INDEXED PARALLEL SPHERE PACKING FOR ARBITRARY DOMAINS

Cuba Lajo Rubén Adrián and Loaiza Fernández Manuel Eduardo

Department of Computer Science,
Universidad Católica San Pablo, Arequipa, Perú

ABSTRACT

Particle packings are used to simulate granular matter, which has various uses in industry. The most outstanding characteristics of these are their density and their construction time, the density refers to the percentage of the space of the object filled with particles, this is also known as compaction or solid fraction. Particle packing seeks to be as dense as possible, work on any object, and have a low build time. Currently there are proposals that have significantly reduced the construction time of a packing and have also managed to increase the density of these, however, they have certain restrictions, such as working on a single type of object and being widely affected by the characteristics of the object. The objective of this work is to present the improvement of a parallel sphere packing for arbitrary domains. The packing to improve was directly affected in time by the number of triangles in the mesh of object. This enhancement focuses on creating a parallel data structure to reduce build time. The proposed method reduces execution time with a high number of triangles, but it takes up a significant amount of memory for the data structure. However, to obtain high densities, that is, densities between 60% and 70%, the sphere packing construction does not overwhelm the memory.

KEYWORDS

Sphere Packing, Geometric Algorithm, Parallel.

1. INTRODUCTION

Things that surround us usually are mostly granular matter, this is a set of solid particles that is widely used in industry and its behaviour is widely studied in physics [1][2]. Granular matter is virtually represented by particle packings. Particle packings are represented by an object filled with particles, where the object is a 3D mesh, this is called a container, and the particles are 3D objects of known volumes, for example, spheres, cubes, tetrahedrons, etc. Within the particles used, the most used are the spheres due to their simple representation of four real numbers, the first three, the centre, referring to its position and the last, the radius, referring to the space it occupies. That is why in this research we work with a sphere packing.

A particle packing seeks to be as dense as possible, work on any object, and have a low build time. The density of a packing is the percentage of the container space occupied by particles, this is also called compaction or solid fraction. Particle packings have two approaches, dynamic and geometric packings, the former focused on physical simulations and the latter on geometric constructions. There are also some particle packings that we will call overlapping packings, these use overlapping particles, that is, they do not take into account their collisions, what they do is accommodate the particles in some containers covering the largest possible space, they are mostly used to detect collisions between objects, these packings are not taken into account in this

David C. Wyld et al. (Eds): AIAPP, NLPML, DMA, CRIS, SEC, CoSIT, SIGL - 2022
pp. 213-226, 2022. CS & IT - CSCP 2022 DOI: 10.5121/csit.2022.120918

research since they move away from its purpose by not having collisions between particles of the same container.

The shortcomings of particle packing are mostly the use of a single container and the density achieved. The objective of this work is to improve a method that already solves the problems mentioned above, the method on which we worked is the method proposed by Cuba and Loaiza [3], which is a method that works in parallel, but does not count with a wide analysis of said parallelism, in addition to having a high delay with 3D meshes that have many triangles. To solve these problems, a data structure will be used which is also parallelized, and the method will be tested on different graphics cards.

This article is organized as follows: Section 2 presents the best particle packing methods today along with their limitations. In Section 3 the proposed sphere packing method is presented. Section 4 shows the tests performed, as well as their results. Finally, we present the conclusions in Section 5.

2. RELATED WORK

Particle packings are varied and seek to be dense. These can be classified into dynamic and geometric. Dynamic packings use the Discrete Element Method (DEM), they are stable and provide the necessary information to perform physical simulations, however, they take a long time, which restricts the number of particles in the packing. Geometric packings do not take as long as dynamic packings and provide more control of particle distribution in the container.

2.1. Dynamic Packings

These methods seek to carry out physical simulations, for which they work with different forms of particles, in this way they are more similar to reality, encompassing a broader study. By vibrating a container full of particles the density increases, with this in mind it was proposed to use composite cubes with superimposed spheres subjected to mechanical vibration to increase the density of the packing, resulting in a maximum density of approximately 70% in a cylindrical container [4]; tetrahedrons composed of superimposed spheres subjected to vibration were also used, but unlike the previous one, a study of the effects of vibration conditions was made, in this study a maximum density of 74.02% was obtained in a cylindrical container [5].

Cylinders composed of superimposed spheres were also used as particles, in this packing it must be fulfilled that the diameter of the container must be larger than the size of the particle, several filling methods were carried out, where in one of them, specifically drop filling, it was observed that the density of the packing is sensitive to height, this packing reached an approximate maximum density of 55% [6]. As you can see, particles composed of spheres are used, this time it was decided to use ellipsoids that are similar to spheres, in this study by using a horizontal orientation in the particles, that is, the elongated part is horizontal, and by increasing the size of the particles increases the density, something interesting, since it is mostly by reducing the size of a particle that greater density is achieved, the maximum density reached in this study was a density of 70% in a cylindrical container [7].

In particle packings, modifying the size of the particles to achieve a higher density is common, taking it further, it was that Zhao et al. [8] proposed to modify the shape of the particles to achieve a higher density, the particles used in this packing are ellipsoids that change shape, this research had a maximum density of about 80% in a rectangular container, which is very good, however, the shape of the ellipsoid that achieved such compaction is similar to a cube. If a cube

is filled with cubes, it can be intuited that it will be a compact packing. As the particles take different shapes to get closer to reality, Rakotonirina et al. [9] carried out research on this, this proposal joins convex particles to build non-convex particles, the distribution and detection of collisions of the particles are based on the convex particles that form the non-convex particles to speed up the calculation, however, the computational cost is increased as well.

