Pontifícia Universidade Católica
do Rio de Janeiro

Elias Fukim Lozano Ching

An Algorithm to Generate Random Sphere Packs in Arbitrary Domains

Dissertação de Mestrado

Dissertation presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática, PUC–Rio as partial fulfillment of the requirements for the degree of Mestre em Informática

Advisor: Prof. Marcelo Gattass

Rio de Janeiro
August 2014

Pontifícia Universidade Católica
do Rio de Janeiro

Elias Fukim Lozano Ching

An Algorithm to Generate Random Sphere Packs in Arbitrary Domains

Dissertation presented to the Programa de Pósgraduação em Informática of the Departamento de Informática do Centro Técnico Científico da PUC–Rio, as partial fulfillment of the requirements for the degree of Mestre.

Prof. Marcelo Gattass
Advisor
Departamento de Informática — PUC–Rio

Prof. Deane de Mesquita Roehl
Departamento de Engenharia Civil — PUC-Rio

Prof. Waldemar Celes Filho
Departamento de Informática — PUC-Rio

Prof. Hélio Côrtes Vieira Lopes
Departamento de Informática — PUC-Rio

Prof. José Eugenio Leal
Coordinator of the Centro Técnico Científico da PUC–Rio

Rio de Janeiro — August 21st, 2014

Elias Fukim Lozano Ching

Graduated from the UNMSM (Universidad Nacional Mayor de San Marcos - Perú) in Systems engineering in 2010.

Bibliographic data

Lozano Ching, Elias Fukim

An Algorithm to Generate Random Sphere Packs in Arbitrary Domains / Elias Fukim Lozano Ching ; advisor: Marcelo Gattass. — 2014.

75 f. : il. ; 30 cm

1. Dissertação (Mestrado em Informática)-Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2014.

Inclui bibliografia

1. Informática – Teses. 2. Empacotamento de esferas. 3. Enfoque de avance frontal. 4. Algoritmo geométrico. 5. Campo de distancia. I. Gattass, Marcelo. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

To my advisor Professor Marcelo Gattass for the guidance and support.
To my parents, Doris and Elias, and brothers, Kiway and Jou.
To the CNPq and the PUC–Rio, for the financial support, without which this work would not have been realized.

Abstract

Lozano Ching, Elias Fukim; Gattass, Marcelo. **An Algorithm to Generate Random Sphere Packs in Arbitrary Domains**. Rio de Janeiro, 2014. 75p. MSc Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The Discrete Element Method (DEM) based on spheres can provide acceptable approximations to many complex physical phenomena both in micro and macro scale.

Normally a DEM simulation starts with an arrangement of spherical particles pack inside a given container. For general domains the creation of the sphere pack may be complex and time consuming, especially if the pack must comply with accuracy and stability requirements of the simulation.

The objective of this work is to extend a 2D disk packing solution to generate random assemblies composed by non-overlapping spherical particles. The constructive algorithm, presented here, uses the advancing front strategy where spheres are inserted one-by-one in the pack, according to a greed strategy based on the previously inserted particles. Advance front strategy requires the existence of an initial set of spheres that defines the boundary of the pack region. Another important extension presented here is the generalization of algorithm to deal with arbitrary objects defined by a triangular boundary mesh. This work presents also some results that allow for some conclusions and suggestions of further work.

Keywords

Sphere Packing. Advancing-Front Approach. Geometric Algorithm. Distance Field.

Resumo

Lozano Ching, Elias Fukim; Gattass, Marcelo. **Um Algoritmo de Geração Randômica de Esferas em Domínios Arbitrários**. Rio de Janeiro, 2014. 75p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O Método dos Elementos Discretos (DEM) com base em esferas pode fornecer aproximações para diversos fenômenos físicos complexos, tanto em escala micro quanto macro.

Normalmente uma simulação DEM começa com um arranjo de partículas esféricas no interior de um determinado recipiente. Para domínios gerais a criação deste pacote de esferas pode ser complexo e demorado, especialmente se ele deve respeitar requisitos de precisão e de estabilidade da simulação.

O objetivo deste trabalho é estender uma solução de empacotamento de discos 2D para gerar conjuntos aleatórios compostos por partículas esféricas não sobrepostas. O algoritmo construtivo proposto utiliza a técnica de frente de avanço, onde as esferas são inseridas uma a uma no pacote, de acordo com uma estratégia gulosa baseada nas partículas previamente inseridas. A técnica de frente de avanço requer a existência de um conjunto inicial de esferas que definem a fronteira do recipiente. Outra extensão importante proposta aqui é uma generalização do algoritmo para lidar com objetos arbitrários definidos por uma malha triangular qualquer. Este trabalho apresenta também alguns resultados que permitem algumas conclusões e sugestões de trabalhos futuros.

Palavras–chave

Empacotamento de esferas. Enfoque de avance frontal. Algoritmo geométrico. Campo de distancia.

Contents

List of Figures

List of Tables

List of Symbols

n	Number of particles in the pack
R_{min}	Minimum radius in the pack
R_{max}	Maximum radius in the pack
$F()$	Distribution function for the radii creation
$lprs$	List of previously rejected radii
$lnrs$	List of newly rejected radii
$\vec{e^{min}}$	Uniform grid lower left point
L_{box}	Size of the neighbourhood box
R_{curr}	Radius of the current sphere in the loop
R_{new}	Radius for the new particle
R_{halo}	Radius of the halo of a sphere
N_p	Number of sample points in the halo intersection
$P(t)$	Circle equation
$\vec{c_3}$	Circle center
r_3	Circle radius
$\vec{U}$	First orthogonal vector on the plane of the circle
$\vec{V}$	Second orthogonal vector on the plane of the circle
$\vec{N}$	Circle normal
ϵ	Displacement factor for the prism calculation
(d_x, d_y, d_z)	Distance field resolution
$MAABB$	Margin Axis aligned bounding box of the mesh
d_s	Margin to create the MAABB
M_p	Mesh proximity distance to consider an initial front
R	Ray traced for the point in polyhedron test
$\vec{R_{start}}$	Ray starting point
$\vec{R_{dir}}$	Ray direction

1
Introduction

For a long time the arrangement of spherical particles inside a given container, also called sphere packing, has been an active research topic due to its academic and industrial importance. In 1611 Kepler stated that are three types of sphere packings of equal size (cubic lattice, face-centered cubic lattice and hexagonal lattice) with a density of approximately $\frac{\pi}{3\sqrt{2}} = 0.74048...$ (38). This is known as the Kepler conjecture, not proved until 1998 by Thomas C. Hales. In 1694 Isaac Newton suggested that the highest number of non-overlapping spheres of the same radius that could be in contact with a central sphere is 12. This problem is known as the contact or coordination number, proved in 1953 by Schütte and van der Waerden (38).

The Discrete Element Method (DEM), proposed by Cundall (9), is an important tool to study the mechanical behavior of phenomena involving grains in both micro and macro scale represented by disks or spheres in virtue that they provide an acceptable approximation to many complex physical phenomena (11). Granular materials, with a vital importance in many industrial processes, are nowadays present in diverse domains such in the medical area in the field of radiosurgical treatment planning (12); pharmaceutical powder and tableting process simulation (25); food modeling (36); cereal production (26, 32) and geotechnical engineering (22).

Normally a DEM simulation starts with an arrangement of spherical particles inside a given container. For general domains the creation of the sphere pack may be complex and time consuming specially if the arrangement must comply with accuracy and stability requirements of the simulation (40). Many factors influence a simulation (28), some of which are: the radial distribution of the particles, the overlaps between them and the density ratio. This last property depends on the type of material; concrete, rock and ceramic require a high density pack while soil for example requires a lower density pack. It is relevant to point out that with experimental tests it was estimated that a random or irregular close packing with spheres of equal size

(or mono-disperse arrangement) has a maximum density of approximately 0.64 (18). Another important feature is the coordination number which determines the transfer of forces in the system (5). In geometric terms the coordination number is the average number of contacts of a sphere in the pack.

The objective of this work is to extend a 2D disk packing solution proposed by Ferreira (30) to generate random assemblies composed by non-overlapping spherical particles. The constructive algorithm presented by Ferreira uses the advancing front strategy where spheres are inserted one-by-one in the pack, according to a greedy strategy based on the previously inserted particles. The method relies on the existence of an initial set of spheres called the *initial front* that defines the boundary of the pack region. Ferreira and many other authors restrict the domain to simple domains, usually boxes. Another important extension presented here is the generalization of the algorithm to deal with 3D objects defined by a triangular boundary mesh. To accomplish this task our algorithm uses a signed distance field (Fuhrmann et al (13)).

This work is organized as follows; Chapter 2 presents some of the existing packing generation solutions and attempts to classify them. The packing algorithm is described in Chapter 3. Chapter 4 presents our strategy to create the initial front for a general 3D boundary represented by a triangle mesh. The results and analyses of our algorithms are presented in Chapter 5. Finally, we present some conclusions and suggest future works.

2
Related Work

Packing algorithms can be classified in two main categories: dynamic and constructive or geometric.

The dynamic approach uses Discrete Element (DEM) simulations to create the sphere packages. These simulations fill progressively the domain with particles letting them interact with each other according to the equations of motion (21). The "Isotropic-compression method" inserts particles inside the pack volume and then moves the walls to compress the package. The particle positions are randomly chosen and when a collision is detected a new position is calculated. After a predefined number of rejections the particle is removed. The process is repeated until the desired density is obtained. Even though the algorithm leads to good densities it is very time consuming. The "Expansion method" was conceived as an evolution of the isotropic-compression method. Small spherical particles are randomly inserted inside the volume, then their radii are increased until they reach their correspondent radii or until a contact with another particle is detected. The "Single layer method" creates a set of particles inside a volume with random positions, then the top wall is moved downwards in order to compress the package. The wall stops when a certain void ratio is reached. The "Multi-layer method", as the name suggests, creates a first set of particles, then they are compressed by the top wall to tighten the package. Next, a new set of particles is created above the first layer, this set is also compressed. The process continues until the volume is filled up with particles. Dynamic methods yield satisfactory results in terms of producing stable arrangements and providing contact force information for every particle. Their drawbacks, however, are the ones inherent to a DEM implementation: it is time consuming and computational intensive. This restriction limits the number of particles of the arrangement.

The constructive methods only uses geometrical calculations, therefore they are more efficient in terms of time and have more control of the package properties such as the radius size distribution of the particles.

A naive geometric approach generates random positions inside the domain and only accepts those particles that do not overlap any of the others. This solution

is very inefficient and generates packages with very low density.

Some constructive methods simulate the gravitational force to make the deposition of the particles with geometric calculations. A compression algorithm was developed in (14) to tightly pack a set of sphere particles inside an arbitrary polyhedron. In the same way as the dynamic multi-layer method, at each step a disperse pack is generated within the empty space and then a random direction is assigned to each particle to execute the compression. To accelerate the process it is used an efficient contact search algorithm called "no binary search" (29). In (15) poly-sized sphere packages are assembled inside parallelepipedic domains using dropping and rolling calculations that simulate the gravitational force with analytical equations. Each sphere is dropped to hit another particle and then it rolls until it hits another sphere or floor. The algorithm considers the sphere dropping finished when it reaches a certain geometric stability.

Another set of geometric algorithms use the advancing front technique firstly proposed by Feng et al (11). An advancing front method computes the new particle position based on the information of the previous inserted particles. Feng et al present two versions of the algorithm in 2D, the "closed form advancing front" and the "open form advancing front". In the first approach three disks in contact with each other are created in the center of the domain. The three triangle segments formed by its centers conform the initial fronts. For each front (segment) a new disk is inserted in contact with other two particles along an outward spiral. The new disk forms two new segments which are added to the front list while the current segment is discarded as a front. The open version fills the container layer by layer from bottom to top inserting new particles from left to right.

Bagi (1) introduced the "inwards packing method", an advancing front algorithm designed for polygonal shapes. The method, similar to the method in this work, uses an initial front composed by disks inside the container placed in contact with the edges of the domain forming a closed chain. New disks, created using a defined distribution function, are positioned in contact with two previous inserted disks and the front is updated. Bagi considers a disadvantage the lack of density control so there are suggested post packing generation ideas such as a random elimination of particles to decrease the density and the use of a dynamic method to compact the pack reducing the porosity. With some similarities to the solutions of Feng et al and Bagi, Zsaki (40) proposed a method to fill polygonal domains. The main difference with the previous approaches is that it does not use an initial front. The method first identifies the lowest vertex in the left corner to insert the particle seed tangent to two

edges. Next, new disks are added layer by layer to be tangent to the current edge and the previous particles. An interesting characteristic of this solution is its capability of being parallelized (41).

Like our solution, some works focus on the generation of particles inside of non trivial domain boundaries generally defined by a triangular mesh.

Cui (8) developed a relatively fast and simple method to generate packs of spheres. The approach handles 2D and 3D boundaries producing dense arrangements in 2D and less dense in 3D. The methods starts by creating random points inside the domain, then with these points, triangles (or tetrahedrons in 3D) are constructed with a Delaunay triangulation. Finally, a single sphere is inserted in each triangle incircle (or tetrahedron insphere). The pack quality depends on the triangulation refinement algorithm being obtained a higher density using equilateral triangles.

Benabbou (2, 3) presents another variation of the initial front approach capable of handling complex containers composed by triangles (2D) or tetrahedra (3D). The front, initially representing the domain, is a set of segments in 2D and triangles in 3D using a weighted Delaunay triangulation. Similar to Bagi's work, Benabbou's method proceeds with an inward packing using initial spheres placed inside the container in contact with the walls. Then new spheres are inserted tangent to the other spheres and the front list is updated. Benabbou introduces the concept of "front level" to guarantee the convergence of the algorithm. The level of a front is defined as the sum of the level of its particles.

Liu et al (27) proposed a sphere packing algorithm with the objective of generate high quality meshes with the Delaunay triangulation. The method packs meshes starting with an initial front constructed by placing particles in the vertices, edges and faces of the mesh. The algorithm uses Horn's theorem (16) to test if a point is inside or outside the polyhedron.

Labra et al (24) use the Stienen model (35), a fast geometrical algorithm that generates a non-dense set of stationary Poisson points to then proceed with the void space minimization obtaining a dense pack inside a finite element mesh. The package optimization considers the minimization of the distances of the particles to the triangles mesh. Jerier et al (19, 20) improves the ideas in (8) to create packs with polydisperse spheres inside tetrahedral meshes. They define a geometric procedure to fill spheres inside tetrahedrons placing particles in the edges, vertices and in inside the tetrahedrons. Weller et al (39) introduced a parallel algorithm in the GPU base in the "Apollonian sphere packing" (7) to fill an arbitrary object with non-overlapping spheres. The basic idea is to fill the model successively with spheres as big as possible. The objective of Weller

is to create $feasible$ packs rather than obtain optimal geometrical properties. Liu et al (28) propose the seed expansion method, and advancing front method to fill an arbitrary domain. Using a Delaunay tessellation and a distance function the package is improved inserting new spheres in the void spaces obtaining dense polidisperse assemblies.

With respect to the techniques relying on the existence of the initial fronts to support the assembly generation, their major difficulty is to create this set for a 3D container. Schiftner (33) proposed a way to pack circles or spheres on arbitrary surfaces introducing a new kind of triangle mesh whose faces incircles form a pack. Using this method would restrict the packing generation to this type of meshes or would imply altering a generic mesh to adapt the front generation. Benabbou (2, 3) places three spheres for each triangle (one in each vertex) to create the first particles. Similarly in (27) the quality of the initial front is restricted to the container triangulation and the level of detail of the mesh. Jerier (20) limits the size of the radii, defined by the mean length of the mesh, which must be previously tetrahedralized. Our goal here is to avoid these restrictions and requirements with a simple solution based on distance fields that yields a set of contacting spheres tangent to the surface of the container. Our initial front may present some gaps between the spheres that can be larger than the minimum diameter of the sphere distribution and, as pointed out by Bagi (1), this may yield some leak. Our method, however, avoids leaking with the continuous use of the distance field function.

3
Package Generation

The algorithm's basic idea is to insert one particle at a time ensuring that there is no sphere interpenetration and the void space is minimized. There is a step before the one by one insertion called the *initial front generation* which consists of replacing the boundaries of the domain with sphere particles. Defining a wall boundary with spheres is a valid and a common strategy in the DEM simulation (31). It happens, however, that most existing algorithms deal with simple forms, such as boxes or cylinders, where it is a straightforward task to convert its planes into spheres. We are interested here in general 3D boundaries and the next chapter presents an algorithm to generate the initial front for these situations.

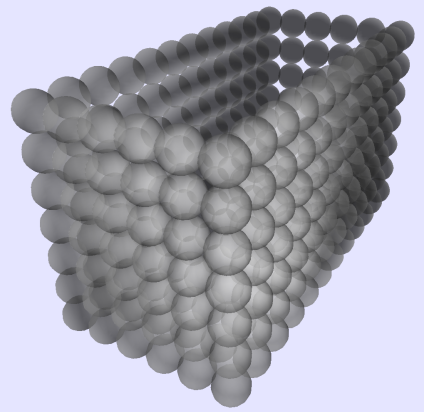


Figure 3.1: Initial fronts for a box representing the boundary.

3.1
Radius generation and particle size-distribution

Some algorithms in the literature deal only with assemblies of identical sphere sizes, while in others the spheres radius can vary according to a certain probability distribution, $F()$. Most of the existing algorithms implement only a uniform distribution between a minimum radius, R_{min} , and a maximum, R_{max} . Our algorithm can deal with more general distributions, such as Truncated gaussian or Bernoulli distributions, that occur in practical particle engineering

simulations, but we still require the definition of the limits R_{min} and R_{max} . We also consider a mechanism to handle possible rejected radii in the generation loop. In order to maintain the desired radius distribution a rejected radius is saved to be used in the next opportunity.

3.2 Neighbour particle search

A critical step in the particle methods is to find the closest neighbors of a given particle. This step is usually in the core of the algorithms and can be a very time consuming process given that the computational cost is $O(n^2)$ (being n the number of particles). Some algorithms use complex spatial index structures. Here we adopt a uniform grid. The uniform grid is a simple and effective space subdivision scheme. It divides the space into a number of regions or voxels of equal size. As suggested by (10), the optimal voxel size

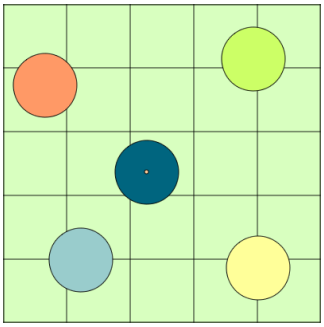


Figure 3.2: Uniform grid.

is the smaller size that can hold the largest object in the scene in all rotated positions. Since we are only dealing with sphere particles the voxel size here depends only on the maximum radius R_{max} given as an input.
Considering a uniform grid with a dimension of (s_x, s_y, s_z) (number of voxels per dimension) and with a lower left point ($e^{\vec{min}}$ .x, $e^{\vec{min}}$ .y, $e^{\vec{min}}$ .z), any point (p_x, p_y, p_z) belongs to the voxel

$$v = i + js_x + ks_xs_y \tag{3-1}$$

with

$$(i, j, k) = (\lfloor \frac{(p_x - e^{\vec{min}}.x)}{R_{max}} \rfloor, \lfloor \frac{(p_y - e^{\vec{min}}.y)}{R_{max}} \rfloor, \lfloor \frac{(p_z - e^{\vec{min}}.z)}{R_{max}} \rfloor) \tag{3-2}$$

Each cell of the index grid has a reference to all particles it overlaps.

