# Collision Detection for Deformable Objects

M. Teschner[1], S. Kimmerle[2], B. Heidelberger[1], G. Zachmann[5], L. Raghupathi[4], A. Fuhrmann[6],
M.-P. Cani[4], F. Faure[4], N. Magnenat-Thalmann[3], W. Strasser[2], P. Volino[3]

[1] ETH Zurich, Switzerland [2] WSI/GRIS, Universität Tübingen, Germany [3] Miralab, University of Geneva, Switzerland

[4] GRAVIR/IMAG, INRIA Grenoble, France [5] Universität Bonn, Germany [6] Fraunhofer Institut, Darmstadt, Germany

**Abstract**

*Interactive environments for dynamically deforming objects play an important role in surgery simulation and entertainment technology. These environments require fast deformable models and very efficient collision handling techniques. While collision detection for rigid bodies is well-investigated, collision detection for deformable objects introduces additional challenging problems. This paper focusses on these aspects and summarizes recent research in the area of deformable collision detection. Various approaches based on bounding volume hierarchies, distance fields, and spatial partitioning are discussed. Further, image-space techniques and stochastic methods are considered. Applications in cloth modeling and surgical simulation are presented.*

## 1. Introduction

For many years, collision detection has been of major interest in computer graphics. Numerous approaches have been investigated to detect interfering objects in applications such as robotics, computational biology, games, surgery simulation, and cloth simulation. While many of the original collision detection methods especially address the problem of rigid bodies, recent approaches have started focusing on deformable objects.

Deformable collision detection is an essential component in interactive physically-based simulation and animation which is a rapidly growing research area with an increasing number of interesting applications. Fig. 1 illustrates one of the major applications for deformable collision detection, namely cloth simulation. In cloth simulation, approaches to dynamically deforming clothes have to be combined with efficient algorithms that handle self-collisions of cloth as well as the interaction of cloth with animated avatars. Such applications require collision detection algorithms that are especially appropriate for deformable and animated objects.

**Figure 1:** *In interactive cloth simulation, efficient deformable collision detection is a key component.*

Surgery simulation illustrated in Fig. 2 is a second major application area for deformable collision detection. In such environments, collisions among deformable organs have to be detected and resolved. Further, collisions between surgical tools and deformable tissue have to be processed. In the case of topological changes due to cutting, self-collisions of tissue can occur and have to be handled. Since interactive behavior of surgery simulation environments is essential, efficient algorithms for deformable collision detection are required.

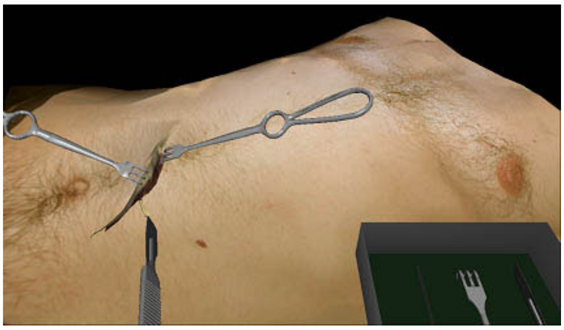


**Figure 2:** *Interactive environments for surgery simulation are one of the major application areas for deformable collision detection.*

In addition to cloth and surgery simulation, deformable collision detection methods are useful in environments with animated objects. Fig. 3 illustrates a sequence of two animated objects. In this example, specific collision detection algorithms for deformable objects can be used to detect interferences of both objects at interactive rates. Further, the intersection volume of both objects can be computed if their surfaces are closed.

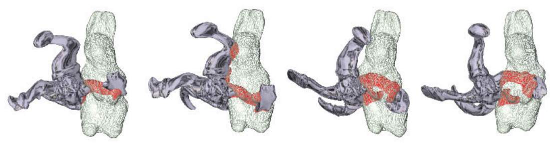


**Figure 3:** *A sequence of two animated objects: Santa and Rabbit. Specific deformable collision detection algorithms can be used to compute the intersection volume of both objects at interactive rates. The intersection volume is shown in red.*

If compared to collision detection approaches for rigid bodies, there are various aspects that complicate the problem for deformable objects.

**Collisions and Self-collisions:** In order to realistically simulate interactions between deformable objects, all contact points including those due to self-collisions have to be considered. This is in contrast to rigid body collision detection, where self-collisions are commonly neglected. Depending on the applications, rigid-body approaches can further be accelerated by only detecting one contact point.

**Pre-processing:** Efficient collision detection algorithms are accelerated by spatial data structures including bounding-box hierarchies, distance fields, or other ways of spatial partitioning. Such object representations are commonly built in a pre-processing stage, and once they are created, perform very well for rigid objects. However, in the case of deforming objects these pre-processed data structures have to be updated frequently. Therefore, pre-processed data structures are less efficient for deforming objects and their practicability has to be examined very carefully.

**Collision Information:** Collision detection algorithms for deformable objects have to consider, that a realistic collision response requires appropriate information. Therefore, it is not sufficient to just detect the interference of objects. Instead, precise information such as penetration depth of objects is desired.

**Performance:** In interactive applications, such as surgery simulation, games, and cloth simulation, the efficiency of collision detection algorithms for deformable modeling environments is especially important. Interactivity is a key characteristics in these applications, resulting in high demands for computing efficiency of collision detection algorithms.

In this paper, we discuss collision detection approaches that especially address the above-mentioned problems in order to meet the requirements of animation and simulation environments with dynamically deforming objects. Although all discussed algorithms are especially appropriate for deformable objects, they are not restricted to deformable objects, but also work with rigid bodies.

The remainder of the paper is organized as follows. Sec. 2 discusses bounding volume hierarchies. Special emphasis is placed on generating and updating the hierarchy. These aspects have to be optimized for deforming objects which require frequent hierarchy updates. In Sec. 3, stochastic methods are presented. These approaches are especially appropriate for interactive simulation environments, since they allow for balancing accuracy and performance. Sec. 4 describes the usage of distance fields for deformable collision detection. These approaches inherently provide information on the penetration depth of colliding objects which is important to compute a realistic collision response in physically-based simulations. Sec. 5 shows, how spatial subdivision can be employed for deformable collision detection. In these approaches, efficient data structures for representing 3D grids are especially important. Sec. 6 discusses recent image-space approaches. Since these methods commonly process projections of objects, they can be accelerated with graphics hardware. Sec. 7 presents applications in cloth modeling and surgery simulation. Finally, Sec. 8 summarizes advantages and drawbacks of the presented algorithms for deformable collision detection.

## 2. Bounding Volume Hierarchies

Bounding-volume hierarchies (BVHs) have proven to be among the most efficient data structures for collision detection. Mostly, they have been applied to rigid body collision detection.

Usually, a BVH is constructed for each object in a preprocessing step. The idea of BVHs is to partition the set of object primitives recursively until some leaf criterion is met. Most often, each leaf contains a single primitive, but one could as well stop when a node contains less than a fixed number of primitives. Here, primitives are the entities which make up the graphical objects, which can be polygons, NURBS patches, etc.

In general, BVHs are defined as follows: Each node in the tree is associated with a subset of the primitives of the object, together with a BV that encloses this subset with a smallest containing instance of some specified class of shapes. See [ZL03] for a thorough discussion of BVHs in general.

One of the design choices with BV trees is the type of BV. In the past, a wealth of BV types has been explored, such as spheres [Hub96, PG95], OBBs [GLM96], DOPs [KHM*98, Zac98], Boxtrees [Zac02, AdG*02], AABBs [vdB97, LAM01], spherical shells [KGL*98], and convex hulls [EL01].

Although a variety of BVs has been proposed (see Fig. 4), two types deserve special mention: OBBs and $k$-DOPs. Note, that AABBs are a special case of $k$-DOPs with $k = 6$. OBBs have the nice property that, under certain assumptions, their tightness increases linearly as the number of polygons decreases [GLM96]. $k$-DOPs, on the other hand, can be made to approximate the convex hull arbitrarily by increasing $k$. Further, $k$-DOPs, especially with $k = 6$, can be computed very efficiently. This is important, since deforming objects require frequent updates of a hierarchy.

### 2.1. Hierarchy Traversal

For the collision test of two objects or the self collision test of one object the BVHs are traversed top-down and pairs of tree nodes are recursively tested for overlap. If the overlapping nodes are leaves then the enclosed primitives are tested for intersection. If one node is a leaf while the other one is a internal node, the leaf node is tested against each of the children of the internal node. If, however, both of the nodes are internal nodes, it is tried to minimize the probability of intersection as fast as possible. Therefore, [vdB97] tests the node with the smaller volume against the children of the node with the larger volume (Fig. 6).

For two given objects with the BVHs $A$ and $B$, most collision detection algorithms implement the following general algorithm scheme:

```
traverse(A,B)
if A and B do not overlap then
    return
end if
if A and B are leaves then
    return intersection of primitives
            enclosed by A and B
else
    for all children A[i] and B do
        traverse(A[i],B)
    end for
end if
```

This algorithm quickly "zooms in" on pairs of nearby polygons. The characteristics of different hierarchical collision detection algorithms lie in the type of BV used, the overlap test for a pair of nodes, and the algorithm for construction of the BV trees.

### 2.2. Construction of Bounding Volume Hierarchies

For rigid bodies, the goal is to construct BVHs such that any subsequent collision detection query can be answered as fast as possible. Such BVHs are called optimal or good in the context of collision detection.

With deformable objects, the main goal is to develop algorithms that can quickly update or refit the BVHs after a deformation has taken place. At the beginning of a simulation, a good BVH is constructed for the undeformed object just like for rigid bodies. Then, during the simulation, often times the structure of the tree is kept, and only the extents of the BVs are updated. Due to the fact that DOPs are generally faster to update for deformable objects, they are preferred to OBBs.

Since the construction of a good initial BVH is important for deformable collision detection, we will discuss some of the issues in the following, while efficient ways of updating the hierarchy are discussed in Sec. 2.3.

There exist three different strategies to build BVHs, namely top-down, bottom-up [RL85], and insertion [GS87]. However, the top-down strategy is most commonly used for collision detection.

The idea is to recursively split a set of object primitives until a threshold is reached. The splitting is guided by a user-specified criterion or heuristic that will yield good BVHs with respect to the chosen criterion.

A very simple splitting heuristic is the following. [GLM96] approximates each polygon by its center. Then, for a given set $B$ of such points, compute its principal components (the eigenvectors of the covariance matrix); choose the largest of them (i.e., the one exhibiting the largest variance); place a plane orthogonal to that principal axis and through the barycenter of all points in $B$; this splits $B$ into two subsets. Alternatively, the splitting plane can be placed through the median of all points. This leads to a balanced tree. However, it is unclear, whether balanced trees provide improved efficiency of collision queries.

