Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2013)
J. Bender, J. Dequidt, C. Duriez, and G. Zachmann (Editors)

Parallel Collision Detection in Constant Time

Rene Weller, Udo Frese and Gabriel Zachmann

University of Bremen, Germany

Abstract

We prove that the maximum number of intersecting pairs spheres between two sets of polydisperse sphere packings is linear in the worst case. This observation is the basis for a new collision detection algorithm. Our new approach guarantees a linear worst case running time for arbitrary 3D objects. Additionally, we present a parallelization of our new algorithm that runs in constant time, even in the worst case. Consequently, it is perfectly suited for all time-critical environments that allow only a fixed time budget for finding collision. Our implementation using CUDA shows collision detection at haptic rates for complex objects.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms

1. Introduction

Collision detection is a fundamental problem that arises in all tasks involving the simulated motion of objects that are not allowed to penetrate one another. For instance, it is required in interactive physically-based real-time simulations that are widely used in computer graphics [BV91], computer games [BEW*98], virtual reality [ES99] or virtual assembly tasks [KKLL95], but also applications in robotics where collision detection helps to avoid obstacles [CG98] and self-collisions between parts of a robot [KNK*02]. Moreover, it is required for path planning [LaV04], molecular docking tasks [Tur89] and multi-axis NC-machining [IEH*05] to name but a few.

In most of these applications, collision detection is the computational bottleneck. The main reason for this is its inherent complexity: consider two objects in a polygonal surface representation, each of them is modelled by n polygons. A brute-force approach for a collision detection algorithm could be to simply test each polygon of one object against each polygon of the other object. This results in a complexity of $O(n^2)$. In practice, a quadratic running-time of the collision detection for complex objects consisting of millions of polygons is not an option. Usually, bounding volume hierarchies (BVHs) are used as the standard acceleration data structure for collision detection: they provide output sensitive queries and they try to prune parts of the objects that can not overlap as early as possible. In many cases, this leads to acceptable running times.

However, it is easy to describe and construct polyhedra that will lead to a quadratic number of *intersections* among the polygons. Note, that the polyhedron depicted in Figure 1 is not particularly pathological. Consequently, each collision detection algorithm, including all polygon-based BVHs, that is able to compute all pairs of colliding polygons has a quadratic worst case running time.

Unfortunately, it is hard to foresee situations when this happens in advance in interactive applications. In computer games or VR, this may result in a few dropped frames and, thus, in a short stuttering of the simulation. However, there exist applications where hard real-time constraints are a necessity. For instance, in robotics and, especially, in haptics, where often simulation frequencies of 1 kHz are required, such a stuttering can damage the expensive hardware or injure people. Hence, there is a need for time-critical collision detection algorithms that are able to guarantee to absolutely never exceed a certain time budget even in the worst case.

In this paper, we present a new collision detection algorithm with a sequential running time that is only *linear* in the worst case; furthermore, its parallel time is *constant* in the worst case while using only a linear number of processors. The main idea is to avoid the polygonal object representation that leads to the quadratic complexity; instead, we represent the object by a volumetric *sphere packing* (see Figure 1). In this context, running times are, therefore, measured in the number of spheres, which is related to the volumetric approximation of the sphere packing.

DOI: 10.2312/PE.vriphys.vriphys13.061-070

Figure 1: Left: the intersection of two Chazelle polyhedra is a worst case for polygon-based collision detection algorithms. It has a quadratic complexity. Right: our algorithm is based on space-filling, polydisperse sphere packings for arbitrary objects and, thus, independent of the object's surface complexity.

More precisely, we use a set of *non-overlapping* spheres that are all located *inside* of the object. We allow different radii of the spheres. This supports *space-filling* sphere-packings. This data structure has several advantages: first, it is independent of the object's representation and can be used for almost all surface descriptions (polygonal, NURBS, point clouds, CSG,...). Second, it is also independent of the object's geometric complexity. Consequently, it provides the user with a natural choice between speed and accuracy. The only pre-condition is that the objects have to be watertight so that they can be filled with spheres.

In addition to the linear running time, our algorithm (to be presented in Section 4) has a number of other benefits: for instance, it does not require any complicated preprocessing steps. Hence, it is, in principle, also suitable for deformable objects. Moreover, our algorithm does not simply check whether a pair of objects collides or not, but it also approximates their *penetration volume*. This penetration measure is known to be "the most complicated yet accurate method" [FL01, Sec. 5.1] to define the extent of intersection. It can be directly used to compute repelling forces for haptics or physically-based simulations.

In Section 3 we will start with a theoretic proof that there are at most $O(n)$ overlapping pairs of spheres if two sets of polydisperse sphere packings collide. This guarantees a linear worst case complexity. In Section 4 we use this observation do define an algorithm with linear worst case running time based on hierarchical grids. Additionally, we show that the construction of the hierarchical grid, as well as the traversal, can be easily parallelized. This results in an, almost, *constant* time parallel algorithm. More precisely, the running time depends only on the sizes of the spheres, but not on their quantity.

Finally, we have implemented our algorithm using NVIDIA's CUDA. Our results show that our new collision detection algorithm can answer collision queries for complex objects at haptic rates in less than 1 msec in the worst case while still providing continuous forces and torques.

2. Related Work

Today, there exist many different algorithms and data structures for collision detection. Often BVHs based on spheres [Hub96, Qui94], AABBs [PML95, vdB98] and their memory optimized derivative called BoxTree [Zac02], k-DOPs [KHM*98, Zac98], a generalization of AABBs, OBBs [GLM96, ASC*06] or convex hull trees [EL01] are used to accelerate collision queries. Also hierarchies on sphere packings has been proven to be very efficient in practice when used for collision detection [WZ09]. The authors included a time-critical version, but this is not able to guarantee a continuity of forces and torques.

Several attempts has been made to port BVH-based collision detection to the GPU. For instance Govindaraju et al. [GKJ*05a] used chromatic decompositions of a meshes to check for collisions between non-adjacent primitives. Greß et al. [GGK06] used the stencil buffer to generate BVHs for deformable objects in real-time. Lauterbach et al. [LMM10] presented a fast GPU-based OBB trees construction.

