Contact Handling for Deformable Point-Based Objects

Richard Keiser, Matthias Müller, Bruno Heidelberger, Matthias Teschner, Markus Gross

Computer Graphics Lab
ETH Zürich
8092 Zürich, Switzerland
Email: keiser@inf.ethz.ch

Abstract

This paper presents an approach to collision detection and response for dynamically deforming point-based objects. Both the volume of an object and its surface are represented by point sets. In case of a collision, response forces are computed for penetrating surface points and distributed to volume points which are used for simulating the object dynamics. The decoupling of collision handling and deformation allows for a very stable collision response while maintaining interactive update rates of the dynamic simulation for environments with moderate complexity. Simulation results are presented for elastically and plastically deforming objects with changing topology.

1 Introduction

Deformable objects play an important role in many interactive virtual environments such as surgery simulators or 3D computer games. Soft objects deform due to external forces exerted during collisions with the environment or with other objects such as a user guided tool. Therefore, both, the collision detection algorithm, as well as the collision response model play a central role in the simulation of deformable objects.

The traditional approach to simulating deformable objects is to represent them by volumetric meshes. The elastic or plastic behavior is then modeled by discretizing the continuous elasticity equations on these meshes. Many collision detection algorithms and collision response models have been proposed for deformable, mesh-based objects.

In recent years, point-based representations for the surface as well as for the volume of deformable objects have been investigated in computer graphics. Since they do not store the connectivity explicitly, they can handle topological changes more easily. While there are a few methods available for simulating deformable, point-based objects, the investigation of robust collision detection and response algorithms for point representations is still an open research topic.

In this paper, we present a collision detection and response algorithm for point-based animation, where both the volume and the surface are point sampled. The method stably resolves collisions for stiff elastic and highly deformable or plastic models. During collisions, it deforms the point-based surface and exerts forces on the points representing the volume yielding plausible collision simulations at interactive speed.

2 Related Work

Collision detection for deformable surfaces has recently gained a lot of attention in the collision detection community. A survey of recent research can be found in Teschner et al [21].

Many different approaches exist for collision handling of deforming objects. Penalty methods, pioneered by Moore and Wilhelms [12], are probably the most widely used solutions in computer graphics. However, these methods cannot ensure that the objects do not penetrate. Baraff and Witkin [2] use a constraint-based method to prevent objects from penetration. This requires solving a linear complementary problem (LCP), which is computationally expensive for complex objects. Impulse-based methods [7, 11] assume short contacts only, and therefore they are not suitable for soft objects. Desbrun and Cani [5, 3] presented a system for animation and collision handling of implicit surfaces generated by skeletons. Exact contact surfaces are achieved by deforming the implicit layer. The compression of the surface yields response forces which

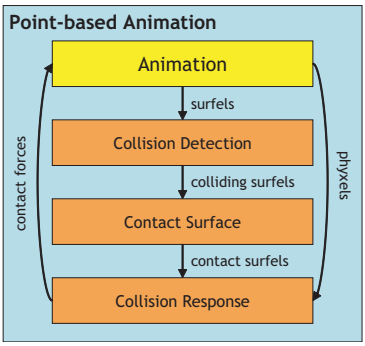


Figure 1: Overview of our contact handling framework, shown in orange.

are transmitted to the skeleton. However, the generated implicit models tend to be blobby.

Point-sampled objects have become very popular in recent years [18, 25, 26, 6]. In this context, the problem of collision detection and response has only been addressed very recently. Pauly et al. [15] were the first dealing with collision detection of point-sampled objects in their shape modeling system. Klein and Zachmann [9] presented an approach for time-critical collision detection of point clouds, where they use a sphere bounding hierarchy. However, they do not deal with collision response. Pauly et al. [16] model contact for point-sampled quasi-rigid objects, i.e. rigid objects with elastic surfaces. They compute exact contact surfaces by setting up linear complementarity constraints and solving for the tractions that act on these surfaces. The wrench on the rigid object is then computed from these tractions.

Recently, Müller et al. [14] presented a physical model derived from continuum mechanics for animating volumetric objects, where both the volume and the surface are point-based. Our collision detection and response framework is built on top of their work.

3 Overview

Figure 1 gives an overview of our collision detection and response framework. It is built on top of a point-based animation, which we describe in Section 4.

After each animation step, the collision detection algorithm gets the point-sampled surfaces (collections of surfels) as input and computes the colliding surfels (Section 5). From the colliding surfels, a contact surface is computed which resolves the intersecting surfaces in a plausible way (Section 6). For the surfels on the contact surface (called contact surfels), penalty forces are compute (Section 7). These forces are then distributed to the neighboring physical points, called phyxels.

