Distance Fields for Rapid Collision Detection in Physically Based Modeling

Arnulph Fuhrmann* IGD
Gerrit Sobottka† UNI-BONN
Clemens Groß‡ IGD

IGD: Fraunhofer Institute for Computer Graphics
Rundeturmstr. 6, 64283 Darmstadt, Germany
UNI-BONN: Institute of Computer Science II, University of Bonn
Römerstr. 164, 53117 Bonn, Germany

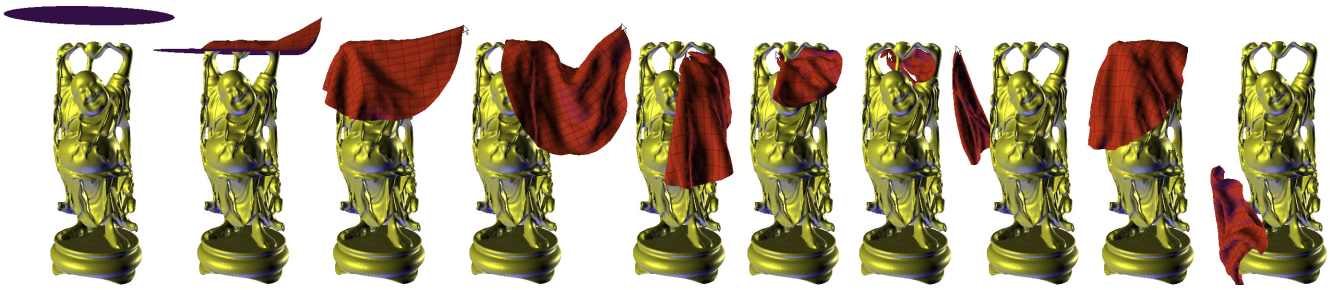


Figure 1: Several snapshots from an interactive session: A tablecloth is dragged around by the user with the mouse.

Abstract

In this paper we address the problem of rapid distance computation between rigid objects and highly deformable objects, which is important in the context of physically based modeling of e.g hair or clothing. Our method is in particular useful when modeling deformable objects with particle systems—the most common approach to simulate such objects.

We combine some already known techniques about distance fields into an algorithm for rapid collision detection. Only the rigid objects of an environment are represented by distance fields. In the context of proximity queries, which are essential for proper collision detection, this representation has two main advantages: First, any given boundary representation can be approximated quite easily, no high-degree polynomials or complicated approximation algorithms are needed. Second, the evaluation of distances and normals needed for collision response is extremely fast and independent of the complexity of the object.

In the course of the paper we propose a simple, but fast algorithm for partial distance field computation. The sources are triangular meshes. Then, we present our approach for collision detection in detail. Examples from an interactive cloth animation system show the advantages of our approach in practice. We conclude that our method allows real-time animations of complex deformable objects in non-trivial environments on standard PC hardware.

Keywords: distance field, collision detection, physically based modeling, real-time animation

*Arnulph.Fuhrmann@igd.fhg.de
†sobotka@cs.uni-bonn.de
‡Clemens.Gross@igd.fhg.de

1 Introduction

Rapid collision detection between different objects is an important part of interactive computer graphics applications. Intense research over the last decade has provided many algorithms for particular applications. Rigid objects which are moving in a static environment are an example for the most common scenario. In this case hierarchical collision detection algorithms provide exact and fast solutions. A bounding volume hierarchy is built for each object and during collision detection the hierarchies are tested recursively against each other for overlap. Since these overlap tests are highly efficient and the number of tests with geometric primitives is usually small, the overall performance allows interactive applications.

But in the case that one of the objects is not rigid anymore things become worse. Just imagine a piece of cloth dragged around in virtual environment. Now it is possible that the whole surface of the deformable object is in contact with another object. In this case the collision detection performance of a bounding volume hierarchy drops considerably. Although the hierarchy is still able to report nearby triangles very efficiently, the number of primitive overlap tests increases to a maximum, since all geometric primitives of the deformable object have to be tested with primitives of the other object. In many applications this means that you have to check lots of pairs of triangles against each other. Clearly, this approach is prohibitive for interactive applications. In particular, if the application has to resolve not only collisions but has to solve differential equations for dynamic motion.

In order to overcome this performance problem we propose an algorithm for rapid collision detection between deformable and rigid objects which uses distance fields as a basis for proximity queries. Our algorithm is not only very efficient but also more exact than previous approaches using voxelisation. Furthermore, only minor pre-computations for auxiliary data structures are needed. For moderately sized meshes all data structures used can be built dur-

ing startup of the application due to our very efficient distance field generation algorithm.

We first describe an efficient method for calculating the distance field. It can be applied to huge oriented triangular meshes which need not necessarily be closed. In the second part of the paper we show how to utilize a distance field for collision detection between rigid objects and deformable objects. We propose a solution to handle concavities and how to deal with multiple distance fields at once. Finally, we present some of our results and the application of our algorithm to the area of cloth modeling for interactive applications.

2 Related Work

In the area of collision detection several methods were proposed in the last years. For convex polyhedra algorithms have been developed that work in expected linear time, see e. g. [10, 17]. In order to overcome the restriction of convexity various hierarchical bounding volume approaches have been developed: axis aligned bounding box trees [23], oriented bounding boxes used in OBBtrees [8], sphere trees [11, 18], discretely oriented polytopes (k-DOPs) [13] and dynamically aligned DOP-trees [26]. Recently, a novel bounding volume hierarchy has been proposed, which is as fast as OBB-Trees or DOP-trees but which uses much less memory [27].

In the context of physically based simulation of cloth more specialized hierarchical collision detection algorithms have been developed [19, 25, 15]. Recently, a computational expensive but very accurate algorithm for robust treatment of collisions was presented [4].

