TU Clausthal

Clausthal University of Technology

Inner Sphere Trees

Rene Weller and Gabriel Zachmann

IfI Technical Report Series **IfI-08-09**

Department of Informatics
Clausthal University of Technology

Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany
Editor of the series: Jürgen Dix
Technical editor: Wojciech Jamroga
Contact: wjamroga@in.tu-clausthal.de
URL: http://www.in.tu-clausthal.de/forschung/technical-reports/
ISSN: 1860-8477

The IfI Review Board

Inner Sphere Trees

Rene Weller and Gabriel Zachmann

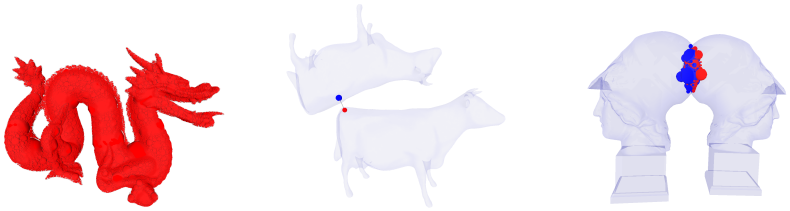

Figure 1: *Our* Inner Sphere Trees *are based on sphere packings of arbitrary polygonal objects (left). They are suitable for different kinds of geometric queries, namely proximity queries (middle) and our new method for computing the penetration depth, the* penetration volume *(right)*

Abstract

We present a novel geometric data structure for collision detection at haptic rates between arbitrary rigid objects. Our data structure, which we call *inner sphere trees*, efficiently supports both proximity queries and the *penetration volume*. The latter is related to the water displacement of the overlapping region and, thus, corresponds to a physically motivated force. Moreover, we present a time-critical version of the penetration volume computation that is able to achieve very good estimates of the penetration volume within a fixed budget of query time. In order to build our new hierarchy, we propose to use an AI clustering algorithm, which we extend and adapt here. The results show extremely good performance of our algorithms both for proximity and penetration volume queries for models consisting of hundreds of thousands of polygons. We also conducted a comparison with an existing package, which shows that our algorithms are about an order of magnitude faster.

1 Introduction

Collision detection between rigid objects plays an important role in many fields of computer science, e.g. in physically-based simulations, robotics, and medical applications. Today, there exist a wide variety of freely available collision detection libraries, (like PQP [Larsen et al., 1999], FreeSolid [van den Bergen, 1999], Opcode [Terdiman, 2001] or CollDet [Zachmann, 1998]) and nearly all of them are able to work at interactive rates, even for very complex objects [Trenkel et al., 2007]. Most collision detection algorithms dealing with rigid objects use the very efficient data

structure of bounding volume hierarchies (BVH). The main idea behind a BVH is to subdivide the primitives of an object hierarchically until there is only one single primitive left at the leaf level. Several kinds of bounding volumes have been proposed in the past, the most popular are axis aligned bounding boxes (AABBs), oriented bounding boxes(OBBs)[Gottschalk et al., 1996], oriented polytopes (k-Dops) [Klosowski et al., 1998] and spheres. BVHs guarantee very fast responses at query time, as long as no further information than the set of intersecting polygons is required for the collision response. However, most applications require much more information in order to compute a proper force.

One way to do this is to compute the exact time of contact for the objects. This method is called continuous collision detection. Algorithms for this kind of collision detection are very time-consuming. Another approach, called penalty methods, is to compute repelling forces based on the penetration depth. However, there does not exist a universally accepted definition of the penetration depth between a pair of polygonal models [Zhang et al., 2007, Hong et al., 2000]. Mostly, the minimum translation vector to separate the objects is used, but this may lead to discontinuous forces.

Another approach is to avoid penetrations or contacts before they really happen. In this case, the minimum distance between the objects can be used to compute repelling forces. However, it can be difficult for the simulation to guarantee that the objects never penetrate each other.

These problems get worse when the collision detection is not only needed to avoid visual artifacts, but the algorithm is used in a haptic environment. Hardware for haptic interaction requires update rates of at least 200 Hz up to 1 kHz to guarantee a stable force feedback.

A remedy could be the use of constant time collision detection algorithms, for example, voxel-based methods like the Voxmap Pointshell algorithm (VPS). In this approach, the world is divided into a static environment that is approximated by voxels, and dynamic objects that are approximated by point clouds and are allowed to move freely. It is independent of the objects' complexity and fast enough for haptic rendering. However, it has to be known in advance which objects are fixed and which are moving. Moreover, collision detection is only available with the static environment, not between pairs of moving objects. Furthermore, the method is very memory consuming and produces strong aliasing artifacts due to the voxelization errors.

1.1 Main Contributions

Our new geometric data structure, the *Inner Sphere Trees (IST)*, combines the advantages and avoids most of the disadvantages of both prior approaches.

The main idea is that we do not build a hierarchy based on the polygons on the boundary of an object. Instead, we fill the interior of the model with a set of non-overlapping simple volumes that approximate the object's volume closely. In our implementation, we used spheres for the sake of simplicity, but the idea of using bounding volumes for lower bounds instead of upper bounds can be extended easily to all kinds of volumes. On top of these inner bounding volumes, we build a hierarchy that allows very

fast collision detection and proximity queries. Moreover, it enables us to estimate the *penetration volume* very efficiently. The penetration volume corresponds to the water displacement of the overlapping parts of the objects and leads to physically motivated and continuous repelling forces, in contrast to the discontinuous forces of voxel- or penetration depth-based algorithms. It is "the most complicated yet accurate method" to define the extent of intersection [Fisher and Lin, 2001, Sec. 5.1], which was also reported earlier by [O'Brien and Hodgins, 1999, Sec. 3.3]. However, to our knowledge, there are no algorithms to compute it efficiently as yet.

The construction of our data structure consists of two main tasks. The first task is to fill an arbitrary object densely with a set of spheres. Our data structure supports all kinds of objects, e.g. polygon meshes or NURBS surfaces. The only precondition is that they be watertight. We use an extended version of a flood filling voxelization in combination with a new sphere creation algorithm for this part. The second task is to build a hierarchy upon these inner spheres. Therefore, we utilize the recently proposed batch neural gas clustering algorithm, which allows us to work in an adaptive manner.

Our inner sphere tree not only allows to compute both separation distance and penetration volume, but it also lends itself very well to time-critical variants, which run only for a pre-defined time budget.

The results shows that our new data structure outperforms state-of-the-art libraries by more than an order of magnitude, with a negligible loss of accuracy.

2 Previous Work

Collision detection has been extensively investigated by researchers in the past decades. There exist a large variety of freely available libraries for collision detection queries. However, the number of libraries that also support the computation of proximity queries or the penetration depth is manageable. Additionally, most of them are not designed to work at haptic refresh rates, or they are restricted to simple point probes [Hudson et al., 1997], or require special objects, such as convex objects as input.

In the following, we will give a short overview of classical and also state of the art approaches to manage these tasks.

2.1 BVH based data structures

[Johnson and Cohen, 1998] present a generalized framework for minimum distance computations that depends on geometric reasoning and includes time-critical properties. The PQP library [Larsen et al., 1999] uses swept sphere volumes as BVs in combination with several speed-up techniques for fast proximity queries. We used it in this paper to compare it with our new data structure. Sphere trees have also been used for distance computation [Quinlan, 1994, Hubbard, 1995, Mendoza and O'Sullivan, 2006]. The algorithm presented there is interruptible and it is able to deliver approximative distances. However, it is not applicable to penetration depth estimation.

