Fast Sphere Packings with Adaptive Grids on the GPU

Jörn Teuber [†], René Weller [*], Gabriel Zachmann [*], Stefan Guthe[†]

[*] University of Bremen, Germany [†] Clausthal University, Germany

Abstract: Polydisperse sphere packings are a new and very promising data representation for several fundamental problems in computer graphics and VR such as collision detection and deformable object simulation. In this paper we present acceleration techniques to compute such sphere packings for arbitrary 3D objects efficiently on the GPU. To do that, we apply different refinement methods for adaptive grids. Our results show a significant speed-up compared to existing approaches.

Keywords: Sphere Packings, Collision Detection, GPU Programming, Acceleration Data Structures

1 Introduction

Sphere packings have diverse applications in a wide spectrum of scientific and engineering disciplines: for example in automated radiosurgical treatment planning, investigation of processes such as sedimentation, compaction and sintering, in powder metallurgy for three-dimensional laser cutting, in cutting different natural crystals, and so forth [HM09].

In contrast, in the field of computer graphics and VR sphere packings have been hardly used for a long time. This has two main reasons: first, computer graphics usually concentrates on the visual parts of the scene, i. e. the surface of the objects. Second, computing sphere packings for arbitrary 3D objects is a highly complex task. Almost all algorithms that are designed to compute sphere packings are computationally very expensive and therefore, they are restricted to very simple geometric objects like cubes or cylinders.

On the other hand, efficient volumetric object representations, such as sphere packings, also have their advantages. For instance, [WZ09] proposed a data structure, called *Inner Sphere Trees*, that is able to perform collision detection at haptic rates for complex objects. Moreover, the sphere packings enabled the authors to achieve an approximation of the penetration volume between a pair of objects efficiently. This is known to be the best measure for contact information that collision detection algorithms can provide. [YMWZ12] shows an application of sphere packings to volume-preserving simulation of deformable objects.

Both examples rely on an algorithm that generates polydisperse, space-filling sphere-packings [WZ10]. Polydisperse means that the radii of the spheres can be an arbitrary real number. This basic algorithm, called *Protosphere*, is inspired by machine-learning techniques and uses a prototype-based greedy choice to extend the idea of Apollonian sphere

packings to arbitrary geometric objects. *Protosphere* uses a uniform grid for two purposes: first, the acceleration of distance computations between the geometric primitives and the prototypes, and second, to guarantee a uniform distribution of the prototypes. However, a static uniform grid has also some drawbacks: e.g., it has a huge memory footprint and it produces a lot of unnecessary computations, especially during the first iterations.

In this paper, we extend the original *Protosphere* algorithm by an adaptive grid refinement to overcome these limitations. To do that, we propose two types of hierarchically refined grids using an implicit as well as an explicit refinement technique: the *implicit grid refinement* allows a faster creation of spheres during in the early phase of the sphere computation, i.e. in sparsely filled objects, while the *explicit grid refinement* accelerates the computation if the object is already densely packed with spheres. We also present a combination both techniques, the *hybrid grid*, that avoids the individual weaknesses of the respective methods while keeping their strengths. Finally, we outline a simple method to reduce the memory footprint of the overall algorithm.

We have implemented all our algorithms in a massively parallel version running completely on the GPU using NVIDIAs CUDA. The improvements described above allow us to improve the performance of the original *Protosphere* algorithm significantly. More precisely, we outperformed the original implementation by more than an order of magnitude, especially on meshes with a high polygon count. The computation of the first spheres is even accelerated by two orders of magnitude. As a by-product, this allows an almost real-time approximation of the medial axis of most objects.

2 Related Work

Sphere packing problems can be classified by several parameters, including the dispersity, the dimension, the orientation of the contacts between the spheres, the kind of container, etc. [ZT99]. The focus of this paper is the computation of space-filling polydisperse sphere packings for arbitrary container objects and arbitrary object representations in any dimension. Because of the wide spectrum of different sphere packing problems, we cannot provide a complete overview of all of them. Therefore, we restrict this review on recent and basic methods that are related to our problem definition or our approaches.

Polydisperse sphere packings are widely used and researched in the field of material science and in simulations that use the Discrete-Element method (DEM). Basically, there exist two different methods to construct polydisperse sphere packings: the *dynamic method* places a pre-defined distribution of spheres inside a given container and then changes the positions and the radii of the spheres until a required density is reached [LS90, KTS02]. In contrast, the *geometric method* places the spheres one after another, following geometric rules [JIDD09, JRID10]. Usually, the geometric method performs better, but the quality of the sphere packing depends heavily on the placement of the initial spheres.

