Position Based Dynamics

Matthias Müller, Bruno Heidelberger, Marcus Hennix, John Ratcliff
3rd Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2006)

Presentation by Daniel Adam
2010/2011

What can you expect?

- To be crushed if you don’t pay attention!

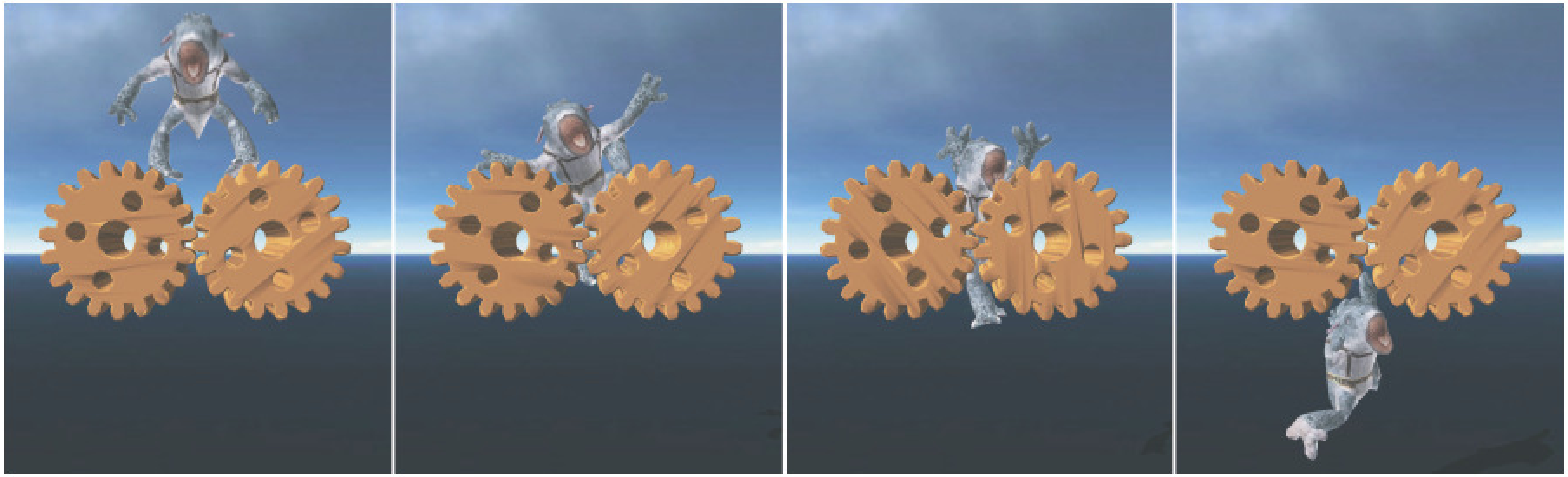


Figure 1: *A known deformation benchmark test, applied here to a cloth character under pressure.*

Overview

- Position based approach to simulation of dynamic systems

- **The Content**
  - Motivation
  - Algorithm
  - Some of the math behind
  - Constraint handling

- **Usage** - Cloth simulation
  - Results

Introduction

- Simulation of physical phenomena such as the dynamics of rigid bodies, deformable objects or fluid flow

- **Computation science:** Accuracy
- **Physical-based animation:** Stability, robustness, speed and visual plausibility

- **Traditional methods** - Force or impulse based
  - Simple explicit methods: Inaccuracy, instability
  - Implicit methods: Large, slow

- **Proposed method** – Position based
  - Directly modify positions

Features and advantages

- Similiar approaches have been used before, but no complete framework has been defined

Position based dynamics:

- gives control over explicit integration
- removes the typical instability problems
- allows direct manipulation of objects and its parts
- allows the handling of general constraints

Representation

- Object representation:
  - dynamic object is represented with a set of N vertices
  - vertex $i \in [1, \ldots, N]$ has a mass m_i, a position x_i and a velocity v_i

- Constraint representation:
  - a cardinality n_j
  - the constraint j is a function $C_j(x): R^{3n_j} \rightarrow R$
  - set of indices
  - stiffness parameter (defines the strength of the constraint)
  - equality constraint j is satisfied if: $C_j(x) = 0$
  - inequality constraint j is satisfied if: $C_j(x) \geq 0$

