3rd Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2006)
C. Mendoza, I. Navazo (Editors)

Position Based Dynamics

Matthias Müller Bruno Heidelberger Marcus Hennix John Ratcliff

AGEIA

Abstract

The most popular approaches for the simulation of dynamic systems in computer graphics are force based. Internal and external forces are accumulated from which accelerations are computed based on Newton's second law of motion. A time integration method is then used to update the velocities and finally the positions of the object. A few simulation methods (most rigid body simulators) use impulse based dynamics and directly manipulate velocities. In this paper we present an approach which omits the velocity layer as well and immediately works on the positions. The main advantage of a position based approach is its controllability. Overshooting problems of explicit integration schemes in force based systems can be avoided. In addition, collision constraints can be handled easily and penetrations can be resolved completely by projecting points to valid locations. We have used the approach to build a real time cloth simulator which is part of a physics software library for games. This application demonstrates the strengths and benefits of the method.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object ModelingPhysically Based Modeling; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and RealismAnimation and Virtual Reality

1. Introduction

Research in the field of physically based animation in computer graphics is concerned with finding new methods for the simulation of physical phenomena such as the dynamics of rigid bodies, deformable objects or fluid flow. In contrast to computational sciences where the main focus is on accuracy, the main issues here are stability, robustness and speed while the results should remain visually plausible. Therefore, existing methods from computational sciences can not be adopted one to one. In fact, the main justification for doing research on physically based simulation in computer graphics is to come up with specialized methods, tailored to the particular needs in the field. The method we present falls into this category.

The traditional approach to simulating dynamic objects has been to work with forces. At the beginning of each time step, internal and external forces are accumulated. Examples of internal forces are elastic forces in deformable objects or viscosity and pressure forces in fluids. Gravity and collision forces are examples of external forces. Newton's second law of motion relates forces to accelerations via the mass. So using the density or lumped masses of vertices, the forces are transformed into accelerations. Any time integration scheme can then be used to first compute the velocities from the accelerations and then the positions from the velocities. Some approaches use impulses instead of forces to control the animation. Because impulses directly change velocities, one level of integration can be skipped.

In computer graphics and especially in computer games it is often desirable to have direct control over positions of objects or vertices of a mesh. The user might want to attach a vertex to a kinematic object or make sure the vertex always stays outside a colliding object. The method we propose here works directly on positions which makes such manipulations easy. In addition, with the position based approach it is possible to control the integration directly thereby avoiding overshooting and energy gain problems in connection with explicit integration. So the main features and advantages of position based dynamics are

- Position based simulation gives control over explicit integration and removes the typical instability problems.

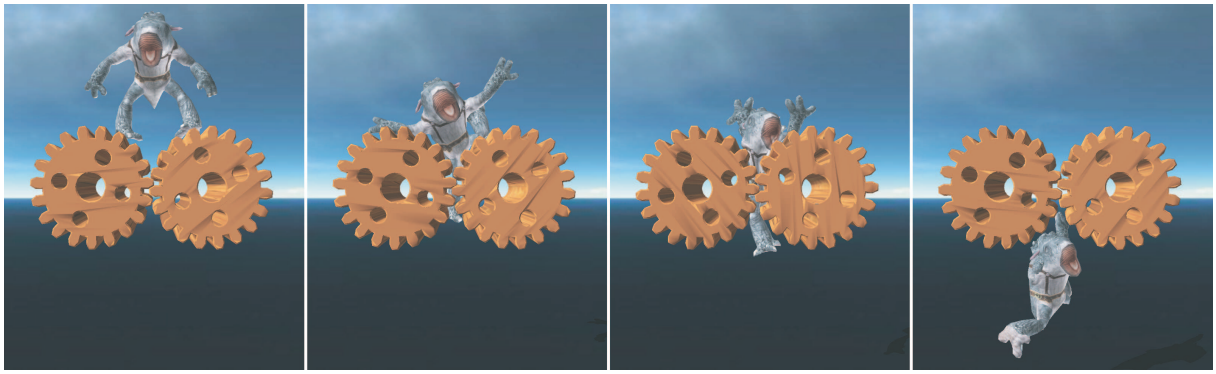

Figure 1: *A known deformation benchmark test, applied here to a cloth character under pressure.*

- Positions of vertices and parts of objects can directly be manipulated during the simulation.
- The formulation we propose allows the handling of general constraints in the position based setting.
- The explicit position based solver is easy to understand and implement.

2. Related Work

The recent state of the art report [NMK*05] gives a good overview of the methods used in computer graphics to simulate deformable objects, e.g. mass-spring systems, the finite element method or finite difference approaches. Apart from the citation of [MHTG05], position based dynamics does not appear in this survey. However, parts of the position based approach have appeared in various papers without naming it explicitly and without defining a complete framework.

Jakobsen [Jak01] built his *Fysix* engine on a position based approach. His central idea was to use a Verlet integrator and manipulate positions directly. Because velocities are implicitly stored by current and the previous positions, the velocities are implicitly updated by the position manipulation. While he focused mainly on distance constraints, he only gave vague hints on how more general constraints could be handled. In this paper we present a fully general approach which handles general constraints. We also focus on the important issue of conservation of linear and angular momenta by position projection. We work with explicit velocities instead of storing previous positions which makes damping and friction simulation much easier.

Desbrun [DSB99] and Provot [Pro95] use constraint projection in mass spring systems to prevent springs from overstretching. In contrast to a full position based approach, projection is only used as a polishing process for those springs that are stretched too much and not as the basic simulation method.

Bridson *et al.* use a traditional force based approach for cloth simulation [BFA02] and combine it with a geometric collision resolving algorithm based on positions to make sure that the collision resolving impulses are kept within stable bounds. The same holds for the kinematical collision correction step proposed by Volino *et al.* [VCMT95].

A position based approach has been used by Clavet *et al.* [CBP05] to simulate viscoelastic fluids. Their approach is not fully position based because the time step appears in various places of their position projections. Thus, the integration is only conditionally stable as regular explicit integration.

Müller *et al.* [MHTG05] simulate deformable objects by moving points towards certain goal positions which are found by matching the rest state to the current state of the object. Their integration method is the closest to the one we propose here. They only treat one specialized global constraint and, therefore, do not need a position solver.

Fedor [Fed05] uses Jakobsen's approach to simulate characters in games. His method is tuned to the particular problem of simulating human characters. He uses several skeletal representations and keeps them in sync via projections.