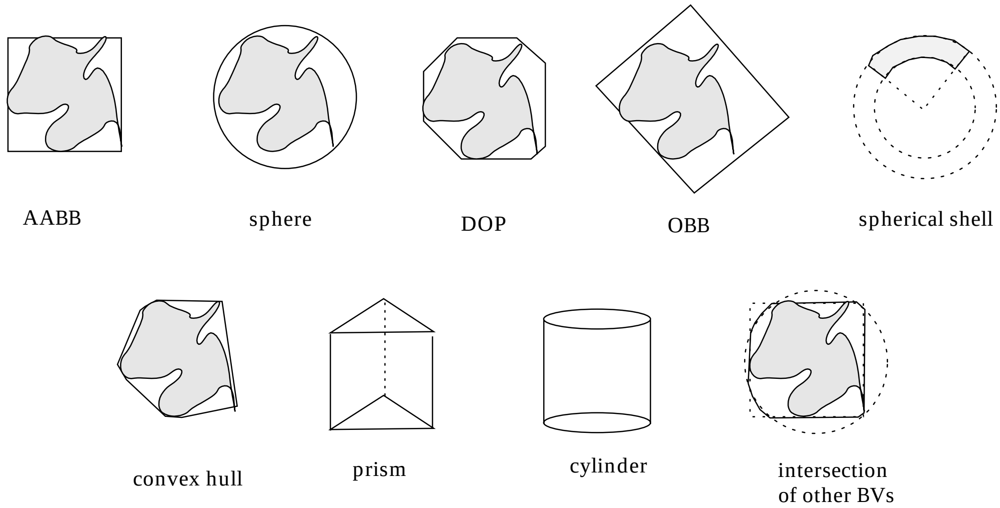



**Figure 4:** *A variety of bounding volumes has been proposed for hierarchy-based collision detection.*

Fig. 5 shows two hierarchy levels for the 18-DOP-hierarchy of an avatar, that was created top-down [MKE03].

For AABBs, it can be shown that any splitting heuristic should try to minimize the volumes of the children [Zac02]. Using Minkowski sums of BVs, one can estimate the geometric probability of a BV overlap during the traversal by

$$P(A_i, B_j) \approx \frac{\mathrm{Vol}(A_1) + \mathrm{Vol}(B_1)}{\mathrm{Vol}(A) + \mathrm{Vol}(B)} \quad (1)$$

in the case of AABBs. Since $\mathrm{Vol}(A) + \mathrm{Vol}(B)$ has already been committed by an earlier step in the recursive construction, Equation 1 can be minimized only by minimizing $\mathrm{Vol}(A_i) + \mathrm{Vol}(B_j)$.

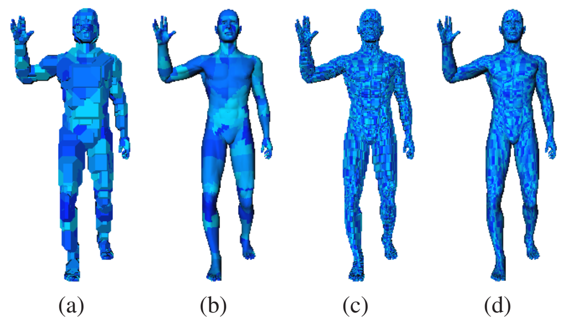



**Figure 5:** *Two levels of an 18-DOP-hierarchy. (a) and (c) illustrate the 18-DOPs, (b) and (d) show corresponding regions on the surface.*

Volino and Magnenat-Thalmann [VMT94, VMT95] as well as Provot [Pro97] use a fairly different approach for deformable objects. In this method, the hierarchy is strictly oriented on the mesh topology of the object, assuming that topology does not change during the simulation. Volino uses a region-merge algorithm to build the hierarchy bottom-up, while Provot uses a top-down algorithm that recursively divides the object in zones imbricating each other. These approaches have the advantage, that they avoid grouping faces close together in the hierarchy that are very close in the initial state, although they are not close at all based on the connectivity. This connectivity-based approach also yields advantages in speeding-up self-collision detection as described in subsection 2.4.

Another crucial point is the arity of the BVH. For rigid objects, binary trees are commonly chosen. In contrast, 4-ary trees or 8-ary trees have shown better performance for deformable objects [LAM01, MKE03]. This is mainly due to the fact that fewer nodes need to be updated and the total update costs are lower. Additionally, the recursion depth during overlap tests is lower and therefore the memory requirements on the stack are lower. Fig. 6 shows the reduction of recursion depth for detecting two overlapping leaves by equivalent 4-ary trees instead of binary trees.

### 2.3. Hierarchy Update

In contrast to hierarchies for rigid objects, hierarchies for deformable objects need to be updated in each time step. Principally, there are two possibilities: updating or rebuilding. Refitting is much faster than rebuilding, but for large deformations, the BVs usually are less tight and have larger overlap volumes. Nevertheless, van den Bergen [vdB97] has found that refitting is about ten times faster compared to a complete rebuild of an AABB hierarchy. Further, as long as the topology of the object is conserved there is no significant performance loss in the actual collision query compared to rebuilding.

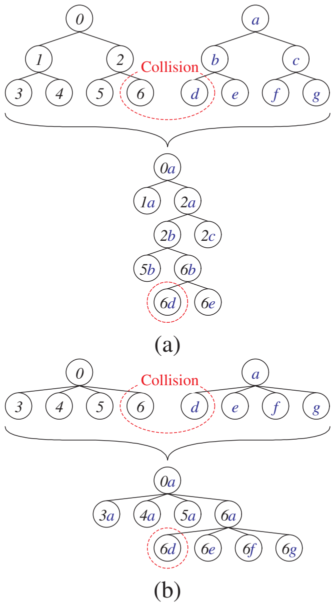



**Figure 6:** *Recursion using binary trees (a) and 4-ary trees (b).*

#### 2.3.1. General Updating

Different strategies have been proposed not only for building a hierarchy but also for the hierarchy update. Larsson and Akenine-Möller [LAM01] compared bottom-up and top-down strategies. They found that if in a collision detection process many deep nodes are reached the bottom-up strategy performs better, while if only some deep nodes are reached the top-down approach is faster. Therefore they proposed a hybrid method, that updates the top half of the tree bottom-up and only if non-updated nodes are reached these are updated top-down. Using this method they reduce the number of unnecessarily updated nodes with the drawback of higher memory requirement because they have to store the leaf information about vertices or faces also in the internal nodes.

Other approaches have been proposed by Mezger et al. [MKE03] to further accelerate the hierarchy update by omitting or simplifying the update process for several time steps. For this purpose the bounding volumes can generally be inflated by a certain distance. Then the hierarchy update is not needed as long as the enclosed primitives did not move farther than that distance.

#### 2.3.2. Refitting for Morphing Objects

An important case of deformation occurs in animation systems, where objects are deformed by morphing or blending. Here, in-between objects are constructed by interpolating between two or more morph targets. This usually requires the target models to have the same number of vertices and the same topology.

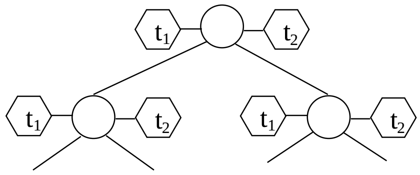



**Figure 7:** *If the deformation is a predefined morph, then a BVH for in-between objects can be constructed by morphing the BVs.*

The idea of [LAM03] is to construct one BVH for one of the morph targets and fit this to the other morph targets, such that corresponding nodes contain exactly the same vertices. With each node of the BVH, all corresponding BVs are stored, i. e. one from each morph target (see Fig. 7). During runtime, a BVH can be constructed for the morphed object just by considering the original BVH and interpolating the BVs. Assume, $n$ morph targets $O^i$ are given, each with vertices $v^i_j$ and weight vectors $w^i$. Then, each vertex $\bar{v}_j$ of the morphed object is an affine combination

$$\bar{v}_j = \sum_{i=1}^{n} w^i_j v^i_j, \quad \text{with} \quad \sum_{i=1}^{n} w^i_j = 1. \tag{2}$$

Let $D^i$ be the $n$ BVs of the corresponding nodes in the BVHs of the $O^i$ (i.e., all $D^i$ contain the same vertices, albeit at different positions). A DOP is denoted by $D^i = (S^i_1, \ldots, S^i_k)$, where $S^i_j = (s_j, e_j)$, $s_j \leq e_j$, is one interval of the DOP. Let $b^l, l = 1 \ldots k$, denote the set of orientations, over which all DOPs are defined. Now, a new DOP $\bar{D} = (\bar{S}_1, \ldots, \bar{S}_k)$ with $\bar{S}_j = (\bar{s}_j, \bar{e}_j)$ can be interpolated from these $n$ DOPs by

$$\bar{s}_j = \sum_{i=1}^{n} w_i s^i_j, \quad \bar{e}_j = \sum_{i=1}^{n} w_i e^i_j. \tag{3}$$

This interpolated DOP $\bar{D}$ will enclose all the interpolated vertices beneath its node. This works just the same for AABBs, since AABBs are a special case of DOPs. A similar approach works for sphere trees.

Overall, we can utilize existing top-down BVH traversal algorithms for collision detection. The only additional work that must be done is the interpolation, i.e., morphing of the BVs, just before they are checked for overlap. The exact positions of the morphed vertices enclosed by a BV are not needed. This deformable collision detection algorithm seems to be faster in practical cases, and its performance depends much less on the polygon count, than the more general method presented in [LAM01].

The method does have a few drawbacks, though: one must find a BVH that yields good performance for *all* in-between models; and, it works only for morphing schemes which allow only *one* weight per morph target.

### 2.4. Self-Collision Detection

BVHs can be easily employed to accelerate self-collisions. As already mentioned in Sec. 1, this is particularly important for deformable objects, such as cloth. In general, collisions and self-collisions are performed the same way using BVHs. If several objects are tested for collisions, the respective BVHs are checked against each other. Analogously, self-collisions of an object are detected by testing one BVH against itself.

However, it has to be noted, that BVs of neighboring regions can overlap, even though there are no self-collisions. To eliminate such cases efficiently, different heuristics have been presented. Volino and Magnenat-Thalmann [VMT94] proposed an exact method to avoid unnecessary self-intersection tests between certain BVs. Therefor, a vector with positive dot product with all face normals of the region is searched. If such a vector exists and the projection of the region onto a plane in direction of the vector does not self-intersect, the region cannot self-intersect.