Algorithm

forall vertices i
initialize $x_i = x_i^0$, $v_i = v_i^0$, $w_i = 1/m_i$
endfor
loop
forall vertices i **do** $v_i = v_i + \Delta t w_i f_{ext}(x_i)$
dampVelocities($v_1, \dots, v_N$)
forall vertices i **do** $p_i = x_i + \Delta t v_i$
forall vertices i **do** generateCollisionConstraints($x_i \rightarrow p_i$)
loop solverIterations **times**
projectConstraints($C_1, \dots, C_{M+Mcoll}; p_1, \dots, p_N$)
endloop
forall vertices i
$v_i = (p_i - x_i) / \Delta t$
$x_i = p_i$
endfor
velocityUpdate($v_1, \dots, v_N$)
endloop

Algorithm description

- **Initialization:**
  - (1)-(3) initialize the state variables.
- **Velocity manipulation:**
  - (5) allows to hook up external forces
  - (6) damps the velocities if necessary
  - (16) the velocities of colliding vertices are modified according to friction and restitution coefficients
- **Constraint manipulation:**
  - (8) generates the M_{coll} collision constraints
  - (10) projects all of the constraints
- **Position based dynamics:**
  - (7) estimates p_i of the vertices are computed using explicit Euler
  - (9)-(11) manipulate these position estimates such that they satisfy the constraints
  - (13-14) vertices are moved to the optimized estimates and the velocities are updated accordingly

Solver

- Input:
  - $M + M_{coll}$ constraints
  - estimates $p_1, \ldots, p_N$

- The solver tries to modify the estimates such that they satisfy all the constraints. The resulting system of equations is non-linear.

- Solution:
  - iterative, similiar to the Gauss-Seidel method
  - the idea is to solve each constraint independently one after the other
  - repeatedly iterate through all the constraints and project the particles to valid locations
  - order of constraints is important

Constraint projection

- moving the points such that they satisfy the constraint
- internal constraints must conserve both linear and angular momentum

- **The Issue:**
  - let us have a constraint with cardinality **n** on the points $p_1, \dots, p_N$ with constraint function **C** and stiffness **k**.
  - let **p** be the concatenation $[p_1^T, \dots, p_N^T]^T$
  - for internal constraints rotating or translating the points does not change the value of the constraint function

- **The Solution:**
  - if the correction **Δp** is chosen to be along the gradient $\nabla_p C(p)$ both momenta are conserved

Constraint projection

- **The Correction:**
  - given **p** we want to find a correction **Δp** such that **C(p+ Δp) = 0 (≥ 0).**
  - approximation: $\mathbf{C(p+\Delta p)} \approx \mathbf{C(p)} + \nabla_p C(p) \,.\, \Delta p = 0$
  - to solve the problem one needs to find a scalar **λ** (lagrange multiplier): $\Delta p = \lambda \nabla_p C(p)$
    - $$\lambda = -\frac{C(p)}{|\nabla_p C(p)|^2}$$
  - solving for **λ** and substituting it into the formula yields the final formula for **Δp:**
    - $$\Delta p = -\frac{C(p)}{|\nabla_p C(p)|^2} \nabla_p C(p)$$

- The result is a non-linear equation, which can be solved iteratively for each point p_i alone

Constraint projection

- For the correction of an individual point p_i we have:
  - $\Delta p_i = -s \nabla_{p_i} C(p_1, \ldots, p_N)$, where **s** is the scaling factor (same for all points)
  - $$s = \frac{C(p_1, \ldots, p_N)}{\sum_j |\nabla_{p_j} C(p_1, \ldots, p_N)|^2}$$
- The methods described so far work if all the points have the same masses

Weighted projection

- If the points have individual masses then the corrections **Δp** must be weighted 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

- Adding the inverse mass to the formula:
  - $\Delta p_i = \lambda \, w_i \nabla_{p_i} C(p)$
  - $s = \dfrac{C(p_1,\dots,p_N)}{\sum_j w_j \, |\nabla_{p_j} C(p_1,\dots,p_N)|^2}$
  - $\Delta p_i = -s \, w_i \nabla_{p_i} C(p_1,\dots,p_N)$

Constraint projection

- Type handling is straightforward:
  - For **equality** constraint always perform a projection
  - For **inequality** constraint perform a projection only when $C(p) < 0$

- Stiffness parameter **k**:
  - simplest variant is to multiply the corrections $\Delta\mathbf{p}$ by $\mathbf{k} \in [0, \ldots, 1]$
  - for multiple iteration loops of the solver, the effect of k is non-linear
  - better solution: multiply by $1-(1-k)^{1/n_s}$ where n_s is the number of iterations
  - resulting material stiffness is applied linearly, but it is still dependent on the time step of the simulation.

