Solid Simulation with Oriented Particles

Matthias Müller Nuttapong Chentanez
NVIDIA PhysX Research

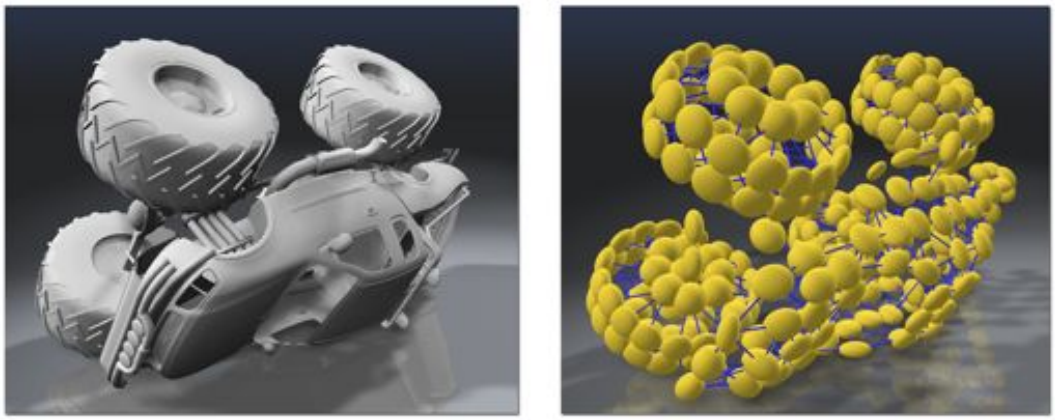

Figure 1: *Using particles with orientation enables us to simulate a complex model like this monster truck with plastically deforming body, free spinning wheels with soft tires, and high fidelity mesh skinning in real time all with a sparse physical representation.*

Abstract

We propose a new fast and robust method to simulate various types of solid including rigid, plastic and soft bodies as well as one, two and three dimensional structures such as ropes, cloth and volumetric objects. The underlying idea is to use oriented particles that store rotation and spin, along with the usual linear attributes, i.e. position and velocity. This additional information adds substantially to traditional particle methods. First, particles can be represented by anisotropic shapes such as ellipsoids, which approximate surfaces more accurately than spheres. Second, shape matching becomes robust for sparse structures such as chains of particles or even single particles because the undefined degrees of freedom are captured in the rotational states of the particles. Third, the full transformation stored in the particles, including translation and rotation, can be used for robust skinning of graphical meshes and for transforming plastic deformations back into the rest state.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically Based Modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation and Virtual Reality

Keywords: oriented particles, shape matching, position based dynamics

Links: DL PDF

1 Introduction

Physical simulation of solids has been investigated for more than two decades in computer graphics. In contrast to the computational sciences, computer graphics is more concerned with creating the overall look and feel of objects than the accurate reproduction of their small scale behavior. Also, artists require easy tuning of the physical attributes as well as full control of object behavior.

Lately, the trend in solid simulation in computer graphics has been to increase the accuracy of the mathematical models. This typically requires an increase in their complexity. Advantages of using representations based on continuum mechanics are that object behavior can be controlled using physical parameters such as Young's modulus, and that the discretization converges toward the continuous solution with increasing mesh resolution.

However, in computer games, where robustness and speed are often more essential than accuracy, simpler unconditionally stable geometric methods such as position based dynamics (PBD) [Müller et al. 2006] can be sufficient to create the desired physical effects. For these reasons we decided to come up with a method that is as simple and as fast as possible, yet able to create the desired visual fidelity required in many computer graphics applications. Our method is based on generalizations of PBD and the shape matching approach [Müller et al. 2005]. The novel idea of using oriented particles in connection with shape matching allows us to create complex dynamic objects with only a small number of simulation particles. This makes turning a visual mesh into a physical object a simple task which can be performed in just a few minutes.