The spheres are widely seen in particle packing, Campello and Casares [10] use the spheres as particles, proposing a layered filling together with a compaction system for the spheres used in rectangular containers, the maximum density was 60%. Dynamic packings when focusing on physical simulations is that they only work with rectangular or cylindrical containers and sometimes both, that is, dynamic packings use at most two domains, this restricts the approach that is desired in this research, which is to cover any domain. Currently there are more dynamic packing, but their results are similar to those mentioned previously, for example, the proposal by Wang et al. [11] that uses octahedrons for vibration experiments.

2.2. Geometric Packings

These methods leave aside physical simulations to focus on the distribution of particles and thus build a dense packing. The most used particles in this type of packing are the spheres. The spheres are widely used, since for all the calculations that involve them, only four numbers must be known, three are their position and the other is their size. These types of packings seek to work in any container. A common and widely used method for geometric packing is the advancing front approach, this generates an initial set of spheres and new spheres are inserted with a strategy based on the previously inserted spheres.

Among the geometric methods, the proposal of Wang et al. [12] stands out, since it is currently a geometric method that plans to be dynamic, this method works in a cubic container, and uses various geometric methods in the construction of a package of non-spherical particles with controllable shapes, this to have realistic particles.

Lozano et al. [13] proposed a method to fill arbitrary containers, this method is based on a 2D advancing front approach using a distance field to achieve contact spheres tangent to the triangles of the container mesh. This method reaches a maximum density of approximately 60% in arbitrary containers.

Li and Ji [14] proposed a method also based on the advancing front approach for arbitrary containers. It is proposed to change the size of the particles while building the packing, the new particles adjust to the previously inserted neighbouring particles according to the trilateration equations. It uses a spatial grid to optimize the detection of particles and accelerate their positioning, reaching a maximum density of 73% in a cubic container.

Weller and Zachmann [15] proposed a parallel method using GPU to rapidly pack spheres, successively inserting spheres of the largest possible size to fit into the empty spaces. However, this method does not display geometric data such as the radii of the spheres or the densities of the packings. Also due to its behaviour, relevant data cannot be input apart from the container mesh. This method was used for object collisions [16].

Cuba and Loaiza [3] proposed a sequential and parallel method using GPU, unlike the previous method, taking into account geometric data such as the radii and densities of the generated packing, this study shows a significant number of results that support their proposal, however, these results have shortcomings such as the time it takes to build the packing when working with

meshes of high number of triangles, and the shortage of tests of the parallel method, so in this research this method is improved and more tests of the parallel methods are carried out.

3. PACKING GENERATION

The proposed method is the improvement of an existing method, this method is the method proposed by Cuba and Loaiza [3] called Parallel Sphere Packing for Arbitrary Domains, for which this method will be explained in a simplified way, and then the improvements made will be explained.

3.1. Parallel Sphere Packing for Arbitrary Domains

This method is the method of Cuba and Loaiza [3], which works with four radii, these are called r_{max}, r_{min}, r_{med}, r_{mid}. The positions of the spheres are given by a hexagon of spheres, which can be seen in Figure 1. The input data for this method is the radius r_{max} and the mesh of the arbitrary container. This parallel packing of spheres leaves randomness aside, therefore, it eliminates the need to detect collisions between spheres, it works in two phases, in the first phase a box is filled, the box is a rectangular container that surrounds the 3D object to be filled, in the second phase a verification is performed on all the spheres to see if they are inside the arbitrary container or not, this verification is performed in parallel.

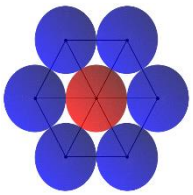


Figure 1. Spheres hexagon [3].

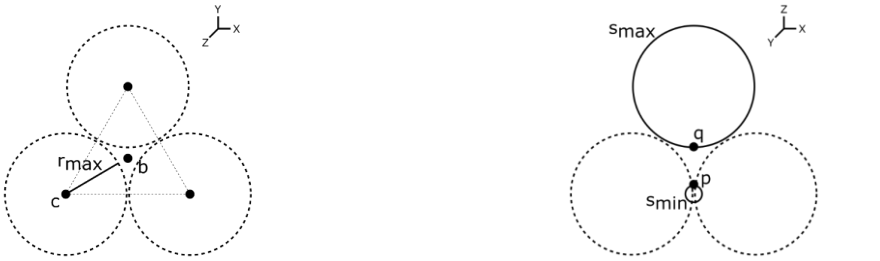
3.1.1. First Phase

First the 3D mesh of the arbitrary container is stored, then the radii r_{min} and r_{med} are calculated based on the input radius r_{max}. To calculate r_{min}, Equation 1 is used, where b is the barycentre of the upper right triangle and c is the centre of the central sphere, this can also be seen in Figure2a. To calculate r_{med} two points are used, r_{max} and r_{min}, the two points are $p = (s_{min}.x, s_{min}.y, s_{min}.z + r_{min})$ and $q = (s_{max}.x, s_{max}.y, s_{max}.z - r_{max})$, where s_{min}represents the sphere with radius r_{min} that must be added and s_{max} represents the sphere with radius r_{max} that only varies its position z in relation to the sphere with radius r_{min}, this can be seen in Equation 2 and Figure 2b. The position of the sphere with radius r_{min} is the barycentre of the upper right triangle. The position of the sphere with radius r_{med} only varies its position on the z axis, therefore, its position on the x axis is the same as that of the x axis of s_{max} and its position on the y axis is also the same as the position on the y axis of s_{max}; its position in z can be seen in Equation 3.

$$\boldsymbol{r_{min} = distance(b, c) - r_{max}} \tag{1}$$

$$r_{med} = \frac{distance(p, q)}{2} \tag{2}$$

$$c_{med}.z = \frac{(s_{max}.z - r_{max}) + (s_{min}.z - r_{min})}{2} \tag{3}$$



(a) Data for r_{min} [3] (b) Data for r_{med} and c_{med} [3]

Figure 2. Data for radii.

