A Poisson-disk sampling based particle-packing generation algorithm for Discrete Element simulations

Haopeng Sun

A Thesis

In

The Department

of

Building, Civil, and Environmental Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science (Geotechnical Engineering) at

Concordia University

Montréal, Québec, Canada

August 2021

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Haopeng Sun

Entitled: A Poisson-disk sampling based particle-packing generation algorithm for Discrete Element simulations

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Geotechnical Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

______________________ *Dr. S. Li*	Chair and Examiner
______________________ *Dr. S. Li*	Examiner
______________________ *Dr. B. Li*	Examiner
______________________ *Dr. A. M. Zsaki*	Supervisor

Approved by: ____

Chair of the Department or Graduate Program Director

______ 2021 ____

Dean of Faculty

ABSTRACT

A Poisson-disk sampling based particle-packing generation algorithm for Discrete Element simulations

Haopeng Sun

The Discrete Element Method (DEM) has been extensively used to model deformation and stresses developed in soils and rocks. The ever-increasing computational power allows the creation of accurate numerical models using the DEM with a significant number of elements.

However, DEM models with equal-sized particles or particles with a narrow range of radii such as those available in current DEM software cannot realistically reflect the physically interactive forces between soil particles, resulting in inaccurate simulation results. This thesis proposes an algorithm to generate circular and spherical particle assemblies that feature particle-size distributions and void ratios derived from actual soil data to improve the accuracy of DEM results. The proposed algorithm can automatically create particle packings with a wide range of radii simulating real soil samples to increase the quality of DEM simulations. The Poisson Disk Sampling and Grid Sampling techniques are introduced to generate models in a random but controllable fashion, meaning that the positions and radii of particles are randomly selected, however, the statistical profile of the particle assembly can be controlled. Similar to soil particle-size analysis, the particle packing is created using a sieve-by-sieve approach. Prior to importing the particle assembly into a DEM simulation system, the algorithm-generated particle assemblies are imported into an open-source DEM framework to complete the model deposition process. This study also includes a number of examples of building 2D and 3D particle assemblies using the proposed algorithm according to laboratory data of pure, mixed, gap graded, uniformly graded, dense, and loose soils to validate the algorithm.

ACKNOWLEDGEMENT

First and foremost, I would like to express my gratitude to my supervisor, Dr. Attila Michael Zsaki, for providing me the opportunity to join his research group. His idea of building an algorithm to improve the DEM simulation accuracy is inspirational. His patient guidance and consistent support helped me to surmount numerous technical difficulties and finally finish the study.

Special thanks to my parents, Yue and Xia, who support me unswervingly and unconditionally.

Finally, I would like to acknowledge the financial support from my supervisor through the GCS FRS funds and from a Concordia SEED Individual research grant held by my supervisor to help complete my research work.

Table of Contents

List of Figures

List of Tables

Notation

act	=	An array of newly generated points
adj	=	An array of distances between the particle and adjacent particles
bdy	=	An array of distances between the particle and boundaries
clr	=	Color of circular particles in 2D model
$cols$	=	Number of cells in the long side of the canvas
d_a	=	Minimum distance between particle center and around particles
d_b	=	Minimum distance between particle center and borders
d_c	=	Distance between one corner of the cell and the particle center
dp	=	Depth of the 3D canvas
e	=	Void ratio
f	=	A boolean variable indicating whether the points are valid
$grid$	=	An array of cells in the canvas
$grow$	=	An boolean variable indicating whether the particle can grow
ht	=	Height of the 2D or 3D canvas
i	=	X coordinate of the cell where the random point is located
j	=	Y coordinate of the cell where the random point is located
max_r	=	Minimum value of the radius range
min_r	=	Maximum value of the radius range
p	=	Percent finer
px	=	A smallest addressable element in 2D model canvas
pos	=	An array of new randomly selected points
$posit$	=	Position of the particles in Circle or Sphere object
r_{min}	=	Minimum radius of the next round of particles insertion
rad	=	Radius of a circular or spherical particle
$ranidx$	=	An integer where a random element in the array is chosen
$rows$	=	Number of cells in the short side of the canvas
s_m	=	Minimum canvas size
v_{max}	=	Individual volume of the biggest particles

w	=	The length of the cell (pixel)
wd	=	Width of the 2D or 3D canvas
x	=	X coordinate of the first randomly selected point
y	=	Y coordinate of the first randomly selected point
C	=	The number of contacts
$Ptcl$	=	An array of the particles generated by void infilling
M	=	Total mass
N	=	The number of particles
V	=	Total volume of the sand sample (mm^3)
X_c	=	X coordinate of a randomly selected cell
Y_c	=	Y coordinate of a randomly selected cell
Z	=	Coordination number
Z_c	=	Z coordinate of a randomly selected cell
ρ	=	Density (g/mm^3)

Chapter 1 Introduction

1.1 Background

Numerical modeling is a vibrant research area in geotechnical engineering. The development of numerical simulation comes hand-in-hand with the rapid introduction of novel computational techniques and the prevalence of powerful personal computers. The development of numerical modeling techniques has considerably accelerated in recent years due to the exponential increase in computational capacity (Zhang et al. 2013).

Numerical simulation methods of rock and soil stress and deformation analysis, in a broad sense, can be categorized into continuum methods and dis-continuum methods (Jing and Hudson 2002; Jing 2003). Continuum methods treat a rock or soil mass as a continuum, and the procedure is to exploit approximations to the connectivity and continuity of displacements and stresses between elements. Dis-continuum methods regard a rock or soil mass as an assemblage of distinct interacting blocks or bodies that can be treated as rigid or could be subdivided into deformable finite-difference meshes that follow linear or non-linear stress-strain laws (Jing and Stephansson 2007).

Continuum methods can also be classified into integral methods and differential methods. For the former, only problem boundaries are defined and discretized. They are more efficient computationally than differential methods but restricted to elastic analyses. For the latter, the whole problem domain is defined and discretized. They can simulate materials with non-linear and heterogeneous properties but are more computationally intensive than the integral methods (Sbirrazzuoli et al. 2009).

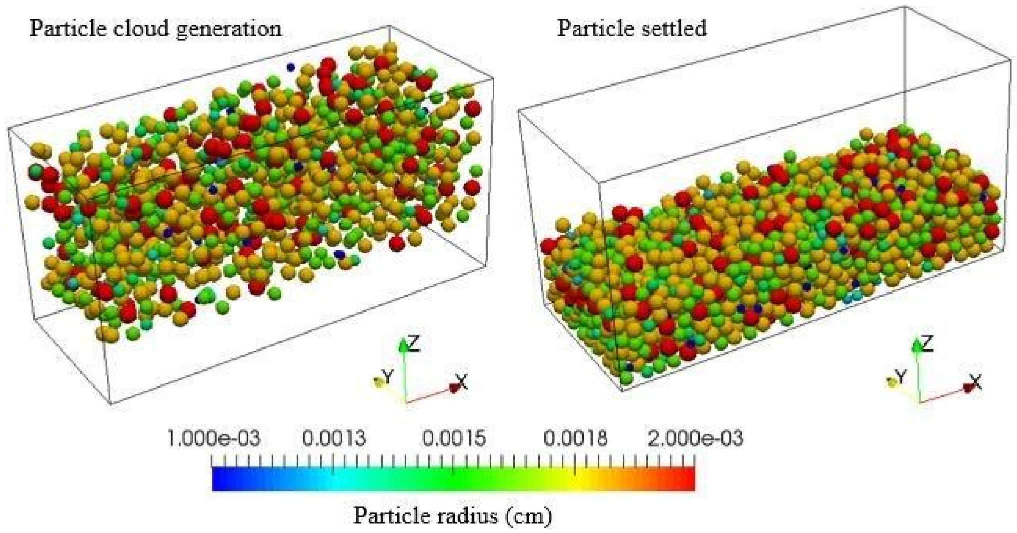


Fig 1.1 Discrete Element Method simulation results using the open-source code LIGGGHTS (Aggarwal and Kumar 2019)

Discrete Element Method (DEM) is a dis-continuum numerical method designed to handle contact conditions for a mass of irregularly shaped particles (Munjiza 2005). It is an effective method of solving engineering problems that deal with granular and discontinuous materials, such as granular flows, powder mechanics, and rock mechanics. Fig 1.1 shows the DEM settlement simulation of a particle assembly in a container (Aggarwal and Kumar 2019).

The advantage of DEM is that it can simulate a wide variety of granular flow and rock mechanics problems and allows a more detailed study of the micro-dynamic analysis of powder flows than physical experiments (Boac et al. 2014). However, it is computationally intensive compared to other numerical methods, which may cause long computation time and limit either the length of a simulation or the number of particles (Bandeira and Zohdi 2018). Another challenge of DEM is generating particle packings that represent realistic conditions (Dang and Meguid 2010).

1.2 Research Objectives and Scope

This research aims to propose an algorithm to generate 2D and 3D DEM simulation models that are close to actual conditions. In particular, the main target is to develop algorithms to create an assembly of circular (in 2D) and spherical (in 3D) particles, which are similar to real soils in terms of particle-size distribution and void ratio, to increase the DEM simulation accuracy. The current simulation software and open-source codes can only create same-sized particles or particles with a narrow range of radii, which cannot reflect the real interactive forces between soil particles. For example, Yade (Donzé et al. 2009), an open-source DEM simulation code, can only generate quasi equal-sized spherical particles based on users' definition on sphere centers, radii, and relative fuzz variation of radii. Current software only generates DEM models with same-sized particles or particles with narrow-range radii because of computational power limitations. However, the speedy development of CPU and GPU calculation capacity makes the creation of more accurate and close-to-reality models possible.

Particle assemblies with particle-size distributions and void ratios that approximate real soil data can yield results that are theoretically and physically more accurate using DEM. In a DEM simulation, two critical components are needed to account for the evolution of a mechanical system: the geometry of particles and the interaction forces between them (Wautier et al. 2018). Fig 1.2 shows the normal and tangential forces between two contacting particles. The dimensions and masses of particles play essential roles in the DEM simulation result. As such, a model with equal-sized particles fails to simulate realistic interactions between soil particles and, therefore, generates inaccurate results.

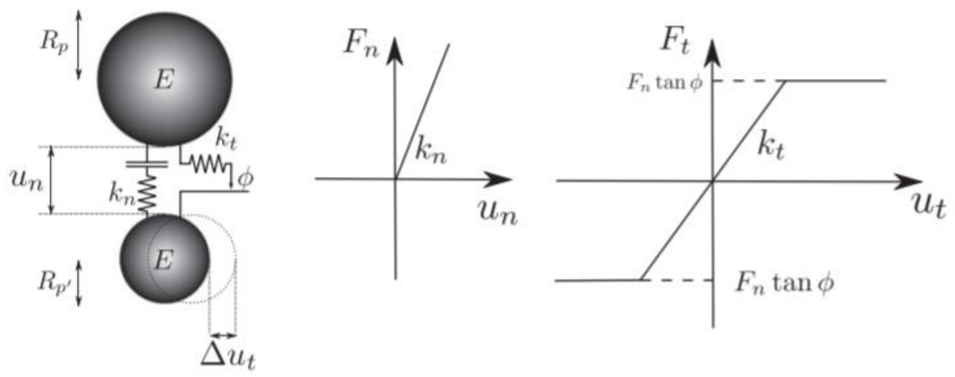


Fig 1.2 Elasto-frictional contact law used in DEM simulations (Wautier et al. 2018)

Fig 1.3 shows the calculation process of the DEM simulation, including time steps, variables, and formulas (O'Sullivan 2011). When contact forces and body forces between particles are calculated, the masses and dimensions of interacting particles are key parameters in the calculation process. As a result, a DEM simulation can produce more accurate results by modeling particles with an extensive radius range that approximates particle-size distributions and void ratios of real soil data rather than models with equal-sized particles or particles with a narrow range of radii.

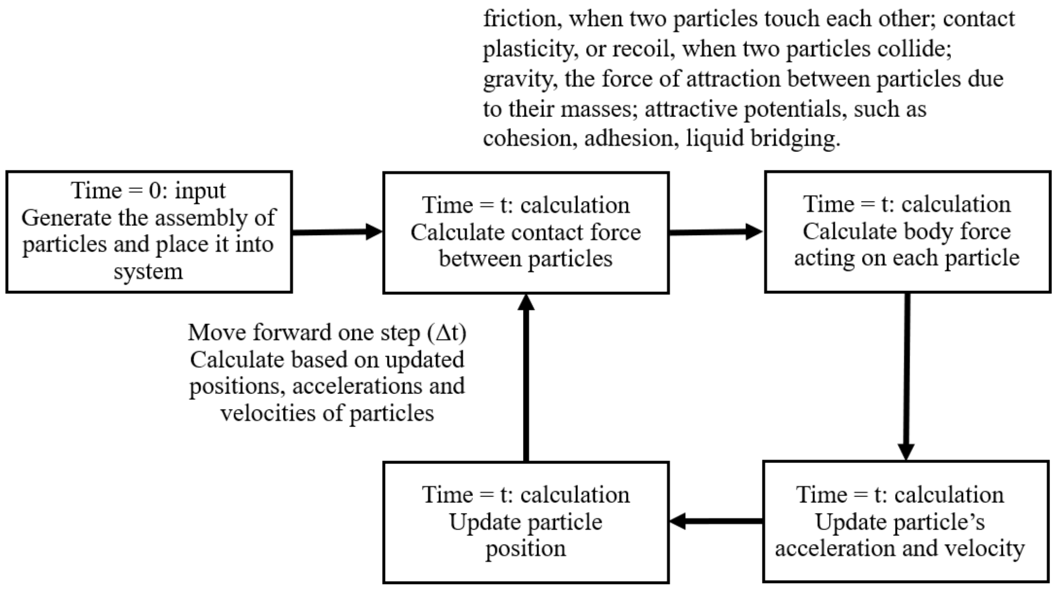

friction, when two particles touch each other; contact plasticity, or recoil, when two particles collide; gravity, the force of attraction between particles due to their masses; attractive potentials, such as cohesion, adhesion, liquid bridging.
Time = 0: input Generate the assembly of particles and place it into system
Time = t: calculation Calculate contact force between particles
Time = t: calculation Calculate body force acting on each particle
Move forward one step (Δt) Calculate based on updated positions, accelerations and velocities of particles
Time = t: calculation Update particle position
Time = t: calculation Update particle's acceleration and velocity

Fig 1.3 The calculation process of Discrete Element Method (O'Sullivan 2011)

The detailed objectives of this study aimed at bridging existing research gaps are as follows:

a) Propose an algorithm to create DEM particle packings that feature particle-size distributions and void ratios close to those encountered in actual soils.

b) Introduce the Poisson Disk Sampling method to generate particles belonging to the largest opening sieve and the Grid Sampling method to insert particles of the remaining sieves by filling the voids between large particles.

c) Achieve particle packing deposition by importing the algorithm-generated models into the DEM simulation software to realize the particle packing densification and obtain the void ratio of the deposited model.

d) Provide examples of 2D and 3D models generated by the algorithm targeting the laboratory data of various soils to validate the algorithm.

1.3 Research Methodology

The methodology is presented below to describe how the research objectives are accomplished:

a) This thesis introduces Poisson Disk Sampling and Grid Sampling methods to randomly but controllably generate particle packings sieve-by-sieve based on user-defined parameters. Poisson Disk Sampling is used to generate samples with an even but random distribution. This method will be applied to determine the positions of particles in the largest opening sieve, laying the model's foundation. Subsequently, the Grid Sampling method locates voids by checking cells that divide the whole area evenly and fills these gaps by inserting particles belonging to the remaining sieves.

b) In computer science, recursion is a method used to handle uncertain reiteration problems. It is used to rerun the particle insertion process when errors exceed the acceptable threshold, or when the particle volume does not reach the target. Particle validation checks if a randomly selected point in the void space can be used to generate a particle. Combining both approaches enables the algorithm to automatically generate particles without manual adjustment based on the user-defined particle-size distributions and void ratios. To pinpoint the void spaces in the canvas, the algorithm randomly generates particles only in non-filled cells.

c) The model deposition simulation is done by Yade (Donzé et al. 2009). The algorithm-generated particle assembly is imported into Yade. It exports the deposited model and several related parameters, including inter-particle forces, unbalance force ratio, coordination number, and void ratio.

d) The algorithm is written by Python (Van 2009). Object-oriented programming (OOP), a computer programming method that organizes software design based on objects (Doherty 2020), is introduced. An object or a data field with unique attributes and behaviors is defined herein as a particle in the DEM model. The algorithm also employs a Python library, NumPy (Travis 2005), to empower the handling of data and arrays.

e) The 2D model visualization is completed by JavaScript and P5.js (McCarthy et al. 2015), a JavaScript-based library specialized in presenting graphics. The 3D model is displayed in Paraview, an open-source, multi-platform visualization application (Röber 2014).

1.4 Thesis Outline

This thesis contains five chapters:

Chapter 1 introduces the research motivations and objectives. The main objective is to propose an algorithm that generates DEM models similar to the real soils in terms of particle size distributions and void ratios. DEM models with similar statistical characteristics to actual soils can provide more accurate simulation results than currently available models.

Chapter 2 presents the literature review. It first introduces simulation methods and places emphasis on DEM. DEM simulation and the current DEM particle packing generation methods that open-source codes and simulation software use are also described. This chapter then covers the Poisson Disk Sampling and Grid Sampling techniques that the algorithm utilizes to generate particles with a random and even distribution. Finally, an overview of programming languages, visualization tools, data formats, and code editors is provided.

Chapter 3 described the algorithm development. The model building process consists of two parts: generating particles for the largest opening sieve using the Poisson Disk Sampling technique, which lays the model's foundation, and then filling the voids to insert particles belonging to the remaining sieves using the Grid Sampling method. This chapter presents the definitions and recommended values of parameters defined in the algorithm and techniques to optimize the model building process and reduce the running time. The chapter ends by outlining how the model is deposited in Yade and how contact friction is used to adjust the void ratio of the deposited models.

Chapter 4 deals with algorithm testing and validation. Examples of building 2D and 3D models using the proposed algorithm targeting a wide variety of soils are provided to demonstrate that the algorithm can generate models for soils with various characteristics. The algorithm-generated models may produce differences compared to the actual soil, and the type of errors is finally described.

Chapter 5 provides concluding remarks and emphasizes the strengths and limitations of the algorithm. Recommendations for future study are also given.

Chapter 2 Literature Review

2.1 Introduction

The DEM is a popular numerical method for computing the motion and response of an assemblage of particles, and it has been widely used to solve problems dealing with granular or discontinuous materials in geotechnical engineering (Donzé et al. 2009). This chapter describes several numerical modeling methods, including DEM, outlines supersampling used by the algorithm to generate particles in a random but controllable fashion, and gives an account of other computational tools that played a pivotal role in the development of the algorithm.

This chapter is divided into four sections organized as follows: Section 2.2 deals with the developments and applications of several numerical simulation methods with a special emphasis on the DEM. Section 2.3 presents DEM model generation methods currently available in simulation software packages and currently explored research avenues in the field. Section 2.4 introduces supersampling techniques, including the Poisson Disk Sampling method and Grid Sampling method and different ancillary algorithms to realize these methods. Section 2.5 describes computational tools, including Python, NumPy, a Python-based data analysis library, and data formats. This chapter ends by describing JavaScript and p5.js, the visualization tools for the 2D particle packing, and Paraview, a 3D model visualization application.

2.2 Numerical Modeling Methods in Soil Mechanics

The commonly applied numerical modeling methods for soil mechanics problems can be classified into two categories, continuum methods, including the finite difference method (FDM), the finite element method (FEM), and the boundary element method (BEM), and discrete or dis-continuum methods, such as the DEM, and discrete fracture network method (DFN). This part briefly introduces these numerical modeling methods as well as their advantages and limitations while focusing on the DEM.

2.2.1 Finite Difference Method

The Finite Difference Method (FDM), based on finite-difference approximations for derivatives occurring in differential equations, is one of the simplest and oldest methods of solving differential equations (Datas 2020). Euler initially proposed FDM in one-dimension space, and this method was extended to two dimensions by Runge in 1908 (Timoshenko and Goodier 1982). Finite-difference techniques were first applied in practice in the early 1950s. Their developments were stimulated by the emergence of computers that offered a convenient framework for dealing with complex technological problems. The relaxation method developed by Southwell in 1964 allowed for systems of equations to be solved quickly and significantly contributed to the widespread use of FDM (Cryer 1970).

Theoretical results were produced regarding the accuracy, stability, and convergence of the finite difference method for partial differential equations. The FDM is based on the following premise: finite differences at grid points can adequately represent the governing differential equations. It operates by discretizing the governing partial differential equations by replacing the partial derivatives with differences defined at neighboring grid points (Datas 2020). The FDM can handle complex non-linear material behavior, such as laterally loaded piles, one-dimensional consolidation, two and three-dimensional seepage (Desai and Christian 1979; Bobet 2010), and solve time-dependent problems (Nikolić et al. 2016). However, this method still has limitations in modeling arbitrarily shaped domains. When the configuration of a field is simple, e.g., a rectangular shape, the mesh points can be adjusted to coincide with the boundaries. In contrast, the mesh points may not fall on the irregular boundary (Desai and Christian 1979).

Newly developed techniques now make it possible to handle irregular meshes, such as triangular grid or Voronoi grid systems, which leads to Control Volume, or Finite Volume, techniques. Voronoi polygons grow from points to fill the space instead of tessellations where the polygons are formed by lines cutting a plane or building up a mosaic from pre-existing polygonal shapes (Gore et al. 2005).

2.2.2 Finite Element Method

The Finite Element Method (FEM) is the most commonly used numerical method in engineering and mathematical models (Rapp 2017). Courant initially developed this method using the Ritz method of numerical analysis and minimization of variational calculus in 1943 (Finlayson 1972). The broader definition of finite element analysis was first coined by Turner and Clough in 1956, who investigated the stiffness and deflection of complex structures (Widas 1997). By the early 1970s, FEM was limited to expensive mainframe computers owned by the aeronautics, defense, and nuclear industries. Due to the rapid decline in computer cost and the phenomenal increase in computing power, it has witnessed considerable improvements in precision and speed (Tan et al. 2011).

The FEM subdivides a large model into small and simple parts, called finite elements, which form a mesh containing a limited number of points representing the numerical domain for the solution. The FEM formulation of a boundary value problem results in equations where the method approximates the unknown function over a given domain. The simple equations representing these finite elements are then assembled into a more extensive system of equations that models the entire problem. The FEM uses variational methods from the calculus of variations to approximate a solution to the system of equations by minimizing an associated error function (Ashcroft and Mubashar 2011). Fig 2.1 illustrates a slope displacement and corresponding safety factors calculated using the FEM method.

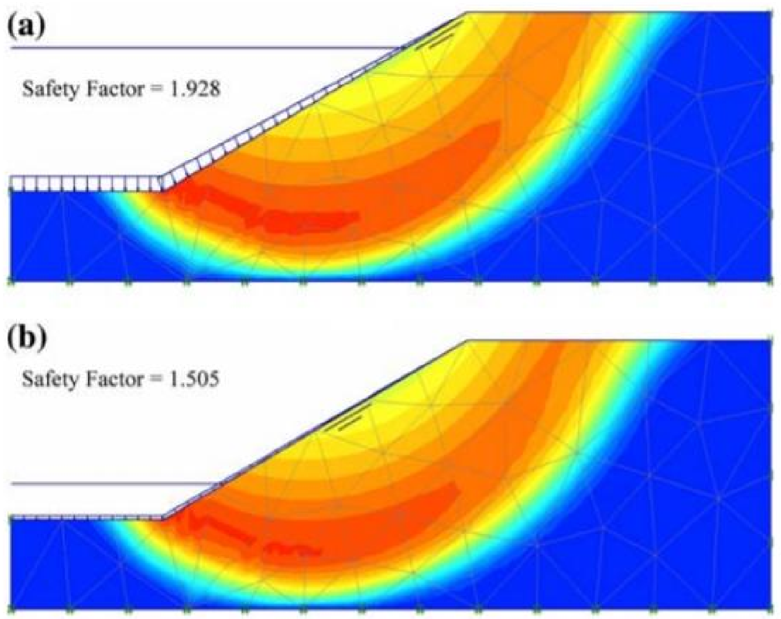


Fig 2.1 Contour of slope displacement using FEM (Hammouri et al. 2008)

Compared to other numerical methods, FEM offers the following advantages: modeling the slope with a degree of a high pragmatism such as complex geometry, sequences of loading, presence of material or reinforcement, capturing the action of water, and use of constitutive laws that characterize complex soil behavior; quickly observing the deformations of soils (Matthews et al. 2014). However, there are several caveats associated with FEM. Indeed, it yields an approximate solution, and only the solution at nodes is accurate. In addition, FEM comes at a high computational cost, and the time needed to solve the problems increases with the degree of mesh fineness (Sharma 2019).

2.2.3 Boundary Element Method

The Boundary Element Method (BEM) is a numerical computational method of solving linear partial differential equations formulated as integral equations. It calculates a weak solution at the global level through a numerical solution of an integral equation derived using Betti's reciprocal theorem and Somigliana's identity (Cheng and Cheng 2005). The diagram presenting the BEM to define the boundary value for an arbitrary shape is shown in Fig 2.2.

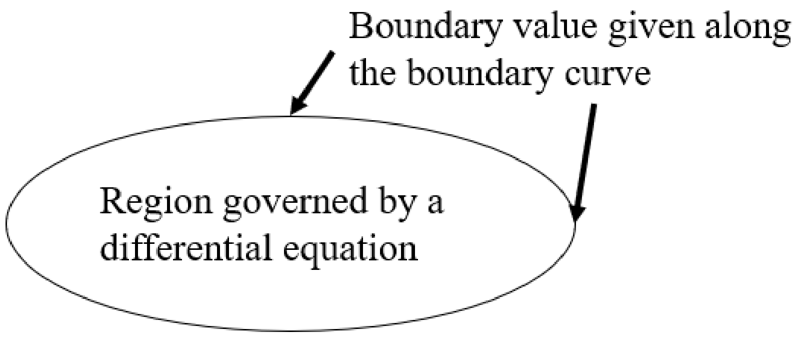

Boundary value given along the boundary curve
Region governed by a differential equation

Fig 2.2 BEM diagram for an arbitrary shape (Pearce 2011)

In the BEM, discretization is performed differently than it is in both the FDM and the FEM. It solves a boundary integral equation only related to the boundary values (Hall 1994). As such, the BEM discretizes only the boundaries of the continuum, while the entire medium and boundaries are discretized using the FEM and FEM (Bobet 2010).

The BEM formulations are particularly well-suited to address static continuum problems with small boundary-to-volume ratios, elastic behavior, and stresses or displacements applied to the boundaries. However, it remains challenging to deal with angular boundaries since the appropriate boundary integral equations give rise to large and dense matrices to be solved (Costabel 1987).

2.2.4 Discrete Element Method

The Discrete Element Method (DEM) is a fundamental physics method that treats each particle of a granular material individually (Hager et al. 2018). Each particle is represented through a descriptive shape and size that interacts with other particles and boundary geometry. These interactions are the core of the DEM and are modeled through different user-defined material properties. The DEM features three aspects: the representation of contacts, modeling solid materials, and the scheme used to detect and update the set of contacts (Cundall and Hart 1992).

The classical DEM first developed by Cundall (1971) was meant for rock mechanics problems. It was further developed by Cundall and Strack (1979), who proposed a method to calculate interaction forces when elements slightly penetrate each other. This force-displacement formulation is typically referred to as a smooth contact method. Other discrete numerical methods, referred to as non-smooth contact methods, exclude possible interpenetration between elements and only deal with unilateral contact (Moreau et al. 1994; Jean et al. 1995 and Luding et al. 1996).

The primary difference between the DEM and other continuum-based methods is that the contact patterns between components are continuously changing during the deformation process for the former but are fixed for the latter (Khan 2010).

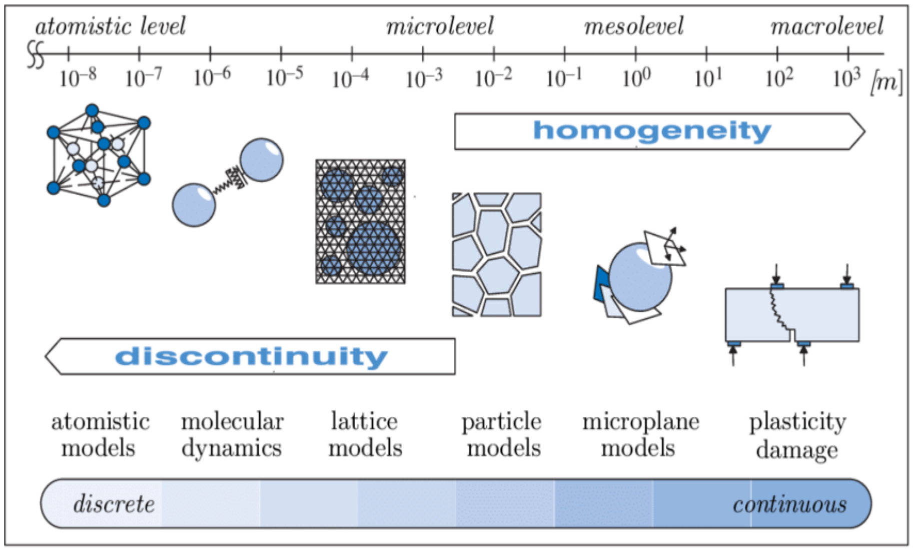


Fig 2.3 Observation scale in the classification of numerical modeling techniques of materials (D'Addetta et al. 2001)

A DEM simulation can reflect the most striking characteristic of granular material, its dual nature lying between a disjoint, discrete material and a continuum. Fig 2.3 shows different materials that can be characterized from discontinuity to homogeneity (D'Addetta et al. 2001). Although individual particles are solid, they are only partially connected at contact points. The DEM treats particles individually and can be regarded as a method that closely simulates reality. Particle types vary due to different properties, including shape and size distribution, density, Young's Modulus, Poisson ratio, adhesion, thermal conductivity, specific energy for breakage, etc. These parameters can capture the unique complexities of the system under different conditions (Yue et al. 2018). Fig 2.4 shows the continuous, discrete-block, and discrete particles models simulated by different numerical software.

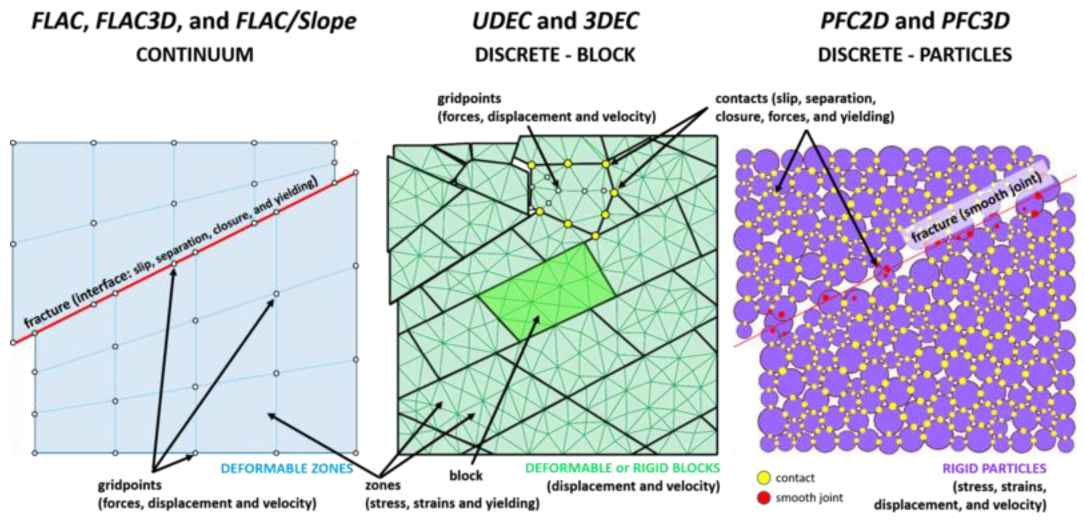

FLAC, FLAC3D, and FLAC/Slope
CONTINUUM
UDEC and 3DEC
DISCRETE - BLOCK
PFC2D and PFC3D
DISCRETE - PARTICLES
gridpoints (forces, displacement and velocity)
contacts (slip, separation, closure, forces, and yielding)
fracture (interface: slip, separation, closure, and yielding)
fracture (smooth joint)
DEFORMABLE ZONES
gridpoints (forces, displacement and velocity)
zones (stress, strains and yielding)
block
DEFORMABLE or RIGID BLOCKS (displacement and velocity)
contact
smooth joint
RIGID PARTICLES (stress, strains, displacement, and velocity)

Fig 2.4 Continuum, discrete-block, discrete-particles models provided by different numerical software (Itasca 2019)

The application of DEM has recently been extended to study physical and geotechnical phenomena such as deformation mechanisms, constitutive soil relations, stability of rock masses, the flow of granular media, ground collapse, etc. (Donzé et al. 2009). This method is designed to deal with contact conditions for a mass of irregular particles (Munjiza 2005).

There are many solution strategies for different DEM formulations. The primary distinguishing feature is the method for treating material deformability. The rigid body analysis is currently a universal method. An explicit time-marching scheme uses finite difference schemes to solve the dynamic equations of motion of a rigid body system or a dynamic relaxation scheme for a quasi-static problem. However, for deformable-body systems, two solution strategies currently exist. An explicit solution with a finite-difference discretization of the body interiors ensures only one system unknown is kept at a local equation at a time step with no matrix equations needed in general. An implicit solution with finite element discretization of the body interior results in a matrix equation representing both motion and deformation of the individual bodies (Jing and Stephansson 2007).

2.2.5 Discrete Fracture Network

The Discrete Fracture Network (DFN) is an advanced method to mimic discrete pathways for fluid flow through a fractured rock mass (Makedonska and Gable 2018). This method explicitly represents the geometrical properties of each fracture (e.g., orientation, size, position, shape, and aperture) and the topological relationships between individual fractures and fracture set within a rock mass, as shown in Fig 2.5 (Lei et al. 2017). The DFN method assists with fragmentation assessment by better describing natural fragmentation distribution than other numerical methods (Elmo et al. 2014).

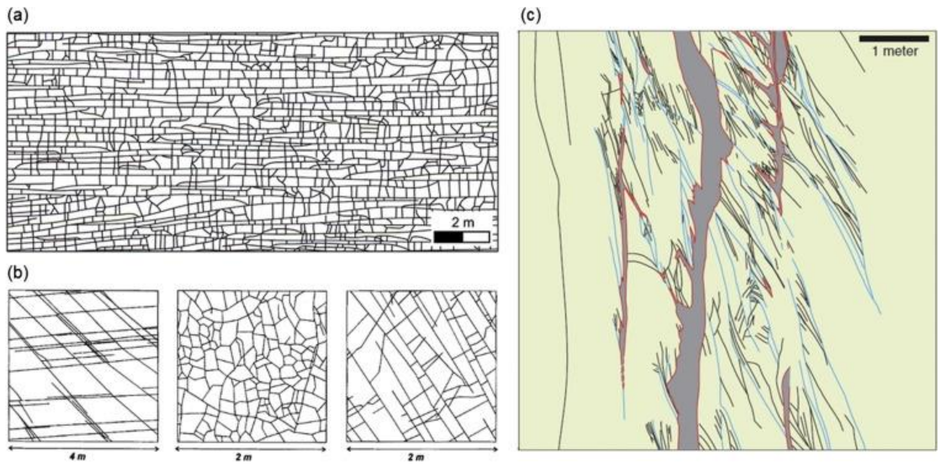


Fig 2.5 Geologically-mapped DFN patterns based on (a) a limestone outcrop at the south margin of the Bristol Channel Basin, UK, (b) sandstone exposures in the Dounreay area, Scotland, and (c) fault zone structures in the Valley of Fire State Park of southern Nevada, USA. (Lei et al. 2017)

However, questions are being raised within the hydrogeologic community over the value of the discrete fracture network method compared to stochastic continuum approaches. Such doubts are fueled by the concept of representative elementary volume on which all continuum approaches hinge (Dershowitz et al. 2004).

2.3 Current Developments in DEM

This section focuses on current developments in Discrete Element Modeling. DEM innovations can be generally categorized into two branches: improving the calculation of the interactive force between particles and novel methods to generate particle packings.

The first part of this section describes new methods for DEM model generation. The second section presents current leading software and open-source code for DEM implementation and their applications. The last part of this section reviews developments in DEM model generation methods used to create particles with various sizes, shapes, and distributions.

2.3.1 Current DEM Model Generation Innovations

Jing and Hudson (2002) designed the granular element method to accurately capture grain shape using Non-Uniform Rational Basis-Splines (NURBS). They proposed a mathematical model commonly used in computer graphics to represent curves and surfaces (Schneider 2011). Their method allows discrete elements to take realistic and complex granular shapes encountered in engineering and science, improving in discrete computational mechanics of granular materials.

Zsaki (2013) developed a reusable library that can generate element assemblages to reduce model generation times drastically. He proposed a fast and straightforward method to accomplish the boundary conformance by building on a previously developed and published algorithm and adding new features.

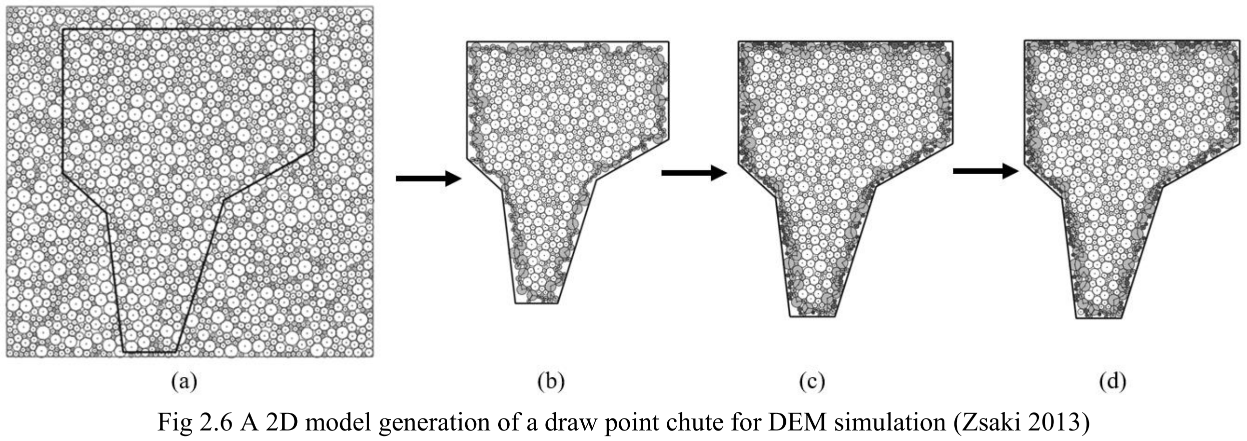
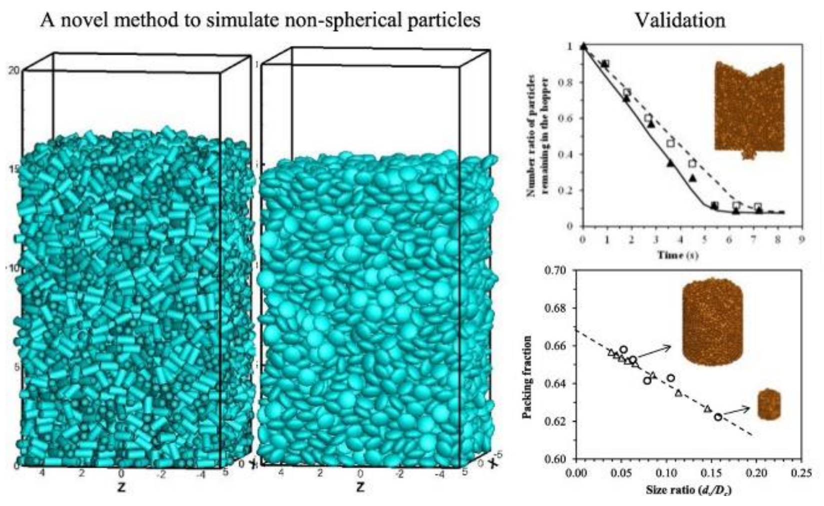


Fig 2.6 A 2D model generation of a draw point chute for DEM simulation (Zsaki 2013)

Fig 2.6 shows the process of a 2D DEM model generation for a draw point chute, typically found in underground mining (Matrix 2010), using the method proposed by Zsaki (2013). The process consists of three steps: i) building the boundary of the problem superimposed on an arrangement of elements selected from the library (Fig 2.6a), ii) identifying an outer layer or ring of disks inside the model boundaries (Fig 2.6b), and iii) adding multiple layers of newly added elements resulting from infilling (Fig 2.6c, Fig 2.6d).


A novel method to simulate non-spherical particles
Validation
Number ratio of particles remaining in the hopper
Time (s)
Packing fraction
Size ratio (d_i/D_c)
Fig 2.7 Discrete element modeling of non-spherical particles (Dong et al. 2015)

Dong et al. (2015) developed a novel method based on orientation discretization for discrete element modeling of representative non-spherical particles, as shown in Fig 2.7. Their approach was based on orientation discretization and pre-calculated databases and can be applied to any shaped particles in a general scheme.

2.3.2 DEM Simulation Software

There are currently many software packages and open-source codes that can run DEM simulations. This section presents several leading DEM simulation software and open-source codes, including Universal Distinct Element Code (UDEC) (Itasca 2019), Particle Flow Code (PFC) (Itasca 2019), Yade (Donzé et al. 2009), ESyS-Particle (Mora et al. 2006), and LIGGGHTS (Kloss et al. 2012).

The Universal Distinct Element Code (UDEC), developed by Itasca Consulting Group Inc. in 1996, is the current representative DEM software. UDEC is a two-dimensional numerical program that simulates the quasi-static or dynamic response to loading of media containing multiple intersecting joint structures. 3DEC, a three-dimensional version of UDEC, is used to simulate the response of discontinuous media, such as jointed rock or masonry bricks, subjected to either static or dynamic loading. Particle Flow Code (PFC), also from Itasca Consulting Group Inc., was released in 2014 and is a general-purpose DEM framework used for two- and three-dimensional simulation (PFC^{2D} and PFC^{3D}, respectively). The PFC^{2D} implements a particle flow model in terms of a collection of circular rigid particles or discs, and PFC^{3D} simulates a collection of rigid spheres. These two programs are based on the idea that a rock mass can be represented by many constituent particles whose contact stiffness and bounding behavior are relatively simple (Jing and Stephansson 2007).

Fig 2.8 shows a spherical particle packing that PFC^{3D} generated based on user-defined particle size distribution (Itasca 2019). The difference between the PFC^{3D} model generation and the algorithm proposed in the study is that particles overlap in the model generated by PFC^{3D} but the proposed algorithm in the study generates models with particles that do not overlap.

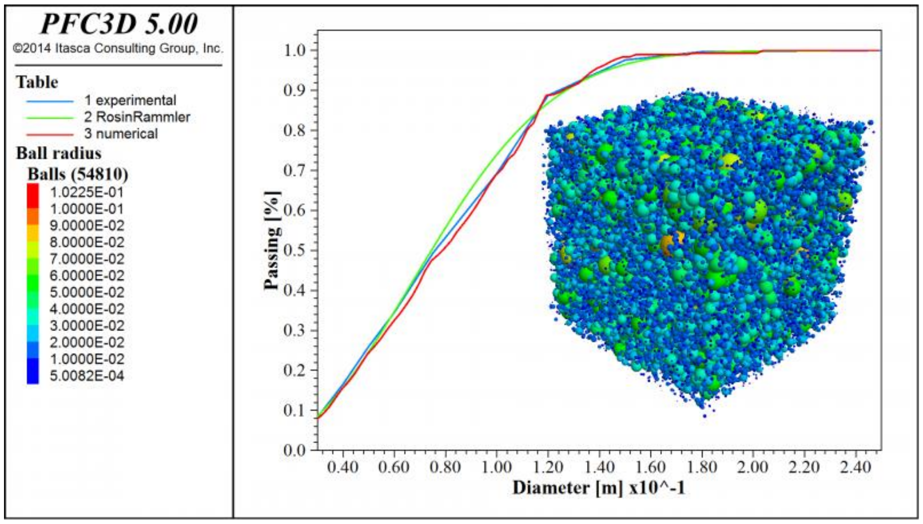


Fig 2.8 3D particle packing generated by PFC^{3D} based on user-defined particle size distribution (Itasca 2019)

Yade is an extensible open-source code for discrete numerical models, developed by Donzé et al. (2009). The computation parts are completed using a flexible object model, allowing the implementation of new algorithms and interfaces independently. Python is introduced to perform rapid and concise scene construction, simulation control, postprocessing, and debugging (Kozicki and Donzé 2009).

ESyS-Particle was created by Mora et al. in 1992 and is another open-source DEM simulation software for particle-based numerical modeling. It is designed for execution on parallel supercomputers, clusters, or multi-core PCs running Linux or Windows. The simulation engine implements spatial domain decomposition via the Message Passing Interface (MPI). A Python wrapper API provides flexibility in numerical models, modeling parameters and contact logic, and analysis of simulation data. ESyS-Particle has been extensively used to simulate earthquake nucleation, comminution in shear cells, silo flow, rock fragmentation, and fault gouge evolution (Weatherley 2008).

LIGGGHTS is an open-source DEM simulation software developed by Sandia National Laboratories, a US Department of Energy institution, in 2011. It can simulate particulate materials and aims to solve industrial problems. The program is based on LAMMPS, a classical molecular dynamics simulation code designed to run efficiently on parallel computers (Kloss et al. 2012).

2.3.3 Application of DEM Simulation with Different Engineering Problems

Anandarajah (1994) proposed a method to carry out research into the stress-strain behavior of cohesive soils. His method approximately simulated double-layer repulsive force between inclined particles and formation of new contacts, deletion of existing contacts, and slip between particles, by using suitable force-displacement laws for the mechanical contacts.

Buttlar and You (2001) investigated the use of asphalt technology to reduce the need for costly tests to characterize asphalt-aggregate mixtures to design flexible pavement structures and materials, which extended traditional discrete element modeling analyses.

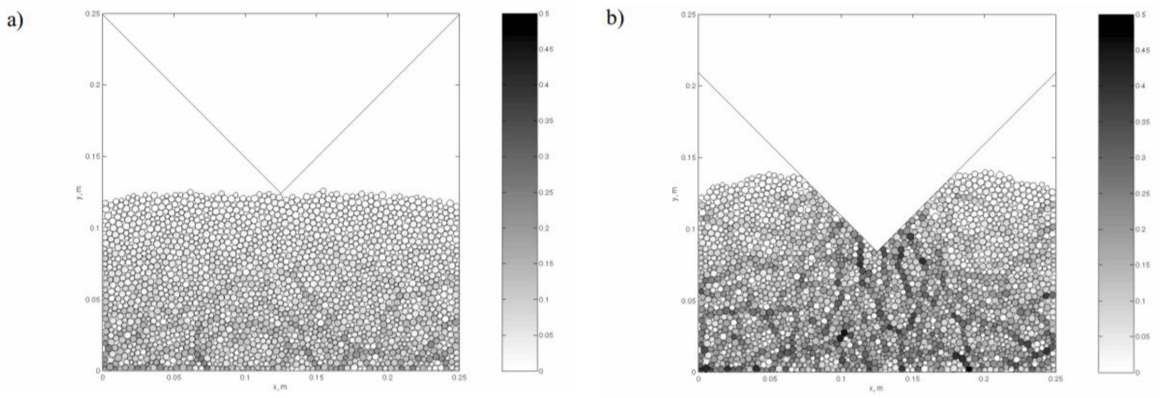


Fig 2.9 2D particles of discrete element method to investigate the penetration in granular media (Balevičius et al. 2004)

Balevičius et al. (2004) investigated keel penetration in a granular model using DEM simulation. DEMMAT code, a DEM software that relies on procedural and object-oriented approaches (Balevičius et al. 2005), was introduced to generate a time integration algorithm. The visco-elastic granular media composed of circular particles is presented in Fig 2.9.

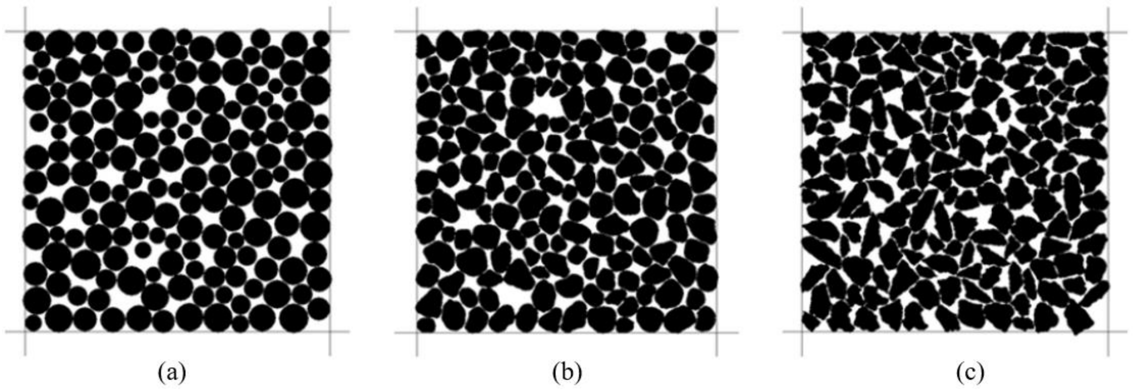


Fig 2.10 Identical grain size distributions prepared at the same void ratio and arranged to similar initial fabrics: (a) Glass beads; (b) Ottawa rounded sand; and (c) Ottawa angular sand. (Ashmawy et al. 2003)

Ashmawy et al. (2003) evaluated the influence of particle shape on liquefaction behavior using 2D DEM. Particle assemblies of varying degrees of angularity were subjected to simulated undrained cyclic shear conditions to assess their liquefaction susceptibility. Fig 2.10 shows the assemblies of particles with different shapes but identical grain size distribution, which were used to evaluate the effect of grain morphology on the cyclic load response.

Balevičius et al. (2005) applied three versions of a software code to simulate granular material dynamics using DEM. Their codes DEMMAT_F90 and DEMMAT_PAS were used to implement a purely procedural method using two programming languages, FORTRAN 90 and OBJECT PASCAL. In contrast, the code DEMMAT_CPP represented a strictly object-oriented programming approach using C++.

Martin et al. (2006) investigated the free sintering of metallic powders using DEM, which allowed the particulate nature of the material to be taken explicitly into account. Ömer (2006) extended the usage of DEM to analyze the bearing capacity of shallow foundations. Peña et al. (2006) used DEM to examine masonry structures represented by discrete elements as assemblies of blocks or particles, an idealization of their discontinuous nature that governs their mechanical behavior.

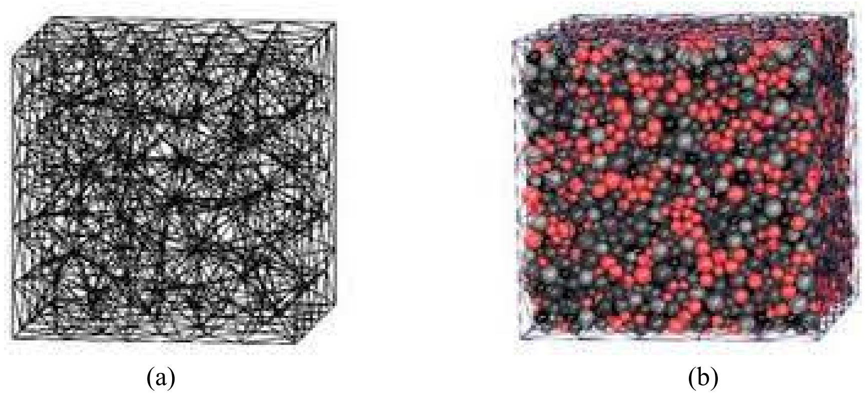


Fig 2.11 3D mesh made of unstructured tetrahedra (a) and its corresponding polydisperse sphere packing (b) (Jérier et al. 2008)

Jérier et al. (2008) developed a new geometric algorithm based on tetrahedral meshes to generate dense isotropic arrangements of non-overlapping spheres with different sizes. The proposed method first fills a tetrahedral mesh with spheres in contact (i.e., hard-sphere clusters) shown in Fig 2.11 (a). Large empty spaces in the model are then detected and filled with new spheres to increase packing Fig 2.11 (b).

Belheine et al. (2009) noted that using spherical elements within the DEM can reduce computational costs, but this oversimplification of the granular geometry has drawbacks when quantitatively assessing the model even for frictional geomaterials. To overcome this limitation, they recommended that the local constitutive law considers the transfer of a moment between elements. Adding normal and shear local interaction forces to particles can improve the simulation accuracy.

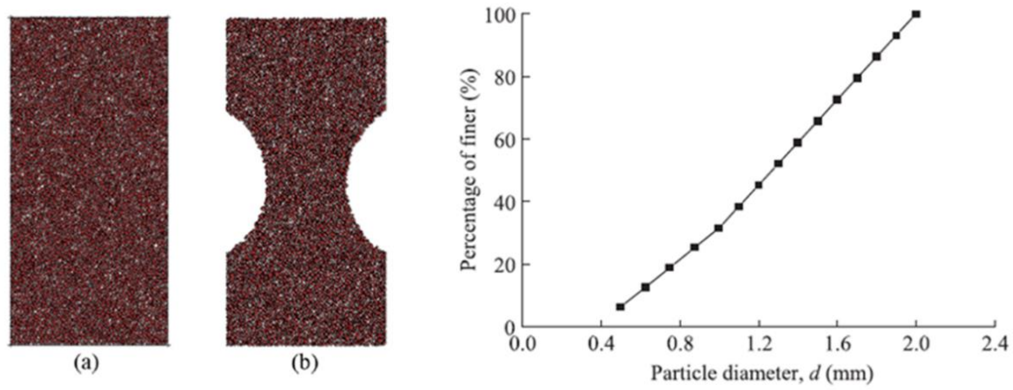
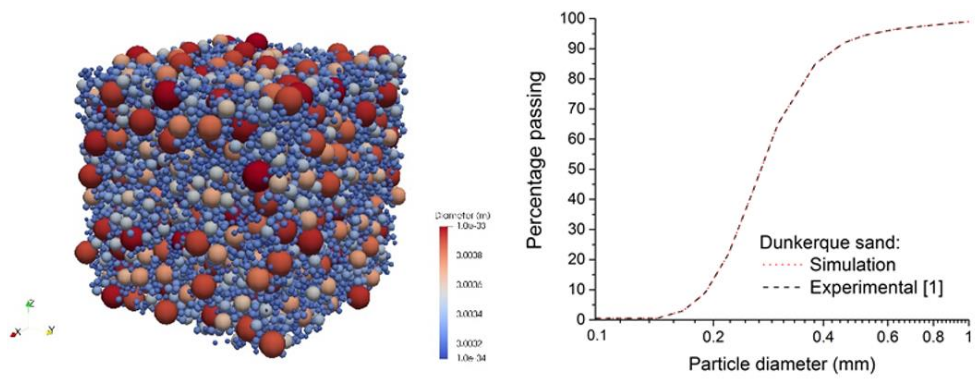


Fig 2.12 Particle-size distribution and rock samples for (a) uniaxial compression test and (b) direct tension test (Shi et al. 2015)

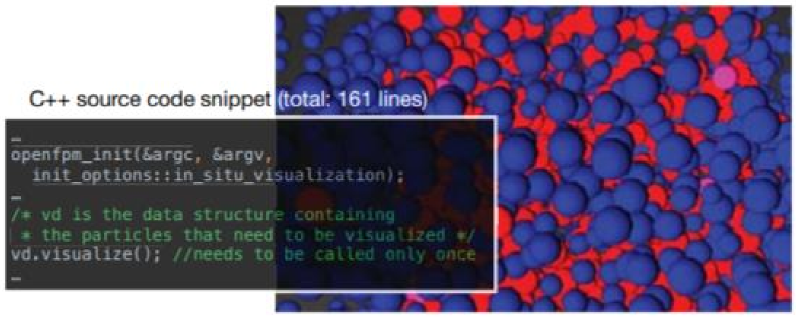
Shi et al. (2015) conducted research on the Distinct Element Method, a variation of DEM, with a novel bond contact model to discover the microscopic physical origin of macroscopic behaviors of weathered rock and capture the changing laws of microscopic parameters observed decaying properties of rocks during weathering. Fig 2.12 shows the particle-size distribution curve and rock particles of their model.



