布料仿真：加速的十年

1

()

摘要：本文主要介绍布料仿真的常见方法以及近十年布料仿真的前沿研究，包括对已有算法的性能优化、新的仿真算法，以及算法在 GPU 上的并行化实现，总结了当前布料仿真的前沿方向

关键词：布料仿真

1 布料仿真的常见方法

1.1 弹簧质点系统

布料在发生形变时，需要考虑系统内部的相对影响，通过弹簧可以模拟布料形变时产生的弹性势能。首先将布料用网格表示，给网格的顶点一定的质量，通过弹簧将网格的顶点相连构成弹簧质点系统。

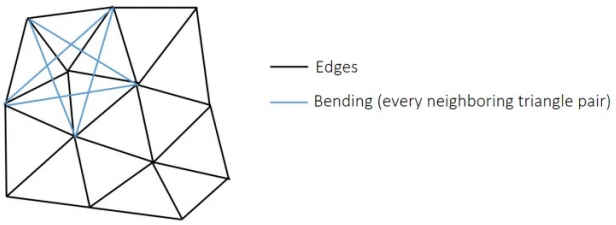

图 1 弹簧质点模型

在不考虑非保守力的作用下，系统中质点的位置可以唯一地确定系统的状态。利用牛顿第二定律：

$$x'' = M^{-1}(\frac{\partial E}{\partial x}) \tag{1}$$

在存在非保守力时：

$$x'' = M^{-1}(\frac{\partial E}{\partial x} + F) \tag{2}$$

在弹簧质点系统中，根据胡克定律 (Hooke's law)：

$$\begin{aligned} F_{spring} &= -k_s \Delta x \\ F_{s_i} &= k_s(1 - \frac{L}{\|x_i - x_j\|})(x_i - x_j) \end{aligned} \tag{3}$$

除此之外还可以加入重力、空气阻力、弹簧阻尼等其它作用力。

在模拟系统随时间的演化时，求解 (2) 式得到：

$$x = \iint M^{-1}(\frac{\partial E}{\partial x} + F)dtdt = \iint adtdt \tag{4}$$

在实际的模拟中，时间是离散的，因此需要做数值积分。直接的方法是显式欧拉方法 (explicit Euler method)：

$$\begin{aligned} v(i+1) &= v(i) + a(i)\Delta t \\ x(i+1) &= x(i) + v(i)\Delta t \end{aligned} \tag{5}$$

此时下一时刻的速度和位移完全由上一时刻决定，导致当 Δt 较大时计算结果不稳定。

另一种方法是隐式欧拉方法 (implicit Euler method)：

$$\begin{aligned} v(i+1) &= v(i) + a(i+1)\Delta t \\ x(i+1) &= x(i) + v(i+1)\Delta t \end{aligned} \tag{6}$$

由于加速度也是位置的函数，此时需要解方程得出下一时刻的速度，它的误差比显式欧拉方法小一些。

在隐式欧拉方法中，将速度方程代入位置方程可以得到：

$$\begin{aligned} x(i+1) &= x(i) + (v(i) + a(i+1)\Delta t)\Delta t \\ &= x(i) + v(i)\Delta t + a(i+1)(\Delta t)^2 \\ &= x(i) + v(i)\Delta t + M^{-1}f(i+1)(\Delta t)^2 \end{aligned} \tag{7}$$

$$\frac{M}{(\Delta t)^2}(x(i+1) - x(i) - v(i)\Delta t) - f(i+1) = 0 \tag{8}$$

在不考虑非保守力时，可以构造一个优化问题，使 $x(i+1)$ 为这个优化问题的解，从而利用求解优化问题的方法求解数值积分：

$$\begin{aligned} F(x) &= \frac{1}{2(\Delta t)^2}(x - \bar{x})^T M(x - \bar{x}) + E(x) \\ \bar{x} &= x(i) + v(i)\Delta t \\ x(i+1) &= \underset{x}{argmin}\, F(x) \end{aligned} \tag{9}$$

