

Position Based Dynamics

Zhipeng Ding

Department of Computer Science
University of North Carolina at Chapel Hill

COMP768, Dec. 2, 2016

Outline

- 1 Introduction
- 2 Algorithm
- 3 Solver
- 4 Specific Constraints
- 5 Implementation
- 6 Limitations

Outline

- 1 Introduction
- 2 Algorithm
- 3 Solver
- 4 Specific Constraints
- 5 Implementation
- 6 Limitations

- Force Based Methods
 - Internal and external forces are accumulated, Newton's second law
 - Employ a time integration method, update velocities and finally positions
- Impulse Based Methods
 - Directly manipulate velocities, one layer of integration can be skipped
- Position Based Methods
 - Omit the velocity layer as well and immediately works on the positions
 - Define general constraints via a constraint function
 - Directly solve for the equilibrium configuration and project positions

Introduction (Continue)

- Main Usage
 - Computer games: it is often desirable to have direct control over positions of objects or vertices of a mesh
- Main features and advantages
 - Position based simulation gives control over explicit integration and removes the typical instability problems
 - Positions of vertices and parts of objects can directly be manipulated during the simulation
 - The formulation allows the handling of general constraints in the position based setting
 - The explicit position based solver is easy to understand and implement

Outline

- 1 Introduction
- 2 Algorithm**
- 3 Solver
- 4 Specific Constraints
- 5 Implementation
- 6 Limitations

- **Main idea:** instead of forces - **constraints**; instead of integration - **constraint projection**
- The dynamic object is represented 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 $k_j \in [0, 1]$ 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 $C_j(\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_{n_j}}) \geq 0$.

Algorithm (Continue)

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

- Damping

- The quality of dynamic simulations can generally be improved by the incorporation of an appropriate damping scheme.
- Generally, a damping term $C\dot{X}$ can be incorporated into the motion equation of an object where \dot{X} denotes the vector of all first time derivatives of positions
- In order to preserve linear and angular momentum of deformable objects, symmetric damping forces, usually referred to as spring damping forces, can be used

Outline

- 1 Introduction
- 2 Algorithm
- 3 Solver**
- 4 Specific Constraints
- 5 Implementation
- 6 Limitations

- **Unified Solver:** Everything is a set of particles connected by constraints
- Simplify collision detection; two-way interaction of all objects type; fits well on the GPU

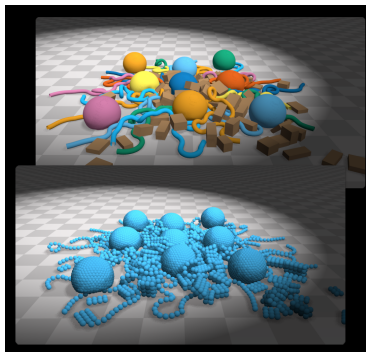


Figure: Particles [Macklin]

Solver (Continue)

- The goal of step (8)-(10) in previous algorithm is to correct the predicted positions of the particles such that they satisfy all constraints
- The problem that needs to be solved comprises of a set of M equations for $3N$ unknown position components
 - $M \geq 3N$: over-determined, $M \leq 3N$: under-determined
 - The equations in general are non-linear
 - Collisions produce inequalities rather than equalities
- Solve a non-symmetric, non-linear system with equalities and inequalities is a TOUGH problem

Solver (Continue)

- Let \mathbf{x} be the concatenation $[\mathbf{x}_1^T, \dots, \mathbf{x}_N^T]$
- Let all the constraint function C_j take the concatenated vector \mathbf{x} as input while only using the subset of coordinates they are defined for
- The system to be solved is

$$C_1(\mathbf{x}) \succ 0$$

...

$$C_M(\mathbf{x}) \succ 0$$

where the symbol \succ denotes either $=$ or \geq

Solver (Continue)

- Each constraint function is then linearized in the neighborhood of the current solution using

$$C(\mathbf{x} + \Delta\mathbf{x}) = C(\mathbf{x}) + \nabla C(\mathbf{x}) \cdot \Delta\mathbf{x} + O(|\Delta\mathbf{x}|^2) = 0$$

- This yields a linear system for the global correction vector $\Delta\mathbf{x}$

$$\nabla C_1(\mathbf{x}) \cdot \Delta\mathbf{x} = -C_1(\mathbf{x})$$

...

$$\nabla C_M(\mathbf{x}) \cdot \Delta\mathbf{x} = -C_M(\mathbf{x})$$

- Then use a non-linear Gauss-Seidel-type iteration and solve each constraint equation separately

Solver (Continue)

- The problem of the system being under-determined is solved by restricting $\Delta \mathbf{x}$ to be in the direction of ∇C which is also a requirement for linear and angular momentum conservation
- This means choosing a scalar λ such that

$$\Delta \mathbf{x} = \lambda \nabla C(\mathbf{x})$$

- Hence the formula for $\Delta \mathbf{x}$

$$\Delta \mathbf{x} = -\frac{C(\mathbf{x})}{|\nabla C(\mathbf{x})|^2} \nabla C(\mathbf{x})$$