[Johnson and Willemsen, 2003] computes local minimum distances for a stable force feedback computation and uses spatialized normal cone pruning for the collision detection. Another classical algorithm for proximity queries is the GJK [Gilbert et al., 1988, van den Bergen, 1999], which computes the distance between a pair of convex objects, by utilizing the Minkowski sum of the two objects. There also exist extensions to the GJK algorithms that allow to measure the penetration depth [Cameron, 1997].

[Zhang et al., 2007] presented an extended definition of the penetration depth that also takes the rotational component into account. This is called the generalized penetration depth. However, the algorithm is not fast enough for haptic interaction rates.

[Redon and Lin, 2006] approximate a local penetration depth by first computing a local penetration direction and then use this information to estimate a local penetration depth on the GPU. The algorithm is restricted to image-precision. Other GPU approaches with very similar problems have been presented by [Kim et al., 2003, Kim et al., 2002] or [Hoff et al., 2002], that also supports proximity queries in image resolution.

The DEEP library [Kim et al., 2004] finds a "locally optimal solution" by walking on the surface of the Minkowski sums and uses a heuristic to estimate the initial features. However, it is restricted to convex polytopes.

Another interesting approach to perform penetration depth computations is the use of continuous collision detection. Unfortunately, most continuous collision detection algorithms are far from real time, not to speak about haptic rendering. So, [Ortega et al., 2007] use a god-object based approach, that computes the collisions and the forces asynchronous in two different processes in order to guarantee constant updating rates. However, between the queries, it does not guarantee collision free configurations.

2.2 Voxel based data structures

The Voxmap Pointshell approach [McNeely et al., 1999] divides 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. The dynamic object is described by a set of points that represent 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.

[Renz et al., 2001] presented extensions to the classic VPS, including optimizations to force calculation in order to increase its stability. However, even these optimizations can not completely avoid the limits of VPS, namely aliasing effects, the huge memory consumption and the strict disjunction between dynamic and static objects.

Closely related to VPS are distance field based methods. [Barbič and James, 2008] use a pointshell of reduced deformable models in combination with distance fields in order to guarantee continuous contact forces.

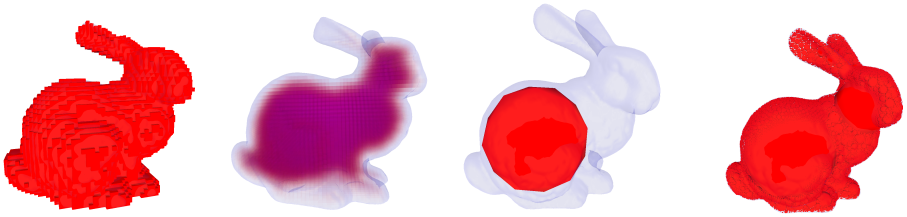

Figure 2: *The stages of our sphere filling method. First, we voxelize the object (left). Additionally, we compute distances from the voxels to the closest triangle. The resulting distance field is visualized in the second image (for the sake of visualization, the transparency is modulated by the distance). Then, we pick the voxel with the largest distance and put a sphere at its center, with its radius set to the distance. We proceed incrementally and, eventually, obtain a dense sphere filling (right).*

3 Creation of the Inner Sphere Tree Hierarchy

In this section we describe the construction of our data structure. The goal is to fill an arbitrary object densely with a set of disjoint (i.e. non-overlapping) spheres such that the volume of the object is covered well while the number of spheres is as small as possible. In a second step, we build a hierarchy over this set of spheres. We chose spheres for volumes, because they offer a trivial and very fast overlap test. Moreover, they are rotationally invariant, and it is easy, in contrast to AABBs or OBBs, to compute the exact intersection volume.

3.1 Let there be Spheres

Filling objects densely with spheres is a highly non-trivial task. Bin packing, even when restricted to spheres, is still a very active field in geometric optimization and far away from being solved for general objects [Birgin and Sobral, 2008, Schuermann, 2006]. In our implementation of the inner sphere trees, we use a simple heuristic that offers a good trade-off between accuracy and speed in practice.

Currently, we voxelize the object as an intermediate step (by a simple flood filling algorithm). But instead of simply storing whether or not a voxel is filled, we additionally store the distance d from the center of the voxel to the nearest point on the surface, as well as the triangle that realizes this distance.

After the voxelization, we generate the inner spheres greedily. We choose the voxel V^* with the largest distance d^* to the surface. We create an inner sphere with radius d^* and centered on the center of V^*. All voxels whose center is contained in this sphere will not be considered any further. Additionally, we have to update all voxels V_i with $d_i > d^*$ and distance $d(V_i, V^*) < 2d$; their d_i must now be set to the new free radius. This is, because they are now closer to the sphere around V^* than to a triangle on the hull (see Figure 3). This process stops, when there is no voxel left.

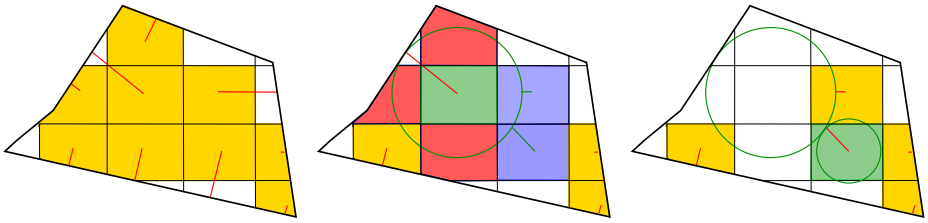

Figure 3: *The first steps of the creation of the inner spheres. Left: voxelized distance field of the object. Middle: we place a maximal sphere (green) at the voxel center with the largest radius; then, the red voxels are deleted, and the free radius of some voxels (blue) is updated. Right: this procedure continues greedily.*

After these steps, the object is filled densely with a set of non-overlapping spheres. The density can be controlled by the number of voxels.

3.2 Building the IST

Our sphere hierarchy is based on the notion of a *wrapped hierarchy* [Agarwal et al., 2004]. In a wrapped hierarchy, a sphere at an inner node is a tight bounding sphere for all its associated leaves, but it does not necessarily bound its direct children (see Figure 4). Compared to layered hierarchies, this has the big advantage that it fits the object more tightly. We use a top-down approach to create our hierarchy, i.e., we start at the root node that covers all inner spheres and divide these into several subsets.

The partitioning of the inner spheres has significant influence on the performance during runtime. Partitioning algorithms for ordinary sphere trees, like the medial axis approach [Bradshaw and O'Sullivan, 2004] work well if the spheres overlap heavily, but in our scenario we use disjoint inner spheres. Other approaches based on the *k-center* problem work only for sets of points and do not support spheres.

So, we decided to use the *batch neural gas* clustering algorithm (BNG) known from artificial intelligence [Cottrell et al., 2006]. BNG is a very robust clustering algorithm, which can be derived as stochastic gradient descent with a cost function closely connected to quantization error. Like *k-means*, the cost function minimizes the mean squared euclidean distance of each data point to its nearest center. But unlike k-means, BNG exhibits very robust behavior with respect to the initial prototype positions: they can be chosen arbitrarily without affecting the convergence. Moreover, BNG can be extended to allow the specification of the importance of each data point, which will be important to create efficient inner spheres trees.

In the following, we will give a quick recap of the basic batch neural gas and then describe our extensions and application to building the inner sphere tree.