求解式 (8) 等价于求解 $F(x)$ 的导数为 0 的点，从而转化为求解 $F(x)$ 的极值。

考虑系统可能发生的碰撞，最终需要求解的问题为：

$$\begin{aligned} &x(i+1) = \underset{x}{argmin}\, F(x) \\ &s.t. \quad x(i+1) \in \Omega \end{aligned} \tag{10}$$

当系统与连续体发生碰撞时：

$$s.t. \quad tx(i+1) + (1-t)x(i) \in \Omega \quad t \in [0,1] \tag{11}$$

为了求解这个优化问题，可以使用牛顿法，利用下面的公式进行迭代直到收敛：

$$x^{k+1} = x^k - (\nabla^2 F(x))^{-1} \nabla F(x) \tag{12}$$

初始的 $x^0 = x(i)$，最终收敛的 $x^k = x(i+1)$

1.2 弹性模型

将布料抽象成三角形的网格，处理当三角形网络发生形变时所带来的的能量变化：

- 当三角形的边长改变时需要能量
- 当两个相邻三角形的夹角发生变化时需要能量
- 当三角形需要变形或剪切时需要能量

在能量发生变化时需要相应的力做功，因此可以通过系统的状态推出系统内部各质点之间的相互作用力。

可以通过建模的方式来计算相应的力，将相互作用力视作边长、相邻三角形二面角的函数。

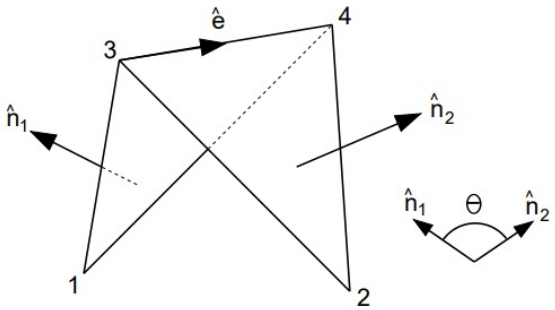

图 2 二面角

首先 u1 和 u2 应该在法线方向 n1 和 n2，其次弯曲不会拉伸边缘 x3x4，因此 u4-u3 应该与边缘正交，即为 n1 和 n2 的线性组合。根据牛顿定律，合，力应为 0，综上所述得到计算结果：

$$\begin{aligned} \mathbf{u}_1 &= \|\mathbf{E}\| \frac{\mathbf{N}_1}{\|\mathbf{N}_1\|^2} \\ \mathbf{u}_2 &= \|\mathbf{E}\| \frac{\mathbf{N}_2}{\|\mathbf{N}_2\|^2} \\ \mathbf{u}_3 &= \frac{(\mathbf{x}_1 - \mathbf{x}_4)\cdot\mathbf{E}}{\|\mathbf{E}\|} \frac{\mathbf{N}_1}{\|\mathbf{N}_1\|^2} + \frac{(\mathbf{x}_2 - \mathbf{x}_4)\cdot\mathbf{E}}{\|\mathbf{E}\|} \frac{\mathbf{N}_2}{\|\mathbf{N}_2\|^2} \\ \mathbf{u}_4 &= -\frac{(\mathbf{x}_1 - \mathbf{x}_3)\cdot\mathbf{E}}{\|\mathbf{E}\|} \frac{\mathbf{N}_1}{\|\mathbf{N}_1\|^2} - \frac{(\mathbf{x}_2 - \mathbf{x}_3)\cdot\mathbf{E}}{\|\mathbf{E}\|} \frac{\mathbf{N}_2}{\|\mathbf{N}_2\|^2} \end{aligned} \tag{13}$$

得到以下有关力的函数

$$\mathbf{f}_i = k \frac{\|\mathbf{E}\|^2}{\|\mathbf{N}_1\| + \|\mathbf{N}_2\|} \sin\left(\frac{\pi - \theta}{2}\right) \mathbf{u}_i \tag{14}$$

