Rapid Random Packing of Poly-disperse Spheres using Adam Stochastic Optimization

Mykhailo Novikov
University of Luxembourg
Esch-sur-Alzette, Luxembourg
mihabayt@gmail.com

Xavier Besseron
University of Luxembourg
Esch-sur-Alzette, Luxembourg
0000-0002-7783-2185

***Abstract*—This paper introduces a machine learning-based approach to solving the 3D sphere packing problem, with a particular focus on applications in Discrete Element Method (DEM) simulations. Diverging from traditional methods that explore abstract mathematical spaces and higher-dimensional constructs, we propose a practical strategy: (1) using generic triangular meshes of convex shapes as containers, and (2) adhering to a predefined size distribution. This approach is highly relevant for real-world applications in engineering and computer graphics.**

Our methodology defines an objective function to penalize both inter-particle overlap and violations of the container's boundary, then minimizes this function using the modern stochastic optimization algorithm, Adam. To achieve high efficiency, we rely on a collective arrangement technique that enables the rapid packing of large numbers of spheres in a feasible time. We further enhance scalability by iteratively adding and optimizing batches of particles, allowing our implementation to handle large packed beds.

The paper presents a detailed description of the algorithm and a thorough numerical evaluation that validates the results and provides insights into performance. With this algorithm, we successfully packed 200,000 particles in a tall vertical container box with a square base in 1 hour and 17 minutes, achieving an average core packing density of approximately 0.6. Finally, we also demonstrate the applicability of the method to complex and real-world configurations.

***Index Terms*—Sphere Packing, Poly-disperse Spheres, Machine Learning, Stochastic Optimization (Adam), Discrete Element Method (DEM), Granular Materials, Computational Geometry**

I. Introduction

The sphere packing problem has been a point of extensive research due to its rich theoretical challenges and wide-ranging applications across various scientific disciplines. In physics and chemistry, efficient sphere arrangement models assist in simulating material behaviors at atomic and molecular levels [1]. They serve as foundational models for understanding the structures of granular materials, liquids, glasses, powders, crystals, colloids, living cells, and polymeric systems [2]–[5]. In geology, sphere packings are crucial for modeling the mechanical properties of geomaterials such as soils, concrete, and rocks [6]–[9].

Multiple approaches have been developed to address the sphere packing problem, each with unique strengths and limitations. Traditional algorithms often employ molecular dynamics to grow spheres until they jam, generating dense packings but at significant computational cost, particularly for large systems. Gradient-based optimization methods minimize potential energy and overlaps, offering efficiency in certain scenarios but facing scalability challenges in complex or large-scale simulations. Lattice-based techniques achieve high densities in structured, periodic packings typical of crystalline materials but are less effective for the random packings found in glasses, sands, and powders.

In this paper, we introduce a novel solution to a more challenging version of the packing problem of poly-disperse spheres (i.e. with varied sizes). Spheres with pre-defined radii are arranged inside a three-dimensional container so they do not overlap and remain fully within the container's boundary. Our method involves a modern stochastic optimization technique commonly used in machine learning, the Adam optimizer, for efficient convergence of a relatively easy-to-compute objective function. The proposed approach employs a collective arrangement technique of successive batches, enabling the packing of millions of spheres in a practical time frame. By minimizing computational costs and taking the desired radii distribution into account, our method addresses key limitations of existing techniques.

The proposed approach is specifically tailored for practical applications in Discrete Element Method (DEM) simulations, and more precisely to create realistic particle packed beds for initial conditions. DEM has emerged as a powerful computational tool for simulating the micro-mechanical properties of granular materials, allowing researchers to investigate macroscopic behaviors by analyzing micro-scale interactions [5], [10], [11]. It has been widely used to optimize processes in steel-making [12] and metallurgy, such as cold compaction during powder transfer, compaction, and sintering stages [4], but also biomass combustion, drying, and pyrolysis [13], [14].

To validate the practical implementation of our proposed framework, we have developed a custom application that demonstrates its effectiveness. We present experimental results obtained from this approach, showcasing its suitability for large-scale applications in engineering, scientific research, and visualization tasks. These results highlight not only the efficiency and scalability of our method but also its potential to advance the state-of-the-art in sphere packing solutions and demonstrate its applicability for various industries.

II. Background and Related Works

The Lubachevsky-Stillinger (L–S) algorithm is a classic approach to sphere packing that has been extensively used for several decades. It is essentially a non-equilibrium protocol that grows spheres gradually over time, starting from an initial, random arrangement until the spheres become jammed and reach a near-maximum packing fraction. The algorithm utilizes molecular dynamics to control the growth rate of spheres, with faster growth leading to more disordered packings and slower growth allowing more ordered and denser configurations [15]. The L–S algorithm has proven effective for both mono-disperse and poly-disperse sphere systems and was widely considered the workhorse for generating dense random packings in three-dimensional space. However, a significant limitation of the L–S approach is its computational cost. Specifically, while the L–S algorithm required approximately 48 hours to pack a system of 10,000 spheres in a study conducted in 2002 [16]. Although modern computers are much faster than those available in 2002, the runtime of the L–S algorithm is influenced by its inherent computational complexity, which may limit the extent of performance gains achievable solely through hardware improvements.

The Adaptive Shrinking Cell (ASC) formulation [17] is another method for packing nonoverlapping spheres, particularly using a lattice-based packing where the fundamental cell structure is repeated periodically across the container. The study offers comprehensive tables comparing its performance against the Lubachevsky-Stillinger method, highlighting its robustness in achieving high-density packings in structured, lattice-type configurations.

The paper by Sutou and Dai [18] presents a global optimization approach for packing unequal spheres in three-dimensional polytopes, which is formulated as a nonconvex quadratic optimization problem. Their method utilizes several optimization techniques such as linearization, linear programming (LP) relaxation, the simplex method, and the branch-and-bound (B&B) algorithm to solve the problem. While these techniques are powerful, they require substantial memory resources, especially when applied to large numbers of particles, making them unsuitable for large-scale simulations. The high memory demands, along with the need for solving numerous linear programs during the optimization process, severely limit the scalability of their approach.

Campello & Cassares [19] introduced a "layer-by-layer" approach for generating dense particle packs, primarily for DEM simulations of granular compacts. Their method begins with the generation of a "layer" of non-overlapping spherical particles, which evolve dynamically under compacting pseudo forces. Their approach also involves a layering mechanism, similar to our method, to create high packing ratios. Their formulation is physics-based, includes considerations of repulsive, attractive, and rotational forces, and utilizes DEM simulations to simulate the compression and jamming of layers. Although they claim their technique to be both fast and simple, no explicit speed metrics are provided, leaving its efficiency relatively unclear.