In this phase, a rectangular container is filled that surrounds the arbitrary container, the filling of the rectangular container is done in layers starting at the minimum point of this container until reaching its maximum point. The filling is done with a small structure called hepta-sphere, that seen in the xy plane moves in the direction of the positive x axis until reaching the limit and then moves in the direction of the positive y axis until reaching the limit, in this way one layer would be filled, then it moves in the direction of the positive z axis to fill the next layer. In the filling of the enveloping rectangular container, only the centres of the spheres are taken into account for the collisions with the container, this is done to cover a greater possible number of spheres. The hepta-sphere can be seen in Figure 3, the value of their positions is shown below:

$$p1 = (p0.x + 2r_{max}, p0.y, p0.z)$$
$$p2 = (p0.x + r_{max}, p0.y + (2r_{max}sin60°), p0.z)$$
$$p3 = (p0.x, p0.y - (r_{max} + r_{min}), p0.z)$$
$$p4 = \left(p0.x + r_{max}, p0.y - \left((r_{max} + r_{min})cos60°\right), p0.z\right)$$
$$p5 = \left(p0.x + r_{max}, p0.y + \left((r_{max} + r_{min})cos60°\right), p0.z\right)$$
$$p6 = (p0.x, p0.y + (r_{max} + r_{min}), p0.z)$$

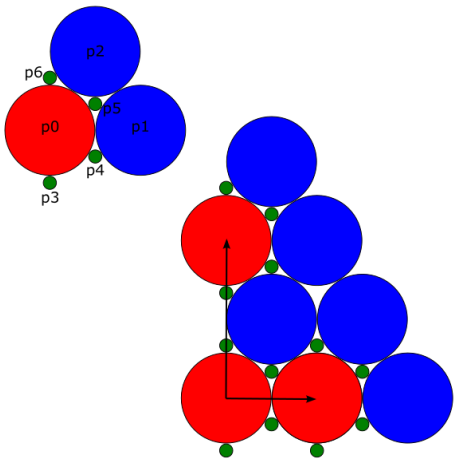


Figure 3. In the left part, a hepta-sphere and in the right, possible movements [3].

The procedure for filling a layer can be seen in Figure 4a, between every three spheres with radius r_{max} there is a sphere with radius r_{min} and two spheres with radius r_{med} as seen in Figure 4b.

(a) One xy layer filling. (b) One min sphere and two med spheres [3].

Figure 4. Data for layers.

3.1.2. Second Phase

In this phase it is verified if a sphere from the previous phase is inside the arbitrary container. For each sphere, it is first verified that the centre of the sphere is inside the arbitrary container, for this purpose raycast is used using the Jimenez et al. [17] algorithm for the intersection of a segment with a triangle, then it is verified if the sphere collides with any triangle of the mesh, for this it is necessary to find the closest point of a triangle, for this the Eberly [18] algorithm is used. If a sphere of radius r_{max} intersects some triangle, it is replaced by 61 spheres of radius r_{mid}, the positions of these spheres and their radius are obtained by recalculating the positions and radius of the dense packing of 61 spheres of Pfoertner [19]. If a sphere of radius r_{med} intersects any triangle of the container's mesh, this sphere is replaced by a sphere of radius r_{min}. Replacements are also checked.

This phase is the part that is parallelized, its form of parallelization is dividing the number of spheres of the first phase, this parallelization was done using CPU threads and GPU threads.

3.2. Method Improvement

As can be inferred from the previous method, the number of spheres is not the only thing to take into account, but also the triangles, since despite being parallel, each sphere is compared with all the triangles, being an operation expensive, then it is planned to reduce the cost of said operation using indexes in a data structure, the data structures to choose were the octree and the uniform grid, finally the last one was chosen as the basis for the data structure to be used due to the amount of memory used, indexes are used so that each sphere is compared only with the triangles that must be compared, the creation of this structure is parallelized to reduce the time as much as possible.

