Position Based Dynamics

Paper Review: Summary with Visualization and Questions

SNU ECE 23
Sanghwa Lee

Muller et al. (2007).

Paper review

Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
J. Vis. Commun. Image R. 18 (2007) 109–118
Journal of VISUAL Communication & IMAGE Representation
www.elsevier.com/locate/jvci

Position based dynamics ☆

Matthias Müller [a,*], Bruno Heidelberger [a], Marcus Hennix [a], John Ratcliff [b]

[a] AGEIA Technologies, Technoparkstrasse 1, 8005 Zürich, Switzerland
[b] AGEIA Technologies, 4041 Forest Park Avenue, St. Louis, MO 63108, USA

Received 8 January 2007; accepted 9 January 2007
Available online 12 February 2007

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.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Physically based simulation; Game physics; Integration schemes; Cloth simulation

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

☆ Reprinted with permission from Proc. Virtual Reality Interactions and Physical Simulations, Vriphys, Madrid, Spain, pp. 71–80, Nov. 6–7, 2006.

* Corresponding author. Fax: +41 44 445 2147.
E-mail address: mmueller@ageia.com (M. Müller).

1047-3203/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jvcir.2007.01.005

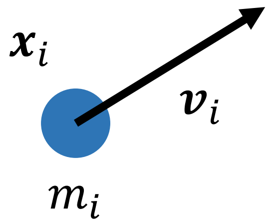
Müller, Matthias, et al. "Position Based Dynamics." Journal of Visual Communication and Image Representation, vol. 18, no. 2, Apr. 2007, pp. 109–118.

Muller et al. (2007).

Algorithm overview

N particles

i th particle



M constraints

j th constraint

$$C_j\left(\boldsymbol{x}_{i_1}, \cdots, \boldsymbol{x}_{n_j}\right) = 0$$

or

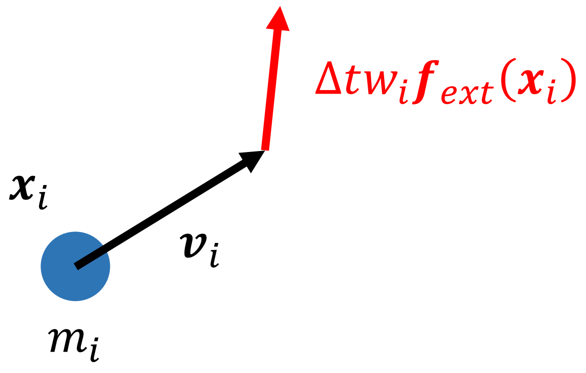
$$C_j\left(\boldsymbol{x}_{i_1}, \cdots, \boldsymbol{x}_{n_j}\right) \leq 0$$

forall vertices i
  initialize $\mathbf{x}_i = \mathbf{x}_i^0$, $\mathbf{v}_i = \mathbf{v}_i^0$, $w_i = 1/m_i$
endfor
loop
  forall vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{\text{ext}}(\mathbf{x}_i)$
  dampVelocities($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
  forall vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
  forall vertices i **do** generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
  loop solverIterations **times**
    projectConstraints($C_1, \ldots, C_{M+M_{\text{coll}}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
  endloop
  forall vertices i
    $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
    $\mathbf{x}_i \leftarrow \mathbf{p}_i$
  endfor
  velocityUpdate($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
endloop

Muller et al. (2007).

Algorithm overview

i th particle



① *Update the velocity*

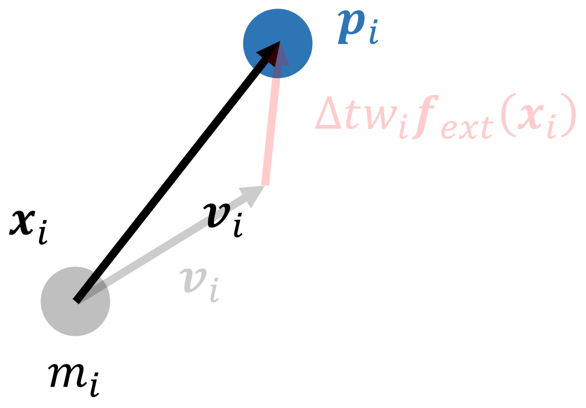
$$\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \Delta t w_i \boldsymbol{f}_{ext}(\boldsymbol{x}_i)$$

The affect of the external force applied to the particle

Muller et al. (2007).

Algorithm overview

$\boldsymbol{i}$ th particle



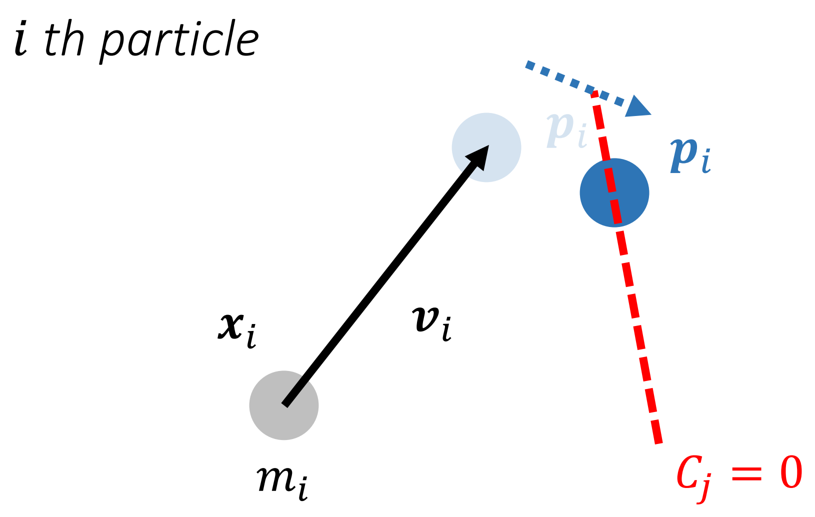
② *Expect the position*

$$\boldsymbol{p}_i \leftarrow \boldsymbol{x}_i + \Delta t \boldsymbol{v}_i$$

Pre-solve the expected position with Euler step

Muller et al. (2007).

Algorithm overview



③ *Consider the constraints*

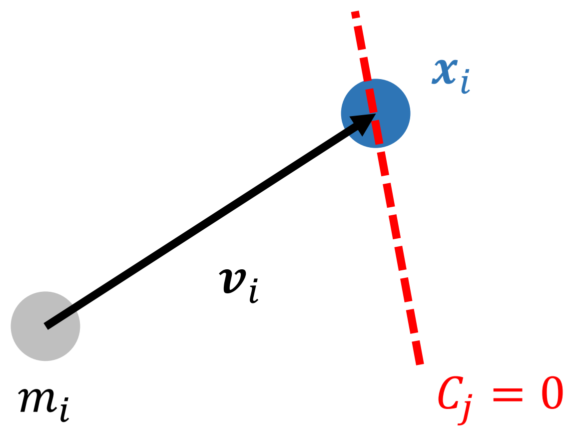
$$projectConstraints(C_j, \cdots, p_i, \cdots)$$

Constraints(initial length, bending, etc.) are considered to correct the expected position

Muller et al. (2007).

Algorithm overview

i *th particle*



④ *Correct the velocity*

$$\boldsymbol{v}_i \leftarrow (\boldsymbol{p}_i - \boldsymbol{x}_i)/\Delta t$$

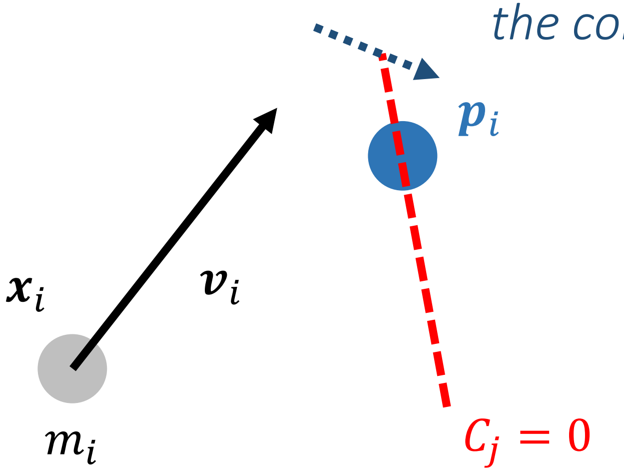
$$\boldsymbol{x}_i \leftarrow \boldsymbol{p}_i$$

Correct the velocity so that the constraints are considered, and update the position

Muller et al. (2007).