Faure [Fau98] uses a Verlet integration scheme by modifying the positions rather than the velocities. New positions are computed by linearizing the constraints while we work with the non linear constraint functions directly.

We define general constraints via a constraint function as [BW98] and [THMG04]. Instead of computing forces as the derivative of a constraint function energy, we directly solve for the equilibrium configuration and project positions. With our method we derive a bending term for cloth which is similar to the one proposed in [GHDS03] and [BMF03] but adopted to the point based approach.

In Section 4 we use the position based dynamics approach for the simulation of cloth. Cloth simulation has been an active research field in computer graphics in recent years. Instead of citing the key papers of the field individually we refer the reader to [NMK*05] for a comprehensive survey.

3. Position Based Simulation

In this section we will formulate the general position based approach. With cloth simulation, we will give a particular application of the method in the subsequent and in the results section. We consider a three dimensional world. However, the approach works equally well in two dimensions.

3.1. Algorithm Overview

We represent a dynamic object by a set of N vertices and M constraints. A vertex $i \in [1,\dots,N]$ has a mass m_i, a position $\mathbf{x}_i$ and a velocity $\mathbf{v}_i$.

A constraint $j \in [1,\dots,M]$ consists of

- a cardinality n_j,
- a function $C_j : \mathbb{R}^{3n_j} \rightarrow \mathbb{R}$,
- a set of indices $\{i_1,\dots i_{n_j}\}$, $i_k \in [1,\dots N]$,
- a stiffness parameter $k_j \in [0\dots 1]$ and
- a type of either *equality* or *inequality*.

Constraint j with type *equality* is satisfied if $C_j(\mathbf{x}_{i_1},\dots,\mathbf{x}_{i_{n_j}}) = 0$. If its type is *inequality* then it is satisfied if $C_j(\mathbf{x}_{i_1},\dots,\mathbf{x}_{i_{n_j}}) \geq 0$. The stiffness parameter k_j defines the strength of the constraint in a range from zero to one.

Based on this data and a time step Δt, the dynamic object is simulated as follows:

(1) **forall** vertices i
(2) initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
(3) **endfor**
(4) **loop**
(5) **forall** vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
(6) dampVelocities($\mathbf{v}_1,\dots,\mathbf{v}_N$)
(7) **forall** vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
(8) **forall** vertices i **do** generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
(9) **loop** solverIterations **times**
(10) projectConstraints($C_1,\dots,C_{M+M_{coll}},\mathbf{p}_1,\dots,\mathbf{p}_N$)
(11) **endloop**
(12) **forall** vertices i
(13) $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
(14) $\mathbf{x}_i \leftarrow \mathbf{p}_i$
(15) **endfor**
(16) velocityUpdate($\mathbf{v}_1,\dots,\mathbf{v}_N$)
(17) **endloop**

Lines (1)-(3) just initialize the state variables. The core idea of position based dynamics is shown in lines (7), (9)-(11) and (13)-(14). In line (7), estimates $\mathbf{p}_i$ for new locations of the vertices are computed using an explicit Euler integration step. The iterative solver (9)-(11) manipulates these position estimates such that they satisfy the constraints. It does this by repeatedly project each constraint in a Gauss-Seidel type fashion (see Section 3.2). In steps (13) and (14), the positions of the vertices are moved to the optimized estimates and the velocities are updated accordingly. This is in exact correspondence with a *Verlet* integration step and a modification of the current position [Jak01], because the Verlet method stores the velocity implicitly as the difference between the current and the last position. However, working with velocities allows for a more intuitive way of manipulating them.

The velocities are manipulated in line (5), (6) and (16). Line (5) allows to hook up external forces to the system if some of the forces cannot be converted to positional constraints. We only use it to add gravity to the system in which case the line becomes $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t \mathbf{g}$, where $\mathbf{g}$ is the gravitational acceleration. In line (6), the velocities can be damped if this is necessary. In Section 3.5 we show how to add global damping without influencing the rigid body modes of the object. Finally, in line (16), the velocities of colliding vertices are modified according to friction and restitution coefficients.

The given constraints $C_1,\dots,C_M$ are fixed throughout the simulation. In addition to these constraints, line (8) generates the M_{coll} collision constraints which change from time step to time step. The projection step in line (10) considers both, the fixed and the collision constraints.

The scheme is unconditionally stable. This is because the integration steps (13) and (14) do not extrapolate blindly into the future as traditional explicit schemes do but move the vertices to a physically valid configuration $\mathbf{p}_i$ computed by the constraint solver. The only possible source for instabilities is the solver itself which uses the Newton-Raphson method to solve for valid positions (see Section 3.3). However, its stability does not depend on the time step size but on the shape of the constraint functions.

The integration does not fall clearly into the category of implicit or explicit schemes. If only one solver iteration is performed per time step, it looks more like an explicit scheme. By increasing the number of iterations, however, a constrained system can be made arbitrarily stiff and the algorithm behaves more like an implicit scheme. Increasing the number of iterations shifts the bottleneck from collision detection to the solver.

3.2. The Solver

The input to the solver are the $M + M_{coll}$ constraints and the estimates $\mathbf{p}_1,\dots,\mathbf{p}_N$ for the new locations of the points. The solver tries to modify the estimates such that they satisfy all the constraints. The resulting system of equations is non-linear. Even a simple distance constraint $C(\mathbf{p}_1,\mathbf{p}_2) = |\mathbf{p}_1 - \mathbf{p}_2| - d$ yields a non-linear equation. In addition, the constraints of type *inequality* yield inequalities. To solve such a general set of equations and inequalities, we use a Gauss-Seidel-type iteration. The original Gauss-Seidel algorithm (GS) can only handle linear system. The part we borrow from GS is the idea of solving each constraint independently one after the other. However, in contrast to GS,

solving a constraint is a non linear operation. We repeatedly iterate through all the constraints and project the particles to valid locations with respect to the given constraint alone. In contrast to a Jacobi-type iteration, modifications to point locations immediately get visible to the process. This speeds up convergence significantly because pressure waves can propagate through the material in a single solver step, an effect which is dependent on the order in which constraints are solved. In over-constrained situations, the process can lead to oscillations if the order is not kept constant.