The ProtoSphere algorithm [20], introduced in 2010, uses a greedy, iterative method to insert the largest possible non-overlapping sphere at each step. This approach, optimized for arbitrarily shaped objects with complex geometries, computes an approximation of Voronoi nodes to determine sphere placement. Designed for applications in computer graphics, ProtoSphere uses CUDA for parallel processing, enabling rapid packing performance and making it particularly suitable for large-scale or real-time scenarios. However, ProtoSphere does not provide control over particle size distributions, making it less suitable for granular material simulations or engineering contexts.

XProtoSphere, introduced in 2023 [21], extends ProtoSphere by enabling multi-sized sphere packing, addressing its limitations in particle size distribution control. By integrating particle size distribution guidance and optional discrete element-based relaxation (DEM), XProtoSphere achieves close adherence to a predefined particle size distribution (PSD). While particles may not exactly match the PSD, this extension significantly improves its suitability for modeling granular materials, such as soils and powders. However, XProtoSphere involves additional computational steps to refine sphere placement and regulate PSD, making it more resource-intensive compared to the original.

J.-F. Jerier et al. in 2009 introduced a geometric method for solving the sphere packing problem using tetrahedral meshes [22]. This approach involves first filling each tetrahedron with spheres in contact, followed by detecting voids and filling them with additional spheres. While the method can achieve a high packing density, it produces lattice-based structures. This approach, typically used to model metals and crystalline ceramics, contrasts with our random packing method that aims to model glasses, sand, and powders. In the subsequent work [23], Jerier et al. refined this geometric algorithm, achieving significant improvements in performance, flexibility, and scalability.

These approaches generally focus on a simpler variant of the sphere packing problem: finding radii that fit within the void spaces to optimize packing density. In contrast, our work brings up a more complex and challenging version of the problem—arranging particles with a predetermined radius distribution. This is particularly relevant in practical applications where particle sizes are dictated by physical properties or manufacturing constraints, requiring a solution that follows the given distribution rather than exploiting void spaces. Our approach utilizes the Adam optimizer [24], commonly employed in machine learning, to directly minimize overlaps while maintaining all spheres inside the container. It enables gradient-based optimization for efficient packing that exactly follows the predefined PSD.

We summarize in Table I the main characteristics of the related works and compare them to our proposed approach.

TABLE I
COMPARISON OF APPROACHES TO SPHERE PACKING

Approach	Container / Scenario	Poly-Disperse Support	Metrics & Performance	Scalability / Limitations
Lubachevsky-Stillinger (L-S) [15], [16]	Grows spheres gradually (nonequilibrium MD)	Mono- or poly-disperse (primarily mono)	Near-max packing fraction; slow for large nb of spheres	High computational cost limits large systems due to sphere-growth simulation
Adaptive Shrinking Cell (ASC) [17]	Lattice-based, periodic tiling of container space	Mainly mono-disperse, with possible extensions	High density in structured, crystalline-like packings	Less suitable for random or large-scale non-lattice arrangements
Sutou & Dai [18]	3D polytopes via B&B + LP relaxation	Poly-disperse (unequal spheres)	Good packing fraction; high memory usage (B&B)	Quickly becomes infeasible for large systems due to memory/computational demands
Campello & Cassares [19]	Layer-by-layer DEM for granular compacts	Poly-disperse	High packing ratio in DEM context; limited direct speed/time metrics	Physics-based with repulsive/attractive forces; efficiency details not deeply specified
ProtoSphere [20], XProtoSphere [21]	Greedy iterative insertion (GPU-accelerated); often for graphics	Mono- or poly-disperse (approx. PSD)	Very fast for large or complex geometry;	Aimed at computer graphics; exact PSD alignment may still deviate
Jerier et al. [22], [23]	Geometric method (tetrahedral meshes)	Typically mono or limited poly-disperse	High packing density, well-suited for metals / crystalline ceramics	Produces lattice-like arrangements; less suitable for random or broad PSD
Proposed Method	Convex container, random packing using Adam	Poly-disperse with user-defined PSD	Packing density ≈ 0.6	Scales linearly in nb of spheres; supports complex shapes; parallel CPU vectorization

III. A NEW APPROACH TO PARTICLE PACKING

We first provide a generic description of the problem followed by a mathematical modeling. Finally, we conclude this section with a detailed presentation of our proposed algorithm.

A. Problem description

The goal is to arrange a given set of spherical particles within a three-dimensional container in such a way that specific objectives are optimized, subject to geometric constraints. This task is non-trivial due to the complexity of the container's shape, the interactions between particles, and the need to balance multiple competing criteria such as density, overlap minimization, and stability.

The container is modeled as a three-dimensional space bounded by a triangular mesh structure. Each particle is modeled as a sphere defined by its radius and position (center coordinates). Particle radii are fixed and given, but their positions within the container are to be optimized. Particles may not significantly overlap, and the degree of overlap is penalized in the objective function. Particles must remain within the boundaries of the container. For computational tractability, the particles are packed in the container in successive batches, allowing localized optimization. Each batch (or layer) of spheres is added to the system along the gravity axis one after another. The spheres of the previous layers are fixed, and only of spheres of the current batch are considered for optimization.

The primary objective in the particle packing process is to reduce the total overlap between particles to ensure feasible packing and avoid physical implausibility, minimize the extent to which particles extend outside the container boundaries, and minimize the total vertical placement of particles to achieve a stable and compact packing arrangement along the gravity axis. With this foundation, we now proceed to detail the mathematical modeling, objective function derivation, and optimization strategy for solving this problem.

B. Problem modeling

In this section, we provide the details of our approach. We describe the key terms and then craft the objective function.

We model the container using a triangular mesh structure. The mesh is defined by a set of vertices $\mathcal{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N\}$, where each vertex $\mathbf{v}_i = (x_i, y_i, z_i)$ is a point in three-dimensional space $\mathbb{R}^3$. From this set of vertices, we construct a set of interconnected triangles $T = \{\Delta_1, \Delta_2, \ldots, \Delta_M\}$. Each triangle $\Delta_j \in T$ is formed by three vertices $(\mathbf{v}_{j1}, \mathbf{v}_{j2}, \mathbf{v}_{j3}) \in \mathcal{V}$ and represents a facet of the container's surface. The triangles are connected along their edges, forming a continuous mesh that defines the boundary of the container.

To facilitate the calculations involved in our optimization process, we require an explicit mathematical representation of the container's boundary in terms of plane equations. While the mesh T provides a detailed description of the container's surface, directly working with all its facets can be computationally intensive. Therefore, we approximate the container's shape by its convex hull, which simplifies the computations while still capturing the essential geometric constraints. The convex hull of $\mathcal{V}$, denoted as $\text{Conv}(\mathcal{V})$, is the smallest convex set that contains all the vertices in $\mathcal{V}$.