Projecting constraints

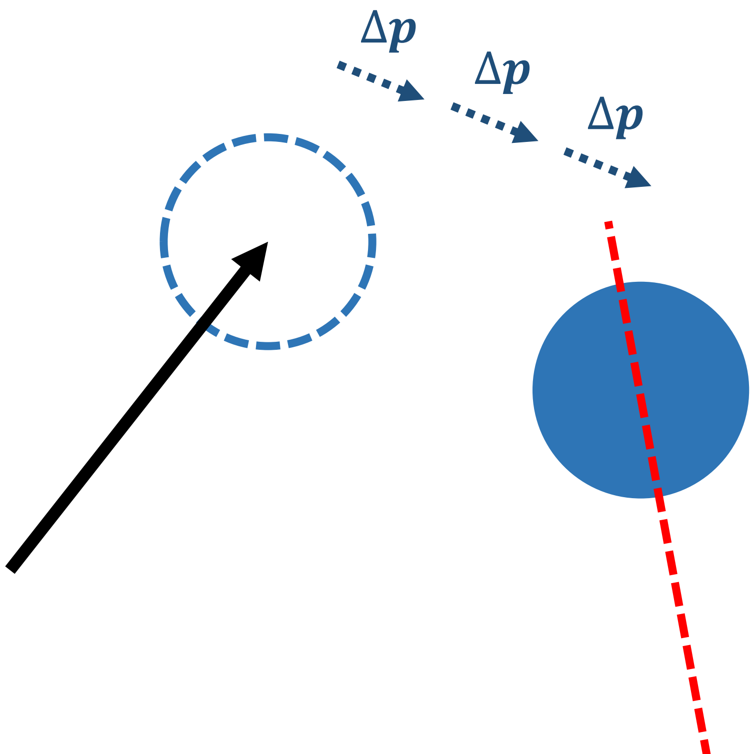
How can we consider the constraints?



$$projectConstraints(C_j, \cdots, p_i, \cdots)$$

Gauss-Seidel-type iteration

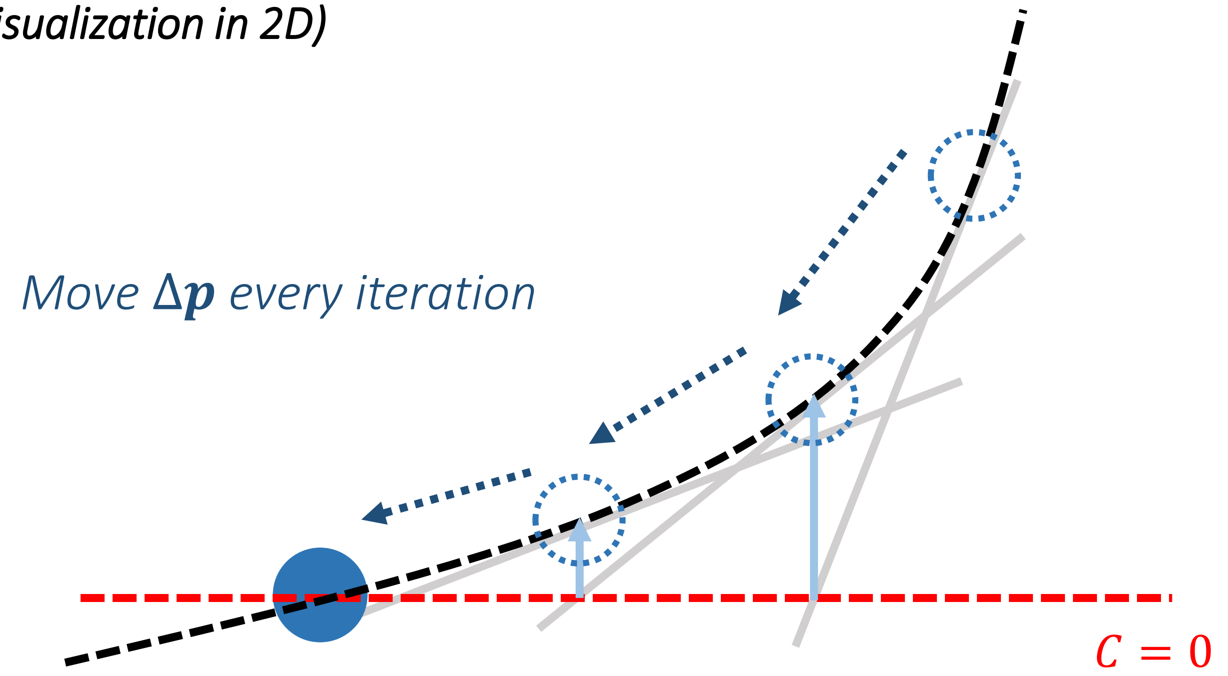
Iteratively find by step $\Delta\boldsymbol{p}$



Muller et al. (2007).

Iteration method

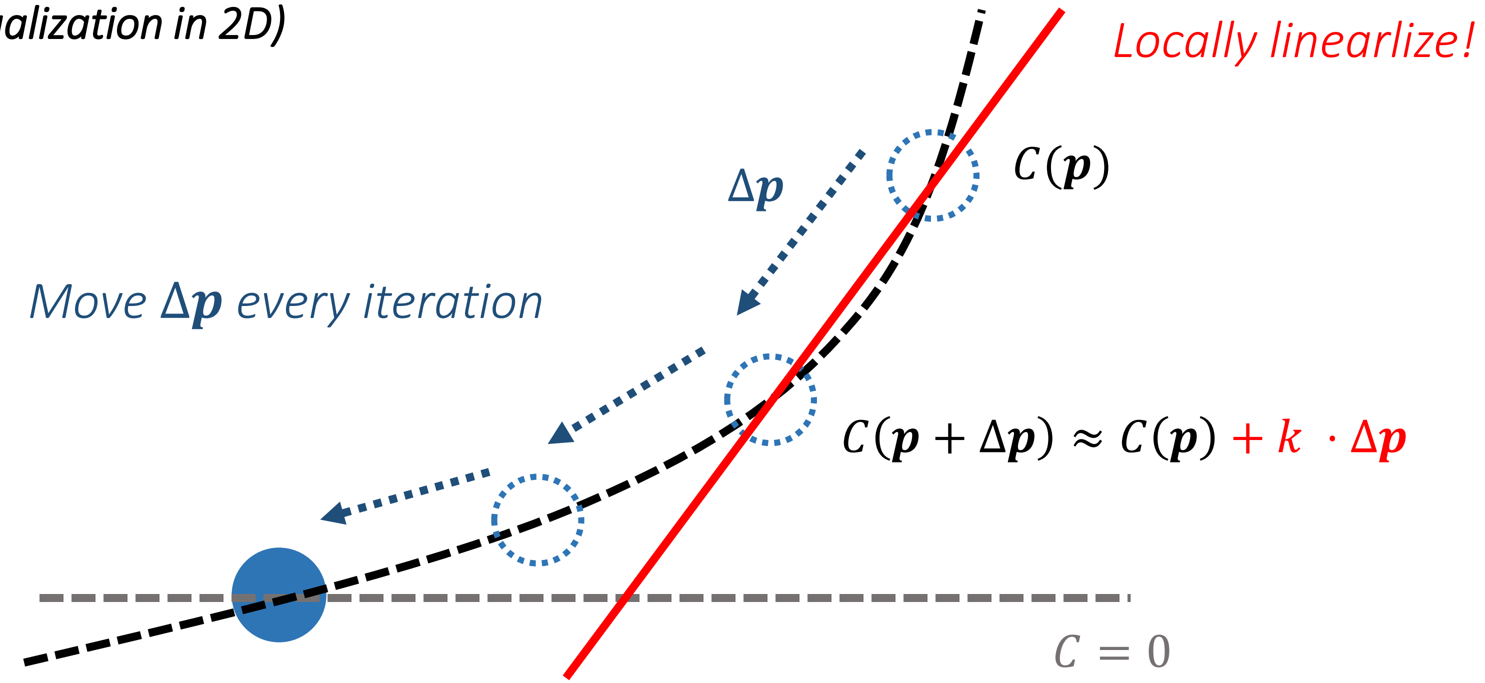
Newton-Raphson process
(visualization in 2D)



Muller et al. (2007).