3.3 Generation loop

Algorithm 3, presented at the end of the chapter, describes the proposed particle generation. The algorithm is based on an active front of spheres that is stored in a list. The active front is initialized with the initial front and the algorithm proceeds inserting and removing spheres from the active front until it is empty. In each step of the loop the algorithm finds a valid position for a new sphere using the information of the neighborhood particles. More precisely, the algorithm selects a sphere in the active front to be the "current sphere" and tries to create a new sphere touching it. Based on the current sphere the algorithm proceeds to find the set of nearby particles that can interfere in the creation of the new sphere. This is accomplished by testing all the spheres in the active front against a neighborhood box, centered at the current sphere center. The size L_{box} of the neighborhood box is given by:

$$L_{box} = 2(R_{curr} + 2.2R_{new}) \tag{3-3}$$

where R_{curr} is the radius of the current sphere and R_{new} is the radius of the new sphere.

The search for the nearby spheres can be accelerated with the help of the spatial index. Here the spatial index is implemented with a uniform grid.

For all the spheres that intersect the neighborhood box our algorithm creates their halos. By a halo we mean a concentric sphere with a greater radius. In the proposed algorithm the intersection of halos defines the locus of the sphere with a radius R_{new} tangent to the current sphere. For this reason the radius of the halo is given by:

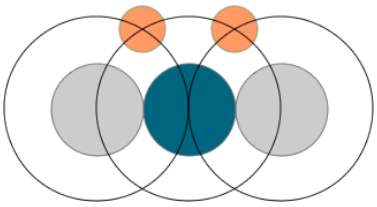
$$R_{halo} = R_{curr} + R_{new} \tag{3-4}$$

This equation differs from the one suggested by (30) where the halo is computed by:

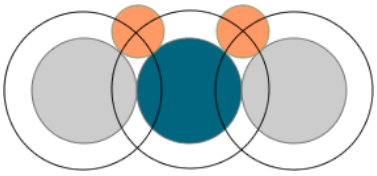
$$R_{halo} = R_{curr} + R_{max} \tag{3-5}$$

For a mono-size package both definitions generate the same outcome, but in the case of a package composed by particles of different radii our modification produces more compact assemblies as illustrated in Figure 3.3. Now, for every $< currentsphere - neighboursphere >$ the algorithm proceeds to compute the intersection of their halos looking for the potential locations to place the new particle.

The neighborhood box size is justified with the halo intersection concept in Figure 3.4. We illustrate two sphere insertions with different radii where the halos are touching creating a single candidate point for the new particle. There



3.3(a): Previous definition creates void spaces.



3.3(b): Our definition avoids empty areas.

Figure 3.3: Different halo outcomes.

are no spheres beyond this distance (the current front radius plus the new particle diameter) that could generate candidate positions. To guarantee the search of neighbor particles in this situation the formula uses 2.2 times the new particle radius.

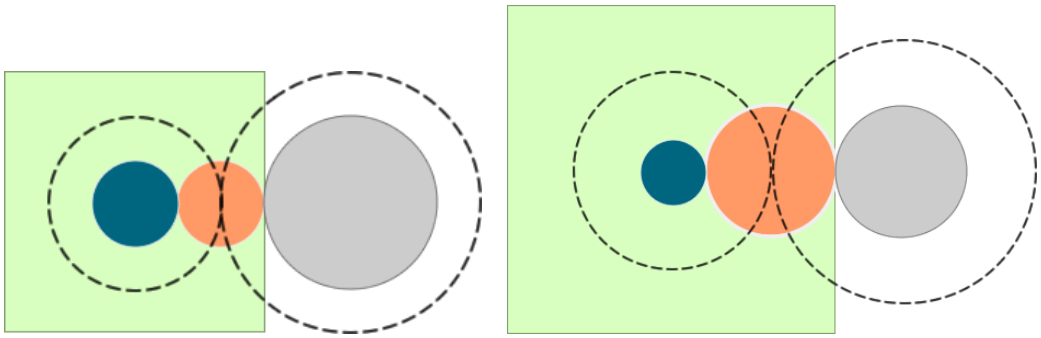
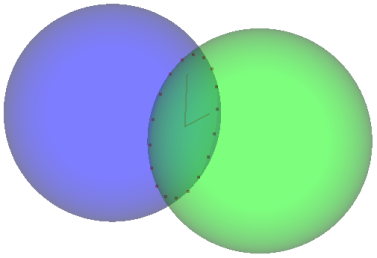


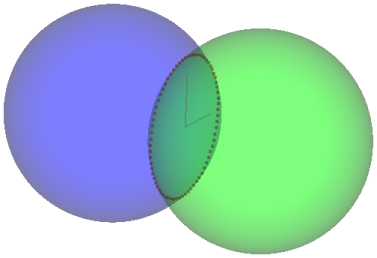
Figure 3.4: Neighborhood box size.

3.4 Halo intersections

While in 2D, halo intersections yield two points, in 3D the intersection of two spheres yields a circle with an infinite number of points. A simple solution for this step is to sample the circle with a predetermined number of equidistant points, N_p having in mind that bigger N_p results in more compact assemblies but the algorithm takes longer.



3.5(a): 18 points



3.5(b): 72 points

Figure 3.5: Sampled points in a circle.

The points in the intercepting circle can be evaluated by:

$$P(t) = \vec{c_3} + r_3(\vec{U}\cos(t) + \vec{V}\sin(t)), 0 \leq t < 2\pi \tag{3-6}$$

where $\vec{c_3}$ and r_3 are the circle's center and radius and $\vec{U}$ and $\vec{V}$ are unit orthogonal vectors laying in the plane of the circle. A point (x,y,z) that lays in the intersection of two spheres with centers and radii $(\vec{c_1}, r_1)$ and $(\vec{c_2}, r_2)$ must satisfy:

$$\begin{aligned}(x - \vec{c_1}.x)^2 + (y - \vec{c_1}.y)^2 + (z - \vec{c_1}.z)^2 - {r_1}^2 = 0\\ (x - \vec{c_2}.x)^2 + (y - \vec{c_2}.y)^2 + (z - \vec{c_2}.z)^2 - {r_2}^2 = 0\end{aligned} \tag{3-7}$$

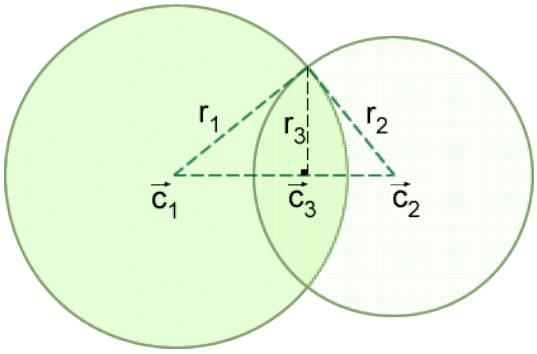


Figure 3.6: Halo intersection in 2D.

We observed in Figure 3.6 the creation of two right triangles with equations:

$${r_1}^2 = \|c_1\vec{c_3}\|^2 + {r_3}^2 \tag{3-8}$$

$${r_2}^2 = (\|c_1\vec{c_2}\| - \|c_1\vec{c_3}\|)^2 + {r_3}^2 \tag{3-9}$$

Because $\|c_1\vec{c_2}\|$ is the known distance between both spheres we subtract Equation 3-8 from Equation 3-9 to calculate the distance $\|c_1\vec{c_3}\|$.

$$\|c_1\vec{c_3}\| = \frac{\|c_1\vec{c_2}\|}{2}(1 - \frac{{r_2}^2 - {r_1}^2}{\|c_1\vec{c_2}\|^2}) \tag{3-10}$$

Then, the circle center could be computed as:

$$\vec{c_3} = \frac{\|c_1\vec{c_2}\| - \|c_1\vec{c_3}\|}{\|c_1\vec{c_2}\|}\vec{c_1} + \frac{\|c_1\vec{c_3}\|}{\|c_1\vec{c_2}\|}\vec{c_2} \tag{3-11}$$

Simplified to:

$$\vec{c_3} = (1 - a)\vec{c_1} + a\vec{c_2} \tag{3-12}$$

with

$$a = \frac{1}{2}(1 - \frac{{r_2}^2 - {r_1}^2}{\|c_1\vec{c_2}\|^2}) \tag{3-13}$$

The radius r_3 is computed using the Pythagoras theorem with any of the right triangles. Here we used Equation 3-8.

$$r_3 = \sqrt{\|c_1\vec{c}_3\|^2 - {r_1}^2} \tag{3-14}$$

To find the first orthogonal vector $\vec{U}$ to the circle normal $\vec{N}$ it is performed a cross product between the normal and a support vector. We choose an axis of the system as this support depending of the minimum normal vector component as detailed in Algorithm 1.
Finally, the second orthogonal vector $\vec{V}$ is the cross product between $\vec{U}$ and the plane normal.

Algorithm 1 System Axis selection

Input: $\vec{N}$, intersection circle normal.
Output: $a\vec{x}is$, support vector.
  if $\vec{N}.x = \min(\vec{N}.x, \vec{N}.y, \vec{N}.z)$ **then**
  $\quad a\vec{x}is = (1,0,0)$
  end if
  if $\vec{N}.y = \min(\vec{N}.x, \vec{N}.y, \vec{N}.z)$ **then**
  $\quad a\vec{x}is = (0,1,0)$
  end if
  if $\vec{N}.z = \min(\vec{N}.x, \vec{N}.y, \vec{N}.z)$ **then**
  $\quad a\vec{x}is = (0,0,1)$
  end if

$$\vec{U} = \frac{\vec{N} \times a\vec{x}is}{\|\vec{N} \times a\vec{x}is\|}$$
$$\vec{V} = \frac{\vec{U} \times \vec{N}}{\|\vec{U} \times \vec{N}\|} \tag{3-15}$$

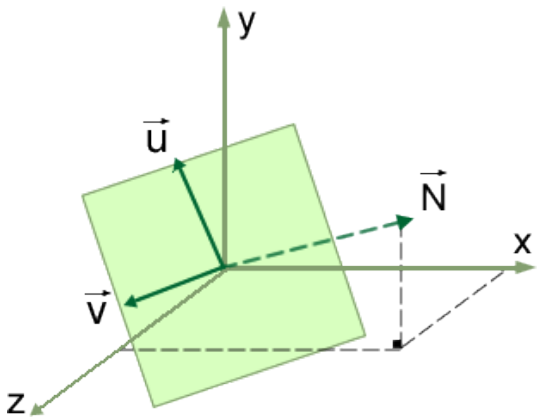


Figure 3.7: Orthogonal vectors for circle plane.

3.5 Particle insertion

Once the set of possible candidates are gathered, the best candidate, if exists, is selected and the others discarded. Our algorithm follows and insertion

criteria based on the y, z, x order that seeks to simulate the deposition process. For example, the points with the lowest y-coordinate (vertical) value receive the highest rank, then the ones with the lowest z-coordinate value and finally we use the x-coordinate value as a tiebreaker.

Once all possible intersecting circles are computed the algorithm loops over the list to see if the new particle's candidate position is inside the boundaries and does not collide with any of the surrounding spheres. The new created sphere is inserted in the active front list, in the assembly and in the spatial index.

When the number of potential candidates is less than two the current sphere is removed from the active front because it is likely that it wont generate new particles and the loop starts again.

It is important to notice that when the algorithm rejects a sphere, the radius distribution of the assembly is no longer the one specified by the user. To avoid this change, the algorithm manages two lists: a list of previously rejected spheres $lprs$ and a list of newly rejected spheres $lnrs$. The first one contains a list of all the sphere sizes rejected in previous loop cycles. When a new sphere radius is needed the stored radius is tried again. If it is again rejected it is stored in $lnrs$. Only when $lprs$ is empty then the algorithm uses the distribution function $F()$ to select a new radius. When an insertion is successful the values of $lnrs$ are emptied into the $lprs$ vector. In this way we guarantee that a rejected particle is always tested until it is finally inserted.

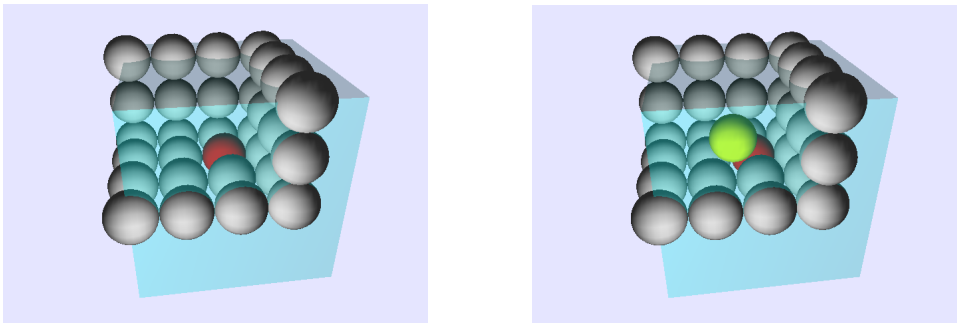


Figure 3.8: Example of current front, its neighborhood box and inserted particle.

Algorithm 2 Select a radius

Input: R_{min} ,R_{max} ,$F()$,$lprs$

```
if lprs is empty() then
    return GetValue(R_min ,R_max ,F() )
else
    return pop(lprs )
end if
```

Algorithm 3 Assembly Generator

Input: mesh,frontSpheres,R_{min} ,R_{max} ,$F()$,N_p

```
assembly ← emptyList()
lprs = lnrs ← emptyList()
gridOfSpheres ← emptyGrid()
for currentSphere ← transverse(frontSpheres) do
    add(currentSphere,gridOfSpheres)
end for
while frontSpheres is not empty do
    currentSphere ← retrieve(frontSpheres)
    R_new ← selectARadius(F() ,R_min , R_max , lprs )
    box ← computeNeighbourhoodBox(currentSphere,R_new ,R_max )
    neighboringSpheres←gatherSpheresThatInstersect(gridOfSpheres,box)
    currentHalo ← generateHalo(currentSphere,R_new )
    candidatePoints ← emptyList()
    for currentNeighbor ← transverse(neighboringSpheres) do
        neighborHalo ← genereteHalo(currentNeighbor,R_new )
        newPoints ← intersectHalos(currentHalo,neighborHalo,N_p )
        candidatePoints + = newPoints
    end for
    numberOfValidPoints ← 0
    bestPoint ← aVeryBadPoint()
    while candidatePoints is not empty do
        currentPoint ← remove(candidatePoints)
        newSphere ← sphere(currentPoint,R_new )
        if checkIfInside(mesh,newSphere) and
not intersectAny(neighboringSpheres,newSphere) then
            numberOfValidPoints++
            if currentPoint is better than bestPoint then
                bestPoint ← currentPoint
            end if
        end if
    end while
    if numberOfValidPoints > 0 then
        bestSphere ← sphere(bestPoint,R_new )
        addAtBeginning(bestSphere,frontSpheres)
        addAtBeginning(bestSphere,assembly)
        add(bestSphere,gridOfSpheres)
        empty lnrs into lprs
    else
        add(R_new , lnrs )
    end if
    if numberOfValidPoints < 2 then
        remove(currentSphere,frontSpheres)
    end if
end while
```

4
Initial Fronts Generation

This chapter presents an algorithm to generate the initial sphere front for an arbitrary object defined by a triangular mesh that represents its external boundary. The proposed algorithm starts with the computation of the distance field of the mesh using the ideas presented by Fuhrmann et al (13). The distance field is used here to determine if any point in space, or the sphere generated in that place, is inside, outside or intersecting any triangle in the mesh. To illustrate the proposed algorithm let's consider the 2D mesh represented in Figure 4.1.

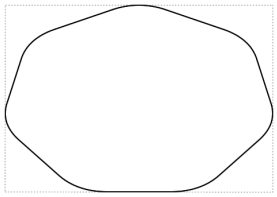


Figure 4.1: Convex boundary and its bounding box.

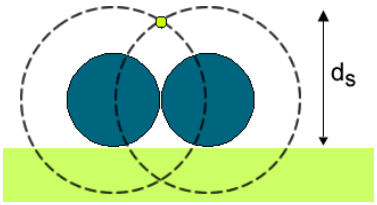
Note also that the algorithm is not restricted to convex boundaries, although, for simplicity, the explanation uses convex figures.

4.1
Grid of distance points

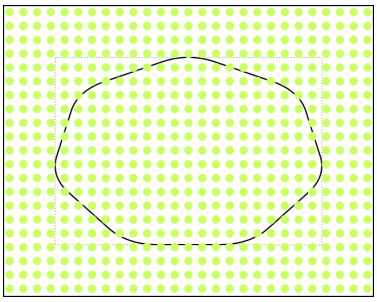
The first step creates the grid of points that holds the distances to the boundary. The distance field must enclose the object and its vicinity where spheres may be created. To this end the distance field is created inside an axis aligned boundary box with a margin $MAABB$ to enclose the initial sphere front. If we assume the front is composed of spheres with R_{max} , the margin d_s can be given by:

$$d_s \geq 3R_{max} \tag{4-1}$$

The grid resolution is defined by the number of points in each Cartesian direction: d_x, d_y and d_z. To store and accelerate the point search it is used a uniform grid, storing a single point inside each voxel.



4.2(a): Minimum margin



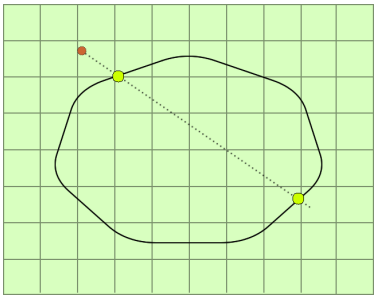
4.2(b): Distance points inside

Figure 4.2: Margin AABB.

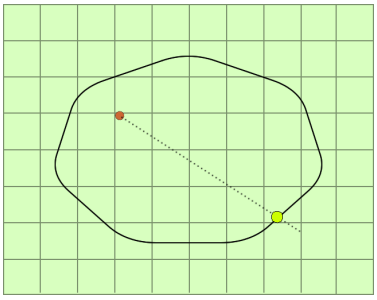
The goal in this first step is to compute, for each grid point, the distance to the boundary and the triangle that defines this distance, i. e.: the closest triangle. Initially all distances in the interior points are assigned a negative infinite value and the ones outside the mesh a positive infinite value.
This assignment can be performed with a "point in polyhedron" test based in the even-odd rule. From each distance point a ray in an arbitrary direction is created.

$$R = \vec{R_{start}} + t(\vec{R_{dir}}), 0 \leq t \tag{4-2}$$

If the ray intersects an even number of triangles in the mesh (Figure 4.3(a)), the point is outside. But if the number of intersections is odd, then the point is inside (Figure 4.3(b)).



4.3(a): Two intersections.



4.3(b): One intersection.

Figure 4.3: 2D Point in polyhedron tests.