We obtain the plane equations defining the convex hull by applying QHULL algorithm [25] to the set of vertices $\mathcal{V}$. QHULL computes the convex hull of the points in $\mathcal{V}$, yielding a set of plane equations H, where each plane corresponds to a facet of the convex hull:

$$H = \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ \vdots & \vdots & \vdots & \vdots \\ a_m & b_m & c_m & d_m \end{bmatrix}$$

Each row in H represents a plane equation in the form:

$$a_k x + b_k y + c_k z + d_k = 0, \quad \text{for } k = 1, 2, \ldots, m.$$

The convex hull can be represented as the intersection of the half-spaces defined by these planes:

$$\text{Conv}(\mathcal{V}) = \bigcap_{k=1}^{m} \left\{ \mathbf{x} \in \mathbb{R}^3 \,\middle|\, a_k x + b_k y + c_k z + d_k \leq 0 \right\}.$$

Since all vertices in $\mathcal{V}$ satisfy these inequalities, they are contained within $\text{Conv}(\mathcal{V})$:

$$\mathcal{V} \subseteq \text{Conv}(\mathcal{V}).$$

C is a matrix of coordinates of sphere centers, which serves as matrix of parameters to optimize, and r is a given vector of spheres radii.

$$C = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ ... & ... & ... \\ x_n & y_n & z_n \end{bmatrix}$$

$$r = [r_1, r_2, ...r_n]$$

where n is the overall number of particles.

In our approach, we define a penetration term $P_{C^2,r^2}^{C^1,r^1}$ to quantify the total overlap between particles (spheres) from the respective sets of coordinates and radii. This function calculates the sum of the absolute values of negative Euclidean distances between particle pairs, effectively measuring the total penetration depth when particles overlap. It starts with calculating the overlap distance (penetration depth) between particles i and j:

$$\delta_{ij} = ||\mathbf{c}_i - \mathbf{c}_j|| - r_i - r_j$$

$$\delta_{ij}^{-} = \min(0, \delta_{ij})$$

where $\mathbf{c}_i$ and $\mathbf{c}_j$ represent the i-th and j-th rows of the matrix C, respectively.

Then, it computes the sum of absolute values of negative distances between each sphere:

$$P_{C^2,r^2}^{C^1,r^1} = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} p_{ij} \tag{1}$$

where

$$n_1 = \text{rows of } C_1 = \text{elements of } r_1$$

$$n_2 = \text{rows of } C_2 = \text{elements of } r_2$$

and with $p_{ij} = |\delta_{ij}^{-}| = -\delta_{ij}^{-}$ if $i \neq j$, and 0 otherwise. In this way, p_{ij} quantifies the penetration depth of sphere i into sphere j.

The second term is an Exterior Distance $E_H^{C,r}$ - a measurement of the positive distances between spheres and container. For each sphere with center (x_i, y_i, z_i) and radius r_i, we compute the signed distance ρ_{ik} from the sphere's center to each plane k:

$$\rho_{ik} = \frac{a_k x_i + b_k y_i + c_k z_i + d_k}{\sqrt{a_k^2 + b_k^2 + c_k^2}}$$

$$\tilde{\rho}_{ik} = \rho_{ik} + r_i$$

If $\tilde{\rho}_{ik} > 0$ it means the sphere extends outside the container through plane k, otherwise $\tilde{\rho}_{ik} \leq 0$. We are interested in penalizing distances for spheres which are outside the container, so the final term looks like:

$$E_H^{C,r} = \sum_{i=1}^{n} \sum_{k=1}^{m} \max(0, \tilde{\rho}_{ik}) \tag{2}$$

Another term A^C represents the total vertical placement of the spheres, aiming to minimize the cumulative height along the gravity axis. In this paper we assume that gravity is influencing particles by z-axis in Cartesian space, however in practice any direction can be used.

$$A^C = \sum_{i=1}^{n} z_i \tag{3}$$

This term cannot always tend to zero, because all particles cannot be at the origin and rather pile on top of each other. It also depends on the position of the container relatively to the origin. However, this term helps push particles downward along the gravity axis, ensuring a realistic arrangement.

As we introduced all the necessary terms, now we can define the objective function we are interested to minimize. Combining 1, 3, 2 we obtain:

$$Z(C) = \alpha P_{C,r}^{C,r} + \beta A^C + \gamma E_H^{C,r} \tag{4}$$

where α, β, γ are weight parameters.

This function is highly non-linear functions with numerous local minima. That is why we will use Adam optimizer with AMSGrad to iteratively optimize this objective function. We introduce machine learning analogies: our model *parameters* or *weights* are represented as C matrix, and an optimization step is one cycle of calculating the objective function value and updating model weights based on their gradient information.

Adam [24] computes individual adaptive learning rates for each parameter. This means that parameters (particle coordinates) with sparse gradients get larger updates, and parameters with dense gradients get smaller updates. In complex loss landscapes with varying curvature and scale, adaptive learning rates help the optimizer make appropriate step sizes in different directions, improving convergence.

Adaptive Moment Estimation with stable steps [26] is a suitable variant of the Adam optimizer. AMSGrad addresses convergence issues observed in the original Adam algorithm when dealing with non-convex optimization problems. AMSGrad modifies Adam to ensure convergence to optimal solutions, even in non-convex settings with multiple local minima. It maintains a non-increasing step size by taking the maximum of past squared gradients, which helps prevent the learning rate from becoming too small. By avoiding the pitfalls of adaptive

learning rates shrinking too rapidly, AMSGrad often leads to solutions that generalize better. Learning rate lr is a crucial parameter which will define how fast our model learns, i.e. how fast particles move in the direction of gradient.

C. Algorithm

Although the objective function and optimization methods are straightforward, the optimization problem rapidly grows in size with the number of particles. As our goal is to pack large amount of particles in feasible amount of time, we can not afford iterative calculation of this objective function for all particles in the system. To alleviate this issue, particles are packed in layers, one batch after the other, similar to Campello & Cassares [19].

In this approach, all the particles of the previous layers are fixed (already optimized) and cannot be moved anymore. Then, a new batch of particles is generated and packed on top of the previous layers.

More precisely, we use C' and r' to denote respectively the coordinates and radii of the particles of the previous layers. For each new batch (or layer), the particles are initially generated randomly in the space above the top of the last layer (defined as $\max_i(C'_i + r'_i)$). Then the particles' positions are optimized according to the objective function 4. For this, an additional penetration term $P^{C,r}_{C',r'}$ is added to the objective function to account for the collisions between the current batch (C, r) and the previous layers (C', r'). Thus, we optimize this objective function:

$$Z(C) = \alpha P^{C,r}_{C,r} + \beta A^C + \gamma E^{C,r}_H + \alpha P^{C,r}_{C',r'} \tag{5}$$

where C are the optimization parameters, i.e. the coordinates of the new particle batch to be optimized, and C' the fixed coordinates of the previously packed particles.

