Inner Sphere Trees for Proximity and Penetration Queries

Rene Weller
Clausthal University, Germany
Email: rwe@tu-clausthal.de

Gabriel Zachmann
Clausthal University, Germany
Email: zach@tu-clausthal.de

***Abstract*— We present a novel geometric data structure for approximate collision detection at haptic rates between rigid objects. Our data structure, which we call *inner sphere trees*, supports different kinds of queries, namely, proximity queries and a new method for interpenetration computation, the penetration *volume*, which is related to the water displacement of the overlapping region and, thus, corresponds to a physically motivated force.**

The main idea is to bound objects from the *inside* with a set of *non-overlapping* spheres. Based on such sphere packings, a "inner bounding volume hierarchy" can be constructed. In order to do so, we propose to use an AI clustering algorithm, which we extend and adapt here.

The results show performance at haptic rates both for proximity and penetration volume queries for models consisting of hundreds of thousands of polygons.

I. Introduction

Collision detection between rigid objects is important for many fields of robotics and computer graphics, e.g. for path-planning, haptic rendering, physically-based simulations, and medical applications. Today, there exist a wide variety of freely available collision detection libraries and nearly all of them are able to work at visual interactive rates, even for very complex objects [TWZ07]. Most collision detection algorithms dealing with rigid objects use some kind of bounding volume hierarchy (BVH). The main idea behind a BVH is to subdivide the primitives of an object hierarchically until there are only single primitives left at the leaves. BVHs guarantee very fast responses at query time, so long as no further information than the set of colliding polygons is required for the collision response. However, most applications require much more information in order to resolve or avoid the collisions.

One way to do this is to compute the exact time of contact of the objects. This method is called continuous collision detection. Another approach, called penalty methods, is to compute repelling forces based on the penetration depth. However, there is no universally accepted definition of the penetration depth between a pair of polygonal models [ZKVM07], [OG96]. 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.

Haptic rendering requires update rates of at least 200 Hz, but preferably 1 kHz to guarantee a stable force feedback. Consequently, the collision detection time should never exceed 5 msec.

One example of an approach that offers fairly constant query times are voxel-based methods like the Voxmap Pointshell algorithm (VPS), where objects, in general, have to be voxelized (the "voxmap") and covered by a point cloud (the "point shell"). This can be very memory consuming and produce aliasing artifacts due to the discretization errors.

A. Main Contributions

This paper contributes the following novel ideas to the area of collision detection:

- a novel geometric data structure, the *Inner Sphere Trees (IST)*, that provides hierarchical bounding volumes from the *inside* of an object;
- a method to compute a dense sphere packing inside a polygonal object;
- we propose to utilize a clustering algorithm to construct a sphere hierarchy;
- a unified algorithm that can compute for a pair of objects, based on their ISTs, both an approximate minimal distance and the approximate penetration volume; the application does not need to know in advance which situation currently exists between the pair of objects;

Our ISTs and, consequently, the collision detection algorithm are independent of the geometry complexity; they only depend on the approximation error.

The main idea is that we do not build an (outer) 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 inner BVs for lower bounds instead of outer BVs for upper bounds can be extended analogously to all kinds of volumes. On top of these inner BVs, we build a hierarchy that allows for fast computation of the approximate proximity and *penetration volume*.

The penetration volume corresponds to the water displacement of the overlapping parts of the objects and, thus, leads to a physically motivated and continuous repulsion force. According to [FL01, Sec. 5.1], it is "the most complicated yet accurate method" to define the extent of intersection, which was also reported earlier by [OH99, Sec. 3.3]. However, to our

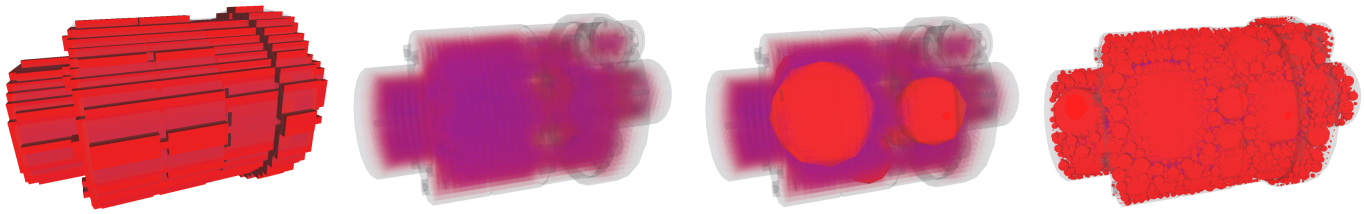

Fig. 1. These images show the different stages of our sphere packing algorithm. First, we voxelize the object (left) and compute distances from the voxels to the closest triangle (second image; transparency = distance). Then, we pick the voxel with the largest distance and put a sphere at its center. We proceed incrementally and, eventually, we obtain a dense sphere packing of the object (right).

knowledge, there are no algorithms to compute it efficiently as yet.