3.3. Constraint Projection

Projecting a set of points according to a constraint means moving the points such that they satisfy the constraint. The most important issue in connection with moving points directly inside a simulation loop is the conservation of linear and angular momentum. Let $\Delta\mathbf{p}_i$ be the displacement of vertex i by the projection. Linear momentum is conserved if

$$\sum_i m_i \Delta\mathbf{p}_i = \mathbf{0}, \tag{1}$$

which amounts to conserving the center of mass. Angular momentum is conserved if

$$\sum_i \mathbf{r}_i \times m_i \Delta\mathbf{p}_i = \mathbf{0}, \tag{2}$$

where the $\mathbf{r}_i$ are the distances of the $\mathbf{p}_i$ to an arbitrary common rotation center. If a projection violates one of these constraints, it introduces so called *ghost forces* which act like external forces dragging and rotation the object. However, only internal constraints need to conserve the momenta. Collision or attachment constraints are allowed to have global effects on the object.

The method we propose for constraint projection conserves both momenta for internal constraints. Again, the point based approach is more direct in that we can directly use the constraint function while force based methods derive forces via an energy term (see [BW98, THMG04]). Let us look at a constraint with cardinality n on the points $\mathbf{p}_1,\ldots,\mathbf{p}_n$ with constraint function C and stiffness k. We let $\mathbf{p}$ be the concatenation $[\mathbf{p}_1^T,\ldots,\mathbf{p}_n^T]^T$. For internal constraints, C is independent of rigid body modes, i.e. translation and rotation. This means that rotating or translating the points does not change the value of the constraint function. Therefore, the gradient $\nabla_{\mathbf{p}}C$ is perpendicular to rigid body modes because it is the direction of maximal change. If the correction $\Delta\mathbf{p}$ is chosen to be along $\nabla C_{\mathbf{p}}$ both momenta are automatically conserved if all masses are equal (we handle different masses later). Given $\mathbf{p}$ we want to find a correction $\Delta\mathbf{p}$ such that $C(\mathbf{p}+\Delta\mathbf{p})=0$. This equation can be approximated by

$$C(\mathbf{p}+\Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla_{\mathbf{p}}C(\mathbf{p})\cdot\Delta\mathbf{p} = 0. \tag{3}$$

Restricting $\Delta\mathbf{p}$ to be in the direction of $\nabla_{\mathbf{p}}C$ means choosing a scalar λ such that

$$\Delta\mathbf{p} = \lambda\nabla_{\mathbf{p}}C(\mathbf{p}). \tag{4}$$

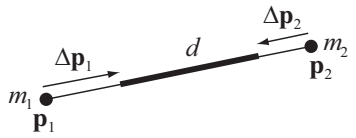

Figure 2: *Projection of the constraint $C(\mathbf{p}_1,\mathbf{p}_2) = |\mathbf{p}_1 - \mathbf{p}_2| - d$. The corrections $\Delta\mathbf{p}_i$ are weighted according to the inverse masses $w_i = 1/m_i$.*

Substituting Eq. (4) into Eq. (3), solving for λ and substituting it back into Eq. (4) yields the final formula for $\Delta\mathbf{p}$

$$\Delta\mathbf{p} = -\frac{C(\mathbf{p})}{|\nabla_{\mathbf{p}}C(\mathbf{p})|^2}\nabla_{\mathbf{p}}C(\mathbf{p}) \tag{5}$$

which is a regular Newton-Raphson step for the iterative solution of the non-linear equation given by a single constraint. For the correction of an individual point $\mathbf{p}_i$ we have

$$\Delta\mathbf{p}_i = -s\,\nabla_{\mathbf{p}_i}C(\mathbf{p}_1,\ldots,\mathbf{p}_n), \tag{6}$$

where the scaling factor

$$s = \frac{C(\mathbf{p}_1,\ldots,\mathbf{p}_n)}{\sum_j |\nabla_{\mathbf{p}_j}C(\mathbf{p}_1,\ldots,\mathbf{p}_n)|^2} \tag{7}$$

is the same for all points. If the points have individual masses, we weight the corrections $\Delta\mathbf{p}_i$ by the inverse masses $w_i = 1/m_i$. In this case a point with infinite mass, i.e. $w_i = 0$, does not move for example as expected. Now Eq. (4) is replaced by

$\Delta\mathbf{p}_i = \lambda w_i \nabla_{\mathbf{p}_i}C(\mathbf{p})$ yielding

$$s = \frac{C(\mathbf{p}_1,\ldots,\mathbf{p}_n)}{\sum_j w_j|\nabla_{\mathbf{p}_j}C(\mathbf{p}_1,\ldots,\mathbf{p}_n)|^2} \tag{8}$$

for the scaling factor and for the final correction

$$\Delta\mathbf{p}_i = -s\,w_i\nabla_{\mathbf{p}_i}C(\mathbf{p}_1,\ldots,\mathbf{p}_n). \tag{9}$$

To give an example, let us consider the distance constraint function $C(\mathbf{p}_1,\mathbf{p}_2) = |\mathbf{p}_1 - \mathbf{p}_2| - d$. The derivative with respect to the points are $\nabla_{\mathbf{p}_1}C(\mathbf{p}_1,\mathbf{p}_2) = \mathbf{n}$ and $\nabla_{\mathbf{p}_2}C(\mathbf{p}_1,\mathbf{p}_2) = -\mathbf{n}$ with $\mathbf{n} = \frac{\mathbf{p}_1-\mathbf{p}_2}{|\mathbf{p}_1-\mathbf{p}_2|}$. The scaling factor s is, thus, $s = \frac{|\mathbf{p}_1-\mathbf{p}_2|-d}{w_1+w_2}$ and the final corrections

$$\Delta\mathbf{p}_1 = -\frac{w_1}{w_1+w_2}(|\mathbf{p}_1-\mathbf{p}_2|-d)\frac{\mathbf{p}_1-\mathbf{p}_2}{|\mathbf{p}_1-\mathbf{p}_2|} \tag{10}$$

$$\Delta\mathbf{p}_2 = +\frac{w_2}{w_1+w_2}(|\mathbf{p}_1-\mathbf{p}_2|-d)\frac{\mathbf{p}_1-\mathbf{p}_2}{|\mathbf{p}_1-\mathbf{p}_2|} \tag{11}$$

which are the formulas proposed in [Jak01] for the projection of distance constraints (see Figure 2). They pop up as a special case of the general constraint projection method.