也可以在相邻三角形上加上弹簧以抵抗弯曲。

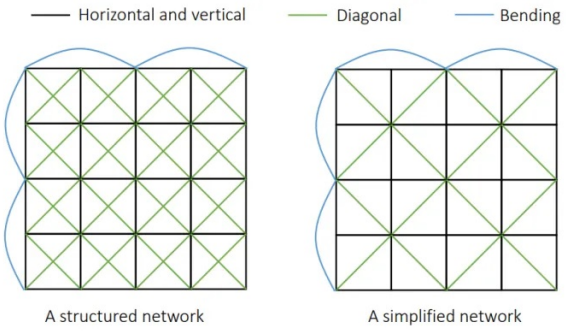

图 3 三种弹簧

质点弹簧模型和其他弯曲模型，假设布料平面变形（伸缩）和布料弯曲变形是独立的，相互不影响，但是实际上并非如此。Locking issue：如下图，当弹簧 stiffnes 很大（即很难形变的时候），右边的网格很难弯曲。

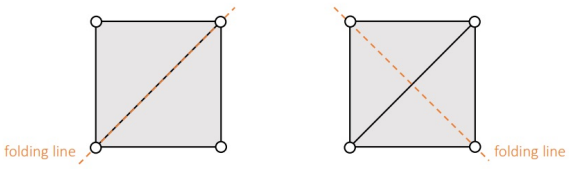

图 4 Locking issue

为了解决这个问题，弹簧在压缩的时候，可以将弹性系数设置的小一点。或者让弹簧在一定范围内没有力，可以自由伸缩。

1.3 基于位置的动力学方法

基于位置的动力学方法 (PBD) 的算法流程如下

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

图 5 PBD 算法

每次迭代时，首先根据外力算出此时的速度更新，然后让速度根据阻尼衰减，再根据速度得到位移的变化量，通过位移生成碰撞约束，之后在位移的基础上对约束条件进行迭代直到收敛，再根据位移变化得出最终的速度。

其中算法的核心是在位移的基础上对约束条件进行迭代直到收敛，这里将之前得到的预估位移投影到约束条件上从而得到最终位移。

假设有一个 n 维约束：

$$\begin{aligned} \boldsymbol{p} &= [\boldsymbol{p_1}, \ldots, \boldsymbol{p_n}] \\ C(\boldsymbol{p}) &= 0 \end{aligned} \tag{15}$$

现在需要找到偏移量 $\Delta \boldsymbol{p}$，使得投影前的 $\boldsymbol{p}$ 在偏移后满足约束：

$$C(\boldsymbol{p} + \Delta \boldsymbol{p}) = 0 \tag{16}$$

对约束函数在 $\boldsymbol{p}$ 处进行展开：

$$C(\boldsymbol{p}+\Delta \boldsymbol{p}) = C(\boldsymbol{p}) + \nabla_p C(\boldsymbol{p}) \cdot \Delta \boldsymbol{p} + o(\Delta \boldsymbol{p}) = 0 \tag{17}$$

将 $\Delta \boldsymbol{p}$ 限定在 $\nabla_p C(\boldsymbol{p})$ 的方向上可以保持动量、角动量守恒：

$$\Delta \boldsymbol{p} = \lambda \nabla_p C(\boldsymbol{p}) \tag{18}$$

联立可得：

$$\Delta \boldsymbol{p} = -\frac{C(\boldsymbol{p})}{\left|\nabla_p C(\boldsymbol{p})\right|^2} \nabla_p C(\boldsymbol{p}) \tag{19}$$

对于单个质点来说：

$$\Delta \boldsymbol{p_i} = -\frac{C(\boldsymbol{p})}{\sum_j \left|\nabla_{p_j} C(\boldsymbol{p})\right|^2} \nabla_{p_i} C(\boldsymbol{p}) \tag{20}$$