Distance constraint

- $C(p_1,p_2) = |p_1 - p_2| - d = 0$
- The gradients:
  - $$\nabla_{p_1} C(p_1, p_2) = \frac{(p_1 - p_2)}{|p_1 - p_2|}$$
  - $$\nabla_{p_2} C(p_1, p_2) = -\frac{(p_1 - p_2)}{|p_1 - p_2|}$$
- The scaling factor **s:**
  - $$s = \frac{|p_1 - p_2| - d}{w_1 + w_2}$$
- Final formula:
  - $$\Delta p_1 = -\frac{w_1}{w_1 + w_2}(|p_1 - p_2| - d)\frac{p_1 - p_2}{|p_1 - p_2|}$$

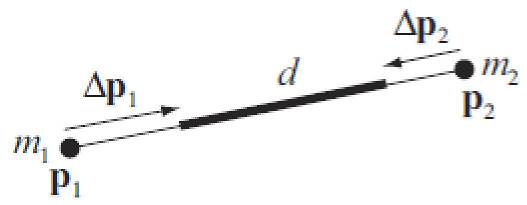
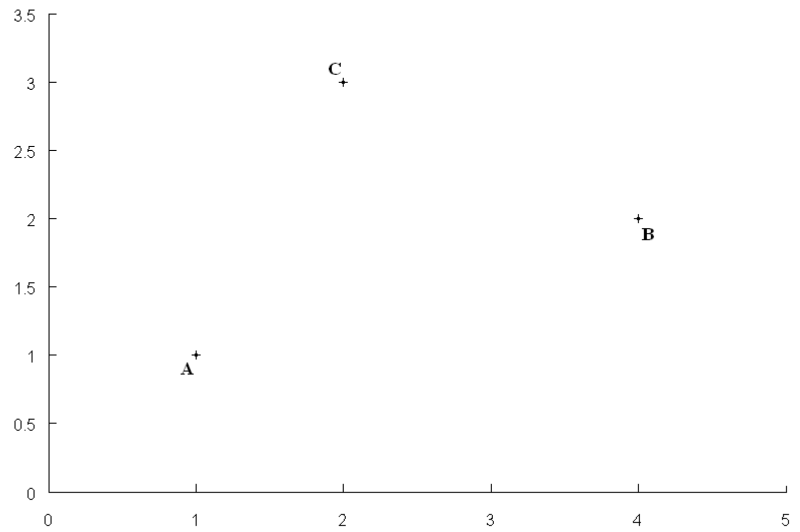


Figure 2: *Projection of the constraint $C(\mathbf{p}_1, \mathbf{p}_2) = |\mathbf{p}_1 - \mathbf{p}_2| - d$. The corrections $\Delta\mathbf{p}_i$ are weighted according to the inverse masses $w_i = 1/m_i$.*

Example – Distance Constraint

- Let us consider a 2D case of 3 vertexes A, B, C bound by 2 distance constraints.
- The parameters:
  - Weights: $m_A = 10$, $m_B = 5$, $m_C = 2$
  - Inverse weights: $w_A = 1/10$, $w_B = 1/5$, $w_C = 1/2$
  - Constraints: $C_1(A, B) = | A – B | - 1$, $C_2(A, C) = | A – C | - 1$
  - New predicted positions: $p_A = [1, 1]$, $p_B = [4, 2]$, $p_C = [2, 3]$
  - Stiffness = 1



Example

- Both constraints are violated: $|A-B|=\sqrt{10}>1$ $|A-C|=\sqrt{5}>1$

- Constraint projection:
  - Lets handle the constraints in order: C_1, C_2
  - Formulas:

$$\Delta p_1 = -\frac{w_1}{w_1 + w_2}(|p_1 - p_2| - d)\frac{p_1 - p_2}{|p_1 - p_2|} \qquad \Delta p_2 = \frac{w_1}{w_1 + w_2}(|p_1 - p_2| - d)\frac{p_1 - p_2}{|p_1 - p_2|}$$