Given points $x_j \in \mathbb{R}^d, j = 0, \ldots, m$ and prototypes $w_i \in \mathbb{R}^d, i = 0, \ldots, n$ initialized randomly, we set the rank for every prototype w_i with respect to every data point x_j as

$$k_{ij} := |\{w_k : d(x_j, w_k) < d(x_j, w_i)\}| \in \{0, \ldots, n\} \tag{1}$$

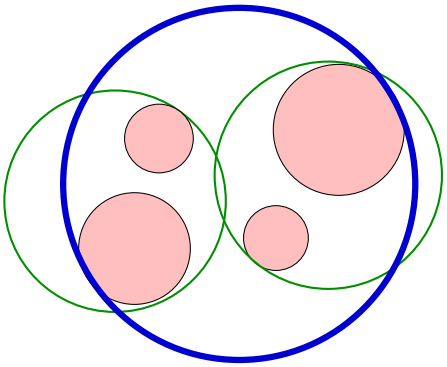

Figure 4: *In a wrapped hierarchy, the blue sphere bounds all its leaf nodes (red spheres), but not its direct children (green spheres).*

In other words, we sort the prototypes with respect to every data point. After the computation of the ranks, we compute the new positions for the prototypes:

$$w_i := \frac{\sum_{j=0}^{m} h_\lambda(k_{ij}) x_j}{\sum_{j=0}^{m} h_\lambda(k_{ij})} \tag{2}$$

These two steps are repeated until a stop criterion is met. In the original paper, a fixed number of iterations is proposed. We propose to use an adaptive version and stop the iteration if the movement of the prototypes is smaller than some ϵ. In our examples, we chose $\epsilon \approx 10^{-5} \times \mathrm{BoundingBoxSize}$, without any differences in the hierarchy compared to the non-adaptive, exhaustive approach. This improvement speeds up the creation of the hierarchy significantly.

The convergence rate is controlled by a monotonically decreasing function $h_\lambda(k) > 0$ that decreases with the number of iterations t. We use the function proposed in the original paper: $h_\lambda(k) = e^{-\frac{k}{\lambda}}$ with initial value $\lambda_0 = \frac{n}{2}$, and reduction $\lambda(t) = \lambda_0 \left(\frac{0.01}{\lambda_0}\right)^{\frac{t}{t_{\max}}}$, where $t_{\max}$ is the maximum number of iterations. These values have been taken according to [Martinetz et al., 1993].

Obviously, the number of prototypes defines the arity of the tree. Experiments with our data structure have shown that a branching factor of 4 produces the best results. Moreover, this has the advantage that we can use the full capacity of SIMD units in modern CPUs.

So far, the BNG only utilizes the location of the centers of the spheres. In our experience, this already produces much better results than other, simpler heuristics, such as greedily choosing the biggest spheres or the spheres with the largest number of neighbors. However, it does not yet take the extent of the spheres into account. As a consequence, the prototypes tend to avoid regions that are covered with a very large sphere, i.e., centers of big spheres are treated as outliers and they are thus placed on very deep levels in the hierarchy. However, it is better to place big spheres at higher

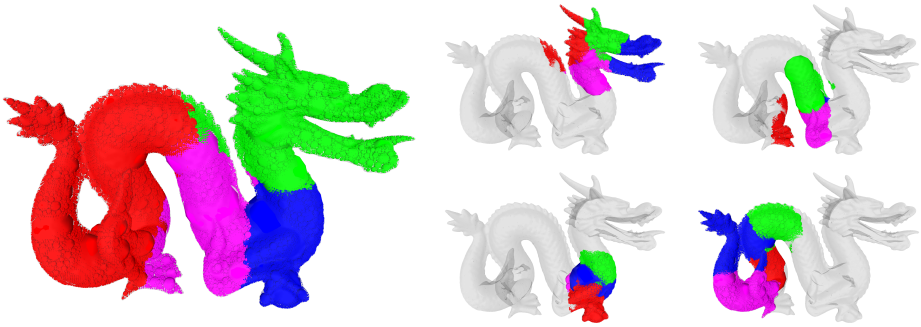

Figure 5: *This figure shows the results of our hierarchy building algorithm based in batch neural gas clustering with magnification control. All of those inner spheres that share the same color are assigned to the same bounding sphere. The left image shows the clustering result of the root sphere, the right images the partitioning of its four children.*

levels of the hierarchy in order to get early lower bounds during distance traversal (see Section 4.1 for details).

Therefore, we use an extended version of the classical batch neural gas, that also takes the size of the spheres into account. Our extension is based on an idea of [Hammer et al., 2006], where *magnification control* is introduced. The idea is to add weighting factors in order to "artificially" increase the density of the space in some areas.

With weighting factors $v(x_j)$, Eq. 2 becomes

$$w_i := \frac{\sum_{j=0}^{m} h_\lambda(k_{ij}) v(x_j) x_j}{\sum_{j=0}^{m} h_\lambda(k_{ij}) v(x_j)} \tag{3}$$

In our scenario, we already know the density, because our spheres are disjoint. Thus, we can directly use the volumes of our spheres to let $v(x_j) = \frac{4}{3}\pi r^3$.

Summing up the hierarchy creation algorithm: we first compute a bounding sphere for all inner spheres (at the leaves), which becomes the root node of the hierarchy. To do that, we use the fast and stable smallest enclosing sphere algorithm proposed in [Gärtner, 1999]. Then, we divide the set of inner spheres into subsets in order to create the children. To do that, we use the extended version of batch neural gas with magnification control. We repeat this scheme recursively.

In the following, we will call the spheres in the hierarchy that are not leaves *hierarchy spheres*. Spheres at the leaves, which were created in Section 3.1, will be called *inner spheres*. Note that hierarchy spheres are not necessarily contained completely within the object.

Algorithm 1: checkDistance(A, B, minDist)

input : A, B = spheres in the inner sphere tree
in/out: minDist = overall minimum distance seen so far
if A *and* B *are leaves* **then**
    // end of recursion
    minDist $= \min\{\text{distance}(\text{A}, \text{B}), \text{minDist}\}$
else
    // recursion step
    forall *children* a[i] *of* A **do**
        forall *children* b[j] *of* B **do**
            if $\text{distance}(\text{a[i]}, \text{b[j]}) < \text{minDist}$ **then**
                checkDistance(a[i], b[j], minDist)

4 BVH Traversal

During runtime, our new data structure supports different kinds of queries, namely proximity queries, which report the separation distance between a pair of objects, and penetration volume queries, which report the common volume covered by both objects. As a by-product, the proximity query can return a witness realizing the distance, and the penetration algorithm can return a partial list of intersecting polygons.

In the following, we describe algorithms for these two query types, but it should be obvious how they can be modified in order to provide an approximate yes-no answer. This would further increase the speed.

First, we will discuss the two query types separately, in order to point out their specific requirements and optimizations. Then, we explain how they can be combined into a single algorithm.

4.1 Proximity Queries

Our algorithm for proximity queries works like most other classical BVH traversal algorithms. We simply have to add the computation of lower bounds for the distance. If a pair of leaves, which are the inner spheres, is reached, we update the lower bound so far (see Algorithm 1). During traversal, there is no need to visit bounding volumes in the hierarchy that are farther away than the current minimum distance, because of the bounding property. This guarantees a high culling efficiency.