考虑到质点的 $\Delta \boldsymbol{p_i}$ 应与质量的倒数 $\omega_i = \frac{1}{m_i}$ 成反比

$$\Delta \boldsymbol{p_i} = -\frac{C(\boldsymbol{p})}{\sum_j \omega_j \left|\nabla_{p_j} C(\boldsymbol{p})\right|^2} \omega_i \nabla_{p_i} C(\boldsymbol{p}) \tag{21}$$

对于布料，两个质点之间的距离约束可以表述为：

$$C(\boldsymbol{p_1}, \boldsymbol{p_2}) = |\boldsymbol{p_1} - \boldsymbol{p_2}| - L \tag{22}$$

梯度为：

$$\begin{aligned} \nabla_{p_1} C(\boldsymbol{p_1}, \boldsymbol{p_2}) &= \frac{\boldsymbol{p_1} - \boldsymbol{p_2}}{|\boldsymbol{p_1} - \boldsymbol{p_2}|} \\ \frac{C(\boldsymbol{p})}{\sum_j \omega_j \left|\nabla_{p_j} C(\boldsymbol{p})\right|^2} &= \frac{|\boldsymbol{p_1} - \boldsymbol{p_2}| - L}{2} \end{aligned} \tag{23}$$

约束投影后的偏移为：

$$\Delta \boldsymbol{p_1} = -\frac{1}{2}(|\boldsymbol{p_1} - \boldsymbol{p_2}| - L)\frac{\boldsymbol{p_1} - \boldsymbol{p_2}}{|\boldsymbol{p_1} - \boldsymbol{p_2}|} \tag{24}$$

以下为两种迭代的方法，其中 Gauss-Seidel 遍历全部约束，难以进行并行。

Projection (by Gauss-Seidel)

For $k = 0 \ldots K$
For every edge $e = \{i, j\}$
$\mathbf{x}_i \leftarrow \mathbf{x}_i - \frac{1}{2}(\|\mathbf{x}_i - \mathbf{x}_j\| - L_e)\frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}$
$\mathbf{x}_j \leftarrow \mathbf{x}_j + \frac{1}{2}(\|\mathbf{x}_i - \mathbf{x}_j\| - L_e)\frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}$

图 6 PBD(Gauss-Seidel)

Jacobi 方法在遍历全部约束后统一更新。

Projection (by Jacobi)

For $k = 0 \ldots K$
For every vertex i
$\mathbf{x}_i^{new} \leftarrow \mathbf{0}$
$n_i \leftarrow 0$
For every edge $e = \{i, j\}$
$\mathbf{x}_i^{new} \leftarrow \mathbf{x}_i^{new} + \mathbf{x}_i - \frac{1}{2}(\|\mathbf{x}_i - \mathbf{x}_j\| - L_e)\frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}$
$\mathbf{x}_j^{new} \leftarrow \mathbf{x}_j^{new} + \mathbf{x}_j + \frac{1}{2}(\|\mathbf{x}_i - \mathbf{x}_j\| - L_e)\frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}$
$n_i \leftarrow n_i + 1$
$n_j \leftarrow n_j + 1$
For every vertex i
$\mathbf{x}_i \leftarrow (\mathbf{x}_i^{new} + \alpha \mathbf{x}_i)/(n_i + \alpha)$

图 7 PBD(Jacobi)

对于两个约束 Ci 和 Cj，Gauss-Seidel 和 Jacobi 方法的直观对比如下

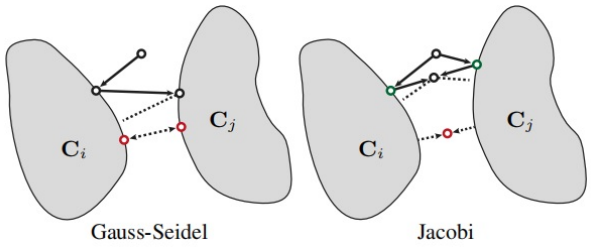

图 8 Gauss-Seidel-Jacobi

PBD 方法实际上是把弹簧质点系统中的弹力变为了约束，系统的刚度取决于迭代的次数，迭代次数越多时约束越强，系统的刚度也就越大，然而很难得到一个物体的力学性质和迭代次数的关系。与隐式欧拉法相比，PBD 优化的目标是 $E(x)$，这使得它忽略了惯性项 $\frac{1}{2(\Delta t)^2}(x-\bar{x})^T M(x-\bar{x})$，同时它不是基于物理推导出来的，不符合物理规律。