3.2.1. Data Structure

As mentioned above, the data structure used is based on a uniform grid. A uniform grid is a set of boxes of the same size that have things inside. The proposed structure is a matrix in which each value represents the type of intersection between the information belonging to the row and column indices. Boxes will be represented taking their size as data for their creation, their positions will come out of their indices using Equation 4, where $bpos$ is the position of the box on an axis, $cmin$ is the minimum point of the grid on the same axis, idx is the index of the box on the same axis and be is half the distance of the box on the same axis. The number of boxes is given by dividing the size of the enveloping rectangular container by axis by the size of the boxes. In this case we must place both triangles and spheres, so it was decided to have one structure for the triangles and another for the spheres. In the first data structure the rows are the

indices of the boxes and the columns are the indices of the triangles, in the second data structure the rows are the indices of the spheres and the columns are the indices of the boxes.

$$bpos = cmin + \big((2idx + 1)be\big) \tag{4}$$

The data structure for triangles has values 0 and 1, where 0 is that the triangle does not intersect the box and 1 is that the triangle intersects the box. The data structure for spheres has values 0, 1, and 2, where 0 is that the sphere does not intersect the box, 1 is that the sphere intersects the box, and 2 is that the centre of the sphere is in the box, this last value is to avoid calculations, since to use the raycast only the centre of the spheres is used. These values are to reduce time, however, having an array consumes significant memory. For the intersection of a triangle in a box, the Separating Axis Theorem (SAT) algorithm will be used, since there would only be thirteen quick checks [20], for the intersection of a sphere with a box the distances per axis are calculated [21], and for the algorithm of a point in a box a simple check per axis is used. The method for box-sphere intersection has some slight modifications, so it is shown in Figure 5, where bc is the centre of the box, be is half the size of the box, c is the centre of the spheres, and $radius$ is the radius of the sphere.

Data: $bc, be, c, radius$
$bxmin \leftarrow bc[0] - be$
$bymin \leftarrow bc[1] - be$
$bzmin \leftarrow bc[2] - be$
$bxmax \leftarrow bc[0] + be$
$bymax \leftarrow bc[1] + be$
$bzmax \leftarrow bc[2] + be$
$dmin \leftarrow 0$
if $c[0] < bxmin$ **then**
  $dmin \leftarrow dmin + (c[0] - bxmin)^2$
else if $c[0] > bxmax$ **then**
  $dmin \leftarrow dmin + (c[0] - bxmax)^2$
end if
if $c[1] < bymin$ **then**
  $dmin \leftarrow dmin + (c[1] - bymin)^2$
else if $c[1] > bymax$ **then**
  $dmin \leftarrow dmin + (c[1] - bymax)^2$
end if
if $c[2] < bzmin$ **then**
  $dmin \leftarrow dmin + (c[2] - bzmin)^2$
else if $c[2] > bzmax$ **then**
  $dmin \leftarrow dmin + (c[2] - bzmax)^2$
end if
return $dmin <= radius^2$

Figure 5. Box-sphere intersection algorithm.

3.2.2. Sphere Inside the Mesh

That changes with respect to the previous method is the use of data structures, both triangles and spheres, to reduce time in knowing if a sphere is inside the 3D mesh of the arbitrary container. As the initial datum is the index of the sphere, the boxes linked to it are searched for, and with the boxes linked to it, the triangles linked to the boxes are searched, that is, the triangles linked to the sphere are searched for, in this way it is not necessary to perform calculations with all the triangles, however, there is a problem with this, in the raycast algorithm a counter is used per triangle segment intersection and if any triangle is found in more than one box there will be problems since will count more than once, this problem is solved with a revision of the next one, that is, it will look in the next box to check if the index of the compared triangle is also there, if so, the current possible intersection will not be performed, of this way only one calculation will be performed per triangle.

3.2.3. Parallelization

The method parallelization works with the previous method parallelization, and adds a parallelization to the generation of the data structures. The parallelization of the method of Cuba and Loaiza [3], worked by dividing the number of spheres for the calculation of the verification of spheres, since it was what took more time, however, due to the creation of the data structures, this time is reduced. considerably, being the generation of the data structures what takes more time now, for which the generation of the data structures is also parallelized. As the data structures are matrices, they can be treated as arrays that are divided to then find the row and column indices, then with these indices make the intersections with boxes, either triangles or spheres. In this way we have two parallel parts, the generation of the data structures and the verification of the spheres inside the mesh of the arbitrary 3D container. The new parallelization of the method considerably reduces the packing construction time, although depending on the number of boxes the memory can become too high, therefore, the amount of this can be decided in the creation of the packing, that is, as input data we will have not only the radius r_{max}, but also the size of all boxes. Implementations of parallelizations are shown in Figure 6, Figure 7a and Figure 7b, where $tgrid$ is the grid of triangles, $sgrid$ is the grid of spheres, $bsizes$ are the sizes of the grid for each axis, $cmins$ are the values of the minimum point of the grid, $triangles$ are the triangles, $spheres$ are the centres of the spheres, $radius$ is the radius used in $spheres$, bxs_{num} is the number of boxes, trs_{num} is the number of triangles, $sphrs_{num}$ is the number of spheres and $valids$ are the positions of the spheres inside the arbitrary domain.

Data: $valids$, $bsizes$, $cmins$, $spheres$, $radius$, $triangles$, $tgrid$, $sgrid$, trs_{num}, bxs_{num}, $sphrs_{num}$

1: $i \leftarrow threadIdx.x + blockIdx.x * blockDim.x$
2: **if** $i < sphrs_{num}$ **then**
3: $valids[i] \leftarrow SphereInsideMesh(i, bsizes, cmins, spheres, radius, triangles, tgrid, sgrid, trs, bxs)$
4: **end if**

Figure 6. Spheres validation algorithm.

