}{N_{rows}}\left(i + \frac{1}{2}\right) - \frac{1}{2}\right] L_x + x_0 \tag{3.2}$$

$$y_R = \left[\frac{1}{N_{rows}}\left(j + \frac{1}{2}\right) - \frac{1}{2}\right] L_y + y_0 \tag{3.3}$$

where $\{R \in \mathrm{Z} \mid 0 \leq R < N_{rows}^2\}$ also represents all possible combinations of:

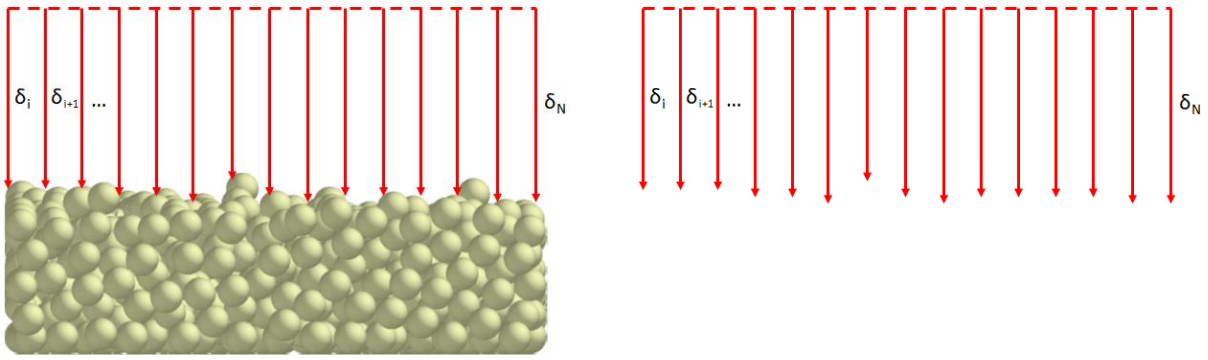
$$i \in \mathrm{Z} \mid 0 \leq i < N_{rows} \tag{3.4}$$

and

$$j \in \mathrm{Z} \mid 0 \leq j < N_{rows} \tag{3.5}$$

to cover each of the cells in the discretized grid (**Figure 3.19**), N_{rows} is the number of cells per row (or column, as we are considering the square region, so both are equal), N_{rows}^2 represents the total number of cells (or rays) initially generated in the square region, L_x and L_y are length of sides of the domain (which are equal in this case), and (x_0, y_0) are the centre coordinates of the domain. The rays are represented by dots '·' at the centre of each cell (**Figure 3.19-b**).

Consider the side view of the powder bed in **Figure 3.20**, where each discretized ray was projected downwards from its origin towards the powder bed surface. The point where they intersected the first particle was used to calculate its total ray-to-particle travel or length, denoted by δ_{RTP}.



(a) **(b)**

Figure 3.20. (a) Depiction of rays falling on the powder bed, up to the point where they intersect the first powder particle; **(b)** Remaining rays with varying lengths illustrating the effect of powder bed surface profile.

To calculate δ for each ray, the mathematical analogue to ray-particle interception, i.e., the line-sphere intersection, was considered, assuming all particles to be spherical (**Figure 3.21**). To develop its algorithm, the data for the particles, including their IDs, locations (i.e., centroid

coordinates), and diameter, were extracted using EDEM's API (Application Programming Interface) and then shortlisted those potentially interfering with the corresponding ray's path.

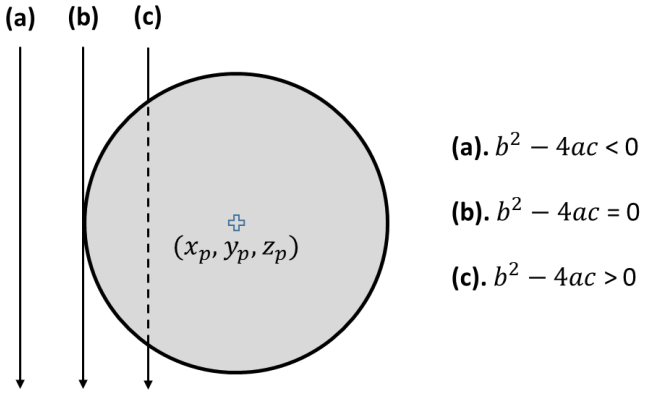


Figure 3.21. Possible ray-particle or line-sphere interactions; **(a).** No interception occurs; **(b).** Tangential contact at a single point; **(c).** Ray path interfering with the particle, with two possible contact points, where only the entry contact point is considered.

The intersection was evaluated by solving for t in the following parametric line **Equation (3.6)**:

$$x = x_R + t\Delta x; y = y_R + t\Delta y;\ z = z_R + t\Delta z \tag{3.6}$$

by substituting their x, y, and z in the sphere **Equation (3.7)**:

$$\left(x - x_p\right)^2 + \left(y - y_p\right)^2 + \left(z - z_p\right)^2 = R_P^2 \tag{3.7}$$

where (x_R, y_R, z_R) are the originating coordinates for the ray, the parameter t represents the scalar magnitude and direction (with respect to particle) of the ray, used to evaluate the coordinates along the line (or ray), $(x_p,\ y_p,\ z_p)$ are centroid coordinates for (spherical) particles, and R_P is the particle radius. The substitution resulted in the following quadratic equation form:

$$at^2 + bt + c = 0 \tag{3.8}$$

In the case of the following condition, the ray does not intersect the considered (**Figure 3.21-a**):

$$b^2 - 4ac < 0 \tag{3.9}$$

while in the condition given below, where the ray only made tangential contact with the particle surface, it was also treated as a missed intersection (**Figure 3.21-b**):

$$b^2 - 4ac = 0 \tag{3.10}$$

The only condition considered for the ray-particle interception (**Figure 3.21-c**) is given as:

$$b^2 - 4ac > 0 \tag{3.11}$$

which results in two solutions (values) for t in the quadratic **Equation (3.8)**, signifying multiple intersection points of the line with the sphere. The smaller t was used to calculate the first intersection point, which was taken as the point where the ray hits the particle. So, this was replaced in the **Equation (3.6)** to obtain the coordinates of the intersection (x_i, y_i, z_i). Finally, the ray-to-particle displacement was calculated as follows:

$$\delta_{RTP} = \sqrt{(x_i - x_R)^2 + (y_i - y_R)^2 + (z_i - z_R)^2} \tag{3.12}$$

Since the ray was projected vertically (i.e., along z-axis) in our case, so the above equation could be simplified to:

$$\delta_{RTP} = |z_i - z_R| \tag{3.13}$$

This process was repeated for all the generated rays in our code, as generally illustrated in **Figure 3.19-b**, to obtain their respective δ_{RTP} (**Figure 3.20-b**). The parameter R_a was used to quantify surface roughness, but it was processed outside of the code. R_a represents the average (absolute) deviation from the mean of obtained data, evaluated as follows:

$$R_a = \frac{1}{N}\sum_{i=0}^{N}\left|\delta_{RTP(i)} - \bar{\delta}\right| \tag{3.14}$$

where N is the total number of rays, $\delta_{RTP(i)}$ is the ray-to-particle displacement for the respective i-th ray, and $\bar{\delta}$ is the mean ray-to-particle displacement for all the rays considered, given by:

$$\bar{\delta} = \frac{1}{N}\sum_{i=0}^{N}\delta_{RTP(i)} \tag{3.15}$$

The discussed process for evaluation of surface roughness has been summarized using a flowchart given in **Figure 3.22** (next page). Following this figure, **Algorithm 3.1** highlights the main steps in developing the corresponding ray tracing C++ code, which can be found in the **Appendix-C** (**Page 229**).

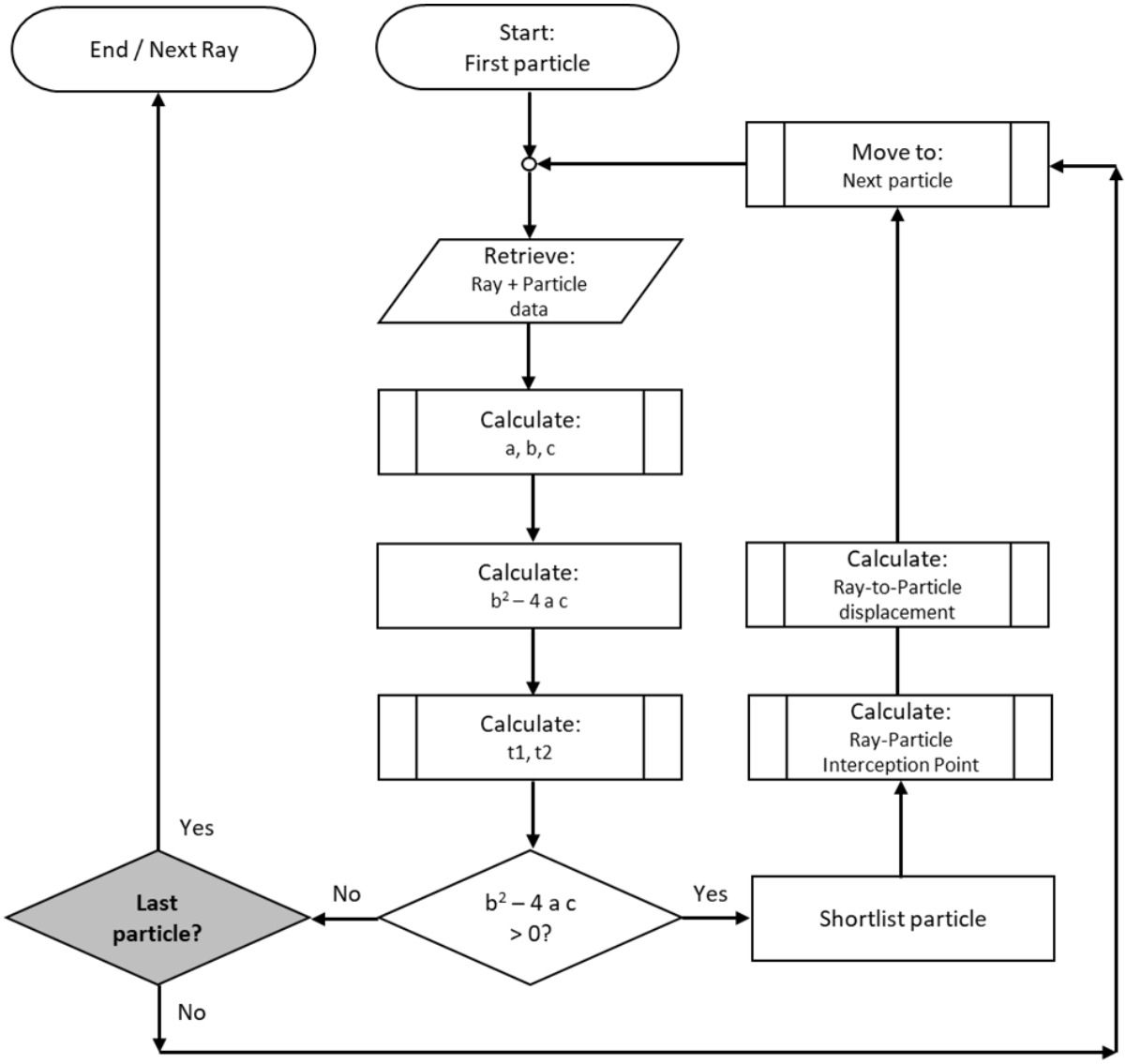


Figure 3.22. Calculation and determination process for ray-particle interception, analogous to line-sphere intersection. This flow chart also represents the expanded/sub process given in **Figure 4.5-a** in the later **Section 4.3.3**.

```
CALCULATE RAY-PARTICLE_INTERCEPTION;

RETRIEVE TOTAL_NUMBER_OF_PARTICLES;
RETRIEVE RAY_ORIGINATING_COORDINATES: (X1, Y1, Z1);
RETRIEVE RAY_DIRECTION (X2-X1, Y2-Y1, Z2-Z1);

SET PARTICLE_ID = 1; SET DISP_SHORT = 1000;

MARK 1

RETRIEVE PARTICLE[PARTICLE_ID] CENTRE POSITION: (XP, YP, ZP)
VOLP = RETRIEVE PARTICLE_VOLUME[PARTICLE_ID];

CALCULATE PARTICLE_RADIUS: RP = CUBE_ROOT [(3/(4*PI))*VOLP]

SOLVE LINE_SPHERE_EQUATION() FOR a, b, c:
CALCULATE b^2 – 4*a*c;
CALCULATE t1, t2

IF b^2 – 4*a*c > 0 AND t1 > 0 AND t2 > 0 THEN
        ADD ID TO SHORTLIST
        SET t = t1;

        IF (t1 > t2) THEN
                SET t=t2;
        END IF

        CALCULATE INTERSECTION Xi = X1 + t*(X2-X1);
        CALCULATE INTERSECTION Yi = Y1 + t*(Y2-Y1);
        CALCULATE INTERSECTION Zi = Z1 + t*(Z2-Z1);

        CALCULATE RAY-TO-PARTICLE (RTP) DISPLACEMENT:
        DISP_RTP = SQRT [(Xi – X1)^2 + (Yi – Y1)^2 + (Zi – Z1)^2]

        IF (DISP_SHORT >= DISP_RTP)
                DISP_SHORT = DISP_RTP;
                INTERSECTED_PARTILCE_ID = PARTICLE_ID
        END IF

END IF

ADD 1 TO PARTICLE_ID;

IF PARTICLE_ID > TOTAL_NUMBER_OF_PARTICLES THEN
        END PROGRAM / MOVE TO NEXT RAY;
ELSE
        GOTO MARK 1;
END IF
```

Algorithm 3.1. Pseudo algorithm for determination and calculation of ray-particle interception. The corresponding C++ code can be found in Appendix.

3.2.6 RESULTS AND DISCUSSION

It can be observed in **Figure 3.23** that by increasing the average size of the powders, i.e., by adding the greater composition of the largest sized SP-5 powders in the mixture, the flowability started to improve relatively. A smaller angle of repose characterises better flowability.

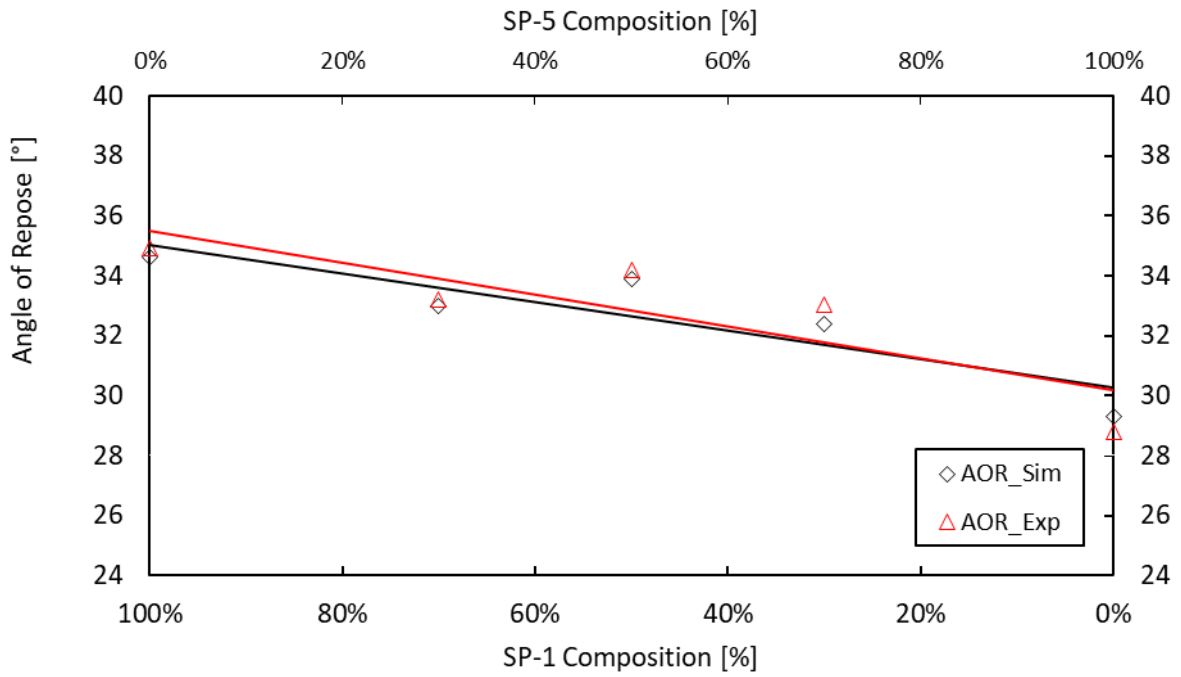


Figure 3.23. Variation in **flowability** of powders in terms of angle of repose (AOR), w.r.t. composition of SP-1 and SP-5 powders.

Considering the graded scale from Riley et al. [45] (**Table 2.2**, **Page 24**) all the powders mixtures in our study presented good to excellent flowability. However, it would be important to note that the smallest SP-1 powders could only be considered borderline 'good' at 34.9°, as the angle of repose formed above 35° fell into the fair category. The reason is that smaller powders have greater contact surface area for the same mass. It led to increased surface cohesivity and friction, reducing their flowability. Vice-versa, the larger-sized particles of the

same total mass had a cumulatively smaller number of inter-particle contacts, which reduced the cohesion and friction between them and led to better flowability.

Figure 3.24 demonstrates the change in packing density of the powders, with decreasing percentage composition of smaller powders (SP-1) and an increasing composition of larger powders (SP-5), for the first and the final (fourth) layer. The **black** coloured lines correspond to **200 µm** layer size, while the **red** and **blue** represent **150 µm** and **100 µm,** respectively, in the figure below.

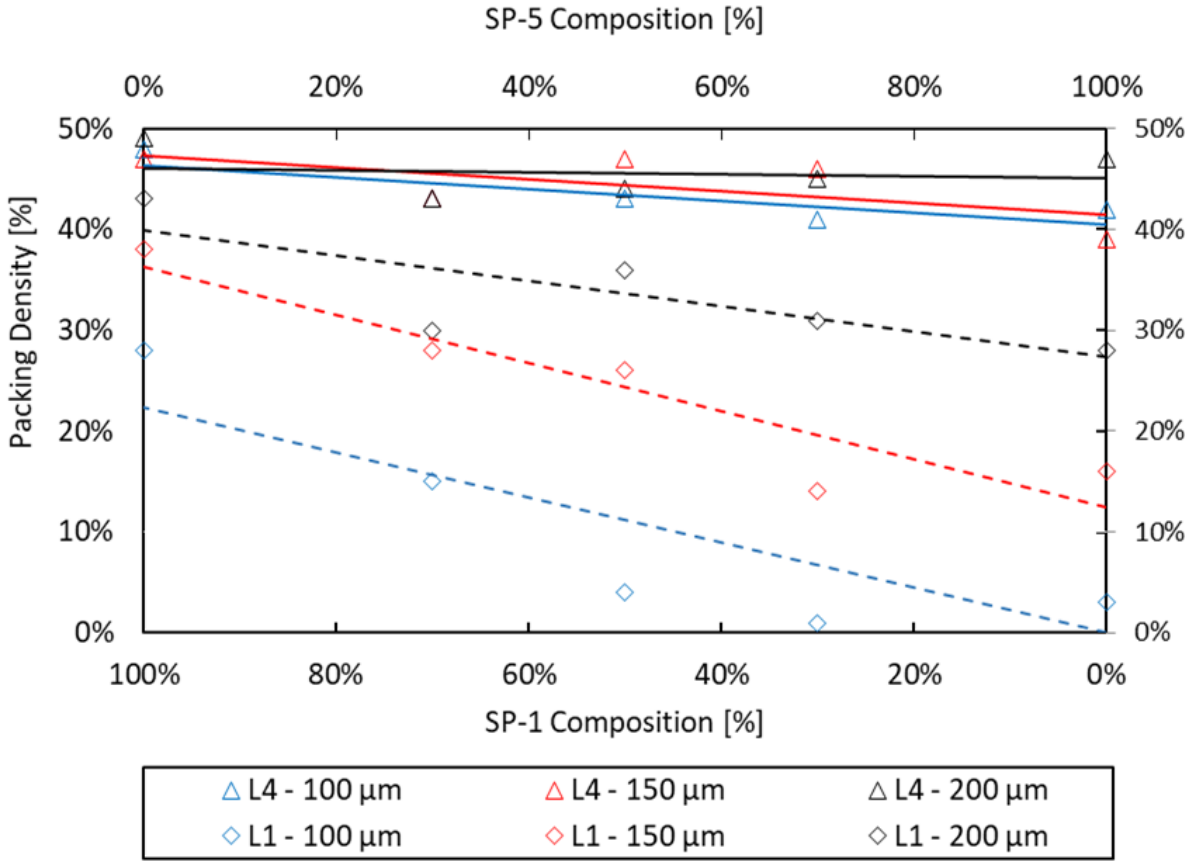
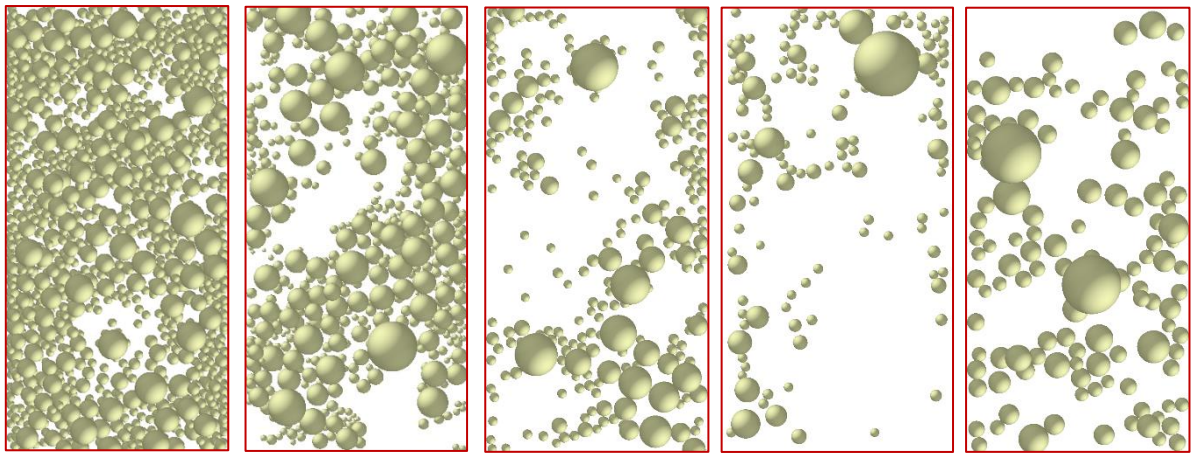


Figure 3.24. Change in **packing density** of alumina powder w.r.t. percentage composition of SP-1 and SP-5 powders for initial and final layers and different layer thicknesses of 100 µm, 150 µm, and 200 µm.

Interestingly, when we considered a 100 µm layer thickness (**Figure 3.25**) and moved towards the larger end of the spectrum, i.e., increased SP-5 composition, the general quality of the first layers started to drop. The main reason was that when the average particle size began to

increase; they left room for fewer particles in that layer. Secondly, for a smaller layer thickness, the ground to blade-tip clearance would be decreased. This meant that larger sized particles were more likely to be pushed off by the blade (**Figure 3.25**-**c,d,e**) compared to smaller (**Figure 3.25**-**a,b**), which led to smaller solid (particles) volume.



(a) SP-1 **(b) SP-2** **(c) SP-3** **(d) SP-4** **(e) SP-5**

Figure 3.25. Illustration of the first layer of 100 µm size producing visually significant voids in all cases, growing with increasing the powder size.

The average PD was only 10% for any mixture composition of SP-1 and SP-5 for the first layer of 100 µm. A similar pattern was followed by the first layers of 150 µm and 200 µm size, i.e., decreased PD quality against reduced SP-1 composition. Compared to 100 µm, the 150 µm layer size produced much better powder bed quality at the start, with an average PD of 24%, with fewer particles swept away by the blade. In contrast, the 200 µm layer size generally produced a denser powder bed from the very beginning, with a PD of 34%.

Moving towards the final layer, the smallest powder SP-1 produced the densest powder bed, averaging 48% PD across the considered layer sizes. The smaller particles naturally tend to create a denser solid volume. Compared to SP-1, the largest SP-5 powders produced a PD of 43% but with a larger variation across the layer sizes. Even though this is 5% lower than SP-1,

it was still not too far from it. The reason is that the SP-5 powders, despite having larger average particles size, also contained the particles towards the smaller end of the spectrum. Those particles tended to act as void fillers, leading to better-than-expected final PD. Moving back to SP-3 powders containing 50% each of SP-1 and SP-5, this powder had a final PD of 45% averaged across the layer sizes in question.

It can be seen in **Figure 3.26** that the overall surface roughness tended to increase with an increasing composition of larger SP-5 powders in the mixture. Both experimentally measured and with our ray tracing method for DEM powder bed exhibit a very similar trend. However, there was an exception with the SP-3 (50% SP-1 and 50% SP-5) powders, where the simulations predict smaller surface roughness than the experiments.

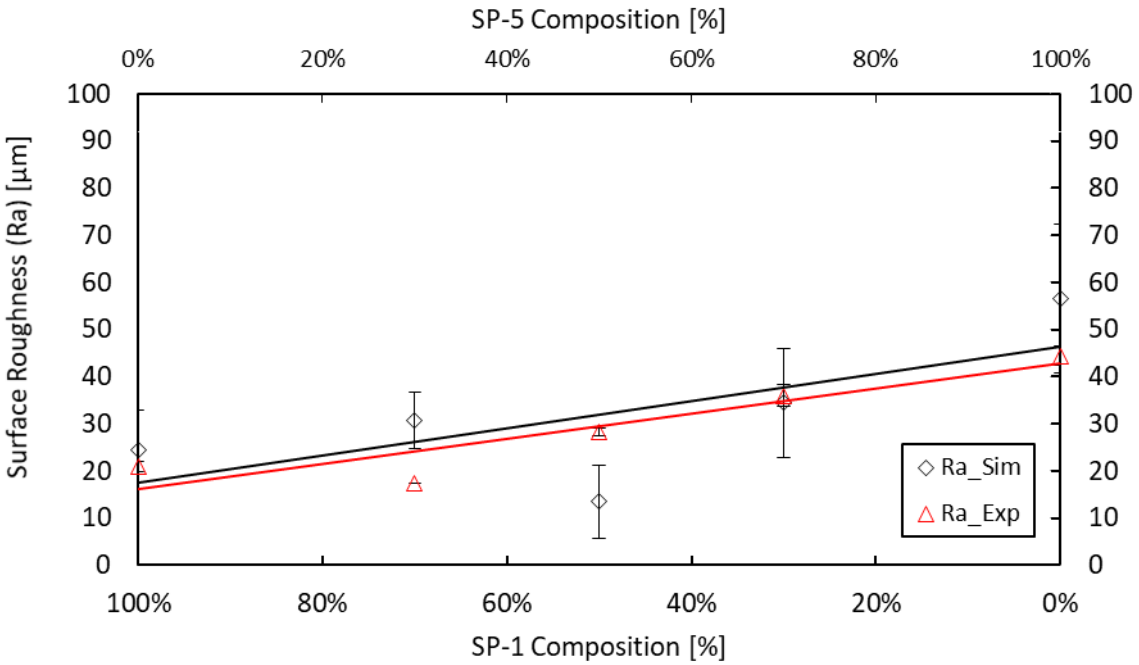


Figure 3.26. Variation in **surface roughness** of alumina powder w.r.t. percentage composition of SP-1 and SP-5 powders for the final layer of 150 µm.

3.3. HYDROXYAPATITE POWDERS

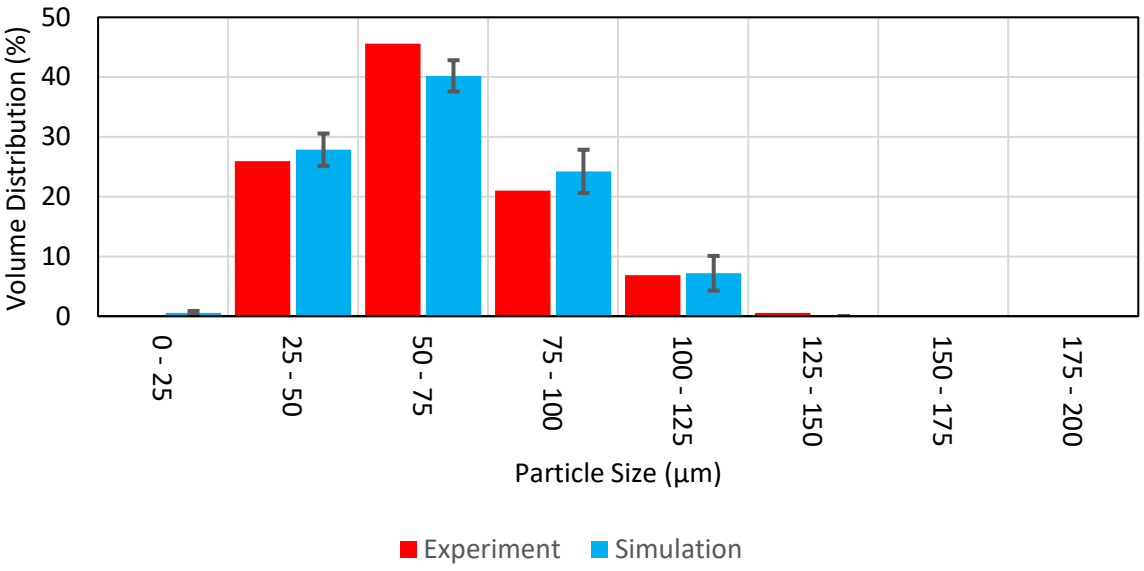
3.3.1 DEM POWDER GENERATION AND VOLUMETRIC DISTRIBUTION

The EDEM parameters adapted to generate the size distributions discussed in the following paragraph are listed in **Table 3.7**.

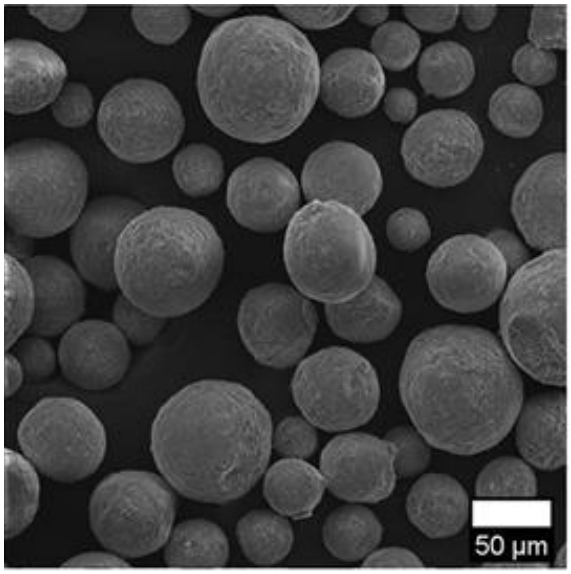
Table 3.7. Parameters used in EDEM to generate the respective size distributions for the powders, for HAP powders.

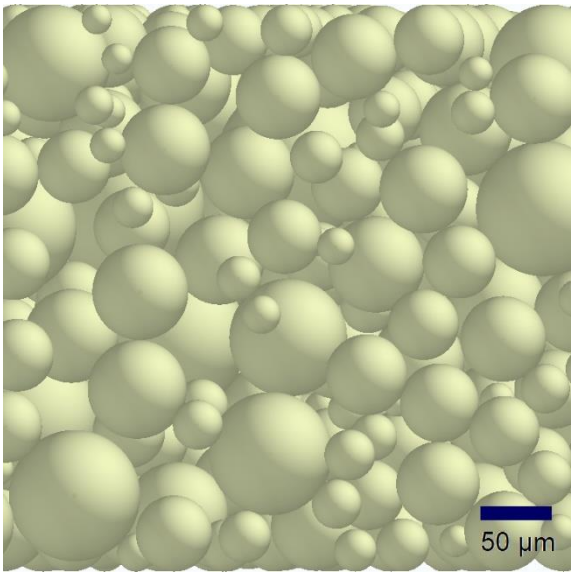
Classification		Parameters				
Powder type	Diameter size range [µm]	Radius [m]	Scaling	Standard deviation	Limit/cap*	
					Lower	Upper
Urodelia® 300-08-2	20 ≤ D <40	$1.5 \cdot 10^{-5}$	1	0.25	0.67	1.33
	40 ≤ D ≤ 120	$3.3 \cdot 10^{-5}$	1.15	0.3	0.75	1.81

The bar chart in **Figure 3.27-a** compares powders' experimentally measured volumetric distribution with the DEM-generated distribution. It can be noted that a very small quantity of powders having less than 25 µm diameters was observed, against which the DEM generated 0.53 (± 0.35) %. 25.92% of the powders lay between 25 and 50 µm, while DEM generated 24.15 (± 2.71) %. The particles in the range 50 to 75 µm held the most significant portion of 45.58%, and DEM generated 40.20 (± 2.62) %. The range 75 to 100 µm carried 21.01% of the particles, and DEM generation was 24.23 (± 3.62) %. In the range of 100 to 125 µm, there were 6.87% of the powders, while DEM generated 7.18 (± 2.91) %. Only 0.61% of the powders had a size larger than 125 µm, but they were ignored in the DEM to avoid the shear effect of larger particles within the small domain.



(a)





(b) **(c)**

Figure 3.27. (a) SEM micrograph for commercial Urodelia® 300-08-2 powder [88] and **(b)** DEM generation of the powders with the same size distribution.

It can further be observed from **Figure 3.27-a** and **Table 3.8** (below) that the powder generated in our DEM simulations presented a very good match with the distribution data provided by CIRMIAT. **Figure 3.27-b,c** illustrates the SEM micrographs for the actual powders [88] against the DEM generation for visualization. Here it would be important to reiterate that an exact match is impossible due to the random generation of particles. Furthermore, it would be pertinent to mention again that the smallest particle determines critical timestep size in the domain. Therefore, the particles below the size of 20 μm were neglected as smaller particles require a smaller timestep for the simulations, which leads to a high computational load.

Table 3.8. Summary of the particle sizes for the HAP (Urodelia® 300-08-2) powders used in this study, with respect to their cumulative percentages of 10%, 50% and 90%.

Powder type	Particle size [μm]					
	D_{10}		D_{50}		D_{90}	
	Physical	**Simulation**	**Physical**	**Simulation**	**Physical**	**Simulation**
Urodelia® 300-08-2	40	45	59	62	92	92

3.3.2 POWDER CALIBRATION FOR SIMULATIONS

Similar to alumina powders, the HAP powders generated in the DEM simulations were also calibrated for their flowability against the actual powders used in experiments. **Figure 3.28** compares the angle of repose against the experimental measurements (image courtesy of Dr Pedro Jesus Navarrete-Segado) from the CIRIMAT.

Table 3.9. Material properties and interaction coefficients for hydroxyapatite powders.

Powder type	ρ [kg m^{-3}]	ν -	E [Pa]	e -	μ -	μ_R -	C [J m^{-2}]
Urodelia® 300-08-2	3180	0.2	$1.68 \cdot 10^8$	0.1	0.2	0.05	$5.0 \cdot 10^{-4}$

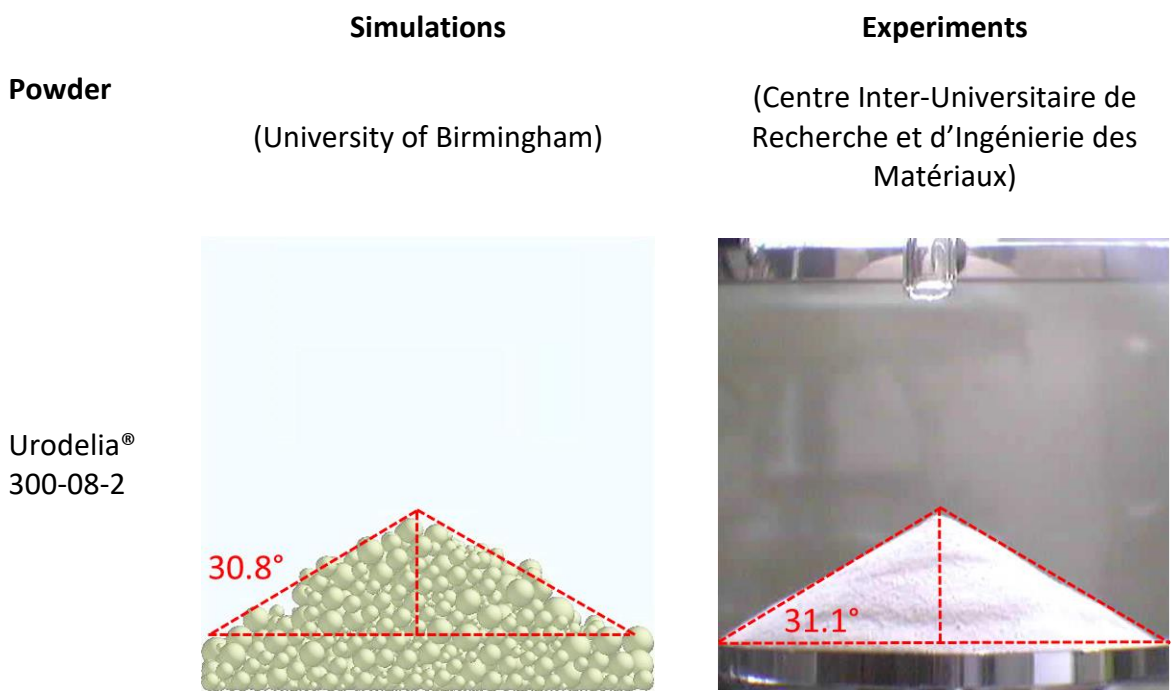


Figure 3.28. Comparison of simulated and experimented static angle of repose used for calibration of hydroxyapatite powders: Urodelia® 300-08-2.

3.3.3 POWDER BED COMPRESSION

Powder beds are subjected to compression before scanning the laser on their surfaces. For instance, Navarrete-Segado et al. stated that they compressed their initially deposited powders prior to laser processing to obtain better packing density [89]. In experimental data provided by CIRIMAT, the 300 µm layer was compressed to 100 µm. However, when this was directly implemented in the DEM program, an unrealistic pressured sinter-like overlap of particles was observed (**Figure 3.29**), as the DEM powders mimic hard spheres (i.e., uncompressible). So, they could not readjust (or reposition) themselves in the (smaller) voids.

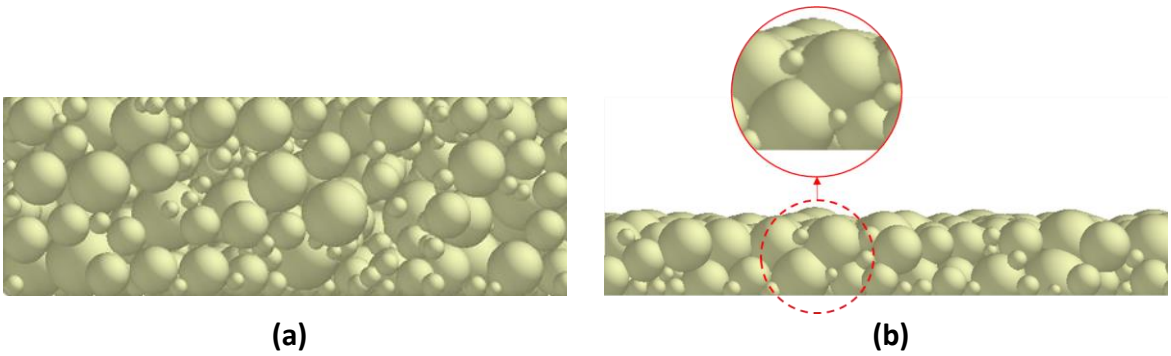


Figure 3.29. (a) Pre-compressed powder bed: ~300 µm. **(b)** Post-compressed powder bed: 100 µm, showing unrealistic overlaps.

In reality, the ceramic particles can only be pressed to a certain level, often due to their brittle nature, before they break down into smaller bits to allow further compression. It would mean that we end up with a new (post-compression) size distribution of particles. Therefore, in our current simulation work, we only compressed the powder to an extent where any significant overlaps did not appear. The original deposited powder thickness in DEM was about 300 µm and compressed down to 150 µm.

4. LASER HEAT SOURCE AND ITS THERMAL INTERACTION

Highlights:

1. Brief introduction to Gaussian heat distribution and FEM

2. Gaussian energy allocation to the discretized laser beam for laser ray tracing

3. Extension of ray tracing algorithm to laser absorption, reflections, and transmissions

4. Potential issues and troubleshooting in the code

5. Consideration of the Biot number of the particle

4.1. INTRODUCTION

The mathematical modelling of laser heat source and its interaction with the powder bed has been discussed in this chapter. We considered two separate methodologies of finite element method (FEM) and discrete element method (DEM), in order to illustrate the comparison.

The laser heat flux is usually defined by Gaussian distribtuion of energy across the surface within the confines of beam diameter. Most of the energy is concentrated about the beam centre and it exponentially decays near the boundary (**Figure 4.1-a**) [92]. However its travel through the surface depends on the material type and distribution. In case of a continuous solid and non-translucent body, a simple 2D surface heat flux can be used. However, if the body is translucent or transparent, the laser rays would penetrate. Their attenuation beneath the surface is then further described by the Beer Lambert's law (**Figure 4.1-b**) [93]. A 3D volumetric heat flux is defined in this case, which combines surface heat flux and depth attenuation. Such implementation is often used in the finite element method (FEM) domain for laser and powder-bed interaction.

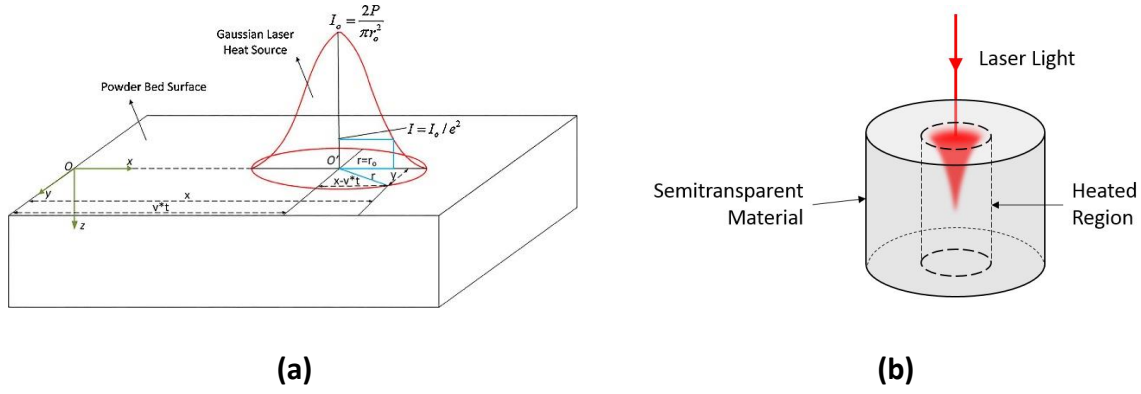


Figure 4.1. (a) Schematic diagram of laser beam irradiating on the powder bed surface [92]; **(b)** Illustration of ion of light intensity attenuation via the Beer-Lamber law [93].

However, the interaction of the laser with the powder bed becomes nonuniform due to discrete particles situated randomly. Each particle absorbs certain amount of energy,

transmits some, and reflects the rest in the direction depending on the ray's strike position on its surface. The reflected and transmitted rays further interact with other particles in the similar manner, forming a haphazard path within the powder bed. Therefore, we developed a laser ray tracing method coupled with the discrete element method (DEM), to define the path of each ray and energy distribution within the powder bed, in order to obtain a closer insight of laser and powder bed interaction.

4.2. GENERAL MODELLING OF LASER HEAT FLUX

A typical laser is represented by the Gaussian heat flux distribution on the material surface [78]. The laser penetrates through the surface of the powder bed to a certain degree. Beer-Lambert's law was used to approximate the attenuation of the laser through the thickness. So, it could not simply be modelled as the surface heat-flux, but its travel through the depth was considered. Therefore, volumetric heat flux was defined as [75]:

$$q(x,y,z) = (1-R)\alpha I_0(x,z)e^{-\alpha y} \tag{4.1}$$

where R is the surface reflectivity, I_0 is the surface laser heat-flux (W·m^{-2}), and α is the absorption coefficient (m^{-1}), and it defines the attenuation of laser energy. The porosity of the powder bed was considered, and the absorption coefficient was further defined as [75]:

$$\alpha = \alpha_0(1-\varphi) \tag{4.2}$$

where α_0 is the absorption coefficient of the bulk material, and φ is the porosity. The Gaussian distribution of the laser is expressed as:

$$I_0(x,z) = \frac{2P}{\pi r_0^2} e^{-2\frac{r^2(x,z,t)}{r_0^2}} \tag{4.3}$$

where P is the laser power (W), r_0 is the radius of the laser spot. The function $r(x,z,t)$ describes the general distance from the laser beam/spot centre at the given time (t):

$$r^2(x,z,t) = (x-x_0)^2 - (z-z_0-vt)^2 \tag{4.4}$$

where x and z are the surface coordinates , v is the velocity (m s^{-1}) of the laser scan. The final form of the **Equation (4.1)** can be represented as:

$$q(x,y,z) = (1-R)\alpha\frac{2P}{\pi r_0^2}e^{-2(\frac{r^2}{r_0^2}+\alpha y)} \quad (4.5)$$

The temperature distribution rapidly changes in space and time due to the laser scan (movement of laser) on the material's surface. Therefore, it can be studied through the 3D transient heat-conduction equation [71]:

$$\rho(T)c_p(T)\frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k(T)\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k(T)\frac{\partial T}{\partial y}\right) + \frac{\partial}{\partial z}\left(k(T)\frac{\partial T}{\partial z}\right) + q \quad (4.6)$$

where ρ is the density (kg·m^{-3}), c_p is the specific heat capacity (J·kg^{-1}·K^{-1}), k is the thermal conductivity (W·m^{-1}·K^{-1}), T is the temperature (K), and q is the internal heat source (W·m^{-3}), respectively. The spatial coordinates are represented by x, y, and z, respectively. The initial condition at t = 0 was assumed uniform and standard room temperature:

$$T(x,y,z,t)|_{t=0} = 293.15\ K \quad (4.7)$$

4.3. RAY-TRACING AND DEM COUPLING

The ray tracing code for laser heating of powder bed shared brief commonalities with the one developed for evaluating surface roughness (**Section 3.2.5, Page 89**), which included discretization of the laser beam into the finite number of rays and evaluation for initial ray-particle interception. However, its construction became much more complex as ray reflections, transmissions, and energy absorption were introduced.

4.3.1 BEAM DISCRETIZATION AND ENERGY ALLOCATION

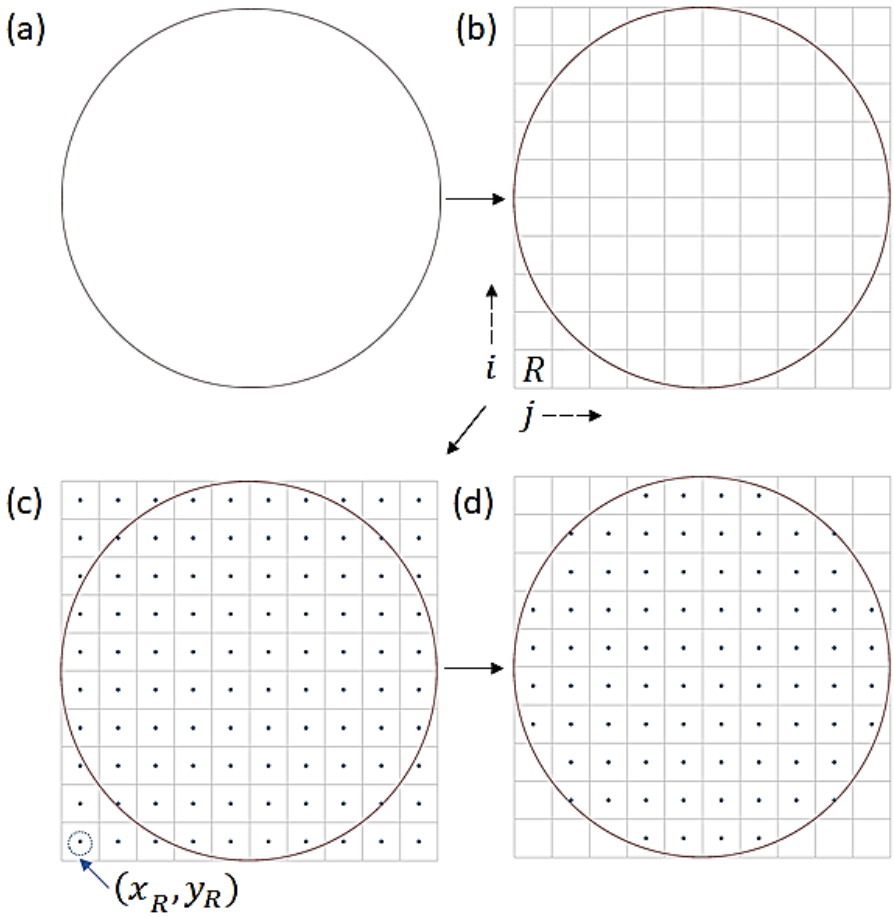


Figure 4.2. Step-by-step discretization of laser beam; **(a).** Defining the circular bound; **(b).** Introducing square of sides equal to diameter of the circular domain, and discretization into smaller square cells (denoted by R); **(c).** Setting up rays' centre positions (x_R, y_R); **(d).** Elimination of rays formed outside the circular domain.

In contrast to the ray-tracing method for evaluating surface roughness, where the square domain was considered, the domain of the laser beam was considered circular bound (**Figure 4.2-a**). Initially, similar to the previous method, a square region with sides equal to the diameter of the domain was generated, and it was discretized into the finite number of square cells, again with their location identified by row (i) and column (j) number (**Figure 4.2-b**). The rays were allocated using the same methodology discussed previously and represented by dots '·' at the centre of each cell (**Figure 4.2-c, d**). However, at this point, all the rays falling outside the circular domain were eliminated (**Figure 4.2-d**) using the criteria given in **Equation (4.8)**. It assessed if the given 'ray to beam-centre distance' was greater than the 'beam radius' ($R_B = \frac{D_B}{2}$):

$$\sqrt{(x_R - x_0)^2 + (y_R - y_0)^2} > R_B \tag{4.8}$$

where (x_0, y_0) are the centre coordinates of starting position of the (circular) laser beam. The energy (e_R) for the original ray (i.e., before reflection or transmission) was allocated to each cell or ray (R), according to Gaussian distribution with time-step size and cell-area factor:

$$e_R = \frac{2P}{\pi R_B^2} e^{-2\frac{(x_R - x_0)^2 + (y_R - y_0)^2}{R_B^2}} \Delta T A_R \tag{4.9}$$

where P is the total laser beam power, ΔT is the timestep size, and A_R is the area of each square cell. It means that the cells located closer to the centre carried greater energy than the ones farther. The energy of the cells closer to the boundary of the beam usually tended to be zero. The colour-coded illustration (**Figure 4.3-a**) below represents the general distribution of cells energies. The red end of the spectrum depicts the highest energy at the centre of the beam, while the blue for the lowest energy. Furthermore, the curve plot in **Figure 4.3-b** shows

the Gaussian energy variation from one end of the border to the other passing through the centre. This was always symmetrical.

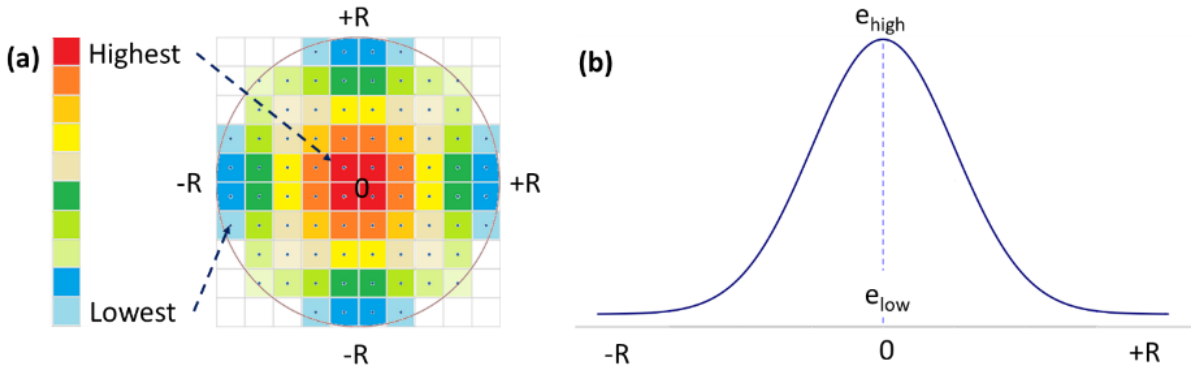


Figure 4.3. Generic representation of discretized energy distribution in the cells.