Figure 4.4 shows a ray R intersecting a triangle mesh representing a torus. The ray, has an axis aligned bound box with a small margin only in the Y and Z axis used to gather all the triangle candidates for a further intersection test.
To simplify the triangle search we decide to trace rays using the X system axis direction ($\vec{R_{dir}}$ =(1,0,0)). The second and most significant advantage of this choice is that now the problem becomes a "point in triangle" test. This is graphically described in Figure 4.5. The triangles are projected into a YZ plane where the ray becomes a point and the ray-triangle intersection in 3D is reduced to a 2D problem.

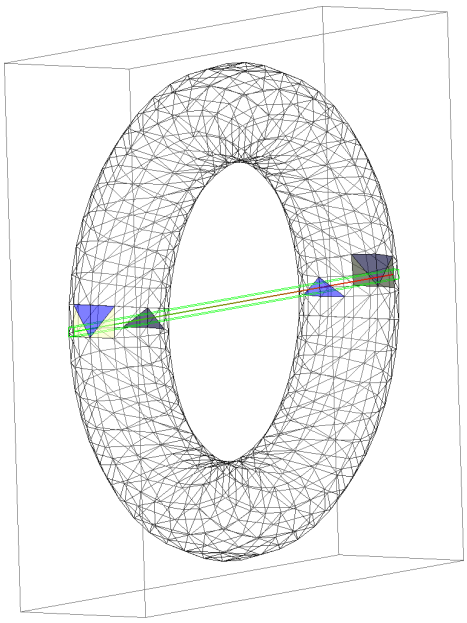


Figure 4.4: Ray traced through a torus

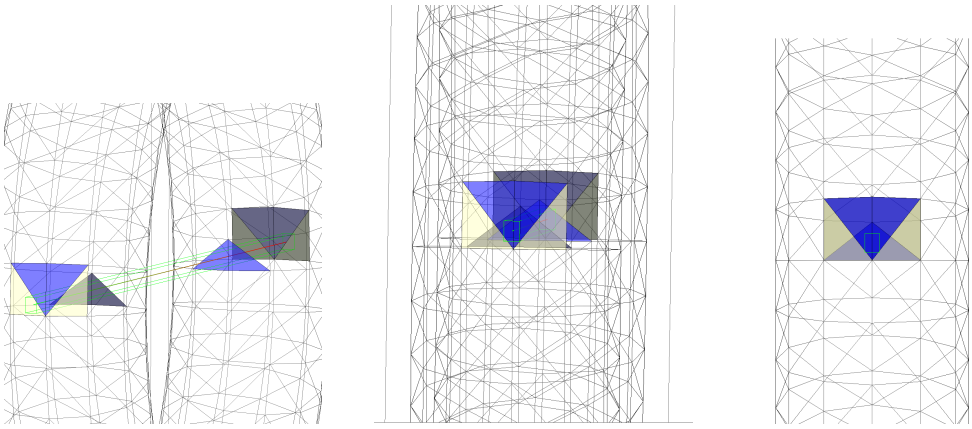


Figure 4.5: From 3D to 2D

We found it convenient to implement the "point in triangle" test computing the barycentric coordinates of the ray because in this way it is easy to detect an edge or vertex collision. Algorithm 4 describes the procedure to compute the barycentric coordinates, note that only the Y and Z coordinates of the input points were used.

Every time the ray R executes a "point in triangle" test and finds collisions, these are classified as: vertex, edge or face. The detected 2D collisions should not be accepted as 3D collisions yet, because our choice to only consider the Y and Z values and discard the X coordinate would probably lead to the computation of a "point in triangle" for a point placed in a lower X coordinate value than $\vec{R_{start}}$ .x, more likely belonging to an indexed triangle in the same voxel as the $\vec{R_{start}}$. The filtering of valid intersections must verify that the corresponding 3D intersection point has a higher or equal X value than $R_{start}.x$.

Algorithm 4 2D Baricentric Coordinates

Input: $\vec{P}$, $\vec{A}$, $\vec{B}$, $\vec{C}$
Output: **u**, **v**, **w**

$\vec{a}$ = (0, 0)
$\vec{b}$ = ($\vec{B}$.y - $\vec{A}$.y, $\vec{B}$.z - $\vec{A}$.z)
$\vec{c}$ = ($\vec{C}$.y - $\vec{A}$.y, $\vec{C}$.z - $\vec{A}$.z)
$\vec{p}$ = ($\vec{P}$.y - $\vec{A}$.y, $\vec{P}$.z - $\vec{A}$.z)
d = $\vec{b}$.x * $\vec{c}$.y - $\vec{c}$.x * $\vec{b}$.y;
u = ($\vec{p}$.x * ($\vec{b}$.y - $\vec{c}$.y) + $\vec{p}$.y * ($\vec{c}$.x - $\vec{b}$.x) + d) / d;
v = ($\vec{p}$.x * $\vec{c}$.y - $\vec{p}$.y * $\vec{c}$.x) / d;
w = ($\vec{p}$.y * $\vec{b}$.x - $\vec{p}$.x * $\vec{b}$.y) / d;

If the 2D collision falls onto a vertex, the 3D X coordinate belongs to the 3D triangle vertex. For edges and faces extra calculations must be made. For the further equations we denote $\vec{R'}$, the projection of the ray R on the edge or face.

If the 2D collision belongs to an edge (Figure 4.6(a)), we consider the parametric equation:

$$\vec{R'} = \vec{V_1} + t(\vec{V_2} - \vec{V_1}), 0 \leq t \leq 1 \tag{4-3}$$

Decomposed in its three components:

$$\begin{aligned} \vec{R'}.x &= \vec{V_1}.x + t(\vec{V_2}.x - \vec{V_2}.x), 0 \leq t \leq 1 \\ \vec{R'}.y &= \vec{V_1}.y + t(\vec{V_2}.y - \vec{V_2}.y), 0 \leq t \leq 1 \\ \vec{R'}.y &= \vec{V_1}.z + t(\vec{V_2}.z - \vec{V_2}.z), 0 \leq t \leq 1 \end{aligned} \tag{4-4}$$

Note that both $\vec{R'}.y$ and $\vec{R'}.z$ are known because the Y and Z values of the ray remain constant. The value of t can be found using any of following two equations, depending on the edge gradient. By substitution of t in 4-4 we can get the value of $R'.x$

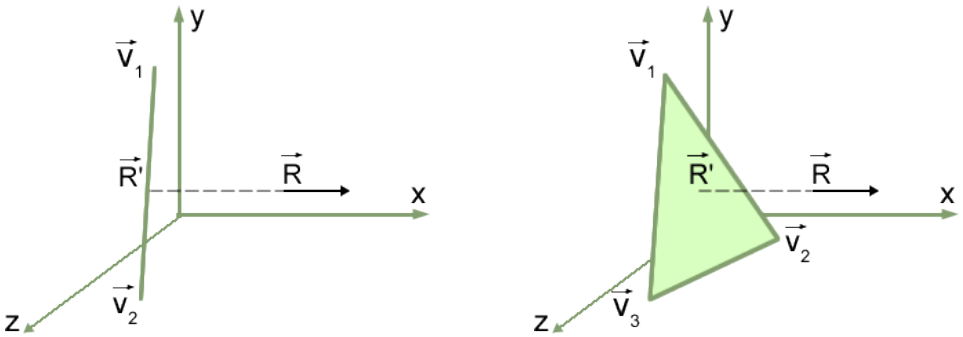
$$t = \begin{cases} \frac{R'.y - \vec{V_1}.y}{\vec{V_2}.y - \vec{V_1}.y}, & \text{if } \vec{V_2}.y \neq \vec{V_1}.y \\ \frac{R'.z - \vec{V_1}.z}{\vec{V_2}.z - \vec{V_1}.z}, & \text{otherwise} \end{cases} \tag{4-5}$$

If the 3D collision belongs to a face (Figure 4.6(b)), we consider the plane equation of the triangle which uses the triangle normal $\vec{n}$:

$$\vec{n} \cdot (\vec{V_1} - \vec{R'}) = 0 \tag{4-6}$$

In this equation only the $\vec{R'}.x$ is unknown so we express:

$$\vec{R'}.x = \vec{V_1}.x + \frac{\vec{n}.y(\vec{V_1}.y - \vec{R'}.y) + \vec{n}.z(\vec{V_1}.z - \vec{R'}.z)}{\vec{n}.x} \tag{4-7}$$



4.6(a): Edge behind the ray

4.6(b): Triangle behind the ray

Figure 4.6: Invalid collisions

In 3D, when the ray intercepts the triangle in one of its edges or vertex the algorithm finds two or more triangles at the same point, causing an erroneous count of mesh intersections by the "point in polyhedron" algorithm. Kalay (23) suggests a strategy to decide whether this point should be accounted for or not computing the inner product of the ray direction and the outward normal of the triangle. If the result is positive the ray is leaving the region. If it is negative the ray is entering it. If in all the intercepting triangles at a point the ray is either entering or leaving the algorithm counts one interception. But if some are leaving and others are entering it counts no interception. In this case is assumed that the ray is tangent to the mesh at that point.

For each ray R two hash maps are managed; one for the ray-vertex intersections $vertexMap$ and another for the ray-edge intersections $edgeMap$. If the barycentric coordinates correspond to a vertex, the triangle vertex is inserted in its respective map along with the dot product of $\vec{R_{dir}}$ with the triangle normal, but since the value of $\vec{R_{dir}}$ =(1,0,0) is constant, then the result will always be the X component of the normal. The same logic is performed for an edge intersection, but inserting the intersecting edge. Only if the ray hits the face of a triangle then the counter for the number of intersections is incremented. Finally, the list of dot products of each map must be checked. Only if the sign of all the values in each list is the same then we consider the intersection and increment the counter. Algorithm 5 summarizes the process.

Algorithm 5 Point in polyhedron

Input: $\vec{P}$, triangleGrid

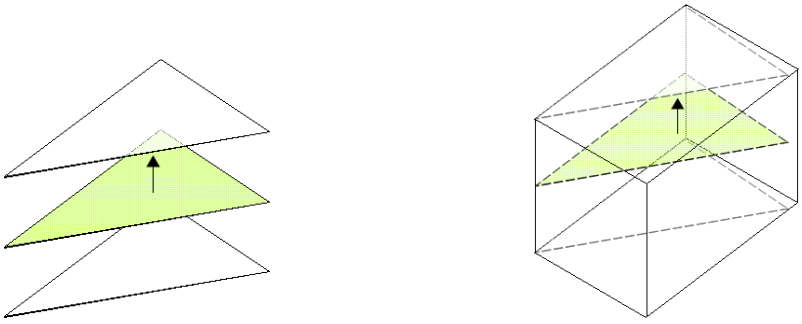
```
intersections ← 0
vertexMap ← emptyMap() //key: common vertex, value: list of dot products
edgeMap ← emptyMap() //key: common edge, value: list of dot products
rayIni ← P
rayEnd ← (triangleGrid.max.x P.y, P.z)
rayDir ← (1,0,0)
rayBox ← computeMarginYZAABB(rayIni, rayEnd)
foundTriangles ← gatherTrianglesThatInstersect(triangleGrid, rayBox)
for currentTriangle ← transverse(foundTriangles) do
    A ← GetVertex(currentTriangle, 0)
    B ← GetVertex(currentTriangle, 1)
    C ← GetVertex(currentTriangle, 2)
    (u, v, w) ← Get2DBaricentricCoordinates(P, A, B, C)
    if (0 ≤ u ≤ 1) AND (0 ≤ v ≤ 1) AND (0 ≤ w ≤ 1) then
        if u == 0 AND v == 0 AND C.x ≥ rayIni.x then
            AddToMap(vertexMap, C, currentTriangle.normal.x)
        else if u == 0 AND w == 0 AND B.x ≥ rayIni.x then
            AddToMap(vertexMap, B, currentTriangle.normal.x)
        else if w == 0 AND v == 0 AND A.x ≥ rayIni.x then
            AddToMap(vertexMap, A, currentTriangle.normal.x)
        else if u == 0 AND GetXValue(BC, rayIni) ≥ rayIni.x then
            AddToMap(edgeMap, BC, currentTriangle.normal.x)
        else if v == 0 AND GetXValue(AC, rayIni) ≥ rayIni.x then
            AddToMap(edgeMap, AC, currentTriangle.normal.x)
        else if w == 0 AND GetXValue(AB, rayIni) ≥ rayIni.x then
            AddToMap(edgeMap, AB, currentTriangle.normal.x)
        else if GetXValue(A, currentTriangle.normal) ≥ rayIni.x then
            intersections++
        end if
    end if
end for
for (V_c, listOfDotProducts) ← transverse(vertexMap) do
    if AllSignsAreEqual(listOfDotProducts) then
        intersections++
    end if
end for
for (E_c, listOfDotProducts) ← transverse(edgeMap) do
    if AllSignsAreEqual(listOfDotProducts) then
        intersections++
    end if
end for
if intersections MOD 2 == 0 then
    return outside
else
    return inside
end if
```

4.2 Mesh shell creation

Prior to the creation of the distance field the algorithm constructs an outer and an inner shell of the mesh. To accomplish this, a prism is built for every triangle, see Figure 4.7(a). The top and bottom base of the prism are, respectively, the reference triangle translated of positive $+\epsilon$ and negative $-\epsilon$ and the direction of its normal. Note that the use of ϵ for both the margin of the distance field and the outer shell ensures that the latter is inside the distance field grid.

$$+\epsilon \leq d_s \tag{4-8}$$

The algorithm also stores the prism bounding box for future reference (see Algorithm 6).



4.7(a): Prism generation.

4.7(b): Prism bounding box.

Figure 4.7: Triangles translation.

The result of the prisms creation generates two shells as presented in Figure 4.8, this will serve to identify the closer points with positive distances (those who are outside the mesh) and negative distances (those who are inside the mesh).

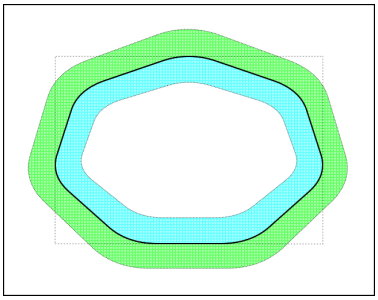


Figure 4.8: Generated shells after the prisms generation.

Algorithm 6 Processing of Triangles

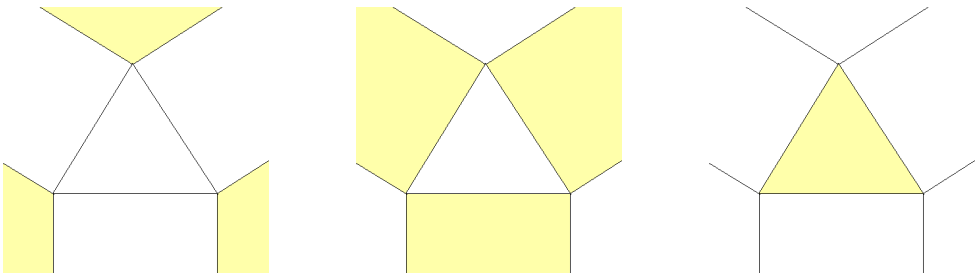
Input: triangle, $+\epsilon$, $-\epsilon$.

```
triangle.normal ← ComputeNormal(triangle)
t1 ← CreateTriangle(triangle, +ϵ )
t2 ← CreateTriangle(triangle, -ϵ )
triangle.box ← computeAABB(t1,t2)
```

4.3 Distance field calculation

To calculate the distance field we iterate over the list of mesh triangles. For each triangle is used the bounding box, computed in the shells creation, intersecting it with the uniform grid of points. In this way we gather all the distance points of the intersecting voxels inside the bounding box.

For each point its distance to the closest point in the current triangle is computed. One efficient way to find the closest point is using the voronoi regions (Figure 4.9). The point P is orthogonally projected onto the triangle plane. Then, it is checked to which region the point projection P_p belongs. If it is a vertex region, then the closest point is the corresponding triangle vertex. If the point falls onto an edge region, then the closest point is one of the edge points and if the point falls onto the face region the closest point is the point P_p itself. For a better understanding of how to determine if a point belongs to a vertex or an edge voronoi triangle region we suggest to review (10) that also discusses some optimizations to reduce expensive vector cross products.



4.9(a): Vertex regions 4.9(b): Edge regions 4.9(c): Face region

Figure 4.9: Triangle voronoi regions.

The sign of the distance depends on the position of the point in relation to the normal of the triangle (Figure 4.10). If the absolute value of the recently obtained distance is smaller than the current absolute distance of the point to the mesh, then the distance and the triangle reference is replaced for the point. A sign replacement issue may occur due to the successive processing of the triangles updating closer points shared by more than one triangle. Fuhrmann (13) explains that when two triangle normals form a convex angle, changing a

Algorithm 7 Closest triangle point from Point

Input: triangle, point.
Output: closesTrianglePoint.

```
voronoiRegions ← computeVoronoiRegions(triangle)
pointInPlane ← orthogonalProjection(triangle, point)
if pointInPlane in vertexRegion(voronoiRegions) then
    vertex = getVertex(voronoiRegions, pointInPlane)
    closesTrianglePoint ← vertex
else if pointInPlane in edgeRegion(voronoiRegions) then
    edge = getEdge(voronoiRegions, pointInPlane)
    closesTrianglePoint ← getPointInEdge(edge, pointInPlane)
else
    closesTrianglePoint ← pointInPlane
end if
```

4.10(a): Positive distance (above the triangle)

4.10(b): Negative distance (below the triangle)

Figure 4.10: Distance sign.

common point from a positive to a negative value is allowed while the opposite is not. And when the triangle normals form a non-convex angle, the change from a negative to positive distance is allowed, while the change from a positive to a negative distance is forbidden.

After the execution of this step we can see in Figure 4.11 the appearance of four differentiated areas; there is one with infinite negative values, another with positive infinite values, a third with finite negative values and a last one with finite positive values. Now we can affirm that if the distance of any point is negative then it must be inside the mesh.

We summarized the distance field calculation in Algorithm 8.

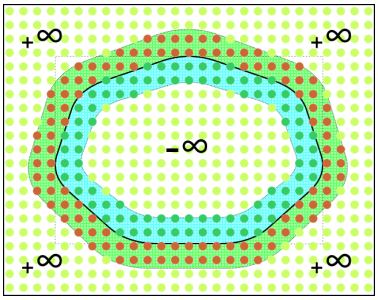


Figure 4.11: Distance field.

Algorithm 8 Distance Field Calculation

Input: meshContainer, (d_x, d_y, d_z) , R_{max} , $+\epsilon$, $-\epsilon$.

```
MAABB ← computeMAABB(meshContainer, R_max )
distancePoints ← definePointsLocation(MAABB , (d_x, d_y, d_z) )
computeInitialDistances(distancePoints, meshContainer)
for currentTriangle ← transverse(meshContainer) do
    trianglePreprocessing(currentTriangle,+ϵ ,-ϵ )
    closerDistancePoints ← findPointsThatIntersect(distancePoints,
currentTriangle.box)
    while closerDistancePoints is not empty do
        currentPoint ← remove(closerDistancePoints)
        closestTrianglePoint ← findClosestPoint(currentTriangle,
currentPoint)
        newDistance ← (closestTrianglePoint - currentPoint).length()
        if absoluteValue(currentPoint.distance) > newDistance then
            sign ← getSign(closestTrianglePoint, currentTriangle.normal,
currentPoint)
            currentPoint.distance ← newDistance * sign
            currentPoint.triangle ← currentTriangle
        end if
    end while
end for
```

4.4 Initial front selection

To proceed with the initial front generation (Algorithm 9), we start by filling the $MAABB$ with spheres. It can be used just a regular package, like the one in Figure 4.12, or a package resulting from the assembly generation described in the previous chapter in the case of the intention to create a front composed by particles of different radii.

