Loose octree: a data structure for the simulation of polydisperse particle packings

S. Raschdorf, M. Kolonko
Clausthal University of Technology

January 8, 2009

Simulated particle packings deliver insight into many properties of granular matter. As many granular mixtures comprise a broad range of particle sizes, a large number of particles has to be included in the sample to reproduce the correct size proportions within the simulation. This is a great challenge not only for the simulation setup, but also for the internal data structures used to keep track of the particles' positions. This paper discusses grid and octree as examples of established spatial indexing techniques and identifies their difficulties with high polydispersity. A comparison shows that the lesser known loose octree is able to overcome these problems as it combines quick particle insertion with a good partitioning scheme.

1 Introduction

Simulating granular media is of great importance in many different fields of material science. It is applied e. g. in pharmaceutics, in powder metallurgy, concrete research or general material design. Often, the aim of the simulation is to obtain a random dense particle packing from a given grain size distribution. Such a packing may be used to analyse the space filling of the grains, the microscopic configuration under compaction or it serves as a starting configuration for the simulation of flow or stress behaviour.

In this paper we are going to discuss some data structure that allows an efficient simulation of particle packings on a computer. To derive the requirements for such a structure, we first have to look at the different algorithms for packing a given sample of particles into a container.

With the *discrete element method* (DEM) [1] a package can be simulated in the most realistic way keeping track of forces, frictions, accelerations and velocities between the

particles during the creation of the package. The computational effort limits this approach to packages with a relatively small number of particles (a few thousand on today's computers). This is sufficient for homogeneous particle mixtures where particles vary only little in size and other relevant properties.

If however, particles in a mixture differ very much from each other, the packing must contain a possibly huge number of particles in order to allow a representative analysis of the mixture. E. g., in a typical concrete mixture particle sizes vary from less than $0.1\,\mu$m up to $200\,\mu$m with even larger particles up to 32 mm if aggregates like gravel or sand are added. Representative samples from such mixtures are much larger than can be handled by today's simulation tools. In [14] a hierarchical approach is presented that subdivides the size range of the sample and allows to restrict the simulation to smaller subsamples. Still, the problem remains to simulate packings that contain several million particles.

Usually, the problem is simplified by 1.) replacing the particles by spheres and 2.) neglecting any property or interaction of the particles other than their geometry. Then the problem is to simulate a random close packing of (non-overlapping) spheres with diameters sampled from a given size distribution.

Algorithms for this problem mostly belong to two categories: the *random sequential addition* (RSA) or the *collective rearrangement* (CR) type.

RSA algorithms [2, 3, 4, 5] add spheres to the packing one by one. The final position of a sphere is found by dropping it onto the packing and rolling it until a stable position is reached [2], or by randomly testing different positions for the sphere and choosing the best one, e. g. the one with lowest potential energy [3]. The packing can either be built upwards from the bottom of a container to its top (see e. g. [5]), or grown as a cluster by adding spheres from all directions [4] or along a spiral on the outer boundaries of the container [6]. RSA algorithms are generally faster than the CR type algorithms described below, but are reported to create packings with a lower density [11]. Therefore, we are going to restrict ourselves to CR algorithms in the sequel.

CR algorithms are iterative algorithms that start with a random initial placement of all spheres in the container. The spheres are allowed to overlap, this may even be enforced by choosing a container that is too small for the spheres. In the following iterations, the spheres are visited sequentially and each one is moved by a small amount such that the overlap with its neighbours is reduced. In this way, the average overlap is reduced as far as possible and remaining overlaps are balanced throughout the packing. From time to time, the container to sphere size ratio is increased, either by growing the container (see e. g. [10]) or by shrinking the spheres (see e.g [11]). This allows to reduce the average overlap until a valid (i. e. more or less overlap-free) packing is obtained.

Data structures suited for CR algorithms are therefore faced with the following requirements:

Insertion insertion of a large number of sphere locations during initial placement,

Polydispersity the diameters of the spheres may vary across several orders of magnitude,

Relocation the spheres are constantly relocated though the changes are typically very small,

Contact search there must be an easy access to all neighbouring spheres that might overlap a given sphere.

The most time-consuming step in CR algorithms (as well as many other types of algorithms) is the identification of possibly overlapping neighbours of a sphere in a changing environment. Therefore, property **Contact search** is the most important requirement for the data structure.

Many sophisticated spatial partitioning and indexing techniques have been developed, see e. g. [15] for a recent comparison of some of them. It seems that most of these data structures and contact serach algorithms are optimized for objects that are of about equal size (e. g. [16]) or (like the dynamical grid structure of [23]) they work well only for systems in which the neighbourhood structure is updated simultaneously for all spheres at once per iteration. This is the case e. g. if an iteration models a time step in a DEM simulation [15]. In CR packing simulations we need a valid neighbourhood information after each relocation of a single sphere.

In this paper, we report some observations made in our experiments with different data structures without attempting to give an exhaustive overview here. In particular, we want to draw attention to the so-called loose octrees, which seem to be well suited for the CR simulation of polydispers particle mixtures. In our experiments, a hybrid data structure performed best with respect to run time. It consists of a loose octree for the global location information of all spheres and a local neighbourhood list for the contact search of each sphere. We compare these to grid-type structures and the rigid octree.