(a) (b)

Fig 2.13 DEM assembly of particles for triaxial testing (a) and particle-size distributions of Dunkerque sand (b) (Aghakouchak 2015)

Aghakouchak (2015) investigated the behavior of Dunkerque sand subjected to cyclic loading using DEM. Fig 2.13 shows the particle assembly and the particle-size distribution curves of the experimental and simulated sands.



```
…
openfpm_init(&argc, &argv,
  init_options::in_situ_visualization);
…
/* vd is the data structure containing
 * the particles that need to be visualized */
vd.visualize(); //needs to be called only once
…
```

Fig 2.14 An OpenFPM-based distributed Molecular Dynamics simulation visualized in situ using the prototype, and the lines of code that needed to be changed to enable in situ visualization (Incardona et al. 2019)

Incardona et al. (2019) introduced OpenFPM, an open and scalable framework, to provide an abstraction layer for numerical simulations using particles and meshes. Fig 2.14 shows the model they built and the C++ source code snippet.

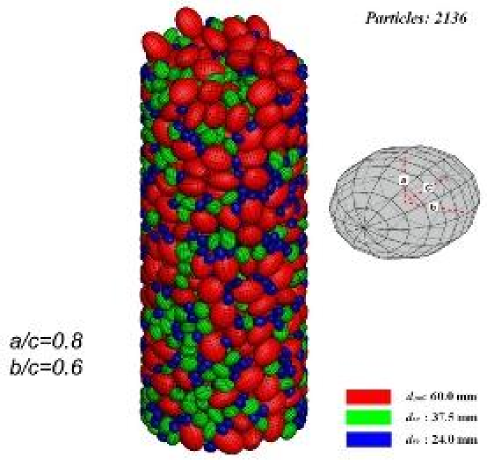


Fig 2.15 The ellipsoidal particles after deposition (Liu et al. 2019)

Liu et al. (2019) investigated the soil micro-scale responses during shield tunnel excavation in a sandy-cobble stratum using DEM. Their model, made up of the ellipsoidal particles, is shown in Fig 2.15.

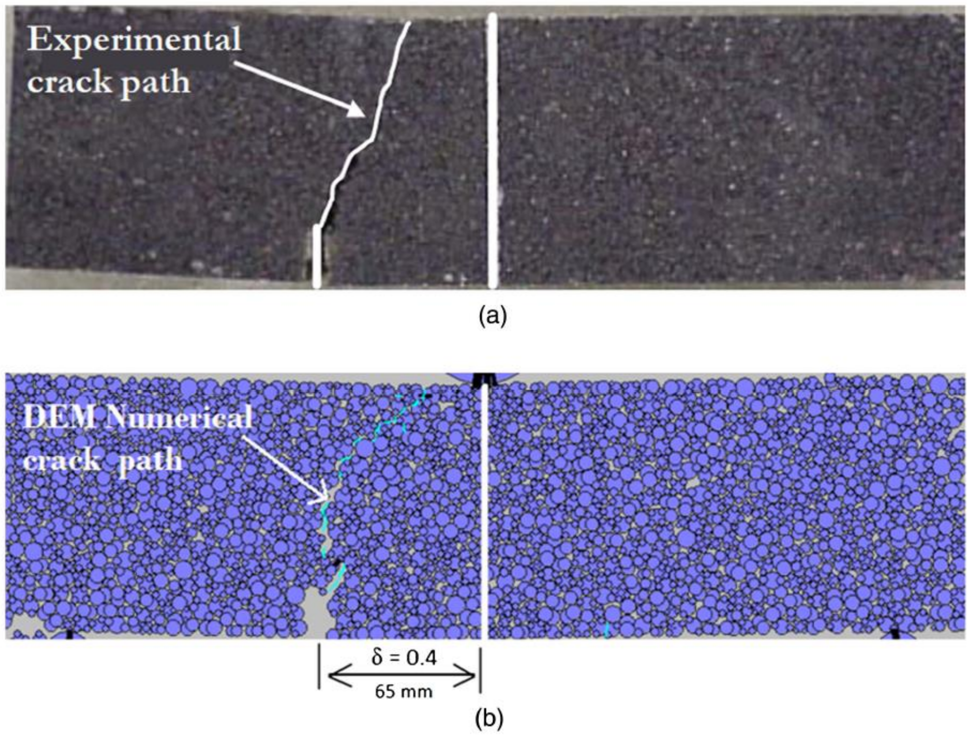


Fig 2.16 Mixed-mode crack test: (a) crack observed in the experimental test; and (b) crack predicted through numerical simulation with the PFC2D in the study of Lopez et al. (2020)

Fig 2.16 compares a DEM mixed-mode crack model to an experimental crack path obtained using the single-edge-notched beam (SEB) test. PFC2D was used to generate the DEM model and perform the SEB test (Lopez et al. 2020).

2.4 Poisson Disk Sampling and Grid Sampling

Supersampling is a spatial anti-aliasing method in computer graphics to reduce the jagged parts at the margin of objects in rendered images (Beets and Barron 2000). Poisson Disk Sampling and Grid Sampling are two extensively used supersampling techniques to find the sample locations in pixel space and decide the color of adjacent pixels (Klassen 1989).

This section first introduces the supersampling technology and focuses on Poisson Disk Sampling and Grid Sampling. The second part presents the current approaches used to perform these two sampling methods.

2.4.1 Supersampling

Supersampling is performed to remove aliasing or jaggies from rendered pictures in computer programs. Real-world objects have continuous smooth curves and lines, but a computer screen shows the viewer using a large number of tiny squares called pixels. These pixels have the same size, and each one has a single color. A line can only be shown as a collection of pixels and therefore appears jagged unless it is perfectly horizontal or vertical. The supersampling technique aims to reduce this effect (Zvekan 2004). The method of deciding the sample positions in pixels, representing the color of adjacent pixels, is the critical aspect of the technology (Cugowski 2016; Zoeken 2004).

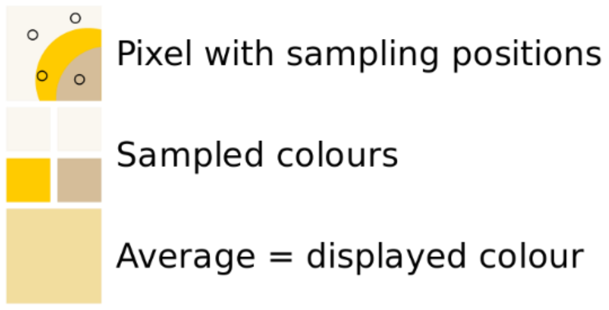


Fig 2.17 How supersampling works on a pixel (Taxel 2016)

Fig 2.17 shows how supersampling works. An average color value is calculated based on color samples taken at several points inside the pixel (not just at the center). It is generally done by rendering the image at a much higher resolution than the one displayed, then shrinking it to the desired size, using the extra pixels for calculation. This method can generate a down-sampled image with smoother transitions from one line of pixels to another along the edges of objects (Cugowski 2016).

Taking samples to characterize the colors in pixels plays a pivotal role in supersampling. Several algorithms presented below are commonly used to decide the samples' positions.

At its simplest, a random algorithm randomly selects the X and Y coordinates of points located within the width and height of a pixel. However, the main limitation of this method is that it cannot generate samples with a random and uniform distribution.

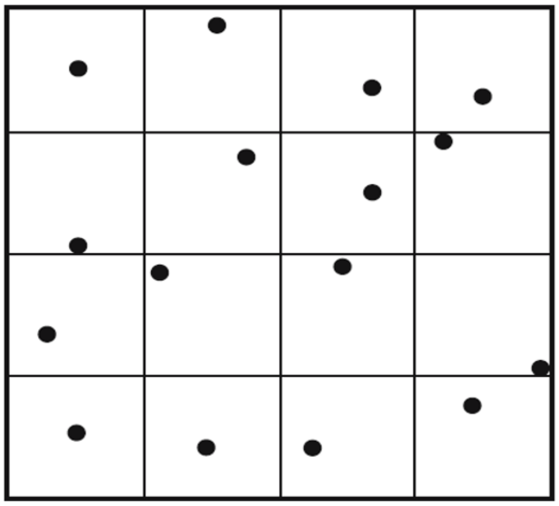


Fig 2.18 Points distributed by Grid Sampling (Chappell et al. 2013)

Fig 2.18 illustrates the Grid Sampling method. The whole canvas is divided into same-sized cells treated as sub-pixels, and then one pixel is randomly selected from every cell in the grid. The distribution of points created in this method is more even than the random algorithm, but two randomly selected points may still be close to one another.

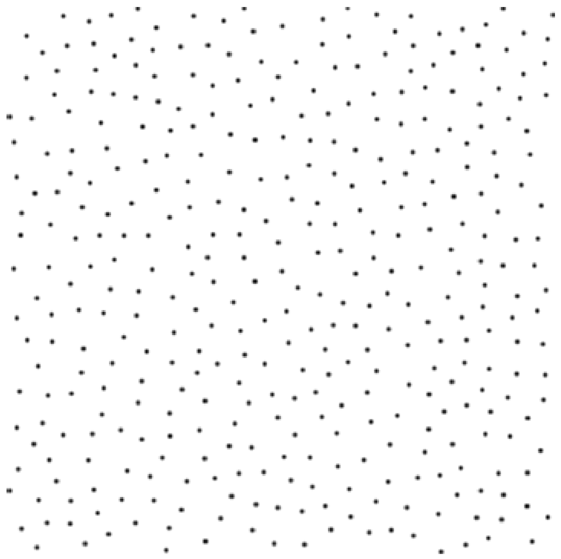


Fig 2.19 Points distributed by Poisson Disk Sampling (Bridson 2007)

The Poisson Disk algorithm can produce a random set of tightly packed points. The distances between them are not smaller than a specified minimum distance. Fig 2.19 shows a point distribution generated with Poisson Disk sampling (Bridson 2007).

The points generated using Poisson Disk Sampling are randomly placed and roughly evenly spaced. This feature makes Poisson Disk Sampling the most popular supersampling technique in computer graphics and several related fields, such as video game design (Hemalatha 2019).

2.4.2 Poisson Disk Sampling and Grid Sampling Generation Algorithms

Due to its excellent blue noise spectral properties, the Poisson Disk distribution is widely used for image sampling. Researchers in computer graphics and computer science developed techniques to generate such a distribution more efficiently (Liang et al. 2015).

Cook (1986) generated blue noise sample patterns with Poisson Disk distributions and used the naive rejection-based approach for generating Poisson Disk samples, such as dart-throwing.

Dunbar and Humphreys (2006) presented a new method for sampling using the dart-throwing method in $O(N \log N)$ time and introduced a novel and efficient variation of generating samples with Poisson Disk distributions in $O(N)$ time and space, which alleviates the usually high computational cost associated with the real-time generation of Poisson Disk distribution.

Bridson (2007) found that existing efficient techniques cannot generalize a blue noise distribution beyond two dimensions. He modified the dart-throwing to generate Poisson Disk samples in $O(N)$ time and easily implemented them in arbitrary dimensions.

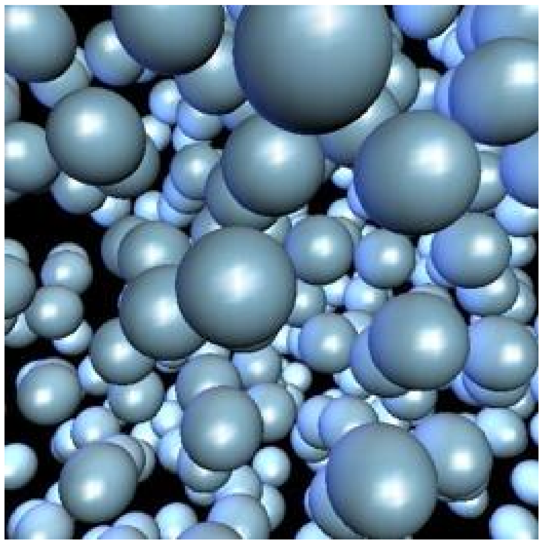


Fig 2.20 Spheres placed at points in a Poisson Disk sample in 3D (Tulleken 2008)

Tulleken (2008) proposed a method that is relatively easy to implement and runs reasonably fast. His idea was to generate points around existing points and check whether they could be added while not disturbing the minimum distance requirement. Cells in a grid are used to perform fast point lookups. Two lists keep track of points that are generated and those that need processing. Fig 2.20 shows 3D spheres based on points with Poisson Disk distribution.

Gamito and Maddock (2009) extended the Poisson Disk distribution of a 2D model to 3D or any higher-dimensional space models. Their method generated distributions with the same statistical properties and yielded more accurate results than methods relying on the brute force dart-throwing algorithm. The technique has $O(N \log N)$ time complexity relative to the number of samples. The method can generate maximal distributions in which no further examples can be inserted after the algorithm is completed.

2.5 Computer Languages and Visualization Tools Used in the Algorithm Development

This section introduces the computational tools that are at the core of this research work, such as the programming language used to build the algorithm, the model visualization tools, the data formats employed to transfer data, and the code editor software.

The first subsection gives a brief description of Python, a programming language developed by Van (2009) used to develop the algorithm, and NumPy (Travis 2015), a Python-based library used to manipulate data. The second subsection presents Yade (Donzé et al. 2009), the open-source framework for discrete numerical models that performs the particle deposition process, the last step before the model is imported into a DEM simulation system. The third subsection introduces JavaScript, a programming language first released by Netscape in 1995, and p5.js (McCarthy et al. 2015), a graphical JavaScript-based library that visualizes 2D models. ParaView, an extensively used 3D model visualization tool to showcase 3D models, is then described. The fourth subsection deals with two data formats, CSV and JSON (Crockford 2011), used as input and output data between the aforementioned programs. Sublime Text, a code editor developed by Jon Skinner in 2008, using which the algorithm is written, is finally presented.

2.5.1 Python and NumPy

Python is an interpreted, high-level, and general-purpose programming language with a design philosophy that emphasizes code readability using significant whitespace. The language constructs and object-oriented approach help programmers write clear, logical code for small and large-scale projects (Kuhlman 2012).

Guido van Rossum first released Python in 1991 (Venners 2003). Python 2.0 was released in 2000 and introduced new features, including list comprehensions and a garbage collection system with reference counting. Python 3.0, released in 2008, was a significant revision of the language that is not entirely backward-compatible, and Python 2 code does not run unmodified on Python 3 (Van 2009; Bill 2003).

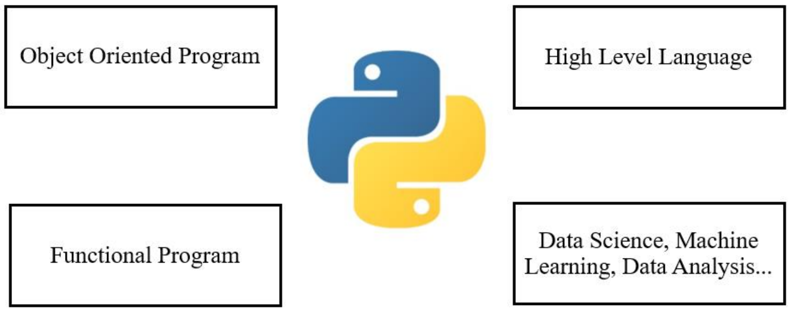


Fig 2.21 Python introduction (Van 2009)

Python is designed for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology, shown in Fig 2.21. It has found many applications in data science, machine learning, data analysis, and web development. (Bhavsar 2020).

NumPy, an open-source Python-based library (Travis 2015), provides functions for large, multi-dimensional arrays and matrices, along with an extensive collection of high-level and complex mathematical functions to deal with data (Harris et al. 2020).

2.5.2 Yade

Yade is an extensible open-source GNU/GPL software framework for discrete numerical models and primarily focuses on DEM (Donzé et al. 2009). The computation parts of Yade are written in C++ using a flexible object model, allowing the independent implementation of algorithms and interfaces. Python is also introduced to realize rapid and concise scene construction, simulation control, postprocessing, and debugging. Fig 2.22 shows a concrete specimen, subject to three-point bending composed of discrete particles, connected using a cohesive law, generated by Yade (Donzé et al. 2009).

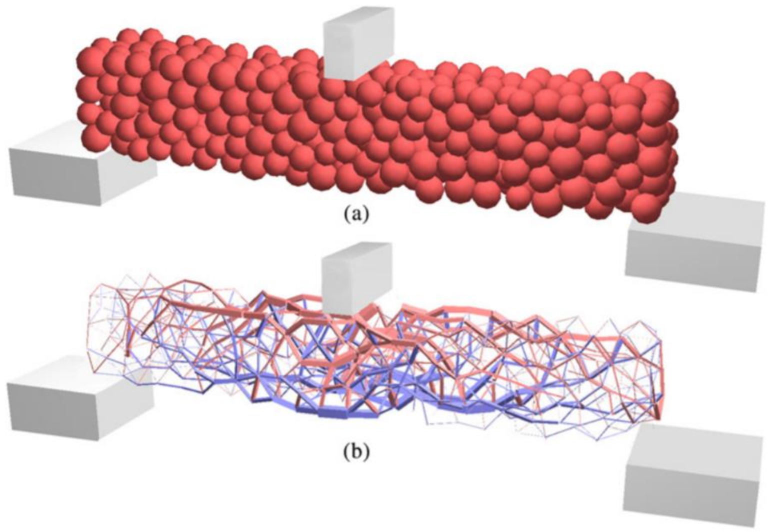
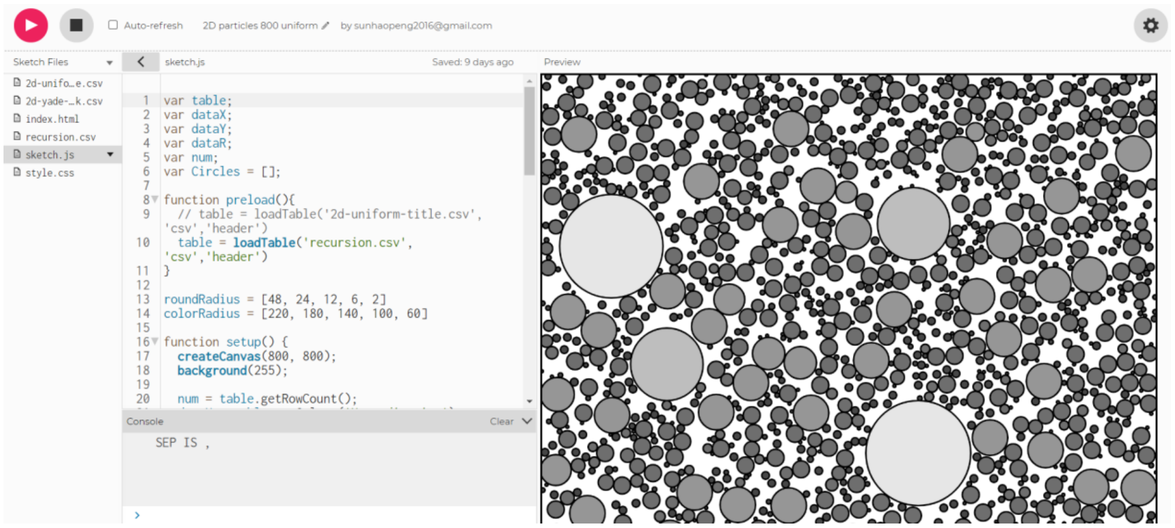


Fig 2.22 A DEM specimen of concrete (aggregates connected using cohesive law), subject to three-point bending, generated by Yade (Donzé et al. 2009)

2.5.3 JavaScript and P5.js

This research uses JavaScript and P5.js as a 2D model visualization tool. JavaScript is a high-level, just-in-time compiled, multi-paradigm programming language (Alex 2018). All popular modern Web browsers currently support JavaScript with built-in execution environments (Flanagan 2011; Alex 2018). JavaScript was first introduced in Netscape (1995) and developed by Brandon Eich.



```
var table;
var dataX;
var dataY;
var dataR;
var num;
var Circles = [];

function preload(){
  // table = loadTable('2d-uniform-title.csv',
'csv','header')
  table = loadTable('recursion.csv',
'csv','header')
}

roundRadius = [48, 24, 12, 6, 2]
colorRadius = [220, 180, 140, 100, 60]

function setup() {
  createCanvas(800, 800);
  background(255);

  num = table.getRowCount();
```

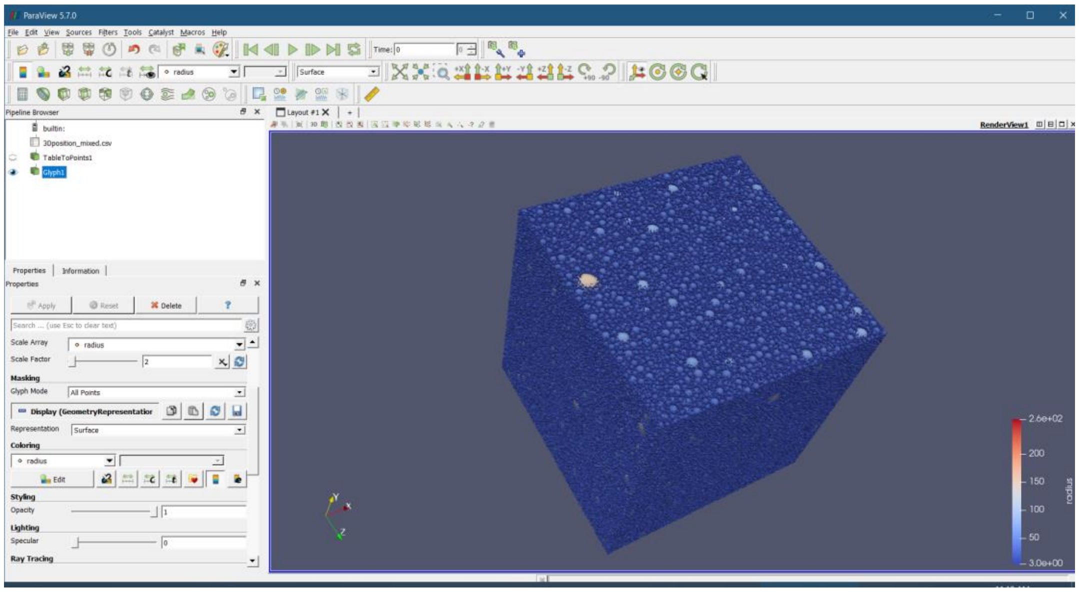
```
SEP IS ,
```

Fig 2.23 The p5.js web editor interface visualizing a 2D packing of circles generated by the algorithm proposed in this thesis

McCarthy developed p5.js in 2013 by collaborators supported by the Processing Foundation and NYU ITP. It is an open-source JavaScript library for creative coding and contains a complete set of drawing functionality (McCarthy et al. 2015). The P5.js web editor interface is shown in Fig 2.23.

2.5.4 ParaView

ParaView, an open-source multiple-platform application for interactive, scientific visualization, is employed as a 3D model visualization tool in this thesis. It was first released in 2020 by Kitware Inc. and Los Alamos National Laboratory (Röber 2014).



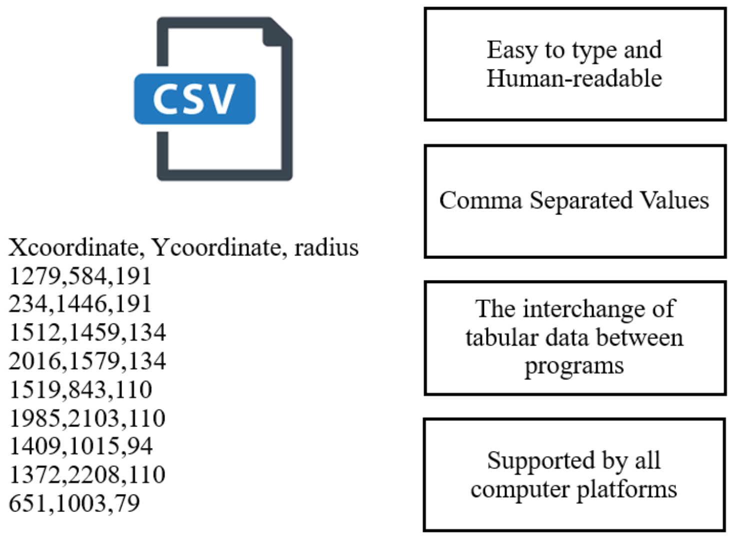
Scale Array: radius
Scale Factor: 2
Glyph Mode: All Points
Representation: Surface
Coloring: radius
Opacity: 1
Specular: 0

Fig 2.24 ParaView software interface visualizing a 3D packing of spheres generated by the algorithm proposed in this thesis

ParaView can handle structured (uniform rectilinear, non-uniform rectilinear, and curvilinear grids), unstructured, polygonal, image, multi-block, and supports many input/output data formats. An intuitive and flexible interface enables users to change filters by directly interacting with the 3D view using 3D widgets. Paraview has been used in many different fields to analyze and visualize scientific data sets (Moreland and Greenfield 2007). The ParaView software interface is shown in Fig 2.24.

2.5.5 JavaScript Object Notation and Comma Separated Values

Comma Separated Values (CSV) is a text file format that this research uses to export data from the algorithm and import it into visualization tools and the deposition simulation program. Data in the CSV file is saved in a tabular form.

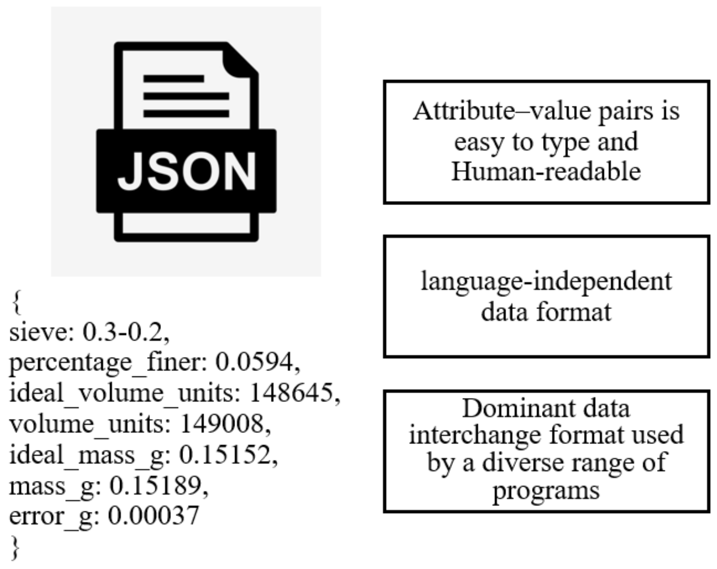

CSV
Easy to type and Human-readable
Comma Separated Values
The interchange of tabular data between programs
Supported by all computer platforms

```
Xcoordinate, Ycoordinate, radius
1279,584,191
234,1446,191
1512,1459,134
2016,1579,134
1519,843,110
1985,2103,110
1409,1015,94
1372,2208,110
651,1003,79
```

Fig 2.25 CSV file format example and features

CSV file is easy to type compared with fixed-column-aligned data and widely used to exchange data in many programs and software (Shafranovich 2005). Fig 2.25 shows a simple CSV file and its characteristics.

In this research project, JavaScript Object Notation (JSON), a lightweight data-interchange format, is used to store the statistical results of the DEM models generated by the algorithm. It is easy for people to read and write and for machines to parse and generate (Freeman 2019). JSON was a subset of the JavaScript scripting language, Standard ECMA-262 3rd Edition (1999) (Stefanov 2010). The data format was first specified by Crockford (2011).


JSON
Attribute–value pairs is easy to type and Human-readable
language-independent data format
Dominant data interchange format used by a diverse range of programs

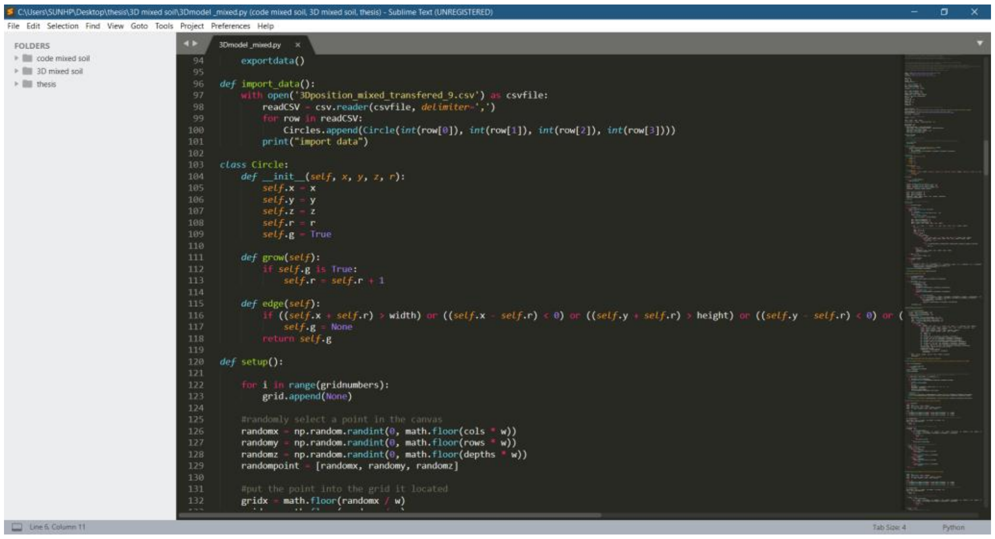
```
{
sieve: 0.3-0.2,
percentage_finer: 0.0594,
ideal_volume_units: 148645,
volume_units: 149008,
ideal_mass_g: 0.15152,
mass_g: 0.15189,
error_g: 0.00037
}
```

Fig 2.26 JSON file format example and features

A JSON example and its outstanding features are shown in Fig 2.26. It is built on a collection of name/value pairs. In various programming languages, this is done as an object, record, struct, dictionary, hash table, keyed list, or associative array, i.e., an ordered list of values (Freeman 2019).

2.5.6 Sublime Text

All codes in this thesis were written and edited in Sublime Text, a shareware cross-platform source code editor developed by Skinner and Sublime HQ company and released in 2008 (Wes 2014). It supports many programming markup languages, and users can insert functions using plugins (Brian 2021). Fig 2.27 shows the interface of the Sublime Text editor.


C:\Users\SUNHP\Desktop\thesis\3D mixed soil\3Dmodel _mixed.py (code mixed soil, 3D mixed soil, thesis) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
code mixed soil
3D mixed soil
thesis
3Dmodel _mixed.py
Line 6, Column 11
Tab Size: 4
Python

```python
    exportdata()

def import_data():
    with open('3Dposition_mixed_transfered_9.csv') as csvfile:
        readCSV = csv.reader(csvfile, delimiter=',')
        for row in readCSV:
            Circles.append(Circle(int(row[0]), int(row[1]), int(row[2]), int(row[3])))
        print("import data")

class Circle:
    def __init__(self, x, y, z, r):
        self.x = x
        self.y = y
        self.z = z
        self.r = r
        self.g = True

    def grow(self):
        if self.g is True:
            self.r = self.r + 1

    def edge(self):
        if ((self.x + self.r) > width) or ((self.x - self.r) < 0) or ((self.y + self.r) > height) or ((self.y - self.r) < 0) or (
            self.g = None
        return self.g

def setup():

    for i in range(gridnumbers):
        grid.append(None)

    #randomly select a point in the canvas
    randomx = np.random.randint(0, math.floor(cols * w))
    randomy = np.random.randint(0, math.floor(rows * w))
    randomz = np.random.randint(0, math.floor(depths * w))
    randompoint = [randomx, randomy, randomz]

    #put the point into the grid it located
    gridx = math.floor(randomx / w)
```

Fig 2.27 Sublime Text code editor interface

Chapter 3 Development of Particle Generation Algorithm

3.1 Introduction

The process of generating the particle assemblies characterized by randomness and controllability is introduced in this chapter. The algorithm builds particle packings sieve-by-sieve. Particles of the sieve with the largest opening size are computed based on the distribution obtained from Poisson Disk Sampling. Particles belonging to the remaining sieves are then inserted during the void-filling process using the Grid Sampling method.

Section 3.2 defines the basic parameters that are set to build models based on real soil data. They include: i) the canvas size, an area where particles are located; ii) the unit size, which is the scale between the length in the simulation model and the length in actual soil samples; iii) the cell size, which initializes grids covering the whole canvas to generate particles with Poisson Disk distribution and implement the void-filling process; iv) the number of particle insertion rounds and the radius ranges of each particle insertion process, used to generate particles sieve-by-sieve; v) the target volumes for particles in each sieve, calculated by the algorithm based on the particle-size distribution and the void ratio of real soils defined by users.

Section 3.3 describes the generation of particles with the Poisson Disk distribution. This procedure generates particles corresponding to the largest opening sieve, laying the model's foundation. For 2D models, the points based on the Poisson Disk distribution are first generated. A "Circle" object, which can be regarded as the template of circular particles with parameters and specific characteristics, is introduced to create particles based on previously generated points. 3D particle assembly is generated using an approach similar to that of 2D particle packing. The differences are that the points of Poisson Disk distribution in 3D are created, and the object used to generate particles is changed to a "Sphere" object by extending the algorithm into the third (Z) dimension.

Section 3.4 deals with the void-filling process using the Grid Sampling method to insert particles belonging to the remaining sieves. The process includes: i) omitting the fully covered cells, which means that when particles are inserted, the algorithm does not check cells entirely occupied by a previously-generated particle to accurately locate the voids and thus reduce the running time; ii) the particle validation is the method of determining whether the algorithm can generate a valid particle based on a randomly selected point in a non-filled cell; iii) recursive call is used to rerun the particle insertion process if the volume error between the algorithm-generated model and the actual soil is beyond the acceptable level or if the particle volume does not reach the target.

Section 3.5 introduces the particle assembly deposition process that is implemented in DEM before the simulation begins. The particle packing is deposited under gravity, becoming denser with particles touching each other. Yade DEM simulation framework is used to implement this process.

This chapter gives details on the algorithm development by providing flow diagrams to illustrate the process and pseudocodes detailing how models are built in the code. There are also several examples showing the results of every model-generated step.

3.2 Setting Basic Parameters

The fundamental parameters that require user input in the algorithm to build models include the unit size, the canvas size, the cell size, the number of particle-insertion rounds, the particle radius range for each sieve, and the target volumes for particles in every sieve. The definition of each parameter, as well as their recommended values, are provided in this section.

3.2.1 Unit Size

The unit size is defined as the length that one unit in the algorithm-generated model is equivalent to in the real condition. It is determined based on the sieve sizes that the particle-size distribution provides. There are two requirements that the unit size must satisfy:

a) The opening size of every sieve should be divisible evenly by the unit size because the algorithm needs to ensure the particle size can perfectly fit every sieve size.
b) The unit size should not be too small because a small value will over-refine the canvas and exponentially increase the algorithm's running time.

Thus, the maximum value that can uniformly divide openings of all sieves in the target particle-size distribution is recommended. For example, if the target is to generate models for soil samples with particle-size distributions ranging from US sieve No.4 to US sieve No.200. Unit sizes of 0.05 mm, 0.025 mm, and 0.0125 mm may be selected, as shown in Table 3.1. If the target distribution has a minimum radius of soil particles of 0.075 mm, setting one unit as 0.025 mm is recommended because it can divide all sieve openings evenly. A unit size of 0.0125 mm can also meet the requirements, but it is smaller than 0.025 mm, and further dividing the canvas, which translates into an increase in running time. A unit size of 0.05 mm is not suitable in this case because 0.075 mm, the opening size of the last sieve, cannot be divided evenly by 0.05 mm, which violates the first requirement.

Table 3.1 The opening sizes of typical sieves and their corresponding units

US sieve No.	Opening (mm)	Unit (1 unit = 0.05mm)	Unit (1 unit = 0.025mm)	Unit (1 unit = 0.0125mm)
4	4.75	95	190	380
10	2.00	40	80	160
20	0.85	17	34	68
30	0.60	12	24	48
40	0.425	9	17	34
60	0.25	5	10	20
100	0.15	3	6	12
200	0.075	-	3	6

3.2.2 Canvas Size

The canvas is an area where circular or spherical particles are generated. For 2D models, it consists of a square, while it is a cube in 3D models. The inequality in Equation 3-1 is the recommended method to estimate the minimum canvas size. The algorithm generates particles sieve-by-sieve. As such, the canvas size should be large enough to contain several particles belonging to the largest opening sieve. The minimum canvas size of 3D models should contain at least three particles of the largest opening sieve. When the 3D canvas size is decided upon, the 2D square canvas size is assigned the same value as the 3D model. Thus, a canvas size that is set greater than this minimum value calculated by Equation 3-1 is suitable.

$$\frac{s_m \ / \ (1+e) \times p}{v_{max}} \geq 3 \tag{3-1}$$

where s_m is the minimum canvas size, e is the void ratio, p is the finer percentage of the particles of the largest opening sieve based on the particle-size distribution that the algorithm aims toward, and v_{max} is the individual volume of particles of the largest opening sieve.

3.2.3 Cell Size

The generation of particles with Poisson Disk distribution requires a grid to cover the whole canvas. The cell size is set to $r/\sqrt{2}$ (r is the minimum distance between points) as proposed by Bridson (2007) to ensure points can be distributed uniformly in the canvas. The minimum distance between points is initially set to half of the canvas length. The algorithm automatically adjusts the value until the volume of particles in the largest opening sieve reaches the target (the details are presented in Section 3.3.4).

After creating particles with Poisson Disk distribution, the algorithm builds a new grid that divides the whole canvas into smaller cells to locate void spaces and fill them up by inserting particles belonging to the remaining sieves.

Fig 3.1 (a) shows the grid with a cell size of $r/\sqrt{2}$ used to generate particles with Poisson Disk distribution. Fig 3.1 (b) presents the refined grid which is used in the void-filling process. The white circular particles are generated based on Poisson Disk distribution, and the grey particles are inserted during the void-filling process.

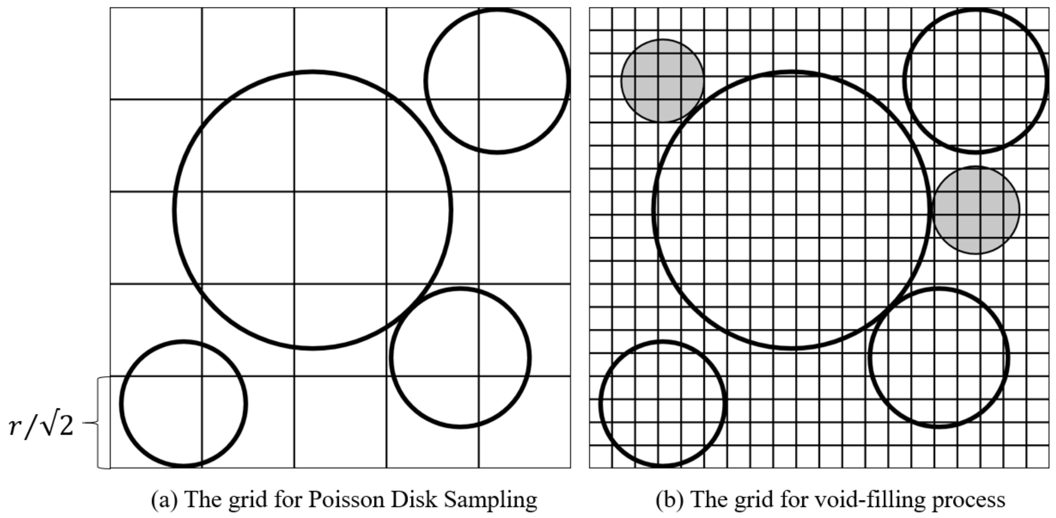

$r/\sqrt{2}$
(a) The grid for Poisson Disk Sampling
(b) The grid for void-filling process

Fig 3.1 Grids for Poisson Disk Sampling and void-filling process

During the void-filling process, the cell size must meet the following requirements:

a) The canvas should be divided evenly by cells because a grid (i.e., all cells) should cover the whole canvas without blank margins.
b) The cell size should be small enough to target tiny voids for the smallest particles, ensuring that each cell contains no more than one of the smallest particles.

However, the cell size should not be too small because the smaller the cell size, the more cells the algorithm needs to check when inserting particles during the void-filling process and the longer the running time. It is recommended to select the largest value that can simultaneously satisfy both requirements.

For example, suppose the unit size is 0.0125mm, the 2D canvas size is 1000unit×1000unit, and the diameter range of minimum particles is from 4 units to 7 units, which is categorized as the lowest sieve. In this case, a cell size of 5unit×5unit is recommended. Using a cell size of 4unit×4unit is also possible, but it will increase the running time because it divides the canvas further, and the algorithm needs to check more cells to find voids. A cell size of 5unit×5unit yields a total of 40,000 cells in this given canvas, while a 4unit×4unit cell size results in 62,500 cells. A cell size of 8unit×8unit can also evenly divide the whole canvas. However, given that the diameter of the smallest particles is 4 units and a square cell of 8unit×8unit can contain at most four smallest particles, the second requirement is violated since one cell cannot contain more than one of the smallest particles. Thus, the cell size of 8unit×8unit cannot be selected.

3.2.4 Number of Rounds of Particle Insertion

The algorithm builds particle packing by generating particles using a sieve-by-sieve method. The number of particle insertion rounds is computed based on the number of sieves that the target particle-size distribution contains.

For example, sandy and fine-grained soils contain eight sieves (US sieves No. 4, 10, 20, 30, 40, 60, 100, and 200). The algorithm generates particles by nine rounds since it must cover all sieves and add a final particle-insertion round for particles with radii smaller than the opening size of US sieve No. 200 and left in the pan at the bottom of the sieve stack.

3.2.5 Radius Range

The algorithm sets the radius ranges for particles in every sieve based on the sieve opening size and the unit size. The various radius ranges ensure that every round of particle insertion is specific to one sieve size. For example, if the unit size is 0.0125 mm and the algorithm is required to build a model based on US sieves, the radius range for every particle insertion round is shown in Table 3.2.

Table 3.2 Radius range for particle-insertion rounds

Round of particles insertion	US sieve No.	Opening(*mm/unit*)	Diameter range (*unit*)
1	4	4.75/380	D > 380
2	10	2.00/160	380 > D ≥ 160
3	20	0.85/68	160 > D ≥ 68
4	30	0.60/48	68 > D ≥ 48
5	40	0.425/34	48 > D ≥ 34
6	60	0.25/20	34 > D ≥ 20
7	100	0.15/12	20 > D ≥ 12
8	200	0.075/6	12 > D ≥ 6
9	pan	-	D < 6

It should be noted that users are required to set the minimum diameters for particles left on the pan at the bottom of the sieve stack. Particle diameters smaller than 6 units make up the diameter range for particles belonging to the lowest sieve. However, the particle size in the last sieve cannot be infinitely small. Thus, the user can set any integer smaller than 6 as the minimum particle diameter. It is preferable to select an even number for the diameter to ensure the particle radius is an integer. Thus, setting the minimum diameter as 4 units or 2 units is recommended.

3.2.6 Target Volume

The algorithm first sets the targets, which are the ideal particle volumes for every sieve, based on the user-defined particle-size distribution and void ratio. Afterward, it generates particles from the first sieve to the last one. When inserting particles, the algorithm simultaneously checks the total volume of particles in a given sieve. If the volume reaches the target, the algorithm stops inserting particles for this sieve and continues generating particles for the next one.

Equation 3-2 is used to calculate the target volume of particles.

$$V_{particle} = V_{canvas}/(1 + e) \tag{3-2}$$

where V is the total volume, e is the void ratio.

Suppose the algorithm aims to create models for homogenous soils (soil densities in all sieves are almost identical). In that case, the "percent finer" between sieves equals the percent of the particle volume in every sieve. Thus, for every sieve, the target volume is calculated by Equation 3-3a.

$$V = (m \times p)/\rho \tag{3-3a}$$

Suppose the soil the algorithm is required to simulate is a soil mixture, such as a sand-silt mixture. In that case, the algorithm allows users to input the densities for sand and silt respectively to calculate the target volumes for every sieve by using Equation 3-3b and 3-3c.

$$V_{sand} = (m \times p)/\rho_{sand} \quad (3\text{-}3b)$$

$$V_{silt} = (m \times p)/\rho_{silt} \quad (3\text{-}3c)$$

where V is the total volume, M is the total mass, p is the "percent finer", and ρ is density.

3.3 Generation of Particles Based on Poisson Disk Distribution

This section describes how 2D and 3D particles are generated for the largest-opening sieve based on Poisson Disk distribution.

The algorithm first creates points with Poisson Disk distribution, which can approximately evenly place samples not too close to one another within a minimum distance in a random fashion. This feature enables the uniform creation of particles with a specific radius, laying the particle packing's foundation. The point generation method the algorithm uses is similar to the approach proposed by Bridson (2007). The next step is to generate particles based on the points created in the previous step. The algorithm builds an "object" as a template to complete the particle generation.

3.3.1 Generation of Points in 2D Based on Poisson Disk Distribution

Points with a Poisson Disk distribution in 2D are created using a three-step process:

a) Initialize a grid that covers the whole canvas,
b) Randomly choose a point in the canvas as the first point,
c) Randomly create points inside the spherical annulus around the existing points and filter them by checking whether those newly-generated points are far enough from other points. If they are, these points are created in the canvas, and if not, they are abandoned.

The third step is repeated until there is no void space left in the canvas where new points can be inserted. The process is described in-depth in the following paragraphs.

a) Build a 2D canvas where the points are placed. Initialize a grid to divide the whole canvas into equal-sized cells and ensure each cell can store at most one point. The cell size is initialized to be bounded by $r/\sqrt{2}$ where r is the minimum distance between points required for a uniform point distribution (Bridson 2007). Implement the cells as a 2D array and assign a default value "undefined" to every cell, indicating no sample inside the cell.

The pseudocode of this step is shown below. ("#" is a symbol in the programming language used to add comments. Lines preceded by this symbol are not executed).

```
function grid ()
    grid = [] # create a grid array and initialize it as empty
    create a 2D canvas
    w = $r/\sqrt{2}$ # set the cell size
    cols = wd of canvas / w # the count of cells on the side of the canvas
    rows = ht of canvas / w
    for every cell inside the cols × rows grid
    # Initialize every element in the grid array as "undefined"
    grid [i] = undefined
```

b) Select the first point randomly inside the canvas, and determine the cell in which the point is located. Initialize an "active" array to temporarily contain the points randomly generated around the existing points and checked as valid. The point is removed from the "active" array after the algorithm checks that there is no void space around this point where new points can be created.

If a point is inside a cell, this cell is assigned as a vector. The vector with a magnitude and a direction simplifies the creation of new points in its spherical annulus from *r* to *2r*. A cell is defined as a vector rather than "undefined", meaning there is already a point inside it. The flow diagram is shown in Fig 3.2.

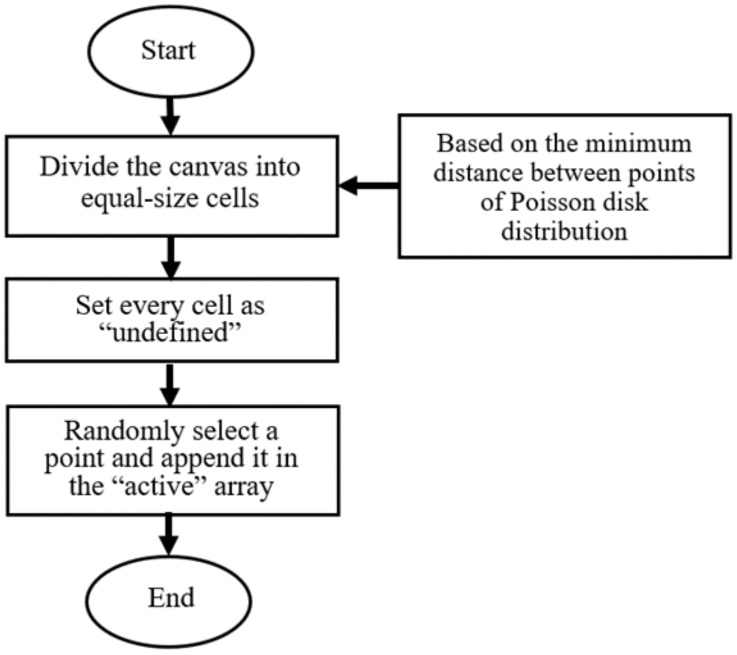

Start
Divide the canvas into equal-size cells
Based on the minimum distance between points of Poisson disk distribution
Set every cell as "undefined"
Randomly select a point and append it in the "active" array
End

Fig 3.2 Flow diagram of steps a) and b)

The pseudocode of the process mentioned above is shown below.

```
function initial_sample ()
    act = [] # create an "active" array and initialize it as an empty array
    # Randomly find a point in the canvas
    x = random (wd)
    y = random (ht)
    i = floor (x / w)
    j = floor (y / w)
    # Pinpoint the cell where this point is placed
    grid [i + j * cols] = pos
```

```
# Create a vector and assign it to the cell
pos = createVector (x, y)
# Append this initial point into act array
 act. push(pos)
```

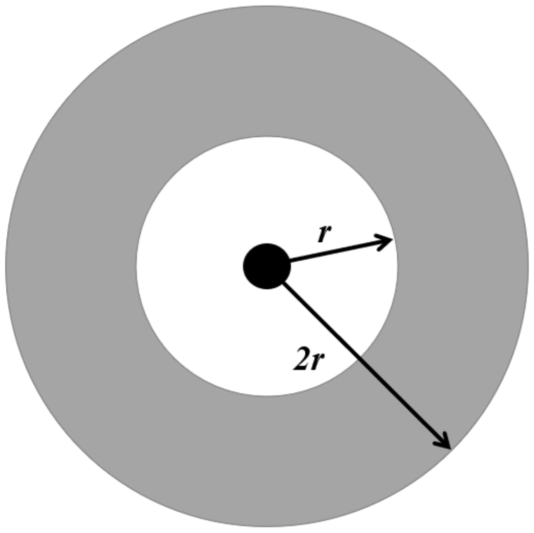


Fig 3.3 Spherical annulus around the chosen point

c) If the "active" array is not empty, there might be void spaces where new points can be inserted. The algorithm then chooses a random sample in the "active" array and generates 30 points, based on Bridson (2007) inside the spherical annulus (the grey annulus shown in Fig 3.3) with the radii ranging between *r* and *2r* (*r* is the minimum distance between points) around this chosen point using the previously generated vector, thereby ensuring the canvas is full packed with points. The vector contains a magnitude that can be used to insert points within the distance from *r* and *2r* and a direction that can randomly generate points 360 degrees inside the spherical annulus. The algorithm then examines each point one by one to determine whether the distance between this newly-generated point and surrounding points ranges from *r* to *2r*.

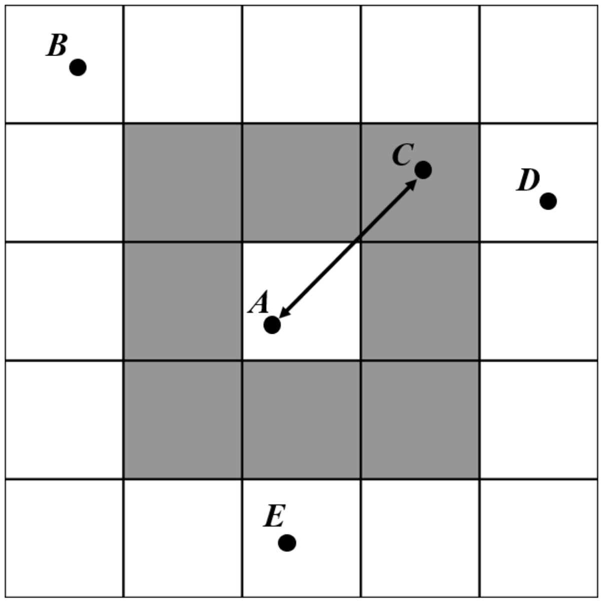


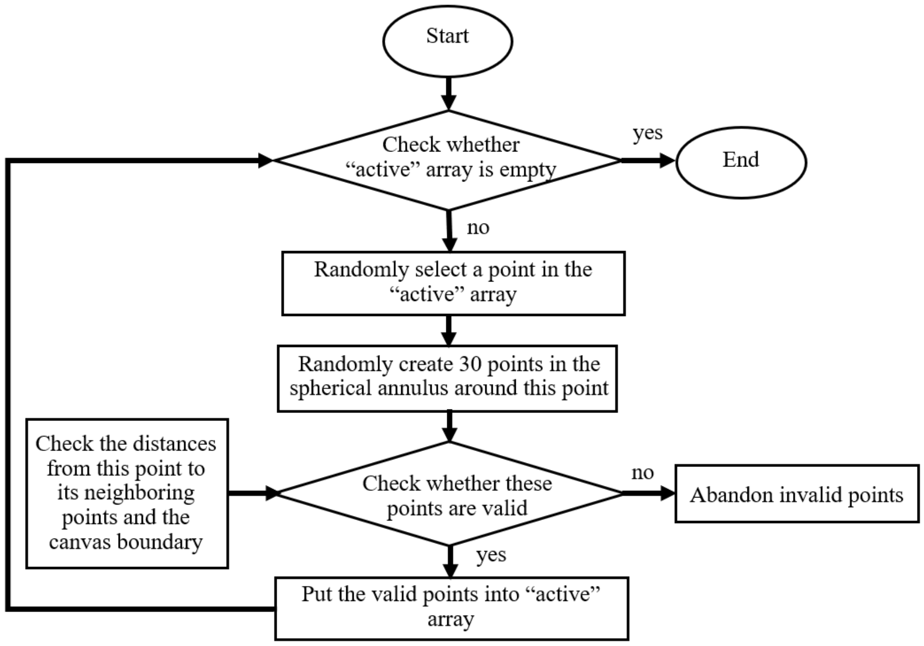
Fig 3.4 Check eight cells around the chosen point *A*

When examining whether a randomly generated point is far enough from neighboring points, the algorithm only checks points inside the eight surrounding cells, such as the grey cells shown in Fig 3.4, to reduce the running time. For example, if a randomly inserted point *A* is checked, the algorithm only checks the distance between point *A* and point *C*. Since points *B*, *D*, and *E* are outside the eight grey cells around point *A*, the algorithm does not check them.