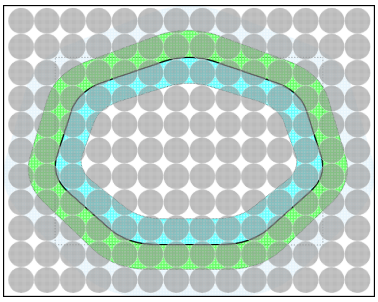


Figure 4.12: Package configuration inside the MAABB.

Next, only those particles placed within a certain distance M_p to the mesh are selected according to the following restriction:

$$0 < M_p \leq min(-\epsilon, +\epsilon) \tag{4-9}$$

The mesh proximity value can not be higher than the minimum factor of displacement because those points beyond the mesh shells will have an infinite value and we will not be able to adjust their positions.
Figure 4.13 illustrates a threshold distance of twice a particle radius to admit spheres from inside and outside the mesh. The gray particles represent the ones that were rejected. The higher the threshold the more spheres are chosen to be part of the front.

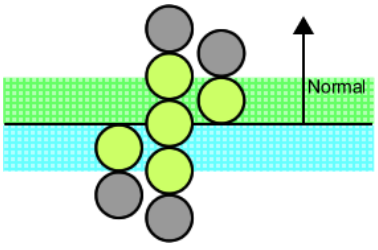


Figure 4.13: Finding the closer particles with equal M_p, $+\epsilon$ and $-\epsilon$).

To compute the distance of an arbitrary point in space we use a trilinear interpolation of the closer distance points (Figure 4.14).
In Figure 4.15 we can see that for each circle center are gathered all the closer distance points. Then all the triangles in the mesh related to this set of points

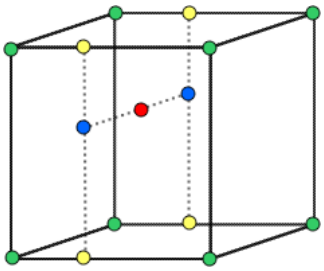


Figure 4.14: Trilinear interpolation.

are identified. Next we find the closest triangle to the circle center. Finally, the circle position is adjusted using the closet mesh triangle normal.

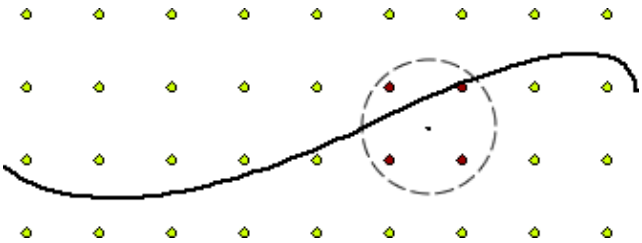


Figure 4.15: Trilinear interpolation.

The result is shown in Figure 4.16. We must remark that the initial front assembled here will not be part of the final sphere pack. The front will be used as seed for the packing generation and along with the distance field it will work as a barrier to contain the pack. Additionally, the interpenetrations between them are not handled and do not constitute an obstacle.

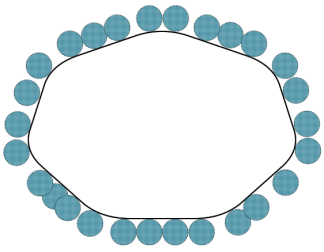


Figure 4.16: Generated Initial Fronts.

Algorithm 9 Initial Fronts Generation

Input: mesh, distanceField, M_p , R_{min} , $F()$, N_p .
Output: meshFronts

```
meshFronts ← empty()
boxFronts ← generateFronts(distanceField.MAABB)
spheres ← assemblyGenerator(mesh,boxFronts,R_min ,R_min ,F() ,N_p )
while spheres is not empty do
    currentSphere ← remove(spheres)
    closerPoints ← findCloserPoints(distanceField,currentSphere.position)
    distance ← trilinearInterpolation(closerPoints,currentSphere.position)
    if absoluteValue(distance) <= M_p then
        minDistance ← aVeryBadPoint()
        while closerPoints is not empty do
            currentPoint ← remove(closerPoints)
            triangle ← currentPoint.triangle
            closestTrianglePoint ← findClosestPoint(triangle,
currentSphere.position)
            newDistance ← (closestTrianglePoint -
currentSphere.position).length()
            if absoluteValue(minDistance) > newDistance then
                minDistance ← newDistance
                normal ← triangle.normal
            end if
        end while
        currentSphere.position += normal * (currentSphere.radius -
minDistance)
        meshFronts.addAtBeginning(currentSphere);
    end if
end while
```

5
Results

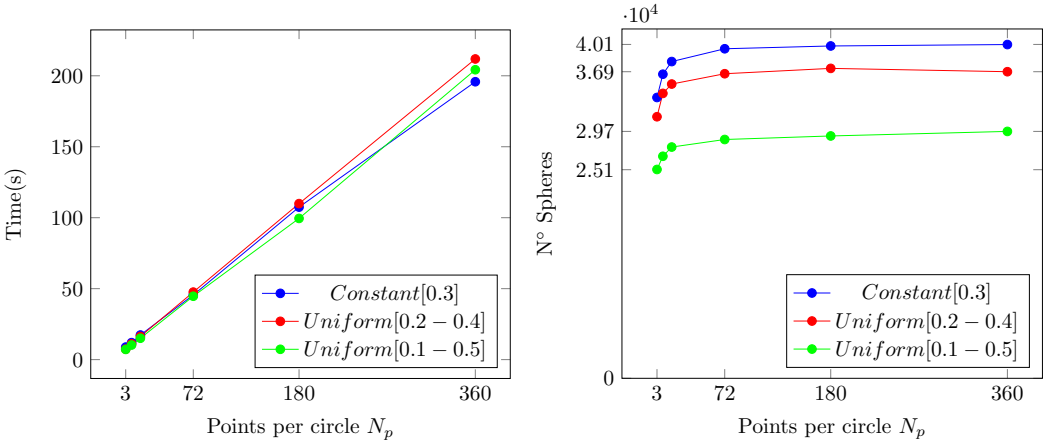
This chapter presents test results based on a C++ implementation of the algorithms described in this dissertation running on an Intel Core i5-3330 @ 3.00GHz with 8GB of RAM memory under Windows 7 64-bit.
In the first part of this chapter we use rectangular containers to evaluate several aspects of the proposed algorithm. We start with the evaluation of the influence of the number of points per halo intersection described in the assembly generation both in time and in number of generated spheres. Next, we study the density, porosity and coordination number of the package generated with the proposed algorithm with different radii distribution functions. To conclude the study with rectangular containers we present some results regarding its efficiency.
The second part of the chapter presents results for containers with arbitrary geometry: cylinder, capsule, torus, bunny and knot. We start with the evaluation of the density of the package generated with the proposed algorithm with a constant and a uniform radii distribution function.
Finally the third part presents results for the point in polyhedron algorithm.

5.1
Points in halo intersection

In this test we fill a box of dimension (10, 10, 10) with several particle size distributions. For each assembling pack we vary the numbers of equidistant points in the circle to test its influence both in processing time and in the final assembly. We consider here three types of packs: one with a constant radius size (0.3), another with a uniform radii varying between [0.2 - 0.4] and a third pack with a uniform radii varying between [0.1 - 0.5] to test a ratio of 1:5.
Table 5.1 exhibits the results (time and number of particles) for the three scenarios. In Figure 5.1(a) we can see that the time grows linearly with the number of points in the circle for all cases. That was expected because the number of candidate points is in the core of the algorithm.
With respect to the number of particles Figure 5.1(b) shows that with more points there are more spheres. This was also expected, with more points

Table 5.1: Pack results varying the number of circle points in a box of (10x10x10)

Points	**Constant [0.3]**		**Uniform [0.2-0.4]**		**Uniform[0.1-0.5]**	
	N° of Spheres	**Time**	**N° of Spheres**	**Time**	**N° of Spheres**	**Time**
3	33762	8.8	31446	7.4	25107	7.2
9	36560	12.0	34254	11.1	26694	10.3
18	38081	17.3	35378	16.3	27803	15.1
72	39619	45.5	36618	47.6	28710	44.7
180	39955	107.	37260	110.	29136	99.5
360	40126	196.	36861	211.	29683	204.



5.1(a): Points vs Time(s)

5.1(b): Points vs N° Spheres

Figure 5.1: Tests varying N° points in halo intersection

increases the probability of finding a valid position. The curves in the figures indicate that over 72 points there is no gain in increasing them. Using 360 candidates creates nearly the same number of spheres than using 180 points, and consumes a much more time.

In the tests that follow in this chapter we use 72 points per halo intersection.

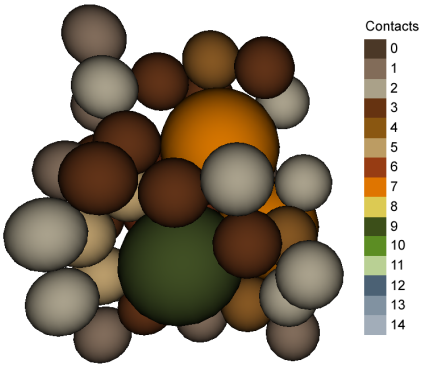


Figure 5.2: Number of contacts per particle.

5.2
Packages for rectangular containers

The quality analysis of the generated assemblies is based on conventional geometric properties also presented in (20, 6, 28, 24).

– The density (d) or volume fraction, is the ratio between the total volume of all the N particles in the assembly and the container volume V_c. Considering that we only use spherical particles:

$$d = \frac{1}{V_c} \sum_{i=1}^{N} \frac{4}{3} \pi r_i^3 \tag{5-1}$$

– The porosity (p), is the ratio between the total volume of the void spaces not occupied by the particles and the container volume V_c. Since the addition of all the particles and void spaces volumes gives the container volume this property is calculated as:

$$p = 1.0 - d \tag{5-2}$$

– The mean coordination number (z), characterizing the local density of the packing. The coordination number is a sphere property representing the number of neighbor particles in contact (c). The mean is computed with:

$$z = \frac{1}{N} \sum_{i=1}^{N} c_i \tag{5-3}$$

Even though some works (1) contemplate the contacts among the particles and the container walls, here we just count the inter particle contacts taking advantage of the uniform grid of particles created in the packing algorithm (Figure 5.2).
To claim a sphere S_1 in contact with another sphere S_2 it is considered a threshold distance of 5% the sum of both radii.

The following test cases are composed by rectangular boxes with different dimensions similar to the experiments in (34). The dimension of a container is represented by the number of particles of the maximum radius in each dimension. For every radii distribution function we present the pack density, porosity, mean coordination number, contact frequency and when applies the radii frequency.

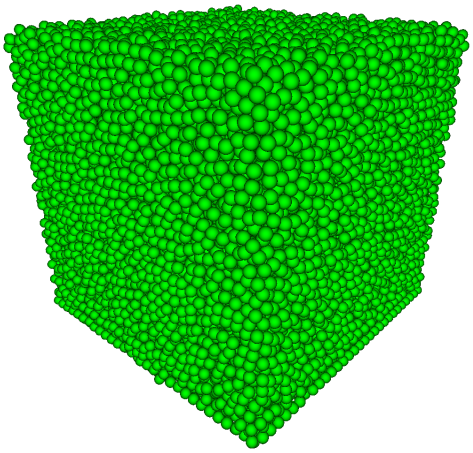
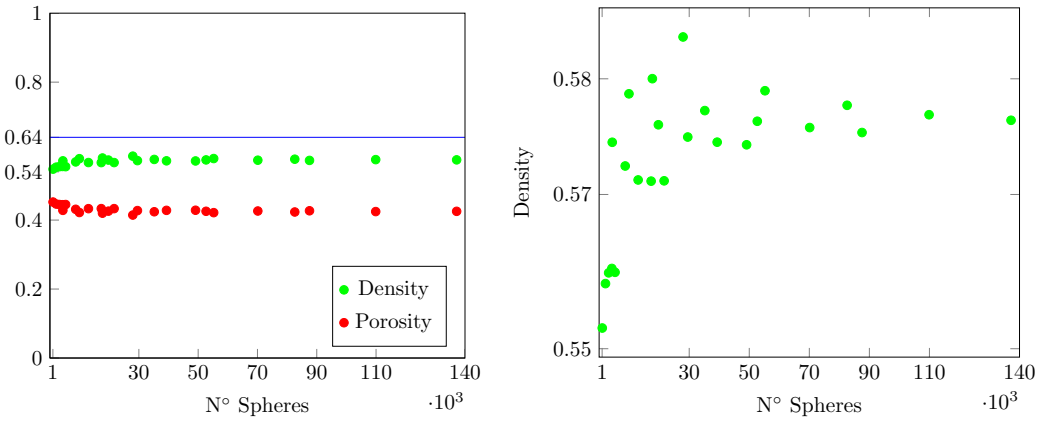


Figure 5.3: Constant radius (0.08) - 30x30x30 pack - 29519 spheres.

Equal distribution

The first test considers an assembly with spheres of the same radius. Figure 5.4 presents in the left side the resulting density and porosity values and in the right side a closer view of the density values between [0.547 - 0.585]. Considering that with experimental tests it is reach a maximum density of approximately 0.64 for packs with constant particle radius (18) our algorithm achieves values with a difference of less than 0.1. Table 5.2 shows the achieved coordination number for each rectangular box tested. According to Bennett's hypothesis (4), the mean number of contacts to guarantee the stability in a mono-disperse pack is 6. Figure 5.5(b) plots the values between [6.17 - 6.91] and Figure 5.5(a) shows the number of contact occurrences in a 30x30x30 pack. We can say then that our solution obtains stable arrangements.



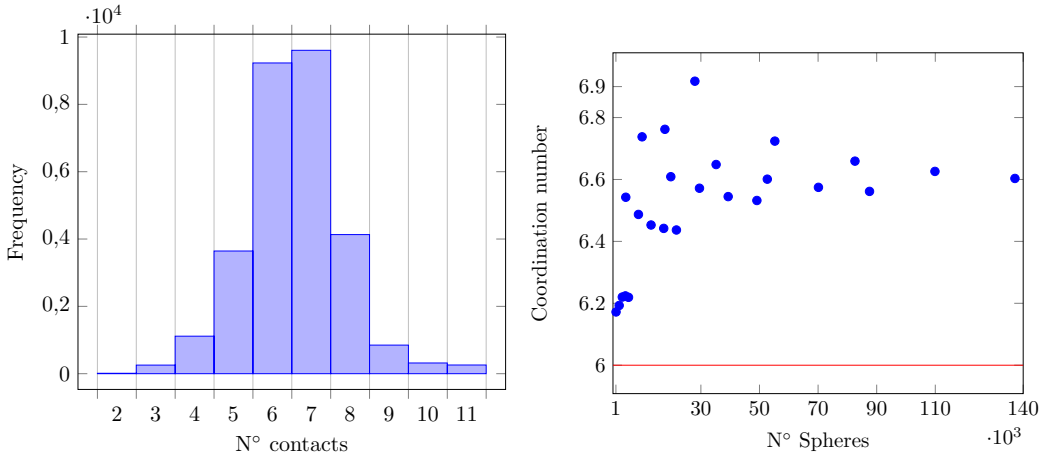
5.4(a): Wider view

5.4(b): Narrow view

Figure 5.4: Constant radius (0.08) - Density and porosity ratios

Table 5.2: Equal distribution [0.08] - Coordination numbers

Length Width Height	**10x10**	**20x20**	**30x30**	**40x40**	**50x50**
10	6.17	6.54	6.74	6.76	6.92
20	6.19	6.49	6.61	6.65	6.72
30	6.22	6.45	6.57	6.60	6.66
40	6.22	6.44	6.54	6.57	6.63
50	6.22	6.44	6.53	6.56	6.60



5.5(a): Frequency - 30x30x30 pack

5.5(b): Coordination number

Figure 5.5: Contacts

Uniform distribution

With the same container dimensions, assemblies using a uniform radii variation were created with a ratio of 2 between the minimum and maximum radius. The density in a poli-disperse arrangement presents a higher ratio (or lower porosity ratio) than a mono-disperse arrangement because with different particle sizes the algorithm achieves less rejections for the minimum radius in the insertion step, the experiments show that the density, going from 0.583 to 0.589, tends to stabilize at the value of 0.586 (Figure 5.7).

The mean coordination falls between [5.98 - 6.40]. Figure 5.8 maps the frequency of the contact numbers. We can observe the appearance of occurrences for 0, 1, 12 and 13 contacts in comparison with the constant radius size experiment, but still persists the predominance of 6 and 7. Figure 5.9 shows that the uniform distribution was preserved in the final pack.

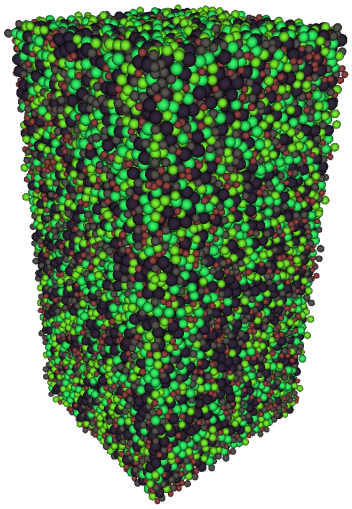
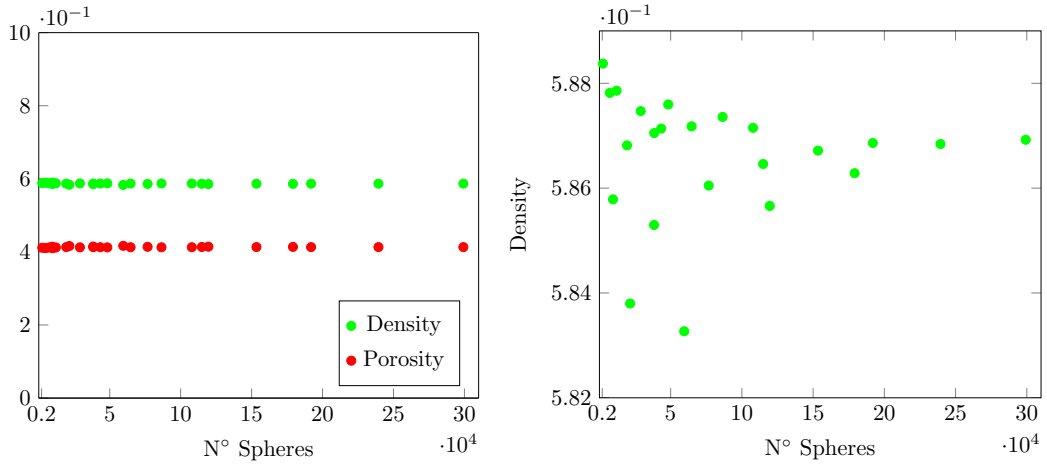


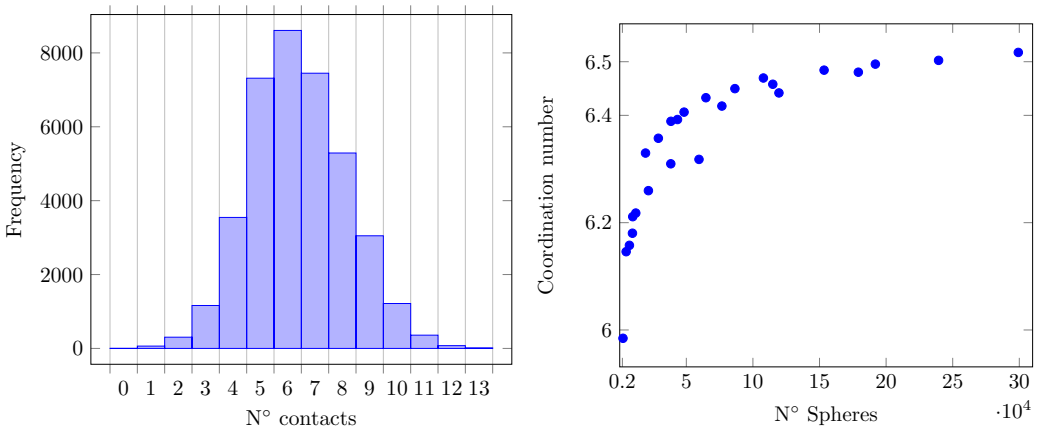
Figure 5.6: Uniform radius variation [0.04-0.08] - 20x40x20 pack - 38454 spheres.



