EUROGRAPHICS 2004 / M.-P. Cani and M. Slater
(Guest Editors)

Volume 23 (2004), Number 3

Point Cloud Collision Detection

Jan Klein[1] and Gabriel Zachmann[2]

[1] Heinz Nixdorf Institute and Institute of Computer Science, University of Paderborn, Germany
[2] Department of Computer Science II, University of Bonn, Germany

Abstract

In the past few years, many efficient rendering and surface reconstruction algorithms for point clouds have been developed. However, collision detection of point clouds has not been considered until now, although this is a prerequisite to use them for interactive or animated 3D graphics.
We present a novel approach for time-critical collision detection of point clouds. Based solely on the point representation, it can detect intersections of the underlying implicit surfaces. The surfaces do not need to be closed.
We construct a point hierarchy where each node stores a sufficient sample of the points plus a sphere covering of a part of the surface. These are used to derive criteria that guide our hierarchy traversal so as to increase convergence. One of them can be used to prune pairs of nodes, the other one is used to prioritize still to be visited pairs of nodes. At the leaves we efficiently determine an intersection by estimating the smallest distance.
We have tested our implementation for several large point cloud models. The results show that a very fast and precise answer to collision detection queries can always be given.

Categories and Subject Descriptors (according to ACM CCS): G.1.2 [Numerical Analysis]: Approximation of surfaces and contours I.3.5 [Computer Graphics]: Computational Geometry and Object-Modeling[Geometric algorithms, languages and systems; object hierarchy; physically-based modeling] I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism[Animation; virtual reality]

1. Introduction

Point sets, on the one hand, have become a popular shape representation over the past few years. This is due to two factors: first, 3D scanning devices have become affordable and thus widely available [RHHL02]; second, points are an attractive primitive for rendering complex geometry for several reasons [PZvBG00, RL00, ZPvBG02, BWG03].

Interactive 3D computer graphics, on the other hand, requires object representations that provide fast answers to geometric queries. Virtual reality applications and 3D games, in particular, often need very fast collision detection queries. This is a prerequisite in order to simulate physical behavior and in order to allow a user to interact with the virtual environment.

So far, however, little research has been presented to make point cloud representations suitable for *interactive* computer graphics. In particular, there is virtually no literature on determining collisions between two sets of points.

In this paper, we present a novel algorithm to check whether or not there is a collision between two point clouds. The algorithm treats the point cloud as a representation of an implicit function that approximates the point cloud.

Note that we never explicitly reconstruct the surface. Thus, we avoid the additional storage overhead and an additional error that would be introduced by a polygonal reconstruction.

We also present a novel algorithm for constructing point hierarchies by repeatedly choosing a suitable subset. This incorporates a hierarchical sphere covering, the construction of which is motivated by a geometrical argument.

This hierarchy allows us to formulate two criteria that guide the traversal to those parts of the tree where a collision is more likely. That way, we obtain a *time-critical* algorithm that returns a “best effort” result should the time budget be exhausted. In addition, the point hierarchy makes it possible that the application can specify a maximum “collision detection resolution”, instead of a time budget.

The next section will review some of the work related to ours. Section 3 describes an overview of our approach, while Section 4 shows the details. Section 6 provides various results and benchmarks of our new approach. Finally, Section 7 draws some conclusions and describes possible avenues for further work.

2. Related Work

Although there is a wealth of literature on the problem of efficient high-quality rendering of point clouds, there is very little on geometric queries of such object representations.

An elementary query, at least, has been considered recently, namely intersecting point clouds with rays [AA03, SJ00, AA04]. The idea of [SJ00] is to intersect a ray with a set of discs (surfels). Although they use an octree as an acceleration structure, the performance seems to be significantly slower than using a polygonal reconstruction.

[AA03, AA04] intersect a ray with a continuously defined surface over a point cloud. The exact intersection point is approximated by using successive polynomial local reconstructions. They also construct a sphere covering, which is, however, not hierarchical. In addition, the exact construction of our hierarchical sphere covering is theoretically better motivated.

The work most related to ours is [AD03]. They present an algorithm to perform Boolean operations on solids. Although the problem of constructing a new solid by Boolean operations and the problem of detecting an intersection in a time-critical scenario are somewhat related, there are many obvious, significant differences. In addition, [AD03] represent objects by surfels. In contrast, we consider a continuous surface defined by a set of points. Furthermore, their approach can handle only solids, because they partition space in "inside" and "outside" by an octree. Our approach is general, i.e., it can handle non-closed geometry.

For efficiently rendering point clouds, most researchers have presented point hierarchies (see e.g. [RL00, DVS03, BWK02]. However, they are not well suited for collision detection.

For collision detection of rigid polygonal objects, bounding volume (BV) hierarchies have proven to be a very efficient data structure (e.g. [Hub96, GLM96, Zac02, vdB97]). Our new algorithm utilizes auxiliary BVs to enclose the points at different levels of the hierarchy. In principle, any of the BV types just mentioned can be used, so that one can choose the one which yields the best performance.

BV hierarchies lend themselves well to time-critical collision detection, where the scheduler interrupts the traversal when the time budget is exhausted [DO00, Hub96, KZ03]. However, these have been applied only to polygonal objects, and for point clouds the estimation of intersection probabilities must be done completely different.