Another approach is proposed by Provot [Pro97]. In this method, normal cones are introduced. The idea is based on the fact, that regions with sufficiently low curvature can not self-intersect, assuming they are convex. Therefore, a cone is calculated for each region. These cones represents a superset of the normal directions. They are built using the hierarchy and updated during the hierarchy update. The apex angle $\alpha$ of the cone represents the curvature, indicating possible intersections if $\alpha \geq \pi$.

### 2.5. Continuous Collision Detection

BVHs are also used to accelerate continuous collision detection, i. e. to detect the exact contact of dynamically simulated objects within two successive time steps. Therefore, BVs do not only cover object primitives at a certain time step. Instead, they enclose the volume described by the linear movement of a primitive within two successive time steps [BFA02, RKC02].

In multi-body systems, the number of collisions at a single time step can increase significantly, causing simple sign checking methods to fail. Therefore, [GK03] developed a reliable method that adjusts the step size of the integration by including the event functions in the system of differential equations, and by robust root detection.

Finding the first point of contact basically corresponds to finding roots of polynomials that describe the distance between the basic geometric entities, i. e. all face/vertex and all edge/edge pairs. These polynomials are easier to process if the motion of objects is a screw motion. Thus, [KR03, RKC02] approximate the general motion by a sequence of screw motions.

In order to quickly eliminate possible collisions of groups of polygons that are part of deformable objects, [MKE03] construct so-called velocity cones throughout their BVHs. Another technique sorts the vertices radially and checks the outer ones first [FW99].

A simple way to augment traditional, static BVH traversals is proposed in [ES99]. During the traversals, for each node a new BV is computed that encloses the static BV of the node at times $t_0$ and $t_1$ (and possibly several $t_i$ in-between). Other approaches utilize quaternion calculus to formulate the equations of motion [SSW95, Can86].

### 2.6. Conclusion

In BVH approaches, the efficiency of the basic BV has to be investigated very carefully. This is due to the fact, that deforming objects require frequent updates of the hierarchy. So far, it has been shown that AABBs should be preferred to other BVs, such as OBBs. Although OBBs approximate objects tighter than AABBs, AABBs can be updated or refit very efficiently. Additionally, 4-ary or 8-ary trees have shown a better overall performance compared to binary trees.

Although deformable modeling environments require frequent updates of BVHs, BVHs are nevertheless well-suited for animations or interactive applications, since updating or refitting of these hierarchies can be done very efficiently. Furthermore, BVHs can be employed to detect self-collisions while applying additional heuristics to accelerate this process. Also, BVHs work with triangles and tetrahedrons as object primitives, which allows for a more sophisticated collision response compared to a pure vertex-based response.

## 3. Stochastic Methods

Recently, "inexact" methods have become a focus in collision detection research. This idea is motivated by several observations. First, polygonal models are just an approximation of the true geometry. Second, the perceived quality of most interactive 3D applications does not depend on exact simulation, but rather on real-time response to collisions [US97]. At the same time, humans cannot distinguish between physically-correct and physically-plausible behavior of objects [BHW96]. Therefore, it can be tolerated to improve the performance of collision detection, while degrading its precision.

In the following, two of these "inexact" methods will be presented in detail. The two methods use probabilistic principles in fairly different ways. The first one uses probabilistic methods to estimate the possibility of collision with respect to a given quality criterion. With this method the "quality" of the collision detection can be specified by the user directly thus ensuring more control. The second method initially "guesses" colliding pairs by a stochastic sampling within the colliding bodies. The exact colliding regions are then narrowed down by using this principle in conjunction with temporal and spatial coherence. In this case, the user has a more indirect control over the quality of collision detection.

### 3.1. An Average-Case Approach

Conceptually, the main idea of this algorithm is to consider sets of polygons at inner nodes of the BVH. During traversal, pairs of these sets of polygons are checked [KZ03]. However, pairs of polygons are never explicitly checked. Therefore, there is no polygon information stored with the nodes of the BVH. Instead, the probability of the existence of a pair of intersecting polygons is estimated.

This has two advantages. First, the algorithm is truly time-critical. The application can control the runtime of the algorithm by specifying the desired quality of the collision detection. Second, the probabilities can guide the algorithm to those parts of the BV hierarchies that allow for faster convergence of the estimate.

In contrast to traditional traversal schemes, the algorithm is guided by the probability that a pair of BVs contains intersecting polygons. Omitting the details, the algorithm works as follows:

```
traverse(A,B)
while q is not empty do
  A,B ← q.pop
  for all children A_i and B_j do
    p ← Pr(A_i,B_j)
    if Pr(A_i,B_j) is large enough then
      return "collision"
    else if Pr(A_i,B_j) > 0 then
      q.insert(A_i,B_j,Pr(A_i,B_j))
    end if
  end for
end while
```

where $q$ is a priority queue, which is initialized with the top BV pair $(A,B)$. $Pr(A_i,B_j)$ denotes the probability of a collision between polygons under nodes $A_i$ and $B_j$. Note, that it is not possible to compute $Pr(A,B)$ exactly. Instead, it is estimated from the distribution of the polygons inside a grid, and combinatorial reasoning about the probability that a cell contains "many" polygons from both $A$ and $B$ (see Fig. 8).

Note, that this approach is a general framework that can be applied to many BVHs utilizing different types of BVs. The BVH has to be augmented by a single number: the number of cells in each node that contain many polygons. In addition, if one never wants to perform exact collision detection, then the polygons do not even have to be stored in the BVH.

### 3.2. Stochastic Collision Detection Based on Randomly Selected Primitives

A naïve approach to stochastic collision detection consists of selecting random pairs of colliding features as an initial

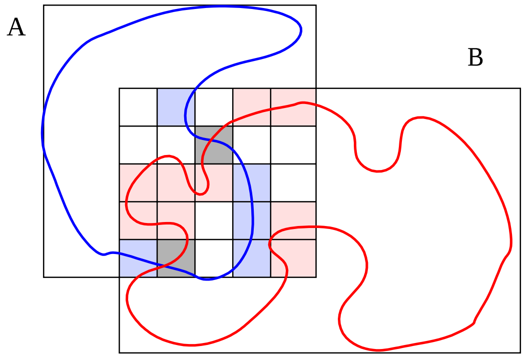



**Figure 8:** *Conceptually, the idea of the average-case approach is to determine the number of cells of a grid covering $A \cap B$ that contain "many" polygons from both A and B.*

guess of the potential intersecting regions. This method can be further augmented by ensuring that the sampling covers features from the entire body and that the features are already close enough. However, this is not sufficient to identify the colliding regions when the object moves or deforms. The solution is to consider temporal coherence proposed by Lin [LC92]. If a pair of features is close enough at a time step, it may still be interesting in the next one. This enables to follow these colliding regions over subsequent time steps as the objects are animated. Further, these pairs are made to converge to the local minima of the distance to efficiently identify collisions.

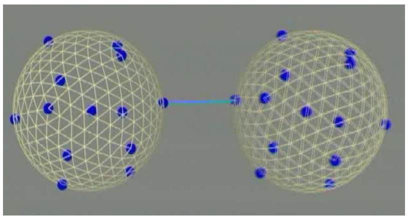
In addition, spatial coherence is also applied by keeping track of this local minimum over the neighborhood features. Each pair is locally updated at a time step, in order to track the local distance variations when the objects move or deform (see Fig. 10 (b)). These pairs are called active pairs. When two initially distant active pairs converge to the same local minimum, one of them is suppressed. A pair is also suppressed if the associated distance is greater than a given threshold. The above process tracks the existing regions of interest but does not detect new ones. This is a serious problem for non-convex object deformations where even a small motion can significantly alter the closest distance location.

Therefore, in addition to the update of the currently active pairs, $n$ additional random pairs are added to the list of active pairs at a time step. The update of these extra active pairs is again similar to the update of the existing ones. The complexity of the detection process thus linearly varies with the user-defined parameter $n$. At each time step, collision detection consists of selecting among the currently active pairs, the pairs which are closer that the sum of their radii. Reaction forces can then be generated between them.

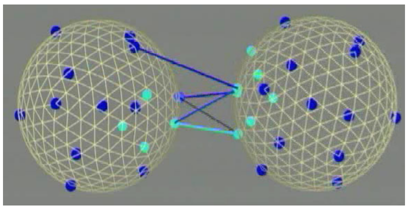
The general, the stochastic approach described above can be applied to several collision detection problems which are explained in Sec. 3.2.1 and Sec. 3.2.2.

#### 3.2.1. Volumetric Elastic Bodies

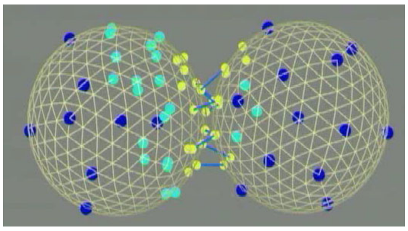
Multiresolution methods have proven to be efficient for the real-time simulation of deformable bodies. In such an environment, Debunne and Guy [GD02] applied a multiresolution, physically-based animation model [DDCB01] in conjunction with the above-described approach to help accelerate collision detection. Here, they start by tracking pairs of features at a coarser level when they are initially far apart and then refine to a finer level as they move closer. The switching between resolutions is done by evaluating a distance metric between pairs and applying spatial coherence between features at different resolutions. Fig. 9 illustrates this concept. If objects approach each other, the collision regions are tracked at different resolutions before it converges to the exact vertices of collisions.



(a)



(b)

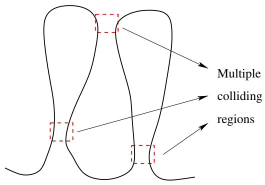


(c)

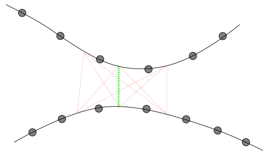
**Figure 9:** *Illustration of stochastic collision detection between two volumetric bodies: Pairs of features (depicted by lines) across different resolutions (different colored points) being tracked as the objects approach each other.*

#### 3.2.2. Thin Self-colliding Structures

As already mentioned, highly flexible structure, such as strands and cloth, have the possibility of self-colliding at multiple places (see Fig. 10 (a)). The general stochastic collision approach was adapted in [RCFC03] to detect collisions and self-collisions for such objects. The proposed method utilizes two optimizations. First, a two-step update method for computing the local distance minima for surface structures is used. This reduces the complexity from $O(n.m)$ to $O(n+m)$, where $n$ and $m$ are the number of neighboring primitives. Second, in order to provide a robust collision response, collisions are propagated from the collision point. When a collision occurs, a recursive algorithm searches the neighborhood for possible collisions and a unique response can be applied.