5.7(a): Wider view

5.7(b): Narrow view

Figure 5.7: Uniform radius variation [0.04-0.08] - Density and porosity ratios



5.8(a): Frequency

5.8(b): Coordination number

Figure 5.8: Uniform radius variation [0.04-0.08] - 20x40x20 pack - Contacts

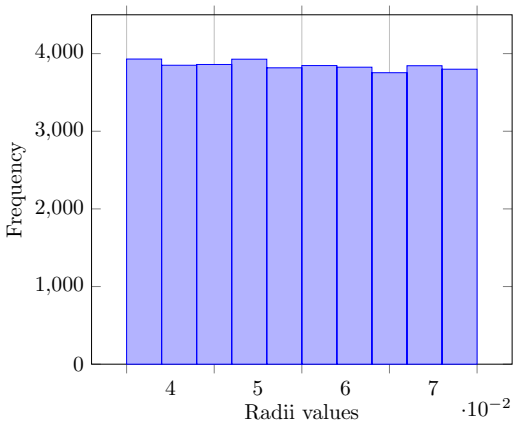


Figure 5.9: Uniform radius variation [0.04-0.08] - 20x40x20 pack - Radii Distribution

Bernoulli distribution

In this type of distribution we only consider two sizes of spheres with a desired probability ρ used as a parameter to create particles with the maximum radius. We explore the packing with a wider variation of the ratio between the radii size (1:2, 1:3, 1:4) for a $\rho = 0.5$. Here the density has the tendency to increase with a bigger container or higher number of particles (Figure 5.11(a)) in the ranges of (1:2)[0.577-0.599], (1:3)[0.561-0.600] and (1:4)[0.553-0.579]. Concerning the mean coordination number (Figure 5.12(b)), it also has the tendency to increase with the number of particles: (1:2)[5.95-6.50], (1:3)[5.69-6.20] and (1:4)[5.59-5.94]. The contact numbers occurrences in Figure 5.12(a) shows the appearance of values from 13 to 17 in comparison to the previous tests.

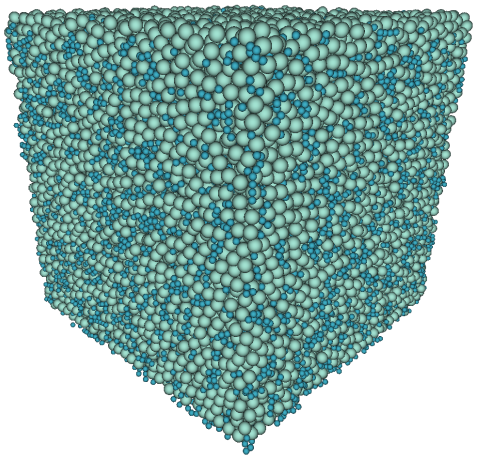


Figure 5.10: Bernoulli test (0.03,0.06) ($\rho = 0.5$) - 30x30x30 pack - 54918 spheres.

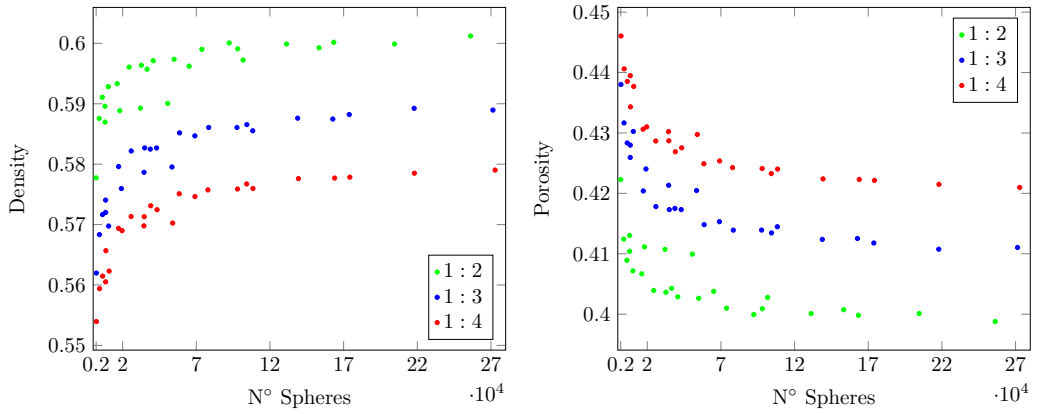


Figure 5.11: Bernoulli test (0.03,0.06) ($\rho = 0.5$) - Density and porosity

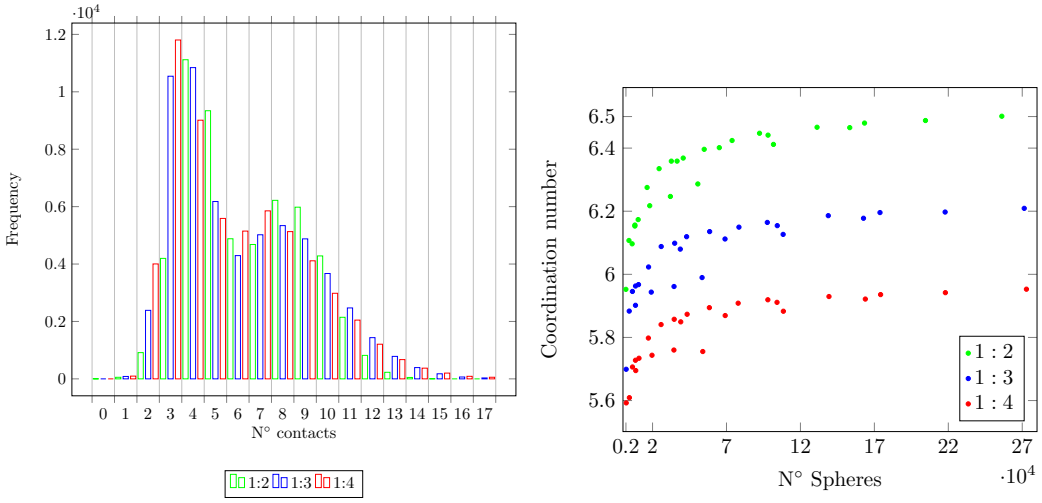


Figure 5.12: Bernoulli test (0.03,0.06) ($\rho = 0.5$) - Number of contacts and coordination number.

To test the contact frequency inside a 30x30x30 pack we use three probabilities ($\rho = 0.25, 0.50, 0.75$) to create particles with the maximum radius. Figure 5.13(a) shows that for each probability were tested three packs varying the ratio between the minimum and maximum radius. Table 5.3 proves that the pack results respect, with a small margin of error, the given probability for the maximum radius. With a higher probability the algorithm inserts fewer particles inside the rectangular boxes because spheres with bigger size occupy more space.

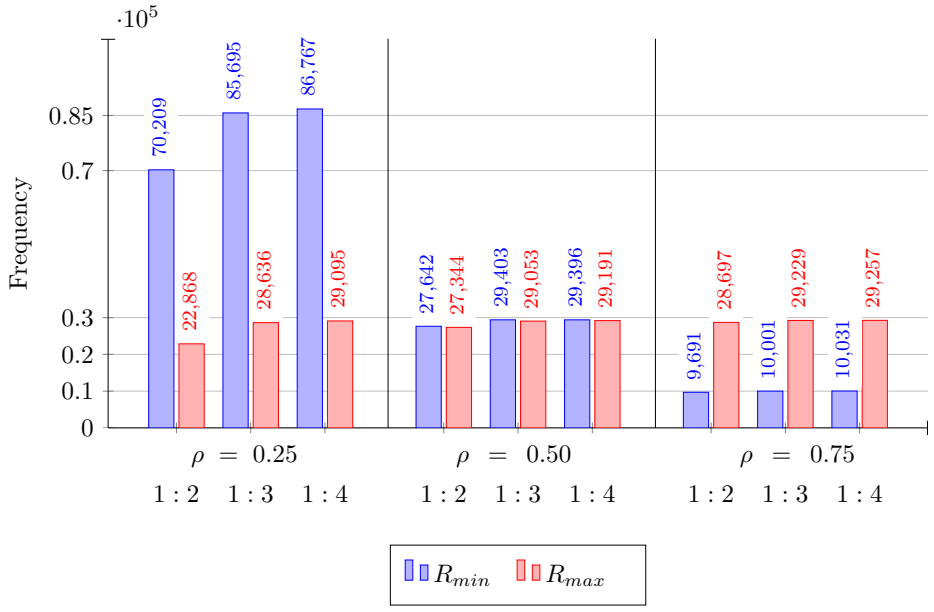


Figure 5.13: Bernoulli test - Pack 30x30x30 - Radii frequencies

Table 5.3: Bernoulli pack with desired ρ and actual ρ_c probabilities

	ρ = **0.25**		ρ = **0.50**		ρ = **0.75**	
Radii	**Particles**	ρ_c	**Particles**	ρ_c	**Particles**	ρ_c
0.030	70209	0.754	27642	0.503	9691	0.246
0.060	22868	**0.246**	27344	**0.497**	29697	**0.754**
	error(0.004)		**error(0.003)**		**error(0.004)**	
0.020	85695	0.750	29403	0.503	10001	0.255
0.060	28636	**0.250**	29053	**0.497**	29229	**0.745**
	error(0.000)		**error(0.003)**		**error(0.005)**	
0.015	86767	0.749	29396	0.502	10031	0.255
0.060	29095	**0.251**	29191	**0.498**	29257	**0.745**
	error(0.001)		**error(0.002)**		**error(0.005)**	

Truncated Gaussian distribution

The Truncated gaussian generator receives as parameters the desired mean μ and standard deviation σ for the radii, besides the R_{min} and R_{max} to discard numbers out of this range. This experiment attempts to create packs with $\mu = 0.05$ and $\sigma = 0.02$ resulting in a density between [0.595-0.605] and a mean coordination number between [6.07-6.50] (Figure 5.16(b)). The contact frequency histogram is presented in Figure 5.16(a).

Figure 5.17 compares the normalized frequency values for a 20x20x20 pack with the Gaussian curve of $\mu = 0.05$ and $\sigma = 0.02$. The analysis of the generated radii presents a current mean $\mu_c = 0.04987$ and current standard deviation $\sigma_c = 0.015$. We observe that the radii sizes were trimmed by the R_{min} and R_{max} values.

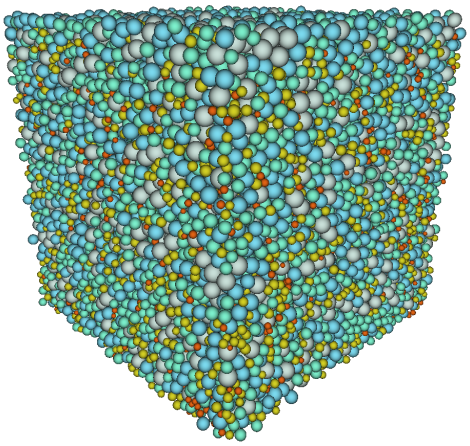
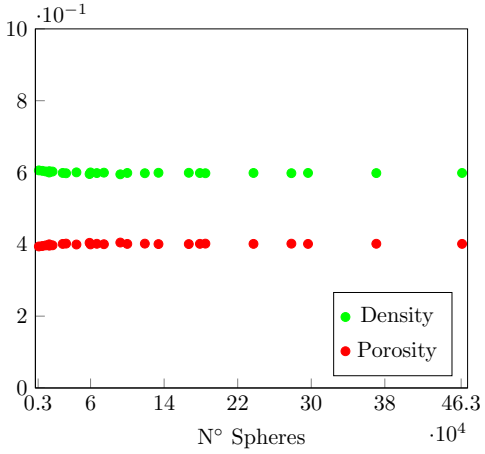
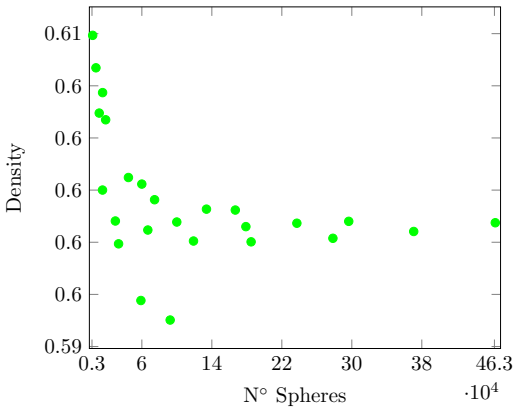


Figure 5.14: Truncated gaussian radius variation [0.02-0.08] [$\mu = 0.05$ and $\sigma = 0.02$] - 20x20x20 pack - 29711 spheres.

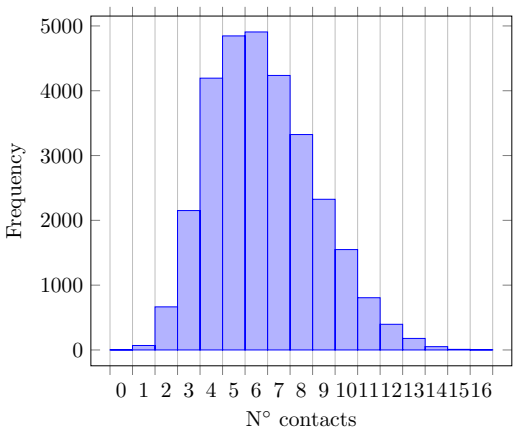


5.15(a): Wide view

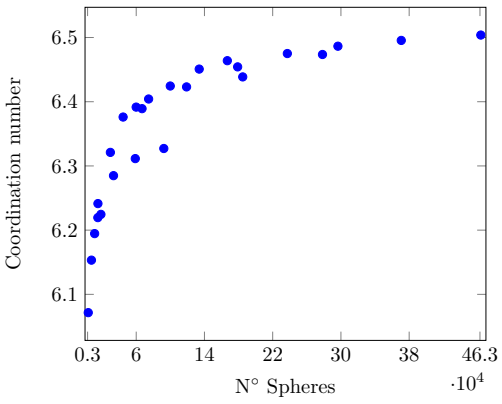


5.15(b): Narrow view

Figure 5.15: Truncated gaussian radius variation [0.02-0.08] [$\mu = 0.05$ and $\sigma = 0.02$] - Density and porosity ratios



5.16(a): Frequency



5.16(b): Coordination number

Figure 5.16: Truncated gaussian radius variation [0.02-0.08] [$\mu = 0.05$ and $\sigma = 0.02$] - Frequencies

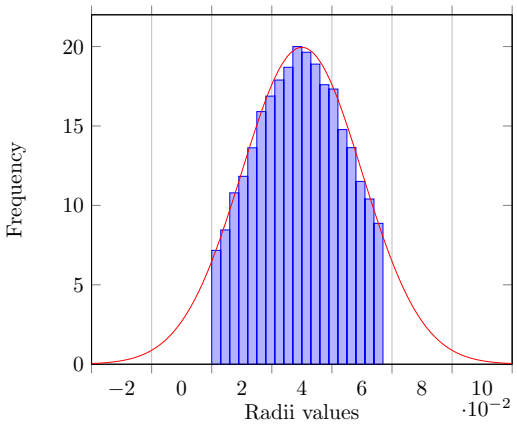


Figure 5.17: Truncated gaussian radius variation [0.02-0.08] [$\mu = 0.05$ and $\sigma = 0.02$] - Curve vs Pack

Another experiment with a wider particle size range [0.1 - 0.3] and desired parameters [$\mu = 0.2$ and $\sigma = 0.05$] displays the following pack radii frequency (Figure 5.18). The particle configuration has a current mean $\mu_c = 0.1994$ and

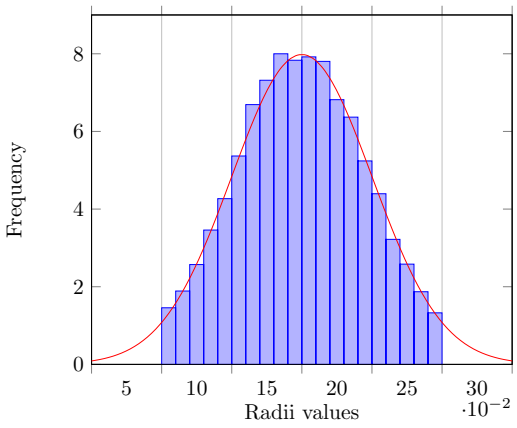


Figure 5.18: Truncated gaussian radius [0.1-0.3] [$\mu = 0.2$ and $\sigma = 0.05$]

current standard deviation $\sigma_c = 0.044$.

Particle rejection

A particular behavior of our packing algorithm is the preference to insert particles with smaller sizes because those have a higher probability to fit into empty spaces not colliding any other previously inserted sphere. That is the justification to use a control mechanism to handle rejected radii and try to insert them again in further loop cycles. We performed some tests without the mechanism of control to examine these situations.

