# Supplemental Material For "MGPBD: A Multigrid Accelerated Global XPBD Solver"

Chunlei Li

## 1 Resiviting XPBD

### 1.1 Dynamical system

The XPBD starts from an implicit verion of Newton's second law of motion $M\ddot{x}^{n+1} = \boldsymbol{f}(\boldsymbol{x}^{n+1})$, which uses the positions of next time step $\boldsymbol{x}^{n+1}$. $M$ is a diagonal matrix representing mass. The elastic force $\boldsymbol{f}$ can be expressed as $\boldsymbol{f} = -\nabla U^T(\boldsymbol{x}^{n+1})$ according to the variational principle, where $U$ is the potential energy. Besides, the acceleration can be discretized with central difference, i.e., $\ddot{\boldsymbol{x}} = (\boldsymbol{x}^{n+1} - 2\boldsymbol{x}^n + \boldsymbol{x}^{n-1})/\Delta t^2$. For simlicity, let $\tilde{\boldsymbol{x}} = 2\boldsymbol{x}^n - \boldsymbol{x}^{n+1}$ to be the predicted postion, which is a known value representing the position at the next time step under only inertia. Replace the only unkown position $\boldsymbol{x^{n+1}}$ with $\boldsymbol{x}$, then the dynamical system can be expressed as the following form:

$$M\frac{\boldsymbol{x}-\tilde{\boldsymbol{x}}}{\Delta t^2} = -\nabla U^T(\boldsymbol{x}) \tag{1}$$

Use the quratic potential energy function to express $U$, we have

$$U(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{C}(\boldsymbol{x})^T\boldsymbol{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{x}), \tag{2}$$

where the $\boldsymbol{C}(\boldsymbol{x}) \in \mathbb{R}^m$ represents the contraints. The $\boldsymbol{\alpha}$, which is a diagonal matrix, stands for a physical parameter representing compliance (the inverse of stiffness). The gradient of the potential energy is calculated as $\nabla U^T(\boldsymbol{x}) = \nabla\boldsymbol{C}(\boldsymbol{x})^T\boldsymbol{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{x})$. For simlicity, let $\tilde{\boldsymbol{\alpha}} = \boldsymbol{\alpha}/(\Delta t^2)$ to encorperate $\Delta t$ into the $\tilde{\boldsymbol{\alpha}}$.

Follows the energy minimization principle, the Newton equation 1 can be transformed into an optimization problem:

$$x = \underset{x}{\operatorname{argmin}}(U(x) + \frac{1}{2\Delta t^2}||x-\tilde{x}||_M^2), \tag{3}$$

where $\frac{1}{2\Delta t^2}||x-\tilde{x}||_M^2 = \frac{1}{2\Delta t^2}(x-\tilde{x})^T M(x-\tilde{x})$ is the inertial energy, which corresponds to the integral of left part of equation Eq. 1. The optimal $\boldsymbol{x}$ is the next time step position, and minimizing total energy $E(x) = U(x) + \frac{1}{2\Delta t^2}||\boldsymbol{x}-\tilde{\boldsymbol{x}}||_M^2$ requires a balance between the potential energy and the inertial energy.

By taking first derivative of $\frac{1}{2}\boldsymbol{C}(\boldsymbol{x})^T\boldsymbol{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{x}) + \frac{1}{2\Delta t^2}||\boldsymbol{x}-\tilde{\boldsymbol{x}}||_M^2$, the optimality condition is $\nabla\boldsymbol{C}(\boldsymbol{x})^T\boldsymbol{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{x}) + M(\boldsymbol{x}-\tilde{\boldsymbol{x}})/\Delta t^2$, which is exactly the Newton's equation of motion Eq. 1.

The XPBD introduce a lagrange multiplier $\boldsymbol{\lambda} = -\boldsymbol{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{x})$ to transforms the original problem into a dual problem:

$$\begin{aligned} M(\boldsymbol{x}-\tilde{\boldsymbol{x}}) - \nabla\boldsymbol{C}(\boldsymbol{x})^T\boldsymbol{\lambda} &= \boldsymbol{0}, \\ \boldsymbol{C}(\boldsymbol{x}) + \tilde{\boldsymbol{\alpha}}\boldsymbol{\lambda} &= \boldsymbol{0}, \end{aligned} \tag{4}$$

where $g(\boldsymbol{x},\lambda) = M(\boldsymbol{x}-\tilde{\boldsymbol{x}}) - \nabla\boldsymbol{C}(\boldsymbol{x})^T\boldsymbol{\lambda}$ is the primal residual and $h(x,\lambda) = \boldsymbol{C}(\boldsymbol{x}) + \tilde{\boldsymbol{\alpha}}\boldsymbol{\lambda}$ is the dual residual. Minimizing these two residuals is the goal. Becaseue the KKT system is a non-linear system, the XPBD applies one Newton step to linearize it:

$$\begin{bmatrix} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} & \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\lambda}} \\ \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}} & \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\lambda}} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = -\begin{bmatrix} g(\boldsymbol{x},\lambda) \\ h(\boldsymbol{x},\lambda) \end{bmatrix}, \tag{5}$$

where the Jacobian of this block matrix is:

$$\begin{bmatrix} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} & \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\lambda}} \\ \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}} & \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} M & \nabla\boldsymbol{C}(\boldsymbol{x})^T \\ \nabla\boldsymbol{C}(\boldsymbol{x}) & \tilde{\boldsymbol{\alpha}} \end{bmatrix}. \tag{6}$$

XPBD does two simplications for this linear system: First, the top-left block of this Eq. 6 should be $\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} = M - \boldsymbol{\lambda}^T\nabla^2\boldsymbol{C} = M - \sum_{j=1}^m \lambda_j\nabla^2\boldsymbol{C_j}$ originally, but XPBD omits the Hessian of constraints. This simplification makes XPBD a quasi-Newton method. Tournier et. al.[4] pointed out that this simplification will cause the unstable in the transverse direction of the constraints, e.g., the perpendicular direction of a distance constraint. They also provides experiements to show that when the stiffness is extreme or mass ratio is large, the geometric stiffness is important to make the simulation more stable.