(a)



(b)

**Figure 10:** *(a) Detecting collisions or self-collisions between different folds of thin objects. (b) Tracking the local minima of the distance (shown by the dark dash line) by performing distance computations between neighboring pairs (shown by lighter dash line).*

### 3.3. Conclusions

Though the two discussed stochastic methods are principally different, they both have common characteristics. The most important one is the possibility to balance the quality of the collision detection or the collision ratio against computation time. It has been shown that stochastic approaches are applicable for real-time applications. However, it has to be considered, that no exact or physically-correct simulation is possible.

The first approach works on BVHs and can therefore extend many of the methods described in section 2. The second one is independent of any hierarchy and is directly employed to pairs of primitives which simplifies the collision response scheme.

## 4. Distance Fields

Distance fields specify the minimum distance to a closed surface for all points in the field. The distance may be signed in order to distinguish between inside and outside. Representing a closed surface by a distance field is advantageous because there are no restrictions about topology. Further, the evaluation of distances and normals needed for collision detection and response is extremely fast and independent of the complexity of the object.

Besides collision detection, distance fields have a wide range of applications. They have been used for morphing [BMWM01, COSL98], volumetric modeling [FPRJ00, BPK*02], motion planning [HKL*99] and recently for the animation of fire [ZWF*03]. Distance fields are sometimes called distance volumes [BMWM01] or distance functions [BMF03].

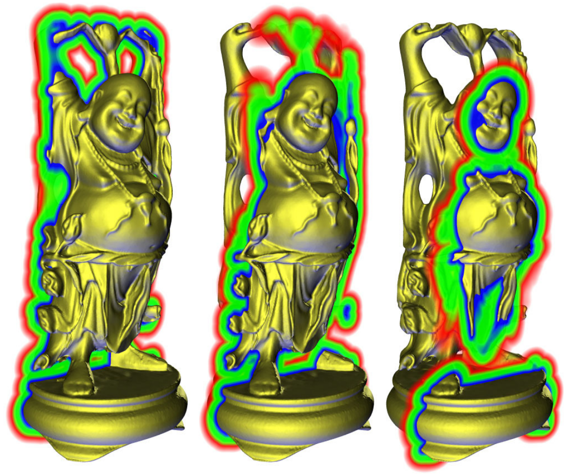


**Figure 11:** *Happy Buddha and three color-mapped distance field slices. Since the distance field is only valid within a band near the surface, the mapping is faded out at larger distance. Blue maps to close distances, whereas red indicates medium distances.*

A distance field $D : \mathbb{R}^3 \rightarrow \mathbb{R}$ defines a surface as the zero level set, i.e. $S = \{p|D(p) = 0\}$. In contrast to other implicit representations, a simple function evaluation also yields the Euclidean distance to the surface. Since a distance field stores a lot of information about a surface, the efficient computation of a distance field for a given surface representations is still a topic and some techniques are presented in the following section. In section 4.2, we discuss how distance fields can be used for collision detection between deformable objects and between rigid and deformable objects.

### 4.1. Distance Field Generation

Different data structures for representing distance fields have been proposed in the literature: Uniform 3D grids, Octrees and BSP-trees. When using uniform grids, distance values are computed for each grid point and intermediate values are reconstructed by trilinear interpolation. This data structure is easy to implement and distance queries can be computed in constant time. The later is important for real-time applications. Also, smooth objects can be represented quite well since a uniform grid provides $C^0$ continuity between different cells and the trilinear interpolation reconstructs smoothly curved surfaces inside each cell with a small approximation error. Most collision response schemes also need normals which can be computed by normalizing the analytic gradient of the trilinear interpolation [FPRJ00]. The drawbacks of uniform grids are the huge memory requirements and the limited resolution when representing objects with sharp features.

In order to overcome these problems, [FPRJ00] proposed to use adaptively sampled distance fields (ADFs). The data is stored in a hierarchy which is able to increase the sampling rate in regions of fine detail. Although various spatial data structures are suitable in general, ADFs are usually stored in an Octree. During construction of an ADF, each cell is subdivided as long as the result of the trilinear interpolation does not properly approximate the original distance field. This subdivision rule differs from standard 3-color Octrees where each cell, which is not completely inside or outside, is subdivided. This subdivision stops when a maximum tree depth is reached. Compared to uniform grids ADFs provide a good compression ratio. For collision detection purposes, special care has to be taken in order to guarantee continuity between different levels of the tree [BMF03]. Whenever a cell is adjacent to a coarser cell, its corner values have to be changed to match those of the interpolated values at the coarser cell [WKE99].

When using a BSP-tree, memory consumption can be reduced even further [WK03]. This is achieved by using a piecewise linear approximation of the distance field, which is not necessarily continuous. In [WK03], several algorithms for selecting appropriate splitting planes of the tree are provided and it is shown that the BSP representation is very compact. Unfortunately, the construction of the BSP-tree is computationally expensive. Another problem may arise from discontinuities between cells since these cracks are not as easily resolved as for ADFs.

In most applications, the surface of a collision object is given as a triangular mesh, possibly deforming over time. For collision detection, it is sufficient to compute distance values only in small band near the surface. Clearly, this reduces the computational effort considerably. Essentially, there are three different approaches for the efficient computation of a distance field: Methods based on Voronoi diagrams, propagation methods and methods using trees. When distances of each grid point are evaluated independently, a tree data structure can be used to cull away distant triangles to speed up computations. An early work of [PT92] applies BVHs. Later, Octrees have been used [JS01]. However, these approaches have shown not to be competitive compared to other methods since the computing times are in the order of minutes.

Propagation methods start with a narrow band of distances computed near the triangular surface. This initial information is distributed over the whole volume. Fast marching methods [Set96] and distance transforms are two examples of propagation methods. In [JS01], different types of distance transforms are compared. However, fast distance transforms are not very accurate.

In [HKL*99], the usage of graphics hardware for computing generalized 2D and 3D Voronoi diagrams is proposed. This technique builds distance meshes for each Voronoi site. In 2D, a simple rendering of these meshes for all sites yields an unsigned distance field in the depth buffer of the graphics hardware. In 3D, the distance meshes are quite complex and the algorithms has to proceed slice by slice. Thus, a huge number of triangles has to be rendered, slowing down this method considerably.

An algorithm with linear complexity is described in [BMWM01]. This method utilizes the Voronoi diagram for faces, edges and vertices of the mesh. Each Voronoi region is represented by a bounding polyhedron. The polyhedrons are cut into slices along grid rows and the resulting polygons are scan-converted in order to determine which grid points lie inside. The distance for inner grid points is easily computed as the distance to the Voronoi site. The orientation of the triangle mesh provides the correct sign for each region. If a grid point is scan-converted several times, the smallest distance is considered. Recently, [SPG03] improved on this algorithm by using graphics hardware to scan-convert the polyhedrons. Further, the authors show how to replace polyhedrons for faces, edges and vertices by constructing only one polyhedron per face. This reduces the number of polyhedra which have to be scan-converted by a factor of three. The authors demonstrated, that it is possible to compute the distance field for the Stanford Bunny, consisting of 69K triangles, within 3.7 seconds. The grid resolution was $256^3$ and the band width was 10% of the model extent.

In some cases, triangular meshes are stored without any adjacency information. Since these are needed for computing Voronoi regions, [FSG03] propose an algorithm which computes distance values independently for each triangle. Except for some sign errors, due to the lack of adjacency information, this technique is able to compute distance fields for finely tessellated objects very fast. It is shown that it takes only about 14 seconds to compute the distance field for the Happy Buddha model (1.1M triangles) on a $167 \times 167 \times 400$ grid.

### 4.2. Distance Field Collision Detection

Collision detection between different objects carried out point-wise when using distance fields. If both deformable objects are volumetric, vertices on the surface of one object are compared against the distance field of the other object and vice versa. A collision has occurred if $D(p) < 0$. When animating deformable surfaces over more or less rigid bodies, distance fields are in particular suitable. In this case, only the vertices of the deformable surface have to be tested for collisions. In order to avoid artifacts during collision response, it is necessary to offset the vertices from the zero isosurface by a predefined $\varepsilon$ (see Fig. 12). In this case, a vertex is marked as collided when $D(p) < \varepsilon$. This $\varepsilon$-offset depends on the sampling density of the deformable objects. Note, that distance fields do not only report collisions, but also compute the penetration depth at the same. This is required for a proper collision response algorithm.

Deformed distance fields have been used to estimate the

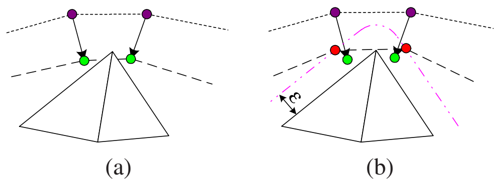



**Figure 12:** *(a) Without offsetting the vertices interpenetration artifacts may occur during collision detection. (b) Introducing an $\varepsilon$-offset solves the problem.*

penetration depth of elastic polyhedral objects [FL01]. In this method, an internal distance field is created by a fast marching level set method, propagating distance information only inside the objects. In order to take deformations of the objects into account, the distance fields are updated before each time step. The actual collision detection is carried out by a hierarchical method. During collision response the distance fields are deformed due to the geometry and used for an approximation of the penetration depth. This method is able to handle self-collisions and collisions. The approach is well-suited for volumetric objects, because thin objects like cloth are not well represented by internal distance fields. Although the distance field is only partially updated in deforming regions of the object, experiments show that this method is not intended for real-time applications.

[BMF03] suggest to use ADFs for detecting collisions between clothing and animated characters. Since the surface deforms over time due to skin and muscle simulation, the authors propose to pre-calculate a distance field at each time step using a fast marching method [Set96]. This can be used for multiple cloth simulations. In order to handle cloth with several intersecting character geometries, a distance field for each body part is created. Thus, cloth vertices, which are inside of two or more distance fields, can be detected and handled properly. For a more thorough analysis of pinched clothing see [BWK03]. No timings were given in the paper, but the necessity of pre-calculation of the distance fields indicates that the proposed approach is tailored for film making.

In [FSG03], the problem of rapid distance computation between rigid and deformable objects is addressed. Rigid objects are represented by distance fields, which are stored in a uniform grid to maximize query performance. Vertices of a deformable object, which penetrate an object, are quickly determined by evaluating the distance field. Additionally, the center of each edge in the deforming mesh is tested in order to improve the precision of collision detection. Since updates of the distance field were considered to costly, the authors also propose to combine multiple rigid bodies for animation—similar to the method of [BMF03]. Experiments suggest, that this technique is able to animate cloth at interactive rates (see Fig. 13). Collisions with a complex non-convex objects are resolved accurately and also self-collisions are treated by using a different approach.