Furthermore, a numerical inaccuracy in this method can be observed in that some of the cells situated closer to the boundary have their partial area draining outside, but their energy is still fully considered. While the other cells, with their centres located outside but closer to the boundary and fractional of their region falling inside, were completely disregarded. This is due to the initial consideration in **Equation (4.8)** that those cells with their (local) centres outside the beam boundary were discarded from the calculation, while those with their centre even marginally inside were still completely assumed. This inaccuracy cannot be completely eliminated but can be further reduced by finer discretization of the domain (i.e., into smaller-sized cells). It would also be important to highlight that these cells have very low energy, which has been exponentially dropped, as given in **Equation (4.9)** and illustrated in **Figure 4.3-b**. So, their overall influence on the temperature change of the particles would also be minimal. However, this would not be the case if a top-hat laser profile (uniformly distributed), in opposition to gaussian, is adopted.

4.3.2 MINIMUM CELL SIZE FOR PARTICLE DETECTION

The beam must be discretized to the point where the smallest generated particle in the simulation does not go undetected if it falls within its boundary. It is important to note that the rays are located at the centre of the cell. So, if the cells are not small enough, the particle can potentially be missed (**Figure 4.4**). Assume that the particle is located exactly at the intersection of 4 cells. The particle would have the highest probability of staying undetected at this location.

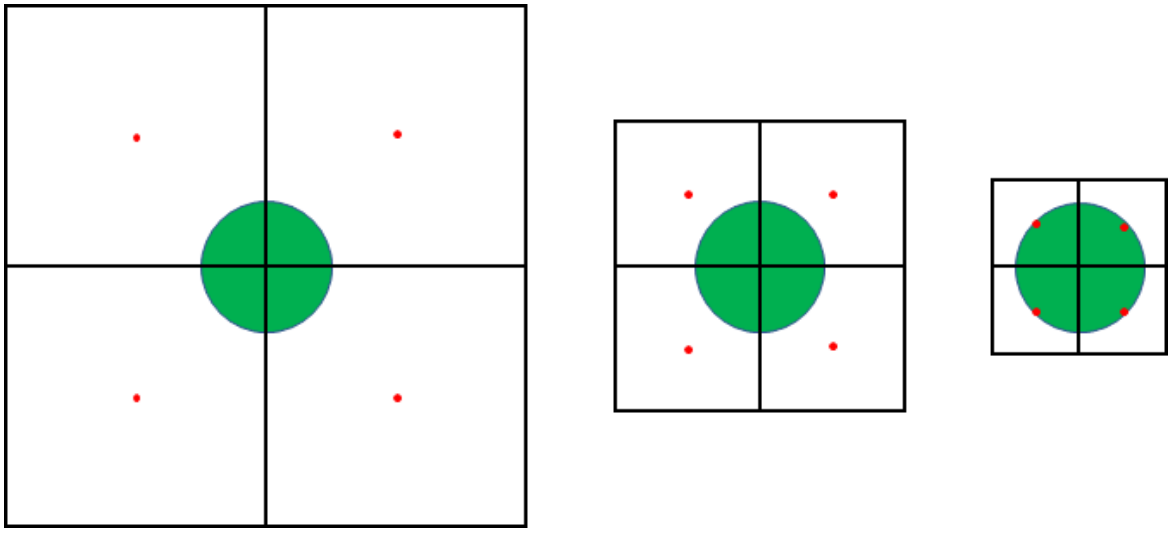


Figure 4.4. Illustration of a particle located within differently sized discretized beam cells. The particle goes potentially undetected in **(a)** and **(b)** due to the coarse discretization of the beam. In case **(c)** the discretization is fine enough to ensure that one or more rays will detect the particle.

Therefore, to eliminate such possibility, the rays must be located within the bounds of the particle. In this case, the condition to help determine the minimum number of cells (N), that the beam must be discretized into, can be derived as follows:

$$Cellsize = \frac{D_B}{N} \tag{4.10}$$

$$\downarrow$$

$$\frac{D_P}{2}\sin 45° > \frac{Cellsize}{2} \tag{4.11}$$

$$\downarrow$$

$$D_P > \sqrt{2}Cellsize \tag{4.12}$$

$$\downarrow$$

$$D_P > \sqrt{2}\frac{D_B}{N} \tag{4.13}$$

$$\downarrow$$

$$\therefore \boldsymbol{N > \sqrt{2}\frac{D_B}{D_P}} \tag{4.14}$$

where $Cellsize$ is the length of the side of the square cell, D_B is the beam diameter, and D_P is the particle diameter.

4.3.3 THE RAY TRACING METHOD AND ALGORITHM

The pseudo algorithm (**Algorithm 4.1**) was devised to implement the overall process for the laser beam's thermal interaction with the powder bed. It enabled the development of a C++ code in **Appendix D (Page 235)**. The main frame of the algorithm (**Figure 4.5**) tracked the travel of each ray originating from the laser source, their subsequent reflection and refraction (transmission) counterparts, and their remaining energies. The calculation for reflection was always performed first, which was followed by refraction for all the given rays. The flow of the algorithm was such that refraction of the 'originating ray' (from the source beam) was checked after its reflection and further corresponding reflection/refraction had been performed. The checks for both reflections $rfl[RR]$ and refraction $rfr[RR]$ for the given particles were recorded to avoid recalculations, given as:

$$rfl[RR] \in [0, 1] \tag{4.15}$$

and

$$rfr[RR] \in [0, 1] \tag{4.16}$$

where RR is the integer identifier for reflection/refraction, 0 represents 'not checked or calculated', and 1 means 'already checked or calculated'. In a given time step, multiple rays (whether original or due to back-reflections or -refractions) could hit the particles. So, the energies gained, as a result, were accumulated. After each hit, part of the ray energy was absorbed, and the rest was allocated to corresponding reflection and refraction rays.

The three sub-algorithms (listed in **Table 4.1**) worked inside this frame of **Algorithm 4.1** that evaluated the ray-particle interception, calculated the reflection on the particle surface and refraction (or transmission) through them. Based on Snell's law, the algorithm for refractions

also considered the total-external-reflection phenomena. They have been discussed separately in the following sections.

Table 4.1. List of devised pseudo algorithms used in developing the ray tracing code for laser heating of powder bed.

Algorithm function	**Cross-references**	
	Flowchart figure	**Algorithm**
Ray tracking and identification	**Figure 4.5, Page 117**	**Algorithm 4.1, Page 119**
Calculation: Ray-particle interception	**Figure 3.22, Page 94**	**Algorithm 3.1, Page 95**
Calculation: Reflection	**Figure 4.9, Page 123**	**Algorithm 4.2, Page 124**
Calculation: Transmission (Refraction)	**Figure 4.10, Page 127**	**Algorithm 4.3, Page 129**

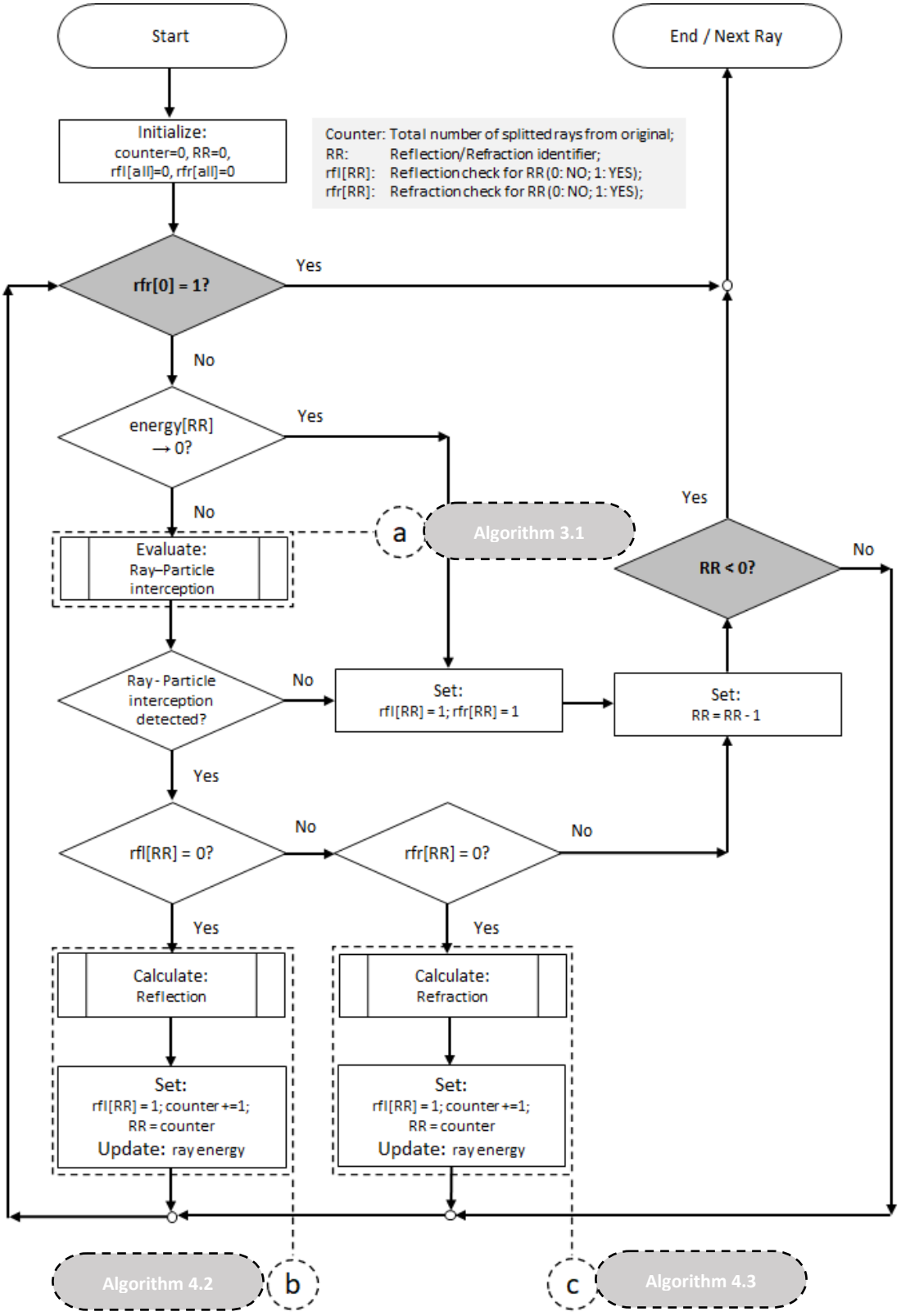


Figure 4.5. A complete rays' identifier and tracer algorithm with integrated single ray-particle interaction algorithm (**Figure 4.6**); Sub-algorithms: **(a).** Checks ray-particle interception; **(b).** Calculates reflected ray's direction and energy; **(c).** Calculates refracted ray's direction after entering, and then emerging from the particle and energy.

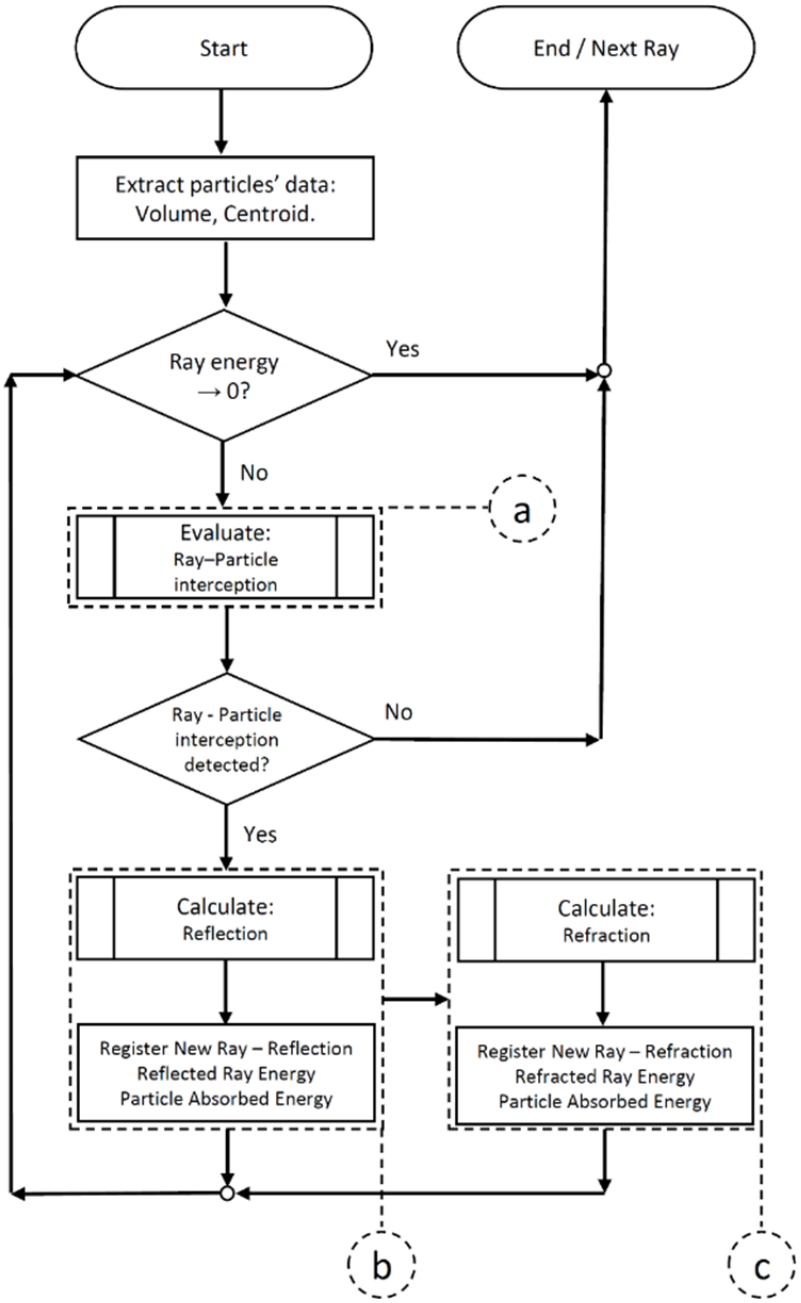


Figure 4.6. Extraction from **Figure 4.5** to represent simplified single ray-particle interaction.

```
CALCULATE RAY_TRACING:

SET COUNTER = 0;        Counter counts the total number of rays splitted from original
SET RR = 0;             RR is the identifier for reflection or refraction
SET RLF[ALL] = 0;       RFL is the check for reflection
SET RFR[ALL] = 0;       RFR is the check for refraction

MARK 1                  Reference point

IF RFR[0] = 1 THEN
        END PROGRAM / MOVE TO NEXT ORIGINAL RAY;
END IF

IF RAY ENERGY = 0 THEN
        SET RFL[RR] = 1;
        SET RFR[RR] = 1;
        IF RR < 0 THEN
                END PROGRAM / MOVE TO NEXT RAY;
        END IF
END IF

CALCULATE RAY-PARTICLE_INTERCEPTION  (Algorithm 3.1, Page 95);

IF RAY-PARTICLE_INTERCEPTION() = 0 THEN
        SET RFL[RR] = 1;
        SET RFR[RR] = 1;
        IF RR < 0 THEN
                END PROGRAM / MOVE TO NEXT RAY;
        END IF
END IF

IF RFL[RR] = 0 THEN
        CALCULATE REFLECTION (Algorithm 4.2, Page 124);
        SET RFL[RR] = 1;
        ADD 1 TO COUNTER;
        SET RR = COUNTER;
        UPDATE RAY_ENERGY;
ELSE IF RFR[RR] = 0 THEN
        CALCULATE REFRACTION (Algorithm 4.3, Page 129);
        SET RFL[RR] = 1;
        ADD 1 TO COUNTER;
        SET RR = COUNTER;
        UPDATE RAY_ENERGY
END IF

GOTO MARK 1
```

Algorithm 4.1. Pseudo algorithm for the ray-tracing method for laser beam interaction with the powder bed.

4.4. INTERCEPTION, REFLECTIONS, AND TRANSMISSIONS

4.4.1 RAY-PARTICLE INTERCEPTION

The mathematical analogue to ray-particle interception is the line-sphere intersection, assuming all particles to be spherical (**Figure 4.7**).

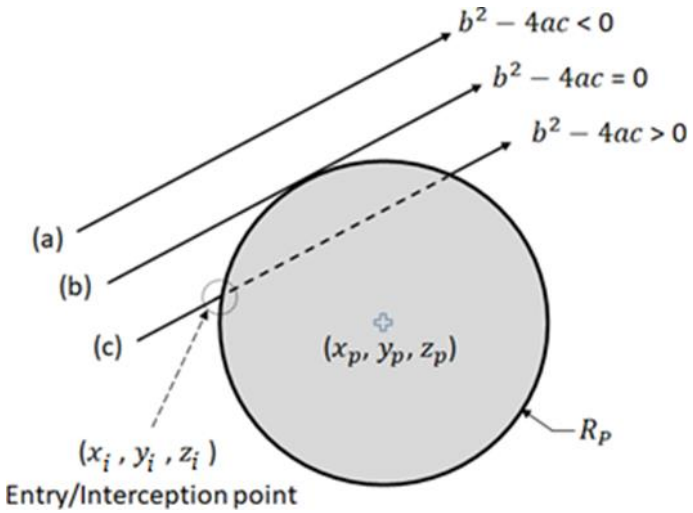


Figure 4.7. Possible ray-particle or line-sphere interactions; **(a).** No interception occurs; **(b).** Tangential contact at a single point; **(c).** Ray path interferes with the particle, with two possible contact points, where only the entry contact point is considered; The direction of the rays/lines is assumed for illustrative purposes only.

This part of **Algorithm 4.1** follows the same procedure for calculating the line-sphere interception, as given in **Section 3.2.5**, from **Equation (3.6)** to **(3.11)**. The implementation flow of this methodology in our code is shown in **Figure 3.22** (**Page 94**) and **Algorithm 3.1** (**Page 95**). This was started by extracting all the particles' IDs, locations (i.e., centroid coordinates) and diameter and then shortlisting those potentially interfering with the ray's path (**Figure 4.6-a**). The solution generally resulted in two values for t in the quadratic **Equation (3.8)**, indicating multiple interceptions. The smaller t was used to calculate the first intersection point, which was taken as the point where the ray hits the particle. In contrast, the larger t was disregarded, as the ray does not travel straight through the particle; instead, it is refracted

during transmission through the particle because of having different refractive indices for atmosphere and particle material. Therefore, the coordinates for an emerging point were calculated differently (discussed later under **Section 4.4.3**).

The energy absorbed by the intercepting particle (e_P) is given by:

$$e_P = \alpha_{abs} e_R \tag{4.17}$$

where e_R is the energy carried by the ray, as given in **Equation (4.9)**, and α_{abs} is the fraction of absorbed laser energy, which can be represented in terms of the fraction of reflected ray (α_{rfl}) energy and refracted (or transmitted) ray energy (α_{rfr}), as following:

$$\alpha_{abs} = 1 - \alpha_{rfl} - \alpha_{rfr} \tag{4.18}$$

Thus, the temperature gained by the particle (ΔT) as a result is given by:

$$\Delta T = \frac{e_P}{m_P c_p} \tag{4.19}$$

where m_P is the mass of the particle, and c_p is the specific heat capacity at constant pressure.

4.4.2 RAY REFLECTION

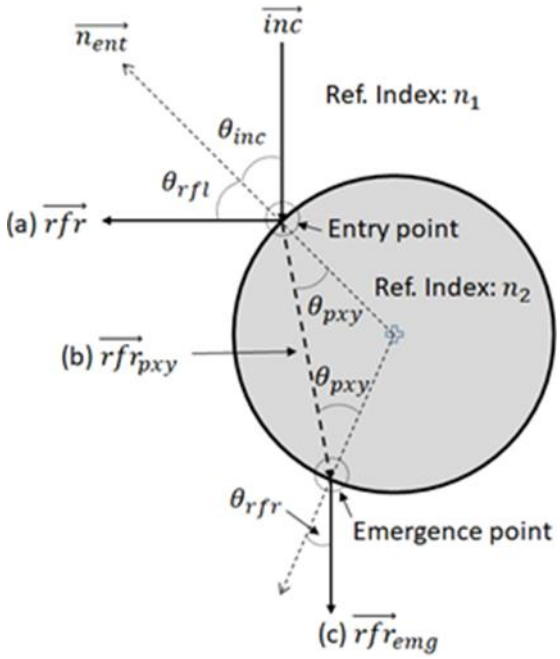


Figure 4.8. Splitting of incident ray; **(a).** Surface reflection; **(b).** Proxy refraction, transmitted through the particle; **(c).** Emergent or final refraction.

The reflection of the incident ray ($\overrightarrow{inc}$) was calculated at the interception/entry point (**Figure 4.8**), in the next step. The vector form for the reflection ($\overrightarrow{rfl}$) is given by (**Figure 4.8-a**) [94]:

$$\overrightarrow{rfl} = \overrightarrow{inc} + 2\cos\theta_{rfl}\,\vec{n} \tag{4.20}$$

where θ_{rfl} is the reflection angle, which is equal to the incidence angle θ_{inc}, given by $\cos\theta_{rfl} = \cos\theta_{inc} = -\vec{n} \cdot \overrightarrow{inc}$, and $\vec{n}$ is the normal vector at the interception (or entry) point. The remaining energy of the reflected ray (e_{rfl}), i.e., less absorbed and transmitted, is given by:

$$e_{rfl} = \alpha_{rfl}\, e_R \tag{4.21}$$

The implementation flow of this methodology in our code is given in **Figure 4.9** and **Algorithm 4.2**.

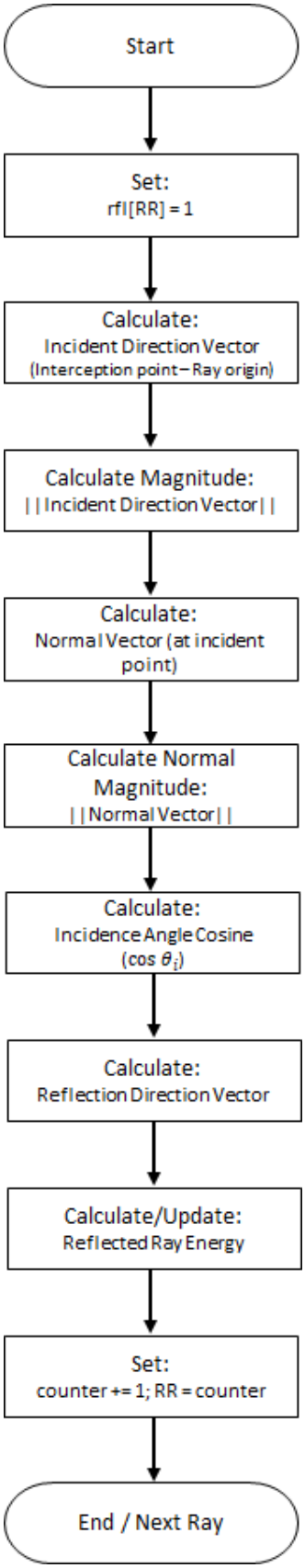


Figure 4.9. Step-by-step calculation of ray reflection. This flow chart represents the expanded/sub-process given in **Figure 4.5-b**.

CALCULATE REFLECTION:

SET RFL[RR] = 1;

RETRIEVE RAY_INC_ENERGY;

SET RAY_INCIDENT_DIRECTION:
(Xinc, Yinc, Zinc) = (X2 - X1, Y2 - Y1, Z2 - Z1);

CALCULATE RAY_INCIDENT_MAGNITUDE:
MAGinc = SQRT ((X2 - X1)^2 + (Y2 - Y1)^2 + (Z2 - Z1)^2);

CALCULATE NORMAL_DIRECTION:
(Xn, Yn, Zn) = (Xi - XP, Yi - YP, Zi - ZP)

CALCULATE NORMAL_MAGNITUDE:
MAGn = SQRT ((Xi - XP)^2 + (Yi - YP)^2 + (Zi - ZP)^2);

CALCULATE RAY_INCIDENT_COSINE:
COSinc= -1 * (Xinc * Xn + Yinc * Yn + Zinc * Zn) / (MAGinc * MAGn);

CALCULATE RAY_REFLECTION_DIRECTION:
Xrfl = (Xinc / MAGinc) + 2 * COSinc * (Xn / MAGn);
Yrfl = (Yinc / MAGinc) + 2 * COSinc * (Yn / MAGn);
Zrfl = (Zinc / MAGinc) + 2 * COSinc * (Zn / MAGn);

SET RAY_RFL_ENERGY = (1 – FRAC_ABSORBED – FRAC_TRANSMITTED) * RAY_INC_ENERGY

Algorithm 4.2. Pseudo algorithm for calculation of reflection.

4.4.3 RAY TRANSMISSION (REFRACTION) AND TOTAL EXTERNAL REFLECTION

Apart from the reflection, the incident ray also splits into refraction, which transmits through the surface. The relation between incidence angle and refraction angle is described using Snell's law:

$$n_1 \sin\theta_{inc} = n_2 \sin\theta_{rfr} \tag{4.22}$$

and the refracted ray ($\overrightarrow{rfr}$) is given by [94]:

$$\overrightarrow{rfr} = \left(\frac{n_1}{n_2}\right)\overrightarrow{inc} + \left(\frac{n_1}{n_2}\cos\theta_{rfl} - \cos\theta_{rfr}\right)\vec{n} \tag{4.23}$$

where n_1 is the refractive index of the region where the incident ray is coming from, n_2 is the refractive index of the region or material the ray travels into. The refractive index (n) indicates the bending of light when it enters from one optical medium into another. θ_{rfr} is refraction angle is given by [94]:

$$\cos\theta_{rfr} = \sqrt{1 - \left(\frac{n_1}{n_2}\right)^2 (1 - \cos^2\theta_{rfl})} \tag{4.24}$$

In our implementation, these calculations were performed twice, first for refraction into the particle (termed proxy refraction - $\overrightarrow{rfr_{pxy}}$, and its counterpart refractive angle as a proxy angle - θ_{pxy}) (**Figure 4.8-b**), and then for the refraction of emerging ray from the particle (**Figure 4.8-c**), which is counted as final refraction. In the second case, two changes were made in the equation (4.24): $\frac{n_1}{n_2}$ is inverted to $\frac{n_2}{n_1}$ and $\theta_{rfl} \equiv \theta_{pxy}$ as initially the proxy ray is inside the particle with refractive index n_2. The emerging ray is also called the transmitted ray. The remaining energy for this transmitted ray is given by:

$$e_{rfr} = \alpha_{rfr}\, e_R \tag{4.25}$$

In some cases, the refraction does not occur, and the ray is completely reflected from the surface. This phenomenon is called total external reflection. It happens due to the ray travelling from a higher refractive index medium to a lower one. The limiting case for incidence angle, which can result in refraction just along the interface surface of two media, i.e., at 90°, is called critical angle. In the case of a spherical particle, this condition can be imagined as a refraction ray's direction resulting in a parallel to the tangent formed at the incident ray's entry point. The critical incidence angle (θ_{ctc}) is given as follows, setting θ_{rfr} in **Equation (4.22)** to 90°:

$$\theta_{ctc} = \sin^{-1}\frac{n_2}{n_1} \tag{4.26}$$

Thus, the calculation for simple reflection is performed instead of refraction, using **Equation (4.20)**, when the following condition holds:

$$\theta_{inc} > \theta_{ctc} \tag{4.27}$$

Therefore, the resulting energy of the reflected ray is completely conserved, i.e., set equal to the energy of the incident ray, as the ray is unable to penetrate the particle to be absorbed and further transmitted:

$$e_{rfl} = e_R \tag{4.28}$$

In our code, the normal reflected ray and the reflected ray resulting from total-external-reflection are counted as two different rays to keep their record. The implementation flow of this methodology in our code is given in **Figure 4.10** and **Algorithm 4.3**.

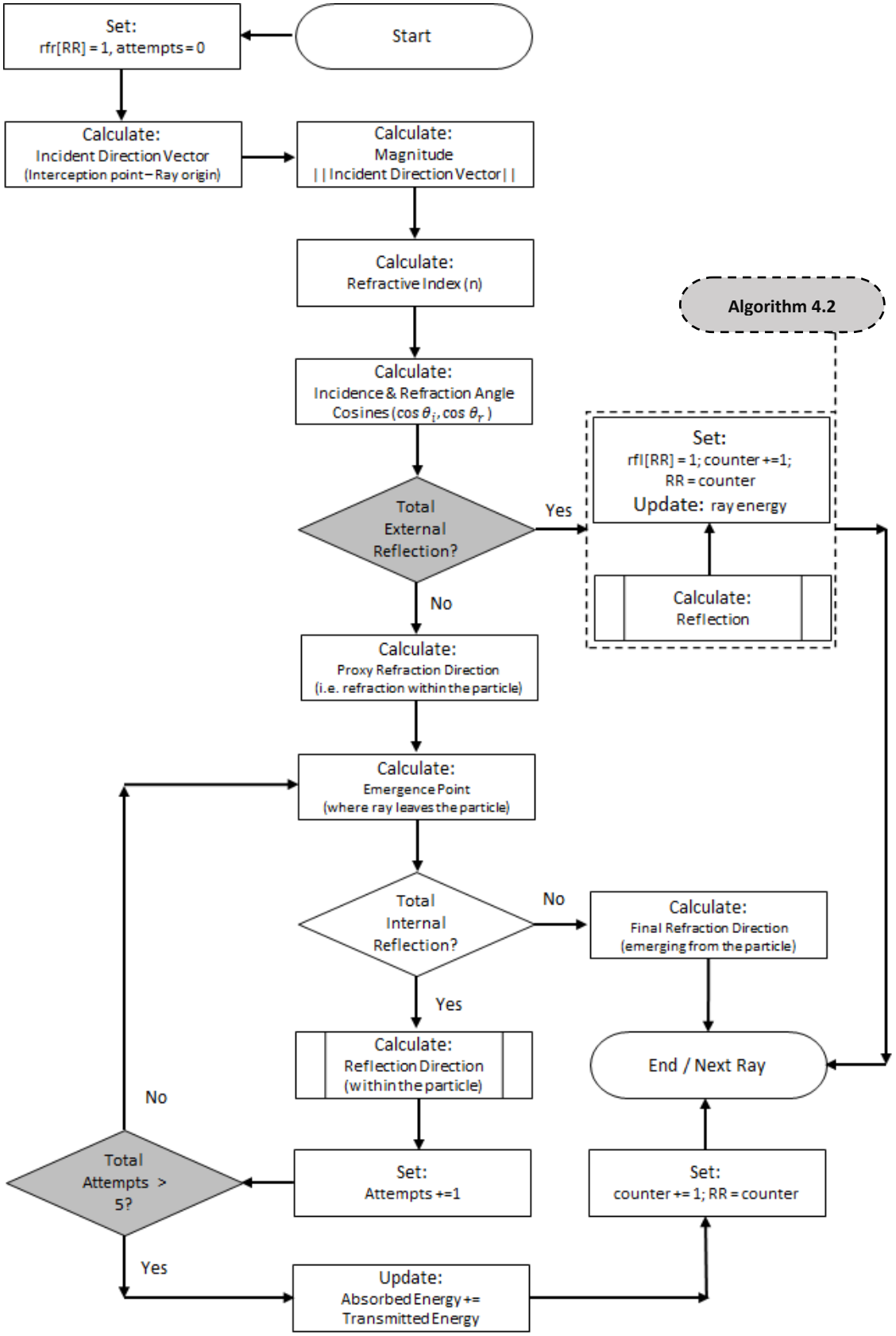


Figure 4.10. Step-by-step process for calculation and determination of refraction or total-internal-reflection and/or total-external-reflection. This flow chart represents the expanded/sub process given in **Figure 4.5-c**.

CALCULATE TRANSMISSION (WITHOUT TOTAL INTERNAL REFLECTION CONDITION);

SET RFR[RR] = 1;
SET ATTEMPTS = 0;
SET REFRACTIVE_INDEX_RATIO (RIR) = n_FLUID / n_PARTICLE;
SET RIR_INVERT = 1 / RIR;

RETRIEVE RAY_INC_ENERGY;

SET RAY_INCIDENT_DIRECTION:
(DXinc, DYinc, DZinc) = (Xi - X1, Yi - Y1, Zi - Z1);

CALCULATE RAY_INCIDENT_MAGNITUDE:
MAGinc = SQRT ((Xi - X1)^2 + (Yi - Y1)^2 + (Zi - Z1)^2);

CALCULATE NORMAL_DIRECTION_ &_MAGNITUDE:
(Xn, Yn, Zn) = (Xi – XP, Yi – YP, Zi – ZP);
MAGn = SQRT ((Xi – XP)^2 + (Yi – YP)^2 + (Zi – ZP)^2);

CALCULATE RAY_INCIDENT_COSINE_&_ANGLE:
COSinc = -(DXinc * Xn + DYinc * Yn + DZinc * Zn) / (MAGinc * MAGn);
ANGinc = COS^{-1} (COSinc);

CALCULATE CRITICAL_INCIDENT_ANGLE:
ANGcrc = SIN^{-1} (RIR_INVERT);

CHECK FOR TOTAL_EXTERNAL_REFLECTION:

IF (ANGinc > ANGcrc) THEN
    CALCULATE REFLECTION (Algorithm 4.2, Page 124);
    OVERRIDE RAY_RFL_ENERGY = RAY_INC_ENERGY
    END PROGRAM / MOVE TO NEXT RAY
ELSE
    CALCULATE RAY_PROXY_COSINE using equation (4.24):
    COSpxy = SQRT (1 – (RIR^2) * (1 – COSinc^2));

    CALCULATE RAY_PROXY_DIRECTION_&_MAGNITUDE using equation (4.23):
    DXpxy = RIR * (DXinc / MAGinc) + (RIR * COSinc – COSpxy) * (Xn / MAGn);
    DYpxy = RIR * (DYinc / MAGinc) + (RIR * COSinc – COSpxy) * (Yn / MAGn);
    DZpxy = RIR * (DZinc / MAGinc) + (RIR * COSinc – COSpxy) * (Zn / MAGn);
    MAGpxy = SQRT (DXpxy^2 + DYpxy^2 + DZpxy^2);

    CALCULATE RAY-PARTICLE_INTERCEPTION (Algorithm 3.1, Page 95);
    RETRIEVE RAY_PROXY_COORDINATES (Xpxy, Ypxy, Zpxy) from above calculation;

    CALCULATE EMERGENCE-POINT_NORMAL_DIRECTION_&_MAGNITUDE:
    (Xn_emg, Yn_emg, Zn_emg) = (XP – Xpxy, YP – Ypxy, ZP – Zpxy);
    MAGn_emg = SQRT ((XP – Xpxy)^2 + (YP – Ypxy)^2 + (YP – Ypxy)^2);

    CALCULATE RAY_REFRACTION-FROM-EMERGENCE_COSINE using equation (4.24):
    COSpxy_emg = COSpxy;

```
        COSrfr_emg = SQRT (1 – (RIR_INVERT^2) * (1 – COSpxy_emg^2));

        CALCULATE RAY_REFRACTION_DIRECTION using equation (4.23):
        DXrfr = RIR_INVERT * (DXpxy / MAGpxy) + (RIR_INVERT * COSpxy_emg – COSrfr_emg) *
        (Xn_emg / MAGn_emg);
        DYrfr = RIR_INVERT * (DYpxy / MAGpxy) + (RIR_INVERT * COSpxy_emg – COSrfr_emg) *
        (Yn_emg / MAGn_emg);
        DZrfr = RIR_INVERT * (DZpxy / MAGpxy) + (RIR_INVERT * COSpxy_emg – COSrfr_emg) *
        (Zn_emg / MAGn_emg);

       SET RAY_RFR_ENERGY = (1 – FRAC_ABSORBED – FRAC_REFLECTED) * RAY_INC_ENERGY
END IF
```

Algorithm 4.3. Pseudo algorithm for calculation of refractive transmission.

4.4.4 ALGORITHM PITFALLS AND TROUBLESHOOTING

Three potential pitfalls in the algorithms were identified during initial testing. They were then addressed to avoid the detection of the incorrect particle against the given ray.

(1). Since there are many particles in the powder bed, each ray potentially intersects multiple of them during each time step, as illustrated in **Figure 4.11-a**. Only the first intercepted particle interacts with the ray; the rest must be discarded. To address this issue, the ray-to-particle displacement (δ_{RTP}) was evaluated for each possible intersection as follows:

$$\delta_{RTP} = \sqrt{(x_R - x_i)^2 + (y_R - y_i)^2 + (z_R - z_i)^2} \tag{4.29}$$

where (x_R, y_R, z_R) are ray's originating coordinates and (x_i, y_i, z_i) are the intersection coordinates located on the particle's surface. The particle with the shortest δ_{RTP} (**Figure 4.11-c**) would only be shortlisted for energy absorption and further operations such as reflection and transmission.

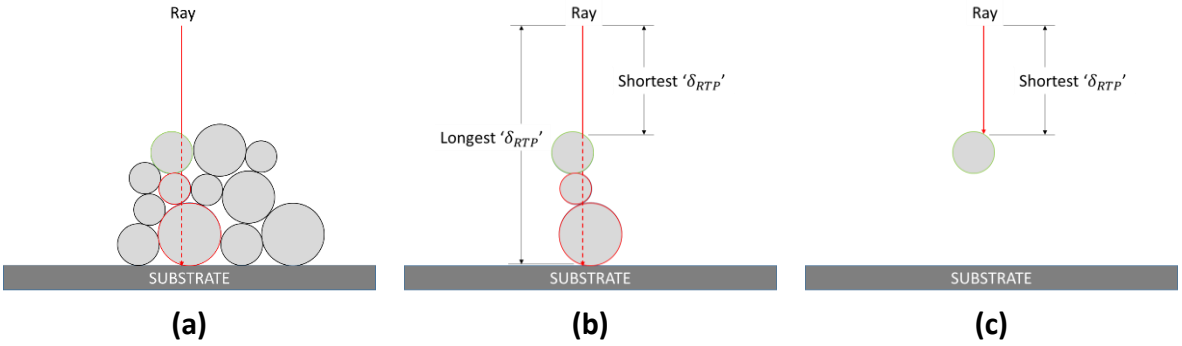


Figure 4.11. Procedure for selection of correct particle intercepted by the ray. The red particles represent potential but false detection, while the green represent the correct detection; **(a).** Ray intersecting multiple number of particles; **(b).** Non-intersected particles eliminated from the count; **(c).** The particle with the shortest ray-to-particle (at intersection point) displacement considered.

(2). The other two issues were primarily related when the reflection calculation was carried out (method discussed under **Section 4.4.2**). The reflected ray would detect the same particle for intersection, i.e., the one it was being reflected from (**Figure 4.12-a**).

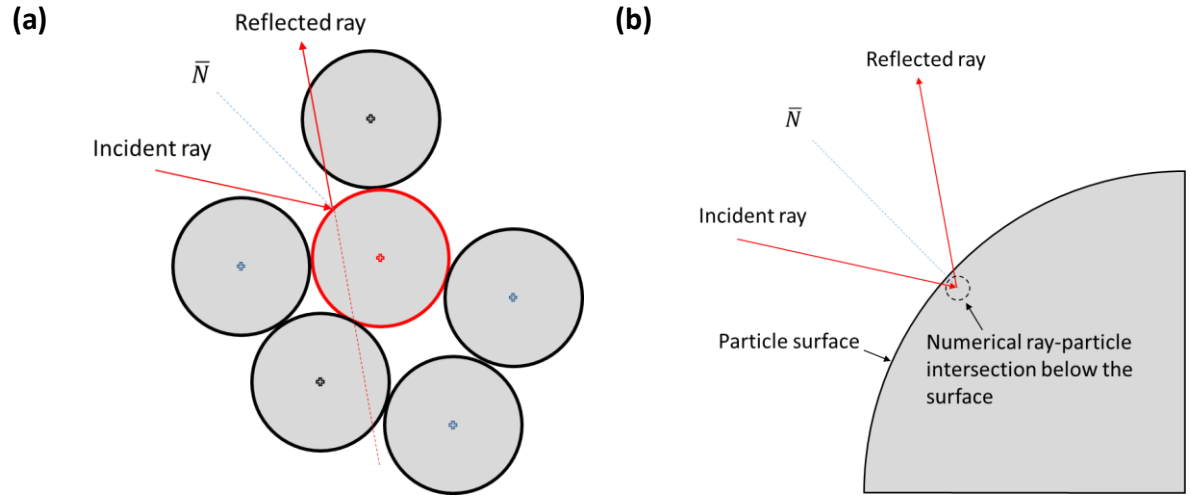


Figure 4.12. Illustration of reflected ray interacting with particles, with potential pitfall: **(a).** False detection: The same particle (red) as for the incident ray being redetected by the reflected ray; **(b).** The intersection point was evaluated slightly below the surface.

It occurred because of a small numerical error or inaccuracy, where the code would evaluate the values for intersection coordinates as situated slightly beneath the particle surface rather than exactly on the surface (**Figure 4.12-b**). It was addressed by introducing the following condition in the C++ code given in **Appendix-D** (**Page 235**):

```
//-------------------------------------------------------------------------------//
// Calculate interception-points coordinates (where applicable):
//-------------------------------------------------------------------------------//

int redetect = 0;

// Check if same particle is being detected by the reflected ray:
if (RR > 0 && ptc_ID == myptc[ray][RR - 1]) redetect = 1;

if (b * b - 4 * a * c > 0 && redetect == 0 && t[0] > 0 && t[1] > 0)
{
```

Figure 4.13. An extract from the ray-tracing code for laser heating of powders is given in **Appendix D**.

The condition checked if the integer identifier for reflection/refraction RR (**Section 4.3.3, Page 114**) was not 0 (i.e., the original ray), and the particle ID being considered for the ray-particle interception check was not the same as detected by the original ray. This was done by initially setting a so-called variable 'redetect' to 0. If the same particle ID is detected as for the original ray, the variable 'redetect' would be set to 1. So, the following interception calculation for this particle would not be carried out.

(3). The last issue was related to both reflected and transmitted rays. The ray would detect the particles in the backward direction (**Figure 4.14-a**). It was resolved by considering the condition $t > 0$, from **Equation (3.8) (Page 91)**, to detect the particle (**Figure 4.13**) correctly. This check ensured that the particle being tested for interception had both potential intersection points situated in the forward direction. This issue was in contrast to when considering the original rays, they were projected from the top of the powder bed in the downward direction, and therefore, no particles could be detected above them (i.e., in the relative backwards direction).

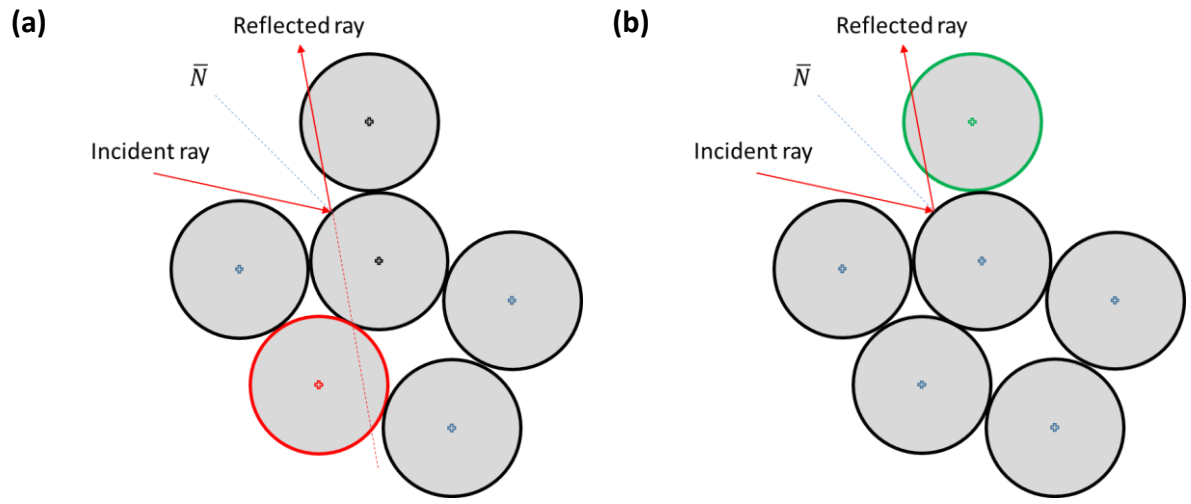


Figure 4.14. Illustration of reflected ray interacting with particles, with potential pitfall: **(a).** False detection: The particle (red) in the backward direction to the ray detected; **(b).** Correct detection (green).

4.5. LASER AND POWDER-BED THERMAL INTERACTION

4.5.1 MINIMUM PARTICLE SIZE FOR LASER INTERACTION

The given condition must be met for the laser-particle interaction, which determines the minimum particle diameter (D_p) that can be used against the given wavelength of laser: [5]:

$$\frac{\pi D_p}{\lambda} \gg 1 \tag{4.30}$$

where λ is the (laser) wavelength. For instance, Tran and Lo (2017) justified the use of 20 µm particle size against an Nd:YAG laser (1.064 µm), where the minimum diameter should have been 0.34 µm [92].

Table 4.2. Minimum particle size to be considered against the given laser wavelength, calculated from equation (4.30).

Laser type	Wavelength (λ) [µm]	Particle Size (D_p) [µm]
CO_2	10.6	3.37
Nd:YAG	1.064	0.34
HPDL	0.81	0.26

4.5.2 LASER ABSORPTION

Tolochko et al. measured the absorptivity of different single-component alumina powders against Nd:YAG and CO_2 lasers. Alumina exhibited a very low absorptivity of 3% for Nd:YAG lasers and, in contrast, a very high absorptivity of 96% for CO_2 lasers [6]. Navarrete-Segado et al. reported the fibre laser absorption for hydroxyapatite (HAP) with and without graphite additives. Very low absorption of 2.4% was observed without additives, while it jumped to 66.8% and 73.7% for 5% graphite and 10% graphite additives, respectively [89] (**Table 4.3**).

Table 4.3. Laser absorption of alumina and hydroxyapatite powders [6], [89].

Laser type	Wavelength (λ) [µm]	Alumina	Hydroxyapatite		
			No additives	5% Graphite	10% Graphite
CO_2	10.6	0.96	-	-	-
Nd:YAG	1.064	0.03	-	-	-
Fibre	1.07	-	0.024	0.668	0.737

4.5.3 REFRACTIVE INDEX

Querry compiled the optical constants for various materials and reported their refractive indices over a range of wavelengths. For α-alumina, it was 0.615 and 1.749 against the wavelengths of 10.6 μm (i.e., CO_2) and 1.064 μm (i.e., Nd:YAG), respectively [95]. Very limited information was available in the literature on the refractive index of HAP, specifically against the fibre laser. Different sources reported 1.63 – 1.65 as the refractive index of Hydroxyapatite [96]–[98] for the visible light spectrum. The values have been summarized in the following **Table 4.4**:

Table 4.4. The refractive indices for α-alumina and hydroxyapaptite [95].

Type	Wavelength (λ) [μm]	Refractive index (n)	
		Alumina	HAP
Laser CO_2	10.6	0.615	0.12658
Laser Nd:YAG	1.064	1.749	0.02
Visible	0.4 – 0.7	-	1.63 – 1.65

4.5.4 MODELLING OF CONVECTIVE AND RADIATIVE HEAT LOSSES

Heat losses were only assumed to occur on the powder bed's top layer (**Figure 4.15**). Convective and radiative heat exchanges within the pores of inner layers were ignored.

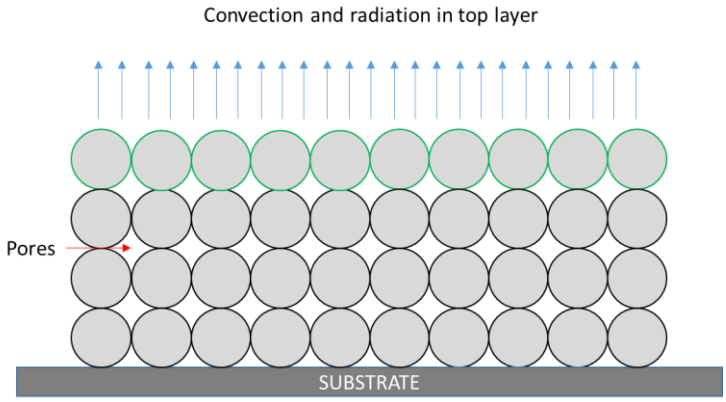


Figure 4.15. The green particles on the top layer only were subjected to convection and radiative heat losses.

The particle in the top layer was approximated by considering if its centroidal coordinate-Z was situated within a distance from the top surface equal to its radius (**Figure 4.16**). The smaller particles were less likely to stay on top as they tended to travel through pores and were likely buried in the layers beneath. So, this condition allowed a proportionate chance of being considered on the top layer. The post-compression powder bed thickness defined the top to bottom depth.

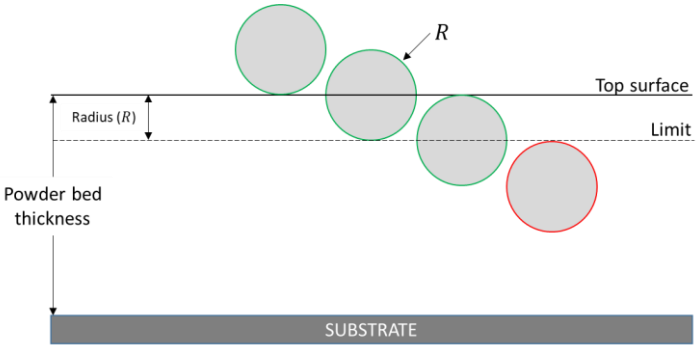


Figure 4.16. The green particles are selected and considered to lose heat. They have their centroids located within the limit of their respective radii from the top surface. While the red particles are off limit and are considered below the top surface.

The resultant heat losses through convection and radiation can be represented as [75]:

$$-k\frac{\partial T}{\partial n} = h(T_S - T_a) + \sigma\varepsilon(T_S^4 - T_a^4) + q \tag{4.31}$$

where T_S is the surface temperature (K), T_a is the ambient temperature (K), n is the normal vector to the surface, h *is* the temperature-dependent heat-transfer coefficient (W·m^{-2}·K^{-1}), ε is the emissivity, σ is the Stefan-Boltzmann constant (5.67×10^{-8} W·m^{-2}·K^{-4}), and q is the input heat-flux from the laser source (W·m^{-2}). Heat transfer through the lower surface was assumed to be absent, so an adiabatic boundary condition was applied:

$$-k\frac{\partial T}{\partial y} = 0 \tag{4.32}$$

Calculating the radiative part of the heat loss was relatively straightforward, as given in **Equation (4.31).** However, to calculate the surface heat loss through convection, the heat transfer coefficient (h) had to be evaluated. The series of steps to calculate this have been outlined below [99]:

(1) Approximation of film temperature T_f (i.e., the temperature inside the convective boundary layer) and calculation of the coefficient of thermal expansion β (K^{-1}):

$$T_f = \frac{T_S + T_a}{2} \Rightarrow \beta \approx \frac{1}{T_f} = \frac{2}{T_S + T_a} \tag{4.33}$$

where T_S and T_a are surface and ambient temperatures (K), respectively.

(2) Calculation of Grashof number (Gr), i.e., ratio of the buoyancy to viscous force on the fluid:

$$Grashof\ Number\ (Gr) = \frac{g\beta(T_S - T_a)L_C^3}{\nu^2} \tag{4.34}$$

where g is the acceleration due to gravity, ν is the kinematic viscosity, and L_C is the characteristic length calculated as follows, using **Appx-A Table 1 (Page 225):**

$$L_c = D \tag{4.35}$$

where is A_s is the surface, and P is the surface perimeter.

(3) Determination of Prandtl number (Pr), i.e., the ratio of momentum diffusivity to thermal diffusivity, from **Appx-A Table 2 (Page 226)**.

(4) Calculation of Rayleigh number (Ra), i.e., a dimensionless number linked with the natural convection:

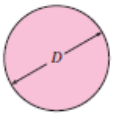
$$Ra = Gr \cdot Pr \tag{4.36}$$

(5) Calculation of Nusselt number (Nu), i.e., ratio of convective to conductive heat-transfer at the fluid boundary, using **Table 4.5** [100]:

Table 4.5. An excerpt from **Appx-A Table 1**, for empirical correlations for the average Nusselt number during natural convection over horizontal surfaces [100].

TABLE 9–1

Empirical correlations for the average Nusselt number for natural convection over surfaces

Geometry	Characteristic length L_c	Range of Ra	Nu
⋮			
Sphere 	D	$Ra_D \le 10^{11}$ ($Pr \ge 0.7$)	$Nu = 2 + \frac{0.589Ra_D^{1/4}}{[1+(0.469/Pr)^{9/16}]^{4/9}}$ (9-26)

(6) In the final step, the heat transfer coefficient (h) is calculated as:

$$Nu = \frac{h \cdot L_C}{k} \Rightarrow h = \frac{Nu \cdot k}{L_C} \tag{4.37}$$

The calculations in the **Table 4.6** have been made assuming ‘air’ as the surrounding fluid.

Table 4.6. Calculation of heat transfer coefficient over the range of temperatures.