Data: $tgrid$, be, $bsizes$, $cmins$, $triangles$, trs_{num}, bxs_{num}

$bc[3] \leftarrow null$
$tr[9] \leftarrow null$
$tg_{id} \leftarrow threadIdx.x + blockIdx.x * blockDim.x$
$tr_{id} \leftarrow tg_{id}$ mod trs
$bx_{id} \leftarrow tg_{id}/trs$
$xpos \leftarrow bx_{id}/(bsizes[1] * bsizes[2])$
$ypos \leftarrow (bx_{id}/bsizes[2])$ mod $bsizes[1])$
$zpos \leftarrow bx_{id}$ mod $bsizes[2]$
if $bx_{id} < bxs_{num}$ **and** $tr_{id} < trs_{num}$ **then**
  $bc[0] \leftarrow cmins[0] + ((2 * xpos + 1) * be)$
  $bc[1] \leftarrow cmins[1] + ((2 * ypos + 1) * be)$
  $bc[2] \leftarrow cmins[2] + ((2 * zpos + 1) * be)$
  $tr[0] \leftarrow triangles[tr_{id} * 9]$
  $tr[1] \leftarrow triangles[tr_{id} * 9 + 1]$
  $tr[2] \leftarrow triangles[tr_{id} * 9 + 2]$
  $tr[3] \leftarrow triangles[tr_{id} * 9 + 3]$
  $tr[4] \leftarrow triangles[tr_{id} * 9 + 4]$
  $tr[5] \leftarrow triangles[tr_{id} * 9 + 5]$
  $tr[6] \leftarrow triangles[tr_{id} * 9 + 6]$
  $tr[7] \leftarrow triangles[tr_{id} * 9 + 7]$
  $tr[8] \leftarrow triangles[tr_{id} * 9 + 8]$
  $tgrid[bx_{id} * trs_{num} + tr_{id}] \leftarrow$ '0'
  if $IntersectBxTr(bc, be, tr) is$ **true then**
    $tgrid[bx_{id} * trs_{num} + tr_{id}] \leftarrow$ '1'
  end if
end if

(a) Triangles grid generation algorithm.

Data: $sgrid$, be, $bsizes$, $cmins$, $spheres$, $radius$, $sphrs_{num}$, bxs_{num}

$bc[3] \leftarrow null$
$c[3] \leftarrow null$
$sg_{id} \leftarrow threadIdx.x + blockIdx.x * blockDim.x$
$bx_{id} \leftarrow sg_{id}$ mod bxs_{num}
$sphr_{id} \leftarrow sg_{id}/bxs_{num}$
$xpos \leftarrow bx_{id}/(bsizes[1] * bsizes[2])$
$ypos \leftarrow (bx_{id}/bsizes[2])$ mod $bsizes[1])$
$zpos \leftarrow bx_{id}$ mod $bsizes[2]$
if $sphr_{id} < sphrs_{num}$ **and** $bx_{id} < bxs_{num}$ **then**
  $c[0] \leftarrow spheres[sphr_{id} * 3]$
  $c[0] \leftarrow spheres[sphr_{id} * 3 + 1]$
  $c[0] \leftarrow spheres[sphr_{id} * 3 + 2]$
  $bc[0] \leftarrow cmins[0] + ((2 * xpos + 1) * be)$
  $bc[1] \leftarrow cmins[1] + ((2 * ypos + 1) * be)$
  $bc[2] \leftarrow cmins[2] + ((2 * zpos + 1) * be)$
  $sgrid[sphr_{id} * bxs_{num} + bx_{id}] \leftarrow$ '0'
  if $IntersectSphrBx(bc, be, c, radius)$ is **true then**
    $sgrid[sphr_{id} * bxs_{num} + bx_{id}] \leftarrow$ '1'
    if $PointInBox(bc, be, c)$ is **true then**
      $sgrid[sphr_{id} * bxs_{num} + bx_{id}] \leftarrow$ '2'
    end if
  end if
end if

(b) Spheres grid generation algorithm.

Figure 7. Grid generation algorithms.

4. RESULTS

This section presents the tests performed as well as their results, the implementation was done in C++ OpenGL using different hardware to provide a further study of the method. The range of densities to be obtained for each container will be between 60% and 70% based on the method of Cuba and Loaiza [3]. First, it will be shown how the proposed method works compared to the method of Cuba and Loaiza [3], then comparisons of both methods will be made using containers with a high number of triangles, finally the comparison of the parallel methods will be made using different graphics cards. The size used for the box is four times the size of the sphere, this is because less memory is used without significantly affecting time.

4.1. Comparison of Methods Using Containers from Cuba and Loaiza [3]

The method of Cuba and Loaiza [3] is compared with the proposed method in different containers, the data of these containers are found in Table 1. The results of the comparisons are found in Table 2. For this comparison, the algorithms are running on an Intel Core i7-8550U @1.80GHz with 12GB of RAM, 8 threads and an NVIDIA GeForce MX130 graphics card under Windows 11 64bits. The radius r_{max} chosen for these tests is $0.2u$, however, in the Torus and the Stanford Dragon, a density between 60% and 70% was not reached due to the shape of their meshes, for which, the radius was reduced until reaching this density range. In the case of the Torus, the density reached with radius r_{max} $0.2u$ was 59.78%. In the case of the Stanford Dragon, since it is smaller, it was decided to start with a radius r_{max} of $0.1u$ where its density was 53.03%.

Table 1. Containers.