We have not considered the type and the stiffness k of the constraint so far. Type handling is straight forward. If the type is *equality* we always perform a projection. If the type is *inequality*, the projection is only performed if $C(\mathbf{p}_1,\ldots,\mathbf{p}_n) < 0$. There are several ways of incorporating the stiffness parameter. The simplest variant is to multiply the corrections $\Delta\mathbf{p}$ by $k \in [0\ldots1]$. However, for multiple iteration loops of the solver, the effect of k is non-linear. The remaining error for a single distance constraint after n_s solver iterations is $\Delta\mathbf{p}(1-k)^{n_s}$. To get a linear relationship we multiply the corrections not by k directly but by $k' = 1-(1-k)^{1/n_s}$. With this transformation the error becomes $\Delta\mathbf{p}(1-k')^{n_s} = \Delta\mathbf{p}(1-k)$ and, thus, becomes linearly dependent on k and independent of n_s as desired. However, the resulting material stiffness is still dependent on the time step of the simulation. Real time environments typically use fixed time steps in which case this dependency is not problematic.

3.4. Collision Detection and Response

One advantage of the position based approach is how simply collision response can be realized. In line (8) of the simulation algorithm the M_{coll} collision constraints are generated. While the first M constraints given by the object representation are fixed throughout the simulation, the additional M_{coll} constraints are generated from scratch at each time step. The number of collision constraints M_{coll} varies and depends on the number of colliding vertices. Both, continuous and static collisions can be handled. For continuous collision handling, we test for each vertex i the ray $\mathbf{x}_i \rightarrow \mathbf{p}_i$. If this ray enters an object, we compute the entry point $\mathbf{q}_c$ and the surface normal $\mathbf{n}_c$ at this position. An *inequality* constraint with constraint function $C(\mathbf{p}) = (\mathbf{p}-\mathbf{q}_c)\cdot\mathbf{n}_c$ and stiffness $k=1$ is added to the list of constraints. If the ray $\mathbf{x}_i \rightarrow \mathbf{p}_i$ lies completely inside an object, continuous collision detection has failed at some point. In this case we fall back to static collision handling. We compute the surface point $\mathbf{q}_s$ which is closest to $\mathbf{p}_i$ and the surface normal $\mathbf{n}_s$ at this position. An *inequality* constraint with constraint function $C(\mathbf{p}) = (\mathbf{p}-\mathbf{q}_s)\cdot\mathbf{n}_s$ and stiffness $k=1$ is added to the list of constraints. Collision constraint generation is done outside of the solver loop. This makes the simulation much faster. There are certain scenarios, however, where collisions can be missed if the solver works with a fixed collision constraint set. Fortunately, according to our experience, the artifacts are negligible.

Friction and restitution can be handled by manipulating the velocities of colliding vertices in step (16) of the algorithm. The velocity of each vertex for which a collision constraint has been generated is dampened perpendicular to the collision normal and reflected in the direction of the collision normal.

The collision handling discussed above is only correct for collisions with static objects because no impulse is transferred to the collision partners. Correct response for two dynamic colliding objects can be achieved by simulating both objects with our simulator, i.e. the N vertices and M constraints which are the input to our algorithm simply represent two or more independent objects. Then, if a point $\mathbf{q}$ of one objects moves through a triangle $\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3$ of another object, we insert an *inequality* constraint with constraint function $C(\mathbf{q},\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3) = \pm(\mathbf{q}-\mathbf{p}_1)\cdot[(\mathbf{p}_2-\mathbf{p}_1)\times(\mathbf{p}_3-\mathbf{p}_1)]$ which keeps the point $\mathbf{q}$ on the correct side of the triangle. Since this constraint function is independent of rigid body modes, it will correctly conserve linear and angular momentum. Collision detection gets slightly more involved because the four vertices are represented by rays $\mathbf{x}_i \rightarrow \mathbf{p}_i$. Therefore the collision of a moving point against a moving triangle needs to be detected (see section about cloth self collision).

3.5. Damping

In line (6) of the simulation algorithm the velocities are dampened before they are used for the prediction of the new positions. Any form of damping can be used and many methods for damping have been proposed in the literature (see [NMK*05]). Here we propose a new method with some interesting properties:

(1) $\mathbf{x}_{cm} = (\sum_i \mathbf{x}_i m_i)/(\sum_i m_i)$
(2) $\mathbf{v}_{cm} = (\sum_i \mathbf{v}_i m_i)/(\sum_i m_i)$
(3) $\mathbf{L} = \sum_i \mathbf{r}_i \times (m_i \mathbf{v}_i)$
(4) $\mathbf{I} = \sum_i \tilde{\mathbf{r}}_i \tilde{\mathbf{r}}_i^T m_i$
(5) $\omega = \mathbf{I}^{-1}\mathbf{L}$
(6) **forall** vertices i
(7) $\quad \Delta\mathbf{v}_i = \mathbf{v}_{cm} + \omega \times \mathbf{r}_i - \mathbf{v}_i$
(8) $\quad \mathbf{v}_i \leftarrow \mathbf{v}_i + k_{\text{damping}}\Delta\mathbf{v}_i$
(9) **endfor**

Here $\mathbf{r}_i = \mathbf{x}_i - \mathbf{x}_{cm}$, $\tilde{\mathbf{r}}_i$ is the 3 by 3 matrix with the property $\tilde{\mathbf{r}}_i\mathbf{v} = \mathbf{r}_i \times \mathbf{v}$, and $k_{\text{damping}} \in [0\ldots1]$ is the damping coefficient. In lines (1)-(5) we compute the global linear velocity $\mathbf{x}_{cm}$ and angular velocity ω of the system. Lines (6)-(9) then only damp the individual deviations $\Delta\mathbf{v}_i$ of the velocities $\mathbf{v}_i$ from the global motion $\mathbf{v}_{cm} + \omega \times \mathbf{r}_i$. Thus, in the extreme case $k_{\text{damping}} = 1$, only the global motion survives and the set of vertices behaves like a rigid body. For arbitrary values of k_{damping}, the velocities are globally dampened but without influencing the global motion of the vertices.

3.6. Attachments

With the position based approach, attaching vertices to static or kinematic objects is quite simple. The position of the vertex is simply set to the static target position or updated at every time step to coincide with the position of the kinematic object. To make sure other constraints containing this vertex do not move it, its inverse mass w_i is set to zero.