Second, the linearized KKT system should originally minimize both the primal and dual residuals, but XPBD only minimizes the dual residual and set the primal residual to zero. This simplification is justified by the observation that $\lambda$ starts from zero and $x$ starts from $\tilde{x}$, which makes the $g$ not far from zero when the iteration number is low. Goldenthal et. al. also reports the small primal residual in their experiments. This simplification brings benefits: $g$ and $\lambda$ will be decoupled, making the computation easier. Besides, it makes the simulation more stable as well according to experiments given by Chen et. al.[1]. However, Chen et. al.[1] pointed out that this will cause a stagnation of Newton residual($\sqrt{g^2+h^2}$), making the problem is not truly converged even though the dual residual is very small. The problem is serious especially when the iteration number is high. Finally, the linearized KKT system is simplified to the following form:

$$\begin{bmatrix} M & -\nabla\boldsymbol{C}(\boldsymbol{x})^T \\ \nabla\boldsymbol{C}(\boldsymbol{x}) & \tilde{\boldsymbol{\alpha}} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{x} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = -\begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{h}(\boldsymbol{x},\lambda) \end{bmatrix}. \tag{7}$$

Applying the Schur complement with respect to $\boldsymbol{M}$ will lead to the $A\Delta\lambda = b$ that XPBD numerically solves:

$$\begin{aligned} A\Delta\lambda &= b, \\ A &= \nabla\boldsymbol{C}(\boldsymbol{x})M^{-1}\nabla\boldsymbol{C}(\boldsymbol{x})^T + \tilde{\boldsymbol{\alpha}}, \\ \boldsymbol{b} &= -\boldsymbol{C}(\boldsymbol{x}) - \tilde{\boldsymbol{\alpha}}\boldsymbol{\lambda}. \end{aligned} \tag{8}$$

The position correction $\Delta\boldsymbol{x}$ is calculated by a substitution of $\Delta\boldsymbol{\lambda}$, which gives

$$\Delta\boldsymbol{x} = M^{-1}\nabla\boldsymbol{C}(\boldsymbol{x})^T\Delta\boldsymbol{\lambda} \tag{9}$$

The new postion is either corrected one by one(a Gauss-Seidel style) or corrected all at once(a Jacobi step). The lagrange multiplier accumulates during the iteration and set to zero at the beginning of a new frame.

### 1.2 Limitations of XPBD numerical solving

The XPBD is a very efficient algorithm for simulating cloth and deformable bodies, etc. The numerical essence of XPBD lies in the

non-linear Gauss-Seidel/Jaocbi. For each constraints $j$, it solves Eq. 8 with following equation:

$$\Delta\lambda_j = \frac{b_j}{A_{j,j}} = \frac{-\boldsymbol{C}_j - \tilde{\boldsymbol{\alpha}}_j \boldsymbol{\lambda}_j}{\nabla C_j M^{-1} \nabla C_j{}^T + \tilde{\alpha}_j} \tag{10}$$

$A_{j,j}$ is the diagonal term of the system matrix. This line of equation is a scalar equation, where all terms are scalars associating line $j$ of matrix $A$. Thus one step of solving in XPBD is extremely cheap. Here two key simlifications are made:

(1) The off-diagonal terms are ignored.
(2) It only has one sweep of Gauss-Seidel/Jaocbi step for each simulation iteration.

These simplifications makes the XPBD a very fast algorithm, because it simplifies the numerical solving step to the extreme. No other numerical method can be faster than this one sweep of diagonal terms in terms of per iteration cost. We think this type of solving is "trotting", which means taking small quick steps.

However, there is no free lunch. The off-diagonal terms contains information of adjacent constrains, and dropping it will make the XPBD a pure local solver. Therefore, we use a global solver that makes up the off-diagonal terms. Our approach is similar to the "Step And Project" method in Goldenthal's work[2].

The global system with off-diagonal terms will obviously make the solving step more costly, but it is worthy as Fig. 8 shows. In this cloth hanging scene, both XPBD and our method converge to $1e-4$ dual residual, and all parameters are the same. But it is clearly that XPBD has strechy issue while our method does not. As Liu[3] pointed out, the XPBD does not converge to a true physical solution, but it reaches a contraint manifold that satisfies $C(\boldsymbol{x}) \approx 0$ as Fig. 2 shows. Our method is similar to Godental et al.[**?** ], which is closer to the true solution compared to XPBD.

If we use a global system, one sweep of GS as XPBD will not succeed anymore. As Fig. 1 shows, on the left side is the global system solved by one step of GS, and on the right side is our method(solved by AMG). Goldenthal et. al. uses a direct solver, which is neither efficient nor scalable for a large sparse system.

Increaseing the interation number of GS will aliviate the problem. However, the iterative methods will have the stalling issue when the problem becomes hard(e.g., high resolution, high stiffness, large time step size, etc.). The residual($b-Ax$) will decrease very quick at the beginning, but it will stall at a certain value and never decrease anymore. This is causeed by the fact that iterative methods are efficient in eliminating the high frequency error, but imcompetent in eliminating the low frequency error. In the contrast, a multigrid solver, which can be seen as a series of iterative solvers running at different resolutions, is able to solve different frequency error. The low frequency error will become high frequency error in the coarse level, which makes it efficient in eliminating the low frequency error.