Usually, the stackless processors on the GPU are not very well suited for a BVH traversal. Therefore, some algorithms have been developed for collision detection on the GPU that avoid BVHs completely. The first GPU-based approaches relied on the fixed-function graphics pipeline and used image space techniques. For instance, Knott and Pai [KP03] implemented a ray-casting algorithm based on frame buffer operations to detect static interferences between polyhedral objects. Heidelberger et al. [HTG04] described an algorithm for computation of layered depth images using depth and stencil buffers.

Later, the fixed function pipeline has been replaced by programmable vertex and fragment processors. This also changed the GPU collision detection algorithms: for example, Zhang and Kim [ZK07] performed massively-parallel

pairwise intersection tests of AABBs in a fragment shader. Kolb et al. [KLRS04] used shaders for the simulation of large particle systems, including collisions between the particles. Today, GPU processors are freely programmable via APIs such as OpenCL or CUDA. This further improved the flexibility of GPU-based collision detection algorithms, like the approach by Pan and Manocha [PM12] that uses clustering and collision-packet traversal or a method based on linear complementary programming for convex objects [Kip07]. Morvan et al. [MRS08] presented an algorithm for proximity queries between a closed rigid object and an arbitrary mesh using distance fields. Mainzer and Zachmann [MZ13] developed a method based on parallel sorting and fuzzy clustering. Faure et al. [FBAF08] computed an approximation of the intersection volume from layered depth images on the GPU. Their approach supports deformable objects and was later extended to continuous forces [AFC*10].

However, all polygon-based approaches have a quadratic worst-case running time. The running time of the image space algorithms depends on the image resolution. Moreover, it is often impossible to guarantee a certain force and torque quality for these approaches.

A known constant time method has been developed especially for 6-DOF haptic rendering – the Voxmap pointshell (VPS) algorithm [MPT99]. The main idea is to divide the virtual environment into a dynamic object, that is allowed to move freely through the virtual space and static objects that are fixed in the world. The static environment is discretized into a set of voxels, whereas the dynamic object is described by a set of points that represents its surface. During query time, for each of these points it is determined with a simple boolean test, whether it is located in a filled volume element or not. Many extension for the classical VPS algorithms have been proposed [MPT05, MPT06, PH05, RPP*01]. However, none of these extensions was able to overcome the huge memory-footprint of the voxmap and the need for different data structures for moving and fixed objects. Additionally, the resulting forces are very noisy [WMS*10].

Due to the inherent quadratic complexity of polygon-based collision detection, most authors simply do not include a theoretic running time analysis, but they use benchmarks to compare the speed of their algorithms [OL03, CRM02, GKJ*05b, TWZ07]. Actually, the literature about the theoretic running time of collision detection algorithms is relatively sparse compared to large number of real implementations. Usually, some constraints about the shape or the motion of the objects was made.

One of the first theoretic results was presented by Dobkin and Kirkpatrick [DK85]. They have shown that the distance of two convex polytopes can be determined in time $O(\log^2 n)$, where n is the number of faces of the objects. For two general polytopes whose motion is restricted to fixed algebraic trajectories, Schömer and Thiel [ST95] have shown that there is an $O(n^{\frac{5}{3}+\varepsilon})$ algorithm for rotational movements. For a more flexible motion that still has to be along fixed known trajectories they presented an $o(n^2)$ algorithm [ST96]. Suri et al. [SHH98] proved that for n convex, well-shaped polytopes (with respect to aspect ratio and scale factor), all intersections can be computed in time $O((n+k)\log^2 n)$, where k is the number of intersecting object pairs.

Also some results about the *expected* running time of BVH-based algorithms has been published: under mild coherence assumptions, Vemuri et al. [VCC98] showed linear expected time complexity for the CD between n convex objects. Weller et al. [WKZ06] presented a model to estimate the expected running time of the simultaneous traversal of two binary BVHs. They showed an average running time of $O(n)$ or even in $O(\log n)$ for realistic cases, depending on the overlap of the root bounding volumes and the diminishing factor.

3. Theoretic Background

In this section, we will prove the theoretic basis of our new linear time collision detection method. As mentioned in the introduction, we do not use the object's *surface* representation, like most other collision detection algorithms do, but we represent the *volume* of an object by a polydisperse sphere packing. Actually, our proof does not rely on an optimized sphere packing, we simply require a *valid* sphere packing. This means, the only pre-condition is that the spheres in such a sphere packing are not allowed to intersect each other. All spheres may have different radii.

Let A and B be two objects with polydisperse sphere packings S_A and S_B, respectively, for each of these objects. We define $n = \max\{|S_A|, |S_B|\}$, the maximum number of spheres. We claim that the maximum number of overlapping pairs of spheres between S_A and S_B is in $O(n)$. Note, if all spheres had the same radius, such a proof would be trivial. However, due to the Kepler conjecture, the maximum density for sphere packings of equally sized spheres is always smaller than 75%. This would result in large errors for the penetration volume. On the other hand, polydisperse sphere packings, like the Apollonian sphere packing, are known to be space-filling. Moreover, they allow a better density with a smaller amount of spheres. Therefore, we decided to use polydisperse sphere packings instead of equally sized sphere packings. However, in this case, the proof is not trivial anymore because a large number of small spheres of S_A can intersect a single large sphere in S_B (see Figure 2).

In order to prove our theorem, we first consider a single sphere: Let r_i be the radius of a sphere s_i. First, we claim that a single sphere $s_i \in S_A$ can intersect at most a constant number of spheres $s_j \in S_B$, where only those spheres $s_j \in S_B$ are considered that have at least the same radius as s_i, this means $r_i \geq r_j$. We will prove this claim later.