4.1.1 Improving runtime

In most collision detection scenarios, there is a high spatial and temporal coherence, especially when rendering at haptic rates. Thus, in most cases, those spheres realizing the minimum distance in a frame are also the closest spheres in the next frames, or they are at least in the neighborhood. Thus, in our implementation we store pointers to

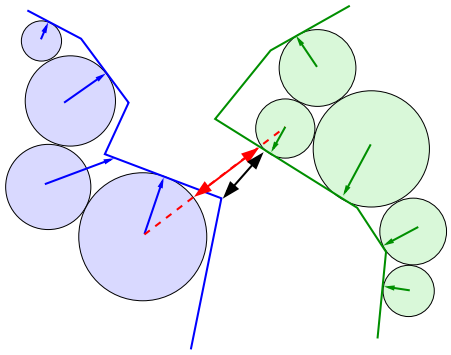

Figure 6: *Every inner sphere (light green/blue) is attached to its closest triangle (dark green/blue). The distance between the closest spheres (red line) is an upper bound for the real distance between the closest triangles (black line)*

the closest spheres as of the last frame and use their distance to initialize minDist in Algorithm 1.

For the moment, we are only interested in proximity queries. Therefore, we can obtain a further speedup by terminating the traversal when the first pair of intersecting inner spheres is found; in that case, the distance is zero.

Moreover, our traversal algorithm is very well suited for parallelization. During recursion, we compute the distances between 4 pairs of spheres in one single SIMD implementation, which is greatly facilitated by our hierarchy being a 4-ary tree.

4.1.2 Improving accuracy

Obviously, Algorithm 1 returns only an approximate minimum distance, because it utilizes only the distances of the inner spheres for the proximity query. Thus, the accuracy depends on their density.

Fortunately, it is very easy to alleviate these inaccuracies by simply assigning the closest triangle (or a set of triangles) to each inner sphere. After determining the closest spheres with Algorithm 1, we add a subsequent test that calculates the exact distance between the triangles assigned to those spheres (see Figure 6). This simple heuristic reduces the error significantly even with relatively sparsely filled objects, and it does not affect the runtime (see first diagram in Figure 13).

4.2 Penetration Volume Queries

In addition to proximity queries, our data structure also supports a new kind of penetration query, namely the *penetration volume*. This is the volume of the intersection of the two objects. In other words, it corresponds to the water displacement, which

Algorithm 2: checkVolume(A, B, totalOverlap)

```
input : A, B = spheres in the inner sphere tree
in/out: totalOverlap = overall volume of intersection
if A and B are leaves then
    // end of recursion
    totalOverlap += overlapVolume( A, B )
else
    // recursion step
    forall children a[i] of A do
        forall children b[j] of B do
            if overlap(a[i], b[j]) > 0 then
                checkVolume( a[i], b[j], totalOverlap )
```

means that it is directly related to a physically motivated measurement to compute the repelling forces.

Obviously, the algorithm to compute the penetration volume (see Algorithm 2) does not differ very much from the proximity query test: we simply have to replace the distance test by an overlap test and maintain an accumulated overlap volume during the traversal.

4.2.1 Filling the gaps

Filling the objects as described in Section 3.1 results in densly filled objects. However, there still remain small gaps between the spheres that can not be completely compensated by increasing the number voxels.

As a remedy, we assign an additional, secondary radius to each inner sphere, such that the volume of the secondary sphere is equal to the volume of all voxels whose centers are contained within the radius of the primary sphere. This guarantees that the total volume of all secondary spheres equals the volume of the object, within the accuracy of the voxelization, because each voxel volume is accounted for exactly once.

Certainly, these secondary spheres may slightly overlap, but this simple heuristic leads to acceptable estimations of the penetration volume. (Note, however, that the secondary spheres are not necessarily larger than the primary spheres.)

4.2.2 Improvements

Similar to the proximity query implementation, we can utilize SIMD parallelization to speed up both, the simple overlap check and the volume accumulation.

Furthermore, we can exploit the observation that a recursion can be terminated if a hierarchy sphere (i.e., an inner node of the sphere hierarchy) is completely contained inside an inner sphere (leaf). In this case, we can simply add the total volume of all of

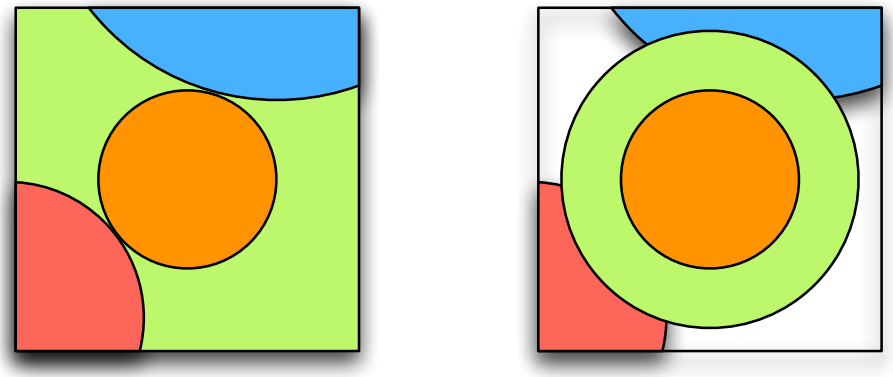

Figure 7: *After the sphere filling algorithm, some voxel are intersected by several non-overlapping spheres (yellow, red, and blue). On the left, the green, hatched volume remains unaccounted for. As a remedy, we blow up the sphere centered at the voxel by the volume of the gap (right).*

its leaves to the accumulated penetration volume. In order to do this quickly, we store the total volume

$$\mathrm{Vol}_l(S) = \sum_{S_j \in \mathrm{Leaves}(S)} \mathrm{Vol}(S_j), \tag{4}$$

where S_j are all inner spheres below S in the BVH. This can be done in a preprocessing step during hierarchy creation.

4.2.3 Time-critical computation of penetration volume

In most cases, a penetration volume query has to visit many more nodes than the average proximity query. Consequently, the runtimes are slower on average, especially in cases with heavy overlaps. Furthermore, Algorithm 2 only provides a lower bound on the penetration volume.

In the following, we will describe a variation of our algorithm for penetration volume queries that guarantees a predefined query time budget while providing an answer "as good as possible". This is essential for time-critical applications such as haptic rendering,

An appropriate strategy to realize time-critical traversals is to guide the traversal by a priority queue Q. Given a pair of hierarchy spheres S and R, a simple heuristic is to use $\mathrm{Vol}(S \cap R)$ for the priority in Q. In our experience, this would yield acceptable upper bounds.

Unfortunately, this simple heuristic also leads to very bad lower bounds, because only a small number of inner spheres will be visited (unless the time budget permits a complete traversal of all overlapping pairs).

A simple heuristic to derive an estimate of the lower bound could be to compute

$$\sum_{(R,S)\in Q} \sum_{\substack{R_i\in \mathrm{ch}(R),\\ S_j\in \mathrm{ch}(S)}} \mathrm{Vol}(R_i \cap S_j), \tag{5}$$

where $\mathrm{ch}(S)$ is the set of all direct children of node S.

Equation 5 amounts to the sum of the intersection of all direct child pairs of all pairs in the p-queue Q. Unfortunately, the direct children of a node are usually not disjoint and, thus, this estimate of the lower bound could actually be larger than the upper bound.

In order to avoid this problem, we introduce a new method to estimate the overlap volume more accurately: the *expected overlap volume*.

The only assumption we make is that for any point inside S, the distribution of the probability that it is also inside one of its leaves is uniform.

Let (R, S) be a pair of spheres in the p-queue. We define the *density* of a sphere as

$$p(S) = \frac{\mathrm{Vol}_l(S)}{\mathrm{Vol}(S)}. \tag{6}$$

with

$$\mathrm{Vol}_l(S) = \sum_{S_j\in \mathrm{Leaves}(S)} \mathrm{Vol}(S_j), \tag{7}$$

where S_j are all inner spheres below S in the IST.