The most suitable collision detection algorithms for interactive applications are image space or voxel-based techniques [14], where each voxel is marked as inside, outside or surface voxel. Since no hierarchy has to be traversed and no primitives have to be tested against each other, this algorithm is very fast. Clearly, this kind of algorithm trades off accuracy for speed. In order to be more accurate Zhang and Matthew [28] proposed to store additional information in the voxel-grid. Thus, primitive overlap tests are still performed. In order to overcome these primitive tests Meyer et al. store only the normal and the closest surface point per voxel [16]. This information is used for collision response computations. This method, although more precise, still introduces inaccuracies since the surface is linearized inside each voxel and not continuous between voxels. Finally, Vassilev et al. [24] described a method which is most closely related to our approach. The authors are using the rendering hardware for constructing two depth and velocity maps of the object—one map for the backside of the object and one map for its front. These maps are used for distance calculations and collision response. Their algorithm is restricted to convex shapes or—in the case of virtual humans—appropriate mapping directions. Furthermore, the accuracy is quite low at the silhouette of the object.

The concept of distance fields is popular in the field of computer graphics. It has been used for metamorphosis of objects [3], acceleration of volume data rendering, for motion planning and for penetration depth estimation purposes in the range of physically based modeling [5]. Distances to the static objects of a scene are mapped to a Cartesian grid as part of the preprocessing. This allows us to rapidly query the distance of a point to an object without testing all geometric primitives. Distance field computation can be computational expensive and, therefore, fast computation methods still remain a topic.

There are standard methods for building up a distance field. Jones and Satherley [12] used distance transforms to propagate an ex ante calculated distance shell throughout the whole volume. Sethian [22] introduced the so-called level set method to the field of computer graphics, which describes an evolving front by a partial differential equation. Once a distance shell has been computed it can be used as initial value for a level set based fast marching algorithm.

3 Distance field computation

A distance field of a surface S is a scalar function

$$D : \mathbb{R}^3 \longrightarrow \mathbb{R},$$

$$D(\mathbf{p}) = \min_{q \in S} \{ \mid \mathbf{p} - \mathbf{q} \mid \}, \forall \mathbf{p} \in \mathbb{R}^3.$$

If the surface is closed, one can define a sign function with

$$\operatorname{sgn}(\mathbf{p}) = \begin{cases} -1 & \text{if } \mathbf{p} \text{ inside} \\ 1 & \text{if } \mathbf{p} \text{ outside} \end{cases}$$

which yields together with the distance function a signed distance field.

Since the concept of inside and outside only makes sense in the case of a closed surface, we use a more simple definition of these perceptions. We assume that S is defined by a set of oriented triangles. Then, a face normal $\mathbf{n}$ is known for every point $\mathbf{q} \in S$. Now let $\mathbf{q} \in S$ be a nearest point of S to $\mathbf{p}$, then we define the sign function as

$$\operatorname{sgn}'(\mathbf{p}) = \begin{cases} -1 & \text{if } \langle \mathbf{p} - \mathbf{q} \mid \mathbf{n} \rangle < 0 \\ 1 & \text{if } \langle \mathbf{p} - \mathbf{q} \mid \mathbf{n} \rangle \geq 0 \end{cases} .$$

Within a small envelope of S for any $\mathbf{p}$ there is a $\mathbf{q}$ such that $\operatorname{sgn}(\mathbf{p}) = \operatorname{sgn}'(\mathbf{p})$. Thus, it is not necessary for the mesh to be closed but it has to be oriented.

Since we are dealing with triangular meshes we have three types of normals: *face-*, *edge-* and *vertex* normals. Due to the fact that edge normals are expensive to compute—information about adjacent triangles are required—only face normals are used for distance field computation.

The algorithm can be summarized as follows (see Algorithm 1; $\vec{v}_i(t)$ denotes a vertex and $\vec{e}_i(t)$ denotes an edge of triangle t): for every triangle of the mesh a prism is calculated by moving its vertices along the face normal by an amount of ε in negative and positive direction (ε is δ times the cell diagonal; using δ we can vary the thickness of the distance envelope). Subsequently, the axis-aligned integer bounding box enclosing the prism is determined. For all grid points that lie in the bounding box the distance to the triangle is computed. Finding the minimum distance between a given point and a triangle can be done efficiently using *Voronoi regions* of the features of the triangle [1]. The current distance is set to the new value if the calculated absolute distance is less than the current. The sign of the distance value results from the sign of the inner product of the face normal and the direction vector, i.e., if the considered point lies below the triangle plane, the sign is negative otherwise positive. Thus, the triangle plane divides the bounding box volume into a positive and a negative part.

The asymptotic time complexity of the algorithm is $\mathcal{O}(nm)$ where n is the number of triangles and m the number of grid points.

At first glance one might wonder if this algorithm produces a valid distance field despite potential sign errors. Indeed, sign errors can become a problem if the size of the triangles becomes too large.

Due to the fact that the distance field is built up successively, changes of sign occur in the overlapping areas of bounding boxes if the angle between the face normals is unequal to 0. In the convex case a change of sign from minus to plus is legal, because it makes an inside point an outside point. In contrast, a change of sign from plus to minus is illegal, because it would make an existing inner

[1] To compute the *Voronoi regions* only a single triangle is considered.

