Efficient Particle Packing Algorithm for Material Modeling

Guilherme Amadio
University of Illinois at Urbana-Champaign

Thomas L. Jackson
University of Florida

(Dated: October 31, 2014)

In this article, we present a particle packing algorithm for the generation of packings of polydisperse convex particles, such as spheres, ellipsoids, cylinders and polyhedra. This algorithm is an event-based method similar to the Lubachevsky–Stillinger (LS) algorithm, originally developed for packing disks and spheres. Similarly to the LS algorithm, our method relies on particle growth to increase packing density over time. In contrast to the LS algorithm, however, our method uses random particle displacements rather than elastic collisions to avoid particle intersections. Intersection queries are performed using the Incremental Separating Axis Gilbert–Johnson–Keerthi (ISA-GJK) algorithm, which is one of the fastest algorithms for convex polyhedra. We present an analysis of the performance of this new method, and provide material modeling examples to highlight the ability of our packing code to create realistic models with a large number of particles.

PACS numbers: 12.60.Rc, 81.05.Rm, 83.10.Rs
Keywords: Particle Packing, Solid Rocket Propellant, Heterogeneous Materials

I. INTRODUCTION

The determination of macroscopic—or *effective*—properties of heterogeneous materials from the properties of their constituents is an important engineering problem that has received a lot of attention in recent years [1–7]. Nevertheless, many studies have employed rudimentary models in which the microstructure of particulate composites is represented with disks and spheres (e.g., [4–6]). This is not an ideal situation in material modeling, since the microstructural information can play a crucial role in determining the behavior of materials in many circumstances. The propagation of mechanical shock waves within heterogeneous explosive materials, for example, is highly dependent on particle geometry, since the interface between the fuel and oxidizer phases is a source of reflected shock waves that can cause the local temperature to increase until detonation is triggered. The burning rate of solid rocket propellants, on the other hand, depends on the surface area of the fuel–oxidizer interface, which—for a fixed volume fraction of oxidizer—is a function of the size distribution of oxidizer particles and their geometry. Therefore, realistic models of the microstructure of these materials are essential for rocket design.

The solid propellants and explosive materials we intend to model typically comprise a mix of oxidizer particles embedded in a polymerized binding matrix. Oxidizer particles are often small crystals, hence their shapes most closely resemble polyhedra. Some common oxidizer materials found in solid propellants are ammonium perchlorate (AP), ammonium dinitramide (ADN), and ammonium nitrate (AN). In explosives, crystalline materials such as HMX, RDX, PETN, and CL–20 are used more often. In some cases, energetic crystals usually found in explosives are used in solid propellants to enhance their burning rate. Among binding materials, hydroxyl-terminated polybutadiene (HTPB), dicyclopentadiene (DCPD), and polybutadiene acrylonitrile (PBAN) are all quite common. In addition, many solid propellants contain additives to tune burning characteristics and reduce smoke (because of its environmental impact), among other things. An extensive discussion of solid rocket propellants can be found in [8].

Aluminum powder is a common additive incorporated into many solid rocket propellants—e.g., in the solid boosters of NASA's now retired space shuttle—to increase specific impulse. A typical composition of such a propellant might be, by weight, 70% AP, 18% aluminum, and 12% binder. The addition of aluminum has other desirable effects as well, such as damping some combustion chamber instabilities. However, the introduction of small aluminum particles makes modeling these propellants more complicated [9–12], since near the burning surface the binding matrix melts, enabling aluminum particles to migrate and agglomerate into larger particles, which then cause nozzle erosion and become a safety concern. Two key parameters that affect agglomeration are the mean size of aluminum particles and the average size of fuel-rich pockets in between the larger oxidizer particles (usually AP). Some additional concerns of rocket designers are shock to detonation transition and the determination of macroscopic properties such as thermal and electrical conductivity [6, 13], elastic properties [6], etc. They are important not only for performance reasons, but also because of safety—the unexpected detonation of a rocket often has severe consequences.

Although our interest in particle packings comes from their utility in the modeling of the microstructure of solid propellants and other energetic materials, they can be applied to a wide variety of other problems. In fact, packings of disks and spheres have played an important role in the modeling of liquids [14–16], glasses [17–19], colloids [20, 21], and granular media [4, 22–24]. Particle packings have also been employed in simulations of physical processes such as fluid flow through packed beds [25–28], and other seemingly unrelated areas of research, such as communication theory [29]. In contrast, packings of polyhedral particles remained relatively unexplored until more recently. Although Betke and Henk [30] have found the densest *lattice* packings of the Platonic and Archimedean solids analytically more than a decade ago, the properties of random packings of the same polyhedra were explored only much later [31–33]. Much effort has been dedicated to specific packing problems, such as the notoriously difficult densest packing of tetrahedra [34–39], while in material modeling, polyhedral packings are less common [40–42].

In this paper, we present a particle packing algorithm for the efficient generation of polydisperse convex hard-particle packings for use in material modeling. Our main goal is to generate realistic microstructures of particulate composites using smooth and polyhedral particles. A secondary—but equally important—goal is to provide a quantitative measure of how realistic the generated microstructures are if compared with real materials. Here, we shall address only the first of these two goals. For a long time, the Lubachevsky–Stillinger algorithm (LS) [18] has been the main choice for generating dense packings of convex particles of various shapes. However, its use has been limited when the particles to be packed have polyhedral shapes. In this case, it has been substituted by more efficient alternatives, such as the Adaptive Shrinking Cell (ASC) algorithm by Torquato and Jiao [31]. In the ASC algorithm, the particle-growth molecular dynamics method used in the LS algorithm is replaced with an optimization scheme that is solved via linear programming for spheres [43], and via a Monte Carlo method for polyhedra [31]. In our work, we retain aspects of both of these algorithms to produce packings of polyhedra using event-driven molecular dynamics and Monte Carlo techniques.

II. PACKING ALGORITHM

The basic idea of the LS packing algorithm is to start the simulation with a set of infinitesimally small disks or spheres, and allow them to grow over time while they undergo elastic collisions that act to prevent intersections. Since collisions are the only form of interaction between the particles, instead of using a fixed time step, the simulation can evolve from event to event, where an event is anything that changes the motion state of a particle. For this reason, the LS algorithm is said to be collision-driven or event-driven. Event-driven molecular dynamics (EDMD) packing algorithms have a strong appeal, since they have a direct connection with the underlying physical processes of liquids, glasses and crystals. Moreover, EDMD algorithms are in general very efficient for packing spheres and other smooth convex shapes, such as ellipsoids. However, when the LS algorithm is used to pack polyhedral particles, not only does the calculation of the collision response between particles become more complicated, but the sharp features of the particles cause them to often get locked into low-density states, or lead to numerical errors when computing the separating plane between colliding partners. Therefore, in order to address this problem, we substitute the elastic collisions with Monte Carlo trial displacements.

In a similar way to the LS algorithm, we begin the simulation by placing infinitesimally small particles inside a periodic domain. In contrast to the LS algorithm, however, the particles are not allowed to move with constant speed, but grow in place until they are about to overlap another particle. At this time, trial displacements are computed until one is found that delays the overlap. An event queue is kept to track the next time an overlap will occur. This improves the efficiency of the algorithm over other Monte Carlo methods in which displacements are applied to particles chosen at random. When no trial displacements can be found that delay the overlap, the neighbors of the colliding pair are also displaced. This procedure is necessary to avoid local jamming.