The algorithm also checks whether the distances between a randomly created point and the canvas boundary are greater than the minimum value of the radius range of the largest opening sieve size. If the point is close to the boundary, it means that based on this point the algorithm cannot generate a particle whose radius is within the radius range and this point will not be inserted in the canvas.

If a newly generated point is far enough from its adjacent points and the canvas boundary, the algorithm inserts it in the canvas and appends it to the "active" array. If the distances between a point and its neighboring points are smaller than the minimum distance *r*, this point is abandoned.

The algorithm removes a point from the "active" array after randomly creating 30 points around it, and no extra point can be added, indicating that there is no void space left around the chosen point where new points can be inserted. The algorithm ends when the "active" array is empty, which means the canvas is fully packed with points, and there is no potential void area where extra points can be inserted. The flow diagram of step 3) is shown in Fig 3.5.


Start
Check whether "active" array is empty
yes
End
no
Randomly select a point in the "active" array
Randomly create 30 points in the spherical annulus around this point
Check the distances from this point to its neighboring points and the canvas boundary
Check whether these points are valid
no
Abandon invalid points
yes
Put the valid points into "active" array
Fig 3.5 Flow diagram of step c)

The pseudocode of the process described above is shown below.

```
while the act array is not empty
  ranidx # an element chosen randomly in the act array
  pos = [] # create an empty array to contain positions of new randomly-selected points
  f = false # create a boolean variable that indicates whether there are valid points founded around this element
  # Randomly generate 30 points inside the spherical annulus around a point in the array act
  for try thirty times
      sample # Create a vector variable that facilitates the generation of new points around this selected point in the array act
      m # Create a variable that sets the magnitude of vector sample from r to 2r
      set m to vector sample and append it to the array pos
      find the cell where a sample is placed
      if cell is valid
        # Assume there is at least one valid point around the checked point
         valid = true # a boolean variable initialized as true, meaning it assumes this point is valid in the first place
         for the eight neighbor cells around the checked cell
            index = serial numbers of eight neighbor cells
            neighbor-cells = grid[index] # an array that contains surrounding cells
            if neighbor is valid # means if there is a point inside this neighbor cell
              create a variable dis, which means the distance between sample and neighbor points
              if dis is smaller than the minimum distance r
                  set valid = false # this sample is invalid and needs to be discarded
```

```
# If distances between the sample and neighbor points are greater than the minimum distance r
if valid = true:
 # Check whether min_bdy, the distance between the point and the canvas boundary, is greater than min_r, the minimum value of the radius range of the largest opening sieve size.
 If min_bdy < min_r:
   set valid = false # this sample is too close to the canvas boundary and needs to be discarded
# There is a point that is valid around the sample
     if valid is true
      f = true # assign f variable as true
      put sample in the corresponding cell
      add sample into array act
    # There is no void space around this sample, and remove it from the array act
    if f is false
     delete ranidx in the array act
```

Fig 3.6 presents the points generated based on Poisson Disk distribution. The parameters are that the canvas size was 800unit × 800unit; the minimum distance r was 80 units.

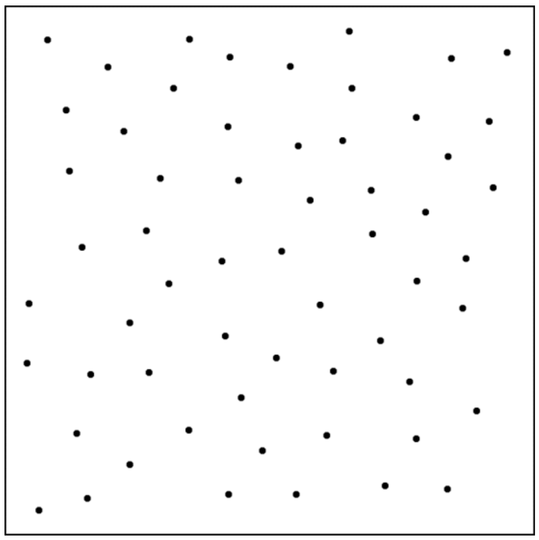


Fig 3.6 Points of Poisson Disk distribution in 2D

3.3.2 Generation of 2D Circular Particles

The 2D particles are generated during a two-step process. The first step consists of building a "Circle" object. It involves parameters including the position, the radius, and the color of a particle, and functions that the algorithm uses to determine whether a particle can continue growing, whether a particle touches the canvas borders, and draw a circular particle in the canvas. In the second step, the algorithm loops through the "grid" array to scan for points inside the cell. If one is found, the algorithm creates a "Circle" instance (an instance, in object-oriented programming, is a specific realization of the object) and starts the particle enlargement process using the object built before.

The positions of circular particle centers are the points that the algorithm generated in the previous section. The algorithm builds a “Circle” object, which can be regarded as a template that contains several parameters and various functions. The “Circle” object is shown in Fig 3.7 and includes the following four parameters:

• the constructor, which is composed of four variables: *X* coordinate, *Y* coordinate, particle radius, and particle color;
• the “grow” function, which determines whether a particle can grow after checking if there is enough space for it to grow;
• the “edge” function, which checks whether a particle touches the canvas borders;
• the “show” function, which draws particles in the canvas using the P5.js library.

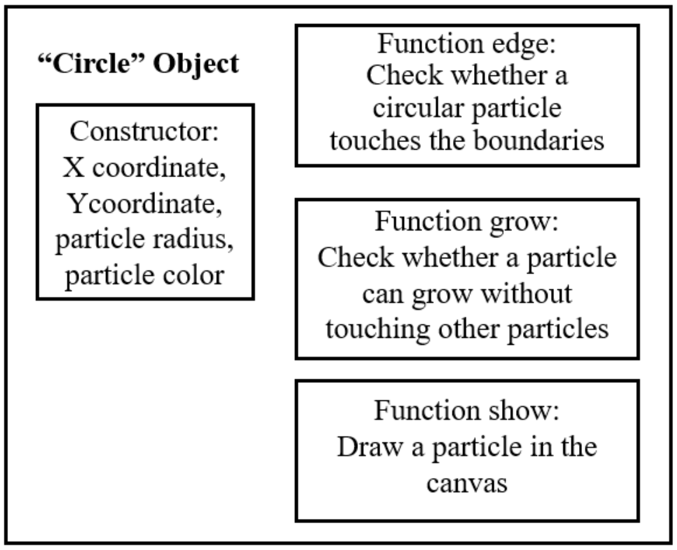

“Circle” Object
Constructor: X coordinate, Ycoordinate, particle radius, particle color
Function edge: Check whether a circular particle touches the boundaries
Function grow: Check whether a particle can grow without touching other particles
Function show: Draw a particle in the canvas

Fig 3.7 “Circle” object used to generate 2D particles

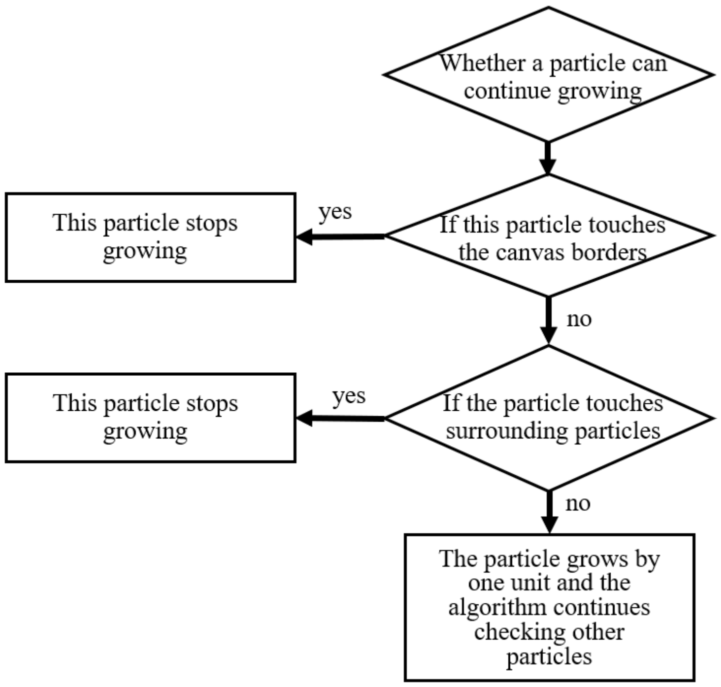
The pseudocode of the process mentioned above is shown below.

```
class Circle
  constructor:
  integer variables: posit, rad, clr
  boolean variable: grow
  grow ()
    if grow is true after checking the distance between this point and the adjacent points
      a particle can continue growing by adding rad
  edge ()
    if the posit shows a particle touches the canvas boundaries
      grow = false
  show ()
      set the stroke weight and clr of the circle
      draw the particle in the 2D canvas
```

After the algorithm builds the "Circle" object, it loops through the "grid" array to create instances for every particle. The particle radius is initially assigned the minimum value of the radius range for the largest opening sieve, and particles are grown in the next step. This minimum value in the range must be equal to or smaller than *r*, i.e., the minimum distance between points in the Poisson Disk distribution. If not, the particles may overlap. The pseudocode is shown below.

```
for each cell in the grid array
    if there is a vector in the cell
        set the weight of the circle stroke using functions offered by P5.js
        # Create an instance using the "Circle" object
        new_circle = new Circle (posit, rad, clr)
        # Add new_circle into array Ptcl
        Ptcl. push(new_circle)
```

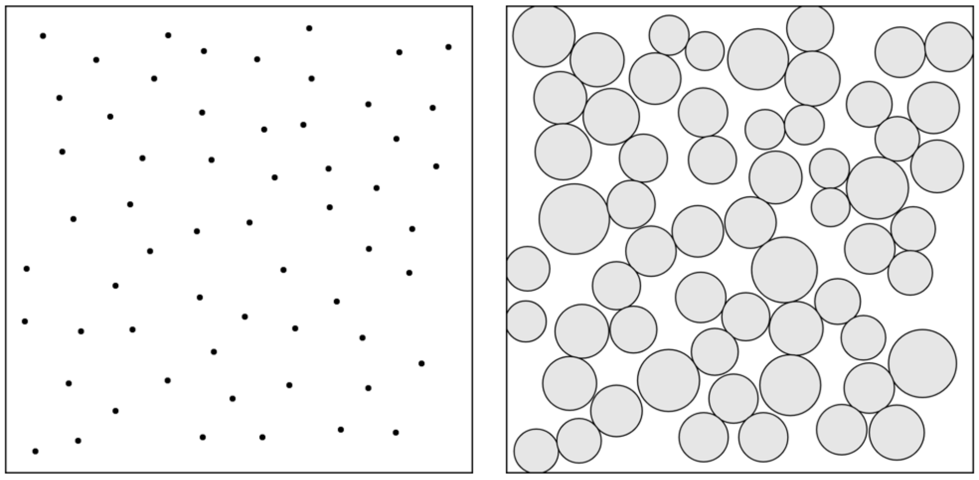
After building instances based on the "Circle" object, the algorithm checks particles one by one to determine whether they can be enlarged. It assesses whether the distances between the center of the checked particle and the centers of the surrounding particles are greater than the sum of two particles' radii. If a particle is too small to touch neighboring particles and the canvas borders, it can continue to grow. It should stop growing if it comes in contact with neighboring particles or boundaries, meaning the particle has finished building. Fig 3.8 shows the process of determining whether a particle can grow.


Whether a particle can continue growing
This particle stops growing
yes
If this particle touches the canvas borders
no
This particle stops growing
yes
If the particle touches surrounding particles
no
The particle grows by one unit and the algorithm continues checking other particles
Fig 3.8 Flow diagram of determining whether a particle can grow

The algorithm needs to ensure a simultaneous growth of all circular particles. It initially assigns particles the same radius, which corresponds to the minimum value in the largest opening sieve radius range. The algorithm then examines particles to determine whether they can grow. If a particle can be enlarged, the algorithm increases its radius by one unit. If it cannot grow, the algorithm skips it, no longer checks it, and continues examining other particles. This particle-growing process is repeated until no particle can grow. The pseudocode is shown below.

```
function generate_circle ()
  for each circle in array Ptcl
      # Originally in the "Circle" object, every circle's grow variable is set as true
      # Two conditions below which set grow as false to stop particle growing
      if the grow of circle is true
        if the circle's edge extends to the borders
          grow = false # stop the particle from growing
       else
        for each circle in array Ptcl
         # For particles surround this checked particle
          if the particles around this circle
            generate dis which is the distance between the checked circle and its adjacent circles
              if dis is smaller than the sum of the radii of these two circles
                grow = false # stop the particle from growing
```

Fig 3.9 (a) shows the point of Poisson Disk distribution generated in the previous step, and Fig 3.9 (b) shows an assembly of circular particles with Poisson Disk distribution. The parameters are: the canvas size was 800unit×800unit; the minimum distance was 80 units; the original radius of the circles was 30 units.


(a) (b)
Fig 3.9 Points bases on Poisson Disk distribution (a) and circular particle assembly (b)

3.3.3 Generation of 3D Spherical Particles Based on Poisson Disk Distribution

Building 3D spherical particle assembly is similar to creating 2D circular particles. The main differences are generating points with Poisson Disk distribution in 3D and the spherical shape of particles. Fig 3.10 shows the process and the modified parts of 3D particle packing generation.

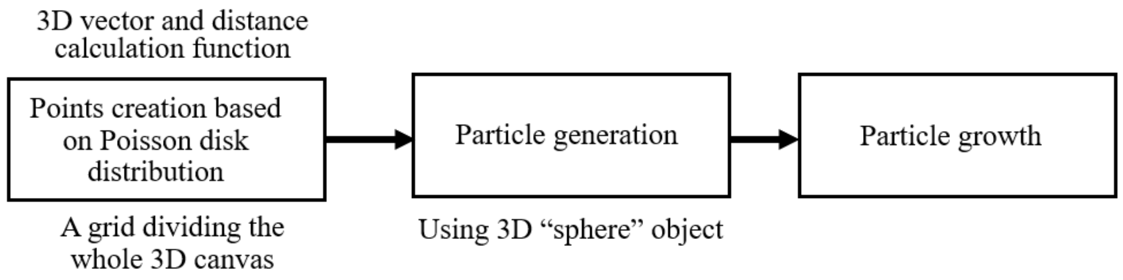

3D vector and distance calculation function
Points creation based on Poisson disk distribution
Particle generation
Particle growth
A grid dividing the whole 3D canvas
Using 3D "sphere" object

Fig 3.10 3D model building process and the modified parts in every step compared to 2D model building

3.3.3.1 Generation of Points in 3D Based on Poisson Disk Distribution

The method to generate points based on Poisson Disk distribution in 3D is similar to the approach proposed by Bridson (2007). The process is divided into three steps:

a) Build a grid covering the whole 3D canvas,
b) Randomly choose a point inside the canvas as the first point,
c) Create new points by randomly generating points inside the shell between two 3D concentric spheres with the existing point as the center and the radii ranging from *r* to *2r*, shown in Fig 3.11, and then check whether those newly generated points are far enough from other points.

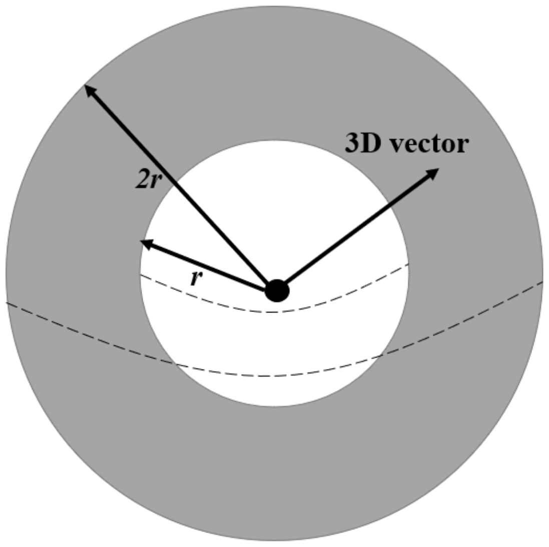


Fig 3.11 The shell between two 3D concentric spheres where the algorithm generates points

If the distances between a new point and its neighboring particles are greater than the minimum distance between points, a 3D vector, represented by the arrow shown in Fig 3.11, is assigned to the cell where the new point is located. This vector is used in the next step to randomly generate 30 points around this point to explore void places in the surrounding area where new points can be inserted.

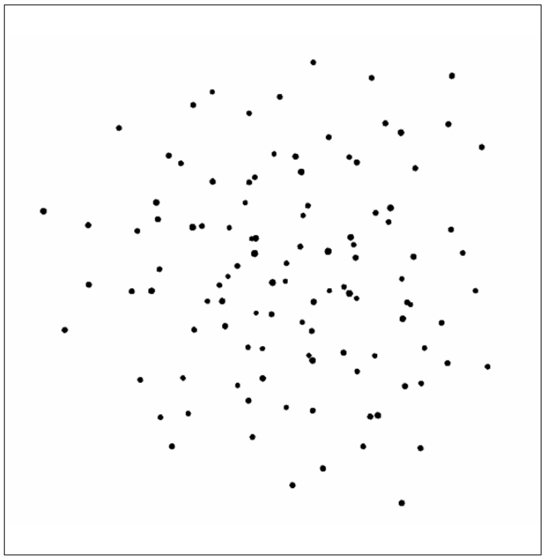


Fig 3.12 Points based on Poisson Disk distribution in 3D

Fig 3.12 shows the points with Poisson Disk distribution in 3D. The parameters used in this case are as follows: were that the 3D canvas size was 600unit×600unit×600unit; the minimum distance between points was 30 units. The picture may not clearly present points distributed in 3D, but the particle packing generated in the next step better illustrates the three-dimensional nature of the result.

3.3.3.2 Generation of 3D Spherical Particles

The algorithm in this step starts by generating spherical particles based on points of Poisson Disk distribution. It consists of two steps: building a “Sphere” object and generating particles using this object.

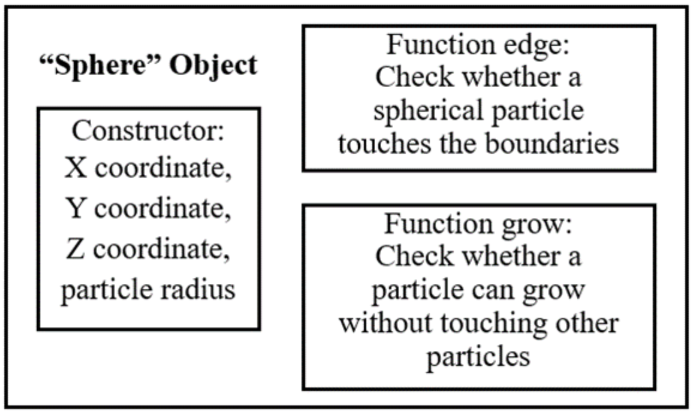

"Sphere" Object
Constructor:
X coordinate,
Y coordinate,
Z coordinate,
particle radius
Function edge:
Check whether a spherical particle touches the boundaries
Function grow:
Check whether a particle can grow without touching other particles

Fig 3.13 "Sphere" object used to generate 3D particles

A "Sphere" object is shown in Fig 3.13. It contains three parameters:

- the constructor, including the particle position and the particle radius;
- the "grow" function, which determines if there is enough space for this particle to grow;
- the "edge" function, which can stop particle growing if it touches the canvas boundaries.

A "Sphere" object does not have a "color" variable and a "show" function like the "Circle" object because ParaView is used to visualize 3D particle assemblies, and parameters to display the model can be manually set in the software.

Based on the object built in the first step, the algorithm starts generating spherical particles. The method of particle growing is the same as in the 2D model. The algorithm determines whether a particle can grow using the "edge" and "grow" functions. If it can, its radius is increased by one unit. If it cannot, the algorithm skips it and no longer checks it, and the generation of this particle is completed. This process is repeated until no further particle can grow.

Fig 3.14 (a) shows points generated based on Poisson Disk distribution in 3D, and the spherical particle assembly created using the previously generated points is shown in Fig 3.14 (b). The original radius of spheres was set to 30 units, and the remaining parameters were the same as the point generation example shown in the former section.

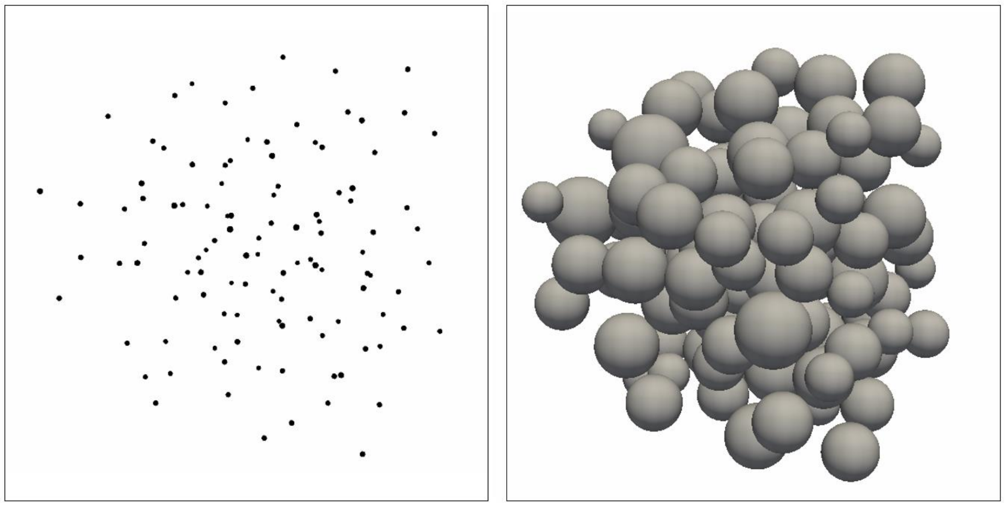

(a) (b)

Fig 3.14 Points of Poisson Disk distribution in 3D (a) and 3D spherical particle packing (b)

3.3.4 Volume Adjustment

The volume of particles with Poisson Disk distribution should be adjusted to make it within the acceptable range. Hogg (2008) discussed issues in interpreting particle size data and different measurement procedures to classify soil particles. He required the relative error of particle-size distributions to be less than 5%. In this thesis, the acceptable range of errors is set to 2%, which means that particle volumes reaching the target but not surpassing it by more than 2% are regarded as acceptable.

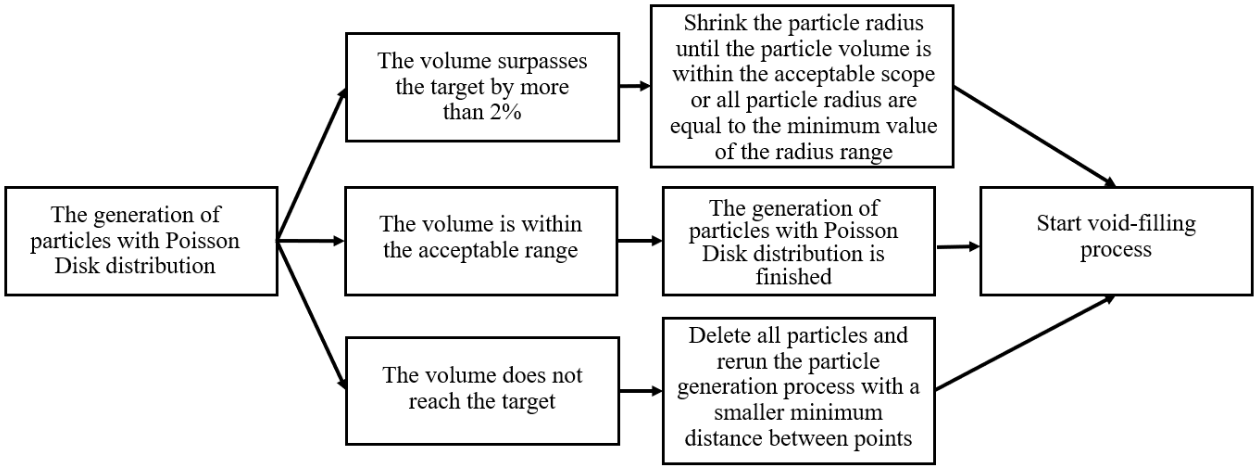
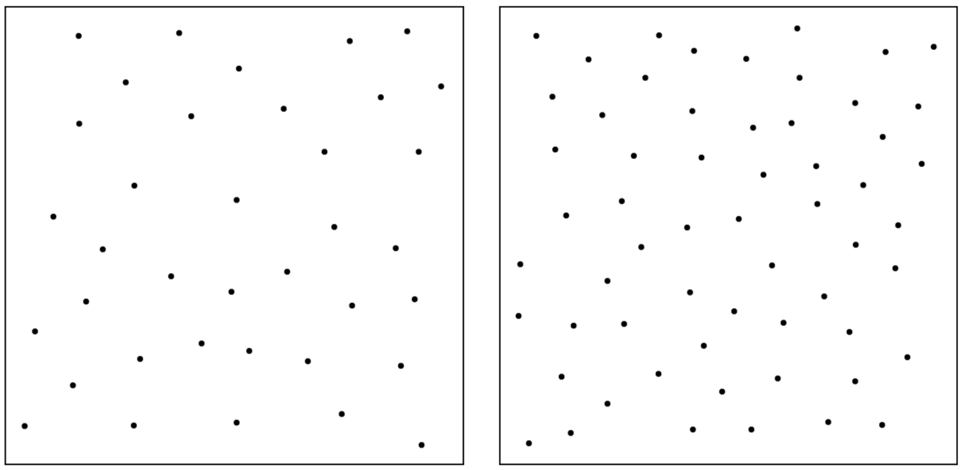

The generation of particles with Poisson Disk distribution
The volume surpasses the target by more than 2%
Shrink the particle radius until the particle volume is within the acceptable scope or all particle radius are equal to the minimum value of the radius range
The volume is within the acceptable range
The generation of particles with Poisson Disk distribution is finished
The volume does not reach the target
Delete all particles and rerun the particle generation process with a smaller minimum distance between points
Start void-filling process

Fig 3.15 The process of particle volume adjustment

The algorithm initially sets the minimum distance between points of Poisson Disk distribution as half of the canvas length and then gradually decreases it until the error is within the acceptable range. If the volume does not reach the target, the algorithm needs to add more particles belonging to the sieve in question. To do so, it deletes all already-generated particles and reruns the point generation process to create particles with a smaller minimum distance between points of Poisson Disk distribution until the error is within the acceptable range. If the particle volume exceeds the target by more than 2%, the algorithm shrinks the particle radii until the volume is within the acceptable limit or all particle radii are equal to the minimum value of the radius range. If the particle volume is within the limit, the algorithm continues to start the void filling process, which is presented in the next section. Fig 3.15 shows the particle volume adjustment process.

The algorithm uses the minimum distance between points of Poisson Disk distribution to adjust the number of points. The smaller the minimum distance between points, the more points are generated based on Poisson Disk distribution. For example, in Fig 3.16, the canvas size was 600unit×600unit, and the minimum distance between points was reduced from 100 units to 80 units. Correspondingly, the number of particles increased from 37 points to 60 points.



(a) $r = 100$ units 37 points (b) $r = 80$ units 60 points

Fig 3.16 The points of Poisson Disk distribution with different minimum distances (r is the minimum distance)

3.4 Void-filling

After creating particles based on the Poisson Disk distribution, the algorithm starts filling voids with particles belonging to the remaining sieves. It pinpoints void spaces in the canvas by checking the positions of non-filled cells and randomly selecting a point inside them. Suppose this point is not inside an existing particle and also far enough from the surrounding particles. In that case, it is identified as valid, and the algorithm inserts a particle based on this point. If it is already inside an existing particle or touches the canvas boundaries, this point is discarded, and the algorithm continues checking the next non-filled cell. Meanwhile, the algorithm calculates the total volume of particles. It stops inserting particles belonging to this sieve size and continues creating particles of the next smaller sieve size when the particle volume reaches the target.

This section first describes the method used to filter the non-filled cells. Only checking empty or partially covered cells can increase the probability of finding a valid point in the canvas and thus reduce the running time. It then presents the particle validation process to fill voids by inserting particles. The recursive call to decrease errors and optimize the algorithm is finally introduced.

3.4.1 Fully Filled Cells Checking

Inserting particles by only checking empty or partially covered cells can help accurately locate the void spaces in the canvas to reduce the running time. The algorithm first finds cells that are fully occupied by existing particles and then ignores these cells in the subsequent particle insertion round.

The method to determine if an existing particle fully covers a cell consists in checking whether all corners of this cell satisfy Equation 3-4.

$$d_c < r_p + r_{min} \tag{3-4}$$

where d_c is the distance between one corner of the cell and the particle center, r_p is the particle radius, and the r_{min} is the minimum value in the radius range for the next round of particle insertion.

For example, the algorithm finishes generating particles for US sieve No.100 (the particle radii range from 6 units to 9 units) and starts inserting particles for the next sieve, US sieve No.200 (the particle radii range from 3 units to 5 units). For a 2D particle with a radius of 7 units that belongs to US sieve No.100, the radius of the extended circular area is 10 units, which is 7 units, i.e., the radius of the existing particle, plus 3 units corresponding to the minimum radius of the subsequent particle insertion round for US sieve No.200.

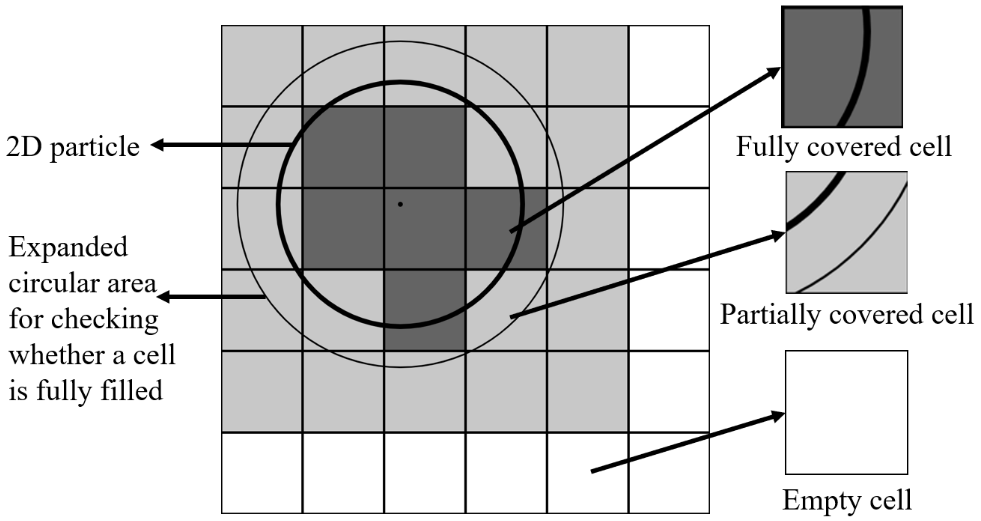

2D particle
Expanded circular area for checking whether a cell is fully filled
Fully covered cell
Partially covered cell
Empty cell

Fig 3.17 Extended area to determine full covered, partially covered, and empty cells

Fig 3.17 illustrates the method of determining whether a cell is fully covered. The gray cells, which are partially covered by the expanded circular area, and the white cells, which are entirely empty, are regarded as non-filled cells, which means the algorithm may insert a valid particle of the next sieve size inside them. Cells inside the extended circular area, which are the black cells, are marked as fully filled cells and are not checked when the algorithm inserts particles for the next sieve. After running several generations of models, neglecting fully-covered cells before the algorithm generates particles for the next sieve can roughly cut the running time in half.

3.4.2 Particle Validation

The particle validation is determined by whether the algorithm can generate a particle within the radius range of the current round of particle insertion based on a randomly selected point in a non-filled cell.

For every randomly selected point, the algorithm calculates two parameters:

- d_b: the minimum distance between this point and the canvas borders;
- d_a: the minimum distance between this point and other adjacent particles.

Based on Equation 3-5, the algorithm decides whether a randomly selected point is valid. If it is valid, the particle radius is then determined, and the new particle is inserted into the canvas. If this point is not valid, the algorithm ignores it and continues inserting particles in other non-filled cells. The algorithm needs four different scenarios to carry out the particle validation. If a valid point is found, the method to determine its corresponding radius is also presented.

If d_b and d_a are in the range of radii of the sieve, shown by Equation 3-5a, the algorithm marks this point as valid and generates a particle with the radius corresponding to the minimum value of d_b and d_a.

$$max_r > d_b , d_a > min_r \tag{3-5a}$$

where max_r and min_r are the maximum and minimum values of the radius range calculated based on the sieve opening size.

If d_b is greater than the maximum value of the radius range and d_a is within the range, presented by Equation 3-5b, the algorithm treats this point as valid and selects d_a as the particle radius.

$$d_b > max_r > d_a > min_r \tag{3-5b}$$

Suppose d_b and d_a are both greater than the maximum value of the range of radii, shown by Equation 3-5c. In that case, the algorithm treats this point as valid and creates a particle with the radius as the minimum value of d_b and d_a.

$$d_b , d_a > max_r \tag{3-5c}$$

Suppose either d_b or d_a is smaller than the minimum value of the range of radii, shown by Equation 3-5d. In that case, the algorithm labels this point as invalid because it is either too close to the borders or touching adjacent particles.

$$d_b < min_r \;\; or \;\; d_a < min_r \tag{3-5d}$$

For example, the algorithm randomly selects point *A* in a non-filled cell, as shown in Fig 3.18, and then calculates d_a, which is the minimum distance between point *A* and four adjacent particles, and d_b, which corresponds to the minimum distance between point *A* and the closest border. If both values are within the radius range, the algorithm generates a circular particle with radius d_a because d_a is smaller than d_b.

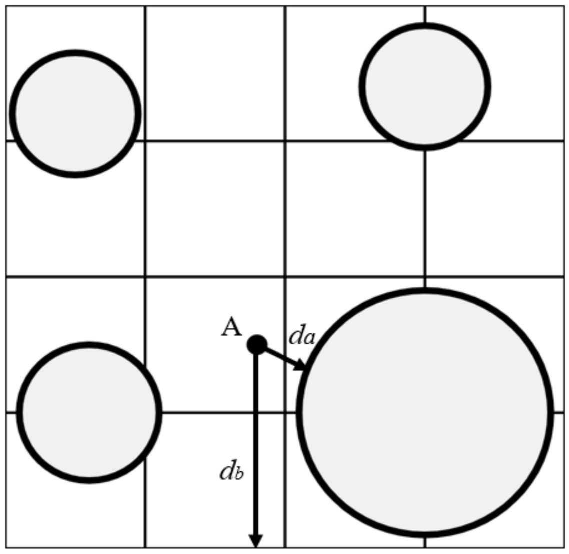


Fig 3.18 Point validation example

The diagram presenting the void-filling process is shown in Fig 3.19.

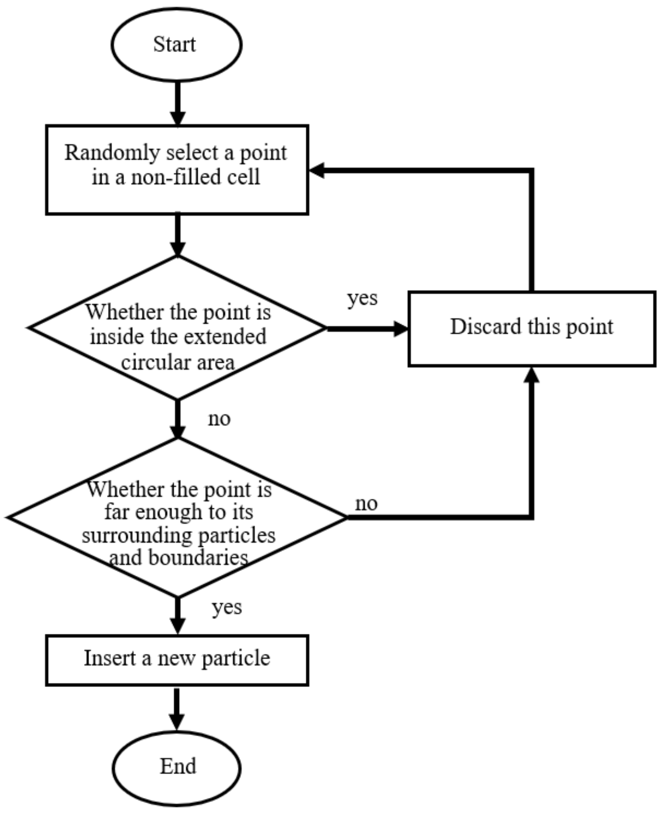

Start
Randomly select a point in a non-filled cell
Whether the point is inside the extended circular area
yes
Discard this point
no
Whether the point is far enough to its surrounding particles and boundaries
no
yes
Insert a new particle
End

Fig 3.19 Flow diagram of filling the voids

The pseudocode below presents the point validation process.

```
Function particle_validation (non-filled cell number, radius range)
  *valid* = true # create a "valid" boolean variable to mark whether a point is valid
  Randomly select a cell in the array of non-filled cells
  # Get the $X_c$, $Y_c$, $Z_c$ coordinates of the cell
  $l_c$ = the side length of the cell
  # Randomly select a point inside this non-filled cell
  $x$ = random ($X_c$, $X_c \times l_c$)
  $y$ = random ($Y_c$, $Y_c \times l_c$)
  $z$ = random ($Z_c$, $Z_c \times l_c$)
  # Create an array of the distances between the point and boundaries
  $bdy = [x, y, z, (wd - x), (ht - y), (dp - z)]$
  # Get the minimum distance in the *boundary* array
  # *min* is the function that outputs the minimum value of an array
  **if** *min(bdy)* < the minimum value of the radius range
    # If the minimum distance is not in the radius range, the point will be marked invalid
    *valid* = false
  # Create an array of the distances between the point and the adjacent particles
  *adj* = distances between point and other adjacent particles
  **if** *min(adj)* < the minimum value of the radius range
```

```
        # If the minimum distance is not in the radius range, the point will be marked invalid
        valid = false
    # If valid is true, the algorithm creates a new particle with the radius based on Equation 3-5
    if valid is true
        if min(bdy) < min(adj)
            if min(bdy) <= the maximum value of radius range:
                create a particle with radius min(bdy)
            else
                create a particle and set the maximum value of radius range as the particle radius
        if min(bdy) > min(adj)
            if min(adj) <= maximum value of radius range
                create a particle with a radius min(adj)
            else
                create a particle and set the maximum value of radius range as the particle radius
```

Fig 3.20 shows the 2D particle packing after the void-filling process. The black circular particles were inserted during the void-filling process. The canvas size was set to 800unit×800unit, and the minimum distance *r* was 80 units. The initial radius of the minimum newly generated particle was 10 units, which was smaller than the initial radius (30 units) of the particles generated with Poisson Disk distribution in the previous step. The assembly of 2D particles became more tightly packed after the void-filling process.

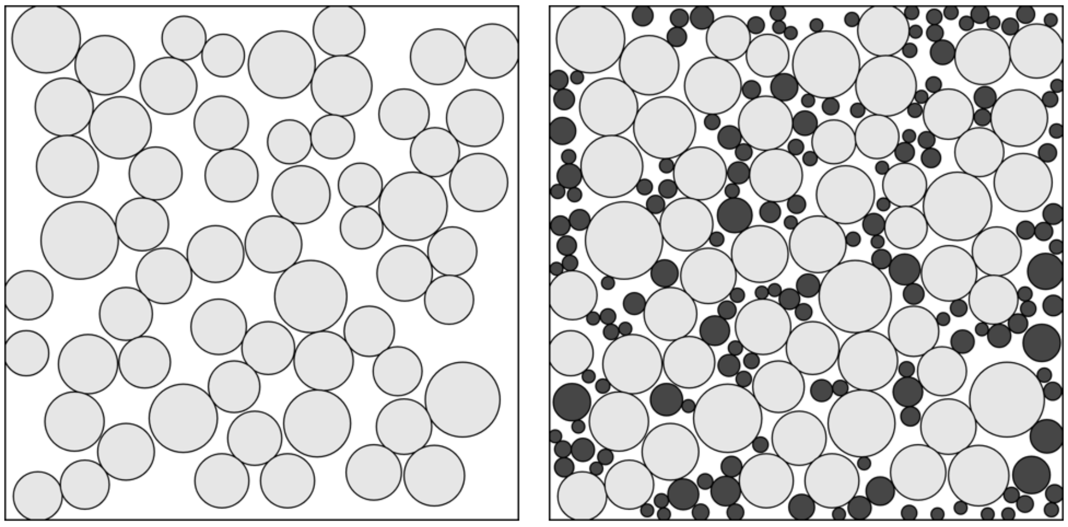

(a) (b)

Fig 3.20 The 2D particle assembly (a) after filling the voids compared with original particle packing (b)

Fig 3.21 shows the 3D particle packing that becomes more packed after the void filling process. The black particles were inserted during the void filling process. The parameters, in this case, are as follows: the canvas size was set to 600unit×600unit×600unit; the minimum distance between points generated with Poisson Disk distribution was 30 units; the initial radius of particles inserted in the void filling process was 10 units.

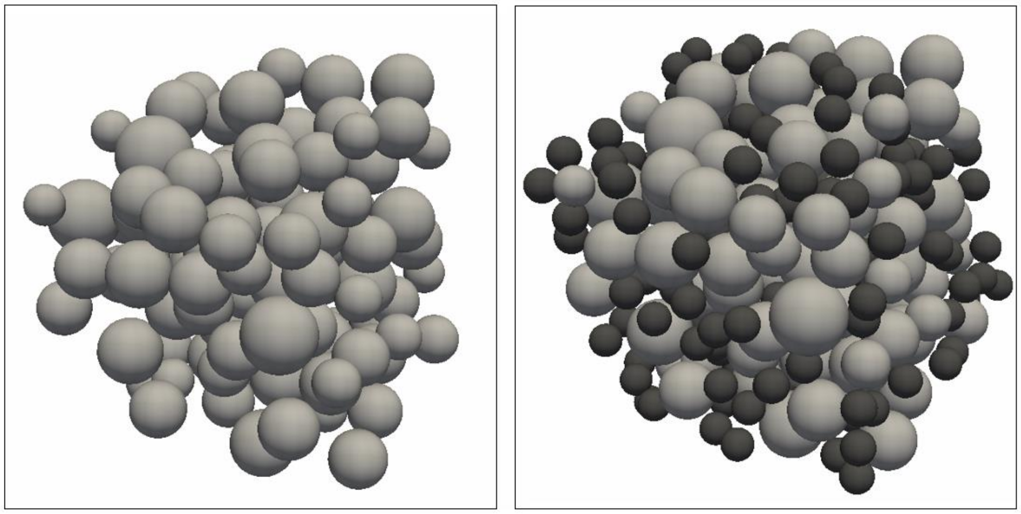


Fig 3.21 The original particle packing (a) and the model after the void-filling process (b)

3.4.3 Recursive Call

The algorithm inserts particles sieve-by-sieve and concurrently calculates the particle volume in each sieve. If the volume of particles created during the void-infilling process for one sieve reaches the target, the algorithm stops inserting particles for this sieve and moves to the next one. It starts a recursive call to rerun the particle insertion function when either of the two following scenarios:

When the algorithm creates particles for large-opening sieves, the following situation may occur: the volume of particles in the sieve reaches the target but exceeds it by more than 2%. The main reason is that large particles have significant individual volumes, and after the last particle in this sieve is generated, the total volume may significantly exceed the target. This situation causes a discrepancy between the simulation model and actual soil data. Thus, a recursive call is introduced to reduce the error within an acceptable range, which means all particles generated for this sieve are eliminated, and the particle insertion process reruns with a shrunken radius range to decrease the individual volumes of the largest particles and the total volume.

For example, for the initial radius range based on Table 3.1, the unit size is 0.0125 mm. For US sieve No.20, which contains particles with a diameter from 2.00mm to 0.68mm (160 units to 68 units), the initial radius range is defined from 80 units to 34 units. If the total volume of this particle insertion round surpasses the target by more than 2%, the algorithm deletes newly inserted particles in this sieve and reruns the void-filling progress with a smaller radius range, reducing the maximum value of a radius range by one unit, which is from 79 units to 34 units. If the overshoot still exists, the algorithm further shrinks the range (from 78 units to 34 units) and reruns the particle insertion until the total value is within the acceptable range or all particle radii are set to the minimum value of the radius range.

Fig 3.22 shows the process used to determine whether the algorithm needs a recursive call. It should be noted that if all radii of particles in this sieve are reduced to the minimum value in the radius range, the algorithm moves to insert particles for the next sieve because it cannot further shrink the radius range.

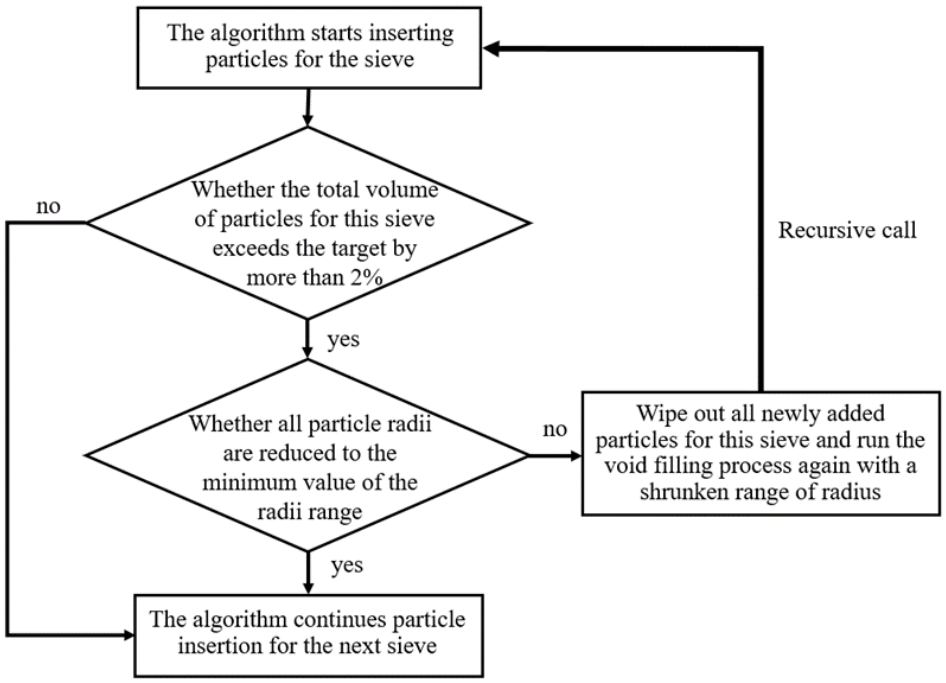

The algorithm starts inserting particles for the sieve
Whether the total volume of particles for this sieve exceeds the target by more than 2%
no
yes
Whether all particle radii are reduced to the minimum value of the radii range
no
yes
Wipe out all newly added particles for this sieve and run the void filling process again with a shrunken range of radius
Recursive call
The algorithm continues particle insertion for the next sieve

Fig 3.22 Recursive call used in the particle generation of large opening sieves

Another problem that may occur is when the algorithm inserts particles for small opening sieves. This condition might be the opposite of the previous scenario. The void-filling process checks all non-filled cells to insert particles of this sieve, but the total particle volume falls short of the target. In this case, the algorithm executes a recursive call to rerun the particle insertion function to generate more particles. For the first time, in a non-filled cell, the algorithm randomly selects a point and then checked it as invalid. However, it may choose another valid point in the same cell in the following void-filling process.

For example, the algorithm randomly selects a non-filled cell, such as the gray cell shown in Fig 3.23(a), and all points in that cell may be chosen as the potential center of a circular particle, as shown in Fig 3.23(b). Still, only the points shown in Fig 3.23(c) are valid because they are far enough from adjacent particles and boundaries. If a point in a non-filled cell is randomly selected and then is checked as invalid in the first round of particle insertion, the algorithm marks this cell as “checked but no particle inserted”. Nevertheless, if the algorithm reruns the particle insertion function and revisits this cell, a valid point may be selected, and the algorithm successfully inserts a particle.

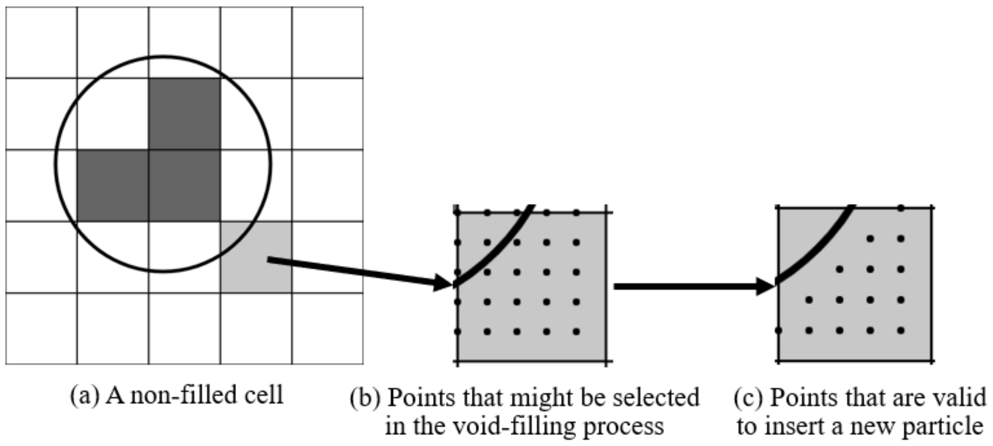

(a) A non-filled cell
(b) Points that might be selected in the void-filling process
(c) Points that are valid to insert a new particle

Fig 3.23 Rerunning the particle insertion function to generate more particles

Suppose the algorithm finds that the void-filling process is completed after running out of non-filled cells instead of reaching the targeted total volume. In that case, the void-filling process reiterates until the total volume reaches the target. The diagram is shown in Fig 3.24.

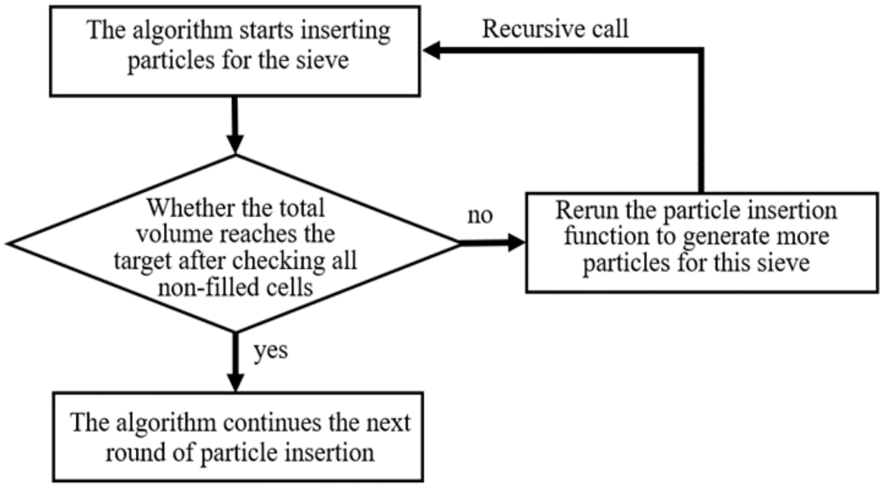

The algorithm starts inserting particles for the sieve
Recursive call
Whether the total volume reaches the target after checking all non-filled cells
no
Rerun the particle insertion function to generate more particles for this sieve
yes
The algorithm continues the next round of particle insertion

Fig 3.24 Recursive call used in the particle generation of small opening-size sieves

The pseudocode below describes the particle insertion process and the conditions when a recursive call is used. It starts with filtering fully occupied cells. The void-filling process then starts to insert particles by randomly selecting points in the non-filled cells. The algorithm reruns the particle insertion process if the particle volume surpasses the target by more than 2% or if the total volume does not reach the target after all non-filled cells are checked.

```
Function particle_insertion ()
    **for** every particle
        # Find the non-filled cells
        Filter the fully covered cells and the algorithm does not check it when inserting particles
```

```
  while total volume < target volume and there are still non-filled cells that are not checked
      Randomly select a point inside a non-filled cell
      if the point is valid:
          Insert a particle
      else
          Mark the cell as "visited but no particle inserted"
# If the total volume surpasses the target by more than 2%
if total volume > ideal volume × 1.02:
    Eliminate the new particles the algorithm created for this sieve
    Rerun the "particle_insertion" function with a shrunken radius range
# If the algorithm checks all non-filled cells, but the total volume does not reach the target
if all non-filled cells checked:
    Rerun the function with the accumulated volume and revisit all the non-filled cells
```

3.5 Particle Deposition

After being imported into a DEM simulator, the particle assembly first collapses and densifies, thereby reducing the void ratio. The void ratio is indeed manifested when particles touch each other. Yade is used to simulate particle packing deposition.

This section first introduces the features of explicit DEM simulations and the model calculation process used by Yade. The model deposition using the "Gravity Deposition Module" in Yade is subsequently described. Finally, contact friction, which is the parameter used to represent inter-particle friction during the deposition simulation, is introduced to investigate the relationship between this parameter and the void ratio of the deposited model. The method of adjusting the contact friction in the Yade simulation to guarantee the 3D model deposition results in a void ratio that slightly decreases and is close to the target is finally presented.

3.5.1 DEM Calculation Process

In Yade (Donzé et al. 2009), when simulating the particle assembly gravity deposition for particles that may establish a new interaction, Yade executes the following steps: i) detecting collision between particles, ii) creating new interactions, and iii) determining their properties.

For interactions that already exist between particles, the strain is evaluated, stresses are computed based on the calculated strains, and forces are applied to particles in interaction, as shown in Fig 3.25 (Jérier et al. 2010).