Let us state the packing problem more formally now. Assume that there are N spheres with diameters ranging from d_{min} to d_{max} that are to be packed into a cubic container with side length s_0. The aim is to obtain a random dense packing of the spheres, but during the CR algorithm spheres may overlap. Typically, the starting volume of the container will be much smaller than the final packing volume, it may even be as low as the net volume of the spheres.

In the next section we start with the local neighbourhood structure, the so-called Verlet lists. We then discuss different grid and tree structures in Section 3, in particular we describe the loose octree. In Section 5, a comparison of the performance of the different data structures for monodisperse and polydisperse packings is presented.

2 Local neighbour search with Verlet-lists

As mentioned in the introduction, a CR iteration sequentially visits each sphere and relocates it. Let us call the sphere presently visited the *active* sphere and name it σ. The algorithm checks the neighbours of σ for possible overlaps and then σ is moved away from its neighbours to reduce the overlap. If σ has no overlap, it is moved towards its nearest neighbour as the aim is a *dense* packing.

In our hybrid data structure we use local neighbour lists to perform the search for potential overlaps more efficiently. These lists serve as local data caches that have to be updated only if one of its members has been moved over a larger distance. In CR iterations movements are typically very small so that updates do not occur too often. Such neighbourhood lists were introduced in [28] (see also [29]) and are therefore also called *Verlet lists*. Although they can also be used alone (an example can be seen in the work of Nolan and Kavanagh [9]), they are widely accompanied by a global spatial index, e. g. a grid structure [30, 31, 32].

The Verlet list of a sphere σ contains all spheres that are located within a *Verlet distance* δ_V from σ. More precisely, let the center of σ be at the position c_σ and let r_σ denote the radius of σ. Then the Verlet list $L_V(\sigma)$ contains all spheres that intersect the *Verlet sphere* $V(\sigma)$ of radius $r_\sigma + \delta_v$ centered at c_σ, see Fig. 1a). The Verlet lists are created after the initial placement. $L_V(\sigma)$ also contains the location c_σ of (the center of) σ at the time the list was created or last updated.

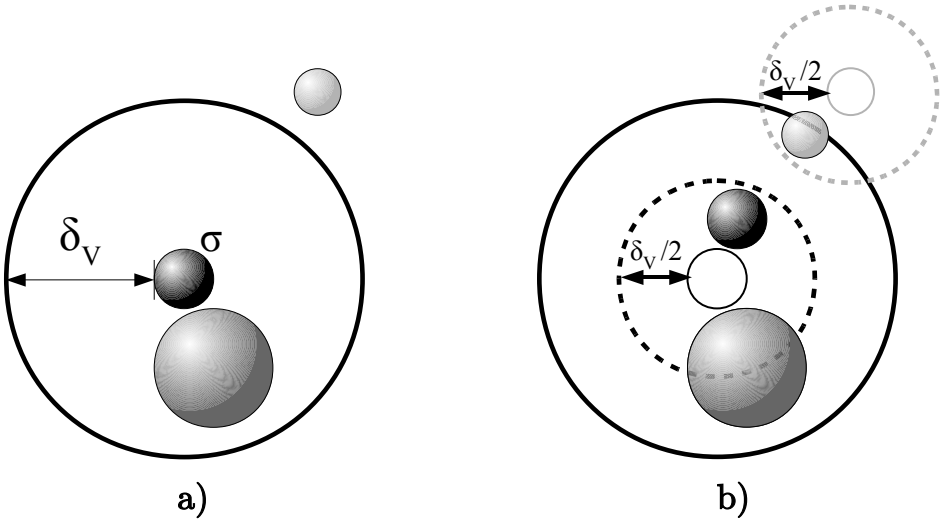

Figure 1: **a)** The black sphere in the center is the active sphere σ, the large gray sphere is in the Verlet list $L_V(\sigma)$ as it intersects the δ_V-zone around σ. The small light gray one is not in $L_V(\sigma)$. **b)** Although the spheres have moved, there is no need to update the neighbour list $L_V(\sigma)$ yet because the spheres' new positions differ less than $\frac{\delta_V}{2}$ from their initial positions.

An update of $L_V(\sigma)$ has to be carried out only if the total displacement of σ from its former location c_σ is more than $\delta_V/2$, see Figure 1b). For an update, the neighbours within distance δ_V from the new location of σ have to be found in the global data structure discussed below. If members from the old list are no longer neighbours of σ, then σ has to be deleted from their Verlet list, too, and if new neighbours appear in the updated list then σ must be appended to their Verlet lists to maintain the symmetry of the neighbourhood relation. This update rule guarantees that all possibly overlapping spheres of σ are in the present Verlet list of σ.

The gain in performance from using Verlet lists obviously depends on the size of

δ_V: if it is chosen too small compared to the sphere movements, the lists have to be updated very often, if it is large the lists contain more neighbours which have to be checked for overlap (the number of neighbours increases with δ_V^3). We cannot give an optimal value because it depends on the sphere movements, size distribution and initial overlaps. However, we recommend to slowly decrease δ_V during a CR simulation run as the movements of the spheres become smaller while the structure approaches the final packing.