We use this observation to define pairs of intersecting

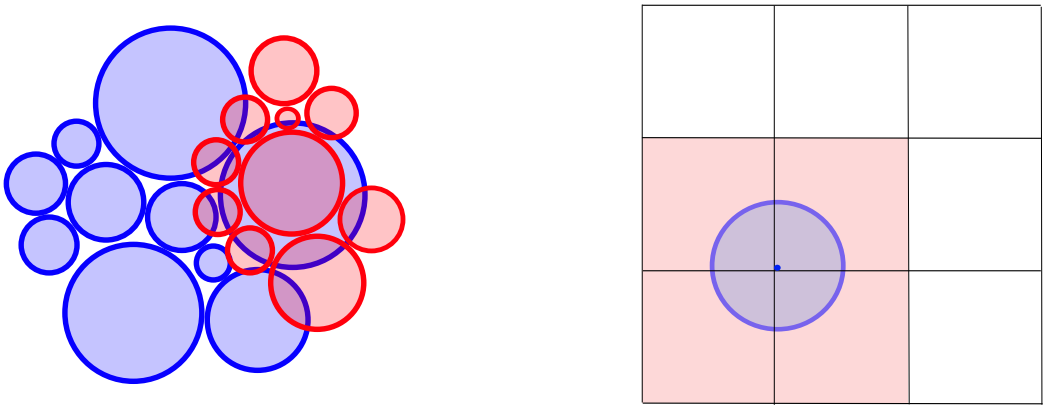


Figure 2: Left: the intersection between the red and the blue sphere packing has a worst case complexity of $O(n)$, even if many small red spheres intersect the large blue sphere on the right side. Right: The diameter of the blue sphere equals the length of the cell. In the worst case, the sphere can intersect four cells (red) in 2D in its neighbourhood (In 3D, this will be eight cells).

spheres with at least the same radius for each individual sphere $s_i \in S_A$ as

$$I^{\geq}_{AB}(s_i) = \{(s_i, s_j) : s_j \in S_B \wedge s_i \cap s_j \neq \varnothing \wedge r_j \geq r_i\}$$

where r_i denotes the radius of s_i and r_j the radius of s_j. Assuming the claim from above is true, we get for each sphere $|I^{\geq}_{AB}(s_i)| \leq k$ for some constant k. Analogously, we can define such sets $I^{\geq}_{BA}(s_j)$ of intersecting spheres for sphere packing S_B. The only difference is that we allow only those intersecting spheres of S_A with strictly larger radius. This avoids double counting of sphere pairs with the same radius. Obviously, the union of all these sets

$$\bigcup_{s_i \in S_A} I^{\geq}_{AB}(s_i) \cup \bigcup_{s_j \in S_B} I^{>}_{BA}(s_j)$$

contains all pairs of intersecting spheres between S_A and S_B: if a sphere $s_i \in S_A$ intersects a sphere $s_j \in S_B$ with a smaller radius, this will be counted in $I^{>}_{BA}(s_j)$ and vice versa.

From the initial assumption we get $|I^{\geq}_{AB}(s_i)| \leq c$, which is also true for $I^{>}_{BA}(s_j)$, Moreover we have $|S_A| \leq n$ and $|S_B| \leq n$ by definition. Combining this delivers

$$\sum_{s_i \in S_A} I^{\geq}_{AB}(s_i) \leq kn$$

and also

$$\sum_{s_j \in S_B} I^{>}_{BA}(s_j) \leq kn$$

. This two inequalities guarantee a total number of intersecting sphere pairs that does not exceed $O(n)$, even in the worst case.

It remains to prove the initial claim:

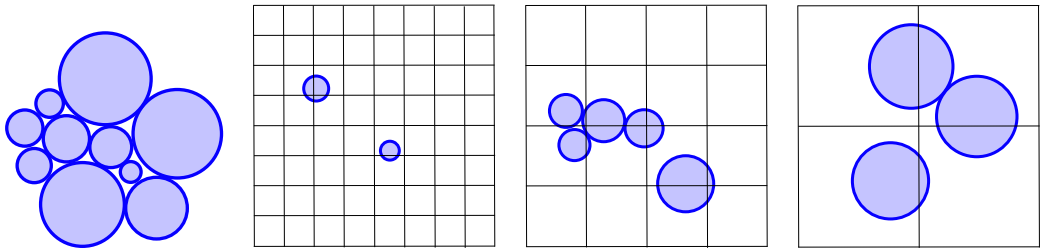
Lemma 3.1 A single sphere s with radius r intersects at most a constant number of spheres that have at least the same radius.

Let I be the set of intersecting spheres. The radius r_i of each sphere $s_i \in I$ has at least the same radius as s, this means, $r_i \geq r$. W.l.o.g. we can assume that $r_i = r$ for all $s_i \in S$; In other words, we assume that all spheres have the same radius. If some sphere s_i has a larger radius $r_i > r$, we can simply replace it by a sphere s^r_i with radius r. To do that, we place s^r_i at the center inside s_i and move it on a straight line between the centers of s_i and s towards s until it first intersects s. Because s_i also intersects s due to the pre-condition and $r_i > r$, s^r_i has to be completely inside s_i. Consequently, replacing larger spheres does not change the total number of intersecting spheres.

All spheres $s_i \in I$ intersect s and all have the same radius r. Therefore, the centers of $s_i \in I$ have a distance of at least $2r$ from the center of s. This means, they all have to be completely inside a sphere of radius $3r$ and the center of s. Obviously, only a constant number of non-overlapping spheres with radius r fits completely inside a sphere of radius $3r$.

The constant is closely related to the kissing number [BL11], but it is not exactly the same. The kissing number is defined as the number of non-overlapping equivalent spheres that can be arranged such that they touch, but do not intersect, another equivalent sphere. This problem is still an active field of research, especially for higher dimensions. In our case, we can simply argue with the spheres' radius and their volume to prove the constant. This is also a simple upper bound for the kissing number. Note, that all our theorems and proofs hold for any dimension, even though we apply it only to 3D collision detection in this paper. However, the constant highly depends on the dimension, ie, higher dimensions lead to larger constants.

The proof of the lemma finishes the proof of the Theorem 3.2. In the next section we will use these observations to achieve an algorithms with linear running time.



(a) Sphere packing (b) Hierarchy level 1 (c) Hierarchy level 2 (d) Hierarchy level 3

Figure 3: Spheres of the sphere packing are assigned to different uniform grids in a grid hierarchy, depending on their radius.

Theorem 3.2 The maximum number of intersecting pairs of spheres of two polydisperse sphere packings with n spheres is in $O(n)$.