Our data structure can support all kinds of object representations, e.g. polygon meshes or NURBS surfaces. The only precondition is that they be watertight. In order to build the hierarchy on the inner spheres, we utilize a recently proposed clustering algorithm that allows us to work in an adaptive manner.

The results shows that our new data structure can answer both kinds of queries at haptic rates with a negligible loss of accuracy.

II. Previous Work

Collision detection has been extensively investigated by researchers in the past decades. There already exist a large variety of techniques and software packages for collision detection queries. They vary greatly in their techniques and computational method. Here we concentrate on those approaches, that are able to compute extended contact informations like a penetration depth or the separation distance between a pair of objects.

Most of them are not designed to work at haptic refresh rates, or they are restricted to simple point probes, 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.

A. BVH based methods

A classical approach for proximity queries is the GJK algorithm [GJK88], [vdB99]. It derives the distance between a pair of convex objects, by utilizing the Minkowski sum. It is also possible to extend GJK in order to compute the penetration depth [Cam97].

In [JC98] a generalized framework for minimum distance computations that depends on geometric reasoning is presented. It also includes time-critical properties.

PQP [LGLM99] creates a hierarchy of rectangle swept spheres. Distance computations are performed between these volumes on the hierarchical tree. Moreover, it uses specialized algorithms to improve the efficiency of distance calculations. We used it in this paper to compute the ground truth for the proximity queries.

Another package, supporting proximity queries between any convex quadrics is SOLID [vdB01]. It uses axis aligned bounding boxes and the Minkowski difference between convex polytopes together with several optimization techniques.

SWIFT++ [EL01] provides a convex surface decomposition scheme and a modified Lin-Canny closest feature algorithm to compute approximate as well as exact distance between general rigid polyhedra.

Sphere trees have also been used for distance computation in the past [Qui94], [Hub95], [MO06]. The algorithms presented there are interruptible and they are able to deliver approximative distances. Moreover, they all compute a lower bound on the distance, because of using outer hierarchies, while our ISTs derive an upper bound. Thus, a combination of these approaches with our ISTs could deliver good error bounds in both directions.

[ZKVM07] presented an extended definition of the penetration depth that also takes the rotational component into account, called the generalized penetration depth. However, this approach is computationally very expensive and, thus, might currently not be fast enough for haptic interaction rates.

[RL06] 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 literature on penetration volume computation is sparse. One approach, proposed by [HS04], constructs the intersection volume of convex polyhedra explicitly. For this reason, it is applicable only to very simple geometries at interactive rates.

Another interesting method is given by [FBAF08]. They compute an approximation of the intersection volume from layered depth images on the GPU. This approach is applicable to deformable geometries but restricted to image space precision.

B. Voxel based data structures

Classical Voxmap Pointshell approaches [MPT99] divide the environment into a dynamic object, that can move freely through space and static objects that are fixed in the world. The static world is discretized into a set of discrete volume elements, the voxels, while the dynamic object is described by a set of points that represent its surface, the pointshell. At query time, for each of these points in the pointshell, it is determined with a simple boolean test, whether it is located in a filled voxel or not.

There also exist extensions to VPS, including optimizations to the force calculation in order to increase its stability [RPP*01]. However, even such optimizations cannot