3 Spatial data structures for the global positioning of polydisperse objects

Beside the local neighbourhood lists we need a global data structure that contains the spheres with their present position within the container, i. e. the absolute coordinates of their centers and their diameters. During the initial placement phase of CR, the spheres are inserted into this data structure sequentially. Changes in the location of a sphere during the overlap reduction phase must be recorded and the present neighbours of an active sphere σ have to be found quickly for the update of the Verlet lists.

3.1 Grid

Probably the most natural approach would be a grid structure that divides the container into cells of equal size. Each sphere σ is assigned to the cell $\nu(\sigma)$ that contains its center. This idea was introduced to molecular dynamics by Alder and Wainwright [19]. [20] accelerated the neighbour search by linking all objects in one cell into a 'linked-cell list', so that all objects in one cell could be accessed via the cell list. This became a standard method in DEM (see e. g. [21]), but is sometimes also used in CR simulations [22].

To obtain a grid in our case, the cubic container with side length s_0 is divided into m^3 equal cubic cells with side length $g := s_0/m$. Insertion of a sphere is particularly easy as the cell $\nu(\sigma)$ can be determined from the coordinates of the sphere center in $O(1)$, independently of the number of spheres.

If g resp. m is chosen such that

$$d_{max} \leq g, \tag{1}$$

where d_{max} is the diameter of the largest sphere, then all spheres will fit into the cells. More importantly, it is guaranteed that all neighbours that may overlap any sphere in a cell c_0 must be contained in one of the adjoining 26 cells (in 3-dimensional space), see Fig. 2 for a 2D example.

This performs well as long as the average number of spheres per cell with side length $g \approx d_{max}$ is small, which is the case for monodisperse systems or particles with only small deviations in diameter.

For highly polydisperse media, the reqirement $d_{max} \leq g$ implies large cells, each possibly containing a huge number of small spheres. Particle size distributions as they are typical e. g. in concrete technology consist of many small and only a few very large

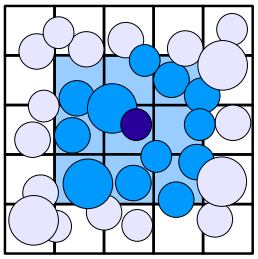

Figure 2: A 2-dimensional grid. Only the nine coloured cells can contain neighbours that overlap the dark sphere in the middle cell.

particles. Here, the grid may even degenerate into a single cell. Hence for polydisperse particle mixtures, the number of potentially overlapping spheres in the adjoining cells increases dramatically which makes a simple grid inefficient for this task. This is also the case for some enhancements of the grid as e. g. the algorithm named 'no binary search' from [16].

There are a number of workarounds for this situation. If there are only few very large particles, one may keep them seperately e. g. in a list and use the grid only for the mass of smaller and less polydisperse particles. This allows smaller cells and accelerates the neighbour search, however it complicates the data structure and requires more effort than the structures introduced below.

In an interesting approach in [23] the cell size may be chosen independently of the sphere sizes. The reference point for a spheres is the origin of the smallest enclosing cube (instead of their center) and it is assigned to the cell that contains this reference point. The algorithm sweeps through the cells starting from the origin of the container and going to the opposite corner. In each cell, all spheres with reference points in that cell are checked for neighbours in the cell itself and the adjoining cells that have not yet been visited. If one of these spheres also intersects another cell not yet visited, it will be checked with that cell again, it 'migrates' to that cell. In this way polydisperse mixtures can be checked with a limited number of cells to be searched. However, this approach is only useful if the neighbourhood of *all* spheres is checked as it must visit all cells in a specified order to be efficient. For the update of Verlet lists as described in Section 2, we only have to do local checks of some spheres.

An extension to the single grid data structure is the use of multiple grids of different sizes [24] that contain the spheres of appropriate size. During the search they have to switch between grids which makes the algorithm a little complicated if the mixture is highly polydisperse and a lot of grids have to be used. Actually, this structure resembles the hierarchical trees discussed next without having their simple recursive structure.

3.2 Octree

Search trees are widely used in computer graphics, geo information systems and similar tasks. Binary search trees for DEM are discussed in [**?**] and [15]. More efficient for the spatial indexing in three dimensions seem to be the so-called octrees [25, 26].

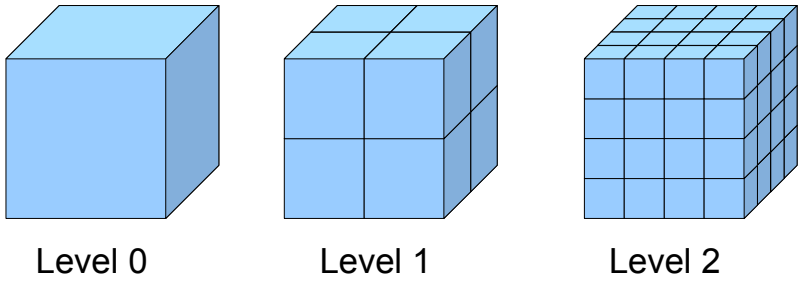

Figure 3: The first three levels of an octree.