The pseudo-code of the packing is provided in the algorithm 1. It features two main loops. The external *while* loop packs the layers one by one. At the beginning of a batch, a set of radii is generated (following the prescribed settings) and initial positions are generated in the area above the previous layer. The batch size is defined initially and decreased dynamically if the packing fails (i.e. if the average overlap between spheres or with the convex hull is above a given threshold). This allows to stop the packing once the containers is full. If the packing of the batch is successful, the best positions found are added to the previous layer and the packing continue with a new batch.

The internal *while* loop tries to find the best positions for the current batch of particles. At every iteration, it calculates the objective function $Z(C)$ with the current values of C and calls the Adam optimizer to update C in the direction of the gradient. This loop ends when there have been no improvement of the objective function for many iterations ($patience$) or when the maximum number of iterations has been reached (max_steps).

Algorithm 1 Particle Packing Optimization

Input: $\mathcal{V}$, α, β, γ, batch_size, nb_max, max_steps, patience, PSD
Output: Positions of packed particles C'

Compute the convex hull: $H \leftarrow \text{Conv}(\mathcal{V})$
Initialize $C' \leftarrow \varnothing$, nb_packed $\leftarrow 0$
while nb_packed $<$ nb_max **and** batch_size > 0 **do**
  Generate radii r according to PSD (particle size distribution)
  Generate new random positions C above the last layer
  Define the objective function:
  $Z(C) = \alpha P^{C,r}_{C,r} + \beta A^C + \gamma E^{C,r}_H + \alpha P^{C,r}_{C',r'}$
  Initialize $Z_{\text{best}} \leftarrow Z$, step $\leftarrow 0$, no_improvement_counter $\leftarrow 0$
  while no_improvement_counter $<$ patience **and** step $<$ max_steps **do**
    Update C using Adam optimization on Z
    Update objective function value: Z
    if $Z > Z_{\text{best}}$ **then**
      no_improvement_counter $\leftarrow 0$
      $C_{\text{best}} \leftarrow C$, $Z_{\text{best}} \leftarrow Z$
    else
      no_improvement_counter++
    end if
    step++
  end while
  if *batch successfully packed* **then**
    Store optimized positions: C_{best}
    Aggregate results: $C' \leftarrow C' \cup C_{\text{best}}$
    nb_packed $\leftarrow$ nb_packed + batch_size
  else
    Reduce the batch size:
      batch_size $\leftarrow$ batch_size$/2$
  end if
end while

IV. Tuning of Hyper-Parameters

Our algorithm is based on numerous hyper-parameters, whose value must be set to actually execute the packing. Some parameters were set manually after empirical trial and errors. Some were defined using a more thorough analysis detailed below. In practice, for this work, we have used the values $patience = 50$ and $max\_steps = 2000$. They were selected empirically to balance execution time against result quality. Although Fig. 3 focuses on the choice of learning rate, it also illustrates how many iterations typically occur under these parameter settings. Likewise, the objective function (5) being a linear combination of individual objectives, we have chosen these values for the weights: $\alpha = 100$, $\beta = 10$ and $\gamma = 100$. The rationale is to penalize inter-particle overlaps (both $P^{C,r}_{C,r}$ and $P^{C,r}_{C',r'}$) and boundary violations ($E^{C,r}_H$) more heavily than the vertical position term (A^C).

For a deeper analysis, we focused the tuning on two hyper-parameters that are critical for our algorithm: the **batch size** and the **learning rate** of the Adam optimizer.

For this study, we ran our algorithm on simple configurations for the packing of mono-disperse spheres in a box, as illustrated in Fig. 1. The packing were executed on a single CPU computing node of the MeluXina super-computer [1].

[1] https://docs.lxp.lu/system/overview/

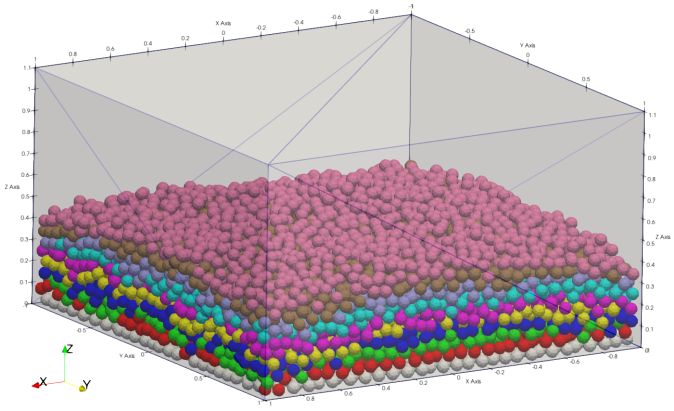

Fig. 1. Result of the packing of 10,000 particles in a box. The batches of 1,000 particles appears in different colors.

It is important to notice that our algorithm is stochastic by nature: particle radii are generated randomly following prescribed distributions, particle positions are initialized randomly, etc. In consequence, every execution returns different results in terms of the packing itself, and gives different execution times. This can be avoided by fixing the seed of the random generator in order to produce deterministic results. However, to better study the behavior of our algorithm, we ran each of our numerical experiments 10 times with different random seeds and display the average, minimum, and maximum values.

A. Impact of the batch size

We executed the same packing of 10,000 particles in a box using different batch sizes and we measured the execution time of the algorithm. Fig. 2 shows the packing time in function of the batch size. Each point is the average of 10 measurements, and the error bars represent the min/max values.

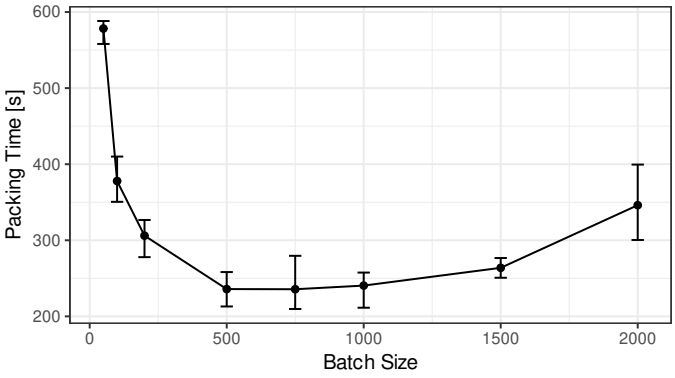

Fig. 2. Packing time in function of the batch size. For a given batch size, the point is the average of 10 measurements and the min/max are shown as error bars.

