XPBD: Position-based simulation of compliant constrained dynamics

SAMMY RASAMIMANANA, Telecom Paris

Many physical simulations are governed by Newton's laws. Due to the complexity of some forces, the computation time of the numerical solution can be long, which is not practical for real-time applications, such as in video games. Another approach is to do a position-based simulation, and this is the technique used in position-based dynamics (PBD). The basic idea is, for a set of vertices, to move them in such a way that certain constraints are satisfied. Although this is a simple and fast method to animate objects, it is not physically accurate and has some notable flaws. One of these problems is the dependence of the stiffness on the number of substeps in the Gauss-Seidel method used to solve the system. In this project, we explore XPBD, an extended version of PBD that addresses this stiffness dependence problem. After interpreting the geometric meaning of some constraints, we tried to implement them in order to render and simulate some simple scenes.

Additional Key Words and Phrases: physics simulation, animation, constraints, stiffness, position-based, gradient

1 INTRODUCTION TO POSITION-BASED SIMULATION

One way to make physics simulation consists in the Euler integration based on Newton's second law. If a particle has a mass m, a position p, a speed v, and is subject to an external force F, then its motion must follow the classic equation $m\ddot{p} = F$. This equation is often numerically solved using semi-implicit Euler integrator

$$v_{n+1} = v_n + \frac{F}{m}\Delta t$$
$$p_{n+1} = p_n + v_{n+1}\Delta t$$

However, instead of relying on forces to simulate physics like above, the postion-based dynamics (PBD) is a method that directly control the position of particles to animate them [Müller et al. 2007].

1.1 Position-Based Dynamics

Let's consider that a dynamic object is a set of N vertices (which will be treated as particles) and M constraints.

Definition 1.1. A constraint $j \in [\![1, M]\!]$ consists of

- a cardinality n_j and a set of indices $i_1, \cdots, i_{n_j} \subset [\![1, N]\!]$
- a function $C_j : \mathbb{R}^{3n_j} \longrightarrow \mathbb{R}$
- a stiffness parameter $k_j \in [0, 1]$
- a type of either *equality* ($C_j = 0$) or *inequality* ($C_j \geqslant 0$)

The idea of the PBD algorithm is simple: we let the system evolves and correct at each step the position of the particles using the gradient of each constraint, so that all of those constraints are satisfied or approximately satisfied.

Algorithm 1 PBD simulation loop

```
p, v, f_ext : vectors containing all the particles (of mass m) positions, speeds and external forces
p_old ⟵ p
v ⟵ v + f_ext · m^{-1} · dt
p ⟵ p + v · dt
while i < nbIterations do
    for all constraints C do
        Δp ⟵ k_j · (−C(p)) / ‖∇C(p)‖² · ∇C^T(p)
        p ⟵ p + Δp
    end for
    i ⟵ i + 1
end while
v ⟵ (p − p_old)/dt
```

Pros. This method not only has the advantage of directly manipulating the positions of the vertices and the objects which makes it very fast, but it also gives control over explicit integration and removes typical instability problems (it is unconditionnally stable). Moreover, non-linear constraints can be handled since we just need to know their gradients.

Cons. However, the PBD method doesn't rely on physics laws, so the result does not have any physcial meaning and can be very inaccurate. Furthermore, one of its main flaw is the sitffness of the model which depends on the number of sub-iterations and the time step when moving along the gradient.

1.2 XPBD, an extension to handle stiffness

To remove this stiffness dependency, [Macklin et al. 2016] tried another point of view that relies on energy potential. Let p be the vector containing all the particles' positions. Given a certain constraint $C(p)$, let $U(p) = \frac{1}{2}C(p)^T\alpha^{-1}C(p)$ its associated energy potential. Using Newton's second law, we obtain

$$M\ddot{p} = -\nabla_p U^T = -\nabla_p C^T(p)\alpha^{-1}C(p)$$

Then, we discretize the problem with a time step Δt and introduce the Lagrange multiplier $\lambda = -\tilde{\alpha}^{-1}C(p)$ where $\tilde{\alpha} = \frac{\alpha}{\Delta t^2}$ so that the equation becomes the following system at step n:

$$M(p_{n+1} - \tilde{p}) - \nabla C^T(p_{n+1})\lambda_{n+1} = 0 \quad (1)$$
$$C(p_{n+1}) + \tilde{\alpha}\lambda_{n+1} = 0 \quad (2)$$

where $\tilde{p} = p_n + v_n\Delta t$ is the predicted position. After linearizing (1) and (2) regarding to p_{n+1} and λ_{n+1}, using $f(x+\Delta x) = f(x) + \Delta x\frac{\partial f}{\partial x}$, we got the following matrix equation:

$$\begin{pmatrix} \frac{\partial(M(p-\tilde{p}) - \nabla C^T(p)\lambda)}{\partial p} & -\nabla C^T(p_n) \\ \nabla C^T(p_n) & \tilde{\alpha} \end{pmatrix} \begin{pmatrix} \Delta p \\ \Delta\lambda \end{pmatrix} = \begin{pmatrix} M(p_n - \tilde{p}) - \nabla C^T(p_n)\lambda_n \\ C(p_n) + \tilde{\alpha}\lambda_n \end{pmatrix}$$

This report is submitted as a part of project for Advanced 3D Computer Graphics (IMA904/IG3DA), Telecom Paris.

The original work is introduced by [Macklin et al. 2016].

From this, two approximations are introduced to make the resolution much simpler:

omission of the geometry stiffness: the computation of the quantity $\frac{\partial(M(p-\tilde{p})-\nabla C^T(p)\lambda)}{\partial p}$ can be time consuming, so we approximate it with the mass M of the particles, thus discarding the geometry stiffness and leading to an error of order $O\left(\Delta t^2\right)$

locality of the solution: equation (1) starts with the predicted position $p_0 = \tilde{p}$, so if we set $\lambda_0 = 0$ then it is verified. Assuming the predicted position is really close to the true solution, the gradient would change very slowly so that at each step we can consider $M(p_n - \tilde{p}) - \nabla C^T(p_n)\lambda_n = 0$.

The solutions are eventually

$$\Delta\lambda = \frac{-C(p_n) - \tilde{\alpha}\lambda_n}{\nabla C(p_n)M^{-1}\nabla C^T(p_n) + \tilde{\alpha}} \tag{3}$$

$$\Delta p = M^{-1}\nabla C^T(p_n)\Delta\lambda \tag{4}$$

and are connected back to PBD using a Gauss-Seidel method. Those formulas can be extended to consider k constraints by writing $C(p) = (C_1(p), \cdots, C_k(p))^T$.

Algorithm 2 XPBD simulation loop

p, v, f_{ext} : vectors containing all the particles (of mass m) positions, speeds and external forces
$\tilde{\alpha} \longleftarrow \frac{\alpha}{dt^2}$
$p_{old} \longleftarrow p$
$v \longleftarrow v + f_{ext} \cdot m^{-1} \cdot dt$
$p \longleftarrow p + v \cdot dt$
Initialize multipliers $\lambda \longleftarrow 0$
while $i <$ `nbIterations` **do**
  for all constraints C **do**
    $\Delta\lambda \longleftarrow \frac{-C(p)-\tilde{\alpha}\lambda}{m^{-1}\left\|\nabla C(p)\right\|^2+\tilde{\alpha}}$
    $\Delta p \longleftarrow m^{-1}\nabla C^T(p)\Delta\lambda$
    $\lambda \longleftarrow \lambda + \Delta\lambda$
    $p \longleftarrow p + \Delta p$
  end for
  $i \longleftarrow i + 1$
end while
$v \longleftarrow (p - p_{old})/dt$

2 SIMULATOR

Our simulator was built based on some scenes we wanted to setup. One scene can contain one or multiple objects, as well as different types of constraints.

2.1 Implementation structure

We divided our structure in two steps, both based on oriented object program and shared pointers in C++. The mathematical operations are mainly handled with the `Eigen` library.

2.1.1 Object class. We designed a generic class `Object` with mass, position, speed and color attributes. Those attributes will be used to set and render the object in the space, but are not going to be updated. Instead, according to section 1.1, each object will have a `std::vector` of `Particle` which will store its vertices. Those vertices are the ones that are going to see their position and speed updated to move the object.