Here, the (cubic) simulation domain (with side length s_0) is divided into eight octants with side length $s_1 := s_0/2$, called *nodes* on *level* 1. Each node is then recursively subdivided into eight octants so that on level $l \geq 0$, there are 8^l nodes (cubic cells) with side length $s_l := s_0/2^l$, see Figure 3. Each sphere is assigned to the smallest node into which it fits. If d_{min} denotes the smallest diameter of a sphere in our sample, then we need an octree with l_{max} levels where

$$s_{l_{max}} \geq d_{min} \quad \text{i. e.} \quad \frac{s_0}{2^{l_{max}}} \geq d_{min} \quad \text{or} \quad l_{max} = \left\lfloor \log_2 \frac{s_0}{d_{min}} \right\rfloor . \tag{2}$$

The single node at level $l = 0$ is called root node and contains the whole simulation domain. An octree can be regarded as a hierarchical grid, at each level providing an adequate grid size for a different class of sphere sizes.

Insertion of a sphere into this spatial index is done by traversing the tree: Beginning with the root node it has to be checked whether the sphere fits completely into one of its eight child nodes without intersecting another node. If this is true, the sphere descends in the tree to the corresponding node where this procedure continues, otherwise it stays in the present (parent) node. This way all spheres are placed as deep into the tree as possible, this needs $O(l_{max})$ steps per sphere in the worst case.

Note that a lot of the nodes at the deepest tree level may remain empty if the associated space is occupied by larger spheres which reside in nodes at higher levels. In a less memory-consuming variant, the tree nodes are only built on demand, i. e. when a sphere would fit into a node's child, this child node is created and the sphere descends. Nodes will also get deleted when their last sphere moves away and they contain no child nodes. However, the dynamic memory allocation and freeing is slower than a static tree.

In contrast to grid cells, the octree nodes enclose their spheres completely, so none of the adjacent nodes of the same tree level have to be searched for overlapping spheres. All possibly overlapping neighbours of a given sphere σ in a node $\nu(\sigma)$ on level l reside in a subset of the following nodes: the node $\nu(\sigma)$ itself, all of its parent nodes (from one level $l-1$ up the root node) and of all of its child nodes on levels $l+1, l+2, \ldots, l_{max}$, see Figure 4. Thus the maximum number of nodes $n_\nu(l)$ whose spheres have to be included

as neighbours for a sphere situated at level l is

$$n_\nu(l) = l + \sum_{i=0}^{l_{max}-l} 8^i, \quad l = 0, \ldots, l_{max}. \tag{3}$$

In particular, the spheres residing in the root node are always included as neighbours of each sphere.

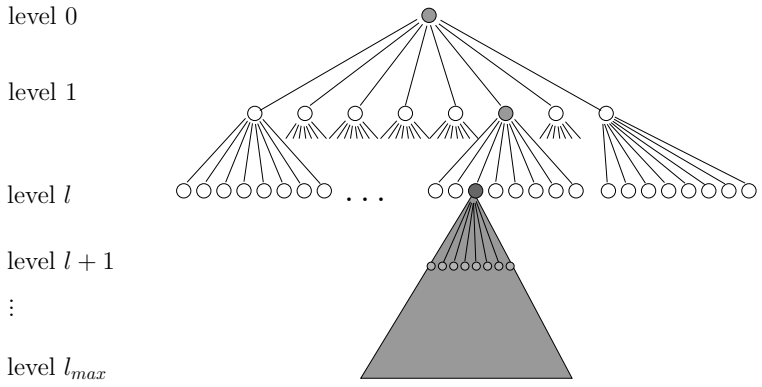

Figure 4: For the Verlet list of a sphere in the black node on level l, only the grey nodes have to be searched.

It is possible to reduce the number of nodes to be searched if intersection checks between σ and the nodes at levels $l+1, \ldots, l_{max}$ are carried out. Whenever there is no intersection between σ and a node ν^*, all further child nodes of ν^* do not have to be searched as they cannot overlap σ either.

A clear disadvantage of octrees is the following: when a sphere is assigned to a node, the level of the node depends not only on the diameter of the sphere, but also on its position. E. g. for a container centered at the origin, all spheres intersecting the coordinate planes will be placed into the root node no matter how large they are because they do not entirely fit into one of the child nodes. The same holds for all spheres that intersect one of the dividing planes of deeper levels: they cannot be placed as deep into the tree as their diameters allow because they would then intersect more than one node. This results in a kind of 'hotspot' pattern along the dividing planes made visible in Fig. 5.

Though this weakens the overall performance of the octree during contact search, the hierarchical approach of the octree suits polydisperse particles quite well as it contains cells of different size. In the next Section we shall relax the requirement that a sphere must fit exactly into one node and obtain a data structure tailored to our needs.

3.3 Loose Octree

The loose octree is an octree variant introduced by Ulrich [33] in order to overcome the 'hotspots' of octrees described above. So far, its main field of application seems to be

[1]Picture generated with a modified version of QuteMol [27].