Updates of a distance field are a common bottleneck of all methods described above. To accelerate theses updates, [VSC01] proposed an image-based approach for computing distance values. Rendering hardware is used for constructing two depth and normal maps of the object, one map for the back of the object and one map for its front. These maps are used for distance calculations and collision response. This algorithm is restricted to convex shapes or appropriate mapping directions in the case of animated characters. In [IZLM01], another image-based approach is proposed that uses the technique described in [HKL*99] to create a 2D distance field using graphics hardware. This 2D distance field is used for collision response. The authors report interactive frame rates for non-convex rigid bodies and also for deformable bodies. The method is applicable to 2D problems.

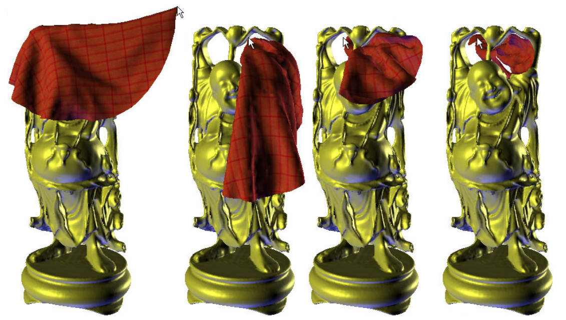


**Figure 13:** *Interactive animation of cloth in a complex collision environment.*

### 4.3. Conclusion

Distance fields have been employed to detect collisions and even self-collisions in non-interactive applications. Although efficient algorithms for computing distance fields have been proposed recently, this generation is still not fast enough for interactive applications, where distance fields have to be updated during run-time due to deforming geometry. However, distance fields provide a highly robust collision detection, since they divide space strictly into inside and outside.

For interactive applications, distance fields can be used to represent all rigid objects contained within the environment. Since distance fields yield not only the penetration depth but also normals needed for collision response at interactive rates, collision detection between deformable objects and the rigid objects is carried out very efficiently. In order to decrease storage requirements or generation time, it is possibly to reduce the resolution of the distance field, which results in lowered accuracy. Thus, distance field approaches can balance performance and accuracy.

## 5. Spatial Subdivision

There exist various approaches that propose spatial subdivision for collision detection. These algorithms employ uniform grids [Tur90], [GDO00], [ZY00] or BSPs [Mel00]. In [Tur90], spatial hashing for collision detection is mentioned for the first time. In [Mir97], a hierarchical spatial hashing approach is presented as part of a robot motion planning algorithm, which is restricted to rigid bodies.

In [THM*03], spatial hashing is employed for the detection of collisions and self-collisions for deformable tetrahedral meshes. Tetrahedral meshes are commonly used in medical simulations, but can also be employed in any physically-based environment for deformable objects that are based on FEM, mass-spring, or similar mesh-based approaches (see Fig. 14 and 15). This algorithm implicitly subdivides $\mathbb{R}^3$ into small grid cells. Instead of using complex 3D data structures, such as octrees or BSPs, the approach employs a hash function to map 3D grid cells to a hash table. This is not only memory efficient, but also provides flexibility, since this allows for handling potentially infinite regular spatial grids. Information about the global bounding box of the environment is not required and 3D data structures are avoided.

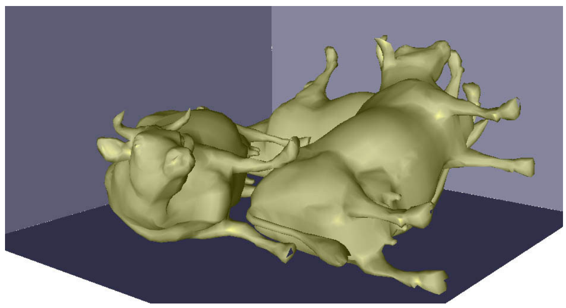


**Figure 14:** *Interactive environment with deformable objects which are represented with tetrahedral meshes. Spatial hashing is employed to detect collisions and self-collisions.*

The algorithm, presented in [THM*03], proceeds in three stages. In a first pass, the information on the implicit 3D grid cells of all vertices are mapped to the hash table. The second pass considers all tetrahedrons of the environment. It maps information on all grid cells touched by a tetrahedron to the hash table. Now, the third stage checks vertices and tetrahedrons within a hash table entry for intersections. If a vertex penetrates a tetrahedron, a collision is detected. If both, the vertex and the tetrahedron belong to the same object, a self-intersection is detected.

Experiments with various setups of deformable objects have been performed (see Tab. 1 and Tab. 2). Setups A, B

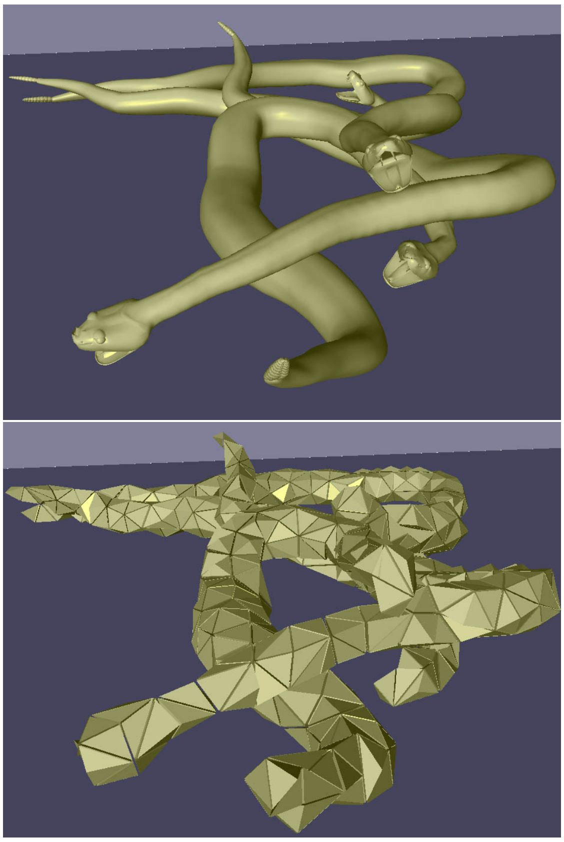


**Figure 15:** *Interactive environment with dynamically deforming objects and collision handling. Surface with high geometric complexity and the underlying tetrahedral mesh are shown.*

and D are illustrated in Fig. 16 and 17. Setup C and E use the same deformable objects like B.

**Table 1:** *The following setups with dynamically deforming objects have been tested in [THM*03].*

| setup | objects | tetras | vertices |
|---|---|---|---|
| A | 100 | 1000 | 1200 |
| B | 8 | 4000 | 1936 |
| C | 20 | 10000 | 4840 |
| D | 2 | 20514 | 5898 |
| E | 100 | 50000 | 24200 |

Experiments indicate, that the detection of all collisions and self-collisions for dynamically deforming objects can be performed at 15 Hz with up to 20k tetrahedrons and 6k vertices on a standard PC. The performance is independent of the number of objects. It only depends on the number of object primitives. The performance varies slightly during simulations due to the changing number of hash collisions and a varying distribution of hash table elements.

**Table 2:** *Performance of collision detection based on spatial subdivision (see [THM*03]). Setups are described in Tab. 1. Average collision detection time, minimum, maximum, and standard deviation for 1000 simulation step are given.*

| setup | ave [ms] | min [ms] | max [ms] | dev [ms] |
|---|---|---|---|---|
| A | 4.3 | 4.1 | 6.5 | 0.24 |
| B | 12.6 | 11.3 | 15.0 | 0.59 |
| C | 30.4 | 28.9 | 34.4 | 1.25 |
| D | 70.0 | 68.5 | 72.1 | 0.86 |
| E | 172.5 | 170.5 | 174.6 | 1.08 |

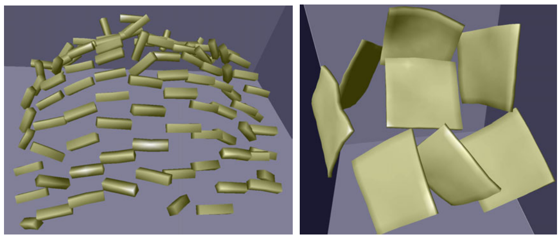


**Figure 16:** *Test setup A (left) and B (right).*

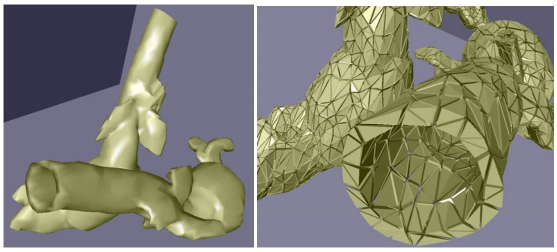


**Figure 17:** *Test setup D (left) and tetrahedral mesh of setup D.*

## 5.1. Conclusion

Spatial subdivision is a simple and fast technique to accelerate collision detection in case of moving and deforming objects. Spatial subdivision can be used to detect collisions and self-collisions. Algorithms based on spatial subdivision are independent of topology changes of objects. They are not restricted to triangles as basic object primitive, but also work with other object primitives if an appropriate intersection test is implemented.

The main difficulty in spatial subdivision is the choice of the data structure that is used to represent the 3D space. This data structure has to be flexible and efficient with respect to computational time and memory. In [THM*03], a hash table is used which has shown to be very efficient in physically-based simulation environments.

## 6. Image-Space Techniques

Recently, several image-space techniques have been proposed for collision detection [MOK95], [BWS99], [BW02], [IZLM01], [KOLM02], [HTG03], [KP03], [GRLM03]. These approaches commonly process projections of objects to accelerate collision queries. Since they do not require any pre-processing , they are especially appropriate for environments with dynamically deforming objects. Furthermore, image-space techniques can commonly be implemented using graphics hardware.

An early approach to image-space collision detection of convex objects has been outlined in [SF91]. In this method, the two depth layers of convex objects are rendered into two depth buffers. Now, the interval from the smaller depth value to the larger depth value at each pixel approximately represents the object and is efficiently used for interference checking. A similar approach has been presented in [BWS99]. Both methods are restricted to convex objects, do not consider self-collisions, and have not explicitly been applied to deforming objects.

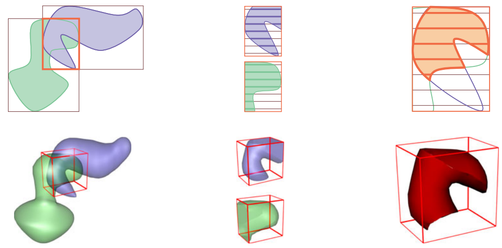
In [MOK95], an image-space technique is presented which detects collisions for arbitrarily-shaped objects. In contrast to [SF91] and [BWS99], this approach can also process concave objects. However, the maximum depth complexity is still limited. Additionally, object primitives have to be pre-sorted. Due to the required pre-processing, this method cannot efficiently work with deforming objects. Self-collisions are not detected.