In Figure 5.19 are plotted the radii frequencies for two bernoulli packs adopting $\rho = 0.5$ without particle rejection (filled bars with the same values as in

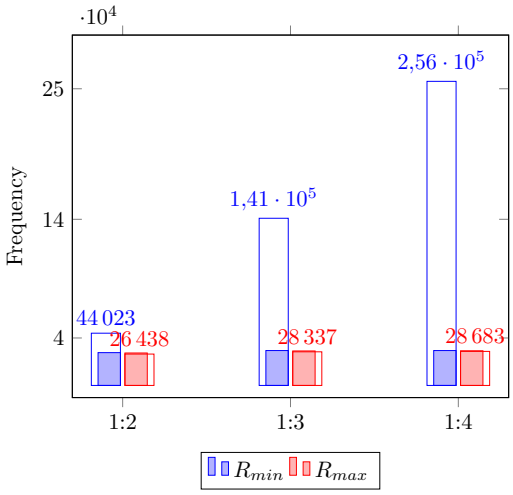


Figure 5.19: Bernoulli 30x30x30 pack ($\rho = 0.5$) - Radii rejection comparison

Figure 5.13) and with rejections (empty wider bars) labeling in this last case the number of particles per radius. Even though we use the probability $\rho = 0.5$, the amount of spheres with R_{min} far exceeds the amount of particles with R_{max} . This difference becomes highly evident with a higher ratio between the minimum and maximum sizes.

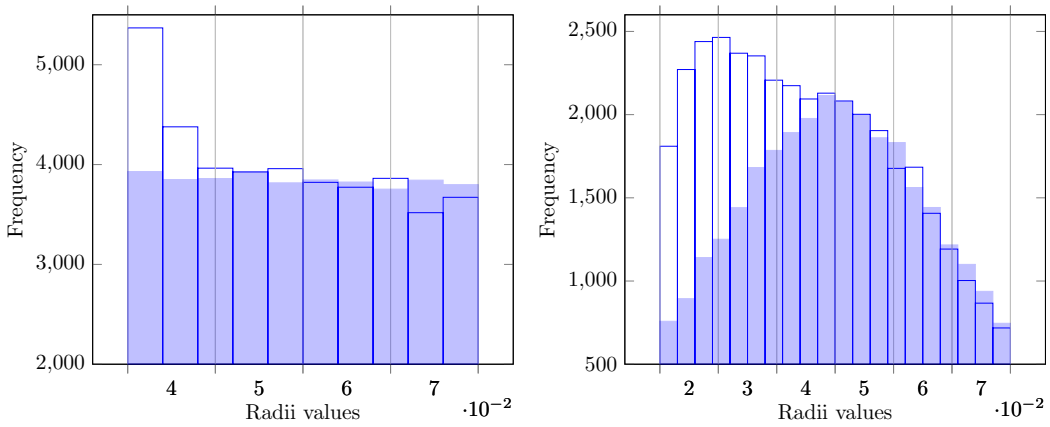
Table 5.4 presents the number of particles in both $lnrs$ and $lprs$ lists after the generation of packs with different ρ probabilities and particle size ratios. The number of rejected radii remaining is very small in comparison to the number of particles in the pack and presents an increase when the ρ probability is higher. For all the test cases in this distribution the rejections lists are composed only of the R_{max} value.

Table 5.4: Bernoulli 30x30x30 pack - N° of rejected particles remaining

		$\rho =$ **0.25**	$\rho =$ **0.50**	$\rho =$ **0.75**
1:2	**N° pack particles**	93077	54986	38388
	lprs + lnrs	474	390	518
	Remaining (%)	**0.51%**	**0.71%**	**1.35%**
1:3	**N° pack particles**	114331	58456	39230
	lprs + lnrs	223	274	467
	Remaining (%)	**0.20%**	**0.47%**	**1.19%**
1:4	**N° pack particles**	115862	58587	39288
	lprs + lnrs	158	265	426
	Remaining (%)	**0.14%**	**0.45%**	**1.08%**

Figures 5.20(a) and 5.20(b) show again, for the Uniform and Truncated gaussian radius variation, that when our algorithm does not use the control mechanism (empty bars) the produced radii do not respect the desired distribution (filled bars) and has the tendency to insert smaller particles.

Tables 5.6 and 5.5 present the number of particles in the rejections lists after



5.20(a): Uniform 20x40x20

5.20(b): Truncated gaussian 20x20x20

Figure 5.20: Uniform and Gaussian packs - Radii rejection comparison

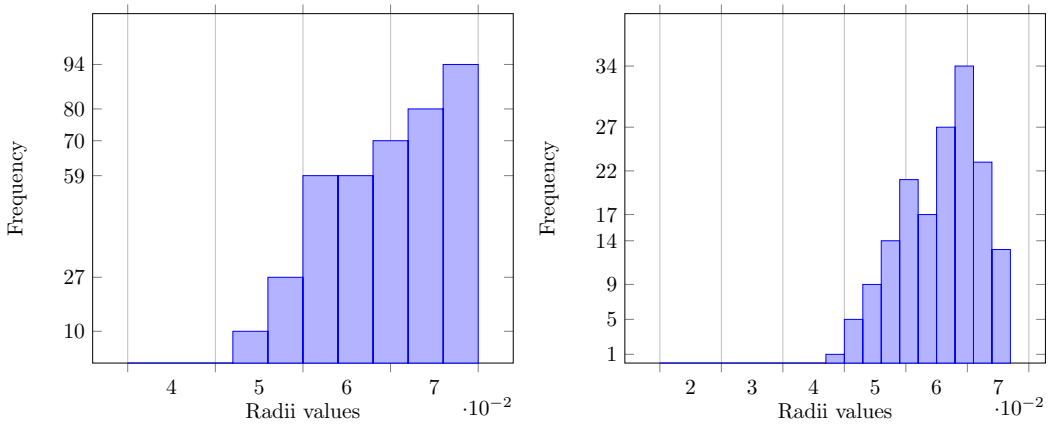
the packs generation. Both numbers represent a small percentage of the total spheres in the pack. Figure 5.21 plots the frequency of the radii occurrences in both experiments. The algorithm tends to reject radii with the higher value.

Table 5.5: Uniform 20x40x20 pack - N° of rejected particles remaining

N° pack particles	38454
lprs + lnrs	399
Remaining (%)	**1.04%**

Table 5.6: Truncated gaussian 20x20x20 pack - N° of rejected particles remaining

N° pack particles	29711
lprs + lnrs	186
Remaining (%)	**0.63%**



5.21(a): Uniform 20x40x20

5.21(b): Truncated gaussian 20x20x20

Figure 5.21: Uniform and Gaussian packs - Rejected radii remaining frequency

Performance

To illustrate the efficiency of the package generation algorithm in rectangular containers this section presents the time required to create packs with each radii distribution function. Figures 5.22 and 5.23 show the required time to create assemblies inside rectangular containers.
The algorithm managed to pack 465K particles in 11 minutes for the truncated Gaussian distribution function and 250K particles in 9 minutes in the Bernoulli tests.

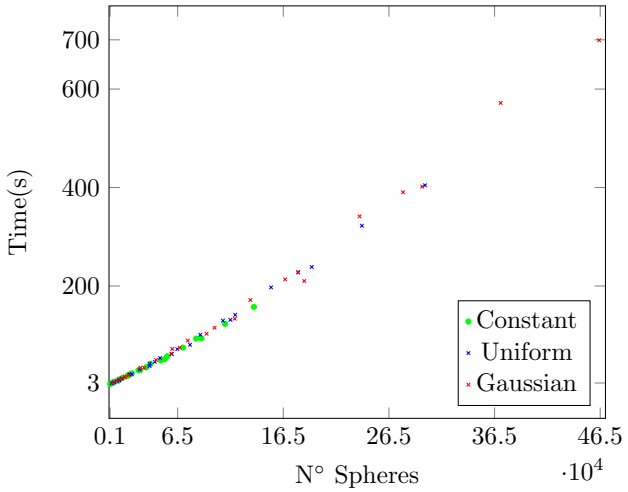


Figure 5.22: Spheres vs Time(s)

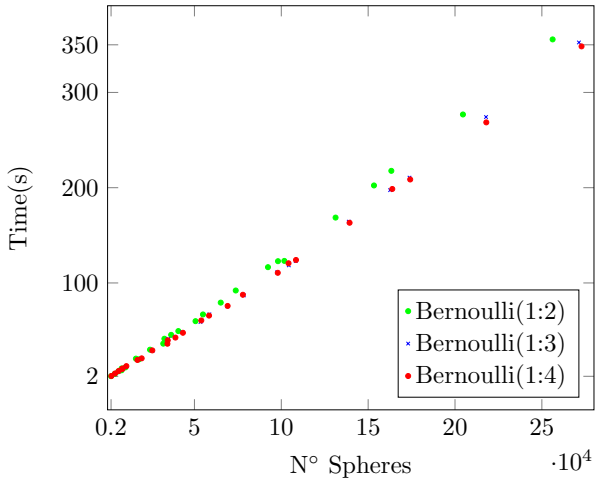


Figure 5.23: Bernoulli tests - Spheres vs Time(s)

Comparison with other packing solutions

Liu et al. (28) define two scenarios to compare their algorithm with previous solutions. Both scenarios take a unitary cube as a container and use a uniform radii variation for the packs with; a) $R_{min} = 1.7 \times 10^{-2}$, $R_{max} = 3.4 \times 10^{-2}$ and b) $R_{min} = 5.1 \times 10^{-3}$, $R_{max} = 5.1 \times 10^{-2}$. In the first test our coordination number was only lower than the one obtained by Liu et al. (28), our density ratio value was the same as the obtained by Jerier et al. (19). In the second test our coordination number was only better than the result of Lubachevsky et al. and our density was the lowest of all. We must say that both Jerier and Liu solutions perform a refilling operation after the package generation to insert more particles inside the void spaces increasing the density and coordination number. Our solution does not.

Table 5.7: Pack comparison for the uniform radii variation (28)

Algorithm	Machine	Coordination number		Density		Time(min)	
		(a)	(b)	(a)	(b)	(a)	(b)
Jerier et al. (19)	Intel Core2, 2GHz	6.1	7.0	0.59	0.66	20	28
Lubachevsky et al.	Intel Core2, 2GHz	5.5	3.0	0.64	0.69	120	577
Liu et al. (28)	Intel Core2, 2.53GHz	9.2	8.1	0.61	0.69	2	4
Our algorithm	**Intel Core5, 3.00GHz**	**6.2**	**5.8**	**0.59**	**0.60**	**0.14**	**0.06**

We also compare our packs with the ones generated with the solution proposed by Yu Shi (34) using a single rectangular box of 30x30x35 for a Truncated Gaussian radii variation using three particle size ranges varying in each range the standard deviation of the pack. Table 5.8 presents the results of Yu Shi and Table 5.9 presents ours. We can see that the densities and coordination numbers are very close.

Table 5.8: Gaussian radii variation results in (34)

Radius range	Density					Coordination number				
	$\sigma = 1$	$\sigma = 2$	$\sigma = 3$	$\sigma = 4$	$\sigma = 5$	$\sigma = 1$	$\sigma = 2$	$\sigma = 3$	$\sigma = 4$	$\sigma = 5$
0.34-3.00	0.600	0.594	0.593	0.592	0.591	5.71	5.66	5.62	5.57	5.64
0.50-2.00	0.592	0.592	0.591	0.591	0.593	5.80	5.83	5.84	5.83	5.81
0.67-1.50	0.585	0.586	0.586	0.587	0.584	5.89	5.88	5.89	5.89	5.89

Table 5.9: Our Gaussian radii variation results

Radius range	Density					Coordination number				
	$\sigma = 1$	$\sigma = 2$	$\sigma = 3$	$\sigma = 4$	$\sigma = 5$	$\sigma = 1$	$\sigma = 2$	$\sigma = 3$	$\sigma = 4$	$\sigma = 5$
0.34-3.00	0.622	0.611	0.614	0.611	0.610	5.49	5.38	5.35	5.48	5.42
0.50-2.00	0.607	0.603	0.604	0.599	0.601	5.95	5.86	5.86	5.86	5.85
0.67-1.50	0.593	0.589	0.594	0.596	0.589	6.02	6.03	6.01	6.05	6.05

5.3 Filling arbitrary objects

To validate the effectiveness of the initial front creation integrated with the assembly generation for triangle meshes we used different geometries as pack containers. The tests contemplate the constitution of packs with equal and uniform size variations using a $N_p = 18$. Also, we wanted to create two types of initial front to let us experiment with the two approaches to fill the $MAABB$: a regular package with fixed particle positions and a random pack resulting from our proposed algorithm with a uniform particle size variation. When creating the $MAABB$ pack it was used an $N_p = 9$, because in this case we are not interested in getting a dense pack.

Convex boundaries

The first test considers a simple geometry commonly used in the package generation beside boxes, a cylinder. Then we generate spheres inside a mesh representing a capsule.

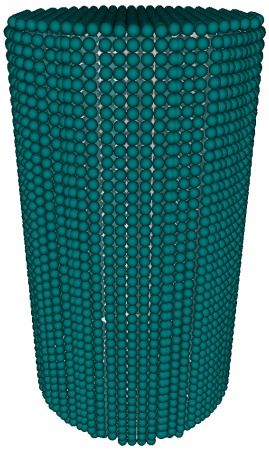
Table 5.10: Cylinder - Mesh properties

Triangles	**Dimension**	**Volume**	**Distance Field Resolution**
192	(2.14,4.00,2.14)	14.31	60x120x60

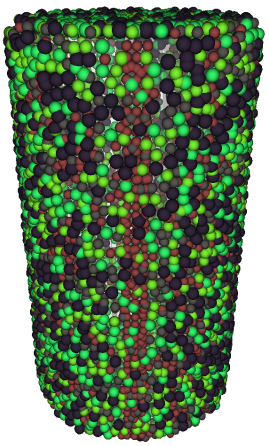
Table 5.11: Cylinder - Initial front and packing parameters

Initial front radii range	[0.05]		[0.04-0.06]	
Mesh pack radii range	[0.04-0.06]	[0.05]	[0.04-0.06]	[0.05]
$+\epsilon$	0.180	0.150	0.180	0.150
$-\epsilon$	0.120	0.100	0.120	0.100
M_p	0.060	0.060	0.072	0.072
Type of MAABB pack	**Regular**		**Random**	
Initial front	4434	4414	4960	4905
Pack particles	14031	14317	13912	14325
Density	0.530	0.524	0.530	0.524

Figure 5.24 and Figure 5.26 illustrate the set of spheres conforming the initial



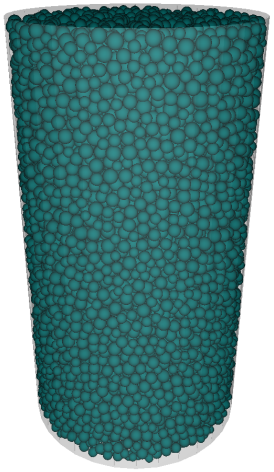
5.24(a): Regular pack



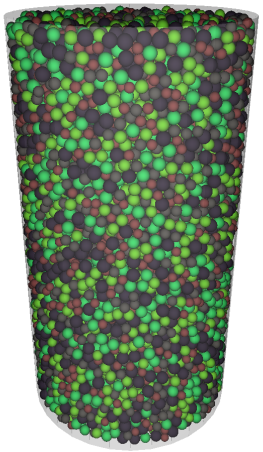
5.24(b): Uniform distribution

Figure 5.24: Cylinder - Initial front

front for the cylinder and the capsule. It is accentuated in Figure 5.24(a) and Figure 5.26(a) the fronts regular composition in contrast with the random fronts in Figure 5.24(b) and Figure 5.26(b).



5.25(a): Equal size

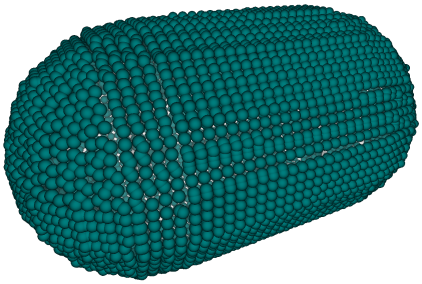


5.25(b): Uniform distribution

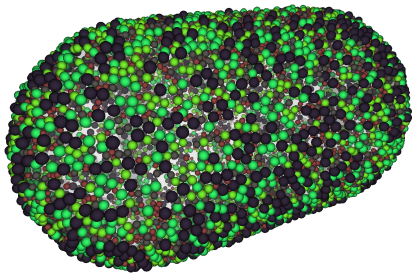
Figure 5.25: Cylinder - Mesh packs

Table 5.12: Capsule - Mesh properties

Triangles	**Dimension**	**Volume**	**Distance Field Resolution**
272	(6.37,3.49,3.49)	48.01	81x41x41

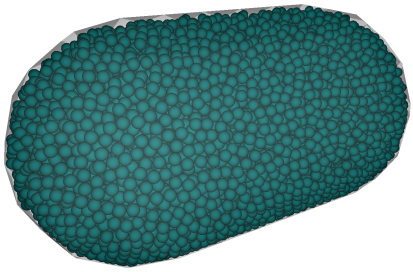


5.26(a): Regular pack

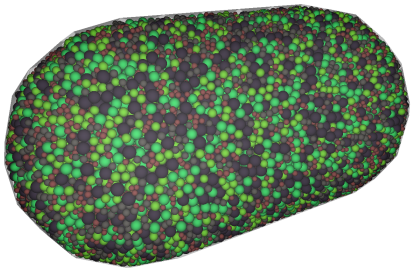


5.26(b): Uniform distribution

Figure 5.26: Capsule - Initial front



5.27(a): Equal size



5.27(b): Uniform distribution

Figure 5.27: Capsule - Mesh packs

Table 5.13: Capsule - Initial front and packing parameters

Initial front radii range	[0.04]		[0.03-0.05]	
Mesh pack radii range	[0.03-0.05]	[0.04]	[0.03-0.05]	[0.04]
$+\epsilon$	0.150	0.120	0.150	0.120
$-\epsilon$	0.100	0.080	0.100	0.080
M_p	0.044	0.044	0.055	0.055
Type of MAABB pack	**Regular**		**Random**	
Initial front	5272	5293	5989	5973
Pack particles	27688	28253	27718	28177
Density	0.536	0.532	0.535	0.531

Non-convex boundaries

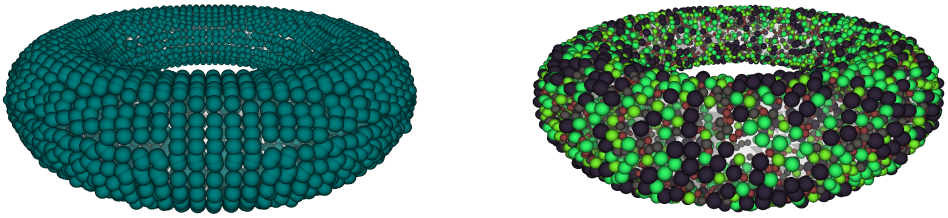
The next experiments consider non-convex meshes: the torus (Figures 5.28 and 5.29), the bunny (Figures 5.30 and 5.31) and the knot (Figures 5.32 and 5.33).