Iteration method

Newton-Raphson process
(visualization in 2D)



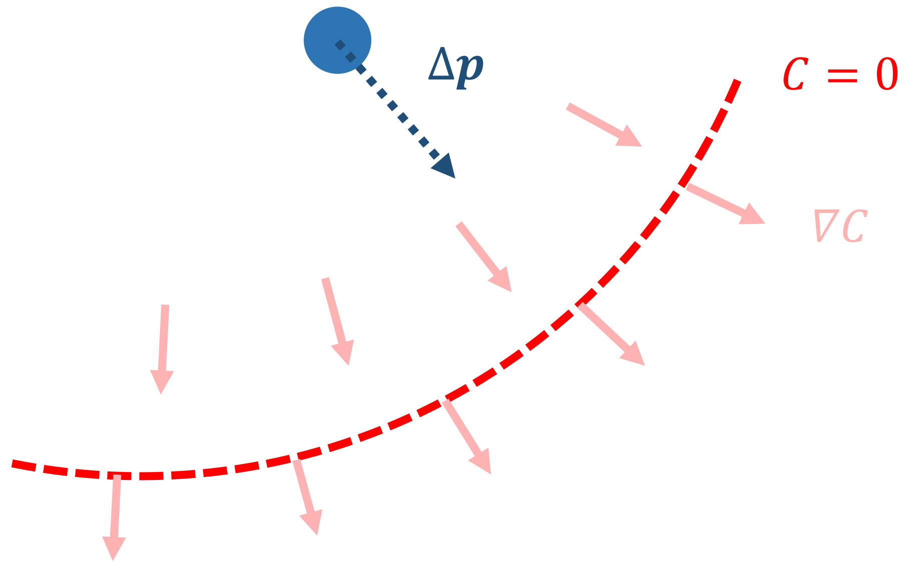
Muller et al. (2007).

Iteration method

C: Constraint of n points

$$\boldsymbol{p} = [\boldsymbol{p}_1^T, \cdots, \boldsymbol{p}_n^T]^T \quad \textit{(Concantation)}$$

$3n$ Dimension



$$C(\boldsymbol{p} + \Delta\boldsymbol{p}) \approx C(\boldsymbol{p}) + \nabla_{\boldsymbol{p}} C(\boldsymbol{p}) \cdot \Delta\boldsymbol{p} = 0$$

Fastest approach $\Delta\boldsymbol{p} \parallel \nabla C$

$$\Delta\boldsymbol{p} = \lambda \nabla_{\boldsymbol{p}} C(\boldsymbol{p}) = -\frac{C(\boldsymbol{p})}{\left|\nabla_{\boldsymbol{p}} C(\boldsymbol{p})\right|^2} \nabla_{\boldsymbol{p}} C(\boldsymbol{p})$$

For a single point $\boldsymbol{i}$,

$$\Delta\boldsymbol{p}_i = -\underbrace{\boxed{\frac{n \cdot w_i}{\sum_j w_j}}}_{\text{Weight distribution}} \underbrace{\boxed{\frac{C(\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n)}{\sum_j \left|\nabla_{\boldsymbol{p}_j} C(\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n)\right|^2}}}_{\text{Scaling factor } s} \nabla_{\boldsymbol{p}_i} C(\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n)$$

Muller et al. (2007).

Constraint type

Type equality $C(\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n) = 0$

$\Rightarrow$ *Always perform a projection*

Type inequality $C(\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n) \leq 0$

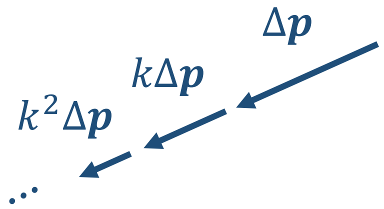
$\Rightarrow$ *Only when* $C < 0$

$+$

Stiffness parameter

$0 \leq k \leq 1$

Multiply by k



$\Delta p \rightarrow k\Delta p$

Error after $\boldsymbol{n_s}$ *iterations*

$\Rightarrow \Delta p(1-k)^{n_s}$ Non-linear

$\downarrow$

Set $k' = 1 - (1-k)^{1/n_s}$

$\Rightarrow \Delta p(1-k')^{n_s} = \Delta p(1-k)$ Linear

Muller et al. (2007).

Collision handling

Total number of constraints $M + M_{coll}$

Collision constraint

(Generated every step)

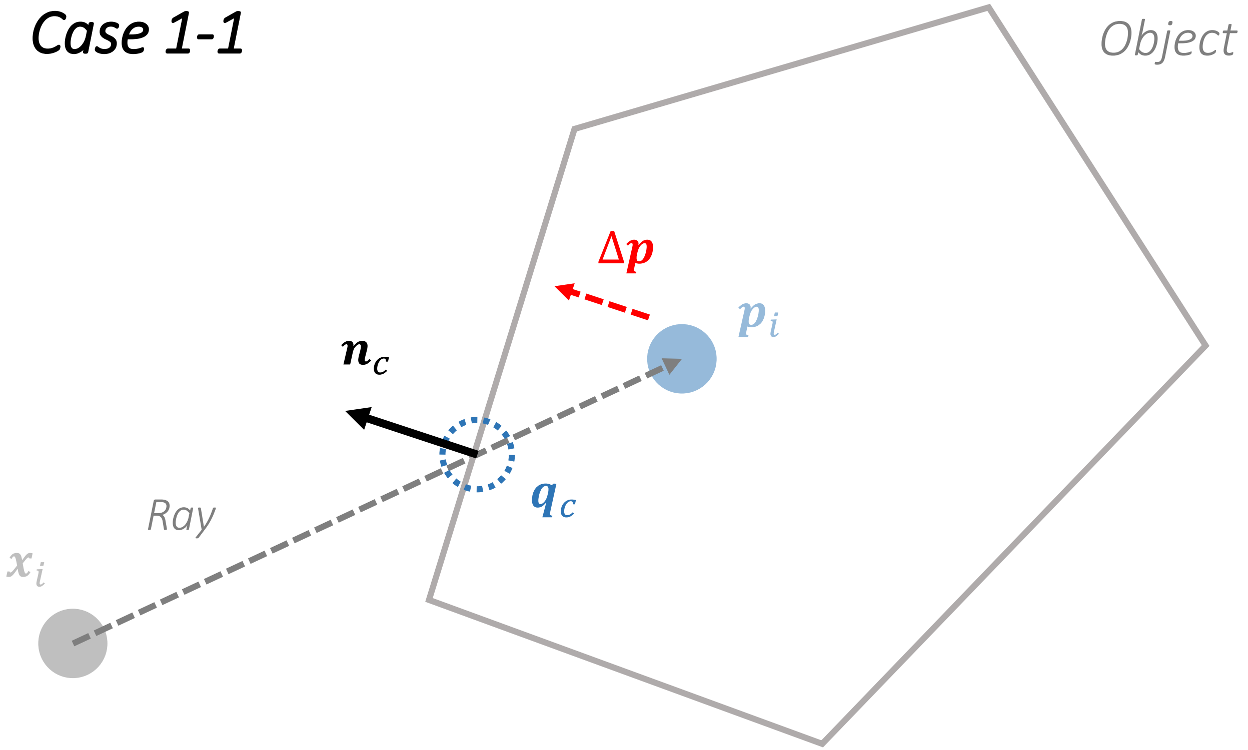
Case 1 *with static objects*

1-1 Continuous collision handling

1-2 Static collision handling

Case 2 *between dynamic objects*

Case 1-1



$$C(\boldsymbol{p}) = (\boldsymbol{p} - \boldsymbol{q}_c) \cdot \boldsymbol{n}_c \leq 0$$

$$k = 1$$

Muller et al. (2007).

Collision handling

Total number of constraints $M + M_{coll}$

Collision constraint

(Generated every step)