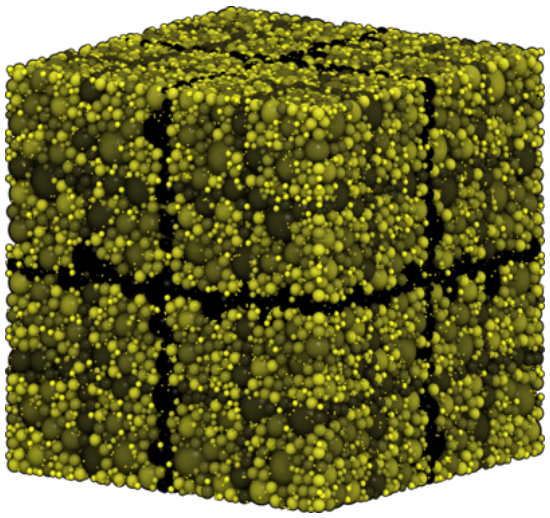

Figure 5: Dense packing of 50 000 spheres[1]. Sphere colors indicate the octree node level: the darker the color, the higher the level. The black spheres intersect the splitting planes and thus reside in the root node.

computer graphics and in particular game programming. The strucure is slightly more complicated than in an ordinary 'strict' octree, but its advantages are more balanced nodes and the possibility to calculate the target cell of a sphere directly without having to traverse the tree.

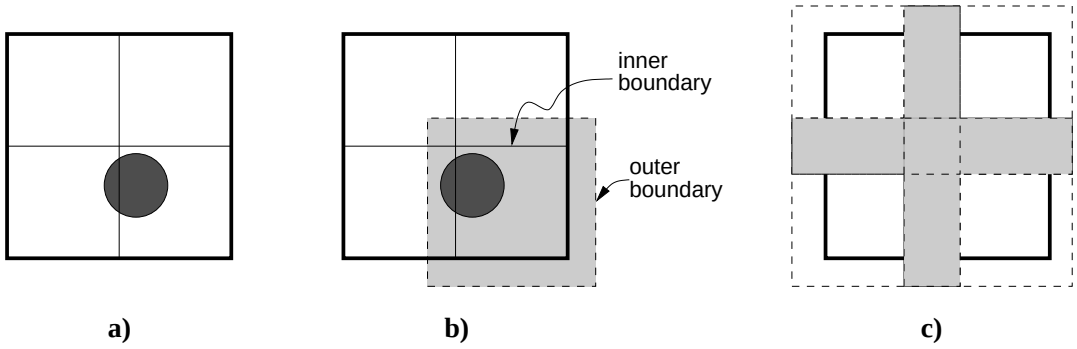

Figure 6: Sphere placement in a two-dimensional equivalent to octree and loose octree: a) The sphere intersects the splitting planes of the octree and must remain on the upper level. b) In the loose octree the sphere fits into the outer boundary, and may therefor go into lower right child node. c) The outer boundaries of the nodes overlap each other and nodes of other parents.

Different to ordinary octrees a node in the loose octree represents a larger part of the container. A node is defined by an *inner boundary* of side length $s^{(in)}$ and an *outer boundary* of length $s^{(out)}$. While $s^{(in)}$ corresponds to the node size s as in an ordinary octrees, the outer length $s^{(out)} := k \cdot s^{(in)}$ causes the nodes at a given octree level to overlap each other for a factor $k > 1$. Now each sphere is assigned to the deepest node

that contains its center within the inner boundary while the sphere fits entirely into the node's outer boundary, see Fig. 6. This node expansion causes the spheres to be placed much deeper into the tree as can bee seen in Fig. 7 when compared to the octree case in Fig. 5.

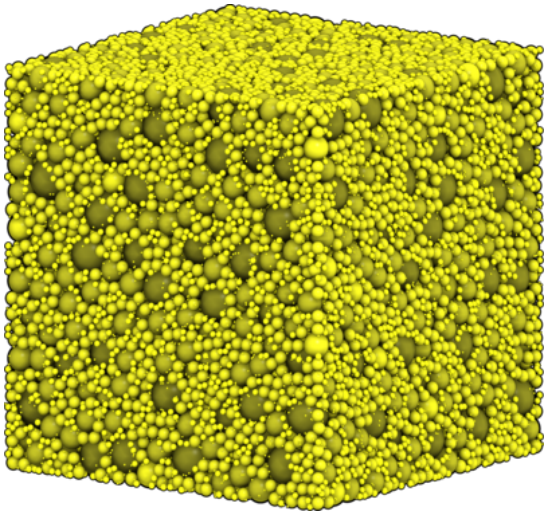

Figure 7: The dense packing of 50 000 spheres[2] from Fig. 5. The same color range is used for the loose octree node level, hence spheres placed deeper into the tree appear brighter.

An appropriately chosen node expansion factor k is of great importance. To avoid the 'hotspots' along the splitting planes large values of k are preferrable. On the other hand, large values of k cause large overlap between nodes of one level. This in turn leads to longer Verlet lists of neighbours: the neighbourhood list $L_V(\sigma)$ of a sphere σ in a node ν on level l contains the spheres of all nodes that have outer boundaries overlapping the outer boundary of ν. Besides the predecessor and successor nodes as in the strict octree these may include all sibling nodes of ν as well as nodes in other subtrees. These nodes have to be determined by a traversal of the tree. Though this seems more effort compared to the strict octree it is made up for by the fact that the average level of spheres is much deeper in the loose octree than in the strict one. Therefore the larger part of overlapping nodes are quite small containing less spheres, see alo experimental results in Section 5.