Input: triangular mesh $\mathscr{T}$ with face normals $\vec{n}$, equidist. cartesian grid $\mathscr{G}$
Output: distance field $\mathscr{F}$

```
F ← +∞
ε ← ‖ (dx, dx, dx) ‖ · δ
for all t ∈ T do
    compute Prism P ← PRISM(t, n(t), ε)
    compute integer bounding box G_p ← IAABB(P)
    for all p ∈ G_p ∩ G do
        d ← ⟨p − v_1(t) | n(t)⟩
        if p ∉ VoronoiReg(t) then
            if p ∈ {VoronoiReg(e_{i∈{1,2,3}}(t))} then
                d ← dist(p, e_i(t)) · sgn(d)
            else
                d ← min{‖ p − v_{1,2,3}(t) ‖} · sgn(d)
            end if
        end if
        F(p) ← min{abs(d), abs(F(p))}
    end for
end for
```

Algorithm 1: DISTANCEFIELD($\mathscr{T}$, $\mathscr{G}$)

point an outside point. Figure 2 depicts these circumstances (projection of two adjacent triangles in the direction of the common edge). Suppose that the triangle t_i has been processed and another

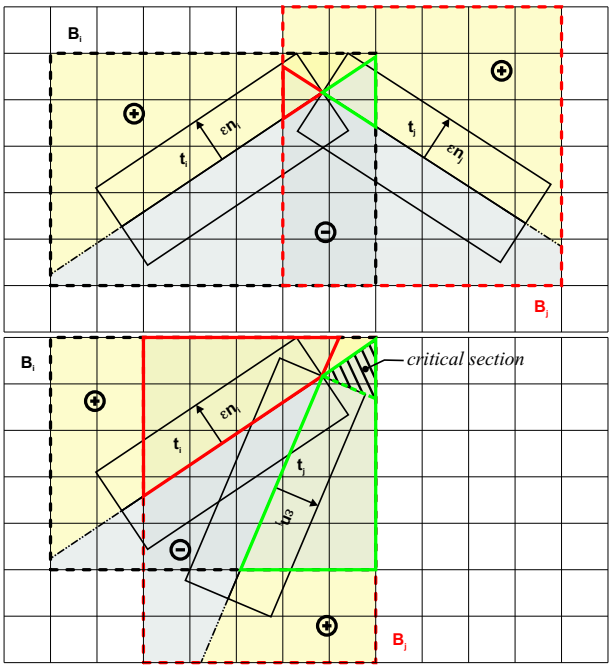


Figure 2: Overlapping bounding boxes of two in succession built in triangles. The green borders mark the areas where a change of sign minus to plus occurs.

triangle t_j which is adjacent to t_i is currently being computed. Let the bounding box of the prism of a triangle be B. We denote the positive part of the volume by $B^{\oplus}$ and the negative by $B^{\ominus}$. A change of sign from plus to minus occurs inside the area $A_{PM} = B_i^{\oplus} \cap B_j^{\ominus}$ and from minus to plus inside the area $A_{MP} = B_i^{\ominus} \cap B_j^{\oplus}$. Since every point of A_{PM} lies above the plane of t_i, its distance to t_j cannot become smaller than its distance to t_i. From this it follows that an existing distance value inside the 'critical section' (red bordered areas in Figure 2) will not be replaced by another and, therefore, the sign still remains valid.

Within the area A_{MP} a change of sign from minus to plus is essential, because all points inside A_{MP} become outer area of the object. For all points inside A_{MP} which are not lying inside the *Voronoi region* of the common edge the distance to t_j is smaller than to t_i. Therefore, the distance values at those points will be set to new values which implies the claimed change of sign (green bordered areas). The problem is that in case of acute angles (i.e., the angle between the face normals $\angle(n_2, n_1)$ lies between $+\frac{\pi}{2}$ and $+\pi$) all points of A_{MP} that are lying inside the *Voronoi region* of the common edge will not undergo a change of sign, because the distance remains the same. The result is a small negative wedge-shaped region (see marked critical section in Figure 2) outside the object that irradiates from the common edge. This all holds for the convex case where the angle between the two face normals lies in $[0, +\pi]$.

In the non-convex case (i.e., angle between the two normals of adjacent triangles is in $[0, -\pi]$) a change of sign from plus to minus is legal and from minus to plus is illegal, because it makes an inside point an outside point. The situation sketched in Figure 2 remains valid except that one has to turn the face normals to the opposite direction and swap the plus and minus signs. A change of sign has to be prevented inside the area A_{MP} and to be promoted in the area A_{PM}. The same circumstances as mentioned before now lead to a small wedge-shaped and positive-signed region inside the object.

Due to numerical inaccuracies we have to use a tolerance value. Otherwise, values and their signs could be replaced in cases where they must not. This effect is conspicuous at the borders of the edge *Voronoi regions*. It leads to spongiform wedges in the area of edges.

In consequence of sign errors the algorithm has problems to compute the distance field correctly in case of simple geometric primitives with acute angles. For convex polytopes with obtuse angles no such sign errors occur.

3.1 Results

We have tested our distance field algorithm with several geometries (triangular meshes with up to 1.8 M faces) at variable resolutions. For complex triangular meshes with predominantly small triangles like those shown in Figure 3 the algorithm produces very pleasing results in short time (see Table 1).

For reconstruction we used the *marching cubes* algorithm. Subsequently, we measured the *Hausdorff distance* between the original data T and the reconstructed surface T' (normally referred to as *forward Hausdorff distance*). The *Hausdorff distance* is defined as $d_H(T, T') = max\, d(p, T')$, $p \in T$, with $d(p, T') = min \parallel p - p' \parallel$, $p' \in T'$. It constitutes an accepted measure for the distortion of surfaces [1].

Figure 4 illustrates the distances from a mesh of a human model to its reconstruction. Based on the assumption that the bust measures approximately 45 cm (from head to the bottom of the chest) the maximum distance amounts to 1.4 mm (mean 7×10^{-3} mm). In the problem area of the non-convex ears the maximum distance is less than 0.5 mm.

3.2 Comparison to other methods

The advantages of this algorithm are obvious. First, we can apply it to very large meshes, because we do not construct any memory consuming data structures for surface representation like trees or *Voronoi diagrams*. Furthermore, we require the meshes to be oriented but it is not necessary for the meshes to be closed. This assumption seems to be more realistic than that of a closed mesh, i.e. think of a car body without window panes or human head models without eyeballs etc.