This is the probability that a point inside S is also inside one of its leaves (which are disjoint). Next, we define the *expected overlap volume* $\overline{\mathrm{Vol}}(R, S)$ as the probability that a point is inside $R \cap S$ and also inside the intersection of one of the possible pairs of leaves, i.e.,

$$\begin{aligned} \overline{\mathrm{Vol}}(R, S) &= p(S) \cdot p(R) \cdot \mathrm{Vol}(R \cap S) \\ &= \frac{\mathrm{Vol}_l(R) \cdot \mathrm{Vol}_l(S) \cdot \mathrm{Vol}(R \cap S)}{\mathrm{Vol}(R) \cdot \mathrm{Vol}(S)} \end{aligned} \tag{8}$$

(see Figure 8).

This estimate can be further refined by taking it one level deeper in the hierarchy. So we define a “second level” expected overlap volume

$$\overline{\mathrm{Vol}}^{(1)}(R, S) = \sum_{\substack{R_i\in \mathrm{ch}(R),\\ S_j\in \mathrm{ch}(S)}} \overline{\mathrm{Vol}}(R_i, S_j) \tag{9}$$

This could be further refined to any level $\overline{\mathrm{Vol}}^{(i)}(R, S)$, but in our experience $\overline{\mathrm{Vol}}^{(1)}(R, S)$ has yielded the best compromise between performance and accuracy.

In summary, for the whole queue we get an expected overlap volume by

$$\sum_{(R,S)\in Q} \overline{\mathrm{Vol}}^{(1)}(R, S) \tag{10}$$

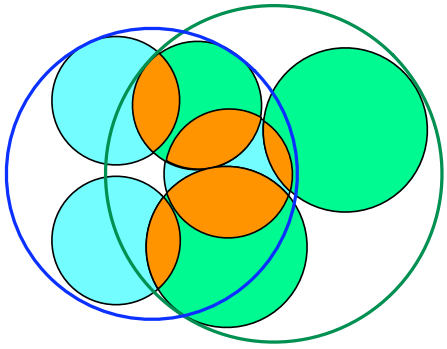

Figure 8: *We estimate the real penetration volume (yellow) during our time-critical traversal by the "density" in the hierarchy spheres (green and blue) and the total volume of leaf spheres.*

Clearly, this volume can be maintained during traversal quite easily.

More importantly, this method provides a much better heuristic for sorting the priority queue: if the gap between the expected overlap $\overline{\mathrm{Vol}}^{(1)}(R, S)$ and the overlap $\mathrm{Vol}(R \cap S)$ is large, then it is most likely that the traversal of this pair will give the most benefit towards improving the bound; consequently, we insert this pair closer to the front of the queue.

Algorithm 3 shows the pseudo code of this approach. (Note that $p(S) = 1$ if S is a leaf, and therefore $\overline{\mathrm{Vol}}(R, S)$ returns the exact intersection volume at the leaves.)

4.3 Combined Traversal

In the previous sections, we introduced the proximity and the penetration volume computation separately. However, it is of course possible to combine Algorithms 1 and 2. This yields a unified algorithm that can compute both the distance and the penetration volume.

To that end, we start with the distance traversal of Algorithm 1. If we find the first pair of intersecting inner spheres, then we simply switch to the penetration volume computation.

This is correct because all pairs of inner spheres we visited so far do not overlap and thus do not extend the penetration volume. Thus, we do not have to visit them again and can continue with the traversal of the rest of the hierarchies using the penetration volume algorithm. If we do not meet an intersecting pair of inner spheres, the unified algorithm still reports the minimal separating distance.

Algorithm 3: checkVolumeTimeCritical(A, B)

input : A, B = root spheres of the two ISTs
estOverlap = $\overline{\mathrm{Vol}}^{(1)}(A, B)$
Q = empty priority queue
Q.push(A, B)
while *Q not empty & time not exceeded* **do**
  $(R, S) =$ Q.pop()
  if *R and S are not leaves* **then**
    estOverlap –= $\overline{\mathrm{Vol}}^{(1)}(R, S)$
    forall $R_i \in$ *children of* R, $S_j \in$ *children of* S **do**
      estOverlap += $\overline{\mathrm{Vol}}^{(1)}(R_i, S_j)$
      Q.push(R_i, S_j)

5 Results

We have implemented our new data structure in C++ on a PC running Windows XP with an Intel Pentium IV 3GHz dual core CPU and 2GB of memory. We used a modified version of Dan Morris' *Voxelizer* [Morris, 2006] to compute the voxelization and the initial distances of the voxels.

We used several hand recorded object paths for benchmarking. In order to test the proximity queries, we moved the objects within a distance range of about 0–10% of the object's BV size. Here, we focused on very close configurations, because these are more stressing and also more interesting in real world scenarios. The paths for the penetration volume tests concentrate on light to medium penetrations of about 0–10% of the object's volume, because this resembles the usage in haptic applications best. But we also included some heavy penetrations of 50% of the object's volume so as to stress our algorithm.

We used several different objects to test the performance of our algorithms, including extremely concave objects, like pigs or the torso (see Figures 9 and 14). The polygon count ranges from 1k to 350k triangles per object. We voxelized each object in different resolutions in order to evaluate the trade-off between the number of spheres and the accuracy.

We compared the performance of our proximity query algorithm with the PQP library. It supports exact and approximate distance computation and make use of coherence and other techniques to improve performance. We used the exact distance computation in order to get reference values for our approximative approach. We also turned on the frame-to-frame coherence in our algorithm.

To our knowledge, there are no publicly available implementations to compute the penetration volume efficiently. In order to evaluate the quality of our penetration volume approximation, we used a tetrahedralization in combination with a sphere hierar-

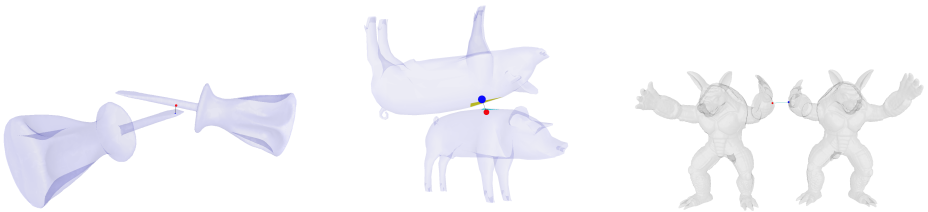

Figure 9: *Some snapshots from our proximity query benchmarks: a pig, a screwdriver (27k polygons) and an armadillo (350k polygons). The red and blue spheres show the closest pair of spheres.*

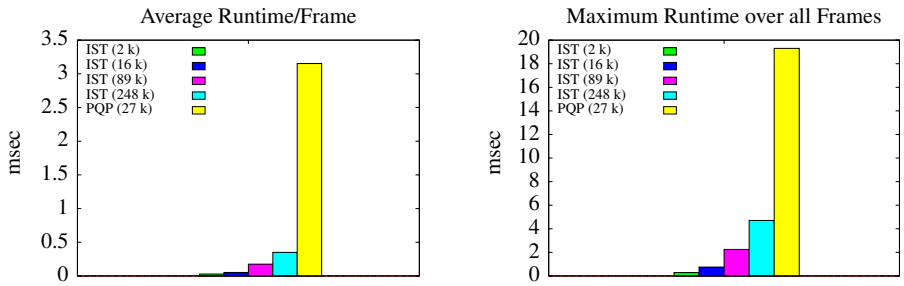

Figure 10: *Benchmark of the proximity query using the screwdriver scene. Left: average total query time of the IST and PQP; the number in parentheses gives the sphere resolution and the number of triangles, resp. Even with a very large number of inner spheres, the IST is 10 times faster than PQP, and this factor increases up to 120 when we reduce the number of spheres. Right: the maximum runtime; in contrast to PQP, even in the worst case it is able to guarantee 200 Hz with the highest resolution.*

chy to compute the exact overlap volume. However, the runtime of this approach is not applicable to real-time applications due to bad BV fitting and the costly tetrahedron-tetrahedron overlap volume calculation.