As already stated by Ulrich [33], $k = 2$ is a value that performs very well. In particular, it allows to insert a sphere into the tree without having to traverse it: a sphere σ of diameter d necessarily fits into the nodes of level l with inner boundary length $d \leq s_l^{(in)} := s_0/2^l$, hence the deepest such level is

$$l(d) := \left\lfloor \log_2 \frac{s_0}{d} \right\rfloor \tag{4}$$

[2]Picture generated with a modified version of QuteMol [27].

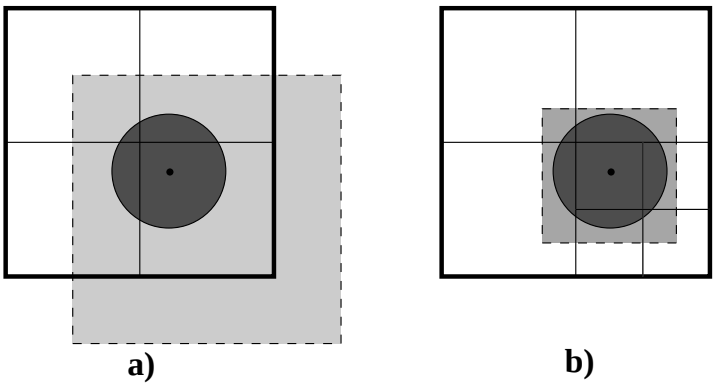

Figure 8: a) With $k = 2$ a sphere with diameter $d \leq s_{in}$ always fits inside the node's outer boundary. b) It may fit into one of its child nodes.

where s_0 is again the side length of the whole simulation domain. Note that the sphere σ can possibly be placed a level deeper into the tree if one of the children of ν entirely contains σ within its outer boundary, see Fig. 8 b).

A complete insertion strategy for a sphere σ of diameter d would therefore be:

1. Calculate the node level $l = l(d)$ with Eq. 3.
2. Choose the node ν on level l that contains the sphere's center. Note that the inner boundaries of the nodes on one level form a grid such that ν can be determined in $O(1)$.
3. Check the one child node of ν that contains the sphere's center within its inner boundary. If the sphere σ happens to fit completely into its outer boundary, store σ there.
4. Else assign σ to ν.

Because of step 3., we need one more level in the loose octree than in the strict octree:

$$\tilde{l}_{max} = \left\lfloor \log_2 \frac{s_0}{d_{min}} \right\rfloor + 1 \tag{5}$$

Remember that the above steps are used not only during the initial insertion of the spheres into the tree but also after each update of a sphere's position during the rearrangement, which makes this fast insertion strategy much more valuable than the one in the 'strict' octree case. The additional effort in step 3. speeds up later neighbour search as a sphere placed one level higher than necessary always adds to the neighbours of all spheres in its node's descendants.

To update the Verlet list $L_V(\sigma)$ of a sphere σ we simply check for intersection between the virtual Verlet sphere $V(\sigma)$ (as described in Section 2) and the nodes of the loose octree, beginning with the root node. If a node is fully located inside $V(\sigma)$, all spheres of all further child nodes in this subtree have to be included. If, on the other hand,

there is no intersection, then all of these child nodes can be skipped. Only at a partial intersection the check has to be carried out for the child nodes, too.

The distribution of spheres among the tree levels will correspond to the particle size distribution: spheres with diameters in the interval $(s_0/2^{l+1}, s_0/2^l]$ reside in the loose octree levels l or $l+1$, so a size distribution typical e. g. for concrete would result in heavily populated deeper levels and much fewer (larger) spheres on the upper levels. This adapts well to the tree structure which provides more nodes in the deeper levels.

4 Comparison

To examine the performance of the different data structures, in particular of the variants of the octree, samples of 10 000 spheres were packed by our collective rearrangement algorithm.

The first experiments used a monodisperse packing with a sphere diameter of $d = 1$. The second sphere mixture was sampled from a *Fuller curve* [35] which is defined by

$$p_i = \left(\frac{d_i}{d_{max}}\right)^{0.5}, \quad i = 1, \ldots, m \tag{6}$$

where the d_i are m different diameters with $d_1 < d_2 < \cdots < d_m$ and p_i is the share (by mass or volume) of spheres with $d \leq d_i$. The Fuller curve is a classical size distribution that results in a high space filling of the packing. For the simulation the parameters 684.75 and $m = 70$ were chosen; however, in the sample of limited size (10 000 spheres) only part of the distribution could be realized, so the diameters ranged from $d_{min} = 1$ to $d_{max} = 52.7$.

The histograms in Fig. 9 show how the spheres are distributed among the levels of the strict octree and the loose variant. The results are averages from the trees for the final packings. In the octree the spheres are widely distributed among the levels of the tree in the monodispers as well as in the polydispers case. If a loose octree is used, the spheres reside mainly within the last few levels of the loose octree. In the monodisperse packing there are actually no spheres on levels 0–3 of the loose octree. As the spheres all have diameter 1, they are inserted into the two deepest tree levels only. In the polydisperse case, however, the outer boundary length $s_7^{(out)}$ of the nodes at level 7 was only slightly larger than the smallest diameter d_{min}, hence only very few spheres slip into the deepest loose octree node level 8 (cp. step 3. above).

As was noted above, octrees have a definite advantage over their loose counterpart: Nodes of an ordinary octree cover exactly the same region as their descendants, while loose octree nodes overlap their siblings of the same level, hence most space within each level is covered by more than one node. This should increases the number of nodes possibly intersecting a sphere σ and hence increase the length of the Verlet neighbour list.