- 1^{st} iteration:
  - C_1 : $\Delta p_A = -\frac{1}{3}(\sqrt{10} - 1)\frac{[-3, -1]}{\sqrt{10}} \cong [0.68, 0.23]$
  - $\Delta p_B = \frac{2}{3}(\sqrt{10} - 1)\frac{[-3, -1]}{\sqrt{10}} \cong [-1.37, -0.46]$

A = [1.68, 1.23], B = [2.63, 1.54]
$|A - B| = 0{,}9986$

  - C_2 : $|A-C| \cong 1.8 > 1$

$$\Delta p_A = -\frac{1}{6}(1.8 - 1)\frac{[-0.32, -1.77]}{1{,}8} \cong [0.02, 0.13]$$

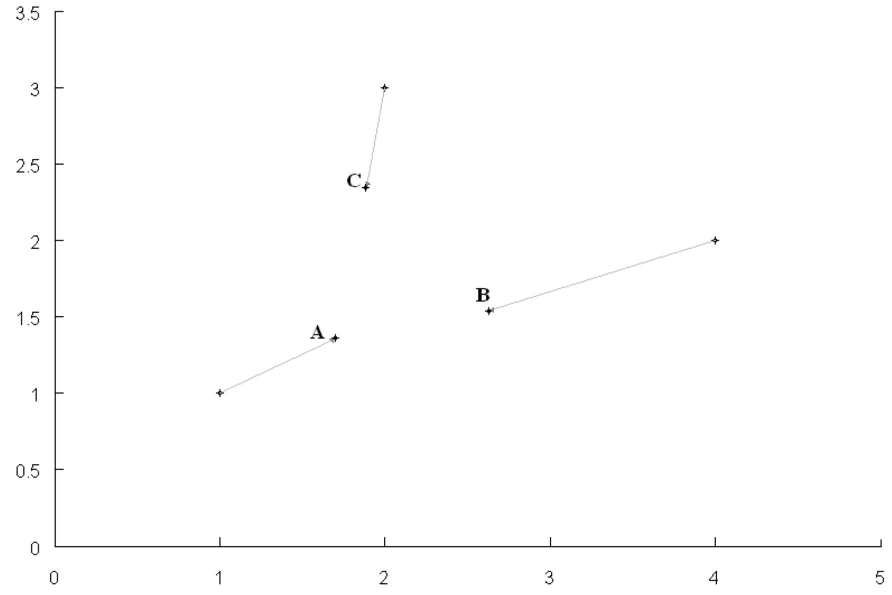
$$\Delta p_C = -\frac{5}{6}(1.8 - 1)\frac{[-0.32, -1.77]}{1{,}8} \cong [-0.12, -0.65]$$

A = [1.7, 1.36], C = [1.88, 2.35]
$|A - C| = 1{,}0125$

Example

- New positions:
  - A = [1.7, 1.36]
  - B = [2.63, 1.54]
  - C = [1.88, 2.35]

- The process is iteratively repeated to get better results



Collision Detection

- Continuous collisions:
  - for each vertex **i** the ray $\mathbf{x}_i \rightarrow \mathbf{p}_i$ is tested if it enters an object
  - compute the entry point $\mathbf{q}_c$ and the surface normal $\mathbf{n}_c$ at this position
  - add a new **inequality** constraint that ensures non-penetration to the list, such constraint has function $\mathbf{C(p)} = (\mathbf{p} - \mathbf{q}_c) \,.\, \mathbf{n}_c \geq \mathbf{0}$ and stiffness $\mathbf{k = 1}$

- Static collisions:
  - compute the surface point $\mathbf{q}_s$ closest to the point $\mathbf{p}_i$ and the surface normal $\mathbf{n}_c$ at this position
  - add add a new **inequality** constraint with $\mathbf{C(p)} = (\mathbf{p} - \mathbf{q}_s) \,.\, \mathbf{n}_s \geq \mathbf{0}$ and stiffness $\mathbf{k = 1}$

- To make the simulation faster, the collision constraint generation is done outside of the solver loop.

Example – Plane Constraint

- Consider a case of a particle (single vertex) that has entered a wall (plane), however the particle is elastic, so it shouldn't penetrate the wall, but bounce off it.