The results of our benchmarking show that our ISTs outperform PQP by an order of magnitude with a negligible loss of accuracy. The speed-up is between 10–120 times, depending on the density of the inner spheres (see Figures 10 and 11).

Our penetration volume algorithm is able to answer queries at haptic rates, even for very large objects with hundreds of thousands of polygons, as long as the interpenetration is not too big (see Figure 15). In the case of deeper penetrations, our time-critical traversal guarantees acceptable estimations of the penetration volume even in worst-case scenarios and multiple contacts (see Figure 16).

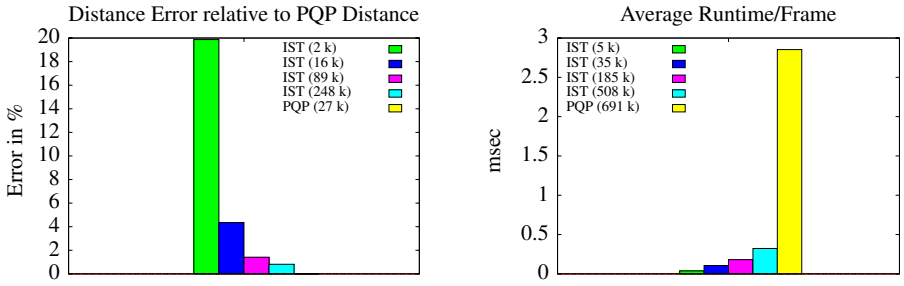

Figure 11: *Left: the error of approximate proximity based on ISTs is relatively small in the screwdriver scene; for the IST with highest resolution, the distance error is less than 0.5% compared to the accurate distance computed by PQP, and even with the IST containing only 89k inner spheres, the error is* $< 1\%$ *while the runtime is two times faster. Right: average total time of IST and PQP using the armadillo scene. Again, the IST is more than 10 times faster than PQP.*

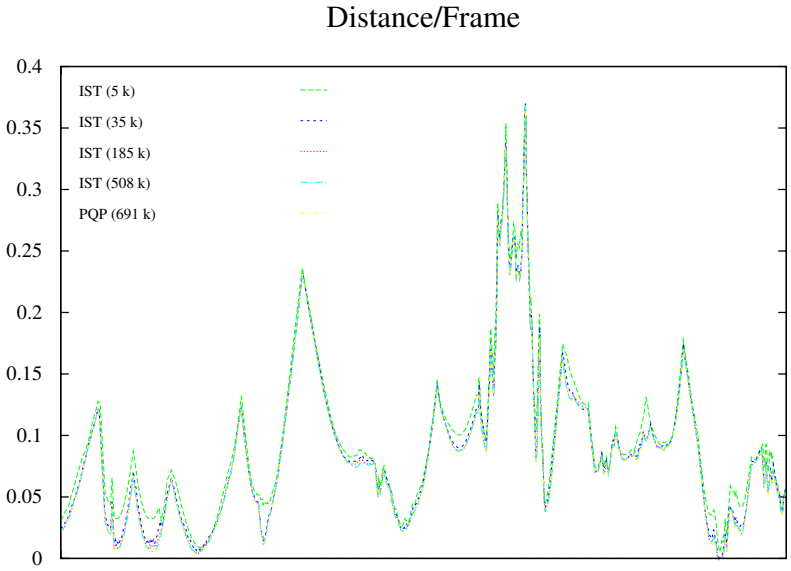

Figure 12: *Distances per frame in the armadillo scene: The overestimation of the lowest resolution IST (green, 5k spheres) is relatively big, whereas the error of the ISTs with higher resolutions (35k, dark blue, 185k, red, 508k, light blue) differ only in insignificant details from the exact distance curve of PQP (yellow).*

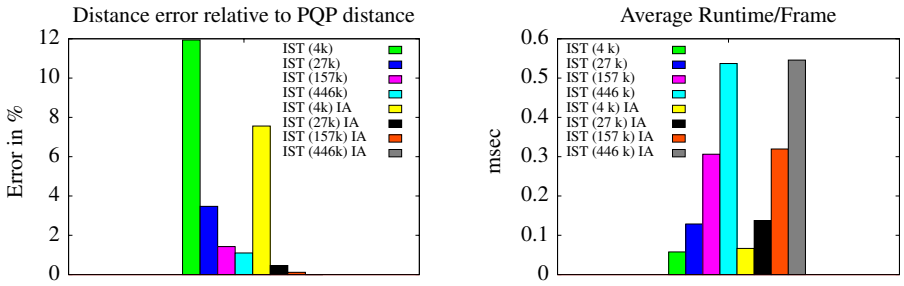

Figure 13: *Left: improvement of the accuracy (IA) in distance queries by the method described in Section 4.1.2) in the pig scene with only one triangle stored for every sphere. Right: the improvement of accuracy does not significantly affect the runtime our traversal algorithm.*

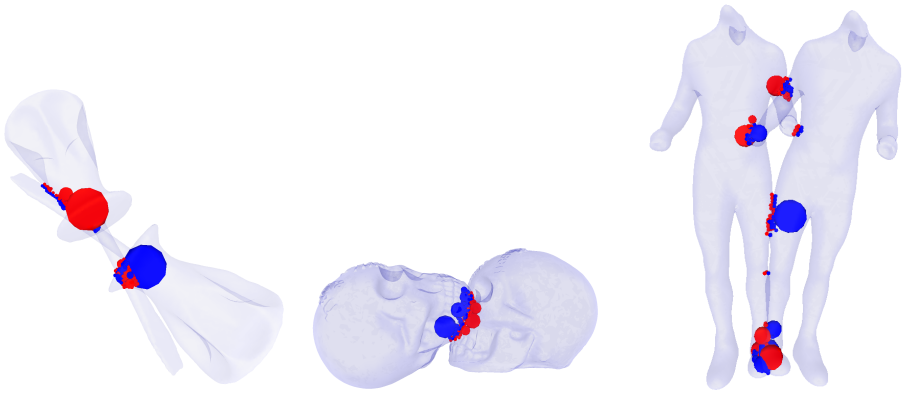

Figure 14: *Some snapshots from our penetration volume benchmarks: another screwdriver scene (27k polygons), a skull (100k polygons) and a human torso (200k polygons). The red and blue spheres show the overlapping inner spheres.*