4 Point-based Animation and Fracturing

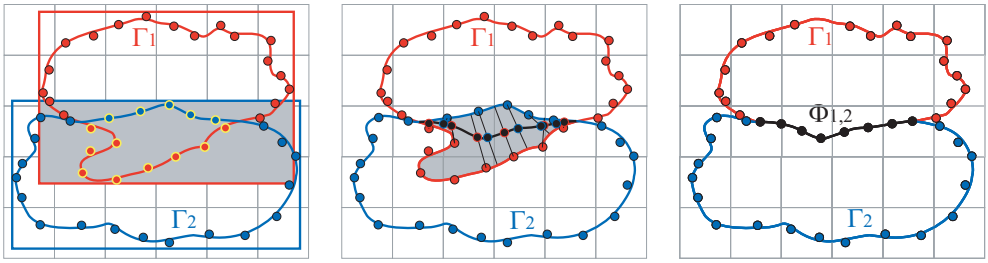
For our collision method to work, all we require from the animation method is that the volumes of the objects are discretized into a set of points (or phyxels) on which external forces can be applied, and a set of surface elements (or surfels) which are animated along with the phyxels. Even though we use the point-based animation method described in [14], the collision algorithm would work with SPH-based approaches [4] as well as with Lennard-Jones-based techniques [22]. In all these approaches, the phyxels typically have a finite influence radius h. During the animation, fracturing naturally occurs when the phyxels can be split in two or more groups where all phyxels in each group have a distance larger than h to all phyxels in other groups. Once an object has fractured, its subobjects should be prevented from merging again when the distance between them becomes smaller than h. We use a technique similar to the time-varying blending graph proposed by Desbrun and Cani [3] to identify fractured parts. Once two phyxels belong to different groups, they receive different marks which prevents them from being merged later.

4.1 Point-based Objects

We define a point-based object Γ_k as a cloud of surfels and phyxels, consisting of n surfels s_k with position $\boldsymbol{s}_k \in \mathbb{R}^3$, and m phyxels p_k with position $\boldsymbol{p}_k \in \mathbb{R}^3$.

For collision detection and response we will often make use of an implicit representation S_k of the surface defined by the surfels $s_k \in \Gamma_k$. We compute the implicit representation as suggested by Alexa et al [1]:

$$S_k = \{\boldsymbol{x} \in \mathbb{R}^3 | f_k(\boldsymbol{x}) = 0\} \quad (1)$$



(a) (b) (c)

Figure 2: (a) Detection of colliding surfels by first intersecting the bounding boxes (Section 5). The points in the shadowed region are collision candidates. Colliding points are outlined in yellow. (b) In the case where both objects have the same stiffness, the contact surface is the middle of the intersecting volume, shown as black line. It is initialized by moving the colliding points onto it (Section 6). (c) The final surfaces after resampling the contact surface.

and $f_k(\boldsymbol{x}) = \hat{\boldsymbol{n}}_k(\boldsymbol{x})^T(\boldsymbol{x} - \boldsymbol{a}_k(\boldsymbol{x}))$, where $\hat{\boldsymbol{n}}_k(\boldsymbol{x})$ is the unit normal vector at $\boldsymbol{x}$ and $\boldsymbol{a}_k(\boldsymbol{x})$ is the weighted average of neighbored points to $\boldsymbol{x}$, i.e.

$$\boldsymbol{a}_k(\boldsymbol{x}) = \frac{\sum_{s_k}(\theta(\|\boldsymbol{x} - \boldsymbol{s}_k\|)\boldsymbol{s}_k)}{\sum_{s_k}\theta(\|\boldsymbol{x} - \boldsymbol{s}_k\|)},$$

where θ is a smooth, positive, and monotonically decreasing weighting function. We choose a truncated Gaussian weighting function. The normal vector $\hat{\boldsymbol{n}}_k(\boldsymbol{x})$ is computed as the weighted average of the neighboring surfel normal vectors $\hat{\boldsymbol{n}}_{s_k}$, i.e.

$$\hat{\boldsymbol{n}}_k(\boldsymbol{x}) = \frac{\sum_{s_k}(\theta(\|\boldsymbol{x} - \boldsymbol{s}_k\|)\hat{\boldsymbol{n}}_{s_k})}{\|\sum_{s_k}\theta(\|\boldsymbol{x} - \boldsymbol{s}_k\|)\hat{\boldsymbol{n}}_{s_k}\|}.$$

In [1], a simple procedure for computing orthogonal projections of a point onto S_k is described. We refer to [1] for details.

5 Collision Detection

Collision detection for point-based objects amounts to finding surfels that are inside other surfel-bound objects.