The following results show however, that the advantages of the loose octree prevail: as most of the spheres are on deeper levels in the loose octree, the neighbourhood search in the (small) overlapping nodes is much more precise resulting in shorter Verlet lists.

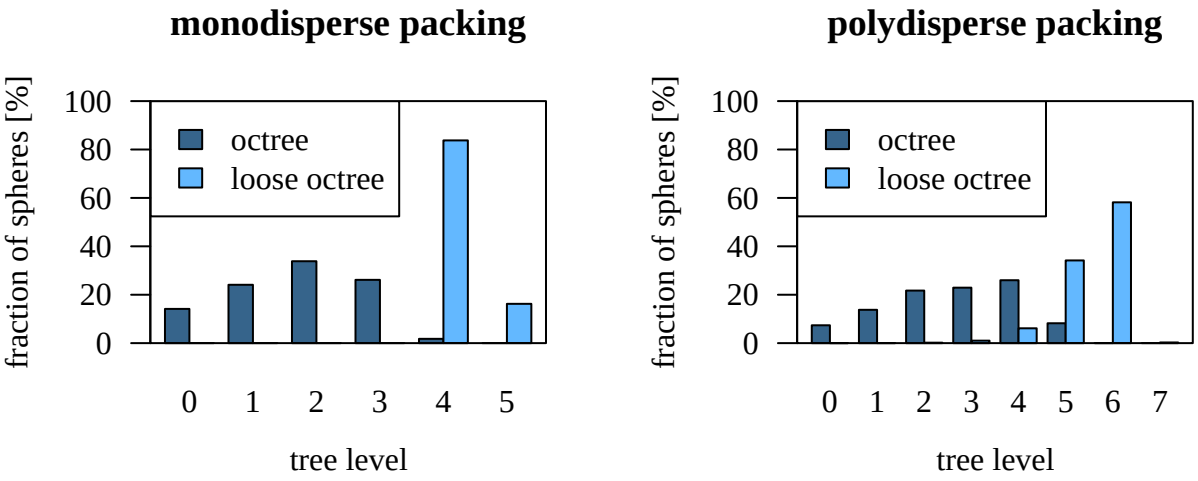

Figure 9: Distribution of spheres among octree and loose octree levels in the final packings, averaged over 10 simulation runs.

Table 1 gives the average number of neighbours per sphere for both the monodisperse and the polydisperse packing for all three global data structures considered in this paper: grid, strict octree and loose octree. While the grid demonstrates its strength at monodisperse packings, it degenerates into a single cell for the polydisperse mixture since the cell size is determined by the largest sphere (see Eq. **??**). Both hierarchical tree structures perform better in the polydisperse than in the monodisperse case, with about 40 % less neighbours in the polydisperse case. Comparing the loose octree to the ordinary octree, the latter delivers about 40 times more neighbours than its loose counterpart.

Considering the fact that the possible overlap with each of its neighbours has to be calculated for every single sphere during each iteration step of the CR algorithm, the loose octree is the only appropriate choice among these three spatial indexing techniques

method	av. no. of neighbours	
	monodisperse	polydisperse
grid	32.9	9 999.0
octree	2 027.6	1 188.1
loose octree	51.7	31.2

Table 1: The average number of neighbours (i. e. overlapping candidates) per sphere for a completed packing of 10 000 spheres, averaged over 10 simulation runs.

5 Conclusion

The loose octree was invented as data structure for the spatial partitioning and indexing in computer graphics. We have shown that it is also works remarkably well for the

simulation of polydisperse granular media. It combines the flexible node size of the hierarchical octree structure with the fast access of grid structures. It allows to insert spheres into the tree in constant time without the need for traversing it. The tree levels are filled in accordance with the particle size distribution which is a great advantage for polydisperse mixtures with many small particles.

Best results for the simulation of polydisperse particle packings with a collective rearrangement algorithm were obtained with a hybrid data structure. It uses the loose octree as a global data base for the spheres supported by Verlet neighbourhood lists as local cache for a fast detection of overlaps.

Acknowledgment

One of the authors (S. R.) acknowledges financial support by the Dyckerhoff-Foundation, project no. T218/15631/2006.

References

[1] P. A. Cundall and O. D. L. Strack. A discrete numerical model for granular assemblies. *Géotechnique*, 29(1):47–65, 1979.

[2] William M. Visscher and M. Bolsterli. Random Packing of Equal and Unequal Spheres in Two and Three Dimensions. *Nature*, 239:504–507, October 1972.

[3] D. J. Adams and A. J. Matheson. Computation of Dense Random Packings of Hard Spheres. *The Journal of Chemical Physics*, 56(5):1989–1994, 1972.

[4] Charles H. Bennett. Serially Deposited Amorphous Aggregates of Hard Spheres. *Journal of Applied Physics*, 43(6):2727–2734, 1972.

[5] Remi Jullien and Paul Meakin. Computer simulations of steepest descent ballistic deposition. *Colloids and Surfaces A: Physicochemical and Engineering Aspects*, 165(1–3):405–422, May 2000.

[6] Katalin Bagi. An algorithm to generate random dense arrangements for discrete element simulations of granular assemblies. *Granular Matter*, 7(1):31–43, April 2005.