T_s [°C]	T_f [°C]	$\beta \approx \frac{1}{T_f}$ [K^{-1}]	k [W m^{-1} K^{-1}]	v [m^2 s^{-1}]	Prandtl number	Rayleigh number	Nusselt Number	h [W m^2 K^{-1}]
25	25	0.00335	0.02551	0.0000156	0.7296	0	0	**0**
30	27.5	0.00333	0.02588	0.0000161	0.7282	0.0000574336	0.005032	**2.605**
35	30.0	0.00330	0.02625	0.0000166	0.7268	0.0001073348	0.006197	**3.253**
40	32.5	0.00327	0.02662	0.0000170	0.7255	0.0001507177	0.006939	**3.694**
45	35.0	0.00325	0.02699	0.0000175	0.7241	0.0001881782	0.007471	**4.033**
50	37.5	0.00322	0.02735	0.0000180	0.7228	0.0002206411	0.007878	**4.309**
60	42.5	0.00317	0.02808	0.0000190	0.7202	0.0002724067	0.00845	**4.746**
70	47.5	0.00312	0.02881	0.0000200	0.7177	0.0003103255	0.008825	**5.085**
80	52.5	0.00307	0.02953	0.0000210	0.7154	0.0003369323	0.00907	**5.357**
90	57.5	0.00302	0.03024	0.0000220	0.7132	0.0003548911	0.009228	**5.581**
100	62.5	0.00298	0.03095	0.0000231	0.7111	0.0003664086	0.009327	**5.773**
120	72.5	0.00289	0.03235	0.0000252	0.7073	0.0003747827	0.009397	**6.080**
140	82.5	0.00281	0.03374	0.0000275	0.7041	0.0003705132	0.009362	**6.317**
160	92.5	0.00273	0.03511	0.0000298	0.7014	0.0003587889	0.009262	**6.504**
180	102.5	0.00266	0.03646	0.0000321	0.6992	0.0003429080	0.009123	**6.653**
200	112.5	0.00259	0.03779	0.0000346	0.6974	0.0003250945	0.008963	**6.774**
250	137.5	0.00244	0.04104	0.0000409	0.6946	0.0002788467	0.008516	**6.990**
300	162.5	0.00230	0.04418	0.0000477	0.6935	0.0002364257	0.008061	**7.123**
400	212.5	0.00206	0.05000	0.0000520	0.7080	0.0002479211	0.008189	**8.189**
600	312.5	0.00171	0.06200	0.0000950	0.7202	0.0000960759	0.005972	**7.406**
800	412.5	0.00146	0.07200	0.0001300	0.7303	0.0000598912	0.005103	**7.348**
1000	512.5	0.00127	0.08100	0.0001950	0.7309	0.0000292512	0.004019	**6.512**
1200	612.5	0.00113	0.09000	0.0002150	0.7402	0.0000260512	0.003867	**6.961**
1400	712.5	0.00102	0.10000	0.0002500	0.7403	0.0000202622	0.003557	**7.114**

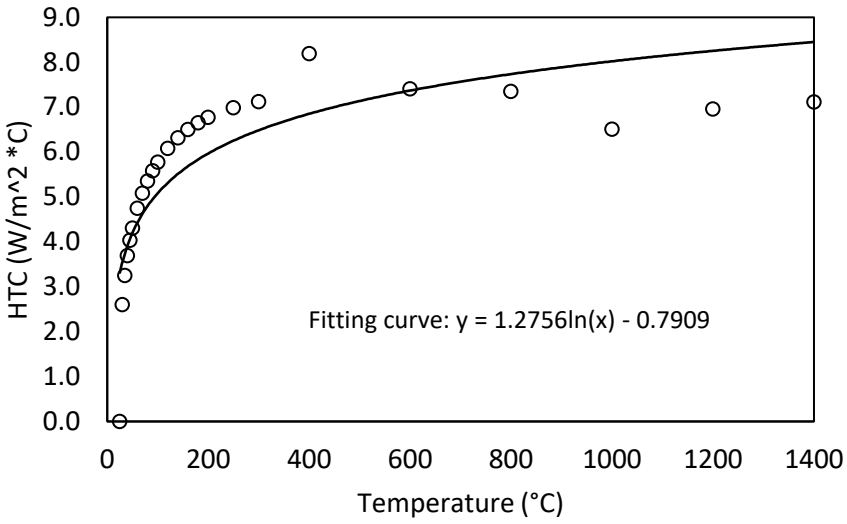


Figure 4.17. Plot of heat transfer coefficient against the temperature.

```
CALCULATE HEAT_LOSSES:

RETRIEVE PARTICLE_TEMP;
RETREIVE PARTICLE_RAD;
RETREIVE PARTICLE_POS (X,Y,Z);

SET POWDERBED_THICK = 150E-6

IF (PARTICLE_POS(Z) <= POWDERBED_THICK-PARTICLE_RAD)
        EXIT ALGORITHM
END IF

SET HEAT_CAPACITY = 628.02;
SET MASS_RESCALING = 1E14;
SET EMMISIVITY = 0.3;
SET AMBIENT_TEMP = 298.15;
SET BOLTZMANN = 1.38064852E-23

SURFACE_AREA = 2*PI*PARTICLE_RAD* PARTICLE_RAD //UPPER HEMISPHERE

CALCULATE HEAT_COEFFICEINT:
HEAT_COEFF = 1.2756*LOG(PARTICLE_TEMP-273.15)-0.7909

CALCULATE HEAT_CONVECTION (NEWTON'S LAW OF COOLING):
HEAT_CONVECTION = HEAT_COEFF*SURFACE_AREA*(PARTICLE_TEMP-AMBIENT_TEMP);

CALCULATE HEAT_RADIATION:
HEAT_RADIATION = EMMISIVITY*SURFACE_AREA*BOLTZMANN*(PARTICLE_TEMP^4 –
AMBIENT_TEMP^4);

CALCULATE TOTAL_HEAT_LOSS:
HEAT_LOSS_TOTAL = HEAT_CONVECTION + HEAT_RADIATION;

UPDATE PARTICLE_TEMP:
PARTICLE_TEMP -= (HEAT_LOSS_TOTAL*MASS_RESCALING*TIMESTEP_SIZE) /
                 (PARTICLE_MASS*HEAT_CAPACITY)
```

Algorithm 4.4. Pseudo algorithm for calculation of temperature change of the particles, due to thermal losses, The corresponding C++ code can be found in the **Appendix-F** (**Page 253**).

4.5.5 BIOT NUMBER

The Biot number was considered to determine if the particles could be treated as isothermal, i.e., with no significant thermal gradient within their bodies. It is a dimensionless number representing the ratio of *internal resistance to heat conduction* to *surface thermal resistance to heat convection* of the body. The smaller Biot number represents smaller resistance to heat conduction, i.e., smaller thermal gradients within the body; therefore, heat conduction within the particle can be neglected [101]. It is given by [99]:

$$Bi = \frac{hL_c}{k} \tag{4.38}$$

where h is the convective heat transfer coefficient, k is the thermal conductivity of the body, and L_c is the characteristic length of the geometry - in our case, it was the particle diameter (D_p). As long as the Biot number is well below unity, the temperature can be considered uniform within the particle [99]:

$$Bi \leq 0.1 \tag{4.39}$$

The change in Biot number against the temperature for a large-sized particle of 500 µm has been plotted in **Figure 4.18**. A larger particle was more likely to reach the above limit (4.39). However, it can be observed that the Biot number stayed well below that limit, even at high temperatures.

The particles used in our simulations were significantly smaller in comparison. By observing their trendlines, no substantial increase was expected for further elevated temperatures that would approach the limit.

Therefore, using the aforementioned equations and the plot below, we safely deduced that any particle could be treated as isothermal for the powder distributions used in our simulations of Alumina (**Section 3.2.1, Page 62**) and HAP (**Section 3.3.1, Page 100**).

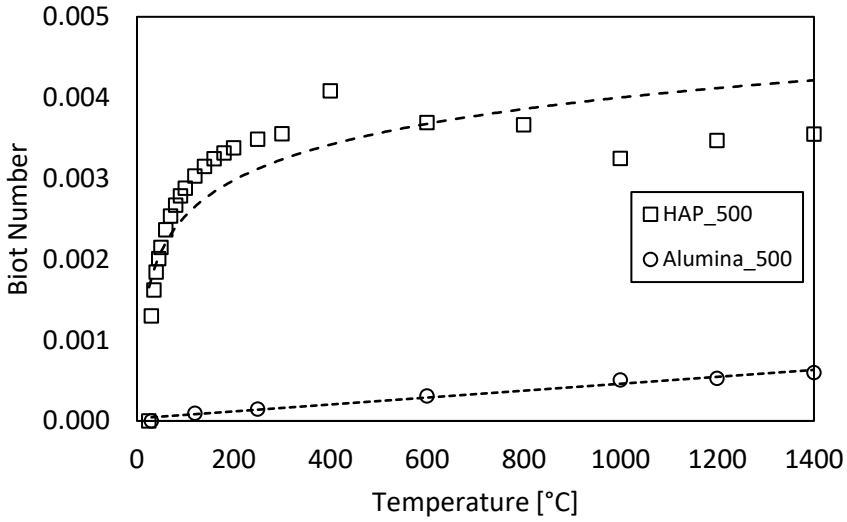
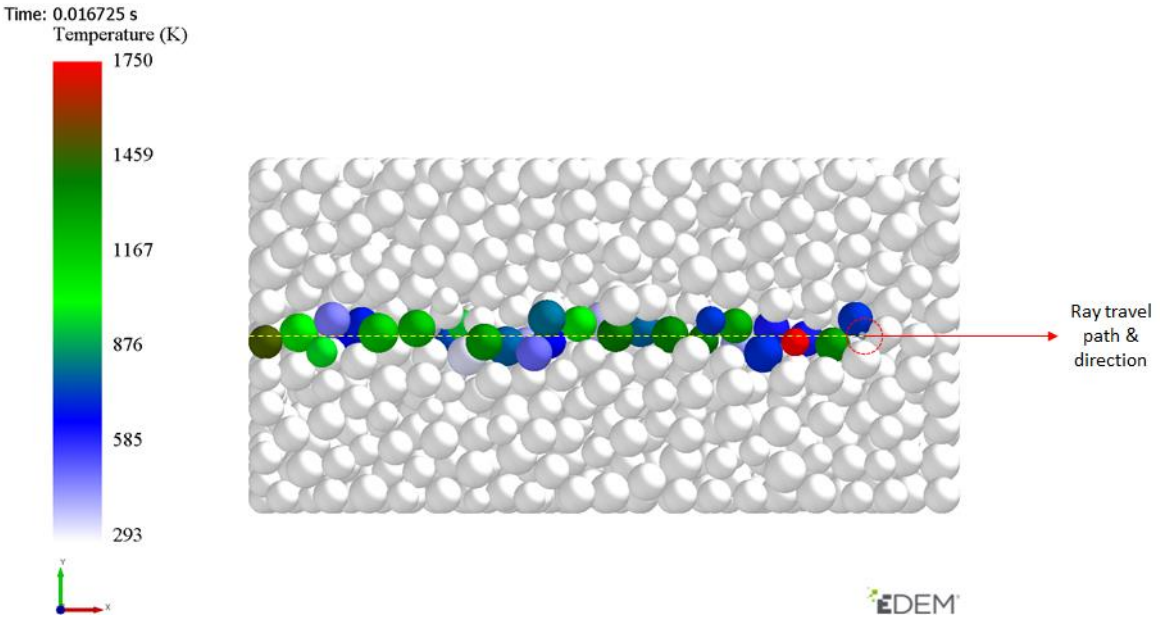


Figure 4.18. Biot number of 500 µm particle (HAP and alumina), up to 1400 °C.

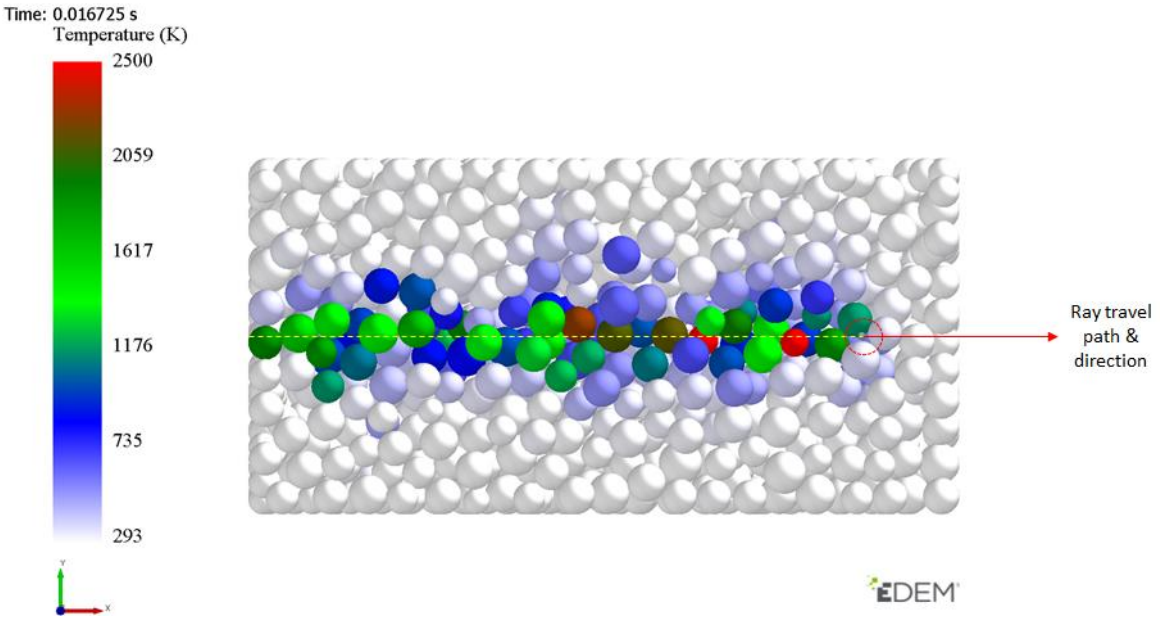
4.6. TESTING AND ANALYTICAL VALIDATION OF SIMPLIFIED CASE

In the initial tests for ray-tracing coupled DEM simulations, the effect of a single ray interacting with the powder bed particles was observed. The heat conduction between the particles was ignored to demonstrate the isolated effect of this interaction. In **Figure 4.19**, the current position of the ray has been encircled red, and the dashed yellow line shows the travel path. In the first case, no scattering effect (due to reflections and/or refractions) was considered (**Figure 4.19-a**). Therefore, the only particles intercepted by the ray's path gained energy. The particles below the top layer can also be seen as gaining energy where cavities are present. It can be observed that no other particle gains energy. The temperature legends given on the figures may be disregarded, as they do not carry any significance for this simplified test.

While in the second case, the scattering effect was considered (**Figure 4.19-b**). In contrast to the first case, it can be observed that the particles outside the intercepting path also gained energy. It was due to subsequent reflections and/or refractions of the given ray.



(a)



(b)

Figure 4.19. Single ray's interaction with the powder particles: **(a) Without** the scattering effect (due to reflections and/or transmissions), where the particles intercepted by the ray travel path gain energy only; **(b) With** the scattering effect, where the particles other than being intercepted by the ray travel path gained energy as well.

4.7. SIMULATIONS SETUP AND PARAMETERS

4.7.1 SIMULATION SETUP IN FEM DOMAIN

The FEM simulations were set up and solved using Abaqus CAE. The subroutines to mathematically define the volumetric heat flux for moving laser source, using **Equation (4.5)**, were written in Fortran (**Appendix B, Page 227**). A cuboidal body of sides 1.0 mm × 0.5 mm × 0.2 mm represented the powder bed, with a mesh size of 10 µm, as illustrated following (**Figure 4.20**):

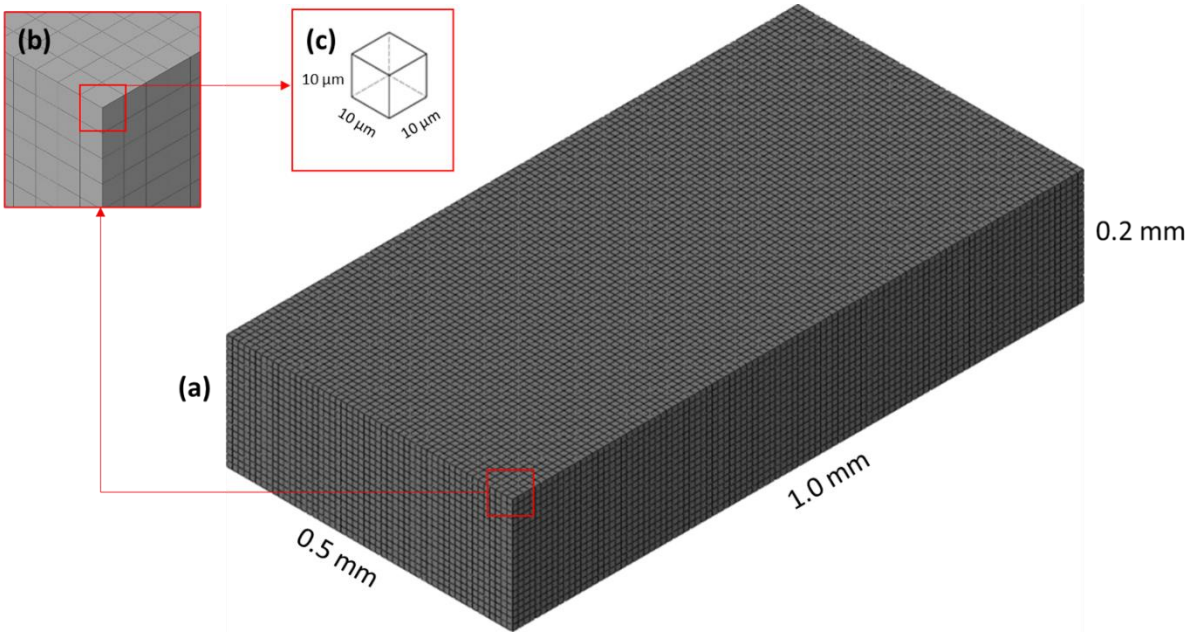


Figure 4.20. Illustration of **(a)** powder bed mesh in FEM, **(b)** a close-up of mesh, and **(c)** the cubical mesh element.

The laser was scanned over the top surface of the powder bed in a single track, and the average temperature was recorded within the square region of 100 µm (**Figure 4.21**). It was set common for both FEM and DEM domains (discussed in the next chapter). The results for FEM have been discussed, along with DEM, in the next chapter. It was done to demonstrate the comparative analysis of the implementation of laser scan simulations in two different domains.

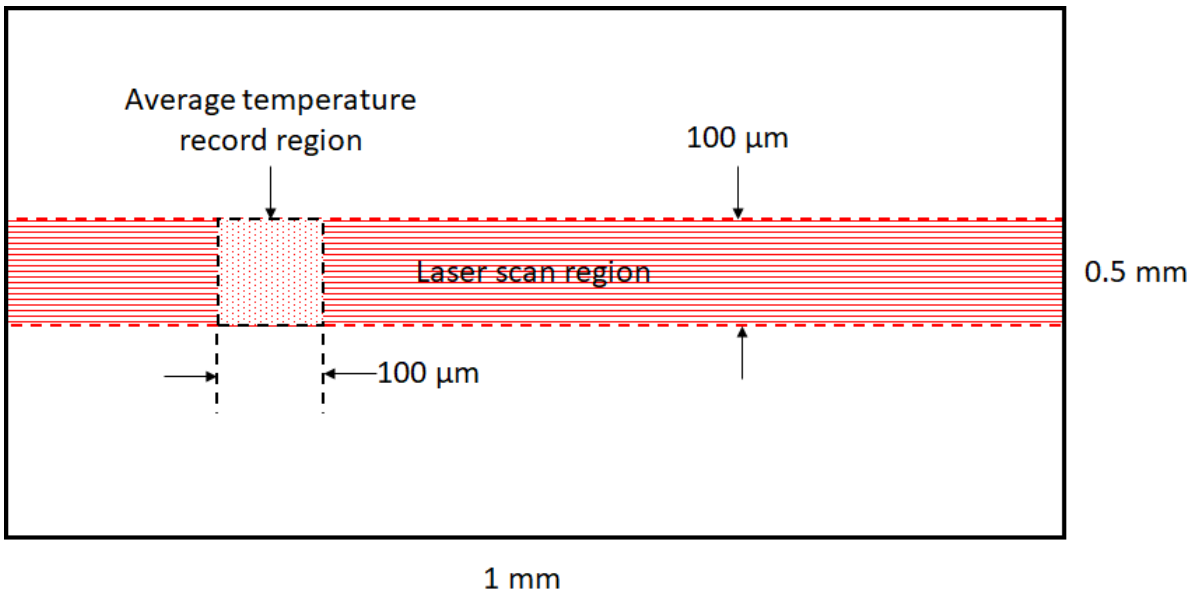


Figure 4.21. Illustration of laser scan and temperature record region on top surface of the powder bed.

4.7.2 SIMULATION SETUP IN DEM DOMAIN

Similar to the FEM setup (**Section 4.7.1**), a domain of sides 1.0 mm × 0.5 mm × 0.2 mm was considered (**Figure 4.22-a**). The discrete spherical particles, with 3 different packing densities (PD) of 48%, 54% and 75%, of both uniform size distribution with 50 µm diameter, and non-uniform size distribution with a mean diameter of 50 µm and standard deviations (SD) of 0.1 and 0.2, were generated within this domain. The discrete particles were produced in a larger volume than the given shell and then compressed to 0.2 mm to attain the desired packing density. In the case of non-uniform particles, it can further be observed from **Figure 4.22-b** and **Figure 4.22-c** that a larger number of smaller particles were used as void fillers to obtain the PD of 75%.

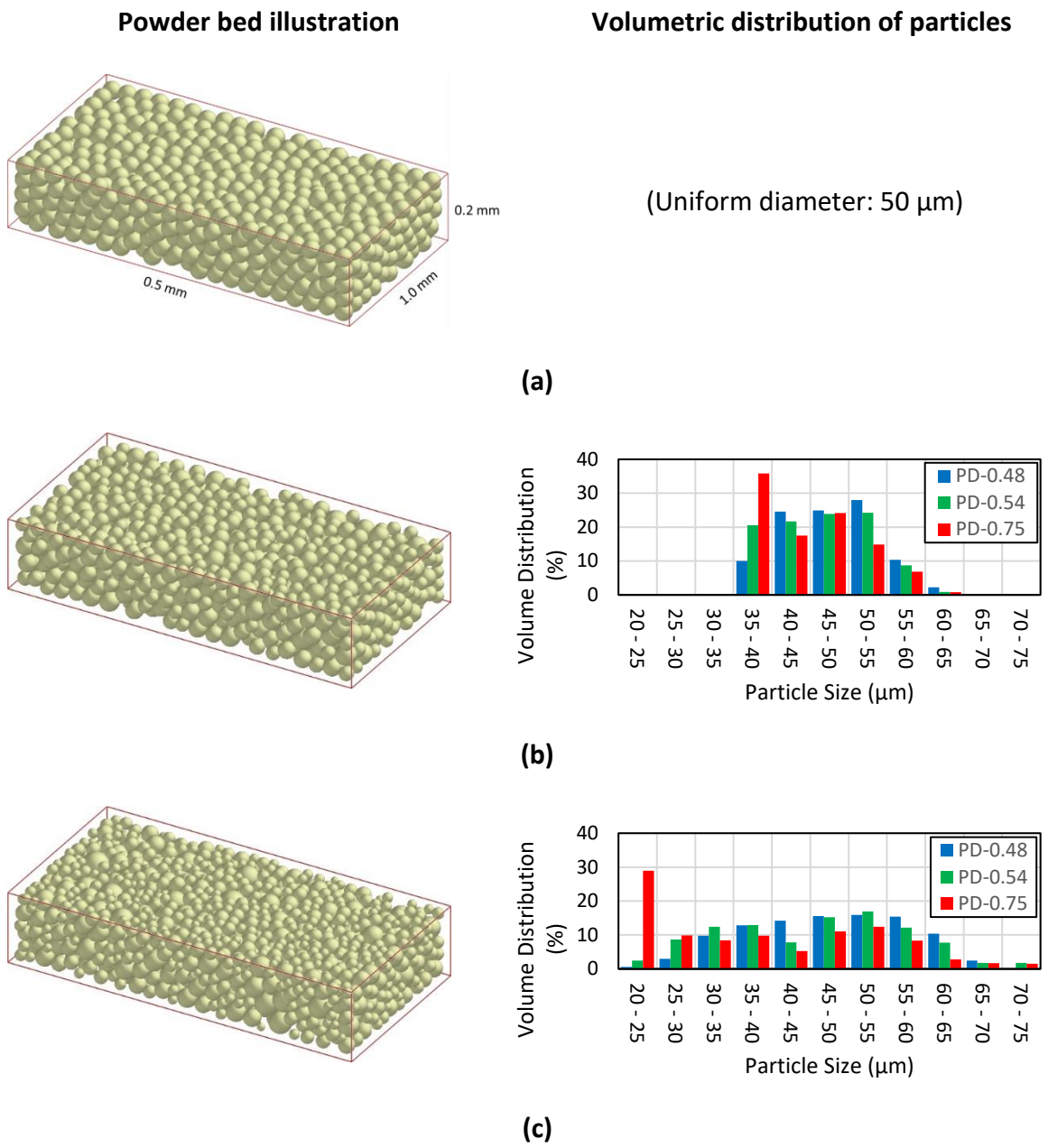


Figure 4.22. Illustration of the **(a)** uniform sized (50 µm) powder bed generated in DEM, within an enclosed shell/domain represented by red wireframe, **(b)** volumetric distribution plot and powder bed visualization for non-uniform sized particles with SD 0.1, and **(c)** volumetric distribution plot and powder bed visualization for non-uniform sized particles with SD 0.2.

4.8. LASER AND MATERIAL PARAMETERS

The following laser parameters (**Table 4.7**) were consider for both FEM and DEM domain:

Table 4.7. Laser parameters for simulations (both FEM and DEM).

Power [W]	Scan speed [m s^{-1}]	Beam diameter [μm]	Scan length [mm]
10	0.5	100	1.0

Alumina was considered as a material in the simulations. A fixed value for **density** of alumina was considered:

3900 kg m^{-3}.

The temperature dependent **specific heat capacity** was considered, plotted as following [102]:

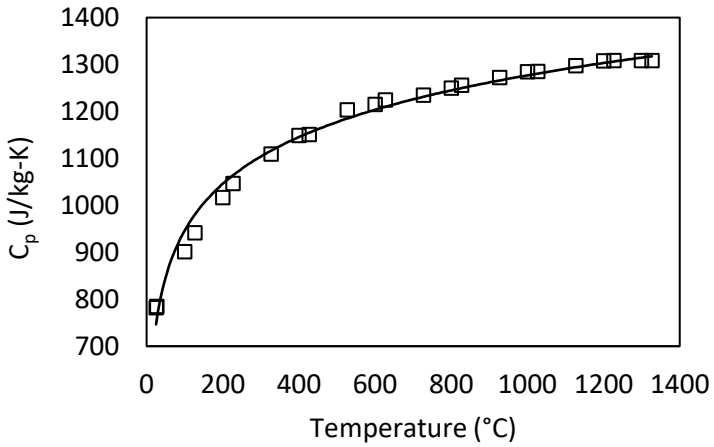


Figure 4.23. Temperature dependent specific heat capacity (constant pressure) of alumina [102].

The **thermal conductivity** of alumina is plotted as following [102]:

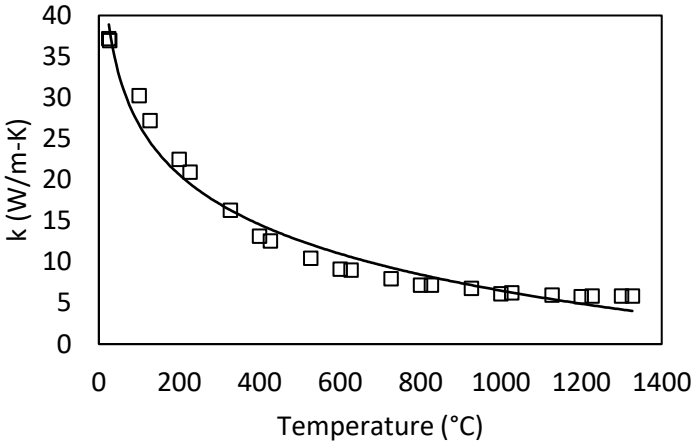


Figure 4.24. Temperature dependent thermal conductivity of alumina [102].

Furthermore, as FEM only deals with continuum material problems, the **effect of porosity** present in the powder bed on thermal conductivity was empirically assumed as following [102], with the dashed lines representing extrapolations at elevated temperatures:

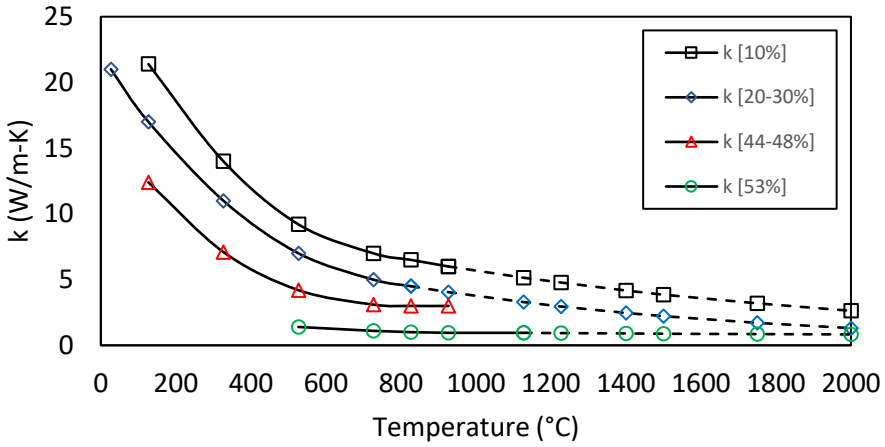


Figure 4.25. Temperature dependent thermal Conductivity of porous Alumina [102].

4.9. RESULTS AND DISCUSSION

The two methodologies were compared to simulate the laser heating of alumina's powder bed surfaces. The first method was developed using Abaqus and Fortran to define the Gaussian heat-flux distribution in FEM. In contrast, the second method was based on ray tracing coupled with discrete particles in DEM, as discussed in this chapter. Thermal losses on the top surface were neglected, as the simulation run time was very brief to have any significant effect. EDEM default thermal conduction model was used.

4.9.1 CO_2 LASER VS Nd:YAG LASER

At first, the thermal simulations using the parameters of the CO_2 laser were carried out. For packing density (PD) of 48%, the graphical results in FEM and DEM (all size distributions) were comparable (**Figure 4.26-a**). A melt pool (colour coded – red) was generated that stretched along the scan path, both in FEM and DEM domains (**Figure 4.27-b**). An almost immediate melting in the record region (**Figure 4.21**, **Page 148**) can be observed.

The case of 54% PD was comparable to 48% PD in terms of induced thermal gradients (**Figure 4.26-b**). However, the FEM simulations demonstrated that the melt pool is partially formed for the laser-travelled path, while the DEM simulations showed a stretched melt pool throughout its course (**Figure 4.27-c**). Furthermore, in the first two cases, the given size distributions (**Figure 4.22**) have no significant effect on thermal gradients.

For 75% PD case, the usual melt pool was generated in both FEM and DEM simulations (**Figure 4.27-d**), except for the powders with the widest size distribution of SD 0.2. This could have been due to the powders' dense packing, leading to greater heat dissipation. The effect of melt pool dynamics was not considered in this study. Therefore, a uniform temperature field represented the melted region.

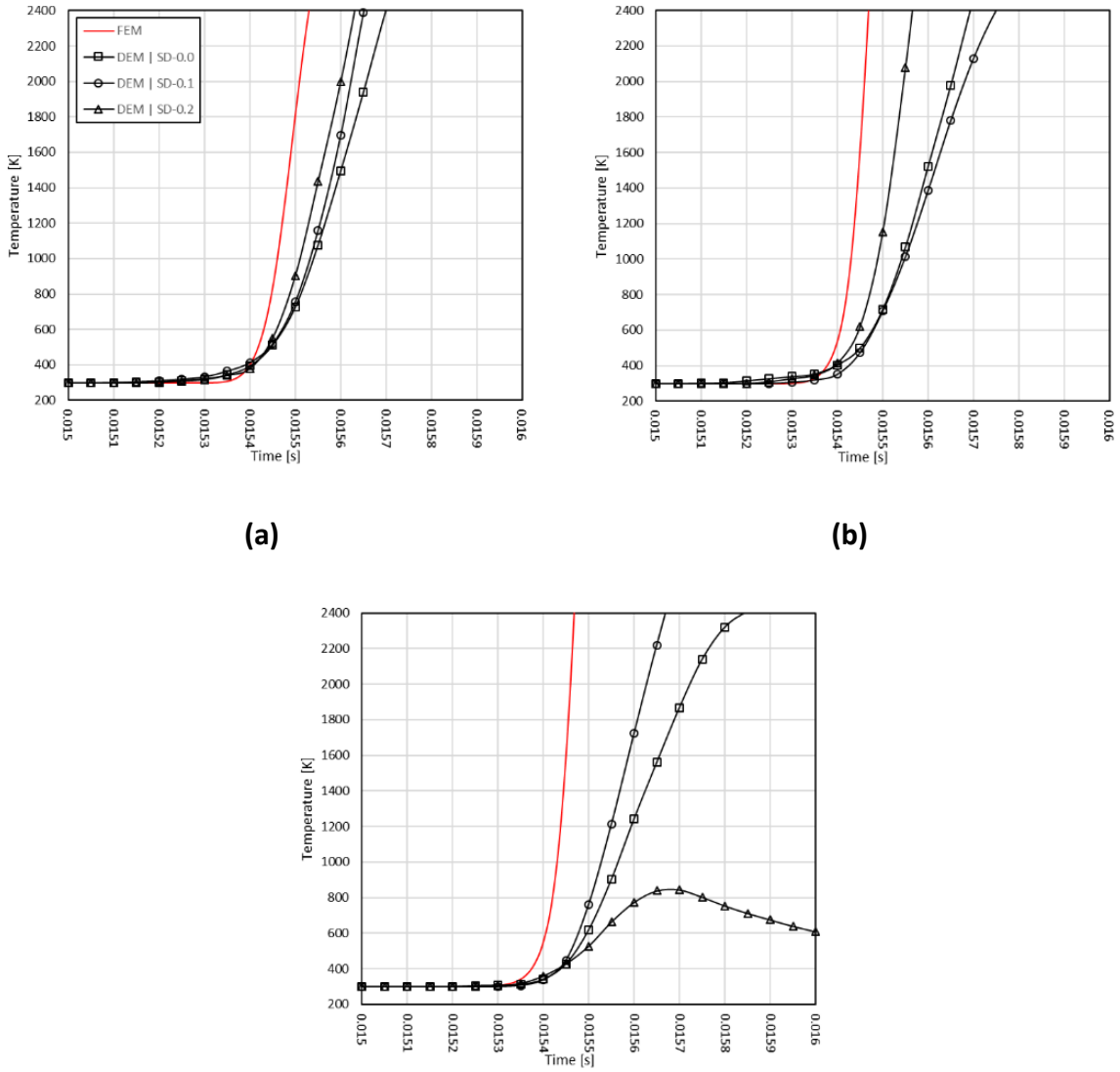
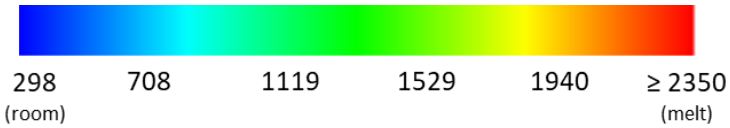


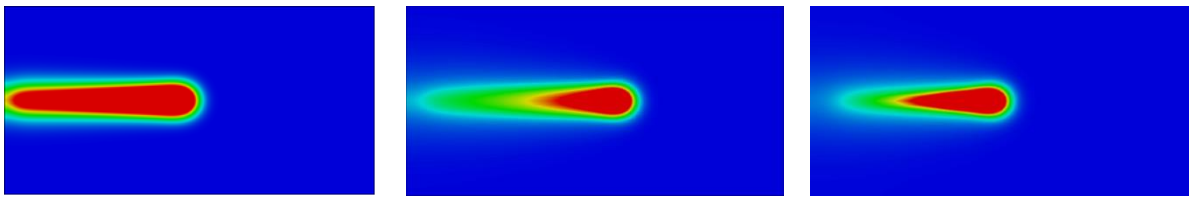
Figure 4.26. Graphical plots for average temperature change in laser scan area (illustrated in **Figure 4.21**, **Page 148**), representing the effect of interaction with **CO_2 laser**, for **(a)** 48% packing density, **(b)** 54% packing density, and **(c)** 75% packing density.

(a). Temperature legend:

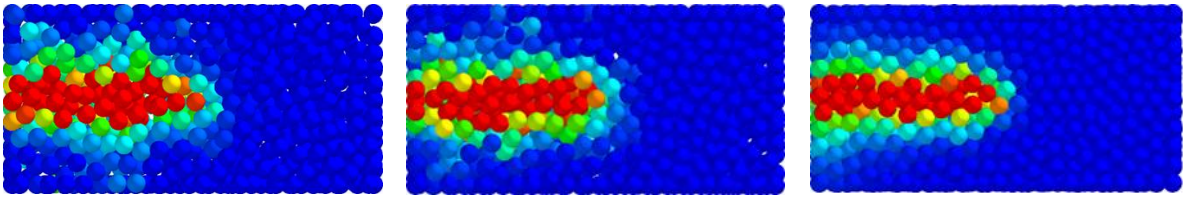


(b). PD 48%: **(c). PD 54%:** **(d). PD 75%:**

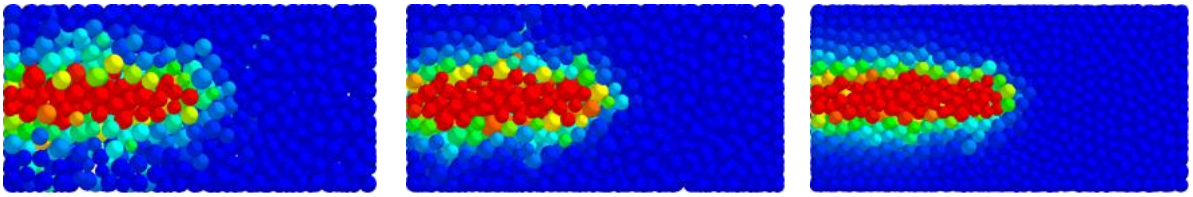
FEM (Uniform particle size distribution):



DEM (Uniform particle size distribution):



DEM (Gaussian particle size distribution, with standard deviation 0.1):



DEM (Gaussian particle size distribution, with standard deviation 0.2):

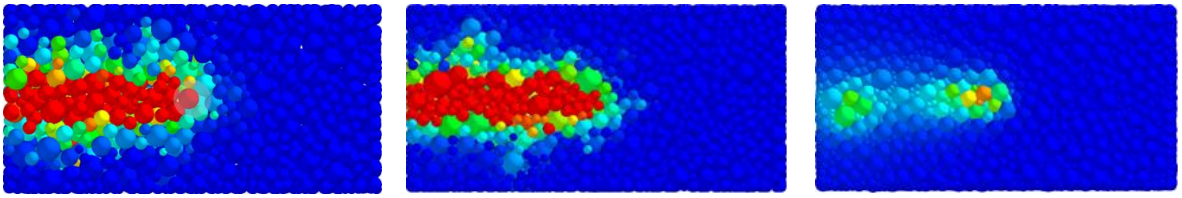


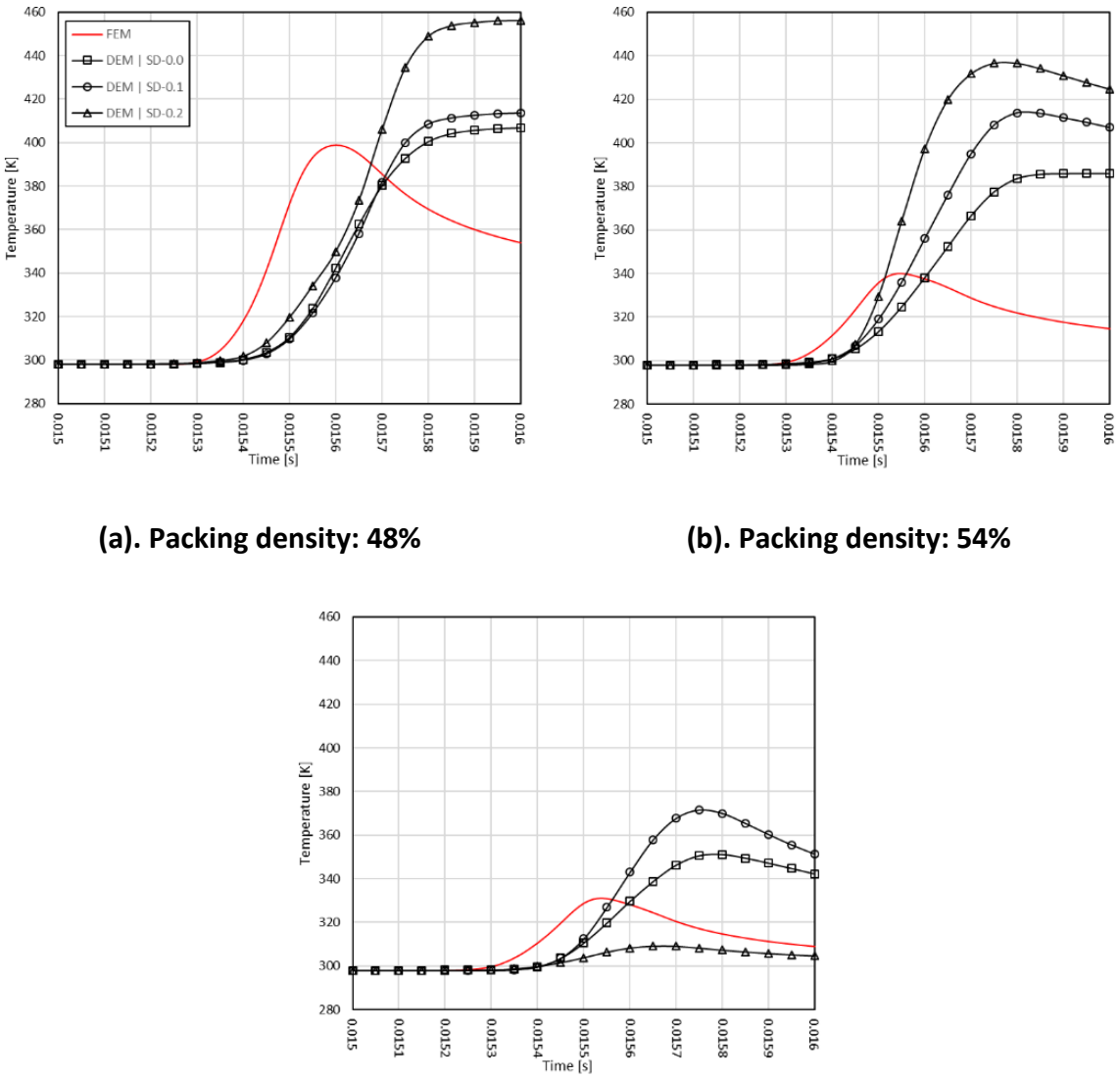
Figure 4.27. Comparison of surface temperature fields resulting from CO_2 laser scans, in FEM and DEM domains, with **(a)** temperature legend, for **(b)** 48% packing density, **(c)** 54% packing density, and **(d)** 75% packing density.

In contrast to the CO_2 laser, the lower temperature rise in the powder bed was observed for Nd:YAG laser (**Figure 4.28**), for all cases of packing densities, for the same parameters (**Table 4.7**). It was due to lower absorption of Nd:YAG laser wavelength (**Table 4.3**). In general, it was observed that the peak temperatures started to decrease with increasing packing densities because closely packed particles dissipated more heat.

For the first case of 48% PD, it was observed that in FEM domain, the peak temperature was comparable to uniform-sized and 0.1 SD powders in DEM. In comparison, 0.2 SD powder size attained a higher peak temperature (**Figure 4.28-a**). This specific distribution contains a larger number of small-sized powder particles (**Figure 4.22-c**) that would gain higher temperatures for the given energies.

As the PD was increased to 54%, a broader difference in peak temperatures was observed between DEM and FEM domains (**Figure 4.28-b**). As Nd:YAG wavelength is highly reflected by the alumina particles (**Table 4.3**), the (ray-traced) back-reflections from other particles lead to higher temperature gain than observed in the FEM domain.

In the last case of 75% PD (**Figure 4.28-c**), the temperatures in the FEM domain approximately represent the average of temperatures corresponding to different size distributions in the DEM domain. The heat distribution from both domains is also similar (**Figure 4.29-d**).



(a). Packing density: 48%

(b). Packing density: 54%

(c). Packing density: 75%

Figure 4.28. Graphical plots for average temperature change in laser scan area (as illustrated in **Figure 4.21**, **Page 148**), representing the effect of interaction with **Nd:YAG laser**, for **(a)** 48% packing density, **(b)** 54% packing density, and **(c)** 75% packing density.

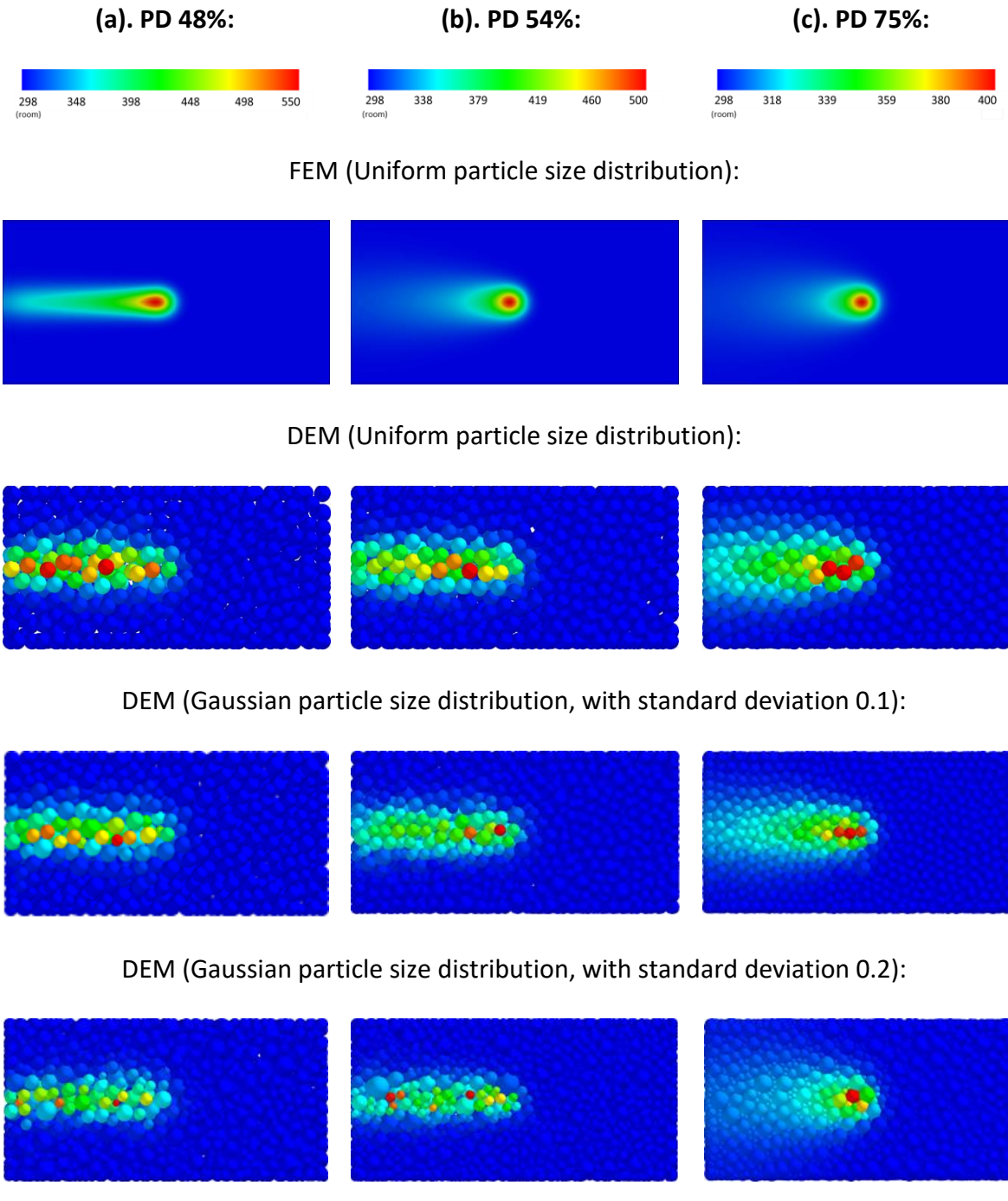


Figure 4.29. Comparison of surface temperature fields resulting from Nd:YAG laser scans, in FEM and DEM domains, with separately defined temperature legends, for **(b)** 48% packing density, **(c)** 54% packing density, and **(d)** 75% packing density.

4.9.2 FEM VS DEM DOMAIN

The difference in results for the two domains can be explained with the combination of the following reasons:

(1). In ray tracing, laser energy scattering could be accounted. This effect was observed in the DEM simulations. The particles along the longer edge of the powder bed were also seen to be gaining energy (leading to a rise in their temperature). Generally, a wider temperature distribution field was formed. In contrast, the results from FEM indicated a thinner or localized temperature distribution field, i.e., generally restricted within the laser scan region.

(2). In addition to the rays scattering effect, the DEM domain could also define non-uniformly distributed powder beds. Due to this, the localized temperature gain would depend on particle size interacting with the ray.

(3). In the FEM domain, the temperature distribution field was always symmetrical comet-shaped due to thermal conductivity being calculated between two nodes, assuming a uniform and continuum material (solid). In comparison, the calculations for thermal conductivity were very different in the DEM domain, as they accounted for particles' sizes, positions, distribution, and contact areas. Therefore, they were almost always different for any given region within the powder bed.

(4). However, as the packing density was increased in the DEM, the temperature distribution started to become symmetrical because closely packed particles began to resemble a continuous solid material. Therefore, the results, especially in the case of Nd:YAG laser, became comparable for both domains. Conversely, the field still appears to be wider for DEM, which was due to the rays scattering effect.

(5). In DEM, for the densely packed powder bed (i.e., 75%), the size distribution in the powder bed displayed a prominent effect in heat dissipation. This can especially be observed for CO_2 laser (**Figure 4.26-c**) (and also for Nd:YAG (**Figure 4.28-c**)), where the widest distribution of size (i.e., SD 0.2) powders exhibited an anomalous behaviour by attaining the significantly lower temperatures than their FEM and rest of the DEM counterparts. As discussed in **Section 2.5.2**, the thermal conductivity in DEM is calculated differently. It depends on a number of factors, such as Hertz-Mindlin interparticle forces and equivalent radii of the interacting particles, which could play an influential role in broader size distributions. In contrast, FEM cannot account for such elements and takes a more direct approach to calculate the node-to-node heat conduction, which only depends on the temperature difference.

THIS PAGE WAS INTENTIONALLY LEFT BLANK

5. SELECTIVE LASER SINTERING

Highlights:

1. Numerical contact model for sintering of ceramics

2. Mass scaling for numerical stability of DEM simulations

3. Combination of sub-models for production of a limited-scale SLS model

5.1. INTRODUCTION

This chapter specifically focuses on developing sintering mechanism, customized heat conduction and heat loss models. These models were combined with ray tracing-based laser heating (**Chapter 4**), to produce a limited-scale selective laser sintering model.

A stopping force algorithm was introduced to eliminate residual forces and torques present within the 'settled' particles. It was done to ensure that they did not interfere with the calculations of sintering. The sintering contact model required very small timesteps for a stable solution, which led to potentially unrealistic computation times. Therefore, this important issue of numerical stability at larger timesteps was addressed by mass scaling method (**Section 2.4.3, Page 33)**.

Furthermore, the conditions and application sequences of the physical and thermal contact models were set. For instance, during settlement and compression of particles only Hertz-Mindlin force was applied. Once the particles had settled, the ray tracing heating, thermal conduction and losses were introduced. The sintering force was set at the end of each timestep, on the condition that interacting particles have surpassed a certain sintering temperature.