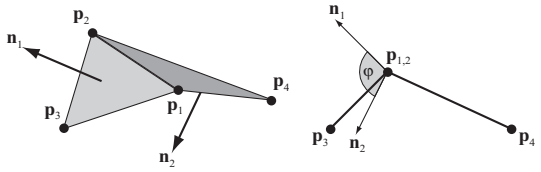

Figure 4: *For bending resistance, the constraint function* $C(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\mathbf{p}_4) = \arccos(\mathbf{n}_1 \cdot \mathbf{n}_2) - \varphi_0$ *is used. The actual dihedral angle* φ *is measure as the angle between the normals of the two triangles.*

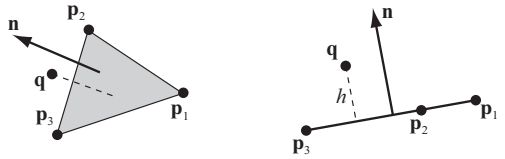

Figure 5: *Constraint function* $C(\mathbf{q},\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3) = (\mathbf{q}-\mathbf{p}_1)\cdot\mathbf{n} - h$ *makes sure that* $\mathbf{q}$ *stays above the triangle* $\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3$ *by the cloth thickness h.*

4. Cloth Simulation

We have used the point based dynamics framework to implement a real time cloth simulator for games. In this section we will discuss cloth specific issues thereby giving concrete examples of the general concepts introduced in the previous section.

4.1. Representation of Cloth

Our cloth simulator accepts as input arbitrary triangle meshes. The only restriction we impose on the input mesh is that it represents a manifold, i.e. each edge is shared by at most two triangles. Each node of the mesh becomes a simulated vertex. The user provides a density ρ given in mass per area $[kg/m^2]$. The mass of a vertex is set to the sum of one third of the mass of each adjacent triangle. For each edge, we generate a stretching constraint with constraint function

$$C_{stretch}(\mathbf{p}_1,\mathbf{p}_2) = |\mathbf{p}_1 - \mathbf{p}_2| - l_0,$$

stiffness $k_{stretch}$ and type *equality*. The scalar ł$_0$ is the initial length of the edge and $k_{stretch}$ is a global parameter provided by the user. It defines the stretching stiffness of the cloth. For each pair of adjacent triangles $(\mathbf{p}_1,\mathbf{p}_3,\mathbf{p}_2)$ and $(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_4)$ we generate a bending constraint with constraint function

$$C_{bend}(\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3,\mathbf{p}_4) = \operatorname{acos}\left(\frac{(\mathbf{p}_2-\mathbf{p}_1)\times(\mathbf{p}_3-\mathbf{p}_1)}{|(\mathbf{p}_2-\mathbf{p}_1)\times(\mathbf{p}_3-\mathbf{p}_1)|}\cdot\frac{(\mathbf{p}_2-\mathbf{p}_1)\times(\mathbf{p}_4-\mathbf{p}_1)}{|(\mathbf{p}_2-\mathbf{p}_1)\times(\mathbf{p}_4-\mathbf{p}_1)|}\right) - \varphi_0,$$

stiffness k_{bend} and type *equality*. The scalar φ_0 is the initial dihedral angle between the two triangles and k_{bend} is a global user parameter defining the bending stiffness of the cloth (see Figure 4). The advantage of this bending term over adding a distance constraint between points $\mathbf{p}_3$ and $\mathbf{p}_4$ or over the bending term proposed by [GHDS03] is that it is *independent of stretching*. This is because the term is independent of edge lengths. This way, the user can specify cloth with low stretching stiffness but high bending resistance for instance (see Figure 3).

Eqns. (10) and (11) define the projection for the stretching constraints. In the appendix A we derive the formulas to project the bending constraints.

4.2. Collision with Rigid Bodies

For collision handling with rigid bodies we proceed as described in Section 3.4. To get two-way interactions, we apply an impulse $m_i\Delta\mathbf{p}_i/\Delta t$ to the rigid body at the contact point, each time vertex i is projected due to collision with that body. Testing only cloth vertices for collisions is not enough because small rigid bodies can fall through large cloth triangles. Therefore, collisions of the convex corners of rigid bodies against the cloth triangles are also tested.

4.3. Self Collision

Assuming that the triangles all have about the same size, we use spatial hashing to find vertex triangle collisions [THM*03]. If a vertex $\mathbf{q}$ moves through a triangle $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$, we use the constraint function

$$C(\mathbf{q},\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3) = (\mathbf{q}-\mathbf{p}_1)\cdot\frac{(\mathbf{p}_2-\mathbf{p}_1)\times(\mathbf{p}_3-\mathbf{p}_1)}{|(\mathbf{p}_2-\mathbf{p}_1)\times(\mathbf{p}_3-\mathbf{p}_1)|} - h, \tag{12}$$

where h is the cloth thickness (see Figure 5). If the vertex enters from below with respect to the triangle normal, the constraint function has to be

$$C(\mathbf{q},\mathbf{p}_1,\mathbf{p}_2,\mathbf{p}_3) = (\mathbf{q}-\mathbf{p}_1)\cdot\frac{(\mathbf{p}_3-\mathbf{p}_1)\times(\mathbf{p}_2-\mathbf{p}_1)}{|(\mathbf{p}_3-\mathbf{p}_1)\times(\mathbf{p}_2-\mathbf{p}_1)|} - h \tag{13}$$

to keep the vertex on the original side. Projecting these constraints conserves linear and angular momentum which is essential for cloth self collision since it is an internal process. Figure 6 shows a rest state of a piece of cloth with self collisions. Testing continuous collisions is insufficient if cloth gets into a tangled state, so methods like the ones proposed by [BWK03] have to be applied.

4.4. Cloth Balloons

For closed triangle meshes, overpressure inside the mesh can easily be modeled (see Figure 7). We add an *equality* constraint concerning all N vertices of the mesh with constraint function

$$C(\mathbf{p}_1,\ldots,\mathbf{p}_N) = \left(\sum_{i=1}^{n_{\text{triangles}}} (\mathbf{p}_{t_1^i}\times\mathbf{p}_{t_2^i})\cdot\mathbf{p}_{t_3^i}\right) - k_{\text{pressure}}V_0 \tag{14}$$