Bounding volume hierarchies are a means to reduce the number of primitives (surfels in our case) to be tested for intersection. However, because we deal with highly deformable models, the update of such a hierarchy is often very costly. Therefore, we approximate the objects by only one bounding volume to efficiently discard collisions if the bounding volumes do not overlap.

We choose axis-aligned bounding boxes (AABBs) [23] because of their efficient computation. Furthermore, the intersection volume of two overlapping AABBs is again an AABB, here called B. Only the surfels s which are inside B are candidates for a collision. We can efficiently find these surfels by first inserting all surfels into a uniform spatial grid and then intersecting B with the cells of this grid (see Figure 2 (a)). The grid is used afterwards as a search data structure for range and neighbor queries. This approach is similar to Teschner et al [20].

In a next step, we determine whether the collision candidates in B are actually penetrating other objects. Pauly et al. [15] used the MLS surface [10] as an implicit representation of a point-sampled model. Instead of the MLS surface, we use the more efficient implicit surface representation S_k described in Section 4.1. Such an implicit representation allows to easily and efficiently specify if a point is inside or outside of a point-based object Γ_k as follows: First, project a point $\boldsymbol{x}$ onto S_k, yielding the projected point $\boldsymbol{x}_k^{proj}$. We then define $\boldsymbol{x}$ as being inside, and therefore colliding, if

$$(\boldsymbol{x} - \boldsymbol{x}_k^{proj}) \cdot \hat{\boldsymbol{n}}_{\boldsymbol{x}_k^{proj}} < 0, \qquad (2)$$

where $\hat{\boldsymbol{n}}_{\boldsymbol{x}_k^{proj}}$ is the normal vector at $\boldsymbol{x}_k^{proj}$.

For simplicity, we assume in the following sections that we have only two colliding point-based objects Γ_1 and Γ_2, as defined in Section 4.1. In Section 8, we discuss contact handling for an arbitrary number of objects.

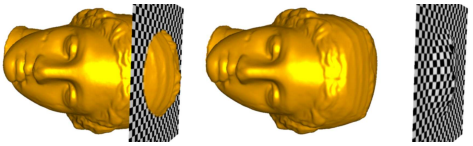


Figure 3: Left: collision of Igea with plane without surface contact handling. Middle and right: surface response of Igea and plane, respectively.

6 Contact Surface

At the time of collision two colliding surfaces intersect. We get a visually consistent contact surface by assuming that the surface is completely elastic. Thus, we *temporarily* recompute the intersecting surfaces such that they touch. The displacement of the surface to the contact surface is used for computing the collision response, as described in the next section, and for rendering.

Assume that we have two intersecting point-based objects Γ_1 and Γ_2. We aim to efficiently compute a reasonable contact surface $\Phi_{1,2}$ which lies in the intersection volume $V = \Gamma_1 \cap \Gamma_2$.

We define distance functions $f_1(\boldsymbol{x})$ and $f_2(\boldsymbol{x})$ as the distance of a point $\boldsymbol{x} \in V$ to the implicit surface representation of Γ_1 and Γ_2, respectively. This distance can be efficiently computed applying the projection operator [1]. By defining an implicit function $F(\boldsymbol{x}) = \lambda_1 f_1(\boldsymbol{x}) - \lambda_2 f_2(\boldsymbol{x})$, the contact surface $\Phi_{1,2}$ can be implicitly defined as

$$\Phi_{1,2} = \{\boldsymbol{x} \in V | F(\boldsymbol{x}) = 0\}, \qquad (3)$$

where λ_1 and λ_2 depend on the surface material. For instance, if we choose a high value for λ_1 compared to λ_2, then the intersecting surface of Γ_1 will adapt to the intersecting surface of Γ_2. We choose λ_i dependent on Young's Modulus E_i of a material, e.g. $\lambda_i = \frac{1}{E_i}$, i.e. elastic materials (low E) adapt to stiff materials (high E).

To get an initial sampling of $\Phi_{1,2}$, we move the colliding surfels onto $\Phi_{1,2}$, see Figure 2 (b). For finding the position $\boldsymbol{s}_{\Phi_{1,2}}$ of a surfel $s_1 \in \Gamma_1$ such that $F(\boldsymbol{s}_{\Phi_{1,2}}) = 0$, the Newton-Raphson method could be used [19]. However, we would need to compute $\nabla F(\boldsymbol{s}_1)$, e.g. using central differences. Furthermore, this method might not converge to a solution. Therefore, we reduce the problem of finding the root to a 1D problem by projecting a surfel $\boldsymbol{s_1}$ onto the implicit surface representation S_2 of Γ_2 as described in Section 4.1. The projected position be $\boldsymbol{x}_2$. Now, we look for the point $\boldsymbol{x}$ with $F(\boldsymbol{x}) = 0$ on the line between $\boldsymbol{s}_1$ and $\boldsymbol{x}_2$. This can be done iteratively using Brent's algorithm [19]. If the boundary of the intersecting volume is not convex, it might happen that the line between $\boldsymbol{s}_1$ and $\boldsymbol{x}_2$ is not inside the volume. Therefore, we check after each iteration if the computed position on this line is inside the volume. If it is not, we simply delete this surfel.