In the first part of the paper we will present our research contributions which are

- An extension of PBD to handle orientation and angular velocity of particles
- A generalized formulation of the shape matching method incorporating particle orientations. This new formulation guarantees stability for arbitrary numbers and arrangements of particles.

- To leverage the orientation information to (1) approximate shapes by ellipsoids instead of spheres for more accurate collision handling and (2) to skin a visual mesh to the simulation nodes.

In the second part we will describe our content creation and simulation framework, which we developed based on our the new method, along with a variety of scenes demonstrating the versatility of the system.

2 Related Work

Formulating a unified solver for various solid material types has been an active topic in computer graphics in recent years. As a unified model, [Stam 2009] represents solids with simplices of various dimensions. [O'Brien et al. 1997], [Jansson and Vergeest 2003] and [Lenoir and Fonteneau 2004] describe ways to couple deformable and rigid bodies. [Sifakis et al. 2007] use soft and hard binding to transmit forces between different representations. Point-based approaches lend themselves especially well to unified solid simulation. One such method based on continuum mechanics was proposed by [Müller et al. 2004] and extended later by [Pauly et al. 2005; Gerszewski et al. 2009]. These approaches use moving least squares (MLS) to derive a deformation field from particle positions. MLS is only stable if local particle neighborhoods are in non-degenerate configurations. This problem was fixed by Martin et al. [2010] who introduced the concept of *elastons*. In addition to the deformation field, the elaston approach also stores derivatives. This way, particles in zero, one, two and three-dimensional configurations can be simulated robustly using generalized MLS (GMLS). Their approach was not designed for real-time use.

Our goal was to apply a similar idea to the geometric approach of shape matching [Müller et al. 2005]. In an analogous way, basic shape matching fails when particles are arranged in singular patterns. We fix this by storing orientation information on the particles. The link to the elaston idea and GMLS is that orientations can be viewed as derivative information of a normalized deformation field. An oriented particle is basically the simplest piece of information to fix the singularity problem. Due to its simplicity, the oriented particle approach is significantly faster then using elastons.

[Becker et al. 2009] use particles in connection with SPH-based forces to simulate various solids. They solve the singularity problem of degenerate cases by stabilizing the polar decomposition as in [Schmedding and Teschner 2008]. The missing rotation information is completed statically and not simulated as in our case. In the extreme case of a single particle, for instance, stabilized polar decomposition always returns the identity matrix for the particle's orientation, while in our case the angular quantities evolve physically in time. Another problem arises when exactly two Eigenvalues of the moment matrix are zero. This happens permanently for one-dimensional structures with stretched rest state. While one missing Eigenvector can be determined uniquely using the cross product of the other two, the directions of two missing Eigenvectors are not unique. To avoid jittering, they need to be chosen consistently over time which is only possible with a state variable. This is exactly what our orientation information on the particles provides.

The problems of simulating one, two and three-dimensional solids have been studied independently as well. To mention only a few papers, [Pai 2002] simulates elastic rods such as hair, wire and threads using the Cosserat theory by formulating a Boundary Value Problem (BVP) which does not have a guaranteed running time bound in general. [Bertails et al. 2006] and [Spillmann and Teschner 2007; Bertails 2009] reduced the complexity of the approach to be quadratic and linear in time, respectively. An approach based on discrete differential geometry was proposed by [Bergou et al. 2010] to allow for simulating thin viscous liquid threads like dripping honey.

The simulation of thin deformable bodies has been an active research field as well. [Provot 1995] were among the first to use mass-spring networks for simulating cloth. Later [Baraff and Witkin 1998] and [Bridson et al. 2003] proposed to use semi-implicit integration to increase stability and allow for larger time steps. [Volino et al. 2009] used non-linear springs to capture cloth behavior more faithfully. Various authors have addressed the problem of cloth stretchiness using the idea of strain limiting such as [Goldenthal et al. 2007]. Beside cloth simulation [Grinspun et al. 2003] and many others investigated the more general problem of thin shell simulation.