We compute a snatchy but not minimal distance envelope around the surface. For collision detection purposes this is sufficient for

Figure 3: Comparison of original data (left) and its reconstruction out of the distance field data (right) — (a) *Happy Buddha* - original data (1.1 M faces). The distance field computation ($167 \times 167 \times 401$) took 14 secs. on a 1.8 GHz Pentium IV; (b) *Turbine Blade* - (1.8 M faces, $356 \times 278 \times 600$, 29 secs., 2.2 GHz Pentium IV).

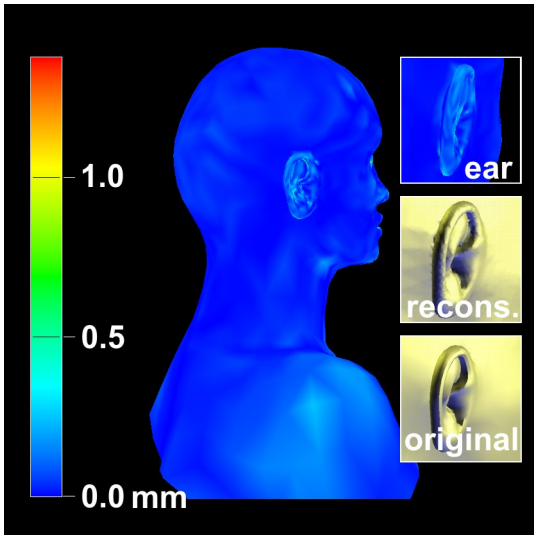


Figure 4: *Hausdorff distance* from the original surface to the reconstruction ($400 \times 257 \times 393$). Blue maps to zero distance. The maximum aberration in the problematic areas of the (non-convex) ears does not exceed 0.5 mm.

two reasons:

- In case of physically based simulation of rigid bodies time steps normally become small during integration of the equations of motion as a consequence of applied collision response forces. The probability for a particle to cross the distance envelope in a single time step is small. If collision response bases upon position adjustment, time steps normally are larger (up to one frame). But we know from experience that the distance envelope is nevertheless sufficient (see pictures below).

- Due to the fact, that particles or rigid bodies in simulation of continuous objects are interconnected by springs or joints (e.g. in cloth simulation) single particles that cross the distance envelope will be pulled out by other particles.

In comparison to our method Breen et al. [3] first built up the whole *Voronoi diagram* for the faces, edges and vertices of the triangular mesh. They used scan conversion to determine which cells of the grid lie in each *Voronoi region*, i.e., the distance to the surface at a set of points within a narrow band and the zero set are computed. The two sets are used as raw data for a fast marching method [21]. Unfortunately Breen et al. do not give any information about computation times.

Jones and Satherley [12, 20] used an *octree*-based approach to accelerate computation; it reduces computational complexity to $\mathcal{O}(n \log n)$. Alternatively one can apply *distance transforms* to precomputed distance shells to speed up the propagation of distance values to the whole grid [12]. Distance transforms are fast but not very accurate.

These methods are two step processes. The first step is the *narrow band* computation, in the second step the narrow band distances are propagated to the remaining grid points.

4 Rapid collision detection

In this chapter we propose a method for rapid collision detection between rigid bodies and deformable objects like cloth or strands of hair. We decided to solve this problem approximately by not testing each triangle of the deformable object, but considering only the vertices. Clearly, we now have to hold those vertices about a predefined ε away from the surface to avoid artifacts, see Figure 5. This distance depends on the size of the triangles. Since we are using a fine triangulation of the deformable object our approach is feasible. Furthermore, the approximations made by our approach allow interactive applications, which would not be possible if a full collision detection was carried out.

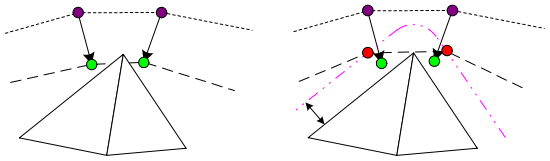


Figure 5: (a) Without offsetting the vertices inter-penetration artifacts occur at sharp corners since only vertices are checked during collision detection. (b) Introducing an ε-offset solves the problem since the maximum distance of two vertices is known in advance.

4.1 The simulation loop

In order to explain our collision detection algorithm we review shortly which steps have to be taken for each simulation iteration. We assume that a particle system is used. In such a system each vertex of the discretized surface or volume represents a particle $\mathbf{p_i}$ with an associated mass m_i. A new simulation step starts with the old positions $\mathbf{x_i^n} \in \mathbb{R}^3$ and velocities $\mathbf{v_i^n} \in \mathbb{R}^3$ of all particles. The movement of each particle is governed by the well-known Newton's equation of motion $\mathbf{f_i} = m_i \cdot \mathbf{a_i} = m_i \cdot \frac{d^2\mathbf{x_i^n}}{dt^2}$ where $\mathbf{f_i} \in \mathbb{R}^3$ denotes the force acting on the particle and $\mathbf{a_i} \in \mathbb{R}^3$ is the acceleration. In order to solve the equation external and internal forces have to be considered and integrated over time to get the new particle positions and

	Triangles	Resolution	Time [secs.]	examined points
Skull	60 339	$252 \times 363 \times 281$	5.177	22 775 185
		$347 \times 500 \times 388$	7.468	41 802 953
Vampire	127 656	$290 \times 289 \times 401$	3.155	13 384 822
		$434 \times 432 \times 600$	4.626	22 769 128
Greta	36 236	$376 \times 85 \times 401$	1.472	6 240 152
		$755 \times 167 \times 804$	3.703	17 736 887
Spaceship	90 800	$157 \times 400 \times 75$	3.825	16 318 608
		$196 \times 500 \times 92$	5.017	21 377 069
		$234 \times 600 \times 110$	6.469	27 288 247
		$273 \times 700 \times 128$	8.112	34 271 815
		$311 \times 800 \times 146$	9.874	41 905 391
Skeleton	87 878	$78 \times 64 \times 400$	1.893	8 583 692
		$153 \times 124 \times 800$	3.157	17 982 883
Bunny	69 451	$200 \times 156 \times 199$	2.203	9 274 604
		$400 \times 311 \times 397$	4.597	19 065 462
Dragon	871 414	$400 \times 182 \times 284$	13.876	79 220 231
		$600 \times 271 \times 425$	17.985	99 091 981
Buddha	1 087 716	$167 \times 167 \times 400$	14.860	86 503 240
		$249 \times 250 \times 600$	18.656	104 837 218
Turbine	1 765 388	$238 \times 186 \times 400$	23.110	132 638 147
		$356 \times 278 \times 600$	29.532	164 197 880