The substitution of elastic collisions with random displacements leads to many advantages. Since particles do not move, collision detection becomes much simpler; only Boolean intersection tests are needed. The time of intersection can be found via simple bisection, since particles must intersect at some point in the future if they grow indefinitely. In practice, however, the search interval can be greatly reduced by first computing the time of overlap of the bounding spheres of the particle pair to be tested. Furthermore, it becomes much simpler to extend the algorithm to work with a wider variety of particle shapes, since the only requirement is a Boolean intersection test.

A. Collision Detection

Fast collision detection is crucial for good performance. A previous effort by to extend the LS algorithm for polyhedra [40] has used a level set method to evaluate intersection tests, which greatly impacted the overall performance of that code, effectively limiting its application to systems with a relatively small number of particles ($< 10^4$). In [31], Torquato and Jiao mention that "the [separating] axis is either perpendicular to one of the faces or perpendicular to a pair of edges, one from each polyhedron. This reduces the number of axes that need to be checked from infinity to $[E(E-1)/2 + 2F]$, where E and F are the number of edges and faces of the polyhedron, respectively." However, testing all of these possibilities can become very expensive as the number of vertices of the polyhedra increases. Moreover, testing a finite number of axes does not allow one to mix polyhedra and smooth shapes in the same packing. Methods reported in the literature for smooth particles are usually based on overlap potentials that are solved algebraically [33, 44]. In our code, we use the Gilbert–Johnson–Keerthi algorithm (GJK) [45, 46] to evaluate intersection tests. In the GJK algorithm, the distance between two convex sets of points is obtained from the distance between their Minkowski difference and the origin. This reduces the problem of finding the distance between two convex set of points to finding the distance between a single convex set of points and the origin. The cleverness of this algorithm lies in the fact that the Minkowski difference of the convex sets need not to be computed explicitly, but may be sampled through the support mapping functions of each set, which are much simpler to calculate. Any convex set with a support mapping function can be used with the GJK: spheres, ellipsoids, cylinders, and polyhedra, for example, are all supported in our code. The GJK algorithm can also benefit from the fact that the same feature in each polyhedron will be the closest to the other over successive calls of the function, which often leads to $O(1)$ running time when the same pair is tested with different sizes multiple times in the bisection search for the time of overlap. The only disadvantage of the GJK algorithm is that support for concave shapes must be added separately via convex decomposition, or via alternative algorithms for concave shapes [47].

B. Event Processing

The event priority queue used in our code is based on a simple heap, represented as a complete binary tree (see e.g. [48] for details). In this implementation, each leaf of the tree is associated with a particle event, and each internal node of the tree has the label of the earliest event between its two children. No deletions need to be performed on the tree since when an event happens, a new prediction is made for that particle and the tree is updated accordingly. This avoids using memory allocations often, which would make the priority queue perform much worse, as the survey in [49] demonstrates. Although a more sophisticated algorithm for a queue that scales better with the number of particles exists [50], it is much more difficult to implement. Since our code spends most of its time evaluating particle intersections and constructing lists of neighbors of displaced particles, the priority queue based on a heap provides a good trade off between performance and complexity.

C. Spatial Partitioning

Collision detection and spatial partitioning have both been extensively discussed in the literature, given their importance to the industry of 3D games [51–53]. Clearly, evaluating pairwise intersection tests for all particles is unpalatable, as it would make the algorithm $O(N^2)$ for a system of N particles. Thus, a spatial partitioning scheme is necessary to speed up event predictions by only checking overlaps against a small set of neighboring particles. The obvious choice for such a partitioning scheme is to use a uniform grid of cells and keep a list of which particles belong to which cells. The cell size should then be carefully chosen to minimize the average number of neighbors. A grid that is too coarse will have many particles in the same cell, while a grid that is too fine will lead to a high number of cell transfers and updates, both leading to performance degradation. In highly polydisperse packings, it is possible that a grid becomes both too coarse and too fine at the same time, as shown in figure 1. In this case, a simple uniform grid is not enough to cope with the problem. Similarly, uniform grids are a suboptimal solution for packings of very elongated particles. Since our simulations rarely contain highly elongated particles, we have decided to use hierarchical grids of cells with an appropriate number of levels to accommodate particles of different sizes in optimally sized cells.

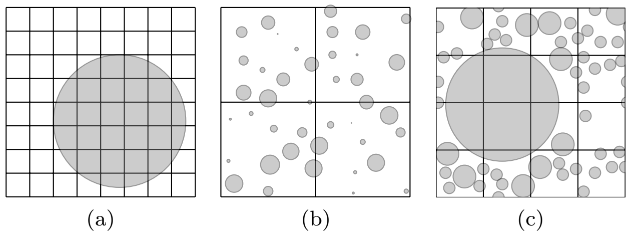
FIG. 1. Issues related to cell size: (a) grid is too fine, (b) grid is too coarse, (c) grid is both too coarse and too fine.

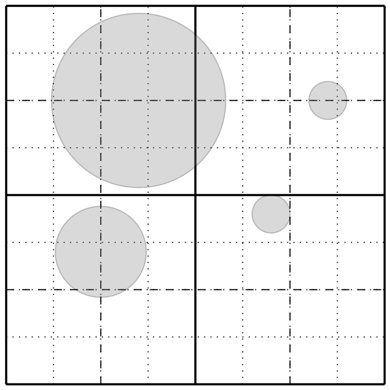
FIG. 2. Hierarchical cell grid. Larger particle uses grid cells with solid lines, medium sized particle uses grid cells with dashed lines, and smaller particles use grid cells with dotted lines. All particles remain in cells of optimal size.

At the highest level, the size of the cells is automatically calculated to hold the largest particle at the end of the simulation. Next, the number of levels needed is determined from the ratio between the largest and smallest particles such that the cells at the lowest level can contain the smallest particles throughout the simulation. At each level, the size of the cells is divided by two, effectively creating a hierarchy similar to an octree, although in our implementation the grid cells are defined only implicitly through a hash function. Each particle stores a hash value of the cell to which it belongs, and for each hash bucket, a doubly linked list is kept of all particles belonging to that bucket. Whenever a bucket becomes empty, it is destroyed. A secondary hash table is used to find the head of each list corresponding to a bucket in constant time. This approach can be implemented without the need for dynamic memory allocation during the simulation, greatly improving performance. It is better than using a static data structure, since it is much more memory efficient (fine grids occupy a lot of memory), and particle insertion and deletion are both constant time operations, whereas in a standard octree these operations take $O(\log N)$ time. Since only up to eight particles physically fit in each cell due to the hierarchical structure, each linked list of particles is kept short throughout the simulation. The hierarchical grid structure has a function to construct a list of neighbors that need to be checked for overlap when an event causes it to be displaced. For simplicity, each particle is stored in the grid based only on its centroid position and bounding radius.

III. PERFORMANCE ANALYSIS

The performance of our event-driven Monte Carlo (EDMC) packing algorithm depends on several factors, such as the number and shape of the particles being packed, polydispersivity, and the parameters that determine the size of trial displacements. Here, we perform an analysis of these factors to understand how this and other codes could be optimized for different types of workloads. We also compare the performance of our algorithm against our implementation of the LS packing algorithm that shares the same event-processing and spatial partitioning schemes. Both computer codes are serial due to the necessity that events be processed one at a time, but there are parts of the EDMC code that could be parallelized—something we intend to do in the future. All runs shown henceforth have been performed on a computer with an Intel Core i7–860 processor (2.8 GHz, 8 M cache), and 4 G of 1067 MHz DDR3 memory.