In this configuration, we can see an optimal batch size in the range of 500 to 1000. With a lower value, we pay the overhead of algorithm managing the batches. With a higher value, the performance are impacted by the cost of the objective function which requires to check all sphere-to-sphere overlaps with a complexity of $O(\text{batch\_size}^2)$.

B. Learning rate

The learning rate in Adam controls the size of the steps taken during the optimization process. It determines how much the parameters are updated in response to the estimated gradients of the loss function. Adam combines the advantages of AdaGrad (adaptive learning rates) and RMSProp (exponential moving average of squared gradients). Although Adam adapts learning rates for individual parameters based on gradient moments, the base learning rate still scales all updates. Our objective function is non-linear, non-convex, and involves multiple constraints, so small changes in parameters can lead to significant changes in the objective function value. This is due to the complex interactions between particles. Too high learning rate may cause the optimizer to take large steps, overshooting minima and also result in oscillations around minima or divergence. Too low learning rate, on the other hand, may lead to slow convergence. The optimizer may get stuck in flat regions or plateaus of the loss surface.

For this experiment, we take a single batch of 500 particles. We study the evolution of the value of the objective function (also called fitness) over the optimization steps. We consider three different fixed learning rates (10^{-2}, 10^{-3}, 10^{-4}) and also two initial learning rates (10^{-2}, 10^{-3}) adapted dynamically with the *ReduceLROnPlateau* scheduler of PyTorch.

The results are displayed in Fig. 3. Each line represents a *learning rate* configuration and shows the value of the fitness over the optimization steps of a single batch. The fitness value decreases with the progress of the optimization, and stops when the patience criteria is reached, i.e., no improvement in a given number of steps (here $patience = 50$), or maximum number of steps reached ($max\_steps = 10,000$ in this case). The final solution is also annotated with the execution time of the batch optimization, which is roughly proportional to the number of steps before convergence to the solution.

The fitness value, calculated by the objective function (5), indicates the quality of the solution: the lower, the better. However, it will not reach zero, even for the optimal solution, because the objective function includes the term A^C that is the sum of the vertical coordinates of spheres, and which depends on the location of the box and its shape.

In our results, a large fixed learning rate of 10^{-2} is able to converge quickly at the beginning but fails to improve the solution after some time. A small fixed learning rate of 10^{-4} is very slow to converge and reaches the maximum number of steps before finding a good solution. The medium fixed learning rate of 10^{-3} is able to converge to a better solution than the two formers in a reasonable time. The use of the *ReduceLROnPlateau* learning rate scheduler gives a significant improvement by dynamically reducing the learning rate when the fitness stopped improving. The changes of the learning rate is visible when the fitness suddenly drops after a plateau. The scheduler-based learning rate with an initial value of 10^{-2} gives the best compromise in this study: the fitness quickly

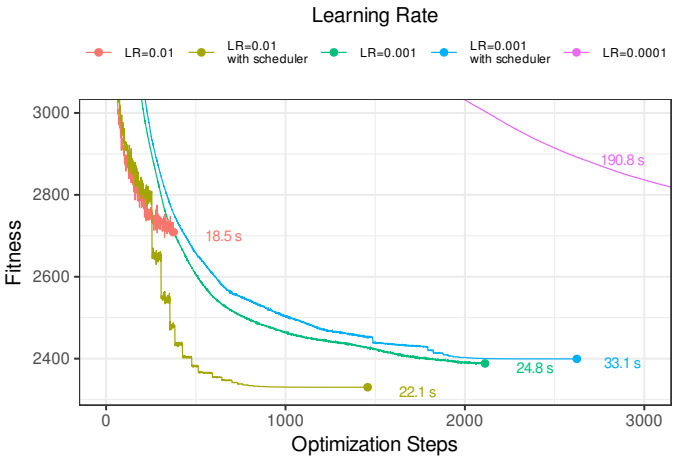

Fig. 3. Fitness value in function of the optimization step for different learning rates configurations.

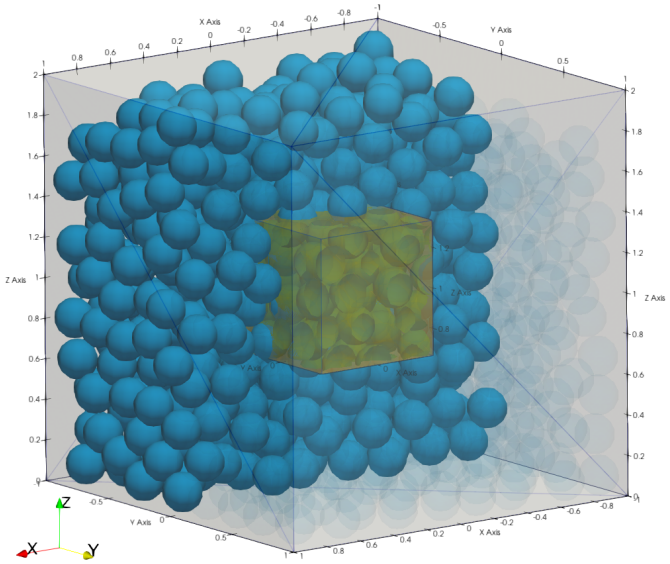

Fig. 4. Representation of the virtual inner box (in orange) used to measure the packing density at the core of the packing.

improves at the beginning and then continues to decrease when the learning rate is refined automatically.

V. Performance of the Algorithm

In this section, we investigate the efficiency of our algorithm focusing first on the quality of the result, and secondly on the execution time.

A. Packing density

We performed numerical experiments to evaluate the packing density achieved by our approach. Here, instead of optimizing the hyper-parameters for a better packing density, we measured the packing density resulting from the hyper-parameter values selected in the previous section.

We considered a simple container box (size $2 \times 2 \times 2$) packed to its maximum capacity with our algorithm using mono-disperse particles of size 0.1. To prevent the void effect caused by the walls of the container, we focus on a virtual inner box $1/3$ smaller located at the center of the container as illustrated on Fig. 4. To calculate precisely the packing density, we rely on the external library *overlap* for the exact calculation of overlap volume and cubes [27].

Our packing algorithm is a stochastic process which mean that every packing is different. We repeated the process 10 times and, in the given configuration, our approach managed to packed between 950 and 1006 particles in the box. The core density (measured in the virtual inner box) for the different executions are shown in Fig. 5 with values between 0.571 and 0.619, and an average of 0.597. Our results provide a packing quality similar to the physical packing methods [28] of *Loose Random Packing* (e.g., dropped into bed or packed by hand) with a density range of [0.59; 0.60] and the *Poured Random Packing* (e.g., spheres poured into bed) with a density range of [0.609; 0.625].