To get an even and hole free sampling of the contact surface, we apply the resampling scheme proposed in [14], consisting of relaxation and resampling steps. During the relaxation steps the surfels are distributed and then reprojected onto $\Phi_{1,2}$, until the whole surface is covered. While the relaxation results in a *locally* uniform distribution, a certain density of surfels ensuring that the whole surface is covered is achieved during the resampling steps by inserting or deleting surfels. The combination of both relaxation and resampling results in an efficient algorithm for covering a certain area, as was already stated by Witkin and Heckbert [24].

For visual accuracy, the normal vectors are recomputed using Principal Component Analysis (PCA), i.e. we compute the Eigenvectors of the covariance matrix of the local neighborhood, where the Eigenvector corresponding to the smallest Eigenvalue is the normal direction [8].

7 Collision Response

We have shown above how a contact surface can be computed from the colliding surfels. We call the surfels on this surface contact surfels. The information gained during the creation of the contact surface is used to apply a penalty method which separates intersecting objects, i.e. forces are computed which act against the penetration. While penalty methods are efficient to compute and yield stable animations, they do not prevent objects from penetrating. However, since we have already handled penetration by deforming the surface, a simple penalty method is sufficient as collision response model for the volume.

In the following subsections, we describe an approach which computes a penalty and friction force for each contact surfel. These forces are then distributed to the phyxels of the colliding objects. We consider two colliding objects Γ_1 and Γ_2 and derive the forces for Γ_1. The forces for Γ_2 are computed

analogous.

7.1 Penalty Force Computation

The penalty force separates two intersecting objects along the deformation direction, i.e. the displacement direction from Γ_1 to $\Phi_{1,2}$ of a surfel $s_c \in \Phi_{1,2}$. We estimate this direction $\boldsymbol{d}_1$ by searching for the closest surfel $s_1 \in \Gamma_1$ to $\boldsymbol{s}_c$, i.e. $\boldsymbol{d}_1 = \boldsymbol{s}_1 - \boldsymbol{s}_c$, see Figure 4. Here we assume small penetrations. For large penetrations, the direction to the closest surfel does not necessarily correspond to the correct deformation direction, see Section 9.

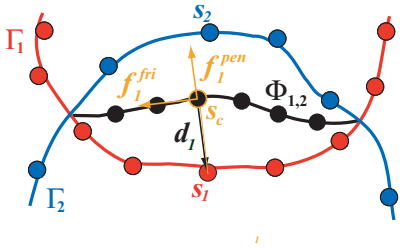


Figure 4: Penalty and friction force computation for Γ_1 of a surfel $s_c \in \Phi_{1,2}$.

We then set up a force-displacement relationship between $\boldsymbol{s}_c$ and $\boldsymbol{s}_1$. We choose a linear elastic model, i.e. we attach a linear spring of zero rest length between $\boldsymbol{s}_c$ and $\boldsymbol{s}_1$ resulting in the following force:

$$\boldsymbol{f}_1^{pen} = -k^{spring}\boldsymbol{d}_1,$$

where k^{spring} is the spring stiffness. This stiffness is a parameter of our collision response model. It controls the trade-off between quality, i.e. small penetrations, and the stability of the simulation. The stability problems in connection with stiff springs can be reduced by adding damping:

$$\boldsymbol{f}_1^{pen} = -(k^{spring}\boldsymbol{d}_1 + k^{damp}\dot{\boldsymbol{d}}_1). \quad (4)$$

The time derivative of the deformation depth $\dot{\boldsymbol{d}}_1$ is equal to the relative velocity in direction of $\boldsymbol{d}_1$, i.e. $\dot{\boldsymbol{d}}_1 = (\boldsymbol{v}_1^{rel} \cdot \hat{\boldsymbol{d}}_1)\hat{\boldsymbol{d}}_1$, where $\hat{\boldsymbol{d}}_1$ is normalized, $\boldsymbol{v}_1^{rel} = (\boldsymbol{v}_1 - \boldsymbol{v}_2)$, $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are the velocities at $\boldsymbol{s}_1$ and at the closest surfel $\boldsymbol{s}_2 \in \Gamma_2$ to $\boldsymbol{s}_c$, respectively.

Note that because the penalty force is linear to the displacement, and the displacement depends on Young's Modulus, the penalty force also dependents on Young's Modulus.

