Titel/Title: ProtoSphere: A GPU-Assisted Prototype Guided Sphere Packing Algorithm for Arbitrary Objects

Autor*innen/Author(s): Rene Weller, Gabriel Zachmann

Veröffentlichungsversion/Published version: Postprint

Publikationsform/Type of publication: Konferenzbeitrag

Verfügbar unter/Available at:
(wenn vorhanden, bitte den DOI angeben/please provide the DOI if available)

10.1145/1899950.1899958

Zusätzliche Informationen/Additional information:

ProtoSphere: A GPU-Assisted Prototype Guided Sphere Packing Algorithm for Arbitrary Objects

Rene Weller, Gabriel Zachmann, Clausthal University, Germany

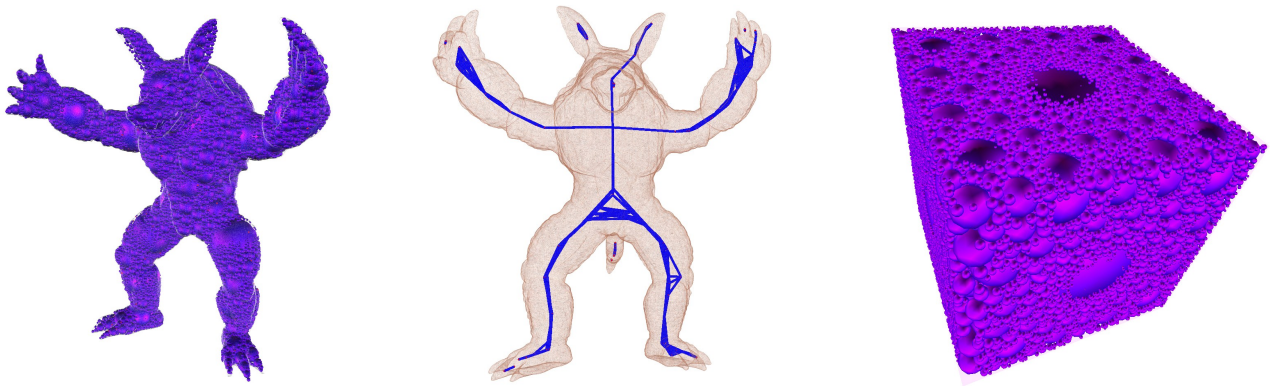

Figure 1: *A model of an armadillo filled with 15000 spheres (left). As a side effect of our algorithm, we get an approximation of the medial axis in the first iteration (middle). The sphere filled cube shows the fractal Apollonian property of our algorithm (right).*

1 Motivation

Filling objects densely with sets of non overlapping spheres has been investigated for centuries. Once started as a pure intellectual challenge, today, 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, the discrete element method is based on them, and so forth.

In computer graphics, sphere packings also have proven to be very efficient when doing collision detection, which was the starting point of our research [Weller and Zachmann 2009].

Additionally, volumetric representations provide advantages for physically based simulations of deformable objects: with simple mass spring systems it is hard to fulfill volume preservation; on the other hand, FEM based methods are hard to do in real-time. So, an approximation of the volume in combination with fast mass spring methods can result in a novel approach for deformable object simulation.

The voxel-based method used so far tends to produce a lot of small and very regularly placed spheres close to the surface. This results in artifacts, in particular temporal aliasing, in the collision response.

In this talk, we would like to present a new algorithm that is able to efficiently compute a space filling sphere packing for arbitrary objects. It is independent of the object's representation (polygonal, NURBS, CSG,...); the only precondition is that it must be possible to compute the distance from any point to the surface of the object. Moreover, our algorithm is not restricted to 3D but can be easily extended to higher dimensions.

The basic idea is very simple and related to *prototype based approaches* known from machine learning. This approach directly leads to a *parallel algorithm* that we have implemented using CUDA. As a byproduct, our algorithm yields an approximation of the object's *medial axis* that has applications ranging from path-planning to surface reconstruction.

2 Our Approach

A simple algorithm to fill an object with a set of non-overlapping spheres is the following greedy method. For a given object we start with the largest sphere that fits in the object. Iteratively, we insert new spheres, under the constraints that a) they must not intersect the already existing spheres and b) that they be completely contained inside the object.

The resulting sphere packing is called an "Apollonian sphere packing". One important property of Apollonian packings is that they are known to be space filling. There exist efficient algorithms to compute Apollonian diagrams for very simple geometrical shapes like cubes or spheres (see e.g. [Mahmoodi-Baram and Herrmann 2004]) but they are hardly expandable to arbitrary objects, let alone their computation time.

2.1 Basic Idea

Let P denote the surface of a closed, simple object in 3D. Consider the largest sphere s inside P. Obviously, s touches at least 4 points of P, and there are no other points of P inside s. This implies that the center of s is a Voronoi node (VN) of P. Consequently, it is possible to formulate the greedy space filling as an iterative computation of a generalized Voronoi diagram (VD) of P plus the set of all spheres existing so far.