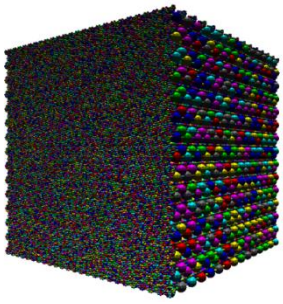
Container	**Measures (u)**	**Triangles**	**Volume (u^3)**
Cube	Width (10), height (10), depth (10)	12	1000.0
Cone	Radius (5), height (10)	62	261.80
Cylinder	Radius (5), height (10)	124	785.40
Capsule	Spherical radius (5), cylindrical height (10)	832	245.44
Sphere	Radius (5)	960	523.60
Torus	Max radius (5), min radius (1.25)	1152	154.21
Stanford Bunny	Domain size ($8.07 \times 8.11 \times 6.17$)	7202	109.90
Stanford Dragon	Domain size ($2.24 \times 3.52 \times 5.00$)	21782	6.95

Table 2. Comparison of times.

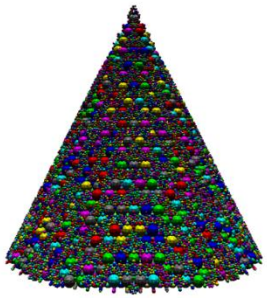
Container	**Radius (u)**	**Density (%)**	**Cuba and Loaiza [3]**		**Proposed Method**	
			Sequential Time (s)	**Parallel Time (s)**	**Sequential Time (s)**	**Parallel Time (s)**
Cube	0.2	71.96	0.07	0.47	2.28	0.81
Cone	0.2	69.34	0.30	0.58	1.56	0.76
Cylinder	0.2	71.11	0.76	0.55	2.42	1.04
Capsule	0.2	72.95	3.06	1.11	0.78	0.82
Sphere	0.2	70.65	8.43	2.01	3.67	1.36
Torus	0.15	63.27	11.44	2.31	4.82	1.89
Stanford Bunny	0.2	64.69	42.97	5.54	4.47	2.24
Stanford Dragon	0.05	64.00	910.12	75.31	78.82	25.35

The results of Table 2 show the advantage of the proposed method compared to the method of Cuba and Loaiza [3] in the cases of the Stanford Bunny and the Stanford Dragon, however, in the other containers, the time differences are small, due to their low number of triangles, so tests are then performed using containers with a high number of triangles.

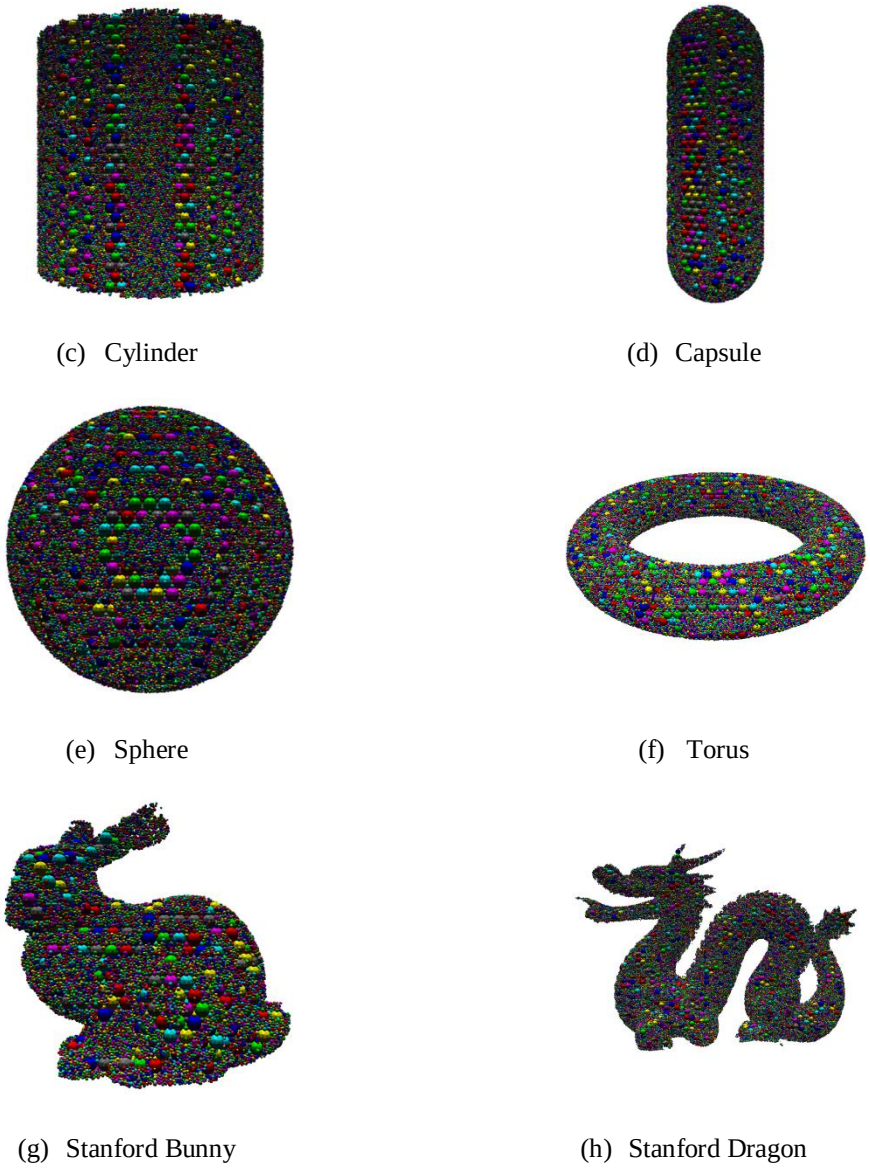
The visual results of the packing made in Table 2 are shown in Figure 8, in these images it is observed that with the densities reached in this table the objects correctly show their shape, including details such as the ears of the Stanford Bunny or the tongue of the Stanford Dragon.