Table 5.14: Torus - Mesh properties

Triangles	**Dimension**	**Volume**	**Distance Field Resolution**
576	(3.81,0.76,3.81)	4.16	61x21x61

Table 5.15: Torus - Initial front and packing parameters

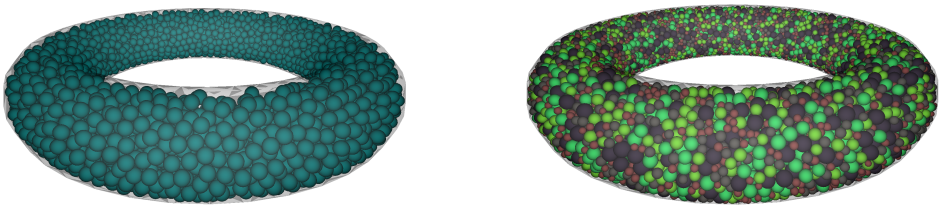
Initial front radii range	[0.03]		[0.02-0.04]	
Mesh pack radii range	[0.02-0.04]	[0.03]	[0.02-0.04]	[0.03]
$+\epsilon$	0.12	0.09	0.12	0.09
$-\epsilon$	0.08	0.06	0.08	0.06
M_p	0.03	0.03	0.04	0.04
Type of MAABB pack	**Regular**		**Random**	
Initial front	6236	6244	7788	7805
Pack particles	17329	18695	17387	18699
Density	0.520	0.508	0.519	0.508



5.28(a): Regular pack

5.28(b): Uniform distribution

Figure 5.28: Torus - Initial front



5.29(a): Equal size

5.29(b): Uniform distribution

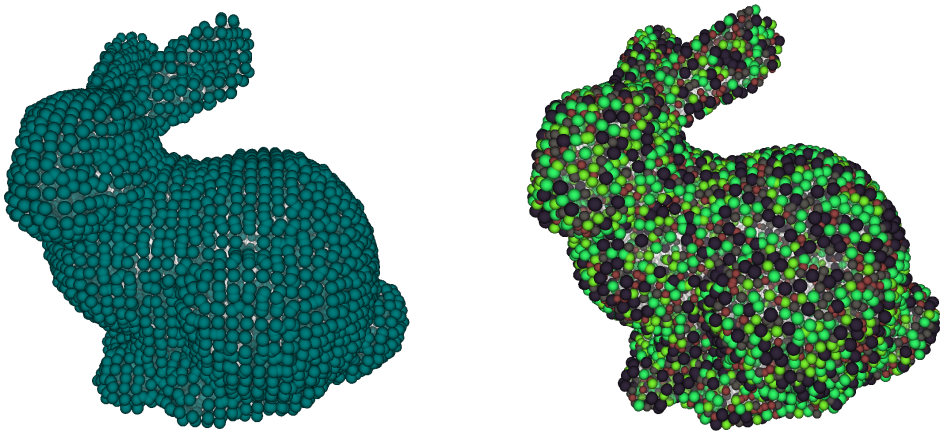
Figure 5.29: Torus - Mesh packs

Table 5.16: Bunny - Mesh properties

Triangles	**Dimension**	**Volume**	**Distance Field Resolution**
34817	(3.11,3.08,2.41)	6.04	60x60x60

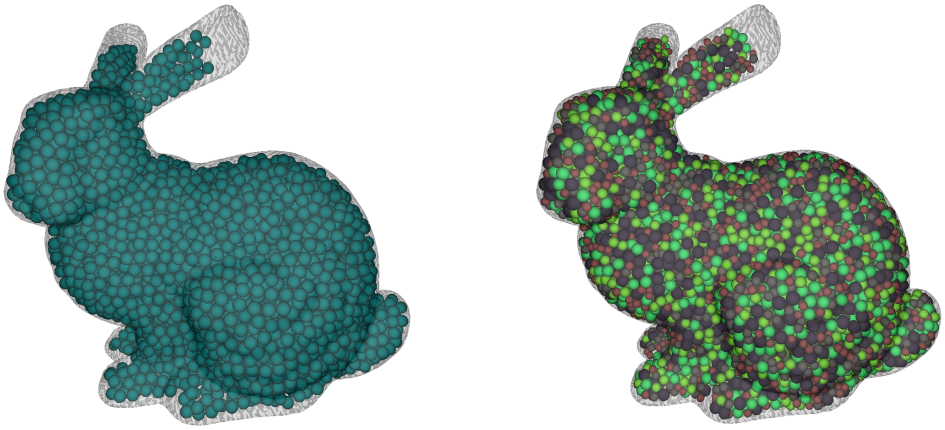
Table 5.17: Bunny - Initial front and packing parameters

Initial front radii range	[0.03]		[0.02-0.04]	
Mesh pack radii range	[0.02-0.04]	[0.03]	[0.02-0.04]	[0.03]
$+\epsilon$	0.12	0.09	0.12	0.09
$-\epsilon$	0.08	0.06	0.08	0.06
M_p	0.03	0.03	0.04	0.04
Type of MAABB pack	**Regular**		**Random**	
Initial front	6482	6483	8189	8118
Pack particles	25771	27688	25550	27696
Density	0.531	0.518	0.531	0.519



5.30(a): Regular pack 5.30(b): Uniform distribution

Figure 5.30: Bunny - Initial front

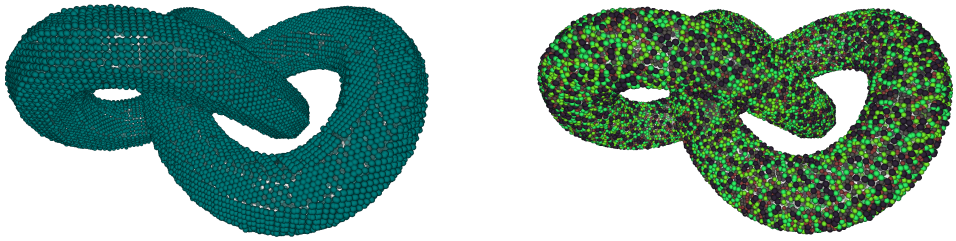


5.31(a): Equal size 5.31(b): Uniform distribution

Figure 5.31: Bunny - Mesh packs

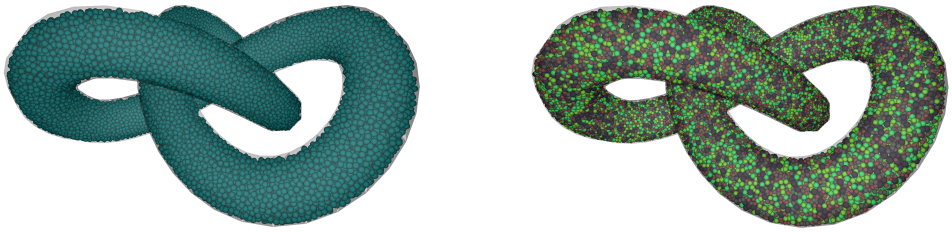
Table 5.18: Knot - Mesh properties

Triangles	**Dimension**	**Volume**	**Distance Field Resolution**
640	(7.17,3.52,7.60)	33.68	111x71x111



5.32(a): Regular pack 5.32(b): Uniform distribution

Figure 5.32: Knot - Initial front



5.33(a): Equal size

5.33(b): Uniform distribution

Figure 5.33: Knot - Mesh packs

Table 5.19: Knot - Initial front and packing parameters

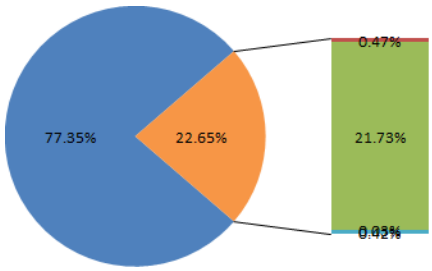
Initial front radii range	[0.06]		[0.05-0.07]	
Mesh pack radii range	[0.05-0.07]	[0.06]	[0.05-0.07]	[0.06]
$+\epsilon$	0.210	0.180	0.210	0.180
-ϵ	0.140	0.120	0.140	0.120
M_p	0.066	0.066	0.077	0.077
Type of MAABB pack	**Regular**		**Random**	
Initial front	9239	9291	10796	10878
Pack particles	18216	18359	18107	18351
Density	0.500	0.493	0.500	0.493

Performance

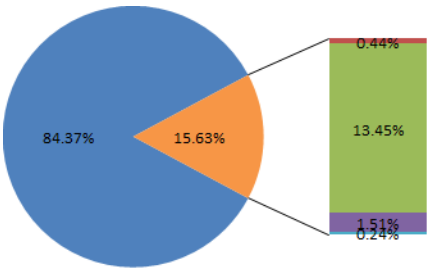
In the following tables and pie charts it is summarized the spent time for each important step in the process. We remark the speed of using a regular package to fill the $MAABB$ in comparison to the use of a random pack. The distance field computation does not constitute a bottleneck. The step which requires most time is the packing of the mesh.

Table 5.20: Time consumption for the cylinder pack results

Initial front radii range	[0.05]		[0.04-0.06]	
Mesh pack radii range	[0.04-0.06]	[0.05]	[0.04-0.06]	[0.05]
Task	**Time(s)**			
Prism generation	0.00	0.00	0.00	0.00
Point creation	1.81	1.79	1.78	1.79
Distance field computation	83.7	70.2	82.6	54.7
Type of pack inside MAABB	**Regular**		**Random**	
MAABB package	0.13	0.12	6.24	6.13
Front selection & relocation	1.61	1.33	1.01	1.00
Pack generation	298.	283.	329.	343.



5.34(a): Regular MAABB pack

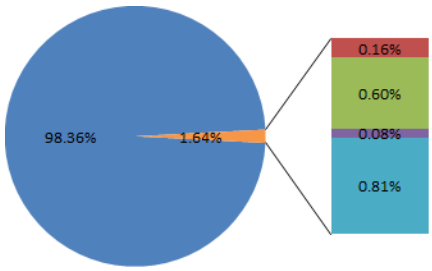


5.34(b): Random MAABB pack

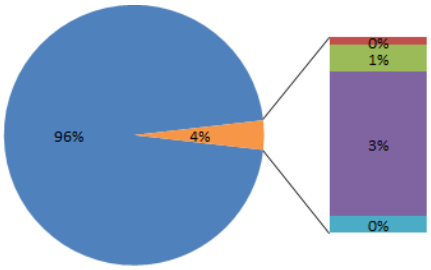
Figure 5.34: Cylinder times - Packs with constant radii

Table 5.21: Time consumption for the capsule pack results

Initial front radii range	[0.04]		[0.03-0.05]	
Mesh pack radii range	[0.03-0.05]	[0.04]	[0.03-0.05]	[0.04]
Task	**Time(s)**			
Prism generation	0.00	0.00	0.00	0.00
Point creation	0.75	0.75	0.74	0.74
Distance field computation	3.34	2.79	3.37	2.54
Type of pack inside MAABB	**Regular**		**Random**	
MAABB package	0.36	0.36	14.4	13.9
Front selection & relocation	4.27	3.76	1.65	1.64
Pack generation	443.	459.	493.	496.



5.35(a): Regular MAABB pack

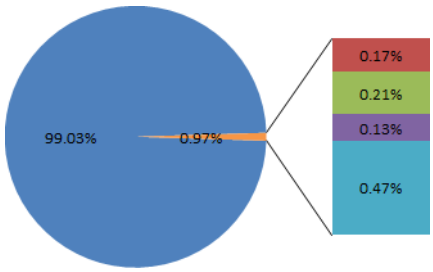


5.35(b): Random MAABB pack

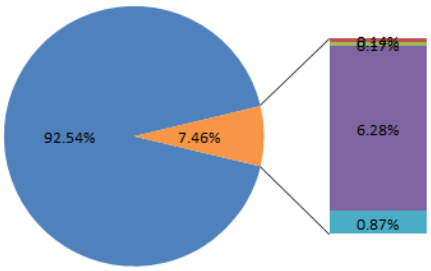
Figure 5.35: Capsule times - Packs with constant radii

Table 5.22: Time consumption for the torus pack results

Initial front radii range	[0.03]		[0.02-0.04]	
Mesh pack radii range	[0.02-0.04]	[0.03]	[0.02-0.04]	[0.03]
Task	**Time(s)**			
Prism generation	0.00	0.00	0.00	0.00
Point creation	0.47	0.50	0.49	0.48
Distance field computation	0.65	0.64	0.65	0.58
Type of pack inside MAABB	**Regular**		**Random**	
MAABB package	0.40	0.40	22.4	22.1
Front selection & relocation	1.38	1.41	2.82	3.08
Pack generation	287.	300.	316.	326.



5.36(a): Regular MAABB pack

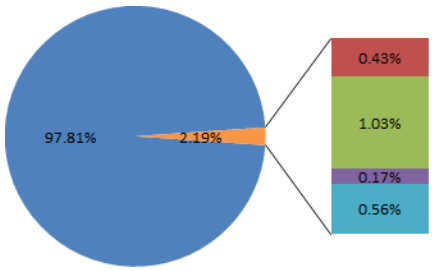


5.36(b): Random MAABB pack

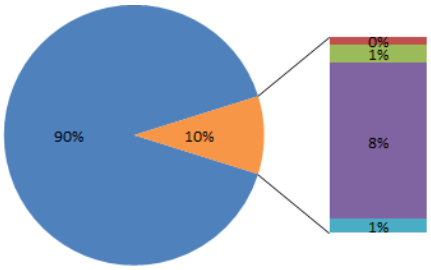
Figure 5.36: Torus times - Packs with constant radii

Table 5.23: Time consumption for the bunny pack results

Initial front radii range	[0.03]		[0.02-0.04]	
Mesh pack radii range	[0.02-0.04]	[0.03]	[0.02-0.04]	[0.03]
Task	**Time(s)**			
Prism generation	0.17	0.02	0.02	0.02
Point creation	2.34	2.31	2.28	2.29
Distance field computation	8.24	5.52	8.15	5.56
Type of pack inside MAABB	**Regular**		**Random**	
MAABB package	0.91	0.91	47.9	48.9
Front selection & relocation	2.79	3.00	4.05	4.29
Pack generation	504.	526.	519.	570.



5.37(a): Regular MAABB pack

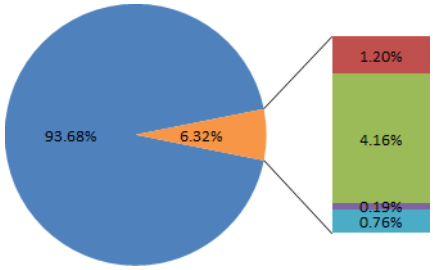


5.37(b): Random MAABB pack

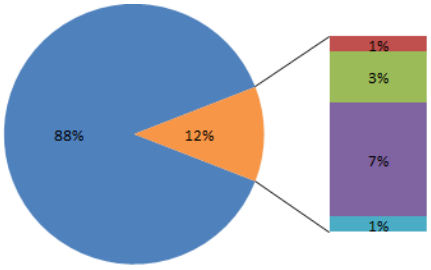
Figure 5.37: Bunny times - Packs with constant radii

Table 5.24: Time consumption for the knot pack results

Initial front radii range	[0.06]		[0.05-0.07]	
Mesh pack radii range	[0.05-0.07]	[0.06]	[0.05-0.07]	[0.06]
Task	**Time(s)**			
Prism generation	0.00	0.00	0.00	0.00
Point creation	6.40	6.57	6.30	6.18
Distance field computation	26.7	22.8	26.8	19.7
Type of pack inside MAABB	**Regular**		**Random**	
MAABB package	1.01	1.05	47.2	45.3
Front selection & relocation	4.00	4.18	6.39	6.02
Pack generation	470.	513.	595.	577.



5.38(a): Regular MAABB pack



5.38(b): Random MAABB pack

Figure 5.38: Knot times - Packs with constant radii

5.4
Point in polyhedron test

Besides the "Point in polyhedron" test there is an alternative method to determine if a point is inside or outside a mesh called the "Winding number". The method counts the number of times the mesh winds around a point. For every face it is computed the *solid angle* $\Omega_{(p)}$ formed by the point. All the angles are summed and only if the result is $\omega_{(p)} = 0$ then the point is considered outside (17).

$$\omega_{(p)} = \sum_{i=1}^{f} \frac{1}{4\pi} \Omega_{(p)} \tag{5-4}$$

We will use this method to compare the results of our "Point in polyhedron" implementation. The test cases will render all the triangles gathered by the AABB with a margin of the ray R . The triangle colors will depend on the type of collision: red for vertex, green for edge and blue for face intersection.

Test case 1

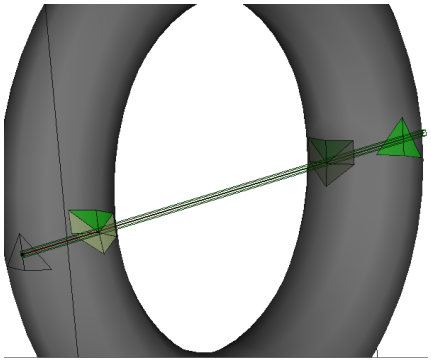
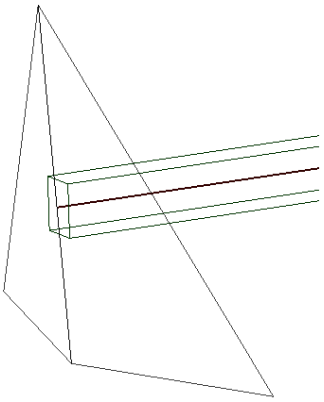
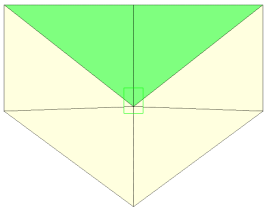


Figure 5.39: Point in polyhedron - Test case 1

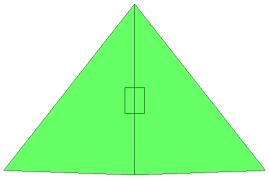
This test case will allow us to identify two situations. The first layer of triangles do not need to be taken in consideration for the collision test because they are behind the ray, nevertheless the 2D "point in triangle" will find an edge collision. Nevertheless the algorithm will compute the real X coordinates of the points and discard the triangles. The ray passes through the middle of three triangle pairs, and the algorithm was able to identify every edge collision as one intersection. The test point $p = (-1.8945, 0.0, -0.14032)$ intersects in our implementation three times the mesh. Using the "Winding number" we have a $\omega_{(p)} = 1.0$ identifying the point as inside the mesh.



5.40(a): First



5.40(b): Second and third



5.40(c): Fourth

Figure 5.40: Layers of triangles

Test case 2

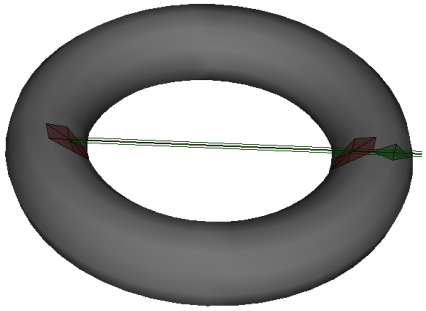
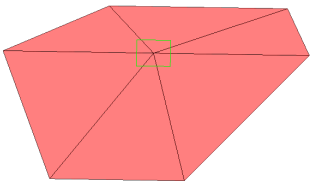
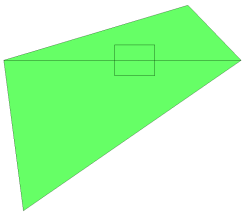


Figure 5.41: Point in polyhedron - Test case 2



5.42(a): First and second



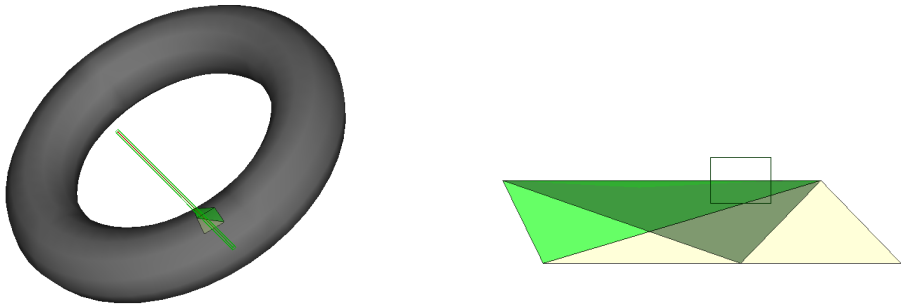
5.42(b): Third

Figure 5.42: Layers of triangles

The second test traces a ray starting from the point $p = (-1.29806, -0.32990, -0.34508)$ located inside the mesh. The issue with

this ray is that it passes through one edge and two vertices (each one shared by 6 triangles). Our "Point in polyhedron" algorithm counts only three intersections. The "Winding number" computes a value of $\omega_{(p)} = 1.0$ for the point.