4. Our Approach

Due to Theorem 3.2, the total number of intersecting pairs of spheres is linear for two polydisperse sphere packings. Consequently, a linear collision check is possible in principle. The next question is: how can we realize this? Fortunately, the proof of the theorem already gives a first hint: basically, for each sphere we have to check only spheres with larger radius. However, checking naively all sphere with larger radius still results in a quadratic running time. To avoid this, we have to constrain the number of spheres to be visited to a constant number.

4.1. Basic Algorithms

If all spheres were of uniform size, a uniform grid would fullfill all requirements: first, it would allow a fast localization of the query sphere. Second, a cell in a grid naturally bounds the number of spheres that are located inside such a cell. However, in our setting, we have a polydisperse sphere packing and all spheres are allowed to have different radii. In order to still benefit from the advantages of a uniform grid data structure, we propose to use a *hierarchy of grids*.

More precisely, let S be a polydisperse sphere packing and $s_{min} \in S$ the sphere with minimum radius r_{min}. We simply set the smallest possible cell size c_{min} in the hierarchical grid as $c_{min} = 2r_{min}$, ie, the diameter of the minimum sphere corresponds to the size of the minimum cell. We construct the higher levels in the grid hierarchy by successively doubling the length of the cells. Consequently, we assign each sphere $s_i \in S$ with radius r_i to a level so that the cell size is *at most the diameter of the sphere, but at least its radius*. Hence, each sphere is assigned to a level l_i with

$$2^{l_i} \cdot c_{min} \leq r_i < 2^{l_i+1} \cdot c_{min}.$$

This binning of the spheres has two advantages: First, it ensures that a sphere in hierarchy level l_i can intersect only spheres whose centers are located in direct neighbour cells on the same hierarchy level. If the center of a sphere s in level l_i with a diameter of at most 2^{l_i+1} is farther away from a cell c, it cannot intersect any sphere whose center is in c.

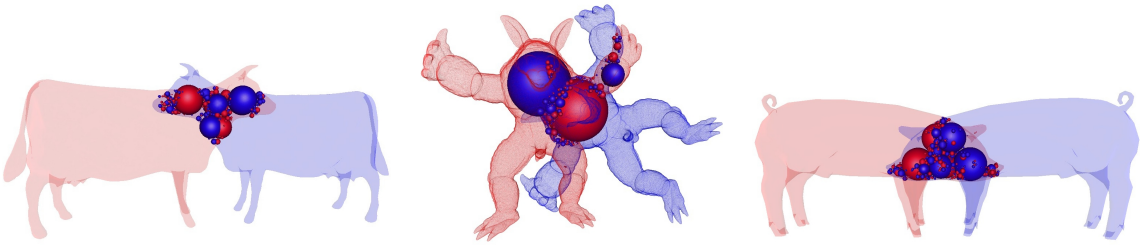
Second, it limits the maximum number of spheres that fits into a cell to a relatively small constant: In order to construct our hierarchical grid data structure, we simply compute the level l_i and assign a pointer to each cell on this level that is intersected by the spheres. Due to the choice of the cell size, this can be at most eight cells – the cell where the center of s_i is located in, and at most seven of its neighbour cells (see Figure 2).

During query time, we want to check which pairs of spheres from two different sphere packings S_A and S_B collide. To do that, we again have to locate the appropriate hierarchy level l_i for each sphere $s_i \in S_A$, but this time in the hierarchical grid of S_B. In order to get colliding spheres of S_B, we initially check all those spheres of S_B that are assigned to cells that are intersected by s_i on that level. Again, the number of intersected cells can be at most eight, with the same argument as above.

In order to get also collisions with larger spheres, we simply ascend in the hierarchy. This means, we compute the cells that are occupied by s_i in the hierarchy levels $l_i + 1 \ldots l_{max}$, where l_{max} denotes the maximum level. l_{max} depends only on the maximum sphere size in S_B. In each of these cells we have to test s_i for collisions with all spheres inside these cells. Obviously, this can be at most eight cells for each level.

Actually, locating the discrete grid coordinates for a sphere can be done in constant time. Each cell is filled with at most a constant number of spheres. Consequently, we get a constant query time for a single cell. Each sphere intersects at most eight cells in all hierarchy levels that have at least the diameter of the sphere.

In the worst case, we have to check $8 \cdot l_{max}$ cells for a single query sphere. However, l_{max} depends only on the *size* of the spheres in the sphere packing, not on the *number* of spheres. More precisely $l_{max} = \log_2(\frac{r_{max}}{r_{min}})$. Hence, we get a



(a) Cow (b) Armadillo (c) Pig

Figure 4: Our test scenes. The triangle count ranges from 20k for the pig up to 1.4 Millions for the armadillos scene. We tested each scene with different sphere resolutions.

constant running time for each sphere. † The overall running time for n spheres is linear.

Obviously, we find all colliding spheres in S_B with at least the same radius for any sphere $s_i \in S_A$. However, there may be spheres in S_B with a smaller radius than a sphere $s_i \in S_A$ that collide with s_i. In order to find also these collisions, we do not only check S_A against S_B, but also the opposite direction, ie, S_B against S_A with the same algorithm. This delivers us the missing colliding sphere pairs automatically. Note that we do not check any pair of spheres twice, because in each direction we test only against spheres with *at least the same radius*.‡ However, this does not change the asymptotic linear running time, even if we have other values for r_{max} and r_{min}.

Summarizing, the use of hierarchical grids allows a constant number of spheres in each grid cell, and a localization in constant time. This guarantees a constant query time for each sphere and, thus, a linear total worst case running time of the whole algorithm. Note that also the construction of the hierarchical grid takes only constant time for each sphere. Consequently, we also get a linear running time for the construction.

† In fact, $\frac{r_{max}}{r_{min}}$ is constant only if the sizes of the spheres do not change during the simulation. Obviously this is true for rigid objects. If we also allow deformable objects, we either can move existing spheres without changing their sizes to fit the deformed object, or we can re-compute or re-size the spheres. Actually, a combination of moving and re-sizing would be more flexible. However, in this case, the ratio between the smallest and the largest sphere may change and $\frac{r_{max}}{r_{min}}$ is not constant anymore. However, the running-time is still independent of the *number* of the spheres and consequently, we still get an *"almost"* constant running-time even in case of deformable objects.