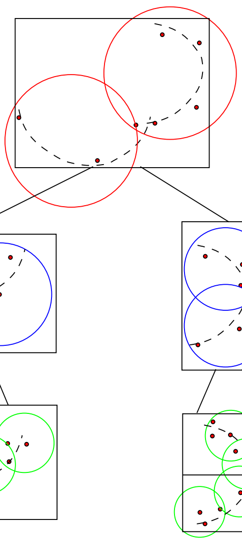


Figure 1: *Our approach constructs a point hierarchy, where each node stores a sample of the points underneath, which yields different levels of detail of the surface. In addition, we store a sphere covering of the surface of each node. Note that we compose a sphere covering of many more spheres.*

3. Overview of Our Approach

A point cloud P_A can be viewed as a way to define a function $f_A(\mathbf{x})$ such that the implicit $f_A(\mathbf{x}) = 0$ approximates P_A. Given two point clouds P_A and P_B, we pursue a hierarchical approach to quickly determine points $\mathbf{x}$ such that $f_A(\mathbf{x}) = f_B(\mathbf{x}) = 0$ by exploiting the spatial knowledge about the surface.

The idea of our algorithm is to create a hierarchy where the points are stored in its leaves. At each inner node, we store a sample of the point cloud underneath, a simple BV (such as a box), and a sphere covering for the part of the surface corresponding to the node (see Fig. 1). The point cloud samples effectively represent a simplified surface, while the sphere coverings define a neighborhood around it that contains the original surface.

The sphere coverings, on the one hand, can be used to quickly eliminate the possibility of an intersection of parts of the surface. The simplified point clouds, on the other hand, together with the sphere coverings, can be used to determine kind of a likelihood of an intersection between parts of the surface.

Given two such point cloud hierarchies, two objects can be tested for collision by simultaneous traversal (see Fig. 2), controlled by a priority queue. For each pair of nodes that still needs to be visited, our algorithm tries to estimate the likelihood of a collision, assigns a priority, and descends first into those pairs with largest priority. A pair of leaves is interrogated by a number of test points.

```
traverse(A, B)
if simple BVs of A and B do not overlap then
    return
if sphere coverings do not overlap then
    return
if A and B are leaves then
    return approx. distance between surfaces inside
for all children $A_i$ and $B_j$ do
    compute priority of pair $(A_i, B_j)$
traverse$(A_i, B_j)$ with largest priority first
```

Figure 2: *Outline of our hierarchical algorithm for point cloud collision detection.*

In order to make our point hierarchy memory efficient, we do not compute an optimal sphere covering, nor do we compute an optimal sample for each inner node. Instead, we combine both of them so that the sphere centers are also the sample.

4. Constructing the Data Structure

In this section, we will describe the construction of data structures being used by our hierarchical collision detection algorithm.

In the following, we treat the terms *bounding volume* (BV) and *node* of a hierarchy synonymous. A and B will always denote BVs of two different hierarchies.

For the sake of accuracy and conciseness, we introduce the following definitions.

Definition 1 (Cloud point) Each point of a given point cloud is denoted as a cloud point. The set of cloud points lying in BV A or its r_ε-border (see previous section) is denoted as P_A.

Definition 2 (Sample point) Each inner node A of our hierarchy stores a sample of all cloud points lying in A or its r_ε-border. These sample points are denoted as P'_A ($P'_A \subset P_A$).

Definition 3 (Test point) A test point is an arbitrary point that is not necessarily contained in a given point cloud.

4.1. Surface Definition

We define the surface of a point cloud implicitly based on moving least squares (MLS). For sake of completeness, we will give a quick recap of that surface definition in this section; please refer to [AA03] for the details.

4.1.1. Implicit distance function

Let N points $\mathbf{p}_i \in \mathbb{R}^3$ be given. Then, the implicit function $f : \mathbb{R}^3 \to \mathbb{R}$ describes the distance of a point $\mathbf{x}$ to a plane given by a point $\mathbf{a}(\mathbf{x})$ and the normal $\mathbf{n}(\mathbf{x})$:

$$f(\mathbf{x}) = \mathbf{n}(\mathbf{x}) \cdot (\mathbf{a}(\mathbf{x}) - \mathbf{x}) \tag{1}$$

The point $\mathbf{a}(\mathbf{x})$ is the weighted average

$$\mathbf{a}(\mathbf{x}) = \frac{\sum_{i=1}^{N} \theta(||\mathbf{x} - \mathbf{p}_i||)\mathbf{p}_i}{\sum_{i=1}^{N} \theta(||\mathbf{x} - \mathbf{p}_i||)} \tag{2}$$

and the normal $\mathbf{n}(\mathbf{x})$ is defined by moving least squares, i.e., $\mathbf{n}(\mathbf{x})$ minimizes

$$\sum_{i=1}^{N} \big(\mathbf{n}(\mathbf{x}) \cdot (\mathbf{a}(\mathbf{x}) - \mathbf{p}_i)\big)^2 \theta(||\mathbf{x} - \mathbf{p}_i||) \tag{3}$$

for fixed $\mathbf{x}$ and under the constraint $\|\mathbf{n}(\mathbf{x})\| = 1$. This is exactly the smallest eigenvector of matrix $\mathbf{B}$ with

$$b_{ij} = \sum_{k=1}^{N} \theta(\|\mathbf{x} - \mathbf{p}_k\|)(p_{k_i} - a(\mathbf{x})_i)(p_{k_j} - a(\mathbf{x})_j) \tag{4}$$

In the following, we will use the kernel

$$\theta(d) = e^{-d^2/h^2} \tag{5}$$

where the global parameter h (called *bandwidth*) allows us to tune the decay of the influence of the points, which is theoretically unbounded. More details are given in Section 5.5.

4.1.2. Horizon of influence

In practice, we consider a point $\mathbf{p}_i$ only if $\theta(||\mathbf{x} - \mathbf{p}_i||) > \theta_\varepsilon$, which defines a *horizon of influence* for each $\mathbf{p}_i$. However, now there are regions in $\mathbb{R}^3$ where only a small number of $\mathbf{p}_i$ are taken into account for computing $\mathbf{a}(\mathbf{x})$ and $\mathbf{n}(\mathbf{x})$. We amend this by dismissing points $\mathbf{x}$ for which the number c of $\mathbf{p}_i$ taken into account would be too small.Note that c and θ_ε are independent parameters. (We remark here that [AA04] proposed an amendment, too, although differently specified and differently motivated.)