- The parameters:
  - Plane given by three points: A = [1, 0, 0], B = [0, 1, 0], C = [0, 0, 1]
  - Particle X position: $p_X = [0, 0, 0]$
  - Stiffness = 1

- Constraint:
  - $C(p) = (p - q_s) \cdot n_s \geq 0$
  - n_s = normal vector = (1, 1, 1); normalized = $\left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)$
  - q_s = parallel projection of X to the plane = [1/3, 1/3, 1/3]
  - Final form: $C(p_X) = (p_X - [1/3, 1/3, 1/3]) \cdot \left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right) \geq 0$

Example

- Constraint projection:
  - $$\Delta p_i = -s\, w_i \nabla_{p_i} C(p_1,\ldots, p_N) \qquad s = \frac{C(p_1,\ldots, p_N)}{\sum_j w_j \,|\nabla_{p_j} C(p_1,\ldots, p_N)|^2}$$
  - Our case with a single particle:

$$\Delta p_X = -\frac{C(p_X)}{|\nabla_{p_X} C(p_X)|^2} \nabla_{p_X} C(p_X)$$

$$C(p_X) = \left([x, y, z] - \frac{1}{3}(1,1,1)\right) . \left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right) = \frac{1}{\sqrt{3}}(x + y + z - 1)$$

$$\nabla_{p_X} C(p_X) = \left(\frac{\partial C(p_X)}{\partial x}, \frac{\partial C(p_X)}{\partial y}, \frac{\partial C(p_X)}{\partial z}\right) = \left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)$$

$$|\nabla_{p_X} C(p_X)|^2 = 1$$

Example

$$\Delta p_X = -\frac{1}{\sqrt{3}}(x+y+z-1).\left(\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}}\right)$$

- Solution:
  - **X = [0, 0, 0]**
  - $\Delta p_X = -\frac{1}{\sqrt{3}}(-1).\left(\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}}\right) = \left(\frac{1}{3},\frac{1}{3},\frac{1}{3}\right)$
  - New position: **X =** $\left(\frac{1}{3},\frac{1}{3},\frac{1}{3}\right)$

Collision Detection

- Friction and restitution can be handled by manipulating the velocities of colliding vertices in step (16) of the algorithm
- The described collision handling is only correct for collisions with static objects, because no impulse is transferred to the collision partners

- Multiple colliding objects:
  - Correct response for multiple colliding objects can be achieved by simulating all objects with the simulator
  - the **N** vertices and **M** constraints which are the input to the algorithm simply represent two or more independent objects.

Collision Detection

- Lets consider a case of two dynamic objects
  - Let **q** be a point of the first object
  - Let $\mathbf{p_1, p_2, p_3}$ be a triangle of the second object

- Example: Point **q** enters the triangle $\mathbf{p_1, p_2, p_3}$
  - the algorithm inserts an **inequality** constraint with constraint function
    $\boldsymbol{C}(\mathbf{q, p_1, p_2, p_3}) = \pm (\mathbf{q} - \mathbf{p_1}) \,.\, [(\mathbf{p_2} - \mathbf{p_2}) \times (\mathbf{p_3} - \mathbf{p_1})]$
  - this keeps the point q on the correct side of the triangle

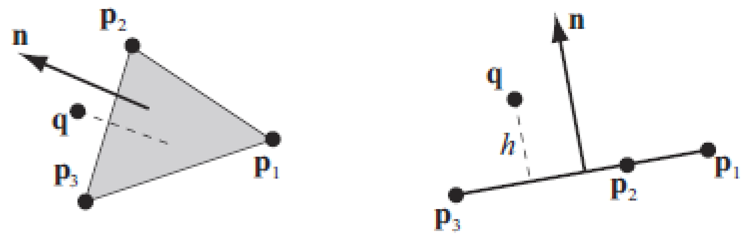


Figure 5: *Constraint function* $C(\mathbf{q}, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3) = (\mathbf{q} - \mathbf{p}_1) \cdot \mathbf{n} - h$ *makes sure that* **q** *stays above the triangle* $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ *by the the cloth thickness h.*

Damping

- the velocities are dampened before they are used for the prediction of the new positions
- local deviations from the global motion is dampened

- Proposed method:

  (1) **forall** vertices i

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

  (3) $v_i \leftarrow v_i + k_d \Delta v_i$

  (4) **endfor**

- The variables:
  - $p_{cm} = (\Sigma_i p_i m_i) / (\Sigma_i m_i)$
  - $v_{cm} = (\Sigma_i v_i m_i) / (\Sigma_i m_i)$ (velocity due to global body motion)
  - $r_i = p_{cm} - p_i$
  - $L = \Sigma_i r_i \times (m_i v_i)$
  - $J = \Sigma_i (r^x_i)(r^x_i)^T m_i$, where r^x_i is the cross product matrix
  - $\omega = J^{-1} L$

Attachments

- Attaching vertices to static or kinematic objects

- How to model it:
  - position of the vertex is simply set to the static target position
  - alternatively update the position 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

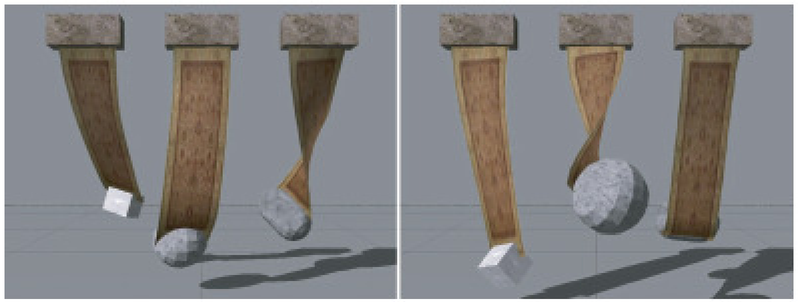


Figure 8: *Cloth stripes are attached via one way interaction to static rigid bodies at the top and via two way constraints to rigid bodies at the bottom.*

Cloth Simulation

- the position based dynamics framework has been used to implement a real time cloth simulator for games

- Representation of cloth:
  - simulator accepts as input arbitrary triangle meshes
  - the input mesh must represent a 2-manifold
  - each node of the mesh becomes a simulated vertex
  - user inputs cloth density and thickness, which are used to calculate the mass of each triangle
  - the mass of a vertex is set to the sum of one third of the mass of each adjacent triangle
  - constraints are defined along edges and faces

Constraints

- Stretching constraints:
  - generated for each edge
  - $\mathbf{C_{stretch}(p_1, p_2) = |p_1 - p_2| - l_0 = 0}$
  - $\mathbf{l_0}$ is the initial length of the edge
  - the stiffness parameter $\mathbf{k_{stretch}}$ is set by the user

- Bending constraints:
  - generated for each pair of adjacent triangles $(\mathbf{p_1, p_3, p_2})$ and $(\mathbf{p_1, p_2, p_4})$
  - $$C_{bend}(p_1, p_2, p_3, p_4) = \operatorname{acos}\left(\frac{(p_2 - p_1) \times (p_3 - p_1)}{|(p_2 - p_1) \times (p_3 - p_1)|} \cdot \frac{(p_2 - p_1) \times (p_4 - p_1)}{|(p_2 - p_1) \times (p_4 - p_1)|}\right) - \varphi_0$$
  - $\boldsymbol{\varphi_0}$ is the initial dihedral angle between the two triangles
  - the stiffness parameter $\mathbf{k_{bend}}$ is set by the user