‡ More precisely, we have to replace this test for *"at least the same radius"* by *"strictly larger radius"* in one direction to avoid double checks between spheres with the same radius.

4.2. Parallelization

All operations of the basic collision detection algorithm described above are formulated individually per sphere. In other words, we process each sphere $s_i \in S_A$ independently from all other spheres. Consequently, the algorithm can be trivially parallelized. The basic call for a collision check between two sphere packings can be described as follows:

Algorithm 1: checkCollisions(SpherePacking S_A, SpherePacking S_B)

In parallel forall *spheres* $s_i \in S_A$ **do**
    checkCollisions(s_i, S_B)
In parallel forall *spheres* $s_j \in S_B$ **do**
    checkCollisions(s_j, S_A)

We simply check all spheres $s_i \in S_A$ in parallel against the sphere packing S_B and vice versa. In the kernel, we have to ascend the hierarchy and check all cells that are intersected by s_i in the respective hierarchy level:

Algorithm 2: collisionKernel(Sphere s_i, SpherePacking S_B)

Get hierarchy level l_i for s_i
forall *hierarchy levels:* $l_i \cdots l_{max}$ **do**
    forall *cells* $c_k \cap s_i \neq \varnothing$ **do**
        forall *spheres* $s_j \in c_k$ **do**
            if $r_i \leq r_j$ **then**
                computeOverlapVolume(s_i, s_j)

All observations with respect to the running time from above still hold: each sphere intersects at most eight cells in each level and each cell is occupied by a constant number of spheres from the sphere packing that we want to test. Consequently, we get the same constant running time for each sphere. If we have $O(n)$ processors for the parallel computation, this results in a constant worst case running time for the whole collision detection algorithm.

Note that also the construction of the hierarchical grid can

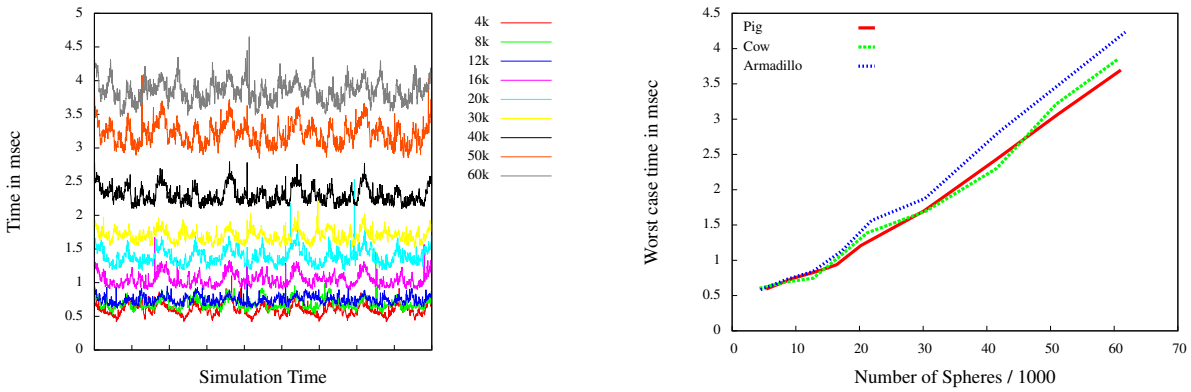


Figure 5: Left: the times for single parallel collision queries over time in the cow scene. We tested this scene for different sphere resolutions, ranging from 4k to 60k spheres. Right: The worst case collision query time for our three test scenes.

be easily parallelized: all spheres can be assigned independently to their cells in the grid hierarchy. Hence, we get a linear parallel construction time for our data structure too.

4.3. Implementation Details

The hierarchical grid is a nice concept to basically prove the theoretic aspects of our algorithm. In practice, uniform grids have serious drawbacks. For instance, we have to maintain a grid of potentially infinite size to cover all cases. Even if we would know the size of the world in advance, uniform grids still have a very huge memory consumption. Moreover, most of the cells in a large grid will probably never be occupied by a sphere. Therefore, we decided to use a more memory efficient data representation than explicitly storing a constant number of uniform grids.

Actually, we can reduce the large memory footprint by using hierarchical hash tables instead, without affecting the constant running time. In detail, we applied the DJB2 hashing function described by Eitz and Lixu [EL07]:

Algorithm 3: calcDJB2HashValue(x, y, z, l)

```
hash = 5381
hash = hash · 33 + x
hash = hash · 33 + y
hash = hash · 33 + z
hash = hash · 33 + l
return hash mod m;
```

where m is the size of the hash table. We used chaining with a fixed size for each bucket. Actually, the size of a bucket can be derived from the constant number of spheres that fits into a cell. This results in the necessity of exclusive writes to the buckets in the hash table for the parallel construction. However, there are at most a constant number of these atomic operations for each bucket. In case of rigid objects, it would be possible to choose a perfect hash function in advance. We decided to use the more flexible DJB2 hashing. In our experiments we did not notice a notable number of hash collisions during the construction. Using hashing instead of explicit grids does not change the running time analysis from the previous sections provided that no unusual amount of hash collisions occur. Hence, we still get a constant running time for the construction as well as for the collision test.

5. Results

We have implemented the massively parallel version of our algorithm prototypically using NVIDIAs CUDA. The testing environment consists of a PC running Windows 7 with a NVIDIA GTX680 graphics card with 4 GByte of memory. All sphere packings were generated with Protosphere [WZ10] that computes a predefined number of spheres for arbitrary objects using a greedy approach. We used a fixed hash table size of 16 MByte. The bucket size was set to 24. Usually, this allows at least a single hash table collision. Actually, it is complicated to determine the exact number of spheres that can intersect a cell in the worst case. Moreover, such a worst case is almost improbable. Therefore, we simply used the kissing number, which is 12 in 3D, as a heuristic. In our experiments, this was sufficient to avoid overflow of the buckets.