7.2 Friction Force Computation

To prevent two objects from freely sliding relative to one another, we apply a Coulomb friction force as proposed in [11] to each surfel s_c:

$$\boldsymbol{f}_1^{fri} = -\mu^{kin}\|\boldsymbol{f}_1^{pen}\|\hat{\boldsymbol{v}}_1^{per}, \quad (5)$$

where $\hat{\boldsymbol{v}}_1^{per}$ is the normalized velocity vector tangential to the normalized penetration direction $\hat{\boldsymbol{d}}_1$, i.e. $\boldsymbol{v}_1^{per} = \boldsymbol{v}_1^{rel} - (\boldsymbol{v}_1^{rel} \cdot \hat{\boldsymbol{d}}_1)\hat{\boldsymbol{d}}_1$, and $\boldsymbol{v}_1^{rel}$ is the relative velocity as defined in the previous section.

7.3 Force Distribution

The total collision response force of a surfel $s_c \in \Phi_{1,2}$ for Γ_1 is the sum $\boldsymbol{f}_1^{pen} + \boldsymbol{f}_1^{fri}$ of the forces defined in Equation (4) and (5).

According to Newton's third law $actio = reactio$, each force yields a force in the opposite direction. Therefore, s_c also receives the reaction force of Γ_2 and vice versa, i.e.

$$\boldsymbol{f}_1 = \frac{(\boldsymbol{f}_1^{pen} + \boldsymbol{f}_1^{fri}) - (\boldsymbol{f}_2^{pen} + \boldsymbol{f}_2^{fri})}{2}, \quad (6)$$

such that $\boldsymbol{f}_{s_c} = \boldsymbol{f}_1 + \boldsymbol{f}_2 = 0$. The factor $\frac{1}{2}$ is needed because each force is considered twice, once as a force onto Γ_1 and once onto Γ_2.

To get a penetration force that is independent of the surface sampling, we scale the force with the local surfel density ρ_{s_c}, i.e.

$$\boldsymbol{f}_1^{scal} = \rho_{s_c}\boldsymbol{f}_1. \quad (7)$$

We use a simple estimation for ρ_{s_c} as suggested by [17]:

$$\rho_{s_c} = \frac{k}{\pi r_{s_c}^2},$$

where r_{s_c} is the distance to the furthest of the k nearest neighbors of s_c.

The scaled forces $\boldsymbol{f}_1^{scal}$ defined on the contact surfels are then distributed to the phyxels $p_1 \in \Gamma_1$ using a weighting function ω, which depends on the distance $d_{s_c,p_1} = \|\boldsymbol{s}_c - \boldsymbol{p}_1\|$. As a weighting function we use the even polynomial kernel proposed by Müller et al. [13] for SPH. It has the nice property that it is zero with vanishing derivatives at the boundary:

$$\omega_{s_c,p_1} = \begin{cases} \frac{315}{64\pi h^9}(h^2 - d_{s_c,p_1}^2)^3 & \text{if } d_{s_c,p_1} \leq h \\ 0 & \text{otherwise,} \end{cases}$$

where h is the support of the kernel. We choose h equal to two times the influence radius of the phyxels. If the number of phyxels within h is locally smaller than k (e.g. $k = 6$), we choose h to be equal to the distance to the furthest of the k nearest neighbors. The forces of all surfels need to be distributed to the phyxels such that the sum of all surfel collision forces is equal to the sum of all phyxel collision forces. The force $\boldsymbol{f}_{s_c,p_1}$ acting on a phyxel p_1 induced by a surfel s_c is then

$$\boldsymbol{f}_{s_c,p_1} = \frac{\omega_{s_c,p_1} \boldsymbol{f}_1^{scal}}{\sum_{p_1 \in \Gamma_1} \omega_{s_c,p_1}}. \quad (8)$$

The total collision force acting on a phyxel p_1 is the accumulation of all surfel forces

$$\boldsymbol{f}_{p_1} = \sum_{s_c \in \Gamma_1} \boldsymbol{f}_{s_c,p_1}. \quad (9)$$

This force will act on p_1 like an external force at the next time step.

8 Implementation & Results

The function `animate` shown below is a pseudocode version of one animation step for all point-based objects $\Gamma_i = (S_i, P_i)$, where S_i is a set of surfels and P_i a set of phyxels. We assume that all data structures are already initialized. First, the phyxels are animated, and along with the phyxels the surfels. A description of our animation framework is beyond this paper, and we refer the interested reader to [14]. For fast neighbor and range queries of the phyxels and surfels, a hash data structure is used as suggested by Teschner et al [20]. With this data structure, insertion and spatial queries for points have constant time complexity.