There are also first approaches that support the parallel computation of polydisperse sphere packings. [Kub09] presented an algorithm to solve the 3D knapsack problem for spheres in a cube. They compute several greedy solutions simultaneously with a master-slave approach.

However, all these methods are restricted to very simple geometric containers like cubes or spheres. In particular, existing dynamic algorithms can be hardly extended to arbitrary container objects, because the dynamic simulation requires a time consuming collision detection of the spheres with the surface of the object.

All the approaches described above try to solve some kind of optimization problem. This means, that either the number or the size of the spheres are defined a priori. Real space-filling sphere packings potentially require an infinite number of spheres. They usually rely on fractal structures like the Apollonian sphere packings [AW00]. Such Apollonian sphere packings can be computed via an inversion algorithm [BPP94, BH04]. Packings like this are used in material science to create very compact materials and to avoid micro fractures in the materials [HBW03]. However, as above, all these algorithms are very time consuming and cannot be extended to arbitrary objects.

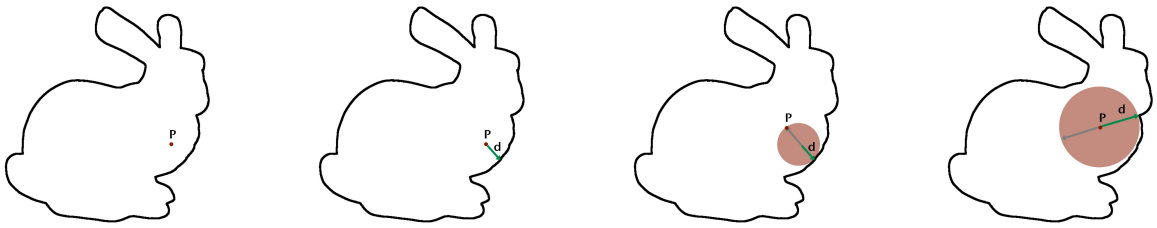
Sphere packings for arbitrary geometries are predominantly investigated in the field of radiosurgical treatment planning. Usually, the tumor region is represented by a polygonal model. A Gamma Knife can shoot spherical beams into this region. In order to avoid hot-spots (this are regions that are irradiated by several beams – they hold the risk of overdosage) but also in order to avoid underdosage in regions that are not hit by a beam, it is essential to compute a good spherical covering of the tumor region. [Wu96] was the first to formulate this problem as a min-max sphere packing. [Wan99] proved that this kind of problem is in fact NP-hard and proposed an approximation allowing arbitrary integer radii. According to [Wan00] there is an optimal solution where the centers of the spheres are located on the medial axis of the 3D region. However, the greedy choice to place the spheres is not optimal in each case. Due to their computational complexity, all these methods are limited to place at most a few hundred spheres in objects consisting of at most a few dozens of polygons.

3 Protosphere Recap

In this section we will start with a short recap of the original *Protosphere* algorithm and we will discuss its major bottlenecks.

The *Protosphere* algorithm, as introduced by [WZ10], is able to efficiently compute a space filling sphere packing for arbitrary container objects and object representations (polygonal, NURBS, CSG, etc.). The only precondition is that it must be possible to compute the distance to the object's surface from any point.

The goal is not so much to compute an optimal packing in some sense, but rather the generation of a *feasible* sphere packing: this means that all spheres are inside the object and



(a) (b) (c) (d)

Figure 1: Visualization of the prototype convergence: (a) Place the prototype P randomly inside the object, (b) calculate the closest point on the surface and the distance d, (c) move P away from the closest point, and (d) repeat this until the prototype converges.

the spheres do not overlap each other. The basic idea is to extend the method of *Apollonian sphere packings*, which are known to be space filling [AW00], to arbitrary container objects. This packing is achieved by successively inserting the largest possible sphere into the object.