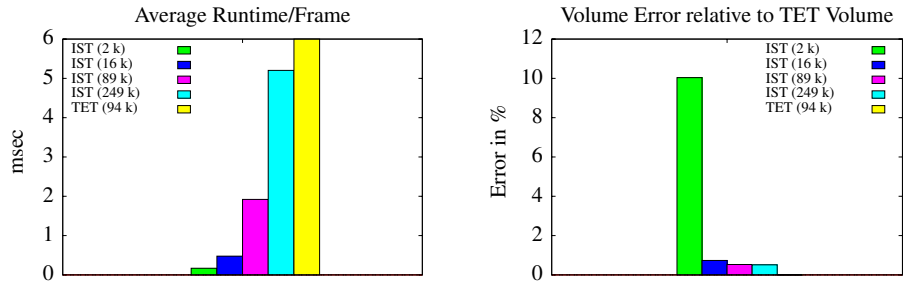

Figure 15: *Left: average runtime for the penetration volume computation in the second screwdriver scene; the average runtime of the tetrahedral tree (TET) is about 1.5 sec/frame with 94k tetrahedrons; the IST allows haptic refresh rates at every sphere resolution. Right: the loss of accuracy is very small compared to the exact overlap volume; clearly, the error decreases sharply with increasing sphere resolution and is about 0.5% for the IST with 89k spheres.*

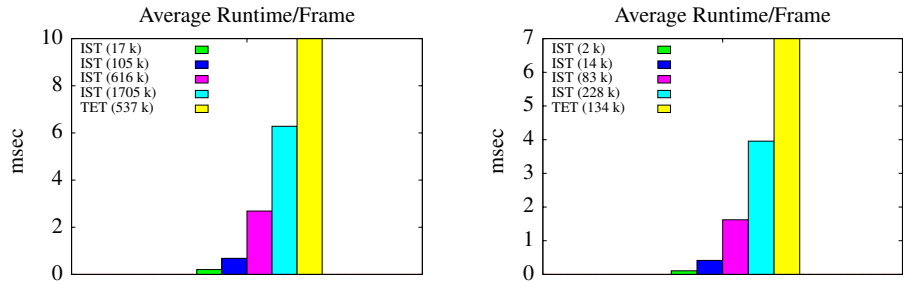

Figure 16: *Average runtime of the penetration volume computation using the scull scene (left) and the torso scene (right); here, some configurations in the object paths incur heavy penetrations of about 50% of object volume. The tetrahedral tree needs more than 2 sec/frame in both scenarios.*

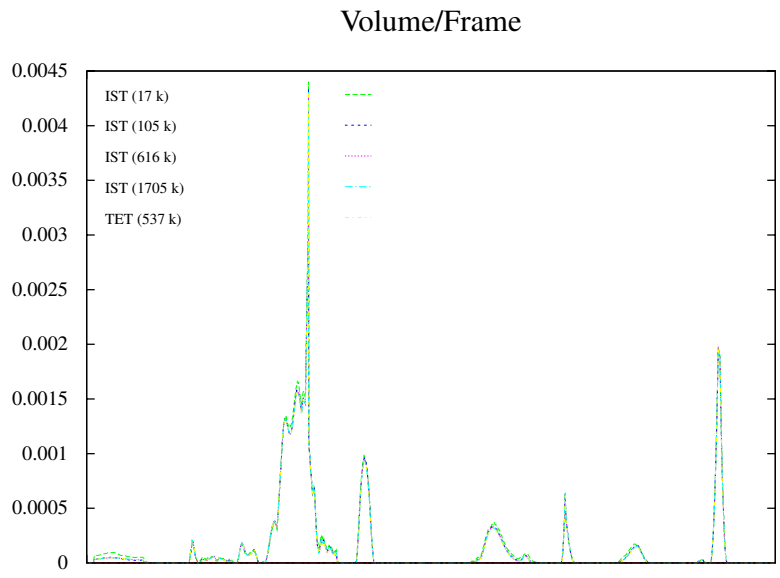

Figure 17: *Volume per frame in the scull scene: Even with the lowest resolution IST (green, 17k spheres) we get a close approximation of the exact volume curve computed with the tetrahedral tree (yellow). The error of the ISTs with higher resolutions (105k, dark blue, 616k, red, 1.7M, light blue) differ only in negligible details from the exact curve.*

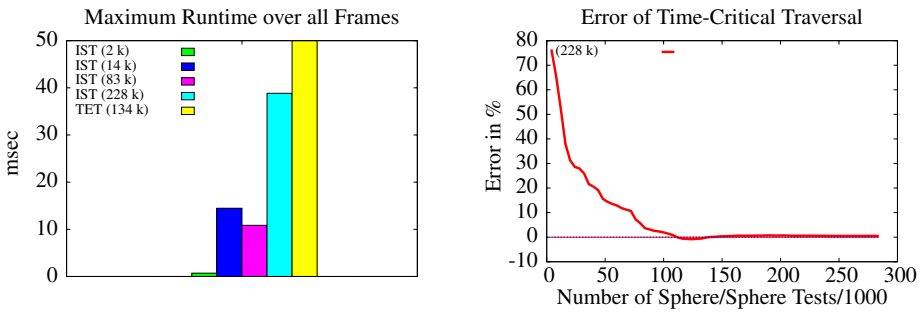

Figure 18: *Benchmark of the penetration volume computation using the torso scene; Left: maximum runtime (the tetrahedral tree needs 20 sec). Right: error of our time-critical penetration volume algorithm based on the exact overlap volume, relative to the number of sphere-sphere intersection tests.*

6 Conclusions and Future Work

We have presented a novel hierarchical data structure, the *inner sphere trees*, that supports different kinds of collision queries, namely proximity queries and penetration volume computations. Both kinds can be answered at rates of about 1 kHz (which makes the algorithm suitable for haptic rendering) even for very complex objects with several hundreds of thousands of polygons. For proximity computations, inner sphere trees achieve runtimes more than 10 times faster than PQP, with a negligible loss of accuracy of about 0.5%. The balance between accuracy and speed can be defined by the user, and this is independent of the object complexity, because the number of leaves of our hierarchy is mostly independent from the number of polygons. Finally, our algorithms for both kinds of queries can be integrated into existing simulation software very easily, because there is only a single entry point, i.e., the application does not need to know in advance whether or not a given pair of objects will be penetrating each other.

Memory consumption of our inner sphere trees is similar to other bounding volume hierarchies, depending on the predefined accuracy. This is very modest compared to voxel-based approaches.

Another big advantage of our penetration volume algorithm over voxel-based approaches is that it yields a continuous measure for penetration and that it utilizes the same data structure for both static and dynamic objects. This implies, that there is no time consuming re-sampling and re-voxeling needed when the roles of the objects or the environment change.

Last but not least, inner sphere trees are perfectly suited for SIMD acceleration techniques and allow algorithms to make heavy use of temporal and spatial coherence.

Our novel approach opens up several avenues for future work.

First of all, the intermediate step of voxelization in order to obtain a sphere packing should be replaced with a better algorithm. This is probably a challenging problem, because several goals should be met: accuracy, query efficiency, and small build times.

Another interesting task is to explore other uses of “inner bounding volume hierarchies”, such as ray tracing or occlusion culling. Note that the type of bounding volume chosen for the “inner hierarchy” probably depends on its use.

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, and it could further improve the heuristics for the time-critical traversal.

It should be straight-forward to derive more contact information, like contact normals, to compute forces. This can possibly be done by extending a method proposed in [Faure et al., 2008].

Finally, a challenging task would be to extend our approach also to deformable objects.

References

[Agarwal et al., 2004] Agarwal, Guibas, Nguyen, Russel, and Zhang (2004). Collision detection for deforming necklaces. *CGTA: Computational Geometry: Theory and Applications*, 28.

[Barbič and James, 2008] Barbič, J. and James, D. L. (2008). Six-dof haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Transactions on Haptics*, 1(1):39–52.

[Birgin and Sobral, 2008] Birgin, E. G. and Sobral, F. N. C. (2008). Minimizing the object dimensions in circle and sphere packing problems. *Computers & OR*, 35(7):2357–2375.