We begin with a simple scalability test to determine how the total packing time varies with the number of particles. Figure 3 shows packing time as a function of the number of spheres for monodisperse packings generated with the LS algorithm, and figures 4–8 show the same information for monodisperse packings of the platonic solids generated with the EDMC packing algorithm. In all runs, the initial condition is a dilute liquid of infinitesimal particles at rest. Both computer codes scale roughly linearly with the number of particles, although the cost of packing polyhedra is much higher than that of packing spheres. Packing spheres with the EDMC algorithm is also significantly slower than with the LS algorithm at high densities ($\phi > 0.60$). The EDMC algorithm is, nevertheless, more efficient than the LS algorithm to pack polyhedra due to the reasons discussed earlier.

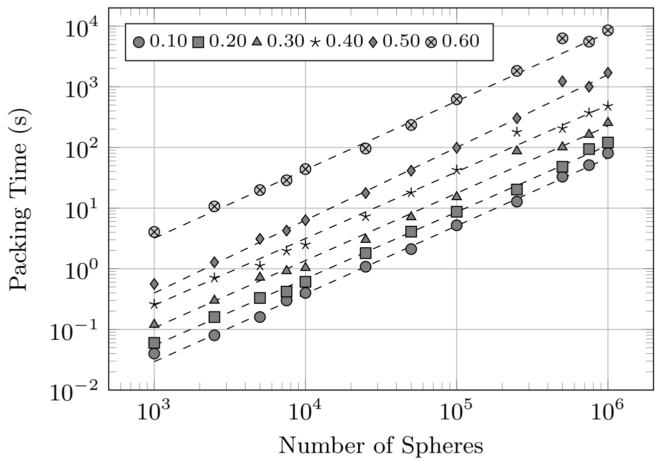

FIG. 3. Packing time of monodisperse spheres using the LS packing algorithm.

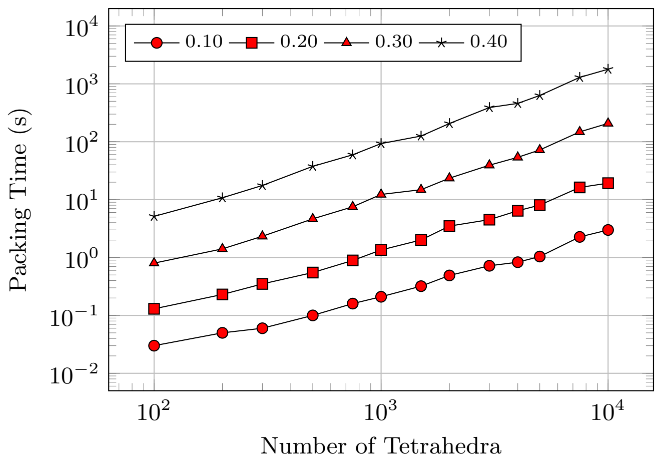

FIG. 4. Packing time of monodisperse tetrahedra using the EDMC packing algorithm.

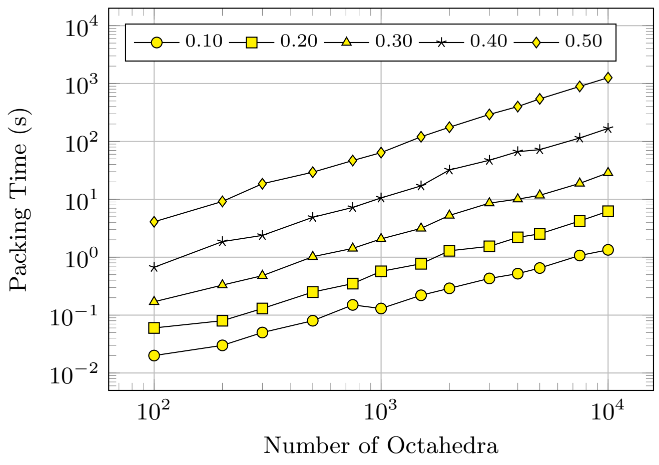

FIG. 5. Packing time of monodisperse octahedra using the EDMC packing algorithm.

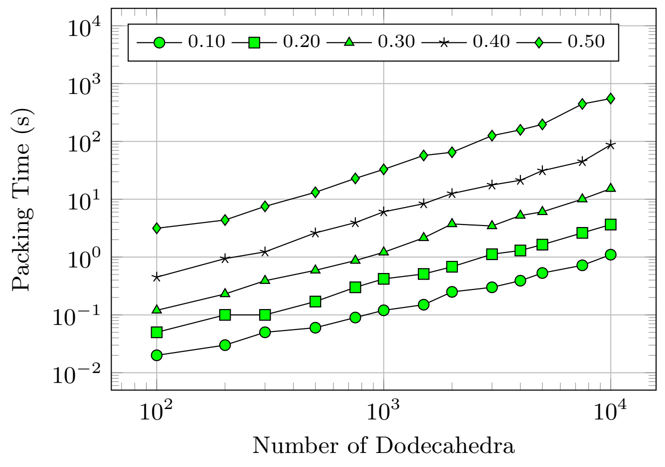

FIG. 6. Packing time of monodisperse dodecahedra using the EDMC packing algorithm.

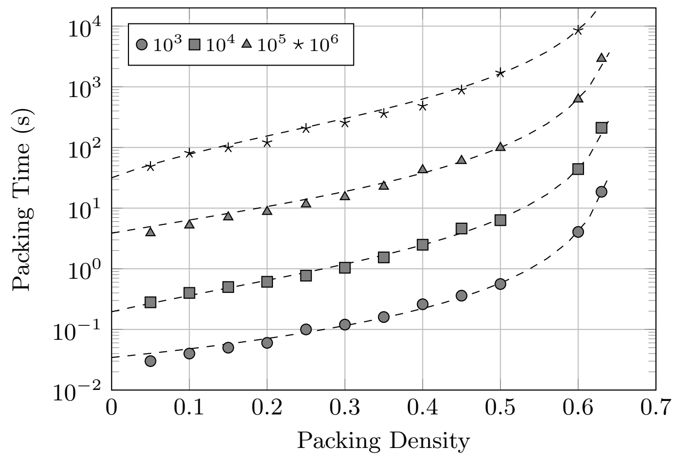

FIG. 9. Packing time of monodisperse spheres as a function of packing density using the LS packing algorithm.

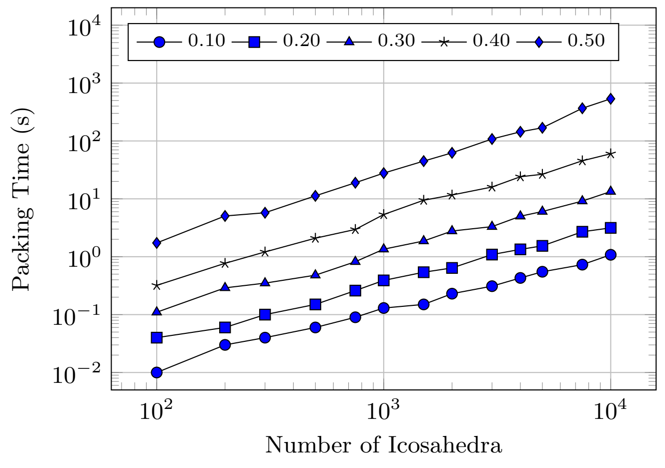

FIG. 7. Packing time of monodisperse icosahedra using the EDMC packing algorithm.