Cloth simulation

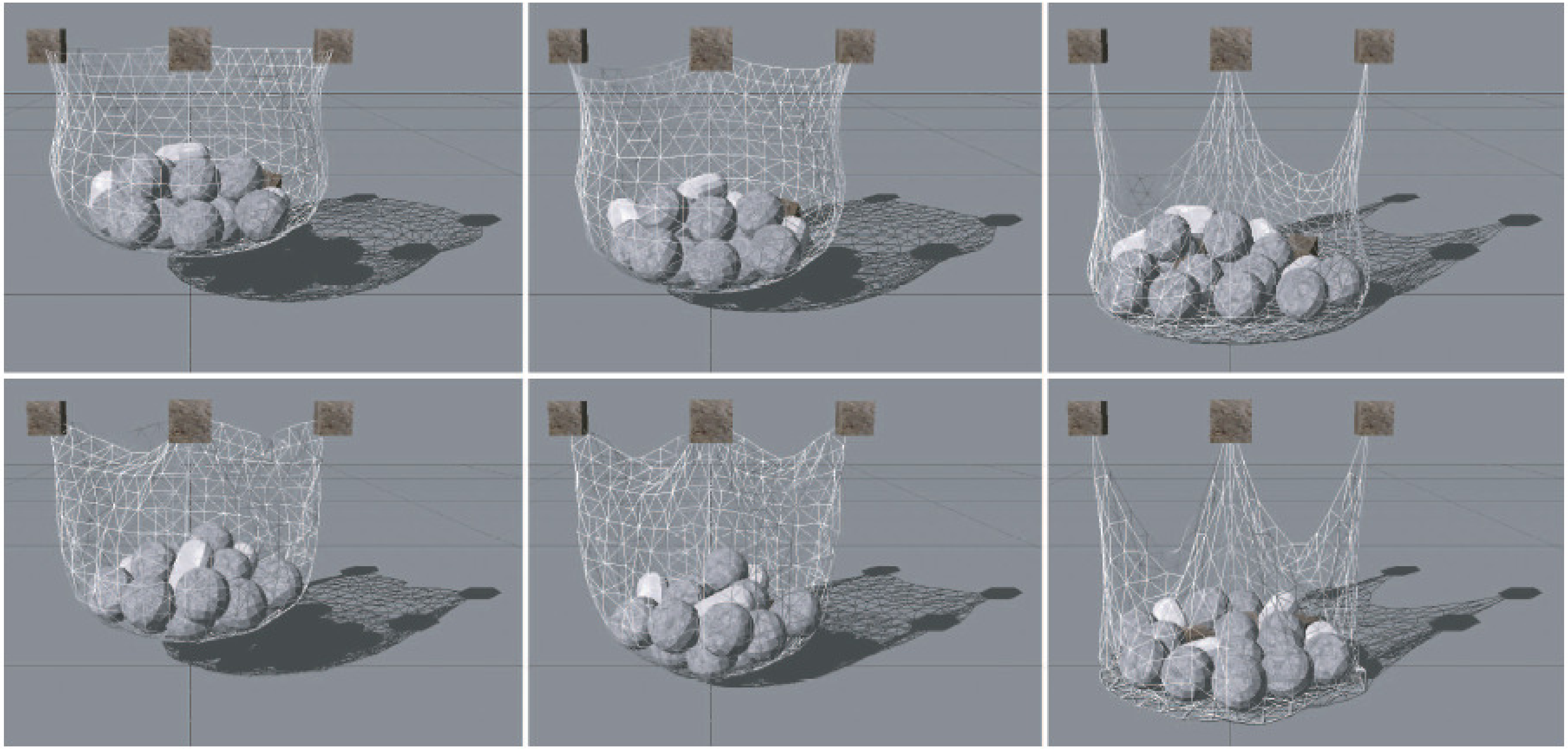


Figure 3: *With the bending term we propose, bending and stretching are independent parameters. The top row shows* $(k_{stretching}, k_{bending}) = (1,1)$, $(\frac{1}{2},1)$ *and* $(\frac{1}{100},1)$. *The bottom row shows* $(k_{stretching}, k_{bending}) = (1,0)$, $(\frac{1}{2},0)$ *and* $(\frac{1}{100},0)$.

Collisions

- Collision with rigid bodies:
  - to get two-way interactions an impulse $\mathbf{m_i \Delta p_i / \Delta t}$ is applied at the contact point each time the vertex i is projected due to collision
- Self-collisions:
  - assume the triangles all have about the same size and use spatial hashing to find vertex triangle collisions
  - if a vertex **q** moves through a triangle $\mathbf{p_1, p_2, p_3}$, use the constraint function:
    - $C(\mathbf{q}, \mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}) = \pm(\mathbf{q} - \mathbf{p_1}) \,.\, [(\mathbf{p_2} - \mathbf{p_2}) \times (\mathbf{p_3} - \mathbf{p_1})] - \mathbf{h}$ (h is cloth thickness)
  - testing continuous collisions is insufficient if cloth gets into a tangled state

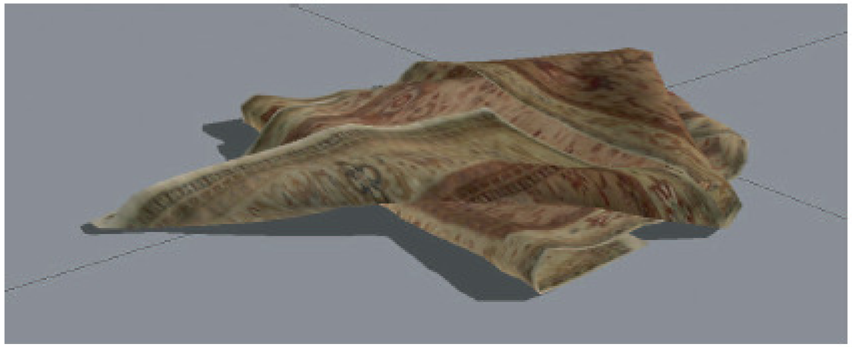


Figure 6: *This folded configuration demonstrates stable self collision and response.*