After both phyxels and surfels are animated, collision handling for each pair of objects is performed. A bounding box is computed for both objects and the surfels in the intersecting box are checked for collision as described in Section 5. The colliding surfels are moved to the contact surface Φ. Afterwards, Φ is resampled and for visual accuracy the normals are recomputed using PCA (Section 6). The penalty forces are computed for each contact surfel on Φ and then distributed to the neighboring phyxels (Section 7). Finally, the new surface S' consists of the non-colliding surfels unified with Φ. Note that after having computed the penalty forces, the new collision free surface S' is used only for rendering. The next time step operates on the original surface S. However, S' is also used for contact handling with further objects. Therefore, contact handling depends on the order of the object pairs. However, in practice there is only little difference.

Function `animate`

```
/* Animation of Point-based Objects */
foreach Γ_i = (S_i, P_i) do
    animatePhyxels(P_i);
    h_{P_i} ← updatePhyxelSearchDS(P_i);
    animateSurfels(S_i, h_{P_i});
/* Contact Handling */
foreach Pair (Γ_j,Γ_k) do
    h_{S_j} ← updateSurfelSearchDS(S_j);
    h_{S_k} ← updateSurfelSearchDS(S_k);
    bb_j ← getBoundingBox(S_j);
    bb_k ← getBoundingBox(S_k);
    bb ← intersectBoundingBoxes(bb_j, bb_k);
    C_j ← getCollidingSurfels(h_{S_j}, bb, h_{S_k});
    C_k ← getCollidingSurfels(h_{S_k}, bb, h_{S_j});
    Φ ← displaceSurfels(C_j, C_k, h_{S_j}, h_{S_k});
    resampleContactSurface(Φ);
    recomputeNormalVectors(Φ);
    computeSurfelForces(Φ, h_{S_k}, h_{S_j});
    computePhyxelForces(Φ, h_{P_j});
    computePhyxelForces(Φ, h_{P_k});
    S'_j ← S_j \ C_j ∪ Φ;
    S'_k ← S_k \ C_k ∪ Φ;
```

An example for contact handling is shown in Figure 6 (see color pages), where three plastic Max Planck busts fall onto each other building a pile, fall apart again and finally come to rest (see our video at http://cg.inf.ethz.ch/∼rkeiser/Papers/VMV2004/).

We use stiff elastic spheres for Newton's Cradle, see Figure 7. In the middle picture the contact surface of the first sphere is shown.

The Santa Claus riding the dragon, shown in Figure 8, is an example for contact handling between two highly complex models. To prevent that the mouth of the dragon self-intersects we store all neighbors within the influence distance. If two phyxels come close which were not in the influence region before, a simple penalty force is applied to the two phyxels.

In Figure 9 we perform the same animation as in [14]. The Igea's hair shock is fixed while the gravitation force is pulling its head down. Because the object is highly plastic, it first stretches and then fractures. Our algorithm detects the two separated parts and converts them into two separated objects. Afterwards, contact handling between the two objects is performed as described above.

An example of a contact surface is shown in Figure 3, where the Igea penetrates a plane deeply due to a small k^{spring} constant. The plane is not animated, but the same surface response is applied to it as to the Igea. This yields a flattening of the Igea's head, to which the plane adopts.

In Figure 5, we show the time measurements of two colliding Max Planck busts with each 10k surfels and 390 phyxels on a Pentium 4 with 3 GHz and 1 GB RAM. For rendering we use a fast surfel renderer exploiting our NVidia GeForce FX 5900 graphics card. We made the busts elastic such that the physical animation performs in about constant time. The green line shows the total time needed for the animation including contact handling and rendering. The time needed for contact handling is shown in blue, and the number of colliding surfels is shown in red.

There are many factors influencing the performance: For a large number c of colliding surfels, computing the phyxel forces is the most expensive step. This can be done in $O(c \cdot p)$, where p is the average number of phyxels within the influence radius h of a colliding surfel. Thereby we assume that range queries can be done in constant time using the hash data structure, see [20]. The time complexity for computing the colliding surfels depends linearly on the number of surfels in the intersected bounding box. For computing the contact surface, the penetration depth is crucial for the performance. Deeper penetration usually means that more iterations are needed to find the contact surface Φ and to resample it.

9 Limitations

Like all discrete collision detection algorithms, our approach fails if an object completely moves through another object between two time steps. Furthermore, the computation of the penalty force depends on an accurate estimation of the deformation direction. Therefore, the algorithm is especially susceptible to fail for deep penetrations, where the penalty force might point to other directions than the actual deformation direction. Also the contact resolving of the surface might be very costly for deep penetrations.