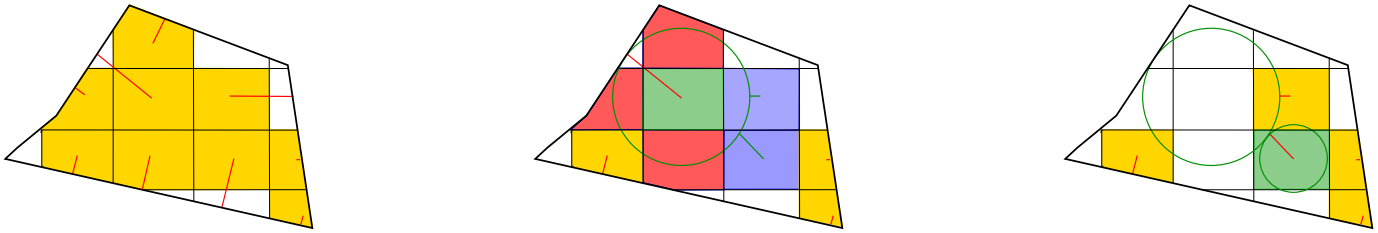

Fig. 2. This figure shows the first steps of the creation of the inner spheres. First, the object is voxelized (left). Then, we compute the shortest distance to the surface (red lines) for interior voxel centers (yellow), i.e., a discretization of the interior distance field. Next, we place a maximal sphere at the voxel center with the largest radius (green sphere). Then, the red colored voxels are deleted, and the shortest distances of some voxels are updated, because they are closer now to an inner sphere (blue). This procedure continues greedily as shown in the drawing to the right.

completely remedy the limits of VPS, namely aliasing effects and the high memory consumption.

Closely related to voxel-based approaches are distance field based methods. [BJ08] use a pointshell of reduced deformable models in combination with distance fields in order to guarantee continuous contact forces.

III. Creation of the Inner Sphere Tree

In this section we describe the construction of our data structure. In a fist step, we want a watertight object to be densely filled with a set of non-overlapping spheres. The volume of the object should be approximated well by the spheres, while their number should be small. In a second step, we create a hierarchy over this set of spheres.

For squared objects, spheres seem not to be a good choice as filling volumes. However, they compensate this disadvantage because of the trivial overlap test and their rotationally invariance. Moreover, it is easy, in contrast to AABBs or OBBs, to compute the exact intersection volume.

A. The Sphere Packing

Filling objects densely with smaller volumes is a highly non-trivial task and still an active field of research, even when restricted to spheres [BS08], [Sch06]. We present a simple heuristic that offers a good trade-off between accuracy and speed in practice.

This heuristic is currently based on a distance field. We start with a flood filling voxelization, 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, together with the triangle that realizes this distance.

After this initialization, we use a greedy algorithm to generate the inner spheres. All voxels are stored in a priority queue, sorted by their distance to the surface. Until the p-queue is empty, we extract the maximum element, i.e. the voxel V^* with the largest distance d^*. We create an inner sphere with radius d^* and centered on the center of the voxel V^*. Then, all voxels whose centers are contained in this sphere are deleted from the p-queue. Additionally, we have to update all voxels V_i with $d_i < d^*$ and distance $d(V_i, V^*) < 2d^*$. This is because they are now closer to the sphere around V^* than to a triangle on the hull (see Figure 2). Their d_i must now be set to the new free radius.

After this procedure, the object is filled densely with a set of non-overlapping spheres. The density, and thus the accuracy, can be controlled by the number of voxels.

B. Building the IST

Our sphere hierarchy is based on the notion of a *wrapped hierarchy* [AGN*04], where inner nodes are tight BVs for all their leaves, but they do not necessarily bound their direct children. Compared to layered hierarchies, the big advantage is that the inner BVs are tighter. 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. Previous algorithms for building ordinary sphere trees, like the medial axis approach [BO04], [Hub95] work well if the spheres constitute a *covering* of the object and have similar size, but in our scenario we use disjoint inner spheres that exhibit a large variation in size. 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 [CHHV06]. BNG is a very robust clustering algorithm, which can be formulated 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 cluster center positions (the *prototypes*): 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; below, we will describe how this can be used to increase the quality of the ISTs.