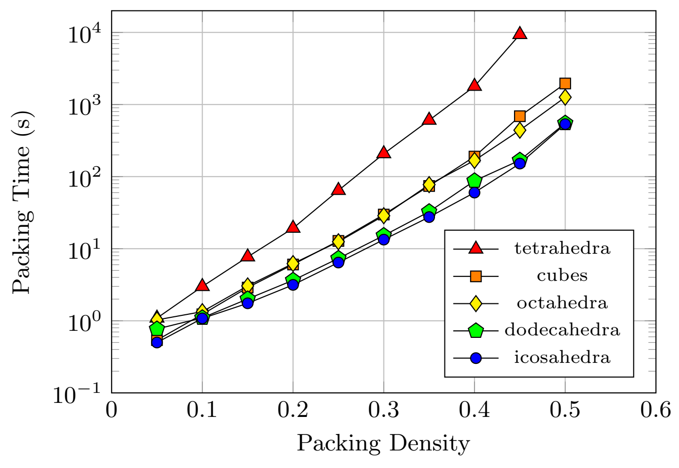

FIG. 10. Packing times as a function of packing density for 10^4 particles of each Platonic solid using EDMC algorithm.

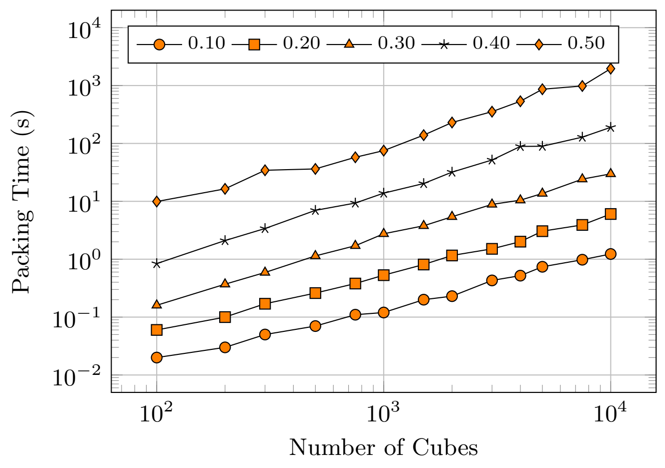

FIG. 8. Packing time of monodisperse cubes using the EDMC packing algorithm.

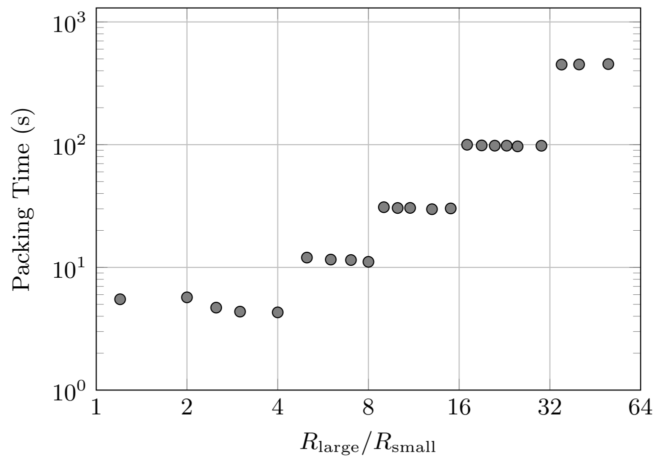

FIG. 11. Packing time of 10^4 spheres with uniform size distribution for different size ratios using the LS packing algorithm.

Fitting the data in figures 3–8 with an expression of the form $T(N) = AN^b$, where A and b are fitting parameters, and N is the number of particles, yields exponents b with values in the range $1.0 \leq b \leq 1.2$. Figure 9 shows the packing times of monodisperse packings of spheres generated with the LS packing algorithm. As the packing density increases, the packing process becomes slower due to the higher collision frequency between the particles. In fact, as the packing approaches jamming, the collision frequency diverges and the packing rate goes to zero. The packing time can be estimated with a function of the form

$$T(\phi) = \frac{A}{(\phi - \phi^*)^2} + B,$$

where A and B are constants proportional to the number of particles in the packing and represent the packing and initialization costs of the algorithm, respectively. The initialization cost is the time it takes to read input files, generate the particles, randomly place them inside the simulation domain, predict the earliest event for each particle, and initialize the event priority queue. It is usually negligible compared with the total cost of packing, but can become large in absolute terms (several seconds) for packings containing more than 10^5 particles. The quantity ϕ^* in the equation represents the jamming density at infinite running time, and is equal to 0.66 ± 0.02 for the runs shown in figure 9. In practice, however, since the code must stop at some finite time, the maximum densities of sphere packings generated with the LS algorithm are around $\phi_{\text{MRJ}} \approx 0.64$, the maximally random jamming density of spheres [54]. The particular values ϕ^* assumes depend on the packing algorithm and the growth rate of the particles being packed, as pointed out by Torquato in [54] and [55].

Other algorithms do not follow the expression for $T(\phi)$ given above; this is the case of the EDMC algorithm. The packing cost of polyhedra in the EDMC algorithm increases roughly exponentially with packing density until it diverges as the packing approaches jamming. Figure 10 shows the packing times as a function of density for 10^4 particles shaped as each of the Platonic solids. It is interesting to note that the cost of packing polyhedra is highly dependent on the shape of the particles. Polyhedra with large asphericity, such as tetrahedra, are more difficult to pack than roughly spherical polyhedra such as icosahedra. Packing tetrahedra to $\phi > 0.55$ requires tuning of the random displacement parameters to avoid suboptimal local jamming. Therefore, more complex polyhedra (with more vertices and edges) do not necessarily take longer to pack than simpler shapes.

Polydispersivity also has a large impact on the performance of both algorithms. This is because large particles need to be tested against a large number of smaller neighbors, and the construction of the list of neighbors that belong to adjacent cells is more expensive, since more cells need to be traversed when the hierarchical grid has additional levels. Since the code that handles spatial partitioning is shared between our extended LS algorithm and the EDMC algorithm, and the cost of computing intersections is lower for spheres than for polyhedra, we can get a clearer picture of the performance impact of polydispersivity alone by using the LS algorithm with packings of polydisperse spheres with a uniform size distribution, such that all levels of the hierarchical grid are evenly populated. Figure 11 shows the packing times to $\phi = 0.5$ of 10^4 spheres with uniform size distribution in the range $[a, b]$, with $a = 1$, and $b = 1.2, 2, \ldots, 50$. We observe that, except for the first extra level, there is a large performance impact when the size ratio is such that additional levels are needed. The uniform particle size distribution is a worst case scenario, but it demonstrates that our spatial partitioning scheme could be further optimized to allow faster simulations of highly polydisperse systems of particles, as a large portion of running time is spent traversing cells to look for neighbors.