Table 1: Duration of distance field computation for triangular meshes on a 1.8 GHz Pentium IV (500 MB); *Dragon*, *Buddha*, and *Turbine* on a 2.2 GHz Pentium IV (2 GB).

velocities. We refer the interested reader to [7] for more information about physically based cloth modeling. After the time integration we get the new positions $\mathbf{x_i^{n+1}}$and the new velocities $\mathbf{v_i^{n+1}}$. Collisions are resolved simply by moving particles back to the surface of the object. Since this relocation changes the trajectory of the particle, the velocities have to be corrected according to the collision response. Finally, our particle system is in a correct state in which no inter-penetrations are visible and the simulation step ends.

Notice, that in our system possible collisions are not affecting the computation of forces of the particle system. Although techniques for incorporating collision information exist [2], we have chosen not to use them because of the increased time complexity.

4.2 Collision Response

During collision response we need to reconstruct two kinds of values from our already calculated distance field: Distances for arbitrary points between the sampled grid points and corresponding surface normals at these points. To compute the distance values we use trilinear interpolation of the 8 corner points of a grid cell. This interpolation provides us with a smooth reconstructed surface inside each grid cell. Between different cells the surface is only C^0. The normals are computed by normalizing the analytic gradient of the trilinear interpolation [6]. In contrast, storing normal information per grid point would be more memory intensive and the trilinear interpolation of the real normal is computationally even more expensive than evaluating the analytic gradient.

With these reconstruction methods we are able to determine the signed distance between a particle of the deformable object and the boundary surface of the rigid object rapidly. At first, we have to transform all necessary coordinates of the particle into the coordinate system of the collision object. Then, if $\mathbf{x_i^{n+1}}$ is detected to be closer to the object's surface than the given threshold ε, it is set back in the direction of the normal $\mathbf{n}$ at $\mathbf{x_i^{n+1}}$. The normal is computed by normalizing the gradient of the distance field. Let d be the distance of the vertex to the surface, then

$$\mathbf{r_n} = \langle \mathbf{n} \mid \varepsilon - d \rangle \tag{1}$$

computes the normal component of our collision response [2]. For an approximate modeling of friction we compute a tangential component $\mathbf{r_t}$ by scaling the tangential part $\Delta\mathbf{x_t}$of the vertex movement vector

$$\Delta\mathbf{x_n} = \Delta\mathbf{x} - \mathbf{n}\langle\Delta\mathbf{x} \mid \mathbf{n}\rangle \tag{2}$$

$$\mathbf{r_t} = -c_f \Delta\mathbf{x_t} \tag{3}$$

where $\Delta\mathbf{x} = \mathbf{x_i^{n+1}} - \mathbf{x_i^n}$ and c_f is the friction parameter (1.0 for friction cancels all the tangential movement). The final vertex position is computed by

$$\mathbf{x_i^{n+1}} = \mathbf{x_i^{n+1}} + \mathbf{r_n} + \mathbf{r_t}. \tag{4}$$

Although our algorithm does not compute the exact intersection point $\mathbf{p_{is}}$ between the particle and the object's surface, we get satisfactory results in practice. This is due to the fact that errors are quite small. The surface normal at $\mathbf{x_i^{n+1}}$ points only at slightly different direction as it would at $\mathbf{p_{is}}$ and the scaling of the tangential component is just a little bit too much which helps the system to stay more stable.

For an exact solution we could determine the distance which the particle may move until he hits the object. Then we could set $\mathbf{x_i^n} = \mathbf{p_{is}}$ and apply our above algorithm. Unfortunately, this introduces two additional distance evaluations: one at $\mathbf{x_i^n}$ for finding $\mathbf{p_{is}}$ and a second one at $\mathbf{p_{is}}$ itself. Since these evaluations are the most costly part of our algorithm we do not use this method but stick to our much simpler algorithm as described above.

Notice, that our algorithm models a totally inelastic collision. So, the energy of the impact is lost. By scaling $\mathbf{r_n}$ appropriately we could also model an elastic collision, if this effect is of any interest. For the examples shown in this paper we used our inelastic model.

4.3 Trapped particles

The algorithm for collision response described above works quite well for convex objects. But in the case of concavities a particle can get stuck as depicted in Figure 6. The force caused by a connected particle is trying to push the particle away, but the collision response process moves the particle back and eventually into the dead end.

In order to solve this problem we introduced additional edge tests into the collision response mechanism. The center $\mathbf{p_{ce}}$of each edge of the particle system is checked for proximity with the object. If necessary the same procedure as described in section 4.2 is applied. The resulting correction vector $\mathbf{r_n} + \mathbf{r_t}$ is added to each of the two particles forming the edge. In most cases not all edges have to be tested. Only if one of the two particles of a edge is in close proximity to the object a check has to be made.

As a pleasing side effect, we do not only free some of the particles from their concave prison, but we are also able to decrease ε considerably without introducing any artifacts. By testing edges in their center we virtually increase the sampling density of our colliding surface which in turn allows to be closer to the object surface.

Unfortunately, our method has the drawback that in the worst case the number of collision queries increases by a factor of three. But since the problem of stuck particles occurred quite often in our test scenarios and the overall precision increases we strongly suggest to implement this edge test.