Overall, the surface S is defined as the constrained zero-set of f, i.e.,

$$S = \big\{\mathbf{x} \mid f(\mathbf{x}) = 0\,, \#\{\mathbf{p} \in P : \|\mathbf{p} - \mathbf{x}\| < r_\varepsilon\} > c\big\} \tag{6}$$

where Equ. 5 implies $r_\varepsilon = h \cdot \sqrt{|\log \theta_\varepsilon|}$.

We approximate the distance of a point $\mathbf{x}$ to the surface S by $f(\mathbf{x})$. Because we limit the region of influence of points, we need to consider only the points inside a BV A plus the points within the r_ε-border around A, if $\mathbf{x} \in A$.

4.2. Point Cloud Hierarchy

In this section, we will describe a method to construct a hierarchy of point sets, organized as a tree, and a hierarchical sphere covering of the surface.

In the first step, we construct a binary tree where each node is associated with a subset of the point cloud. In order to do this efficiently, we recursively split the set of points by a top-down process. We create a leaf when the number of cloud points is below a threshold. We store a suitable BV with each node to be used during the collision detection process. Since we are striving for maximum collision detection

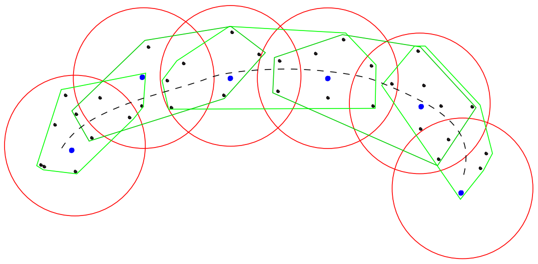


Figure 3: *The set of convex hulls induced by the leaves underneath an inner node of our hierarchy can be covered by spheres thus obtaining a neighborhood around P containing the surface.*

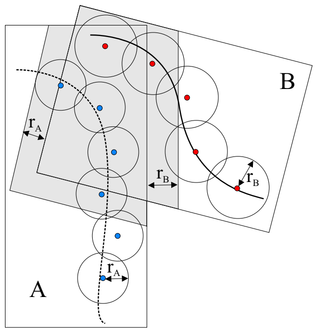


Figure 4: *Using the BVs and sphere coverings stored for each node, we can quickly exclude intersections of parts of the surfaces.*

performance, we should split the set so as to minimize the volume of the child BVs [Zac02].

Note that so far, we have only partitioned the point set and assigned the subsets to leaves.

In the second step, we construct a simplified point cloud and a sphere covering for each level of our hierarchy. Actually, we will do this such that the set of sphere centers are exactly the simplified point cloud. One of the advantages is that we need virtually no extra memory to store the simplified point cloud.

In the following, we will derive the construction of a sphere covering for one node of the hierarchy, such that the centers of the spheres are chosen from the points assigned to the leaves underneath. In order to minimize memory usage, all spheres of that node will have the same radius. (This problem bears some relationship to the general mathematical problem of thinnest sphere coverings, see [CS93] for instance, but here we have different constraints and goals.)

More specifically, let A be the node for which the sphere covering is to be determined. Let $L_1,\ldots,L_n$ be the leaves underneath A. Denote by P_i all cloud points lying in L_i or its r_ε-border, and let $\mathrm{CH}(P_i)$ be its convex hull. Let $P_A = \bigcup P_i$.

For the moment, assume that the surface in A does not have borders (such as intentional holes). Then

$$\forall \mathbf{x} \in L_i : \mathbf{a}(\mathbf{x}) \in \mathrm{CH}(P_i).$$

Therefore, if $\mathbf{x} \in A$ and $f(\mathbf{x}) = 0$, then $\mathbf{x}$ must be in $H = \bigcup_i \mathrm{CH}(P_i)$.

So instead of trying to find a sphere covering for the surface contained in A directly, our goal is to find a set $K = \{K_i\}$ of spheres, centered at $\mathbf{k}_i$, and a common radius r_A, such that $\mathrm{Vol}(K) = \mathrm{Vol}(\bigcup K_i)$ is minimal, with the constraints that $\mathbf{k}_i \in P_A$, K covers H, and bounded size $|K| \le c$ (see also Fig. 3). This problem can be solved by a fast randomized algorithm, which does not even need an explicit representation of the convex hulls (see below). (An exact solution would be an expensive combinatorial optimization problem over an exponential number of possible sets $\{\mathbf{k}_i\}$. In addition, constructing the convex hulls and minimal enclosing spheres are fairly expensive, too.) In Section 5.5, we will derive suitable bounds c on the size of K.

Therefore, we propose a randomized algorithm that first tries to determine a "good" sample $P'_A \subset P_A$ as sphere centers $\mathbf{k}_i$, and then computes an appropriate r_A. In both stages, the basic operation is the construction of a random point within the convex hull of a set of points, which is trivial.

4.2.1. Construction of P'_A:

The idea is to choose sample points $\mathbf{k}_i \in P_A$ in the interior of H so that the distances between them are of the same order. Then, a sphere covering using the $\mathbf{k}_i$ should be fairly tight and thin.

We choose a random point $\mathbf{q}$ lying in BV A; then, we find the closest point $\mathbf{p} \in P_A$ (this is equivalent to randomly choosing a Voronoi cell of P_A with probability depending on its size); finally, we add $\mathbf{p}$ to the set P'_A. We repeat this random process until P'_A contains the desired number of sample points (see Section 5.5). In order to obtain more evenly distributed $\mathbf{k}_i$'s, and thus a better P'_A, we can use quasi-random number sequences.

Since we want to prefer random points in the interior over points close to the border of H, we compute $\mathbf{q}$ as the weighted average of *all* points P_i of a randomly chosen L_i.