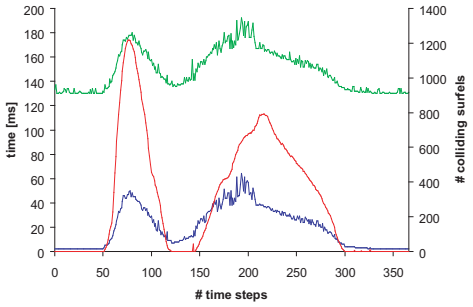


Figure 5: Performance measurement for two colliding Max Plancks. Green: total animation time. Blue: contact handling time. Red: number of colliding surfels.

10 Conclusion & Future Work

In this paper, we have presented a novel approach for interactive contact handling of point-based animated objects.

We have presented an algorithm for efficiently computing a contact surface for intersecting surfaces. For the surfels on this contact surface a collision response force is computed. These forces are then distributed to the phyxels.

By using two different object representations for collision detection and deformation, i.e. a high resolution surface (surfels) for contact handling and a low resolution representation (phyxels) for the deformation, the animation is both stable and efficient. Furthermore, we never observed oscillations for resting contacts, as they often arise in meshes. We believe that this is due to the smoothing effect of the distribution of the surfel forces to the phyxels. Furthermore, computing a contact surface is efficient using points because a simple and fast resampling can be applied without having to deal with consistency constraints.

Note that our collision detection and response algorithms could be applied for meshes as well. The forces could be distributed to the vertices of the tetrahedra similar to the phyxels. If a high resolu-

tion mesh is used for the surface, only the vertices could be checked for collision similar to the surfels.

A difficult problem to deal with in the future is self-collision detection for point-based objects. This is far from trivial because no explicit connectivity information is given. To make contact handling even more efficient, spatial and temporal coherence could be exploited.

11 Acknowledgements

This research is supported by the Swiss National Science Foundation and by the Swiss National Commission for Technology and Innovation (KTI). The project is part of the Swiss National Center of Competence in Research on Computer Aided and Image Guided Medical Interventions (NCCR Co-Me) and KTI project no. 6310.1 KTS-ET.

References

[1] M. Alexa and A. Adamson. On normals and projection operators for surfaces defined by point sets. In *Proceedings of Eurographics Symposium on Point-Based Graphics 2004*, pages 150–155, 2004.

[2] D. Baraff and A. Witkin. Dynamic simulation of non-penetrating flexible bodies. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, 1992.

[3] M. Desbrun and M.-P. Cani. Animating soft substances with implicit surfaces. In *SIGGRAPH 95 Conference Proceedings*.

[4] M. Desbrun and M.-P. Cani. Smoothed particles: A new paradigm for animating highly deformable bodies. In *6th Eurographics Workshop on Computer Animation and Simulation '96*, pages 61–76, 1996.

[5] Marie-Paule Gascuel. An implicit formulation for precise contact modeling between flexible solids. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, 1993.

[6] G. Guennebaud, L. Barthe, and M. Paulin. Deferred splatting. In *Proceedings of EUROGRAPHICS 2004*, 2004. To appear.

[7] J. K. Hahn. Realistic animation of rigid bodies. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, 1988.

[8] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th annual conference on computer graphics and interactive techniques*, 1992.

[9] J. Klein and G. Zachmann. Point cloud collision detection. In *Proceedings of EUROGRAPHICS 2004*. To appear.

[10] David Levin. Mesh-independent surface interpolation. 2003.

[11] B. Mirtich and J. Canny. Impulse-based simulation of rigid bodies. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, 1995.

[12] M. Moore and J. Wilhelms. Collision detection and response for computer animation. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, 1988.

[13] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of 2003 ACM SIGGRAPH Symposium on Computer Animation*, pages 154–159, 2003.

[14] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2004*, 2004. To appear.

[15] M. Pauly, R. Keiser, L. P. Kobbelt, and M. Gross. Shape modeling with point-sampled geometry. In *Computer Graphics*, Siggraph 2003 Proceedings, pages 641–650, 2003.

[16] M. Pauly, D. Pai, and L. Guibas. Quasi-rigid objects in contact. In *Proceedings of Eurographics Symposium on Computer Animation 2004*. To appear.

[17] Mark Pauly. *Point Primitives for Interactive Modeling and Processing of 3D Geometry*. PhD thesis, Department of Computer Science, ETH Zurich, 2003.

[18] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000.

[19] W. H. Press, S. A. Teukolsky, W. T. Vettering, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[20] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranerts, and M. Gross. Optimized spatial hashing for collision detection of deformable object. In *Proceedings of Vision, Modeling, Visualization VMV03*, pages 47–54.

[21] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, L. Raghupathi, A. Fuhrmann, M-P. Cani, F. Faure, N. Magnetat-Thalmann, and W. Strasser. Collision detection for deformable objects. In *Proc. Eurographics State-of-the-Art Report*. EG Association, 2004. To appear.