Let O denote the surface of a closed, simple object in 3D. Consider the largest sphere s inside O. Obviously, s touches at least four points of O, and there are no other points of O inside s. This implies that the center of s is a Voronoi node (VN) of O. Consequently, we can formulate the Apollonian filling as an iterative computation of the VNs of the objects hull O plus the set of all spheres existing so far. However, computing the Voronoi Diagram (VD) explicitly is too time consuming. Hence, the authors propose to use an approximation scheme based on techniques known from machine learning. To do that, they approximate the VNs by placing a single point, the *prototype*, inside the object and let it move away from the objects surface in a few iterations (See Fig. 1). By choosing a clever movement, the prototype converges automatically towards a VN (See Algorithm (1). The last step of the algorithm guarantees that, after each single step, p is still inside the object, because the entire sphere around p with radius $\|p - q_c\|$ is inside the object.

Moreover, moving p away from the border, into the direction $(p - q_c)$, leads potentially to bigger spheres in the next iteration. Usually, $\varepsilon(t)$ denotes a cooling function that allows large movements in early iterations and only small changes in the later steps.

Algorithm 1: convergePrototype(prototype p, object O)

place p randomly inside O ;
while *p has not converged* **do**
$q_c = \arg\min\{ \|p - q\| \ : \ q \in$ surface of $O \}$;
choose $\varepsilon(t) \in [0, 1]$;
$p = p + \varepsilon(t) \cdot (p - q_c)$;

However, inserting just a single prototype may end up in a local optimum instead of converging toward the global optimum (see Fig. 3a). Therefore, the authors propose to use a set of prototypes that are allowed to move independently. This can be easily paral-

Figure 2: The original *Protosphere* algorithms simply places one prototype in each cell of a uniform grid (left). However, only very few of these prototypes contribute to potential centers of spheres (center). The yellow area denotes the discrete sphere in Manhattan distance for the green prototype in its center. All these cells are processed in parallel during the search for the closest point (right).

lelized. But a naive approach would results in a clumping of many prototypes at almost the same position and thus a lot of unneeded computations (see Fig. 3b). To overcome this problem, the authors apply a uniform grid that guarantees a uniform density of the prototypes. During the iterations, the prototypes are confined to their cells in order to maintain this property and to avoid a prototype clumping. Moreover, they use the grid to accelerate the distance queries by simply storing a discrete distance with each cell that denotes the number of cells between this and the closest border cell (see Fig. 3c). Only cells within less than this distance are candidates for possible closest points on the objects surface. Obviously, the discrete distance field has to be updated whenever a sphere is added.

However, simply using a grid of fixed resolution has some serious drawbacks. Especially in the early steps of the algorithm, when the object is filled only sparsely with spheres, most of the prototypes are *not* candidates for a VN, because their cells are completely contained inside Voronoi cells, hence they can not converge to a VN (see Fig. 2). Consequently, all distance computations that are performed for these prototypes are wasted. This results in a very slow filling rate during the early steps. Thus, a low resolution of the grid would be preferable at the beginning.

However, simply using a low resolution grid also has its drawbacks. Slightly different from the basic idea described above, the authors do not insert only a single sphere during each iteration, but they greedily insert spheres to all VNs that are not already covered. Consequently, a finer grid resolution in the later steps leads to a higher filling rate, because more spheres can be inserted simultaneously.

Moreover, the fixed grid has a very large memory footprint: for each cell we have to store a list of contained geometric primitives (e.g. triangles, spheres...), its discrete distance, some status information,etc. On the other hand, in the early steps of the algorithm,

(a) (b) (c)

Figure 3: (a) Depending on the start position of the prototype, the algorithm does not necessarily converge to the global optimum. (b) If the prototypes are allowed to move completely free, many of them clump together at the same VN. (c) Visualization of a discrete distance field with transparency = distance.

most of the cells are either completely outside or do not contain any geometric primitives. In the later steps, many cells are completely contained in spheres. Overall, allocating a huge amount of memory for these cells results in a waste of memory and it limits the resolution of the grid.

4 Our Approach

In this section, we will present several ideas to overcome the aforementioned limitations of the original *Protosphere* algorithm. In detail, we propose two different techniques for an adaptive adjustment of the grid. Each of these methods has individual advantages in different phases of the sphere packing process. Additionally, we combine these techniques to a hybrid approach in order to guarantee the maximum acceleration for the whole process. Finally, we propose a simple method to save memory that allows us to store much finer grids. All these improvements lead to a significant speed-up of the original algorithm (see Section 6).