4.2.2. Determining r_A:

Conceptually, we could construct the Voronoi diagram of the $\mathbf{k}_i$, intersect that with $H = \bigcup_i \mathrm{CH}(P_i)$, determine the radius for the remainder of each Voronoi cell, and then take the maximum. Since the construction of the Voronoi diagram in 3D takes $O(n^2)$ (n = number of sites) [dBvKOS00], we propose a method similar to Monte-Carlo integration as follows.

Initialize r_A with 0. Generate randomly and independently

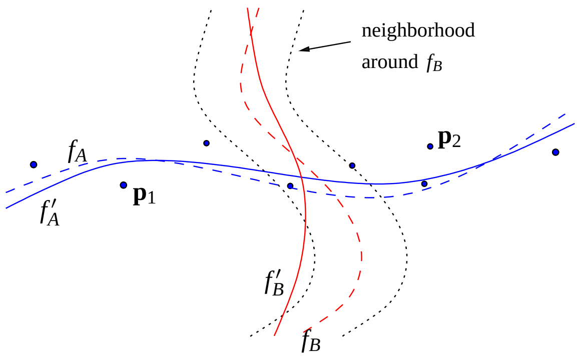


Figure 5: *Using the sample of two nodes and their r-neighborhoods, we can efficiently determine whether or not an intersection among the two nodes is likely.*

test points $\mathbf{q} \in H$. If $\mathbf{q} \notin K$, then determine the minimal distance d of $\mathbf{q}$ to P'_A, and set $r_A = d$. Repeat this process until a sufficient number of test points has been found to be in K.

In other words, we continuously estimate

$$\frac{\mathrm{Vol}(K \cap H)}{\mathrm{Vol}(H)} \approx \frac{\#\text{ points} \in K \cap H}{\#\text{ points} \in H} \tag{7}$$

and increase r_A whenever we find that this fraction is less than 1. In order to improve this estimate, we can apply kind of a stratified sampling: when $\mathbf{q} \notin K$ was found, we choose the next r test points in the neighborhood of $\mathbf{q}$ (for instance, by a uniform distribution confined to a box around $\mathbf{q}$).

5. Simultaneous Traversal of Point Cloud Hierarchies

In this section we will explain the details of the algorithm that determines an intersection, given two point hierarchies as constructed above.

5.1. Exclusion Criterion

Utilizing the sphere coverings of each node, we can quickly eliminate the possibility of an intersection of parts of the surface (see Fig. 4). Note that we do not need to test all pairs of spheres. Instead, we use the BVs of each node to eliminate spheres that are outside the BV of the other node.

5.2. A Priority Criterion

As mentioned above, we strive for a time-critical algorithm. Therefore, we need a way to estimate the likelihood of a collision between two inner nodes A and B, which can guide our algorithm shown in Fig. 2.

Assume for the moment that the sample points in A and B describe closed manifold surfaces $f_A = 0$ and $f_B = 0$, resp. Then, we could be certain that there is an intersection between A and B, if we would find two points on f_A that are on different sides of f_B.

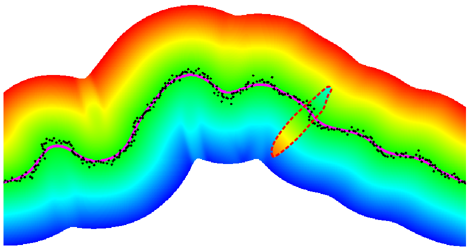


Figure 6: *Visualization of the implicit function $f(\mathbf{x})$ over a 2D noisy point cloud (black dots). Points $\mathbf{x} \in \mathbb{R}^2$ with $f(\mathbf{x}) \approx 0$, i.e., points on or close to the surface, are shown magenta. Red denotes $f(\mathbf{x}) \gg 0$ and blue denotes $f(\mathbf{x}) \ll 0$. The normal $\mathbf{n}(\mathbf{x})$ flips only across the red dashed line.*

Here, we can achieve only a heuristic. Assuming that the points P'_A are close to the surface, and that f'_B is close to f_B, we look for two points $\mathbf{p}_1, \mathbf{p}_2 \in P'_A$ such that $f'_B(\mathbf{p}_1) < 0 < f'_B(\mathbf{p}_2)$ (Fig. 5).

In order to improve this heuristic, we consider only test points $\mathbf{p} \in P'_A$ that are outside the r_B-neighborhood around f_B, because this decreases the probability that the sign of $f_B(\mathbf{p}_1)$ and $f_B(\mathbf{p}_2)$ is equal.

Overall, we estimate the likelihood of an intersection proportional to the number of points on both sides.

This argument holds only, of course, if the normal $\mathbf{n}_B(\mathbf{x})$ in Equation 1 does not "change sides" within a BV B. In our experience, fortunately, this appears to be rarely the case, in particular, if one uses the covariance matrix centered at $\mathbf{a}(\mathbf{x})$ as proposed in Equ. 4 (see Fig. 6).

5.3. Intersection Test in Leaf Nodes

When the traversal has reached two leaf nodes, A and B, we could just apply the traversal criterion again, and return an intersection if it is met.

Ideally, however, we would like to find a test point $\mathbf{p}$ such that $f_A(\mathbf{p}) = f_B(\mathbf{p}) = 0$ (where f_A and f_B are defined over P_A and P_B, resp.).

In practice, such a point cannot be found in a reasonable amount of time, so we generate randomly and independently a constant number of test points $\mathbf{p}$ lying in the sphere covering of object A (see left of Fig. 7). Then we take

$$d_{AB} \approx \min_{\mathbf{p}} \{ |f_A(\mathbf{p})| + |f_B(\mathbf{p})| \} \tag{8}$$

as an estimate of the distance of the two surfaces (see right of Figure 7), and report an intersection if $d_{AB} < d_\varepsilon$.