Therefore, we will introduce our method to deal with the three above limitations of XPBD:

(1) We assemble a full system matrix with off-diagonal terms
(2) We use multiple iterations to solve the system matrix instead of just one sweep.
(3) We use a multigrid method instead of the iterative methods

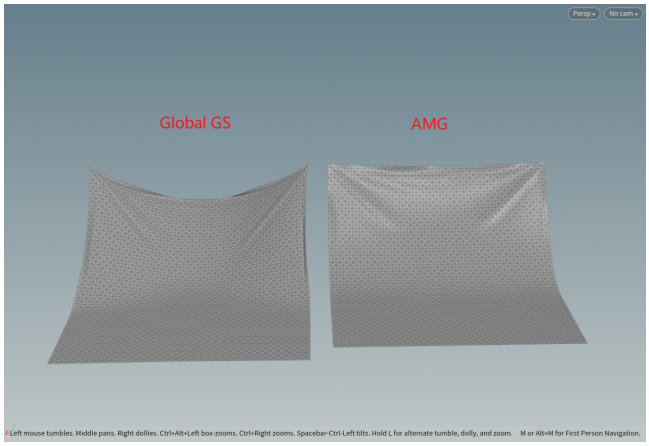


**Figure 1: One sweep of GS with global system matrix will not succeed anymore. Left:global system solved by one step of GS, Right(Ours):global system solved by AMG.**

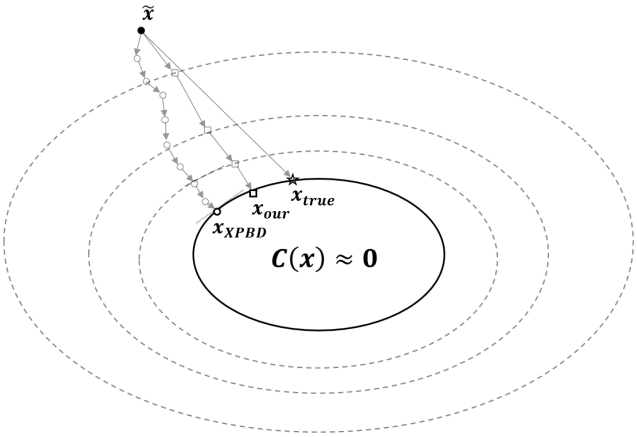


**Figure 2: XPBD has a drift solution issue: it does not converge to a true physical solution, but reaches a contraint manifold that satisfies $C(\boldsymbol{x}) \approx 0$.**

A full system will obviously make each step of solving costly, however, a multigrid method will make each step large towards the final solution. This is the reason why we think our method is "striding", which means taking large steps.

## 1.3 Insight of system matrix of XPBD

The system matrix $A$ expressed in Eq. 8 is a SPD(Symmetric Positive Definite) matrix. Let alone diagonal matrix $\tilde{\boldsymbol{\alpha}}$, which does not change the structure of matrix, $\nabla \boldsymbol{C}(\boldsymbol{x}) M^{-1} \nabla \boldsymbol{C}(\boldsymbol{x})^T$ is a triple matrix product. Here $M^{-1}$ is also a diagonal matrix, and multiplying a diagonal matrix on the left is equivalent to scaling each row of the matrix. Reversely, multiplying a diagonal matrix on the right is equivalent to scaling each column of the matrix. Thus, the $M^{-1}$ does not affect the structure of the matrix as well. Therefore, the structure of the system matrix $A$ is determined by the product of gradient matrix with itself, $\nabla C \nabla C^T$.

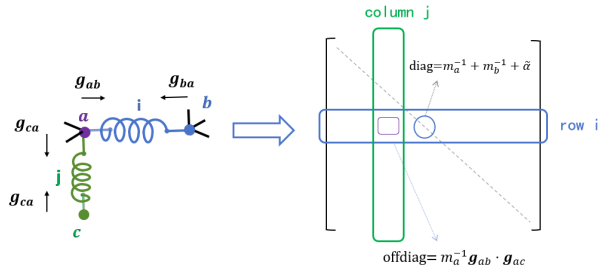


**Figure 3: Structure of A in mass-spring system. The diagonal terms are $m_a^{-1}+m_b^{-1}+\tilde{\alpha}$, and the off-diagonal terms are $m_a^{-1}\boldsymbol{g}_{ab}\cdot\boldsymbol{g}_{ac}$. i:current spring, j:adjacent spring, a: shared vertex, b&c: other vertices**

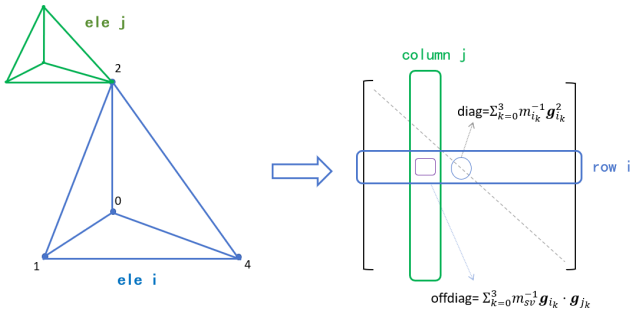


**Figure 4: Structure of A in tetrahedral system. The diagonal terms are $\sum_{k=0}^{3} m_{i_k}^{-1}\boldsymbol{g}_{i_k}^2$, and the off-diagonal terms are $\sum_{k=0}^{3} m_{sv}^{-1}\boldsymbol{g}_{i_k}\cdot\boldsymbol{g}_{j_k}$. i:current element, j:adjacent element, k: local vertex number(0-3), sv: shared vertex**