[Bradshaw and O'Sullivan, 2004] Bradshaw, G. and O'Sullivan, C. (2004). Adaptive medial-axis approximation for sphere-tree construction. In *ACM Transactions on Graphics*, volume 23(1), pages 1–26. ACM press.

[Cameron, 1997] Cameron, S. (1997). Enhancing GJK: Computing minimum and penetration distances between convex polyhedra. In *Proc. of Int'l Conf. on Robotics and Automation*, pages 3112–3117.

[Cottrell et al., 2006] Cottrell, M., Hammer, B., Hasenfuss, A., and Villmann, T. (2006). Batch and median neural gas. *Neural Networks*, 19:762–771.

[Faure et al., 2008] Faure, F., Barbier, S., Allard, J., and Falipou, F. (2008). Image-based collision detection and response between arbitrary volumetric objects. In *ACM Siggraph/Eurographics Symp. on Computer Animation, SCA 2008, July, 2008*, Dublin, Irlande.

[Fisher and Lin, 2001] Fisher, S. M. and Lin, M. C. (2001). Fast penetration depth estimation for elastic bodies using deformed distance fields.

[Gärtner, 1999] Gärtner, B. (1999). Fast and robust smallest enclosing balls. In Nesetril, J., editor, *ESA*, volume 1643 of *Lecture Notes in Computer Science*, pages 325–338. Springer.

[Gilbert et al., 1988] Gilbert, E. G., Johnson, D. W., and Keerthi, S. S. (1988). A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE J. of Robotics and Automation*, 4:193–203.

[Gottschalk et al., 1996] Gottschalk, S., Lin, M. C., and Manocha, D. (1996). OBB-Tree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30(Annual Conference Series):171–180.

[Hammer et al., 2006] Hammer, B., Hasenfuss, A., and Villmann, T. (2006). Magnification control for batch neural gas. In *ESANN*, pages 7–12.

[Hoff et al., 2002] Hoff, K. E., Zaferakis, A., Lin, M., and Manocha, D. (2002). Fast 3d geometric proximity queries between rigid & deformable models using graphics hardware acceleration. Technical Report TR02-004, Department of Computer Science, University of North Carolina - Chapel Hill. Fri, 8 Mar 2002 20:06:33 GMT.

[Hong et al., 2000] Hong, S.-M., Yeo, J.-H., and Park, H.-W. (2000). A fast procedure for computing incremental growth distances. *Robotica*, 18(4):429–441.

[Hubbard, 1995] Hubbard, P. M. (1995). Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230.

[Hudson et al., 1997] Hudson, T. C., Lin, M. C., Cohen, J., Gottschalk, S., and Manocha, D. (1997). V-COLLIDE: Accelerated collision detection for VRML. In Carey, R. and Strauss, P., editors, *VRML 97: Second Symp. on the Virtual Reality Modeling Language*, New York City, NY. ACM Press.

[Johnson and Cohen, 1998] Johnson, D. E. and Cohen, E. (1998). A framework for efficient minimum distance computations. In *Proc. of the IEEE Int'l Conf. on Robotics and Automation (ICRA-98)*, pages 3678–3684, Piscataway. IEEE Computer Society.

[Johnson and Willemsen, 2003] Johnson, D. E. and Willemsen, P. (2003). Six degree-of-freedom haptic rendering of complex polygonal model. In *HAPTICS*, pages 229–235. IEEE Computer Society.

[Kim et al., 2003] Kim, Otaduy, Lin, and Manocha (2003). Fast penetration depth estimation using rasterization hardware and hierarchical refinement (short). In *COMPGEOM: Annual ACM Symp. on Computational Geometry*.

[Kim et al., 2002] Kim, Y., Otaduy, M., Lin, M., and Manocha, D. (2002). Fast penetration depth computation for physically-based animation. In Spencer, S. N., editor, *Proc. of the ACM SIGGRAPH Symp. on Computer Animation (SCA-02)*, pages 23–32, New York. ACM Press.

[Kim et al., 2004] Kim, Y. J., Lin, M. C., and Manocha, D. (2004). Incremental penetration depth estimation between convex polytopes using dual-space expansion. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):152–163.

[Klosowski et al., 1998] Klosowski, J. T., Held, M., Mitchell, J. S. B., Sowizral, H., and Zikan, K. (1998). Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36.

[Larsen et al., 1999] Larsen, E., Gottschalk, S., Lin, M., and Manocha, D. (1999). Fast proximity queries with swept sphere volumes. In *Technical Report TR99-018*.

[Martinetz et al., 1993] Martinetz, T. M., Berkovich, S. G., and Schulten, K. J. (1993). 'Neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Trans. on Neural Networks*, 4(4):558–569.

[McNeely et al., 1999] McNeely, W. A., Puterbaugh, K. D., and Troy, J. J. (1999). Six degrees-of-freedom haptic rendering using voxel sampling. In Rockwood, A., editor, *Siggraph 1999*, Annual Conference Series, pages 401–408, Los Angeles. ACM Siggraph, Addison Wesley Longman.

[Mendoza and O'Sullivan, 2006] Mendoza, C. and O'Sullivan, C. (2006). Interruptible collision detection for deformable objects. *Computers & Graphics*, 30(3):432–438.

[Morris, 2006] Morris, D. (2006). Algorithms and data structures for haptic rendering: Curve constraints, distance maps, and data logging.

[O'Brien and Hodgins, 1999] O'Brien, J. F. and Hodgins, J. K. (1999). Graphical modeling and animation of brittle fracture. In *SIGGRAPH '99*, pages 137–146, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

[Ortega et al., 2007] Ortega, M., Redon, S., and Coquillart, S. (2007). A six degree-of-freedom god-object method for haptic display of rigid bodies with surface properties. *IEEE Trans. Vis. Comput. Graph*, 13(3):458–469.

[Quinlan, 1994] Quinlan, S. (1994). Efficient distance computation between non-convex objects. In *In Proc. of Int'l Conf. on Robotics and Automation*, pages 3324–3329.

[Redon and Lin, 2006] Redon, S. and Lin, M. C. (2006). A fast method for local penetration depth computation. *J. of Graphics Tools: JGT*, 11(2):37–50.

[Renz et al., 2001] Renz, M., Preusche, C., Pötke, M., peter Kriegel, H., and Hirzinger, G. (2001). Stable haptic interaction with virtual environments using an adapted voxmap-pointshell algorithm. In *In Proc. Eurohaptics*, pages 149–154.

[Schuermann, 2006] Schuermann, A. (2006). On packing spheres into containers (about kepler's finite sphere packing problem).

[Terdiman, 2001] Terdiman, P. (2001). Memory-optimized bounding-volume hierarchies.

[Trenkel et al., 2007] Trenkel, S., Weller, R., and Zachmann, G. (2007). A benchmarking suite for static collision detection algorithms. In Skala, V., editor, *Inter'l Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, Plzen, Czech Republic. Union Agency.

[van den Bergen, 1999] van den Bergen, G. (1999). A fast and robust GJK implementation for collision detection of convex objects. *J. of Graphics Tools: JGT*, 4(2):7–25.

[Zachmann, 1998] Zachmann, G. (1998). Rapid collision detection by dynamically aligned DOP-trees. In *Proc. of IEEE Virtual Reality Annual Int'l Symp.; VRAIS '98*, pages 90–97, Atlanta, Georgia.

[Zhang et al., 2007] Zhang, L., Kim, Y. J., Varadhan, G., and Manocha, D. (2007). Generalized penetration depth computation. *Computer-Aided Design*, 39(8):625–638.