In computer graphics one of the most popular approach to simulating volumetric solids is the Finite Element Method (FEM) with linear tetrahedral elements. This technique was used in [O'Brien and Hodgins 1999] to simulate fracture and by [O'Brien et al. 2002; Bargteil et al. 2007; Wojtan and Turk 2008] to model plastic materials. The co-rotational formulation was introduced by [Müller and Gross 2004] to reduce visual artifacts in connection with linear elements and large deformations. To reduce the number of elements required for simulation Martin et al. [2008] generalized the traditional method to include general polyhedral elements.

The idea of attaching orientation information to particles and the term "oriented particle" was introduced by [Szeliski and Tonnesen 1992]. They used the additional information to define special energy potentials that let non-connected particles form surfaces rather then volumetric objects. Their approach is quite different from ours since they do not use a mesh, their particles are isotropic, they do not handle collision detection nor use the particles for skinning a visual mesh.

3 Generalized Shape Matching

As mentioned previously, the main idea behind our approach is to use particles that have both position and orientation. We will first show how the shape matching approach [Müller et al. 2005] benefits from this additional piece of information because shape matching lies at the core of our method.

Given n particles with rest position $\bar{\mathbf{x}}_i$, current position $\mathbf{x}_i$ and mass m_i we are looking for a global translation $\mathbf{t}$ and rotation $\mathbf{R}$ of the rest state that matches the current positions optimally in a least squares sense. For this the moment matrix $\mathbf{A}$ has to be computed as

$$\mathbf{A} = \sum_i m_i(\mathbf{x}_i - \mathbf{c})(\bar{\mathbf{x}}_i - \bar{\mathbf{c}})^T \in \mathbb{R}^{3\times 3}, \tag{1}$$

where the mass centers are defined as

$$\mathbf{c} = \sum_i m_i\mathbf{x}_i / \sum_i m_i \quad \text{and} \tag{2}$$

$$\bar{\mathbf{c}} = \sum_i m_i\bar{\mathbf{x}}_i / \sum_i m_i. \tag{3}$$

The polar decomposition $\mathbf{A} = \mathbf{RS}$ yields the least squares optimal rotation $\mathbf{R}$ of the original shape into the actual configuration while for the translation we have $\mathbf{t} = \mathbf{c} - \bar{\mathbf{c}}$. Using this transformation, the goal position of each particle becomes

$$\mathbf{g}_i = \mathbf{R}(\bar{\mathbf{x}}_i - \bar{\mathbf{c}}) + \mathbf{c}. \tag{4}$$

If the particles are close to co-planar or co-linear, $\mathbf{A}$ becomes ill-conditioned or even singular. In this case, the optimal rotation $\mathbf{R}$ is not well defined, causing the simulation to be unstable. This

problem can be fixed by using the orientation information of the particles.

Let us assume we have two groups of particles with their respective moment matrices $\mathbf{A}_1$ and $\mathbf{A}_2$. To compute the total moment matrix $\mathbf{A}$ of the union of the particles, the two matrices cannot simply be added because each one is computed w.r.t. its own center of mass. Fortunately there is an easy way to fix this problem. In their paper, [Rivers and James 2007] reformulated Eq. (1) for fast summation into the following form:

$$\mathbf{A} = \sum_i m_i \mathbf{x}_i \bar{\mathbf{x}}_i^T - M\mathbf{c}\bar{\mathbf{c}}^T, \tag{5}$$

where $M = \sum_i m_i$. Now let us assume we could define a moment matrix $\mathbf{A}_i$ for a single particle w.r.t. its center of mass $\mathbf{x}_i$. Using Eq. (5) we shift the center of mass of each $\mathbf{A}_i$ to the global one as

$$\mathbf{A} = \sum_i \left(\mathbf{A}_i + m_i \mathbf{x}_i \bar{\mathbf{x}}_i^T - m_i \mathbf{c}\bar{\mathbf{c}}^T \right) \tag{6}$$

and Eq. (1) generalizes to

$$\boxed{\mathbf{A} = \sum_i \left(\mathbf{A}_i + m_i \mathbf{x}_i \bar{\mathbf{x}}_i^T \right) - M\mathbf{c}\bar{\mathbf{c}}^T} \tag{7}$$