The Gradient matrix $\nabla C \in \mathbb{R}^{m\times 3n}$ is a sparse matrix, where $m$ denotes the number of contraints and $n$ denotes number of vertices. Each row of $\nabla C$ corresponds to a constraint, and each column corresponds to a vertex. The structure of $\nabla C$ is:

$$\nabla C = \begin{bmatrix} \frac{\partial C_1}{\partial \boldsymbol{x}_1} & \frac{\partial C_1}{\partial \boldsymbol{x}_2} & \cdots & \frac{\partial C_1}{\partial \boldsymbol{x}_n} \\ \frac{\partial C_2}{\partial \boldsymbol{x}_1} & \frac{\partial C_2}{\partial \boldsymbol{x}_2} & \cdots & \frac{\partial C_2}{\partial \boldsymbol{x}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial C_m}{\partial \boldsymbol{x}_1} & \frac{\partial C_m}{\partial \boldsymbol{x}_2} & \cdots & \frac{\partial C_m}{\partial \boldsymbol{x}_n} \end{bmatrix}, \tag{11}$$

where each $\frac{\partial C_j}{\partial \boldsymbol{x}_i}$ is a 3 by 1 row vector for 3 dimensions $x, y, z$. The $\nabla C$ is sparse because each constraint only affects a few vertices. For example, a distance constraint $\boldsymbol{C}_1$ only affects two vertices $\boldsymbol{x_1}, \boldsymbol{x_2}$. So for each row, there are only 6 non-zero elements that are $\frac{\partial C_1}{\partial \boldsymbol{x}_{1,x}}$, $\frac{\partial C_1}{\partial \boldsymbol{x}_{1,y}}$, $\frac{\partial C_1}{\partial \boldsymbol{x}_{1,z}}$, $\frac{\partial C_1}{\partial \boldsymbol{x}_{2,x}}$, $\frac{\partial C_1}{\partial \boldsymbol{x}_{2,y}}$, $\frac{\partial C_1}{\partial \boldsymbol{x}_{2,z}}$.

$\nabla \boldsymbol{C}(\boldsymbol{x})\nabla \boldsymbol{C}(\boldsymbol{x})^T$ is a $m \times m$ matrix, where each element is the dot product of two rows of $\nabla C$. There are two situations: 1) If the two rows are from the same constraint, then the dot product is the sum of the square of the partial derivatives of the constraint with respect to the vertices. This term is the diagonal term of the matrix $A_{j,j}$. 2) If the two rows are from different constraints, then only those vertices that are shared by the two constraints will have non-zero dot product. This term is the off-diagonal term of the matrix $A_{i,j}$.

Finally, we get the structure of $A$ considering the $M^{-1}$ and $\tilde{\boldsymbol{\alpha}}$. For a mass-spring system, it is shown as Fig. 3. For a tetrahedral system, it is shown as Fig. 3.

The CSR(Compressed Sparse Row) format is a good way to store the sparse matrix, because each row of the matrix coincidentally corresponds to a constraint. With the structure of the matrix, we can directly fill the matrix non-zeros in parallel, which lead to a very efficient way to assemble the system matrix.

## 1.4 Geometric Stiffness

Tournier[4] et. al. pointed out that the geometric stiffness is important to make the simulation more stable when the stiffness is extreme or mass ratio is large. The geometric stiffness is the omitted Hessian term of the upper left term in equation 6. We denote it as $\tilde{K} = \boldsymbol{\lambda}^T\nabla^2 C = \sum_{j=1}^{m}\nabla^2 C_j\lambda_j$ to keep the symbol same as Tourinier's[4] paper.

The Hessian $\nabla^2\boldsymbol{C} \in \mathbb{R}^{m\times 3n\times 3n}$ is a tensor can be expressed as the following form:

$$\nabla^2 \boldsymbol{C} = \begin{bmatrix} \frac{\partial^2 C}{\partial \boldsymbol{x}_1^2} & \frac{\partial^2 C}{\partial \boldsymbol{x}_1 \partial \boldsymbol{x}_2} & \cdots & \frac{\partial^2 C}{\partial \boldsymbol{x}_1 \partial \boldsymbol{x}_n} \\ \frac{\partial^2 C}{\partial \boldsymbol{x}_2 \partial \boldsymbol{x}_1} & \frac{\partial^2 C}{\partial \boldsymbol{x}_2^2} & \cdots & \frac{\partial^2 C}{\partial \boldsymbol{x}_2 \partial \boldsymbol{x}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 C}{\partial \boldsymbol{x}_n \partial \boldsymbol{x}_1} & \frac{\partial^2 C}{\partial \boldsymbol{x}_n \partial \boldsymbol{x}_2} & \cdots & \frac{\partial^2 C}{\partial \boldsymbol{x}_n^2} \end{bmatrix}, \tag{12}$$

where $\boldsymbol{C} = [C_1, C_2, \cdots, C_m]^T \in \mathbb{R}^{m\times 1}$, each $\boldsymbol{x}_j \in \mathbb{R}^{3\times 1}$ and each element $\frac{\partial^2 C_k}{\partial \boldsymbol{x}_i \partial \boldsymbol{x}_j}$ is a $3\times 3$ block. For $\tilde{K} = \boldsymbol{\lambda}^T\nabla^2 C$, we can simply first multiply $\boldsymbol{\lambda}$ with $\boldsymbol{C}$ and get a scalar $\boldsymbol{\lambda}^T\boldsymbol{C} = \sum_{j=1}^{m}\lambda_j C_j$. Then replace all above $\boldsymbol{C}$ with $\sum_{j=1}^{m}\lambda_j C_j$ to get the $\tilde{K}$.

For a mass-spring system, the distance constraint $\boldsymbol{C}$, $\nabla\boldsymbol{C}$ and $\nabla^2\boldsymbol{C}$ are like:

$$\begin{aligned} \boldsymbol{C} &= ||\boldsymbol{p}-\boldsymbol{q}|| - l_0, \\ \nabla\boldsymbol{C} &= \frac{(\boldsymbol{p}-\boldsymbol{q})}{||\boldsymbol{p}-\boldsymbol{q}||}, \\ \nabla^2\boldsymbol{C} &= \frac{\boldsymbol{I}}{||\boldsymbol{p}-\boldsymbol{q}||} - \frac{(\boldsymbol{p}-\boldsymbol{q})(\boldsymbol{p}-\boldsymbol{q})^T}{||\boldsymbol{p}-\boldsymbol{q}||^3}, \end{aligned} \tag{13}$$

where $I$ denotes the identity matrix, $\boldsymbol{p}$ and $\boldsymbol{q}$ are DOF of two vertices of the spring, and $l_0$ is the rest length of the spring. If we denote the unit direction vector as $\boldsymbol{n} = \frac{\boldsymbol{p}-\boldsymbol{q}}{||\boldsymbol{p}-\boldsymbol{q}||}$ and currect length as $l = ||\boldsymbol{p}-\boldsymbol{q}||$. Then only takes the diagonal term of $\tilde{K}$, the geometric stiffness for each spring is $\tilde{K}_{j,j} = \frac{\lambda_j}{l_j}(\boldsymbol{I}-\boldsymbol{n_j}\boldsymbol{n_j}^T) \in \mathbb{R}^{3\times 3}$. Takes the diagonal of this $3\times 3$ small matrix again, the diagonal term of the geometric stiffness matrix is $K_{diag} = \frac{\lambda_j}{l_j}diag(1-n_{j,x}^2, 1-n_{j,y}^2, 1-n_{j,z}^2)$.