Note that it would not be sufficient to compute the distances between the points stored in A and B, because an intersection point is not necessarily close to a cloud point.

In order to obtain a better estimate, one could perform an

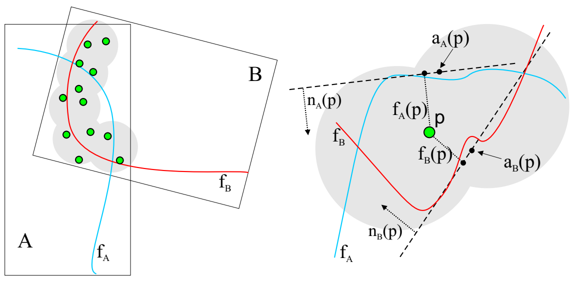


Figure 7: *In order to efficiently estimate the distance between the surfaces contained in a pair of leaves, we generate a number of random test points (left) and estimate their distance from A and B (right).*

iterative approximation of a pair of closest points on both surfaces, but in our experience, our simplistic approach has worked remarkably well.

5.4. Time-Critical Collision Detection

The traversal prioritization and the leaf intersection test described above facilitate a time-critical approach: on the one hand, if the time budget is exhausted, the collision detection process returns a "best effort" answer to the collision query. This is needed in time-critical applications where a real-time response is needed under all circumstances. On the other hand, if there is still time left, our algorithm can spend more time on the collision detection in leaf nodes to increase the accuracy.

This is done by trying to spend the same time $t_{\max}$ for each collision query by adjusting the number of test points and the distance ε that has to be found between the objects (see Section 5.3). If the time needed is larger than $t_{\max}$, the number of test points is gradually decreased and d_ε is increased, and vice versa otherwise.

5.5. Automatic Bandwidth Detection and Sampling Radius Estimation

Our algorithm has to evaluate $f(\mathbf{x})$ for subsets $P'_A \subset P$, which have different sampling densities. As a consequence, we should automatically adjust the bandwidth h (Section 4.1.1) to the sampling density, so that no holes appear higher up in our point cloud hierarchy. It is, of course, inevitable that intentional holes in the surface are closed at higher levels, but this just produces a few "false positives" during the traversal.

Definition 4 (Sampling radius) Consider a volume A, a point set P_A in A, and a subset $P'_A \subseteq P_A$. Consider a set of spheres, centered at P'_A, that cover the surface defined by P_A (not P'_A) that is inside A, where all spheres have equal radius. We define the sampling radius $r(P'_A)$ as the minimal radius of such a sphere covering.

Moreover, we can define the sampling radius for the set P_A as a special case with $P'_A = P_A$ in the definition above. And, as a special case of that, we define the sampling radius $r(P)$ of the whole point cloud P where A is the root BV in the definition above.

Since our surfaces do not interpolate the point cloud, we do not use the notion of ε-*sampling* [ACDL00]. In addition, we believe our definition is more practical.

Given $r(P'_A)$, we can determine the bandwidth h such that points up to a distance of about m times the sampling radius will have an influence in Equ. 1, if $\mathbf{x}$ is close to the surface:

$$h = \sqrt{-\frac{(r(P'_A)\cdot m)^2}{\log \theta_\varepsilon}} \tag{9}$$

with $\theta_\varepsilon < 1$. This follows from Equ. 5 and the notion of the horizon of influence (see Section 4.1.2).

Obviously, we could plug in r_A as sampling radius $r(P'_A)$ at inner nodes. However, this can be an overestimate, because the spheres, as constructed in Section 4.2, could cover intentional holes, which results in an imprecise h.

Alternatively, we estimate $r(P'_A)$ as follows. Assume that the implicit surface S_A of the point set P_A is approximated by surfels (2D discs) of equal size. Assume further that it is also approximated by surfels in P'_A, and that both P_A and P'_A do not contain significant discrepancies (i.e., the local sampling radius does not vary too much). Then, the surface area F_A can be estimated as

$$|P'_A|\,\pi r(P'_A)^2 = F_A = |P_A|\,\pi r(P_A)^2$$

from which follows

$$r(P'_A) = \sqrt{\frac{|P_A|}{|P'_A|}}\cdot r(P_A). \tag{10}$$

Using this, we can derive an estimate for the number of sample points needed in a node A in order to achieve a radius $r(P'_A) \le c\cdot r(P_A)$:

$$|P'_A| = |P_A|/c^2. \tag{11}$$

As a consequence, the sampling radius is at most $c\cdot r(P)$ throughout the point hierarchy, if $|P|/c^2$ points per node are stored. Then, the largest sampling radius $c\cdot r(P)$ can be found in the root node, while the sampling radius in the leaves is $r(P)$. For instance, if the sampling radius in every node is to be at most $50\cdot r(P)$ and the point cloud consists of 75 000 points, then at most 30 points per node need to be stored. In practice, $c = 50$ seems to be a good compromise between the number of points and the corresponding quality.

5.6. Fast Function Evaluation

During the BV traversal, Equation 1 needs to be evaluated many times. In order to achieve maximum performance, we use the following procedure.

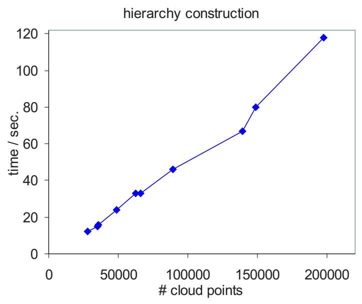


Figure 8: *This plot shows the build time of our point cloud hierarchies for various objects.*

First, we compute the three eigenvalues by determining the roots of the cubic characteristic polynomial of $\mathbf{B}$. [PFTV93] Let λ be the smallest of them. Then, we compute the associated eigenvector using the Cholesky decomposition of $\mathbf{B}-\lambda\mathbf{I}$.