To compute the moment matrix of a particle with orthonormal orientation matrix $\mathbf{R}$ we integrate Eq. (1) over the particle's volume yielding

$$\begin{aligned} \mathbf{A}_{\text{sphere}} &= \int_{V_r} \rho(\mathbf{R}\mathbf{x})\mathbf{x}^T dV = \rho\mathbf{R} \int_{V_r} \mathbf{x}\mathbf{x}^T dV \\ &= \frac{4}{15}\pi r^5 \rho \mathbf{R} = \frac{4}{15}\pi r^5 \frac{m}{V_r}\mathbf{R} \\ &= \frac{1}{5} m r^2 \mathbf{R} \end{aligned} \tag{8}$$

for a sphere, where V_r is the volume of a sphere of radius r. For an ellipsoid with radii a, b and c we get

$$\mathbf{A}_{\text{ellipsoid}} = \frac{1}{5} m \begin{bmatrix} a^2 & 0 & 0 \\ 0 & b^2 & 0 \\ 0 & 0 & c^2 \end{bmatrix} \mathbf{R} \tag{9}$$

With this extension we always get a full rank moment matrix $\mathbf{A}$ even for a single particle! Eq. (7) can be viewed as a specialized version of GMLS [Martin et al. 2010] for the case that the deformation derivatives only contain rotations. However, in contrast to GMLS, Eq. (7) takes into consideration particles of finite size through the integral in Eq. (8) instead of single points only.

4 Generalized Position Based Dynamics

Before describing our overall simulation model, we explain how to evolve oriented particles in time. As an integration scheme we use PBD because shape matching is designed to work with this approach. To handle the rotational quantities we had to generalize PBD.

We will briefly recap basic position based dynamics and then explain our generalization. PBD evolves a set of particles with positions $\mathbf{x}$ and velocities $\mathbf{v}$ in three stages per time step. In the prediction stage a predicted position $\mathbf{x}_p$ is computed for each particle using explicit Euler integration so $\mathbf{x}_p \leftarrow \mathbf{x} + \mathbf{v}\Delta t$. In the second stage, the solver corrects the predicted positions (not the velocities, thus the name of the method) such that they satisfy a set of constraints by iterating through all constraints multiple times. Finally the particles' state variables are updated in the integration step as $\mathbf{v} \leftarrow (\mathbf{x}_p - \mathbf{x})/\Delta t$ and $\mathbf{x} \leftarrow \mathbf{x}_p$. Note that the solver's modifications on the predicted positions influence the velocities of the particles. Only this way does the resulting system become second order in time. PBD is straight forward to implement yet unconditionally stable because positional corrections never overshoot.

As in regular PBD, we define stiffness, friction and damping coefficients to be scalars $s \in [0 \dots 1]$. Since we use constant time steps, this definition is intuitive and works as expected. However, if the time steps are not constant, the coefficients should be defined as $s = s'\Delta t$ to reduce time step size dependence.

4.1 Integration

In addition to position $\mathbf{x}$ and velocity $\mathbf{v}$, our particles carry an orientation unit quaternion $\mathbf{q}$ and the angular velocity ω. The prediction step using explicit forward Euler integration then becomes

$$\mathbf{x}_p \leftarrow \mathbf{x} + \mathbf{v}\Delta t \tag{10}$$

$$\mathbf{q}_p \leftarrow \left[\frac{\omega}{|\omega|} \sin(\frac{|\omega|\Delta t}{2}), \cos(\frac{|\omega|\Delta t}{2}) \right] \mathbf{q}, \tag{11}$$