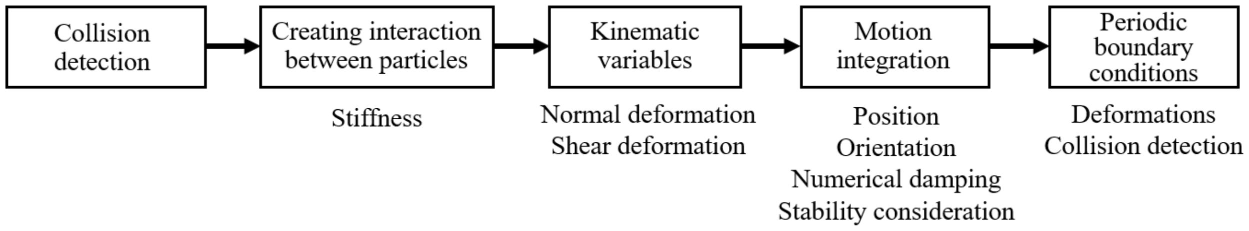


Fig 3.25 Yade calculation process of particle forces (Jérier et al. 2010)

The detailed calculation process includes:

a) Yade detects potential collisions between particles and establishes interactions,
b) Stiffness and shear stiffness are defined according to the basic DEM interaction rules,
c) The normal deformation and the shear deformation between particles are calculated,
d) Yade accumulates the generalized forces that every particle is subjected to in the contacts, and these generalized forces are then used to integrate motion equations for each particle separately,
e) Particles' boundary conditions are updated, and the unbalanced force ratio, which is the ratio of maximum contact force and maximum per-body force, is calculated.

If the unbalanced force ratio is smaller than 0.05, the particle assembly is regarded as stable, and the deposition simulation process is finalized. If it is not, the program continues finding potential interactions between particles and reruns the procedure until the unbalanced force ratio is lower than 0.05.

3.5.2 Gravity Deposition Module

The process through which particles touch each other and the entire particle packing densifies is called particle collapsing. It is performed by using Yade's Gravity Deposition Module. The module comprises three parts:

a) importing the algorithm-generated model into the Yade module,
b) simulating the particle assembly deposition,
c) outputting results, including the updated particle positions and forces between particles.

The diagram describing this process is shown in Fig 3.26.

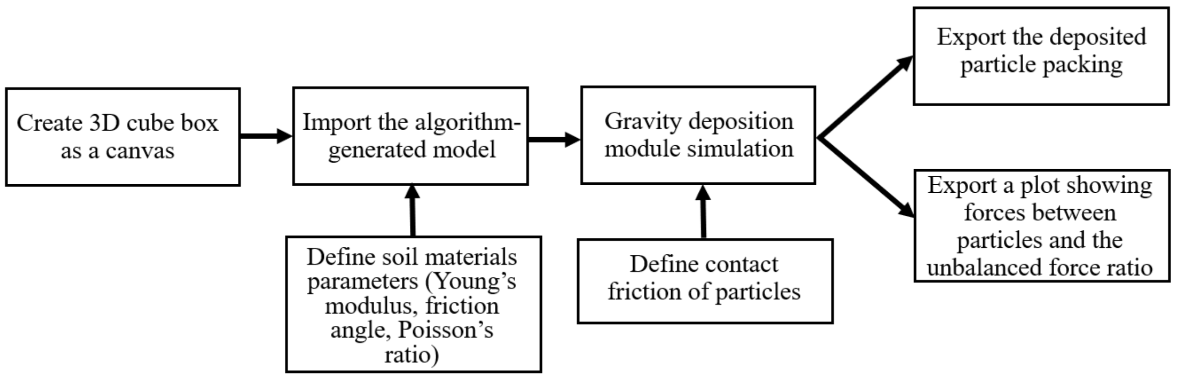
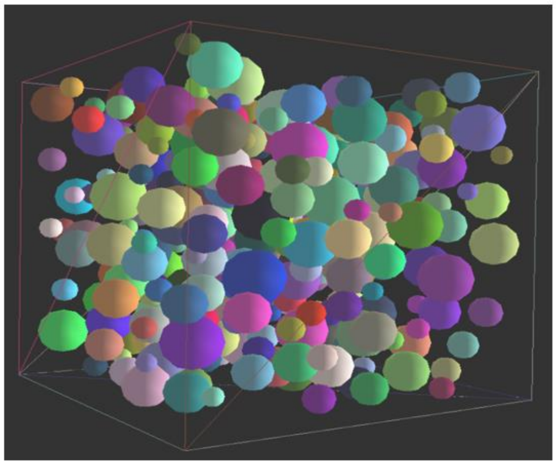

Create 3D cube box as a canvas
Import the algorithm-generated model
Gravity deposition module simulation
Export the deposited particle packing
Export a plot showing forces between particles and the unbalanced force ratio
Define soil materials parameters (Young's modulus, friction angle, Poisson's ratio)
Define contact friction of particles

Fig 3.26 Particle deposition process simulated by Yade

The particle assembly deposition simulation includes three steps:

The first step consists in generating a model. Yade first builds a 3D canvas using the *facet* function and then generates a particle packing using the *SpherePack* and *makeCloud* functions.


(a) 3D particle assembly

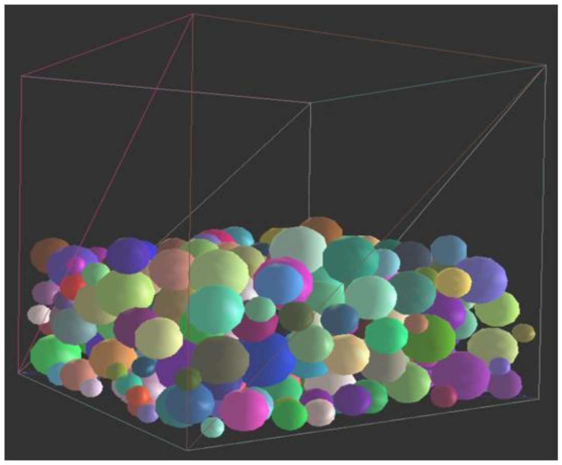

(b) 3D particle assembly after deposition

Fig 3.27 3D particle assembly deposition simulated by Yade

In the second step, the forces between particles are calculated, including contact plasticity, gravity, kinetic energy, plastic dissipation, and damping dissipation. All these forces are added up to calculate the total force acting on each particle. An integration method is introduced to compute the change in the position and the velocity of each particle during a specific time step from Newton's laws of motion. Then, the new particle positions are updated to compute the forces in the next step. This loop repeats until the unbalanced force ratio falls below 0.05, indicating the particle packing is stable. The original model and the model after the deposition process are shown in Fig 3.27.

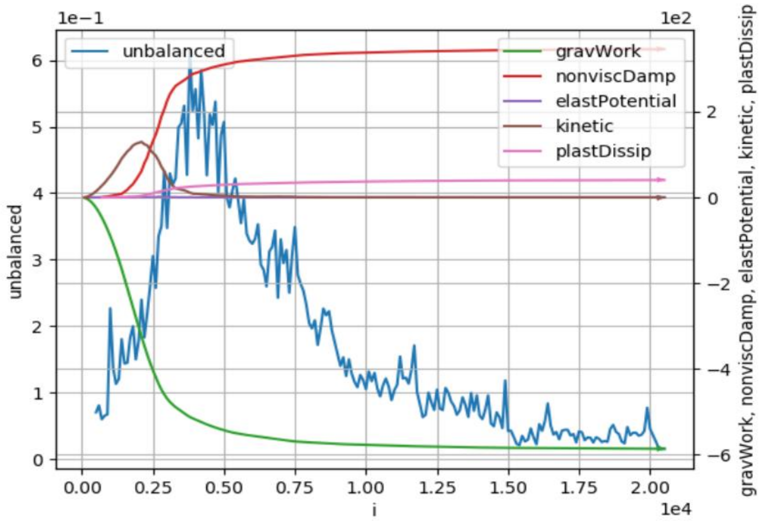


Fig 3.28 The plot of forces between particles provided by Yade

Finally, Yade exports the updated particle positions, calculates the deposited model's void ratio, and plots the result of all forces. Fig 3.28 shows the resulting plot, where the Y-axis represents all forces between particles and the X-axis is the time step.

The “Gravity Deposition Module” is designed specifically for 3D spherical particle models. For the 2D model, this thesis proposed using this model by assigning the same Z-axis value to all 2D particles and transferring them to spherical particles. During the 2D deposition simulation, the freedom of particles in the Z plane is restricted, meaning particles can only move in the X-Y plane. Fig 3.29 shows the 2D model placed in Yade and its deposition result.

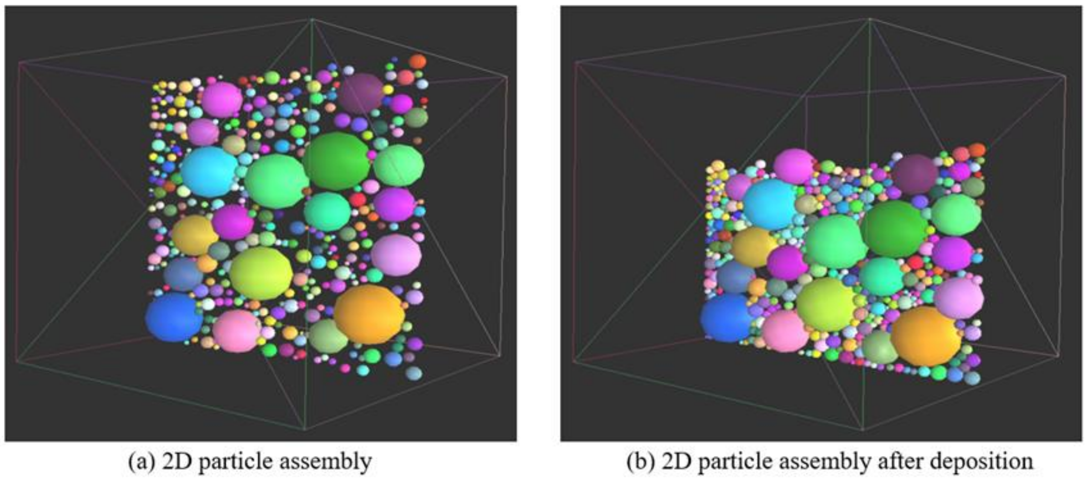


Fig 3.29 2D particle model deposition of Yade simulation

3.5.3 Contact Friction

Contact friction is a parameter that reflects the inter-particle friction in the model deposition. Contact friction, one of the micro properties of particle packings, is not equal to the overall friction angle of particles. But, to simplify the simulation process, Yade sets the minimum value of the friction angles of two particles equal to the contact friction. It also provides the *setContactFriction* function to adjust contact friction manually.

The stability of the particle packing is parametrically influenced by varying inter-particle friction. Suppose the particles during the deposition process are set to frictionless. In that case, the void ratio of the deposited particle packing decreases sharply compared to the original value because particles with frictionless surfaces can easily slide over each other. On the other hand, if contact friction is infinitely rough, the contact forces between particles become friction-driven, and the void ratio after the model deposition may remain unchanged or just decrease slightly. So, changing contact friction, a micro property between particles, can adjust the void ratio, a macro property, of particle packings.

The ideal model deposition results in a densified particle packing with particles touching each other. Meanwhile, the void ratio should slightly decrease while still remaining close to the target value.

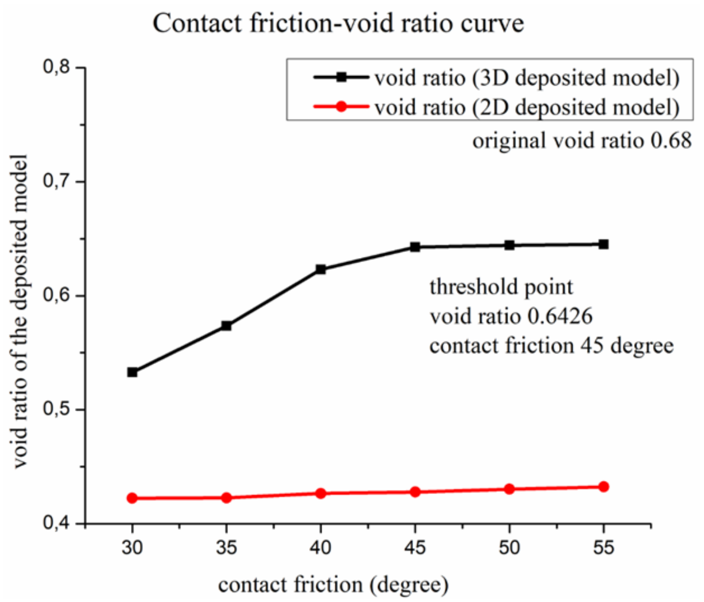


Fig 3.30 Curves of the relationship between contact friction and void ratio of deposited models

For example, Fig. 3.30 shows the relationship between the void ratio of the deposited model and contact friction obtained by changing the contact frictions of 2D and 3D particle packings. The curves illustrate that, for the 3D particle packing, the model's void ratio initially increased with increasing contact friction. However, once the contact friction exceeded 45º, this increasing trend slowed down, and the curve tended to be flat. Thus, a contact friction of 45º can be regarded as the threshold value. When the contact friction is greater than this value, the model deposition can be achieved with a slightly reduced void ratio. Therefore, setting the contact friction greater than 45º in the 3D model deposition is recommended for this case. The trend is similar to the conclusion of Thornton (2000). He found that the increasing value of interparticle friction induces an increase in the void ratio of spherical particles subjected to triaxial stress condition in 3D DEM simulation but the effect of contact friction is limited when it reaches some threshold value.

It should be noted that contact frictions set in the Yade program start from 30º to 55º. The reason is that based on Dai et al. (2016), interparticle friction angles that range from 0º to 15º are considered relatively low. Contact friction angles larger than 15º are regarded as relatively high. Larger contact friction should be set to minimum the void ratio reduction of the deposited model. In addition, after running several model position simulations, the study found that threshold values are normally within the range from 40º to 50º. The initial value of contact friction should be greater than 15º and outside the possible range of threshold values. So, in this study, contact frictions that are set in Yade software start from 30º. The study did not run deposition simulations with contact friction larger than 55º, since void ratios of deposited models did not change after contact friction was set larger than the threshold value.

However, contact friction does not significantly impact the void ratio of the 2D deposited model, and the void ratio also reduced sharply compared to the original value and the target. The potential reasons might include:

a) For 2D models, since the movements of 2D particles are limited in the X-Y plane, inter-particle contacts only exit in the 2D plane. But they occur in all directions for the 3D model. Coordination number, i.e., the average number of interactions per particle, can reflect this difference.

The coordination number *Z* is defined as:

$$Z = 2C/N \tag{3-6}$$

where *C* is the number of contacts and *N* is the number of particles.

The coordination number of a stable 3D particle assembly is generally greater than 4, while it is just around 2 for 2D particle packings (Rothenburg and Kruyt 2004).

Coordination number also indicates the stability of a particle assembly. A particle packing can be regarded as dense and stable, and particles are closely packed if the coordination number is higher than 4. Therefore, inter-particle friction can play a critical role in the density of particle assemblies and thus influences the void ratio (Masanobu 1977).

In addition, mechanical properties of compacted soils could be influenced by coordination number. Strong frictional resistance combined with a high coordination number would restrict the possible axis of rotation, eventually leading to rotational arrest or frustration (Santamarina 2003; Minabe et al. 2016). 2D particles can only move in the X-Y plane during the deposition process, and 2D particles are not as tightly packed as 3D particles. Thus, contact friction can barely influence the void ratio of the 2D model. Moreover, 3D particle packings with a high coordination number mean that the load that works on every particle, which is the gravity during the deposition for this case, can be distributed by neighboring particles. Thus, it is more stable than the 2D particle assembly with a low coordination number (Hagerty et al. 1993).

Yade provides the *avgNumInteractions* function to calculate the coordination numbers of particle assemblies. In the case presented above, the coordination number of the 3D deposited model was 4.22. By contrast, that of the 2D deposited model was only 1.84.

b) The particle arrangement, namely particle-size distributions and initial void ratios, also plays a significant role in the void ratio reduction after the model deposition. For example, Wang et al. (2018) studied the minimum void ratio of gap graded soil-rock mixtures (SRM) with varying particle breakage. They concluded that if the particles in a gap interval play the role of filling components, their absence will increase the void ratio of the SRM.

c) The 2D particle shape in the algorithm-generated model is circular. Rounded particles can rotate and slide over one another more easily than angular and ellipsoidal ones (Lin and Ng 1997; Ke and Takahashi 2012; Fei and Narsilio 2020). In addition, coordination numbers of rounded particles are usually lower than angular particles, indicating rounded particles are generally less packed (De Bono and McDowell 2015; Estrada et al. 2008). Therefore, circular particles easily fill voids during the deposition process, causing the void ratio to reduce dramatically.

d) There are many more opportunities for particle interlocking to form in a 3D particle assembly than in a 2D particle packing since particle contacts only exist in the X-Y plane in 2D models, but they occur in all directions in 3D models. Therefore, weak particle interlocking also results in the model's void ratio reduction after the deposition process.

For 3D models, the relationship between contact friction and the void ratio of the deposited model is similar to the curve shown in Fig 3.30, but the threshold value may vary. The void ratio first increases with the growth of contact friction and remains constant when the contact friction reaches some threshold value. The curve indicates the recommended contact friction to perform the model deposition with a void ratio that remains close to the original value. However, for 2D models, altering contact friction cannot achieve the same effect. The void ratio decreases considerably after the deposition process. The reasons include weak particle interlocking, low coordination number, initial arrangement of particles, and rounded shape of particles.

3.6 Conclusion

This chapter introduces the details of the algorithm developed to generate 2D and 3D models. The algorithm can generate particle assemblies sieve-by-sieve using Poisson Disk Sampling and Grid Sampling techniques based on user-defined particle-size distributions and void ratios.

The parameters the algorithm requires to build models are defined, and their recommended values are provided. The algorithm generates particles for the largest opening sieve based on Poisson Disk distribution and then fills void spaces by inserting particles belonging to the remaining sieves. The particle validation and recursion are used to reduce errors during the void-filling process and optimize the algorithm. Omitting the fully-covered cells prior to the particle insertion process can also reduce the running time. Finally, Yade is used to simulate the particle packing deposition. Adjusting the contact friction can help realize the 3D model deposition with a slightly reduced void ratio. However, contact friction barely influences the 2D model density, and the void ratio of the 2D model reduces significantly after the gravity deposition. Coordination number, initial particle arrangement, particle shape, and particle interlocking may influence the void ratio reduction.

Chapter 4 Testing and Validation

4.1 Introduction

This chapter provides examples showing how 2D and 3D models representing a wide variety of actual soils are generated using the proposed algorithm.

Section 4.2 contains six examples of real soil samples with different characteristics. The algorithm creates models aiming to produce well-graded pure sands and soil mixtures with narrow and wide ranges of particle radii. The algorithm then generates particle assemblies for poorly graded soils, including gap graded and uniformly graded soils. Building models simulating well-graded dense and loose soils also shows that the algorithm can generate particle packings with a broad range of void ratios. The particle packing deposition result is completed by Yade, and the relationship between the void ratio of the deposited model and contact friction is also presented. Since the algorithm-generated models may bring errors compared to real soils, an error analysis is presented in Section 4.3.

A sieve-by-sieve demonstration is used to illustrate the model-building process. In order to show the process of particle generation, the particle colors are set as different contrasting shades of grey to distinguish particles generated in different particle insertion rounds, where particles get darker as the insertion process proceeds.

4.2Verification of Algorithm Implementation

4.2.1 Pure Sand Example

The first example aims to recreate pure sands using the proposed algorithm. The actual soil data is based on Fragaszy and Sneider (1991). The algorithm creates 2D and 3D particle packings that simulate the fine portion of the soil sample. The particle-size distribution of the pure sand with a void ratio of 0.6 is shown in Table 4.1.

Table 4.1 Particle-size distribution of the pure sand sample (Fragaszy and Sneider 1991)

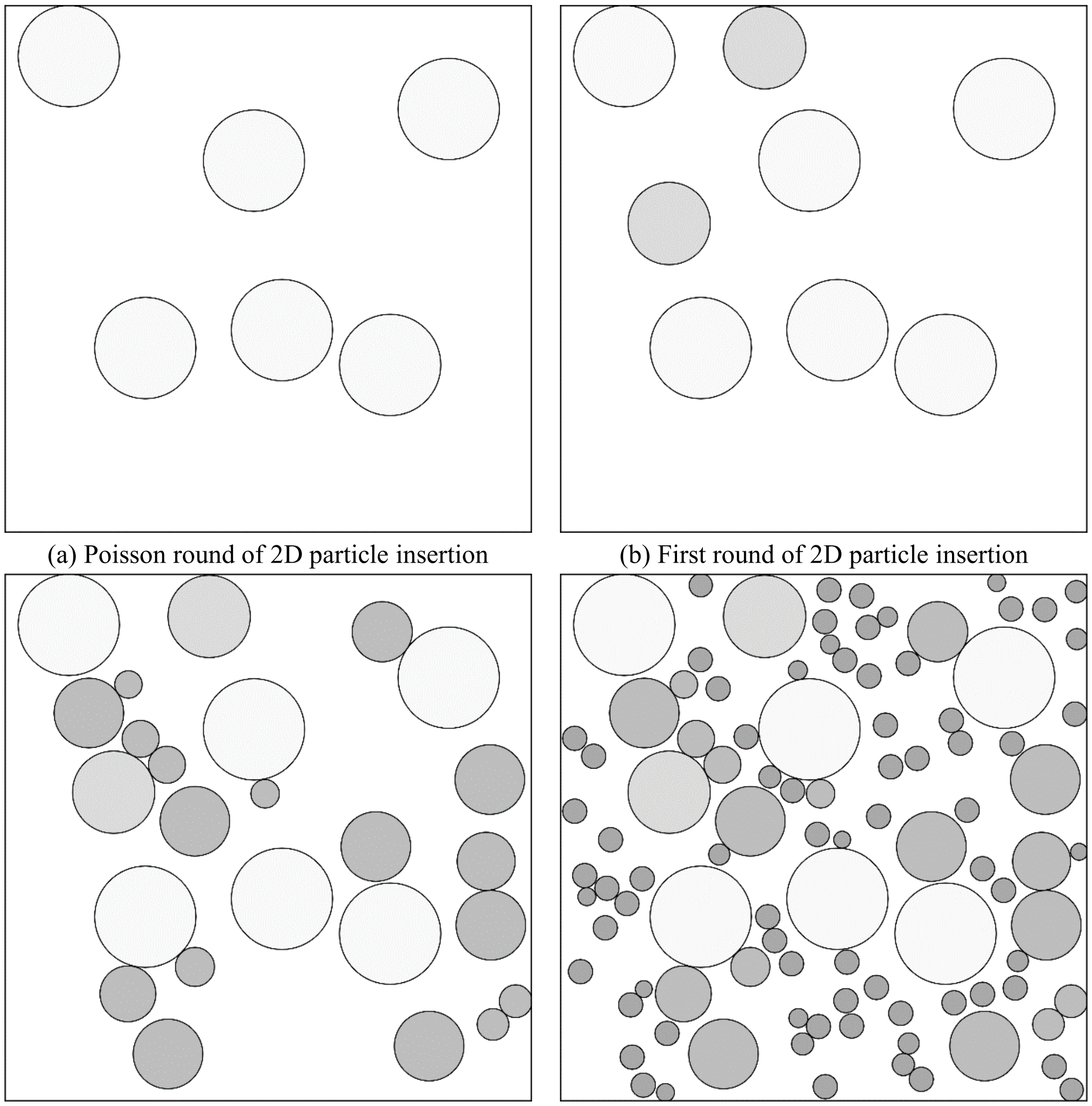
Sieve(*mm*)	4.75-2.36	2.36-1.7	1.7-0.6	0.6-0.4	0.4-0.3	0.3-0.15	0.15-0.075	<0.075
Radius(*unit*)	190-95	95-68	68-24	24-16	16-12	12-6	6-3	<3
Finer percentage	100%	76%	70%	47%	29%	18%	5%	2%
Mass percentage for each sieve	24%	6%	23%	18%	11%	13%	3%	2%

The model's unit size was set to 0.0125mm based on the soil sieve sizes. The 2D canvas size was 12.5mm×12.5mm/1000unit×1000unit, and the total volume of 2D particle assembly was $1000^2 \div 1.6 = 625000$ unit^2. The algorithm calculated the target volumes for each sieve using Equations 3-2 and 3-3, shown in Table 4.2.

Table 4.2 Finer percentage and target volumes of 2D particles for every sieve

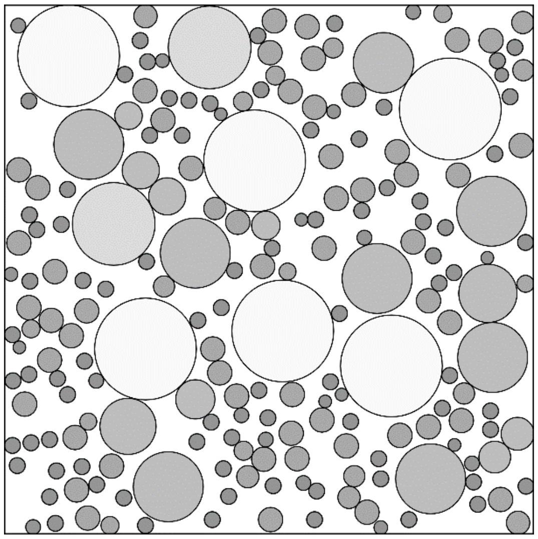
Radius(*unit*)	190-95	95-68	68-24	24-16	16-12	12-6	6-3	<3
Mass percentage for each sieve	24%	6%	23%	18%	11%	13%	3%	2%
Target volume ($\times 10^4$ *unit*^{2})	15	3.75	14.375	11.25	6.875	8.125	1.875	1.25

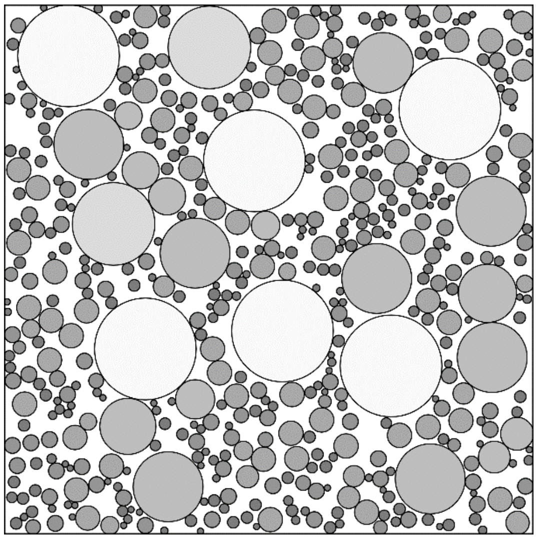
The cell size used in the void-filling process was 5unit×5unit. This cell size was small enough to target the tiny voids of the smallest sieve particles while ensuring the canvas was evenly covered. Fig 4.1 shows the 2D model building process sieve-by-sieve.

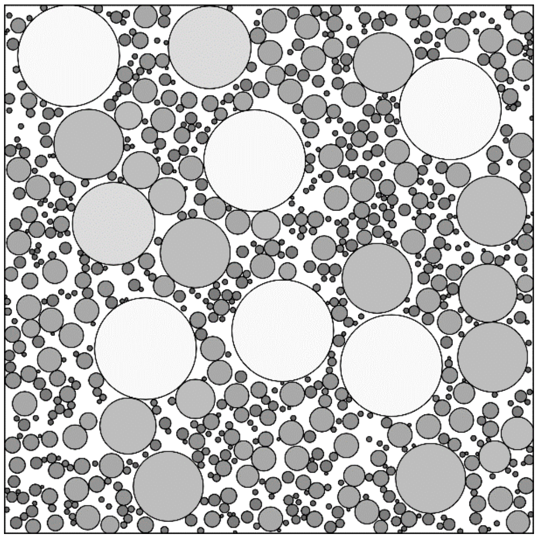

(a) Poisson round of 2D particle insertion (b) First round of 2D particle insertion

(c) Second round of 2D particle insertion (d) Third round of 2D particle insertion

Fig 4.1 2D model generation for pure sand


(e) Fourth round of 2D particle insertion


(f) Fifth round of 2D particle insertion


(g) Sixth round of 2D particle insertion

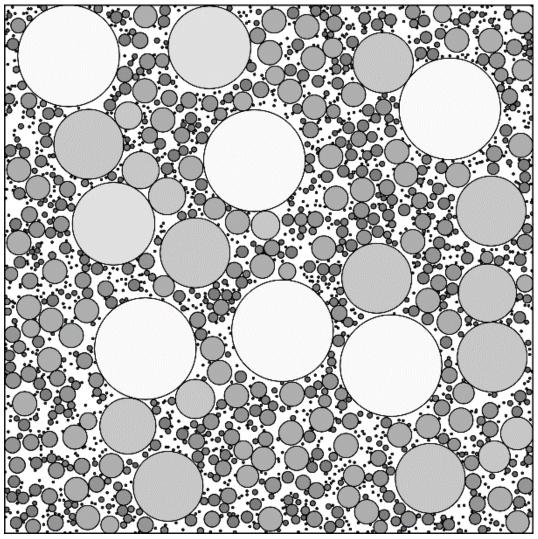

(h) Seventh round of 2D particle insertion

Fig 4.1 (Continued) 2D model generation for pure sand

Table 4.3 shows the results of the 2D particle packing. A total of 1790 circular particles were generated in the model, and the running time was 3.5 seconds. It must be noted that all computations and model deposition simulations in this thesis were completed on a ROG Strix G15 with an AMD Ryzen7 4800HS CPU @2.9 GHz and 16 GB of memory.

Table 4.3 Details of 2D particle packing

Round of particles insertion	Opening of sieves *mm*	Mass $g \times 10^{-3}$	Ideal mass $g \times 10^{-3}$	Finer percentage	Ideal finer percentage	Percent difference
Poisson	4.75-2.36	44.24	39.23	100%	100%	0%
1	2.36-1.7	9.67	9.56	73.40%	76%	2.60%
2	1.7-0.6	37.34	36.63	67.59%	70%	2.41%
3	0.6-0.4	28.70	28.67	45.17%	47%	1.83%
4	0.4-0.3	17.66	17.52	27.94%	29%	1.06%
5	0.3-0.15	20.72	20.71	17.33%	18%	0.67%
6	0.15-0.075	4.81	4.79	4.89%	5%	0.11%
7	<0.074	3.33	3.19	2%	2%	0%

The particle-size distributions of the algorithm-generated model and the actual soil are shown in Fig 4.2. The model's particle-size distribution curve closely matches that of the actual soil. The average error was 1.44%, while the maximum error occurred during the first round of particle insertion and had a value of 2.60%. The primary source of error is that the volume of particles belonging to the largest-opening sieve exceeded the target by more than 2%. Due to the considerable individual particle volume, the total volume exceeded the target after the algorithm created the last particle. However, without this particle, the volume would have fallen short of the target. This problem persisted when the algorithm used recursion to generate particles with a shrunken radius range, as described in Section 3.4.3. The errors for the particles generated for the remaining sieves were considered acceptable.

The void ratio of the particle assembly was 0.531, which is slightly lower than the target of 0.600. The main reason is that the volume of particles in the sieve with the largest opening size exceeded the target, causing the model's void ratio to be smaller than the target.

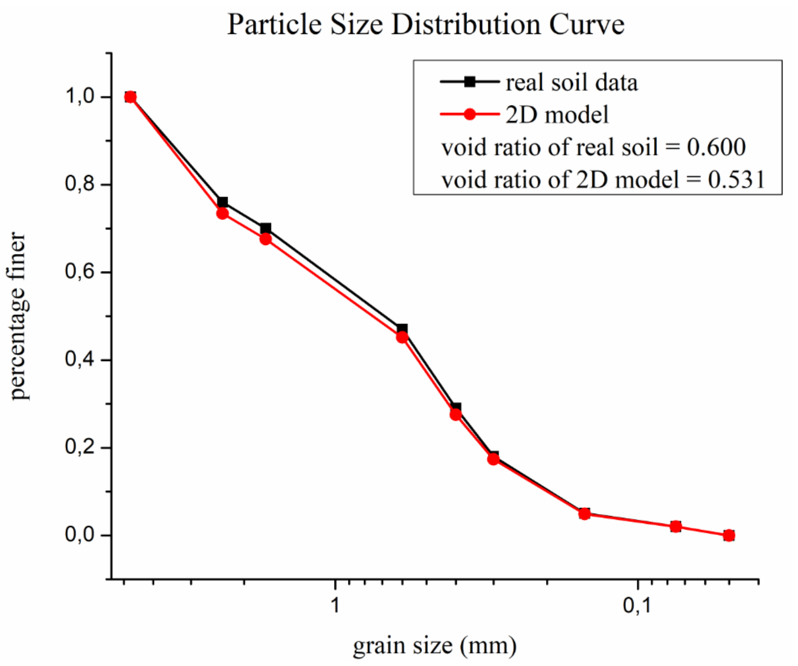


Fig 4.2 Comparison of particle-size distributions of the algorithm-generated 2D model and the real soil

The curve showing the relationship between contact friction and the void ratio of the 2D deposited model is shown in Fig 4.3. It indicates that contact friction had little influence on the void ratio of the 2D deposited model. The void ratio increased slightly when contact friction rose from 35º to 40º. The reason is that the particle packing can be regarded as stable and tightly packed if the coordination number is higher than 4 (Masanobu 1977), however, the coordination number of the 2D particle assembly was only 1.86, meaning that particles were loosely packed. Thus, inter-particle friction that takes place when particles are in contact with one another hardly impacts the deposition results.

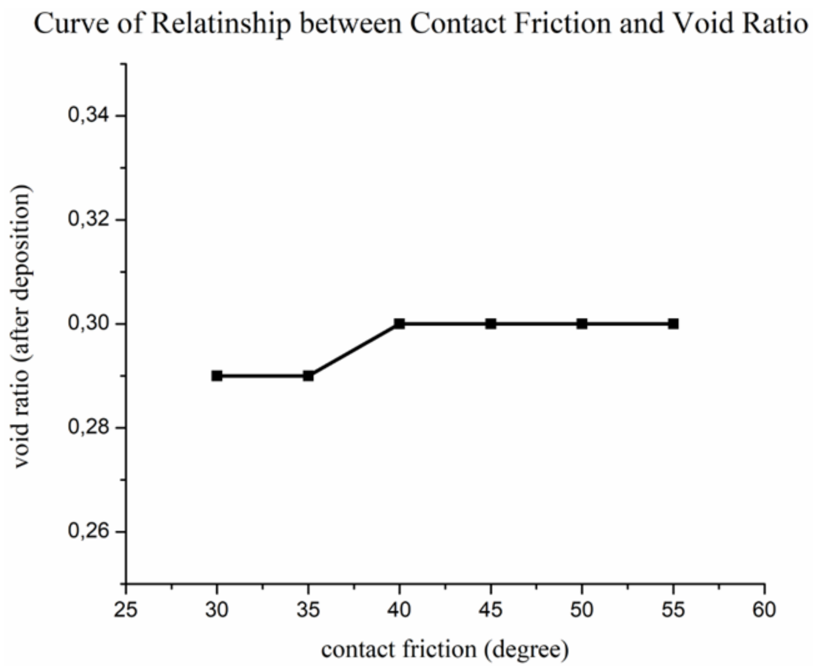
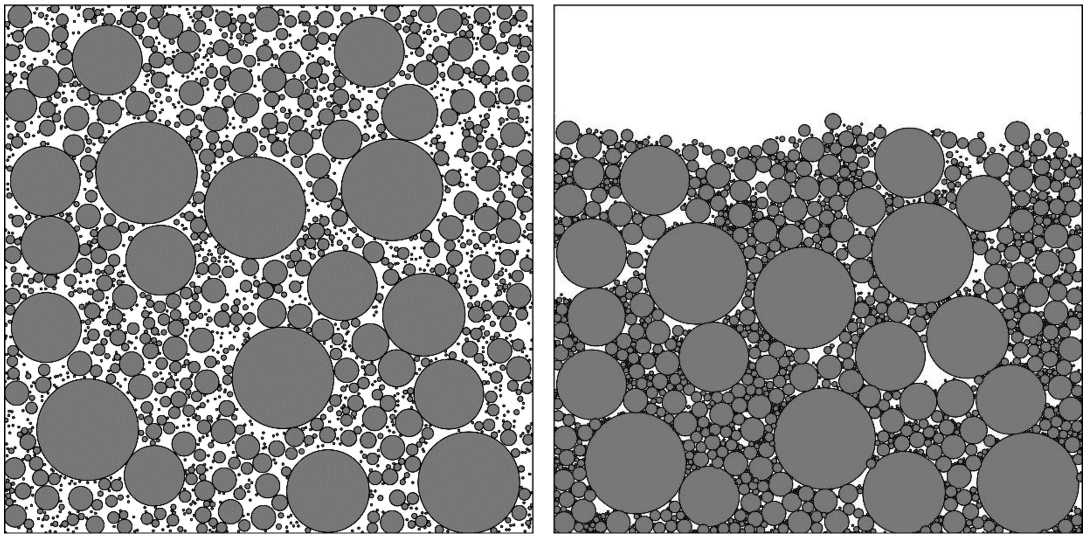


Fig 4.3 The relationship between contact friction and void ratio of 2D deposited model

Fig 4.4 shows the result of the particle assembly deposition. The model's volume decreased from 12.5mm×12.5mm to 12.5mm×10.3625mm, and accordingly, the post-deposition void ratio was 0.300 compared with 0.531 prior to deposition. The running time was 45 seconds.

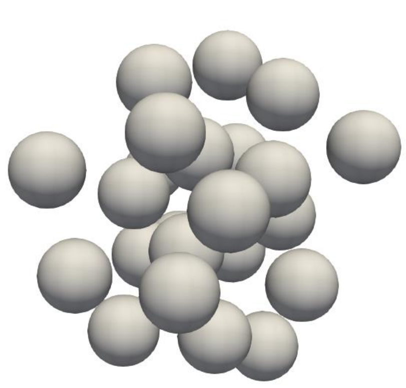

(a) (b)
Fig 4.4 The comparison of the original model (a) and the deposited model (b)

The void ratio significantly decreased after the model deposition. This is mainly attributed to the initial void ratio being 0.531, indicating there were many voids in the original model. Particles belonging to the last three sieves accounted for a high percentage of the total soil mass (17.33%), meaning that a large number of small particles slid into the voids between large particles during the deposition process, causing the void ratio to decrease significantly. In addition, the 2D model's coordination number was low, meaning that the 2D particles were loosely packed. As such, after the gravity densified, the particle packing densified, and the void ratio decreased significantly.

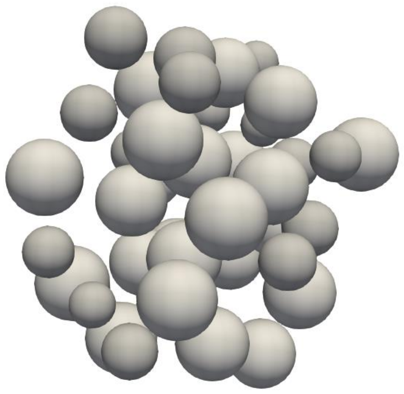
For the 3D model, the total volume was $1000^3 \div 1.6 = 6.25 \times 10^8$ unit^3, and the cell size was 5unit ×5unit×5unit. The remaining parameters were assigned the same values as in the 2D model. The finer percentage and target volumes for each sieve are shown in Table 4.4. The algorithm-generated 3D model is shown in Fig 4.5 sieve-by-sieve.

Table 4.4 Finer percentage and target volumes of 3D particles for every sieve

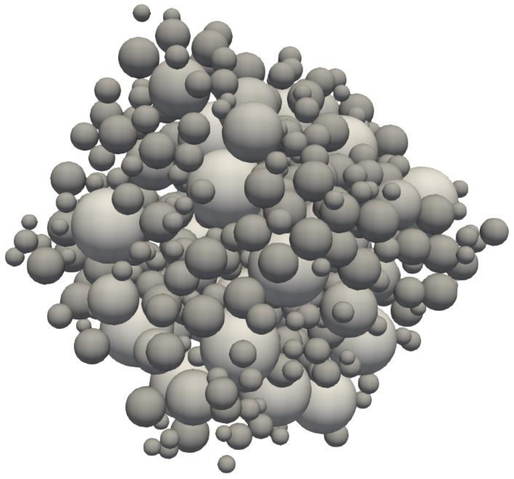
Radius (*unit*)	190-95	95-68	68-24	24-16	16-12	12-6	6-3	<3
Finer percentage	24%	6%	23%	18%	11%	13%	3%	2%
Target volume ($\times 10^8 unit^3$)	1.5	0.375	1.4375	1.125	0.6875	0.8125	0.1875	0.125



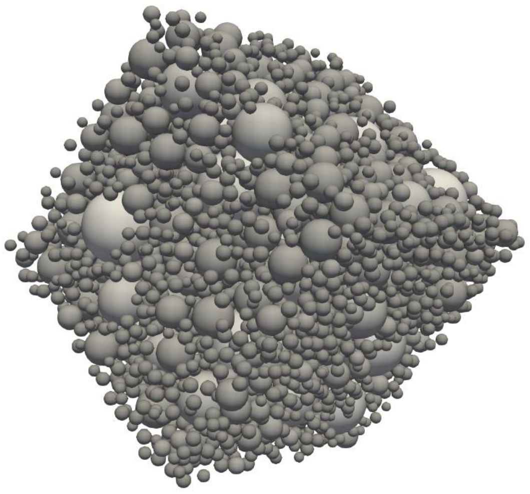
(a) Poisson round of 3D particle insertion



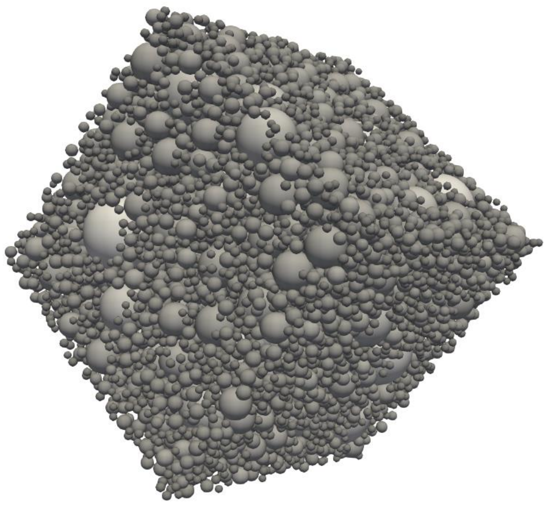
(b) First round of 3D particle insertion



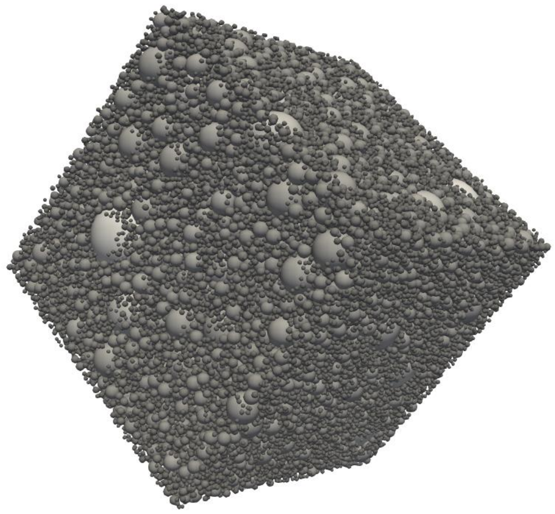
(b) Second round of 3D particle insertion



(c) Third round of 3D particle insertion

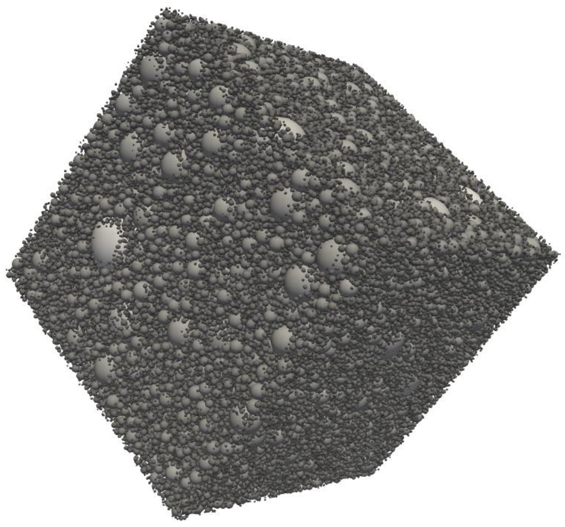


(d) Fourth round of 3D particle insertion



(e) Fifth round of 3D particle insertion

Fig 4.5 3D model generation for a pure sand


(f) Sixth round of 3D particle insertion

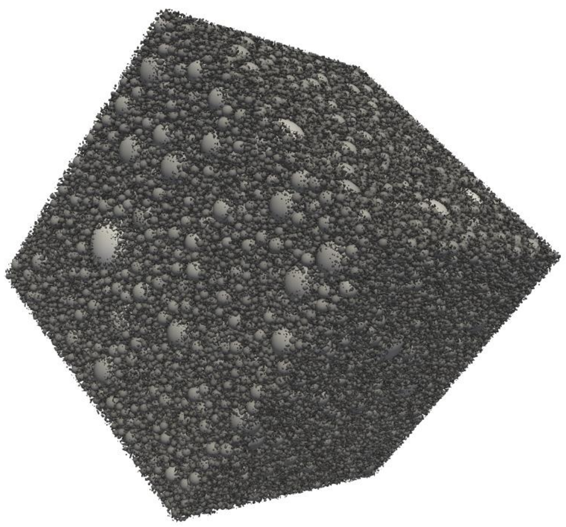

(g) Seventh round of 3D particle insertion

Fig 4.5 (Continued) 3D model generation for a pure sand3D

Table 4.5 shows the details of the 3D model. A total of 525,333 spherical particles were generated for the model, and the running time was 56,906 seconds. The particle generation for the smallest opening sieve was time-consuming, accounting for roughly 40% of the total running time.

Table 4.5 Details of 3D particle packing

Round of particles insertion	Opening of sieves *mm*	Mass $g\times10^{-2}$	Ideal mass $g\times10^{-2}$	Finer percentage	Target finer percentage	Percent difference
Poisson	4.75-2.36	49.25	46.82	100%	100%	0%
1	2.36-1.7	11.94	11.70	75.42%	76%	0.58%
2	1.7-0.6	45.79	44.87	69.46%	70%	0.54%
3	0.6-0.4	35.82	35.12	46.61%	47%	0.39%
4	0.4-0.3	21.89	21.46	28.73%	29%	0.27%
5	0.3-0.15	25.88	25.36	17.81%	18%	0.19%
6	0.15-0.075	5.86	5.85	4.89%	5%	0.11%
7	<0.074	3.96	3.90	2.00%	2%	0%

The particle-size distribution curves for the model generated by the algorithm and the actual soil are shown in Fig 4.6. The model's particle-size distribution closely matches that of the original soil. The average error between the 3D model and the actual soil was 0.29%, and the maximum error was 0.58%. It occurred in the Poisson round of the particle insertion. Errors were regarded as small and acceptable. The model's void ratio of 0.584 was close to the target of 0.600.

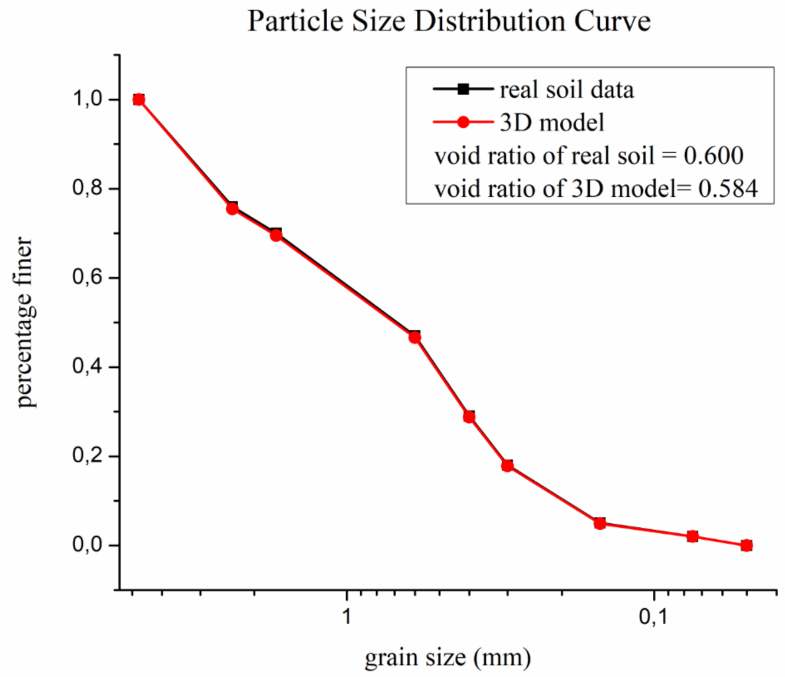


Fig 4.6 Comparison of particle-size distributions of the algorithm-generated 3D model and the real soil

The curve in Fig 4.7 shows the relationship between the void ratio of the deposited model and the contact friction defined in Yade. The void ratio initially increased with rising contact friction but remained constant once the contact friction exceeded 45°, which can be considered the threshold value. Setting contact frictions greater than this point guarantees that the final void ratio closely matches the target void ratio after the deposition process. The particle packing is deemed stable and tightly packed if the coordination number is higher than 4 (Masanobu 1977). The 3D model's coordination number was 4.77, meaning that particles were densely packed after the gravity deposition. Thus, inter-particle friction can be used to adjust the void ratio.

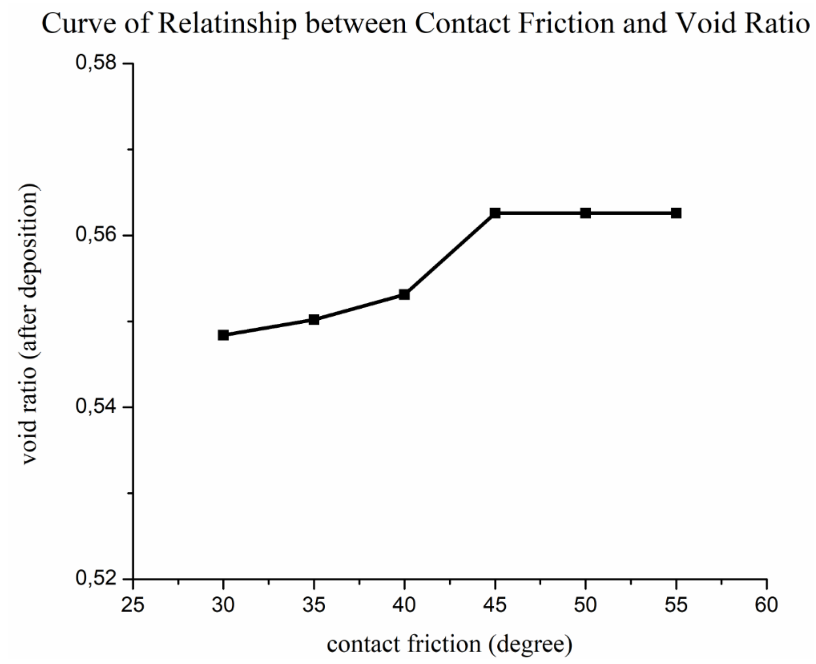


Fig 4.7 The relationship between contact friction and void ratio of 3D deposited model

The result of 3D model deposition is shown in Fig 4.8. The friction contact was 45º. The volume of the 3D particle packing after the model deposition shrank to 12.5mm×12.5mm×12.36mm from the original 12.5mm×12.5mm×12.5mm. The void ratio was 0.563, which reduced slightly from the original value of 0.584 and was close to the target of 0.6. The running time was 1,987 seconds.

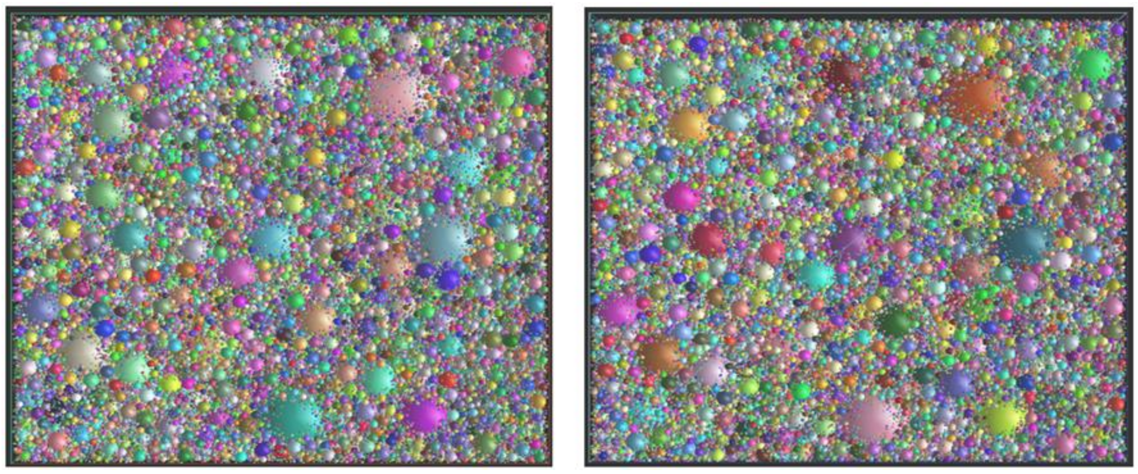

(a) (b)

Fig 4.8 The comparison of the original model (a) and the deposited model (b)

4.3.2 Mixed Sand Example

The second example illustrating the process of building DEM models aims to simulate silica sand, a gravel-sand mixture widely used as a proppant by companies in the oil and natural gas recovery industry (Campos et al. 2018). The particle-size distribution shown in Table 4.6 is taken from the Geotechnical Aspects of Pavements Reference Manual (2006). The void ratio of typical silica sand is 0.49~0.79 (Ahn and Jung 2017). The void ratio for this example is assigned a value of 0.55.