Case 1 *with static objects*

1-1 Continuous collision handling

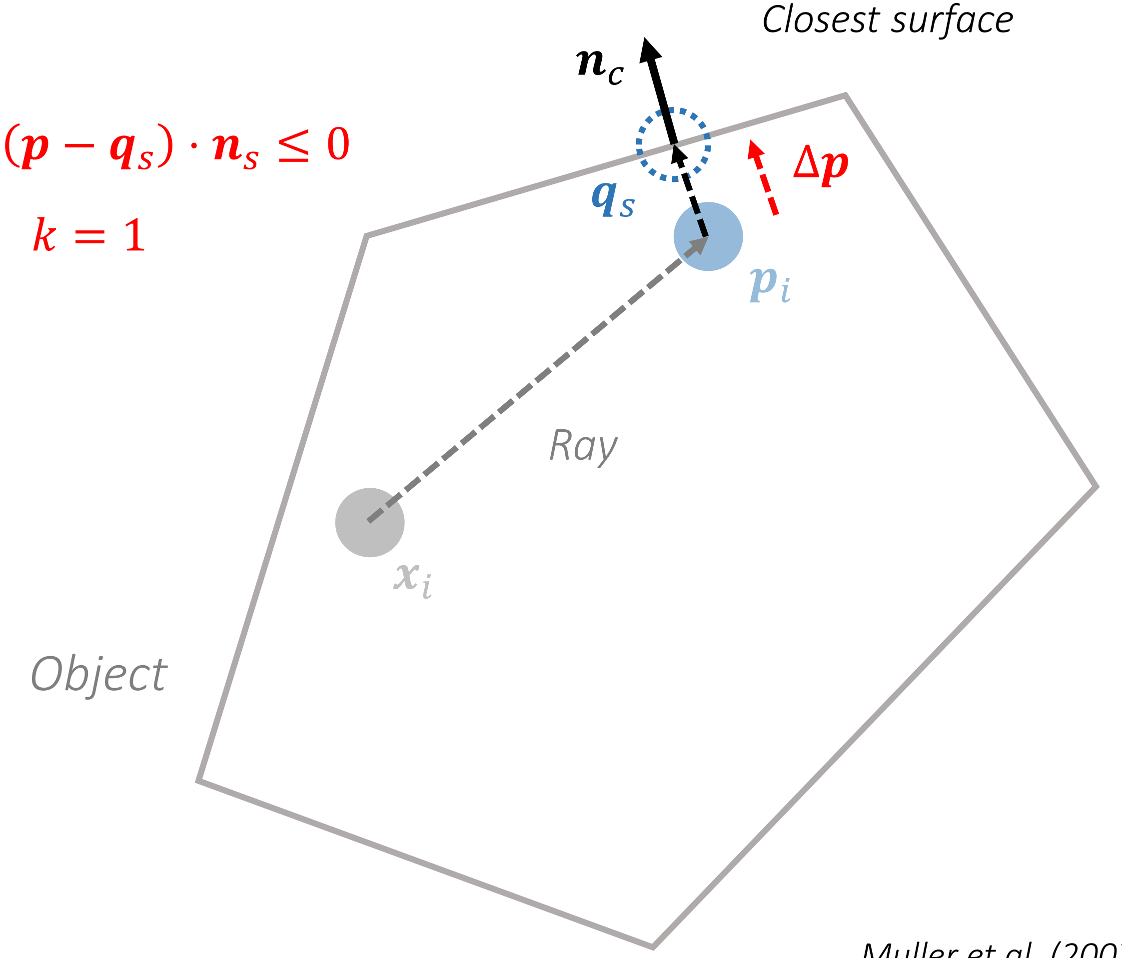
1-2 Static collision handling

Case 2 *between dynamic objects*

Case 1-2

$$C(\boldsymbol{p}) = (\boldsymbol{p} - \boldsymbol{q}_s) \cdot \boldsymbol{n}_s \leq 0$$

$$k = 1$$



Muller et al. (2007).

Collision handling

Total number of constraints $M + M_{coll}$

Collision constraint

(Generated every step)

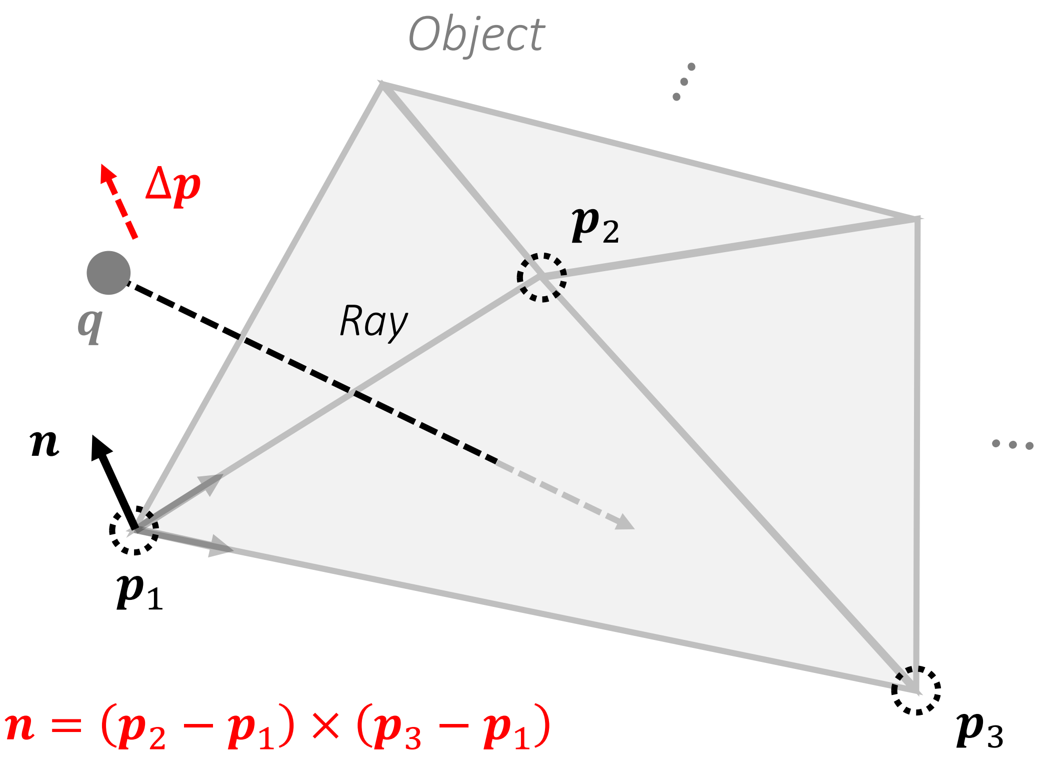
Case 1 *with static objects*

1-1 Continuous collision handling

1-2 Static collision handling

Case 2 *between dynamic objects*

Case 2



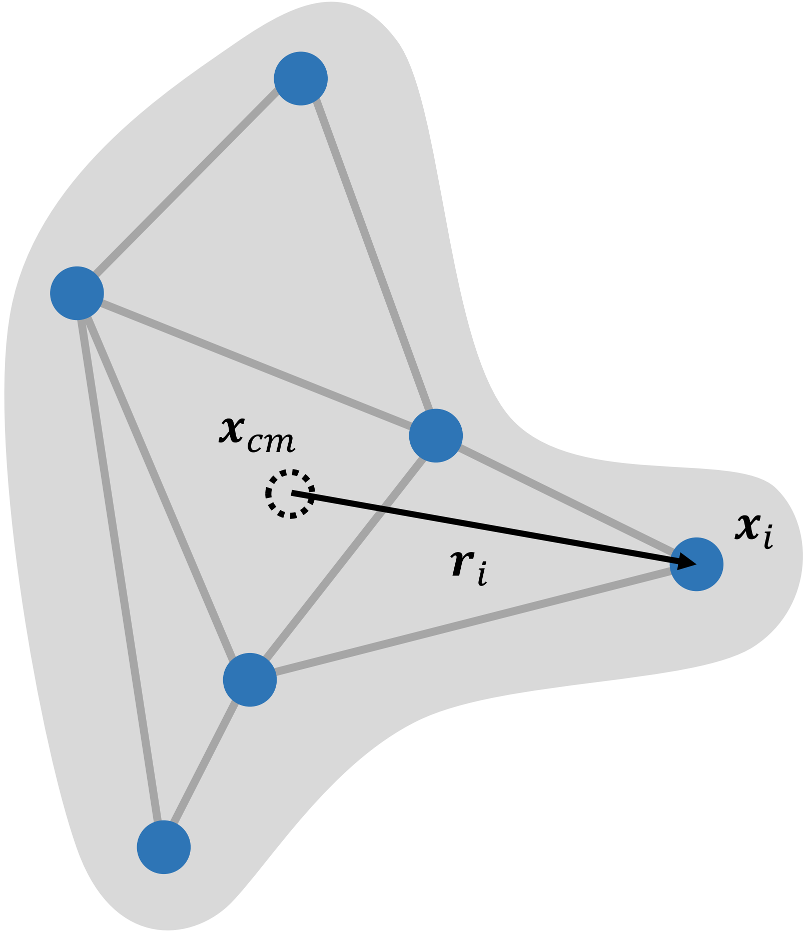
$$\boldsymbol{n} = (\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_3 - \boldsymbol{p}_1)$$

$$C(\boldsymbol{q}, \boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3) = \pm(\boldsymbol{q} - \boldsymbol{p}_1) \cdot \boldsymbol{n} \le 0$$

Muller et al. (2007).