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

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

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

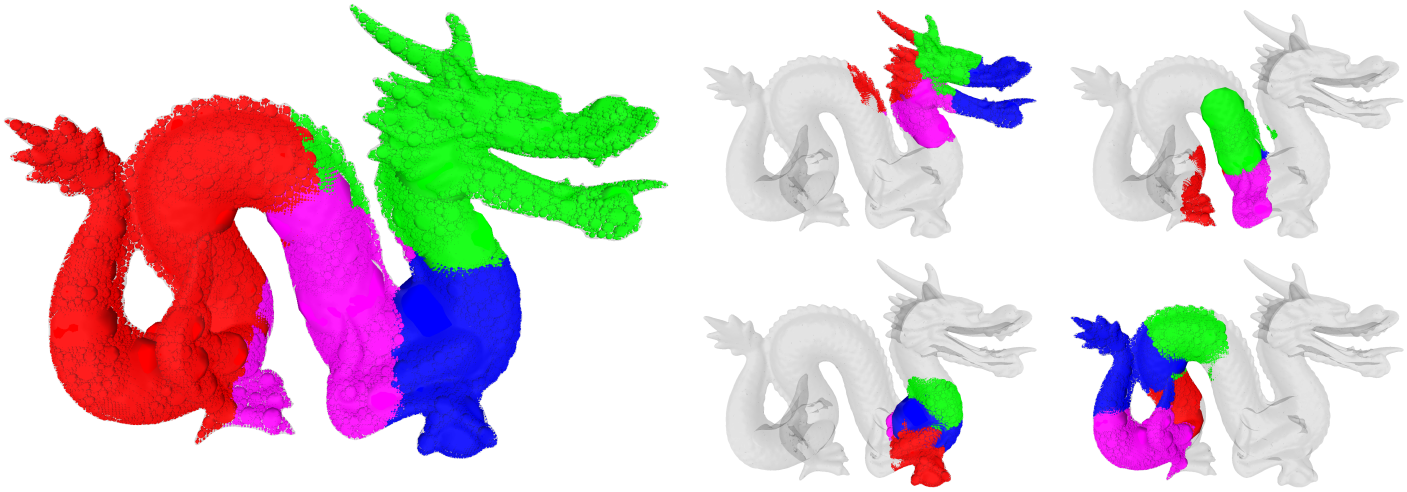

Fig. 3. This figure shows the results of our hierarchy building algorithm based on batch neural gas clustering with magnification control. All of those inner spheres that share the same color are assigned to the same bounding sphere. The left image shows the clustering result of the root sphere, the right images the partitioning of its four children.

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 $\varepsilon \approx 10^{-5} \times \text{BoundingBoxSize}$, 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 [MBS93].

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. Additionally, this has the benefit 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 levels of the hierarchy in order to get early lower bounds during distance traversal (see Section IV-A 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 [HHV06], 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 [Gär99]. 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 (See Figure 3 for some clustering results).

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 III-A, will be called *inner spheres*. Note that hierarchy spheres are not necessarily contained completely within the object.

IV. BVH Traversal

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

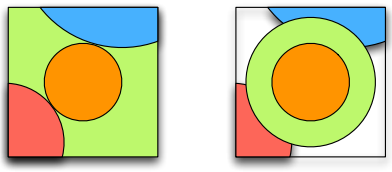

Fig. 4. After constructing the sphere packing (see Section III-A), every voxel can be intersected by several non-overlapping spheres (left). These do not necessarily account for the whole voxel space (green space in the left picture). In order to account for these voids, too, we simply increase the radius of the sphere that covers the center of the voxel (right).