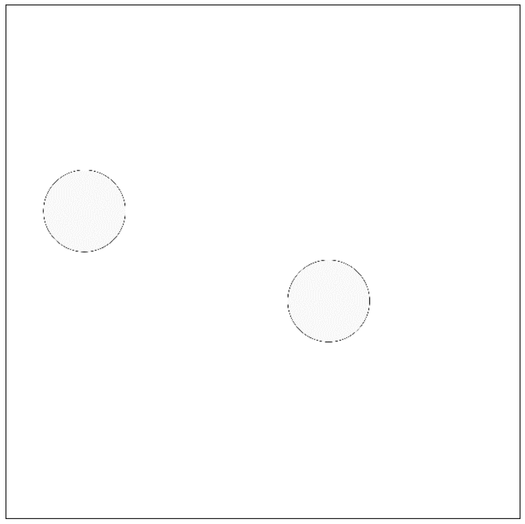
Table 4.6 Particle-size distribution of Silica Sand (Geotechnical Aspects of Pavements Reference Manual 2006)

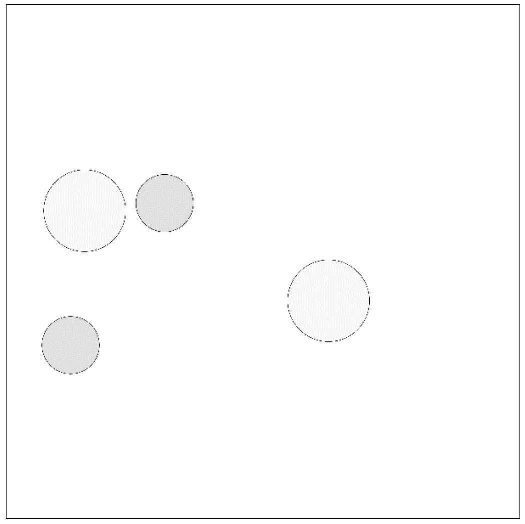
Sieve (*mm*)	10-4.75	4.75-3.35	3.35-2	2-1	1-0.85	0.85-0.5	0.5-0.3	0.3-0.2	0.2-0.1	<0.1
Radius (*unit*)	400-190	190-134	134-80	80-40	40-34	34-20	20-12	12-8	8-4	<4
Finer percentage	100%	95%	93%	90%	68%	44%	20%	8%	2%	0.5%
Mass percentage for each sieve	5%	3%	2%	22%	24%	24%	12%	6%	1.5%	0.5%

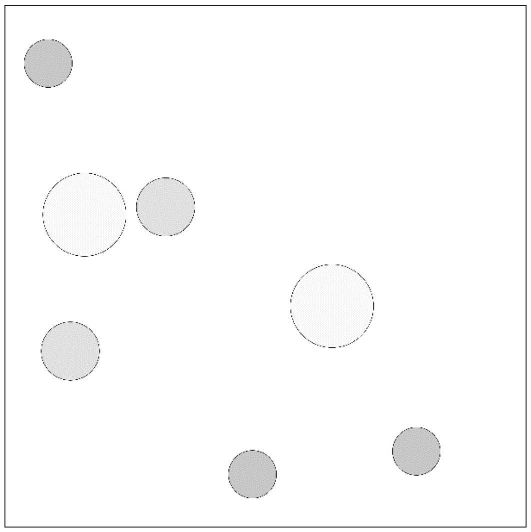
Based on the sieve opening sizes, the unit size was set to 0.025mm. Due to a wide range of particle radii, the canvas size needed to be larger than that used for the pure sand example and was set to 36mm×36mm/2400unit×2400unit. It took ten rounds of particle insertion to generate the model. The cell size used in the void-filling process was 5unit×5unit. The target volume of the particle assembly was $2400^2 \div 1.55 = 3.7 \times 10^6$ unit^2. Table 4.7 shows the target volumes for every sieve in the 2D model. The corresponding 2D particle packing generated by the algorithm is shown in Fig 4.9.

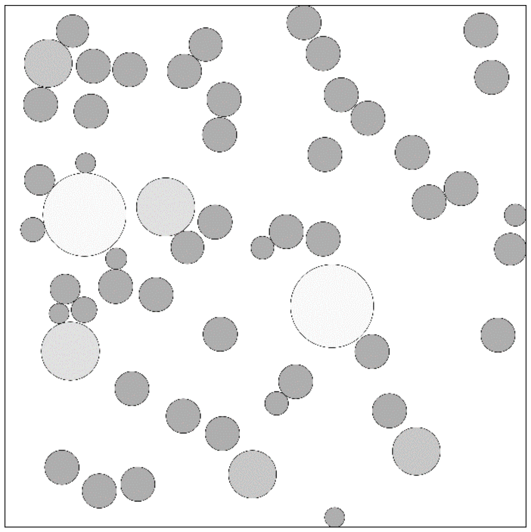
Table 4.7 Finer percentage and target volumes of 2D model for every sieve

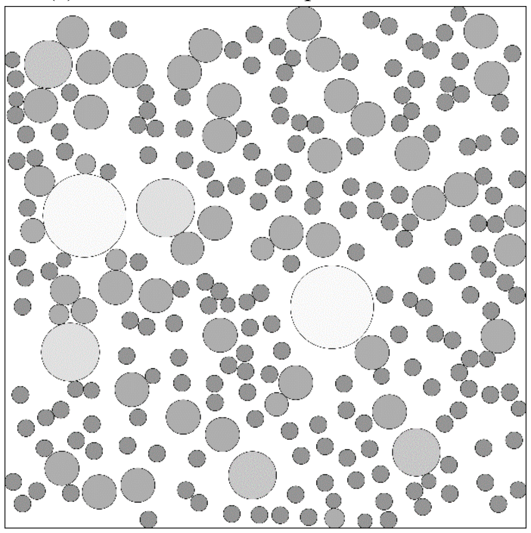
Radius (*unit*)	400-190	190-134	134-80	80-40	40-34	34-20	20-12	12-8	8-4	<4
Percentage finer	5%	3%	2%	22%	24%	24%	12%	6%	1.5%	0.5%
Target volume ($\times 10^5$ *unit*^{2})	1.85	1.11	0.74	8.14	8.88	8.88	4.44	2.22	0.56	0.18


(a) Poisson round of 2D particle insertion


(b) First round of 2D particle insertion


(c) Second round of 2D particle insertion


(d) Third round of 2D particle insertion


(e) Fourth round of 2D particle insertion

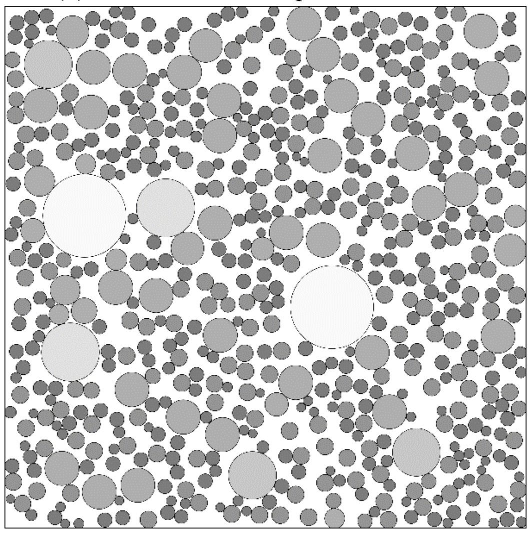
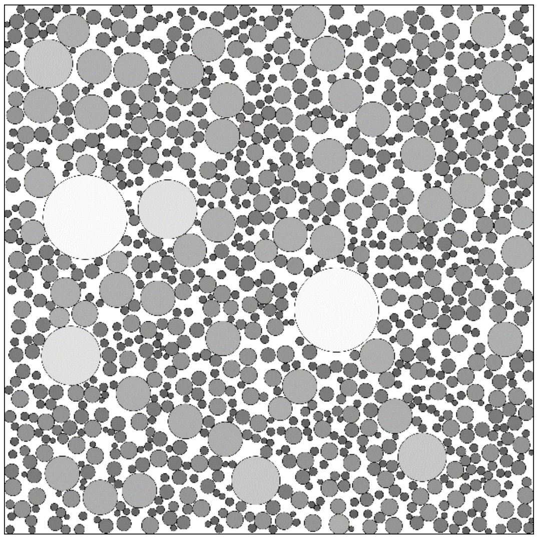
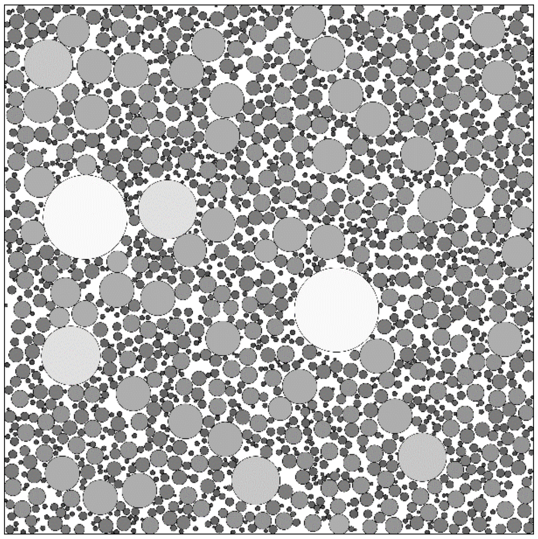
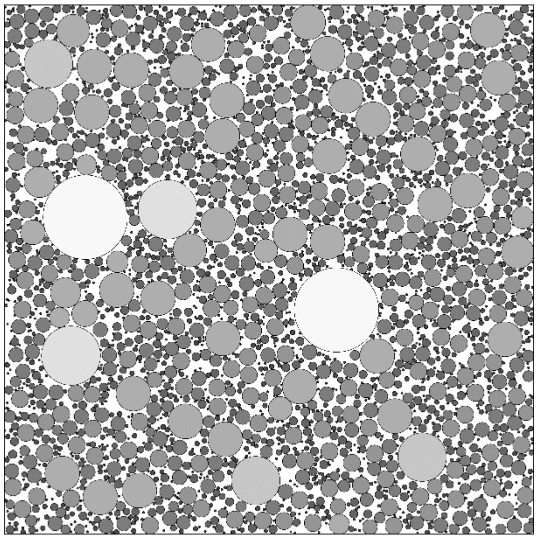

(f) Fifth round of 2D particle insertion

Fig 4.9 2D model generation for mixed sand


(g) Sixth round of 2D particle insertion


(h) Seventh round of 2D particle insertion


(i) Eighth round of 2D particle insertion

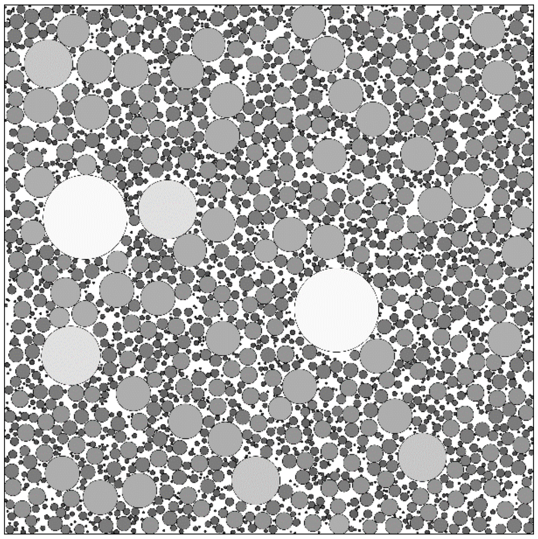

(j) Ninth round of 2D particle insertion

Fig 4.9 (Continued) 2D model generation for mixed sand

The properties of the model generated with the 2D algorithm are shown in Table 4.8. There were in total 3,391 particles in the model, and the running time was 9.7 seconds.

Table 4.8 Details of 2D particle packing

Round of particles insertion	Opening of sieves *mm*	Mass g×10^{-3}	Ideal mass g×10^{-3}	Finer percentage	Target finer percentage	Present difference
poisson	10-4.75	233.53	189.41	100%	100%	0%
1	4.75-3.35	114.94	75.76	93.99%	95%	1.01%
2	3.35-2	116.47	113.64	91.03%	93%	1.97%
3	2-1	838.97	833.39	88.03%	90%	1.97%
4	2-0.85	910.08	909.15	66.43%	66%	0.43%
5	0.85-0.5	911.24	909.15	42.95%	42%	0.95%
6	0.5-0.3	454.59	454.57	19.53%	20%	0.47%
7	0.3-0.2	227.67	227.28	7.83%	8%	0.17%
8	0.2-0.1	57.20	56.82	1.97%	2%	0.03%
9	<0.01	19.12	18.93	0.49%	0.5%	0.01%

Fig 4.10 shows the particle-size distribution curves of the generated model and the actual soil sample. The model created by the algorithm agrees with the actual soil in terms of particle-size distribution. The average error was 0.78%, and the maximum error was 1.97%, which occurred in the second and third rounds of particle insertion due to volumes of particles for the first two sieves exceeding the target by over 2% and resulting in volume errors. Using recursion with a shrunken radius range to reduce the total volume can decrease the errors, as described in Section 3.4.3. However, in this case, even setting all radii of particles to the minimum value in the radius range, the overshoot still existed. When the algorithm generated the last particle belonging to the two aforementioned sieves, the volumes substantially surpassed the target. However, without the last particle, the volume would not reach the target. The errors of the particles generated for the other sieves were regarded as small and acceptable. The void ratio of the 2D model was 0.523, which is close to 0.550, which is that of the target sand mixture.

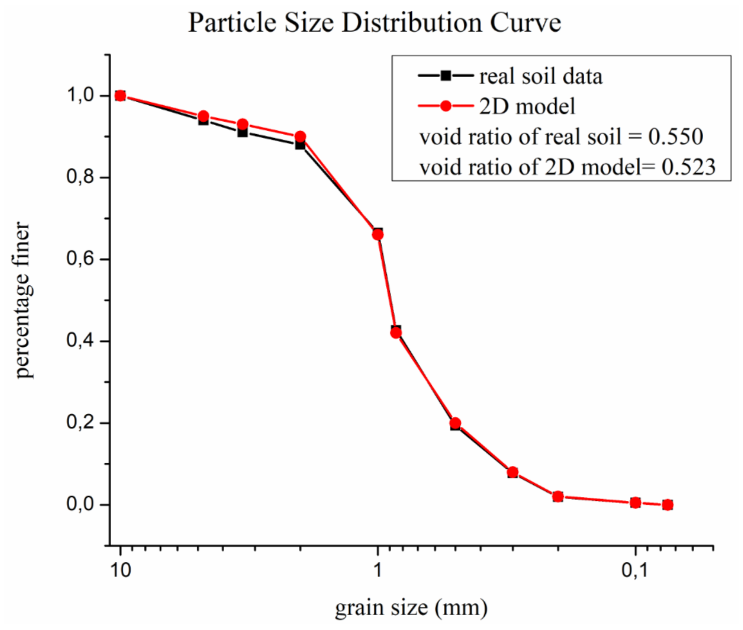


Fig 4.10 Comparison of particle-size distributions of the algorithm-generated 2D model and the real soil

Fig 4.11 shows that the relationship between the void ratio of the deposited particle assembly and contact friction. The void ratio slightly increased as contact friction rose, indicating that contact friction barely impacted the void ratio of the 2D deposited model. The coordination number of the 2D particle assembly was 2.47, which is lower than 4, meaning particles were not tightly packed (Masanobu 1977). Thus, since inter-particle friction relies on particle contact, it cannot influence soil density.

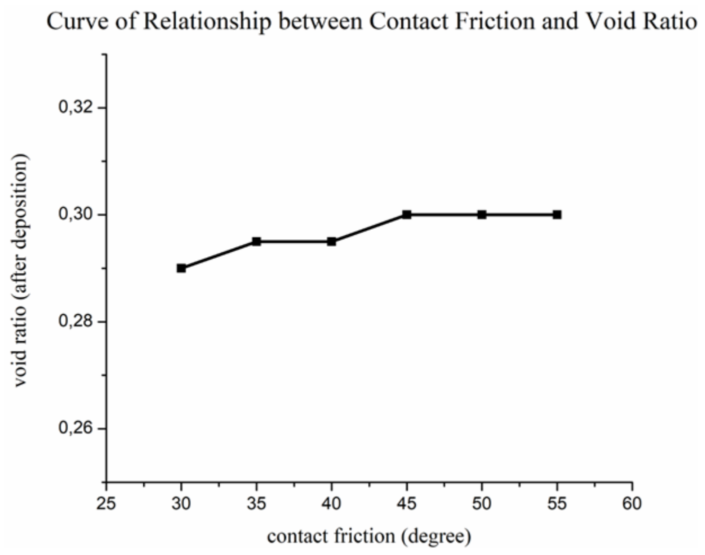


Fig 4.11 The relationship between contact friction and void ratio of 2D deposited model

The 2D particle assembly deposition simulated by Yade is shown in Fig 4.12. The void ratio of the deposited model decreased to 0.300 compared with the original 0.523, and the model's volume correspondingly shrank from 36mm×36mm to 36mm×27mm. It took 53 seconds to run the algorithm.

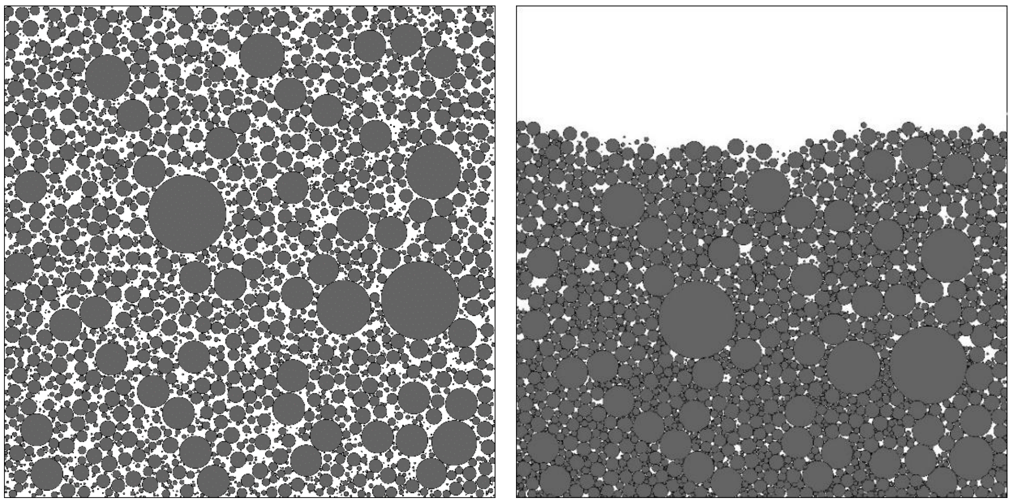


Fig 4.12 The comparison of the original model (a) and the deposited model (b)

The void ratio decreased sharply after the model deposition process, which can be attributed to the pivotal role of the particle assembly's particle size distribution. Compared to the pure sand sample, the gravel-sand mixture had a more comprehensive radius range, and large particles of the first four sieves accounted for just 10% of the total volume. Thus, a large number of small particles filled nearly all the voids in the model, causing the void ratio to reduce considerably.

For the 3D model generation, the canvas size was set to 36mm×36mm×36mm/2400unit×2400unit ×2400unit, and the remaining parameters were the same as those of the 2D model. The target volume of the 3D particles was $2400^3 \div 1.55 = 8.9 \times 10^9$ unit^3. Table 4.9 shows the target volumes for every sieve in the 3D model. The 3D model generated by the algorithm is shown in Fig 4.13 sieve-by-sieve.

Table 4.9 Finer percentage and target volume of 3D model for every sieve

Radius (*unit*)	400-190	190-134	134-80	80-40	40-34	34-20	20-12	12-8	8-4	<4
Percentage finer	5%	3%	2%	22%	24%	24%	12%	6%	1.5%	0.5%
Target volume ($\times 10^8$ *unit*^{3})	4.55	2.67	1.78	19.58	21.36	21.36	10.68	5.34	1.34	0.45

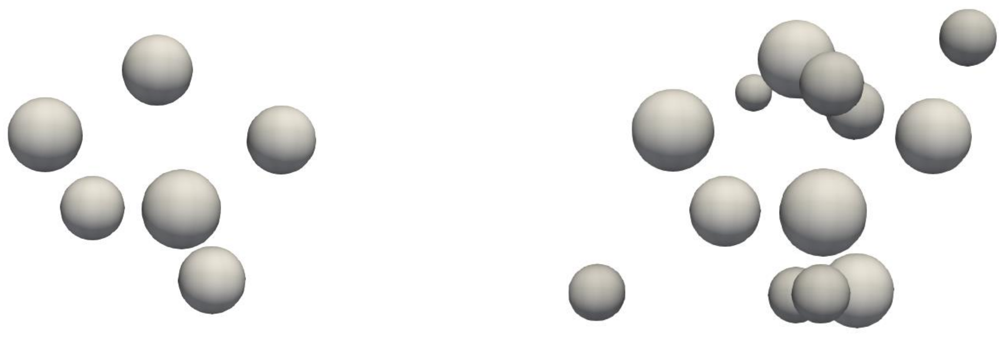
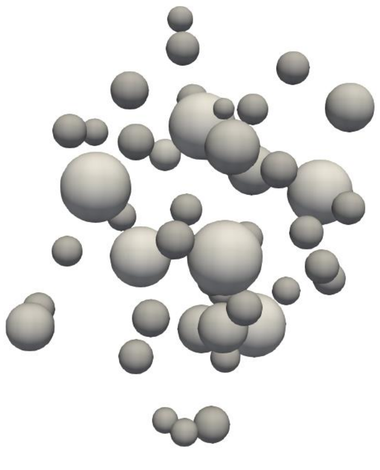
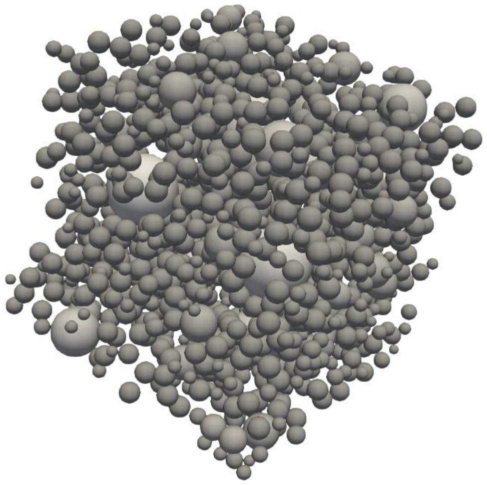
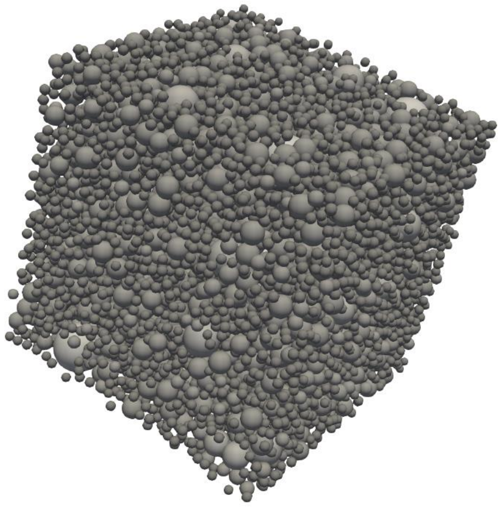
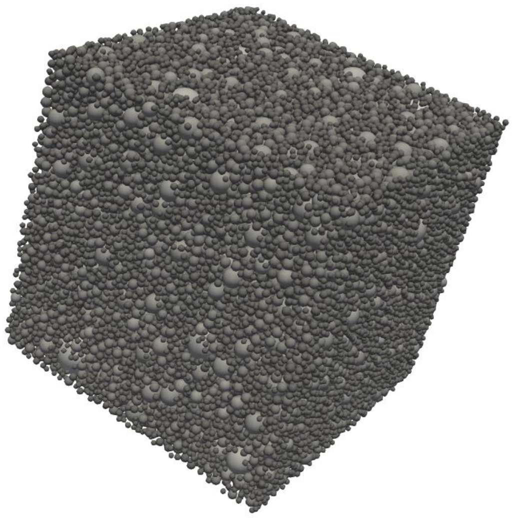

(a) Poisson round of 3D particle insertion (b) First round of 3D particle insertion

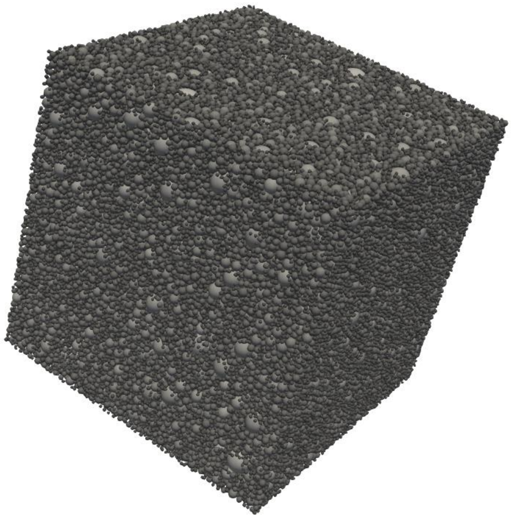
Fig 4.13 3D model generation for mixed sand


(c) Second round of 3D particle insertion


(d) Third round of 3D particle insertion


(e) Fourth round of 3D particle insertion


(f) Fifth round of 3D particle insertion


(g) Sixth round of 3D particle insertion

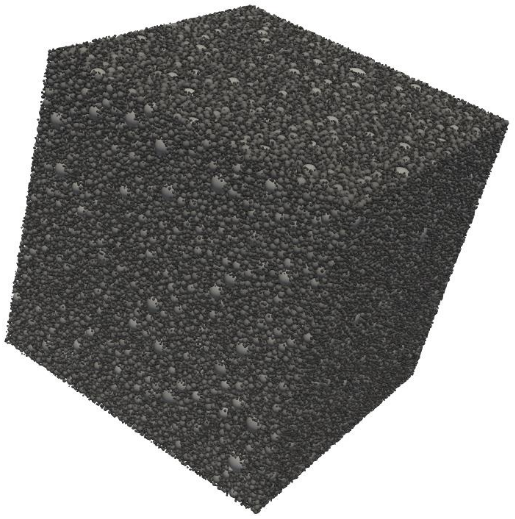
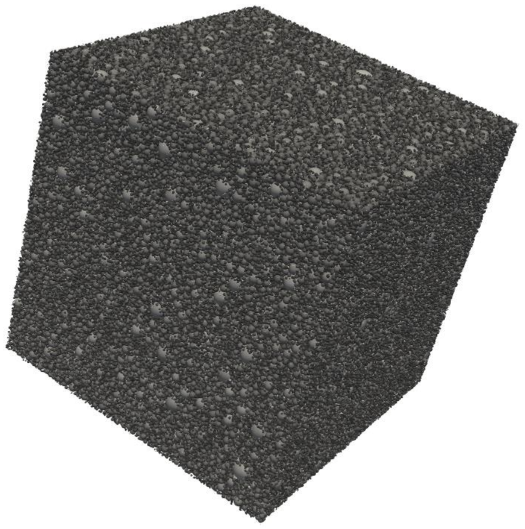

(h) Seventh round of 3D particle insertion

Fig 4.13 (Continued) 3D model generation for mixed sand


(i) Eighth round of 3D particle insertion

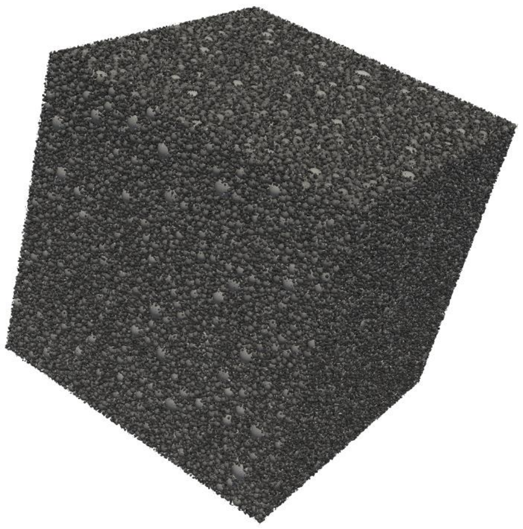

(j) Ninth round of 3D particle insertion

Fig 4.13 (Continued) 3D model generation for mixed sand

Table 4.10 shows details of the 3D model. The total number of spherical particles in the algorithm-generated model was 1,048,576, and the running time was 100,766 seconds. The generation of particles in the two lowest sieves was the most time-consuming, accounting for 50% of the running time.

Table 4.10 Details of 3D particle packing

Round of particles insertion	Opening of sieves *mm*	Mass $g \times 10^{-1}$	Target mass $g \times 10^{-1}$	Finer percentage	Target finer percentage	Percent difference
poisson	10-4.75	115.19	113.64	100%	100%	0
1	4.75-3.35	45.74	45.45	94.94%	95%	0.06%
2	3.35-2	68.27	68.18	92.93%	93%	0.07%
3	2-1	500.33	500.03	89.93%	90%	0.07%
4	2-0.85	545.52	545.49	65.96%	66%	0.04%
5	0.85-0.5	545.54	545.49	41.99%	42%	0.01%
6	0.5-0.3	272.93	272.74	20.00%	20%	0%
7	0.3-0.2	136.43	136.37	8.00%	8%	0%
8	0.2-0.1	34.35	34.09	2.00%	2%	0%
9	<0.01	11.47	11.36	0.50%	0.5%	0%

The particle-size distribution curves of the 3D model and the real soil are shown in Fig 4.14. The particle-size distribution of the algorithm-generated model is consistent with that of the actual soil sample. The average error between the 3D model and the real soil data was 0.06%, and the maximum error was 0.07% which occurred during the first and second rounds of particle insertion. Errors were considered acceptable. The void ratio of the model was 0.547, which is also close to the target of 0.550.

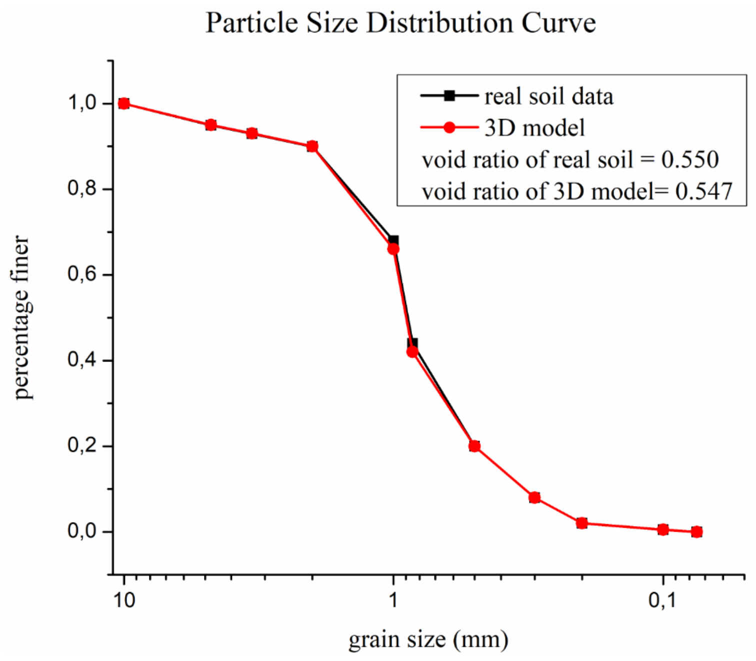


Fig 4.14 Comparison of particle-size distributions of the algorithm-generated 3D model and the real soil

If the model with a considerable number of particles is imported into the Yade simulation program, the running time will be extremely long. This thesis proposed a method to simplify the imported model to avoid this problem. The void ratio of the particle packing after the deposition process can be estimated by importing the model without the smallest sieve particles, which accounts for 0.5% of the total volume in this case. After neglecting the smallest particles, the total number of particles in the model decreased considerably to a little over 700,000, a number that Yade can easily handle. The effect of deleting the smallest particles in the model was deemed small and acceptable.

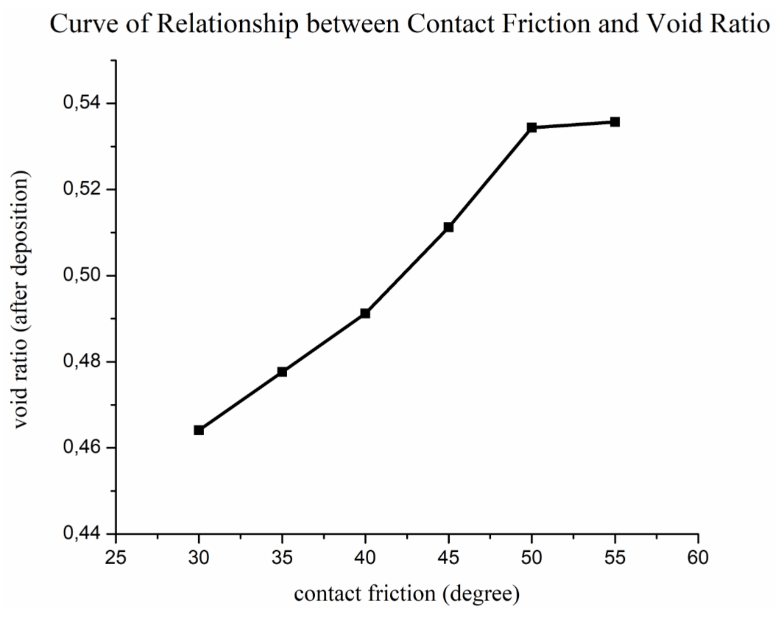


Fig 4.15 The relationship between contact friction and void ratio of 3D deposited model

Fig 4.15 shows the relationship between the void ratio of the deposited particle packing and contact friction. The model's void ratio significantly increased as contact friction grew but remained roughly constant when the contact friction was greater than 50º, which can be regarded as the threshold point. Using a contact friction that exceeds the threshold point guarantees the void ratio experiences only a minor reduction and stays close to the target void ratio after the model deposition process. For the 3D deposited model, the coordination number was 4.94, which is higher than 4, indicating that the particle packing was dense and stable (Masanobu 1977). Thus, inter-particle friction that functions based on particle contacts can help to adjust the void ratio.

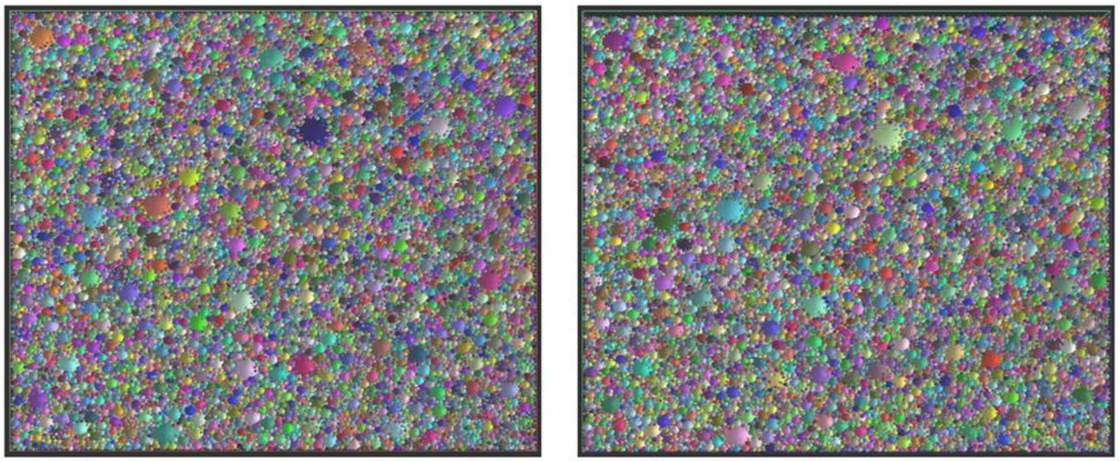


Fig 4.16 The comparison of the original model (a) and the deposited model (b)

Fig 4.16 shows the model deposition simulation results. The contact friction was set to 50º. The void ratio was 0.547 initially and decreased slightly to 0.534 after the model deposition. The model volume accordingly shrank from 36mm×36mm×36mm to 36mm×36mm×35.685mm. The particle deposition process took a total of 890 seconds.

4.3.3 Gap Graded Sand Example

The third example consists in creating models to simulate gap graded soils. The real soil data is based on Reboul et al. (2010). The particles of the actual gap graded soil whose radii range from 2 mm to 0.3mm are missing. The particle-size distribution of the real soil data is shown in Table 4.11, and the void ratio of the soil sample was 0.55.

Table 4.11 Particle-size distribution of gap graded sand (Reboul et al. 2010)

Opening of sieves (*mm*)	10.5-10	10-8	8-2	2-0.3	0.3-0.25	0.25-0.2	0.2-0.15	0.15-0.125	<0.125
Radius (*unit*)	420-400	400-320	320-80	80-12	12-10	10-8	8-6	6-5	<5
Finer percentage	100%	83%	52%	15%	15%	6%	3.5%	1.5%	0.5%
Mass percentage for each sieve	17%	21%	37%	0%	9%	2.5%	2%	1%	0.5%

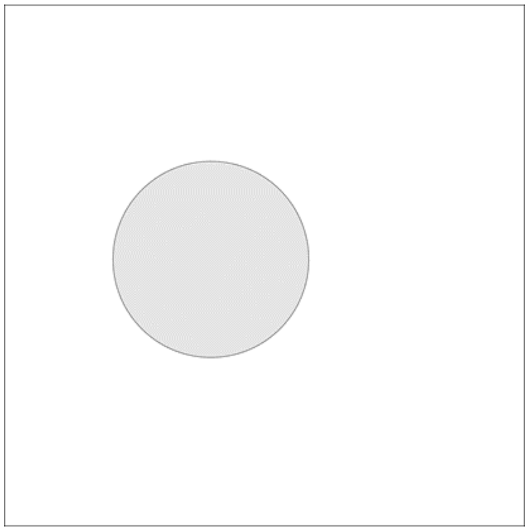
For the 2D model generation, the unit size was set to 0.025 mm. The canvas size had to be greater than the one used in previous models and was 2400unit×2400unit/60mm ×60mm.The total volume was $2400^2 \div 1.55 = 3.72 \times 10^6$ unit^2. The percentage passing and the target volumes for each sieve are shown in Table 4.12.

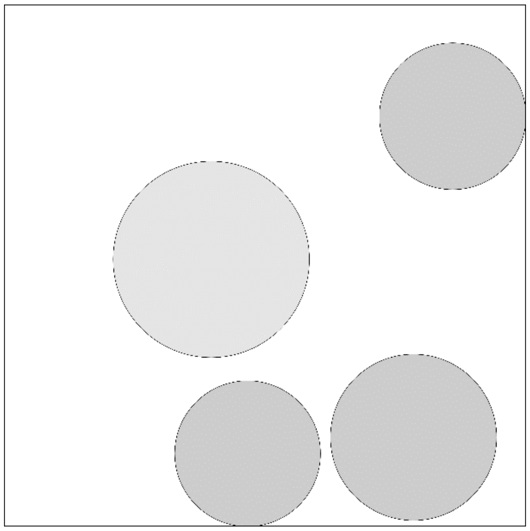
Table 4.12 Finer percentage and target volumes for each sieve of 2D model

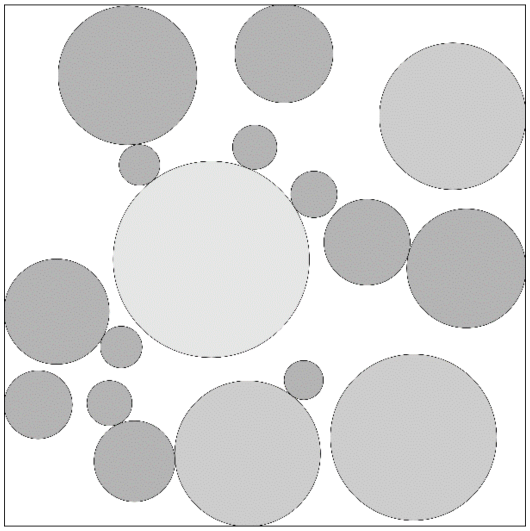
Opening of sieves(*mm*)	10.5-10	10-8	8-2	2-0.3	0.3-0.25	0.25-0.2	0.2-0.15	0.15-0.125	0.125-0.1
Finer percentage	17%	31%	37%	0%	9%	2.5%	2%	1%	0.5%
Target volume ($\times 10^4 unit^3$)	63	115	138	0	33.5	9.3	7.4	0.372	0.186

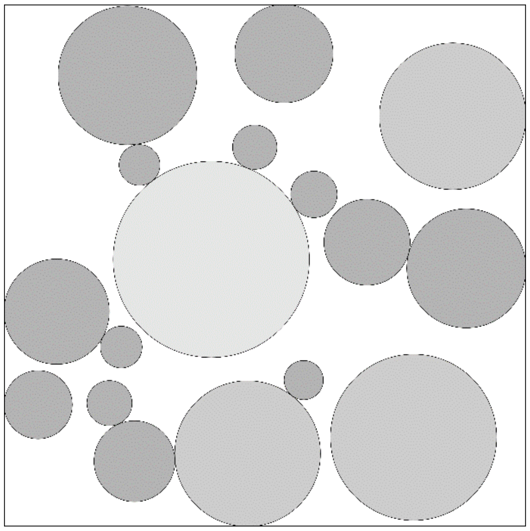
The model generation process was divided into nine rounds of particle insertion. The third round of particle insertion generated particles with radii ranging from 2mm to 0.3mm, which were missing in the gap graded soil. Thus, the algorithm did not generate particles in this round. The cell size used in the void-filling process was 5unit×5unit.

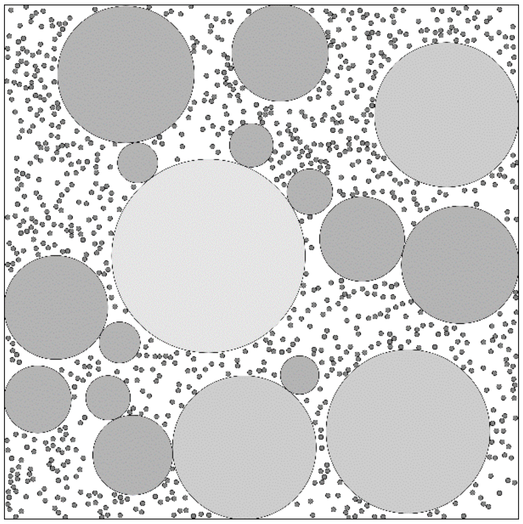
The 2D algorithm-generated model is shown in Fig 4.17 sieve-by-sieve.


(a)   Poisson round of 2D particle insertion


(b) First round of 2D particle insertion


(c) Second round of 2D particle insertion


(d) Third round of 2D particle insertion


(e) Fourth round of 2D particle insertion

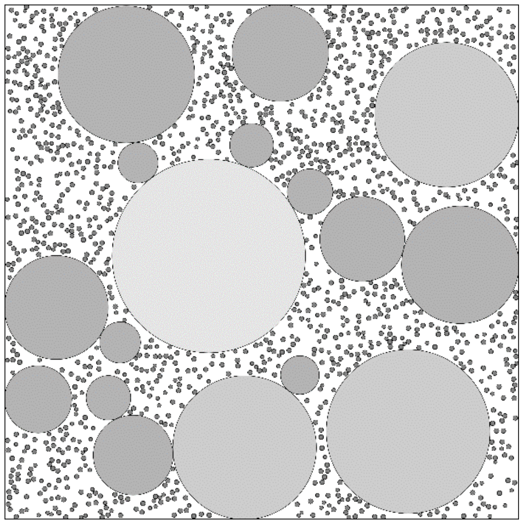
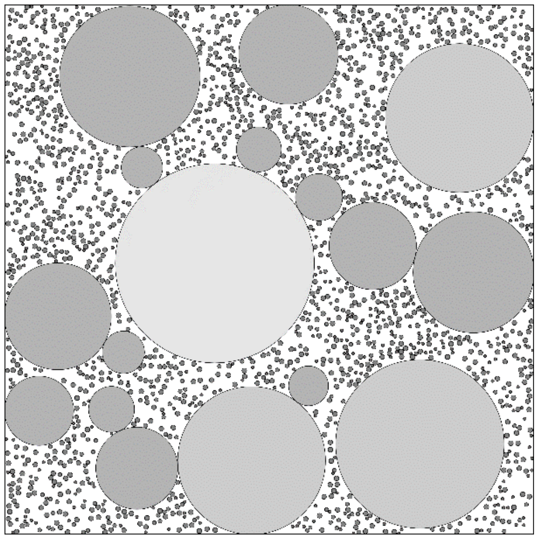
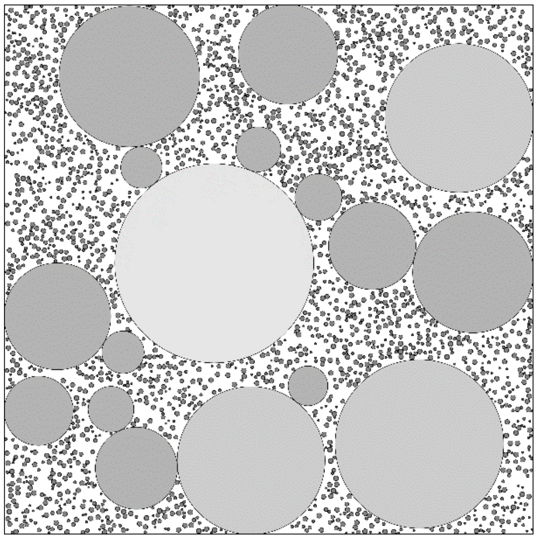

(f) Fifth round of 2D particle insertion

Fig 4.17 2D model generation for gap graded sand


(g) Sixth round of 2D particle insertion


(h) Seventh round of 2D particle insertion

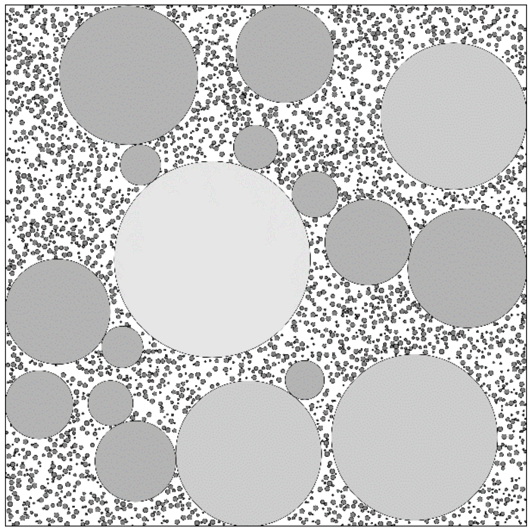

(i) Eighth round of 2D particle insertion

Fig 4.17 (Continued) 2D model generation for gap graded sand

The results for 2D particle packing are shown in Table 4.13. There were 2,625 particles in the 2D model, and the running time was 19 seconds.

Table 4.13 Details of 2D particle packing

Round of particle insertion	Opening of sieves *mm*	Mass $g\times10^{-2}$	Target mass $g\times10^{-2}$	Finer percentage	Target finer percentage	Percent difference
Poisson	10.5-10	65.14	64.40	100%	100%	0%
1	10-8	119.04	117.43	82.90%	83%	0.10%
2	8-2	142.11	140.16	51.95%	52%	0.05%
3	2-0.3	0	0	14.87%	15%	0.13%
4	0.3-0.25	34.13	34.09	14.87%	15%	0.13%
5	0.25-0.2	9.51	9.47	5.96%	6%	0.04%
6	0.2-0.15	7.62	7.57	3.48%	3.5%	0.02%
7	0.15-0.125	3.82	3.79	1.49%	1.5%	0.01%
8	0.125-0.1	1.91	1.89	0.5%	0.5%	0%

The particle-size distributions of the 2D model and the soil sample shown in Fig 4.18 are very close to each other. The average error was 0.06%. The maximum error was 0.12% and took place during the third round of particle insertion. The volume errors of particles generated for the remaining sieves were deemed acceptable. The void ratio of the 2D model is 0.532, close to the real soil of 0.550.

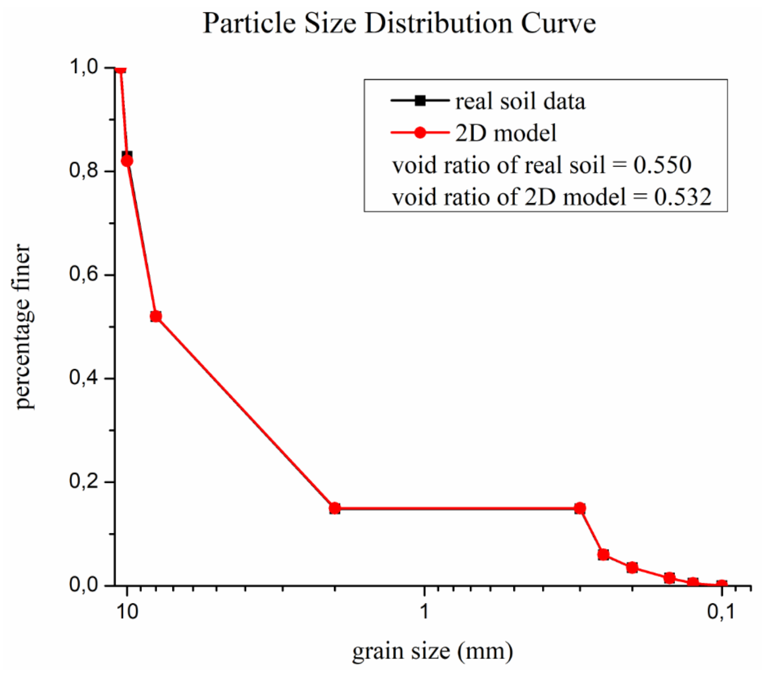


Fig 4.18 Comparison of particle-size distributions of the algorithm-generated 2D model and the real soil

The contact friction had little influence on the void ratio of the 2D deposited model, shown in Fig 4.19. The void ratio increased slightly only when the contact friction was changed from 40º to 45º. Contact friction could barely influence the post-deposition void ratio. The coordination number of the 2D deposited model was 2.67, which is lower than 4, meaning the particles were loosely packed (Masanobu 1977). Thus, inter-particle friction just has a slight influence on the model density.

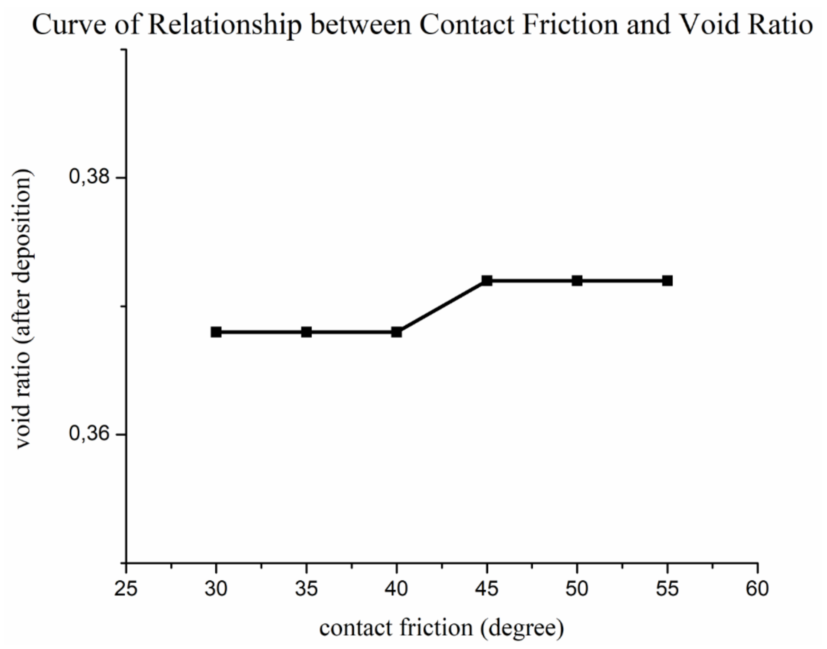


Fig 4.19 The relationship between contact friction and void ratio of 2D deposited model

Fig 4.20 shows the result of the 2D model deposition. The void ratio of the deposited model was 0.373, which decreased compared with the original void ratio of 0.532, and the model's volume shrank from 60mm×60mm to 60mm×46mm. The running time was 22 seconds.

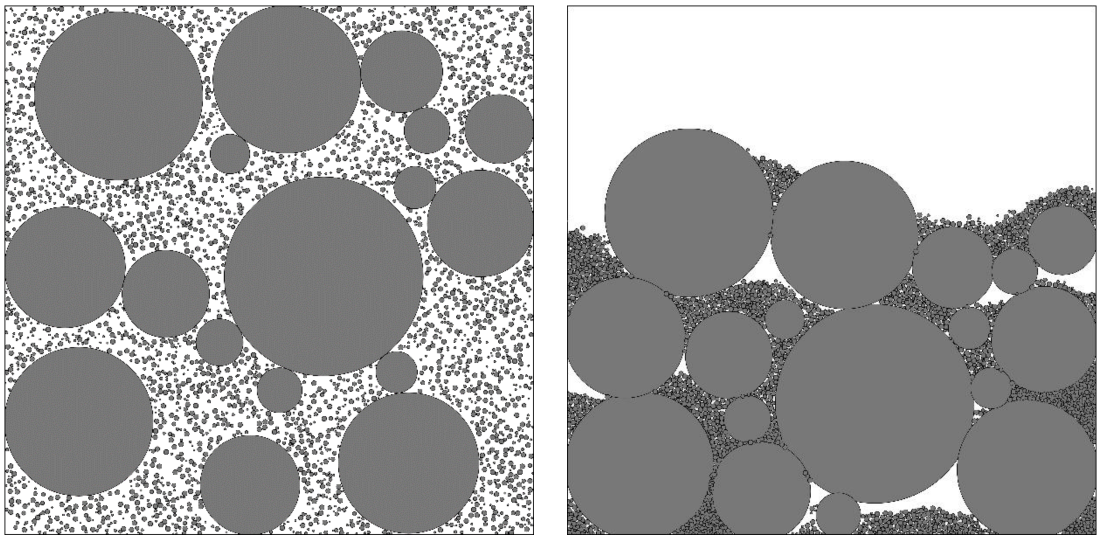


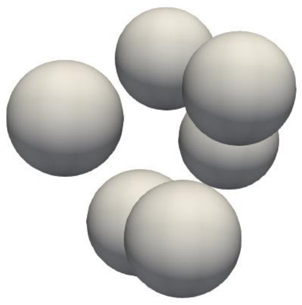
Fig 4.20 The comparison of the original model (a) and the deposited model (b)

The void ratio decreased significantly after the model was deposited. This is due to the fact that in gap graded soils, large particles make up the soil skeleton, and their interlocking plays a decisive role in determining the void ratio of the deposited model (Ke and Takahashi 2012). Small particles, on the other hand, act as separators and merely fill the gaps between the larger particles during the deposition process, having only a minor influence on the void ratio. In addition, well-graded soils develop better particle interlocking than poorly graded ones (Khan 2012). Thus, weak interlocking in gap graded soils played a role in the void ratio reduction after the model deposition.

For the 3D model, the canvas size was set to 2400unit×2400unit×2400unit/60mm×60mm ×60mm. The remaining parameters of the 3D model were the same as in the 2D model. The target volume of 3D particles was $2400^3 \div 1.55 = 8.919 \times 10^9$ unit^3. Based on Equations 3-2 and 3-3, the particle-size distribution and target volumes of the 3D particle packing for every sieve are shown in Table 4.14. The 3D model building process is shown in Fig 4.21 sieve-by-sieve.