The magnitude and number of attempted trial moves in the EDMC packing algorithm have a significant effect on performance, but they are difficult to optimize because the best parameters vary with particle shape and size distribution. We have used heuristics to determine reasonable default parameters for most simulations. Here, we present a brief discussion of the effects of changing these parameters to demonstrate their importance. We pack 10^3 icosahedra for a fixed amount of time, and show how the packing density evolves with running time when these parameters are changed. The magnitude of the displacements and number of trials both vary with the packing density during the simulation according to the formulas

$$S_n = S_{\max} \frac{(1-\phi)^3}{n} + \epsilon \qquad \text{and} \qquad N_{\text{trials}} = \frac{N_0}{(1.1-\phi)^2}.$$

We employ these functions to account for the reduced mobility of particles as the packing approaches jamming. The particular dependence on ϕ was determined *ad hoc*, and is not necessarily optimal in all cases. In essence, the displacements should become smaller and the number of trials increase as the packing density increases. In the formula for S_n, n is simply the current iteration number. Decreasing S_n with n increases the probability of a successful displacement that delays intersection in each iteration. To simplify the discussion, we only change $S_{\max}$ and N_0 between runs and compare the results. Figure 12 shows the effects of changing $S_{\max}$. If $S_{\max}$ is too large, each trial is more likely to fail, increasing the number of iterations necessary for an accepted move. This causes progress to be more random and unsteady, sometimes faster, but often slower than if $S_{\max}$ was smaller. If $S_{\max}$ is too small, although progress is more predictable and steady, each displacement only delays overlap for a small amount of time. Therefore, many more events are needed to bring packings to higher densities, which means longer running times. Each line in figure 12 shows a single run of the code. Due to the random nature of the EDMC algorithm, there is some variability in consecutive runs with similar parameters. In a similar manner, changing the number of trial moves also affects performance. If N_0 is too large, the size of the displacements becomes too small—thus, inefficient—as the number of iterations increases. On the other hand, if N_0 is too small, the trial displacement will fail to delay intersection more frequently, unnecessarily triggering all neighbors to be moved more frequently as well. The effects of changing N_0 are shown in figure 13. Interestingly, although the use of a smaller number of trials has a strong negative effect on performance during the first half of the simulation, the increased frequency of rearrangement of all neighbors seems to be beneficial as the packing becomes denser. If the displacements of neighboring particles is disabled, the simulation stalls before reaching $\phi = 0.5$. Therefore, a scheme to avoid local jamming is essential for simulations with tightly packed materials.

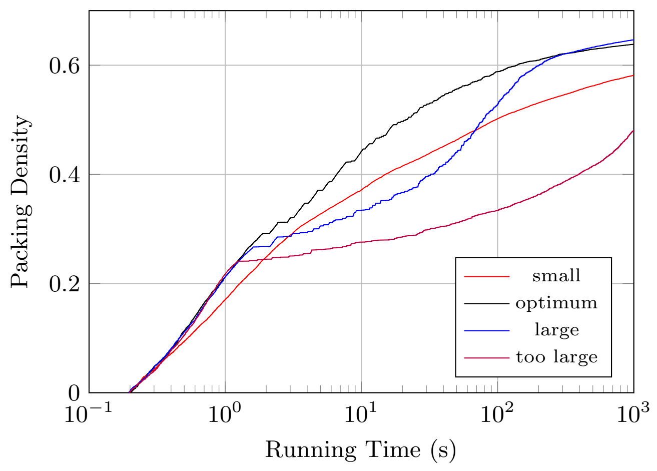

FIG. 12. Evolution of packing density with running time and different maximum displacement sizes.

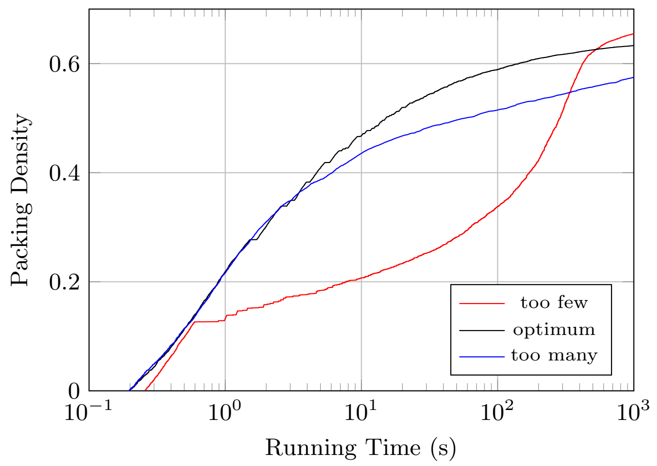

FIG. 13. Evolution of packing density with running time and different number of trials.

IV. APPLICATIONS OF THE PACKING ALGORITHM TO MATERIAL MODELING

In this section, we present a few examples of microstructure models generated with the EDMC packing algorithm. The models are constructed based on general information about the shape and size distribution of the particles. Each model is constructed inside a three-dimensional periodic cell.

A. Aluminized Rocket Propellant

The first model we present is for an aluminized solid propellant consisting of AP particles and fine aluminum powder. Coarse AP particles used in propellants generally have ellipsoidal shape due to milling and can be up to a few hundred microns across. Hence in this example we model AP particles using ellipsoids. We use a gaussian distribution of particle sizes with mean size $\mu = 100$ µm, and standard deviation $\sigma = 10$ µm. Fine aluminum powder particles are roughly spherical and usually no larger than a few tens of microns. Here, they are modeled using perfectly spherical particles with a gaussian size distribution. For the aluminum particles, the mean size is $\mu = 5$ µm, and standard deviation is $\sigma = 1$ µm. In this simplified model, all AP particles have the same ellipsoidal shape, with axes $a = 100$ µm, $b = 80$ µm, and $c = 60$ µm. If the shape distribution is known, it is also possible to create particles according to that distribution. Although the size distributions chosen for this model are close to what is typically found in aluminized propellants, they are not necessarily realistic. For instance, a propellant may contain different grades of AP particles that would need to be properly modeled to be realistic. Figure 14 shows the simulated packing. It contains 2×10^2 ellipsoidal AP particles and 4×10^5 spherical aluminum particles.

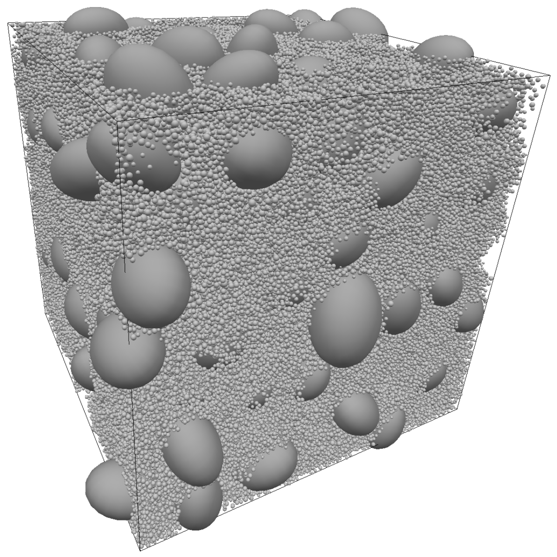

FIG. 14. Model of aluminized solid rocket propellant.

The final particle volume fraction is $\phi = 0.50$. It should be noted that the relative volume fraction of AP and aluminum, as well as the total particle volume fraction both depend on the particular type of propellant to be simulated. Figure 15 shows a cross section through the packing shown in figure 14.