A first application of image-space collision detection to dynamic cloth simulation has been presented in [VSC01]. In this approach, an avatar is rendered from a front and a back view to generate an approximate representation of its volume. This volume is used to detect penetrating cloth particles. A first image-space approach to collision detection in medical applications is presented in [LCN99], where intersections of a surgical tool with deformable tissue are detected by rendering the interior of the tool.

In [IZLM01], an image-space method is not only employed for collision detection, but also for proximity tests. This method is restricted to 2D objects. In [KOLM02] and [KmLD03], closest-point queries are performed using bounding-volume hierarchies along with a multipass-rendering approach. In [KP03], edge intersections with surfaces can be detected in multi-body environments. This approach is very efficient. However, it is not robust in case of occluded edges. In [GRLM03], several image-space methods are combined for object and sub-object pruning in collision detection. The approach can handle objects with changing topology. The setup is comparatively complex and self-collisions are not considered.

In [HTG03], an image-space technique is used for collision detection of arbitrarily-shaped, deformable objects. This approach computes a Layered Depth Image LDI [SGwHS98] for an object to approximately represent its volume. This approach is similar to [SF91], but not restricted to convex objects. Still, a closed surface is required in order to have a defined object volume.

The algorithm presented in [HTG03] proceeds in three stages. First, the intersection of axis-aligned bounding boxes of pairs of objects is calculated. If an intersection is non-empty, a second stage computes an LDI representation of each object within the bounding-box intersection. Finally, two volumetric collision queries can be applied. The first query detects intersecting volumes of two objects, while the second query detects vertices or other object primitives that penetrate another object. Self-collisions cannot be found. Fig. 21 illustrates the three stages of the algorithm.



(a) Stage 1. (b) Stage 2. (c) Stage 3a.

**Figure 21:** *Image-space collision detection in 2-D and 3-D. (a) AABB intersection. (b) LDI generation within the VoI. (c) Computation of the intersection volume.*

In [HTG04], an improved algorithm is presented. In contrast to existing approaches that do not consider self-collisions, this approach combines the image-space object representation with information on face orientation to overcome this limitation.

LDIs are the basic data structure in many image-space collision detection approaches. Therefore, it is very important to optimize their computation and [HTG04] provides a comparison of three different implementations for LDI generation. Two implementations based on graphics hardware and one software solutions have been compared. Results suggest, that graphics hardware accelerates image-space collision detection in geometrically complex environments, while CPU-based implementations provide more flexibility and better performance in case of small environments.

In Fig. 18 and Fig. 19, collisions and self-collisions are computed using a software implementation. Computing times are 8-11 ms using a standard PC. In this environment with low geometric complexity, a CPU-based implementation for LDI generation is more efficient compared

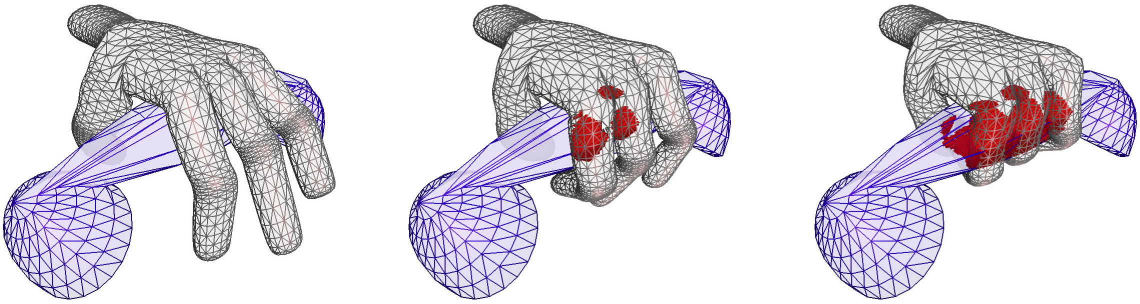


**Figure 18:** *Dynamic animation of a hand with 4800 faces and a phone with 900 faces. Collisions (red) are detected.*

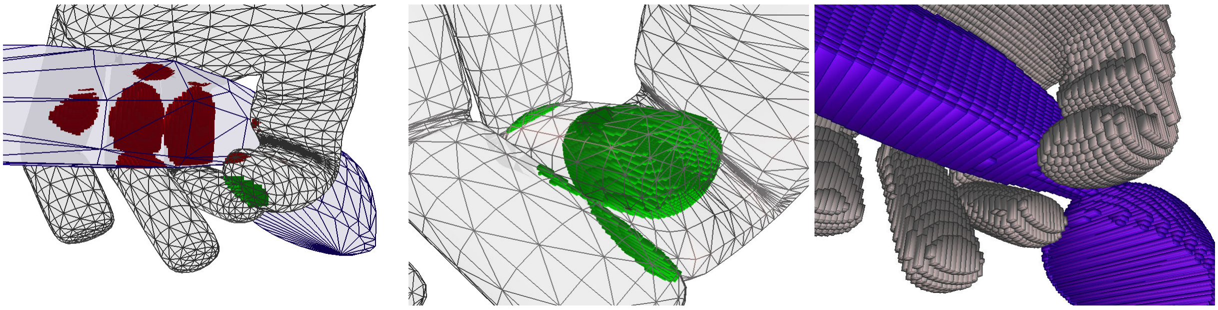


**Figure 19:** *Left: Collisions (red) and self-collisions (green) of the hand are detected. Middle: Self-collisions (green) are detected. Right: LDI representation with a resolution of 64x64. Collisions and self-collisions are detected in 8-11 ms using a standard PC.*

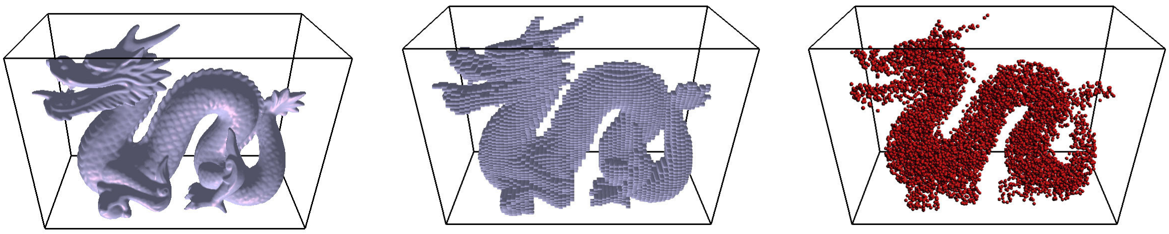


**Figure 20:** *Left: Dragon with 500k faces. Middle: LDI representation with a resolution of 64x64. Right: Particles penetrating the volume of the dragon are detected. In this environment with 500k faces, 100k particles can be tested for penetration in 225 ms using a standard PC.*

to a GPU-based implementation. In contrast, Fig. 20 illustrates an environment with 500k faces and 100k particles. In this case, the GPU-based implementation for detecting penetrating particles outperforms the CPU-based implementation. Thorough comparisons can be found in [HTG04].

### 6.1. Conclusion

In contrast to other collision detection methods, image-space techniques do not require time-consuming pre-processing. This makes them especially appropriate for dynamically deforming objects. They can be used to detect collisions and self-collisions. Image-space techniques usually work with triangulated surfaces. However, they could also be used for other object primitives as long as these primitives can be rendered. Topology changes of objects do not cause any problems.

Since image-space techniques work with discretized representations of objects, they do not provide exact collision information. The accuracy of the collision detection depends on the discretization error. Thus, accuracy and performance can be balanced in a certain range by changing the resolution of the rendering process.

Image-space techniques can be accelerated with graphics hardware. However, due to buffer read-back delays and the limited flexibility of programmable graphics hardware, it is not always guaranteed that implementations on graphics hardware are faster than software solutions in all cases (see [HTG04]). As a rule of thumb, graphics hardware should only be used for geometrically complex environments.

While image-space techniques efficiently detect collisions, they are limited in providing information that can be used for collision response in physically-based simulation environments. In many approaches, further post-processing of the provided result is required to compute or to approximate information such as the penetration depth of colliding objects.

## 7. Applications

This section summarizes various applications of deformable collision detection algorithms in cloth modeling and surgery simulation.

### 7.1. Cloth Simulation

For the simulation of cloth, Mezger et al. [MKE03] used a BVH with a bottom-up update of the hierarchy. This was combined with an approach called "lazy hierarchy update", that omits the update process in the areas of the hierarchy without significant motion (see Fig. 22). To accelerate self-collision detection, a heuristic derived from Provot's normal cones is employed [Pro97].

Impressive results for virtual clothing have been presented over the last decade by the MiraLab. Many algorithms for versatile cloth simulations have been developed. (see Fig. 23 and 24).

As cloth simulation is a very popular field in computer graphics, in recent years also major work has been published on this topic not focussed on collision detection, but on the physical model [CK02, BMF03], the integration schemes [BW98, HE01] or collision response [BWK03, VT00].

Additionally work on real time simulation of cloth has been published, that employs collision detection methods presented in this report or tries to avoid complex collision detection [CMT02].

### 7.2. Virtual Liver Surgery

Lombardo et al. [LCN99] developed a method to address the detection of collisions between a rigid surgical tool and a deformable organ model in a surgical simulation environment. It exploits GPU-based computation by using the OpenGL

**Figure 22:** *Efficient collision and self-collision detection and handling for cloth objects.*

**Figure 23:** *Catwalk scene showing a big variety of clothing.*

clipping process for collision detection. A surgical tool such as a grasper or cautery tool is simple enough to be modeled as a orthographic or perspective viewing volume with clipping planes. Thus, by rendering in feedback mode, they identify the triangles of the organ that are "visible" to this volume as the colliding ones. This simply provides a static collision test when the tool is stationary. Hence, a dynamic collision detection taking into account the volume covered by the tool between consecutive time steps was proposed. Though, this method is extremely fast, it is very specific to be applicable only to simple object shapes such as cylinders. Fig. 25 illustrates this method. Here, collision detection was implemented in conjunction with a physically-based model

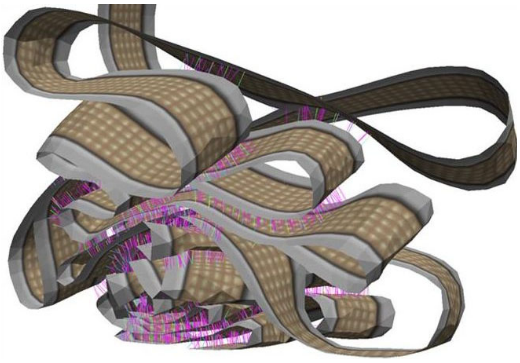


**Figure 24:** *Well-known falling ribbon showing the power of collision response algorithms.*

of a liver. Collision response forces are considered in the dynamic simulation.