Algorithm 1: computeVolume(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 )
```

realizing the distance, and the penetration algorithm can return a partial list of intersecting polygons.

In this section, we concentrate on these two types of queries, because they are more interesting for physically-based simulations that need some contact information to compute proper forces. But it should be obvious that the traversal can be easily modified in order to provide also approximate yes-no answer, which would further increase the speed of collision detection.

We start with a separate discussion of the two query types in order to point out their specific requirements. In Section IV-C we describe how to combine these traversal schemes to a unified algorithm that is able to provide distance and overlap volume informations, without the user has to know in advance, whether the objects overlap or not.

A. Proximity Queries

Our proximity query algorithm works like most other classical BVH traversal algorithms: We check whether two bounding volumes overlap or not. If this is the case, we recursively step to their children. In order to compute lower bounds for the distance, we simply have to add an appropriate distance test at the right place. This has to be done, when we reach a pair of inner spheres (i.e., leaves) during traversal. Due to Section III-A, these inner spheres are located completely inside the object and provide, thus, a lower bound on the sought-after distance. 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 results in a high culling efficiency.

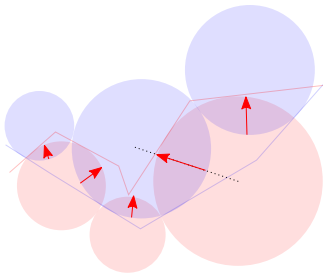

Fig. 5. The direction of the penalty force can be derived from the weighted average of all vectors between the centers of colliding pairs of spheres, weighted by their overlap.

It should be obvious, that this algorithm can be easily extended to triangle accuracy by additionally taking the triangles into account that are closest to an inner sphere.

B. Penetration Volume Queries

As stated before, our data structure does not only support proximity queries, but also a new kind of penetration query, namely the *penetration volume*. This is the volume of the intersection of the two objects, which can be interpreted directly as the amount of the repulsion force, if it is considered as the amount of water being displaced.

The algorithm to compute the penetration volume (see Algorithm 1) 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. The overlap volume of a pair of spheres can be easily derived by adding the volumes of the spherical caps.

Due to the non-overlapping constraint of the inner spheres, the accumulated overlap volumes provides a lower bound on the real overlap volume of the objects.

1) Filling the gaps: The algorithm described in Section III-A results in densely filled objects. However, there still remain small voids between the spheres that cannot be completely compensated by increasing the number of voxels. This results in bad lower bounds.

As a remedy, we propose a simple heuristic to compensate this problem: We additionally assign a *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 (see Figure 4). 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.)

2) Collision response: In order to apply penalty forces in haptic environments or simulations, we also need the direction of the force in addition to its amount.

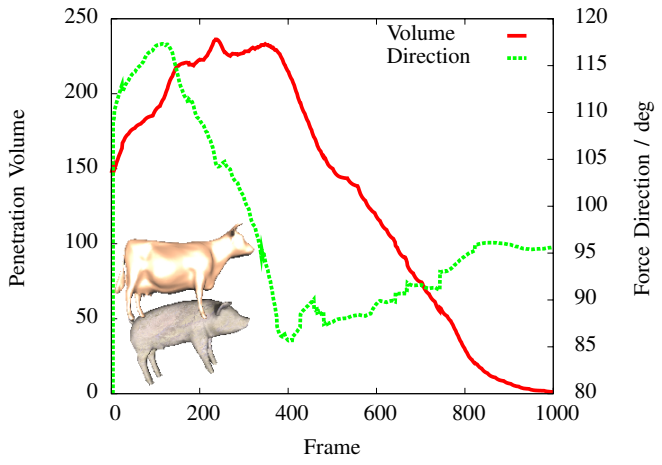

Fig. 6. Penetration volume (red) and direction of the force vector (green) during the path of a cow scraping alongside a pig.

This can be derived easily from our ISTs by considering all overlapping pairs of spheres (R_i, S_j) separately. Let $\mathbf{c}_i, \mathbf{c}_j$ be their sphere centers and $\mathbf{n}_{ij} = \mathbf{c}_i - \mathbf{c}_j$. Then, we compute the overall direction of the penalty force as the weighted sum $\mathbf{n} = \sum_{i,j} \mathrm{Vol}(R_i \cap S_j) \cdot \mathbf{n}_{ij}$ (see Figure 5). Obviously, this direction is continuous, provided the path of the objects is continuous (see Figure 6).