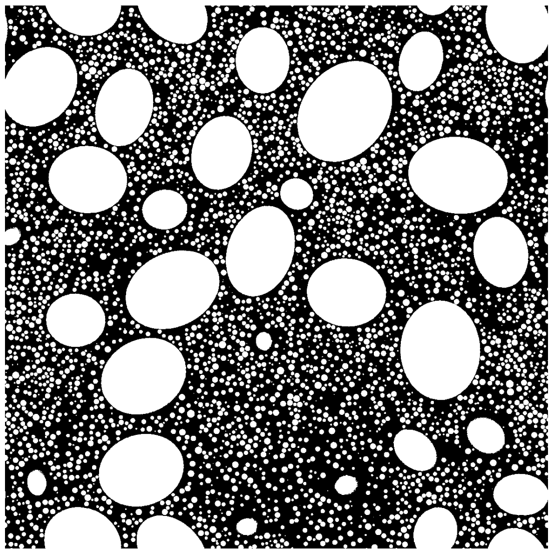
FIG. 15. Cross section of aluminized propellant model.

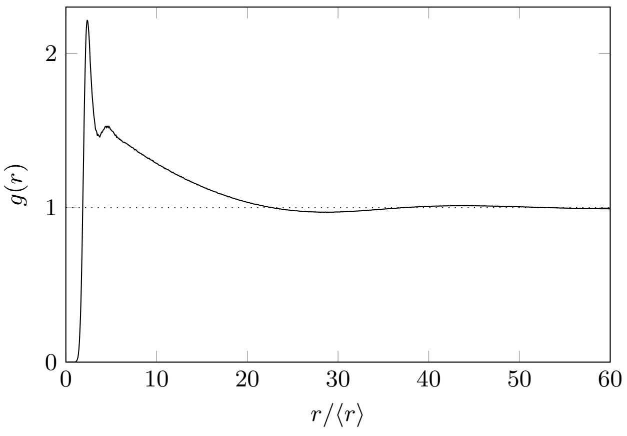

FIG. 16. Radial distribution function for aluminum particles.

Once a model of the material is constructed, useful statistical information can be extracted to be used in other models. Aluminum agglomeration models, for example, depend on the number of particles that fill the interstices between AP particles. The radial distribution function of the aluminum particles, shown in figure 16, can be used to produce estimates for the average size of these aluminum-rich pockets. These estimates usually rely on the position of the minimum in the radial distribution function, which in the case of our model is around 15 mean particle diameters, as seen in the figure.

B. Energetic Materials

Some energetic materials, such as solid rocket propellants and explosives, use crystalline oxidizer particles with shapes that are best represented by polyhedra. In [41], single crystals of the nitramines HMX, RDX, and PETN were used to create polyhedra with a typical crystal shape for each material. These polyhedra can be used to construct realistic models of propellants and explosive materials containing nitramines. Figure 17 is an example simulation using the polyhedron created for HMX particles. The packing density is $\phi = 0.61$. In this case, we have used 10^4 particles with a lognormal size distribution $p(x|\mu,\sigma) = \frac{1}{\sigma x \sqrt{2\pi}} \exp[-(\ln x - \mu)^2/2\sigma^2]$, where $\mu = 1.0$ and $\sigma = 0.2$. Figure 18 shows a cross section of the packing shown in figure 17.

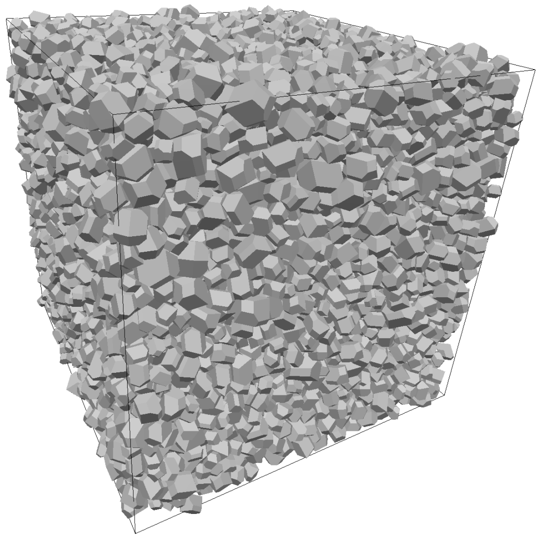
FIG. 17. Propellant model with polyhedral HMX particles.

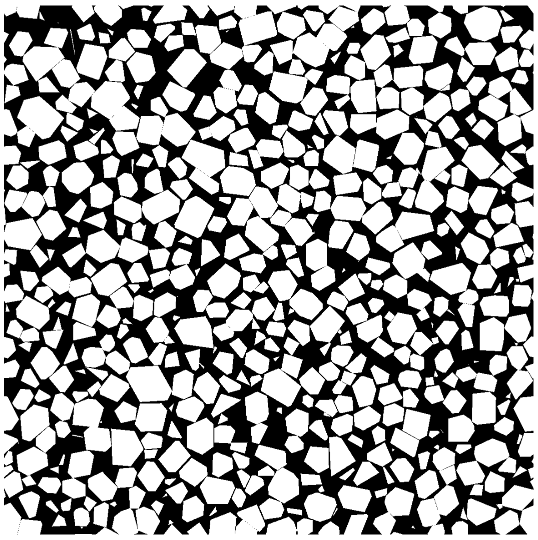
FIG. 18. Cross section of HMX propellant model.

C. Fiber-reinforced Material

Fiber-reinforced materials are commonly used in the aerospace, automotive, marine, and construction industries. Fibers usually increase the strength of flexible materials such as plastics to create strong but light materials. The extent to which the strength and elasticity of the composite are enhanced depends on the mechanical properties of the fibers and matrix materials, but also on the length and orientation of the fibers inside the matrix, and the volume fraction of fibers in the matrix. Figure 19 shows a simple model of a fiber-reinforced material in which fibers were modeled with square rods of aspect ratio equal to 10. The volume fraction of the fibers is $\phi = 0.10$.

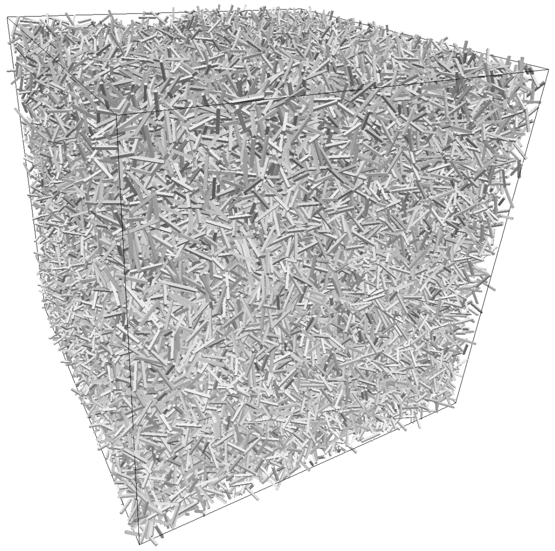
FIG. 19. Model of fiber-reinforced material.

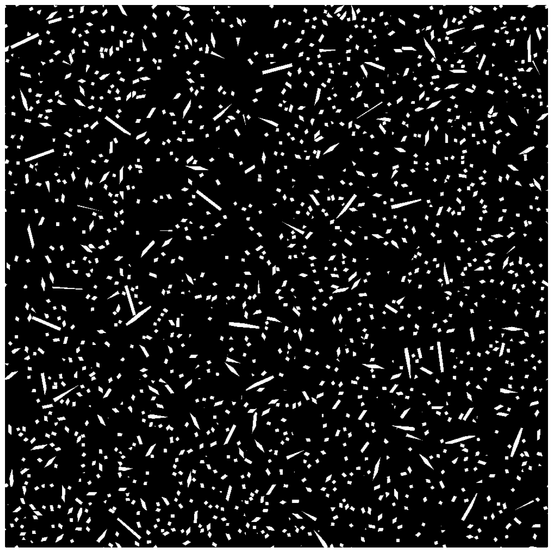
FIG. 20. Cross section of fiber-reinforced material.