For stability reasons, $\mathbf{q}_p$ should directly be set to $\mathbf{q}$ if $|\omega| < \varepsilon$. After the solver has modified the predicted state $(\mathbf{x}_p, \mathbf{q}_p)$, as we shall describe in Section 5.1, the current state is updated using the integration scheme

$$\mathbf{v} \leftarrow (\mathbf{x}_p - \mathbf{x})/\Delta t \tag{12}$$

$$\mathbf{x} \leftarrow \mathbf{x}_p \tag{13}$$

$$\omega \leftarrow \text{axis}(\mathbf{q}_p\mathbf{q}^{-1}) \cdot \text{angle}(\mathbf{q}_p\mathbf{q}^{-1})/\Delta t \tag{14}$$

$$\mathbf{q} \leftarrow \mathbf{q}_p, \tag{15}$$

where axis() returns the normalized direction of a quaternion and angle() its angle. Again, for stability reasons, ω should be set to zero directly if $|angle(\mathbf{q}_p\mathbf{q}^{-1})| < \varepsilon$. There are two rotations, $\mathbf{r} = \mathbf{q}_p\mathbf{q}^{-1}$ and $-\mathbf{r}$ transforming $\mathbf{q}$ into $\mathbf{q}_p$. It is important to always choose the shorter one, i.e. if $r.w < 0$ use $-\mathbf{r}$. As in traditional PBD for translation, changing the rotational quantity $\mathbf{q}_p$ in the solver also affects its time derivate ω through the integration step creating the required second order effect. Note that this formulation is a simplification of the true rigid body dynamics since it omits precession. The error introduced is zero for spherical particles and grows with the aspect ratio of the particle shape. Also, correct precession of bodies composed of multiple particles emerges automatically so the error shows only for bodies composed of a small number of particles.

4.2 Friction

In PBD, friction is handled by scaling down the linear velocity by a constant factor s after the update step. If a particle has collided with a solid object, we modify both, linear and angular velocities as

$$\mathbf{v} \leftarrow \mathbf{v} + (\mathbf{v}_s - \mathbf{v})_{\perp \mathbf{n}} \cdot s_{lin} \tag{16}$$

$$\omega \leftarrow \omega + \frac{\mathbf{r}}{|\mathbf{r}|^2} \times (\mathbf{v}_s - \mathbf{v} - \omega \times \mathbf{r}) \cdot s_{rot}, \tag{17}$$

where $\mathbf{v}_s$ and $\mathbf{n}$ are the velocity and the normal of the solid at the impact point, $\mathbf{r} = r\mathbf{n}$ and r is the particle radius. The two scalars $s_{lin} \in [0 \dots 1]$ and $s_{rot} \in [0 \dots 1]$ control the amount of linear and angular friction.

The equations for handling friction in case of two particles colliding with each other look similar

$$\mathbf{v}_1 \leftarrow \mathbf{v}_1 + (\frac{\mathbf{v}_1+\mathbf{v}_2}{2} - \mathbf{v}_1)_{\perp \mathbf{n}} \cdot s_{lin} \tag{18}$$

$$\mathbf{v}_2 \leftarrow \mathbf{v}_2 + (\frac{\mathbf{v}_1+\mathbf{v}_2}{2} - \mathbf{v}_2)_{\perp \mathbf{n}} \cdot s_{lin} \tag{19}$$

and

$$\omega_1 \leftarrow \omega_1 + \frac{\mathbf{r}_1}{|\mathbf{r_1}|^2} \times (\mathbf{v}_{avg} - \mathbf{v}_1 - \omega_1 \times \mathbf{r}_1) \cdot s_{rot} \tag{20}$$

$$\omega_2 \leftarrow \omega_2 + \frac{\mathbf{r}_2}{|\mathbf{r_2}|^2} \times (\mathbf{v}_{avg} - \mathbf{v}_2 - \omega_2 \times \mathbf{r}_2) \cdot s_{rot} \tag{21}$$

where in this case $\mathbf{n} = (\mathbf{x}_2 - \mathbf{x}_1)/|\mathbf{x}_2 - \mathbf{x}_1|$, $\mathbf{r}_1 = r\mathbf{n}$, $\mathbf{r}_2 = -r\mathbf{n}$ and $\mathbf{v}_{avg} = (\mathbf{v}_1 + \omega_1 \times \mathbf{r}_1 + \mathbf{v}_2 + \omega_2 \times \mathbf{r}_2)/2$.