4.1 Explicit Grid Refinement

In a sparsely filled object, a dense grid and therefore, a dense distribution of prototypes, leads to a large amount of unnecessary computations. As a consequence, it seems to be a good idea to simply use a coarser grid in the early phase of the sphere filling process and a finer grid at the end to be more likely to place prototypes inside the voids between the spheres. This is exactly the idea of our *explicit grid refinement*: We start with a low resolution grid and refine it consecutively in the later phases. To do that, we simply split each cell into 8 sub-cells (see Fig. 4).

Although this means that we have to re-compute the discrete distances and we have to re-assign the geometric primitives (e.g. triangles and spheres) to each sub-cell during

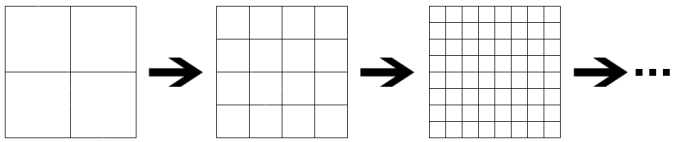


Figure 4: 2D visualization of the explicit refinement.

the refinement, it this does not take much more time than the standard updating of the distances after the insertion of new spheres

But, this method is not very well suited for parallelization any more: you will find the basic parallel version of the *Protosphere* algorithm in Algorithm 2.

First, a lower number of prototypes results in a lower number of threads that can be processed in parallel. The second problem is hidden in the search for the closest point in the `convergePrototype`-function: actually, we do not only process all prototypes in parallel, but we also parallelize the search for the closest neighbour for each prototype individually. To do that, we process all cells that are in a discrete sphere (Manhattan distance) around the prototype in parallel (see Fig. 2).

Consequently, a small amount of prototypes together with a sparse grid results in a low degree of parallelism.

Algorithm 2: parallelSpherePacking(object O)

In parallel: initialize discrete distance field

while *number of required spheres is not met* **do**

In parallel: place p_i randomly inside grid cell c_i ;

In parallel: convergePrototype(p_i, $O \cup$ inserted spheres)

In parallel: sort $\{p_i\}$ by max distance d_i

In parallel: find VNs $p_m \in \{p_i\}$ that are not overlapped by any p_i with bigger d_i

In parallel: insert spheres at positions p_m with radii d_m

In parallel: update discrete distance field

4.2 Implicit Grid Refinement

In order to overcome this drawback, we propose a second hierarchical grid, that is based on *implicit grid refinement*. Again, the basic idea is very simple: we maintain different grids, one for the parallel distance computation and one for the distribution of the prototypes (see Fig. 5).

Basically, we construct a dense grid and combine its *explicit* cells to construct an *implicit* grid on the top of that. We use the larger *implicit* cells to place the prototypes and the small

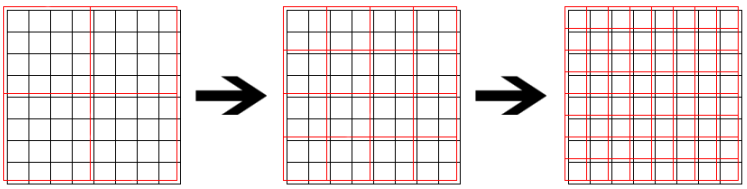
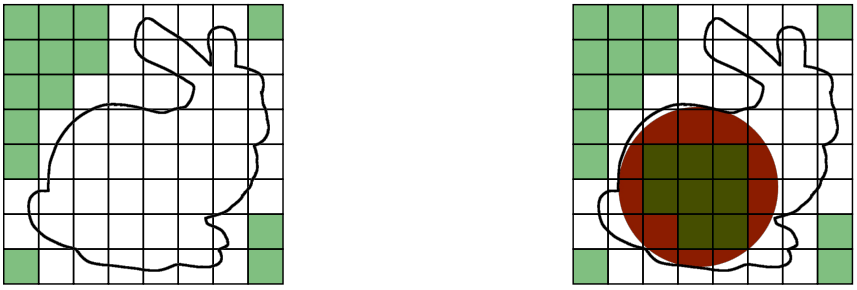


Figure 5: 2D visualization of the implicit refinement, the explicit grid colored black, the implicit red



(a) (b)

Figure 6: At the beginning, only cells that are outside the object are marked as inactive (green), this means that we do not have to assign a prototype to them (a). If the object becomes filled with spheres, also inside cells that are completely occupied by spheres become inactive (b).

explicit cells for the parallel distance computation. Moreover, the prototypes are confined to their *implicit* cell, but they are allowed to pass through the borders of the *explicit* cells.