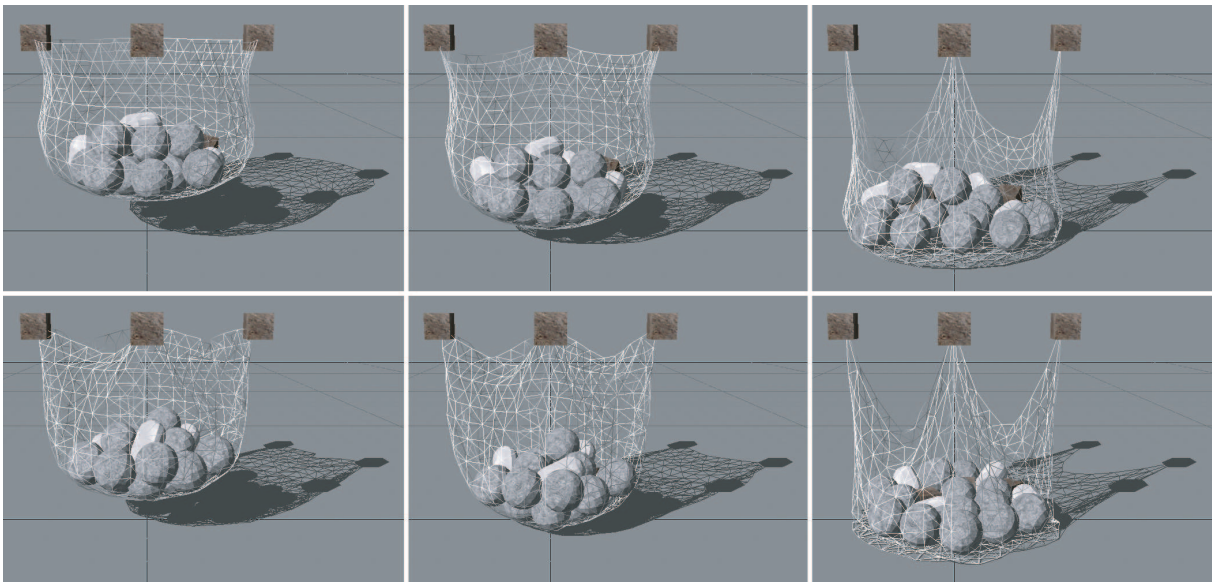

Figure 3: *With the bending term we propose, bending and stretching are independent parameters. The top row shows* $(k_{stretching}, k_{bending}) = (1,1)$, $(\frac{1}{2},1)$ *and* $(\frac{1}{100},1)$. *The bottom row shows* $(k_{stretching}, k_{bending}) = (1,0)$, $(\frac{1}{2},0)$ *and* $(\frac{1}{100},0)$.

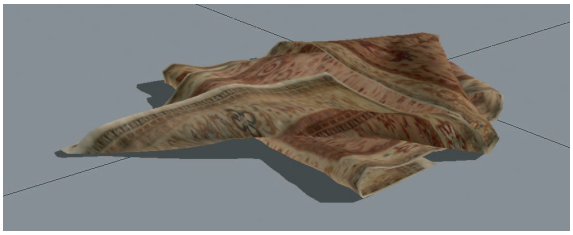

Figure 6: *This folded configuration demonstrates stable self collision and response.*

Figure 7: *Simulation of overpressure inside a character.*

and stiffness $k = 1$ to the set of constraints. Here t_1^i, t_2^i and t_3^i are the three indices of the vertices belonging to triangle i. The sum computes the actual volume of the closed mesh. It is compared against the original volume V_0 times the overpressure factor k_{pressure}. This constraint function yields the gradients

$$\nabla_{\mathbf{p}_i} C = \sum_{j:t_1^j=i} (\mathbf{p}_{t_2^j} \times \mathbf{p}_{t_3^j}) + \sum_{j:t_2^j=i} (\mathbf{p}_{t_3^j} \times \mathbf{p}_{t_1^j}) + \sum_{j:t_3^j=i} (\mathbf{p}_{t_1^j} \times \mathbf{p}_{t_2^j}) \tag{15}$$

These gradients have to be scaled by the scaling factor given in Eq. (7) and weighted by the masses according to Eq. (9) to get the final projection offsets $\Delta\mathbf{p}_i$.

5. Results

We have integrated our method into *Rocket* [Rat04], a game-like environment for physics simulation. Various experiments have been carried out to analyze the characteristics and the performance of the proposed method. All test scenarios presented in this section have been performed on a PC Pentium 4, 3 GHz.

Independent Bending and Stretching. Our bending term only depends on the dihedral angle of adjacent triangles, not on edge lengths, so bending and stretching resistances can be chosen independently. Figure 3 shows a cloth bag with various stretching stiffnesses, first with bending resistance enabled and then disabled. As the top row shows, bending does not influence stretching resistance.

Attachments with Two Way Interaction. We can simulate both, one way and two way coupled attachment constraints. The cloth stripes in Figure 8 are attached via one way constraints to the static rigid bodies at the top. In addition, two way interaction is enabled between the stripes and the bottom rigid bodies. This configuration results in realistically looking swing and twist motions of the stripes. The scene features 6 rigid bodies and 3 pieces of cloth which are simulated and rendered with more than 380 fps.

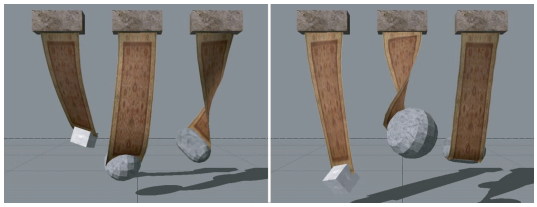

Figure 8: *Cloth stripes are attached via one way interaction to static rigid bodies at the top and via two way constraints to rigid bodies at the bottom.*

Real Time Self Collision. The piece of cloth shown in Figure 6 is composed of 1364 vertices and 2562 triangles. The simulation runs at 30 fps on average including self collision detection, collision handling and rendering. The effect of friction is shown in Figure 9 where the same piece of cloth is tumbling in a rotating barrel.

Tearing and stability. Figure 10 shows a piece of cloth consisting of 4264 vertices and 8262 triangles that is torn open by an attached cube and finally ripped apart by a thrown ball. This scene is simulated and rendered with 47 fps on average. Tearing is simulated by a simple process: Whenever the stretching of an edge exceeds a specified threshold value, we select one of the edge's adjacent vertices. We then put a split plane through that vertex perpendicular to the edge direction and split the vertex. All triangles above the split plane are assigned to the original vertex while all triangles below are assigned to the duplicate. Our method remains stable even in extreme situations as shown in Figure 1, a scene inspired by [ITF04]. An inflated character model is squeezed through rotating gears resulting in multiple constraints, collisions and self collisions acting on single cloth vertices.

Complex Simulation Scenarios. The presented method is especially suited for complex simulation environments (see Figure 12). Despite the extensive interaction with animated characters and geometrically complex game levels, simulation and rendering of multiple pieces of cloth can still be done at interactive speed.