Damping

$\boldsymbol{I} = ?$



Skew-symmetric matrix

$$\tilde{\boldsymbol{r}}_i \boldsymbol{v} = \boldsymbol{r}_i \times \boldsymbol{v}$$

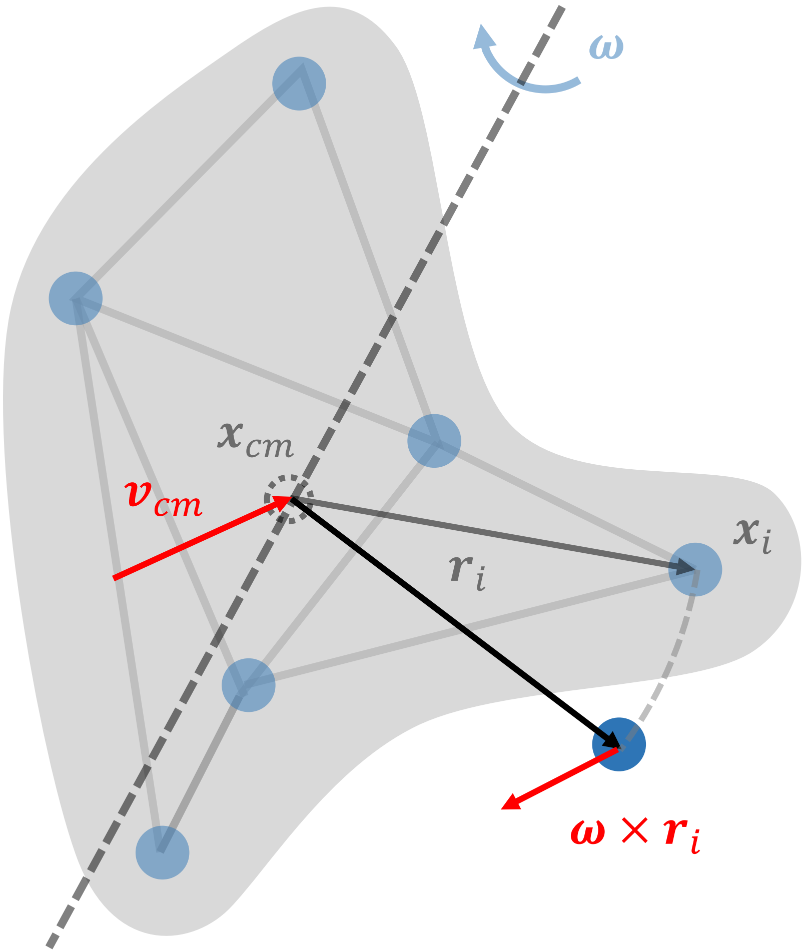
$$\boldsymbol{r}_i = \left(r_x, r_y, r_z\right)_i \qquad \tilde{\boldsymbol{r}}_i = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}_i$$

$$\tilde{\boldsymbol{r}}_i \tilde{\boldsymbol{r}}_i^T m_i = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}_i = \boldsymbol{I}_i$$

Inertia tensor $\quad \boldsymbol{I} = \sum_i \tilde{\boldsymbol{r}}_i \tilde{\boldsymbol{r}}_i^T m_i$

Muller et al. (2007).

Damping



$$I = \sum_i \tilde{r}_i \tilde{r}_i^T m_i \qquad L = \sum_i r_i \times (m_i v_i)$$

$$\omega = I^{-1} L$$

Global motion $\quad v_{cm} + \omega \times r_i$

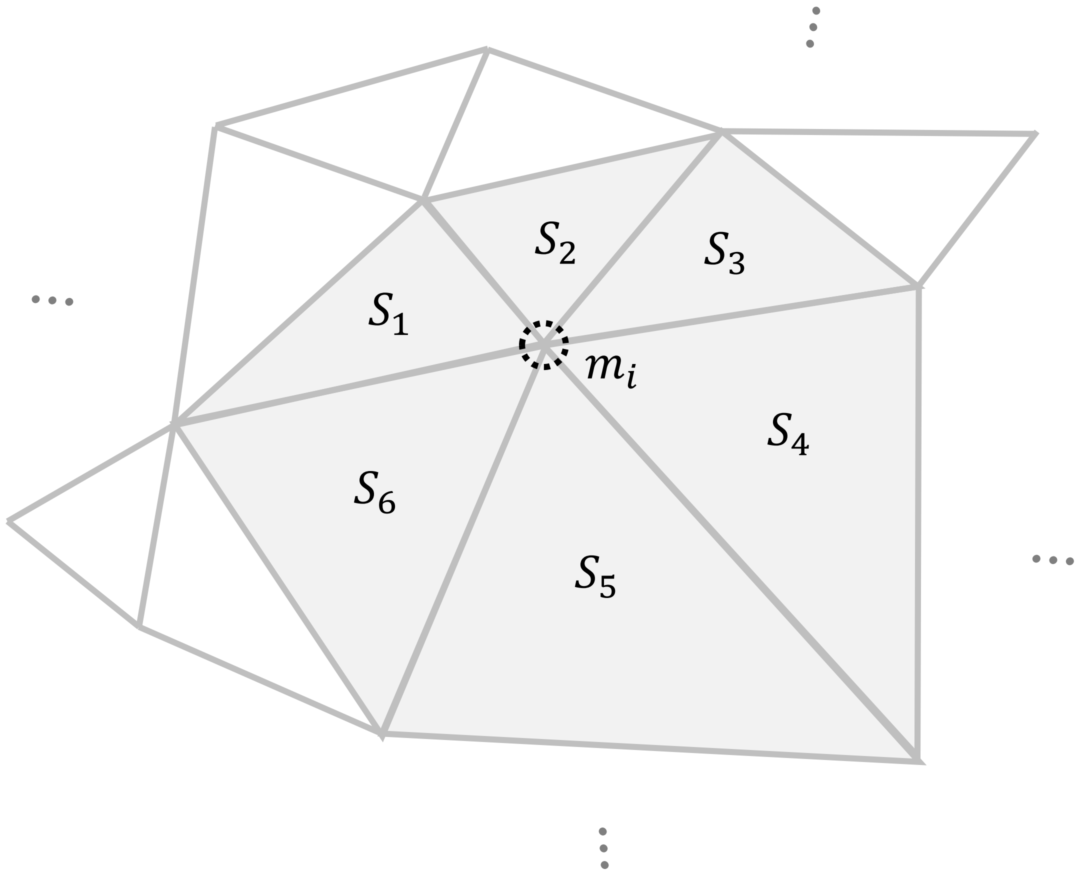
Damping for each i

$$\Delta v_i = v_{cm} + \omega \times r_i - v_i$$

$$v_i \leftarrow v_i + k_{damping} \Delta v_i$$

Muller et al. (2007).

Cloth simulation



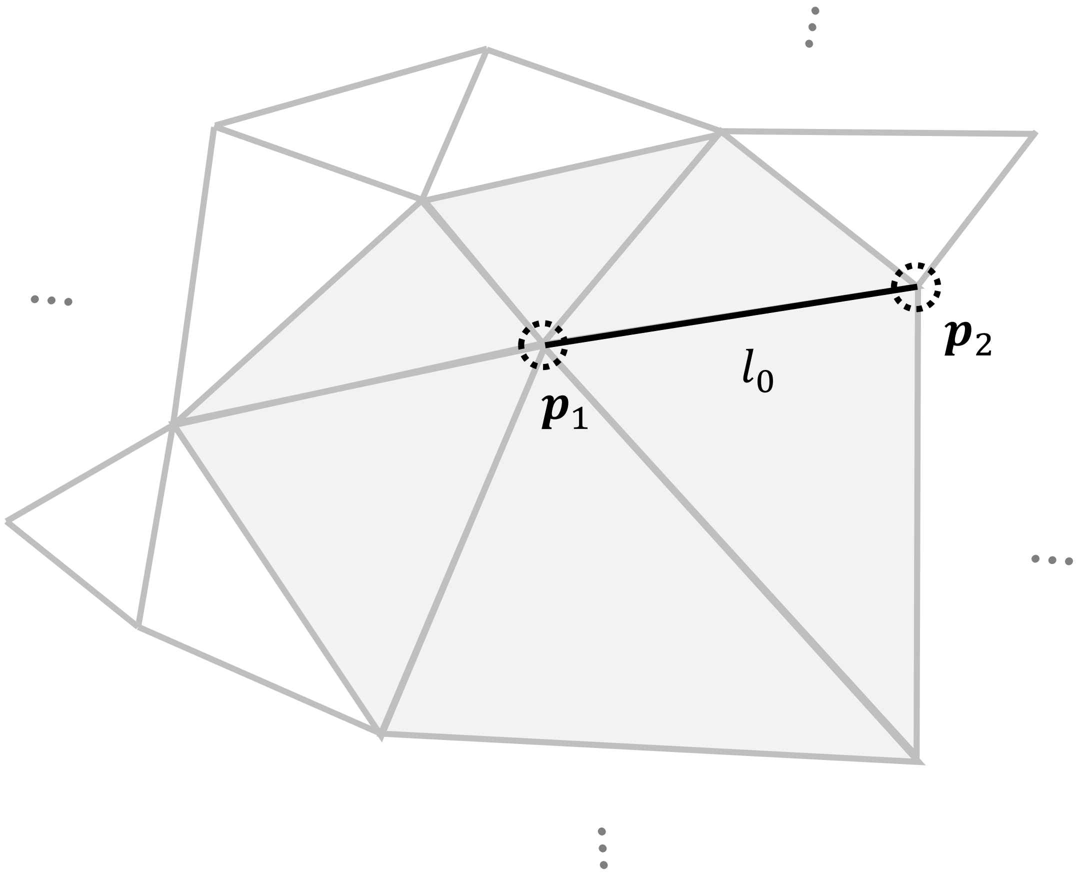
Point mass

Density ρ

$$m_i = \frac{1}{3}\sum_j \rho S_j$$

Muller et al. (2007).