5 Simulation Model

We represent objects as a set of oriented particles and a set of edges connecting them. The resulting mesh does not need to have the topology of a triangle or tetrahedral mesh. It might look similar to a tetrahedral mesh locally in places where the model is volumetric. In other locations, where thin structures are present, it can take the form of a particle chain as in Fig. 3(d).

5.1 Implicit Shape Matching

This data structure is simulated by defining one shape matching group per particle. A group contains the corresponding particle and all the particles connected to it via a single edge. In sparse regions of the mesh, regular shape matching would become immediately unstable in this setting while in our case there are no limitations to the connectivity structure.

After the prediction step, the solver iterates multiple times through all shape match constraints in a Gauss-Seidel type fashion. For each constraint the goal positions are computed using Eq. (4). All the particles of the group are then moved towards their goal position by the same fraction $s_{\text{stiffness}}$ which mimics stiffness as in [Müller et al. 2006]. This stiffness can be specified per particle as Fig. 3(c) shows.

In terms of orientation, we only update the orientation of the center particle by replacing it with the optimal rotation provided by shape matching. Generalized shape matching, as we formulated it in Section 3, has a nice property: it only influences the orientation of the particle along the directions contained in the moment matrix $\mathbf{A}$. Let us have a look at two extreme cases. If there is only one particle in the group, generalized shape matching will return the orientation of that particle (see Eq. (7)) so the solver does not change it as expected. If the number of particles in the group and their positions are such that they robustly span a 3D space, the new orientation of the particle is dominated by the orientation of the entire group. All situations in between smoothly interpolate these two cases. In case of a chain of particles, for instance, the orientations of the particles along the direction of the chain are determined by shape matching, while they can freely rotate about the axis along the chain.

5.2 Stretching vs. Bending

Shape matching per node models both stretching and bending resistance at the same time. This is sufficient in many cases. If the artist wishes to specify them separately, we support regular PBD distance constraints on the edges as well. So in order to reduce bending resistance only, the shape match stiffness can be made small and the distance constraints activated. However, if the shape matching stiffness is set to zero, shape matching still has to be performed to update the orientations of the particles for collisions and skinning since the distance constraints only act on the positions.

5.3 Explicit Shape Matching

We also support additional explicit shape matching groups defined by the user which can cover arbitrary subsets of the particles in the mesh. The implicit shape matching as described above is not performed for particles in explicit groups. Their positions and orientations are controlled by the explicit group only. In contrast to implicit shape matching, all participating particles get the shape match rotation. This results in rigid components, as shown in several examples in Section 9. There is one exception: particles belonging to more than one explicit shape matching group are treated as non-oriented (i.e. the matrix $\mathbf{A}_i$ in Eq. (7) is set to zero for particle i). This allows us to model various joints as in the monster truck sample. Without this exception rotation information would propagate from one group to the other and prevent the wheels from rotating freely.

5.4 Plastic Deformation

We allow the explicit shape matching groups to deform plastically as well. Plastic deformation starts whenever one of the group particles is involved in a collision with relative velocity higher than a user defined threshold. When this happens, we simply deactivate the shape matching group for a fixed number of simulation frames (5 in our case) and let implicit shape matching take over. After the deformation phase, we reactivate the explicit group and absorb the deformation into the rest state.

Special care has to be taken when updating the rest state. It is crucial that explicit shape matching groups do not store individual rest positions of particles. The rest configuration of the model needs to be consistent over all shape matching groups otherwise ghost forces appear. Also, when distorting the rest poses, the original poses are still needed. We call these original poses the bind poses.