In case of deep penetrations, it can be necessary to flip some of the directions $\mathbf{n}_{ij}$. Computing normal cones for all spheres throughout the hierarchy can help to identify these pairs. It is also possible to extend this approach to provide continuous torques. Details are omitted here due to space constraints.

C. The Unified Algorithm

In the previous sections, we introduced the proximity and the penetration volume computation separately. However, it is quite easy to combine both algorithms. This yields a unified algorithm that can compute both the distance and the penetration volume, without the user having to know in advance, whether the objects overlap or not.

We start with the distance traversal. If we find the first pair of intersecting inner spheres, then we simply switch to the penetration volume computation.

The correctness is based on the fact, that all pairs of inner spheres we visited so far during distance traversal 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.

V. Results

We have implemented our new data structure in C++. The testing environment consists of a PC running Windows XP with an Intel Pentium IV 3GHz dual core CPU and 2GB of memory. The initial distance field was computed using a slightly modified version of Dan Morris' *Voxelizer* [Mor06].

The benchmark includes hand recorded object paths with distances ranging from about 0–20% of the object's BV size for the proximity queries. We concentrated on very close configurations, because they are more interesting in real world scenarios and more challenging regarding the running time. The paths for the penetration volume queries concentrate on light to medium penetrations of about 0–10% of the object's volume. This scenario resembles the usage in haptic applications best, because the motive for using collision detection algorithms is to avoid heavy penetrations. However, we also included some heavy penetrations of 50% of the object's volume to stress our algorithm.

We used highly detailed objects with a polygon count ranging up to 370k to test the performance and the quality of our algorithm.[1] The quality of the resulting distances and penetration volumes is closely related to the quality of the underlying voxelization. Consequently, we voxelized each object in different resolutions in order to evaluate the trade-off between the number of spheres and the accuracy.

We computed the ground truth data for the proximity queries with the PQP library. We also included the running time of PQP in our plots, even if the comparison seems to be somewhat unfair, because PQP computes exact distances. However, it shows the impressive speed-up that is achievable when using approximative approaches. Moreover, it is possible to extend ISTs to support exact distance calculations, too.

To our knowledge, there are no implementations available to compute the exact penetration volume efficiently. In order to still evaluate the quality of our penetration volume approximation, we used a tetrahedralization to compute the exact volume. Even though we speed it up by a hierarchy built on the tetrahedra, the running time of this approach is in the order of 0.5 sec/frame.[2]

The results of our benchmarking show that our ISTs with the highest sphere resolution have an average speed-up of 50 compared to PQP, while the average error is only 0.15%. Even in the worst case, they are suitable for haptic rendering with response rates of less than 2 mesc in the highest resolution (see Figure 7).