Many algorithms have been devised for the calculation of the classic VD and for its many generalizations. However, there are relatively few works dedicated to the construction of VDs for spheres in 3D. E.g. [Wang 1999] used 3D Voronoi diagrams for spheres to approximately solve the optimal sphere packing problem. But, this approach can handle only points and spheres and moreover it is very slow. [Baran and Popović 2007] used an approximative greedy sphere packing algorithm for automatic rigging of animated characters. The spheres are all located on the medial axis and they are allowed to overlap. Even if it is possible to extend this approach to more general sphere packings, the re-computation of intermediate data structures like an octree and a kd-tree would make it inefficient, because it already needs several seconds of computation time to compute just an approximation of the medial axis. To our knowledge, there is no algorithm that supports the computation of VDs for a mixed set of triangles and spheres, let alone a fast and stable implementation.

Fortunately, a closer look at the simple algorithm we proposed above shows that we do not need the whole VD, but only the VNs. Hence, the core of our novel algorithm (which we call "ProtoSphere") is the approximation of the VNs. Again, the basic idea is very simple: we let a single point, the *prototype*, iteratively move

Figure 2: *The space filling rate of our sphere packing algorithm (left). The models used for the timings: A cow, a pig, a bust and a dragon (middle). Timings for different objects on a Geforce GTX480 (right; please note that the code is not optimized yet).*

towards one of the VNs:

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 $\varepsilon \in [0, 1]$
  $p = p + \varepsilon \cdot (p - q_c)$

The last line 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, ε is not a constant, but a cooling function that allows large movements in early iterations and only small changes in the later steps.

The accuracy of the approximated VN depends on the choice of this cooling function and on the number of iterations.

2.2 Parallelization

Using a single prototype does not guarantee to find the global optimum (which is the sought-after VN), because the algorithm presented in the previous section depends on the starting position of the prototype. Hence, we use a set of independently moving prototypes instead of only a single one. This can be easily parallelized if the prototypes are allowed to move independently.

Therefore, we compute a uniform grid and start with a prototype in each cell that is located inside the object. During the movement step of Algorithm 1, the prototypes are confined to their cells. This results in a uniform density of the prototypes, and moreover the grid can be used to speed up the distance computations. For the latter, we additionally compute the discrete distance from each cell to the surface. The complete parallelized algorithm can be written as follows:

Algorithm 2: spherePacking(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: Find VN $p_m \in \{p_i\}$ with max distance d_m
  Insert sphere at position p_m with radius d_m
  In parallel: Update discrete distance field

Please note that after the convergence of the initial set of prototypes, we get an approximation of the medial axis. Its accuracy depends on the number of initial prototypes and thus on the size of the grid. In addition, our algorithm extends Apollonian sphere packings to arbitrary objects. This is the reason for the space filling property of our algorithm.

3 Results

We have implemented our algorithm using CUDA. Figure 2 shows the results for some of the objects that we have filled with spheres. The triangle count reaches from 10.000 for the pig up to 300.000 for the dragon. We are able to fill all objects with 100.000 spheres within a few seconds using a NVIDIA GTX480 graphics card. The number of iterations of Algorithm 1 was set to 50. The accuracy of the computed Voronoi nodes is $> 99.99\%$ compared to the exact Voronoi nodes position.

Surprisingly, the filling rate depends only on the number of spheres but is independent of the objects shapes, at least with all objects that we have tested.

4 Conclusions and Future Work

Summarizing, we have presented a novel method for filling arbitrary objects very quickly and stably with sets of non-overlapping spheres. It is optimal in the sense, that it produces space filling sphere packings due to the Apollonian property, but it is not optimal in the number of spheres, which is known to be an NP complete problem. However, small modifications of our algorithm could also help to solve sphere packing problems with other objectives, e.g. replacing the greedy choice by dynamic programming would maximize the covered volume for a predefined number of unequal spheres.

Beyond that, our algorithm offers numerous other starting points for future work. For instance, further optimization of the implementation could make, at least the first step and thus the approximation of the medial axis, capable for real-time applications. This could be helpful e.g. for path-planning in deformable environments. Moreover our algorithm can be used as a blueprint for several generalized Voronoi diagram problems.

References

BARAN, I., AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3d characters. *ACM Trans. Graph. 26*, 3, 72.

MAHMOODI-BARAM, R., AND HERRMANN, H. J. 2004. Self-similar space-filling packings in three dimensions. *Fractals 12*, 293–301. cond-mat/0312345.

WANG, J. 1999. Packing of unequal spheres and automated radiosurgical treatment planning. *Journal of Combinatorial Optimization 3*, 453–463. 10.1023/A:1009831621621.

WELLER, R., AND ZACHMANN, G. 2009. A unified approach for physically-based simulations and haptic rendering. In *Sandbox 2009: ACM SIGGRAPH Video Game Proceedings*, ACM Press, New Orleans, LA, USA.