Table 4.14 Finer percentage and target volumes for each sieve of 3D model

Opening of sieves(*mm*)	10.5-10	10-8	8-2	2-0.3	0.3-0.25	0.25-0.2	0.2-0.15	0.15-0.125	0.125-0.1
Finer percentage	17%	31%	37%	0%	9%	2.5%	2%	1%	0.5%
Target volume ($\times 10^8 unit^3$)	15.2	27.6	33	0	8.03	2.23	1.78	0.89	0.45


(a) Poisson round of 3D particles

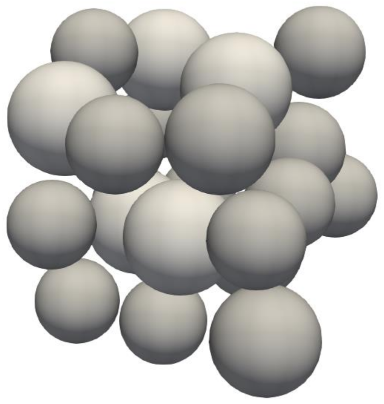
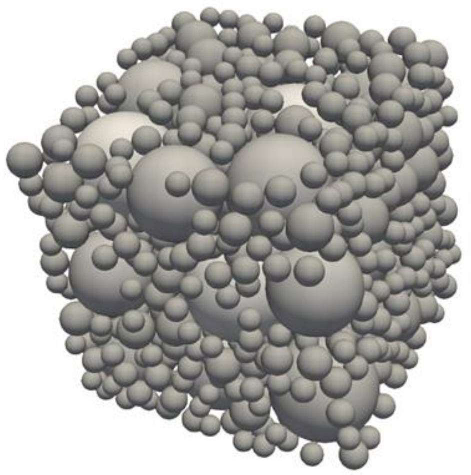
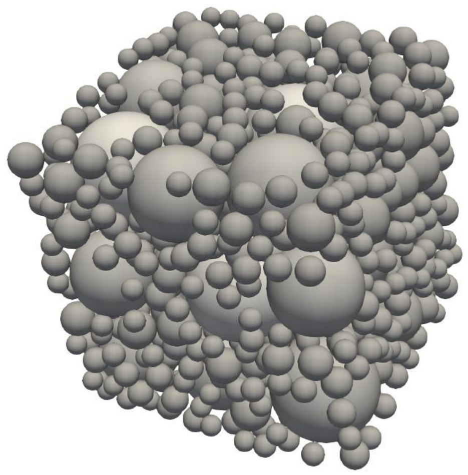
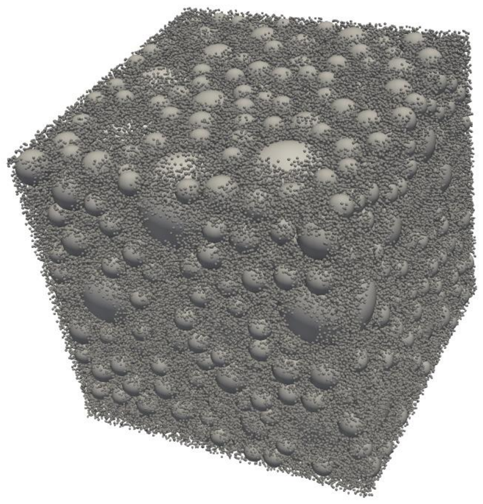
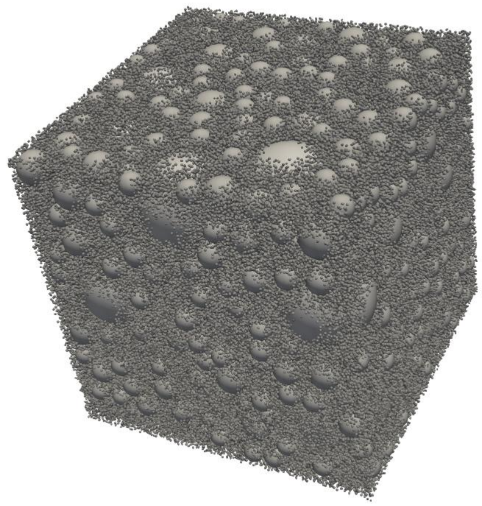

(b) First round of 3D particles insertion

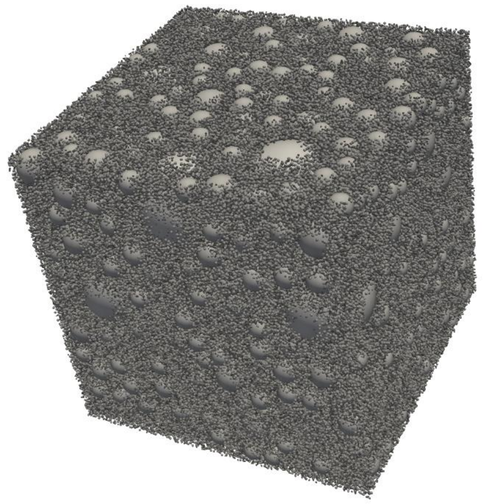
Fig 4.21 3D model generation for gap graded sand


(c) Second round of 3D particles insertion


(d) Third round of 3D particles insertion


(e) Fourth round of 3D particles insertion


(f) Fifth round of 3D particles insertion


(g) Sixth round of 3D particles insertion

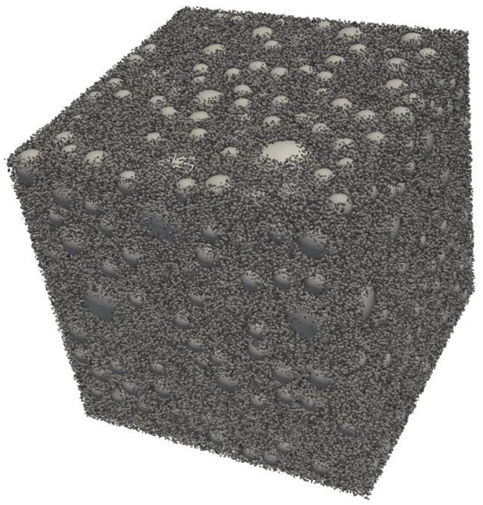

(h) Seventh round of 3D particles insertion

Fig 4.21 (Continued) 3D model generation for gap graded sand

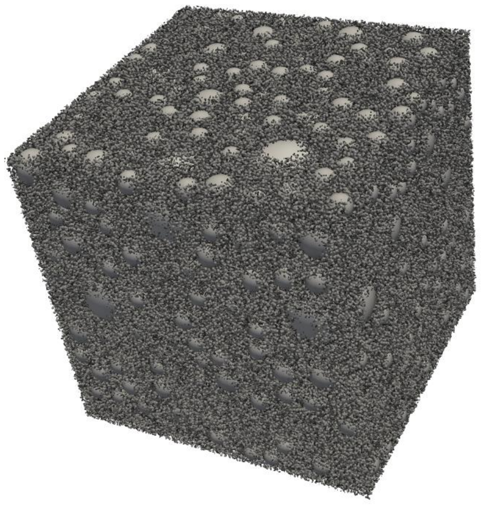

(i) Eighth round of 3D particles insertion

Fig 4.21 (Continued) 3D model generation for gap graded sand

The results of the 3D model are shown in Table 4.15. There were 583,777 spherical particles in the 3D model. The running time was 117,377 seconds. Creating particles belonging to the smallest opening sieve was the most time-consuming, accounting for roughly 40% of the total running time.

Table 4.15 Details of 3D particle packing

Round of particles insertion	Opening of sieves (*mm*)	Mass $g\times10^0$	Target mass $g\times10^0$	Finer percentage	Target finer percentage	Percent difference
Poisson	10.5-10	40.29	38.64	100%	100%	0%
1	10-8	71.70	70.46	82.25%	83%	0.75%
2	8-2	84.10	84.09	51.4%	52%	0.6%
3	2-0.3	0	0	14.78%	15%	0.12%
4	0.3-0.25	20.48	20.46	14.78%	15%	0.12%
5	0.25-0.2	5.68	5.68	5.91%	6%	0.09%
6	0.2-0.15	4.55	4.55	3.45%	3.5%	0.15%
7	0.15-0.125	2.28	2.27	1.48%	1.5%	0.02%
8	0.125-0.1	1.13	1.13	0.49%	0.5%	0.01%

The particle-size distribution of the 3D model shown in Fig 4.22 compares well with the actual soil sample. The average error was 0.23%, and the maximum error was 0.75%, which occurred in the first round of the particle insertion. Differences were regarded as small and acceptable. The void ratio of the model was 0.526, close to the target of 0.550.

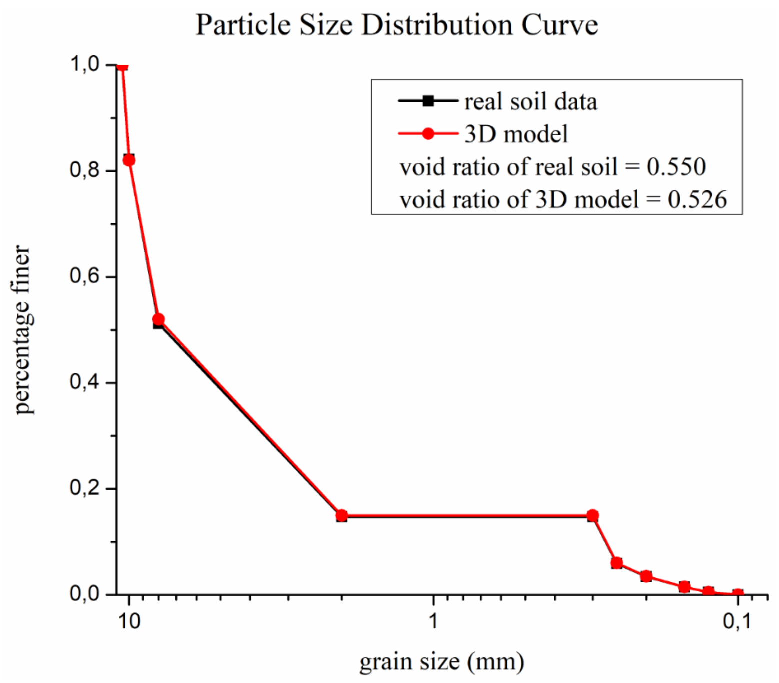


Fig 4.22 Comparison of particle-size distributions of the algorithm-generated 3D model and the real soil

The curve in Fig 4.23 shows the relationship between contact friction and the void ratio of the 3D deposited model. The model's void ratio initially increased with the growth of contact friction and remained roughly constant once the contact friction exceeded 40º. Thus, contact friction of 40º can be regarded as the threshold point. Using contact frictions greater than 40° guarantees a particle assembly only experiences minor changes in void ratio during densification. The particle packing is deemed stable and tightly packed if the coordination number is greater than 4 (Masanobu 1977). The coordination number of the 3D model was 4.52, which means spherical particles were tightly packed after the deposition. Thus, inter-particle friction played a significant role in the model's void ratio.

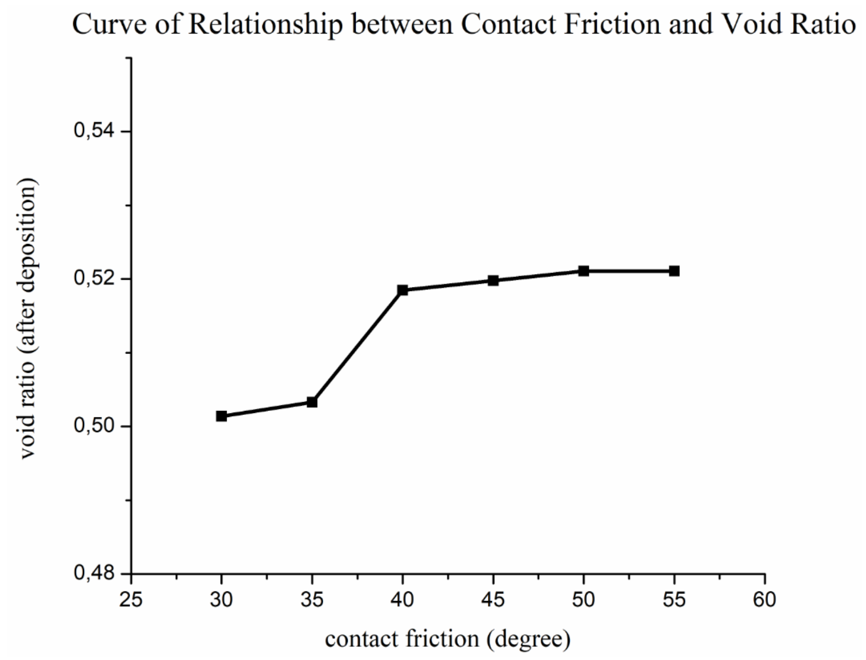


Fig 4.23 The relationship between contact friction and void ratio for 3D model deposition

Fig 4.24 shows the model deposition of 3D particle packing. The contact friction was set to 50º. The model void ratio decreased slightly to 0.521 after the deposition process, compared with an original value of 0.526, and the volume correspondingly contracted from 60mm×60mm×60mm to 60mm×60mm×59.825mm. The running time was 738 seconds.

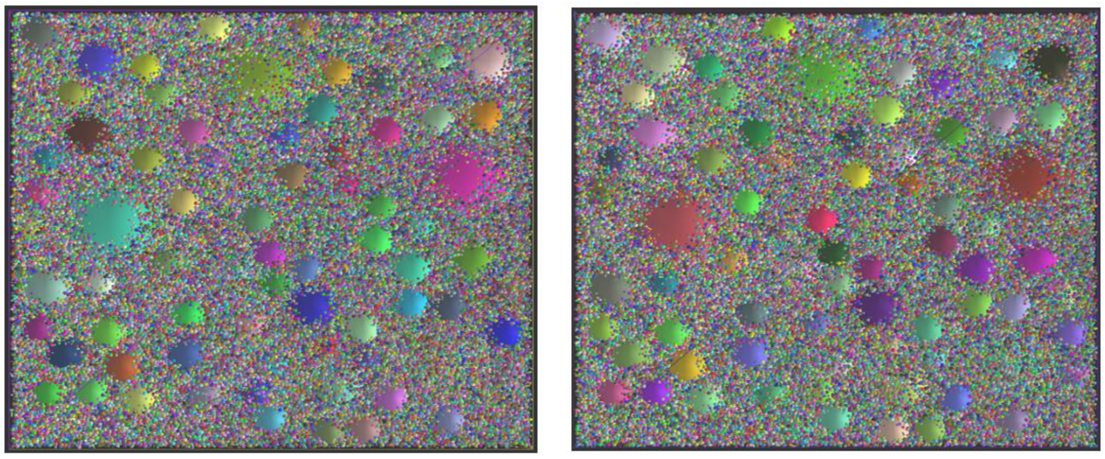

(a) (b)

Fig 4.24 The comparison of the original model (a) and the deposited model (b)

4.3.4 Uniformly Graded Sand Example

The fourth example simulated uniformly graded soils, and the data of actual uniformly graded river soils is based on Opara et al. (2008). The uniformity coefficient of the soil sample was 2.11, less than that of well-graded soils, which usually range from 4 to 6. In addition, there were only five particle insertion rounds due to the small scope of particle radii. The particle-size distribution of the real soil sample is shown in Table 4.16. The typical void ratio for poorly graded river sands is 0.8 (Dunning 2006).

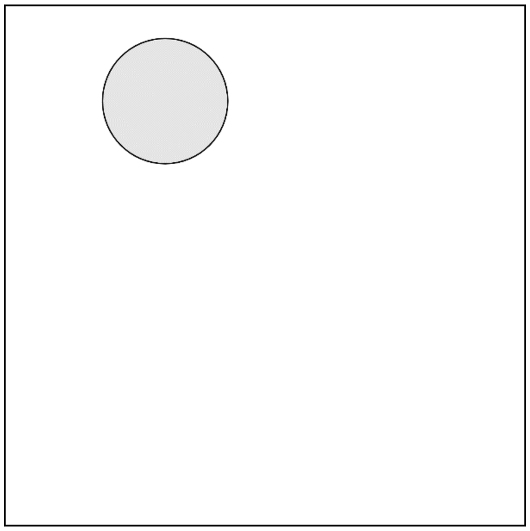
Table 4.16 Particle-size distribution of uniformly graded river sand (Opara et al. 2008)

Opening of sieve (*mm*)	4.75-2.36	2.36-1.18	1.18-0.6	0.6-0.3	0.3-0.15
Radius (*unit*)	94-48	48-24	24-12	12-6	6-2
Finer percentage	100%	92%	82%	58%	14%
Mass percentage for each sieve	8%	10%	28%	44%	15%

The unit size was set to 0.05 mm based on the sieve opening sizes. Due to a narrow range of particle radii, the canvas size was set to 800unit×800unit/40mm×40mm, which is relatively smaller than previous models. The total volume of 2D particle packing was $800^2 \div 1.8 = 3.55 \times 10^5$ unit^2. The size of cells used in the void-filling process was set to 2unit × 2unit. The finer percentage and the target volumes for each sieve are shown in Table 4.17. The 2D model created by the algorithm is shown in Fig 4.25 sieve-by-sieve.

Table 4.17 Finer percentage and target volumes for each sieve in 2D model

Opening of sieve (*mm*)	4.75-2.36	2.36-1.18	1.18-0.6	0.6-0.3	0.3-0.15
Mass percentage for each sieve	8%	10%	28%	44%	15%
Target volume ($\times 10^4$*unit*^{2})	2.84	3.55	9.94	15.6	5.33


(a) Poisson round of 2D particles insertion

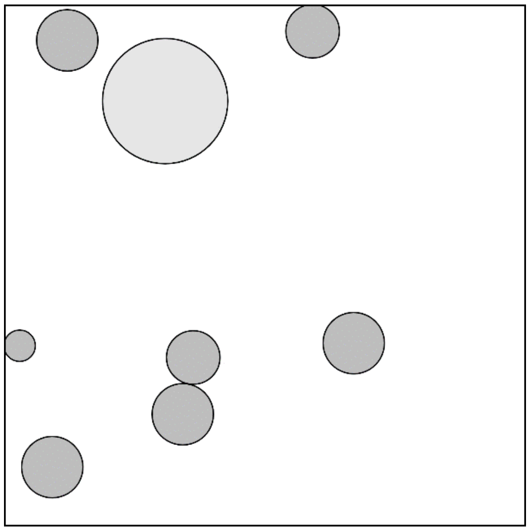
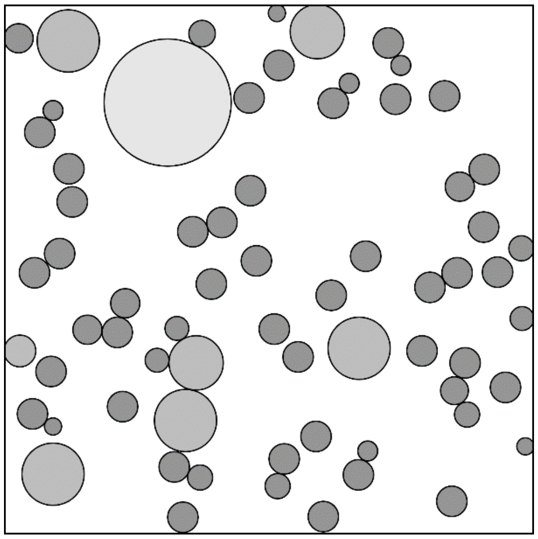
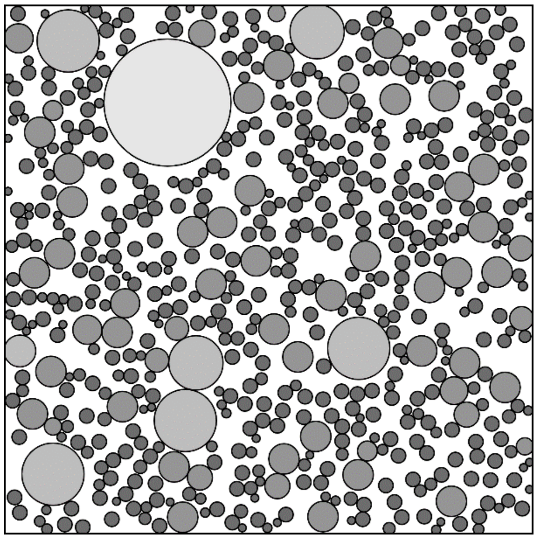

(b) First round of 2D particles insertion

Fig 4.25 2D model generation for uniformly graded sand


(c) Second round of 2D particles insertion


(d) Third round of 2D particles insertion

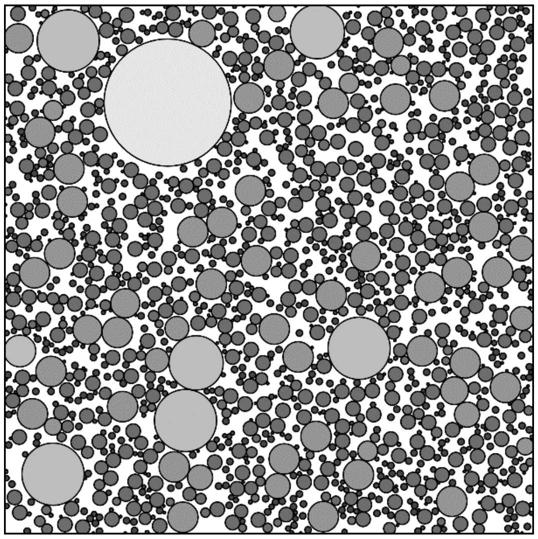

(e) Fourth round of 2D particles insertion

Fig 4.25 (Continued) 2D model generation for uniformly graded sand

Table 4.18 presents the details of 2D particle packing. There were 1,576 circular particles in total in the 2D model, and the running time was 15 seconds.

Table 4.18 Details of 2D particle packing

Round of particle insertion	Opening of sieves mm	Mass $g\times10^{-2}$	Target mass $g\times10^{-2}$	Finer percentage	Target finer percentage	Percent difference
Poisson	4.75-2.36	11.59	11.07	100%	100%	0%
1	2.36-1.18	16.36	14.49	91.45%	92%	0.55%
2	1.18-0.6	34.88	34.79	81.50%	82%	0.5%
3	0.6-0.3	63.88	63.79	57.52%	58%	0.48%
4	0.3-0.15	20.50	20.29	13.98%	14%	0.02%

The particle-size distribution curves of the model generated by the algorithm and the actual soil are shown in Fig 4.26. The particle-size distribution of the 2D model agrees with that of the real soil sample. The mean error was 0.39%, and the maximum error was 0.55%, which took place in the first round of particle insertion. The volume of particles generated in the first void-filling round exceeded the target by more than 2%. The generation of particles with a shrunken radius range, as described in Section 3.4.3, could have reduced the difference, but the overshoot still existed. After the algorithm inserted the last particle belonging to the second-largest sieve into the model, the volume significantly exceeded the target. However, without this particle, the volume would not reach the target. The errors of particles in the remaining sieves were considered acceptable. The void ratio of the model was 0.779, which is close to the target of 0.800.

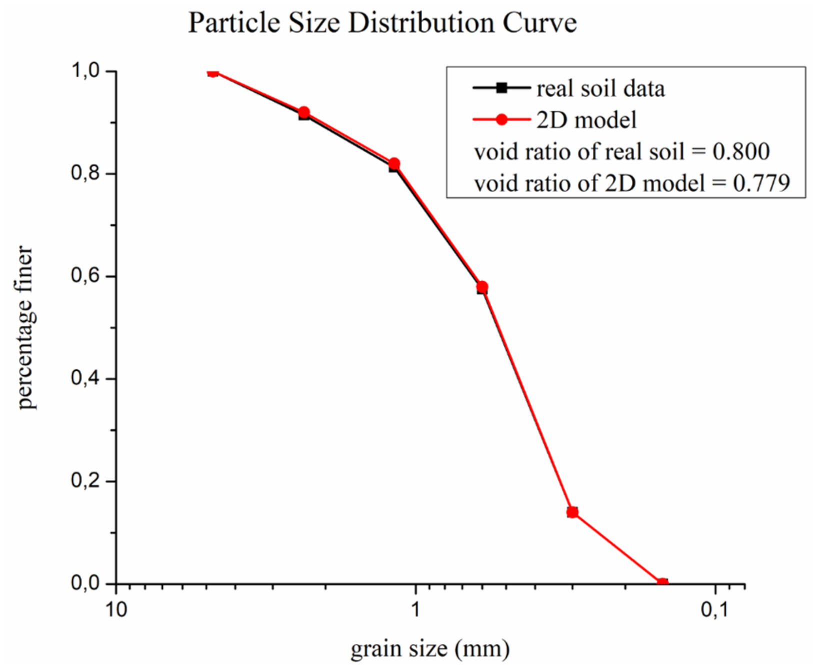


Fig 4.26 Comparison of particle-size distributions of the algorithm-generated 2D model and the real soil

The curve that describes the relationship between the void ratio of the 2D deposited model and contact friction is shown in Fig 4.27. The void ratio of the deposited model only increased slightly when contact friction rose from 35º to 40º. This is attributed to the fact that the number of inter-particle contacts in the 2D particle packing was only 1.94, which is lower than 4, meaning that particles were loosely packed, and the 2D model was unstable (Masanobu 1977). Thus, the inter-particle friction could barely influence the soil density.

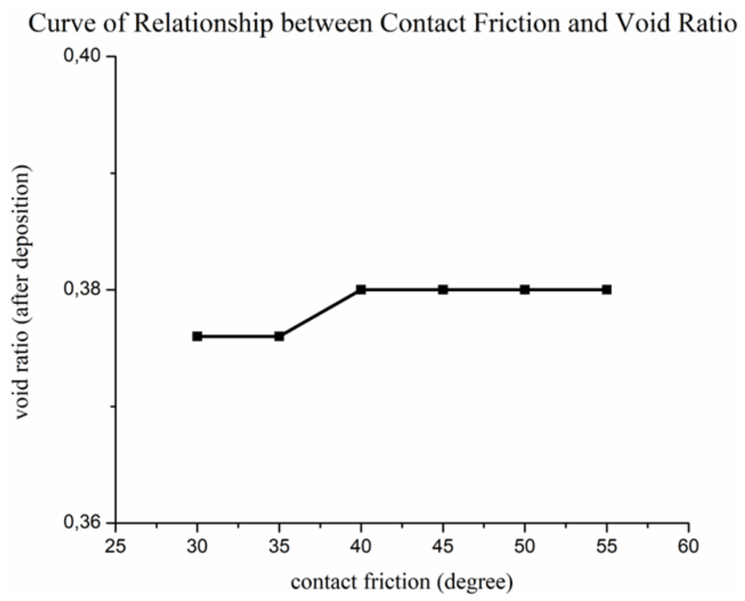


Fig 4.27 The relationship between contact friction and void ratio for 2D model deposition

Fig 4.28 shows the result of the 2D particle deposition simulated by Yade. The deposited model's volume shrank from 40mm×40mm to 40mm×27mm. Accordingly, the void ratio was reduced to 0.380 in comparison to the original 0.779.

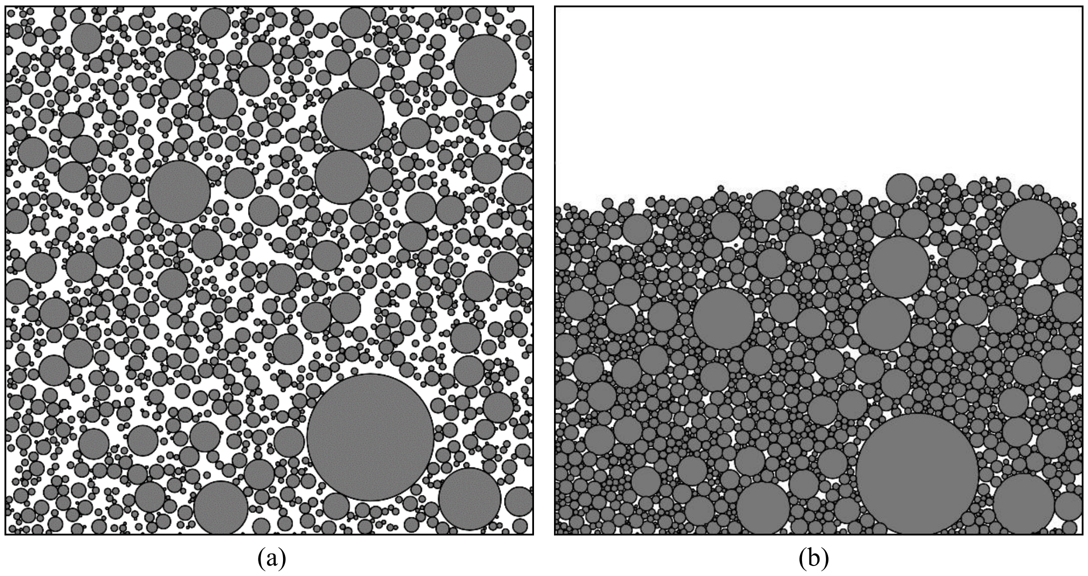


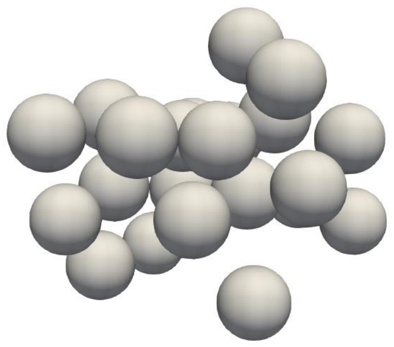
Fig 4.28 The comparison of the original model (a) and the deposited model (b)

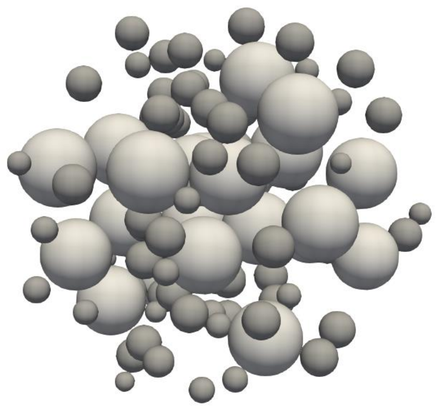
The void ratio of the 2D model decreased significantly after the deposition process compared with the original value. The void ratio of the original model was 0.779, indicating the particle packing was classified as a loose soil, and many voids existed in the model before it was deposited. In addition, particles belonging to the smallest-opening sieve accounted for 13.98% of the total mass. As a result, it can be concluded that a considerable number of small particles filled nearly all the voids between large particles during the deposition process, causing the void ratio to decrease dramatically.

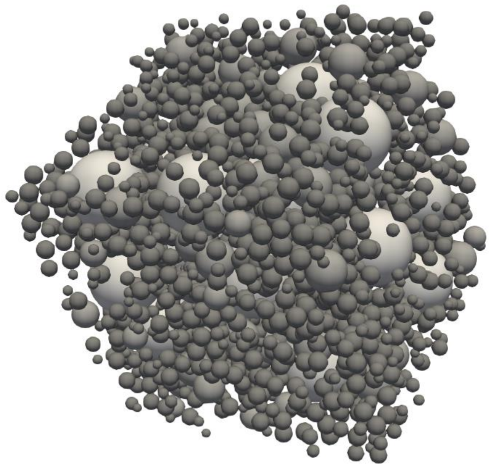
The total volume of the 3D model was $800^3 \div 1.8 = 2.84 \times 10^8$ unit^3, and the remaining parameters needed by the algorithm to generate 3D models were the same as that of 2D models. The finer percentage and the target volumes for each sieve are presented in Table 4.19. The 3D algorithm-generated model for uniformly graded soils is shown in Fig 4.29 sieve-by-sieve.

Table 4.19 Finer percentage and target volumes for each sieve in 3D model

Opening of sieve (*mm*)	4.75-2.36	2.36-1.18	1.18-0.6	0.6-0.3	0.3-0.15
Mass percentage for each sieve	8%	10%	28%	44%	15%
Target volume (*unit*^{3})	2.27×10^7	2.84×10^7	7.95×10^7	1.25×10^8	4.26×10^7


(a) Poisson round of 3D particles insertion


(b) First round of 3D particles insertion


(c) Second round of 3D particles insertion

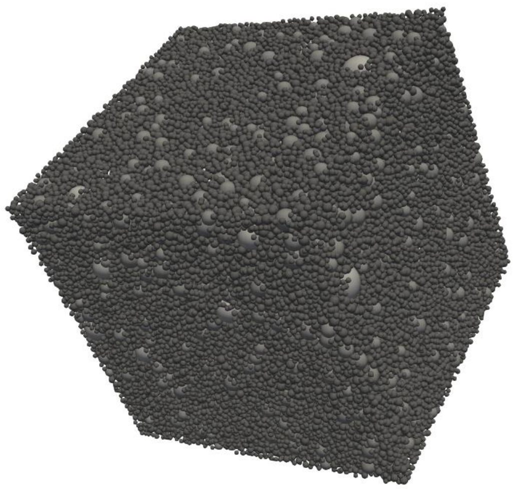

(d) Third round of 3D particles insertion

Fig 4.29 3D model generation for uniformly graded sand

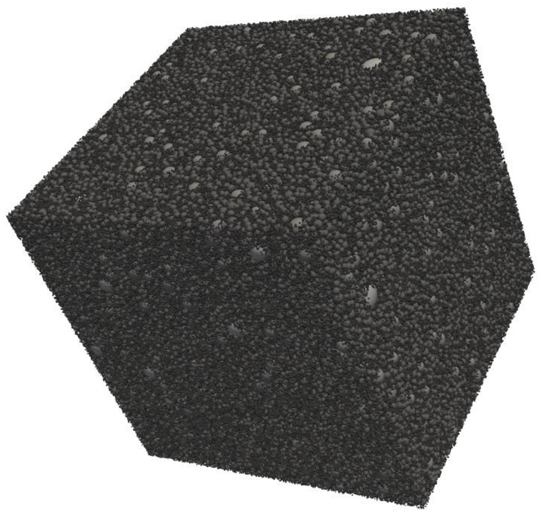

(f) Fourth round of 3D particles insertion

Fig 4.29 (Continued) 3D model generation for uniformly graded sand

Table 4.20 presents the result of the 3D model. There were 898,458 spherical particles in the model, and the running time was 142,639 seconds. Generating particles for the smallest-opening sieve is the most time-consuming, accounting for 50% of the running time.

Table 4.20 Details of 3D particle packing

Round of particle insertion	Opening of sieve mm	Mass $g \times 10^0$	Target mass $g \times 10^0$	Finer percentage	Target finer percentage	Percent difference
Poisson	4.75-2.36	4.67	4.64	100%	100%	0%
1	2.36-1.18	5.81	5.79	91.22%	92%	0.78%
2	1.18-0.6	13.92	13.92	81.31%	82%	0.69%
3	0.6-0.3	25.52	25.51	57.62%	58%	0.38%
4	0.3-0.15	8.19	8.12	13.97%	14%	0.03%

The particle-size distribution curves of the model generated by the algorithm and the real soil are shown in Fig 4.30. The particle-size distribution of the 3D model compares well with that of the real soil sample. The average error was 0.47%, and the maximum error was 0.78% and occurred in the first round of particle insertion. Errors were deemed small and acceptable. The void ratio of the model was 0.799, also close to the target of 0.800.

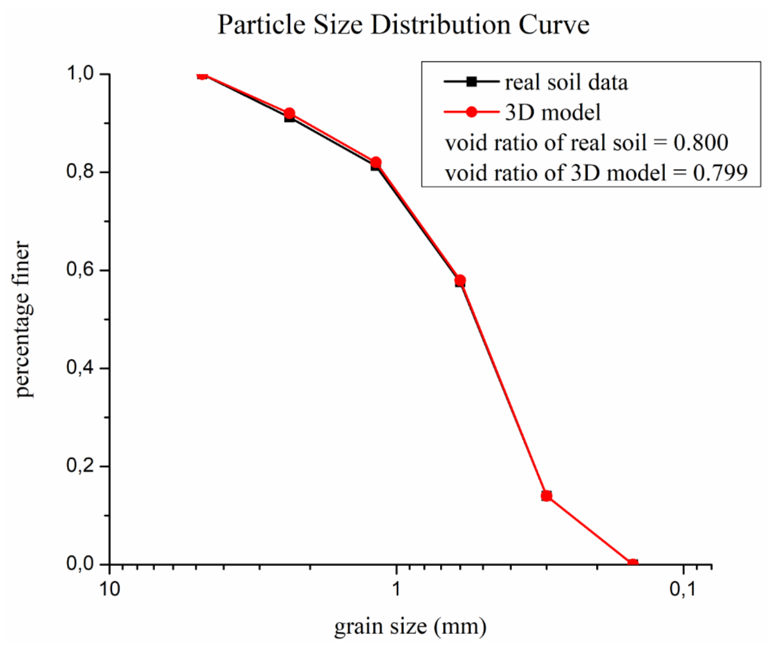


Fig 4.30 Comparison of particle-size distributions of the algorithm-generated 3D model and the real soil

The relationship between the void ratio of the deposited model and contact friction is shown in Fig 4.31. It indicates that the void ratio increased significantly as contact friction rose. The curve leveled off when the contact friction exceeded 50º, which can be considered the threshold point. Contact frictions set greater than this point can ensure the model deposition only triggers a slight reduction in void ratio. The coordination number of the particle assembly is 4.32, which is higher than 4, indicating that particles were tightly packed (Masanobu 1977). Thus, inter-particle friction can be used to adjust the void ratio.

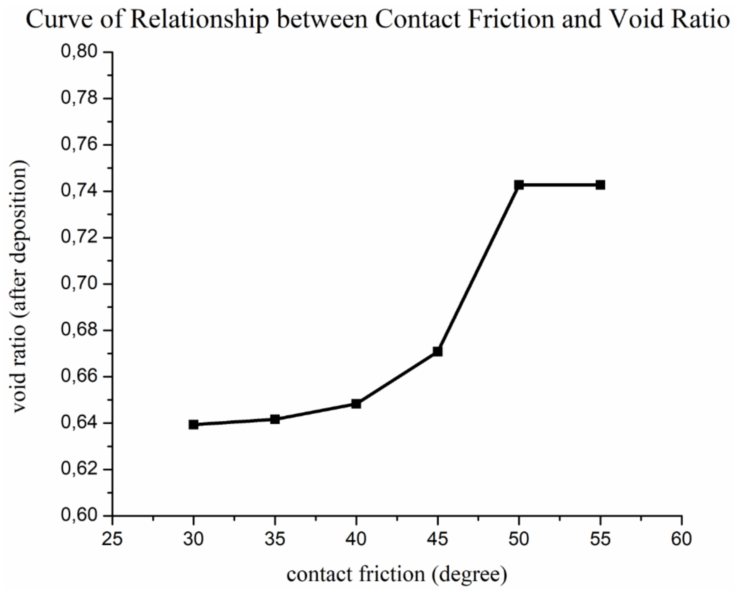


Fig 4.31 The curve of the relationship between contact friction and void ratio of 3D deposited model

The 3D model deposition is shown in Fig 4.32. The contact friction was set to 50º. The void ratio decreased from 0.799 to 0.743 after the model was deposited, and accordingly, the volume shrank from 40mm×40mm×40mm to 40mm×40mm×38.75mm. The running time was 560 seconds.

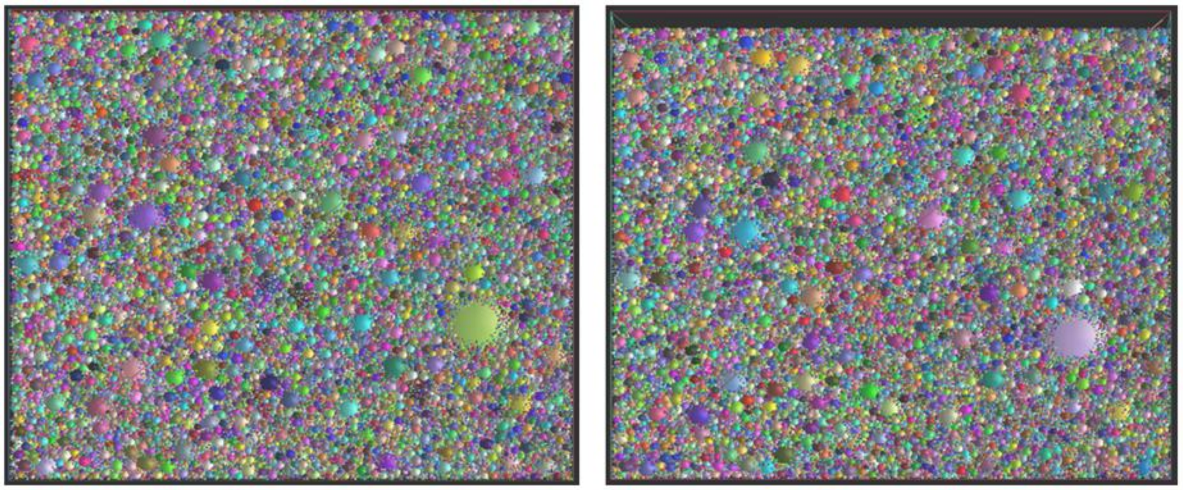

(a) (b)

Fig 4.32 The comparison of the original model (a) and the deposited model (b)

The void ratio of the uniformly graded soil model decreased more significantly compared to examples of the other 3D models. The primary reason is similar to that of the 2D uniformly graded particle assembly. The particle packing had a high void ratio and small particles that accounted for a high percentage of the total mass and collapsed further after the deposition since small particles nearly filled all voids between large particles.

4.3.5 Dense Sand Example

The fifth example aimed to create 2D and 3D models for the well-graded dense sand samples, using real soil data based on Skaggs et al. (2001). The particle-size distribution of dense soil samples is shown in Table 4.21, and the void ratio of a typical dense sand is 0.4 (Das 2003).

Table 4.21 Particle-size distribution of the dense sand sample (Skaggs et al. 2001)

Opening of sieve *mm*	3.35-2	2-1.7	1.7-1.18	1.18-1	1-0.85	0.85-0.5	0.5-0.355	0.355-0.25	0.25-0.18	0.18-0.106	0.106-0.053	0.053-0.038	<0.038
Radius units	268-160	160-136	136-94	94-80	80-68	68-40	40-28	28-20	20-14	14-8	8-4	4-3	3-2
Target finer percentage	100%	90%	82%	66%	50%	36%	22%	13%	8%	5%	3%	2%	1%
Mass percentage for each sieve	10%	8%	16%	16%	14%	14%	9%	5%	3%	2%	1%	1%	1%

The simulation parameters were as follows: the canvas size was 1500unit×1500unit/18.75mm×

18.75mm; the unit size was 0.0125 mm; the total volume of 2D particles was $1500^2 \div 1.4 = 1.6 \times 10^6$ unit^2; the cell size was 5unit×5unit. The canvas size is relatively larger than previous models since dense soils had a more comprehensive radius range. Table 4.22 shows the finer percentage and target volumes of 2D particles for every sieve. The 2D model generated by the algorithm based on well-graded dense sand data shows in Fig 4.33 sieve-by-sieve.

Table 4.22 Finer percentage and target volumes of 2D particles for every sieve

Radius *unit*	268-160	160-136	136-94	94-80	80-68	68-40	40-28	28-20	20-14	14-8	8-4	4-3	3-2
Mass percentage for each sieve	10%	8%	16%	16%	14%	14%	9%	5%	3%	2%	1%	1%	1%
Target volume $\times 10^5$ *unit*^{2}	1.6	1.28	2.56	2.56	2.24	2.24	1.44	0.8	0.48	0.32	0.16	0.16	0.16

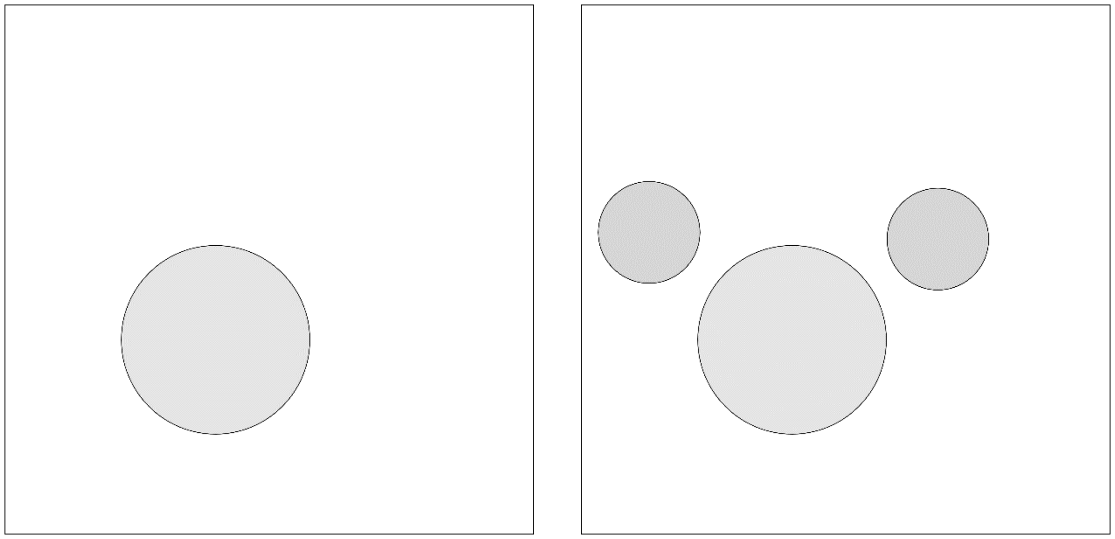
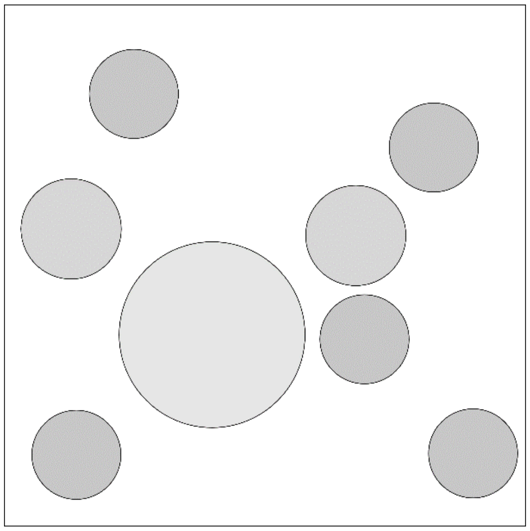
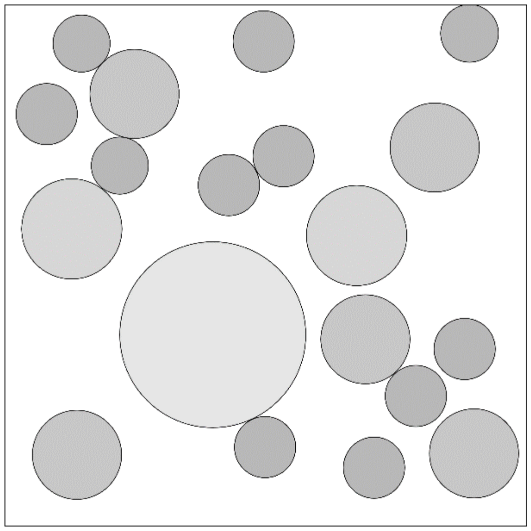
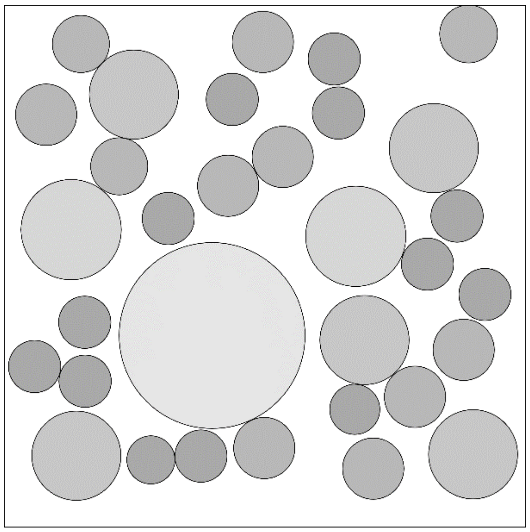
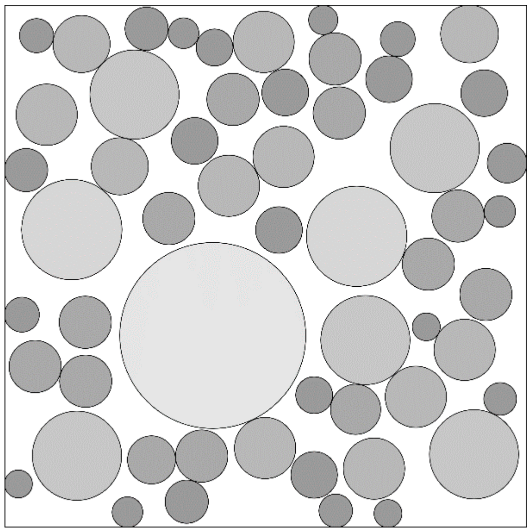

(a) Poisson round of 2D particles insertion (b) First round of 2D particles insertion

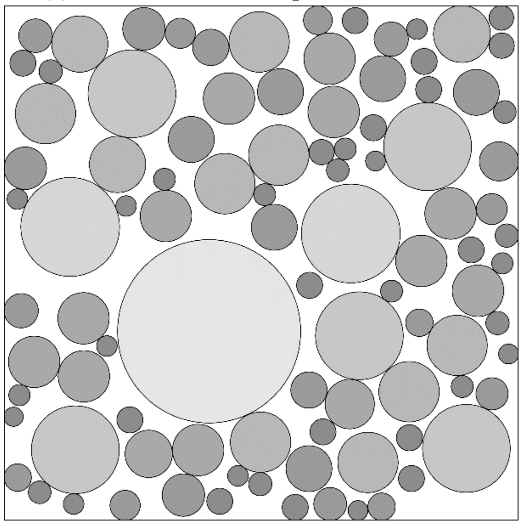
Fig 4.33 2D model generation for dense sand


(c) Second round of 2D particles insertion


(d) Third round of 2D particles insertion


(e) Fourth round of 2D particles insertion


(f) Fifth round of 2D particles insertion


(g) Sixth round of 2D particles insertion

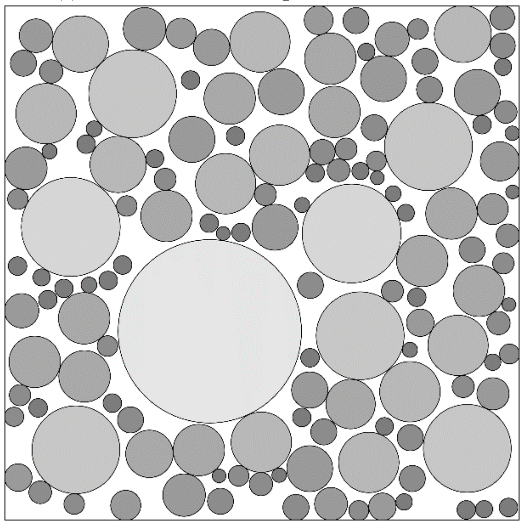
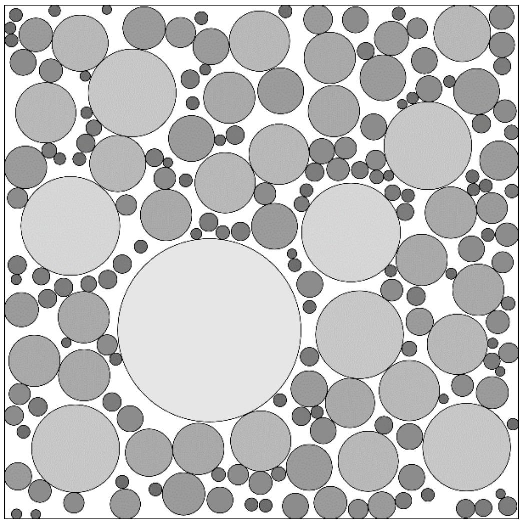
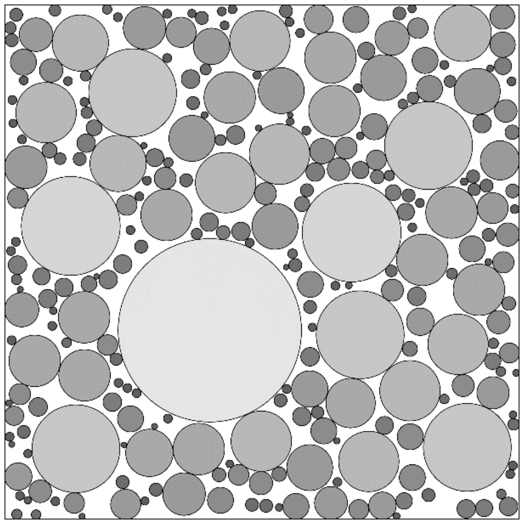
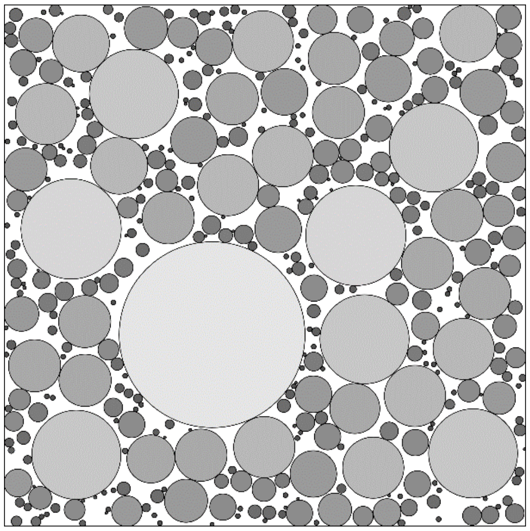
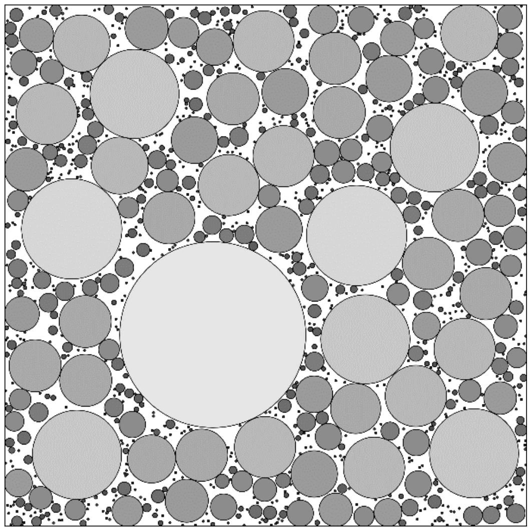

(h) Seventh round of 2D particles insertion

Fig 4.33 (Continued) 2D model generation for dense sand


(i) Eighth round of 2D particles insertion


(j) Ninth round of 2D particles insertion


(k) Tenth round of 2D particles insertion


(l) Eleventh round of 2D particles insertion

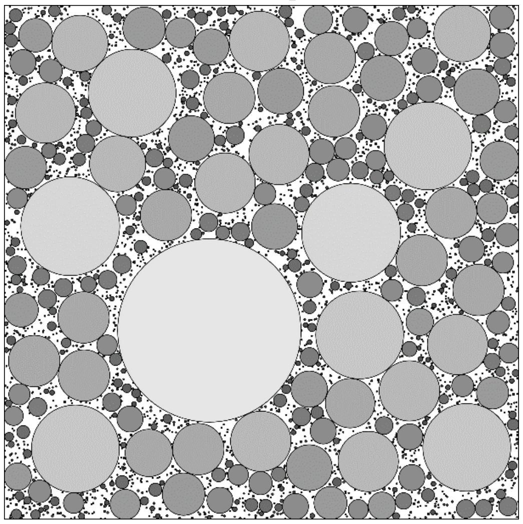

(m) Twelfth round of 2D particles insertion

Fig 4.33 (Continued) 2D model generation for dense sand

The results of the 2D model are presented in Table 4.23. There were 2,312 circular particles in the model, and the running time was 20 seconds.