We used a simple artificial benchmark scenario: one object is fixed while a rotating copy of it is translated by half of the distance of its bounding box. This results in light to heavy penetrations. In practice, eg, in physically-based simulations, usually only much smaller penetrations are allowed. However, in this benchmark we tried to stress our algorithm.

We tested our algorithm with different objects and differ-

ent densities for the sphere packings. The number of spheres was ranging from 4000 up to 60000 spheres (see Figure 4). This means a filling rate of about 85 – 95% of the object's volumes. Our results show that our algorithm can compute collision checks in about 1 msec with a reasonable accuracy, ie, density of the spheres (see Figure 5). Moreover, we get a constant running-time for all filling rates, almost independent of the object's configuration. We only get small peaks in cases of penetrations, because theses cases require a little more work that non-penetration cases: for instance we have to ascend the hierarchy, compute sphere-sphere overlap volumes, etc. Hence, in case of colliding spheres the query time slightly increases.

However, although if the query time for a specific sphere resolution is constant, the *constant factor* seems to depend linearly on the number of spheres (see Figure 5). Actually, our proof implies a *constant* running time, independent of the sphere resolution. This seems to hold only for up to 20000 spheres. The reason for this observation is relatively simple: our theoretic considerations require $O(n)$ processors. In the real world benchmark, the processors of our graphics card seem to be fully loaded by 20000 spheres. However, this border can be easily increased with upcoming GPU generations with more parallel processors or by simply using more GPUs.

The forces and torques are exactly the same as for the non time-critical version of the *Inner Sphere Trees* [WZ09], because we get exactly the same pairs of intersecting spheres and the same overlap volumes. This guarantees a continuous and noise-free signal for the collision response. Also the approximation error is the same as for the *Inner Sphere Trees*.

Moreover, the construction of the hierarchical grid takes only a fraction of a millisecond (see Figure 6). Basically, this qualifies it for online usage and thus, will enable us to apply it to deformable objects in the future.

5.1. Conclusions and Future Work

We have presented a new massively parallel algorithm that enables collision detection in constant time. Our algorithm does not simply check whether or not two objects collide, but it approximates their penetration volume – the best method to define the extent of interpenetration. This guarantees almost noise-free continuous forces and torques for both direction and magnitude. Collision queries can be answered at rates of 1 kHz, which makes it suitable for haptic rendering. Additionally, the running time is independent of the objects' geometric complexity and their surface representation. Our algorithm allows a simple choice between accuracy and speed. The memory consumption is low compared to other linear time algorithms like VPS and it is very easy to implement. The foundation of our new algorithm is our theoretic worst case analysis. We have shown that the number of intersecting pairs of spheres for two sphere packings is linear in the worst case.

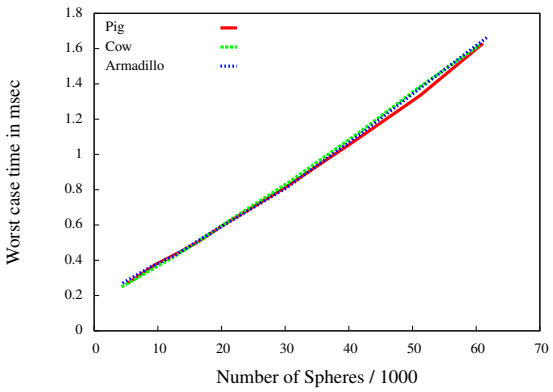


Figure 6: The parallel construction time in the worst case for the hierarchical grid in our scenarios for different sphere resolutions.

Our novel approach opens up several avenues for future work. For instance, our recent implementation is not optimized yet. We are confident that a better, more "hierarchy compatible" hashing function would be able to increase the performance significantly. Moreover, such a hashing function should be locality-preserving in order to better utilize cashing. Probably Morton codes in combination with closed hashing and a reduced cell size could be an option. An interesting question is the analytical determination of exact error bounds. This could lead to an optimal number of inner spheres with well-defined errors.

Finally, our approach is restricted to rigid objects until now. On the other hand, the bounding-volume-less constant time construction of the hierarchical grid makes it perfectly suited for deformable objects. However, a simple rebuild of the sphere packing will be too slow. Therefore, we plan to combine our approach with a new deformation scheme in the future. The so called *Sphere-Spring-Systems* are an extension of the classic mass spring systems. The main difference is the replacement of the dimensionless mass points by volumetric spheres. This approach will also solve the missing volume preservation of mass spring systems automatically.

References

[AFC*10] ALLARD J., FAURE F., COURTECUISSE H., FALIPOU F., DURIEZ C., KRY P. G.: Volume contact constraints at arbitrary resolution. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2010) 29*, 3 (Aug. 2010). URL: http://www.sofa-framework.org/projects/ldi. 3

[ASC*06] ALBOCHER D., SAREL U., CHOI Y.-K., ELBER G., WANG W.: Efficient continuous collision detection for bounding boxes under rational motion. In *ICRA* (2006), IEEE, pp. 3017–3022. URL: http://dblp.uni-trier.de/db/conf/icra/icra2006.html. 2

[BEW*98] BISHOP L., EBERLY D., WHITTED T., FINCH M., SHANTZ M.: Designing a pc game engine. *IEEE Computer Graphics and Applications 18*, 1 (1998), 46–53. 1

[BL11] BELOTTI P., LIBERTI L.: The kissing number problem, Modification of: 18:07:41, August 19 2011. Available from CyberInfrastructure for MINLP, a collaboration of Carnegie Mellon University and IBM Research. 4

[BV91] BOUMA W. J., VANECEK G. J.: Collision detection and analysis in a physically based simulation. In *In Eurographics Workshop on Animation and Simulation* (1991), pp. 191–203. 1

[CG98] CHAKRAVARTHY A., GHOSE D.: Obstacle avoidance in a dynamic environment: a collision cone approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part A 28*, 5 (1998), 562–574. 1

[CRM02] CASELLI S., REGGIANI M., MAZZOLI M.: Exploiting advanced collision detection libraries in a probabilistic motion planner. In *WSCG* (2002), pp. 103–110. 3