The second step is possible because $\mathbf{B}-\lambda\mathbf{I}$ is positive semi-definite (because λ is the smallest of all eigenvalues). Thus, the Cholesky decomposition can be performed if full pivoting is done [Hig90].

In our experience, this method is faster than the Jacobi method by a factor of 4, and it is faster than singular value decomposition by a factor 8.

6. Results

We implemented our new algorithm in C++. As of yet, the implementation is not fully optimized. In the following, all results have been obtained on a 2.8 GHz Pentium-IV.

For timing the performance and measuring the quality, we have used a set of objects (see Fig. 13), most of them with varying complexities (with respect to the number of points). Benchmarking is performed by the procedure proposed in [Zac02], which computes average collision detection times for a range of distances between two identical objects.

6.1. Hierarchy Construction

Our point cloud hierarchies can be built in a fairly short time, so that the construction can be performed at startup time (see Fig. 8).

The memory consumption of a hierarchy is fairly low: with each node, we store a BV, a pointer to the child nodes, 1 float for the radius r_A, a constant number c_1 of pointers to the sample or to the cloud points lying in A, and also a number c_2 of pointers to points in the r_ε-border of the BV. That means, we need $32+4(c_1+c_2)$ bytes for each node. In practice, c_1+c_2 is between 15 and 30 so that at most 150 bytes per node is needed. For example, the hierarchy of our largest model (the grid) consisting of about 65,000 nodes consumes about 9 MB main memory. Of course, we also have to store the cloud points in main memory. Table 1 gives an overview of the number of generated sample points as well as the average depth of a node in the hierarchy.

6.2. Time and Quality

Each plot in Fig. 10 (left) and Fig. 11 (left) shows the average runtime for a model of our test suite, which is in the range 0.5–2.5 millisec (the two artificial models are considered later in this section). This makes our new algorithm suitable for real-time applications, and, in particular, physically-based simulation in interactive applications. Using our time-critical approach (see Section 5.4), the detection time can be decreased even further (e.g., when there are too many collisions going on).

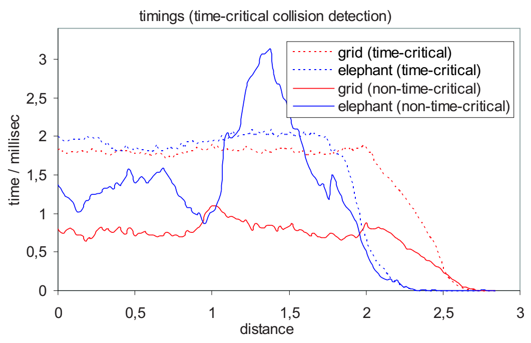
For each object of our test suite, we have also compared the outcome of our new algorithm with a traditional polygonal collision detection using a very high-resolution polygonal model. That way, we can give some experimental hints about the error probability of our new algorithm. Note that the polygonal models are *not* a tessellation of the true implicit surface, but just a tessellation of the given point cloud. The results in Fig. 10 (right) and Fig. 11 (right) show that the difference is always relatively low on average. For distances between 0.6 and 2 about 1.2% (happy buddha), 1.06%(elephant), 1.20% (aphrodite) and 0.64% (sharan) different answers are reported. Here, only collision tests were considered where at least the root BVs intersect. The differences can be explained by two facts: first implicit surface defined by the vertices of a polygonal object is obviously different from the polygonal model. Second, our intersection finding algorithm in the leaf nodes is very simplistic at the moment.

Equivalent measurements for our two artificial models can be found in Fig. 12. Note that the models have boundaries, and that the spheres model consists of several unconnected components. Obviously, our approach achieves results as good as for the other models. The grid model causes more differences compared to polygonal collision detection (up to 10%). Probably, this is because the surface definition in Section 4.1 is valid only for manifold objects.

6.3. Time-Critical Collision Detection

Our time-critical algorithm (Section 5.4) tries to spend the same amount of time $t_{\max}$ for each collision query. The results can be found in Fig. 9. Obviously, the time-critical algorithm always spends almost the same amount of time on the collision detection. Therefore, it can adapt the number of test points in the leaves of the point hierarchy much better.

Note further that the measured average collision query time is sometimes lower than $t_{\max}$ because sometimes a result can be achieved earlier, and sometimes the traversal doesn't reach any leaf nodes.



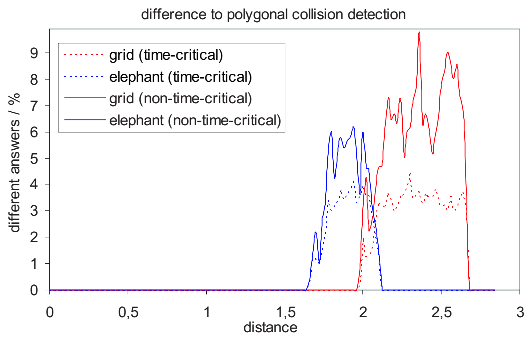


Figure 9: *Timings and differences using both the time-critical and non-time-critical algorithms. The differences measure not errors but the number of different reports from the point-based versus the polygonal collision detection algorithms.*

7. Conclusion and Future Work

In this paper we have presented a novel approach for collision detection of point clouds, which is, to the best of our knowledge, the first one. It works for non-closed surfaces.

We have described an efficient algorithm for constructing point hierarchies incorporating a hierarchical sphere covering that are well-suited for collision detection. Since our point cloud surfaces are based on moving least squares, we derive a method for automatic bandwidth and sampling rate detection for the different levels of the hierarchy. Based on the sphere covering and the implicit function, we have derived a criterion for prioritizing the traversal so as to find intersections more quickly. Finally, we have presented a time-critical collision detection algorithm that, given a time budget, returns a "best effort" result, so that we can guarantee a real-time response under all circumstances.