2.1.2 Constraint class. The `Constraint` is also a generic class. It contains all the main quantities for a constraint to execute the XPBD loop, such as its value, its gradient, its type (`EQUALITY` or `INEQUALITY`), its compliance α, the lagrange multiplier λ and the particles it will affect. Then, each different subclass of `Constraint` defines a particular constraint with its own intermediate quantities and gradient.

2.2 Implemented constraints

We present in this section all the constraints we implemented for our scenes, as well as our geometric interpretation for them. Table 1 sums up the different formulas. Some of the gradients where a little bit complex to calculate due to cross product, but we managed to get usable expressions.

2.2.1 Distance Constraint. The goal of the distance constrained is to maintain two particles at distance l_0. It gives a similar effect than a spring between those two particles.

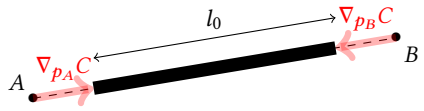


Fig. 1. Distance constraint illustration

2.2.2 Fixed Constraint. It makes a particle staying at a given position p_0. We could have implemented it as a ghost distance constraint with a ghost particle or a fixed position but since we know where the particle should stay, we set p to p_0 at each step of the solver.

2.2.3 Collision Constraint. It is basically the inequality version of the distance constraint, and allows us to avoid the collision between two particles.

2.2.4 Penetration Constraint. This constraint prevents a vertex q from entering the triangle formed by particles (p_1, p_2, p_3) if q comes from the half-plane pointed by the normal n. It can be used to model mesh-mesh collisions or particle-mesh collisions.

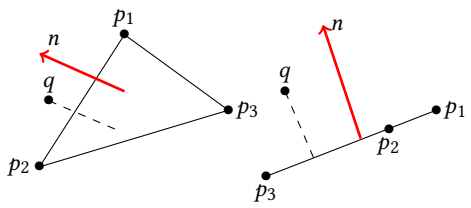


Fig. 2. Penetration constraint illustration

2.2.5 Wall Constraint. It is the particular case of the penetration constraint when (p_1, p_2, p_3) are not particles but static points of a given plane that acts like a wall or a floor.

2.2.6 Bending Constraint. Instead of adding a diagonal distance constraint in the quad (p_1, p_2, p_3, p_4), we constraint the two half-triangles to keep the same angle φ_0. This way, this bending term is somehow independent of stretching, allowing for instance to model cloth with low stretching stiffness but high bending resistance.

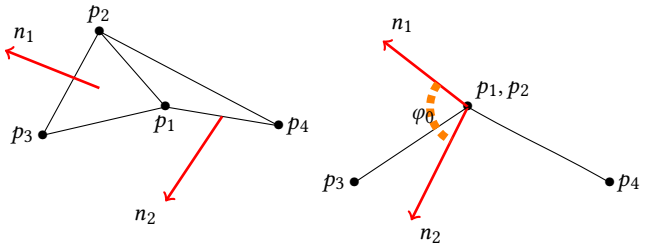


Fig. 3. Bending constraint illustration

2.2.7 Isometric bending Constraint. When we first tried to implement the bending stress, the gradient seemed complicated to calculate. So we first opted for the isobending constraint, which is convenient for inextensible surfaces and cloth simulations. The idea of this constraint is to conserve the bending potential energy $\frac{1}{2}\sum_{i,j} Q_{i,j} p_i \cdot p_j$ of the quad, where Q is the initial Hessian matrix.