Let b_i, r_i and d_i be rigid transformations of the form $f(\mathbf{x}) = \mathbf{R}\mathbf{x} + \mathbf{t}$, where $\mathbf{R}$ is a rotation matrix. The bind pose transformations b_i map a zero centered and axis aligned particle to its original position and orientation defined by the user and the algorithm we will describe in Section 6.2. Before plastic deformation occurs, the rest poses r_i correspond to the bind poses so $\forall i : r_i = b_i$. The transformations d_i map the particles from the rest pose to the current pose and, thus, describe the deformation of the object. These are the transformations that shape matching gives us. Without plastic deformation, they can directly be used for mesh skinning (see Section 7). However, as soon as the rest poses do not correspond to the bind poses anymore, the transformations to be used for skinning are $d_i \circ r_i \circ b_i^{-1}$ (application from right to left). In contrast, the poses of the particles for collision detection are $d_i \circ r_i$.

After the deformation phase, particle i deviates from the rest pose r_i. This deviation is stored in the transformation d_i. We want to absorb this transformation by changing the rest pose r_i. However, since the object in world space is not aligned with the rest state during the simulation, the world space deformation d_i needs to be transformed back into the rest state space before being absorbed. Since the transformation from rest to world space varies over the object, we have to pick a specific one and choose the transformation s returned by the shape matching group at the first frame after reactivation. At this point in time we let $r_i \leftarrow s^{-1} \circ d_i \circ r_i$.

5.5 Torsion Resistance

To control torsion resistance we iterate through all the edges and update the rotations of the adjacent particles similar to friction handling as

$$\mathbf{q}_1 \leftarrow \text{slerp}\left(\mathbf{q}_1, \mathbf{q}_2, \frac{1}{2} s_{\text{torsion}}\right) \tag{22}$$

$$\mathbf{q}_2 \leftarrow \text{slerp}\left(\mathbf{q}_2, \mathbf{q}_1, \frac{1}{2} s_{\text{torsion}}\right). \tag{23}$$

The function $\text{slerp}(\mathbf{q}_1, \mathbf{q}_2, s)$ returns $\mathbf{q}_1$ if $s = 0$, $\mathbf{q}_2$ if $s = 1$ and the spherical interpolation of $\mathbf{q}_1$ and $\mathbf{q}_2$ for values in between.

6 Collision Handling

In traditional PBD particles are represented by spheres. Using these spheres as collision primitives results in bumpy collision representations of objects. This is problematic because bumpy surfaces introduce unnatural friction and other visual artifacts. The fact that our particles have orientation information lets us represent them by ellipsoids which can more accurately approximate flat surfaces, as demonstrated in [Yu and Turk 2010] in the case of fluid simulations (see Fig. 3(c)). Two questions arise. First, how to handle ellipsoids correctly in the collision detection step and, second, how to find their principal radii prior to the simulation.

6.1 Ellipsoid Collision

When using spheres, a collision with a plane occurs when the particle gets closer to the plane than its radius r. In this case, PBD shifts the particle up along the plane normal $\mathbf{n}$ such that it touches the plane as shown in Fig. 2(a). For an ellipsoid the situation gets a bit more complicated. A simple approximate way to resolve the collision is to use the ellipse's radius in the direction of $\mathbf{n}$ for collision handling. As Fig. 2(b) shows, the collision is only resolved correctly if $\mathbf{n}$ is aligned with the principal axes of the ellipsoid. In Appendix A.1 we show how to compute the correct distance d shown in Fig. 2(c). This is slightly more expensive. The situation is similar in the case of particle-particle collisions shown in Fig. 2(d)-(f) for which we derive the necessary equations in Appendix A.2.