5.2. SINTERING CONTACT USING DEM

5.2.1 STOPPING FORCE AND TORQUE

After the particles had settled down in the powder bed, residual translational and angular velocities were still present in them. It was due to the gravity-induced acceleration that initially assisted in dropping the particles into the simulation box and a counteracting Hertz-Mindlin (HM) force between them that brought the overall particulate system to an equilibrium state. As discussed before, HM was disabled when the sintering temperature was reached, but the residual relative velocities were still present between the interacting particles. So, they were brought down to zero to avoid the presence of initial velocities when the sintering force was applied, as they could hinder the free sintering mechanism. However, EDEM's API (Application Programming Interface) does not allow manually overriding particles' velocities. Therefore, each particle in the powder bed was subjected to a stopping force before implementing the laser scan code. This force brought their velocities down to zero, and is given by:

$$F_{STOP} = -m_p \frac{\Delta v_p}{\Delta t} \tag{5.1}$$

where m_p is the particle's mass, Δv_p is the amount of its translational velocity being brought to rest in the timestep of size Δt. Similarly, a torque is applied to stop the angular velocity, as follows:

$$\tau_{STOP} = -I_p \frac{\Delta \omega_p}{\Delta t} \tag{5.2}$$

where $\Delta \omega_p$ is the amount of angular velocity of (spherical) particle with moment of inertia (I_p) being brought to rest, given by:

$$I_p = \frac{2}{5} m_p r_p^2 \tag{5.3}$$

where m_p is the particle's mass and r_p is its radius. This algorithm brings the particles to rest in one timestep. Before the start of this timestep, gravity was also set to 0 so that we did not have to apply additional counteracting force. The methodology to implement this force is outlined in **Algorithm 5.1**. The typical result of the application of this code can be observed in **Figure 5.1** and **Figure 5.2**.

```
CALCULATE STOPPING_FORCE_TORQUE:

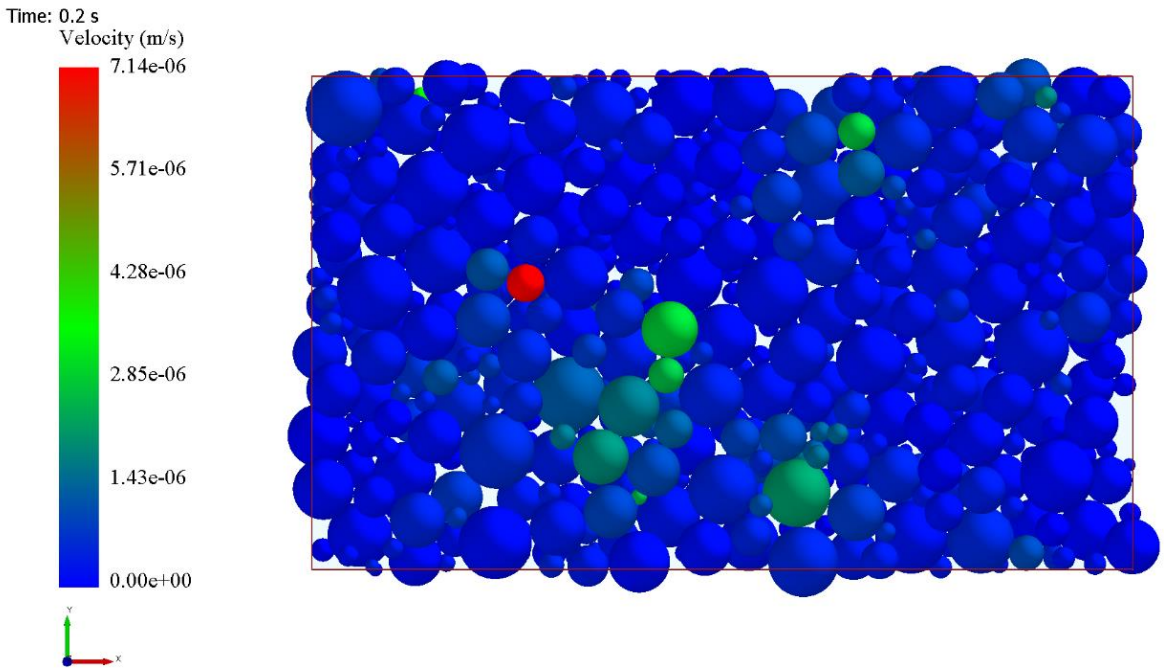
RETRIEVE TIMESTEP_SIZE;
RETRIEVE PARTICLE_MASS;
RETRIEVE PARTICLE_RADIUS;
RETRIEVE PARTICLE_VELOCITY_X, PARTICLE_VELOCITY_Y, PARTICLE_VELOCITY_Z;
RETRIEVE PARTICLE_ANGVEL_X, PARTICLE_ANGVEL_Y, PARTICLE_ANGVEL_Z;

FOR ALL PARTICLES -> CALCULATE STOPPING_FORCE:
FORCE_STOP_X = (PARTICLE_MASS * PARTICLE_VELOCITY_X) / TIMESTEP_SIZE;
FORCE_STOP_Y = (PARTICLE_MASS * PARTICLE_VELOCITY_Y) / TIMESTEP_SIZE;
FORCE_STOP_Z = (PARTICLE_MASS * PARTICLE_VELOCITY_Z) / TIMESTEP_SIZE;

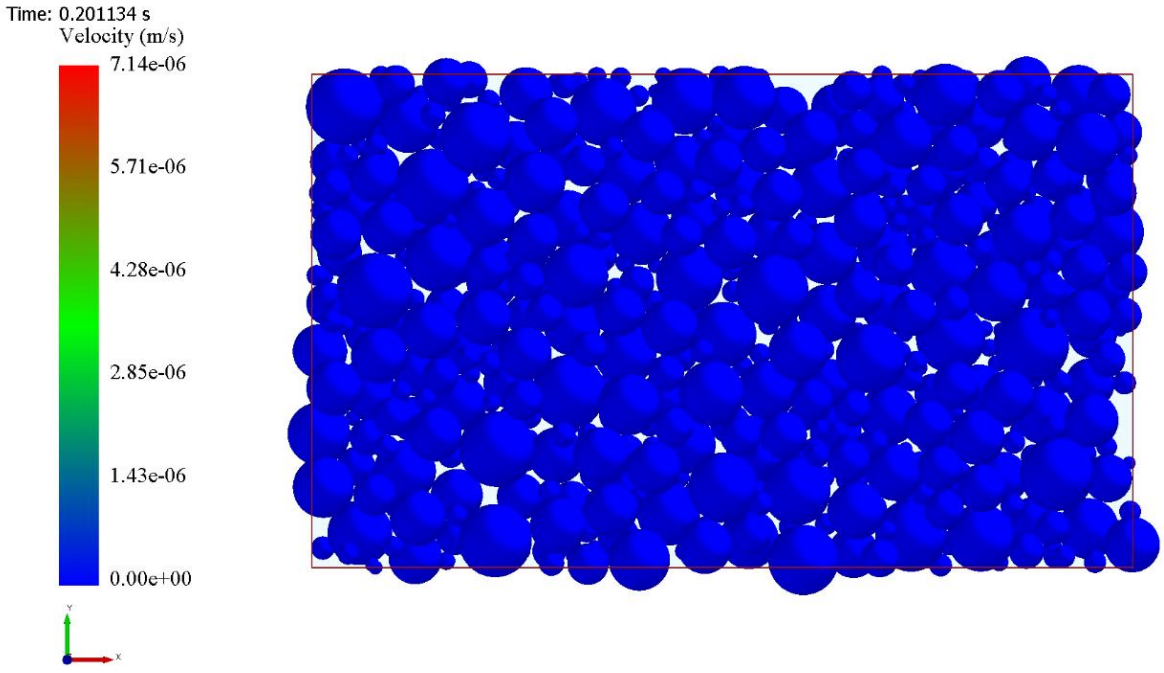
FOR ALL PARTICLES -> CALCULATE MOMENT_OF_INERTIA:
PARTICLE_MOI = 0.4 * PARTICLE_MASS * PARTICLE_RADIUS * PARTICLE_RADIUS;

FOR ALL PARTICLES -> CALCULATE STOPPING_TORQUE:
TORQUE_STOP_X = (PARTICLE_MOI * PARTICLE_ANGVEL_X) / TIMESTEP_SIZE;
TORQUE_STOP_Y = (PARTICLE_MOI * PARTICLE_ANGVEL_Y) / TIMESTEP_SIZE;
TORQUE_STOP_Z = (PARTICLE_MOI * PARTICLE_ANGVEL_Z) / TIMESTEP_SIZE;
```

Algorithm 5.1. Pseudo algorithm for calculation of stopping force and torque to bring the particle to complete rest. The corresponding C++ code can be found in **Appendix-G (Page 253)**.

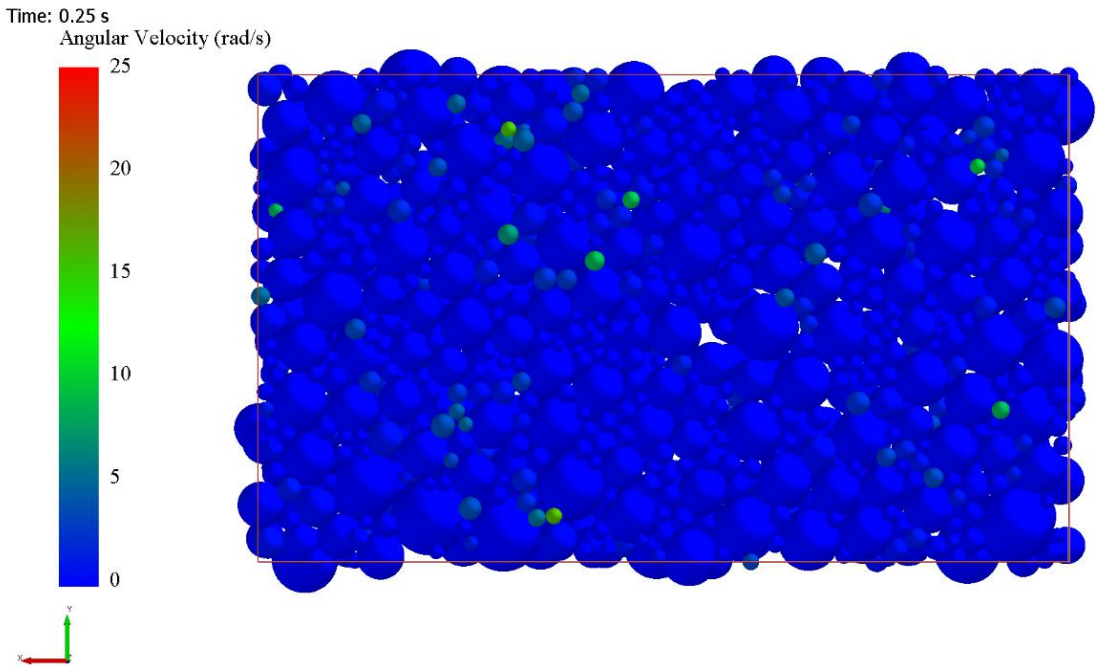


(a)

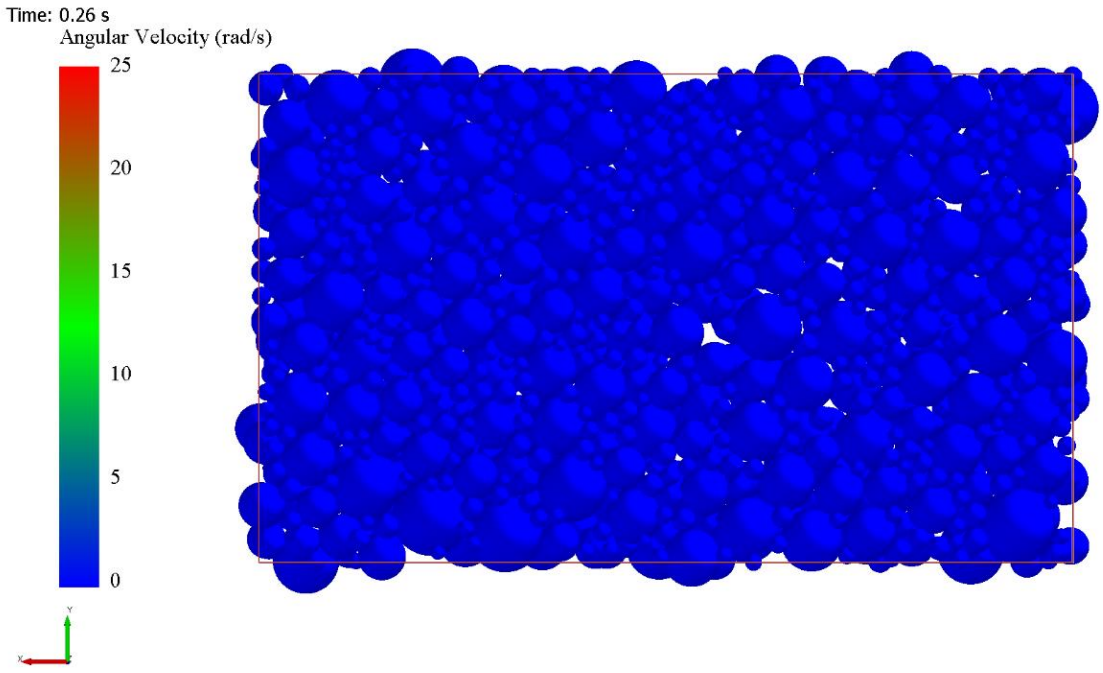


(b)

Figure 5.1. Illustration of **translational velocity** distribution in the powder bed after they have settled. **(a)** Presence of residual velocities in a number of particles; **(b)** Velocities of all the particles reduced to zero after applying the Stopping Force algorithm.



(a)



(b)

Figure 5.2. Illustration of **angular velocity** distribution in the powder bed after they have settled. **(a)** Presence of residual angular velocities in a number of particles; **(b)** Angular velocities of all the particles reduced to zero after applying the Stopping Force algorithm.

5.2.2 STRESS RELAXATION

Before the introduction of the sintering contact force, the compressive stresses induced by the Hertz-Mindilin (HM) forces between the particles, represented by the overlap (δ_{actual}) between them (**Figure 5.3**), were relaxed (i.e., removed). It was also done during the sintering process in each time step. So, when the interacting particles' average temperature had exceeded the sintering temperature, those compressive stresses would again be removed to avoid influence over the numerical calculation of the sintering force.

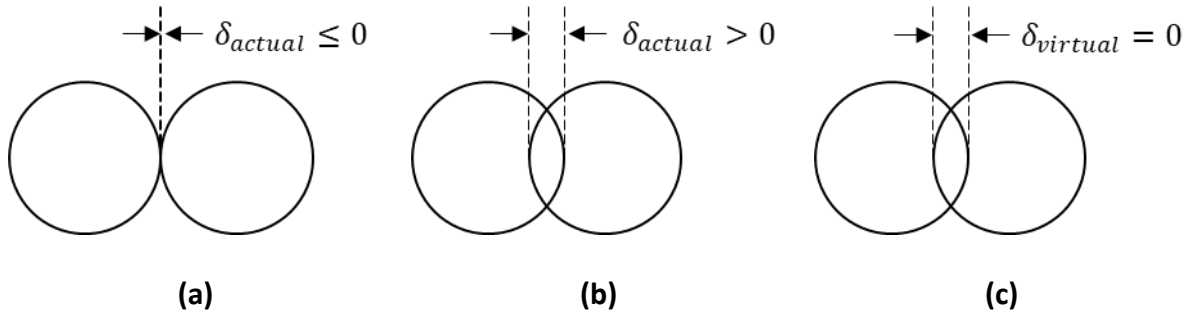


Figure 5.3. Particle overlaps: **(a)** Point or no contact between particles, numerically represented as $\delta = 0$ or $\delta < 0$ respectively, **(b)** particles having actual overlap, considered as in compressive stress state, and **(c)** particle are still under compression, i.e., have an existing physical overlap, but their overlap is numerically overridden to 0 and passed to Hertz-Mindilin part of the algorithm while keeping their original positions unaffected.

Both sintering and HM were dependent on the particles overlap (δ) input, which meant they could influence each other resulting in unstable simulation runs. Also, a larger overlap occurring during the implementation of the sintering force led to a very high repulsive force due to the in-parallel calculation of HM, sending the particles into an unreal explosive trajectory. The small overlaps between the particles could still be visually observed in the GUI (graphical user interface), as their positions were unaffected. However, those overlaps ($\delta_{virtual}$) were virtually offset to zero (**Figure 5.3-c**), which were used to calculate the inter-particle forces, and then passed on to the HM part of the code. It resulted in zero force (or compressive stress) from HM.

5.2.3 SINTERING, REPULSIVE AND BONDING FORCES

The algorithm and code for the sintering contact model discussed in **Section 2.4.2** (**Page 31**) was developed and implemented. It worked under the conditions that the average temperature of the interacting particles had reached above the sintering temperature. However, the overall numerical scheme followed some additional forces, including repulsion and bonding.

The sintering force caused the centres of the interacting particles to move closer, increasing their overlap and contact angle. However, when the contact angle exceeded the dihedral angle, another repulsive force was introduced to reduce their overlap to the point where the contact angle was equal to the dihedral angle and to keep the equilibrium. The repulsive force is given by [55]:

$$F_{REP} = K_n \delta_n^{REP} - \beta_n v \tag{5.4}$$

where δ_n^{REP} is the normal overlap in the repulsion region, v is the approach velocity between the particles, and K_n is the contact stiffness given by [55]:

$$K_n = 2E^* \sqrt{R^* \delta_n^{HM}} \tag{5.5}$$

β_n is the sintering parameter given by [55]:

$$\beta_n = 1.8257 \mu \sqrt{K_n m^*} \tag{5.6}$$

The parameter μ is calculated as [55]:

$$\mu = \frac{-\ln^2 e}{\sqrt{\pi^2 + \ln^2 e}} \tag{5.7}$$

After the initial repulsive force implementation, the particles were permanently bonded together. The normal and tangential forces ($F_{n,t}$) and moments ($M_{n,t}$) were set to zero at this point. They were incrementally readjusted every time step, as follows [19]:

$$\Delta F_n = -\nu_n S_n A \Delta t \tag{5.8}$$

$$\Delta F_t = -\nu_t S_t A \Delta t \tag{5.9}$$

$$\Delta M_n = -\omega_n S_t J \Delta t \tag{5.10}$$

$$\Delta M_t = -\omega_t S_n \frac{J}{2} \Delta t \tag{5.11}$$

where:

$$A = \pi R_B^2 \tag{5.12}$$

$$J = \frac{1}{2}\pi R_B^4 \tag{5.13}$$

where $\nu_{n,t}$ are the normal and tangential velocities of the interacting particles, respectively. $S_{n,t}$ are the normal and shear stiffnesses, respectively. $\omega_{n,t}$ are the normal and tangential angular velocities, respectively. Δt is the timestep size. R_B is the glue radius, set as radius of the smaller particle [19].

The bond strength was defined in terms of normal (σ_{max}) and tangential (τ_{max}) shear stresses. It would be broken with the following conditions [19]:

$$\sigma_{max} < -\frac{F_n}{A} + \frac{2M_t}{J} R_B \tag{5.14}$$

$$\tau_{max} < -\frac{F_t}{A} + \frac{M_n}{J} R_B \tag{5.15}$$

The conditions and flow of the forces are given in **Figure 5.4**. Furthermore, the pseudo algorithm that helped develop the C++ code is given in **Algorithm 5.2**.

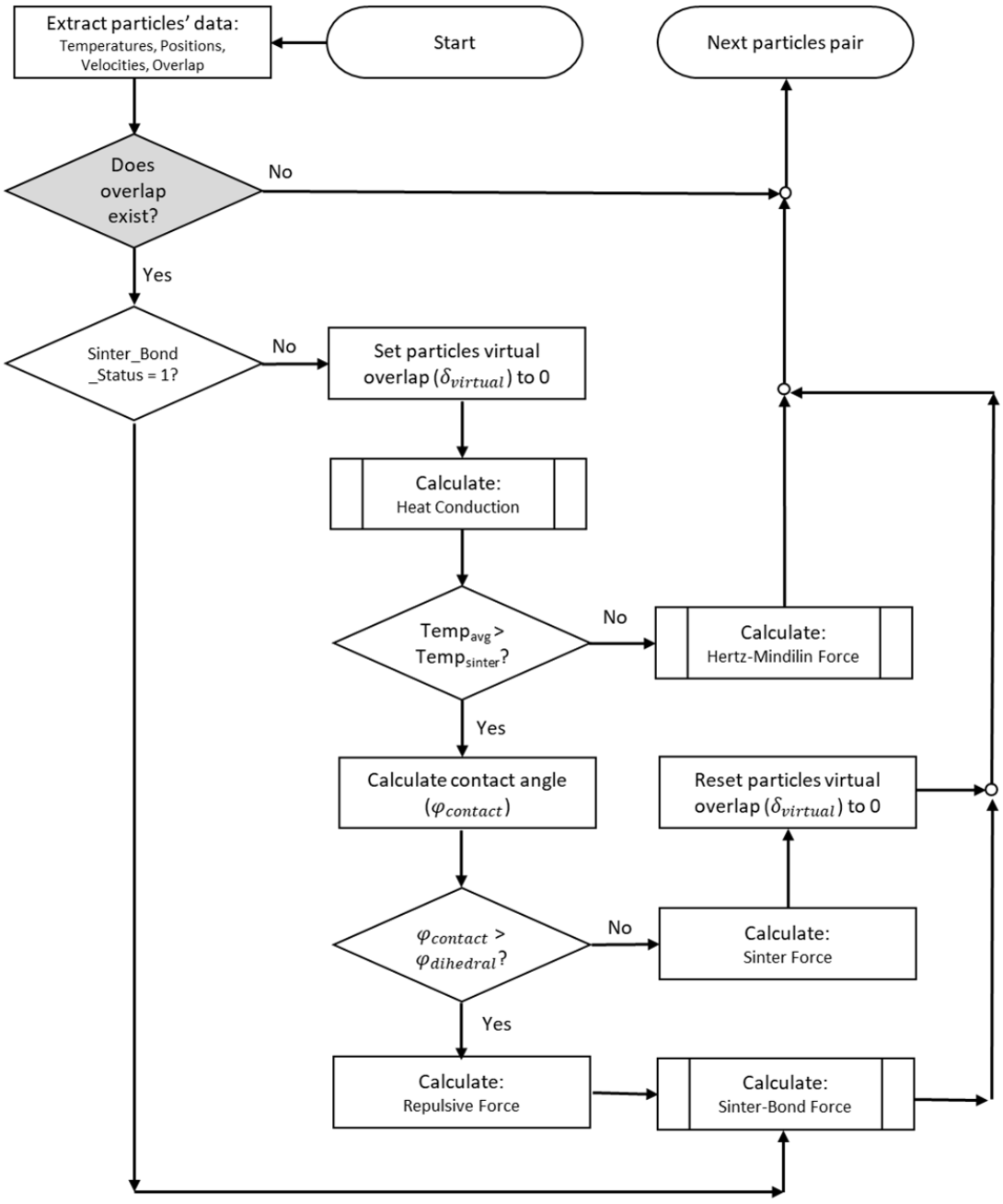


Figure 5.4. Flowchart for determination and calculation of sintering and bonding force between the given two interacting particles.

```
CALCULATE SINTERING_FORCE:

RETRIEVE P1_TEMPERATURE, P2_TEMPERATURE;
RETREIVE P1_POSITION(X,Y,Z), P2_POSITION(X,Y,Z);
RETREIVE P1_VELOCITY, P2_VELOCITY;
RETREIVE MELTBOND_STATUS(P1,P2);

CALCULATE PARTICLES_OVERLAP(P1_POSITION, P2_POSITION);

IF (PARTICLES_OVERLAP <= 0)
        EXIT ALGORITHM
END IF

CALCULATE AVG_TEMPERATURE(P1_TEMPERATURE, P2_TEMPERATURE);
CALCULATE CONTACT_ANGLE(P1_POSITION, P2_POSITION);

IF (AVG_TEMPERATURE < SINTERING TEMPERATURE && MELTBOND_STATUS == 0) THEN
        CALCULATE HERTZ_MINDILIN(P1,P2)

ELSE IF (AVG_TEMPERATURE >= SINTERING_TEMPERATURE &&
        AVG_TEMPERATURE < MELTING_TEMPERATURE &&
        SINTERBOND_STATUS == 0) THEN

        IF (CONTACT_ANGLE <= DIHEDRAL_ANGLE)
                CALCULATE SINTERING_FORCE(P1,P2);
        ELSE
                CALCULATE REPULSION_FORCE(P1,P2);
        END IF

ELSE IF (CONTACT_ANGLE >= DIHEDRAL_ANGLE &&
        SINTERBOND_STATUS == 0) THEN
        CALCULATE MELTBOND_FORCE(P1,P2);
ELSE
        CALCULATE MELTBOND_RETENTION(P1,2P);
END IF
```

Algorithm 5.2. Pseudo algorithm for calculation of relevant interaction forces between the particles. The corresponding C++ code can be found in **Appendix-H (Page 255)**.

5.2.4 MASS SCALING OF PARTICLES

The mass of the particles was scaled up by factor 10^{14} to achieve a stable numerical solution with a reasonable timestep. However, it would also be important to note that when particle mass has been scaled up, the temperature gained by it would be scaled down by the same factor for the given energy input. Therefore, the same scale-up factor (S_{mass}) was applied to the specific heat (c_p) equation in the code to ensure that the temperature change remained unaffected:

$$\Delta T = \frac{e}{mc_p} \rightarrow \Delta T = S_{mass}\frac{e}{mc_p} \tag{5.16}$$

where e is the laser energy (in our case), m is the particle mass, and ΔT is the temperature change.

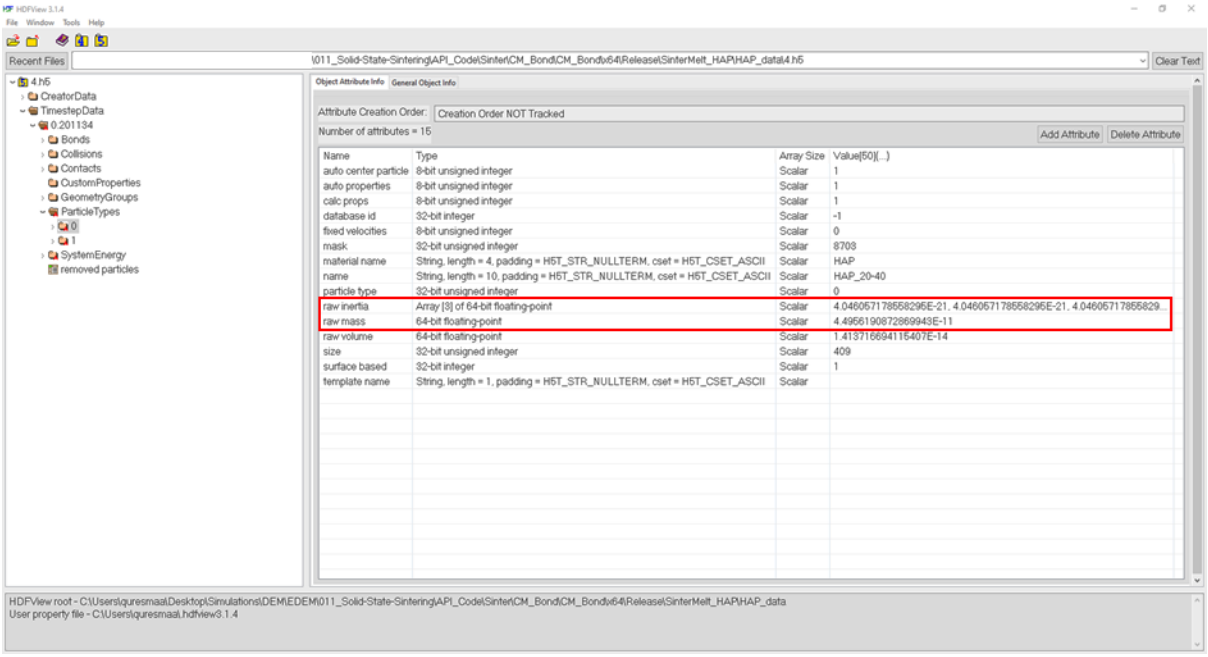
The mass scaling during the simulation run in EDEM is generally not a straightforward process. EDEM generates h5 file format for each timestep to store data for each time step. The material properties applied during the first timestep (or initialization) are carried throughout the simulation run. Therefore, any changes made in those properties within the GUI (Graphical User Interface) in the subsequent timesteps (i.e., during the simulation) are not updated in the stored files. It could be a preventive measure from the developers to avoid any anomalies or numerical instabilities to the dynamic particulate system introduced by the sudden change of properties.

However, we needed the new scaled-up density for the settled particles only, which would not introduce any abrupt change to the system. Since the unrealistically large inertia of the falling microparticles could lead to unpredictable results. Therefore, we allowed the particles to settle using their standard (unscaled) material properties. Then, we manually modified the

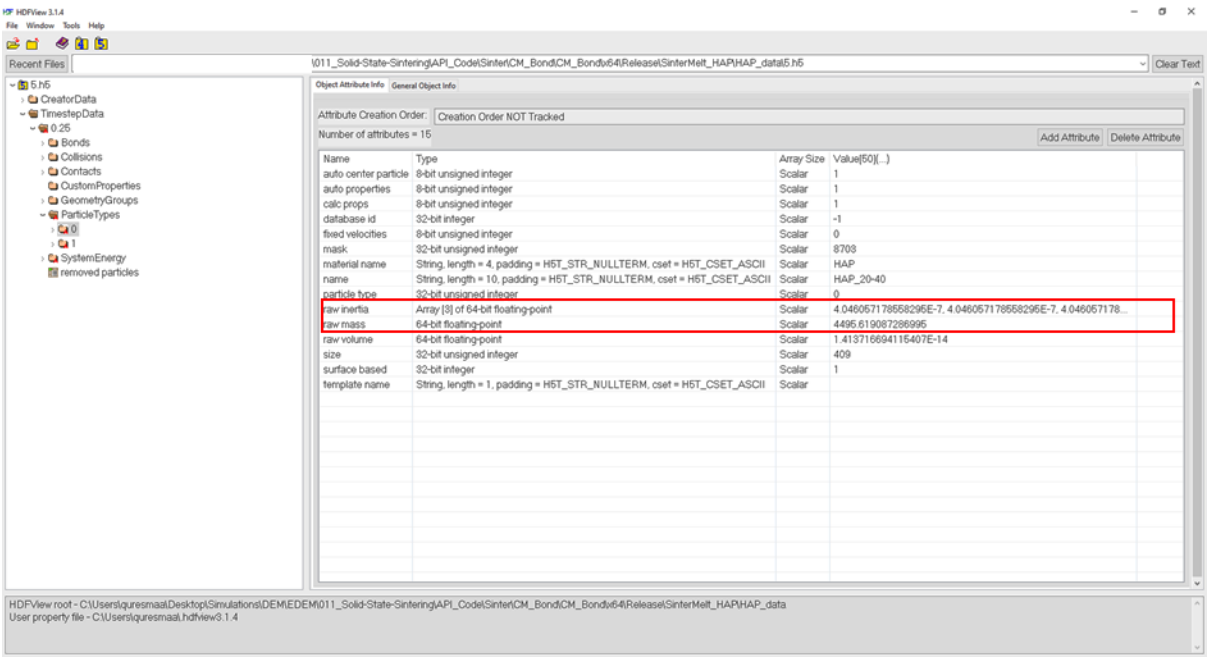
file from the latest timestep using an opensource software HDFView [95], allowing access to individual h5 format files. The example in **Figure 5.5** shows the location of the particle's raw inertia and raw mass, which was modified as given in **Table 5.1**. The file stored only the average value of the particles set in EDEM during the first timestep. For nonuniform definitions of particle sizes, as in our case, they are automatically readjusted by the solver by the same factor.

Table 5.1. Modification (scaling up) of actual values in the simulations.

Parameter		Actual value		Scale-up Factor		Final/modified value
Raw inertia	I_x	$4.046\ldots \cdot 10^{-21}$				$4.046\ldots \cdot 10^{-7}$
	I_y	$4.046\ldots \cdot 10^{-21}$	$\longrightarrow$	10^{14}	$\longrightarrow$	$4.046\ldots \cdot 10^{-7}$
	I_z	$4.046\ldots \cdot 10^{-21}$				$4.046\ldots \cdot 10^{-7}$
Raw mass	**m**	$4.495\ldots \cdot 10^{-11}$				4495.6190…



(a)



(b)

Figure 5.5. The timestep files in the h5 format opened in the HDFView for modification: **(a)** Actual raw inertia and mass in the timestep 4; **(b)** Modified (scaled up by the factor 10^{14}) raw inertia and mass in the timestep 5.

5.2.5 SUMMARY

When the particles' temperature rose above the required sintering temperature, the sintering force was executed between them. However, during the application of the sintering force, if the contact angle exceeded the dihedral angle, a repulsive force was applied to bring them back to the equilibrium state for sintering. At this point, a parameter in the code 'Bond Status' was set to 1, implying that the particles had completely fused. The discussed conditions for choosing the contact forces have been summarized in **Table 5.2**:

Table 5.2. Summary of conditions to calculate the relevant contact force.

Force	**Condition(s)**
Hertz-Mindlin	1. Particles temperature < Sintering temperature 2. Melt bond status = 0
Sinter	1. Particles temperature ≥ Sintering temperature 2. Particles temperature < Melting temperature 3. Contact angle ≤ Dihedral angle 4. Melt bond status = 0
Repulsion	1. Particles temperature ≥ Sintering temperature 2. Particles temperature < Melting temperature 3. Contact angle > Dihedral angle 4. Melt bond status = 0
Sinter/Melt bond retention	1. Contact angle > Dihedral angle 2. Melt bond status = 1

5.3. THERMAL CONDUCTIVITY MODEL AND OPTIMISATION

EDEM has an integrated model based on the work of Chaudhuri et al. [69]. As discussed in **Section 2.5.2** (**Page 38**), this model depends upon the presence of resultant interparticle forces from the Hertz-Mindlin (HM) model. However, we did not consider HM forces in our case, as we relaxed the compressive stresses. So, this model became void. Therefore, we developed a code for another thermal conductivity model based on Batchelor and O'Brien [68]. We introduced an additional resistivity (or optimisation) factor (RF) to the conduction **Equation (2.36)** (**Page 38**) in order to bring the results as close as possible to the EDEM's validated original model, as follows:

$$Q_{i,j} = RF \cdot 2\lambda a_{ij} \Delta T_{i,j} \tag{5.17}$$

In order to achieve the optimisation, we set up a simplified 5 particles (of 50 μm) system (**Figure 5.6**). 1 particle was placed in the centre, and 4 others were attached to it on the left, right, above, and below. The external walls were compressed slightly to make sure that a small overlap would exist between the particles. Otherwise, the code would ignore conduction between them in case of no overlap or just a point contact. The initial temperature of the centre particle was set at 500 K, while the rest were at 1000 K (**Figure 5.6-b**). Following, they were allowed to freely conduct heat, without any loss to the environment (i.e., in the form of convection or radiation), for 0.1 s. By this time, all the particles reached 900 K, and no further heat exchange occurred (**Figure 5.6-c**). The pseudo algorithm to develop the C++ code for this method is given in **Algorithm 5.3** (**Page 180**).

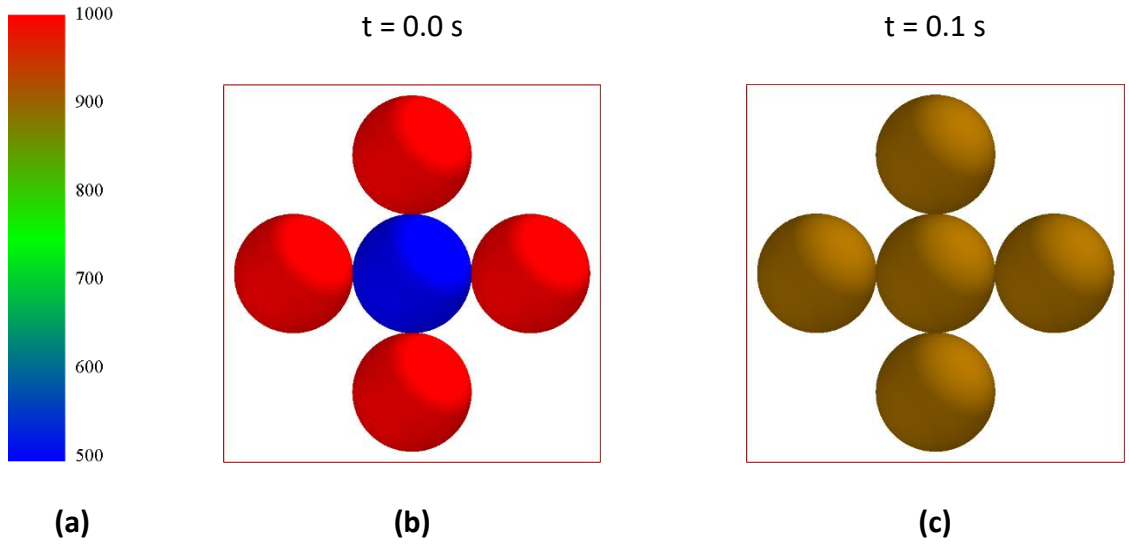


Figure 5.6. (a) Temperature legend in Kelvin; **(b)** The simplified setup of 5 particles. The inner particle is initially set at 500 K, while the outer particles at 1000K; **(c)** Final state, where all the particles have reached 900 K, and no further heat exchange exists.

In our first simulation run, we used EDEM's standard model, which was set as the optimisation benchmark. In the following runs, we replaced this with our custom-developed model and tested against different values of RF, ranging from 0.75 to 0.9. As a result, RF was set to 0.9, which appeared to produce the closest match with the EDEM's model. In this case, the final temperature of 900 K was reached at the same time as EDEM (**Figure 5.7**).

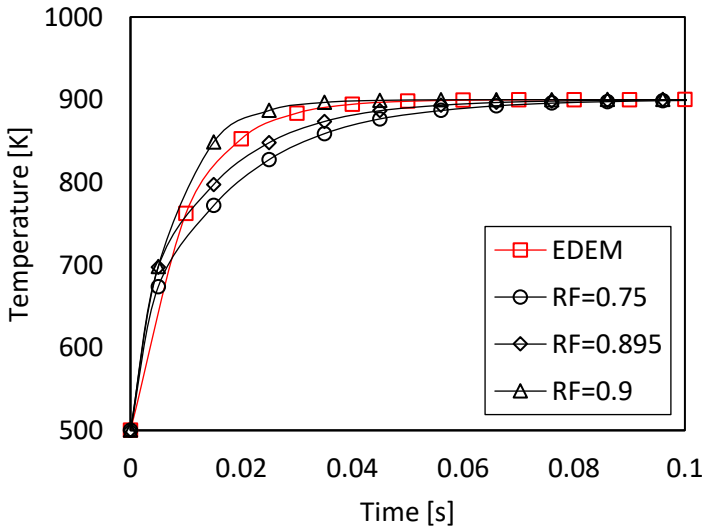


Figure 5.7. Graphical plots comparison temperature evolution of centre particle, for different values of RF against the EDEM's standard model (i.e., Chaudhuri et al. [69]).

```
CALCULATE TEMPERATURE_CHANGE:

RETRIEVE P1_TEMPERATURE, P2_TEMPERATURE;
RETREIVE P1_RADIUS, P2_RADIUS;
RETREIVE P1_POSITION(X,Y,Z), P2_POSITION(X,Y,Z);

CALCULATE CENTROID_DISTANCE(P1_POSITION, P2_POSITION):
PTC_DIST = SQRT((X1-X2)^2 + (Y1-Y2)^2 + (Z1-Z2)^2);

CALCULATE PARTICLES_OVERLAP(P1_POSITION, P2_POSITION):
PTC_OVRLP = P1_RADIUS + P2_RADIUS – PTC_DIST

CALCULATE EQUIVALENT_RADIUS(P1_RADIUS, P2_RADIUS):
EQUI_RAD = (P1_RADIUS*P2_RADIUS)/(P1_RADIUS+P2_RADIUS)

IF (PARTICLES_OVERLAP <= 0)
        EXIT ALGORITHM
END IF

SET THERMAL_CONDUCTIVITY = 1.0;
SET HEAT_CAPACITY = 628.02;
SET MASS_RESCALING = 1E-14;
SET RF = 0.9;

CALCULATE CONTACT_RADIUS:
CNT_RAD = SQRT(2 * EQUIVALENT_RADIUS * PARTICLES_OVERLAP);

CALCULATE HEAT_COEFFICIENT:
HEAT_COEFF = 2*RF*CNT_RAD*THERMAL_CONDUCTIVITY;

CALCULATE HEATFLUX_CHANGE:
CHANGE = HEAT_COEFF*(P2_TEMPERATURE-P1_TEMPERATURE);
P1_HEATFLUX_CHANGE += CHANGE;
P2_HEATFLUX_CHANGE -= CHANGE;

P1_HEATFLUX = (P1_HEATFLUX_CHANGE*MASS_RESCALING)/(MASS*HEAT_CAPACITY);
P2_HEATFLUX = (P2_HEATFLUX_CHANGE*MASS_RESCALING)/(MASS*HEAT_CAPACITY);

CALCULATE TEMPERATURE_CHANGE:
P1_TEMPERATURE += HEATFLUX_P1*TIMESTEP_SIZE;
P2_TEMPERATURE += HEATFLUX_P2*TIMESTEP_SIZE;
```

Algorithm 5.3. Pseudo algorithm for calculation of temperature change of two interacting particles, due to thermal conductivity, based on Batchelor and O'Brien [68] model. The corresponding C++ code can be found in the **Appendix-E** (**Page 250**).

5.4. COMBINATION OF SUB-MODELS

5.4.1 CONDITIONS AND SEQUENCE FOR APPLYING RELEVANT FORCES

A sequence and certain conditions were set to ensure that the correct contact force was applied to the interacting particles. They have been illustrated in **Figure 5.8** (next page). In the first phase, a simple Hertz-Mindlin (HM) (**Section 2.3.5**) (as standard EDEM standalone) contact model was applied to allow the settlement of particles in the domain. After this, the powder bed was compressed, and the standard GUI-based HM was removed.

In the second phase, the gravity was removed, and the stopping force and torque (**Section 5.2.1**) were applied to all particles to bring them to complete rest. This was followed by the stresses from all the particles being relaxed (**Section 5.2.2**). Afterwards, particle density was scaled up.

In the last phase, the ray tracing laser heating was implemented parallel to the sintering contact model. When the particles' average temperature was below the sintering temperature and not already fused together, a code for HM contact force was applied to them. It would be important to note that while HM was removed from EDEM GUI, its code was still incorporated within the sintering contact model's code, which now worked under certain conditions.

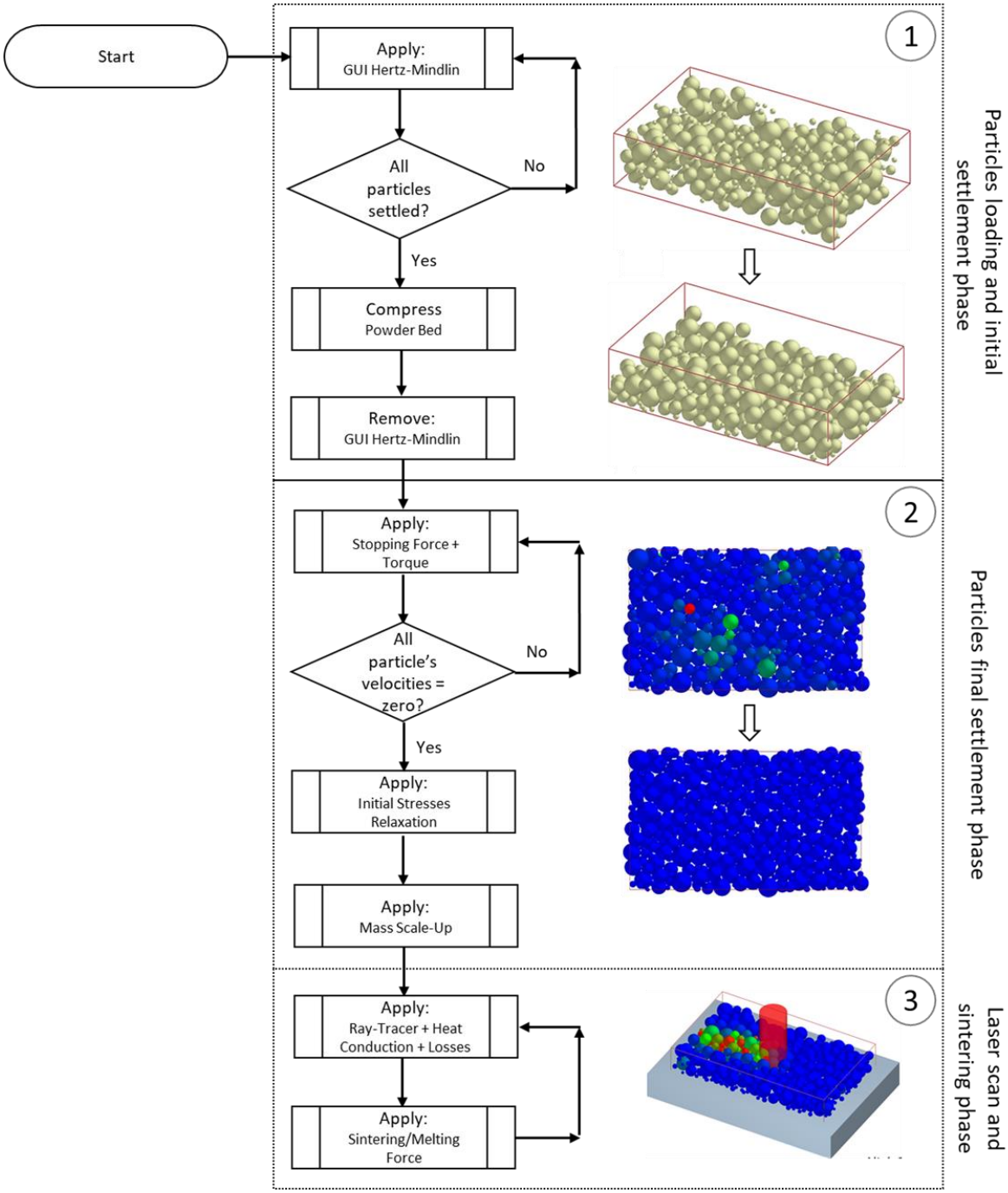


Figure 5.8. Sequence of implementation of three main phases of simulation and associated conditions for application of various forces.

5.4.2 CONDITIONS AND SEQUENCE FOR APPLYING THERMAL MODELS

In parallel to the contact forces, the conditions for selecting the thermal conductivity contact models were also established. EDEM uses the model from Chaudhuri et al. as standard (**Section 2.5.2, Page 38**). However, this model is restricted to the Hertz-Mindlin contact model working in parallel [69]. In our case, the Hertz-Mindilin would be conditionally disabled once the laser heating (ray tracing) and sintering contact models were implemented (**Table 5.2**) for the reasons discussed in **Section 5.2.2**. This would render the EDEM's standard model void. Therefore, the heat conduction algorithm based on Batchelor and O'Brien [68] was developed and applied since it is an independent model. **Table 5.3** summarizes the conditions:

Table 5.3. Summary of conditions to select the thermal contact model.

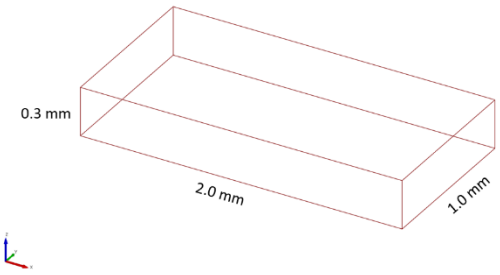
Thermal contact model	**Condition(s)**
Batchelor and O'Brien	1. Particles temperature < Melting temperature 2. Sinter bond status = 0
Standard solid conductivity	1. Sinter bond status = 1 2. Any particle temperature

5.5. SIMULATIONS SETUP, PARAMETERS, AND OPTIMIZATION

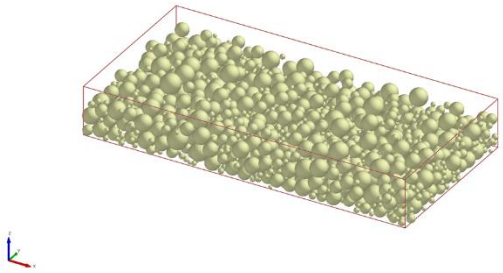
5.5.1 SETUP, EXECUTION, AND RESULTS VISUALIZATION

Even though the numerical contact and laser models for SLS were dominantly custom developed, the EDEM GUI was used in generating particles, and geometries, as a DEM solver and for results visualization. The following steps highlight the workflow, from setup to visualization:

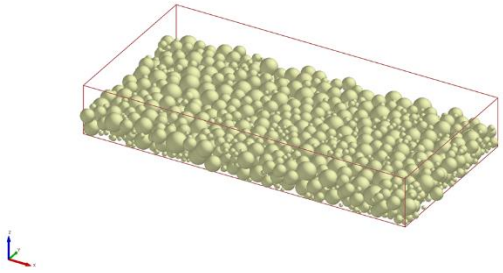
(1) Generation of the domain of dimensions: 1.0 mm × 2.0 mm × 0.3 mm



(2) Powder particles generation, as discussed in the **Section 3.3 (Page 100)**.



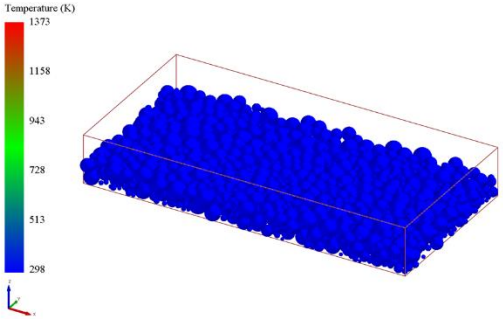
(3) Powder bed compression, to the point no significant overlaps are observed.



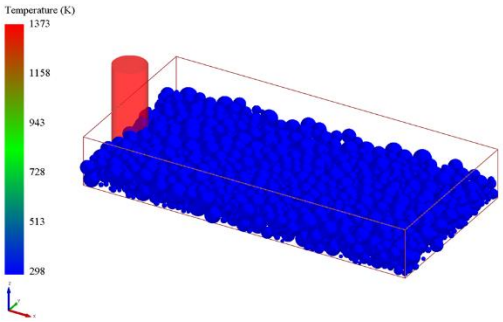
(4) Mass scaling and plugin contact models: Stopping force, ray-tracing, thermal conductivity, heat losses, and sintering.



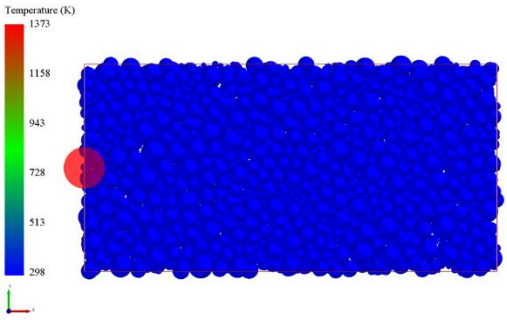
(5) Particles colouring by ‘temperature’ and show legend.



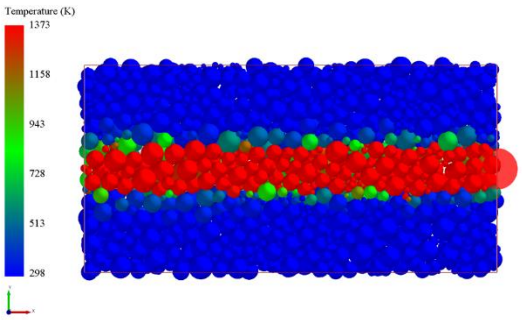
(6) Introduce transparent virtual geometry for representation of ‘laser’ size and position.



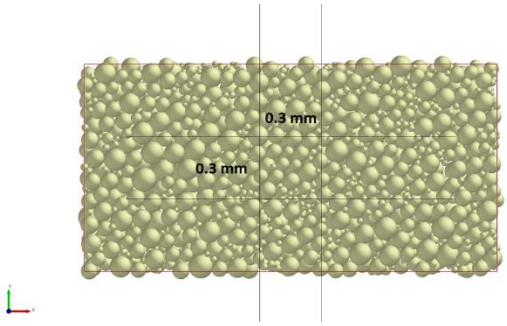
(7) Execution of solver.



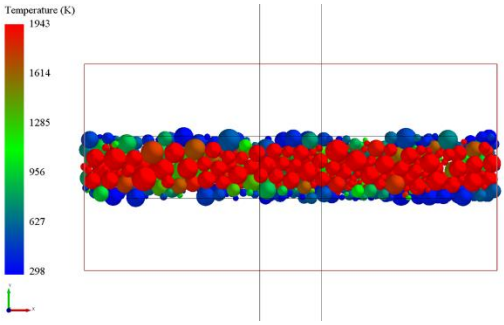
(8) Visualization of temperature field generated by the solver.



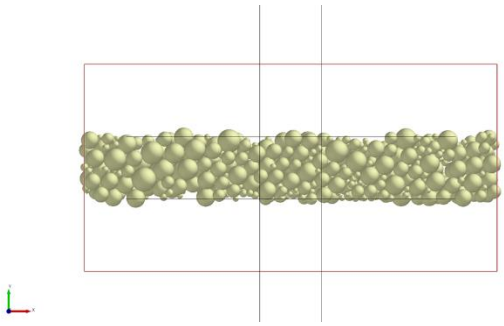
(9) Clipping/slicing the scan region



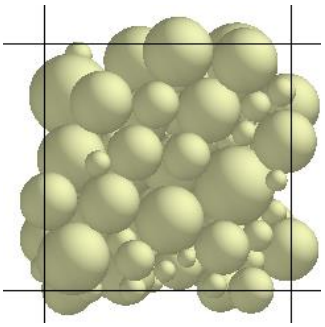
(10) Visualization of temperature field in the clipped scanned region. Post scan average temperatures were taken from this region.



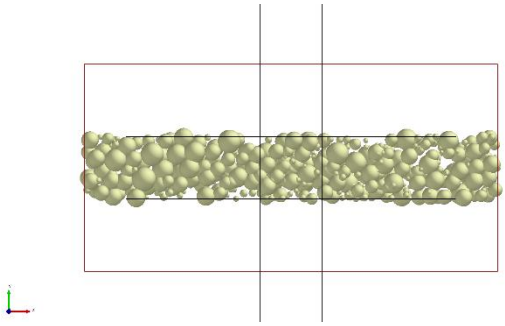
(11) Visualization of pre-sintered particles in the clipped scanned region.