[2] Where $\langle . \mid . \rangle$ denotes the dot product.

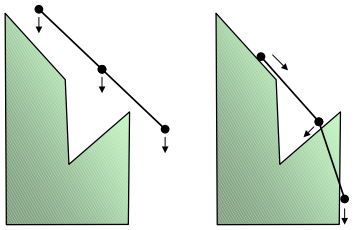


Figure 6: The collision response can fail if only particles are considered: The particle in the middle is dragged into the concave region as a result of collision response.

4.4 Enhancing friction

The model for friction described above is very simple and does not reflect real friction behavior. A better way would be to use Coulomb's model for friction. This model distinguishes between static and kinetic friction and, more importantly, takes into account the force pressing two objects together, i.e. the force due to friction is proportional to the force in normal direction of the surface ($F_f = c_f F_N$). The latter property is clearly missing in the formulas we described above. In practice, we experienced particles sliding slowly over vertical surfaces, an effect certainly not intended.

Therefore we derived a more elaborate model of friction, which does not affect performance too much and uses only the change of positions of the particles. For sake of simplicity we treat static and kinetic friction alike. In order to approximate the forces needed for friction computation we assume that the forces are proportional to the changes in position of a particle. Therefore, we can use $\Delta\mathbf{x_n}$ as an approximation of the force in normal direction of the surface and $\Delta\mathbf{x_t}$ for the force in tangential direction. We get

$$\beta = -\max\left(\frac{\|\Delta\mathbf{x_t}\| - c_f\|\Delta\mathbf{x_n}\|}{\|\Delta\mathbf{x_t}\|}, 0\right) \tag{5}$$

as scaling factor for the tangential movement where $\beta = 0$ means no movement and $\beta = 1$ no friction. In the case that no tangential movement occurs, i.e. $\|\Delta\mathbf{x_t}\| = 0$, the computation of the above equation can be skipped. For the correction vector in tangential direction follows

$$\mathbf{r_t} = (\beta - 1)\Delta\mathbf{x_t}. \tag{6}$$

This enhanced friction model increases the realism of the simulated objects without introducing much additional computations. Also, no changes in prior steps of the simulation loop have to be made, which eases implementation.

4.5 Multiple distance fields

In most virtual environments not only one object is contained, but many objects — some of them moving — have to be considered. We propose to use two stages of collision detection. First, one could use some space partitioning or bounding volume tree to find out which objects are in proximity to the simulated object. Then, our distance field algorithm is applied only to nearby objects for precise collision detection.

Another problem are all objects which are not fully rigid and therefore can not be animated by affine transformations, e. g. partially deformable objects like human avatars or kinematic chains like a robot arm. In these cases we use one distance field per rigid object. During collision detection we use bounding boxes to find all nearby objects. For these we compute all distances d_i. For further computations only the distance field with the smallest d_i is considered. In Figure 7 several snapshots of a tablecloth draped over a bending rod are shown.

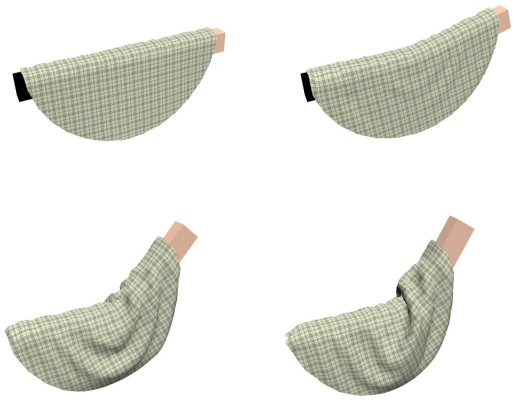


Figure 7: Four snapshots taken from an animation of a tablecloth draped over a rod which is bend in the center

5 Results

We validated our method for collision detection by integrating it into a cloth animation system. It is able to produce real-time animations of complex pieces of cloth consisting of around 1000 particles. During animation cloth dynamics, self collisions and collisions between the cloth and the environment are handled with a time step of 0.01s. In Figure 9 (a) several snapshots of a tablecloth draped over a head are shown. The head consists of around 293 000 triangles and the distance field was sampled at a resolution of $168 \times 272 \times 199$. Table 8 shows that only half of the time of one simulation step is used up for collision detection. The rest of the time can be used for dynamics and self collision handling, which are treated separately [7]. Table 8 also shows that computing the normal is not time-critical. Also, our enhanced friction model is just slightly slower than the simple one.

In order to demonstrate the precision of our algorithm a male avatar wearing a trouser is shown in Figure 9 (b) and (c). The garment was automatically pre-positioned by the algorithm described in [9]. After setting the friction coefficient to zero, the trouser begins to slide down the legs slowly. We have not noticed any jumping particles or jitter effects. Since the distance function is continuous the particles can slide smoothly along the surface. The discontinuities of the normals at neighboring grid-points were not visible at all.

In Figure 1 a sequence of an animated tablecloth, which is taken from the accompanying video, is shown. The user, which is moving the tablecloth around with the mouse, gets the impression of dragging real cloth around. The interaction is done by mapping mouse movements to the *xy* plane and by using the mouse-wheel to change the *z* values. The mesh of the tablecloth consists of 1300 particles. Since the original Buddha model is too complex to render it at interactive rates with Java3D, the mesh was replaced by the marching cubes reconstruction with 390 000 triangles. Since our collision detection algorithm is independent of the number of triangles, we are still able to achieve interactive simulation and rendering.