In all the repeated executions, the average overlap measured at the contact points was always below 1.1% of the particle radius. This is due to the line 19 of our algorithm 1 that prevents large overlaps by discarding batches non-conforming solutions (or batches) as explained in section III-C.

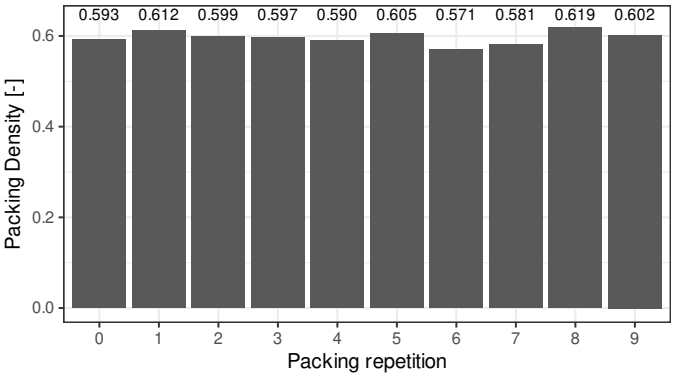

Fig. 5. Core packing density, for 10 executions, measured at the virtual inner box. The average packing density is 0.597.

B. Parallel execution with multiple threads

Our implementation is based on *PyTorch* [29] and relies heavily on vectorized tensor operations. Thus, it is able to benefit from significant acceleration when executed on a multi-cores system.

To evaluate the parallel performance of our implementation, we considered the packing of 10,000 particles of size 0.3 in a container box of a size size $2 \times 2 \times 2$ using a batch size of 500. We measured the time of the packing using different numbers of CPU cores on a single CPU compute node of the MeluXina supercomputer (128 physical cores).

The measurements are reported in Fig. 6. Each point is the average of 10 executions and the min/max values are shown using error bars. The speedup (relative to the sequential execution) is plotted in Fig. 7. The results show an acceleration

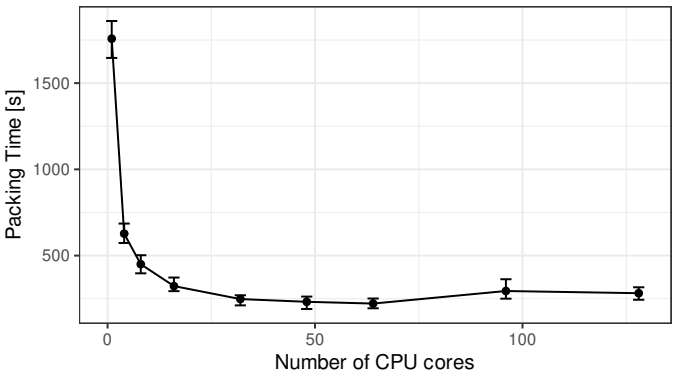

Fig. 6. Packing time in function of the number of CPU cores.

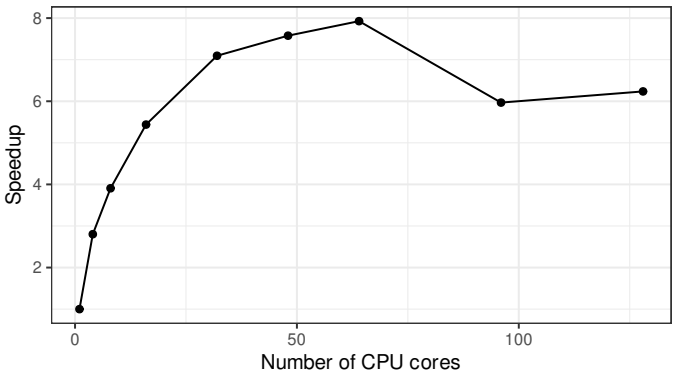

Fig. 7. Speedup in function of the number of CPU cores.

of 7.93 time compared to sequential for an execution on 64 cores.

C. Scalability with the number of particles

Finally, we study the performance of our algorithm according to the number of particles. This is an important characteristic in order to pack large number of particles as it can be found in industrial setup.

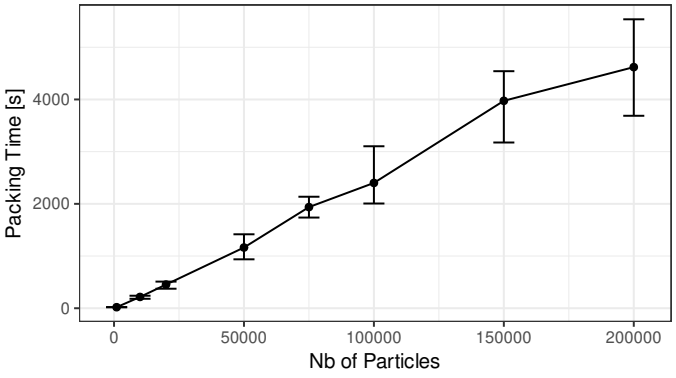

Fig. 8. Execution time in function of the number of particles.

This setup is based on a tall vertical container box with a square base of size 2×2 in which we pack particles of size 0.03 using a batch size of 500. The numerical experiment is repeated 10 times for each fixed number of particles. The results are displayed in Fig. 8 where each point is the average of 10 executions and the min/max values are shown as error bars. The execution time appears to be linear with the number of particles with an average of 1 hour and 17 minutes to pack 200,000 particles in this simple configuration. However, we can also see that the variability of the execution time increases significantly with the number of particles. This could be due to more frequent ill settlements of the particles that cause difficulty for the optimizer to find suitable non-overlapping configuration, and thus batches to fail.

VI. Implementation for Real-World Scenarios

In this section, we thoroughly describe the application we developed as well as introducing a realistic case of a blast furnace packing with a predefined radii distribution.

A. Software Implementation

The application is written in Python. For working with triangular mesh structure we use the Trimesh library [30]. For obtaining the convex hull H, we use Scipy's [31] *ConvexHull()* function. The library *overlap* [27] is used to calculate the overlap between spheres and cubes and estimated the packing density. We use *PyTorch* [29] to perform matrix operations, as well, as their implementation of Adam optimizer.

Parameters for each run in our application are configured via a configuration file written in YAML. Our application follows a flexible design, based on a Abstract Algorithm Runner, in order to ease the addition (and comparison) of new packing algorithms in the future.

The packing configuration is based on a few elements:

- A **container** that is defined as a (convex) STL shape that delimits the space area in which the spheres should be contained.
- **Particle sets** that defines a set of particles with common properties, in particular a radius distribution. Different radius distributions are supported: $Constant(value)$, $Uniform(min, max)$ and $Normal(mean, stddev)$.
- **Zones** represent a space area of the container to be filled with particles of given particle sets. This covers the cases where different types of particles should be used in the different parts of the container: For example, small particles at the bottom, and large particles at the top. In practice, a zone delimits a subspace of the container (using a slice or a STL shape) and specifies a mix of particle sets to use (e.g., 70% set 1 and 30% set 2).