Table 4.23 Details of 2D particle packing

Round of particle insertion	Opening of sieves mm	Mass g×10^{-3}	Target mass g×10^{-3}	Finer percentage	Target finer percentage	Percent difference
Poisson	3.35-2	57.0	40.9	100%	100%	0%
1	2-1.7	33.2	32.7	86.66%	90%	3.34%
2	1.7-1.18	65.6	65.5	78.90%	82%	1.10%
3	1.18-1	65.9	65.5	63.57%	66%	2.43%
4	1-0.85	57.5	57.3	48.17%	50%	1.83%
5	0.85-0.5	57.4	57.3	34.73%	36%	1.27%
6	0.5-0.355	37.2	36.8	21.32%	22%	0.68%
7	0.355-0.25	2.08	2.04	12.62%	13%	0.38%
8	0.25-0.18	12.4	12.2	7.75%	8%	0.25%
9	0.18-0.106	8.21	8.19	4.85%	5%	0.15%
10	0.106-0.053	4.13	4.09	2.93%	3%	0.07%
11	0.053-0.038	4.13	4.09	1.97%	2%	0.03%
12	<0.038	4.29	4.09	1.00%	1%	0%

The particle-size distribution curves of the 2D model and the real soil are shown in Fig 4.34. The algorithm-generated model agrees with the actual soil sample in terms of particle-size distribution. The average error was 0.81%, and the maximum error was 3.34% and occurred in the first round of the particle insertion. The reason is that the volume of particles belonging to the largest opening sieve significantly surpassed the target. Due to the considerable individual particle volume, the total volume exceeded the target volume by more than 2% after the algorithm created the last particle for this sieve. Nevertheless, without this particle, the volume would not have reached the target. The recursive call presented in section 3.4.3 was introduced to avoid this problem by shrinking the radius range. However, even setting the particle radius as the minimum value in the range still caused the volume to be beyond the acceptable limit. The differences between particles in the remaining sieves and real soils were regarded as acceptable.

The void ratio of the 3D model was 0.343, which is slightly lower than the target of 0.400, mainly due to the volume of particles in the largest opening sieve that exceeded the target.

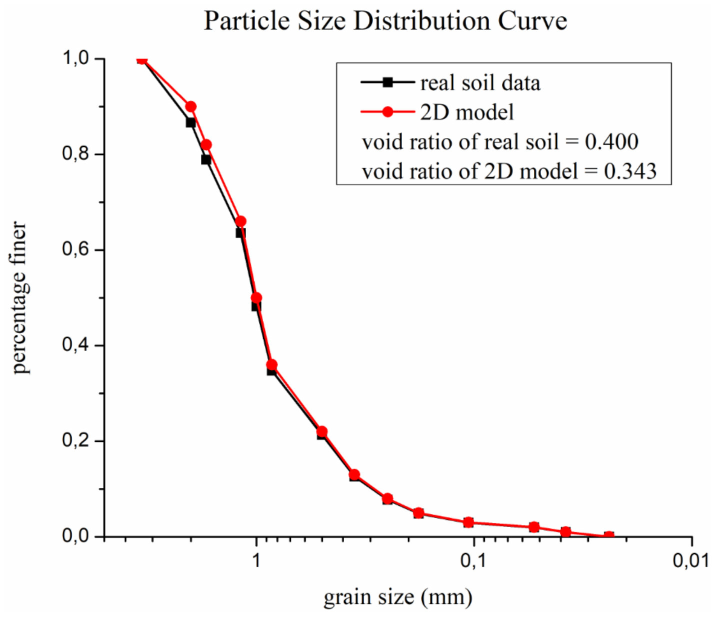


Fig 4.34 Comparison of particle-size distributions of the algorithm-generated 2D model and the real soil

Contact friction barely influenced the void ratio of the deposited model, as shown in Fig 4.35. The void ratio only slightly increased when contact friction rose from 40º to 45º. The reason is that the coordination number of the 2D particle assembly was 2.73, which is lower than 4, indicating that this particle packing was unstable and not tightly packed (Masanobu 1977). Thus, inter-particle friction cannot impact the void ratio by much.

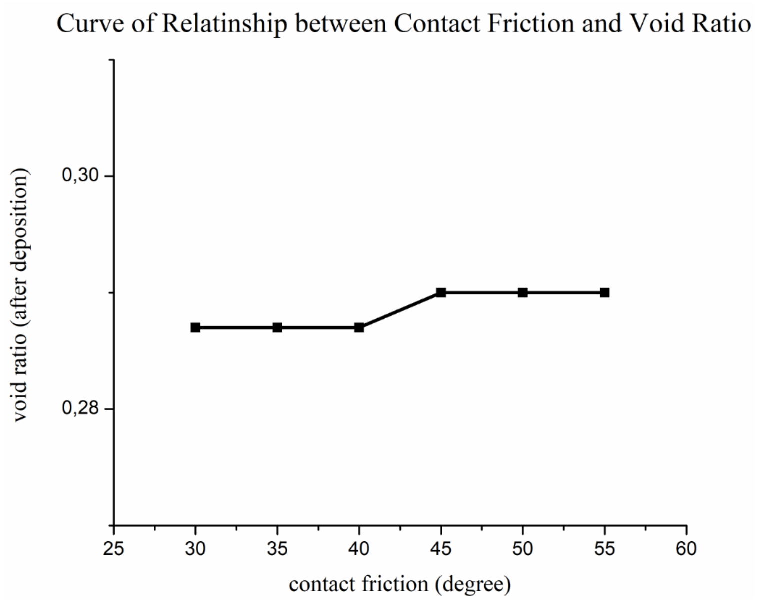


Fig 4.35 The curve of the relationship between contact friction and void ratio of 2D deposited model

The result of the 2D model deposition is shown in Fig 4.36. The void ratio decreased moderately to 0.290 compared with the original value of 0.343. Correspondingly, the volume shrank from

18.75mm×18.75mm to 18.75mm×16.93mm. The running time was 24 seconds.

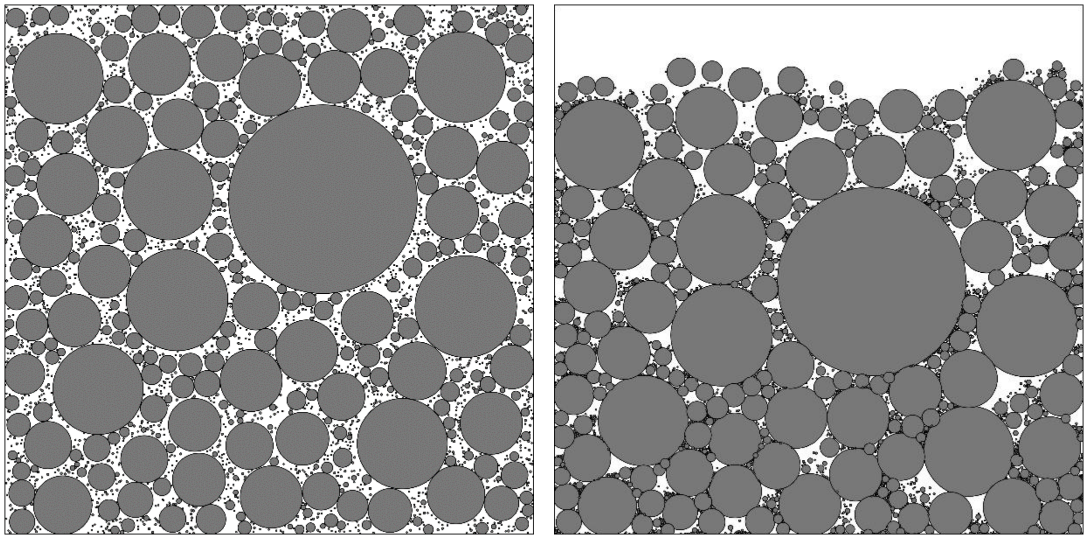

(a) (b)

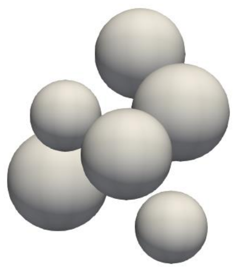
Fig 4.36 The comparison of the original model (a) and the deposited model (b)

The void ratio decreased moderately compared with the original value. Compared to other 2D model deposition simulations presented before, the void ratio reduction was smaller in this case. The main reason is that the initial void ratio was low, meaning that the particle assembly was dense before the deposition simulation. In addition, the number of small particles in the last five sieves was low, accounting for just 8% of the total mass. After the model deposition, small particles did not fill all the voids, causing only minor changes in the void ratio. Additionally, the 2D dense soil model had the highest coordination number of all the 2D examples. Thus, relatively strong interlocking between particles also prevented the particles from collapsing.

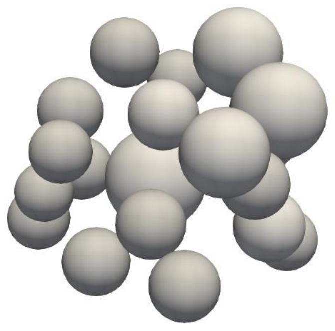
The canvas size for the 3D model was set to 1500 unit×1500unit×1500 unit/18.75mm×18.75mm ×18.75mm, and the volume of 3D particles was $1500^3 \div 1.4 = 2.4 \times 10^9$ unit^3. The remaining parameters the algorithm requires to build 3D models were the same as in the 2D model. Table 4.24 shows the percent finer and the target volumes of the 3D model for every sieve. The 3D algorithm-generated model for well-graded dense sand is shown in Fig 4.37 sieve-by-sieve.

Table 4.24 Finer percentage and target volumes of 3D particles for every sieve

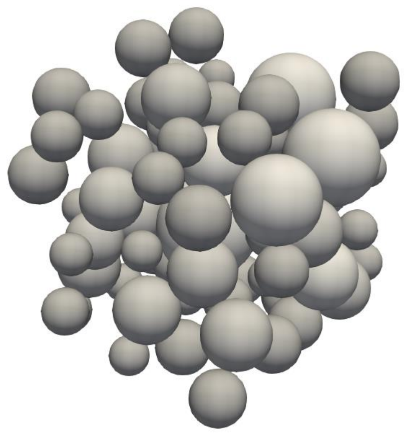
Radius *unit*	268-160	160-136	136-94	94-80	80-68	68-40	40-28	28-20	20-14	14-8	8-4	4-3	3-2
Mass percentage for each sieve	10%	8%	16%	16%	14%	14%	9%	5%	3%	2%	1%	1%	1%
Target volume $\times$ *10^8 unit^3*	2.4	1.92	3.84	3.84	3.36	3.36	2.16	1.2	0.72	0.48	0.24	0.24	0.24



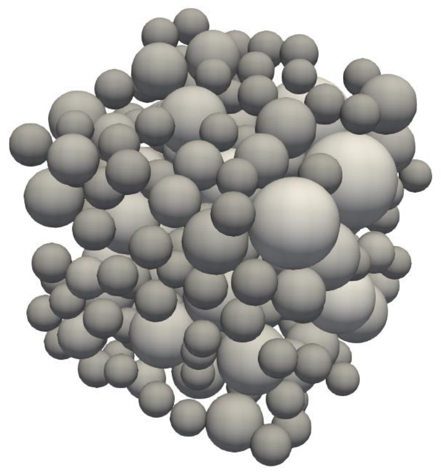
(a) Poisson round of 3D particles insertion



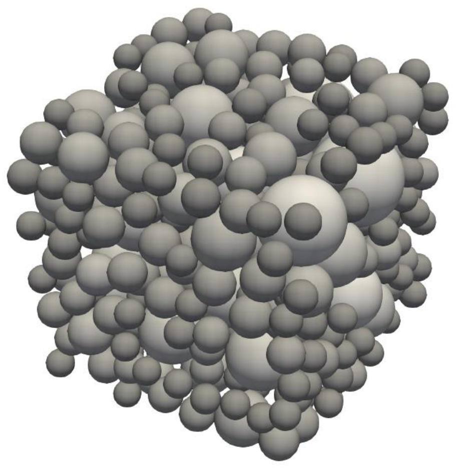
(b) First round of 3D particles insertion



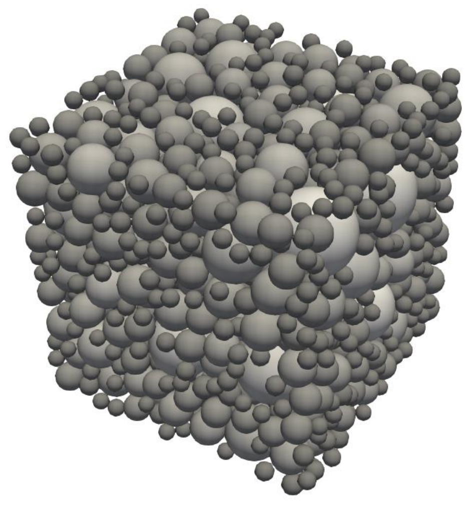
(c) Second round of 3D particles insertion



(d) Third round of 3D particles insertion

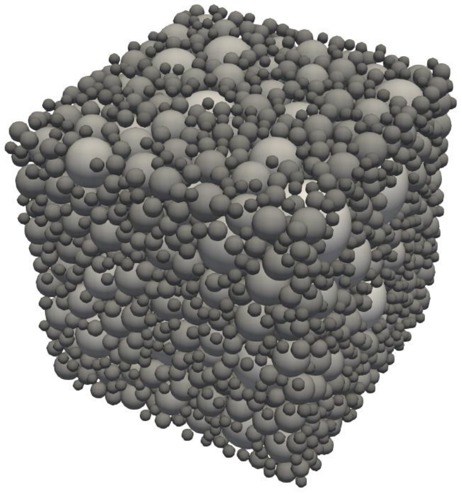


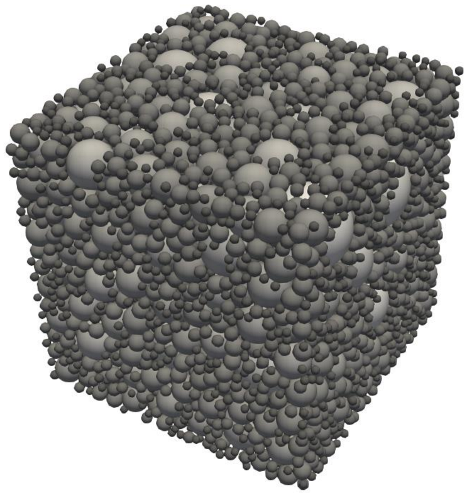
(e) Fourth round of 3D particles insertion

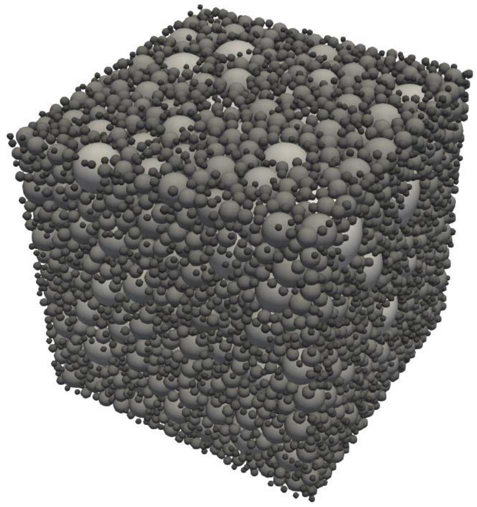


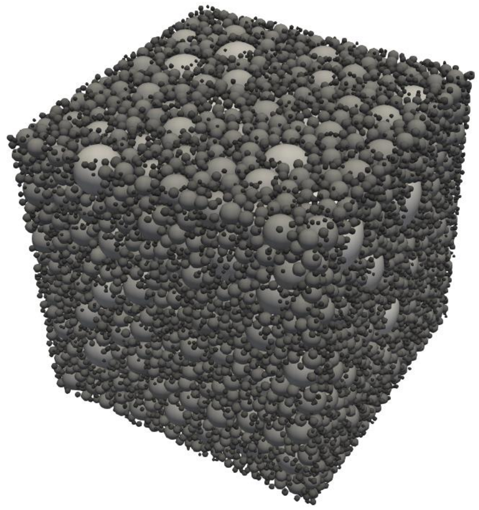
(f) Fifth round of 3D particles insertion

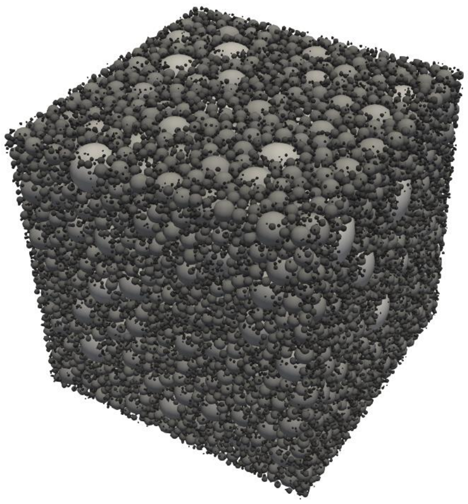
Fig 4.37 3D model generation for dense sand


(g) Sixth round of 3D particles insertion


(h) Seventh round of 3D particles insertion


(i) Eighth round of 3D particles insertion


(j) Ninth round of 3D particles insertion


(k) Tenth round of 3D particles insertion

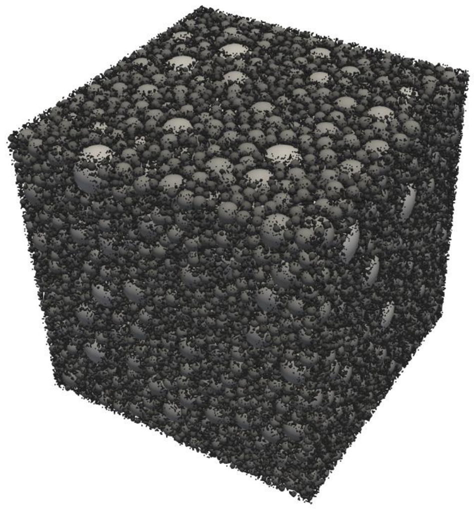

(l) Eleventh round of 3D particles insertion

Fig 4.37 (Continued) 3D model generation for dense sand

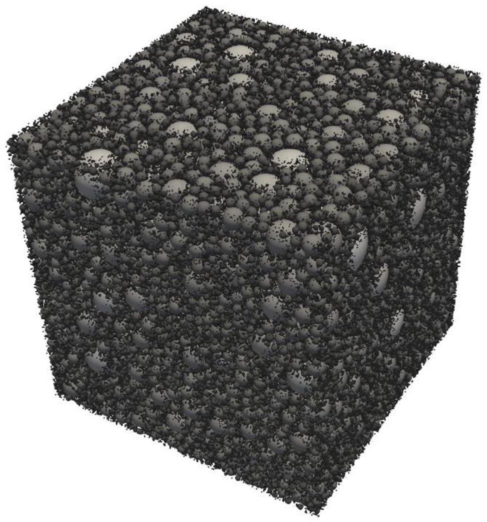

(m) Twelfth round of 3D particles insertion

Fig 4.37 (Continued) 3D model generation for dense sand

Details of the 3D model are presented in Table 4.25. There were 998,390 spherical particles in the model. The running time was 132,171 seconds. Generating particle generation belonging to the smallest opening sieve was the most time-consuming step, accounting for roughly 35% of the total running time.

Table 4.25 Details of 3D particle packing

Round of particles insertion	Opening of sieves mm	Mass $g\times10^{-2}$	Target mass $g\times10^{-2}$	Finer percentage	Target finer percentage	Percent difference
Poisson	3.35-2	77.82	76.79	100%	100%	0%
1	2-1.7	66.66	61.43	89.99%	90%	0.01%
2	1.7-1.18	125.9	122.87	81.42%	82%	0.58%
3	1.18-1	123.1	122.87	65.63%	66%	0.37%
4	1-0.85	107.61	107.51	49.45%	50%	0.55%
5	0.85-0.5	107.55	107.51	35.57%	36%	0.43%
6	0.5-0.355	69.13	69.15	21.74%	22%	0.26%
7	0.355-0.25	38.40	38.39	12.85%	13%	0.15%
8	0.25-0.18	23.04	23.04	7.92%	8%	0.08%
9	0.18-0.106	15.36	15.35	4.95%	5%	0.05%
10	0.106-0.053	7.68	7.67	2.98%	3%	0.02%
11	0.053-0.038	7.68	7.67	1.99%	2%	0.01%
12	<0.038	7.79	7.67	1%	1%	0%

The particle-size distribution curves of the algorithm-generated particle packing and the actual soil are shown in Fig 4.38. The particle size distribution of the 3D model compares well with that of the actual soil sample. The average error was 0.21%, and the maximum error was 0.58%, which occurred in the second round of particle insertion. The generation of particles for the sieve with an opening size ranging from 2mm to 1.7mm caused errors. The particle volume of this sieve exceeded the target by more than 2%. The main reason is that particles of this sieve accounted for just 8% of the total mass but had a significant individual particle volume. As such, when the algorithm inserted the last particle for this sieve into the model, the volume considerably surpassed the target. However, without this particle, the volume would not have reached the target. Shrinking the radius range could have alleviated this problem, but the error was still out of the acceptable range. Errors of particle percentages for the remaining sieves were deemed as small and acceptable. The void ratio of the model was 0.382, which is close to the target of 0.400.

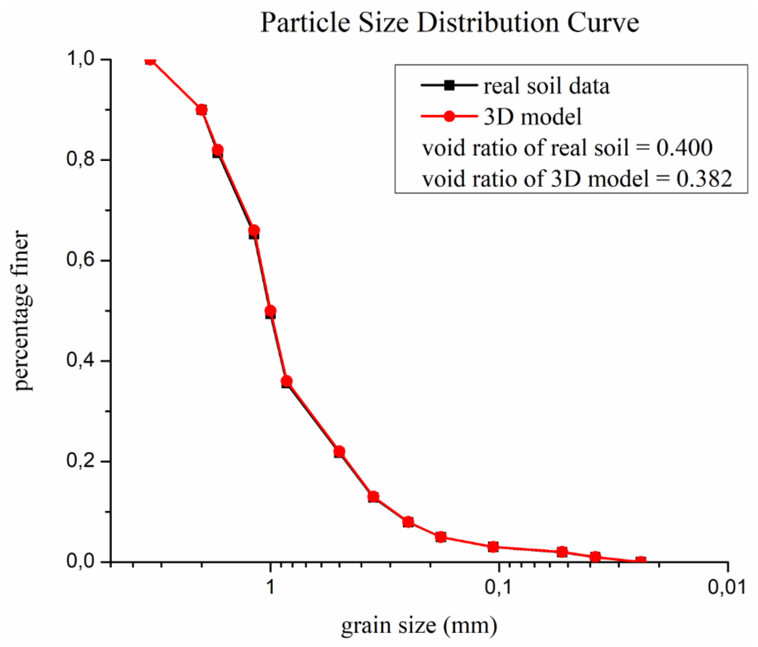


Fig 4.38 Comparison of particle-size distributions of the algorithm-generated 3D model and the real soil

Since the 3D particle packing contained a large number of particles, the simplification method used in the mixture sand example was employed. The particle assembly without the particles in the smallest opening sieve was imported into Yade. This significantly reduces the running time of the Yade simulation. Omitting the smallest opening sieve particles, which accounted for 1% of the total mass in this case, was deemed acceptable.

The curve showing the relationship between the void ratio of the deposited model and contact friction is presented in Fig 4.39. The void ratio initially increased with the rising contact friction. Then the curve started to flatten, and the void ratio remained constant once the contact friction exceeded 45º, which can be regarded as the threshold point. Using a contact friction greater than this point ensures the model only experiences a slight decrease in void ratio during the deposition process. After the deposition process, the coordination number of the model was 5.72, which is higher than 4, meaning the particles were closely packed (Masanobu 1977). Thus, inter-particle friction could be used to adjust the void ratio.

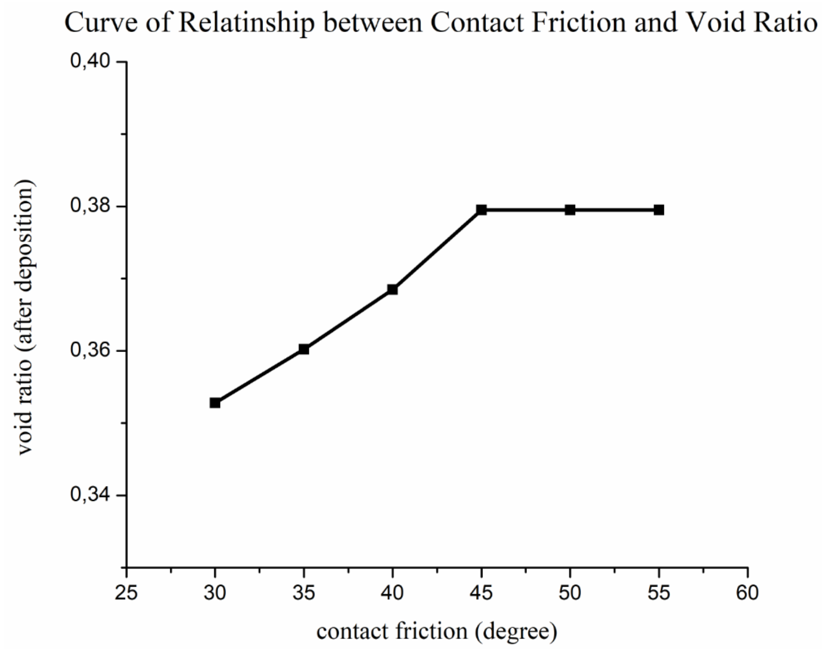


Fig 4.39 The relationship between contact friction and void ratio of 3D deposited model

The 3D model deposition simulated by Yade is shown in Fig 4.40. The contact friction was set to 45 degrees. The void ratio of the 3D deposited model decreased slightly from 0.382 to the original value of 0.380. The model volume shrank from 18.75mm×18.75mm×18.75mm to 18.75mm× 18.75mm×18.7125 mm. The running time was 892 seconds.

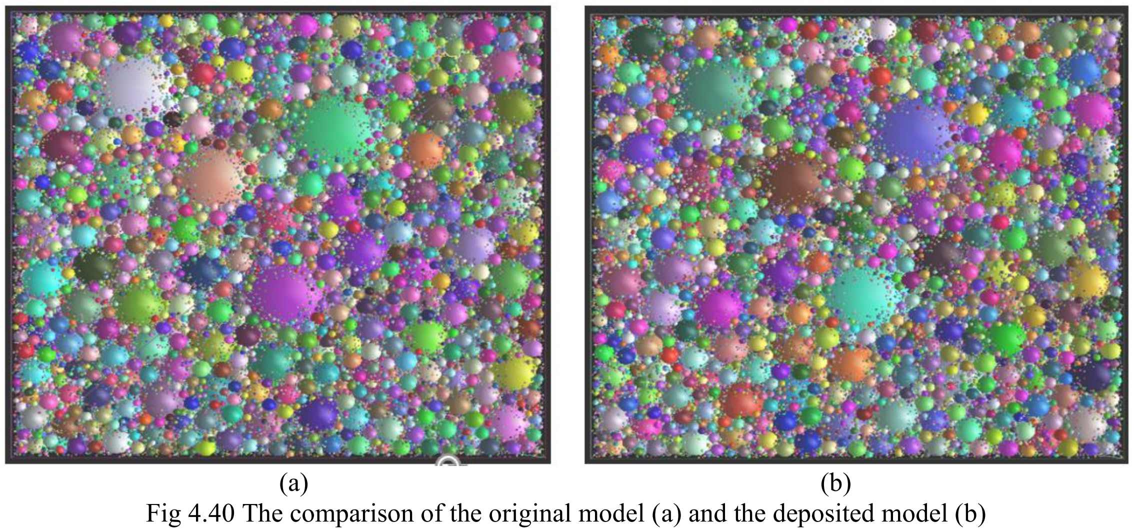


Fig 4.40 The comparison of the original model (a) and the deposited model (b)

4.3.6 Loose Sand Example

In this example, 2D and 3D models simulating well-graded loose sand samples are generated. The particle-size distribution of the soil sample is shown in Table 4.26 and is taken from Skaggs et al. (2001). The void ratio of typical loose sands is 0.65, according to Das (2003).

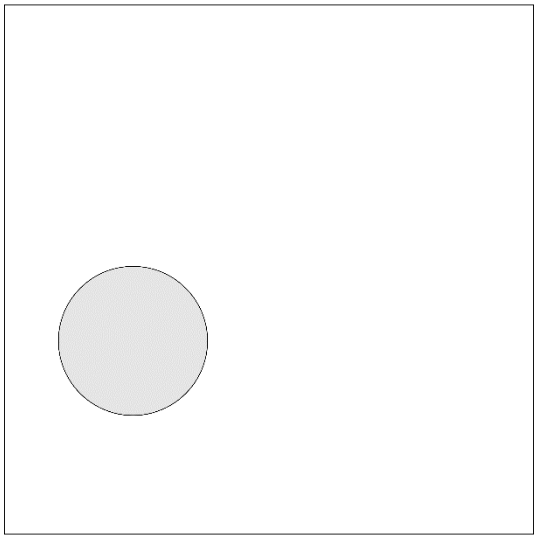
Table 4.26 Particle-size distribution of the loose sand sample (Skaggs et al. 2001)

Opening of sieves *mm*	3.35-2	2-1.7	1.7-1.18	1.18-1	1-0.85	0.85-0.5	0.5-0.355	0.355-0.25	0.25-0.18	0.18-0.106	0.106-0.053	0.053-0.038	<0.038
Radius units	268-160	160-136	136-94	94-80	80-68	68-40	40-28	28-20	20-14	14-8	8-4	4-3	3-2
Ideal finer percentage	100%	90%	82%	66%	50%	36%	22%	13%	8%	5%	3%	2%	1%
Mass percentage for each sieve	10%	8%	16%	16%	14%	14%	9%	5%	3%	2%	1%	1%	1%

For the 2D model, the canvas size was 1500unit×1500unit/18.75mm×18.75mm, and the unit size was 0.0125 mm. The target total volume was $1500^2 \div 1.65 = 1.36 \times 10^6$ unit^2. The cell size for the void-filling process was 5unit×5unit. Table 4.27 shows the target volumes of 2D particles for every sieve. The 2D model generated by the algorithm targeting well-graded loose sands is shown in Fig 4.41 sieve-by-sieve.

Table 4.27 Finer percentage and target volumes of 2D particles for every sieve

Radius *unit*	268-160	160-136	136-94	94-80	80-68	68-40	40-28	28-20	20-14	14-8	8-4	4-3	3-2
Mass percentage for each sieve	10%	8%	16%	16%	14%	14%	9%	5%	3%	2%	1%	1%	1%
Target volume $\times 10^5$ $unit^2$	1.36	1.08	2.18	2.18	1.9	1.9	1.22	0.68	0.408	0.272	0.136	0.136	0.136


(a) Poisson round of 3D particles insertion

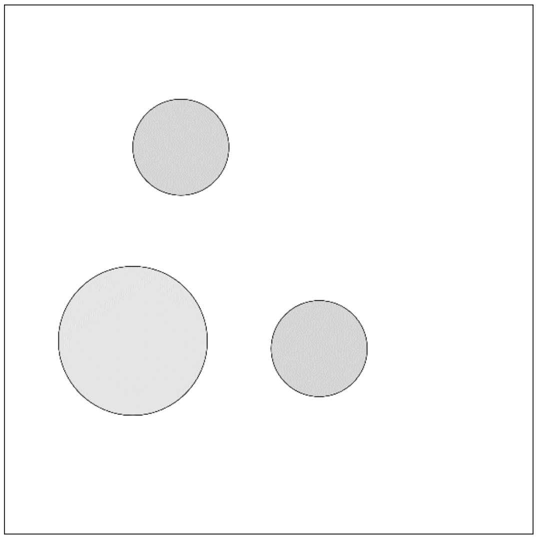
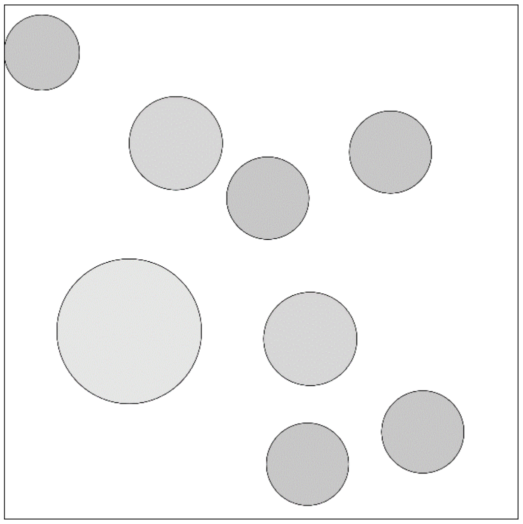
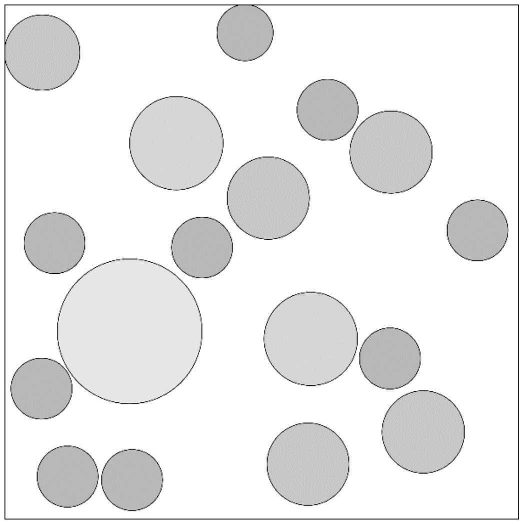
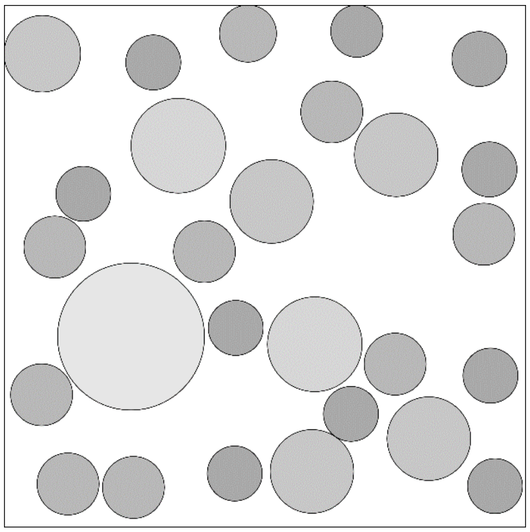
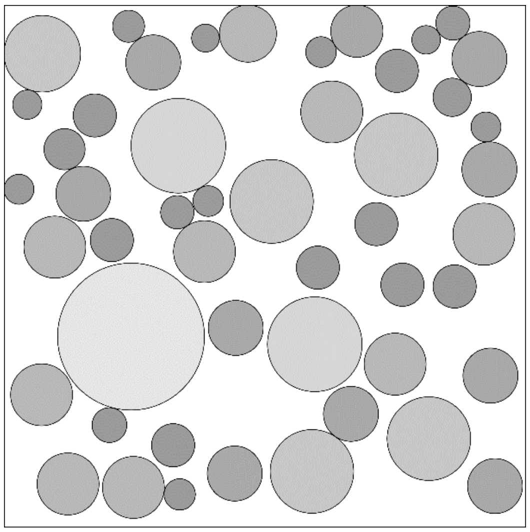

(b) First round of 3D particles insertion

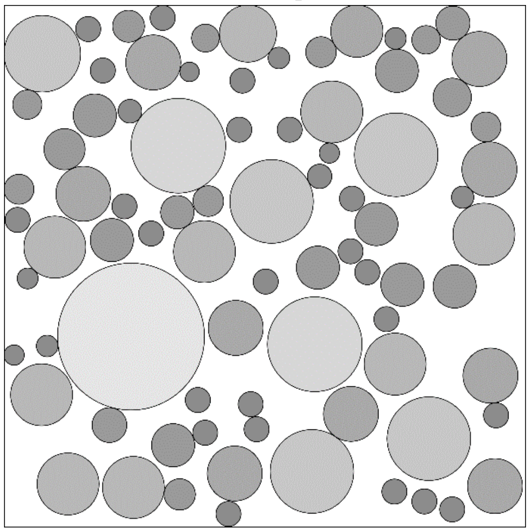
Fig 4.41 2D model generation for loose sand


(c) Second round of 3D particles insertion


(d) Third round of 3D particles insertion


(e) Fourth round of 3D particles insertion


(f) Fifth round of 3D particles insertion


(g) Sixth round of 3D particles insertion

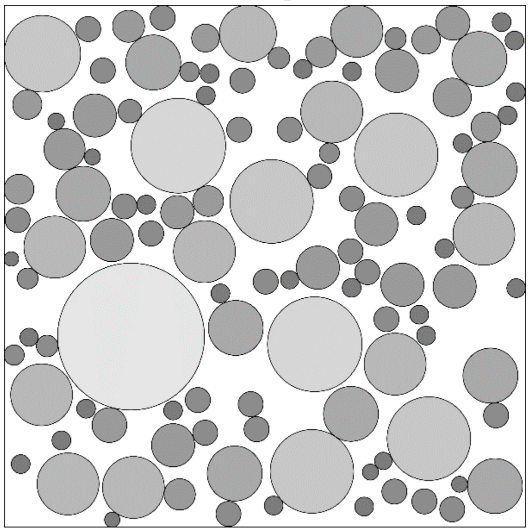
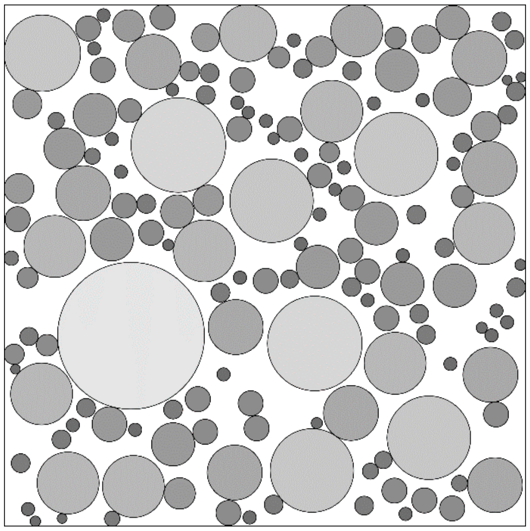
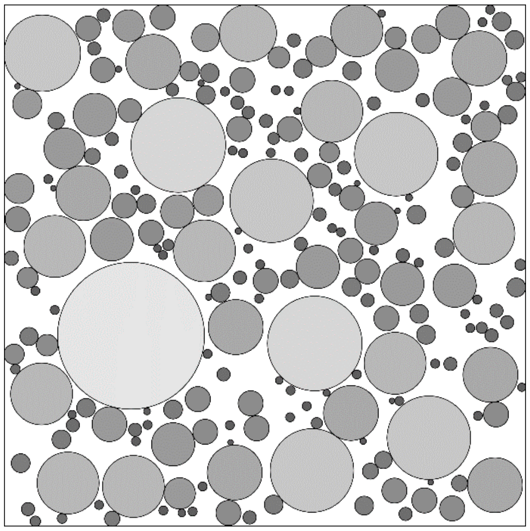
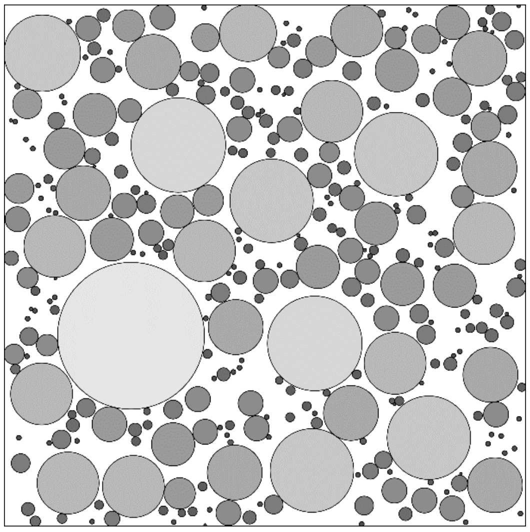
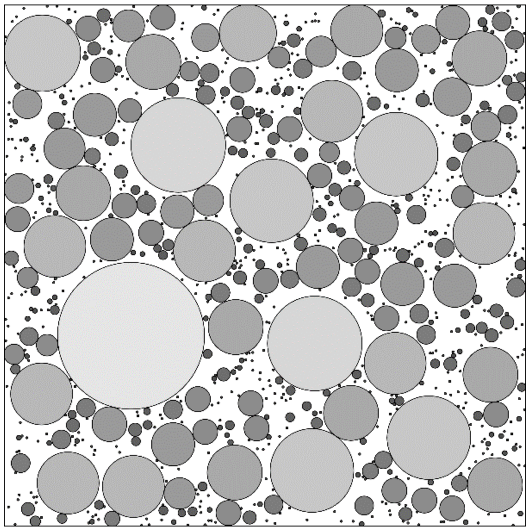

(h) Seventh round of 3D particles insertion

Fig 4.41 (Continued) 2D model generation for loose sand


(i) Eighth round of 3D particles insertion


(j) Ninth round of 3D particles insertion


(k) Tenth round of 3D particles insertion


(l) Eleventh round of 3D particles insertion

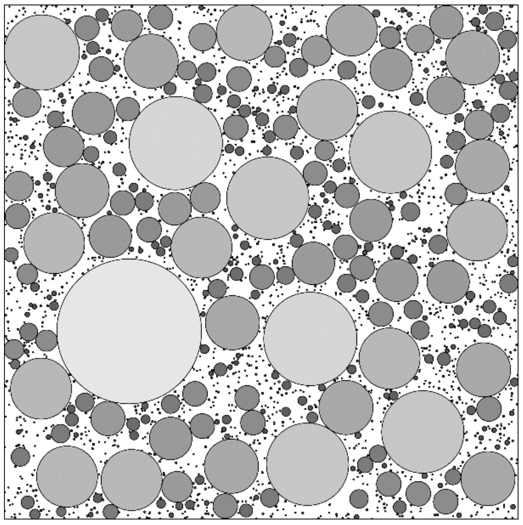

(m) Twelfth round of 2D particles insertion

Fig 4.41 (Continued) 2D model generation for loose sand

The results of the 2D model are shown in Table 4.28. There were 1,939 circular particles in the model, and the running time was 19 seconds.

Table 4.28 Details of 2D particle packing

Round of particles insertion	Opening of sieves mm	Mass $g\times10^{-3}$	Target mass $g\times10^{-3}$	Finer percentage	Target finer percentage	Present difference
Poisson	3.35-2	35.64	34.75	100%	100%	0%
1	2-1.7	29.62	27.80	89.99%	90%	0.01%
2	1.7-1.18	55.80	55.60	81.51%	82%	0.49%
3	1.18-1	56.12	55.60	65.69%	66%	0.31%
4	1-0.85	49.47	48.65	49.79%	50%	0.21%
5	0.85-0.5	49.01	48.65	35.77%	36%	0.23%
6	0.5-0.355	31.59	31.27	21.88%	22%	0.12%
7	0.355-0.25	17.47	17.37	12.92%	13%	0.08%
8	0.25-0.18	10.44	10.42	7.97%	8%	0.03%
9	0.18-0.106	7.03	6.95	5%	5%	0%
10	0.106-0.053	3.49	3.47	3%	3%	0%
11	0.053-0.038	3.51	3.47	2%	2%	0%
12	<0.038	3.64	3.47	1%	1%	0%

The particle-size distribution curves of the algorithm-generated model and the real soil are shown in Fig 4.42. The particle-size distribution of the 2D model agrees with that of the real soil sample. The average error was 0.12%, and the maximum error was 0.49%, which occurred in the second round of the particle insertion. Errors were considered small and acceptable. The void ratio of the model was 0.625, close to the target of 0.650.

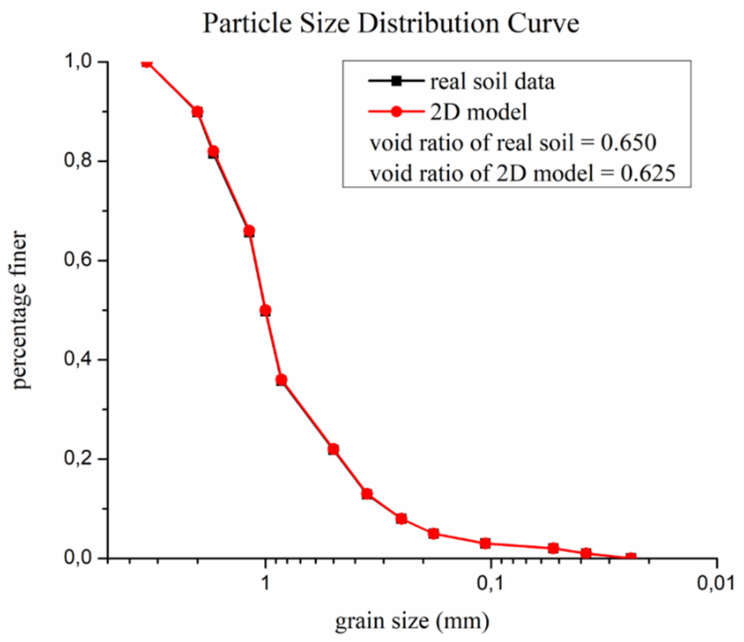


Fig 4.42 Comparison of particle-size distributions of the algorithm-generated 2D model and the real soil

The curve describing the relationship between contact friction and the void ratio of the 2D model is shown in Fig 4.43. The void ratio increased slightly along with an increase in contact friction. Contact friction could not influence the model's void ratio by much. This is due to the model having a low coordination number of 1.79, indicating the soil was loosely packed. As such, since inter-particle friction relies on particle contacts, it only had a limited impact on the soil's density.

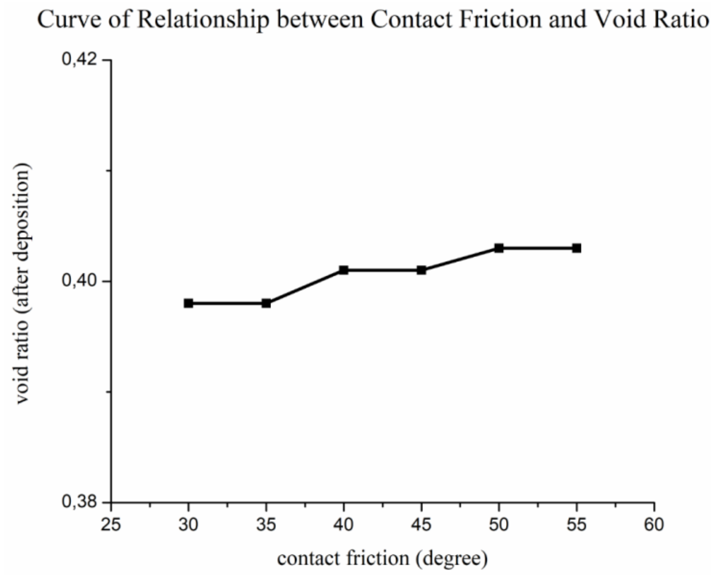


Fig 4.43 The relationship between contact friction and void ratio of 2D deposited model

The result of the 2D model deposition is shown in Fig 4.44. The model's void ratio decreased from 0.625 to 0.403 after the deposition. Accordingly, the volume contracted from 18.75mm×18.75mm to 18.75mm×14.32mm. The running time was 39 seconds.

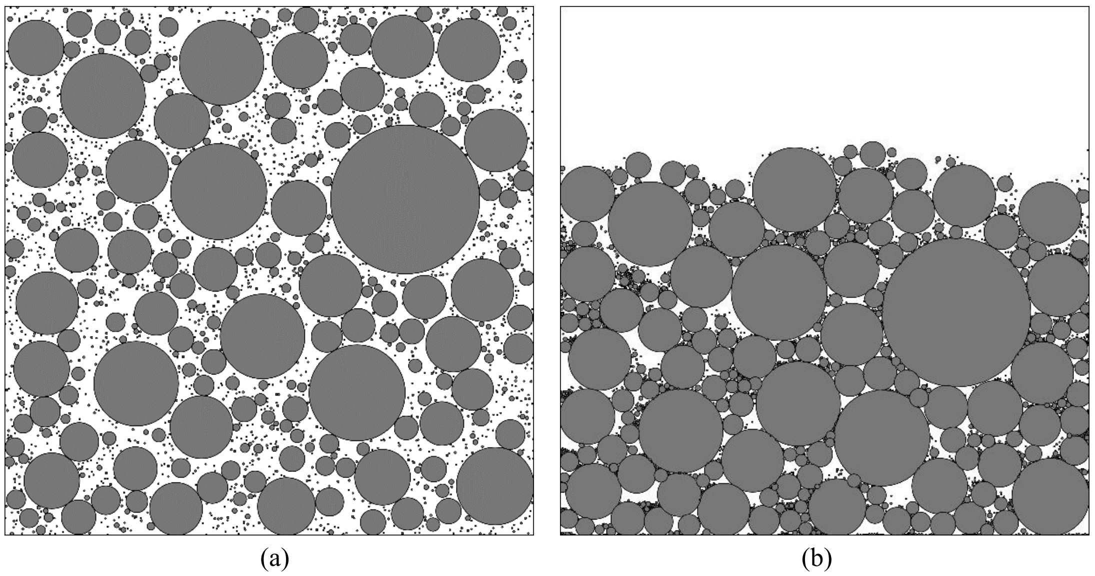


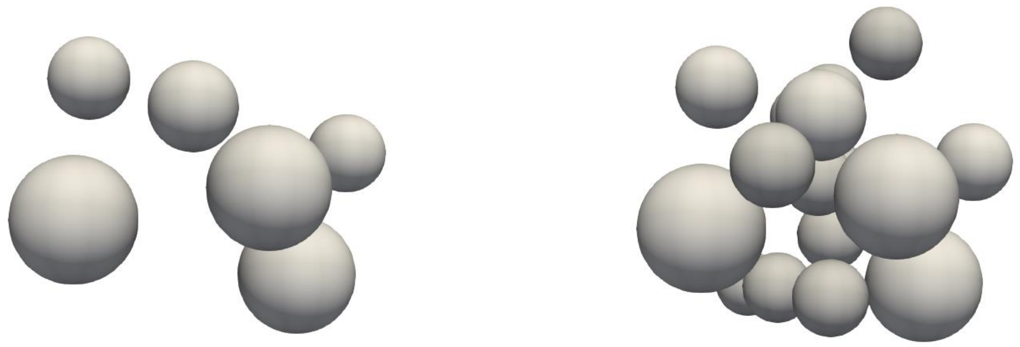
Fig 4.44 The comparison of the original model (a) and the deposited model (b)

The void ratio decreased sharply compared with the original value after the deposition simulation. The main reason is that compared to the dense soil model, the loose soil model had more voids, causing the void ratio to reduce considerably after the deposition. In addition, since 2D particles were loosely packed and had a low coordination number, weak interlocking between the circular particles contributed to the void ratio reduction.

The canvas size for the 3D model was set to 1500unit×1500unit×1500unit/18.75mm×18.75mm ×18.75mm, and the total volume of the model is $1500^3 \div 1.65 = 2.05 \times 10^9$ unit^3. The remaining parameters used in the algorithm were the same as in the 2D model. Table 4.29 shows the percent passing and the target volumes for every sieve. The 3D algorithm-generated model for well-graded loose sand is shown in Fig 4.45 sieve-by-sieve.