Cloth simulation



Stretching constraint

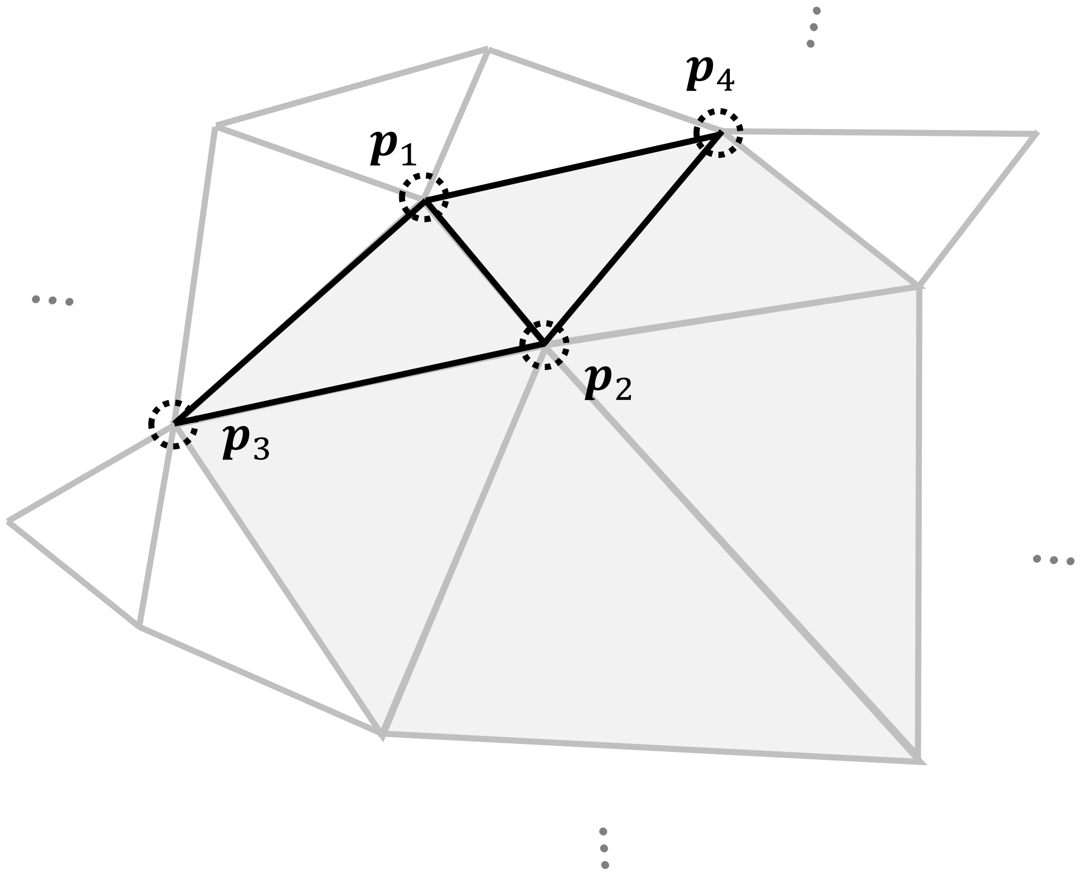
(Type equality)

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

$$0 \leq k_{stretch} \leq 1$$

Muller et al. (2007).

Cloth simulation



Bending constraint

(Type equality)

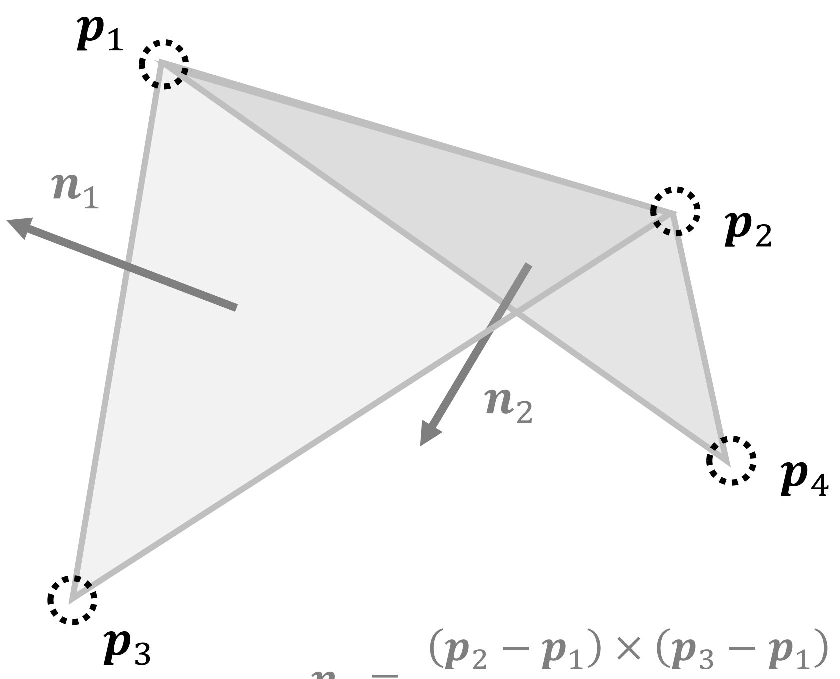
$$C_{bend}(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4)$$

$$= \arccos\left(\frac{(\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_3 - \boldsymbol{p}_1)}{|(\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_3 - \boldsymbol{p}_1)|} \cdot \frac{(\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_4 - \boldsymbol{p}_1)}{|(\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_4 - \boldsymbol{p}_1)|}\right) - \varphi_0$$

$$0 \leq k_{bend} \leq 1$$

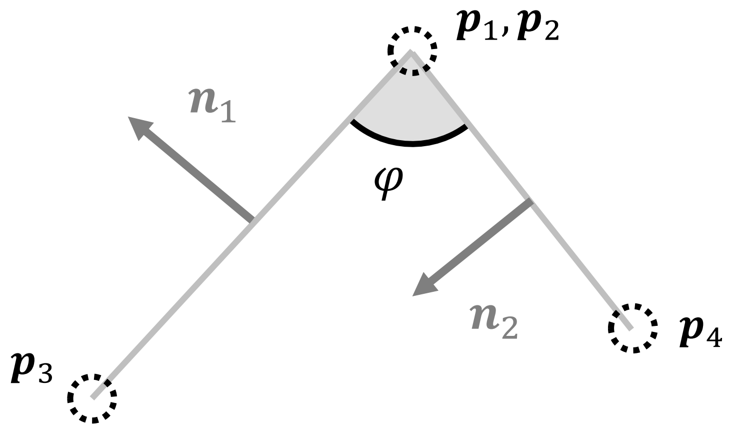
Muller et al. (2007).