6.2 Ellipsoid Representation of Objects

Let us assume the user has already placed the particles to simulate a given graphical mesh and specified radii for each of them. We use the same radius for all particles depending on the spacing of the particles to be able to use spatial hashing [Teschner et al. 2003] effectively for finding overlapping particles.

We now wish to automatically compute the principal axis directions and radii for each particle. Our algorithm to do this is similar to the method proposed in [Yu and Turk 2010] for creating smooth surfaces of particle fluids. For a certain particle we collect all graphical vertices within the particle's radius. We then compute the co-variance matrix of this vertex cloud relative to the particle's center and use polar decomposition to determine the orientation of the ellipsoid. In contrast to [Yu and Turk 2010] we do not use the eigenvalues of the co-variance matrix to determine the radii of the ellipsoid because the eigenvalues are related to the sum of squared distances and not the distances themselves. Instead, we use the extents of the oriented bounding box (OBB) of the mesh vertices as principal radii. For stability reasons we clamp the radii to the range $[\frac{1}{\sigma} r \dots r]$, where σ is a limit on the aspect radio. In our examples we use $\sigma = 2$.

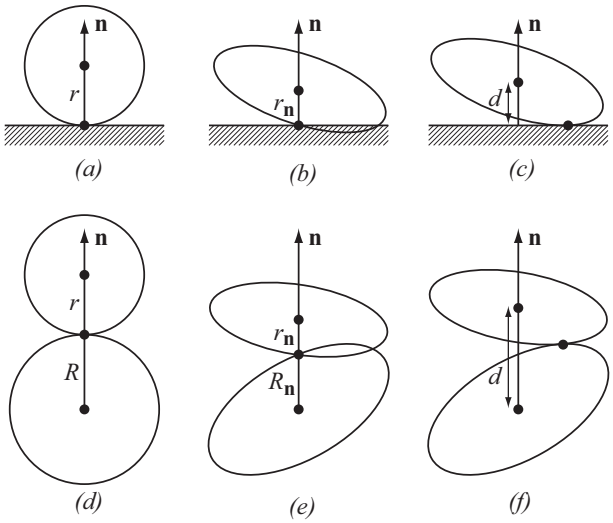

Figure 2: *Rows: particle-plane collision, particle-particle collision. Left to right: sphere collision, approximate and correct ellipsoid collision*

This approach can be extended in a useful way. Instead of adjusting only the orientations of particles, one can also move them to the center of mass of the surrounding vertices. By iterating these two steps multiple times, particles placed on the surface of the graphical mesh typically drift towards the center of a feature such as an arm. We used this property to create the skeleton shown in Fig. 3(c).

7 Visual Mesh Skinning

A common method to animate an arbitrary graphical mesh is to embed it in a tetrahedral simulation mesh as in [Müller and Gross 2004]. While the tetrahedral mesh deforms, each graphical vertex is transformed along with it, using barycentric weighting of the vertex positions of the closest tetrahedron. The method is only robust if each graphical vertex is surrounded by a tetrahedron because only in that case all the barycentric coordinates are positive, as noticed by [Twigg and Kacic-Alesic 2010]. Many tetrahedra are typically required to get an accurate approximation of the graphical mesh's surface, particularly if the graphical mesh has a complex structure with many branches that need to be resolved in order to move independently.

The fact that, in our case, each particle stores a full transformation composed of translation and rotation from the rest state to the current state lets us use a more robust approach for binding the graphical to the simulation particles. For each graphical vertex we precompute links and weights to nearby particles (at most 4 in our case) and use linear blend skinning to compute its transformation. This yields plausible deformations even if the visual geometry is far away from the simulated particles and for very sparse simulation meshes. Skinning remains well defined for the case even of a single simulation particle. We found that simple linear blend skinning was sufficient. However, to further improve the results, dual quaternion blending [Kavan et al. 2008] could be used as well.

8 Simulation Framework

We built a design and simulation framework to create various demo scenes. The design tool is used to create a physical representation of an object. After loading a graphical mesh, the user places