Test case 3



5.43(a): Mesh

5.43(b): Single layer

Figure 5.43: Point in polyhedron - Test case 3

This example shows a ray starting at point $p = (-0.29806, -0.38094, -0.34508)$ tangent to the mesh. The AABB of the ray intercepts three triangles but only hits one edge. Because the dot products of the ray direction $\vec{R_{dir}}$ and the triangle normals have different signs, the collision is not counted and the point is recognized to be outside. This interpretation is confirmed by the "Winding number" output $\omega_{(p)} = 0.0$

Test case 4

This case is similar to the previous example, the ray starting at the point $p = (-1.3626, 2.7450, -0.37704)$ is tangent to the mesh. But now the ray intersects a vertex shared by 5 triangles. Since the dot products of the triangle normals and the $\vec{R_{dir}}$ have different signs, the collision is not counted as a mesh intersections. One again the "Winding number" value is $\omega_{(p)} = 0.0$.

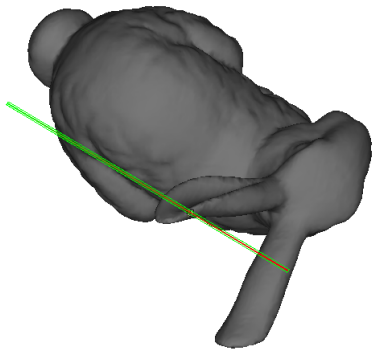
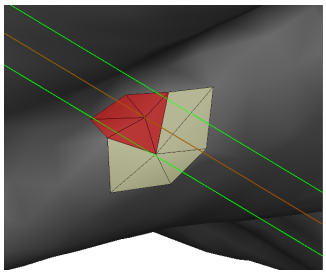
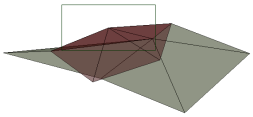


Figure 5.44: Point in polyhedron - Test case 4



5.45(a): Zoom

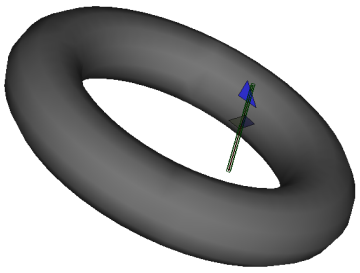


5.45(b): Triangles

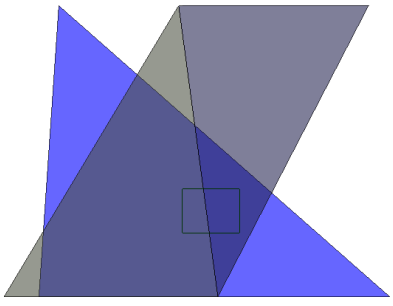
Figure 5.45: Intersections

Test case 5

We left for the last example the simplest situation. A point $p = (-0.0806, 0.13432, -0.59241)$ outside the mesh intersects two triangles through their faces. There is a third triangle gathered by the ray AABB but does not intersect the ray. The "Winding number" output for this case is $\omega_{(p)} = 0.0$.



5.46(a): Mesh



5.46(b): Two layers

Figure 5.46: Point in polyhedron - Test case 5

Performance

It was executed the computation of all the points in the distance fields of different meshes using the two available methods. We contemplate the sum of every individual call to the method and then it was calculated a mean time for a single call based on the distance field resolution.

Table 5.25: Time consumption for the point in polyhedron tests

Mesh (N° triangles f)	**Distance Field Resolution d_xxd_yxd_z**	**Total Time(s)**		**Mean Time(s) per Point**	
		Winding Number	**Point in polyhedron**	**Winding Number**	**Point in polyhedron**
Capsule (272)	81x41x41	8.0	0.9	59×10^{-6}	6×10^{-6}
Torus (576)	61x21x61	9.7	0.5	125×10^{-6}	6×10^{-6}
Bunny (34817)	60x60x60	2042.	1.9	9456×10^{-6}	9×10^{-6}
Knot (640)	111x71x111	122.	5.6	140×10^{-6}	6×10^{-6}
Cylinder (192)	60x120x60	5.8	1.4	13×10^{-6}	3×10^{-6}

Clearly the "Point in polyhedron" consumes a lot less time. The disadvantage of the "Winding number" is that has a $O(d_x d_y d_z f)$ complexity. The bunny mesh, for example, took almost half an hour due to its high number of faces. In contrast, our method took only 2 seconds.
For every point in all the distance fields, the outcomes for our implementation matched the outcome of the winding number method.

6
Conclusion and Future Works

In this work we introduced a 3D packing method for spherical particles derived from a 2D advancing front solution. The proposed algorithm generates packages with acceptable geometric properties in a reasonable time in all tested containers. The use of a distance field was effective in the proposed algorithm. Comparing with the 2D version the proposed algorithm yields tighter assemblies due to a better halo definition.

Due to its nature, the average number of contact points in the assemblies produced by the proposed algorithm is just above 6, regardless of the complexity of the mesh container and the radius distribution function. In the case of monodisperse assemblies this number of contacts is enough to prove that our packages are stable.

The resulting density is also almost constant. Around 0.6 for rectangular containers, which is close to the maximum 0.64 density for monodisperse assemblies, and 0.5 for arbitrary meshes.

The suggested discretization of the intersection halo is around 72, or 5° degrees in order to get packs with high densities.

As for future work, there are certain aspects that could be explored to improve the packing generation. A suggestion is to find an analytic solution for the best position for a new particle in the intersection halo. A different approach might consider insert particles tangent to three previously inserted spheres instead of using the halo intersection. This way the new particle will have to check only two possible positions. It is expected that both alternatives should produce packages in less time. A possible difficulty is to find a criteria of how to select the triplet of spheres.

We also suggest that in the insertion of a new sphere the algorithm could consider different directions to improve the fitting of the spheres.

As the halo intersection tests suggest, the number of sampled points can be used to slightly increase the porosity ratio, nonetheless this is not enough as some simulations requires. Hence, a future work could study the application of some mechanism of control to produce packs with a given porosity ratio.

A property of interest related to the porosity is the permeability, which is the

pack ability to allow fluids to pass through it. An idea to study this property within our algorithm is with the update of the distance field on each particle insertion to obtain a distance field of the voids.

As for the front generation, we could improve the assembly of spheres on the surface of the containers with the relocation of those spheres to avoid any interpenetration. With less spheres in the front the algorithm becomes more efficient.

A
Random number generators

The proposed assembly and front generation algorithms have as an input the desired distribution function for the radii creation. We will explain how the radii based on random number generators we created.
For the number generation we use the common function for random generation numbers in C++ *rand*(). Since this function creates numbers in the range of [0-$RAND\_MAX$] we divide the result by $RAND\_MAX$ to obtain numbers in the range of [0-1]. In the case of the Uniform distribution we produce the particle radii by the following algorithm.

Algorithm 10 Uniform number generator

Input: minimumRadius, maximumRadius.
Output: newRadius.

```
random ← rand() / RAND_MAX
newRadius ← minimumRadius + random * (maximumRadius - minimumRadius)
```

For the Bernoulli distribution we use an additional parameter, a probability ρ to create radii with the minimum value.

Algorithm 11 Bernoulli number generator

Input: minimumRadius, maximumRadius, probability.
Output: newRadius.

```
if rand() / RAND_MAX < 1.0 - probability then
    newRadius ← minimumRadius
else
    newRadius ← maximumRadius
end if
```

Our Truncated gaussian distribution generator implements a polar version of the Box Muller transform (37) which creates a pair of random numbers with standard normal distribution ($\mu = 0$ and $\sigma^2 = 1$) in a single execution. Then the numbers are scaled and translated multiplying them by the desired standard deviation and adding the desired mean. Only those numbers between the minimum and maximum radius are accepted.

Algorithm 12 Truncated gaussian number generator

Input: minimumRadius, maximumRadius, mean, standardDeviation.
Output: newRadius.

```
static n2 ← 0
static n2Cached ← 0
repeat
    if not n2Cached then
        repeat
            x ← 2 * rand() / RAND_MAX - 1
            y ← 2 * rand() / RAND_MAX - 1
            r = x * x + y * y
        until r != 0 and r <= 1
        d ← sqrt(-2 * log(r) / r)
        n1 ← x * d
        n2 ← y * d
        n2Cached ← 1
        newRadius ← mean + n1 * standardDeviation
    else
        n2Cached ← 0
        newRadius ← mean + n2 * standardDeviation
    end if
until newRadius >= minimusRadius and newRadius <= maximumRadius
```

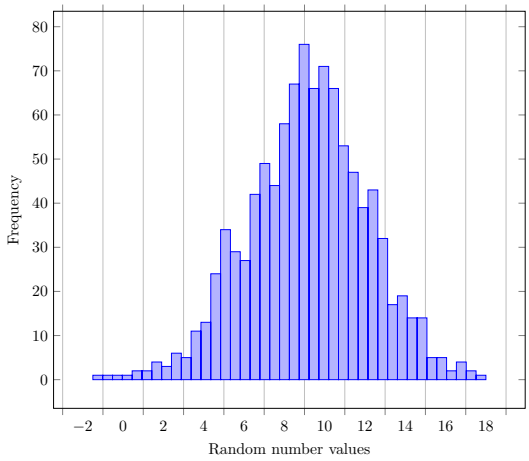


Figure A.1: 1000 values using the Truncated gaussian number generator ($\mu = 10$, $\sigma^2 = 9$)

Bibliography

[1] BAGI, K. **Granular Matter**. An algorithm to generate random dense arrangements for discrete element simulations of granular assemblies, journal, v.7, n.1, p. 31–43, 2005.

[2] BENABBOU, A.; BOROUCHAKI, H.; LAUG, P. ; LU, J. **Sphere packing and applications to granular structure modeling**. In: PROCEEDINGS OF THE 17TH INTERNATIONAL MESHING ROUNDTABLE, p. 1–18. Springer, 2008.

[3] BENABBOU, A.; BOROUCHAKI, H.; LAUG, P. ; LU, J. **International journal for numerical methods in engineering**. Geometrical modeling of granular structures in two and three dimensions. application to nanostructures, journal, v.80, n.4, p. 425–454, 2009.

[4] BENNETT, C. H. **Journal of Applied Physics**. Serially deposited amorphous aggregates of hard spheres, journal, v.43, n.6, p. 2727–2734, 1972.

[5] BEZRUKOV, A.; BARGIEŁ, M. ; STOYAN, D. **Particle & Particle Systems Characterization**. Statistical analysis of simulated random packings of spheres, journal, v.19, n.2, p. 111–118, 2002.

[6] BEZRUKOV, A.; STOYAN, D. ; BARGIEL, M. **Image Anal. Stereol**. Spatial statistics for simulated packings of spheres, journal, v.20, p. 203–206, 2001.

[7] BORKOVEC, M.; DE PARIS, W. ; PEIKERT, R. **Fractals**. The fractal dimension of the apollonian sphere packing, journal, v.2, n.04, p. 521–526, 1994.

[8] CUI, L.; O SULLIVAN, C. **Granular Matter**. Analysis of a triangulation based approach for specimen generation for discrete element simulations, journal, v.5, n.3, p. 135–145, 2003.

[9] CUNDALL, P. A.; STRACK, O. D. **Geotechnique**. A discrete numerical model for granular assemblies, journal, v.29, n.1, p. 47–65, 1979.

[10] ERICSON, C. **Real-Time Collision Detection**. The Morgan Kaufmann Series in Interactive 3D Technology. Elsevier Science, 2004.

[11] FENG, Y.; HAN, K. ; OWEN, D. **International Journal for Numerical Methods in Engineering**. Filling domains with disks: an advancing front approach, journal, v.56, n.5, p. 699–713, 2003.

[12] FLICKINGER, J. C.; WU, A.; MAITZ, A. H.; KALEND, A. M. ; OTHERS. **International Journal of Radiation Oncology* Biology* Physics**. Treatment planning for gamma knife radiosurgery with multiple isocenters, journal, v.18, n.6, p. 1495–1501, 1990.

[13] FUHRMANN, A.; SOBOTKA, G. ; GROSS, C. **Distance fields for rapid collision detection in physically based modeling**. In: PROCEEDINGS OF GRAPHICON 2003, p. 58–65, 2003.

[14] HAN, K.; FENG, Y. ; OWEN, D. **Powder Technology**. Sphere packing with a geometric based compression algorithm, journal, v.155, n.1, p. 33–41, 2005.

[15] HITTI, K.; BERNACKI, M. **Applied Mathematical Modelling**. Optimized dropping and rolling (odr) method for packing of poly-disperse spheres, journal, v.37, n.8, p. 5715–5722, 2013.

[16] HORN, W. P.; TAYLOR, D. L. **Computer Vision, Graphics, and Image Processing**. A theorem to determine the spatial containment of a point in a planar polyhedron, journal, v.45, n.1, p. 106–116, 1989.

[17] JACOBSON, A.; KAVAN, L. ; SORKINE-HORNUNG, O. **ACM Trans. Graph.** Robust inside-outside segmentation using generalized winding numbers., journal, v.32, n.4, p. 33, 2013.

[18] JALALI, P.; LI, M. **The Journal of Chemical Physics**. An estimate of random close packing density in monodisperse hard spheres, journal, v.120, n.2, 2004.

[19] JERIER, J.-F.; IMBAULT, D.; DONZE, F.-V. ; DOREMUS, P. **Granular Matter**. A geometric algorithm based on tetrahedral meshes to generate a dense polydisperse sphere packing, journal, v.11, n.1, p. 43–52, 2009.

[20] JERIER, J.-F.; RICHEFEU, V.; IMBAULT, D. ; DONZÉ, F.-V. **Computer Methods in Applied Mechanics and Engineering**.

Packing spherical discrete elements for large scale simulations, journal, v.199, n.25, p. 1668–1676, 2010.

[21] JIANG, M.; KONRAD, J. ; LEROUEIL, S. **Computers and geotechnics**. An efficient technique for generating homogeneous specimens for dem studies, journal, v.30, n.7, p. 579–597, 2003.

[22] JIANG, M.; YU, H.-S. **Application of discrete element method to geomechanics**. Springer, 2006.

[23] KALAY, Y. E. **Computer Graphics and Image Processing**. Determining the spatial containment of a point in general polyhedra, journal, v.19, n.4, p. 303–334, 1982.

[24] LABRA, C.; ONATE, E. **Communications in Numerical Methods in Engineering**. High-density sphere packing for discrete element method simulations, journal, v.25, n.7, p. 837–849, 2009.

[25] LEWIS, R. W.; GETHIN, D. T.; YANG, X. S. ; ROWE, R. C. **International journal for numerical methods in engineering**. A combined finite-discrete element method for simulating pharmaceutical powder tableting, journal, v.62, n.7, p. 853–869, 2005.

[26] LI, J.; WEBB, C.; PANDIELLA, S. ; CAMPBELL, G. **Food and bioproducts processing**. A numerical simulation of separation of crop seeds by screening effect of particle bed depth, journal, v.80, n.2, p. 109–117, 2002.

[27] LIU, J.; LI, S. ; CHEN, Y. **Acta Mechanica Sinica**. A fast and practical method to pack spheres for mesh generation, journal, v.24, n.4, p. 439–447, 2008.

[28] LIU, J.; YUN, B. ; ZHAO, C. **International Journal for Numerical and Analytical Methods in Geomechanics**. An improved specimen generation method for dem based on local delaunay tessellation and distance function, journal, v.36, n.5, p. 653–674, 2012.

[29] MUNJIZA, A.; ANDREWS, K. **International Journal for Numerical Methods in Engineering**. Nbs contact detection algorithm for bodies of similar size, journal, v.43, n.1, p. 131–149, 1998.

[30] PINTO, A. L. F. **Algoritmo para geração de arranjos de particulas para utilização no método dos elementos discretos**. 2009. Master's thesis - Pontifícia Universidade Católica do Rio de Janeiro.

[31] PÖSCHEL, T.; SCHWAGER, T. **Computational Granular Dynamics: Models and Algorithms**. SCIENTIFIC COMPUTATION. Springer, 2005.

[32] SAKAGUCHI, E.; FAVIER, J. **International Agrophysics**. Analysis of the shear behaviour of a grain assembly using dem simulation, journal, v.14, n.2, p. 241–248, 2000.

[33] SCHIFTNER, A.; HÖBINGER, M.; WALLNER, J. ; POTTMANN, H. **Packing circles and spheres on surfaces**. In: ACM TRANSACTIONS ON GRAPHICS (TOG), volume 28, p. 139. ACM, 2009.

[34] SHI, Y.; ZHANG, Y. **Simulation of random packing of spherical particles with different size distributions**. In: ASME 2006 INTERNATIONAL MECHANICAL ENGINEERING CONGRESS AND EXPOSITION, p. 539–544. American Society of Mechanical Engineers, 2006.

[35] STOYAN, D. **International Statistical Review**. Random sets: Models and statistics, journal, v.66, n.1, p. 1–27, 1998.

[36] SUN, Y.; XU, W. **International Journal of Computer Applications in Technology**. Simulation of food mastication based on discrete element method, journal, v.39, n.1, p. 3–11, 2010.

[37] THOMAS, D. B.; LUK, W.; LEONG, P. H. ; VILLASENOR, J. D. **ACM Comput. Surv.** Gaussian random number generators, journal, v.39, n.4, Nov. 2007.

[38] WEAIRE, D.; ASTE, T. **The pursuit of perfect packing**. CRC Press, 2008.

[39] WELLER, R.; ZACHMANN, G. **Protosphere: A gpu-assisted prototype guided sphere packing algorithm for arbitrary objects**. In: ACM SIGGRAPH ASIA 2010 SKETCHES, p. 8. ACM, 2010.

[40] ZSAKI, A. **Computers and Geotechnics**. An efficient method for packing polygonal domains with disks for 2d discrete element simulation, journal, v.36, n.4, p. 568–576, 2009.

[41] ZSAKI, A. **International journal for numerical and analytical methods in geomechanics**. Parallel generation of initial element assemblies for two-dimensional discrete element simulations, journal, v.33, n.3, p. 377–389, 2009.