Our penetration volume algorithm is able to answer queries at haptic rates between 0.1 msec and 2.5 msec on average, depending on the voxel resolution, even for very large objects with hundreds of thousands of polygons (see Figure 8). The average accuracy using the highest sphere resolution is around 0.5%.

The per-frame quality displayed in Figure 9 re-emphasizes the accuracy of our approach and, additionally, shows the continuity of the distance and the volume.

VI. Conclusions and Future Work

We have presented a novel hierarchical data structure, the *inner sphere trees*. The ISTs support different kinds of

[1] Please visit cg.in.tu-clausthal.de/research/ist to watch some videos of our benchmarks.

[2] This is due to bad BV tightness and the costly tetrahedron-tetrahedron overlap volume calculation.

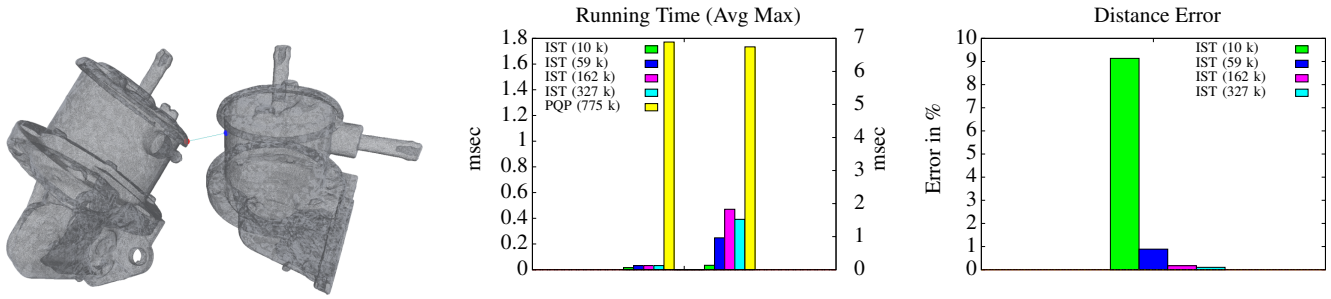

Fig. 7. Left: Snapshot from our oil pump scene (330k triangles). The red and blue spheres show the closest pair of spheres. Center: average and maximum time/frame. Left: relative error compared to accurate distance.

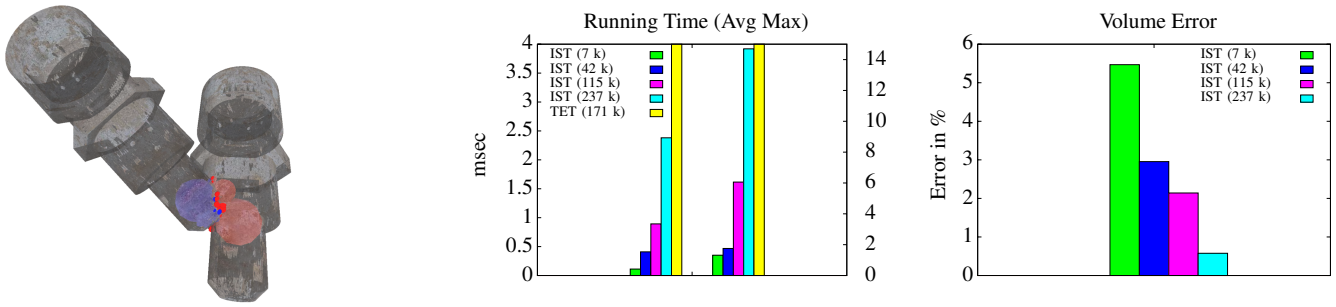

Fig. 8. Left: Snapshot from our bolt scene (171k triangles). The red and blue spheres show the overlapping inner spheres. Center: average and maximum time/frame. Right: relative error compared to accurate penetration volume.

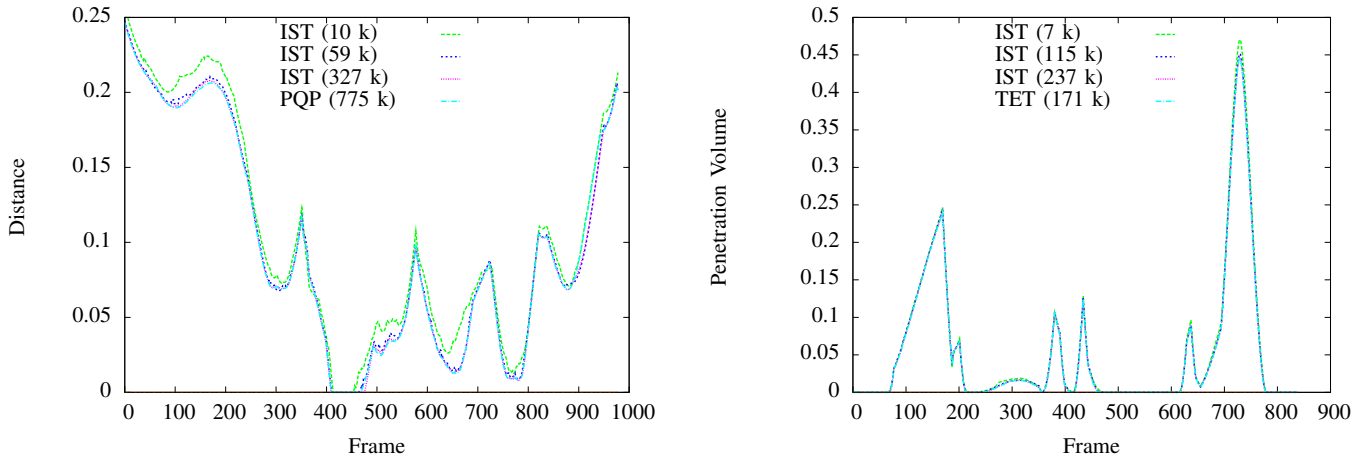

Fig. 9. Left: Distance per frame in the oil pump scene. Right: Penetration volume per frame in the bolt scene.

collision detection queries, including proximity queries and penetration volume computations with one unified algorithm. All types of queries 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 situations, typical average running times are in the order of 0.05 msec with 327 000 spheres per object and an error of about 0.1%. In penetration situations, the running times depend, obviously, much more on the intersection volume; here, we are in the order of around 2.5 msec on average with 237 000 spheres and an error of about 0.5%.

The balance between accuracy and speed can be defined by the user. Moreover, the speed is independent of the object complexity, because the number of leaves of our hierarchy is mostly independent of the number of polygons.

Our algorithm 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 (in our experiments, it was always in the order

of a few MB). This is very modest compared to voxel-based approaches.

Another big advantage of our penetration volume algorithm, when utilized for penalty-based simulations, is that it yields continuous directions and magnitudes of the force and the torque, even cases of deep penetrations.

Our novel approach opens up several avenues for future work.

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

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.

On the whole, ISTs are fast enough for haptic refresh rates. However, there exist configurations, especially in cases of heavy penetrations, where the 1 kHz constraint may not always be met. Therefore it would be nice to apply time critical techniques to the traversal algorithms in order to guarantee fixed response times.

Finally, there might also be some room for improving the hierarchy. For example, it could be better, especially at the borders of an object, to minimize the volume of those parts of hierarchy spheres that are outside of the object, instead of minimizing their volume.

ACKNOWLEDGMENT

This work was partially supported by DFG grant ZA292/1-1 and BMBF grant Avilus / 01 IM 08 001 U.

REFERENCES

[AGN*04] AGARWAL, GUIBAS, NGUYEN, RUSSEL, ZHANG: Collision detection for deforming necklaces. *CGTA: Computational Geometry: Theory and Applications 28* (2004).

[BJ08] BARBIČ J., JAMES D. L.: Six-dof haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Transactions on Haptics 1*, 1 (2008), 39–52.

[BO04] BRADSHAW G., O'SULLIVAN C.: Adaptive medial-axis approximation for sphere-tree construction. In *ACM Transactions on Graphics*, vol. 23(1). ACM press, 2004, pp. 1–26.