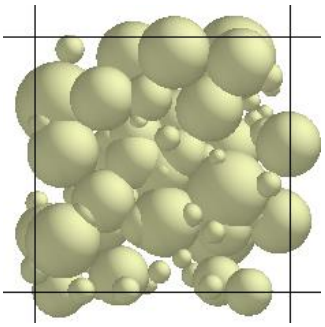
(12) Closer look at the pre-sintered particles in the central region



(13) Visualization of post-sintered particles in the clipped scanned region, as a result of heating.



(14) Closer look at the post-sintered particles in the central region, as a result of heating.



5.5.2 SIMULATION PARAMETERS

The laser processing parameters have been given in **Table 5.4 (Page 189)**, and the material properties, optical properties, and sintering parameters used in the simulations have been listed in **Table 5.5** (**Page 190**). The material considered for simulations was hydroxyapatite without additives and with 10% graphite, based on the study of Navarrete-Segado et al. [89]. Asif ur Rehman (Ermaksan, Turkey) provided the optical images of single-track experiments.

The molecular weight (M_R) of HAP is 502.3 g mol^{-1} [103]. This value was used to evaluate the molecular volume (Ω) as follows:

$$\Omega = \frac{M_R}{\rho L} = \frac{502.3}{3150000 \times 6.02214129 \cdot 10^{23}} = 2.65 \cdot 10^{-28}\ m^3 \tag{5.18}$$

where ρ is the material density and L is the Avogadro constant.

Furthermore, the minimum sintering temperature was set at 1100 °C (1373.15 K) [2], and the maximum temperature for the sintering of HAP was set at 1670 °C (1943.15 K) [104]. This may also be evaluated using the phase diagram of the CaO-P_2O_5 system given in **Figure 5.9** [105]. After this temperature, the powders' melted/liquid phase was assumed to have formed.

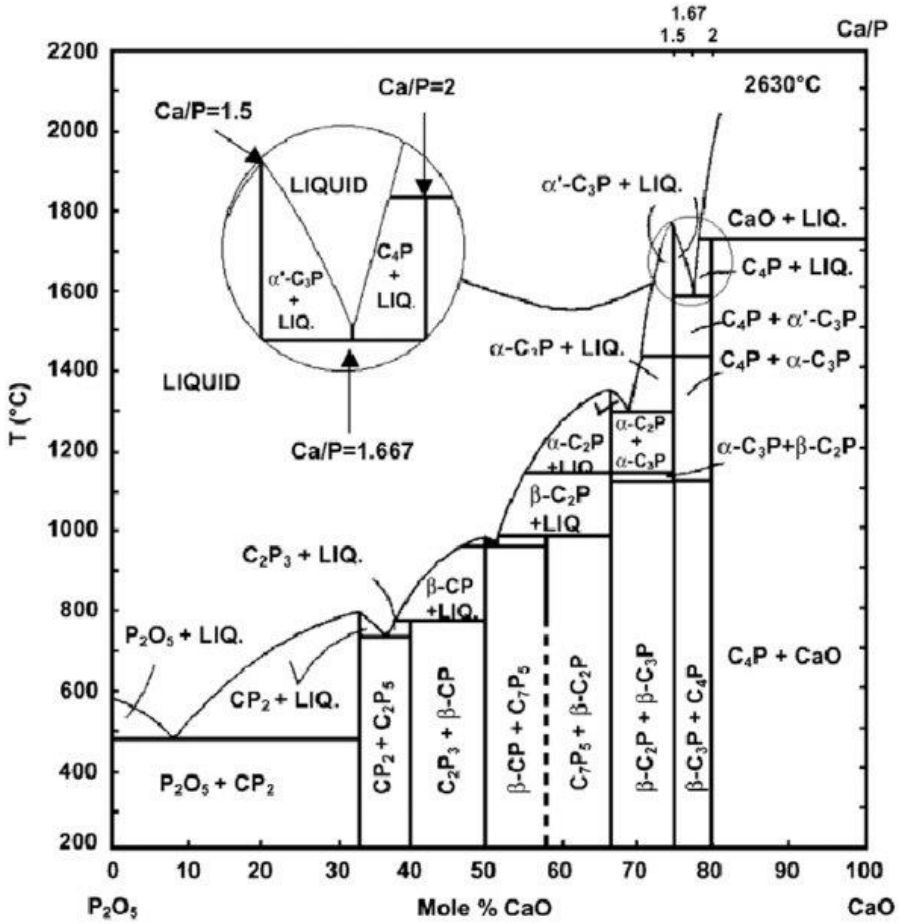


Figure 5.9. Phase diagram of the system CaO–P_2O_5 (C = CaO, P = P_2O_5) at elevated temperatures. Each line represents a phase boundary. Here C7P5 represents $7CaO{\cdot}5P_2O_5$; the other abbreviations can be written out in the same manner [105].

Table 5.4. Laser processing parameters, adapted from Navarrete-Segado et al. [89].

Index	Power [W]	Scan speed [mm s^{-1}]	Beam diameter [µm]	Scan length [mm]
1	30	25	200	2.0 (Single track)
2		50		
3		75		
4		100		
5		125		
6	36	25	200	2.0 (Single track)
7		50		
8		75		
9		100		
10		125		
11	42	25	200	2.0 (Single track)
12		50		
13		75		
14		100		
15		125		
16	48	25	200	2.0 (Single track)
17		50		
18		75		
19		100		
20		125		
21	54	25	200	2.0 (Single track)
22		50		
23		75		
24		100		
25		125		

Table 5.5. Summary of material, optical and sintering parameters for SLS of hydroxyapatite.

Parameter	Symbol	Value	Unit	Reference / Cross-reference
Thermomechanical				
Specific heat capacity	c_p	628.02	J kg^{-1} K^{-1}	[106]
Thermal conductivity	k	1.0	W m^{-1} K^{-1}	[106]
Density	ρ	3150	kg m^{-3}	[106], [107]
Heat transfer coefficient	h	Variable	W m^2 K^{-1}	**Table 5.3**
Emissivity	ϵ	0.3	-	[64]
Atmospheric temperature	T_a	298.15	K	[89]
Optical				
Laser type	-	Fibre laser	-	[89]
Laser wavelength	λ	1070	nm	[89]
Laser energy absorption	α_{abs}	Variable	-	[89]
Refractive index	n	1.63	-	[96]
Absorption (HAP)	α_{abs}	0.024	-	**Table 4.3**
Absorption (HAP + 10G)	α_{abs}	0.737	-	**Table 4.3**
Sintering				
Surface energy	γ_S	0.089	J m^{-2}	[108]
Molecular volume	Ω	$2.65 \cdot 10^{-28}$	m^3	**Equation (5.18)**
GB diffusion	$D_g\delta_g$	$8.6 \cdot 10^{-10}$	m^3 s^{-1}	[109], [110]
Activation energy	Q_b	438	kJ mol^{-1}	[111]
Dihedral angle	φ	120	°	-
Sintering temperature	T_{sinter}	1373.15	K	[2]
Melting temperature	T_{melt}	1943.15	K	**Figure 5.9**
Boltzmann constant	k_B	$1.38 \cdot 10^{-23}$	m^2 kg s^{-2} K^{-1}	[112]
Gas constant	R	8.3145	J mol^{-1} K^{-1}	[112]

5.5.3 SIMULATION OPTIMIZATION PARAMETER

It would be important to reiterate that PBSLP is a very complex process. There have been very limited studies on HAP ceramics and investigations on its thermal behaviour of different phases, especially in terms of measurement of thermal conductivities, specific heat, latent heat of fusion, grain boundary diffusion rates during sintering, and laser optical parameters. Furthermore, there was a significant difference in the scales of the actual physical process and the model in this work, which was expected to affect the heat exchange in the overall system, as we were only considering about 2 mm laser scan. So, producing a simulation digital twin to the experiments became an intricate task. The scope of this research at this stage was limited to sintering only, which meant any melting behaviour also could not be undertaken.

So, owing to the discussed issues, we could not incorporate the phase change models that would keep the temperatures in check. This resulted in temperatures of the particles shooting up to an unrealistic degree, i.e., as high as tens of thousands, as the absorbed energy caused a linear rise. On contrary, during phase transformation process the temperature of the body becomes constant. Moreover, after a certain temperature the liquid also starts to evaporate. In addition, HAP undergoes multiple chemical phase changes in solid state as well, such as β-Tricalcium phosphate, α-Tricalcium Phosphate and calcium oxide, before entering the actual liquid state.

Therefore, to address the discrepancies between the simulations and experiments, simulation optimization factors (SOF) were introduced to the heat transfer equation for ray tracing (RT), heat losses (HL), and heat conduction (HC). The overall heat transfer ($\dot{Q}_{tot}$) to each particle was expressed as:

$$\dot{Q}_{tot} = SOF_{RT} \cdot \dot{Q}_{laser} + SOF_{HL} \cdot (\dot{Q}_{rad} + \dot{Q}_{conv}) + SOF_{HC} \sum \dot{Q}_{cond} \qquad (5.19)$$

where $\dot{Q}_{laser}$ is the laser input energy, $\dot{Q}_{rad}$ is heat loss due to radiation, $\dot{Q}_{conv}$ is the heat loss due to convection to the environment, and $\sum \dot{Q}_{cond}$ is the accumulated heat transfer (i.e., after energy gains and losses) due to conduction with the neighbouring particles.

The values of SOF (**Table 5.6**) were set constant for all the simulation runs against the laser parameters listed in **Table 5.4** to ensure consistency and reliability. They were evaluated using the results from Navarrete-Segado et al. [83] and single-track laser experiments data provided by Ermaksan (Turkey), which indicated sintering observation for the given laser parameters.

Table 5.6. Constant values of simulation optimization factor (SOF) for different modes of heat transfer: Input via laser ray tracing (RT), heat losses (HL), and heat conduction (HC).

SOF_{RT}	SOF_{HL}	SOF_{HC}
$1.5 \cdot 10^{-1}$	10^{-3}	10^{-4}

5.6. RESULTS AND DISCUSSION

5.6.1 HYDROXYAPATITE WITHOUT ADDITIVES

Pure HAP exhibited very low absorption (2.4%) of fibre laser wavelength. So, the direct laser sintering experiments by Navarrete-Segado et al. demonstrated that the powder bed could not be sintered with any of the (earlier listed) laser parameters (**Figure 5.10**) [89].

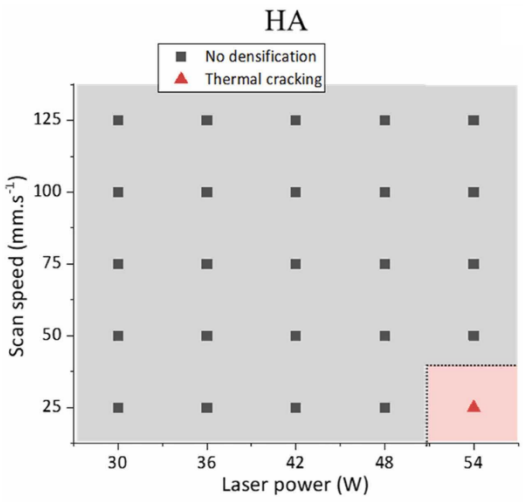


Figure 5.10. PBSLP process window comparison of the HAP without (graphite) additives. Navarrete-Segado et al. [89].

Therefore, we began our simulations by choosing the highest power (54 W) and lowest scan speed (25 mm s^{-1}). The highest power meant that the maximum possible energy was supplied to the powder bed, while the lowest speed implied that the greatest exposure time was given to any of the particles in the scan region. Therefore, the particles would ultimately reach the highest possible temperatures in the powder bed from our given set of parameters.

It was observed that most of the particles failed to reach the minimum sintering temperature of 1373.15 K (**Figure 5.11**). This simple strategy helped curtail any further simulation runs with other parameters. Consequently, it was inferred that lowering the laser power or increasing the scan speed would become ineffective for further analysis. Therefore, it was concluded that

simulation model was in complete agreement with the experiments for pure HAP, within our working set of laser parameters.

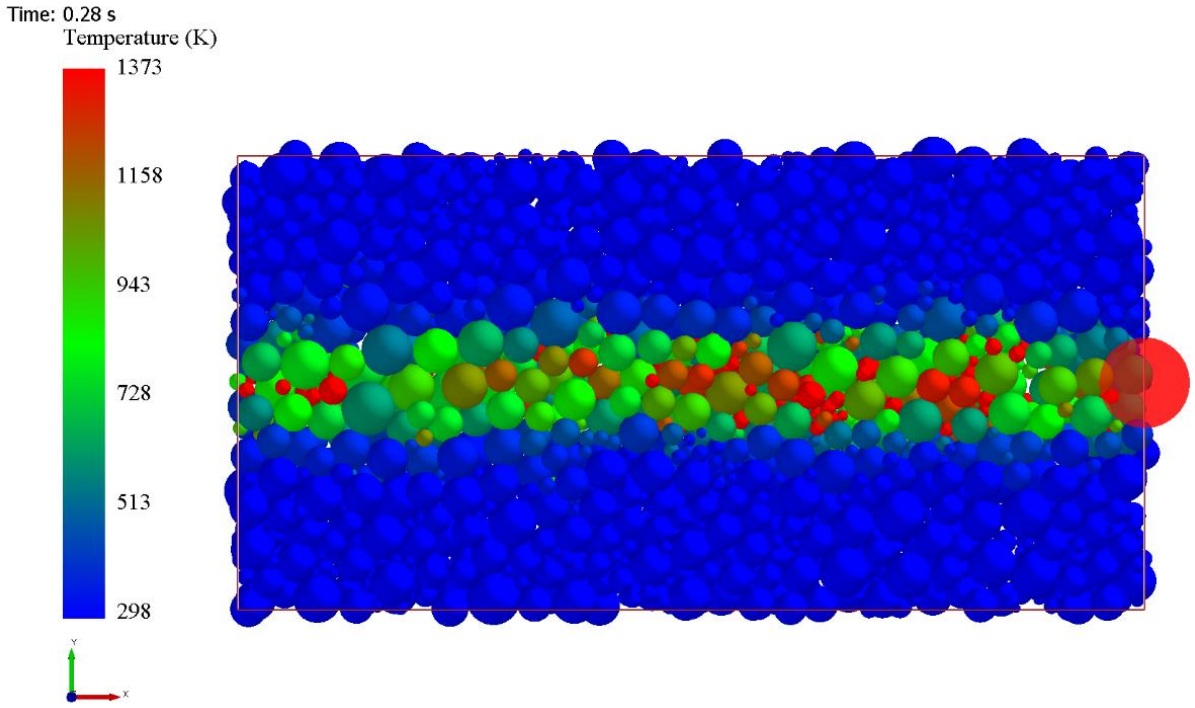


Figure 5.11. Visualization of temperature field on the scan region of pure HAP at 54 W power and 25 mm s^{-1} speed. The minimum temperature on the legend was set at 298 K (25 °C), which was the standard room temperature. In contrast, the maximum temperature was set at 1373 K (1100 °C) to help visually identify the (potential) particles above the sintering temperature.

5.6.2 HYDROXYAPATITE WITH 10% GRAPHITE ADDITIVE

The laser absorption in our ray tracing code was increased to 73.7% for HAP + 10% Graphite. As mentioned earlier in the simulation parameters (**Table 5.5**, **Page 190**), the sintering temperature was set at 1373.15 K (1100 °C), and the indicative melting temperature was set at 1943.15 K (1670 °C). So, the scan region where average temperatures crossed the sintering temperature was considered to have its particles actively sintered in simulations. The region that crossed the melting temperature was both sintered and formed a dense clump. The average temperatures produced by the simulations in the scan region were mapped against all the power and speed settings (**Figure 5.12-a**), while the experimental sintering response was mapped in parallel (**Figure 5.12-b**).

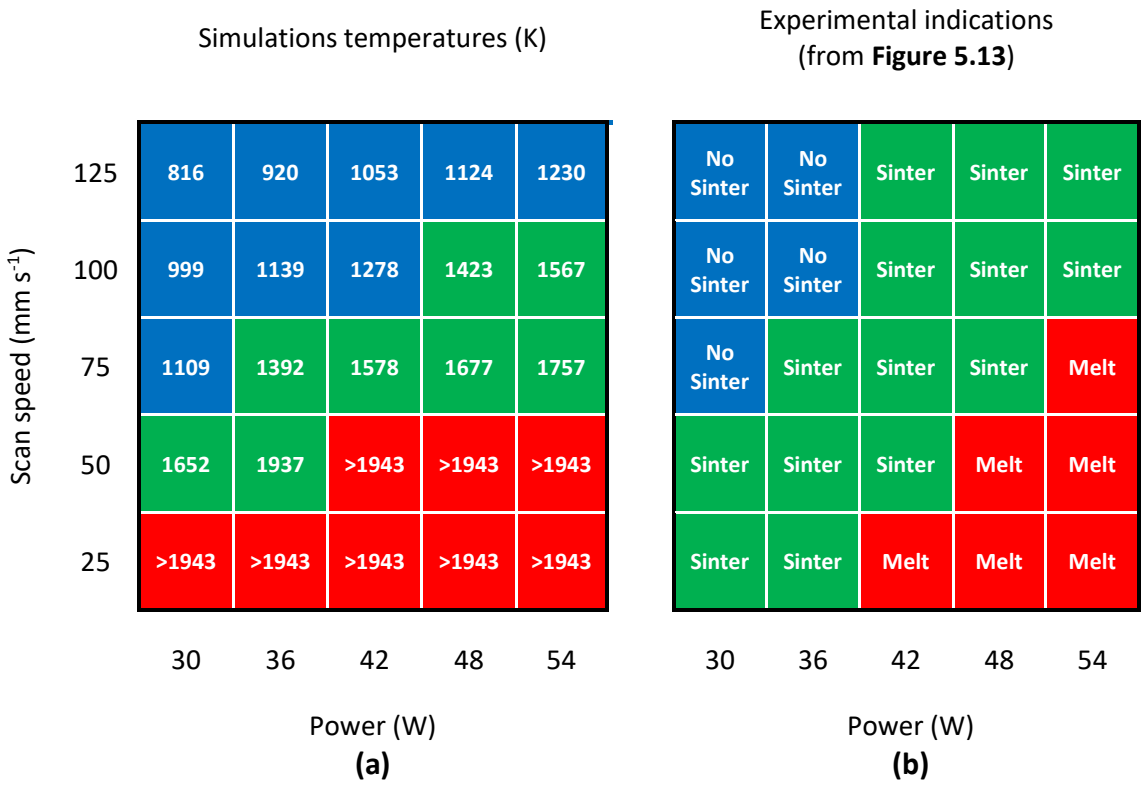


Figure 5.12. Comparison of simulation and experimental indication of sintering response. **(a)** The response in the simulations is indicated by the average temperature recorded in the scan region, **(b)** while visually in the experiment, from **Figure 5.13**. The **blue** cells represent the **negative sintering** response. The **green** and **red** represent the **positive sintering** and **melting/consolidation** response, respectively.

There were three main observations in the experimental results, either the particle did not sinter, or they were sintered, or they were melted and consolidated (**Figure 5.13**, **Page 197**). The images highlighted in blue represent the case where no sintering was recorded, green with observed sintering, and red demonstrated sintering and consolidation. The particles were at least sintered in the laser scan region in the lower speed settings of 25 mm s^{-1} and 50 mm s^{-1}. At 75 mm s^{-1} speed, the particles did not sinter on the 30 W power setting only, while they continued to sinter at higher powers. The particles also did not sinter at higher speeds of 100 mm s^{-1} and 125 mm s^{-1} at lower powers of 30 W and 36 W. Above and including 42 W, particles sintered or consolidated for all scan speeds. The main reason for the sintering response was the addition of 10% graphite to the HAP powder bed, which improved the laser absorption (from 2.4% to 73.7%).

Similar to experiments, sintering was observed across all the power settings at the lower scan speeds of 25 mm s^{-1} and 50 mm s^{-1} in simulations. Moreover, the simulation results showed that powers 30 W and 36 W at 25 mm s^{-1} also led to melting temperatures, in contrast to the experiments where only sintering was observed. Similarly, a complete agreement between the simulations and experimental sintering response was seen at 75 mm s^{-1} scan speed across all the powers. Additionally, the experimental results indicated melting at 54 W. Contrarily, 4 discrepancies were observed between the simulations and experiments at 100 mm s^{-1} and 125 mm s^{-1} scan speeds. At these speeds, lower powers of 30 W and 36 W exhibited no sintering in simulations and experiments. However, in the simulations, no sintering response could be seen for 42 W at 100 mm s^{-1} and 42 W and above at 125 mm s^{-1}, while the experiment displayed a positive response for these settings.

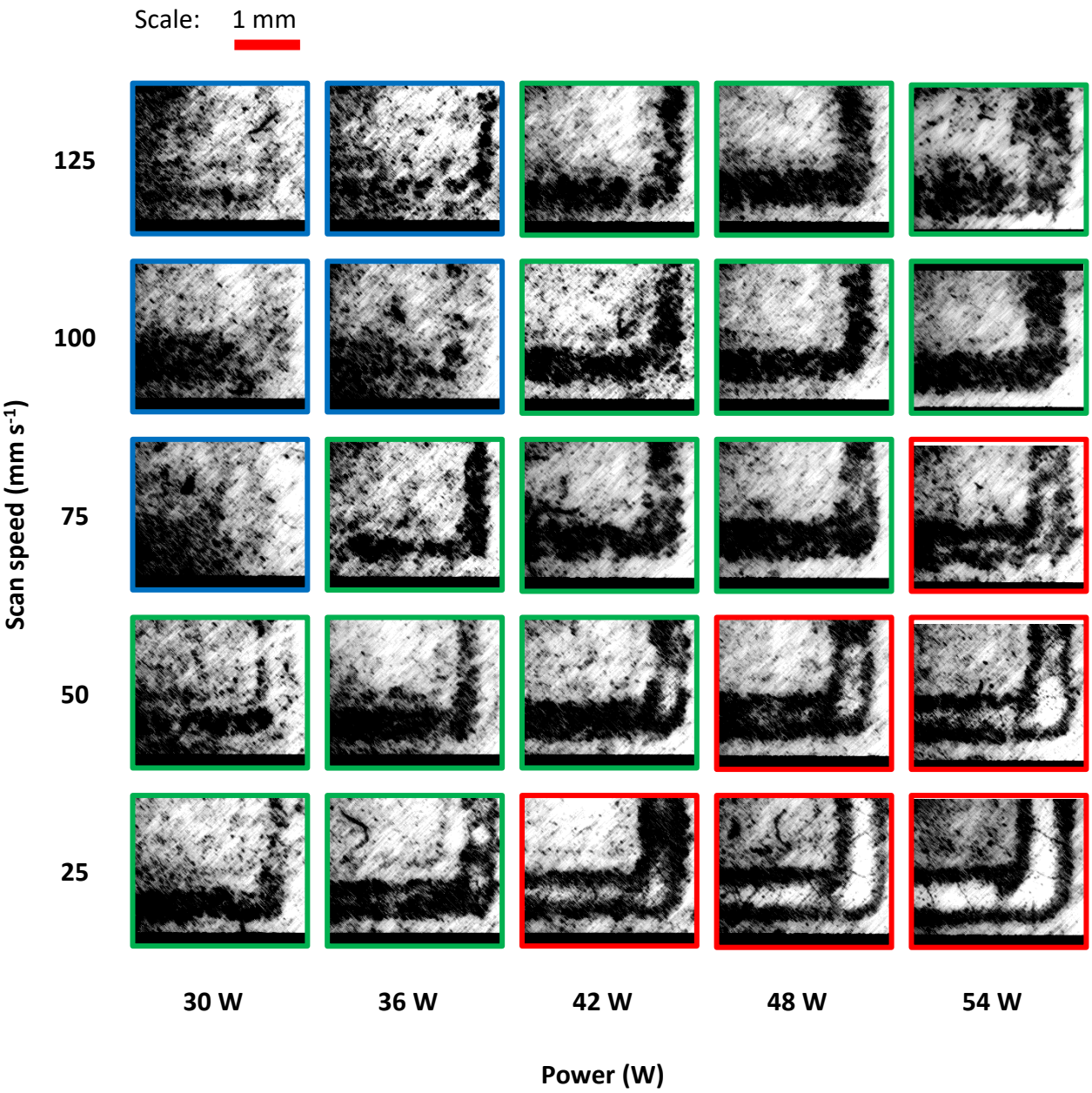


Figure 5.13. Processed SEM images of single track experimental results for the given laser powers and scan speeds. The images highlighted in **blue** represent no sintering, the **green** hightlight indicate sintering took place, while the **red** indicate consolidation (after melting). Images courtesy of Dr Asif ur Rehman (ERMAKSAN, Turkey) and Dr Pedro Jesus Navarrete-Segado (CIRIMAT - INPT, France).

84% of the simulations correctly predicted whether the temperatures crossed the sintering point. However, in 12% of the total cases, simulations overestimated the temperatures going above the melting point, where experiments only indicated sintering below the melting point. Compared to experiments, 16% of the simulations underestimated the temperatures by predicting lower than sintering temperatures. Overall, 72% of the simulations precisely forecasted the 3 main temperature regions: Under sintering, between sintering and melting, and above melting.

Furthermore, sintering and melting temperatures presented a contrast in the fusion of particles in simulations. For instance, in a 42 W power setting, where above melting average temperatures were recorded at 50 mm s^{-1} and at least sintering at 75 mm s^{-1}, the comparison of temperature fields can be observed in **Figure 5.14** below. At 50 mm s^{-1}, most of the particles within the scan region had topped 1943 K (bright red coloured), while at 75 mm s^{-1}, the particles below 1373 K (bright green coloured) were prominent as well, and they tended to bring the average temperatures of the scan region down. The higher of the two speeds triggered sintering, but the lower speed allowed denser formation due to longer laser exposure time (**Figure 5.15**). The graphical results for all other combinations of power and scan speeds can be found in the appendix.

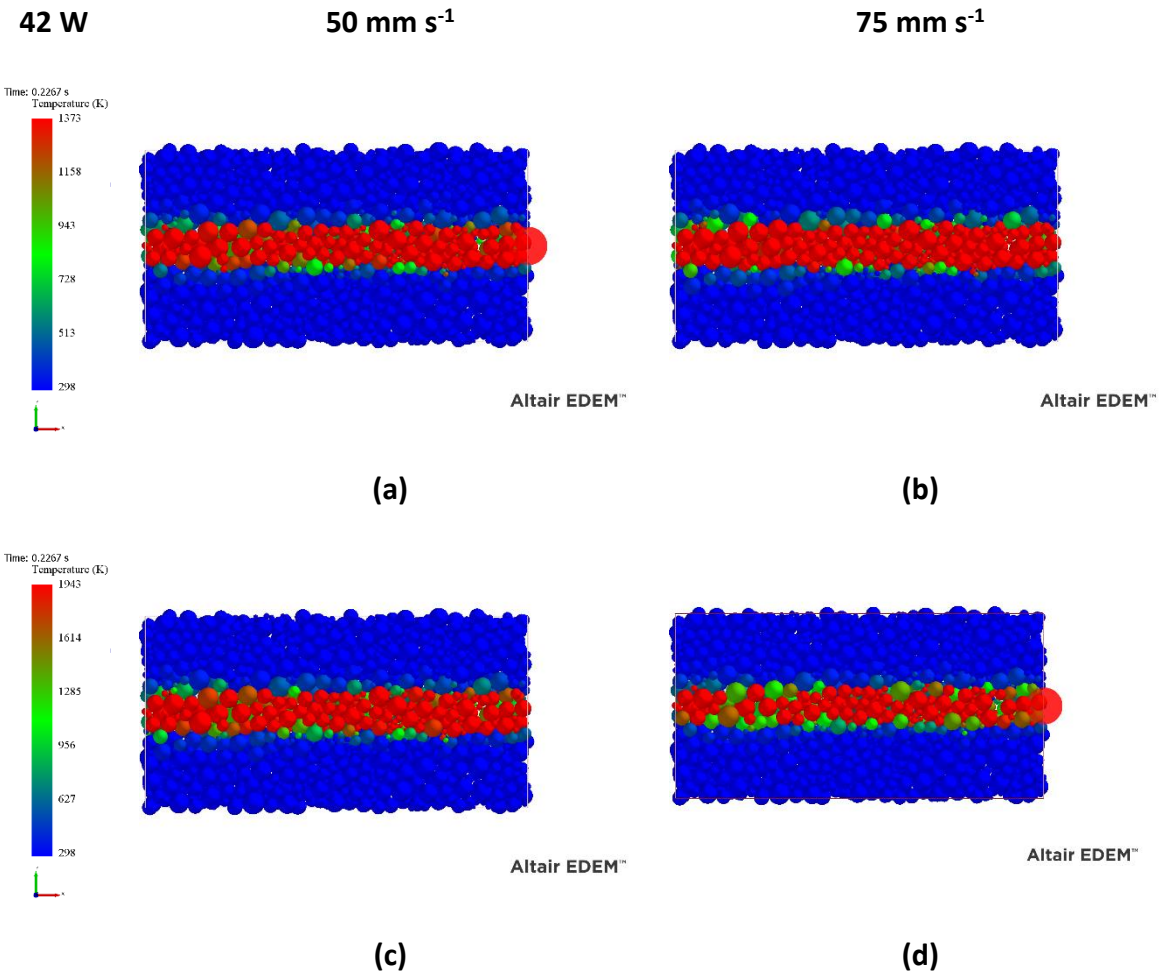
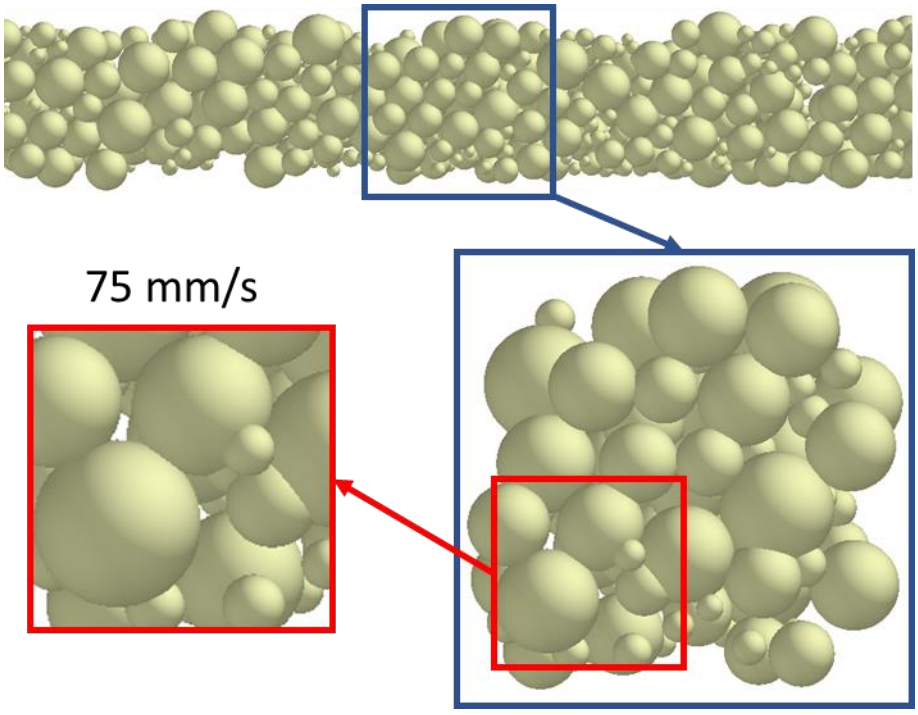
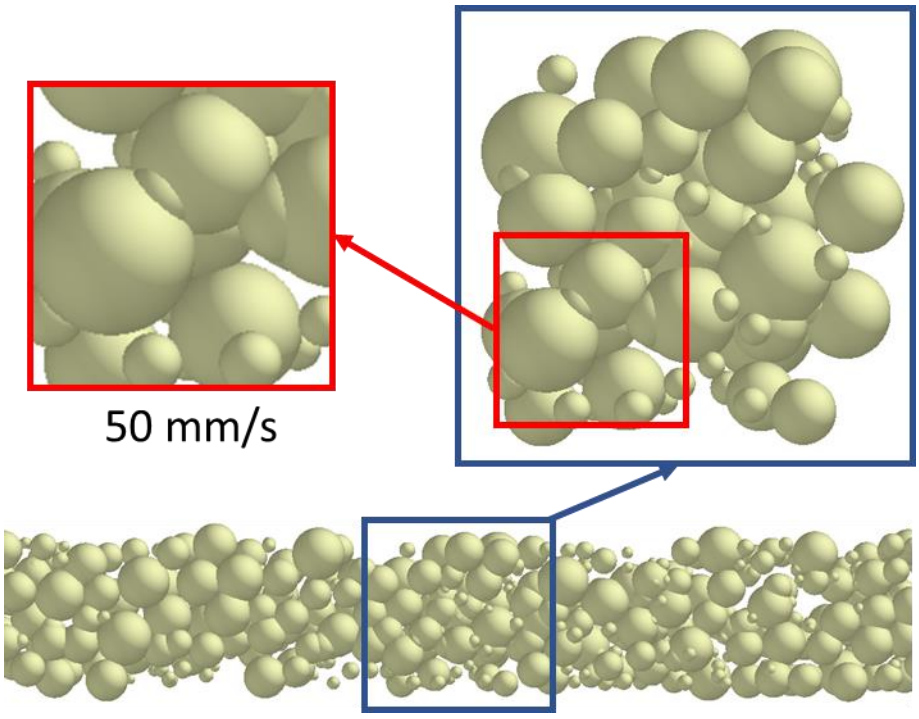


Figure 5.14. Temperature distributions after laser scan with 42 W power, at 50 and 75. The legends of figures **(a)** and **(b)** represent the maximum temperature of 1373 K (sintering), while **(c)** and **(d)** represent the maximum 1943 (melting).



(a)



(b)

Figure 5.15. Cropped and magnified views of laser scanned region for **(a)** 75 mm s^{-1} and **(b)** 50 mm s^{-1}. Sintering was triggered at both speeds, but lower speed allowed denser formation.

6. CONCLUSIONS AND OUTLOOK

Highlights:

1. Thesis summary, conclusions, and final remarks

2. Critical analysis of the research work, potential improvements and future work

6.1. CONCLUSIONS

The objective of the research work presented in this thesis was to simulate the powder bed selective laser processing (PBSLP) of ceramics to develop a better understanding at microscale. The models were divided into two parts mainly: Pre-processing and processing. The purpose of pre-processing models was to investigate the powder delivery and layer formation of alumina powders using blade recoater. In contrast, the laser processing models served to provide an understanding on effect of laser parameters on sintering response of the hydroxyapatite (HAP) powder beds. Therefore, we adopted a discrete element method (DEM) solver 'Altair EDEM' that helped simulate the dynamic behaviour of the powders during these processes.

Therefore, to understand the preprocessing powder delivery phase, we collaborated with the Belgian Ceramic Research Centre (BCRC). The research focused on alumina powder deposition using a blade coater. 5 different powder samples (**Table 3.1**, **Page 62**) were considered and their DEM volumetric size distributions were generated that closely matched the actual powder distributions. These powders were then calibrated using angle of repose method, to ensure that their dynamic behaviour resembled the physical powders. All the samples exhibited good flowability on Riley et al. [45] scale, which improved with the coarser powders due to reduced contact surface area, leading to lower cohesivity and friction.

During the powder deposition process using the blade recoater, the DEM simulations provided a closer look on evolution of packing densities of 3 different layer thickness of, i.e., 100, 150, and 200 µm. In contrast to 200 µm, 100 and 150 µm produced significant voids that rendered them completely ineffective for further processing. However, the voids were observed to reduce with addition of further layers. By the deposition of the fourth layer, the packing

densities became comparable regardless of size distribution and layer thickness. The finer powder, nonetheless, still produced relatively denser packing, but there was a smaller variation towards the coarser powders.

We developed a ray-tracing code that helped us investigate the surface quality of the powder bed. Its methodology followed setting up of a virtual square domain vertically offset from the top surface of the powder. This domain was then discretized into smaller cells, and a ray was projected from the centre of each towards the powder bed. The travel length of each ray was recorded up to the point of first interception. The lengths data was accumulated to evaluate the surface roughness parameter R_a. The finer powders produced a more level surface, while the coarser powders generated rougher surfaces.

In the processing phase, we developed codes for the laser ray tracing that were capable of evaluating light scattering (using reflections and transmissions, principles) and particle energy absoprtion. As a result, it enabled us to precisely map the heat distribution field, in randomly sized powder beds of different packing densities, resulting from laser thermal interaction. In contrast, the FEM demonstrated limited capability to analyse impact of powder bed packing density, but none whatsoever with respect to size distributions, on the thermal distribution field.

Besides laser ray tracing, we also developed heat conduction, losses, and most importantly sintering codes. They were coupled with the DEM solver and helped produce a powder bed laser sintering model on a limited scale. They helped us investigate the sintering response to fibre laser, of both pure HAP and HAP with graphite additivies powder beds. We used these models to simulate the temperature field and sintering against the 25 combinations of laser parameters. The power varied from 30 W to 54 W, with a 6 W increment. The scan speed was

changed from 25 mm s^{-1} to 125 mm s^{-1} with a 25 mm s^{-1} increment against each given power (**Table 5.4**, **Page 189**). The laser spot size was set constant at 200 µm. The simulation correctly predicted 18 settings compared to the experiments. 2 simulated instances of 30 W and 36 W at 25 mm s^{-1} produced overestimated results, where the simulations inferred that the temperatures exceeded the melting point, while experiments inferred sintering only. In contrast, 4 laser settings of 42 W at 100 mm s^{-1} and 42, 48, and 54 W at 125 mm s^{-1} demonstrated no sintering in the simulations, while it was observed in the experiments (**Figure 6.1**, **Page 204**). In addition to general sintering responses, we observed that same laser power at slower speeds produced either higher sintering density or completely melted the powders. With this methodology, we could also classify the laser parameters leading to solid-state sintering response, which were identified to be in the region of lower speeds and lower powers, and higher speed and higher powers (**Figure 5.12**, **Page 195**).

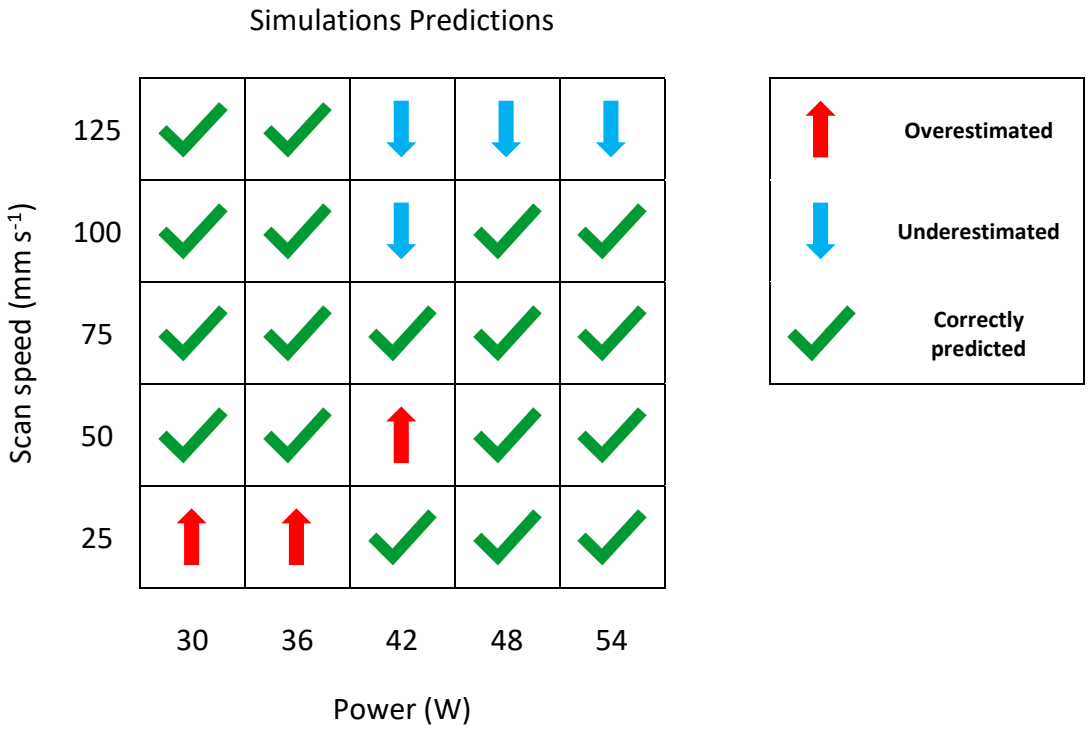


Figure 6.1. Sintering prediction in our simulations, compared to the experiments shown in **Figure 5.13** (**Page 197**).

6.2. LIMITATIONS, FUTURE WORK, AND IDEAS

One of the restrictions to the codes developed for our current simulation models (given in the appendix) is that they are confined to a closed-source solver, i.e., EDEM. Therefore, we plan to transfer our codes to an open-source environment, especially for ray tracing laser heating and sintering. No doubt, the reprogramming and adaptation to a differently formatted solver would be expected in this case. Nevertheless, a successful transfer would allow access to a broader outreach towards the AM research community in a bid to improve further.

Currently, the scope of this thesis was limited to dealing with the process under the melting phase, even when the temperatures crossed this point, which left room for further investigation. It would also be worth mentioning that HAP undergoes a phase transition at elevated temperatures below the melting point. The experimental measurement of thermal conductivities and specific heat of different phases of HAP and latent heat of fusion for forming the melt phase would, therefore, be necessary to improve the accuracy and precision of the models.

One of the most challenging and complex aspects of simulating the powder bed selective laser processes (PBSLP) is multiphase sintering and melting. The particles enter the solid-state-sintering phase under the melting point. During melting, they undergo the liquid-state-sintering phase. Finally, post-melting, they would be in the liquid state, which would then be followed by consolidation upon cooling. The models of melt or liquid pools are developed with entirely different approaches, e.g., involving Marangoni convection [113], [114]. Theoretically, we may be able to address this issue by considering the development of a versatile fusion model that would cover solid-state sintering, liquid-state sintering, and melting altogether. The idea would be to utilize the DEM principles in the solid regions and the volume of fluid

(VOF) method in the liquid region. Those regions could be discretized in the form of cells and classified for their state depending on temperatures. However, in such an implementation, our current ray tracing method will no longer be appropriate after melt pool formation, as the spherical particles would be deformed.

In contrast, an alternative would be a mesh-free method similar to DEM, called smoothed-particle hydrodynamics (SPH). SPH can simulate both solid mechanics and fluid flows. The current ray tracing method would still be utilizable.

There was another issue of powder bed compression, discussed in **Section 3.3.3** (**Page 104**). One of the experimental ways is to take a sample of the compressed powder and measure its renewed size distribution. The data obtained would then be applied in DEM, and the compression step could be eliminated. An alternative would be implementing the Tavares UFRJ Particle Breakage model. This model converts the large particles under compression into smaller particles and redistributes their energies (**Figure 6.2**) by utilizing the particle replacement approach [115]. However, an experimental optimization study would still be required to define the required material parameters for this model. Once the powder has been optimized, repeated experimental measurements for size distributions (as suggested earlier) will no longer be necessary.

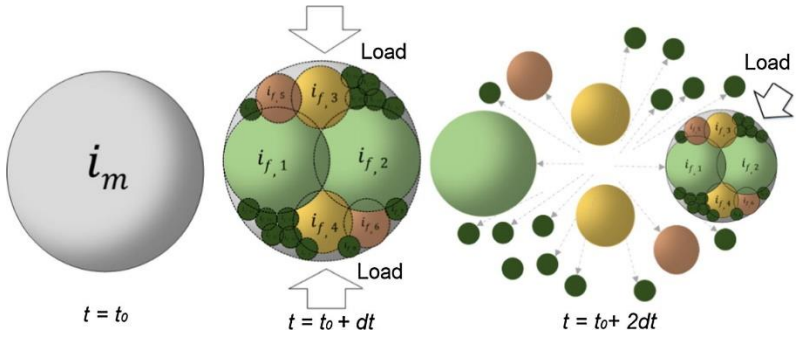


Figure 6.2. Illustration of the particle replacement model (PRM) [115].

REFERENCES

[1] D. Grossin *et al.*, "A review of additive manufacturing of ceramics by powder bed selective laser processing (sintering / melting): Calcium phosphate, silicon carbide, zirconia, alumina, and their composites," *Open Ceramics*, vol. 5, no. 100073, 2021, doi: 10.1016/j.oceram.2021.100073.

[2] J. Rojek, S. Nosewicz, K. Pietrzak, and M. Chmielewski, "Simulation of powder sintering using a discrete element model," *Acta Mechanica et Automatica*, vol. 7, no. 3, pp. 175–179, 2013, doi: 10.2478/ama-2013-0030.

[3] L. C. de Jonghe and M. N. Rahaman, "4.1 Sintering of Ceramics," *Handbook of Advanced Ceramics: Materials, Applications, Processing and Properties*, vol. 1–2, pp. 187–264, Jan. 2003, doi: 10.1016/B978-012654640-8/50006-7.

[4] W. H. Lee, Y. Zhang, and J. Zhang, "Discrete element modeling of powder flow and laser heating in direct metal laser sintering process," *Powder Technol*, vol. 315, pp. 300–308, Jun. 2017, doi: 10.1016/j.powtec.2017.04.002.

[5] D. Moser, S. Pannala, and J. Murthy, "Computation of effective radiative properties of powders for selective laser sintering simulations," *The Journal of The Minerals, Metals & Materials Society (TMS)*, vol. 67, pp. 1194–1202, Mar. 2015, doi: 10.1007/s11837-015-1386-8.

[6] N. K. Tolochko, T. Laoui, Y. V. Khlopkov, S. E. Mozzharov, V. I. Titov, and M. B. Ignatiev, "Absorptance of powder materials suitable for laser sintering," *Rapid Prototyp J*, vol. 6, no. 3, Sep. 2000, doi: 10.1108/13552540010337029.

[7] N. K. Tolochko, M. K. Arshinov, A. V. Gusarov, V. I. Titov, T. Laoui, and L. Froyen, “Mechanisms of selective laser sintering and heat transfer in Ti powder,” *Rapid Prototyp J*, vol. 9, no. 5, Dec. 2003, doi: 10.1108/13552540310502211.

[8] C. R. : Deckard, J. J. : Beaman, and J. F. Darrah, “Method and Apparatus for Producing Parts by Selective Sintering,” *U.S. Patent 4,863,538*, 1992.

[9] J. P. Kruth, X. Wang, T. Laoui, and L. Froyen, “Lasers and materials in selective laser sintering,” *Assembly Automation*, vol. 23, no. 4, Dec. 2003, doi: 10.1108/01445150310698652.

[10] K. F. Leong, D. Liu, and C. K. Chua, “Tissue Engineering Applications of Additive Manufacturing,” *Comprehensive Materials Processing*, vol. 10, pp. 251–264, Jan. 2014, doi: 10.1016/B978-0-08-096532-1.01010-4.

[11] W. H. Lee, Y. Zhang, and J. Zhang, “Discrete element modeling of powder flow and laser heating in direct metal laser sintering process,” *Powder Technol*, 2017, doi: 10.1016/j.powtec.2017.04.002.

[12] “The Finite Element Method (FEM),” *COMSOL*, Mar. 15, 2016. https://www.comsol.com/multiphysics/finite-element-method (accessed Jan. 23, 2023).

[13] C. R. K. Windows-Yule, D. R. Tunuguntla, and D. J. Parker, “Numerical modelling of granular flows: a reality check,” *Comput Part Mech*, vol. 3, pp. 311–3, Dec. 2016, doi: 10.1007/s40571-015-0083-2.

[14] S. Luding, “Introduction to discrete element methods: Basic of contact force models and how to perform the micro-macro transition to continuum theory,”

European Journal of Environmental and Civil Engineering, vol. 12, no. 7–8, pp. 785–826, Oct. 2008, doi: 10.1080/19648189.2008.9693050.

[15] S. J. Burns, P. T. Piiroinen, and K. J. Hanley, "Critical time step for DEM simulations of dynamic systems using a Hertzian contact model," *Int J Numer Methods Eng*, vol. 119, no. 5, pp. 432–451, Aug. 2019, doi: 10.1002/nme.6056.

[16] B. A. Wills and J. A. Finch, "Modeling and Characterization," in *Wills' Mineral Processing Technology*, 2016, pp. 449–462. doi: 10.1016/b978-0-08-097053-0.00017-0.

[17] P. A. Cundall and O. D. L. Strack, "A discrete numerical model for granular assemblies," *Geotechnique*, 1979, doi: 10.1680/geot.1979.29.1.47.

[18] J. Zhang, Y. Tan, T. Bao, Y. Xu, X. Xiao, and S. Jiang, "Discrete element simulation of the effect of roller-spreading parameters on powder-bed density in additive manufacturing," *Materials*, vol. 13, no. 10, May 2020, doi: 10.3390/ma13102285.

[19] Altair Engineering Inc., "EDEM 2022.2," 2022. https://help.altair.com/EDEM/index.htm#t=Introducing_EDEM.htm (accessed Feb. 10, 2023).

[20] A. P. Thompson *et al.*, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Comput Phys Commun*, vol. 271, p. 108171, Feb. 2022, doi: 10.1016/J.CPC.2021.108171.

[21] DCS Computing GmbH, "Aspherix DEM." https://www.aspherix-dem.com/ (accessed Mar. 25, 2023).

[22] C. Kloss and C. Goniva, “LIGGGHTS - Open Source Discrete Element Simulations of Granular Materials Based on Lammps,” in *Supplemental Proceedings*, Hoboken, NJ, USA: John Wiley & Sons, Inc., 2011, pp. 781–788. doi: 10.1002/9781118062142.ch94.

[23] BECKER 3D GmbH, “ThreeParticle/CAE - Discrete Element Method (DEM) Software.” https://www.becker3d.com/ (accessed Mar. 25, 2023).

[24] T. Weinhart *et al.*, “Fast, flexible particle simulations — An introduction to MercuryDPM,” *Comput Phys Commun*, vol. 249, p. 107129, Apr. 2020, doi: 10.1016/J.CPC.2019.107129.

[25] ESSS, “Rocky DEM - The Most Powerful Particle Simulation Software.” https://rocky.esss.co/ (accessed Mar. 25, 2023).

[26] Šmilauer V. et al., “Yade Documentation 3rd ed. The Yade Project.,” *Documentation*, vol. 3, p. 876, Nov. 2021, doi: 10.5281/zenodo.5705394.

[27] Inc. US Minneapolis - Itasca Consulting Group, “PFC.” https://www.itascacg.com/software/PFC (accessed Mar. 25, 2023).

[28] N. Govender, D. N. Wilke, and S. Kok, “Blaze-DEMGPU: Modular high performance DEM framework for the GPU architecture,” *SoftwareX*, vol. 5, pp. 62–66, Jan. 2016, doi: 10.1016/J.SOFTX.2016.04.004.

[29] M. Dosta and V. Skorych, “MUSEN: An open-source framework for GPU-accelerated DEM simulations,” *SoftwareX*, vol. 12, p. 100618, Jul. 2020, doi: 10.1016/J.SOFTX.2020.100618.

[30] S. Bin Yeom, E. Ha, M. Kim, S. H. Jeong, S. J. Hwang, and D. H. Choi, “Application of the discrete element method for manufacturing process simulation in the pharmaceutical industry,” *Pharmaceutics*, vol. 11, no. 8, Aug. 2019, doi: 10.3390/pharmaceutics11080414.

[31] S. Enayatpour, R. Khaledialidusti, and T. W. Patzek, “Assessment of thermal fracturing in tight hydrocarbon formation using DEM,” in *48th US Rock Mechanics / Geomechanics Symposium 2014*, Minneapolis, Minnesota, Jun. 2014.

[32] R. D. Mindlin and H. Deresiewicz, “Elastic Spheres in Contact Under Varying Oblique Forces,” *J Appl Mech*, vol. 20, no. 3, pp. 327–344, Sep. 1953, doi: 10.1115/1.4010702.

[33] R. D. Mindlin, “Compliance of Elastic Bodies in Contact,” *J Appl Mech*, vol. 16, no. 3, pp. 259–268, Sep. 1949, doi: 10.1115/1.4009973.

[34] H. P. Zhu, Z. Y. Zhou, R. Y. Yang, and A. B. Yu, “Discrete particle simulation of particulate systems: Theoretical developments,” *Chem Eng Sci*, vol. 62, no. 13, pp. 3378–3396, Jul. 2007, doi: 10.1016/j.ces.2006.12.089.

[35] L. C. Capozzi, A. A. Barresi, and R. Pisano, “Supporting data and methods for the multi-scale modelling of freeze-drying of microparticles in packed-beds,” *Data Brief*, vol. 22, pp. 722–755, Feb. 2019, doi: 10.1016/j.dib.2018.12.061.

[36] A. Di Renzo and F. P. Di Maio, “Comparison of contact-force models for the simulation of collisions in DEM-based granular flow codes,” *Chem Eng Sci*, vol. 59, no. 3, pp. 525–541, Feb. 2004, doi: 10.1016/j.ces.2003.09.037.