Consequently, we do not have to re-compute the underlying *explicit* grid after an *implicit* refinement, and we have less prototypes in the early phases of the algorithm and more prototypes in the later phases. However, the resolution of the underlying *explicit* grid that is constructed once in a pre-processing step is limited by the memory.

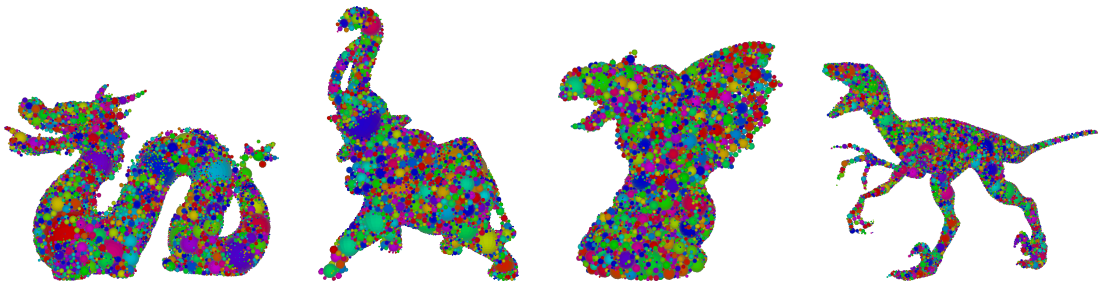
4.3 Adaptive Grid Storage

When we fill an object with spheres, there are some cells in the grid that we do not have to consider, e.g. because they are completely outside the object (see Fig. 6). In the following we will denote them as *inactive* cells. The number of these inactive cells may increase because some of the cells are completely covered by an inserted sphere. Obviously, we don't have to place prototypes into them. Each *active* cell has to store some information, e.g. a list of the intersecting geometric primitives, the discrete distance, some status information, etc. In order to avoid storing this information also for *inactive* cells, we propose an indirect addressing scheme: we store the whole grid in a large array of pointers that simply point to the respective information of the cell if the cell is active.

Due to the increasing number of inactive cells, the amount of active cells decreases during the algorithm and so does the memory that is required to store them. Conse-

(a) (b) (c) (d)

Figure 7: The models used for the timings: (a) A dragon, (b) an elephant, (c) a gargoyle and (d) a raptor



(a) (b) (c) (d)

Figure 8: The same models as in Fig.7 but filled with spheres: (a) A dragon, (b) an elephant, (c) a gargoyle and (d) a raptor

quently, we will be able to build finer *explicit* grids in the later phases of the algorithm, because there is more memory available. This observation directly implies a combination of the both aforementioned hierarchical grid approaches.

4.4 Hybrid Grid

The *hybrid grid* uses this by inserting refinements of the *explicit* grid whenever we refine the *implicit* grid. The first few iterations of Protosphere cover a lot of cells with spheres, so a lot of cells can become inactive. We do refinements as long as we have enough memory available to do them. This adds the benefit of being able to sustain an ideal relationship between the resolutions of the *implicit* and the *explicit* grid.

5 Results

We have implemented all our algorithms using NVIDIAs CUDA for the parallelization on the GPU. In this section we will present the results we obtained with the different grids that we introduced in the chapters before. We tested the algorithms with different settings and models. All timings presented here were done using a nVidia GTX680 with 4 GB VRAM and the models that you can see in Fig. 7. The number of triangles range