[BS08] BIRGIN E. G., SOBRAL F. N. C.: Minimizing the object dimensions in circle and sphere packing problems. *Computers & OR 35*, 7 (2008), 2357–2375.

[Cam97] CAMERON S.: Enhancing GJK: Computing minimum and penetration distances between convex polyhedra. In *Proceedings of International Conference on Robotics and Automation* (1997), pp. 3112–3117.

[CHHV06] COTTRELL M., HAMMER B., HASENFUSS A., VILLMANN T.: Batch and median neural gas. *Neural Networks 19* (jul 2006), 762–771.

[EL01] EHMANN S. A., LIN M. C.: Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *in Computer Graphics Forum* (2001), pp. 500–510.

[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).

[FL01] FISHER S. M., LIN M. C.: Fast penetration depth estimation for elastic bodies using deformed distance fields, 2001.

[Gär99] GÄRTNER B.: Fast and robust smallest enclosing balls. In *ESA* (1999), Nesetril J., (Ed.), vol. 1643 of *Lecture Notes in Computer Science*, Springer, pp. 325–338.

[GJK88] GILBERT E. G., JOHNSON D. W., KEERTHI S. S.: A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation 4* (1988), 193–203.

[HHV06] HAMMER B., HASENFUSS A., VILLMANN T.: Magnification control for batch neural gas. In *ESANN* (2006), pp. 7–12.

[HS04] HASEGAWA S., SATO M.: Real-time rigid body simulation for haptic interactions based on contact volume of polygonal objects. *Comput. Graph. Forum 23*, 3 (2004), 529–538.

[Hub95] HUBBARD P. M.: Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics 1*, 3 (Sept. 1995), 218–230.

[JC98] JOHNSON D. E., COHEN E.: A framework for efficient minimum distance computations. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-98)* (Piscataway, May 16–20 1998), IEEE Computer Society, pp. 3678–3684.

[LGLM99] LARSEN E., GOTTSCHALK S., LIN M., MANOCHA D.: Fast proximity queries with swept sphere volumes. In *Technical Report TR99-018* (1999).

[MBS93] MARTINETZ T. M., BERKOVICH S. G., SCHULTEN K. J.: 'Neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Trans. on Neural Networks 4*, 4 (1993), 558–569.

[MO06] MENDOZA C., O'SULLIVAN C.: Interruptible collision detection for deformable objects. *Computers & Graphics 30*, 3 (2006), 432–438.

[Mor06] MORRIS D.: Algorithms and data structures for haptic rendering: Curve constraints, distance maps, and data logging. In *Technical Report 2006-06* (2006).

[MPT99] MCNEELY W. A., PUTERBAUGH K. D., TROY J. J.: Six degrees-of-freedom haptic rendering using voxel sampling. In *Siggraph 1999* (Los Angeles, 1999), Rockwood A., (Ed.), Annual Conference Series, ACM Siggraph, Addison Wesley Longman, pp. 401–408.

[OG96] ONG C., GILBERT E.: Growth distances: New measures for object separation and penetration. *T-RA 12* (1996), 888–903.

[OH99] O'BRIEN J. F., HODGINS J. K.: Graphical modeling and animation of brittle fracture. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), ACM Press/Addison-Wesley Publishing Co., pp. 137–146.

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

[RL06] REDON S., LIN M. C.: A fast method for local penetration depth computation. *Journal of Graphics Tools: JGT 11*, 2 (2006), 37–50.

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

[Sch06] SCHUERMANN A.: On packing spheres into containers (about kepler's finite sphere packing problem). In *Documenta Mathematica* (Sept. 09 2006), vol. 11, pp. 393–406.

[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.

[vdB99] VAN DEN BERGEN G.: A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools: JGT 4*, 2 (1999), 7–25.

[vdB01] VAN DEN BERGEN G.: Proximity queries and penetration depth computation on 3d game objects. In *Game developers conference* (2001).

[ZKVM07] ZHANG L., KIM Y. J., VARADHAN G., MANOCHA D.: Generalized penetration depth computation. *Computer-Aided Design 39*, 8 (2007), 625–638.