[37] P. Prokopovich and S. Perni, “Comparison of JKR- and DMT-based multi-asperity adhesion model: Theory and experiment,” *Colloids Surf A Physicochem Eng Asp*, vol. 383, no. 1–3, pp. 95–101, 2011, doi: 10.1016/j.colsurfa.2011.01.011.

[38] M. Marigo and E. H. Stitt, “Discrete element method (DEM) for industrial applications: Comments on calibration and validation for the modelling of cylindrical pellets,” *KONA Powder and Particle Journal*, vol. 32, pp. 236–252, 2015, doi: 10.14356/kona.2015016.

[39] A. Yousefi and T. T. Ng, “Dimensionless input parameters in discrete element modeling and assessment of scaling techniques,” *Comput Geotech*, vol. 88, pp. 164–173, 2017, doi: 10.1016/j.compgeo.2017.03.017.

[40] C. J. Coetzee, “Particle upscaling: Calibration and validation of the discrete element method,” *Powder Technol*, vol. 344, pp. 487–503, Feb. 2019, doi: 10.1016/j.powtec.2018.12.022.

[41] H. Chen, Y. G. Xiao, Y. L. Liu, and Y. S. Shi, “Effect of Young’s modulus on DEM results regarding transverse mixing of particles within a rotating drum,” *Powder Technol*, vol. 318, pp. 507–517, Aug. 2017, doi: 10.1016/j.powtec.2017.05.047.

[42] M. Lupo, D. Barletta, D. Sofia, and M. Poletto, “Calibration of DEM for Cohesive Particles in the SLS Powder Spreading Process,” *Processes*, vol. 9, no. 10, Sep. 2021, doi: 10.3390/pr9101715.

[43] P. Tan, F. Shen, W. S. Tey, and K. Zhou, “A numerical study on the packing quality of fibre/polymer composite powder for powder bed fusion additive

manufacturing," *Virtual Phys Prototyp*, vol. 16, pp. S1–S18, May 2021, doi: 10.1080/17452759.2021.1922965.

[44] A. B. Spierings, M. Voegtlin, T. Bauer, and K. Wegener, "Powder flowability characterisation methodology for powder-bed-based metal additive manufacturing," *Progress in Additive Manufacturing*, vol. 1, no. 1–2, pp. 9–20, Jun. 2016, doi: 10.1007/s40964-015-0001-4.

[45] G. S. Riley, S. Mann, and R. O. Jesse, "Angles of repose of cohesive powders," *Journal of powder & bulk solids technology*, vol. 2, no. 4, 1978.

[46] G. Š. Parezanović, M. Lalić-Popović, S. Goločorbin-Kon, N. Todorović, N. Pavlović, and J. Jovičić-Bata, "The effect of magnesium stearate and sodium starch glycolate on powder flowability," *Acta Periodica Technologica*, vol. 50, 2019, doi: 10.2298/APT1950304S.

[47] J. L. Shi, "Solid state sintering of ceramics: Pore microstructure models, densification equations and applications," *J Mater Sci*, vol. 34, pp. 3801–3812, 1999, doi: 10.1023/A:1004600816317.

[48] J. G. J. Peelen, "Alumina : sintering and optical properties," Technische Hogeschool Eindhoven, 1977. doi: 10.6100/IR4212.

[49] K. Wawrzyk, P. Kowalczyk, S. Nosewicz, and J. Rojek, "A constitutive model and numerical simulation of sintering processes at macroscopic level," in *AIP Conference Proceedings*, Jerzy Podgórski, Ewa-Błazik Borowa, Jarosław Bęc, Tadeusz Burczyński, Mieczysław Kuczma, and Jarosław LatalskiJerzy Warmiński, Eds., Lublin, Poland: AIP Publishing, 2018. doi: 10.1063/1.5019045.

[50] D. Bouvard and R. M. McMeeking, “Deformation of interparticle necks by diffusion-controlled creep,” *Journal of the American Ceramic Society*, vol. 79, no. 3, pp. 666–672, Mar. 1996, doi: 10.1111/j.1151-2916.1996.tb07927.x.

[51] F. Parhami and R. M. McMeeking, “A network model for initial stage sintering,” *Mechanics of Materials*, vol. 27, no. 2, 1998, doi: 10.1016/S0167-6636(97)00034-3.

[52] F. Parhami, R. M. McMeeking, A. C. F. Cocks, and Z. Suo, “A model for the sintering and coarsening of rows of spherical particles,” *Mechanics of Materials*, vol. 31, no. 1, 1999, doi: 10.1016/S0167-6636(98)00049-0.

[53] R. L. COBLE, “Initial Sintering of Alumina and Hematite,” *Journal of the American Ceramic Society*, vol. 41, no. 2, 1958, doi: 10.1111/j.1151-2916.1958.tb13519.x.

[54] C. L. Martin, Z. Yan, D. Jauffres, D. Bouvard, and R. K. Bordia, “Sintered ceramics with controlled microstructures: Numerical investigations with the Discrete Element Method,” *Journal of the Ceramic Society of Japan*, vol. 124, no. 4, 2016, doi: 10.2109/jcersj2.15269.

[55] M. H. P. Teixeira *et al.*, “High heating rate sintering and microstructural evolution assessment using the discrete element method,” *Open Ceramics*, vol. 8, 2021, doi: 10.1016/j.oceram.2021.100182.

[56] S. Martin, M. Guessasma, J. Léchelle, J. Fortin, K. Saleh, and F. Adenot, “Simulation of sintering using a Non Smooth Discrete Element Method. Application to the study of rearrangement,” *Comput Mater Sci*, vol. 84, 2014, doi: 10.1016/j.commatsci.2013.11.050.

[57] B. Henrich, A. Wonisch, T. Kraft, M. Moseler, and H. Riedel, “Simulations of the influence of rearrangement during sintering,” *Acta Mater*, vol. 55, no. 2, pp. 753–762, Jan. 2007, doi: 10.1016/J.ACTAMAT.2006.09.005.

[58] M. Dosta, K. P. Furlan, V. Skorych, S. Heinrich, and R. Janssen, “Influence of pores arrangement on stability of photonic structures during sintering,” *J Eur Ceram Soc*, vol. 40, no. 13, 2020, doi: 10.1016/j.jeurceramsoc.2020.04.019.

[59] V. Iacobellis, A. Radhi, and K. Behdinan, “Discrete element model for ZrB2-SiC ceramic composite sintering,” *Compos Struct*, vol. 229, 2019, doi: 10.1016/j.compstruct.2019.111373.

[60] S. Yagi and D. Kunii, “Studies on effective thermal conductivities in packed beds,” *AIChE Journal*, vol. 3, no. 3, pp. 373–381, Sep. 1957, doi: 10.1002/aic.690030317.

[61] G. Bugeda, M. Cervera, and G. Lombera, “Numerical prediction of temperature and density distributions in selective laser sintering processes,” *Rapid Prototyp J*, vol. 5, no. 1, 1999, doi: 10.1108/13552549910251846.

[62] D. Sofia, M. Granese, D. Barletta, and M. Poletto, “Laser sintering of unimodal distributed glass powders of different size,” in *Procedia Engineering*, 2015. doi: 10.1016/j.proeng.2015.01.180.

[63] C. T. Hsu, P. Cheng, and K. W. Wong, “Modified Zehner-Schlunder models for stagnant thermal conductivity of porous media,” *Int J Heat Mass Transf*, vol. 37, no. 17, pp. 2751–2759, Nov. 1994, doi: 10.1016/0017-9310(94)90392-1.

[64] S. S. Sih and J. W. Barlow, “The prediction of the emissivity and thermal conductivity of powder beds,” *Particulate Science and Technology*, 2004, doi: 10.1080/02726350490501682.

[65] Y. Qian, Z. Han, J. H. Zhan, X. Liu, and G. Xu, “Comparative evaluation of heat conduction and radiation models for CFD simulation of heat transfer in packed beds,” *Int J Heat Mass Transf*, 2018, doi: 10.1016/j.ijheatmasstransfer.2018.06.127.

[66] L. Huang and M. S. El-Genk, “Thermal conductivity measurements of alumina powders and molded Min-K in vacuum,” *Energy Convers Manag*, 2001, doi: 10.1016/S0196-8904(00)00085-6.

[67] Y. Liang and X. Li, “A new model for heat transfer through the contact network of randomly packed granular material,” *Appl Therm Eng*, vol. 73, no. 1, 2014, doi: 10.1016/j.applthermaleng.2014.08.063.

[68] G. K. Batchelor and R. W. O’Brien, “THERMAL OR ELECTRICAL CONDUCTION THROUGH A GRANULAR MATERIAL.,” *Proc R Soc London Ser A*, vol. 355, no. 1682, 1977, doi: 10.1098/rspa.1977.0100.

[69] B. Chaudhuri, F. J. Muzzio, and M. S. Tomassone, “Modeling of heat transfer in granular flow in rotating vessels,” *Chem Eng Sci*, vol. 61, no. 19, pp. 6348–6360, Jun. 2006, doi: 10.1016/j.ces.2006.05.034.

[70] A. V. Gusarov, T. Laoui, L. Froyen, and V. I. Titov, “Contact thermal conductivity of a powder bed in selective laser sintering,” *Int J Heat Mass Transf*, vol. 46, no. 6, pp. 1103–1109, Mar. 2003, doi: 10.1016/S0017-9310(02)00370-8.

[71] C. Shuai, P. Feng, C. Gao, Y. Zhou, and S. Peng, “Simulation of dynamic temperature field during selective laser sintering of ceramic powder,” *Math Comput Model Dyn Syst*, vol. 19, no. 1, 2013, doi: 10.1080/13873954.2012.689769.

[72] C. H. Fu and Y. B. Guo, “3-Dimensional Finite Element Modeling of Selective Laser Melting Ti-6Al-4V Alloy,” in *Solid Freeform Fabrication Symposium*, David L. Bourell, Ed., Austin: University of Texas, Aug. 2014.

[73] F. R. Liu, Q. Zhang, W. P. Zhou, J. J. Zhao, and J. M. Chen, “Micro scale 3D FEM simulation on thermal evolution within the porous structure in selective laser sintering,” *J Mater Process Technol*, vol. 212, no. 10, pp. 2058–2065, 2012, doi: 10.1016/j.jmatprotec.2012.05.010.

[74] A. Hussein, L. Hao, C. Yan, and R. Everson, “Finite element simulation of the temperature and stress fields in single layers built without-support in selective laser melting,” *Mater Des*, vol. 52, pp. 638–647, 2013, doi: 10.1016/j.matdes.2013.05.070.

[75] Z. Fan, M. Lu, and H. Huang, “Selective laser melting of alumina: A single track study,” *Ceram Int*, vol. 44, no. 8, pp. 9484–9493, 2018, doi: 10.1016/j.ceramint.2018.02.166.

[76] P. Conti, F. Cianetti, and P. Pilerci, “Parametric Finite Elements Model of SLM Additive Manufacturing process,” *Procedia Structural Integrity*, vol. 8, pp. 410–421, Feb. 2018, doi: 10.1016/j.prostr.2017.12.041.

[77] C. M. Olivier, G. A. Oosthuizen, and N. Sacks, “Improving the R&D process efficiency of the selective laser sintering industry through numerical thermal modeling,” in *Procedia Manufacturing*, Günther Seliger, I.S. Jawahir, and Fazleena BadurdeenHolger Kohl, Eds., Lexington Kentucky: Elsevier Ltd, 2019, pp. 131–138. doi: 10.1016/j.promfg.2019.04.017.

[78] X. Li and W. Tan, “Numerical Investigation of Laser Absorption by Metal Powder Bed in Selective Laser Sintering Processes,” *Proceedings of the 27th Solid Freeform Fabrication Symposium (SFF)*, 2016.

[79] L. Xin, M. Boutaous, S. Xin, and D. A. Siginer, “Numerical modeling of the heating phase of the selective laser sintering process,” *International Journal of Thermal Sciences*, vol. 120, pp. 50–62, Oct. 2017, doi: 10.1016/j.ijthermalsci.2017.05.017.

[80] Y. Yang, O. Ragnvaldsen, Y. Bai, M. Yi, and B.-X. Xu, “3D non-isothermal phase-field simulation of microstructure evolution during selective laser sintering,” *NPJ Comput Mater*, vol. 5, no. 81, Aug. 2019, doi: 10.1038/s41524-019-0219-7.

[81] O. D. Quintana-Ruiz and E. M. B. Campello, “Discrete element modeling of selective laser sintering additive manufacturing processes,” *Comput Methods Appl Mech Eng*, vol. 410, no. 115994, May 2023, doi: 10.1016/j.cma.2023.115994.

[82] Z. Yan, C. L. Martin, O. Guillon, D. Bouvard, and C. S. Lee, “Microstructure evolution during the co-sintering of Ni/BaTiO3 multilayer ceramic capacitors modeled by discrete element simulations,” *J Eur Ceram Soc*, vol. 34, no. 13, 2014, doi: 10.1016/j.jeurceramsoc.2014.04.013.

[83] S. Haeri, Y. Wang, O. Ghita, and J. Sun, “Discrete element simulation and experimental study of powder spreading process in additive manufacturing,” *Powder Technol*, vol. 306, pp. 45–54, 2017, doi: 10.1016/j.powtec.2016.11.002.

[84] M. Y. Shaheen, A. R. Thornton, S. Luding, and T. Weinhart, “The influence of material and process parameters on powder spreading in additive manufacturing,” *Powder Technol*, vol. 383, pp. 564–583, May 2021.

[85] S. Haeri, “Optimisation of blade type spreaders for powder bed preparation in Additive Manufacturing using DEM simulations,” *Powder Technol*, vol. 321, pp. 94–104, 2017, doi: 10.1016/j.powtec.2017.08.011.

[86] F. Wakai and G. Okuma, “Rigid body motion of multiple particles in solid-state sintering,” *Acta Mater*, vol. 235, no. 118092, Aug. 2022, doi: 10.1016/j.actamat.2022.118092.

[87] Q. Chen, E. Juste, M. Lasgorceix, F. Petit, and A. Leriche, “Binder jetting process with ceramic powders: Influence of powder properties and printing parameters,” *Open Ceramics*, vol. 9, 2022, doi: 10.1016/j.oceram.2022.100218.

[88] P. Navarrete-Segado, C. Frances, D. Grossin, and M. Tourbin, “Tailoring hydroxyapatite microspheres by spray-drying for powder bed fusion feedstock,” *Powder Technol*, vol. 398, p. 117116, Jan. 2022, doi: 10.1016/j.powtec.2022.117116.

[89] P. Navarrete-Segado, C. Frances, M. Tourbin, C. Tenailleau, B. Duployer, and D. Grossin, “Powder bed selective laser process (sintering/melting) applied to

tailored calcium phosphate-based powders," *Addit Manuf*, vol. 50, no. 102542, Feb. 2022, doi: 10.1016/j.addma.2021.102542.

[90] AZoNetwork, "Properties: Alumina - Aluminium Oxide - Al2O3 - A Refractory Ceramic Oxide." https://www.azom.com/properties.aspx?ArticleID=52 (accessed Jul. 14, 2023).

[91] S. Geer, M. L. Bernhardt-Barry, E. J. Garboczi, J. Whiting, and A. Donmez, "A more efficient method for calibrating discrete element method parameters for simulations of metallic powder used in additive manufacturing," *Granul Matter*, vol. 20, no. 4, 2018, doi: 10.1007/s10035-018-0848-4.

[92] H. C. Tran and Y. L. Lo, "Heat transfer simulations of selective laser melting process based on volumetric heat source with powder size consideration," *J Mater Process Technol*, vol. 255, pp. 411–425, 2018, doi: 10.1016/j.jmatprotec.2017.12.024.

[93] Walter Frei, "Modeling Laser-Material Interactions with the Beer-Lambert Law," Apr. 13, 2015. https://www.comsol.com/blogs/modeling-laser-material-interactions-with-the-beer-lambert-law/ (accessed Feb. 06, 2023).

[94] J. M. Cychosz, "An introduction to ray tracing. Edited by Andrew S. Glassner, published by Academic Press, New York, 1989, 327 pages," *Computers and Graphics*. 1993. doi: 10.1016/0097-8493(93)90060-m.

[95] M. Querry, "Optical Constants, contractor report," *US Army Chemical Research, Development and Engineering Center (CRDC), Aberdeen Proving Ground, MD*, 1985.

[96] D. Holzmann, D. Holzinger, G. Unter Hesser, T. Schmidt, and G. Unther Kn€, "Hydroxyapatite nanoparticles as novel low-refractive index additives for the long-term UV-photoprotection of transparent composite materials †," *Materials Chemistry*, vol. 19, no. 43, pp. 8102–8106, 2009, doi: 10.1039/b912116a.

[97] M. Shahmoradi, R. Rohanizadeh, F. Sonvico, M. Ghadiri, and M. Swain, "Synthesis of stabilized hydroxyapatite nanosuspensions for enamel caries remineralization," *Aust Dent J*, vol. 63, no. 3, 2018, doi: 10.1111/adj.12624.

[98] I. M. Berke, J. P. Miola, M. A. David, M. K. Smith, and C. Price, "Seeing through musculoskeletal tissues: Improving in situ imaging of bone and the lacunar canalicular system through optical clearing," *PLoS One*, vol. 11, no. 3, 2016, doi: 10.1371/journal.pone.0150268.

[99] Y. A. Cengel, *Heat Transfer: A Practical Approach*. McGraw-Hill Education, 2002.

[100] Y. A. Cengel, "Property Tables and Charts," in *Heat Transfer: A Practical Approach*, 2002.

[101] J. Rojek, R. Kasztelan, and R. Tharmaraj, "Discrete element thermal conductance model for sintered particles," *Powder Technol*, vol. 405, p. 117521, Jun. 2022, doi: 10.1016/J.POWTEC.2022.117521.

[102] Y. S. Touloukian, *Thermophysical properties of high temperature solid materials. Volume 4. Oxides and their solutions and mixtures. Part 1. Simple oxygen compounds and their mixtures.* New York ; London : Macmillan (N.Y.) Collier-Macmillan, 1967, 1967. [Online]. Available: https://apps.dtic.mil/sti/citations/AD0649951

[103] “PubChem Compound Summary for CID 14781, Hydroxyapatite.,” *National Center for Biotechnology Information.*, 2022. https://pubchem.ncbi.nlm.nih.gov/compound/Hydroxyapatite (accessed Nov. 09, 2022).

[104] “Calcium hydroxyapatite - CAMEO,” Dec. 02, 2022. https://cameo.mfa.org/wiki/Calcium_hydroxyapatite (accessed Jul. 15, 2023).

[105] S. V. Dorozhkin, “Biphasic, triphasic and multiphasic calcium orthophosphates,” *Acta Biomater*, vol. 8, no. 3, pp. 963–977, 2012, doi: 10.1016/J.ACTBIO.2011.09.003.

[106] M. W. Bilton, *Nanoparticulate hydroxyapatite and calcium-based CO2 sorbents*. University of Leeds, 2012.

[107] “Hazardous Substances Data Bank (HSDB) : 5804 - PubChem.” https://pubchem.ncbi.nlm.nih.gov/source/hsdb/5804#section=Molecular-Formula (accessed Sep. 23, 2022).

[108] H. J. Busscher, H. P. de Jong, and J. Arends, “Surface free energies of hydroxyapatite, fluorapatite and calcium fluoride,” *Mater Chem Phys*, vol. 17, no. 6, 1987, doi: 10.1016/0254-0584(87)90014-9.

[109] W. Luo and J. Pan, “Effects of Surface Diffusion and Heating Rate on First-Stage Sintering That Densifies by Grain-Boundary Diffusion,” *Journal of the American Ceramic Society*, vol. 98, no. 11, pp. 3483–3489, Nov. 2015, doi: https://doi.org/10.1111/jace.13662.

[110] T. Matsuda, “Development of a DEM taking account of neck increments caused by surface diffusion for sintering and application to analysis of the initial stage of sintering,” *Comput Mater Sci*, vol. 196, 2021, doi: 10.1016/j.commatsci.2021.110525.

[111] S. K. Pratihar, M. Garg, S. Mehra, and S. Bhattacharyya, “Phase evolution and sintering kinetics of hydroxyapatite synthesized by solution combustion technique,” *J Mater Sci Mater Med*, vol. 17, no. 6, 2006, doi: 10.1007/s10856-006-8932-4.

[112] W. Luo and J. Pan, “Effects of Surface Diffusion and Heating Rate on First-Stage Sintering That Densifies by Grain-Boundary Diffusion,” *Journal of the American Ceramic Society*, vol. 98, no. 11, pp. 3483–3489, Nov. 2015, doi: https://doi.org/10.1111/jace.13662.

[113] M. Bayat *et al.*, “Keyhole-induced porosities in Laser-based Powder Bed Fusion (L-PBF) of Ti6Al4V: High-fidelity modelling and experimental validation,” *Addit Manuf*, vol. 30, 2019, doi: 10.1016/j.addma.2019.100835.

[114] E. Li, Z. Zhou, L. Wang, Q. Zheng, R. Zou, and A. Yu, “Melt pool dynamics and pores formation in multi-track studies in laser powder bed fusion process,” *Powder Technol*, vol. 405, p. 117533, Jun. 2022, doi: 10.1016/J.POWTEC.2022.117533.

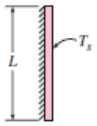
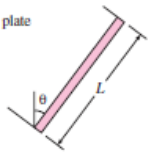
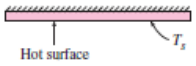
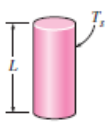
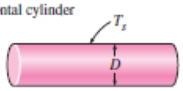
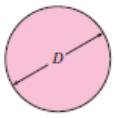
[115] N. Jiménez-Herrera, G. K. P. Barrios, and L. M. Tavares, “Comparison of breakage models in DEM in simulating impact on particle beds,” *Advanced Powder Technology*, vol. 29, no. 3, 2018, doi: 10.1016/j.apt.2017.12.006.

APPENDIX-A: TABLES

Appx-A Table 1. Yunus Cengel (2002). "Table 9-1". Heat Transfer: A Practical Approach (2nd ed). McGraw-Hill Higher Education. ISBN-13: 978-0072458930

TABLE 9–1

Empirical correlations for the average Nusselt number for natural convection over surfaces

Geometry	Characteristic length L_c	Range of Ra	Nu
Vertical plate 	L	10^4–10^9 10^9–10^{13} Entire range	$\text{Nu} = 0.59\text{Ra}_L^{1/4}$ (9-19) $\text{Nu} = 0.1\text{Ra}_L^{1/3}$ (9-20) $\text{Nu} = \left\{0.825 + \frac{0.387\text{Ra}_L^{1/6}}{[1 + (0.492/\text{Pr})^{9/16}]^{8/27}}\right\}^2$ (9-21) (complex but more accurate)
Inclined plate 	L		Use vertical plate equations for the upper surface of a cold plate and the lower surface of a hot plate Replace g by $g\cos\theta$ for $\text{Ra} < 10^9$
Horiontal plate (Surface area A and perimeter p) (a) Upper surface of a hot plate (or lower surface of a cold plate)  (b) Lower surface of a hot plate (or upper surface of a cold plate) 	A_s/p	10^4–10^7 10^7–10^{11} 10^5–10^{11}	$\text{Nu} = 0.54\text{Ra}_L^{1/4}$ (9-22) $\text{Nu} = 0.15\text{Ra}_L^{1/3}$ (9-23) $\text{Nu} = 0.27\text{Ra}_L^{1/4}$ (9-24)
Vertical cylinder 	L		A vertical cylinder can be treated as a vertical plate when $D \geq \frac{35L}{\text{Gr}_L^{1/4}}$
Horizontal cylinder 	D	$\text{Ra}_D \leq 10^{12}$	$\text{Nu} = \left\{0.6 + \frac{0.387\text{Ra}_D^{1/6}}{[1 + (0.559/\text{Pr})^{9/16}]^{8/27}}\right\}^2$ (9-25)
Sphere 	D	$\text{Ra}_D \leq 10^{11}$ ($\text{Pr} \geq 0.7$)	$\text{Nu} = 2 + \frac{0.589\text{Ra}_D^{1/4}}{[1 + (0.469/\text{Pr})^{9/16}]^{4/9}}$ (9-26)

Appx-A Table 2. Yunus Cengel (2002). "Table A-22". Heat Transfer: A Practical Approach (2nd ed). McGraw-Hill Higher Education. ISBN-13: 978-0072458930

TABLE A–22

Properties of air at 1 atm pressure

Temp. T, °C	Density ρ, kg/m^3	Specific Heat c_p, J/kg · K	Thermal Conductivity k, W/m · K	Thermal Diffusivity α, m^2/s	Dynamic Viscosity μ, kg/m · s	Kinematic Viscosity ν, m^2/s	Prandtl Number Pr
−150	2.866	983	0.01171	4.158×10^{-6}	8.636×10^{-6}	3.013×10^{-6}	0.7246
−100	2.038	966	0.01582	8.036×10^{-6}	1.189×10^{-6}	5.837×10^{-6}	0.7263
−50	1.582	999	0.01979	1.252×10^{-5}	1.474×10^{-5}	9.319×10^{-6}	0.7440
−40	1.514	1002	0.02057	1.356×10^{-5}	1.527×10^{-5}	1.008×10^{-5}	0.7436
−30	1.451	1004	0.02134	1.465×10^{-5}	1.579×10^{-5}	1.087×10^{-5}	0.7425
−20	1.394	1005	0.02211	1.578×10^{-5}	1.630×10^{-5}	1.169×10^{-5}	0.7408
−10	1.341	1006	0.02288	1.696×10^{-5}	1.680×10^{-5}	1.252×10^{-5}	0.7387
0	1.292	1006	0.02364	1.818×10^{-5}	1.729×10^{-5}	1.338×10^{-5}	0.7362
5	1.269	1006	0.02401	1.880×10^{-5}	1.754×10^{-5}	1.382×10^{-5}	0.7350
10	1.246	1006	0.02439	1.944×10^{-5}	1.778×10^{-5}	1.426×10^{-5}	0.7336
15	1.225	1007	0.02476	2.009×10^{-5}	1.802×10^{-5}	1.470×10^{-5}	0.7323
20	1.204	1007	0.02514	2.074×10^{-5}	1.825×10^{-5}	1.516×10^{-5}	0.7309
25	1.184	1007	0.02551	2.141×10^{-5}	1.849×10^{-5}	1.562×10^{-5}	0.7296
30	1.164	1007	0.02588	2.208×10^{-5}	1.872×10^{-5}	1.608×10^{-5}	0.7282
35	1.145	1007	0.02625	2.277×10^{-5}	1.895×10^{-5}	1.655×10^{-5}	0.7268
40	1.127	1007	0.02662	2.346×10^{-5}	1.918×10^{-5}	1.702×10^{-5}	0.7255
45	1.109	1007	0.02699	2.416×10^{-5}	1.941×10^{-5}	1.750×10^{-5}	0.7241
50	1.092	1007	0.02735	2.487×10^{-5}	1.963×10^{-5}	1.798×10^{-5}	0.7228
60	1.059	1007	0.02808	2.632×10^{-5}	2.008×10^{-5}	1.896×10^{-5}	0.7202
70	1.028	1007	0.02881	2.780×10^{-5}	2.052×10^{-5}	1.995×10^{-5}	0.7177
80	0.9994	1008	0.02953	2.931×10^{-5}	2.096×10^{-5}	2.097×10^{-5}	0.7154
90	0.9718	1008	0.03024	3.086×10^{-5}	2.139×10^{-5}	2.201×10^{-5}	0.7132
100	0.9458	1009	0.03095	3.243×10^{-5}	2.181×10^{-5}	2.306×10^{-5}	0.7111
120	0.8977	1011	0.03235	3.565×10^{-5}	2.264×10^{-5}	2.522×10^{-5}	0.7073
140	0.8542	1013	0.03374	3.898×10^{-5}	2.345×10^{-5}	2.745×10^{-5}	0.7041
160	0.8148	1016	0.03511	4.241×10^{-5}	2.420×10^{-5}	2.975×10^{-5}	0.7014
180	0.7788	1019	0.03646	4.593×10^{-5}	2.504×10^{-5}	3.212×10^{-5}	0.6992
200	0.7459	1023	0.03779	4.954×10^{-5}	2.577×10^{-5}	3.455×10^{-5}	0.6974
250	0.6746	1033	0.04104	5.890×10^{-5}	2.760×10^{-5}	4.091×10^{-5}	0.6946
300	0.6158	1044	0.04418	6.871×10^{-5}	2.934×10^{-5}	4.765×10^{-5}	0.6935
350	0.5664	1056	0.04721	7.892×10^{-5}	3.101×10^{-5}	5.475×10^{-5}	0.6937
400	0.5243	1069	0.05015	8.951×10^{-5}	3.261×10^{-5}	6.219×10^{-5}	0.6948
450	0.4880	1081	0.05298	1.004×10^{-4}	3.415×10^{-5}	6.997×10^{-5}	0.6965
500	0.4565	1093	0.05572	1.117×10^{-4}	3.563×10^{-5}	7.806×10^{-5}	0.6986
600	0.4042	1115	0.06093	1.352×10^{-4}	3.846×10^{-5}	9.515×10^{-5}	0.7037
700	0.3627	1135	0.06581	1.598×10^{-4}	4.111×10^{-5}	1.133×10^{-4}	0.7092
800	0.3289	1153	0.07037	1.855×10^{-4}	4.362×10^{-5}	1.326×10^{-4}	0.7149
900	0.3008	1169	0.07465	2.122×10^{-4}	4.600×10^{-5}	1.529×10^{-4}	0.7206
1000	0.2772	1184	0.07868	2.398×10^{-4}	4.826×10^{-5}	1.741×10^{-4}	0.7260
1500	0.1990	1234	0.09599	3.908×10^{-4}	5.817×10^{-5}	2.922×10^{-4}	0.7478
2000	0.1553	1264	0.11113	5.664×10^{-4}	6.630×10^{-5}	4.270×10^{-4}	0.7539

Note: For ideal gases, the properties c_p, k, μ, and Pr are independent of pressure. The properties ρ, ν, and α at a pressure P (in atm) other than 1 atm are determined by multiplying the values of ρ at the given temperature by P and by dividing ν and α by P.

Source: Data generated from the EES software developed by S. A. Klein and F. L. Alvarado. Original sources: Keenan, Chao, Keyes, Gas Tables, Wiley, 1984; and Thermophysical Properties of Matter. Vol. 3: Thermal Conductivity, Y. S. Touloukian, P. E. Liley, S. C. Saxena, Vol. 11: Viscosity, Y. S. Touloukian, S. C. Saxena, and P. Hestermans, IFI/Plenun, NY, 1970, ISBN 0-306067020-8.

APPENDIX-B: FORTRAN SUBROUTINES FOR ABAQUS CAE

```
      !DOC-3D-PRINTING
      !SUBROUTINES FOR ABAQUS
      !MATERIAL STATE + LASER HEAT-FLUX MOVEMENT

      !*********************************************************
      !SUBROUTINE: STATE INITIATION
      !*********************************************************

      SUBROUTINE UEXTERNALDB(LOP,LRESTART,TIME,DTIME,KSTEP,KINC)

      INCLUDE 'ABA_PARAM.INC'

      DIMENSION TIME(2)

      COMMON /PHASE/PHASE !PHASE: 0-POWDER, 1-SOLID+LIQUID
      REAL*8, DIMENSION(59333, 8) :: PHASE

      IF(LOP==0)THEN
          PHASE=0
      ENDIF

      RETURN
      END

      !*********************************************************
      !SUBROUTINE: USER-FIELD
      !*********************************************************

      SUBROUTINE USDFLD(FIELD,STATEV,PNEWDT,DIRECT,T,CELENT,
     1 TIME,DTIME,CMNAME,ORNAME,NFIELD,NSTATV,NOEL,NPT,LAYER,
     2 KSPT,KSTEP,KINC,NDI,NSHR,COORD,JMAC,JMATYP,MATLAYO,LACCFLA)
!
      INCLUDE 'ABA_PARAM.INC'C
!
      CHARACTER*80 CMNAME,ORNAME
      CHARACTER*3  FLGRAY(15)
      DIMENSION FIELD(NFIELD),STATEV(NSTATV),DIRECT(3,3),
     1 T(3,3),TIME(2)
      DIMENSION ARRAY(15),JARRAY(15),JMAC(*),JMATYP(*),COORD(*)

      COMMON /PHASE/PHASE !PHASE: 0-POWDER, 1-SOLID+LIQUID
      REAL*8, DIMENSION(59333, 8) :: PHASE

      FIELD(1)=PHASE(NOEL,NPT) !(ELEMENT-NUMBER,INTEGRATION-POINT)

      RETURN
      END

      !*********************************************************
      !SUBROUTINE: LASER MOVEMENT
      !*********************************************************

      SUBROUTINE DFLUX(FLUX,SOL,KSTEP,KINC,TIME,NOEL,NPT,COORDS,
     1 JLTYP,TEMP,PRESS,SNAME)

      INCLUDE 'ABA_PARAM.INC'

      DIMENSION FLUX(2), TIME(2), COORDS(3)
      CHARACTER*80 SNAME

      COMMON /PHASE/PHASE
      REAL*8, DIMENSION(59333, 8) :: PHASE

      IF(SOL>=2054)THEN
          PHASE(NOEL,NPT)=1
      ENDIF


      X=COORDS(1)
      Y=COORDS(2)
      Z=COORDS(3)

      JLTYP=1         !0: SURFACE, 1: BODY/VOLUMETRIC
```

```
X0=0.0              !0.0 m
Y0=0.0              !0.0 m
Z0=0.0              !0.0 m

!INPUT VARIABLES
DIA=0.0001          !100 um
V=0.5               !0.5 m/s
P=10                !10 W

RAD=0.5*DIA

ABP=0.96        !CO2
ABC=3300            !33.16 cm^-1

POS=V*TIME(1)

R=SQRT((X-X0)**2+(Z-Z0+POS)**2)

FLUX(1)=((2*P*ABP*ABC)/(3.142*RAD**2))*EXP(-2*(R**2)/(RAD**2))
1 *EXP(-ABC*(Y0-Y))

RETURN
END
```

APPENDIX-C: RAY-TRACING CODE FOR SURFACE ROUGHNESS

```
#include "Craytracingsurface.h"
#include "IFieldApi_1_0.h"
#include "IFieldManagerApi_1_0.h"
#include "IParticleManagerApi_1_3.h"
#include "IApiManager_1_0.h"
#include <fstream>
#include <cstring>
#include <iostream>
#include <algorithm>
#include <random>
#include <cmath>
#include <ctime>

#pragma warning(disable : 4996)

using namespace std;
using namespace NApi;
using namespace NApiCore;
using namespace NApiPbf;
using namespace NApiHelpersV3_0_0;

NApiCore::IParticleManagerApi_1_3* m_particleManager;

const string Craytracingsurface::particle_temperature        = "Temperature";
const string Craytracingsurface::heat_flux                   = "Heat Flux";

char fieldDataName[NApi::API_BASIC_STRING_LENGTH]            = "Temperature";

Craytracingsurface::Craytracingsurface(){   ;}
bool Craytracingsurface::isThreadSafe(){return true;}

bool Craytracingsurface::setup(NApiCore::IApiManager_1_0& apiManager,
        const char prefFile[],
        char customMsg[NApi::ERROR_MSG_MAX_LENGTH])
{
        //This is executed during the loading of the plugin
        //Open the preference file
        ifstream prefsFile(prefFile);

        m_particleManager =
        static_cast<NApiCore::IParticleManagerApi_1_3*>(apiManager.getApi(eParticleManager, 1, 3));

        return true;
}
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
// SIMULATION INPUT-DATA (SI UNITS):
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

ofstream output("raytracingsurface_output.txt", std::ios_base::app);
ofstream rays_distribution("raytracingsurface_raysdistribution.txt", std::ios_base::app);

const int max_rays = 1000;

double ray_x0 = 0.0;                        // Starting x-coord. of beam centre        [m];
double ray_y0 = 0.0;                        // Starting y-coord. of beam centre        [m];
double ray_z0 = 0.0005;                     // Starting z-coord. of beam centre        [m];

int const total_rays = 100;                 // Total number of rays, per row/column of beam domain;
int N;                                      // Total number of rays in the beam;

double ray_xc0[total_rays * total_rays];    // Beam start coordinate X;
double ray_yc0[total_rays * total_rays];    // Beam start coordinate Y;
double ray_zc0[total_rays * total_rays];    // Beam start coordinate Z;

double ray_xc1[total_rays * total_rays];    // Beam end coordinate X;
double ray_yc1[total_rays * total_rays];    // Beam end coordinate Y;
double ray_zc1[total_rays * total_rays];    // Beam end coordinate Z;

double drays = 1 / double(total_rays);      // Distance between adjacent rays  [m];

double a, b, c, t1, t2;
double t[2] = { 0 };                        //t1[num], t2[num];
double x_int_1, y_int_1, z_int_1;           // Intersection-1 coordinates/points       [m];
double x_int_2, y_int_2, z_int_2;           // Intersection-2 coordinates/points       [m];
double x_int[2] = { 0 }, y_int[2] = { 0 }, z_int[2] = { 0 }; // Shortlisted Intersection coordinates    [m];
double my_x_int, my_y_int, my_z_int, my_t1, my_t2;

int myptc[total_rays * total_rays] = { 0 }; // Potential number of particles to be intercepted by the ray;
double myptc_posZ = 1;                      // Arbitrary z-coord of the above particle;
double myptc_disp = 1;                      // Arbitrary ray-to-particle displacement;

int shortlist = 0;

double rtp_disp_1, rtp_disp_2;              // Ray-to-Particle displacement            [m];
double rtp_disp[2] = { 0 };

double shortest_disp = 1000;

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
// Calculate discretized cell area (uniform):
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

// ray_area = (beam_side / number of rays per row or column)^2;

double beam_side =        0.001;
double ray_area = pow((beam_side / total_rays), 2);
```

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
// UNIFORM RAYS GENERATOR / LASER BEAM DISCRETIZATION:
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

// (1) Generate uniformly distributed numbers, between 0 and 1;
// (2) Scale their positions to beam-diameter;
// (3) Offset their value by beam-radius, to distribute them on both sides of the origin.

bool Craytracingsurface::starting(NApiCore::IApiManager_1_0& apiManager, int numThreads)
{
        // This is executed at the start of the simulation run:
        //
        //mydata << "SIMULATION STARTED!" << endl << endl;

        //mydata.close();
        //rays_distribution.close();

        N =     0;
        int RR = 0;
        //double ray_energy[total_rays * total_rays][100] = { 0 };

//----------------------------------------------------------------------------------------------------------------//
// Rays formation/initialization:
//----------------------------------------------------------------------------------------------------------------//

#pragma omp parallel for

        for (int j = 0; j < total_rays; ++j)
                for (int i = 0; i < total_rays; ++i)
                {
                        ray_xc0[N] = (double(i) + 0.5) * drays * beam_side - 0.5 * beam_side + ray_x0;
                        ray_yc0[N] = (double(j) + 0.5) * drays * beam_side - 0.5 * beam_side + ray_y0;
                        ray_zc0[N] = ray_z0;
                        //
                        ray_xc1[N] = ray_xc0[N];
                        ray_yc1[N] = ray_yc0[N];
                        ray_zc1[N] = -ray_zc0[N];
                        //
                        rays_distribution << ray_xc0[N] << "        " << ray_yc0[N] << endl;
                        //
                        ++N;
                        //
                }
        return true;
}
```

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
// RAY-TRACING ALGORITHM:
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

void Craytracingsurface::getGpuFileName(char nameFile[NApi::FILE_PATH_MAX_LENGTH])
{
        ;
}

void Craytracingsurface::getPreferenceFileName(char prefFileName[FILE_PATH_MAX_LENGTH])
{
        // Copy in pref file name
        ;
}

void Craytracingsurface::stopping(NApiCore::IApiManager_1_0& apiManager)
{
        // This is executed at the end of the simulation run:

        int const total_particles = m_particleManager->getTotalNumberParticles();
        int flag = 0;

        //output << "Calculation  started" << endl;
        //output << "Total particles:     " << total_particles << endl;

        for (int ray = 0; ray < N; ray++) // Rays loop:
        {
                shortest_disp = 1000; // Arbitrarily set to large number;

                for (int i = 0; i < total_particles; i++) // Particles loop:
                {

                //------------------------------------------------------------------------------------//
                // Retrieve particle data:
                //------------------------------------------------------------------------------------//

                        double ptc_xc =   m_particleManager->getParticleData(i + 1).position.getX();
                        double ptc_yc =   m_particleManager->getParticleData(i + 1).position.getY();
                        double ptc_zc =   m_particleManager->getParticleData(i + 1).position.getZ()
                        double ptc_vol =  m_particleManager->getParticleData(i + 1).volume;
                        double ptc_rad =  pow((3.0 * ptc_vol) / (4.0 * 3.142), 1.0 / 3.0);
                        int ptc_ID =      m_particleManager->getParticleData(i + 1).ID;


                //------------------------------------------------------------------------------------//
                // Solve parametric line-sphere equations:
                //------------------------------------------------------------------------------------//

                        // a = (X1-X0)^2 + (Y1-Y0)^2 + (Z1-Z0)^2:

                        a =      (ray_xc1[ray] - ray_xc0[ray]) * (ray_xc1[ray] - ray_xc0[ray]) +
                                 (ray_yc1[ray] - ray_yc0[ray]) * (ray_yc1[ray] - ray_yc0[ray]) +
                                 (ray_zc1[ray] - ray_zc0[ray]) * (ray_zc1[ray] - ray_zc0[ray]);

                        // b = -2(X1-X0)(PX-X0) - 2(Y1-Y0)(PX-Y0) - 2(Z1-Y0)(PZ-Y0):

                        b =      - 2 * (ray_xc1[ray] - ray_xc0[ray]) * (ptc_xc - ray_xc0[ray])
                                 - 2 * (ray_yc1[ray] - ray_yc0[ray]) * (ptc_yc - ray_yc0[ray])
                                 - 2 * (ray_zc1[ray] - ray_zc0[ray]) * (ptc_zc - ray_zc0[ray]);

                        // c = (PX-X0)^2 + (PX-X0)^2 + (PX-X0)^2 – P_RAD^2

                        c =      (ptc_xc - ray_xc0[ray]) * (ptc_xc - ray_xc0[ray]) +
                                 (ptc_yc - ray_yc0[ray]) * (ptc_yc - ray_yc0[ray]) +
                                 (ptc_zc - ray_zc0[ray]) * (ptc_zc - ray_zc0[ray]) -
                                 ptc_rad * ptc_rad;

                        t[0] =   (-b + sqrt(b * b - 4 * a * c)) / (2 * a);
                        t[1] =   (-b - sqrt(b * b - 4 * a * c)) / (2 * a);
```

```
                //-------------------------------------------------------------------------------------//
                // Calculate interception-points coordinates (where applicable):
                //-------------------------------------------------------------------------------------//


                        if (b * b - 4 * a * c > 0 && t[0] > 0 && t[1] > 0)
                        {
                        shortlist += 1;

                                for (int j = 0; j < 2; j++) //Test for shorter 't': t[0] & t[1]
                                {
                                x_int[j] =      ray_xc0[ray] + (ray_xc1[ray] - ray_xc0[ray]) * t[j];
                                y_int[j] =      ray_yc0[ray] + (ray_yc1[ray] - ray_yc0[ray]) * t[j];
                                z_int[j] =      ray_zc0[ray] + (ray_zc1[ray] - ray_zc0[ray]) * t[j];

                                rtp_disp[j] =   sqrt(
                                                (ray_xc0[ray] - x_int[j]) * (ray_xc0[ray] - x_int[j]) +
                                                (ray_yc0[ray] - y_int[j]) * (ray_yc0[ray] - y_int[j]) +
                                                (ray_zc0[ray] - z_int[j]) * (ray_zc0[ray] - z_int[j]));

                                //-----------------------------------------------------------------//
                                // Set the first interception point as 'incident' point:
                                //-----------------------------------------------------------------//

                                        if (shortest_disp >= rtp_disp[j])
                                        {
                                        shortest_disp =  rtp_disp[j];

                                        my_x_int =      x_int[j];
                                        my_y_int =      y_int[j];
                                        my_z_int =      z_int[j];

                                        myptc[ray] =    ptc_ID;
                                        }
                                }

                        }

                }

                if (shortest_disp < 1000)
                {
                output << ray << "      " << ray_xc0[ray] << "    " << ray_yc0[ray] << "    " <<
                shortest_disp << "      " << myptc[ray] << endl;
                }

        }

        rays_distribution << "Calculation ended" << endl;

}

ECalculateResult Craytracingsurface::externalForce(int threadID,
        const NExternalForceTypesV3_0_0::STimeStepData& timeStepData,
        const NExternalForceTypesV3_0_0::SParticle& particle,
        NApiCore::ICustomPropertyDataApi_1_0* particleCustomProperties,
        NApiCore::ICustomPropertyDataApi_1_0* simulationCustomProperties,
        NExternalForceTypesV3_0_0::SResults& results)
{
        return eSuccess;
}
```

```
//=============================================================================================================//
// Number of properties:
//=============================================================================================================//

unsigned int Craytracingsurface::getNumberOfRequiredProperties(const NApi::EPluginPropertyCategory category)
{
        return 0;
}



//=============================================================================================================//
// Units definition:
//=============================================================================================================//

bool Craytracingsurface::getDetailsForProperty(
                                        unsigned int propertyIndex,
                                        NApi::EPluginPropertyCategory category,
                                        Char name[NApi::CUSTOM_PROP_MAX_NAME_LENGTH],
                                        NApi::EPluginPropertyDataTypes& dataType,
                                        unsigned int& numberOfElements,
                                        NApi::EPluginPropertyUnitTypes& unitType,
                                        Char initValBuff[NApi::BUFF_SIZE])
{
        return true;
}
```

APPENDIX-D: RAY-TRACING CODE FOR LASER HEATING OF POWDERS

```
#include "CHeatTransferFieldQuery.h"
#include "IFieldApi_1_0.h"
#include "IFieldManagerApi_1_0.h"
#include "IParticleManagerApi_1_3.h"
#include "IApiManager_1_0.h"
#include <fstream>
#include <cstring>
#include <iostream>
#include <algorithm>
#include <random>
#include <cmath>
#include <ctime>

#pragma warning(disable : 4996)

using namespace std;
using namespace NApi;
using namespace NApiCore;
using namespace NApiPbf;
using namespace NApiHelpersV3_0_0;

NApiCore::IParticleManagerApi_1_3* m_particleManager;

const string CHeatTransferFieldQuery::PARTICLE_TEMPERATURE  = "Temperature";
const string CHeatTransferFieldQuery::HEAT_FLUX             = "Heat Flux";

char fieldDataName[NApi::API_BASIC_STRING_LENGTH]           = "Temperature";

CHeatTransferFieldQuery::CHeatTransferFieldQuery() :
m_fieldManager(0),
m_field(0)
{
    ;
}

void CHeatTransferFieldQuery::getPreferenceFileName(char prefFileName[FILE_PATH_MAX_LENGTH])
{
    // Copy in pref file name
    strncpy(prefFileName, PREFS_FILE.c_str(), FILE_PATH_MAX_LENGTH);
}

bool CHeatTransferFieldQuery::isThreadSafe()
{
    // thread safe
    return true;
}

bool CHeatTransferFieldQuery::usesCustomProperties()
{
    // Use custom properties
    return true;
}


bool CHeatTransferFieldQuery::setup(NApiCore::IApiManager_1_0& apiManager,
                        const char prefFile[],
                        char customMsg[NApi::ERROR_MSG_MAX_LENGTH])
{
    //This is executed during the loading of the plugin
    //Open the preference file
    ifstream prefsFile(prefFile);

        m_particleManager = static_cast<NApiCore::IParticleManagerApi_1_3
*>(apiManager.getApi(eParticleManager, 1, 3));

    return true;
}
```

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
// SIMULATION INPUT-DATA (SI UNITS):
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

//----------------------------------------------------------------------------------------------------------------------------//
// Frequently used (to be checked every time!):
//----------------------------------------------------------------------------------------------------------------------------//

ofstream rays_distribution("rays_distribution.txt", std::ios_base::app);

const int max_rays = 1000;

double ray_x0 = -0.0005;                    // Starting x-coord. of beam centre         [m];
double ray_y0 = 0.0;                        // Starting y-coord. of beam centre         [m];
double ray_z0 = 0.0005;                     // Starting z-coord. of beam centre         [m];

double timestepsize = 5e-07;                // Only 'fixed' timesteps                   [s];
double laser_velocity = 0.5;                // Laser velocity                           [m/s];

double laser_start = 0.015;                 // Laser movement start time                [s];

double beam_dia = 1e-4;                     // Beam diameter                            [m];

int const total_rays = 100;                 // Total number of rays, per row/column of beam domain;

double laser_power = 50;                    // Laser power                              [W];

double ray_energy[total_rays * total_rays][max_rays] = { 0 };// Ray energy             [J];

double n1 = 1.000293;                       // Refractive-Index of Medium 1 / Fluid - Air;
double n2 = 0.61457;                        // Refractive-Index of Medium 2 / Particle - CO2 Laser;

// Nd:YAG Laser:
// double energy_trans = 0.16;              // Transmitted energy;
// double energy_rfl = 0.81;                // Reflected energy;

// CO2 Laser:
double energy_trans = 0.02;                 // Transmitted energy;
double energy_rfl = 0.02;                   // Reflected energy;

//----------------------------------------------------------------------------------------------------------------------------//
// Laser beam characteristics:
//----------------------------------------------------------------------------------------------------------------------------//

double beam_rad = 0.5 * beam_dia;           // Beam radius                              [m];
double beam_radsq = beam_rad * beam_rad;    // Beam radius squared                      [m^2];

int N;                                      // Total number of rays in the beam;

double ray_xc0[total_rays * total_rays][max_rays];  // Beam start coordinate X;
double ray_yc0[total_rays * total_rays][max_rays];  // Beam start coordinate Y;
double ray_zc0[total_rays * total_rays][max_rays];  // Beam start coordinate Z;

double ray_xc1[total_rays * total_rays][max_rays];  // Beam end coordinate X;
double ray_yc1[total_rays * total_rays][max_rays];  // Beam end coordinate Y;
double ray_zc1[total_rays * total_rays][max_rays];  // Beam end coordinate Z;


double drays = 1 / double(total_rays);      // Distance between adjacent rays  [m];

//----------------------------------------------------------------------------------------------------------------------------//
// Ray-Particle Interaction:
//----------------------------------------------------------------------------------------------------------------------------//

double rtp_disp_1, rtp_disp_2;              // Ray-to-Particle displacement             [m];
double rtp_disp[2] = { 0 };                 //double shortest_disp[num] = { 1000 };
double shortest_disp = 1000;

double a, b, c, t1, t2;
double t[2] = { 0 };                        //t1[num], t2[num];
double x_int_1, y_int_1, z_int_1;           // Intersection-1 coordinates/points        [m];
double x_int_2, y_int_2, z_int_2;           // Intersection-2 coordinates/points        [m];
double x_int[2] = { 0 }, y_int[2] = { 0 }, z_int[2] = { 0 }; // Shortlisted Intersection coordinates    [m];
double my_x_int, my_y_int, my_z_int, my_t1, my_t2;
double my_x_emg, my_y_emg, my_z_emg;        // Coordinates of emergence point           [m];

int myptc[total_rays * total_rays][max_rays] = { 0 };        // Potential number of intercepted particles;
double myptc_posZ = -1000.0;                // Arbitrary z-coord of the above particle  [m];
double myptc_disp = 1000;                   // Arbitrary ray-to-particle displacement   [m];
```

```
int SL = 0;
int SL_num = 0;

int counter = 0, counter_max = 0;

double myptc_energy[total_rays * total_rays][max_rays] = { 0 };

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
// Calculate discretized cell area (uniform):
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

// ray_area = (beam_dia / number of rays per row or column)^2;

double ray_area = pow((beam_dia / total_rays), 2);
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
// UNIFORM RAYS GENERATOR / LASER BEAM DISCRETIZATION:
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

// (1) Generate uniformly distributed numbers between 0 and 1;
// (2) Scale their positions to beam-diameter;
// (3) Offset their value by beam-radius to distribute them on both sides of the origin.

bool CHeatTransferFieldQuery::starting(NApiCore::IApiManager_1_0& apiManager, int numThreads)
{
        // This is executed at the start of the simulation run:
        //
        //mydata << "SIMULATION STARTED!" << endl << endl;

        //mydata.close();
        rays_distribution.close();

                N = 0;
                int RR = 0;
                //double ray_energy[total_rays * total_rays][100] = { 0 };

//*************************************************************************************************************//
// Rays formation/initialization:
//*************************************************************************************************************//

                #pragma omp parallel for

                for (int j = 0; j < total_rays; ++j)
                for (int i = 0; i < total_rays; ++i)
                {
                        ray_xc0[N][RR] = (double(i) + 0.5) * drays * beam_dia - beam_rad + ray_x0;
                        ray_yc0[N][RR] = (double(j) + 0.5) * drays * beam_dia - beam_rad + ray_y0;
                        ray_zc0[N][RR] = ray_z0;
                        //
                        ray_xc1[N][RR] = ray_xc0[N][RR];
                        ray_yc1[N][RR] = ray_yc0[N][RR];
                        ray_zc1[N][RR] = -ray_zc0[N][RR];
                        //