D. Gravel, Cement, and Concrete

Many granular materials have complex microstructures with randomly shaped particles of various sizes, such as gravel, cement, and concrete. Concrete, in particular, is one of the oldest and most important composites ever invented. It has allowed ancient civilizations to build architectural masterpieces such as the Roman Pantheon—which remained as the largest dome in the world for over a thousand years!—and the Colosseum—perhaps the most iconic symbol of the Roman empire. Today, concrete is still the most widely used composite material. Therefore, it is also an important material to be able to model realistically, in order to allow us to study how the shape and size of the particles affect macroscopic material properties, hydration, etc. Concrete, however, is a difficult material to model, due to the random shapes and sizes of the particles that it comprises. Most concrete models found in the literature still employ spherical or ellipsoidal particles. Here, we attempt to create a more realistic model of concrete to showcase the capabilities of our particle packing code.

We generate random polyhedra for the particle shapes by computing the convex hulls of random point distributions using the software Qhull [56]. Each shape is generated from 50 random points distributed inside a unit cube. On average, each random polyhedron has about 17 vertices and 30 faces. We generate 1000 different polyhedra via this process, and use 100 particles of each type with a lognormal size distribution for a total of 10^5 particles in the simulation. The parameters for the lognormal size distribution are $\mu = 1.0$ and $\sigma = 0.35$, and the final packing density is $\phi = 0.60$. Figures 21 and 22 show the packing and a cross section. The statistical properties of the simulated packing are shown in figures 23 and 24, where L is the width of the domain.

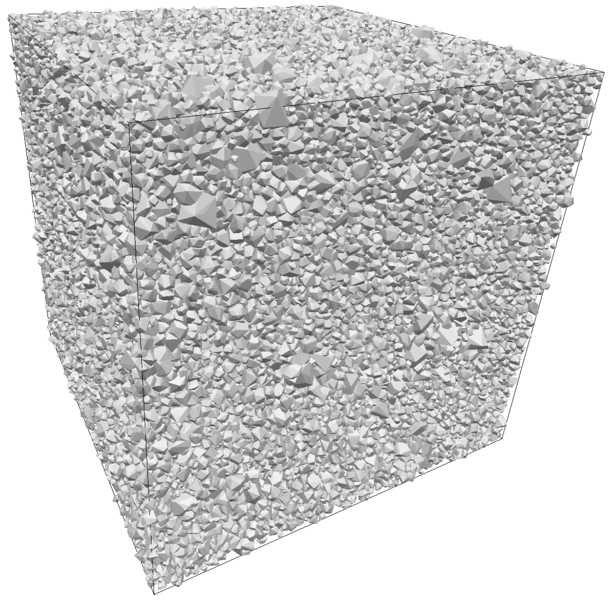
FIG. 21. Concrete model.

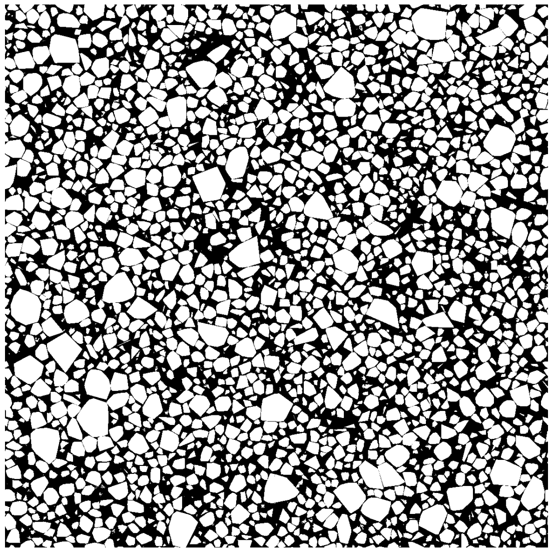

FIG. 22. Cross section of concrete model.

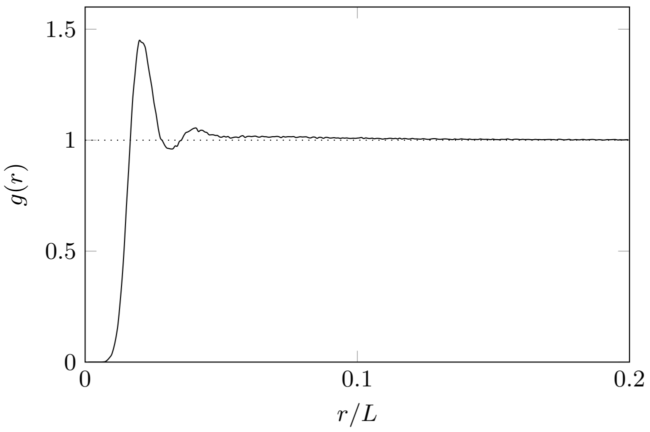

FIG. 23. Radial distribution function for concrete model.

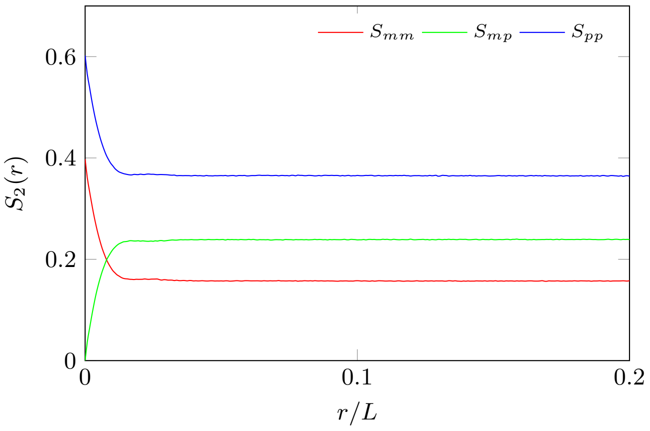

FIG. 24. Two-point statistical functions for concrete model; S_{mm} corresponds to the matrix–matrix, S_{mp} to particle–matrix, and S_{pp} to particle–particle functions, respectively.

V. DISCUSSION AND CONCLUSIONS

We have presented a particle packing algorithm for the modeling of complex microstructures of particulate heterogeneous materials. This algorithm allows realistic, large scale simulations of representative volume elements containing a large number of convex particles of various shapes and size distributions.

A performance analysis of the algorithm demonstrates that an efficient spatial partitioning scheme is essential for high performance. The suitability of partitioning schemes based on hash tables is reduced for highly polydisperse systems due to the large number of cells that need to be checked for neighbors. A sparse tree representation of the hierarchical grid may provide better performance in this case—something that should be investigated in the future.

We note that the shape of polyhedral particles has a stronger effect on performance than their complexity (number of vertices and faces). Moreover, the parameters that control the magnitude and number of trial displacements during collision resolution significantly affect performance as well.

Material models constructed with this algorithm can be used to determine statistical characteristics of materials—such as the n-point probability functions—that can in turn be used to compute their effective properties, or in direct numerical simulations.