2 常见的线性方程组求解方法

2.1 直接法：LU 分解、LDLT 分解、Cholesky 分解

- 代价高，得到精确解
- 对矩阵限制少
- 适合 CPU 计算

2.2 迭代法：Jacobi

- 如果需要得到精确解的话代价大，但是可以根据容差控制
- 要让方法收敛的话，对矩阵有比较严格的限制（例如正定）
- 可使用 GPU 并行
- 实现比较容易
- 有很多加速算法

The Jacobi Method

$\Delta\mathbf{x} \leftarrow \mathbf{0}$

For $k = 0 \ldots K$

$\mathbf{r} \leftarrow \mathbf{b} - \mathbf{A}\Delta\mathbf{x}$

If $\|\mathbf{r}\| < \varepsilon$ break

$\Delta\mathbf{x} \leftarrow \Delta\mathbf{x} + \alpha\mathbf{D}^{-1}\mathbf{r}$

Residual error

Convergence condition ε

Update by $\mathbf{D}$, the diagonal of $\mathbf{A}$

图 9 Jacobi

3 基于块式梯度下降的隐式欧拉法[1]

在隐式欧拉法中，最后需要求解一个优化问题，除了用牛顿法外，还可以用块式梯度下降的方法，与牛顿法相比它们收敛到同一结果，虽然在渐进收敛速度上比牛顿法慢，但在不追求精度的情况下，并不需要迭代到最终收敛，这个方法在单步迭代的速率上快于牛顿法，因此在初始的几步迭代中能够快速减小误差同时取得与牛顿法相似的结果。

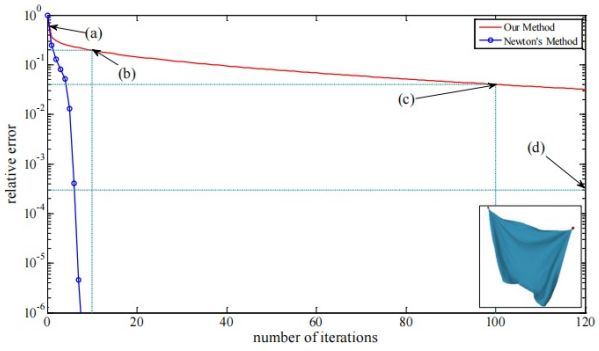

图 10 块式梯度下降与牛顿法

隐式欧拉法的优化目标为：

$$\Delta \boldsymbol{p_1} = -\frac{1}{2}(|\boldsymbol{p_1}-\boldsymbol{p_2}|-L)\frac{\boldsymbol{p_1}-\boldsymbol{p_2}}{|\boldsymbol{p_1}-\boldsymbol{p_2}|} \tag{25}$$

通常来说，外力的势能都不是关于位置的线性函数，但可以将其拆分为线性部分和非线性部分，将非线性部分用显式欧拉法计算，从而使整体变为线性方程，利用坐标下降的方法，交替优化非线性部分和整体。

利用胡克定律 (Hooke's law) 的另一形式：

$$(||\mathbf{p}_1-\mathbf{p}_2||-r)^2 = \min_{||\mathbf{d}||=r} ||(\mathbf{p}_1-\mathbf{p}_2)-\mathbf{d}||^2 \tag{26}$$

对于弹簧的弹性势能：

$$\frac{1}{2}\sum_{i=1}^{s} k_i||\mathbf{p}_{i_1}-\mathbf{p}_{i_2}-\mathbf{d}_i||^2 = \frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{L}\mathbf{x} - \mathbf{x}^{\mathsf{T}}\mathbf{J}\mathbf{d} \tag{27}$$