                        //-----------------------------------------------------------------------------//
                        // Ray displacement from centre:
                        //-----------------------------------------------------------------------------//

                        // dr^2 = (x - x0)^2 + (y - y0)^2;

                        double dr = sqrt (
                                          pow(ray_xc0[N][RR] - ray_x0, 2)
                                        + pow(ray_yc0[N][RR] - ray_y0, 2)
                                        );

                        //
                        if (dr < beam_rad)
                        {

                        //-----------------------------------------------------------------------------//
                        // Assign initial rays' energies:
                        //-----------------------------------------------------------------------------//

                        // e = P * t;
                        // ray_energy = (2*P / PI*rad^2) x exp(-2*dr^2 / rad^2) x ray_area x timestepsize;

                        // ray_energy[N][RR] = ((2 * laser_power) / (PI * beam_rad * beam_rad))
                        //                     * exp((-2 * // dr * dr) / (beam_rad * beam_rad));

                                ray_energy[N][RR] = timestepsize * ray_area * ((2 * laser_power)
                                                  / (PI * beam_rad * beam_rad)) * exp((-2 * dr * dr)
                                                  / (beam_rad * beam_rad));

                                rays_distribution << "Ray number: " << N << "       < " << ray_xc0[N][0]
                                << ", " << ray_yc0[N][0] << " >" << "       dr: " << dr << "  Energy: "
                                << ray_energy[N][0] << endl << endl;

                                ++N;
                                }
                        }

    return true;
}
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
// RAY-TRACING ALGORITHM:
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

// 0: To be checked, 1: Do not check / Already checked;
int rfl[total_rays * total_rays][max_rays] = { 1 };
int rfr[total_rays * total_rays][max_rays] = { 1 };


void CHeatTransferFieldQuery::configForTimeStep(NApiCore::ICustomPropertyDataApi_1_0* simData,
        NApiCore::IParticleManagerApi_1_0* particleManager,
        double time)
{
        // This is executed only 'once' at the start of each timestep:

        //int rfl[total_rays * total_rays][max_rays] = { 1 };
        //int rfr[total_rays * total_rays][max_rays] = { 1 };

        if (time >= laser_start) //To avoid unnecessary computations before laser movement starts;
        {

        int const num = m_particleManager->getTotalNumberParticles();

        // 0: To be checked, 1: Do not check / Already checked;
        // int rfl[total_rays * total_rays][max_rays] = { 1 };
        // int rfr[total_rays * total_rays][max_rays] = { 1 };

        int ray_type[total_rays * total_rays][max_rays] = { 0 };

        //myptc_energy[total_rays * total_rays][max_rays]= ;
        #pragma omp parallel for collapse(2)
        for (int ray = 0; ray < N; ray++) // Rays loop:
        {
                // ray_energy[ray][0] = 100;

                double energy_int = ray_energy[ray][0];

                counter = 0;
                int RR = 0;
                int update_velocity = 0;
                int origin_tracer = 0;     // Check the number of times algorithm reached back to 0-th RR;
                ray_type[ray][0] = 0;      //energy[ray][0] = 5;

                while (rfr[ray][0] < 2)
                {
                        if (RR == 0) { ray_type[ray][0] = 0; }
                        SL = 0;
                        myptc_disp = 1000;
                        myptc[ray][counter] = 0;
                        if (RR == 0 && update_velocity == 0)
                        {
                                // Only update original ray position, i.e. not the subsequent
                                reflections/refractions

                                ray_xc0[ray][RR] += laser_velocity * timestepsize;
                                ray_xc1[ray][RR] += laser_velocity * timestepsize;

                                // Switch set to 1 to make sure that velocity is updated only once per
                                timestep;

                                update_velocity = 1;
                                }

                                shortest_disp = 1000; // Arbitrarily set to large number;

                        //*************************************************************************//
                        // RAY ENERGY + RAY-PARTICLE INTERCEPTION CHECK:
                        //*************************************************************************//

                        if (ray_energy[ray][RR] >= 0.05 * ray_energy[ray][0])
                        {

                        for (int i = 0; i < num; i++) // Particles loop:
                        {

                        //-------------------------------------------------------------------------//
                        // Retrieve particle data:
                        //-------------------------------------------------------------------------//
```

```
double ptc_xc = m_particleManager->getParticleData(i + 1).position.getX();
double ptc_yc = m_particleManager->getParticleData(i + 1).position.getY();
double ptc_zc = m_particleManager->getParticleData(i + 1).position.getZ();
double ptc_vol = m_particleManager->getParticleData(i + 1).volume
double ptc_rad = pow((3.0 * ptc_vol) / (4.0 * 3.142), 1.0 / 3.0);
int ptc_ID = m_particleManager->getParticleData(i + 1).ID;

//-------------------------------------------------------------------------------//
// Solve parametric line-sphere equations:
//-------------------------------------------------------------------------------//

// a = (X1-X0)^2 + (Y1-Y0)^2 + (Z1-Z0)^2

a = (ray_xc1[ray][RR] - ray_xc0[ray][RR]) * (ray_xc1[ray][RR] - ray_xc0[ray][RR]) +
    (ray_yc1[ray][RR] - ray_yc0[ray][RR]) * (ray_yc1[ray][RR] - ray_yc0[ray][RR]) +
    (ray_zc1[ray][RR] - ray_zc0[ray][RR]) * (ray_zc1[ray][RR] - ray_zc0[ray][RR]);

// b = -2(X1-X0)(PX-X0) - 2(Y1-Y0)(PX-Y0) - 2(Z1-Y0)(PZ-Y0)

b = -2 * (ray_xc1[ray][RR] - ray_xc0[ray][RR]) * (ptc_xc - ray_xc0[ray][RR])
   - 2 * (ray_yc1[ray][RR] - ray_yc0[ray][RR]) * (ptc_yc - ray_yc0[ray][RR])
   - 2 * (ray_zc1[ray][RR] - ray_zc0[ray][RR]) * (ptc_zc - ray_zc0[ray][RR]);

// c = (PX-X0)^2 + (PY-Y0)^2 + (PZ-Z0)^2 - P_RAD^2

c = (ptc_xc - ray_xc0[ray][RR]) * (ptc_xc - ray_xc0[ray][RR]) +
    (ptc_yc - ray_yc0[ray][RR]) * (ptc_yc - ray_yc0[ray][RR]) +
    (ptc_zc - ray_zc0[ray][RR]) * (ptc_zc - ray_zc0[ray][RR]) -
    ptc_rad * ptc_rad;


t[0] = (-b + sqrt(b * b - 4 * a * c)) / (2 * a);
t[1] = (-b - sqrt(b * b - 4 * a * c)) / (2 * a);

//-------------------------------------------------------------------------------//
// Calculate interception-points coordinates (where applicable):
//-------------------------------------------------------------------------------//

int redetect = 0;

// Check if same particle is being detected by the reflected ray:
if (RR > 0 && ptc_ID == myptc[ray][RR - 1]) redetect = 1;

if (b * b - 4 * a * c > 0 && redetect == 0 && t[0] > 0 && t[1] > 0)
{
        SL += 1;

        for (int j = 0; j < 2; j++)
        {

        // Test for shorter 't': t1 & t2, as t[0] & t[1]:

        x_int[j] = ray_xc0[ray][RR] + (ray_xc1[ray][RR] - ray_xc0[ray][RR]) *
                   t[j];
        y_int[j] = ray_yc0[ray][RR] + (ray_yc1[ray][RR] - ray_yc0[ray][RR]) *
                   t[j];
        z_int[j] = ray_zc0[ray][RR] + (ray_zc1[ray][RR] - ray_zc0[ray][RR]) *
                   t[j];

        rtp_disp[j] = sqrt(
             (ray_xc0[ray][RR] - x_int[j]) * (ray_xc0[ray][RR] - x_int[j]) +
             (ray_yc0[ray][RR] - y_int[j]) * (ray_yc0[ray][RR] - y_int[j]) +
             (ray_zc0[ray][RR] - z_int[j]) * (ray_zc0[ray][RR] - z_int[j]));

        //-------------------------------------------------------------------//
        // Set the first interception point as 'incident' point:
        //-------------------------------------------------------------------//

                if (shortest_disp >= rtp_disp[j])
                {
                        shortest_disp = rtp_disp[j];
                        my_x_int = x_int[j];
                        my_y_int = y_int[j];
                        my_z_int = z_int[j];
                        myptc[ray][RR] = ptc_ID;
                }
        }
```

```
                    //-------------------------------------------------------------------------------//
                    // Calculate energy absorbed by the shortlisted particle:
                    //-------------------------------------------------------------------------------//

                    myptc_energy[ray][RR] = (1 - energy_rfl - energy_trans) * ray_energy[ray][RR];

                    }}}
                    else
                    {
                            SL = 0;
                    }

                    // If no particle is shortlisted, then eliminate the ray (0- Not checked):

                    if (SL == 0)
                    {
                            rfl[ray][counter] = 1; rfr[ray][counter] = 1;
                    }


//*************************************************************************************************//
// SPLIT RAY INTO REFLECTION + REFRACTION:
//*************************************************************************************************//

//-------------------------------------------------------------------------------------------------//
// Calculate normal at first interception point:
//-------------------------------------------------------------------------------------------------//

double nrm_dx_1 = 2 * (my_x_int - m_particleManager->
                        getParticleData(myptc[ray][RR]).position.getX());
double nrm_dy_1 = 2 * (my_y_int - m_particleManager->
                        getParticleData(myptc[ray][RR]).position.getY
double nrm_dz_1 = 2 * (my_z_int - m_particleManager->
                        getParticleData(myptc[ray][RR]).position.getZ());

double   nrm_mag_1 = sqrt( 4*
                        (
                        (my_x_int - m_particleManager->
                        getParticleData(myptc[ray][RR]).position.getX()) *
                        (my_x_int - m_particleManager->
                        getParticleData(myptc[ray][RR]).position.getX())

                        +

                        (my_y_int - m_particleManager->
                        getParticleData(myptc[ray][RR]).position.getY()) *
                        (my_y_int - m_particleManager->
                        getParticleData(myptc[ray][RR]).position.getY())

                        +

                        (my_z_int - m_particleManager
                        ->getParticleData(myptc[ray][RR]).position.getZ()) *
                        (my_z_int - m_particleManager->
                        getParticleData(myptc[ray][RR]).position.getZ())));
```

```
//*************************************************************************************************//
// CALCULATE REFLECTION:
//*************************************************************************************************//

if (rfl[ray][RR] == 0)
{
        ray_type[ray][counter + 1] = 1;

        //-----------------------------------------------------------------------------------------//
        // Set reflected-ray to 'checked' [1]:
        //-----------------------------------------------------------------------------------------//

        rfl[ray][RR] = 1;

        //-----------------------------------------------------------------------------------------//
        // Set incident-ray direction:
        //-----------------------------------------------------------------------------------------//

        double ray_dx = my_x_int - ray_xc0[ray][RR];
        double ray_dy = my_y_int - ray_yc0[ray][RR];
        double ray_dz = my_z_int - ray_zc0[ray][RR];
        double ray_mag = sqrt(ray_dx * ray_dx + ray_dy * ray_dy + ray_dz * ray_dz);

        //-----------------------------------------------------------------------------------------//
        // Calculate cosine for incident + reflection angles:
        //-----------------------------------------------------------------------------------------//

        // rfl = ray - (2 * (ray.dot.normal) * normal) / ||normal||^2

        double cos_theta_inc = (-1 / (ray_mag * nrm_mag_1)) *
                               ((ray_dx * nrm_dx_1) + (ray_dy * nrm_dy_1) + (ray_dz * nrm_dz_1));

        //-----------------------------------------------------------------------------------------//
        // Calculate reflection direction:
        //-----------------------------------------------------------------------------------------//

        double rfl_dx = (ray_dx / ray_mag) + 2 * cos_theta_inc * (nrm_dx_1 / nrm_mag_1);
        double rfl_dy = (ray_dy / ray_mag) + 2 * cos_theta_inc * (nrm_dy_1 / nrm_mag_1);
        double rfl_dz = (ray_dz / ray_mag) + 2 * cos_theta_inc * (nrm_dz_1 / nrm_mag_1);

        //-----------------------------------------------------------------------------------------//
        // Set reflected-ray coordinates + energy:
        //-----------------------------------------------------------------------------------------//

        ray_xc0[ray][counter + 1] = my_x_int;
        ray_yc0[ray][counter + 1] = my_y_int;
        ray_zc0[ray][counter + 1] = my_z_int;

        ray_xc1[ray][counter + 1] = my_x_int + rfl_dx;
        ray_yc1[ray][counter + 1] = my_y_int + rfl_dy;
        ray_zc1[ray][counter + 1] = my_z_int + rfl_dz;

        ray_energy[ray][counter + 1] = energy_rfl * ray_energy[ray][RR];

        RR = counter + 1;
        counter += 1;

}
```

```
//*************************************************************************************************//
// CALCULATE REFRACTION:
//*************************************************************************************************//

// 'rfr[ray][0] == 1' check is made to make sure the loop does not end prematurely

else if (rfr[ray][RR] == 0)
{
        ray_type[ray][counter + 1] = 2;

        //-------------------------------------------------------------------------------------------//
        // Set 'Total External Reflection' & 'Total Internal Reflection' to NOT DETECTED state:
        //-------------------------------------------------------------------------------------------//

        int TER_detect = 0; // Set TER Detection to '0' (NO)
        int TIR_detect = 0; // Set TIR Detection to '0' (NO)

        //-------------------------------------------------------------------------------------------//
        // Set refracted-ray to checked [1]:
        //-------------------------------------------------------------------------------------------//

        rfr[ray][RR] = 1;

        //-------------------------------------------------------------------------------------------//
        // Retrieve particle data:
        //-------------------------------------------------------------------------------------------//

        double ptc_xc = m_particleManager->getParticleData(myptc[ray][RR]).position.getX();
        double ptc_yc = m_particleManager->getParticleData(myptc[ray][RR]).position.getY();
        double ptc_zc = m_particleManager->getParticleData(myptc[ray][RR]).position.getZ();
        double ptc_vol = m_particleManager->getParticleData(myptc[ray][RR]).volume;
        double ptc_rad = pow((3.0 * ptc_vol) / (4.0 * 3.142), 1.0 / 3.0);

        //-------------------------------------------------------------------------------------------//
        // Set incident ray direction:
        //-------------------------------------------------------------------------------------------//

        double ray_dx = my_x_int - ray_xc0[ray][RR];
        double ray_dy = my_y_int - ray_yc0[ray][RR];
        double ray_dz = my_z_int - ray_zc0[ray][RR];
        double ray_mag = sqrt(ray_dx * ray_dx + ray_dy * ray_dy + ray_dz * ray_dz);

        //-------------------------------------------------------------------------------------------//
        // Calculate cosines for incident + refraction angles:
        //-------------------------------------------------------------------------------------------//

        double r1 = n1 / n2; // Ratio of refractive-indices (fluid : particle)

        double cos_theta_inc = (-1 / (ray_mag * nrm_mag_1)) * ((ray_dx * nrm_dx_1) + (ray_dy *
                               nrm_dy_1) + (ray_dz * nrm_dz_1));
        double cos_theta_rfr_sqr = 1 - (r1 * r1) * (1 - cos_theta_inc * cos_theta_inc);
        double cos_theta_rfr = sqrt(cos_theta_rfr_sqr);

        //===========================================================================================//
        // Check for total external reflection:
        //===========================================================================================//

        double pxy_rfr_dx = 0, pxy_rfr_dy = 0, pxy_rfr_dz = 0;
        double pxy_x_int = 0, pxy_y_int = 0, pxy_z_int = 0;

        //=============================================================//
        // TOTAL EXTERNAL REFLECTION DETECTED: NO
        //=============================================================//

        double longer_disp = -1000; // Arbitrarily set to small number;

        if (cos_theta_rfr_sqr >= 0) // FALSE
        {

                //-----------------------------------------------------------------------//
                // Set proxy-refracted ray direction (i.e., within the particle):
                //-----------------------------------------------------------------------//

                pxy_rfr_dx =      r1 * (ray_dx / ray_mag) +
                                  (r1 * cos_theta_inc - cos_theta_rfr) * (nrm_dx_1 / nrm_mag_1);
                pxy_rfr_dy =      r1 * (ray_dy / ray_mag) +
                                  (r1 * cos_theta_inc - cos_theta_rfr) * (nrm_dy_1 / nrm_mag_1);
                pxy_rfr_dz =      r1 * (ray_dz / ray_mag) +
                                  (r1 * cos_theta_inc - cos_theta_rfr) * (nrm_dz_1 / nrm_mag_1);
```

```
                pxy_x_int =       my_x_int + pxy_rfr_dx;
                pxy_y_int =       my_y_int + pxy_rfr_dy;
                pxy_z_int =       my_z_int + pxy_rfr_dz;

                //-------------------------------------------------------------------------------//
                // Calculate emergence point coordinates:
                //-------------------------------------------------------------------------------//

                // a = (X1-X0)^2 + (Y1-Y0)^2 + (Z1-Z0)^2:

                a =       (pxy_x_int - my_x_int) * (pxy_x_int - my_x_int) +
                          (pxy_y_int - my_y_int) * (pxy_y_int - my_y_int) +
                          (pxy_z_int - my_z_int) * (pxy_z_int - my_z_int);

                // b = -2(X1-X0)(PX-X0) - 2(Y1-Y0)(PX-Y0) - 2(Z1-Y0)(PZ-Y0):

                b =       - 2 * (pxy_x_int - my_x_int) * (ptc_xc - my_x_int)
                          - 2 * (pxy_y_int - my_y_int) * (ptc_yc - my_y_int)
                          - 2 * (pxy_z_int - my_z_int) * (ptc_zc - my_z_int);

                // c = (PX-X0)^2 + (PX-X0)^2 + (PX-X0)^2 – P_RAD^2

                c =       (ptc_xc - my_x_int) * (ptc_xc - my_x_int) +
                          (ptc_yc - my_y_int) * (ptc_yc - my_y_int) +
                          (ptc_zc - my_z_int) * (ptc_zc - my_z_int) -
                          ptc_rad * ptc_rad;


                t[0] = (-b + sqrt(b * b - 4 * a * c)) / (2 * a);
                t[1] = (-b - sqrt(b * b - 4 * a * c)) / (2 * a);

                // Test for longer 't': t1 & t2, as t[0] & t[1]:

                for (int j = 0; j < 2; j++)
                {

                          x_int[j] = my_x_int + (pxy_x_int - my_x_int) * t[j];
                          y_int[j] = my_y_int + (pxy_y_int - my_y_int) * t[j];
                          z_int[j] = my_z_int + (pxy_z_int - my_z_int) * t[j];

                          rtp_disp[j] = sqrt(
                                    (my_x_int - x_int[j]) * (my_x_int - x_int[j]) +
                                    (my_y_int - y_int[j]) * (my_y_int - y_int[j]) +
                                    (my_z_int - z_int[j]) * (my_z_int - z_int[j]));

                          //-------------------------------------------------------------------//
                          // Set the second interception point as 'emergence' point:
                          //-------------------------------------------------------------------//

                          if (longer_disp <= rtp_disp[j])
                          {
                                    longer_disp = rtp_disp[j];
                                    my_x_emg = x_int[j]; my_y_emg = y_int[j]; my_z_emg = z_int[j];
                          }
                }
}

//=============================================================//
// TOTAL EXTERNAL REFLECTION DETECTED: YES
//=============================================================//

else // TRUE
{

                //-------------------------------------------------------------------------------//
                // Set Total External Reflection (TER) to '1' (DETECTED) state:
                //-------------------------------------------------------------------------------//

                TER_detect = 1;

                //-------------------------------------------------------------------------------//
                // Calculate external reflection direction:
                //-------------------------------------------------------------------------------//

                // ref = ray + 2 * cos(theta_inc) * normal:

                double rfl_dx = (ray_dx / ray_mag) + 2 * cos_theta_inc * (nrm_dx_1 / nrm_mag_1);
                double rfl_dy = (ray_dy / ray_mag) + 2 * cos_theta_inc * (nrm_dy_1 / nrm_mag_1);
                double rfl_dz = (ray_dz / ray_mag) + 2 * cos_theta_inc * (nrm_dz_1 / nrm_mag_1);
```

```
        ray_type[ray][counter + 1] = 3;

        //-------------------------------------------------------------------------------//
        // Set external reflection ray coordinates + energy:
        //-------------------------------------------------------------------------------//

        ray_xc0[ray][counter + 1] = my_x_int;
        ray_yc0[ray][counter + 1] = my_y_int;
        ray_zc0[ray][counter + 1] = my_z_int;

        ray_xc1[ray][counter + 1] = my_x_int + rfl_dx;
        ray_yc1[ray][counter + 1] = my_y_int + rfl_dy;
        ray_zc1[ray][counter + 1] = my_z_int + rfl_dz;

        ray_energy[ray][counter + 1] = (1 - energy_rfl) * ray_energy[ray][RR];

        //-------------------------------------------------------------------------------//
        // Set particle's absorbed energy (back) to ZERO:
        //-------------------------------------------------------------------------------//

        // This is done since the ray has completely externally reflected, i.e., no
        // transmission or travel through the particle;

        myptc_energy[ray][RR] = 0;

        RR = counter + 1;
        counter += 1;
}

//=================================================================================================//
// Check for total internal reflection (if TER in NOT DETECTED):
//=================================================================================================//

if (TER_detect == 0)
{

        double nrm_dx_2 = 0, nrm_dy_2 = 0, nrm_dz_2 = 0, nrm_mag_2 = 0, ray_dot_n_2 = 0;
        double r2 = n2 / n1; // case: emergence from particle
        int attempts = 0;

        do
        {

                // Incident proxy refracted ray (within the particle) towards emergence-
                // point:

                ray_dx = my_x_emg - my_x_int;
                ray_dy = my_y_emg - my_y_int;
                ray_dz = my_z_emg - my_z_int;
                ray_mag = sqrt(ray_dx * ray_dx + ray_dy * ray_dy + ray_dz * ray_dz);

                // Emergence point normal (pointed towards centre) calculation:

                nrm_dx_2 = -2 * (my_x_emg - ptc_xc); // -2 to invert the direction
                nrm_dy_2 = -2 * (my_y_emg - ptc_xc);
                nrm_dz_2 = -2 * (my_z_emg - ptc_xc);

                nrm_mag_2 = sqrt(4 * (
                                (my_x_emg - ptc_xc) * (my_x_emg - ptc_xc)
                                +
                                (my_y_emg - ptc_yc) * (my_y_emg - ptc_yc)
                                +
                                (my_z_emg - ptc_zc) * (my_z_emg - ptc_zc)
                                ));

                // cos(theta_rfr) - This is different than for reflection:

                ray_dot_n_2 = (-1 / (ray_mag * nrm_mag_2)) * ((pxy_rfr_dx * nrm_dx_2) +
                              (pxy_rfr_dy * nrm_dy_2) + (pxy_rfr_dz * nrm_dz_2));

                //=====================================================================//
                // TOTAL INTERNAL REFLECTION DETECTED: NO
                //=====================================================================//

                if (1 - r2 * r2 * (1 - ray_dot_n_2 * ray_dot_n_2) > 0)
                {
                        break;
                } // Total internal reflection cos(theta_rfr) < 0;
```

```
                //=========================================================================//
                // TOTAL INTERNAL REFLECTION DETECTED: YES
                //=========================================================================//

                double ray_dot_n =        (ray_dx * nrm_dx_2) +
                                          (ray_dy * nrm_dy_2) +
                                          (ray_dz * nrm_dz_2);

                // rfl = ray - (2 * (ray.dot.normal) * normal) / ||normal||^2

                double rfl_dx =   ray_dx –
                                  (2 * ray_dot_n * nrm_dx_2) / (nrm_mag_2 * nrm_mag_2);
                double rfl_dy = ray_dy –
                                  (2 * ray_dot_n * nrm_dy_2) / (nrm_mag_2 * nrm_mag_2);
                double rfl_dz = ray_dz –
                                  (2 * ray_dot_n * nrm_dz_2) / (nrm_mag_2 * nrm_mag_2);

                my_x_int = my_x_emg; my_y_int = my_y_emg; my_z_int = my_z_emg;
                my_x_emg += rfl_dx; my_y_emg += rfl_dy; my_z_emg += rfl_dz;

                attempts += 1;

        } while (attempts < 5); // Make 5 attempts to test if the ray escapes from TIR;

        //=============================================================================//
        // Emergent refraction direction vector + energy:
        //=============================================================================//

        if (attempts < 5)
        {

                double ray_dx = my_x_emg - my_x_int;
                double ray_dy = my_y_emg - my_y_int;
                double ray_dz = my_z_emg - my_z_int;
                double ray_mag = sqrt(ray_dx * ray_dx + ray_dy * ray_dy + ray_dz *
                                  ray_dz);

                // -normal.dot.ray

                double cos_theta_inc =(-1 / (ray_mag * nrm_mag_2)) * ((ray_dx * nrm_dx_2)
                                      + (ray_dy * nrm_dy_2) + (ray_dz * nrm_dz_2));
                double cos_theta_rfr_sqr = 1 –
                                       (r2 * r2) * (1 - cos_theta_inc * cos_theta_inc);
                double cos_theta_rfr = sqrt(cos_theta_rfr_sqr);

                // rfr = r.dot.ray + (r_dot_c - sqrt(1 - r^2 * (1 - c^2))) * normal:

                double rfr_dx =   r2 * (ray_dx / ray_mag) +
                                  (r2 * cos_theta_inc – cos_theta_rfr) *
                                  (nrm_dx_2 / nrm_mag_2);
                double rfr_dy =   r2 * (ray_dy / ray_mag) +
                                  (r2 * cos_theta_inc - cos_theta_rfr) *
                                  (nrm_dy_2 / nrm_mag_2);
                double rfr_dz =   r2 * (ray_dz / ray_mag) +
                                  (r2 * cos_theta_inc - cos_theta_rfr) *
                                  (nrm_dz_2 / nrm_mag_2);

                ray_xc0[ray][counter + 1] = my_x_emg;
                ray_yc0[ray][counter + 1] = my_y_emg;
                ray_zc0[ray][counter + 1] = my_z_emg;

                ray_xc1[ray][counter + 1] = my_x_emg + rfr_dx;
                ray_yc1[ray][counter + 1] = my_y_emg + rfr_dy;
                ray_zc1[ray][counter + 1] = my_z_emg + rfr_dz;

                ray_energy[ray][counter + 1] = energy_trans * ray_energy[ray][RR];

                myptc_energy[ray][RR] = (1 - energy_rfl - energy_trans) *
                                        ray_energy[ray][RR];

                RR = counter + 1;
                counter += 1;
        }
        else
        {
                myptc_energy[ray][RR] = (1 - energy_rfl) * ray_energy[ray][RR];
        }
}
```

```
        }

        //*********************************************************************************************//
        //ELSE: MOVE BACK TO PREVIOUS RAY
        //*********************************************************************************************//

        // This will only be executed when both Reflection + Refraction of the given ray have been checked;

        else
        {
                --RR;
                if (RR < 0) { break; }
        }

        //---------------------- IF LOOP ENDS HERE ---------------------

} // WHILE LOOP ENDS HERE

// The check "rfr[ray][0] < 2" is made to make sure loop is not terminated without check the subsequent rays
// for rfr[ray][0];

// counter_max makes sure that maximum number of subsequent rays for any given ray are counted, so all the
// corresponding intercepted particles could be recorded;

// otherwise, the loop in the ECaclulateResult class would only run up to last recorded counter:

if (counter > counter_max) { counter_max = counter; }

}}}
```

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
// PARTICLE TEMPERATURE UPDATE:
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

ECalculateResult CHeatTransferFieldQuery::externalForce(int threadID,
                                        const NExternalForceTypesV3_0_0::STimeStepData& timeStepData,
                                        const NExternalForceTypesV3_0_0::SParticle& particle,
                                        NApiCore::ICustomPropertyDataApi_1_0* particleCustomProperties,
                                        NApiCore::ICustomPropertyDataApi_1_0* simulationCustomProperties,
                                        NExternalForceTypesV3_0_0::SResults& results)
{

const double* particleTemperature = particleCustomProperties->getValue(PARTICLE_TEMPERATURE.c_str());
double* particleTemperatureDelta = particleCustomProperties->getDelta(PARTICLE_TEMPERATURE.c_str());

if (timeStepData.time >= laser_start)
{

        //First particle intercepting the ray:

        #pragma omp parallel for collapse(2)
        for (int i = 0; i < N; i++)
        {
                for (int j = 0; j <= counter_max; j++)
                {
                        if (particle.ID == myptc[i][j])
                        {
                                double ptc_mass = m_particleManager->getParticleData(myptc[i][j]).mass;
                                double heatCapacity;

                                if (*particleTemperature < 298)
                                {
                                        heatCapacity =    782.218;
                                }
                                else if (*particleTemperature >= 298 && *particleTemperature <= 1199.85)
                                {
                                        heatCapacity =    - 6.439791e-10 * pow(*particleTemperature, 4)
                                                          + 2.2515160465e-6 * pow(*particleTemperature, 3)
                                                          - 2.9606709100e-3 * pow(*particleTemperature, 2)
                                                          + 1.8972212911167 * particleTemperature[0]
                                                          + 738.686985719856;
                                }
                                else
                                {
                                        heatCapacity =    1308.38;
                                }

                                double temperatureChange = myptc_energy[i][j] / (ptc_mass * heatCapacity);
                                *particleTemperatureDelta += temperatureChange;

                        }
                }
        }
}

return eSuccess;

}
```

```
//=============================================================================================================//
// Number of properties:
//=============================================================================================================//

unsigned int CHeatTransferFieldQuery::getNumberOfRequiredProperties(const NApi::EPluginPropertyCategory
category)
{
    if (eParticle == category)
    {
        return 2;
    }
    else
    {
        return 0;
    }
}



//=============================================================================================================//
// Units definition:
//=============================================================================================================//

bool CHeatTransferFieldQuery::getDetailsForProperty(unsigned int propertyIndex,
                                        NApi::EPluginPropertyCategory category,
                                        char name[NApi::CUSTOM_PROP_MAX_NAME_LENGTH],
                                        NApi::EPluginPropertyDataTypes& dataType,
                                        unsigned int& numberOfElements,
                                        NApi::EPluginPropertyUnitTypes& unitType,
                                        Char initValBuff[NApi::BUFF_SIZE])
{
    if (eParticle == category &&
        0 == propertyIndex)
    {
        strcpy(name, PARTICLE_TEMPERATURE.c_str());
        dataType = eDouble;
        numberOfElements = 1;
        unitType = eTemperature;
        return true;
    }
    if (eParticle == category &&
        1 == propertyIndex)
    {
        strcpy(name, HEAT_FLUX.c_str());
        dataType = eDouble;
        numberOfElements = 2;
        unitType = eHeatFlux;
        return true;
    }
}
```

APPENDIX-E: THERMAL CONDUCTIVITY CODE

```
#include <cstring>
#include <fstream>
#include <iostream>
using namespace std;

#pragma warning(disable : 4996)

using namespace NApi;
using namespace NApiCore;
using namespace NApiCm;
using namespace NApiHelpersV3_0_0;

//conduction custom properties
const std::string CHeatConduction::PARTICLE_TEMPERATURE =    "Temperature";
const std::string CHeatConduction::HEAT_FLUX =               "Heat Flux";
const std::string CHeatConduction::RESET_PARAM =             "Reset Parameter";

bool CHeatConduction::isThreadSafe()
{
    // thread safe
    return true;
}

bool CHeatConduction::usesCustomProperties()
{
    // Uses custom properties
    return true;
}

NApi::EPluginModelType CHeatConduction::getModelType()
{
    return EPluginModelType::eOptional;
}

NApi::EPluginExecutionChainPosition CHeatConduction::getExecutionChainPosition()
{
    return EPluginExecutionChainPosition::eAfterBasePos;
}

bool CHeatConduction::setup(NApiCore::IApiManager_1_0& apiManager,
                            const char prefFile[],
                            char customMsg[NApi::ERROR_MSG_MAX_LENGTH])
{
        return true;
}
```

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
// RESET HEAT-FLUX TO 'ZERO':
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

void CHeatConduction::configForTimeStep(
        NApiCore::ICustomPropertyDataApi_1_0* simPropData,
        NApiCore::IParticleManagerApi_1_3* particleManager,
        NApiCore::IGeometryManagerApi_1_2* geometryManager,
        double time)
{
        // IMPORTANT: Heat-Flux must be set to 'zero' at the start of each time-step.
        // This is done to avoid accumulation from the previous time-step.

        particleManager->resetCustomProperty("HAP_20-40", HEAT_FLUX.c_str(), 0.0);
        particleManager->resetCustomProperty("HAP_40-120", HEAT_FLUX.c_str(), 0.0);
        particleManager->resetCustomProperty("PTC_50um", HEAT_FLUX.c_str(), 0.0);
        particleManager->resetCustomProperty("Alumina", HEAT_FLUX.c_str(), 0.0);
        particleManager->resetCustomProperty("HAP", HEAT_FLUX.c_str(), 0.0);
        geometryManager->resetCustomProperty("HAP_Substrate", HEAT_FLUX.c_str(), 0.0);
        particleManager->resetCustomProperty("PTC_50um", RESET_PARAM.c_str(), 0.0);
}

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//
// HEAT-CONDUCTION CALCULATION:
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++//

double resistivityFactor = 0;

ECalculateResult CHeatConduction::calculateForce(
                Int                                             threadID,
                const NCalcForceTypesV3_0_0::STimeStepData&     timeStepData,
                const NCalcForceTypesV3_0_0::SDiscreteElement&  element1,
                NApiCore::ICustomPropertyDataApi_1_0*           elem1CustomProperties,
                const NCalcForceTypesV3_0_0::SDiscreteElement&  element2,
                NApiCore::ICustomPropertyDataApi_1_0*           elem2CustomProperties,
                NApiCore::ICustomPropertyDataApi_1_0*           contactCustomProperties,
                NApiCore::ICustomPropertyDataApi_1_0*           simulationCustomProperties,
                const NCalcForceTypesV3_0_0::SInteraction&      interaction,
                const NCalcForceTypesV3_0_0::SContact&          contact,
                NApiHelpersV3_0_0::CSimple3DVector&             tangentialPhysicalOverlap,
                NCalcForceTypesV3_0_0::SContactResult&          contactResults)
{

        //*************************************************************************************************
        // HEAT CONDUCTION MODEL:
        //*************************************************************************************************//

        //=============================================================//
        // ELEMENT-2 IS SPHERE (I.E. PARTICLE): YES
        //=============================================================//

        if (contact.normalPhysicalOverlap < 0)
        {
                return eSuccess;

        }

        // Cannot use this on surface-geometry contacts

        if (element2.isSphere != 0)
        {
                //-----------------------------------------------------------------------------------//
                // Obtain/get current (time-step's) temperatures for each element:
                //-----------------------------------------------------------------------------------//

                const double* elem1Temp = elem1CustomProperties->getValue(PARTICLE_TEMPERATURE.c_str());
                const double* elem2Temp = elem2CustomProperties->getValue(PARTICLE_TEMPERATURE.c_str());

                double thermalConductivity = 1;
                double scalingParameter = 1;
                double resistivityFactorBase = 0.90;
                double lowerOLLimit = 0.03; // lower overlap limit
                double upperOLLimit = 0.05; // upper overlap limit

                if (contact.normalPhysicalOverlap <= lowerOLLimit * equivRadius(element1, element2))
                {
                        resistivityFactor = resistivityFactorBase;
                }
```

```
                else if (contact.normalPhysicalOverlap > lowerOLLimit* equivRadius(element1, element2) &&
                contact.normalPhysicalOverlap < upperOLLimit * equivRadius(element1, element2))
                {
                        resistivityFactor = resistivityFactorBase
                        + ((1 - resistivityFactorBase) / (upperOLLimit - lowerOLLimit)) *
                        ((contact.normalPhysicalOverlap / equivRadius(element1, element2)) - lowerOLLimit);
                }
                else
                {
                        resistivityFactor = 1;
                }

                double contactRadius = sqrt(2 * equivRadius(element1, element2) *
                contact.normalPhysicalOverlap);

                double heatCoefficient = 2 * scalingParameter * contactRadius * resistivityFactor *
                thermalConductivity;

                //-----------------------------------------------------------------------------------------//
                // Check whether this is sink or source:
                //-----------------------------------------------------------------------------------------//

                double change = heatCoefficient * (*elem2Temp - *elem1Temp);

                //-----------------------------------------------------------------------------------------//
                // Update heat fluxes:
                //-----------------------------------------------------------------------------------------//

                const double* elem1FluxVal = elem1CustomProperties->getValue(HEAT_FLUX.c_str());
                const double* elem2FluxVal = elem2CustomProperties->getValue(HEAT_FLUX.c_str());

                double* elem1Flux = elem1CustomProperties->getDelta(HEAT_FLUX.c_str());
                double* elem2Flux = elem2CustomProperties->getDelta(HEAT_FLUX.c_str());

                *elem1Flux += change;
                *elem2Flux -= change;

        }

    return eSuccess;
}
```

APPENDIX-F: THERMAL LOSS CODE

```
ECalculateResult CHeatLoss::externalForce(
        int threadID,
        const NExternalForceTypesV3_0_0::STimeStepData& timeStepData,
        const NExternalForceTypesV3_0_0::SParticle& particle,
        NApiCore::ICustomPropertyDataApi_1_0* particleCustomProperties,
        NApiCore::ICustomPropertyDataApi_1_0* simulationCustomProperties,
        NExternalForceTypesV3_0_0::SResults& results)
{
        heatloss.close();

        const double* temp_surface = particleCustomProperties->getValue(Particle_Temperature.c_str());
        double* temp_surface_delta = particleCustomProperties->getDelta(Particle_Temperature.c_str());

        double temp_ambient = 298.15;
        double temp_particle = *temp_surface;

        //-------------------------------------------------------------------------------------------//
        // Heat loss condition:
        //-------------------------------------------------------------------------------------------//

        double particle_rad = cbrt(0.75 * particle.volume / PI);
        double surface_area = 2 * PI * particle_rad * particle_rad;

        double powderbedthickness = 200e-6;

        // The following condition is only valid when the top surface of the powder bed is exactly aligned
        with the origin.

        // Redefine the condition otherwise.

        if (abs(particle.position.getZ()) >= particle_rad)
        {
                return eSuccess;
        }

        //-------------------------------------------------------------------------------------------//
        // Heat loss calculation:
        //-------------------------------------------------------------------------------------------//

        double mass_rescaling     = 1e14;
        double SOF                                = 1000;
        double heat_capacity      = 628.02;
        double emmisivity         = 0.3;

        double boltzmann_constant = 1.38064852e-23;

        double heat_coefficient           = 1.2756 * log(temp_particle - 273.15) - 0.7909;

        double heat_loss_convection       = (heat_coefficient * surface_area * (temp_particle -
                                          temp_ambient));

        double heat_loss_radiation        = emmisivity * surface_area * boltzmann_constant *
                                          (pow(*temp_surface, 4) - pow(temp_ambient, 4));

        double heat_loss_total            = heat_loss_convection + heat_loss_radiation;

        double temp_change                = (heat_loss_total * timeStepData.timeStep * mass_rescaling) /
                                          (particle.mass * heat_capacity * SOF);

        // Update temperature:

        *temp_surface_delta                  -= temp_change;

        return eSuccess;
}
```

APPENDIX-G: STOPPING FORCE AND TORQUE CODE

```
ECalculateResult CVelocityRegulator::externalForce(
        int threadID,
        const NExternalForceTypesV3_0_0::STimeStepData& timeStepData,
        const NExternalForceTypesV3_0_0::SParticle& particle,
        NApiCore::ICustomPropertyDataApi_1_0* particleCustomProperties,
        NApiCore::ICustomPropertyDataApi_1_0* simulationCustomProperties,
        NExternalForceTypesV3_0_0::SResults& results)
{

        //*****************************************************************************************//
        // Linear velocity stop
        //*****************************************************************************************//

        //force = mass x acceleration

        if (particle.velocity.length() > velocitylimit)
        {
                results.force -= {
                        particle.mass* (abs(particle.velocity.getX()) - abs(diff)) / timeStepData.timeStep,
                        particle.mass* (abs(particle.velocity.getY()) - abs(diff)) / timeStepData.timeStep,
                        particle.mass* (abs(particle.velocity.getZ()) - abs(diff)) / imeStepData.timeStep};
        }

        //*****************************************************************************************//
        // Angular velocity stop
        //*****************************************************************************************//

        //particle radius: Vol = (4/3)*PI*radius^3:

        double particle_radius = cbrt((3 * particle.volume) / (4 * PI));

        //particle moment of interia: I=(2/5)*mass*radius^2:

        double particle_momentofinertia = 0.4 * particle.mass * particle_radius * particle_radius;

        //torque = moment of inertia x angular acceleration

        results.torque -= {
                        particle_momentofinertia* particle.angVel.getX() / timeStepData.timeStep,
                        particle_momentofinertia* particle.angVel.getY() / timeStepData.timeStep,
                        particle_momentofinertia* particle.angVel.getZ() / timeStepData.timeStep};

        return eSuccess;
}
```

APPENDIX-H: SOLID STATE SINTERING CODE

```
#include "CBonded.h"
#include "ICustomPropertyManagerApi_1_0.h"
#include "IParticleManagerApi_1_3.h"
#include "HelpersV3_0_0.h"
#include "time.h"
#include <cstring>
#include <cmath>

#include <iostream>
#include <fstream>
#include <sstream>

using namespace std;

using namespace NApi;
using namespace NApiCore;
using namespace NApiCm;
using namespace NApiHelpersV3_0_0;

namespace
{
	const std::string GPU_FILE = "hertzMindlin_v3_1_0";

	const double sqrt_5_6 = 0.91287092917527685576161630466 8;

	double equivRadius(const NCalcForceTypesV3_0_0::SDiscreteElement& element1, const
	NCalcForceTypesV3_0_0::SDiscreteElement& element2)
	{
		return (element1.physicalRadius * element2.physicalRadius) / (element1.physicalRadius +
		element2.physicalRadius);
	}

	double sqrtEquivRadius(const NCalcForceTypesV3_0_0::SDiscreteElement& element1, const
	NCalcForceTypesV3_0_0::SDiscreteElement& element2)
	{
		return sqrt(equivRadius(element1, element2));
	}

	double equivMass(const NCalcForceTypesV3_0_0::SDiscreteElement& element1, const
	NCalcForceTypesV3_0_0::SDiscreteElement& element2)
	{
		return (element1.mass * element2.mass) / (element1.mass + element2.mass);
	}

	double sqrtEquivMass(const NCalcForceTypesV3_0_0::SDiscreteElement& element1, const
	NCalcForceTypesV3_0_0::SDiscreteElement& element2)
	{
		return sqrt(equivMass(element1, element2));
	}

	double equivYoungsMod(const NCalcForceTypesV3_0_0::SDiscreteElement& element1, const
	NCalcForceTypesV3_0_0::SDiscreteElement& element2)
	{
		return (2.0 * element1.shearModulus * element2.shearModulus) /
			(element2.shearModulus * (1.0 - element1.poisson) +
				element1.shearModulus * (1.0 - element2.poisson));
	}

	double equivShearMod(const NCalcForceTypesV3_0_0::SDiscreteElement& element1, const
	NCalcForceTypesV3_0_0::SDiscreteElement& element2)
	{
		return (element1.shearModulus * element2.shearModulus) /
			(element2.shearModulus * (2.0 - element1.poisson) +
				element1.shearModulus * (2.0 - element2.poisson));
	}

	double dampingRatio(const NCalcForceTypesV3_0_0::SInteraction& interaction)
	{
		double B = 0.0;
		if (interaction.coeffRest > 0.0)
		{
			double myLog = log(interaction.coeffRest);
```

```
                double sl = myLog * myLog;
                double tl = NApiHelpersV3_0_0::PI * NApiHelpersV3_0_0::PI;
                B = -myLog / sqrt(sl + tl);
        }
        return B;
}

void dampingNormalForce(const CSimple3DVector& relVel_n,
        const CSimple3DVector& unitCPVect,
        const CSimple3DVector& F_n,
        const double equivYoungsMod,
        const double sqrtEquivRadius,
        const double sqrtEquivMass,
        const double sqrtOverlapN,
        const double dampingRatio,
        CSimple3DVector& F_nd,
        CSimple3DVector& newF_n)
{
        double S_n = 2.0 * equivYoungsMod * sqrtEquivRadius * sqrtOverlapN;
        double factor = 2.0 * sqrt_5_6 * dampingRatio * sqrt(S_n) * sqrtEquivMass *
        relVel_n.length();
        F_nd = unitCPVect * factor;

        //check for loading
        if (relVel_n.dot(unitCPVect) > 0.0)
        {
                F_nd = -F_nd;
        }

        newF_n = F_n + F_nd;
}

double tangentialStiffness(const double equivShearMod,
        const double sqrtEquivRadius,
        const double sqrtOverlapN)
{
        return 8.0 * equivShearMod * sqrtEquivRadius * sqrtOverlapN;
}

CSimple3DVector tangentialForce(const CSimple3DVector& tangentialOverlap, const double
tangentialStiffness)
{
        return CSimple3DVector(tangentialOverlap.getX() * -tangentialStiffness,
        tangentialOverlap.getY() * -tangentialStiffness, tangentialOverlap.getZ() * -
        tangentialStiffness);
}


void dampingTangentialForce(const CSimple3DVector& relVel_t,
        const NCalcForceTypesV3_0_0::SInteraction& interaction,
        const CSimple3DVector& F_n,
        const CSimple3DVector& F_t,
        const double tangentialStiffness,
        const double sqrtEquivMass,
        const double dampingRatio,
        CSimple3DVector& tangentialOverlap,
        CSimple3DVector& F_td,
        CSimple3DVector& newF_t)
{
        double ft2 = F_t.lengthSquared();
        double fn2 = F_n.lengthSquared() * interaction.staticFriction * interaction.staticFriction;

        if (ft2 > fn2)
        {
                newF_t = F_t * sqrt(fn2 / ft2);
                                tangentialOverlap = newF_t * (-1.0 / tangentialStiffness);
        //slippage has occurred so the tangential overlap is reduced a bit


        //at this point we get energy loss from the sliding!
                F_td = newF_t;
        }
        else
        {
                //at this point we get energy loss from the damping!
                F_td = relVel_t * (-2.0 * sqrt_5_6 * dampingRatio * sqrt(tangentialStiffness) *
                sqrtEquivMass);
                newF_t = F_t + F_td;
        }
```

```
        }
}

NApiCore::IParticleManagerApi_1_3* m_particleManager;

//Add the name for the preference file
const string CBonded::PREFS_FILE = "bonded_particle_prefs.txt";

//Add the name for the openCl file
const string CBonded::GPU_FILE = "BondedParticle_v3_1_0";

//Add the name for the custom property
const string CBonded::BOND_STATUS = "BondStatus";
const string CBonded::BOND_NORMAL_FORCE = "NormBondForce";
const string CBonded::BOND_TANGENTIAL_FORCE = "TangBondForce";
const string CBonded::BOND_NORMAL_TORQUE = "NormBondTorque";
const string CBonded::BOND_TANGENTIAL_TORQUE = "TangBondTorque";
const string CBonded::BOND_SEPERATION = "BondSeperation";
const string CBonded::BOND_PREFS = "BondProps";
const string CBonded::PARTICLE_TEMPERATURE = "Temperature";
const string CBonded::OFFSET_OVERLAP_STATUS = "OffsetOverlapStatus";
const string CBonded::initialOverlap_Status = "initialOverlapStatus";
const string CBonded::sinter_status = "SinterStatus";

char fieldDataName[NApi::API_BASIC_STRING_LENGTH] = "Temperature";

CBonded::CBonded() :
                m_requestedBondTime(0.0)
{
    ;
}

CBonded::~CBonded()
{

}

void CBonded::getPreferenceFileName(char prefFileName[FILE_PATH_MAX_LENGTH])
{
    //Copy in the name of the preference file
    //Example code
    strncpy(prefFileName, PREFS_FILE.c_str(), FILE_PATH_MAX_LENGTH);
}

bool CBonded::isThreadSafe()
{
    //Thread safe
    return true;
}

bool CBonded::usesCustomProperties()
{
    //Does not use custom properties
    return true;
}

NApi::EPluginModelType CBonded::getModelType()
{
    return EPluginModelType::eOptional;
}

NApi::EPluginExecutionChainPosition CBonded::getExecutionChainPosition()
{
    return EPluginExecutionChainPosition::eAfterBasePos;
}

void CBonded::setFilePath(const char simFile[])
{
    ;
}

void CBonded::getGpuFileName(char nameFile[NApi::FILE_PATH_MAX_LENGTH])
{
    strncpy(nameFile, GPU_FILE.c_str(), NApi::FILE_PATH_MAX_LENGTH);
}

bool CBonded::setup(NApiCore::IApiManager_1_0& apiManager,
        const char                 prefFile[],
        char                       customMsg[NApi::ERROR_MSG_MAX_LENGTH])
```

```
{
        ifstream prefsFile(prefFile);

        m_particleManager =
static_cast<NApiCore::IParticleManagerApi_1_3*>(apiManager.getApi(eParticleManager, 1, 3));

        if (!prefsFile)
        {

                return false;
        }
        else
        {

                srand(time(NULL));
                string description;

                //time at which the bonds form
                prefsFile >> description >> m_requestedBondTime;

                //torque feedback to be applied to particles as well as bonds? not applied in EDEM internal
                model
                prefsFile >> description >> torque_feedback;

                //want to break in compression as well as tension?
                prefsFile >> description >> break_compression;

                //bond damping value (0-1)
                prefsFile >> description >> damping_coeff;

                //bond rotation friction (typically similar values to edem rolling friction)
                prefsFile >> description >> rotation_coeff;

                //scale of particle physical radius wrt contact radius
                //if you have a very large contact radius may only want to bond to particles within x% of
radius
                prefsFile >> description >> m_contactRadiusScale;
                m_contactRadiusScale = (m_contactRadiusScale + 100) / 100;      //convert to usable format

          //read in bond type info to memory
                double nNormalStiffness, nNormalRange, nTangStiffness, nTangRange, nNormalStrength,
                nShearStrength, nBondDiskScale;
                string str;

                while (prefsFile)
                {
                        prefsFile >> str
                                >> description >> nNormalStiffness >> description >> nNormalRange
                                >> description >> nTangStiffness >> description >> nTangRange
                                >> description >> nNormalStrength
                                >> description >> nShearStrength
                                >> description >> nBondDiskScale;

                        if ((nNormalStiffness - nNormalRange) < 0.0)
                        {

                                return false;
                        }
                        if ((nTangStiffness - nTangRange) < 0.0)
                        {
                                return false;
                        }

                        string::size_type i(str.find(':'));
                        string surfA = str.substr(0, i);
                        str.erase(0, i + 1);
                        string surfB = str;

                        SBondParameters bondParams;

                        bondParams.m_nNormalStiffness = nNormalStiffness;
                        bondParams.m_nNormalRange = nNormalRange;
                        bondParams.m_nTangStiffness = nTangStiffness;
                        bondParams.m_nTangRange = nTangRange;
                        bondParams.m_nNormalStrength = nNormalStrength;
                        bondParams.m_nShearStrength = nShearStrength;
                        bondParams.m_nBondDiskScale = nBondDiskScale;

                        m_bondParameters.addBondParameters(surfA, surfB, bondParams);
```

```
                }
        }

        return true;
}

//ofstream bonddata("bonddata.txt", std::ios_base::app);

bool CBonded::starting(NApiCore::IApiManager_1_0& apiManager, int numThreads)
{
        NApiCore::ICustomPropertyManagerApi_1_0* contactCustomPropertyManager
                =
static_cast<NApiCore::ICustomPropertyManagerApi_1_0*>(apiManager.getApi(eContactCustomPropertyManager, 1,
0));

        iBOND_STATUS = contactCustomPropertyManager->getPropertyIndex(BOND_STATUS.c_str());;
        iBOND_PREFS = contactCustomPropertyManager->getPropertyIndex(BOND_PREFS.c_str());;
        iBOND_NORMAL_FORCE = contactCustomPropertyManager->getPropertyIndex(BOND_NORMAL_FORCE.c_str());;
        iBOND_TANGENTIAL_FORCE = contactCustomPropertyManager-
>getPropertyIndex(BOND_TANGENTIAL_FORCE.c_str());;
        iBOND_NORMAL_TORQUE = contactCustomPropertyManager->getPropertyIndex(BOND_NORMAL_TORQUE.c_str());;
        iBOND_TANGENTIAL_TORQUE = contactCustomPropertyManager-
>getPropertyIndex(BOND_TANGENTIAL_TORQUE.c_str());;
        iBOND_SEPERATION = contactCustomPropertyManager->getPropertyIndex(BOND_SEPERATION.c_str());;

        //

        return true;
}

void CBonded::stopping(NApiCore::IApiManager_1_0& apiManager)
{
    ;
}
```

```
//=============================================================================================================//
// Sinter:
//=============================================================================================================//

CSimple3DVector zeroVector{ 0.0, 0.0, 0.0 };