Bending constraint



$$\boldsymbol{n}_1 = \frac{(\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_3 - \boldsymbol{p}_1)}{|(\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_3 - \boldsymbol{p}_1)|}$$

$$\boldsymbol{n}_2 = \frac{(\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_4 - \boldsymbol{p}_1)}{|(\boldsymbol{p}_2 - \boldsymbol{p}_1) \times (\boldsymbol{p}_4 - \boldsymbol{p}_1)|}$$



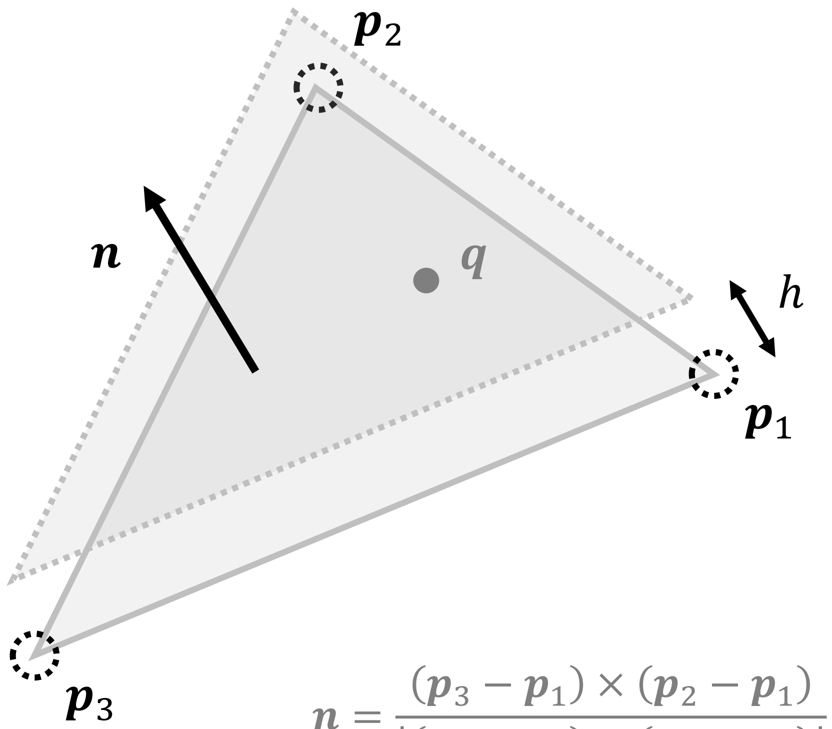
Dihedral angle φ

$$\boldsymbol{n}_1 \cdot \boldsymbol{n}_2 = \cos\varphi$$

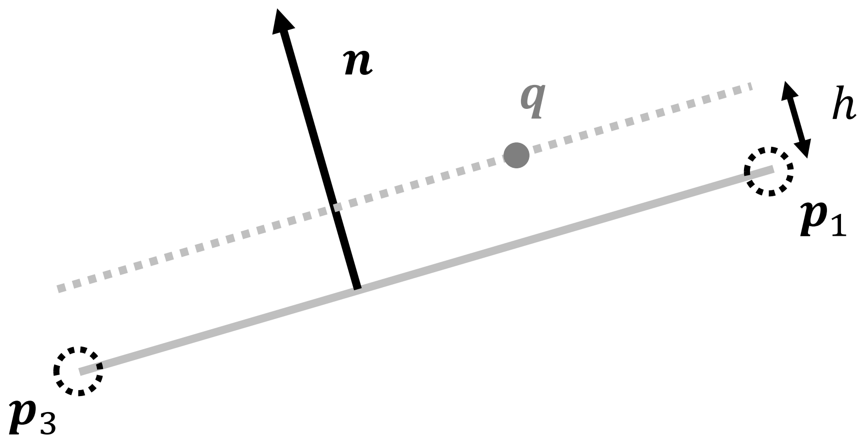
$$C_{bend} = \arccos(\boldsymbol{n}_1 \cdot \boldsymbol{n}_2) - \varphi_0$$

Muller et al. (2007).

Self collision



$$\boldsymbol{n} = \frac{(\boldsymbol{p}_3 - \boldsymbol{p}_1) \times (\boldsymbol{p}_2 - \boldsymbol{p}_1)}{|(\boldsymbol{p}_3 - \boldsymbol{p}_1) \times (\boldsymbol{p}_2 - \boldsymbol{p}_1)|}$$



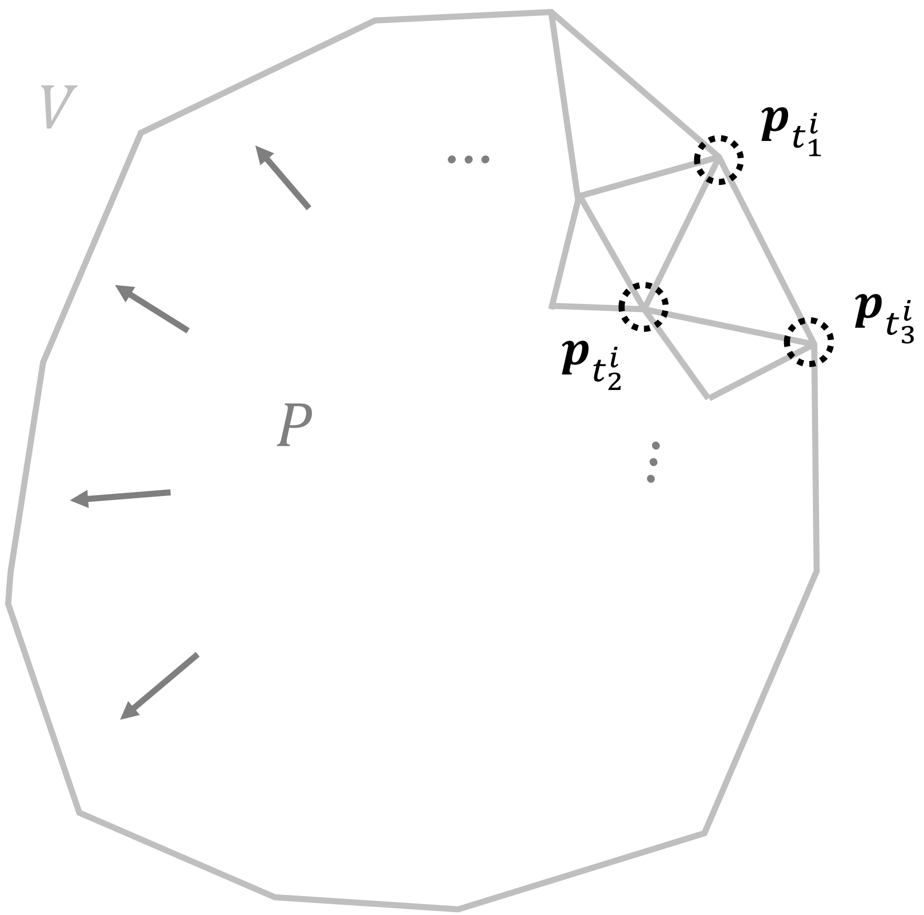
Thickness h

$$C(\boldsymbol{q}, \boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3) = (\boldsymbol{q} - \boldsymbol{p}_1) \cdot \boldsymbol{n} - h$$

$$= (\boldsymbol{q} - \boldsymbol{p}_1) \cdot \frac{(\boldsymbol{p}_3 - \boldsymbol{p}_1) \times (\boldsymbol{p}_2 - \boldsymbol{p}_1)}{|(\boldsymbol{p}_3 - \boldsymbol{p}_1) \times (\boldsymbol{p}_2 - \boldsymbol{p}_1)|} - h$$

Muller et al. (2007).

Cloth balloons



$P \cdot V = const$

Constant

Volume constraint

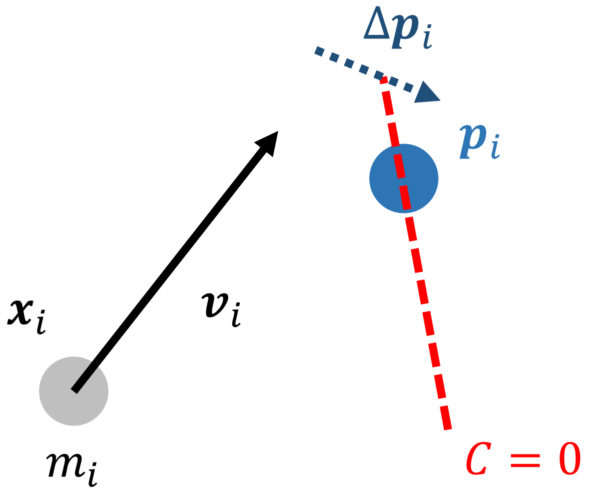
(Type equality)

Initial volume V_0

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

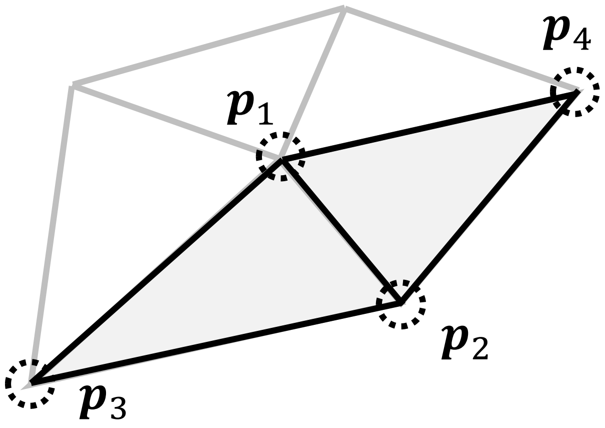
Muller et al. (2007).