Cloth Balloons

- For closed triangle meshes, overpressure inside the mesh can easily be modeled

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

- The model:
  - an **equality** constraint concerning all **N** vertices of the mesh
  - compute the actual volume of the closed mesh and compare it against the original volume $\mathbf{V_0}$ times the overpressure factor $\mathbf{k_{pressure}}$
  - $$C(p_1,\ldots,p_N)=\left(\sum_{i=1}^{n_{triangles}}(p_{t_1^i}\times p_{t_2^i}).p_{t_3^i}\right)-k_{pressure}V_0$$
  - t_1^i, t_2^i, t_3^i are the three indices of the vertices belonging to triangle i

Cloth Tearing

- Tearing is simulated by a simple process:
  - When the stretching of an edge exceeds a threshold, select one of the adjacent vertices
  - 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
  - All triangles below are assigned to the new vertex
  - Method remains stable even in extreme situations

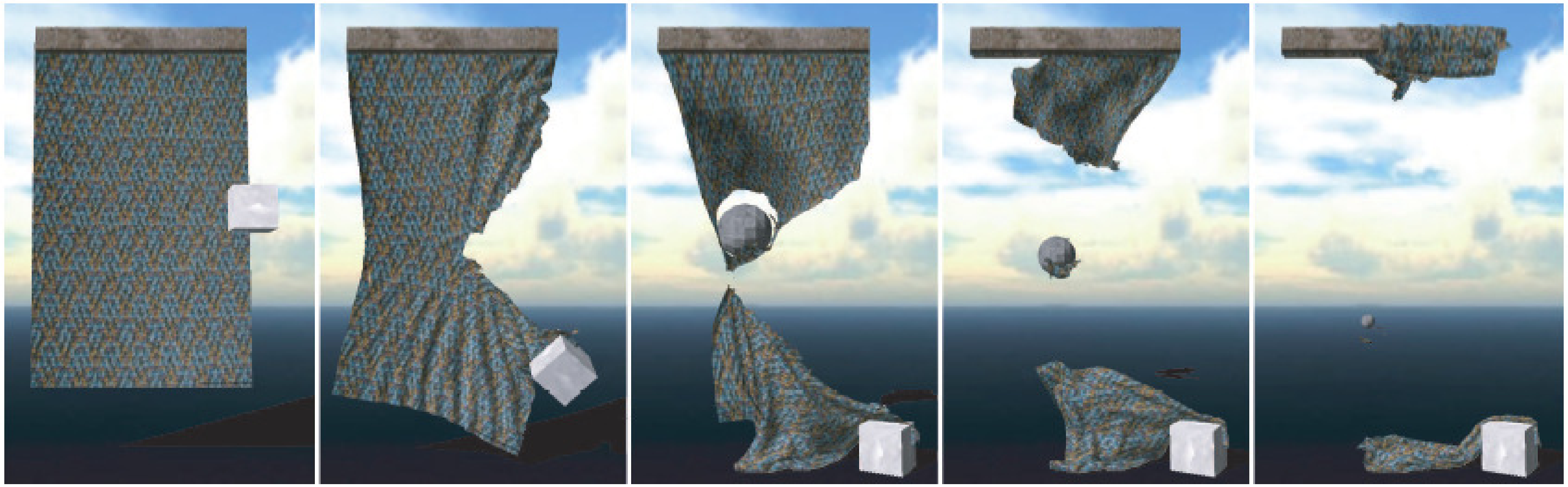


Figure 10: *A piece of cloth is torn open by an attached cube and ripped apart by a thrown ball.*

Conclusions

- 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.
- It significantly simplifies the handling of collisions, attachment constraints and explicit integration and it makes direct and immediate control of the animated scene possible.
- The approach presented could quite easily be extended to handle rigid objects as well

Eye Candy

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

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

Eye Candy

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).*

The End

Thank you for your attention.