6. Conclusions

We have presented a position based dynamics framework that can handle general constraints formulated via constraint functions. With the position based approach it is possible to manipulate objects directly during the simulation. This significantly simplifies the handling of collisions, attachment constraints and explicit integration and it makes direct and immediate control of the animated scene possible.

We have implemented a robust cloth simulator on top of this framework which provides features like two way interaction of cloth with rigid bodies, cloth self collision and response and attachments of pieces of cloth to dynamic rigid bodies.

7. Future Work

A topic we have not treated in this paper is rigid body simulation. However, the approach we presented could quite easily be extended to handle rigid objects as well. Instead of computing a set of linear and angular impulses for the resolution of collisions as regular rigid body solvers typically do, movements and rotations would be applied to the bodies at the contact points and the linear and angular velocities would have to be adjusted accordingly after the solver has completed.

Figure 9: *Influenced by collision, self collision and friction, a piece of cloth tumbles in a rotating barrel.*

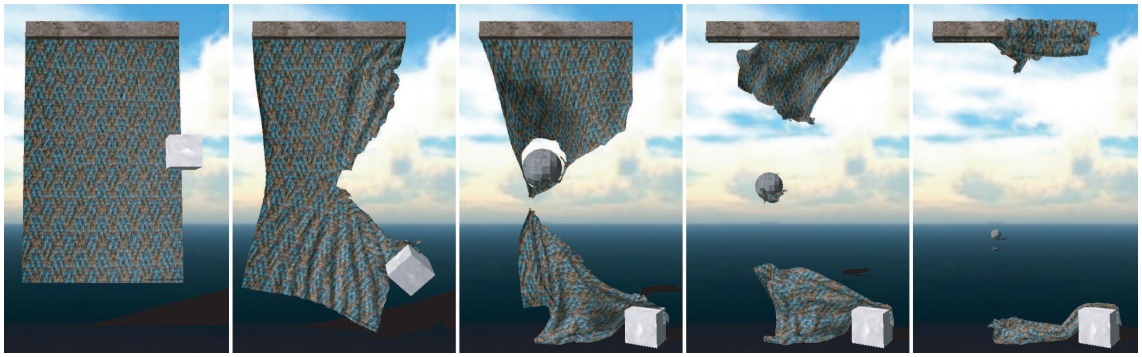

Figure 10: *A piece of cloth is torn open by an attached cube and ripped apart by a thrown ball.*

Figure 11: *Three inflated characters experience multiple collisions and self collisions.*

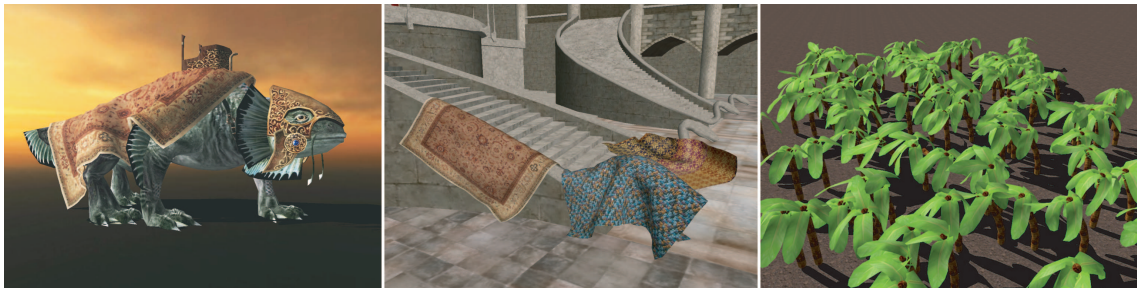

Figure 12: *Extensive interaction between pieces of cloth and an animated game character (left), a geometrically complex game level (middle) and hundreds of simulated plant leaves (right).*

References

[BFA02] BRIDSON R., FEDKIW R., ANDERSON J.: Robust treatment of collisions, contact and friction for cloth animation. *Proceedings of ACM Siggraph* (2002), 594–603.

[BMF03] BRIDSON R., MARINO S., FEDKIW R.: Simulation of clothing with folds and wrinkles. In *ACM SIGGRAPH Symposium on Computer Animation* (2003), pp. 28–36.

[BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. *Proceedings of ACM Siggraph* (1998), 43–54.

[BWK03] BARAFF D., WITKIN A., KASS M.: Untangling cloth. In *Proceedings of the ACM SIGGRAPH* (2003), pp. 862–870.

[CBP05] CLAVET S., BEAUDOIN P., POULIN P.: Particle-based viscoelastic fluid simulation. *Proceedings of the ACM SIGGRAPH Symposium on Computer Animation* (2005), 219–228.

[DSB99] DESBRUN M., SCHRÖDER P., BARR A.: Interactive animation of structured deformable objects. In *Proceedings of Graphics Interface '99* (1999), pp. 1–8.

[Fau98] FAURE F.: Interactive solid animation using linearized displacement constraints. In *Eurographics Workshop on Computer Animation and Simulation (EGCAS)* (1998), pp. 61–72.

[Fed05] FEDOR M.: Fast character animation using particle dynamics. *Proceedings of International Conference on Graphics, Vision and Image Processing, GVIP05* (2005).

[GHDS03] GRINSPUN E., HIRANI A., DESBRUN M., SCHRODER P.: Discrete shells. In *Proceedings of the ACM SIGGRAPH Symposium on Computer Animation* (2003).

[ITF04] IRVING G., TERAN J., FEDKIW R.: Invertible finite elements for robust simulation of large deformation. In *Proceedings of the ACM SIGGRAPH Symposium on Computer Animation* (2004), pp. 131–140.

[Jak01] JAKOBSEN T.: Advanced character physics Ű the fysix engine. *www.gamasutra.com* (2001).

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHER M., GROSS M.: Meshless deformations based on shape matching. *Proceedings of ACM Siggraph* (2005), 471–478.

[NMK*05] NEALEN A., MÜLLER M., KEISER R., BOXERMAN E., CARLSON M.: Physically based deformable models in computer graphics. *Eurographics 2005 state of the art report* (2005).

[Pro95] PROVOT X.: Deformation constraints in a mass-spring model to describe rigid cloth behavior. *Proceedings of Graphics Interface* (1995), 147Ű–154.

[Rat04] RATCLIFF J.: Rocket - a viewer for real-time phyics simulations. *www.physicstools.org* (2004).

[THM*03] TESCHNER M., HEIDELBERGER B., MÜLLER M., POMERANERTS D., GROSS M.: Optimized spatial hashing for collision detection of deformable objects. *Proc. Vision, Modeling, Visualization VMV 2003* (2003), 47–54.