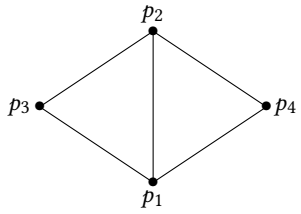


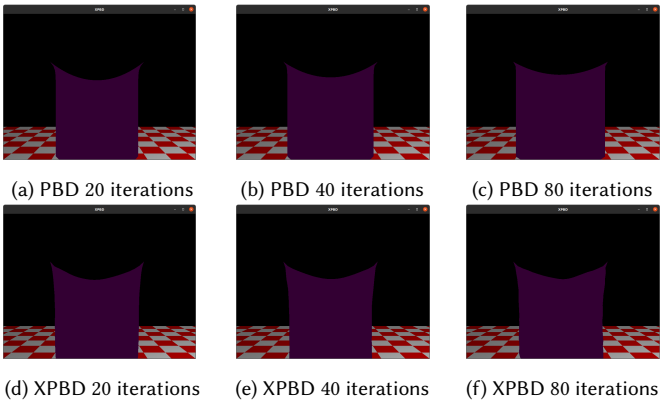
Fig. 4. Isometric Bending constraint illustration

3 DISCUSSIONS

3.1 Stiffness dependance of PBD

One of the shortcomings of PBD is its dependence on the number of iterations used in the Gauss-Seidel process. Figure 5 shows the result for a cloth simulation using PBD and XPBD. If you look at the first line, the curvature changes with the number of iterations: the more substeps there are, the larger the curvature becomes. This is especially visible at the beginning of the simulation, when the cloth starts to fall.

On the contrary, the XPBD loop leaves the curvature qualitatively unchanged. This is the effect of the Lagrangian multiplier λ and the compliance α which act as regularization terms and allow to simulate an arbitrary elastic and dissipative energy potential.



(a) PBD 20 iterations (b) PBD 40 iterations (c) PBD 80 iterations

(d) XPBD 20 iterations (e) XPBD 40 iterations (f) XPBD 80 iterations

Fig. 5. Stiffness dependance on cloth simulation

3.2 Stability

However, it should be kept in mind that XPBD does not solve all the drawbacks of PBD. In particular, it does not improve either the convergence or the accuracy of the simulation. Therefore, it is still possible to obtain undesirable results or even a divergent simulation, depending on the setting of all the parameters that can affect the error propagation.

Since PBD is an approximation and does not have an exact physical interpretation, we find it acceptable to modify the coefficient in front of $\nabla C^T(p)$ in order for our simulation to work. Thus, in order to take into account possible particles with a mass of their own, we choose to multiply in the PBD solver our Δp by an average factor $\frac{m^{-1}}{\sum_{i \in \text{particles}} m_i^{-1}}$.

Moreover, in the PBD solver, $||\Delta p|| \sim \frac{C(p)}{||\nabla C(p)||}$, so if the gradient is close to 0, then Δp might diverge. This can create problems if p is a potential extremum of the constraint different from the true solution, so we put a threshold to try to avoid this phenomenon, taking the bet that the predicted position is the closest extremum point. In the XPBD loop, if the inverse mass is too high (i.e. the mass is very small), then $\Delta p \approx (-C(p) - \tilde{\alpha}\lambda)\frac{\nabla C^T(p)}{||\nabla C(p)||^2}$, so there is potentially the same problem.

4 POTENTIAL FUTURE DIRECTIONS

PBD and XPBD also depend strongly on the order in which the constraints are processed, since two constraints can affect the same set of particles. To improve the behavior of the particles, it might be interesting to try a hierarchical approach of either the particles in space or the constraints. This could remove convergence problems and give better stability.

Also, our implementation structure relies on pointers in C++, so if the number of particles or constraints is too high, the simulation could slow down considerably. One way to work on this would be to do a parallelization on the constraints with a harmonization of

all Δp at the end (e.g. by dividing them with the number of implicit constraints). It would also be possible to use another solving method than the Gauss-Seidel/Jacobi approach.

5 CONCLUSION

In this project, we implemented an extended version of the PBD algorithm. This XPBD, which stores only one more scalar than PBD, removes one of the main shortcomings of PBD: the dependence of the number of iterations on the stiffness. Thus, an arbitrary elastic and dissipative energy potential can be taken into account. However, other typical problems of PBD such as convergence speed and stability are still remaining. In the end, this project allowed us to explore and better understand a common technique used in real-time physical animation.