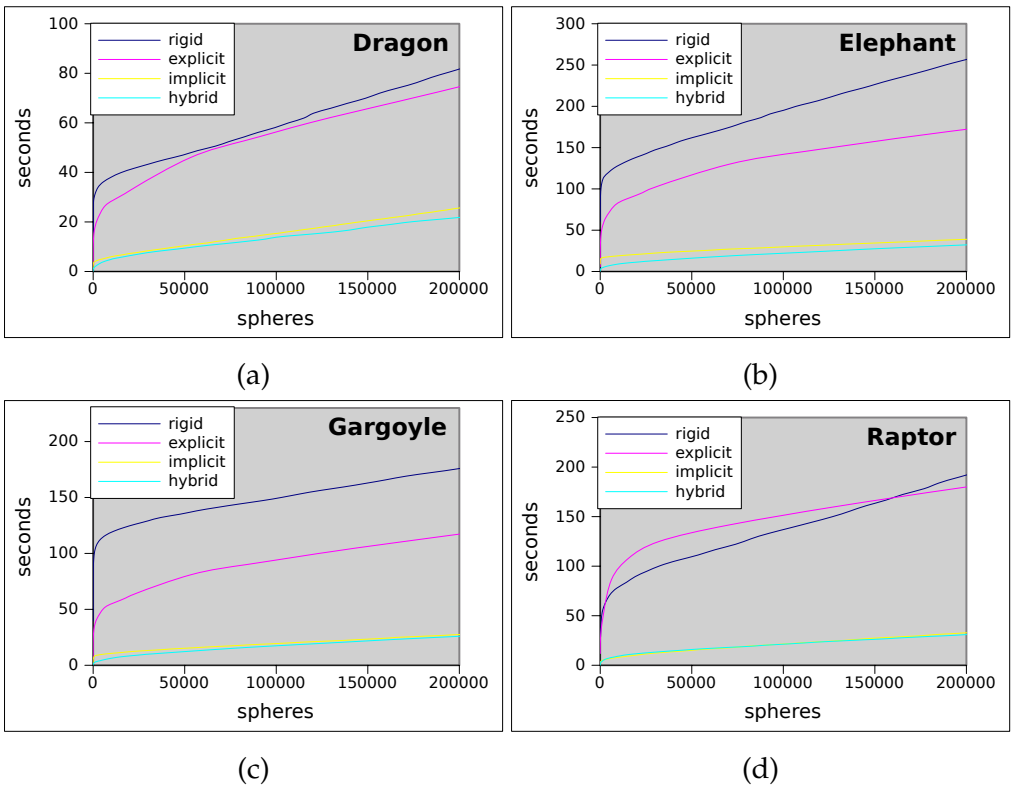


Figure 9: Comparison of the sphere filling speed for the different grids on all objects.

from 174k (the dragon) up to 800k (the raptor).

The graphs in Figure 9 show the performance of all kinds of grids with a manually chosen configuration for each object. Actually, the grid resolution for the original *protosphere* algorithm is between 70-140 cells on the longest side of the objects. For all other grids we start with 4-9 cells on the longest side for the rough grids and perform up to five refinements. We do a refinement after every 2 iterations. The resolutions of the *explicit* grid in the *hybrid* grids start with 16-72 cells on the longest side and we perform up to four additional refinements.

As you can see, a significant speed-up for all objects can be achieved using the *implicit grid refinement* and the *hybrid grid* compared to the *explicit grid refinement* and the original *Protosphere* algorithm. We were able to compute the sphere packings more than an order of magnitude faster with our new optimized algorithms. Especially in the first phase, when the objects were filled only very sparsely with spheres, we achieved a speed-up of more than two orders of magnitude. Surprisingly, the difference between the *implicit grid refinement* and the *hybrid grid* is very small in the long run in most cases. This is mainly, because the *hybrid grid* and the *implicit grid refinement* behave very similarly as soon as the explicit grids are at the highest resolution. That is why the *hybrid grid* speeds up mostly the beginning of the filling. The computation of the initial spheres can be up to an order of magnitude higher with hybrid refinement. For example with 0.445 seconds compared to 9.2 seconds for the first spheres on the elephant. There is a negative impact on the quality of the fillings when using very rough grids.

6 Conclusion and Future Work

We have presented four easy-to-implement extensions to the sphere packing algorithm *Protosphere*. Namely, these are *implicit grid refinement* to accelerate the early phase of the algorithm when the object is only sparsely filled with spheres, and the *explicit grid refinement* for the later phase. While the explicit refinement speeds up the end of the packing by allowing very fine grids, the implicit refinement speeds up the beginning by better distributing the computing workload of finding the closest point on the objects surface. Moreover, we combined both approaches in the *Hybrid Grid* algorithm that combines the benefits of the other kinds of grids, thus effectively eliminating their drawbacks. Finally, we have presented a basic technique to improve the memory load of all grid methods.

Our massively parallel implementation of our new algorithms using CUDA proves that they are able to outperform the original approach by more than an order of magnitude also in practice. Additionally, the overall finer grid structure allows us to fill very filigree objects. This allows new applications for our sphere packings, e.g. in the field of molecular simulations.