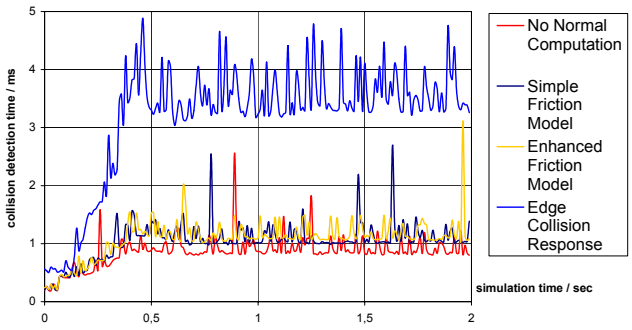


Figure 8: Comparison of collision detection and response timings during a simulation run of 2 seconds (tablecloth draped over a head, see Figure 9 (a)). When testing edges for collision the timings increase as predicted by a factor of three. Dropping the computation of normals for collision response does not gain much.

The implementation was carried out in Java and uses Java3D for the visualization. The system runs on a dual Intel Xeon^{TM} processor at 2.0 GHz. We did not yet parallelize our animation algorithm. Only the rendering routines of Java3D are using the second processor.

6 Conclusion

We have proposed an algorithm for rapid collision detection and response between rigid and deformable objects. The algorithm is in particular suited for physically based modeling of cloth as we have shown in several examples. We also believe that other areas like hair simulation may benefit from our method. Since our algorithm is very efficient, we are able to produce real-time animations of cloth.

Our method is extremely robust since due to the signed distance field we can distinguish between inside and outside. If for any reason a particle is within the object — maybe due to slightly wrong initial position or just a jump in the collision objects' movement — it will be brought back to the surface by the next collision response procedure. This method will only fail if the particle is deeper in the object than the pre-computed distance shell.

We believe that it is even possible to use our algorithm as a basis for a collision detection hardware. First, the actual algorithm is quite easy to implement and only a few conditional branches have to be made during its execution. Second, since no hierarchies are involved, each execution of a proximity query takes about the same time. This is crucial when implementing some pipelining scheme, which in turn is a must for an efficient hardware implementation.

Currently, we are investigating more complex examples of deforming rigid bodies like moving human avatars. We hope that our algorithm can be extended for real time animation of garment on avatars. Also the storage scheme of the distance field needs to be optimized in order to support high sampling rates and simultaneously low memory footprint. Possible directions are hierarchies like 2^{3N}-trees [14] or ADF's [6]. The impact of the increased distance query times of those hierarchies has to be investigated.

7 Acknowledgments

This work was partially supported by the bmb+f (Bundesministerium für Bildung und Forschung) Virtual Try-On grant. We would like to thank our Virtual Try-On project partners at the University of Bonn for providing the textures used for rendering the cloth.

We used the *MESH*-tool [1] for computation of the *Hausdorff distance*. The avatar was generated with Curious Lab's *Poser*. The *Buddha* and *Turbine* meshes have been taken from the *Large Geometric Models Archive* at `www.cc.gatech.edu/projects/large_models`. The Joseph von Fraunhofer bust model was provided by the department of Cognitive Computing and Medical Imaging at the Fraunhofer Institute of Computer Graphics.

References

[1] ASPERT, N., SANTA-CRUZ, D., AND EBRAHIMI, T. Mesh: Measuring errors between surfaces using the hausdorff distance. In *Proceedings of the IEEE International Conference on Multimedia and Expo* (2002), vol. I, pp. 705 – 708. `http://mesh.epfl.ch`.

[2] BARAFF, D., AND WITKIN, A. Large steps in cloth simulation. In *SIGGRAPH 98 Conference Proceedings* (Orlando, FL, USA, July 1998), M. Cohen, Ed., Annual Conference Series, ACM SIGGRAPH, pp. 43–54.

[3] BREEN, D. E., MAUCH, S., WHITAKER, R. T., AND MAO, J. 3d metamorphosis between different types of geometric models. *Eurographics 2001 Proceedings 20(3)* (2001), 36–48.

[4] BRIDSON, R., FEDKIW, R., AND ANDERSON, J. Robust treatment of collisions, contact and friction for cloth animation. In *ACM Transactions on Graphics (SIGGRAPH 2002)* (2002), vol. 21.

[5] FISHER, S., AND LIN, M. Fast penetration depth estimation for elastic bodies using deformed distance fields. In *In Proc. International Conf. on Intelligent Robots and Systems (IROS)* (2001).

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

[7] FUHRMANN, A., GROSS, C., AND LUCKAS, V. Interactive animation of cloth including self collision detection. *Journal of WSCG 11*, 1 (Feb. 2003), 141–148.

[8] GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. OBBTree: a hierarchical structure for rapid interference detection. In *SIGGRAPH 96 Conference Proceedings* (New Orleans, LA, USA, Aug. 1996), Annual Conference Series, ACM SIGGRAPH, pp. 171–180.

[9] GROSS, C., FUHRMANN, A., AND LUCKAS, V. Automatic prepositioning of virtual clothing. In *Proceedings of the Spring Conference on Computer Graphics* (Apr. 2003), pp. 113–122.

[10] GUIBAS, L. J., HSU, D., AND ZHANG, L. H-walk: hierarchical distance computation for moving convex bodies. In *Proceedings of 15th ACM Symposium on Computational Geometry* (1999), pp. 344–351.

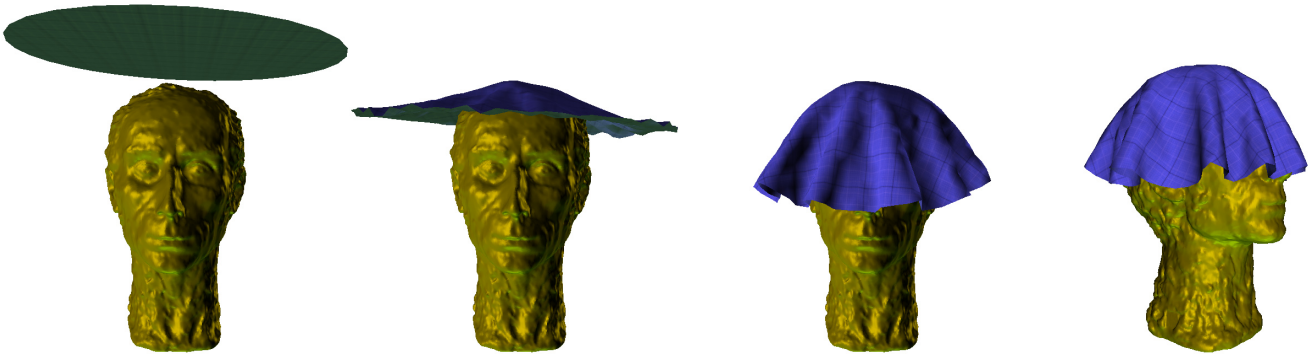
[11] HUBBARD, P. M. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics 15*, 3 (1996), 179–210.