[7] J. Mościński, M. Bargieł, Z. A. Rycerz, and P. W. M. Jacobs. The Force-Biased Algorithm for the Irregular Close Packing of Equal Hard Spheres. *Molecular Simulation*, 3(4):201–212, 1989.

[8] Boris D. Lubachevsky and Frank H. Stillinger. Geometric properties of random disk packings. *Journal of Statistical Physics*, 60(5):561–583, September 1990.

[9] G. T. Nolan and P. E. Kavanagh. Computer simulation of random packing of hard spheres. *Powder Technology*, 72(2):149–155, 1992.

[10] D. He, N. N. Ekere, and L. Cai. Computer simulation of random packing of unequal particles. *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics*, 60(6):7098–7104, December 1999.

[11] A. Bezrukov, M. Bargieł, and Dietrich Stoyan. Statistical Analysis of Simulated Random Packings of Spheres. *Particle & Particle Systems Characterization*, 19(2):111–118, 2002.

[12] G. Georgalli and M. Reuter. A particle packing algorithm for packed beds with size distribution. *Granular Matter*, 10(4):257–262, June 2008.

[13] Hiroshi Mio, Atsuko Shimosaka, Yoshiyuki Shirakawa, and Jusuke Hidaka. Optimum Cell Size for Contact Detection in the Algorithm of the Discrete Element Method. *Journal of Chemical Engineering of Japan*, 38(12):969–975, 2005.

[14] Michael Kolonko, Steffen Raschdorf, and Dominic Wäsch. A Hierarchical Approach to Estimate the Space Filling of Particle Mixtures with Broad Size Distributions. submitted to Powder Technology, 2009.

[15] K. Han, Y.T. Feng, and D.R.J. Owen. Performance comparisons of tree-based and cell-based contact detection algorithms. *Engineering Computations*, 24(2):165–181, 2007.

[16] A. Munjiza and K. R. F. Andrews. NBS contact detection algorithm for bodies of similar size. *International Journal for Numerical Methods in Engineering*, 43(1):131–149, 1998.

[17] Mihaly Mezei. A Near-neighbour Algorithm for Metropolis Monte Carlo Simulations. *Molecular Simulation*, 1(3):169–171, 1988.

[18] Y. T. Feng, K. Han, and D. R. J. Owen. Asynchronous/Multiple Time Integrators for Multifracturing Solids and Discrete Systems. In E. Oñate and D. R. J. Owen, editors, *VIII International Conference on Computational Plasticity*, 2005.

[19] B. J. Alder and T. E. Wainwright. Studies in Molecular Dynamics. I. General Method. *The Journal of Chemical Physics*, 31(2):459–466, August 1959.

[20] R. W. Hockney, S. P. Goel, and J. W. Eastwood. Quiet high-resolution computer models of a plasma. *Journal of Computational Physics*, 14(2):148–158, 1974.

[21] Simo Siiriä and Jouko Yliruusi. Particle packing simulations based on Newtonian mechanics. *Powder Technology*, 174(3):82–92, 2007.

[22] A. Yang, C. T. Miller, and L. D. Turcoliver. Simulation of correlated and uncorrelated packing of random size spheres. *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics*, 53(2):1516–1524, February 1996.

[23] John R. Williams, Eric Perkins, and Ben Cook. A contact algorithm for partitioning N arbitrary sized objects. *Engineering Computations*, 21(2/3/4):235–258, 2004.

[24] Hiroshi Mio, Atsuko Shimosaka, Yoshiyuki Shirakawa, and Jusuke Hidaka. Optimum Cell Condition for Contact Detection Having a Large Particle Size Ratio in the Discrete Element Method. *Journal of Chemical Engineering of Japan*, 39(4):409–416, 2006.

[25] Chris L. Jackins and Steven L. Tanimoto. Oct-trees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing*, 14(3):249–270, 1980.

[26] Donald Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982.

[27] Marco Tarini, Paolo Cignoni, and Claudio Montani. Ambient Occlusion and Edge Cueing for Enhancing Real Time Molecular Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1237–1244, 2006.

[28] Loup Verlet. Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Physical Review*, 159(1):98, July 1967.

[29] M.P. Allen and D.J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, USA, 1989.

[30] Gary S. Grest, Burkhard Dunweg, and Kurt Kremer. Vectorized link cell Fortran code for molecular dynamics simulations for a large number of particles. *Computer Physics Communications*, 55(3):269–285, October 1989.

[31] Zhenhua Yao, Jian-Sheng Wang, Gui-Rong Liu, and Min Cheng. Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method. *Computer Physics Communications*, 161(1-2):27–35, August 2004.

[32] G. Sutmann and V. Stegailov. Optimization of neighbor list techniques in liquid matter simulations. *Journal of Molecular Liquids*, 125(2-3):197–203, April 2006.

[33] Thatcher Ulrich. Loose Octrees. In Mark DeLoura, editor, *Game Programming Gems*, volume 1, chapter 4.11, pages 444–453. Charles River Media, 2000.

[34] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[35] W. B. Fuller and S. E. Thompson. The laws of proportioning concrete. *Trans. Am. Soc. Civ. Eng.*, 59:67–172, 1907.