We have also performed several measurements that show that for all our models the collision queries can be done within a very short time, and with very little error.

There are many avenues for further work. On the one hand, performance and accuracy can be increased during the traversal of the point hierarchies. For instance, a faster convergence in the leaves towards an intersection point could be found. Also, at inner nodes, the traversal criteria can probably be improved, for instance, by applying knowledge about the separation of the eigenvalues.

On the other hand, the point hierarchy and the hierarchical sphere covering could be improved so as to allow for faster collision detection. For instance, the sphere covering could be made tighter, and the definition of the surface itself could be improved. Additionally, our method could be combined with the one presented by [CBC*01].

Acknowledgements

This work is partially supported by DFG grant DA155/29-1 "Benutzerunterstützte Analyse von Materialflusssimulationen in virtuellen Umgebungen" (BAMSI), and the DFG program "Aktionsplan Informatik" by grant ZA292/1-1.

References

[AA03] Adamson A., Alexa M.: Approximating and intersecting surfaces from points. In *Proc. of Eurographics Symposium on Geometry Processing (SPG-03)* (Aachen, Germany, June23–25 2003), pp. 230–239.

[AA04] Adamson A., Alexa M.: Approximating bounded, non-orientable surfaces from points. In *Shape Modeling International* (2004). accepted.

[ACDL00] Amenta N., Choi S., Dey T. K., Leekha N.: A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the sixteenth annual symposium on Computational geometry* (2000), ACM Press, pp. 213–222.

[AD03] Adams B., Dutré P.: Interactive boolean operations on surfel-bounded solids. In *Proc. SIGGRAPH* (July 2003), vol. 22, pp. 651–656.

[BWG03] Bala K., Walter B., Greenberg D. P.: Combining edges and points for interactive high-quality rendering. In *Proc. of SIGGRAPH* (July 2003), vol. 22, pp. 631–640.

[BWK02] Botsch M., Wiratanaya A., Kobbelt L.: Efficient high quality rendering of point sampled geometry. In *Proc. of Eurographics Workshop on Rendering* (2002), pp. 53 – 64.

[CBC*01] Carr J. C., Beatson R. K., Cherrie J. B., Mitchell T. J., Fright W. R., McCallum B. C., Evans T. R.: Reconstruction and representation of 3D objects with radial basis functions. In *Proc. of SIGGRAPH* (2001), pp. 67–76.

	Sharan	Happy Buddha	Aphrodite	Elephant	Grid
# cloud points	35,056	62,299	89,036	148,689	197,315
# sample points	45,012	90,068	134,376	180,180	360,404
avg. depth of a node	11	12	13	13	14

Table 1: *Comparison of the number of cloud points and sample points as well as the average depth of a node in the hierarchy (only objects with appreciably different values are listed).*

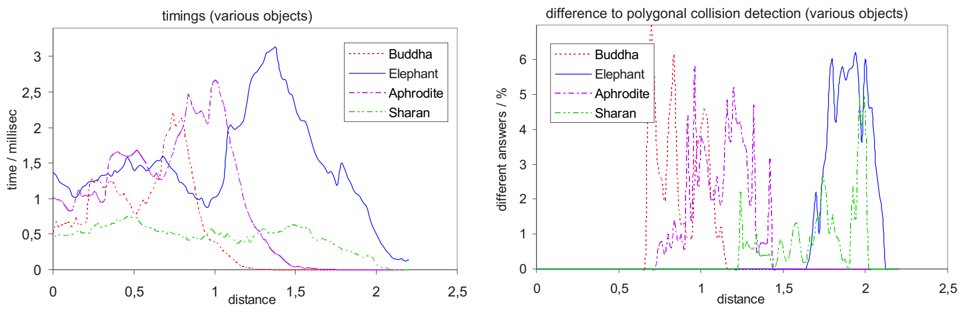


Figure 10: *Left: timings for different objects. Right: differences to polygonal collision detection of the objects; note that the polygonal models are* not *a tessellation of the true implicit surface, but just a mesh of the point cloud. The results for the teddy are very similar to that of the sharan, and are therefore omitted.*

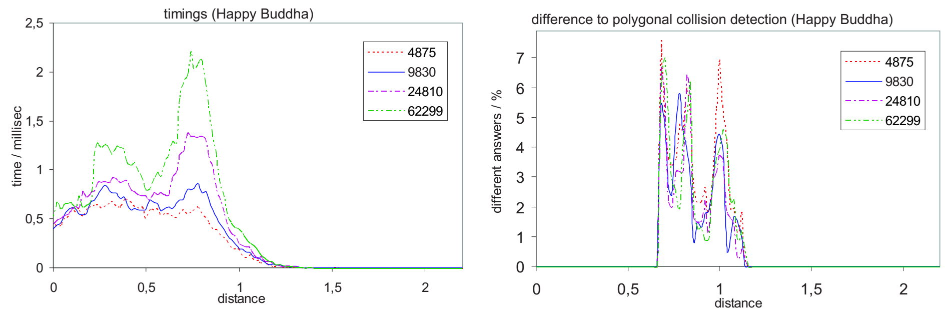


Figure 11: *Left: timings for different object complexities (# points) of the happy buddha model. Right: differences to polygonal collision detection of the objects.*