## 1.5 XPBD vs. Newton's method

The Newton's method is a conventional method that solves the non-linear system. It uses the Jacobian and Hessian of the total energy.

$$
\begin{aligned}
E(x) &= \frac{1}{2\Delta t^2}||\boldsymbol{x}-\tilde{\boldsymbol{x}}||_M^2 + \frac{1}{2}\boldsymbol{C}(\boldsymbol{x})^T\boldsymbol{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{x}) \\
\frac{\partial E}{\partial x} &= \frac{1}{\Delta t^2}M(x-\tilde{x}) + \nabla\boldsymbol{C}(\boldsymbol{x})^T\boldsymbol{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{x}) \\
\frac{\partial^2 E}{\partial x^2} &= \frac{1}{\Delta t^2}M + \nabla\boldsymbol{C}(\boldsymbol{x})^T\alpha^{-1}\nabla\boldsymbol{C}(\boldsymbol{x}) + \nabla^2\boldsymbol{C}(\boldsymbol{x})\boldsymbol{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{x}).
\end{aligned} \tag{14}
$$

In a standard Newton method, it solves the following linear system:

$$
\frac{\partial^2 E}{\partial x^2}\Delta x = -\frac{\partial E}{\partial x}. \tag{15}
$$

Notice that if we substitute back the $\lambda = -\alpha\boldsymbol{C}(\boldsymbol{x})$ and express primal residual solely in x, $g(x) = M(\boldsymbol{x}-\tilde{\boldsymbol{x}}) - \nabla\boldsymbol{C}(\boldsymbol{x})^T\alpha^{-1}\Delta t^2 C(x)$, it is the same with the energy Jacobian except $\Delta t^2$, i.e., $\frac{\partial E}{\partial x} = g(x)/\Delta t^2$. And thus the $\frac{\partial g}{\partial x}/\Delta t^2$ will be total energy Hessian. XPBD omits the third term of $\frac{\partial^2 E}{\partial x^2}$ which makes it a quasi-Newton method.

For one spring, the gradient and Hessian of the potential for one point $\boldsymbol{p}$ are:

$$
\begin{aligned}
U &= \frac{1}{2}k(l-l_0) \\
\nabla_{\boldsymbol{p}}U &= k(1-l_0/l)(\boldsymbol{p}-\boldsymbol{q}) \in \mathbb{R}^{3\times 1} \\
\nabla^2_{\boldsymbol{pp}}U &= \frac{l_0}{l^3}(\boldsymbol{p}-\boldsymbol{q})(\boldsymbol{p}-\boldsymbol{q})^T + k(1-\frac{l_0}{l})\boldsymbol{I}_3 \in \mathbb{R}^{3\times 3}
\end{aligned}. \tag{16}
$$

Because of the symmetry, $\nabla_{\boldsymbol{q}}U = -\nabla_{\boldsymbol{p}}U$. Similarly, Hessian has $\nabla^2_{\boldsymbol{pp}}U = \nabla^2_{\boldsymbol{qq}}U = -\nabla^2_{\boldsymbol{pq}}U = -\nabla^2_{\boldsymbol{qp}}U \equiv B$. So gradient and Hessian for one spring j are:

$$
\begin{aligned}
\nabla U_j &= \begin{bmatrix}\nabla_{\boldsymbol{p}}U \\ \nabla_{\boldsymbol{q}}U\end{bmatrix} \in \mathbb{R}^{6\times 1} \\
\nabla^2 U_j &= \begin{bmatrix}\nabla^2_{\boldsymbol{pp}}U & \nabla^2_{\boldsymbol{pq}}U \\ \nabla^2_{\boldsymbol{qp}}U & \nabla^2_{\boldsymbol{qq}}U\end{bmatrix} = \begin{bmatrix}B & -B \\ -B & B\end{bmatrix} \in \mathbb{R}^{6\times 6}
\end{aligned}. \tag{17}
$$

Compared to the Newton's method, XPBD solves a dual system instead of the primal system. Dropping the constraint Hessian term $\nabla^2\boldsymbol{C}(\boldsymbol{x})\boldsymbol{\alpha}^{-1}\boldsymbol{C}(\boldsymbol{x})$ in Newton's method as well, their system matrices are

$$
\begin{aligned}
primal: A &= \nabla\boldsymbol{C}(\boldsymbol{x})^T\alpha^{-1}\nabla\boldsymbol{C}(\boldsymbol{x}) + \frac{1}{\Delta t^2}M \\
dual: A &= \nabla\boldsymbol{C}(\boldsymbol{x})M^{-1}\nabla\boldsymbol{C}(\boldsymbol{x})^T + \tilde{\boldsymbol{\alpha}}
\end{aligned} \tag{18}
$$

Let alone the $\frac{1}{\Delta t^2}M$ and $\tilde{\boldsymbol{\alpha}}$ terms, which are diagonal matrices, the key difference in these two system matrices is $\nabla\boldsymbol{C}(\boldsymbol{x})^T\alpha^{-1}\nabla\boldsymbol{C}(\boldsymbol{x})$ versus $\nabla\boldsymbol{C}(\boldsymbol{x})M^{-1}\nabla\boldsymbol{C}(\boldsymbol{x})^T$. They have two differences: First, the primal system has dimension $m\times m$ while the dual system has dimension $3n\times 3n$, where $m$ is number of constraints, and $n$ is the number of particles. In a tetrahedral system the number of constrains usually exceed number of particles, while it may be reversed in a mass-spring system. Second, the primal system matrix is scaled by $\alpha^{-1}$, while the dual system matrix is scaled by $M^{-1}$. Thus Macklin et. al.[**?** ] pointed out that the primal system can stands large mass ratio while the dual system can stands large stiffness.