An example of a YAML packing configuration file illustrating the different features is given in Fig. 9. The output of this complex packing configuration is shown in Fig. 10.

B. Packing of a Blast Furnace

Here we present a real scenario of usage of a industrial application. With this example, the objective is to create a packed bed to serve as initial conditions for a Midrex blast furnace simulation, similar to the settings described in [12]. In this work, we consider a Midrex blast furnace with a total

```yaml
container:
    path: "cone.stl"

algorithm: "COLLECTIVE_ARRANGEMENT"

params:
    lr: 0.01
    n_epoch: 1000
    patience: 50
    verbosity: 10

gravity_axis: z

particle_sets:
  - radius_distribution: "uniform"
    radius_min: 0.05
    radius_max: 0.08

  - radius_distribution: "normal"
    radius_mean: 0.04
    radius_std_dev: 0.005

zones:
  - n_particles: 200
    location:
        stl:
            path: "sphere.stl"
    set_proportions: [0.0, 1.0,]

  - n_particles: 300
    location:
        slice:
            axis: z
            min_bound: 0.8
            max_bound: 1.5
    set_proportions: [1.0, 0.0]
```

Fig. 9. Example of a YAML packing configuration for packing a cone with a sphere zone and a slide zone.

height of 32 m and a maximum diameter of 6.5 m. It features 4 gas inlets at the mid-height and 4 gas outlets (unrelevant for the packing). The furnace is filled with spherical particles with radii uniformly distributed between 5.2 cm and 7.5 cm. This setup is then used to run a coupled multi-physics simulation of the fluid with Computational Fluid Dynamics (CFD) and the particles with Discrete Element Method (DEM).

The result of the blast furnace packing is presented in Fig. 11. Using our approach, it was packed automatically with 430,062 particles in 31 hours and 11 minutes on a single compute node. While this time is significantly higher than the ones presented on Fig. 8, it is important to notice that this configuration represents a real-world scenario with a much more complex container and wider area that causes more interactions between the particles.

VII. Conclusion

In this paper, we introduced a novel approach for the random packing of poly-disperse spheres. Our aim is to provide a practical solution that is driven by real-world and industrial examples. In particular, we support complex convex shape for the container that can be provided as a generic STL file. Furthermore, the spheres follow a predefined size distribution that can be specified in an advanced configuration file.

Our proposed algorithm is based on the PyTorch machine learning framework and leverage modern steepest descent technique provided by the Adam optimizer. The numerical experimentations show a packing density in the range of 0.571 to 0.619, close to the *Loose Random Packing* and the *Poured Random Packing* methods. The performance evaluation demonstrates a speedup for parallel executions and highlights the linear behavior of the packing time in function of the number of particles.

In the end, our implementation provides a practical solution for the packing of complex systems, and future work will focus on improving the results and performance of real-world scenarios.

Acknowledgment

The simulations were performed on the Luxembourg national supercomputer MeluXina. The authors gratefully acknowledge the LuxProvide teams for their expert support.

References

[1] R. Zallen, *The Physics of Amorphous Solids*, 1st ed. Wiley, May 1998.
[2] E. Rentsen, "Recent Advances in Sphere Packing Problems," in *Mathematical Optimization Theory and Operations Research: Recent Trends*, Y. Kochetov, A. Eremeev, O. Khamisov, and A. Rettieva, Eds. Cham: Springer Nature Switzerland, 2022.
[3] P. A. Cundall and O. D. L. Strack, "A discrete numerical model for granular assemblies," *Géotechnique*, Mar. 1979.
[4] J.-F. Jerier *et al.*, "Study of cold powder compaction by using the discrete element method," *Powder Technology*, Mar. 2011.
[5] F. Nicot, L. Sibille, F. Donze, and F. Darve, "From microscopic to macroscopic second-order work in granular assemblies," *Mechanics of Materials*, Jul. 2007.
[6] G. Caumon, P. Collon-Drouaillet, C. Le Carlier de Veslud, S. Viseur, and J. Sausse, "Surface-Based 3D Modeling of Geological Structures," *Mathematical Geosciences*, Nov. 2009.
[7] F. V. Donze, V. Richefeu, and S.-A. Magnier, "Advances in Discrete Element Method Applied to Soil, Rock and Concrete Mechanics," *Electronic Journal of Geotechnical Engineering*, 2009.
[8] F. Camborde, C. Mariotti, and F. Donzé, "Numerical study of rock and concrete behaviour by discrete element modelling," *Computers and Geotechnics*, Dec. 2000.
[9] G. G. Gray, J. K. Morgan, and P. F. Sanz, "Overview of continuum and particle dynamics methods for mechanical modeling of contractional geologic structures," *Journal of Structural Geology*, Feb. 2014.
[10] C. Martin, D. Bouvard, and S. Shima, "Study of particle rearrangement during powder compaction by the Discrete Element Method," *Journal of the Mechanics and Physics of Solids*, Apr. 2003.
[11] B. Peters *et al.*, "XDEM multi-physics and multi-scale simulation technology: Review of DEM–CFD coupling, methodology and engineering applications," *Particuology*, Jun. 2019.
[12] X. Besseron, P. Adhav, and B. Peters, "Parallel Multi-Physics Coupled Simulation of a Midrex Blast Furnace," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Workshops*, ser. HPCAsia '24 Workshops. New York, NY, USA: Association for Computing Machinery, Jan. 2024.
[13] A. H. Mahmoudi, F. Hoffmann, B. Peters, and X. Besseron, "Numerical study of the influence of particle size and packing on pyrolysis products using XDEM," *International Communications in Heat and Mass Transfer*, Feb. 2016.
[14] X. Besseron, H. Rusche, and B. Peters, "Parallel Multi-Physics Simulation of Biomass Furnace and Cloud-based Workflow for SMEs," in *Practice and Experience in Advanced Research Computing (PEARC)*. Boston, MA, USA: ACM, Jul. 2022.
[15] B. D. Lubachevsky and F. H. Stillinger, "Geometric properties of random disk packings," *Journal of Statistical Physics*, Sep. 1990.
[16] A. R. Kansal, S. Torquato, and F. H. Stillinger, "Computer generation of dense polydisperse sphere packings," *The Journal of Chemical Physics*, Nov. 2002.
[17] S. Torquato and Y. Jiao, "Robust algorithm to generate a diverse class of dense disordered and ordered sphere packings via linear programming," *Physical Review E*, Dec. 2010.