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

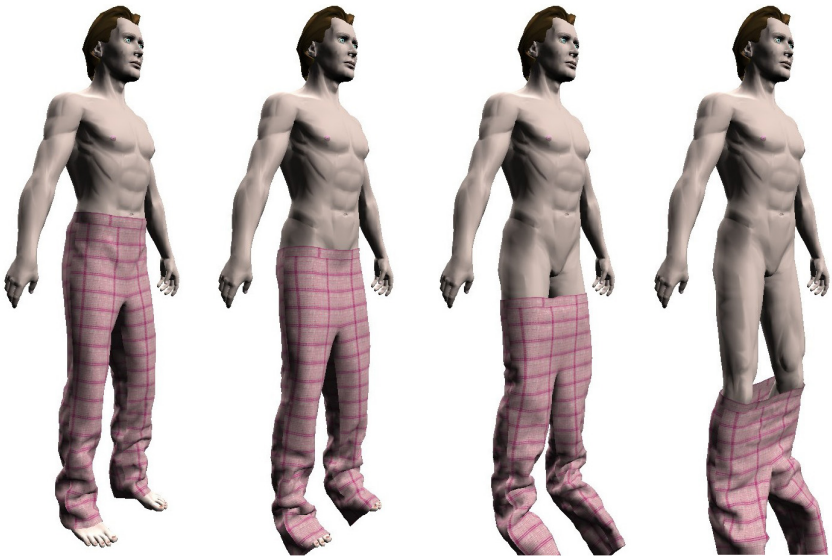
[13] KLOSOWSKI, J., HELD, H., MITCHELL, J., ZIKAN, K., AND SOWIZRAL, H. Efficient collision detection using bounding volume hierarchies of *k*-DOPs. *IEEE Trans. Visual. Comput. Graph. 4*, 1 (1998).

[14] MCNEELY, W. A., PUTERBAUGH, K. D., AND TROY, J. J. Six degree-of-freedom haptic rendering using voxel sampling. In *SIGGRAPH 99 Conference Proceedings* (Los Angeles, CA, USA, Aug. 1999), Annual Conference Series, ACM SIGGRAPH, pp. 401–408.

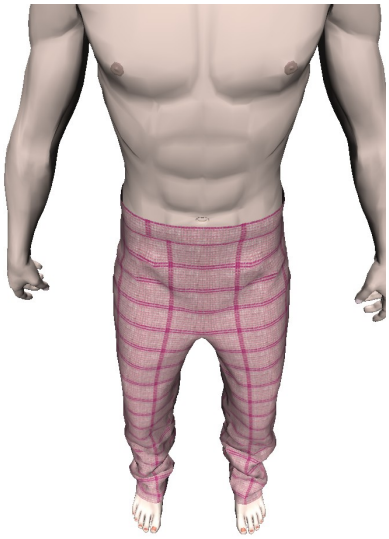
[15] METZGER, J., KIMMERLE, S., AND ETZMUSS, O. Hierarchical techniques in collision detection for cloth animation. *Journal of WSCG 11*, 2 (Feb. 2003), 322–329.



(a) Several snapshots from draping a tablecloth over a head.



(b) After setting the friction coefficient to zero, the trouser slides smoothly down the legs.



(b) Accurate collision detection at the hip.

Figure 9: Some screen shots taken from our interactive cloth animation system demonstrating the capabilities of our collision detection algorithm.

[16] MEYER, M., DEBUNNE, G., DESBRUN, M., AND BARR, A. H. Interactive animation of cloth-like objects for virtual reality. *The Journal of Visualization and Computer Animation 12* (May 2001), 1–12.

[17] MIRTICH, B. V-Clip: fast and robust polyhedral collision detection. *ACM Transactions on Graphics 17*, 3 (1998), 177–208.

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

[19] PROVOT, X. Collision and self-collision handling in cloth model dedicated to design garments. In *Graphics Interface 97* (1997), pp. 177–189.

[20] SATHERLEY, R., AND JONES, M. W. Hybrid distance field computation. In *Volume Graphics 2001* (Wien New York, 2001), K. Mueller and A. Kaufman, Eds., Springer, pp. 195–209.

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

[22] SETHIAN, J. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, Cambridge, UK, 1999.

[23] VAN DEN BERGEN, G. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools 2*, 4 (1997), 114.

[24] VASSILEV, T., SPANLANG, B., AND CHRYSANTHOU, Y. Fast cloth animation on walking avatars. In *Proceedings of Eurographics 2001* (2001).

[25] VOLINO, P., AND MAGNENAT-THALMANN, N. *Virtual Clothing, Theory and Practice*. Springer, 2000.

[26] ZACHMANN, G. Rapid collision detection by dynamically aligned dop-trees. In *Proc. of IEEE Virtual Reality Annual International Symposium; VRAIS '98* (Atlanta, Georgia, March 1998).

[27] 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.

[28] ZHANG, D., AND YUEN, M. M. Collision detection for clothed human animation. In *Proceedings of the 8th Pacific Graphics Conference on Computer Graphics and Application* (2000), pp. 328–337.