### 1.6 Relation to Fast Projection: Regularization Term

Our system has two key differences with Goldental et. al.'s Fast Projection[2]:

(1) We have a regularization term while Fast Projection does not.
(2) We use a multigrid method while Fast Projection uses a direct solver.

Simply put, the regularization term $\tilde{\boldsymbol{\alpha}}$ added in the diagonal:

$$
\begin{aligned}
Ours: A &= \nabla\boldsymbol{C}(\boldsymbol{x})M^{-1}\nabla\boldsymbol{C}(\boldsymbol{x})^T + \tilde{\boldsymbol{\alpha}} \\
Goldental's: A &= \nabla\boldsymbol{C}(\boldsymbol{x})M^{-1}\nabla\boldsymbol{C}(\boldsymbol{x})^T
\end{aligned} \tag{19}
$$

This term helps to stabilize the system. From physcial viewpoint, dropping it equals to a always infinite stiffness system. The PBD has no regularization term as well, which is a key improvement of XPBD over PBD. From numerical viewpoint, a system has larger diagonal will prone to diagonal dominance, which results in a smaller condition number. From optimization viewpoint, for any given objective function $g$, if we add a penalty term $\frac{1}{2}\Delta\boldsymbol{x}^T\tilde{\boldsymbol{\alpha}}\Delta\boldsymbol{x}$ to its Taylar expansion, it becomes:

$$
g(\boldsymbol{x}) \approx g(\boldsymbol{x}^k) + \nabla g(\boldsymbol{x}^k)^T\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^T\nabla^2 g(\boldsymbol{x}^k)\Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^T\tilde{\alpha}\Delta\boldsymbol{x} \tag{20}
$$

The optimal condition is:

$$
\nabla g(\boldsymbol{x}^k) + \nabla^2 g(\boldsymbol{x}^k)\Delta\boldsymbol{x} + \tilde{\boldsymbol{\alpha}}\Delta\boldsymbol{x} = 0. \tag{21}
$$

Reformulate it and we get:

$$
\Delta\boldsymbol{x} = (\nabla^2 g(\boldsymbol{x}^k) + \tilde{\boldsymbol{\alpha}})^{-1}\nabla g(\boldsymbol{x}^k) \tag{22}
$$

This is Newton's method with a regularization term. Without the regularization term, if the Hessian is too small, the Hessian inverse will becomes too large and generates a too large step size. So the regularization term prevents a too large step size. This is the reason why we think the regularization term can stabilize the system.

## 2 AMG

### 2.1 Setup phase and solve phase

A multigrid solving process can be divided into two phases: the setup phase and the solving phase. The goal of setup phase is to produce a good prolongator P form a sparse matrix. To define the goodness of prolongator, we should evaluate from two aspects: convergency and efficiency. There are many methods to construct setup phase, but for the solving phase, all different multigrid methods are the same. Two mainstream AMG methods to construct setup phase are classical AMG(C-AMG) and aggregation based AMG(A-AMG). Our approach adopts the aggregation based AMG because we found it has less layers and slightly better convergency than classical AMG in our experiements.

### 2.2 Algebraically smooth and near-kernel components

To make the multigrid method accurate, one key is to make the error that passed down to the coarser grid as smooth as possible. For AMG, where we only has the information of the matrix, the

smoothness of the error is measured by "algebraically smoothness". Heuristically, when the error is slow to converge, it can be seen as "algebraically smooth".

There are also other equvilant decriptions of "algebraically smooth". For the residual equations $A\mathbf{e} = \mathbf{r}$ where A is reversible, the error vector $\mathbf{e}$ can be decomposed into a linear combination of the eigenvectors and eigenvalues of A, as Eq. 23

$$\mathbf{e} = \lambda_0 \mathbf{v_0} + \lambda_1 \mathbf{v_1} + \cdots + \lambda_n \mathbf{v_n}, \tag{23}$$

where $\lambda$s are eigenvalues in descending order, $\lambda_0 > \lambda_1 > ... > \lambda_n$. $\mathbf{v_0}, \mathbf{v_1}, \cdots, \mathbf{v_n}$ are corresponding eigenvectors.

The smallest eigenvalue and its corresponding eigenvector have special meaning. We think this eigenvector is the "algebraically smooth error". That is, if applying smoother enough times, finally the error will be sufficiently smooth and the remaining is this part. This vector is called "near-kernel component", $B$, also known as "near-nullspace component".

As its name shown, the near-kernel compoenent is approximately the solution vector of homogeneous system $A\mathbf{x} = \mathbf{0}$. For a reversible A, $A\mathbf{x} = \mathbf{0}$ has the only solution 0. So $B$ is just an approximate solution. Because it is an approxmation, B may not be just one vector, but multiple vectors.

From another aspect, the algebraically smooth means low frequency, so the near-kernel components are low-frequency errors.

## 2.3 Smoother

Every multigrid method are basically consist of two components: smoother and coarse-grid correction. For the smoother, its task is to make the errors as smooth as possible. For the coarse-grid correction, its task is to catch the smoothed error and eliminate them. The bridge between smoother and coarse-grid correction is the interpolation operator P.

The smoother is especially important for a multigrid's convergency. Poor smoother will lead to the failure of multigrid. One intuitive explanation is from the aliasing. AMG is required to capture the signal in the subspace, i.e., the low frequency part of the original input wave. So the wave in the coarse space should approximate the original wave as closer as possible. To achieve that, the orignal wave has to be captured precisely, ortherwise the aliasing issue as shown in the Fig.5 will occur. shows the aliasing phenomenon caused by the inadequet smoothing, where he black wave is the original error, and the red wave is the re-composed wave in the subspece. Red crosses are the sampled points in the restricting process. Green crosses are the sampled points after interpolation. To solve the problem, there are two key factors: 1) Before restriction, applying adequet smoothing to lower down the frequency of the original wave. 2) In the meantime, a precise P is required.