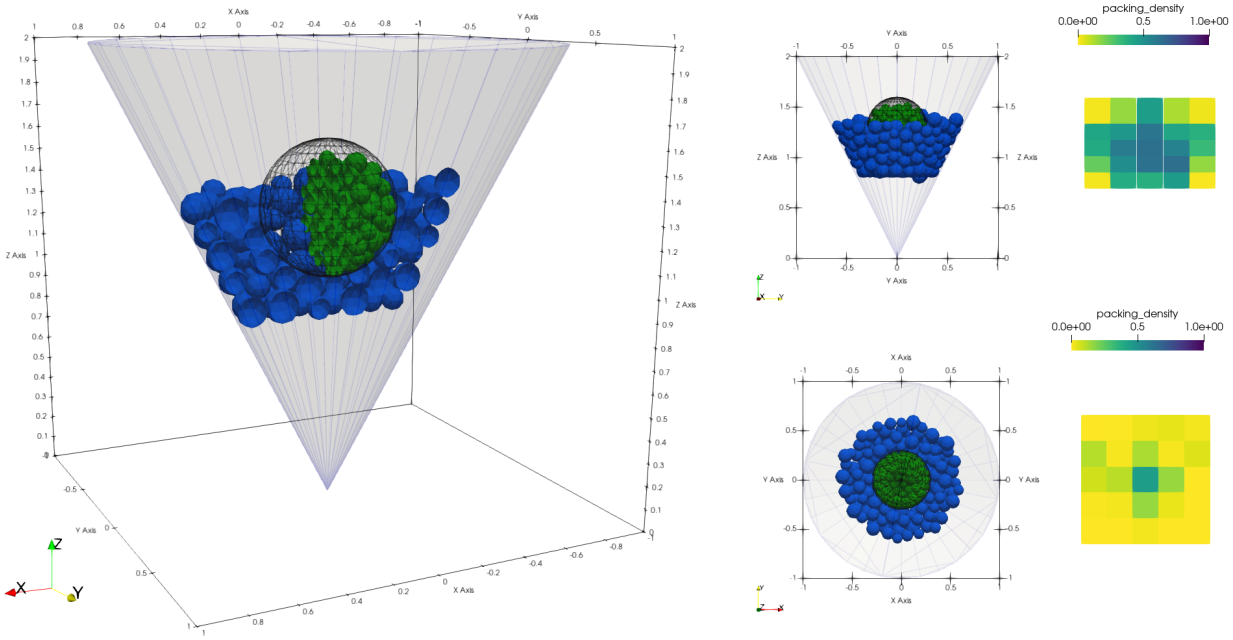

Fig. 10. Resulting packing of the cone-packing configuration listed in Fig. 9. The green particles are the ones generated for the first zone in the sphere STL shape. The blue particles are the one generated in the second zone in the horizontal slice.

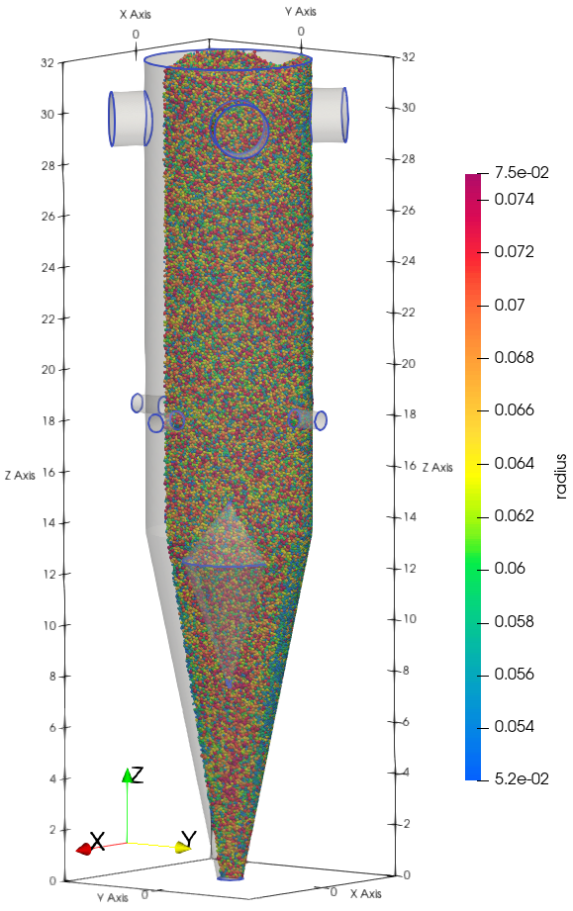

Fig. 11. Resulting packing of the real-world configuration for a blast furnace with 430,062 particles. Only a vertical half of the particles are displayed.

[18] A. Sutou and Y. Dai, "Global Optimization Approach to Unequal Global Optimization Approach to Unequal Sphere Packing Problems in 3D," *Journal of Optimization Theory and Applications*, Sep. 2002.

[19] E. M. B. 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*, Jan. 2016.

[20] R. Weller and G. Zachmann, "ProtoSphere: A GPU-assisted prototype guided sphere packing algorithm for arbitrary objects," in *ACM SIGGRAPH ASIA 2010 Sketches*. ACM, Dec. 2010.

[21] X. Wang, M. Fujisawa, and M. Mikawa, "XProtoSphere: An eXtended multi-sized sphere packing algorithm driven by particle size distribution," *The Visual Computer*, Aug. 2023.

[22] J.-F. Jerier, D. Imbault, F.-V. Donze, and P. Doremus, "A geometric algorithm based on tetrahedral meshes to generate a dense polydisperse sphere packing," *Granular Matter*, Jan. 2009.

[23] J.-F. Jerier, V. Richefeu, D. Imbault, and F.-V. Donzé, "Packing spherical discrete elements for large scale simulations," *Computer Methods in Applied Mechanics and Engineering*, May 2010.

[24] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," Jan. 2017.

[25] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Trans. on Mathematical Software*, Dec. 1996.

[26] S. J. Reddi, S. Kale, and S. Kumar, "On the Convergence of Adam and Beyond," Apr. 2019.

[27] S. Strobl, A. Formella, and T. Pöschel, "Exact calculation of the overlap volume of spheres and mesh elements," *Journal of Computational Physics*, Apr. 2016.

[28] D. P. Haughey and G. S. G. Beveridge, "Structural properties of packed beds — A review," *The Canadian Journal of Chemical Engineering*, Apr. 1969.

[29] J. Ansel *et al.*, "Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation," in *Proc. of the 29th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, Apr. 2024.

[30] Dawson-Haggerty et al., "Trimesh," Dec. 2019. https://trimesh.org/

[31] P. Virtanen *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, 2020.