A APPENDIX

The calculation of some gradients relies on the derivatives $\left(\frac{\partial n}{\partial p_i}\right)_{i=1,2}$ where $n = \frac{p_1 \times p_2}{||p_1 \times p_2||}$. Those are given by

$$\frac{\partial n}{\partial p_1} = \frac{1}{||p_1 \times p_2||}\left(-\begin{pmatrix} 0 & -p_{2,z} & p_{2,y} \\ p_{2,z} & 0 & -p_{2,x} \\ -p_{2,y} & p_{2,x} & 0 \end{pmatrix} + n(n \times p_2)^T\right)$$

$$\frac{\partial n}{\partial p_2} = \frac{-1}{||p_1 \times p_2||}\left(-\begin{pmatrix} 0 & -p_{1,z} & p_{1,y} \\ p_{1,z} & 0 & -p_{1,x} \\ -p_{1,y} & p_{1,x} & 0 \end{pmatrix} + n(n \times p_1)^T\right)$$

Table 1. Constraints formula

Constraint	**Value**	**Gradient**
Distance	$C = \lvert p_A - p_B\rvert - l_0 = 0$	$\nabla_{p_A} C = \frac{p_A - p_B}{\lvert\lvert p_A - p_B\rvert\rvert}$ $\nabla_{p_B} C = \frac{-(p_A - p_B)}{\lvert p_A - p_B\rvert}$
Collision	$C = \lvert p_A - p_B\rvert - l_0 \geqslant 0$	$\nabla_{p_A} C = \frac{p_A - p_B}{\lvert\lvert p_A - p_B\rvert\rvert}$ $\nabla_{p_B} C = \frac{-(p_A - p_B)}{\lvert p_A - p_B\rvert}$
Penetration	$C = (q - p_1) \cdot n \geqslant 0$ where $n = \frac{(p_2 - p_1) \times (p_3 - p_1)}{\lvert\lvert (p_2 - p_1) \times (p_3 - p_1)\rvert\rvert}$	$\nabla_q C = n$ $\nabla_{p_1} C = -\nabla_q C - \nabla_{p_2} C - \nabla_{p_3} C$ $\nabla_{p_2} C = \left(\frac{\partial n}{\partial p_2}\right)^T (q - p_1)$ $\nabla_{p_3} C = \left(\frac{\partial n}{\partial p_3}\right)^T (q - p_1)$
Bending	$C = \arccos(d) - \varphi_0 = 0$ where $n_1 = \frac{(p_2 - p_1) \times (p_3 - p_1)}{\lvert\lvert (p_2 - p_1) \times (p_3 - p_1)\rvert\rvert}$ and $n_2 = \frac{(p_2 - p_1) \times (p_4 - p_1)}{\lvert\lvert (p_2 - p_1) \times (p_4 - p_1)\rvert\rvert}$ and $d = n_1 \cdot n_2$	$\nabla_{p_1} C = -\nabla_q C - \nabla_{p_2} C - \nabla_{p_3} C$ $\nabla_{p_2} C = -\frac{1}{\sqrt{1-d^2}}\left[\left(\frac{\partial n_1}{\partial p_2}\right)^T n_2 + \left(\frac{\partial n_2}{\partial p_2}\right)^T n_1\right]$ $\nabla_{p_3} C = \frac{1}{\sqrt{1-d^2}}\left(\frac{\partial n_1}{\partial p_3}\right)^T n_2$ $\nabla_{p_4} C = \frac{1}{\sqrt{1-d^2}}\left(\frac{\partial n_2}{\partial p_4}\right)^T n_1$
Isobending	$C = \frac{1}{2}\sum_{i,j} Q_{i,j} p_i \cdot p_j = 0$	$\nabla_{p_i} C = \sum_j Q_{i,j} p_j$

ACKNOWLEDGMENTS

The author would like to thank Mr. Kiwon Um of Telecom Paris for monitoring this project, as well as Mr. Amal Dev Parakkat and Mr. Jonathan Fabrizio for teaching the Advanced 3D Computer Graphics (IMA904/IG3DA) class.

REFERENCES

Jan Bender, Matthias Müller, and Miles Macklin. 2015. Position-Based Simulation Methods in Computer Graphics.. In *Eurographics (tutorials)*. 8.

Miles Macklin, Matthias Müller, and Nuttapong Chentanez. 2016. XPBD: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*. 49–54.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2 (2007), 109–118.