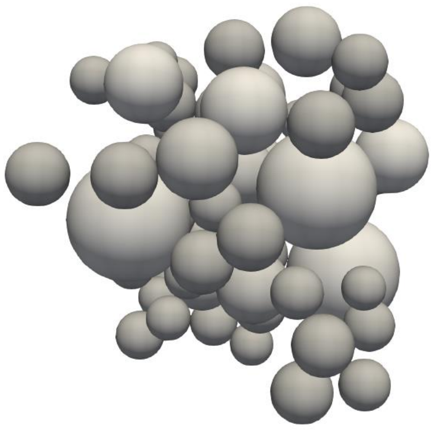
Table 4.29 Finer percentage and target volumes of 3D particles for every sieve

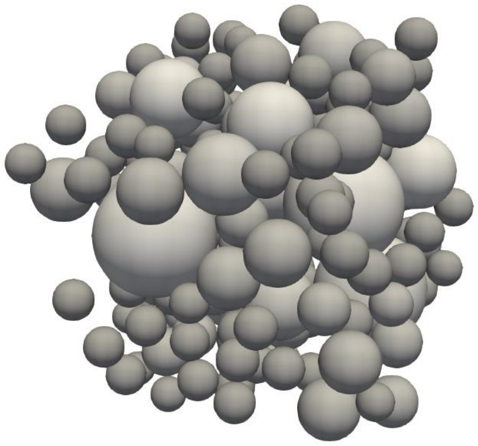
Radius *unit*	268-160	160-136	136-94	94-80	80-68	68-40	40-28	28-20	20-14	14-8	8-4	4-3	3-2
Mass percentage for each sieve	10%	8%	16%	16%	14%	14%	9%	5%	3%	2%	1%	1%	1%
Target volume × 10^8 *unit*^{3}	2.05	16.4	3.28	3.28	2.87	2.87	1.85	1.03	0.615	0.410	0.205	0.205	0.205

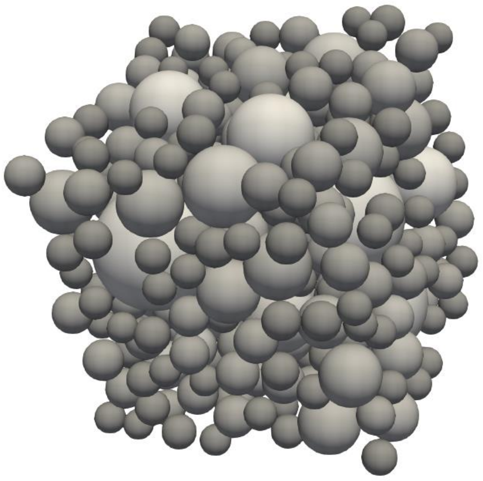


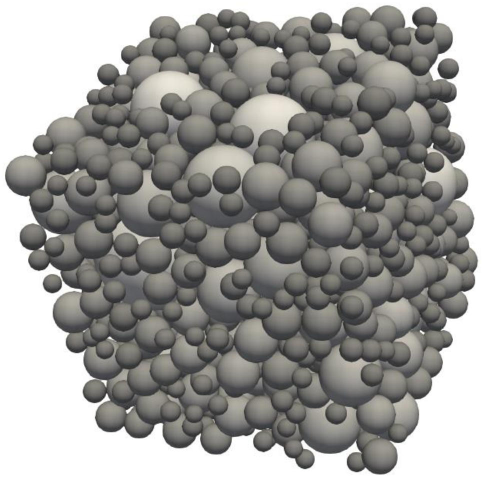
(a) Poisson round of 3D particles insertion (b) First round of 3D particles insertion

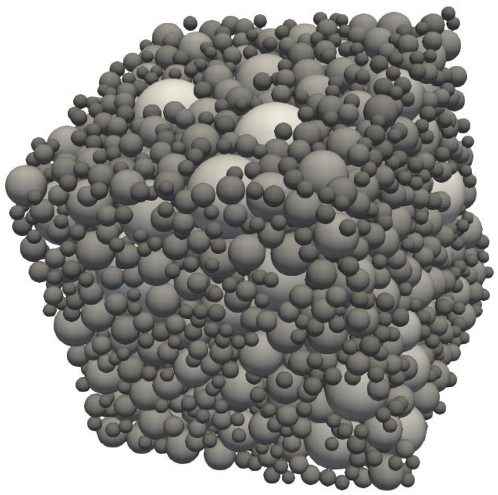
Fig 4.45 3D model generation for loose sand


(c) Second round of 3D particles insertion


(d) Third round of 3D particles insertion


(e) Fourth round of 3D particles insertion


(f) Fifth round of 3D particles insertion


(g) Sixth round of 3D particles insertion

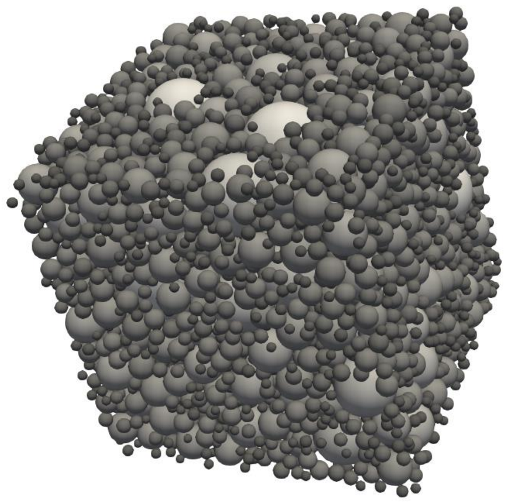
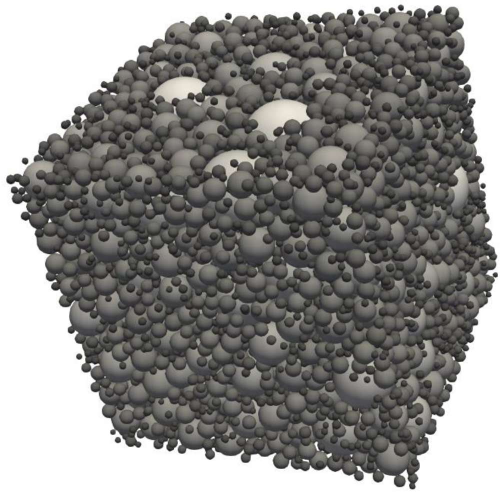
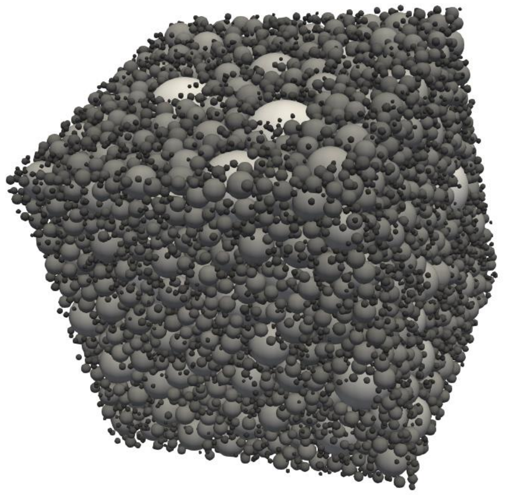

(h) Seventh round of 3D particles insertion

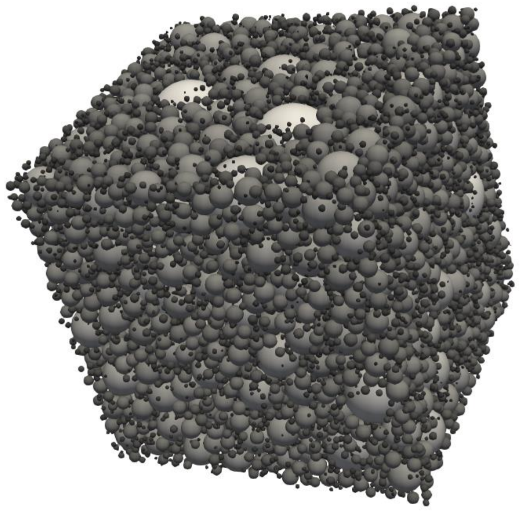
Fig 4.45 (Continued) 3D model generation for loose sand



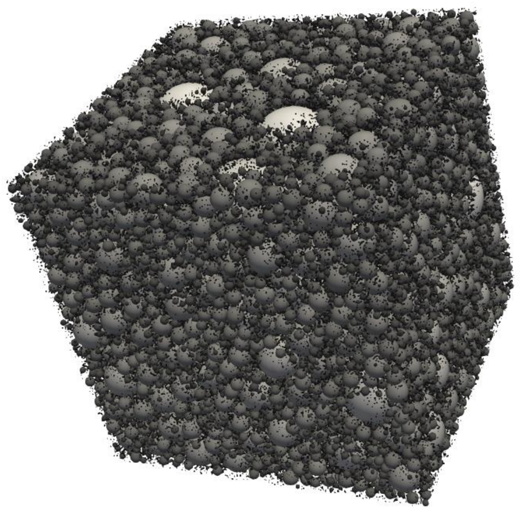
(i) Eighth round of 3D particles insertion



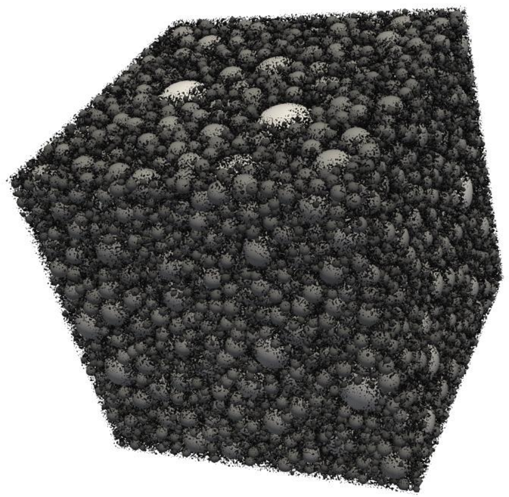
(j) Ninth round of 3D particles insertion



(k) Tenth round of 3D particles insertion



(l) Eleventh round of 3D particles insertion



(m) Twelfth round of 3D particles insertion

Fig 4.45 (Continued) 3D model generation for loose sand

The details of the 3D model are shown in Table 4.30. There were 835,674 particles in total in the model, and the running time was 105,508 seconds. The generation of particles belonging to the smallest opening sieve was time-consuming, accounting for roughly 30% of the running time.

Table 4.30 Details of 3D particle packing

Round of particles insertion	Opening of sieves mm	Mass $g\times10^{-2}$	Ideal mass $g\times10^{-2}$	Finer percentage	Ideal finer percentage	Percent difference
Poisson	3.35-2	68.24	65.15	100%	100%	0
1	2-1.7	52.51	52.12	89.62%	90%	0.38%
2	1.7-1.18	105.71	104.25	81.64%	82%	0.36%
3	1.18-1	104.77	104.25	65.56%	66%	0.34%
4	1-0.85	91.62	91.22	49.36%	50%	0.64%
5	0.85-0.5	91.25	91.22	35.70%	36%	0.3%
6	0.5-0.355	58.68	58.64	21.82%	22%	0.18%
7	0.355-0.25	32.58	32.58	12.90%	13%	0.1%
8	0.25-0.18	19.55	19.54	7.94%	8%	0.04%
9	0.18-0.106	13.03	13.03	4.97%	5%	0.03%
10	0.106-0.053	6.52	6.52	3.00%	3%	0
11	0.053-0.038	6.53	6.52	2.00%	2%	0
12	<0.038	6.62	6.52	1.00%	1%	0

The particle-size distribution curves of the algorithm-generated model and the real soil sample are shown in Fig 4.46. The particle-size distribution of the 3D model compares well with the real soil sample. The average error was 0.20%, and the maximal error was 0.64%, which occurred in the fourth round of particle insertion. Differences were regarded as acceptable. The void ratio of the model was 0.635, close to the target of 0.650.

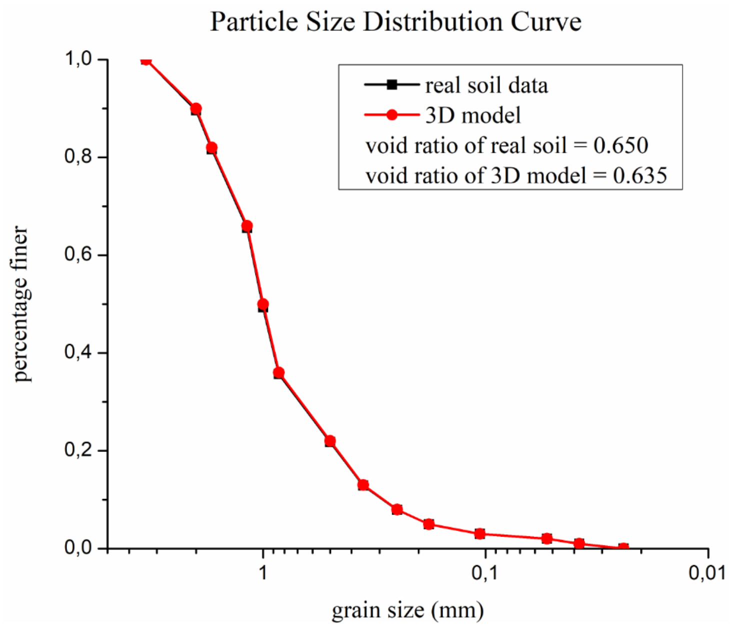


Fig 4.46 Comparison of particle-size distributions of the algorithm-generated 3D model and the real soil

The curve that presents the relationship between the void ratio of the deposited model and contact friction is shown in Fig 4.47. The void ratio first increased as contact friction rose but remained roughly constant after contact friction exceeded 35º. Contact friction of 35º can be considered the threshold point. Setting contact friction higher than this point ensures particle packing deposition results only in a minor reduction in void ratio. The coordination number of the 3D particle assembly was 4.80, which is greater than 4, indicating the particles were tightly packed (Masanobu 1977). As such, inter-particle friction could be used to adjust the void ratio.

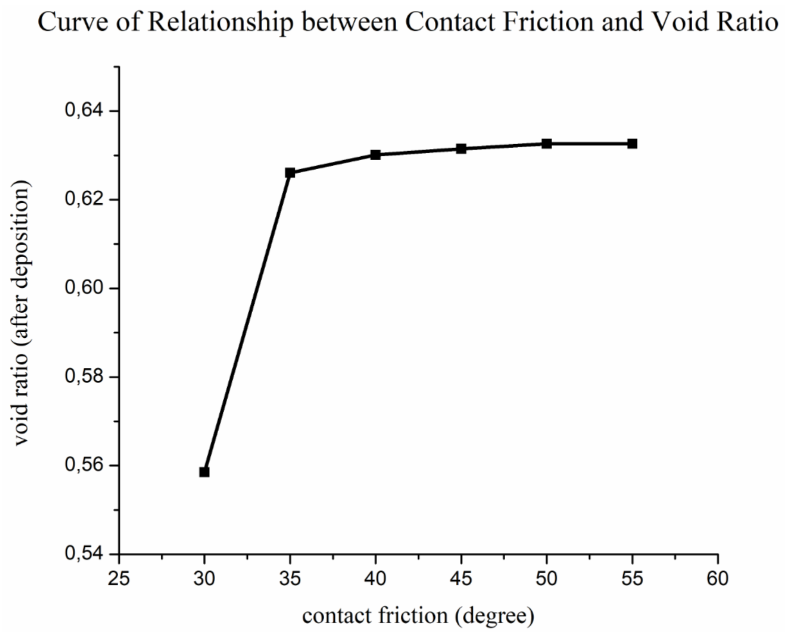


Fig 4.47 The relationship between contact friction and void ratio of 3D deposited model

The results of the 3D model deposition simulated by Yade are shown in Fig 4.48. The contact friction was set to 45º. The void ratio of the deposited model slightly decreased from the original value of 0.635 to 0.633. The model's volume correspondingly shrank from 18.75mm×18.75mm× 18.75mm to 18.75mm×18.75mm×18.725mm. The running time was 548 seconds.

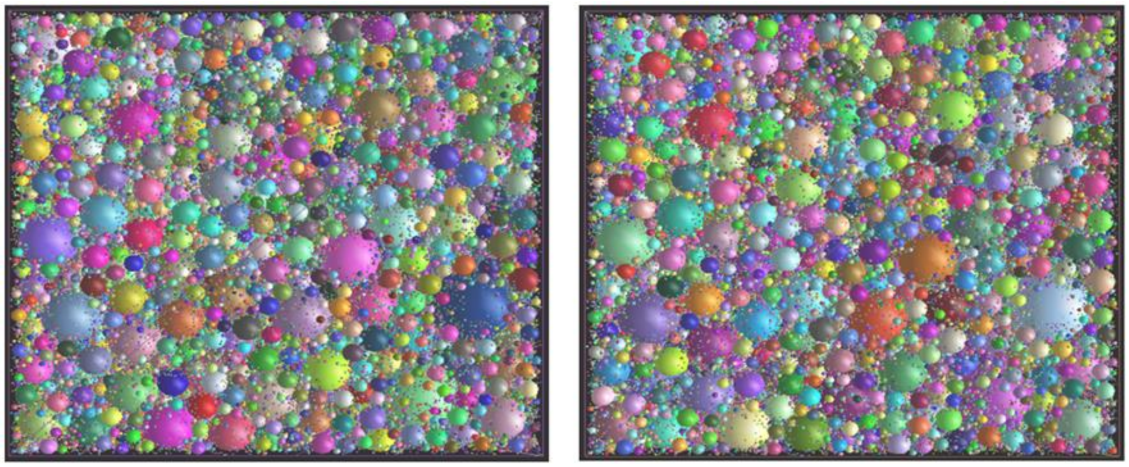

(a) (b)

Fig 4.48 The comparison of the original model (a) and the deposited model (b)

4.3 Error Analysis

The differences in the volume of particles belonging to the large sieves and the resulting void ratio variations in the 2D deposited models lead to errors between the algorithm-generated models and actual soils.

In 2D models, generating particles belonging to large-opening sieves is likely to result in errors. Errors were typically caused by the considerable volume of large sieve particles that led the total soil volume to exceed the target beyond the acceptable tolerance of 2% after the last particle was inserted. However, without this particle, the volume would have fallen short of the target. In 3D models, the overshoot could usually be minimized by shrinking radius ranges using recursion, as described in Section 3.4.3. However, in some cases, the particle volume still surpassed the target even when the algorithm shrank the radius range or set it equal to the minimum value. This problem most likely occurs when the individual volume of particles is large, and the difference between two sieve sizes is narrow, but particles account for a small percentage of the total mass. Adjusting the total volume with a narrow range of radii and large individual particle volumes is challenging, hence the emergence of errors.

Errors caused by volume overshoots are analyzed as follows:

a) In the 2D pure sand sample, the average error was 1.44%. The maximum error was 2.6%, and it occurred when the algorithm generated particles for the largest sieve. There were five circular particles in this sieve in the model, and their radii were all reduced to the minimum value in the radius range. Once the algorithm inserted the last particle for this sieve, the volume considerably exceeded the target. However, the volume would not have reached the target had the last large particle not been inserted. In the 3D model, the average error was 0.29%, and the maximum error was 0.58%. They were deemed acceptable, and the recursion method to rerun the particle insertion process with a reduced radius range could decrease the particle volume to an acceptable level.

b) In the 2D soil mixture sample, the average error was 0.78%, and a maximum error of 1.97% occurred when particles were created for the largest and second-largest sieves. The individual volumes of particles in these two sieves were considerable. In contrast, they only accounted for 5% of the total volume of the largest sieve particles and 2% of the second-largest sieve particles. As such, the particle volume could easily exceed the target. Using the recursion to shrink the particle radii, the algorithm set the radii of particles in these two sieves as the smallest value in the radius ranges, but the overshoot still existed. In the 3D model, the average error was 0.06%, and the maximum error was 0.07%. Both were considered acceptable.

c) In the 2D model of gap graded soils, the average error was 0.06%, and the maximum error was 0.12%, and it occurred in the third round of particle insertion. Due to the large fractions of particles in the first two sieves, with the particles in the largest sieve accounting for 17% of the total and those in the second-largest sieve accounting for 31% of the total, reducing the particle radius could adjust the volume to ensure errors were within the acceptable limit. In the 3D model, the average error was 0.23%, and the maximal maximum error was 0.75%, which occurred in the first round of the particle insertion. Errors in the 3D model were slightly higher than in the 2D model, and this is attributed to the fact that the particle volume in the largest sieve surpassed the target. There were six spherical particles in the largest opening sieve, and their radii were reduced to the smallest value of their radius range by the algorithm. The recursion used in the algorithm to reduce the radius range was not able to minimize the error.

d) In the 2D uniformly graded soil sample, the mean error was 0.39%, and a maximum error of 0.55% occurred in the first round of particle insertion. The main reason is that the individual particle volume in the first round of particle insertion was significant. In contrast, the percentage of particles in this sieve was relatively small, representing only 10% of the total volume. The sizeable individual particle volume combined with a small percent passing for this sieve caused the volume to easily exceed the target. In the 3D model, the average error was 0.47%, while the maximum error was 0.78%. They were deemed small and acceptable.

e) In the 2D model of the dense soil sample, the average error was 0.81%, while a maximum error of 3.34% occurred when particles belonging to the largest sieve were generated. This was caused by the fact that the largest opening sieve was 3.35-2mm, and the individual particle volumes in this sieve were considerable. In contrast, particles in this sieve accounted for only 10% of the total soil mass. Errors are likely to occur when the individual particle volume is large while the percent finer for this sieve is small. The radii of all the particles in this sieve were reduced to the minimum value in the radius range by the algorithm, but the overshoot could not be minimized. Thus, the volume exceeded the target when the algorithm inserted the last particle, however removing this particle would have caused the volume not to reach the target. In the 3D model, the average error was 0.21%, and the maximum error was 0.58%. The errors were considered acceptable.

f) In the 2D loose soil sample, the average error was 0.12%, and the maximum error was 0.49%. For the 3D model, the average error was 0.20%, and the maximal error was 0.64%. By shrinking the particle radius range in the large sieves, the algorithm ensured that the particle volumes of each sieve did not surpass the target by more than 2%. The errors were acceptable.

Another significant difference between 2D algorithm-generated models and real soils is the void ratios of the particle packings after the deposition process. For 3D models, the critical value of contact friction ensures that the void ratio of the deposited model reduces slightly and is close to the original value after the model deposition. The main reason is that the coordination numbers of spherical particle assemblies were generally above 4, which means 3D models were dense and stable. Tightly packed particles mean inter-particle friction played an essential role in the model density during the deposition process. As such, contact friction could be used to adjust the model's void ratio. Additionally, the high coordination number of most 3D models indicate that particle interlocking prevents particle packings from collapsing

However, since the coordination number of 2D models was generally around 2, contract friction could not be used to control the void ratio as it was in 3D models. In addition, the particle-size distribution also influenced the void ratio reduction. Small particles could easily slide over each other and fill nearly all the void spaces between large particles in models with a broad radius range or a high percentage of small sieve particles or models simulating poorly graded or loose soils, causing the void ratio to decrease sharply. The error analyses for the void ratio reduction in the 2D models after the deposition process are detailed below:

a) For the 2D pure sand model, the void ratio decreased sharply after the model deposition. This is attributed to the fact that the model had a pre-deposition void ratio of 0.531, indicating that it was loosely packed. Additionally, the model consisted of a large number of particles belonging to the three smallest sieves, which accounted for 17.33% of the total mass. As such, a large number of small particles slid into void spaces between large particles during the deposition process, causing the void ratio to decrease dramatically.

b) For the 2D mixture soil model, the void ratio decreased considerably to 0.300 from the original value of 0.523 after the model deposition simulation. The model's particle-size distribution played a significant role in reducing the void ratio. Compared to the pure sand sample, the gravel-sand mixture had a more comprehensive radius range. In addition, the particles of the first four largest sieves only accounted for 10% of the total mass, meaning that small particles accounted for a high percentage of the total mass. Therefore, it can be concluded that a large number of small particles filled nearly all the voids, causing the void ratio to decrease considerably.

c) For the 2D model of the gap graded soil sample, the void ratio decreased considerably to 0.373 after the model deposition process compared with the original value of 0.532, which is the most significant reduction in all the 2D examples. The main reason is that the percent finer of the three largest sieves accounted for 85% of the total mass. Thus, the positions of the large particles can essentially decide the void ratio. Small particles filled the voids between the large particles during the deposition process and hardly affected the void ratio. In addition, interlocking between the particles in well-graded soils is stronger than in poorly-graded ones (Khan 2012). Thus, weak interlocking in gap graded soils contributed to the void ratio reduction.

d) In the 2D uniformly graded soil sample, the model's void ratio decreased considerably to 0.380 after the deposition process compared with the original value of 0.779, which is the most significant reduction in all the 2D examples. This is attributed to the particle size distribution. Indeed, the high void ratio of the original model indicates the particles were loosely packed, and many voids existed in the model before the deposition simulation. Additionally, particles belonging to the smallest sieve accounted for a high proportion of the total mass (13.98%). It can be concluded that a considerable number of small particles filled nearly all the voids after the model was deposited, leading to a considerable void ratio reduction.

e) In the 2D dense soil example, the void ratio decreased moderately to 0.290 from the original value of 0.343 after the deposition process. Compared to the other 2D model deposition simulations, this void ratio reduction was the smallest. This is mainly due to the tight packing of the particles before the deposition simulation. Additionally, the proportion of particles in the last five smallest sieves was small, accounting for just 8% of the total mass. Accordingly, the small particles could not fill all the voids, leaving many empty spaces in the post-deposition model.

Moreover, the coordination number of the dense soil model was the highest of all the 2D models, indicating the relatively strong interlocking compared to the other 2D models helped prevent the particles from sliding. Thus, the void ratio did not reduce considerably.

f) The void ratio decreased sharply to 0.400 compared with the original value of 0.625 after the model deposition in the 2D loose soil example. This occurred because there were many voids in the loose soil sample, causing the particle assembly to collapse and its void ratio to be dramatically reduced following the deposition process.

4.4 Conclusion

This chapter provides examples aiming to simulate real soils to demonstrate the usability of the proposed algorithm. The 3D particle assembly generated by the algorithm showed excellent agreement with the actual soil in terms of the particle-size distribution and the void ratio. In 3D models with a high coordination number and a tight particle packing, contact friction can help adjust void ratios to ensure the deposition process only triggers a minor void ratio reduction. The particle-size distributions of 2D models compare well with the real soils, but there were still errors in the void ratios of deposited models. The primary sources of error are loosely packed 2D particles, weak interlocking, and initial particle arrangement.

Chapter 5 Conclusions and Future Work

5.1 Conclusion

This thesis proposes an algorithm to generate DEM models simulating the particle size distribution and void ratio of actual soil samples. The resulting models provide more accurate simulation results than currently available DEM software packages since the algorithm-generated models closely replicate real-world conditions. The conclusions are summarized as follows:

a) The proposed algorithm can create 3D particle assemblies that closely simulate the particle-size distribution and void ratio of real soils, thereby improving the accuracy of DEM simulations. It can also generate models in 2D that compare well with the real soils in terms of particle-size distribution, but the void ratios of the deposited models are significantly smaller than the target.

b) The algorithm can build particle assemblies sieve-by-sieve. The particle generation for the largest opening sieve is based on Poisson Disk distribution, and then the void-filling process creates particles belonging to the remaining sieves.

c) Particle validation enables determining whether a randomly selected point in a non-filled cell is valid. The recursive call ensures that the algorithm generates particles based on the user-defined soil data, reducing the error. Omitting fully covered cells before the particle insertion reduces the algorithm running time.

d) The model deposition simulation is done by Yade. Due to the closely packed particles in the 3D models, contact friction can help to adjust the void ratio of deposited models. There is a critical value of contact friction that can be used to ensure the model's void ratio is only slightly reduced by the deposition process. However, in 2D models, the void ratio decreased considerably after the model deposition. This is primarily caused by weak interlocking, lower coordination numbers, particle-size distribution, and initial void ratio.

5.2 Limitations

Although the DEM models created by the proposed algorithm are similar to real soils in terms of particle-size distributions and void ratios, there are two limitations in this study.

The first limitation is that building 3D models is time-consuming even though several acceleration measures were introduced. There are two potential solutions. The first consists in running the algorithm using a machine with a greater computational capacity. All the computations in this thesis were performed using an AMD Ryzen™ 7 4800H with 7nm FinFET. Recently, a CPU with 5nm FinFET with excellent calculation speed has been introduced. Given the speedy development of chip manufacturing technologies, computational power will undoubtedly increase. Another alternative is to simplify the models. The algorithm generates particles sieve-by-sieve. Shrinking the number of sieves that the particle packing contains, such as omitting particles of small-opening sieves, can sharply decrease the number of particles in the model and the running time. The process of particle insertion for the smallest-opening sieve is the most time-consuming, accounting for roughly 30%-40% of the total running time. The generation of particle assemblies without the smallest sieve particles substantially saves time but decreases the accuracy of the simulation results.

The second limitation is errors generated by the algorithm in 2D models. They may occur in two cases. The first one is that the void ratio of the 2D deposited model reduced significantly compared to the original model and the target void ratio. This is attributed to the interlocking phenomenon, the particle size distribution, the model's pre-deposition void ratio, and the rounded shape of the particles. Particles in 2D models were not as closely packed as in 3D models. Thus, contact friction cannot be used to adjust the void ratio. The second case is that errors are likely to arise when circular particles belonging to the large opening sieves are generated. Due to their large volumes of individual particles, the total volume may surpass the target by more than 2%. Shrinking the radius range can reduce the particle volume, but the overshoot may still persist.

5.4 Future Work

The innovations of DEM simulation are mainly focused on building models that replicate actual soil data with a high degree of fidelity and more accurate simulations of forces between particles. More research could be done in modeling particles of various shapes and assemblies of such particles with a particle-size distribution and a void ratio that compare well with actual soil data.

References

1. Aggarwal, A., and A. Kumar. 2019. "Particle scale modelling of porosity formation during selective laser melting process using a coupled DEM – CFD approach." *IOP Conference Series: Materials Science and Engineering*. 529: 012001. https://doi.org/10.1088/1757-899X/529/1/012001.

2. Aghakouchak, A. 2015. "Advanced laboratory studies to explore the axial cyclic behavior of driven piles." Ph.D. thesis, Dept. of Civil and Environmental Engineering, Imperial College London.

3. Ahn, J., and J. Jung. 2017. "Effects of Fine Particles on Thermal Conductivity of Mixed Silica Sands." *Applied Sciences*. 7(7): 650. https://doi.org/10.3390/app7070650.

4. Alex, M. 2018. "A Brief History of Web Browsers and How They Work." *Smartbear*. Accessed January 09, 2019. https://smartbear.com/blog/history-of-web-browsers/.

5. Anandarajah, A. 1994. "Discrete-Element Method for Simulating Behavior of Cohesive Soil." *Journal of Geotechnical Engineering*. 120(9): 1593–1613. https://doi.org/10.1061/(asce)0733-9410(1994)120:9(1593).

6. Ashcroft, A. I., and A. Mubashar. 2011. *Handbook of Adhesion Technology*. Berlin: Springer.

7. Ashmawy, A. K., B. Sukumaran, and V. V. Hoang. 2003. "Evaluating the Influence of Particle Shape on Liquefaction Behavior Using Discrete Element Modeling." *Proceedings of The Thirteenth (2003) International Offshore and Polar Engineering Conference*. 542-549.

8. Balevičius, R., A. Džiugys, and R. Kačianauskas. 2004. "Discrete element method and its application to the analysis of penetration into granular media." *Journal of Civil Engineering and Management*. 10(1): 3–14. https://doi.org/10.1080/13923730.2004.9636280.

9. Balevičius, R., R. Kačianauskas, A. Džiugys, A. Maknickas, and K. Vislavičius. 2005. "DEMMAT Code for Numerical Simulation of Multi-Particle Systems Dynamics." *Information Technology and Control*. 34(1): 71–77.

10. Bandeira, A. A., and T. I. Zohdi. 2018. "3D numerical simulations of granular materials using DEM models considering rolling phenomena." *Computational Particle Mechanics*. 06: 97–131. https://doi.org/10.1007/s40571-018-0200-0.

11. Beets, K., and D. Barron. 2000. "Super-sampling Anti-aliasing Analyzed." *Beyond3D*. Accessed December 19, 2020. http://www.x86-secret.com/articles/divers/v5-6000/datasheets/FSAA.pdf.

12. Belheine, N., J. P. Plassiard, F. V. Donzé, F. Darve, and A. Seridi. 2009. "Numerical simulation of drained triaxial test using 3D discrete element modeling." *Computers and Geotechnics*. 36(1-2): 320–331. https://doi.org/10.1016/j.compgeo.2008.02.003.

13. Bhavsar, P. D. 2020. "Why Python is One of the Most Preferred Languages for Data Science." Accessed April 17, 2021. https://www.kdnuggets.com/2020/01/python-preferred-languages-data-science.html.

14. Bill, V. 2003. "The Making of Python." Accessed March 22, 2020.

https://www.artima.com/articles/the-making-of-python.

15. Boac, J. M., R. P. Ambrose, M. E. Casada, R. G. Maghirang, and D. E. Maier. 2014. "Applications of Discrete Element Method in Modeling of Grain Postharvest Operations." *Food Engineering Reviews*. 06: 128–149. https://doi.org/10.1007/s12393-014-9090-y.

16. Bobet, A. 2010. "Numerical methods in geomechanics." *The Arabian Journal for Science and Engineering*. 35(1B): 27–48.

17. Brian, J. 2021. "13 Best Text Editors to Speed up Your Workflow." Accessed April 17, 2021. https://kinsta.com/blog/best-text-editors/.

18. Bridson, R. 2007. "Fast Poisson disk sampling in arbitrary dimensions." *ACM SIGGRAPH 2007 sketches on - SIGGRAPH '07*. 22–es. https://doi.org/10.1145/1278780.1278807.

19. Buttlar, W. G., and Z. You. 2001. "Discrete Element Modeling of Asphalt Concrete: Microfabric Approach." *Transportation Research Record: Journal of the Transportation Research Board*. 1757(1): 111–118. https://doi.org/10.3141/1757-13.

20. Campos, V. P., E. C. Sansone, and G. F. Silva. 2018. "Hydraulic fracturing proppants." *Cerâmica*. 64(370): 219–229. https://doi.org/10.1590/0366-69132018643702219.

21. Chappell, A., J. A. Baldock, and R. A. Viscarra. 2013. *Sampling soil organic carbon to detect change over time*. Canberra: CSIRO.

22. Cheng, A. H. D., and D. T. Cheng. 2005. "Heritage and early history of the boundary element method." *Engineering Analysis with Boundary Elements*. 29(3): 268–302. https://doi.org/10.1016/j.enganabound.2004.12.001.

23. Cook, R. L. 1986. "Stochastic sampling in computer graphics." *ACM Transactions on Graphics*. 5(1): 51–72. https://doi.org/10.1145/7529.8927.

24. Costabel, M. 1987. "Principles of boundary element methods." *Computer Physics Reports*. 6(1-6): 243–274. https://doi.org/10.1016/0167-7977(87)90014-1.

25. Crockford, D. 2011. "Douglas Crockford: The JSON Saga." *YouTube*. Accessed September 23, 2020. https://www.youtube.com/watch?v=-C-JoyNuQJs&gt.

26. Cryer, C. W. 1970. "On the Approximate Solution of Free Boundary Problems Using Finite Differences." *Journal of the ACM*. 17(3): 397–411. https://doi.org/10.1145/321592.321593.

27. Cugowski, T. 2016. "Anti-aliasing techniques comparison." Accessed September 23, 2020. https://www.sapphirenation.net/anti-aliasing-comparison-performance-quality.

28. Cundall, P. A. 1971. "A computer model for simulating progressive, large-scale movement in blocky rock system." *Proceedings of the International Symposium on Rock Mechanics*. 02: 2–8.

29. Cundall, P. A., and O. D. Strack. 1979. "A discrete numerical model for granular assemblies." *Géotechnique*. 29(1): 47–65. https://doi.org/10.1680/geot.1979.29.1.47.

30. Cundall, P. A., and R. D. Hart. 1992. "Numerical modelling of discontinua." *Engineering Computations*. 9(2): 101–113. https://doi.org/10.1108/eb023851.

31. D'Addetta, G. A., F. Kun, E. Ramm, and H. J. Herrmann. 2001. "From solids to granulates Discrete element simulations of fracture and fragmentation processes in geomaterials." *Continuous and Discontinuous Modelling of Cohesive-Frictional Materials*. 231–258.

https://doi.org/10.1007/3-540-44424-6_17.

32. Dai, Bei Bing, Jun Yang, F. and Cui Ying Zhou. 2016. "Observed Effects of Interparticle Friction and Particle Size on Shear Behavior of Granular Materials." *International Journal of Geomechanics*. 16(1): 04015011. https://doi.org/10.1061/(ASCE)GM.1943-5622.0000520.

33. Dang, H. K., and M. A. Meguid. 2010. "Algorithm to Generate a Discrete Element Specimen with Predefined Properties." *International Journal of Geomechanics*. 10(2): 85–91. https://doi.org/10.1061/(ASCE)GM.1943-5622.0000028.

34. Das, B. M. 2003. *Principles of Foundation Engineering*. Chonburi: CL Engineering.

35. Datas, A. 2020. *Ultra-High Temperature Thermal Energy Storage, Transfer and Conversion*. Madrid: Woodhead Publishing.

36. De Bono, J. P., and G. R. McDowell. 2015. "An insight into the yielding and normal compression of sand with irregularly-shaped particles using DEM." *Powder Technology*. 271: 270–277. https://doi.org/10.1016/j.powtec.2014.11.013.

37. Dershowitz, W. S., P. R. La Pointe, and T. W. Doe. 2004. "Advances in discrete fracture network modeling." Accessed December 15, 2020. https://www.clu-in.org/products/siteprof/2004fracrockconf/cdr_pdfs/indexed/group1/882.pdf.

38. Desai, C. S., and J. T. Christian. 1979. *Numerical methods in geotechnical engineering*. New York: McGraw-Hill.

39. Doherty, E. 2020. "What is Object Oriented Programming? OOP Explained in Depth." Accessed December 09, 2020. https://www.educative.io/blog/object-oriented-programming.

40. Dong, N., C. Zhu, and Y. Wang. 2015. "Hydro-Mechanical Analysis of Hydraulic Fracturing Based on an Improved DEM-CFD Coupling Model at Micro-Level." *Journal of Computational and Theoretical Nanoscience*. 12(9): 2691–2700. https://doi.org/10.1166/jctn.2015.4164.

41. Donzé, F. V., O. Galizzi, and J. Kozicki. 2009. "Welcome to Yade - Open-Source Discrete Element Method." Accessed December 15, 2020. https://yade-dem.org/doc/.

42. Dunbar, D., and G. Humphreys. 2006. "A spatial data structure for fast Poisson-disk sample generation." *ACM Transactions on Graphics*. 25(3): 503–508. https://doi.org/10.1145/1141911.1141915.

43. Dunning, D. 2006. "Void Ratio for Common Gravel & Sand." Accessed December 09, 2020. https://sciencing.com/void-ratio-common-gravel-sand-7958152.html.

44. Elmo, D., S. Rogers, D. Stead, and E. Eberhardt. 2014. "Discrete Fracture Network approach to characterize rock mass fragmentation and implications for geomechanical upscaling." *Mining Technology*. 123(3): 149–161. https://doi.org/10.1179/1743286314Y.0000000064.

45. Engineering Simulation and Scientific Software. 2021. "Rocky DEM—Advanced particle simulation software for better and faster results." Accessed December 15, 2020. https://rocky.esss.co/software/.

46. Estrada, N., A. Taboada, and F. Radjaï. 2008. "Shear strength and force transmission in granular media with rolling resistance." *Physical Review E*. 78(2): 1–11. https://doi.org/10.1103/physreve.78.021301.

47. Fei, W., and G. A. Narsilio. 2020. "Impact of Three-Dimensional Sphericity and Roundness on Coordination Number." *Journal of Geotechnical and Geoenvironmental Engineering*. 146(12): 06020025. https://doi.org/10.1061/(asce)gt.1943-5606.0002389.

48. Finlayson, B. A. 1972. *Method of Weighted Residuals and Variational Principles*. Cambridge, Massachusetts: Academic Press.

49. Flanagan, D. 2011. *JavaScript: The Definitive Guide: Activate Your Web Pages*. Boston: O'Reilly Media.

50. Fragaszy, R. J, and C. A. Sneider. 1991. "Compaction control of granular soils." Accessed December 15, 2020. https://trid.trb.org/view/355671.

51. Freeman, J. 2019. "What is JSON? A better format for data exchange." Accessed April 17, 2021. https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html

52. Gamito, M. N., and S. C. Maddock. 2009. "Accurate multidimensional Poisson-disk sampling." *ACM Transactions on Graphics*. 29(1): 1–19. https://doi.org/10.1145/1640443.1640451.

53. Geotechnical Aspects of Pavements Reference Manual. 2006. "Geotechnical Inputs for Pavement, Federal Highway Administration of U.S. Department of Transportation Design." Accessed November 20, 2020. https://www.fhwa.dot.gov/engineering/geotech/pubs/05037/05a.cfm.

54. Gore, S. P., D. F. Burke, and T. L. Blundell. 2005. "PROVAT: a tool for Voronoi tessellation analysis of protein structures and complexes." *Bioinformatics*. 21(15): 3316–3317. https://doi.org/10.1093/bioinformatics/bti523.

55. Hager, A., C. Kloss, and C. Goniva. 2018. "Combining Open Source and Easy Access in the field of DEM and coupled CFD-DEM: LIGGGHTS®, CFDEM®coupling and CFDEM®workbench." *Computer Aided Chemical Engineering*. 43: 1699–1704. https://doi.org/10.1016/B978-0-444-64235-6.50296-5.

56. Hagerty, M. M., D. R. Hite, C. R. Ullrich., and D. J. Hagerty. 1993. "One-Dimensional High-Pressure Compression of Granular Media." *Journal of Geotechnical Engineering*. 119(1): 1–18. https://doi.org/10.1061/(asce)0733-9410(1993)119:1(1).

57. Hall, W. S. 1994. *The boundary element method*. Dordrecht: Kluwer Academic Publishers.

58. Hammouri, N. A., A. I. Malkawi, and M. M. Yamin. 2008. "Stability analysis of slopes using the finite element method and limiting equilibrium approach." *Bulletin of Engineering Geology and the Environment*. 67: 471–478. https://doi.org/10.1007/s10064-008-0156-z .

59. Harris, C. R., K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. 2020. "Array programming with NumPy." *Nature*. 585: 357–362. https://doi.org/10.1038/s41586-020-2649-2.

60. Hemalatha, M. 2019. "Poisson Disc Sampling." *Medium*. Accessed April 17, 2021. https://medium.com/@hemalatha.psna/implementation-of-poisson-disc-sampling-in-javascript-17665e406ce1.

61. Hogg, R. 2008. "Issues in Particle Size Analysis." *KONA Powder and Particle Journal*. 26: 81–93. https://doi.org/10.14356/kona.2008009.

62. Incardona, P., A. Leo, Y. Zaluzhnyi, R. Ramaswamy, and I. F. Sbalzarini. 2019. "OpenFPM: A scalable open framework for particle and particle-mesh codes on parallel computers." *Computer Physics Communications*. 241: 155–177. https://doi.org/10.1016/j.cpc.2019.03.007.

63. Itasca. 2019. "UDEC, distinct-element modeling of jointed and blocky material." Accessed December 9, 2020. https://www.itascacg.com/software/udec.

64. Jean, M., M. Raous, and J. J. Moreau. 1995. *Contact Mechanics*. Berlin: Springer.

65. Jérier, J. F., D. Imbault, F.V. Donzé, and P. Doremus. 2008. "A geometric algorithm based on tetrahedral meshes to generate a dense polydisperse sphere packing." *Granular Matter*. 11: 43–52. https://doi.org/10.1007/s10035-008-0116-0.

66. Jérier, J. F., V. Richefeu, D. Imbault, and F. V. Donzé. 2010. "Packing spherical discrete elements for large scale simulations." *Computer Methods in Applied Mechanics and Engineering*. 199(25-28): 1668–1676. https://doi.org/10.1016/j.cma.2010.01.016.

67. Jing, L. 2003. "A review of techniques, advances and outstanding issues in numerical modelling for rock mechanics and rock engineering." *International Journal of Rock Mechanics and Mining Sciences*. 40(3): 283–353. https://doi.org/10.1016/s1365-1609(03)00013-3.

68. Jing, L., and J. A. Hudson. 2002. "Numerical methods in rock mechanics." *International Journal of Rock Mechanics and Mining Sciences*. 39(4): 409–427. https://doi.org/10.1016/s1365-1609(02)00065-5.

69. Jing, L., and O. Stephansson. 2007. *Fundamentals of Discrete Element Methods for Rock Engineering–Theory and Applications*. Amsterdam: Elsevier Science.

70. Ke, L., and A. Takahashi. 2012. "Strength reduction of cohesionless soil due to internal erosion induced by one-dimensional upward seepage flow." *Soils and Foundations*. 52(4): 698–711. https://doi.org/10.1016/j.sandf.2012.07.010.

71. Khan, M. S. 2010. "Investigation of Discontinuous Deformation Analysis for Application in Jointed Rock Masses." Ph.D. thesis, Rotman School of Management, University of Toronto.

72. Khan, Z. 2012 "Soil Classification and Identification (With Diagram)." *Your Article Library*. Accessed January 15, 2021. https://www.yourarticlelibrary.com/soil/soil-classification-and-identification-with-diagram/45407.

73. Kitware. 2000. "Kitware Signs Contract to Develop Parallel Processing Tools." Accessed April 20, 2021. https://blog.kitware.com/kitware-signs-contract-to-develop-parallel-processing-tools/.

74. Klassen, R. V. 1989. "Device Dependent Image Construction for Computer Graphics." Ph.D. thesis, University of Waterloo.

75. Kloss, C., C. Goniva, A. Hager, S. Amberger, and S. Pirker. 2012 "Models, algorithms and validation for opensource DEM and CFD-DEM." *Progress in Computational Fluid Dynamics*. 12(2/3): 140–152. https://doi.org/10.1504/pcfd.2012.047457.

76. Kozicki, J., and F. V. Donzé. 2009. "YADE-OPEN DEM: an open-source software using a

discrete element method to simulate granular material." *Engineering Computations*. 26(7): 786–805. https://doi.org/10.1108/02644400910985170.

77. Kuhlman, D. 2012. *A Python Book: Beginning Python, Advanced Python, and Python Exercises*. New York: Platypus Global Media.

78. Lei, Q., J. P. Latham, and C. F. Tsang. 2017. "The use of discrete fracture networks for modelling coupled geomechanical and hydrological behaviour of fractured rocks." *Computers and Geotechnics*. 85: 151–176. https://doi.org/10.1016/j.compgeo.2016.12.024.

79. Liang, G., L. Lu, Z. Chen, and C. Yang. 2015. "Poisson disk sampling through disk packing." *Computational Visual Media*. 01: 17–26. https://doi.org/10.1007/s41095-015-0003-7.

80. Lin, X., and T. T. Ng. 1997. "A three-dimensional discrete element model using arrays of ellipsoids." *Géotechnique*. 47(2): 319–329. https://doi.org/10.1680/geot.1997.47.2.319.

81. Liu, C., L. Pan, F. Wang, Z. Zhang, J. Cui, H. Liu, Z. Duan, and X. Ji. 2019. "Three-Dimensional Discrete Element Analysis on Tunnel Face Instability in Cobbles Using Ellipsoidal Particles." *Materials*. 12(20): 3347. https://doi.org/10.3390/ma12203347.

82. Lopez, J. M., N. Noreña, C. Meza, and C. Romanel. 2020. "Modeling of Asphalt Concrete Fracture Tests with the Discrete-Element Method." *Journal of Materials in Civil Engineering*. 32(8): 1–11. https://doi.org/10.1061/(asce)mt.1943-5533.0003305.

83. Luding, S., E. Clément, J. Rajchenbach, and J. Duran. 1996. "Simulations of pattern formation in vibrated granular media." *Europhysics Letters (EPL)*. 36(4): 247–252. https://doi.org/10.1209/epl/i1996-00217-9.

84. Makedonska, N., and C. W. Gable. 2018. "FracMan/dfnWorks: From Geological Fracture Characterization to Multiphase Subsurface Flow and Transport Simulation." *Computers & Geosciences*. 84: 10–19. https://doi.org/10.2172/1479909.

85. Martin, C. L., L. C. R. Schneider, L. Olmos, and D. Bouvard. 2006. "Discrete element modeling of metallic powder sintering." *Scripta Materialia*. 55(5): 425–428. https://doi.org/10.1016/j.scriptamat.2006.05.017.

86. Masanobu, O. 1977. "Co-Ordination Number and its Relation to Shear Strength of Granular Material." Soils and Foundations. 17(2): 29–42. https://doi.org/10.3208/sandf1972.17.2_29.

87. Matrix. 2010. "Chutes can increase conveyor belt life in mines." Accessed December 09, 2020. https://im-mining.com/2010/11/09/chutes-can-increase-conveyor-belt-life-in-mines/.

88. Matthews, C., Z. Farook, and P. Helm. 2014. "Slope stability analysis–limit equilibrium or the finite element method?" *Ground Engineering*. 48(5): 22-28.

89. McCarthy, L., C. Reas, and B. Fry. 2015. *Getting Started with p5.js: Making Interactive Graphics in JavaScript and Processing*. Santa Rosa: Make Community LLC.

90. Minabe, Y., S. Kawajiri, T. Kawaguchi, D. Nakamura, and S. Yamashita. 2016. "Correlation between Mechanical Properties and Suction Calculated by X-ray CT of Unsaturated Sandy Soil." *Procedia Engineering*. 143: 292–299. https://doi.org/10.1016/j.proeng.2016.06.037.

91. Mora, J., R. Otín, P. Dadvand, E. Escolano, M. A. Pasenau, and E. Oñate. 2006. "Open tools for electromagnetic simulation programs." *COMPEL–The international journal for computation*

and mathematics in electrical and electronic engineering. 25(3): 551–564. https://doi.org/10.1108/03321640610666709.

92. Moreau, A., A. Dompmartin, B. Castel, B. Remond, M. Michel, and D. Leroy. 1994. "Contact dermatitis from a textile flame retardant." *Contact dermatitis*. 31(2): 86–88. https://doi.org/10.1111/j.1600-0536.1994.tb01922.x.

93. Moreland, K., and J. Greenfield. 2007. "Large Scale Visualization with ParaView." Accessed April 20, 2021. https://www.osti.gov/servlets/purl/1719002.

94. Munjiza, A. 2005. *The combined finite-discrete element method*. Chichester: John Wiley & Sons.

95. Netscape. 1995. "Netscape and Sun announce JavaScript." Accessed May 15, 2021. https://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html.

96. Nikolić, M., T. Roje-Bonacci, and A. Ibrahimbegović. 2016. "Overview of the numerical methods for the modeling of rock mechanics problems." *Tehnički vjesnik*. 23(2): 627–637. https://doi.org/10.17559/TV-20140521084228.

97. O'Sullivan, C. 2011. "Particle-Based Discrete Element Modeling: Geomechanics Perspective." *International Journal of Geomechanics*. 11(6): 449–464. https://doi.org/10.1061/(asce)gm.1943-5622.0000024.

98. Ömer, A. 2006. "Analysis of Bearing Capacity Using Discrete Element Method." M.S. thesis, Middle East Technical University.

99. Opara, E. H., U. G. Eziefula, and B. I. Eziefula. 2018. "Comparison of physical and mechanical properties of river sand concrete with quarry dust concrete." *Journal of Civil Engineering*. 13(s1): 127–134. https://doi.org/10.1515/sspjce-2018-0012.

100. Pearce, B. W. 2011. "Ventilated Supercavitating Hydrofoils for Ride Control of High-Speed Craft." Ph.D. thesis, University of Tasmania.

101. Peña, F., P. B. Lourenço, and J. V. Lemos. 2006. "Modeling the Dynamic Behavior of Masonry Walls as Rigid Blocks." *European Conference on Computational Mechanics*. 282–282. https://doi.org/10.1007/1-4020-5370-3_282.

102. Rapp, B. E. 2017. *Microfluidics: Modelling, Mechanics and Mathematics*. San Francisco: Elsevier.

103. Reboul, N., E. Vincens, and B. Cambou. 2010. "A computational procedure to assess the distribution of constriction sizes for an assembly of spheres." *Computers and Geotechnics*. 37(1-2): 195–206. https://doi.org/10.1016/j.compgeo.2009.09.002.

104. Röber, N. 2014. "Paraview Tutorial for the Visualization of Earth and Climate Science Data." Accessed December 12, 2020. https://www.dkrz.de/mms/pdf/vis/paraview.pdf.

105. Rothenburg, L., and N. P. Kruyt. 2004. "Critical state and evolution of coordination number in simulated granular materials." *International Journal of Solids and Structures*. 41(21): 5763–5774. https://doi.org/10.1016/j.ijsolstr.2004.06.001.

106. Santamarina, J. C. 2003. "Soil Behavior at the Microscale: Particle Forces." *12th*

Panamerican Conference on Sil Mechanics and Geotechnical Engineering. 25–56. https://doi.org/10.1061/40659(2003)2.

107. Sbirrazzuoli, N., L. Vincent, A. Mija, and N. Guigo. 2009. "Integral, differential and advanced isoconversional methods." *Chemometrics and Intelligent Laboratory Systems*. 96(2): 219–226.

108. Schneider, P. J. 2011. "NURB Curves: A Guide for the Uninitiated." Accessed December 20, 2020. https://dokumen.tips/documents/nurb-curves-a-guide-for-the-uninitiated.html.

109. Shafranovich, Y. 2005. "Common Format and MIME Type for Comma-Separated Values (CSV) Files." Accessed December 22, 2020. https://www.rfc-editor.org/rfc/pdfrfc/rfc4180.txt.pdf.

110. Sharma, P. 2019. "What are the limitations of the finite element method." *Prakhar's Blog*. Accessed May. 21, 2021. https://prakhar962.wordpress.com/2019/06/26/what-are-the-limitations-of-the-finite-element-method/.

111. Shi, Z., T. Jiang, M. Jiang, F. Liu, and N. Zhang. 2015. "DEM investigation of weathered rocks using a novel bond contact model." *Journal of Rock Mechanics and Geotechnical Engineering*. 7(3): 327–336. https://doi.org/10.1016/j.jrmge.2015.01.005.

112. Skaggs, T. H., L. M. Arya, P. J. Shouse, and B. P. Mohanty. 2001. "Estimating Particle-Size Distribution from Limited Soil Texture Data." *Soil Science Society of America Journal*. 65(4): 1038. https://doi.org/10.2136/sssaj2001.6541038x.

113. Stefanov, S. 2010. *JavaScript Patterns: Build Better Applications with Coding and Design Patterns*. Newton: O'Reilly Media.

114. Sublime HQ Pty Ltd. "A sophisticated text editor for code, markup, and prose." Accessed December 9, 2020. https://www.sublimetext.com/.

115. Tan, C. M., Z. Gan, W. Li, and Y. Hou. 2011. *Applications of Finite Element Methods for Reliability Studies on ULSI Interconnections*. London: Springer.

116. Taxel, P. 2016. "Calculating the end color value." *Wikipedia*. Accessed December 09, 2020. https://en.wikipedia.org/wiki/Supersampling#/media/File:Supersampling.svg.

117. Thornton, C. 2000. "Numerical Simulations of Deviatoric Shear Deformation of Granular Media." *Geotechnique*. 50:43-53. http://dx.doi.org/10.1680/geot.2000.50.1.43.

118. Timoshenko, S. P., and J. N. Goodier. 1982. *Theory of elasticity*. New York: McGraw-Hill.

119. Travis, E. O. 2015. *Guide to Numpy*. North Charleston: CreateSpace Independent Publishing Platform.

120. Tulleken, H. 2008. "Poisson Disk Sampling". *Dev. Mag*. Accessed December 9, 2020. http://devmag.org.za/2009/05/03/poisson-disk-sampling/.

121. Van, R. G. 2009. "The History of Python: A Brief Timeline of Python." *Blogger*. Accessed March 5, 2021. https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html.

122. Venners, B. 2003. "The Making of Python—A Conversation with Guido van Rossum." *Artima*. Accessed December 09, 2020. https://www.artima.com/articles/the-making-of-python.

123. Wang, T., S. Liu, Y. Feng, and J. Yu. 2018. "Compaction Characteristics and Minimum Void Ratio Prediction Model for Gap-Graded Soil-Rock Mixture." *Applied Sciences*. 8(12): 2584.

https://doi.org/10.3390/app8122584.

124. Wautier, A., S. Bonelli, and F. Nicot. 2018. "DEM investigations of internal erosion: Grain transport in the light of micromechanics." *International Journal for Numerical and Analytical Methods in Geomechanics*. 43(1): 339–352. https://doi.org/10.1002/nag.2866.

125. Weatherley, D. 2008. "ESyS-Particle in Launchpad." *Launchpad*. Accessed December 9, 2020. https://launchpad.net/esys-particle.

126. Wes, B. 2014. *Sublime Text Power User: A Complete Guide*. Birmingham: Packt Publishing.

127. Widas, P. 1997. "Introduction to Finite Element Analysis." Accessed December 09, 2020. http://www.sv.rkriz.net/classes/MSE2094_NoteBook/97ClassProj/num/widas/history.html.

128. Yue, Y., B. Smith, Y. P. Chen, M. Chantharayukhonthorn, K. Kamrin, and E. Grinspun. 2018. "Hybrid Grains: Adaptive Coupling of Discrete and Continuum Simulations of Granular Media." *ACM Transactions on Graphics*. 37(6): 283. https://doi.org/10.1145/3272127.3275095.

129. Zhang, G., P. Fu, and F. Liang. 2013. "Mathematical and Numerical Modeling in Geotechnical Engineering." *Journal of Applied Mathematics*. 2013: 1–1. https://doi.org/10.1155/2013/123485.

130. Zoeken. 2004. "Super-sampling Anti-aliasing Analyzed (Multisampling)." *Beyond3D*. Accessed December 09, 2019. https://forum.beyond3d.com/threads/multisampling-antialiasing-explained.8659.

131. Zsaki, A. M. 2013. "Filling 2D domains with disks using templates for discrete element model generation." *Granular Matter*. 15(1): 109–117. https://doi.org/10.1007/s10035-012-0379-3.

132. Zvekan, J. 2004. "Multisampling Anti-Aliasing: A Closeup View." Accessed December 09, 2020. http://alt.3dcenter.org/artikel/multisampling_anti-aliasing/index_e.php.