- If the points have individual masses, then weight the corrections $\Delta \mathbf{x}$ 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 as expected

- Thus, the correction vector of a single particle i is

$$\Delta \mathbf{x}_i = -\lambda w_i \nabla_{\mathbf{x}_i} C(\mathbf{x})$$

where

$$\lambda = \frac{C(\mathbf{x})}{\sum_j w_j |\nabla_{\mathbf{x}_j} C(\mathbf{x})|^2}$$

Solver (Continue)

- To deal with stiffness k , the simplest way is to multiply the corrections $\Delta\mathbf{x}$ by k
- 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 $\Delta\mathbf{x}(1 - k)^{n_s}$
- To get a linear relationship, multiply the corrections not by k directly but by $k' = 1 - (1 - k)^{1/n_s}$
- With this transformation the error becomes $\Delta\mathbf{x}(1 - k')^{n_s} = \Delta\mathbf{x}(1 - k)$ and thus becomes linearly dependent on k and independent of n_s

Outline

- 1 Introduction
- 2 Algorithm
- 3 Solver
- 4 Specific Constraints**
- 5 Implementation
- 6 Limitations

- Constraint types
 - Distance (clothing)
 - Shape (rigid bodies, plastics)
 - Density (fluids)
 - Volume (inflatables)
 - Contact (non-penetration)
- Combine constraints
 - Melting, phase-changes
 - Stiff cloth, bent metal

Example: Cloth Simulation

- For simplicity, define $\mathbf{x}_{i,j} = \mathbf{x}_i - \mathbf{x}_j$
- Stretching

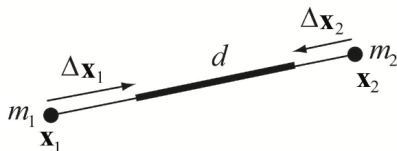


Figure: Stretching [Bender 15]

- Distance constraint function $C(\mathbf{x}_1, \mathbf{x}_2) = |\mathbf{x}_{1,2}| - d$

Example: Cloth Simulation

- The derivatives with respect to the points are

$$\nabla_{\mathbf{x}_1} C(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{n} \quad \& \quad \nabla_{\mathbf{x}_2} C(\mathbf{x}_1, \mathbf{x}_2) = -\mathbf{n}$$

where $\mathbf{n} = \frac{\mathbf{x}_{1,2}}{|\mathbf{x}_{1,2}|}$

- The scaling factor λ is, thus,

$$\lambda = \frac{|\mathbf{x}_{1,2}| - d}{\omega_1 + \omega_2}$$

- and the final corrections

$$\Delta \mathbf{x}_1 = -\frac{\omega_1}{\omega_1 + \omega_2} (|\mathbf{x}_{1,2}| - d) \mathbf{n}$$

$$\Delta \mathbf{x}_2 = +\frac{\omega_2}{\omega_1 + \omega_2} (|\mathbf{x}_{1,2}| - d) \mathbf{n}$$

Example: Cloth Simulation (Continue)

- Bending

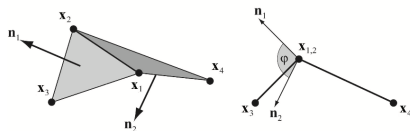


Figure: Bending [Bender 15]

- Bilateral bending constraint function

$$C_{bend}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = \arccos\left(\frac{\mathbf{x}_{2,1} \times \mathbf{x}_{3,1}}{|\mathbf{x}_{2,1} \times \mathbf{x}_{3,1}|} \cdot \frac{\mathbf{x}_{2,1} \times \mathbf{x}_{4,1}}{|\mathbf{x}_{2,1} \times \mathbf{x}_{4,1}|}\right) - \varphi_0$$

Example: Cloth Simulation (Continue)

- Self Collision

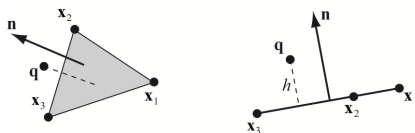


Figure: Collision [Bender 15]

- Constraint function $C(\mathbf{q}, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) = (\mathbf{q} - \mathbf{x}_1) \cdot \frac{\mathbf{x}_{2,1} \times \mathbf{x}_{3,1}}{|\mathbf{x}_{2,1} \times \mathbf{x}_{3,1}|} - h$

Example: Cloth Simulation (Continue)

- Pressure



Figure: Simulation of overpressure inside a character [Muller 06]

- Constraint function

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

Here t_1^i , t_2^i and t_3^i are the three indices of the vertices belonging to triangles i

Example: Cloth Simulation (Continue)

- **Video** Position Based Dynamics
- There are also other interesting examples that have more constraints to explore, but the PBD framework is the same. So, we stop here to discuss other issues