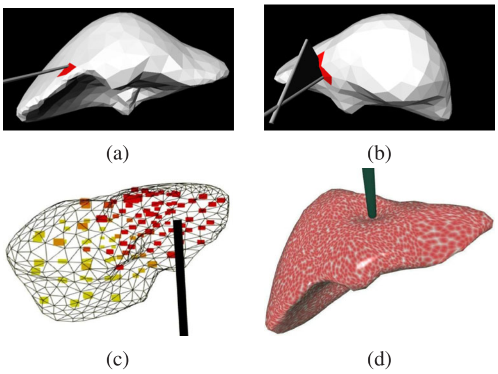



**Figure 25:** *Illustration of a fast, OpenGL clipping-based collision detection method in virtual liver surgery using a standard PC. (a) Collision detected (the dark patch) at a given time step when the tool is static. (b) Dynamic collision detection by sweeping the viewing volume over subsequent time steps as the tool probes the organ. (c) Dynamic simulation with input from the collision response driving a physically-based model. (d) Final textured image.*

### 7.3. Collision Detection Between Virtual Organs

Debunne and Guy [GD02] applied their multiresolution technique (section 3.2.1) to the collision detection between two volumetric elastic organs (a liver and a prostate) manipulated by a rigid tool in a surgical simulation environment (see Fig. 26). Experiments show favorable performances when compared with OBBs for rigid bodies and with AABBs for deformable bodies.

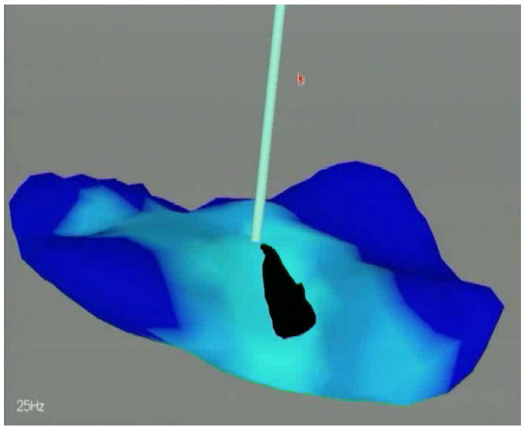



**Figure 26:** *Collision detection during virtual surgery between liver, prostate (shown as a dark-colored organ) and a rigid tool.*

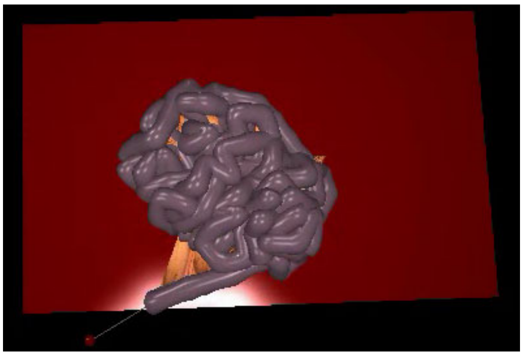
### 7.4. Virtual Intestinal Surgery

Raghupathi et al. [RCFC03] adapted the stochastic technique to detect collisions and self-collisions occurring in the intestinal region when the surgeon manipulates the small intestine before the removal of colon cancer. The intestine being a flexible organ can easily self-collide when moved or deformed using a surgical tool. In addition, it also collides with a thin membrane called the mesentery that connects the colon with the main vessels. Here, the intestine is modeled as a set of skeletal segments with a radius and the mesentery as a thin triangulated mesh. The stochastic collision detection approach (section 3.2.2) was applied between the segments of intestine-intestine and intestine-mesentery while neglecting mesentery-mesentery collisions. A response is applied when a pair actually collides. The skeletal model used for animation and collision processing was fed into a fast adaptive sampling and skinning algorithm for real-time rendering (Fig. 27). For an intestine-mesentery model with 300 animated segments with the possibility of a few hundred collisions, the system achieved real-time performances on a standard PC.

## 8. Conclusion

In this paper, a variety of deformable collision detection approaches has been presented. The discussed approaches are based on bounding volume hierarchies, distance fields, or spatial partitioning. Further, image-space techniques and stochastic methods have been described. All presented methods address problems which especially occur in deformable simulation environments. While all approaches provide promising solutions to certain aspects in deformable collision detection, there exist no general or optimal approach.

Approaches based on bounding volume hierarchies have



(a)

(b)

**Figure 27:** *Real-time treatment of collisions between intestine and mesentery (shown as a light-colored, thin membrane beneath the intestine) and self-collisions in intestine during virtual intestinal surgery in a standard PC. The organs are manipulated using a virtual probe (shown as a tiny red sphere) as in a real surgery.*

shown to be very efficient. In these methods, the basic bounding volume and the strategy for generating and updating the hierarchy have to be chosen very carefully in order to handle frequent update requests in simulation environments with deformable objects. Stochastic methods are a promising approach to time-critical applications, since they allow for balancing performance and accuracy.

Distance fields are especially appropriate for collision detection between rigid and deformable objects. In this case, pre-computed distance fields do not only detect collisions, but also provide the penetration depth which is essential for a physically-correct collision response. Collision detection based on spatial subdivision has also shown to be very efficient in deformable simulation environments. In these approaches, research is mainly focussed on efficient data structures.

Further interesting approaches to deformable collision detection are based on image-space techniques. These algorithms are commonly accelerated with graphics hardware. This makes them especially appropriate for environments with geometrically complex objects and promising results have been presented.

## References


[AdG*02] Agarwal P., de Berg M., Gudmundsson J., Hammar M., Haverkort H.: Box-trees and r-trees with near-optimal query time. *Discrete and Computational Geometry 28*, 3 (2002), 291–312. 3

[BFA02] Bridson R., Fedkiw R., Anderson J.: Robust treatment of collisions, contact and friction for cloth animation. In *Proceedings of ACM SIGGRAPH* (2002), pp. 594–603. 6

[BHW96] Barzel R., Hughes J., Wood D. N.: Plausible motion simulation for computer graphics animation. In *Proceedings of the Eurographics Workshop Computer Animation and Simulation* (1996), Boulic R., Hégron G., (Eds.), Springer, pp. 183–197. 6

[BMF03] Bridson R., Marino S., Fedkiw R.: Simulation of clothing with folds and wrinkles. In *Proc. ACM/Eurographics Symposium on Computer Animation* (2003), pp. 28–36. 9, 10, 15

[BMWM01] Breen D. E., Mauch S., Whitaker R. T., Mao J.: 3d metamorphosis between different types of geometric models. *Eurographics 2001 Proceedings 20(3)* (2001), 36–48. 8, 9, 10

[BPK*02] Bremer P.-T., Porumbescu S., Kuester F., Hamann B., Joy K. I., Ma K.-L.: Virtual clay modeling using adaptive distance fields. In *Proceedings of the 2002 International Conference on Imaging Science, Systems, and Technology (CISST 2002)* (2002). 8

[BW98] Baraff D., Witkin A.: Large Steps in Cloth Simulation. *Computer Graphics 32*, Annual Conference Series (1998), 43–54. 15

[BW02] Baciu G., Wong W. S.-K.: Hardware-assisted self-collision for deformable surfaces. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology (VRST)* (2002), ACM Press, pp. 129–136. 13

[BWK03] Baraff D., Witkin A., Kass M.: Untangling cloth. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003) 22*, 3 (2003), 862–870. 10, 15

[BWS99] Baciu G., Wong W. S.-K., Sun H.:

RECODE: an image–based collision detection algorithm. *The Journal of Visualization and Computer Animation 10* (1999), 181–192. 13

[Can86] CANNY J.: Collision detection for moving polyhedra. *IEEE Trans. Pattern Anal. Mach. Intell. PAMI-8*, 2 (1986), 200–209. 6

[CK02] CHOI K., KO H.: Stable but responsive cloth. In *Proceedings of ACM SIGGRAPH* (2002), pp. 604–611. 15

[CMT02] CORDIER F., MAGNENAT-THALMANN N.: Real-time animation of dressed virtual humans. In *Eurographics Conference Proceedings* (2002), pp. ?–? 15

[COSL98] COHEN-OR D., SOLOMOVIC A., LEVIN D.: Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics 17*, 2 (1998), 116–141. 8

[DDCB01] DEBUNNE G., DESBRUN M., CANI M.-P., BARR A.: Dynamic real-time deformations using space and time adaptive sampling. In *Proc. SIGGRAPH '01* (2001), ACM Press. 8

[EL01] EHMANN S. A., LIN M. C.: Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *Computer Graphics Forum* (2001), vol. 20, pp. 500–510. ISSN 1067-7055. 3

[ES99] ECKSTEIN J., SCHÖMER E.: Dynamic collision detection in virtual reality applications. In *Proc. The 7-th Int'l Conf. in Central Europe on Comp. Graphics, Vis. and Interactive Digital Media '99 (WSCG'99)* (Plzen, Czech Republic, Feb. 1999), University of West Bohemia, pp. 71–78. 6

[FL01] FISHER S., LIN M.: Deformed distance fields for simulation of non-penetrating flexible bodies. In *Proc. of Eurographics Workshop on Computer Animation and Simulation* (2001). 10

[FPRJ00] FRISKEN S. F., PERRY R. N., ROCKWOOD A. P., JONES T. R.: Adaptively sampled distance fields: A general representation of shape for computer graphics. *SIGGRAPH 2000, Computer Graphics Proceedings* (2000), 249–254. 8, 9

[FSG03] FUHRMANN A., SOBOTKA G., GROSS C.: Distance fields for rapid collision detection in physically based modeling. In *Proceedings of GraphiCon 2003* (Sept. 2003), pp. 58–65. 10

[FW99] FAHN C.-S., WANG J.-L.: Efficient time-interrupted and time-continuous collision detection among polyhedral objects in arbitrary motion. *Journal of Information Science and Engineering 15* (1999), 769–799. 6

[GD02] GUY S., DEBUNNE G.: Layered shells for fast collision detection, 2002. Technical Report, GRAVIR/IMAG, INRIA Grenoble, France. 8, 16

[GDO00] GANOVELLI F., DINGLIANA J., O'SULLIVAN C.: Buckettree: Improving collision detection between deformable objects. In *Proc. of Spring Conference on Computer Graphics SCCG '00* (2000). 11