[DK85] DOBKIN D. P., KIRKPATRICK D. G.: A linear algorithm for determining the separation of convex polyhedra. *J. Algorithms 6*, 3 (1985), 381–392. 3

[EL01] EHMANN S. A., LIN M. C.: Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of EUROGRAPHICS 2001) 20*, 3 (2001), 500–510. 2

[EL07] EITZ M., LIXU G.: Hierarchical spatial hashing for real-time collision detection. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications 2007* (Washington, DC, USA, 2007), SMI '07, IEEE Computer Society, pp. 61–70. URL: http://dx.doi.org/10.1109/SMI.2007.18, doi:10.1109/SMI.2007.18. 7

[ES99] ECKSTEIN J., SCHÖMER E.: Dynamic collision detection in virtual reality applications. In *WSCG'99 Conference Proceedings* (1999), Skala V., (Ed.). URL: citeseer.ist.psu.edu/eckstein99dynamic.html. 1

[FBAF08] FAURE F., BARBIER S., ALLARD J., FALIPOU F.: Image-based collision detection and response between arbitrary volumetric objects. In *ACM Siggraph/Eurographics Symposium on Computer Animation, SCA 2008, July, 2008* (Dublin, Irlande, July 2008). 3

[FL01] FISHER S., LIN M.: Fast penetration depth estimation for elastic bodies using deformed distance fields. In *Proc. International Conf. on Intelligent Robots and Systems (IROS)* (2001), pp. 330–336. 2

[GGK06] GRESS A., GUTHE M., KLEIN R.: Gpu-based collision detection for deformable parameterized surfaces. *Computer Graphics Forum 25*, 3 (Sept. 2006), 497–506. 2

[GKJ*05a] GOVINDARAJU N. K., KNOTT D., JAIN N., KABUL I., TAMSTORF R., GAYLE R., LIN M. C., MANOCHA D.: Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans. Graph. 24*, 3 (2005), 991–999. URL: http://dblp.uni-trier.de/db/journals/tog/tog24.html#GovindarajuKJKTGLM05. 2

[GKJ*05b] GOVINDARAJU N. K., KNOTT D., JAIN N., KABUL I., TAMSTORF R., GAYLE R., LIN M. C., MANOCHA D.: Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans. Graph. 24*, 3 (2005), 991–999. 3

[GLM96] GOTTSCHALK S., LIN M. C., MANOCHA D.: Obbtree: a hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 171–180. URL: http://doi.acm.org/10.1145/237170.237244, doi:10.1145/237170.237244. 2

[HTG04] HEIDELBERGER B., TESCHNER M., GROSS M.: Detection of collisions and self-collisions using image-space techniques. In *Proceedings of the 12-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2004 (WSCG'2004)* (University of West Bohemia, Czech Republic, Feb. 2004), pp. 145–152. 2

[Hub96] HUBBARD P. M.: Approximating polyhedra with spheres for time-critical collision detection. *ACM Trans. Graph. 15*, 3 (1996), 179–210. 2

[IEH*05] ILUSHIN O., ELBER G., HALPERIN D., WEIN R., KIM M.-S.: Precise global collision detection in multi-axis nc-machining. *Comput. Aided Des. 37*, 9 (Aug. 2005), 909–920. URL: http://dx.doi.org/10.1016/j.cad.2004.09.018, doi:10.1016/j.cad.2004.09.018. 1

[KHM*98] KLOSOWSKI J. T., HELD M., MITCHELL J. S. B., SOWIZRAL H., ZIKAN K.: Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (Jan. 1998), 21–36. URL: http://dx.doi.org/10.1109/2945.675649, doi:10.1109/2945.675649. 2

[Kip07] KIPFER P.: LCP algorithms for collision detection using CUDA. In *GPUGems 3* (2007), Nguyen H., (Ed.), Addison-Wesley, pp. 723–739. 3

[KKLL95] KIM H. S., KO H., LEE K., LEE C.-W.: A collision detection method for real time assembly simulation. *Assembly and Task Planning, IEEE International Symposium on 0* (1995), 0387. doi:http://doi.ieeecomputersociety.org/10.1109/ISATP.1995.518799. 1

[KLRS04] KOLB A., LATTA L., REZK-SALAMA C.: Hardware-based simulation and collision detection for large particle systems. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (New York, NY, USA, 2004), HWWS '04, ACM, pp. 123–131. URL: http://doi.acm.org/10.1145/1058129.1058147, doi:10.1145/1058129.1058147. 3

[KNK*02] KUFFNER J., NISHIWAKI K., KAGAMI S., KUNIYOSHI Y., INABA M., INOUE H.: Self-collision detection and prevention for humanoid robots. In *in Proceedings of the IEEE International Conference on Robotics and Automation* (2002), pp. 2265–2270. 1

[KP03] KNOTT D., PAI D.: Cinder: Collision and interference detection in real–time using graphics hardware, 2003. URL: citeseer.ist.psu.edu/knott03cinder.html. 2

[LaV04] LAVALLE S. M.: Planning algorithms, 2004. 1

[LMM10] LAUTERBACH C., MO Q., MANOCHA D.: gproximity: Hierarchical gpu-based operations for collision and distance queries. *Comput. Graph. Forum 29*, 2 (2010), 419–428. URL: http://dblp.uni-trier.de/db/journals/cgf/cgf29.html#LauterbachMM10. 2

[MPT99] MCNEELY W. A., PUTERBAUGH K. D., TROY J. J.: Six degrees-of-freedom haptic rendering using voxel sampling. *ACM Transactions on Graphics (SIGGRAPH 1999) 18*, 3 (1999), 401–408. 3

[MPT05] MCNEELY W. A., PUTERBAUGH K. D., TROY J. J.: Advances in voxel-based 6-dof haptic rendering. In *ACM SIGGRAPH 2005 Courses* (New York, NY, USA, 2005), SIGGRAPH '05, ACM. URL: http://doi.acm.org/10.1145/1198555.1198606, doi:10.1145/1198555.1198606. 3

[MPT06] MCNEELY W. A., PUTERBAUGH K. D., TROY J. J.: Voxel-based 6-dof haptic rendering improvements. *Hapticse The Electronic Journal of Haptics Research 3*, 7 (2006). 3