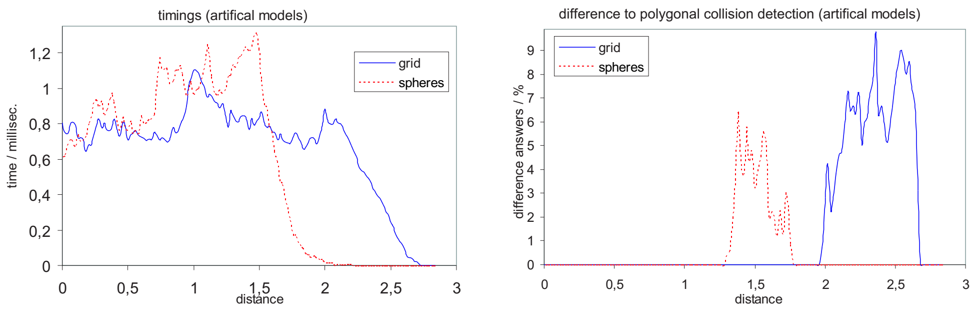


Figure 12: *Timings and difference to polygonal collision detection of artificial models. The differences can be reduced by increasing the number of test points if the time budget is not exhausted (see Fig. 9).*

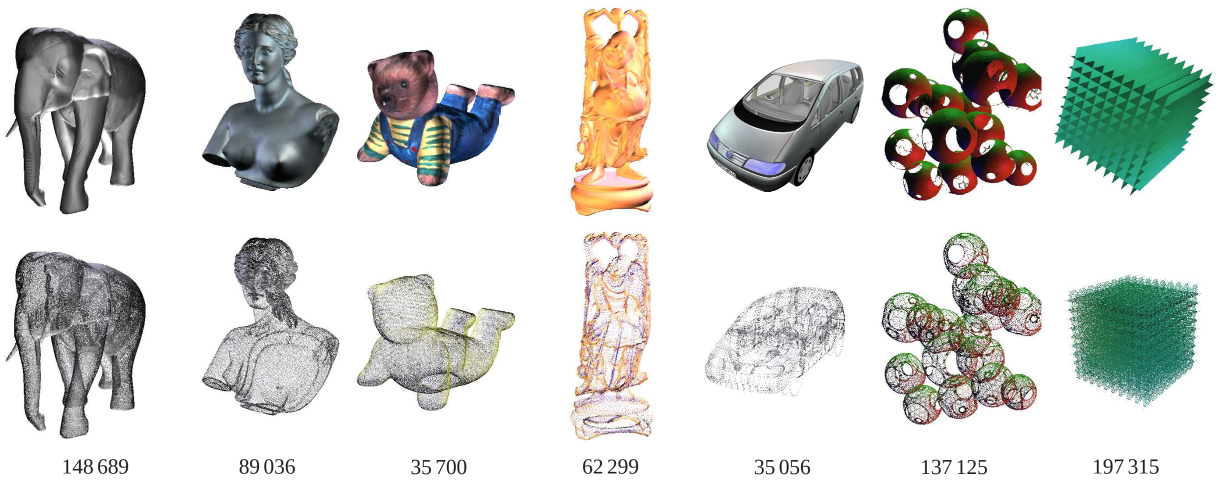


Figure 13: *Some of the models of our test suite, by courtesy of (left to right): Polygon Technology Ltd, Stanford, Volkswagen. The two artificial models (spheres and grid) show that our approach works well with non-closed geometry, too. The numbers are the sizes of the respective point clouds.*

[CS93] CONWAY J. H., SLOANE N. J. A.: *Sphere Packings, Lattices, and Groups*, 2 ed. Springer-Verlag, New York, 1993.

[dBvKOS00] DE BERG M., VAN KREVELD M., OVERMARS M., SCHWARZKOPF O.: *Computational Geometry: Algorithms and Applications*, 2nd ed. Springer-Verlag, Berlin, Germany, 2000.

[DO00] DINGLIANA J., O'SULLIVAN C.: Graceful degradation of collision handling in physically based animation. In *Proc. of EUROGRAPHICS* (Aug. 2000), vol. 19, pp. 239–247.

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential point trees. In *Proc. of the Siggraph (ACM Trans. on Graphics)* (San Diego, California, July 2003), vol. 22, pp. 657–662.

[GLM96] GOTTSCHALK S., LIN M., MANOCHA D.: OBB-Tree: A hierarchical structure for rapid interference detection. In *Proc. of SIGGRAPH* (Aug. 1996), pp. 171–180.

[Hig90] HIGHAM N. J.: Analysis of the Cholesky decomposition of a semi-definite matrix. In *Reliable Numerical Computation* (1990), Cox M. G., Hammarling S. J., (Eds.), Oxford University Press, pp. 161–185.

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

[KZ03] KLEIN J., ZACHMANN G.: Time-critical collision detection using an average-case approach. In *Proc. ACM Symp. on Virtual Reality Software and Technology (VRST)* (Osaka, Japan, Oct.1–3 2003), pp. 22–31.

[PFTV93] PRESS W. H., FLANNERY B. P., TEUKOLSKY S. A., VETTERLING W. T.: *Numerical Recipes in C*, 2nd ed. Cambridge University Press, Cambridge, England, 1993.

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. In *Proc. of SIGGRAPH* (2000), pp. 335–342.

[RHHL02] RUSINKIEWICZ S., HALL-HOLT O., LEVOY M.: Real-time 3D model acquisition. *ACM Transactions on Graphics 21*, 3 (July 2002), 438–446.

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: A multiresolution point rendering system for large meshes. In *Proc. of SIGGRAPH* (2000), pp. 343–352.

[SJ00] SCHAUFLER G., JENSEN H. W.: Ray tracing point sampled geometry. In *Proc. of Eurographics Workshop on Rendering Techniques* (Brno, Czech Republic, June26–28 2000), Springer Verlag, pp. 319–328.

[vdB97] VAN DEN BERGEN G.: Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools 2*, 4 (1997), 1–14.

[Zac02] ZACHMANN G.: Minimal hierarchical collision detection. In *Proc. ACM Symposium on Virtual Reality Software and Technology (VRST)* (Hong Kong, China, Nov. 2002), pp. 121–128.

[ZPvBG02] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: EWA splatting. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (July/Sept. 2002), 223–238.