[22] David Tonnesen. *Dynamically Coupled Particle Systems for Geometric Modeling, Reconstruction, and Animation*. PhD thesis, University of Toronto, November 1998.

[23] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *J. Graph. Tools*, 2(4), 1997.

[24] A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. In *Computer Graphics Proceedings*, pages 269–277. ACM SIGGRAPH, 1994.

[25] M. Zwicker, M. Pauly, O. Knoll, and M. Gross. Pointshop 3d: An interactive system for point-based surface editing. In *Computer Graphics*, SIGGRAPH 2002 Proceedings, pages 322–329.

[26] M. Zwicker, J. Rasanen, M. Botsch, C. Dachsbacher, and M. Pauly. Perspective accurate splatting. 2004.

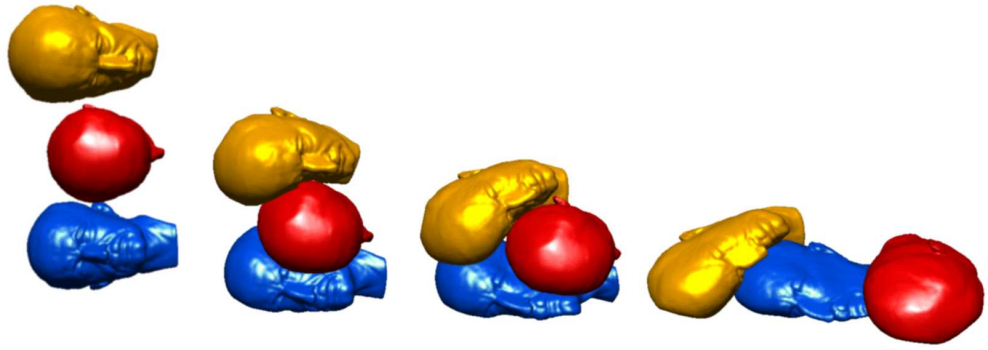


Figure 6: Plastic Max Plancks building a pile and falling apart again.

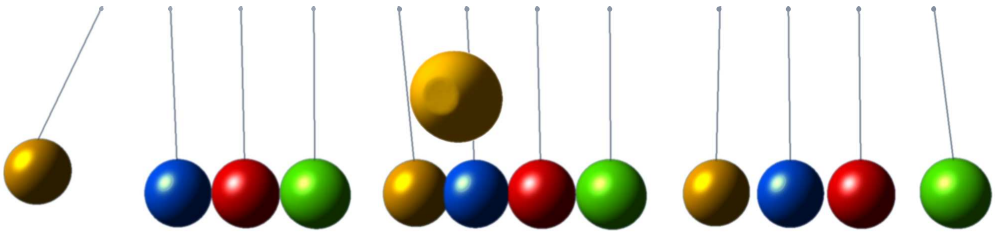


Figure 7: Newton’s Cradle with stiff elastic spheres.

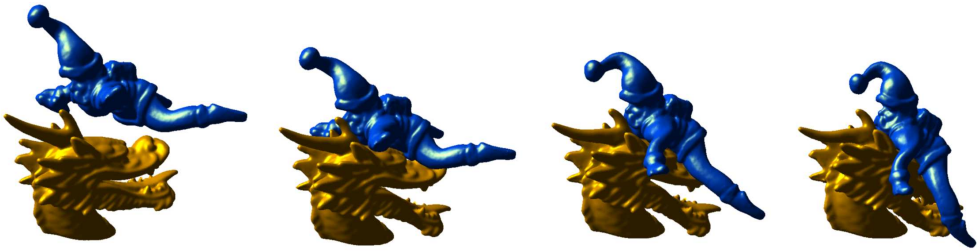


Figure 8: Santa Claus riding the dragon.

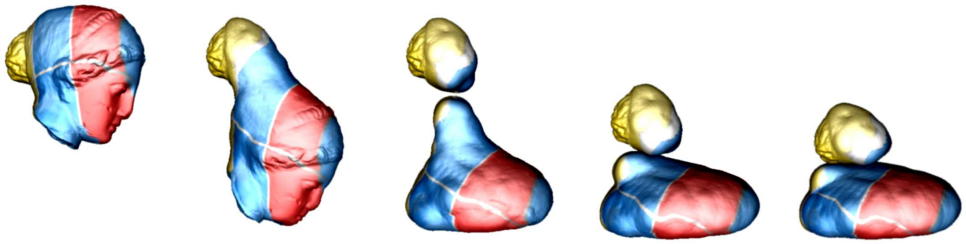


Figure 9: Fracturing with subsequent contact handling of the plastic Igea model.