其中：

$$\mathbf{L} = \left(\sum_{i=1}^{s} k_i\mathbf{A}_i\mathbf{A}_i^{\mathsf{T}}\right)\otimes\mathbf{I}_3,\ \mathbf{J} = \left(\sum_{i=1}^{s} k_i\mathbf{A}_i\mathbf{S}_i^{\mathsf{T}}\right)\otimes\mathbf{I}_3 \tag{28}$$

考虑到其它保守力的存在：

$$E(\mathbf{x}) = \min_{d\in U}\ \frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{L}\mathbf{x} - \mathbf{x}^{\mathsf{T}}\mathbf{J}\mathbf{d} + \mathbf{x}^{\mathsf{T}}\mathbf{f}_{ext} \tag{29}$$

得到最终的优化问题：

$$\min_{\mathbf{x}\in\mathbf{R}^3,\mathbf{d}\in U}\ \frac{1}{2}\mathbf{x}^{\mathbf{T}}(\mathbf{M}+\Delta t^2\mathbf{L})\mathbf{x} - \Delta t^2\mathbf{x}^{\mathbf{T}}\mathbf{J}\mathbf{d} + \mathbf{x}^{\mathbf{T}}\mathbf{b} \tag{30}$$

从 x 的初始猜测（显式欧拉法的结果）开始，交替优化 x 和 d，其中优化 x 时是一个凸二次最小化问题，$\mathbf{M}+\Delta t^2\mathbf{L}$ 是一个在迭代中不变的正定矩阵，可以对其预先计算其 Cholesky 分解从而加快速度。

这个方法与 PBD 相比，正确地考虑了惯性项 $\frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{M}\mathbf{x}$ 的存在，同时弹簧的刚性也被包括在 **L** 和 **J** 中。

对于阻尼的作用，可以简单将其转换为速度的百分比衰减：

$$\tilde{\mathrm{v}}_n = \alpha \mathrm{v}_n \tag{31}$$

对于碰撞，可以直接将 x 移动到无碰撞的区域，这部分由碰撞响应线程实现。

4 投影动力学 (Projective Dynamics)[2]

PD 是一个对物理系统进行隐式欧拉积分的新方法，可以将约束转换为势能部分，将势能表示为两部分的和：

$$W(\mathbf{q},\mathbf{p}) = d(\mathbf{q},\mathbf{p}) + \delta_{\mathrm{E}}(\mathbf{p}) \tag{32}$$

其中 $\delta_{\mathrm{E}}(\mathbf{p})$ 形式化了 p 应该位于约束流形上的要求，$d(\mathbf{q},\mathbf{p})$ 用于衡量 p 和 q 之间的距离，对 p 最小化方程 (30) 等价于寻找 q 在约束流形上的投影，因此对于材料模型 Ψ 产生的弹性势 $W(\mathbf{q}) = \Psi(\mathbf{E}(\mathbf{q})) = \min_{\mathbf{P}} W(\mathbf{Q},\mathbf{p})$

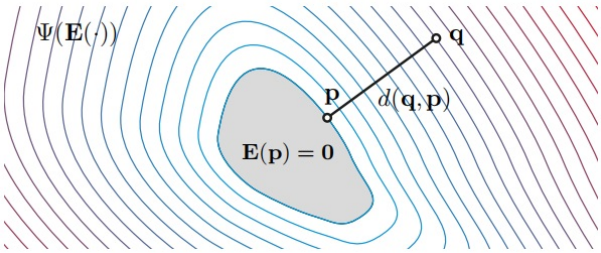

图 11 材料弹性势

将势能代入隐式欧拉积分中，采取交替优化的方法，先基于当前的 q 求出 p，再根据新的 p 更新 q，反复迭代直到收敛。

$d(\mathbf{q},\mathbf{p})$ 可取为简单的线性函数，这样在 p 固定时优化 q 变为线性优化问题，从而可以快速计算，而非线性的部分在优化 p 的过程中进行，这与 [1] 的思想类似。