CSimple3DVector force_sinter{ zeroVector };
CSimple3DVector force_viscous{ zeroVector };
CSimple3DVector force_normal_total{ zeroVector };            // force_sinter + force_viscous
CSimple3DVector force_tangential_total{ zeroVector };
CSimple3DVector force_stop{ zeroVector };

CSimple3DVector newNormalForce;
CSimple3DVector usNormalForce;
CSimple3DVector dampedTangentialForce;
CSimple3DVector unsymTangentialForce;

double overlapOffsetFactor = 0.01;
double restrictedPhysicalOverlap = 0.0;

//=============================================================================================================//
// Sinter Parameters:
//=============================================================================================================//

double normalPhysicalOverlap_int[10000][10000] = { 0.0 };
double normalOverlap_Repulsive = 0;
double neck_radius = 0.0;
int testvariable = 0;

double surface_energy = 0.0;                // [J m^-2]
double atomic_volume = 0.0;                 // [m^3]
double boltzmann_constant = 0.0;            // [m^2 kg s^-2 K-1]
double gas_constant = 0.0;                  // Molar gas constant / Universal gas constant [J mol^-1 K^-1]
double GB_thick_diff = 0.0;                 // Grain-boundary thickness * diffusion coefficient [m^3 s^-1]
double activation_energy = 0.0;             // [J mol^-1]
double dihedral_angle = 0.0;                // [degree]
double dihedral_angle_rad = 0.0;            // [rad]
double sinter_temperature = 0.0;            // [K]
double melt_temperature = 0.0;              // [K]
double boil_temperature = 0.0;
double AV_BC = 0.0;                         // Atomic volume / Boltzmann constant (8.47 / 1.38064852) * 1e-7;


int const totalnumberofparticles = m_particleManager->getTotalNumberParticles();
int sinter_container(totalnumberofparticles);
```

```
NApi::ECalculateResult CBonded::calculateForce(int threadID,
const NCalcForceTypesV3_0_0::STimeStepData&    timeStepData,
const NCalcForceTypesV3_0_0::SDiscreteElement& element1,
NApiCore::ICustomPropertyDataApi_1_0*          elem1CustomProperties,
const NCalcForceTypesV3_0_0::SDiscreteElement& element2,
NApiCore::ICustomPropertyDataApi_1_0*          elem2CustomProperties,
NApiCore::ICustomPropertyDataApi_1_0*          contactCustomProperties,
NApiCore::ICustomPropertyDataApi_1_0*          simulationCustomProperties,
const NCalcForceTypesV3_0_0::SInteraction&     interaction,
const NCalcForceTypesV3_0_0::SContact&         contact,
NApiHelpersV3_0_0::CSimple3DVector&            tangentialPhysicalOverlap,
NCalcForceTypesV3_0_0::SContactResult&         contactResults)

//-------------------------------------------------------------------------------------------------------------//
// Retrieve custom properties:
//-------------------------------------------------------------------------------------------------------------//

// Bonding and sintering status:

const double* m_BondStatus = contactCustomProperties->getValue(iBOND_STATUS);
double* m_BondStatusDelta = contactCustomProperties->getDelta(iBOND_STATUS);

const double* m_BondSep = contactCustomProperties->getValue(iBOND_SEPERATION);
double* m_BondSepDelta = contactCustomProperties->getDelta(iBOND_SEPERATION);

const double* sinterstatus = contactCustomProperties->getValue(sinter_status.c_str());
double* sinterstatusdelta = contactCustomProperties->getDelta(sinter_status.c_str());

// Properties for static values:

const double* m_BondPrefs = contactCustomProperties->getValue(iBOND_PREFS);

// Particles temperature:

const double* elem1_temperature = elem1CustomProperties->getValue(PARTICLE_TEMPERATURE.c_str());
double* elem1_temperaturedelta = elem1CustomProperties->getDelta(PARTICLE_TEMPERATURE.c_str());

const double* elem2_temperature = elem2CustomProperties->getValue(PARTICLE_TEMPERATURE.c_str());
double* elem2_temperaturedelta = elem2CustomProperties->getDelta(PARTICLE_TEMPERATURE.c_str());

double particles_temperature = 0.5 * (*elem1_temperature + *elem2_temperature);

//check if in bond formation radius
CSimple3DVector nPos1(element1.position);
CSimple3DVector nPos2(element2.position);

//single sphere only if using this!
double nDistance = (nPos1 - nPos2).length();//center to center particle distance

// Some temporary variables
CSimple3DVector   F_bond_n,
                  F_bond_t,
                  F_bond_n_damp,
                  F_bond_t_damp,
                  F_total_n,
                  F_total_t,
                  T_bond_1,
                  T_bond_2,
                  T_total_1,
                  T_total_2;

SBondParameters bondParams = m_bondParameters.getBondParameters(element1.type, element2.type);

/*************************************************************************/
/* Making / obtaining bonds from records                                 */
/*************************************************************************/
//only particle-particle

int elem1_ID = element1.ID;
int elem2_ID = element2.ID;
int N = element1.ID;

if (contact.normalPhysicalOverlap < 0)
{
        //contactstate << "NO CONTACT DETECTED" << endl;
        return eSuccess;
}
else
{
```

```
;//contactstate << "CONTACT DETECTED" << endl;
}

if (true == element1.isSphere && true == element2.isSphere)
{
        ; //contactstate << "BOTH ELEMENTS ARE SPHERE" << endl;
}
else
{
        return eSuccess;
}

//--------------------------------------------------------------------------------------------------------------//
// Material choice:
//--------------------------------------------------------------------------------------------------------------//

int material_choice = 3;

// 1: Alumina / Al2O3
if (material_choice == 1)
{
        surface_energy = 1.1;                               // [J m^-2]
        atomic_volume = 8.47e-30;                           // [m^3]
        boltzmann_constant = 1.38064852e-23;                // [m^2 kg s^-2 K-1]
        gas_constant = 8.3145;                              // Universal gas constant [J mol^-1 K^-1]
        GB_thick_diff = 1.3e-8;           // Grain-boundary thickness * diffusion coefficient [m^3 s^-1]
        activation_energy = 4.18e5;                         // [J mol^-1]
        dihedral_angle = 130;                               // [degree]
        dihedral_angle_rad = (dihedral_angle * PI) / 180;   // [rad]
        sinter_temperature = 1773.15;                       // [K] - 1500 *C
        melt_temperature = 2327.15;                         // [K] - 2054 *C
        boil_temperature = 3250.15;                         // [K] - 2977 *C
        AV_BC = 6.13e-7;                  // Atomic volume / Boltzmann constant (8.47 / 1.38064852) * 1e-7;

        // Data reference: https://doi.org/10.1016/j.commatsci.2021.110525
}

// 2: Copper / Cu
else if (material_choice == 2)
{
        surface_energy = 1.72;                              // [J m^-2]
        atomic_volume = 1.18e-29;                           // [m^3]
        boltzmann_constant = 1.38064852e-23;                // [m^2 kg s^-2 K-1]
        gas_constant = 8.3145;                              // Universal gas constant [J mol^-1 K^-1]
        GB_thick_diff = 3.832e-19;        // Grain-boundary thickness * diffusion coefficient [m^3 s^-1]
        activation_energy = 105000;                         // [J mol^-1]
        dihedral_angle = 146;                               // [degree]
        dihedral_angle_rad = (dihedral_angle * PI) / 180;   // [rad]
        sinter_temperature = 1023.15;                       // [K] - 750 *C
        melt_temperature = 1358.15;                         // [K] - 1085 *C
        boil_temperature = 2835.15;                         // [K] - 2562 *C
        AV_BC = 8.55e-7;

        // Data reference: https://doi.org/10.1016/j.powtec.2013.05.020
}

// 3: Hydroxyapatite / HAP
else if (material_choice == 3)
{
        surface_energy = 0.05;                              // [J m^-2]
        atomic_volume = 2.65e-28;                           // [m^3]
        boltzmann_constant = 1.38064852e-23;                // [m^2 kg s^-2 K-1]
        gas_constant = 8.3145;                              // Universal gas constant [J mol^-1 K^-1]
        GB_thick_diff = 8.6e-8;           // Grain-boundary thickness * diffusion coefficient [m^3 s^-1]
        //activation_energy = 1020000;              // [J mol^-1]
        Ref: https://doi.org/10.1016/j.biomaterials.2004.05.027
        activation_energy = 438000;
        dihedral_angle = 105;                               // [degree]
        dihedral_angle_rad = (dihedral_angle * PI) / 180;   // [rad]
        sinter_temperature = 1525;                          // [K] - 1252 *C
        melt_temperature = 1943;                            // [K] - 1670 *C
        boil_temperature = 0;
        AV_BC = atomic_volume / boltzmann_constant;
}
```

```
//-----------------------------------------------------------------------------------------------------------//
// Calculate: Relative/approach velocity between the particles
//-----------------------------------------------------------------------------------------------------------//

CSimple3DVector relVel = element1.velocity - element2.velocity;

//-----------------------------------------------------------------------------------------------------------//
// Calculate: Normal and tangential components of the relative velocity
//-----------------------------------------------------------------------------------------------------------//

// Unit-vector from particle 1 to contact point:
CSimple3DVector unitCPVect = contact.contactPoint - element1.position;

// Unit-vector from particle 1 to contact point (inverted):
CSimple3DVector unitCPVect_inv = element1.position - contact.contactPoint;

unitCPVect.normalise();
unitCPVect_inv.normalise();

CSimple3DVector relVel_n = unitCPVect * unitCPVect.dot(relVel);
CSimple3DVector relVel_t = relVel - relVel_n;

//-----------------------------------------------------------------------------------------------------------//
// Calculate: Normal overlap with initial offset (δ_pl)
//-----------------------------------------------------------------------------------------------------------//

const double* initialOverlapStatus = contactCustomProperties->getValue(initialOverlap_Status.c_str());
double* initialOverlapStatusDelta  = contactCustomProperties->getDelta(initialOverlap_Status.c_str());

if (*initialOverlapStatus == 0) // <- consider replacing this condition with zero particles' velocities
(i.e., settled)
{
        normalPhysicalOverlap_int[element1.ID][element2.ID] = contact.normalPhysicalOverlap;
        normalPhysicalOverlap_int[element2.ID][element1.ID] =
                                                normalPhysicalOverlap_int[element1.ID][element2.ID];
        *initialOverlapStatusDelta += 1;
}

        // δ_HM = δn - δ_pl:
        double normalPhysicalOverlapHM = contact.normalPhysicalOverlap -
        normalPhysicalOverlap_int[element1.ID][element2.ID];

//-----------------------------------------------------------------------------------------------------------//
// Calculate: Neck radius
//-----------------------------------------------------------------------------------------------------------//

double sqrtEquivMass = ::sqrtEquivMass(element1, element2);
double equivRadius = ::equivRadius(element1, element2);

neck_radius = sqrt(2 * contact.normalContactOverlap * equivRadius);

double contact_angle = 2 * asin(neck_radius / (2 * equivRadius));

// Normalzied neck radius:
double norm_neck_radius = neck_radius / (2 * equivRadius);

// Maximum neck radius at equillibrium:
double neck_radius_max = 2 * equivRadius * sin(dihedral_angle_rad / 2);

// Normaziled maximum neck radius:
double maxnorm_neck_radius = neck_radius_max / (2 * equivRadius);

// Reference: Initial Sintering of Alumina and Hematite, R. L. Coble (1958)
```

```
//*****************************************************************************************************//
// CALCULATE HERTZ-MINDLIN:
//*****************************************************************************************************//

if (particles_temperature < sinter_temperature &&
        m_BondStatus[0] == 0.0 && *sinterstatus == 0.0)
{


        // Although a contact might exist, we may not have an actual physical overlap.
        if (contact.normalPhysicalOverlap <= 0.0)
        {
                return eSuccess;
        }

        CSimple3DVector newNormalForce;
        CSimple3DVector usNormalForce;
        CSimple3DVector dampedTangentialForce;
        CSimple3DVector unsymTangentialForce;

        // relative velocity of both surfaces at contact point
        CSimple3DVector relVel = element1.velocityAtContactPoint - element2.velocityAtContactPoint;

        CSimple3DVector unitCPVect = contact.contactPoint - element1.position;
        unitCPVect.normalise();

        //I'm going to assume that the physical and contact radius are the same for this model.
        //CSimple3DVector relVel_n = unitCPVect * unitCPVect.dot(relVel);
        //CSimple3DVector relVel_t = relVel - relVel_n;

        //double sqrtOverlapN = std::sqrt(contact.normalPhysicalOverlap);
        double sqrtOverlapN = std::sqrt(normalPhysicalOverlapHM);

        // sqrt Equivalent radius
        double sqrtEquivRadius = ::sqrtEquivRadius(element1, element2);
        // sqrt Equivalent Mass
        double sqrtEquivMass = ::sqrtEquivMass(element1, element2);
        // Effective Young's Modulus
        double equivYoungsMod = ::equivYoungsMod(element1, element2);
        // Effective Shear Modulus
        double equivShearMod = ::equivShearMod(element1, element2);

        // Normal spring constant and normal force
        double normalStiffness = (4.0 / 3.0) * equivYoungsMod * sqrtEquivRadius;
        double factor = normalStiffness * sqrtOverlapN * sqrtOverlapN * sqrtOverlapN;
        CSimple3DVector F_n = unitCPVect * -factor;

        // coeff B
        double dampingRatio = ::dampingRatio(interaction);

        // Damping calculation
        dampingNormalForce(relVel_n, unitCPVect, F_n, equivYoungsMod, sqrtEquivRadius,
        sqrtEquivMass, sqrtOverlapN, dampingRatio, usNormalForce, newNormalForce);
        // compute contact stiffness
        double tangentialStiffness = ::tangentialStiffness(equivShearMod, sqrtEquivRadius, sqrtOverlapN);

        // tangential force
        CSimple3DVector F_t = tangentialForce(tangentialPhysicalOverlap, tangentialStiffness);

        // tangential damping force
        dampingTangentialForce(relVel_t, interaction, F_n, F_t, tangentialStiffness,
        sqrtEquivMass, dampingRatio, tangentialPhysicalOverlap, unsymTangentialForce, dampedTangentialForce);

        //contactResults.normalForce += newNormalForce;
        //contactResults.usNormalForce += usNormalForce;
        //contactResults.tangentialForce += dampedTangentialForce;
        //contactResults.usTangentialForce += unsymTangentialForce;

        return eSuccess;
}
```

```
//*****************************************************************************************************//
// CALCULATE SINTER FORCE:
//*****************************************************************************************************//

else if (particles_temperature >= sinter_temperature &&
         particles_temperature <= melt_temperature &&
         norm_neck_radius <= maxnorm_neck_radius &&
         m_BondStatus[0] == 0.0)
         {

         if (*sinterstatus == 0) { *sinterstatusdelta += 1; }

         //=================================================================================================//
         // NORMAL COMPONENT
         //=================================================================================================//

         //-------------------------------------------------------------------------------------------------//
         // Calculate normal force: Sinter component
         //-------------------------------------------------------------------------------------------------//

         double alpha = 9 / 2; // Grain boundary diffusion dominance: 9/2; Surface diffusion dominance: 5/2
         double beta = 4;

         CSimple3DVector force_sinter_normal = unitCPVect * PI * surface_energy *
         (4 * (2 * equivRadius) * (1 - cos(contact_angle / 2)) + neck_radius * sin(contact_angle / 2));

         //-------------------------------------------------------------------------------------------------//
         // Calculate: Diffusion parameter
         //-------------------------------------------------------------------------------------------------//

         // Δb = (Ω/KT)*(δb*D0b)*exp(-Qb/(RT))

         double delta_b = ((atomic_volume * GB_thick_diff) / (boltzmann_constant * particles_temperature)) *
         exp(-activation_energy / (gas_constant * particles_temperature));

         //-------------------------------------------------------------------------------------------------//
         // Calculate normal force: Viscous component
         //-------------------------------------------------------------------------------------------------//

         double upper_term = PI * pow(neck_radius, 4) * relVel_n.length();
         double lower_term = 2 * beta * delta_b;

         CSimple3DVector force_viscous_normal = unitCPVect * (upper_term / lower_term);

         //-------------------------------------------------------------------------------------------------//
         // Calculate normal force: Resulting sintering force (normal)
         //-------------------------------------------------------------------------------------------------//

         force_normal_total = force_sinter_normal - force_viscous_normal;
         contactResults.normalForce += force_normal_total;

         //=================================================================================================//
         // TANGENTIAL COMPONENT
         //=================================================================================================//

         double visc_coeff = 0.1;

         CSimple3DVector force_sinter_tangential = unitCPVect * ((visc_coeff * PI * equivRadius *
         pow(norm_neck_radius, 2)) / (beta * delta_b)) * relVel_t.length();

         //-------------------------------------------------------------------------------------------------//
         // Offset: Compressive/tractive overlaps:
         //-------------------------------------------------------------------------------------------------//

         // This is done to eliminate/ignore compressive stresses;
         // Once this step is through, Hertz-Mindlin force will not be calculated for those particles that
         have started
         // to sinter even if their respective temperatures drop below sintering temperature;
         // Thus numerical instability is avoided.

         // δ_pl = δn;
         normalPhysicalOverlap_int[element1.ID][element2.ID] = contact.normalPhysicalOverlap;
         normalPhysicalOverlap_int[element2.ID][element1.ID] =
                                                  normalPhysicalOverlap_int[element1.ID][element2.ID];

         return eSuccess;
         }
```

```
//*****************************************************************************************************//
// CALCULATE REPULSIVE FORCE:
//*****************************************************************************************************//

else if (particles_temperature >= sinter_temperature &&
         particles_temperature <= melt_temperature &&
         norm_neck_radius > maxnorm_neck_radius &&
         m_BondStatus[0] == 0.0)
{

         //-------------------------------------------------------------------------------------------------//
         // Calculate: Repulsive normal stiffness
         //-------------------------------------------------------------------------------------------------//

         double equivYoungsMod = ::equivYoungsMod(element1, element2);
         double equivRadius = ::equivRadius(element1, element2);

         // K_n = 2E*√(R* δ-HM_n)
         double normalStiffness = 2 * equivYoungsMod * sqrt(equivRadius * normalPhysicalOverlapHM);

         //-------------------------------------------------------------------------------------------------//
         // Calculate: Repulsive normal overlap
         //-------------------------------------------------------------------------------------------------//

         // δ-rep_n = δ_n - δ_max
         normalOverlap_Repulsive = contact.normalPhysicalOverlap - 0.8 * equivRadius;

         //-------------------------------------------------------------------------------------------------//
         // Calculate: Beta_normal (β_n)
         //-------------------------------------------------------------------------------------------------//

         double dampingRatio = ::dampingRatio(interaction);
         double beta_n = 2.0 * sqrt_5_6 * dampingRatio * sqrt(normalStiffness) * sqrtEquivMass;

         //-------------------------------------------------------------------------------------------------//
         // Calculate: Repulsive normal force
         //-------------------------------------------------------------------------------------------------//

         CSimple3DVector force_repulsive = unitCPVect * (+1) * (normalStiffness * normalOverlap_Repulsive –
                                           beta_n * relVel.length());
         contactResults.normalForce -= force_repulsive;


}
```

```
//*************************************************************************************************//
// CALCULATE BONDING FORCE:
//*************************************************************************************************//

// This part of the code is adaptation of EDEM's original code, which has been combined with sintering

if (m_BondStatus[0] == 0.0 && norm_neck_radius > maxnorm_neck_radius && true == element2.isSphere)
{



        if (*sinterstatus == 0) { *sinterstatusdelta += 1; }

        double nBondSeperation = nDistance - (element2.physicalRadius * m_contactRadiusScale) -
        (element1.physicalRadius * m_contactRadiusScale);

        if (m_BondStatus[0] == 0.0 && nBondSeperation < 0.0) // also add dihedral angle condition!
        {
                //change bond status to set that it exists
                m_BondStatusDelta[0] += 1.0;

                //record center to center seperation
                m_BondSepDelta[0] += nDistance;

                //this is only recorded once, can later be used to check if bond is in compression/tension

                double* m_BondPrefsDelta = contactCustomProperties->getDelta(iBOND_PREFS);

                SBondParameters bondParams = m_bondParameters.getBondParameters(element1.type,
                element2.type);

                double random_number = rand() % 101; //creates random number between 0 and 100
                random_number = (random_number - 50) / 50; //number between -1 and 1

                m_BondPrefsDelta[0] = bondParams.m_nNormalStiffness + (bondParams.m_nNormalRange *
                random_number);
                m_BondPrefsDelta[1] = bondParams.m_nTangStiffness + (bondParams.m_nTangRange *
                random_number);

                //use smallest radius of two particles for contacts
                if (element1.physicalRadius <= element2.physicalRadius)
                {

                        m_BondPrefsDelta[2] = element1.physicalRadius * bondParams.m_nBondDiskScale;
                        //1 off calc for AREA
                        m_BondPrefsDelta[5] = PI * element1.physicalRadius * element1.physicalRadius *
                        bondParams.m_nBondDiskScale * bondParams.m_nBondDiskScale;
                }
                else
                {

                        m_BondPrefsDelta[2] = element2.physicalRadius * bondParams.m_nBondDiskScale;
                        //1 off calc for AREA if elem2 is smaller
                        m_BondPrefsDelta[5] = PI * element2.physicalRadius * element2.physicalRadius *
                        bondParams.m_nBondDiskScale * bondParams.m_nBondDiskScale;
                }

                m_BondPrefsDelta[3] = bondParams.m_nNormalStrength;
                m_BondPrefsDelta[4] = bondParams.m_nShearStrength;

                }

}
```

```
unitCPVect.normalise();

double nEquivMass = element1.mass * element2.mass / (element1.mass + element2.mass);


if (m_BondStatus[0] == 1.0)
{

        const double* m_NForce = contactCustomProperties->getValue(iBOND_NORMAL_FORCE);
        double* m_NForceDelta = contactCustomProperties->getDelta(iBOND_NORMAL_FORCE);
        const double* m_TForce = contactCustomProperties->getValue(iBOND_TANGENTIAL_FORCE);
        double* m_TForceDelta = contactCustomProperties->getDelta(iBOND_TANGENTIAL_FORCE);
        const double* m_NTorque = contactCustomProperties->getValue(iBOND_NORMAL_TORQUE);
        double* m_NTorqueDelta = contactCustomProperties->getDelta(iBOND_NORMAL_TORQUE);
        const double* m_TTorque = contactCustomProperties->getValue(iBOND_TANGENTIAL_TORQUE);
        double* m_TTorqueDelta = contactCustomProperties->getDelta(iBOND_TANGENTIAL_TORQUE);

        CSimple3DVector ndOverlap_n = relVel_n * timeStepData.timeStep;
        CSimple3DVector ndOverlap_t = relVel_t * timeStepData.timeStep;

        CSimple3DVector dF_n = -ndOverlap_n * m_BondPrefs[0] * m_BondPrefs[5];
        CSimple3DVector dF_t = -ndOverlap_t * m_BondPrefs[1] * m_BondPrefs[5];

        double J = 0.5 * PI * m_BondPrefs[2] * m_BondPrefs[2] * m_BondPrefs[2] * m_BondPrefs[2];
        double I = J / 2.0;

        CSimple3DVector angle = (element1.angVel - element2.angVel) * timeStepData.timeStep;
        CSimple3DVector normalAngle = unitCPVect * unitCPVect.dot(angle);
        CSimple3DVector tangAngle = angle - normalAngle;

        CSimple3DVector dT_n = -normalAngle * J * m_BondPrefs[1];
        CSimple3DVector dT_t = -tangAngle * I * m_BondPrefs[0];

        F_bond_n = CSimple3DVector(m_NForce[0], m_NForce[1], m_NForce[2]);
        F_bond_n = F_bond_n + dF_n;

        F_bond_t = CSimple3DVector(m_TForce[0], m_TForce[1], m_TForce[2]);
        F_bond_t = F_bond_t + dF_t;

        CSimple3DVector T_n = CSimple3DVector(m_NTorque[0], m_NTorque[1], m_NTorque[2]);
        T_n = T_n + dT_n;

        CSimple3DVector T_t = CSimple3DVector(m_TTorque[0], m_TTorque[1], m_TTorque[2]);
        T_t = T_t + dT_t;

        //check if bond is in compression or tension
        double sep = nDistance - m_BondSep[0];

        double maxStress;

        //* check for damage */
        if (sep <= 0.0 && break_compression != 1)
        {
                //if in compression only consider the moments when breaking bonds
                maxStress = T_t.length() / I * m_BondPrefs[2];

        }
        else
        {
                //else bond will break in compression as well as tension
                maxStress = -F_bond_n.length() / m_BondPrefs[5] + T_t.length() / I * m_BondPrefs[2];
        }

        double maxTorque = F_bond_t.length() / m_BondPrefs[5] + T_n.length() / J * m_BondPrefs[2];

        if (fabs(maxStress) <= (m_BondPrefs[3]) && fabs(maxTorque) <= (m_BondPrefs[4]))
        {
                //if bonds are not broken
                //update forces on bonds
                //N.B normal forces are still added in compression
                //BUT are not used in fracture calcs unless specified in prefs file
                //only when in tension (-ve normal)

                T_bond_1 = T_n + T_t;
                T_bond_2 = -T_bond_1;

                m_NForceDelta[0] += dF_n.getX();
                m_NForceDelta[1] += dF_n.getY();
                m_NForceDelta[2] += dF_n.getZ();
```

```
                m_TForceDelta[0] += dF_t.getX();
                m_TForceDelta[1] += dF_t.getY();
                m_TForceDelta[2] += dF_t.getZ();

                m_NTorqueDelta[0] += dT_n.getX();
                m_NTorqueDelta[1] += dT_n.getY();
                m_NTorqueDelta[2] += dT_n.getZ();

                m_TTorqueDelta[0] += dT_t.getX();
                m_TTorqueDelta[1] += dT_t.getY();
                m_TTorqueDelta[2] += dT_t.getZ();
        }
        else
        {
                F_bond_n = CSimple3DVector();
                F_bond_t = CSimple3DVector();
                m_BondStatusDelta[0] += 1.0;//set as broken
        }
}

        /*****************************************/
        /* Normal and Tangential Bond Force Damping */
        /*****************************************/

        //Simple equation: damping coeff * stiffness per unit area * area * rel_vel -
        //This is not used due to difficulty of setting damping coefficient

        //F_bond_n_damp = relVel_n *damping_coeff * m_BondPrefs[0] * m_BondPrefs[5];
        //F_bond_t_damp = relVel_t *damping_coeff * m_BondPrefs[1] * m_BondPrefs[5];

        //Reference for damping equations implemented below:
        //Y. Guo, J. Curtis, C. Wassgren, W. Ketterhagan, B. Hancock. Granular shear flows of flexible rod-
        like particles. Proceedings of Powders & Grains 2013, 8-12 July 2013, Sydney, Australia.

        F_bond_n_damp = relVel_n * damping_coeff * pow((m_BondPrefs[0] * m_BondPrefs[5] * 2.0 * nEquivMass),
        0.5);
        F_bond_t_damp = relVel_t * damping_coeff * pow((m_BondPrefs[1] * m_BondPrefs[5] * 2.0 * nEquivMass),
        0.5);

        /*****************************************/
        /* Fill in parameters we were passed in */
        /*****************************************/

        F_total_n = F_bond_n - F_bond_n_damp;
        F_total_t = F_bond_t - F_bond_t_damp;

        if (torque_feedback == 1)
        {
                T_total_1 += T_bond_1;
                T_total_2 += T_bond_2;
        }

        contactResults.normalForce += F_total_n;
        contactResults.tangentialForce += F_total_t;

        contactResults.additionalTorque1 += T_total_1;
        contactResults.additionalTorque2 += T_total_2;

        /*****************************************/
        /* Bond rotational 'friction' based on particle rolling fricton */
        //Not damping but simple implementation to help energy dissipation
        //values typically can range between 0 and 1 (1 is high damping) but better range is 0-0.1, could
        //also have value higher than 1.
        /*****************************************/

        //see Hertz-Mindling code for EDEM - rolling friction equation
        if (!isZero(element1.angVel.lengthSquared()))
        {
                CSimple3DVector torque1 = element1.angVel;
                torque1.normalise();
                double coM1toContactPointDistance = (contact.contactPoint - element1.CoM).length();
                torque1 *= -F_total_n.length() * coM1toContactPointDistance * rotation_coeff;
                contactResults.additionalTorque1 += torque1;
                contactResults.usAdditionalTorque1 += torque1;
        }

        if (!isZero(element2.angVel.lengthSquared()))
        {
                CSimple3DVector torque2 = element2.angVel;
```

```
                torque2.normalise();
                double coM2toContactPointDistance = (contact.contactPoint - element2.CoM).length();
                torque2 *= -F_total_n.length() * coM2toContactPointDistance * rotation_coeff;
                contactResults.additionalTorque2 += torque2;
                contactResults.usAdditionalTorque2 += torque2;
        }

        return eSuccess;
}
```

APPENDIX-I: GRAPHICAL RESULTS FOR ALL LASER SETTINGS

[Given from next page]

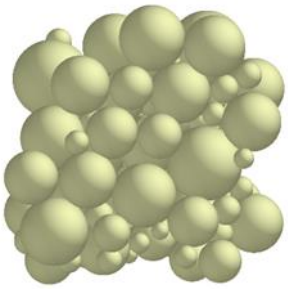
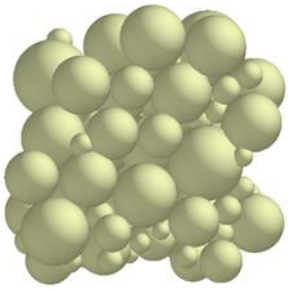
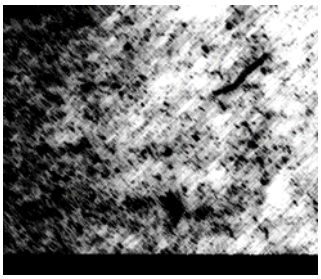
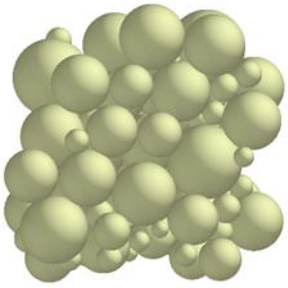
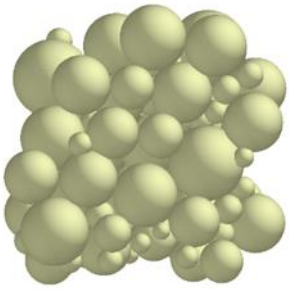
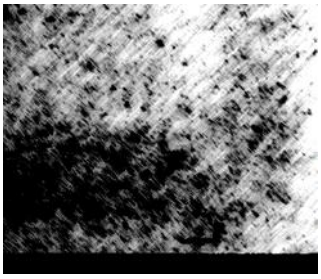
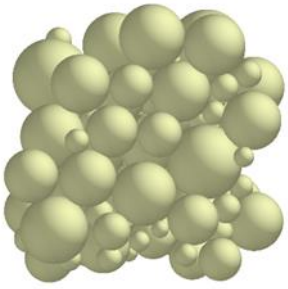
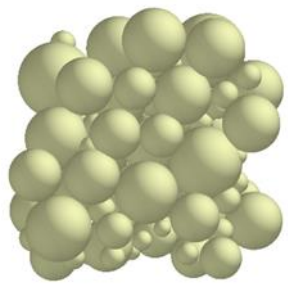
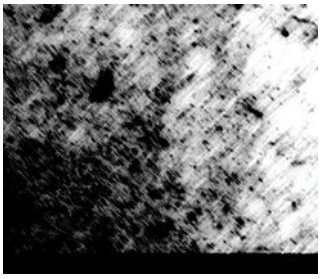
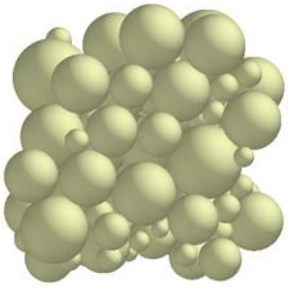
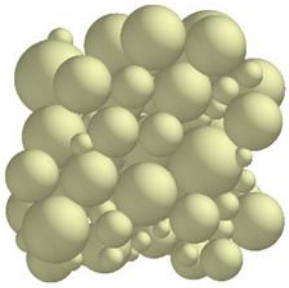
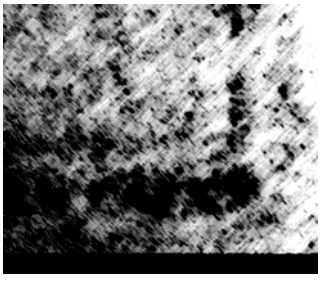
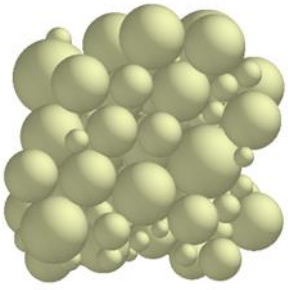
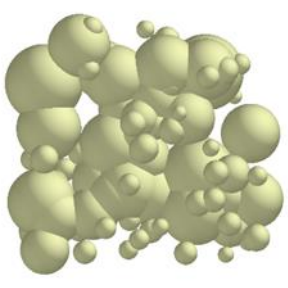
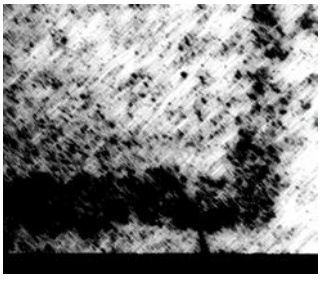
Laser power: 30 W (Lowest setting)			
Speed [mm s^{-1}]	**Before laser scan**	**After laser scan**	
	Simulation	**Simulation**	**Experimental**
(a) 125			
(b) 100			
(c) 75			
(d) 50			
(e) 25			

Figure 0.1. Simulations show orientation of particles before and after laser scan, and experiments (optical images) demonstrate effect of laser scan on the powder surface, for **different scan speeds** at **30 W power**.

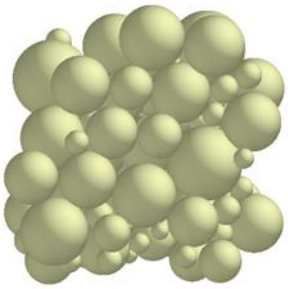
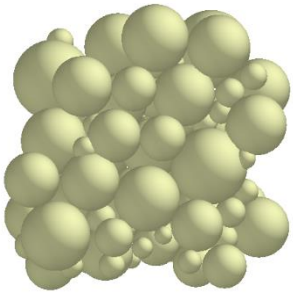
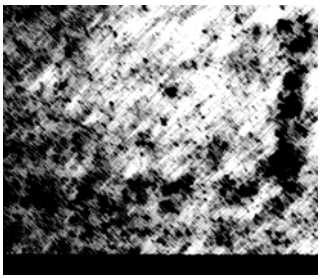
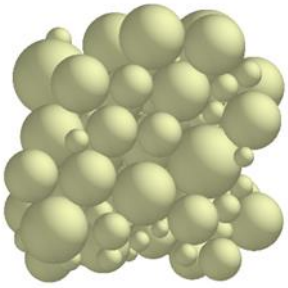
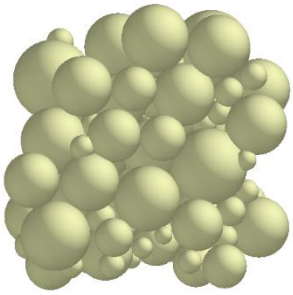
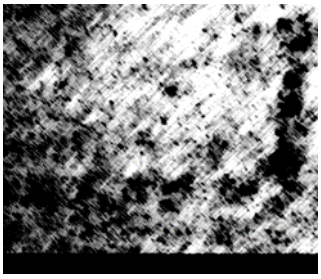
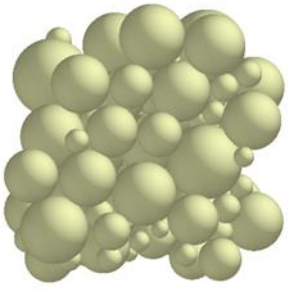
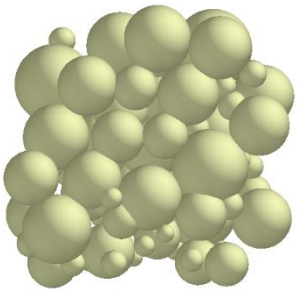
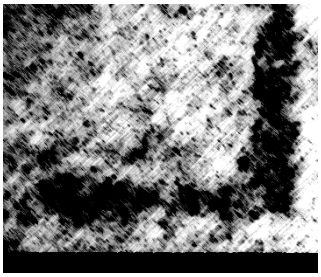
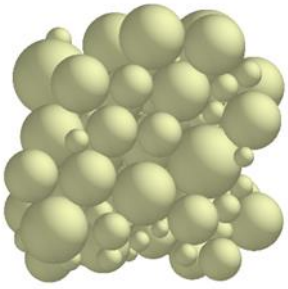
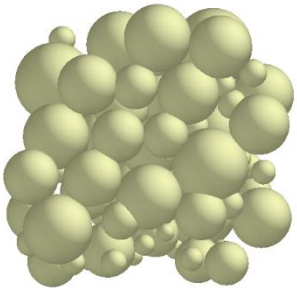
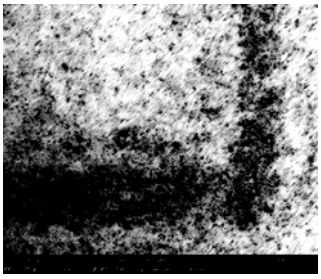
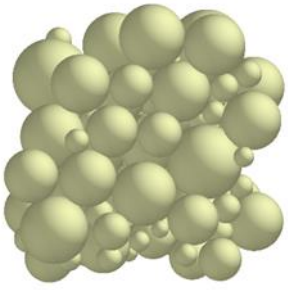
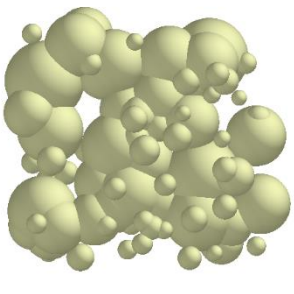
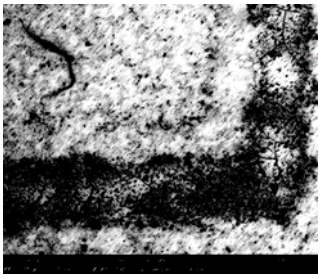
Laser power: 36 W			
Speed [mm s^{-1}]	**Before laser scan**	**After laser scan**	
	Simulation	**Simulation**	**Experimental**
(a) 125			
(b) 100			
(c) 75			
(d) 50			
(e) 25			

Figure 0.2. Simulations show orientation of particles before and after laser scan, and experiments (optical images) demonstrate effect of laser scan on the powder surface, for **different scan speeds** at **36 W power**.

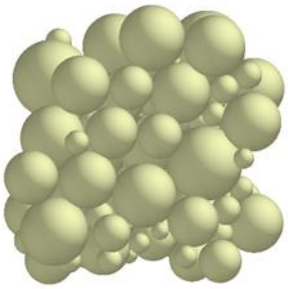
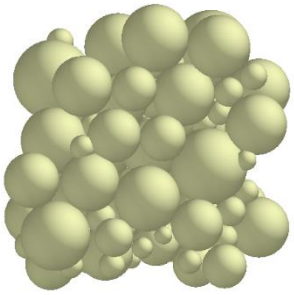
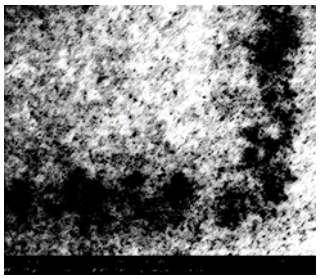
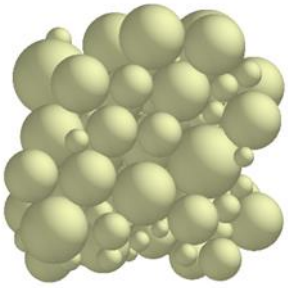
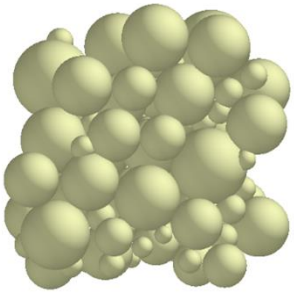
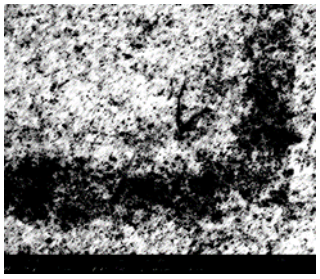
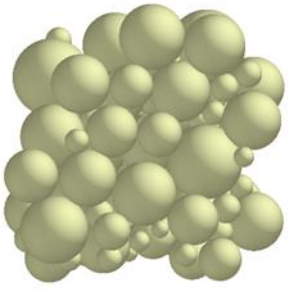
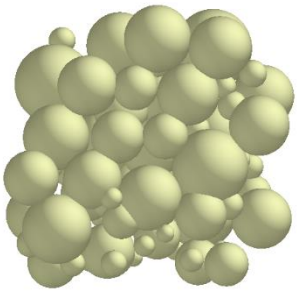
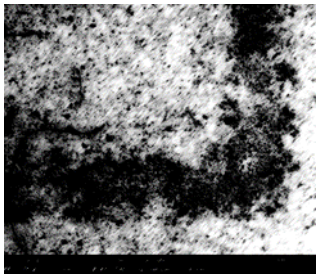
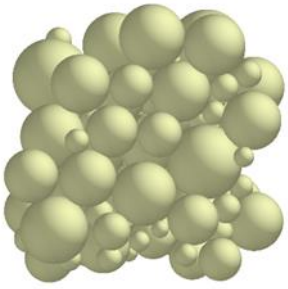
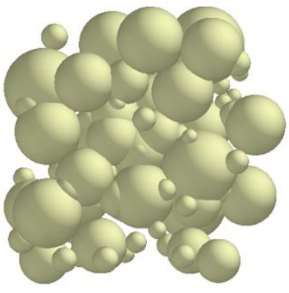
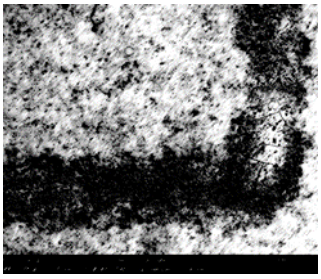
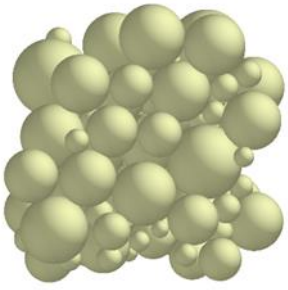
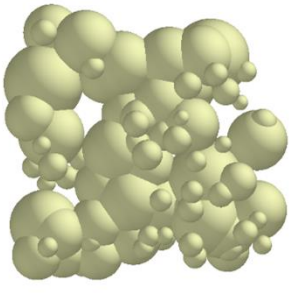
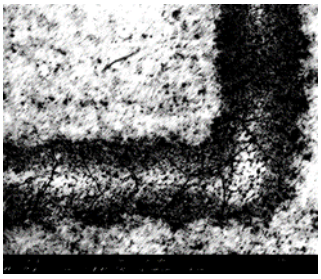
Laser power: 42 W			
Speed [mm s^{-1}]	**Before laser scan**	**After laser scan**	
	Simulation	**Simulation**	**Experimental**
(a) 125			
(b) 100			
(c) 75			
(d) 50			
(e) 25			

Figure 0.3. Simulations show orientation of particles before and after laser scan, and experiments (optical images) demonstrate effect of laser scan on the powder surface, for **different scan speeds** at **42 W power**.

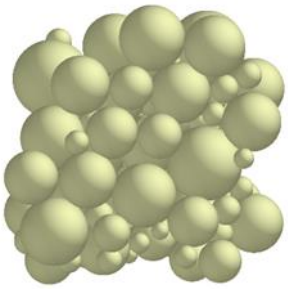
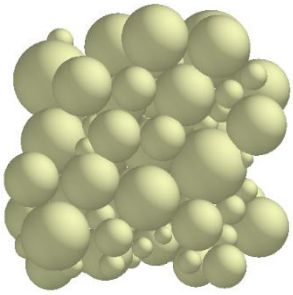
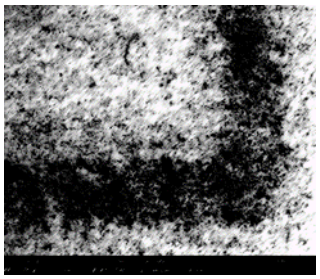
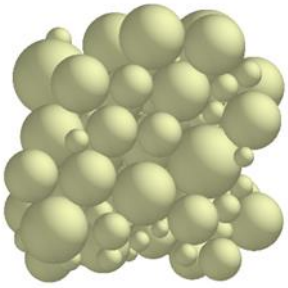
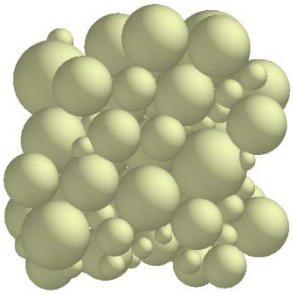
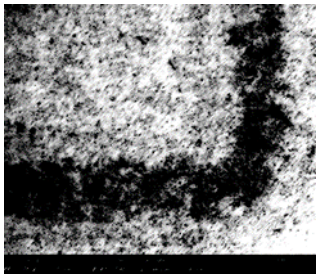
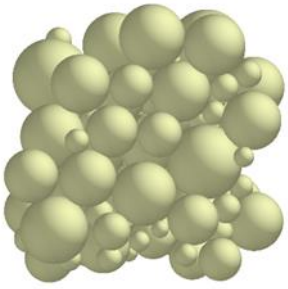
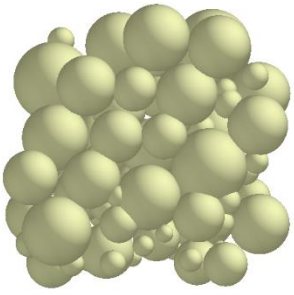
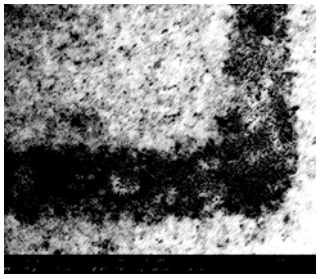
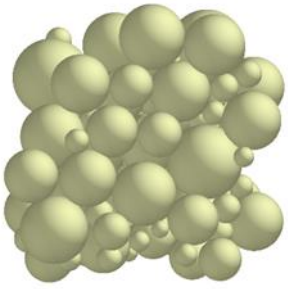
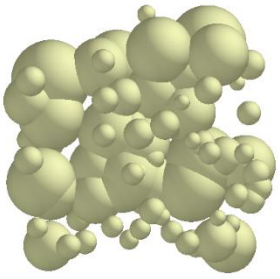
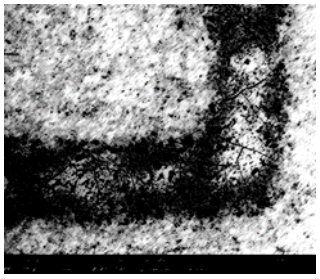
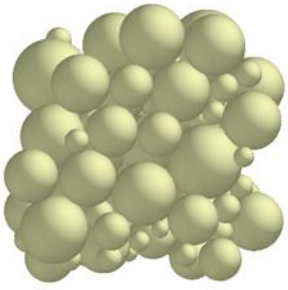
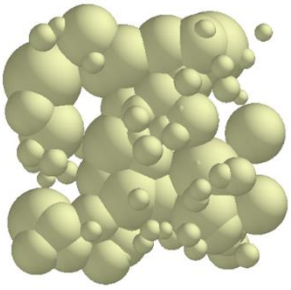
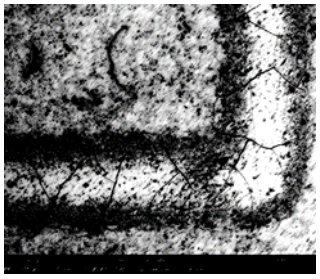
Laser Power: 48 W			
Speed [mm s^{-1}]	**Before laser scan**	**After laser scan**	
	Simulation	**Simulation**	**Experimental**
(a) 125			
(b) 100			
(c) 75			
(d) 50			
(e) 25			

Figure 0.4. Simulations show orientation of particles before and after laser scan, and experiments (optical images) demonstrate effect of laser scan on the powder surface, for **different scan speeds** at **48 W power**.

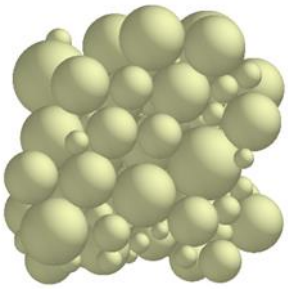
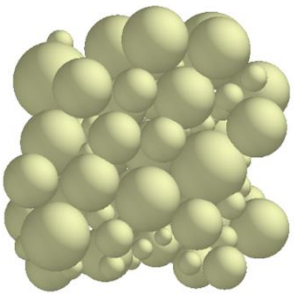
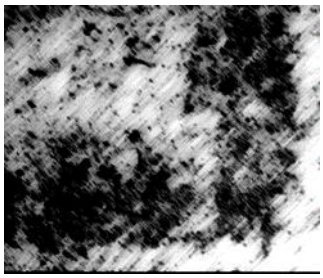
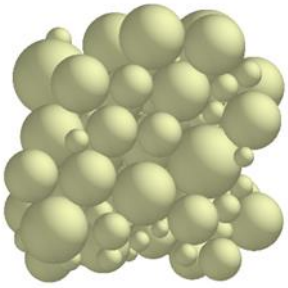
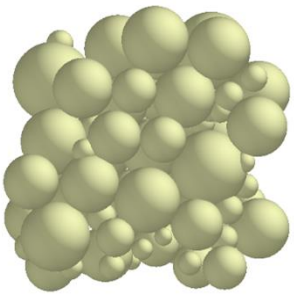
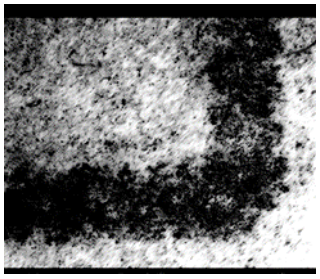
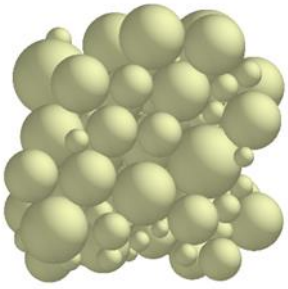
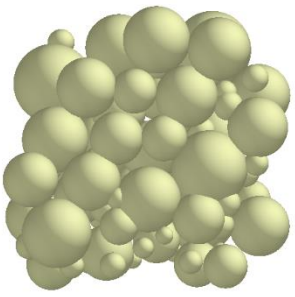
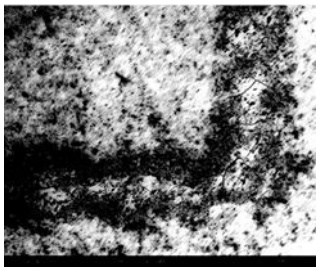
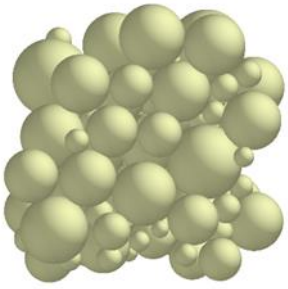
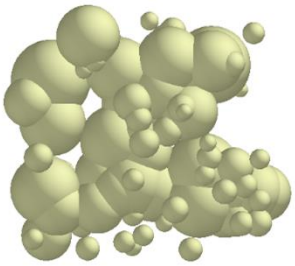
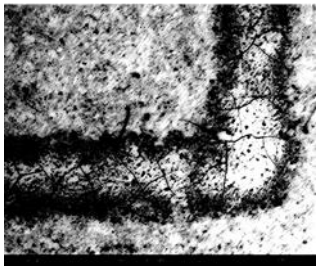
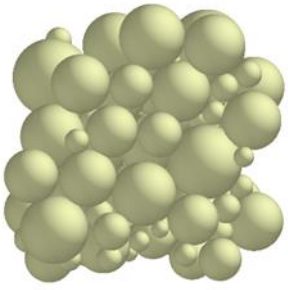
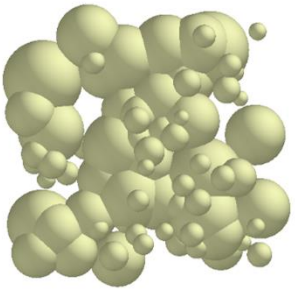
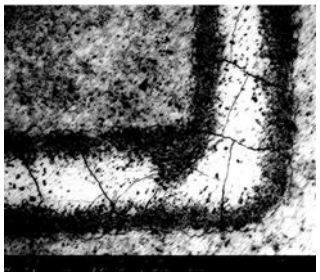
Laser Power: 54 W (Highest setting)			
Speed [mm s^{-1}]	**Before laser scan**	**After laser scan**	
	Simulation	**Simulation**	**Experimental**
(a) 125			
(b) 100			
(c) 75			
(d) 50			
(e) 25			

Figure 0.5. Simulations show orientation of particles before and after laser scan, and experiments (optical images) demonstrate effect of laser scan on the powder surface, for **different scan speeds** at **54 W power**.