[THMG04] TESCHNER M., HEIDELBERGER B., MÜLLER M., GROSS M.: A versatile and robust model for geometrically complex deformable solids. *Proceedings of Computer Graphics International (CGI)* (2004), 312–319.

[VCMT95] VOLINO P., COURCHESNE M., MAGNENAT-THALMANN N.: Versatile and efficient techniques for simulating cloth and other deformable objects. *Proceedings of ACM Siggraph* (1995), 137–144.

Appendix A:

Gradient of the Normalized Cross Product

Constraint functions often contain normalized cross products. To derive the projection corrections, the gradient of the constraint function is needed. Therefore it is useful to know the gradient of a normalized cross product with respect to both arguments. Given the normalized cross product $\mathbf{n} = \frac{\mathbf{p}_1 \times \mathbf{p}_2}{|\mathbf{p}_1 \times \mathbf{p}_2|}$, the derivative with respect to the first vector is

$$\frac{\partial \mathbf{n}}{\partial \mathbf{p}_1} = \begin{bmatrix} \frac{\partial n_x}{\partial p_{1x}} & \frac{\partial n_x}{\partial p_{1y}} & \frac{\partial n_x}{\partial p_{1z}} \\ \frac{\partial n_y}{\partial p_{1x}} & \frac{\partial n_y}{\partial p_{1y}} & \frac{\partial n_y}{\partial p_{1z}} \\ \frac{\partial n_z}{\partial p_{1x}} & \frac{\partial n_z}{\partial p_{1y}} & \frac{\partial n_z}{\partial p_{1z}} \end{bmatrix} \tag{16}$$

$$= \frac{1}{|\mathbf{p}_1 \times \mathbf{p}_2|} \left(\begin{bmatrix} 0 & p_{2z} & -p_{2y} \\ -p_{2z} & 0 & p_{2x} \\ p_{2y} & -p_{2x} & 0 \end{bmatrix} + \mathbf{n}(\mathbf{n} \times \mathbf{p}_2)^T \right) \tag{17}$$

Shorter and for both arguments we have

$$\frac{\partial \mathbf{n}}{\partial \mathbf{p}_1} = \frac{1}{|\mathbf{p}_1 \times \mathbf{p}_2|} \left(-\tilde{\mathbf{p}}_2 + \mathbf{n}(\mathbf{n} \times \mathbf{p}_2)^T \right) \tag{18}$$

$$\frac{\partial \mathbf{n}}{\partial \mathbf{p}_2} = -\frac{1}{|\mathbf{p}_1 \times \mathbf{p}_2|} \left(-\tilde{\mathbf{p}}_1 + \mathbf{n}(\mathbf{n} \times \mathbf{p}_1)^T \right) \tag{19}$$

$$\tag{20}$$

where $\tilde{\mathbf{p}}$ is the matrix with the property $\tilde{\mathbf{p}}\mathbf{x} = \mathbf{p} \times \mathbf{x}$.

Bending Constraint Projection

The constraint function for bending is $C = \arccos(d) - \varphi_0$, where $d = \mathbf{n}_1 \cdot \mathbf{n}_2 = \mathbf{n}_1^T \mathbf{n}_2$. Without loss of generality we set $\mathbf{p}_1 = \mathbf{0}$ and get for the normals $\mathbf{n}_1 = \frac{\mathbf{p}_2 \times \mathbf{p}_3}{|\mathbf{p}_2 \times \mathbf{p}_3|}$ and $\mathbf{n}_2 = \frac{\mathbf{p}_2 \times \mathbf{p}_4}{|\mathbf{p}_2 \times \mathbf{p}_4|}$. With $\frac{d}{dx} \arccos(x) = -\frac{1}{\sqrt{1-x^2}}$ we get the following gradients:

$$\nabla_{\mathbf{p}_3} C = -\frac{1}{\sqrt{1-d^2}} \left(\left(\frac{\partial \mathbf{n}_1}{\partial \mathbf{p}_3} \right)^T \mathbf{n}_2 \right) \tag{21}$$

$$\nabla_{\mathbf{p}_4} C = -\frac{1}{\sqrt{1-d^2}} \left(\left(\frac{\partial \mathbf{n}_2}{\partial \mathbf{p}_4} \right)^T \mathbf{n}_1 \right) \tag{22}$$

$$\nabla_{\mathbf{p}_2} C = -\frac{1}{\sqrt{1-d^2}} \left(\left(\frac{\partial \mathbf{n}_1}{\partial \mathbf{p}_2} \right)^T \mathbf{n}_2 + \left(\frac{\partial \mathbf{n}_2}{\partial \mathbf{p}_2} \right)^T \mathbf{n}_1 \right) \tag{23}$$

$$\nabla_{\mathbf{p}_1} C = -\nabla_{\mathbf{p}_2} C - \nabla_{\mathbf{p}_3} C - \nabla_{\mathbf{p}_4} C \tag{24}$$

Using the gradients of normalized cross products, first compute

$$\mathbf{q}_3 = \frac{\mathbf{p}_2 \times \mathbf{n}_2 + (\mathbf{n}_1 \times \mathbf{p}_2) d}{|\mathbf{p}_2 \times \mathbf{p}_3|} \tag{25}$$

$$\mathbf{q}_4 = \frac{\mathbf{p}_2 \times \mathbf{n}_1 + (\mathbf{n}_2 \times \mathbf{p}_2) d}{|\mathbf{p}_2 \times \mathbf{p}_4|} \tag{26}$$

$$\mathbf{q}_2 = -\frac{\mathbf{p}_3 \times \mathbf{n}_2 + (\mathbf{n}_1 \times \mathbf{p}_3) d}{|\mathbf{p}_2 \times \mathbf{p}_3|} - \frac{\mathbf{p}_4 \times \mathbf{n}_1 + (\mathbf{n}_2 \times \mathbf{p}_4) d}{|\mathbf{p}_2 \times \mathbf{p}_4|} \tag{27}$$

$$\mathbf{q}_1 = -\mathbf{q}_2 - \mathbf{q}_3 - \mathbf{q}_4 \tag{28}$$

Then the final correction is

$$\Delta \mathbf{p}_i = -\frac{w_i \sqrt{1-d^2} (\arccos(d) - \varphi_0)}{\sum_j w_j |\mathbf{q}_j|^2} \mathbf{q}_i \tag{29}$$