As mentioned before, the first iteration of the Protosphere algorithm delivers an approximation of the objects' Voronoi nodes and its medial axis. With the implicit grid it should be possible to compute a generalized Voronoi diagrams in almost real-time. One possible application could be real-time path-planning in dynamic environments

However, there is still room for improvements for both our new methods and the fast computation of sphere packings. For instance, in our timings, we tweaked the grid size and the refinement parameters manually. It would be nice to compute the optimal parameters automatically. Another optimization could be the use of even more memory efficient spatial hashing for the explicit refinement phase, because then, the number of inactive grid cells is relatively small.

Finally, it would be interesting to apply our massively parallel hierarchical method also to other optimization problems. Many approaches in machine learning are based on the movement of single points in some data sets. A hierarchical optimization could probably help to find some globally interesting points and then use these as a basis for further exploration with a finer level of detail.

References

[AW00] Tomaso Aste and Denis Weaire. *The pursuit of perfect packing*. Institute of Physics Publishing, Bristol, UK and Philadelphia, PA, USA, 2000.

[BH04] Reza M. Baram and Hans J. Herrmann. Self-similar space-filling packings in three dimensions. *Fractals*, 12:293–301, 2004.

[BPP94] Micha Borkovec, Walter De Paris, and Ronald Peikert. The fractal dimension of the apollonian sphere packing. *Fractals An Interdisciplinary Journal On The Complex Geometry Of Nature*, 2(4):521–526, 1994.

[HBW03] H.J. Herrmann, R. Mahmoodi Baram, and M. Wackenhut. Searching for the perfect packing. *Physica A: Statistical Mechanics and its Applications*, 330(1-2):77 – 82, 2003. RANDOMNESS AND COMPLEXITY: Proceedings of the International Workshop in honor of Shlomo Havlin's 60th birthday.

[HM09] Mhand Hifi and Rym M'Hallah. A literature review on circle and sphere packing problems: Models and methodologies. *Adv. Operations Research*, 2009, 2009.

[JIDD09] Jean-Francois Jerier, Didier Imbault, Frederic-Victor Donze, and Pierre Doremus. A geometric algorithm based on tetrahedral meshes to generate a dense polydisperse sphere packing. *Granular Matter*, 11(1):43–52, 2009.

[JRID10] Jean-Francois Jerier, Vincent Richefeu, Didier Imbault, and Frederic-Victor Donze. Packing spherical discrete elements for large scale simulations. *Computer Methods in Applied Mechanics and Engineering*, 199(25-28):1668 – 1676, 2010.

[KTS02] Anuraag R. Kansal, Salvatore Torquato, and Frank H. Stillinger. Computer generation of dense polydisperse sphere packings. *The Journal of Chemical Physics*, 117(18):8212–8218, 2002.

[Kub09] T. Kubach. Parallel greedy algorithms for packing unequal spheres into a cuboidal strip or a cuboid. 2009.

[LS90] Boris D. Lubachevsky and Frank H. Stillinger. Geometric properties of random disk packings. *Journal of Statistical Physics*, 60(5-6):561–583, 1990.

[Wan99] Jie Wang. Packing of unequal spheres and automated radiosurgical treatment planning. *Journal of Combinatorial Optimization*, 3(4):453–463, 1999.

[Wan00] Jie Wang. Medial axis and optimal locations for min-max sphere packing. *Journal of Combinatorial Optimization*, 4:487–503, 2000. 10.1023/A:1009889628489.

[Wu96] Qing Rong Wu. *Treatment planning optimization for gamma unit radiosurgery*. PhD thesis, Biophysical Science–Biomedical Imaging–Mayo Graduate School, 1996.

[WZ09] Rene Weller and Gabriel Zachmann. A unified approach for physically-based simulations and haptic rendering. In *Sandbox 2009: ACM SIGGRAPH Video Game Proceedings*, New Orleans, LA, USA, August 2009. ACM Press.

[WZ10] René Weller and Gabriel Zachmann. Protosphere: A gpu-assisted prototype guided sphere packing algorithm for arbitrary objects. In *ACM SIGGRAPH ASIA 2010 Sketches*, SA '10, pages 8:1–8:2, New York, NY, USA, 2010. ACM.

[YMWZ12] Weiyu Yi, Stefan Mock, Rene Weller, and Gabriel Zachmann. Sphere-spring systems and their application to hand animation. In *VR/AR*, pages 131–142, 2012.

[ZT99] C. Zong and J. Talbot. *Sphere packings*. Universitext (1979). Springer, 1999.