## 2.4 Convergency

In theory, it is hard to give a rigorious proof of the multigrid method. But heuristically define the precision of MG is possible. The essence of MG is to use the error of a small number of points to approximate the error of a large number of points. This approximation is achieved by interpolation. Thus the more precise of interpolation, the better of approximation. A inequality called "Strong Approximation Property" decribe how precise of this approximation:

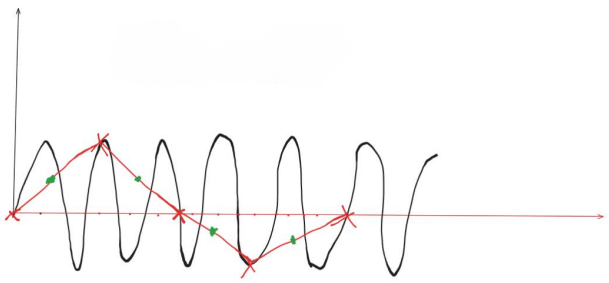

**Figure 5: Aliasing phenomenon**

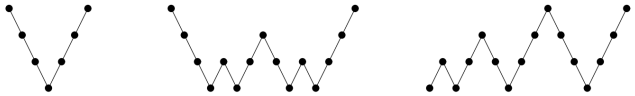

**Figure 6: V-cycle, W-cycle, and F-cycle**

(24)

The AMG is not guranteeted converge in many cases. It depends on many factors: smoother, prolongator, the problem itself, etc.

## 2.5 Efficiency

The performance of the multigrid solve phase depends on two parts: 1)The convergence rate. 2)The per iteration cost. The convergency has been disscussed, so now we will focus on the latter. Sometimes the method with better convergence rate has more per iteration cost, which cause the worse performance. So a balance between the two factors is needed. We did a lot of experiements and found out the sparsity of matrix is important to per iteration cost, since the most inefficient part of one multigrid cycle is sparse matrix-vector(spmv) and smoothers(which are iterative methods). Both are related to the nonzeros of the matrix. So a complex P may produce better convergency, but results in poor performance because it generates complex $Ac$. The simpler of each level system matrix, the better performance. Operator complexity $C$ can measure the complexity of the multi-level system. It is the nonzeros of all levels matrices divided by the finest level matrix as Eq.25,

$$C = \frac{\sum_l nnz(A_l)}{nnz(A_0)} \tag{25}$$

Apparently, $C > 1$ and the less the better.

The cycle type can affect the performance of MG. There are three frequently used types of multigrid cycle: V-cycle, W-cycle and F-cycle(Full cycle). As Fig.6 shows, the main difference between these three types of cycles lies in the arrangement of smoothers. W-cycle and F-cycle solves more coarse level which is cheap compared to fine level. The F-cycle starts from the coarsest level by first restricting to to the bottom level without smoothing. From experiments, we found that a v-cycle is better, since it maintains the simlicity of the structure, which results in a lower operator complexity.

## 2.6 Setup phase

We adopt a aggregation based AMG method to construct the setup phase. The aggregation based AMG is a two-step process: 1)Aggregation. 2)Prolongator construction. The Algorithm 1 shows the process of building P.

In Algorithm 1, for each level, first filter out weak connection with a strength threshold $\theta_s$. This step is called "strength". For each row, the relatively small off-diagonal values are deleted according to the Eq.26, and the row is scaled so that maintain the sum unchanged. The output after this step is a strength of connection matrix(S)

Through our experiements, we found that threshold has a great impact on the performance of AMG. A small threshold will result in a complex P, which leads to a high operator complexity. A large threshold will result in a simple P, but the convergency will be poor. The threshold is set to be a small value, e.g., 0.25.

$$|A_{ij}| < \theta_s \cdot \sqrt{|A_{ii}||A_{jj}|} \tag{26}$$

**Algorithm 1** build P

**Input:** $A$, $maxcoarse$, $maxlevels$
**Output:** $P_0, ..., P_{nl-2}$

$A_0 = A$
$B_0 = (1, 1, ..., 1)^T$
**for** $l = 0, 1, ..., maxlevels$ **do**
  $S_l = strength(A_l)$
  $Agg_l = aggregate(S_l)$
  $P_l, B_{l+1} = inject(Agg_l, B_l)$
  **if** number of unkowns ≤ maxcoarse **then**
    break
  **end if**
**end for**
nl = number of levels

## 2.7 Solve phase

The goal of solve phase is to solve the $Ax = b$ with given prolongators. Firstly, we need to construct the system for each level. A Galerkin style construction with variational principle leads to,

$$A_{l+1} = R_l A_l P_l R_l = P_l^T, \tag{27}$$

The first line comes from the Galerkin principle while the second line comes from the variational principle. The R is the restriction operator, which is the tranpose of P by variational principle.

In each level, AMG applies iterative solver to the error equation $Ae = r$. Fig.7 shows a three layers AMG system with v-cycle. For each layer, there is a linear system $Ax = b$, where the x is error to be corrected in this layer, and b is the residual of this layer. V-cycle starts from layer 0(finest layer), pre-smoothes by iterative method(e.g., a few GS or Jacobi sweeps), and transfer the residual to the next level by R. Repeat until it reaches the bottom level and get the error of the bottom level by solving the system with direct solver or iterative method. In the right half of "V" shape, it goes up by first correcting the error prolongated from coarse level, then applying post-smoother.