[MRS08] MORVAN T., REIMERS M., SAMSET E.: High performance gpu-based proximity queries using distance fields. In *Computer graphics forum* (2008), vol. 27, Wiley Online Library, pp. 2040–2052. 3

[MZ13] MAINZER D., ZACHMANN G.: Collision detection based on fuzzy scene subdivision. In *Symposium on GPU Computing and Applications* (Singapore, 2013). 3

[OL03] OTADUY M. A., LIN M. C.: CLODs: Dual hierarchies for multiresolution collision detection. In *Symposium on Geometry Processing* (2003), pp. 94–101. 3

[PH05] PRIOR A., HAINES K.: The use of a proximity agent in a collaborative virtual environment with 6 degrees-of-freedom voxel-based haptic rendering. In *Proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems* (Washington, DC, USA, 2005), WHC '05, IEEE Computer Society, pp. 631–632. URL: http://dx.doi.org/10.1109/WHC.2005.137, doi:10.1109/WHC.2005.137. 3

[PM12] PAN J., MANOCHA D.: Gpu-based parallel collision detection for fast motion planning. *Int. J. Rob. Res. 31*, 2 (Feb. 2012), 187–200. URL: http://dx.doi.org/10.1177/0278364911429335, doi:10.1177/0278364911429335. 3

[PML95] PONAMGI M., MANOCHA D., LIN M. C.: Incremental algorithms for collision detection between solid models. In *Proceedings of the third ACM symposium on Solid modeling and applications* (New York, NY, USA, 1995), SMA '95, ACM, pp. 293–304. URL: http://doi.acm.org/10.1145/218013.218076, doi:10.1145/218013.218076. 2

[Qui94] QUINLAN S.: Efficient distance computation between non-convex objects. In *In Proceedings of International Conference on Robotics and Automation* (1994), pp. 3324–3329. 2

[RPP*01] RENZ M., PREUSCHE C., POTKE M., KRIEGEL H.-P., HIRZINGER G.: Stable haptic interaction with virtual environments using an adapted voxmap-pointshell algorithm. In *In Proc. Eurohaptics* (2001), pp. 149–154. 3

[SHH98] SURI S., HUBBARD P. M., HUGHES J. F.: Collision detection in aspect and scale bounded polyhedra. In *SODA* (1998), pp. 127–136. 3

[ST95] SCHÖMER E., THIEL C.: Efficient collision detection for moving polyhedra. In *11th Annual Symposium on Computational Geometry* (June 1995), pp. 51–60. 3

[ST96] SCHÖMER E., THIEL C.: Subquadratic algorithms for the general collision detection problem. In *12th European Workshop on Computational Geometry* (March 1996), pp. 95–101. 3

[Tur89] TURK G.: *Interactive collision detection for molecular graphics*. Tech. rep., University of North Carolina at Chapel Hill, 1989. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.4927. 1

[TWZ07] TRENKEL S., WELLER R., ZACHMANN G.: A benchmarking suite for static collision detection algorithms. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)* (Plzen, Czech Republic, 29 January–1 February 2007), Skala V., (Ed.), Union Agency. URL: http://cg.in.tu-clausthal.de/research/colldet_benchmark. 3

[VCC98] VEMURI B. C., CAO Y., CHEN L.: Fast collision detection algorithms with applications to particle flow. *Computer Graphics Forum 17*, 2 (1998), 121–134. 3

[vdB98] VAN DEN BERGEN G.: Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools 2*, 4 (Jan. 1998), 1–13. URL: http://dl.acm.org/citation.cfm?id=763345.763346. 2

[WKZ06] WELLER R., KLEIN J., ZACHMANN G.: A model for the expected running time of collision detection using aabb trees. In *Eurographics Symposium on Virtual Environments (EGVE)* (Lisbon, Portugal, May 8–10 2006), Hubbold R., Lin M., (Eds.), Eurographics Association, pp. 11–17. URL: http://www.gabrielzachmann.org/. 3

[WMS*10] WELLER R., MAINZER D., SAGARDIA M., HULIN T., ZACHMANN G., PREUSCHE C.: A benchmarking suite for 6-dof real time collision response algorithms. In *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology (VRST)* (New York, NY, USA, Nov. 2010), ACM, pp. 63–70. URL: http://cg.in.tu-clausthal.de/publications.shtml#vrst2010, doi:http://doi.acm.org/10.1145/1889863.1889874. 3

[WZ09] WELLER R., ZACHMANN G.: Inner sphere trees for proximity and penetration queries. In *Robotics: Science and Systems (RSS)* (28 June–1 July 2009). URL: http://cg.in.tu-clausthal.de/. 2, 8

[WZ10] WELLER R., ZACHMANN G.: Protosphere: A gpu-assisted prototype guided sphere packing algorithm for arbitrary objects. In *ACM SIGGRAPH ASIA 2010 Sketches* (New York, NY, USA, Dec. 2010), ACM, pp. 8:1–8:2. URL: http://cg.in.tu-clausthal.de/research/protosphere, doi:http://doi.acm.org/10.1145/1899950.1899958. 7

[Zac98] ZACHMANN G.: Rapid collision detection by dynamically aligned dop-trees. In *Proceedings of the Virtual Reality Annual International Symposium* (Washington, DC, USA, 1998), VRAIS '98, IEEE Computer Society, pp. 90–. URL: http://dl.acm.org/citation.cfm?id=522258.836122. 2

[Zac02] ZACHMANN G.: Minimal hierarchical collision detection. In *Proceedings of the ACM symposium on Virtual reality software and technology* (New York, NY, USA, 2002), VRST '02, ACM, pp. 121–128. URL: http://doi.acm.org/10.1145/585740.585761, doi:10.1145/585740.585761. 2

[ZK07] ZHANG X., KIM Y. J.: Interactive collision detection for deformable models using streaming aabbs. *IEEE Transactions on Visualization and Computer Graphics 13*, 2 (Mar. 2007), 318–329. URL: http://dx.doi.org/10.1109/TVCG.2007.42, doi:10.1109/TVCG.2007.42. 2