Outline

- 1 Introduction
- 2 Algorithm
- 3 Solver
- 4 Specific Constraints
- 5 Implementation**
- 6 Limitations

- In a single CPU implementation, the solver processes the constraints one by one in a Gauss-Seidel-type fashion
- However, GPU based parallelization is more preferable
 - Graph-Coloring Methods
 - Jacobi Methods
 - Hybrid Methods

- Graph-Coloring Methods
 - Need to split the constraints into groups or phases. In each phase, none of the constraints are allowed to share a common particle
 - In general case, splitting constraints into phases corresponds to the graph coloring problem, where each constraint corresponds to a node of the graph and two constraints are connected by an edge if they affect one or more common particles
 - The minimum number of colors determines how many phases are needed in the parallel execution of PBD.
 - Keeping the number of phases small is not the only optimization criterion. The set also need to have similar size for good load balancing

Implementation (Continue)

- Jacobi Methods

- In a Jacobi solver, each constraint may be processed in parallel, and the position delta for each particle obtained by summing the delta from each constraint at the end of an iteration. But, converge significantly slower and may not converge at all
- **Constraint averaging:**

$$\Delta \tilde{\mathbf{x}}_i = \frac{1}{n_i} \sum \Delta \mathbf{x}_i$$

- Introduce a global user-parameter ω to control the rate of successive over-relaxation (SOR)

$$\Delta \tilde{\mathbf{x}}_i = \frac{\omega}{n_i} \sum \Delta \mathbf{x}_i$$

recommand using $1 \leq \omega \leq 2$

- Hybrid Methods

- To take advantage of both Gauss-Seidel and Jacobi solvers, Fratarcangeli et al. proposed a hybrid approach
- Use graph coloring and modify the graph such that it produces a desired number of k colors by splitting high valence particles, i.e. solving them Jacobi style

Outline

- 1 Introduction
- 2 Algorithm
- 3 Solver
- 4 Specific Constraints
- 5 Implementation
- 6 Limitations**

- The dependency of the results on the amount of solver iterations

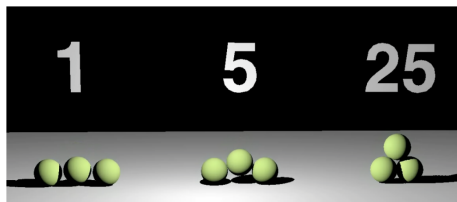


Figure: The free-flowing particles fall onto each other, three different times [Bartels 15]

Limitations (Continue)

- One-frame delay in collision detection due to the collision detection happening in between the external forces and the constraint solve

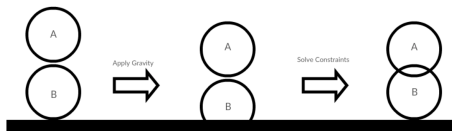


Figure: Two particles, situated above each other but not touching, fall onto a floor [Bartels 15]

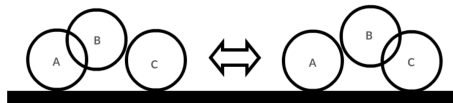


Figure: The one-frame delay pushes particle B back and forth between A and C [Bartels 15]

Limitations (Continue)

- A problem inherent to the Jacobi iterations is the constraint averaging. However, as explained by Macklin et al. (2014), it is almost impossible to efficiently parallelize the Gauss-Seidel version of PBD

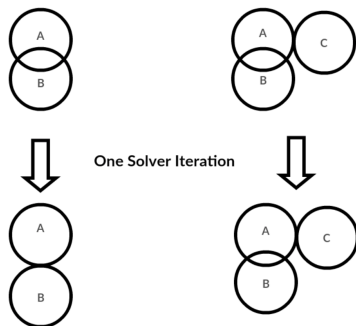









Figure: Constraint averaging makes the amount of necessary solver iterations depend on the amount of constraints [Bartels 15]

For Further Reading I

-  Mller, M., Heidelberger, B., Hennix, M., and Ratcliff, J. (2007). Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2), 109-118. Chicago
-  Mller, M. (2008). Hierarchical position based dynamics.
-  Bender, J., Mller, M., Otaduy, M. A., and Teschner, M. (2013). Position-based methods for the simulation of solid objects in computer graphics. *Eurographics*.
-  Macklin, M., and Mller, M. (2013). Position based fluids. *ACM Transactions on Graphics (TOG)*, 32(4), 104.
-  Bender, J., Mller, M., Otaduy, M. A., Teschner, M., and Macklin, M. (2014, September). A Survey on Position-Based Simulation Methods in Computer Graphics. In *Computer graphics forum* (Vol. 33, No. 6, pp. 228-251).

For Further Reading II

-  Bender, J., Muller, M., and Macklin, M. (2015). Position-based simulation methods in computer graphics. EUROGRAPHICS Tutorial Notes.
-  Bartels, P. (2015). Position Based Dynamics.