VI. ACKNOWLEDGMENTS

This work was supported in part by the Defense Threat Reduction Agency, Basic Research Award No. HDTRA1-13-0010 to the University of Illinois at Urbana-Champaign and HDTRA1-14-0031 to the University of Florida; Dr. Suhithi Peiris, program manager. Guilherme Amadio was also supported by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) and Fulbright. Thomas L. Jackson was also supported in part by the U.S. Department of Energy, National Nuclear Security Administration, Advanced Simulation and Computing Program, as a Cooperative Agreement under the Predictive Science Academic Alliance Program, under Contract No. DE-NA0002378.

[1] J. C. Michel, H. Moulinec, and P. Suquet, Computer Methods in Applied Mechanics and Engineering **172**, 109 (1999).

[2] S. Torquato, International Journal of Solids and Structures **37**, 411 (2000).

[3] V. Kouznetsova, W. A. M. Brekelmans, and F. P. T. Baaijens, Computational Mechanics **27**, 37 (2001).

[4] F. Ballani, D. J. Daley, and D. Stoyan, Computational Materials Science **35**, 399 (2006).

[5] J. Segurado and J. LLorca, Mechanics of Materials **38**, 873 (2006).

[6] S. R. Annapragada, D. Sun, and S. V. Garimella, Computational Materials Science **40**, 255 (2007).

[7] S. Torquato, Annual Review of Materials Research **40**, 101 (2010), http://www.annualreviews.org/doi/pdf/10.1146/annurev-matsci-070909-104517.

[8] G. P. Sutton and O. Biblarz, *Rocket Propulsion Elements* (Wiley, 2010).

[9] L. Massa, T. L. Jackson, and J. Buckmaster, Journal of Propulsion and Power **21**, 914 (2005).

[10] T. L. Jackson, F. Najjar, and J. Buckmaster, Journal of Propulsion and Power **21**, 925 (2005).

[11] T. L. Jackson, AIAA Journal **50**, 993 (2012).

[12] Y. Yavor, A. Gany, and M. W. Beckstead, Propellants, Explosives, Pyrotechnics (2013), 10.1002/prep.201300073.

[13] S. Gallier, International Journal of Multiscale Computational Engineering **8** (2010).

[14] B. J. Alder and T. E. Wainwright, Journal of Chemical Physics **31**, 459 (1959).

[15] J. D. Bernal, Nature **185**, 68 (1960).

[16] J. D. Bernal, Elsevier, Amsterdam , 25 (1965).

[17] F. H. Stillinger, E. A. DiMarzio, and R. L. Kornegay, Journal of Chemical Physics **40**, 1564 (1964).

[18] B. D. Lubachevsky and F. H. Stillinger, Journal of Statistical Physics **60**, 561 (1990).

[19] R. J. Speedy, Journal of Chemical Physics **100**, 6684 (1994).

[20] W. B. Russel, D. A. Saville, and W. R. Schowalter, *Colloidal dispersions* (Cambridge University Press, 1992).

[21] C. W. Hong, Journal of the American Ceramic Society **80**, 2517 (1997).

[22] C. H. Bennet, Journal of Applied Physics **43**, 2727 (1972).

[23] H. A. Makse, D. L. Johnson, and L. M. Schwartz, Physical Review Letters **84**, 4160 (2000).

[24] T. Pöschel and T. Schwager, *Computational granular dynamics: models and algorithms* (Springer, 2005).

[25] L. V. Woodcock and C. A. Angell, Physical Review Letters **47**, 1129 (1981).

[26] D. Grolimund, M. Elimelech, M. Borkovec, K. Barmettler, R. Kretzschmar, and H. Sticher, Environmental Science and Technology **32**, 3562 (1998), http://pubs.acs.org/doi/pdf/10.1021/es980356z.

[27] S. Khirevich, *High-Performance Computing of Flow, Diffusion, and Hydrodynamic Dispersion in Random Sphere Packings*, Ph.D. thesis, Philipps-Universität Marburg, Germany (2010).

[28] H. S. Choi, H. C. Park, C. Huh, and S. Kang, Energy Procedia **4**, 3786 (2011).

[29] C. E. Shannon, Bell System Technical Journal **27**, 379 (1948).

[30] U. Betke and M. Henk, Computational Geometry **16**, 157 (2000).

[31] S. Torquato and Y. Jiao, Physical Review E **80**, 041104 (2009).

[32] S. Torquato and Y. Jiao, Nature **460**, 876 (2009).

[33] Y. Jiao and S. Torquato, Physical Review E **84**, 041309 (2011).

[34] E. R. Chen, Discrete Comput Geom **40**, 214 (2008).

[35] A. Haji-Akbari, M. Engel, A. S. Keys, X. Zheng, R. G. Petschek, P. Palffy-Muhoray, and S. C. Glotzer, Nature **462**, 773 (2009).

[36] Y. Kallus, V. Elser, and S. Gravel, Discrete & Computational Geometry **44**, 245 (2010).

[37] E. R. Chen, M. Engel, and S. C. Glotzer, arXiv:1001.0586 [cond-mat.stat-mech] (2010), arXiv:1001.0586.

[38] S. Torquato and Y. Jiao, Physical Review E **81**, 041310 (2010).

[39] S. Torquato and Y. Jiao, arXiv:0912.4210 [cond-mat.stat-mech] (2010), arXiv:0912.4210.

[40] D. S. Stafford and T. L. Jackson, Journal of Computational Physics **229**, 3295 (2010).

[41] T. L. Jackson, D. E. Hooks, and J. Buckmaster, Propellants, Explosives, Pyrotechnics **36**, 252 (2011).

[42] S. R. Buechler and S. M. Johnson, International Journal for Numerical Methods in Engineering **94**, 1 (2013).

[43] S. Torquato and Y. Jiao, Physical Review E **82**, 061302 (2010), arXiv:1008.2747.

[44] A. Donev, R. Connelly, F. H. Stillinger, and S. Torquato, Physical Review E **75**, 051304 (2007).

[45] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, Robotics and Automation, IEEE Journal of **4**, 193 (1988).

[46] E. G. Gilbert and C.-P. Foo, Robotics and Automation, IEEE Transactions on **6**, 53 (1990).

[47] X. Zhang, M. Lee, and Y. J. Kim, available at: http://graphics.ewha.ac.kr/FAST/ (2004).

[48] D. E. Knuth, *The Art of Computer Programming, Volume 3, Sorting and Searching* (Addison-Wesley, 1973).

[49] M. Marín and P. Cordero, Computer Physics Communications **92**, 214 (1995).

[50] G. Paul, Journal of Computational Physics **221**, 615 (2007).

[51] C. Ericson, *Real-Time Collision Detection* (Morgan Kaufmann Publishers, Elsevier, 2005).

[52] G. van den Bergen, *Collision Detection in Interactive 3D Environments* (Morgan Kaufmann Publishers, Elsevier, 2003).

[53] D. Eberly, *Game Physics* (Morgan Kaufmann Publishers, Elsevier, 2003).

[54] S. Torquato, T. M. Truskett, and P. G. Debenedetti, Physical Review Letters **84**, 2064 (2000).

[55] S. Torquato and F. H. Stillinger, Review of Modern Physics **82**, 2633 (2010).

[56] C. B. Barber, D. P. Dobkin, and H. T. Huhdanpaa, ACM Trans. on Mathematical Software **22**, 469 (1996), http://www.qhull.org.