Summary



① *Constraint function*

② *Iteration* $$\Delta \boldsymbol{p}_i = -\frac{n \cdot w_i}{\sum_j w_j} \frac{C(\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n)}{\sum_j \left|\nabla_{\boldsymbol{p}_j} C(\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n)\right|^2} \nabla_{\boldsymbol{p}_i} C(\boldsymbol{p}_1, \cdots, \boldsymbol{p}_n)$$



③ *Cloth simulation*

$$C_{stretch}, C_{bend}, C_{collision}, \cdots$$

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

$$C_{bend}(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4) = \arccos(\boldsymbol{n}_1 \cdot \boldsymbol{n}_2) - \varphi_0$$

Muller et al. (2007).

Implementation

Implementation

(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** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{\text{ext}}(\mathbf{x}_i)$
(6) dampVelocities($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
(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($C_1, \ldots, C_{M+M_{\text{coll}}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
(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($\mathbf{v}_1, \ldots, \mathbf{v}_N$)
(17) **endloop**

→

```
void ClothSimulator::updatePBD() {
  explicitEuler();
  for (int n = 0; n < itr; n++) {
    calDistanceConstraint();
    calBendingConstraint();
  }
  integrationForPBD();
  }
```

Muller et al. (2007).

Implementation

```
void ClothSimulator::calDistanceConstraint() {
  for (int i = 0; i < nE; i++) {
    Vector3d p1 = pred_pos[edges[i][0]];
    Vector3d p2 = pred_pos[edges[i][1]];
    Vector3d n = (p1 - p2) / (p1 - p2).norm();

    double w1 = inv_mass[edges[i][0]];
    double w2 = inv_mass[edges[i][1]];
    double C = (p1 - p2).norm() - restLengthsOfBending[i];

    Vector3d dp1 = -(w1 / (w1 + w2)) * C * n;
    Vector3d dp2 = (w2 / (w1 + w2)) * C * n;

    pred_pos[edges[i][0]] += dp1;
    pred_pos[edges[i][1]] += dp2;
  }
}
```

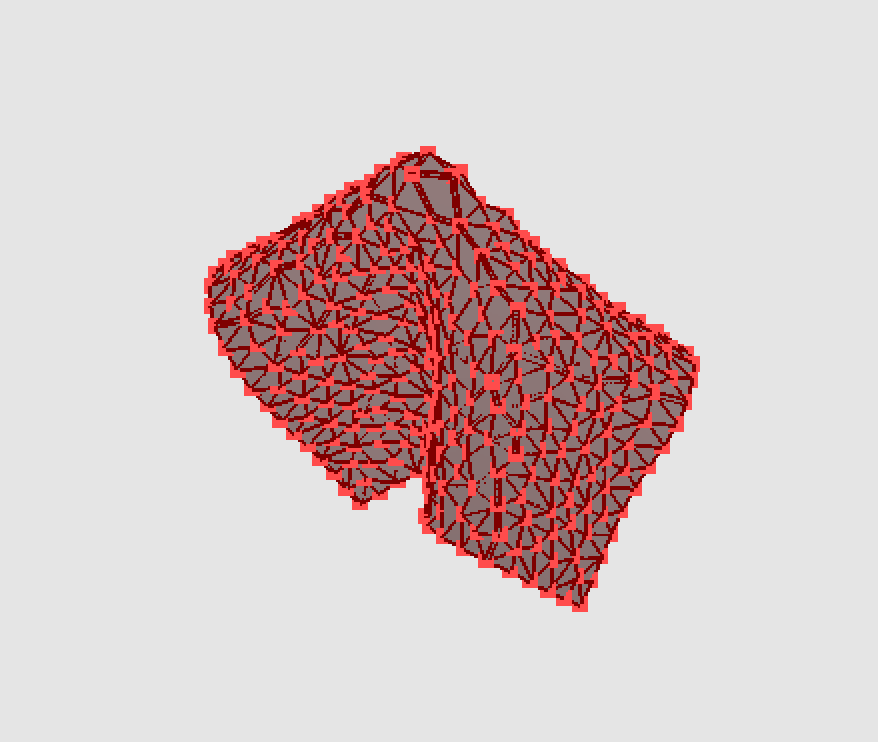
Muller et al. (2007).

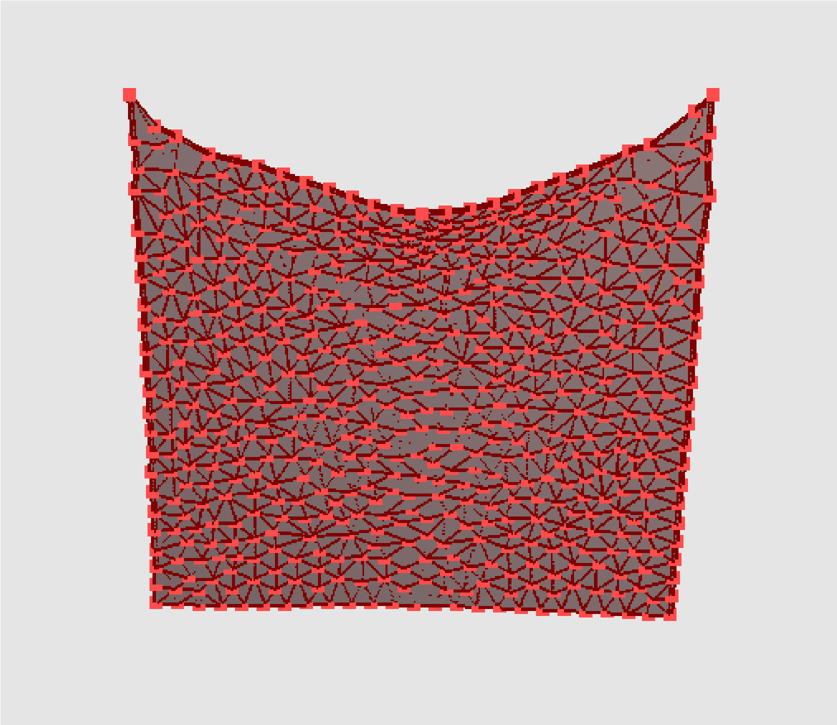
Implementation

```
void ClothSimulator::calBendingConstraint() {
  for (int i = 0; i < bendingElementIdx.size() / 6; i++) {
    int idx1 = bendingElementIdx[6 * i + 3];
    int idx2 = bendingElementIdx[6 * i + 2];
    int idx3 = bendingElementIdx[6 * i + 0];
    int idx4 = bendingElementIdx[6 * i + 1];

    Vector3d p2 = pred_pos[idx2] - pred_pos[idx1];
    Vector3d p3 = pred_pos[idx3] - pred_pos[idx1];
    Vector3d p4 = pred_pos[idx4] - pred_pos[idx1];
    Vector3d n1 = cross(p2, p3) / cross(p2, p3).norm();
    Vector3d n2 = cross(p2, p4) / cross(p2, p4).norm();
    double d = dot(n1, n2);
    if (d < -1) d = -1;

    Vector3d q3 = (cross(p2, n2) + cross(n1, p2) * d) / cross(p2, p3).norm();
    Vector3d q4 = (cross(p2, n1) + cross(n2, p2) * d) / cross(p2, p4).norm();
    Vector3d q2 = - (cross(p3, n2) + cross(n1, p3) * d) / cross(p2, p3).norm() - (cross(p4, n1) + cross(n2, p4) * d) / cross(p2, p4).norm();
    Vector3d q1 = - q2 - q3 - q4;

    double w1 = inv_mass[idx1];
    double w2 = inv_mass[idx2];
    double w3 = inv_mass[idx3];
    double w4 = inv_mass[idx4];

    double s = 4 / (w1 + w2 + w3 + w4);
    double dp = - s * sqrt(1 - d * d) * (acos(d) - restAngles[i]) / (q1.norm() + q2.norm() + q3.norm() + q4.norm());

    pred_pos[idx1] += dp * w1 * q1;
    pred_pos[idx2] += dp * w2 * q2;
    pred_pos[idx3] += dp * w3 * q3;
    pred_pos[idx4] += dp * w4 * q4;
  }
}
```

Muller et al. (2007).

Implementation





Muller et al. (2007).