[GK03] GRABNER G., KECSKEMÉTHY A.: Reliable multibody collison detection using runge-kutta integration polynomials. In *Proceedings of IDMEC/IST* (Lisbon, Portugal, July1–4 2003). 6

[GLM96] GOTTSCHALK S., LIN M., MANOCHA D.: OBB-Tree: A hierarchical structure for rapid interference detection. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), Rushmeier H., (Ed.), ACM SIGGRAPH, Addison Wesley, pp. 171–180. 3

[GRLM03] GOVINDARAJU N., REDON S., LIN M., MANOCHA D.: Cullide: Interactive collision detection between complex models in large environments using graphics hardware. In *Proc. of ACM Graphics Hardware* (2003). 13

[GS87] GOLDSMITH J., SALMON J.: Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications 7*, 5 (May 1987), 14–20. 3

[HE01] HAUTH M., ETZMUSS O.: A high performance solver for the animation of deformable objects using advanced numerical methods. In *Proc. Eurographics 2001* (2001), Chalmers A., Rhyne T.-M., (Eds.), vol. 20(3) of *Computer Graphics Forum*, pp. 319–328. 15

[HKL*99] HOFF III K. E., KEYSER J., LIN M., MANOCHA D., CULVER T.: Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of ACM SIGGRAPH 1999* (1999), pp. 277–286. 8, 9, 11

[HTG03] HEIDELBERGER B., TESCHNER M., GROSS M.: Real-time volumetric intersections of deforming objects. In *Proc. of Vision, Modeling, Visualization VMV'03* (2003), pp. 461–468. 13

[HTG04] HEIDELBERGER B., TESCHNER M., GROSS M.: Detection of collisions and self-collisions using image-space techniques. In *Proc. of WSCG'04* (2004), p. to appear. 13, 14, 15

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

[IZLM01] III K. E. H., ZAFERAKIS A., LIN M. C., MANOCHA D.: Fast and simple 2d geometric proximity queries using graphics hardware. In *Symposium on Interactive 3D Graphics* (2001), pp. 145–148. 11, 13

[JS01] JONES M. W., SATHERLEY R.: Using distance fields for object representation and rendering. In *Proc. 19th Ann. Conf. of Eurographics (UK Chapter)* (London, 2001), pp. 37–44. 9

[KGL*98] KRISHNAN S., GOPI M., LIN M., MANOCHA D., PATTEKAR A.: Rapid and accurate contact determination between spline models using ShellTrees. *Computer Graphics Forum 17*, 3 (Sept. 1998). 3

[KHM*98] KLOSOWSKI J. T., HELD M., MITCHELL J. S. B., SOWRIZAL H., ZIKAN K.: Efficient collision detection using bounding volume hierarchies of *k*-DOPs. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (Jan. 1998), 21–36. 3

[KmLD03] KIM Y. J., HOFF III K. E., LIN M. C., D.MANOCHA: Closest point query among the union of convex polytopes using rasterization hardware. *Journal of Graphics Tools* (2003). 13

[KOLM02] KIM Y., OTADUY M., LIN M., MANOCHA D.: Fast penetration depth computation for physically-based animation. In *Proc. of SIGGRAPH Symposium on Computer Animation '02* (2002), pp. 23–31. 13

[KP03] KNOTT D., PAI D.: Cinder: Collision and interference detection in real–time using graphics hardware. In *Proc. of Graphics Interface '03* (2003). 13

[KR03] KIM B., ROSSIGNAC J.: Collision prediction for polyhedra under screw motions. In *ACM Symposium on Solid Modeling and Applications* (Seattle, Washington, USA, June16–20 2003), pp. 4–10. 6

[KZ03] KLEIN J., ZACHMANN G.: Adb-trees: Controlling the error of time-critical collision detection. In *8th International Fall Workshop Vision, Modeling, and Visualization (VMV)* (University München, Germany, Nov.19–21 2003). 7

[LAM01] LARSSON T., AKENINE-MÖLLER T.: Collision detection for continuously deforming bodies. In *Eurographics* (2001), pp. 325–333. short presentation. 3, 4, 5

[LAM03] LARSSON T., AKENINE-MÖLLER T.: Efficient collision detection for models deformed by morphing. *The Visual Computer 19*, 2 (May 2003), 164–174. 5

[LC92] LIN M. C., CANNY J. F.: Efficient Collision Detection for Animation. In *Proc. 3rd Eurographics Workshop on Animation and Simulation* (1992). 7

[LCN99] LOMBARDO J., CANI M.-P., NEYRET F.: Real-time Collision Detection for Virtual Surgery. In *Proc. Comp. Anim. '99* (1999), IEEE CS Press, pp. 82–91. 13, 15

[Mel00] MELAX S.: Dynamic plane shifting bsp traversal. In *Proc. of Graphics Interface '00* (2000), pp. 213–220. 11

[Mir97] MIRTICH B.: *Efficient algorithms for two-phase collision detection*. Tech. Rep. TR-97-23, Mitsubishi Electric Research Laboratory, 1997. 11

[MKE03] MEZGER J., KIMMERLE S., ETZMUSS O.: Hierarchical Techniques in Collision Detection for Cloth Animation. *Journal of WSCG 11*, 2 (2003), 322–329. 4, 5, 6, 15

[MOK95] MYSZKOWSKI K., OKUNEV O., KUNII T.: Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer 11*, 9 (1995), 497–512. 13

[PG95] PALMER I. J., GRIMSDALE R. L.: Collision detection for animation using sphere-trees. *Computer Graphics Forum 14*, 2 (June 1995), 105–116. 3

[Pro97] PROVOT X.: Collision and Self-Collision Handling in Cloth Model Dedicated to Design Garments. In *Graphics Interface '97* (May 1997), Canadian Information Processing Society, Canadian Human-Computer Communications Society, pp. 177–189. 4, 6, 15

[PT92] PAYNE B. A., TOGA A. W.: Distance field manipulation of surface models. *IEEE Computer Graphics and Applications 12(1)* (January 1992), 65–71. 9

[RCFC03] RAGHUPATHI L., CANTIN V., FAURE F., CANI M.-P.: Real-time Simulation of Self-collisions for Virtual Intestinal Surgery. In *Surg. Sim. & Soft Tis. Model.* (2003), Ayache N., Delingette H., (Eds.), LNCS no. 2673, Springer, Heidelberg, pp. 15–26. 8, 16

[RKC02] REDON S., KHEDDARY A., COQUILLART S.: Fast continuous collision detection between rigid bodies. *Computer Graphics Forum (Eurographics) 21*, 3 (Sept. 2002), 279–288. 6

[RL85] ROUSSOPOULOS N., LEIFKER D.: Direct spatial search on pictorial databases using packed R-trees. In *Proceedings of ACM-SIGMOD 1985 International Conference on Management of Data* (Austin, Texas, May28–31 1985), pp. 17–31. 3

[Set96] SETHIAN J.: A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Science 93(4)* (1996), 1591–1595. 9, 10

[SF91] SHINYA M., FORGUE M.: Interference detection through rasterization. *Journal of Visualization and Computer Animation 2* (1991), 132–134. 13

[SGwHS98] SHADE J., GORTLER S., WEI HE L., SZELISKI R.: Layered depth images. In *Proceedings of SIGGRAPH '98* (1998), pp. 231–242. 13

[SPG03] SIGG C., PEIKERT R., GROSS M.: Signed distance transform using graphics hardware. In *Proceedings of IEEE Visualization '03* (October 2003), IEEE Computer Society Press. 10

[SSW95] SCHÖMER E., SELLEN J., WELSCH M.: Exact geometric collision detection. In *Proc. 7th Canad. Conf. Comput. Geom.* (1995), pp. 211–216. 6

[THM*03] TESCHNER M., HEIDELBERGER B., MUELLER M., POMERANETS D., GROSS M.: Optimized spatial hashing for collision detection of deformable objects. In *Proceedings of Vision, Modeling, Visualization VMV'03* (2003), pp. 47–54. 11, 12

[Tur90] TURK G.: *Interactive collision detection for molecular graphics*. Tech. Rep. TR90-014, University of North Carolina at Chapel Hill, 1990. 11

[US97] UNO S., SLATER M.: The sensitivity of presence to collision response. In *Proc. of IEEE Virtual Reality Annual International Symposium (VRAIS)* (Albuquerque, New Mexico, Mar.01–05 1997), p. 95. 6

[vdB97] VAN DEN BERGEN G.: Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools 2*, 4 (1997), 1–14. 3, 4

[VMT94] VOLINO P., MAGNENAT-THALMANN N.: Efficient Self-Collision Detection on Smoothly Discretized Surface Animations using Geometrical Shape Regularity. *Computer Graphics Forum 13*, 3 (1994), 155–166. 4, 6

[VMT95] VOLINO P., MAGNENAT-THALMANN N.: Collision and Self-Collision Detection: Efficient and Robust Solutions for Higly Deformable Surfaces. In *Comp. Animation and Simulation* (1995), Springer Verlag, pp. 55–65. 4

[VSC01] VASSILEV T., SPANLANG B., CHRYSANTHOU Y.: Fast Cloth Animation on Walking Avatars. In *Computer Graphics Forum (Proc. of Eurographics)* (2001). 11, 13

[VT00] VOLINO P., THALMANN N. M.: Implementing Fast Cloth Simulation with Collision Response. *Computer Graphics Interface* (2000). 15

[WK03] WU J., KOBBELT L.: Piecewise linear approximation of signed distance fields. In *Vision, Modeling and Visualization 2003 Proceedings* (2003), pp. 513–520. 9

[WKE99] WESTERMANN R., KOBBELT L., ERTL T.: Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces. *The Visual Computer 15*, 2 (1999), 100–111. 9

[Zac98] ZACHMANN G.: Rapid collision detection by dynamically aligned DOP-trees. In *Proc. of IEEE Virtual Reality Annual International Symposium; VRAIS '98* (Atlanta, Georgia, Mar. 1998), pp. 90–97. 3

[Zac02] ZACHMANN G.: Minimal hierarchical collision detection. In *Proc. ACM Symposium on Virtual Reality Software and Technology (VRST)* (Hong Kong, China, Nov.11–13 2002), pp. 121–128. 3, 4

[ZL03] ZACHMANN G., LANGETEPE E.: Geometric data structures for computer graphics. In *Proc. of ACM SIGGRAPH*. ACM Transactions of Graphics, 27–31July 2003. 3

[ZWF*03] ZHAO Y., WEI X., FAN Z., KAUFMAN A., QIN H.: Voxels on fire. In *Proceedings of IEEE Visualization* (2003). 8

[ZY00] ZHANG D., YUEN M.: Collision detection for clothed human animation. In *Proceedings of Pacific Graphics '00* (2000), pp. 328–337. 11