(a) Cube



(b) Cone



(c) Cylinder (d) Capsule

(e) Sphere (f) Torus

(g) Stanford Bunny (h) Stanford Dragon

Figure 8. The proposed method in different containers.

4.2. Comparison of Methods Using Containers with a High Amount of triangles

The method of Cuba and Loaiza [3] is compared with the method proposed in the Stanford Bunny and the Stanford Dragon used previously, but with more triangles, these two models were modified by McGuire [22]. The Stanford Bunny used for this comparison has 144046 triangles and is called Stanford Bunny HT. The Stanford Dragon used for this comparison has 871306 triangles and is called Stanford Dragon HT. This comparison occurs in more powerful hardware since the triangles of each mesh are many. The comparison is made on an Intel Core i7-8700 @3.70GHz with 64GB of RAM, 12 threads and an NVIDIA GeForce RTX 2060 graphics card under Windows 10 64 bits. The results of this comparison are shown in Table 3.

Table 3. Comparison of times in container with high number of triangles.

Container	Radius (u)	Density (%)	Cuba and Loaiza [3]		Proposed Method	
			Sequential Time (s)	Parallel Time (s)	Sequential Time (s)	Parallel Time (s)
Stanford Bunny HT	0.2	64.78	411.39	21.56	47.56	4.70
Stanford Dragon HT	0.05	64.07	17639.96	454.52	1144.77	50.24

The results of Table 3 show a clear advantage of the proposed method compared to the method of Cuba and Loaiza [3] both in its sequential forms and in its parallel forms. To have a deeper analysis of their parallel forms, a comparison of both methods on three different graphics cards is made below.

4.3. Comparison of Parallel Methods Using Different Hardware

A comparison of the parallel method of Cuba and Loaiza [3] and the proposed parallel method is made using the containers of Table 1 and Table 3, these results are shown in Table 4.

Table 4. Comparison of times in parallel methods.

Container	Radius (s)	Density (%)	Times (s)					
			Cuba and Loaiza [3]			Proposed Method		
			Intel Core i7-8550U @1.80GHz, 12GB RAM, NVIDIA MX130	Intel Core i7-8700 @3.70GHz, 64GB RAM, NVIDIA RTX 2060	Intel Core i5-10400F @2.90GHz, 16GB RAM, NVIDIA RTX 3060	Intel Core i7-8550U @1.80GHz, 12GB RAM, NVIDIA MX130	Intel Core i7-8700 @3.70GHz, 64GB RAM, NVIDIA RTX 2060	Intel Core i5-10400F @2.90GHz, 16GB RAM, NVIDIA RTX 3060
Cube	0.2	71.96	0.47	0.13	0.20	0.81	0.21	0.22
Cone	0.2	69.34	0.58	0.15	0.17	0.76	0.21	0.21
Cylinder	0.2	71.11	0.55	0.19	0.19	1.04	0.22	0.24
Capsule	0.2	72.95	1.11	0.28	0.22	0.82	0.17	0.18
Sphere	0.2	70.65	2.01	0.44	0.40	1.36	0.27	0.27
Torus	0.15	63.27	2.31	0.52	0.50	1.89	0.33	0.31
Stanford Bunny	0.2	64.69	5.54	1.28	1.22	2.24	0.45	0.39
Stanford Dragon	0.05	64.00	75.31	11.44	12.14	25.35	1.95	1.65
Stanford Bunny HT	0.2	64.78	97.90	21.56	21.50	38.26	4.70	4.32
Stanford Dragon HT	0.05	64.07	4854.48	454.52	472.89	1602.52	50.24	47.28

The results of Table 4 show that the proposed method obtains densities between 60% and 70% in less than a minute in all the tests carried out, except for the Stanford Dragon HT using the NVIDIA GeForce MX130 graphics card, where this container has 871306 triangles. This indicates that the method works well, since its execution time is very low, except in cases with a very high number of triangles on moderately powerful hardware.

5. CONCLUSIONS

The limitations of the proposed method are aimed at the number of triangles contained in the model and the size of the radius chosen as input, since reducing this radius increases the number of spheres. An excessive number of triangles would saturate the GPU memory and consume more time, the same happens when reducing the input radius too much. This excessive number of triangles and spheres is in the millions. This indicates that the method would fail when constructing a packing of spheres with a density close to or greater than 60% in a highly detailed 3D model.

The proposed method does not affect the use in arbitrary domains or the densities reached by the method of Cuba and Loaiza [3], these densities are between 60% and 70%. In the tests carried out, it is observed that the improvement in time is high both in sequential and in parallel, for which it is considered a successful improvement of the algorithm. The times achieved by the method are relatively low even with a high number of triangles and the memory is not saturated for densities between 60% and 70%. This indicates the effectiveness of the method for arbitrary containers.

6. FUTURE WORKS

As future works, it is planned to create a software for the industry with indications and restrictions of the method so that it can be used in 3D printing. This software will be used in the material reduction of a 3D print, the spheres will be holes in the 3D model that allow to print a 3D model with internal supports. When printing a model, an amount of material of approximately the density percentage of the packaging is saved, that is, when using the proposed method in 3D printing, between 60% and 70% of material will be saved in a model in the internal part. For external supports, it is planned to modify the method by reducing the thickness of the possible supports generated, in this way they can be broken without damaging the quality of the 3D printing. This software will save a large amount of material in a 3D print.