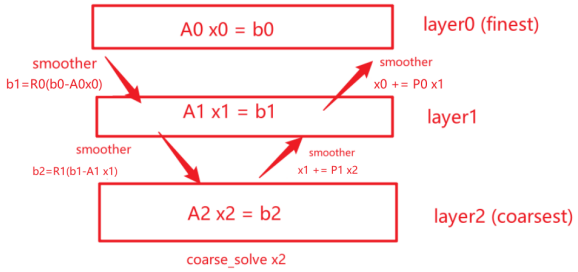


**Figure 7: A three layers AMG system**

The v-cycle algorithm can be implemented as either recursive or non-recursive. We adopts a non-recursive version, which is shown in Algorithm 2.

**Algorithm 2** v-cycle algorithm

**Input:** $A_0, ...A_{nl-1}, P_0, ...P_{nl-1}, R_0, ...R_{nl-1}, x_0, b$
**Output:** new $x_0$ (x in level 0)

$r_0 = b$
**for** $l = 0, 1, ..., nl - 2$ **do**
  $x_l = 0$
  presmooth($A_l$,$x_l$,$r_l$)
  $r_{l+1} = R_l(r_l - A_l x_l)$
**end for**
$x_{nl-1} = coarse\_solver(A_{nl-1}, r_{nl-1})$
**for** $l = nl - 2, nl - 1, ..., 0$ **do**
  $x_l += P_l x_{l+1}$
  postsmooth($A_l$,$x_l$,$r_l$)
**end for**

As Stuben[**?** ] pointed out, the aggregation based AMG is more robust combined with a krylov method such as conjugate gradient(CG), bicgstab, etc. In our experiements, we also found that AMG-CG coupled algorithm is more effient than a stand-alone AMG solver. The AMG-CG algorithm is shown in Algorithm 3. The AMG procedure(i.e., one iteration of v-cycle) is served as a preconditioner for the CG method in the solving z step.

# 3 Implementation

# 4 experiements

# References

[1] Yizhou Chen, Yushan Han, Jingyu Chen, Shiqian Ma, Ronald Fedkiw, and Joseph Teran. 2023. Primal Extended Position Based Dynamics for Hyperelasticity. In *Proceedings of the 16th ACM SIGGRAPH Conference on Motion, Interaction and Games* (Rennes, France) *(MIG '23)*. Association for Computing Machinery, New York, NY, USA, Article 21, 10 pages. https://doi.org/10.1145/3623264.3624437

[2] Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. 2007. Efficient Simulation of Inextensible Cloth. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)* 26, 3 (2007), to appear.

[3] Tiantian Liu, Adam W. Bargteil, James F. O'Brien, and Ladislav Kavan. 2013. Fast simulation of mass-spring systems. *ACM Trans. Graph.* 32, 6, Article 214 (Nov. 2013), 7 pages. https://doi.org/10.1145/2508363.2508406

[4] Maxime Tournier, Matthieu Nesme, Benjamin Gilles, and François Faure. 2015. Stable Constrained Dynamics. *ACM Transactions on Graphics* 34, 4 (Aug. 2015), 132:1–132:10. https://doi.org/10.1145/2766969

**Algorithm 3** amg cg solve

$r_0 = b - Ax_0$
$z_0 = Vcycle(levels, 0, z_0, r_0)$
$p_0 = z_0$
**for** k=0,1,..., maxiter **do**
  $\alpha_k = \frac{r_k^T z_k}{p_k^T A p_k}$
  $x_{k+1} = x_k + \alpha_k p_k$
  $r_{k+1} = r_k - \alpha_k A p_k$
  **if** $\|r_{k+1}\|_2 < tol$ **then**
    break
  **end if**
  $z_{k+1} = Vcycle(levels, 0, z_{k+1}, r_{k+1})$
  $\beta_k = \frac{r_{k+1}^T z_{k+1}}{r_k^T z_k}$
  $p_{k+1} = z_{k+1} + \beta_k p_k$
**end for**

**Algorithm 4** substep of MG-PBD

$\tilde{\boldsymbol{x}}, \boldsymbol{x}, \boldsymbol{x_{old}} \leftarrow$ semiEuler($\boldsymbol{v}, \Delta t, \boldsymbol{f_{ext}}$)
$\lambda \leftarrow (0, ..., 0)^T$
**for** $i = 0, 1, ..., maxiter$ **do**
  calculate constraints C and constraint gradients G
  assemble $A \leftarrow \nabla C M^{-1} \nabla C^T + \tilde{\alpha}$
  calculate $b \leftarrow -C - \tilde{\alpha}\lambda$
  setup AMG for every 20 frames
  solve $A\Delta\lambda = b$ using MGPCG
  $\lambda \leftarrow \lambda + \Delta\lambda$
  $\Delta \boldsymbol{x} \leftarrow M^{-1} \nabla C^T \Delta\lambda$
  $\boldsymbol{x} \leftarrow \boldsymbol{x} + \omega \Delta \boldsymbol{x}$
  **if** dual residual is small **then**
    break
  **end if**
**end for**
$\boldsymbol{v} \leftarrow (\boldsymbol{x} - \boldsymbol{x_{old}})/\Delta t$

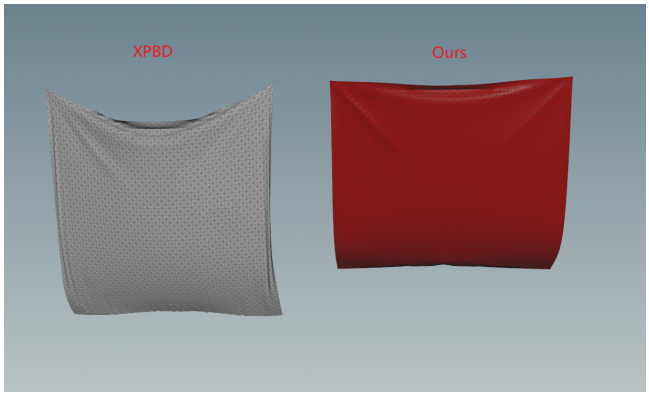


**Figure 8: The XPBD cloth is more strechy compared to our method under the same tolerance and physcial parameters.**