ACKNOWLEDGEMENTS

M. E. LOAIZA acknowledges the financial support of the CONCYTEC – BANCO MUNDIAL Project “Mejoramiento y Ampliación de los Servicios del Sistema Nacional de Ciencia Tecnología e Innovación Tecnológica” 8682-PE, through its executing unit PROCIENCIA, within the framework of the call E041-01, Contract No. 002-2020-FONDECYT.

REFERENCES

[1] H. M. Jaeger, S. R. Nagel, and R. P. Behringer, “Granular solids, liquids, and gases,” *Reviews of modern physics*, vol. 68, no. 4, p. 1259, 1996.
[2] J. Duran, *Sands, powders, and grains: an introduction to the physics of granular materials*. Springer Science & Business Media, 2012.
[3] R. A. Cuba Lajo and M. E. Loaiza Fernandez, “Parallel sphere packing for arbitrary domains,” in *International Symposium on Visual Computing*. Springer, 2021, pp. 447–460.
[4] Y. Wu, X. An, and A. Yu, “Dem simulation of cubical particle packing under mechanical vibration,” *Powder technology*, vol. 314, pp. 89–101, 2017.
[5] B. Zhao, X. An, Y. Wang, Q. Qian, X. Yang, and X. Sun, “Dem dynamic simulation of tetrahedral particle packing under 3d mechanical vibration,” *Powder technology*, vol. 317, pp. 171–180, 2017.
[6] H. Tangri, Y. Guo, and J. S. Curtis, “Packing of cylindrical particles: Dem simulations and experimental measurements,” *Powder technology*, vol. 317, pp. 72–82, 2017.

[7] J. Gan and A. Yu, "Dem study on the packing density and randomness for packing of ellipsoids," *Powder Technology*, vol. 361, pp. 424–434, 2020.
[8] S. Zhao, N. Zhang, X. Zhou, and L. Zhang, "Particle shape effects on fabric of granular random packing," *Powder technology*, vol. 310, pp. 175–186, 2017.
[9] A. D. Rakotonirina, J.-Y. Delenne, F. Radjai, and A. Wachs, "Grains3d, a flexible dem approach for particles of arbitrary convex shape—part iii: extension to non-convex particles modelled as glued convex particles," *Computational Particle Mechanics*, vol. 6, no. 1, pp. 55–84, 2019.
[10] E. Campello and K. R. Cassares, "Rapid generation of particle packs at high packing ratios for dem simulations of granular compacts," *Latin American Journal of Solids and Structures*, vol. 13, no. 1, pp. 23–50, 2016.
[11] L. Wang, X. An, Y. Wu, Q. Qian, R. Zou, and K. Dong, "Dem simulation of vibrated packing densification of mono-sized regular octahedral particles," *Powder Technology*, vol. 384, pp. 29–35, 2021.
[12] X. Wang, Z.-Y. Yin, D. Su, X. Wu, and J. Zhao, "A novel approach of random packing generation of complex-shaped 3d particles with controllable sizes and shapes," *Acta Geotechnica*, pp. 1–22, 2021.
[13] E. Lozano, D. Roehl, W. Celes, and M. Gattass, "An efficient algorithm to generate random sphere packs in arbitrary domains," *Computers & Mathematics with Applications*, vol. 71, no. 8, pp. 1586–1601, 2016.
[14] Y. Li and S. Ji, "A geometric algorithm based on the advancing front approach for sequential sphere packing," *Granular Matter*, vol. 20, no. 4, pp. 1–12, 2018.
[15] R. Weller and G. Zachmann, "Protosphere: A gpu-assisted prototype guided sphere packing algorithm for arbitrary objects," in *ACM SIGGRAPH ASIA 2010 Sketches*. Association for Computing Machinery, 2010, pp. 1–2.
[16] R. Weller, U. Frese, and G. Zachmann, "Parallel collision detection in constant time," in *Workshop on Virtual Reality Interaction and Physical Simulation*. The Eurographics Association, 2013.
[17] J. J. Jimenez, R. J. Segura, and F. R. Feito, "A robust segment/triangle intersection algorithm for interference tests. efficiency study," *Computational Geometry*, vol. 43, no. 5, pp. 474–492, 2010.
[18] D. Eberly, "Distance between point and triangle in 3d," *Geometric Tools*, 2020.
[19] H. Pfoertner, "Numerical results for densest packing of n equal spheres in a larger sphere with radius=1," 2013, Accessed on: Aug. 2, 2021. [Online]. Available: http://oeis.org/A084827/a084827.txt
[20] C. Ericson, *Real-time collision detection*. CRC Press, 2004.
[21] J. Arvo, "A simple method for box-sphere intersection testing," in *Graphics gems*, 1990, pp. 335–339.
[22] M. McGuire, "Computer graphics archive," July 2017, Accessed on: Dec. 1, 2021. [Online]. Available: https://casual-effects.com/data

AUTHORS

Cuba Lajo Rubén Adrián is currently a Bachelor of Computer Science from Universidad Católica San Pablo, Arequipa, Perú (2022). His current research interest focus on Computer Graphics and Videogames.

Loaiza Fernández Manuel Eduardo holds a PhD in Computer Science from Pontifícia Universidade Católica Do Rio De Janeiro, Rio de Janeiro, Brasil (2009). His current research interest focus on Computer Graphics and Computer Vision.