例如使用以下的二次距离度量：

$$W(\mathbf{q},\mathbf{p}) = \frac{w}{2}\|\mathrm{A}\mathbf{q}-\mathbf{B}\mathbf{p}\|_F^2 + \delta_{\mathrm{C}}(\mathbf{p}) \tag{33}$$

在优化 p 时，优化问题为：

$$\min_{\mathbf{p}_i} \frac{w_i}{2}\|\mathbf{A}_i\mathbf{S}_i\mathbf{q}-\mathbf{B}_i\mathbf{p}_i\|_F^2 + \delta_{\mathrm{C}_i}(\mathbf{p}_i) \tag{34}$$

在优化 q 时，由于为线性优化，由极值点梯度为 0 得到

$$(\frac{\mathrm{M}}{h^2} + \sum_i w_i \mathrm{S}_i^T\mathbf{A}_i^T\mathbf{A}_i\mathbf{S}_i)\mathbf{q} = \frac{\mathrm{M}}{h^2}\mathbf{s}_n + \sum_i w_i\mathbf{S}_i^T\mathbf{A}_i^T\mathbf{B}_i\mathbf{p}_i \tag{35}$$

此时可以选用直接法或迭代法求解该线性方程。

5 SOR 在 PBD 算法上的应用[3]

在优化视角下，PBD 算法等价于：

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\Delta x^T M \Delta x \\ &\text{subject to} \quad C_i(x+\Delta x) = 0, \quad i = 1,\cdot\cdot,n \end{aligned} \tag{36}$$

也就是寻找满足约束条件的动能的最小变化，这与高斯最小拘束原理一致。

由于约束通常是非线性非凸函数，使用顺序二次规划 (SQP) 进行，求解二次最小化序列：

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\Delta\mathbf{x}^T\mathbf{M}\mathbf{x} \\ &\text{subjec to} \quad J\Delta x = \mathbf{b} \end{aligned} \tag{37}$$

这种类型的问题可以转化为以下的最优化条件：

$$\begin{aligned} M\Delta x &= \mathrm{J}^T\lambda \\ J\Delta x &= \mathbf{b} \end{aligned} \tag{38}$$

其中第一个式子由拉格朗日乘数法得到，第二个式子为可行性条件。

从中消除 Δx 得到：

$$\left[\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\right]\lambda = \mathbf{b} \tag{39}$$

这是一个线性方程，可以使用迭代法求解。

为了解决 Gauss-Seidel 方法不能并行的问题，可以先并行地处理每个约束，在迭代结束时将它们的均值作为最终的结果，并引入一个全局参数控制 SOR 的速率：

$$\Delta\tilde{\mathbf{x}}_i = \frac{\omega}{n_i}\Delta\mathbf{x}_i \tag{40}$$

它不能保证动量守恒，但一般视觉误差并不明显。

利用约束平均可以避免迭代法在矩阵不满足要求时不收敛的问题，通常 $1 \leqslant \omega \leqslant 2$，不需要额外的欠松弛 ($\omega < 1$)，因为约束平均已经足以避免发散。

在实践中，通常希望某些类型的约束优先于其它类型的约束，可以分组处理约束，在处理下一个约束组前，将累计的位移变化立即应用到当前位移上。

6 PD 算法的 Chebyshev 加速[4]

注意到PD算法的收敛性与线性系统高度相似，因此可以使用 Chebyshev 加速来加速 PD 算法的收敛，其中谱半径从其收敛速度中估计得到。

Chebyshev 加速方法易于实现，且与 GPU 加速兼容，其算法流程如下

Algorithm 1 Chebyshev_PD_Solve($\mathbf{q}_t, \mathbf{v}_t, \rho, h, K$)

$\mathbf{q}^{(0)} \leftarrow \mathbf{s}_t \leftarrow \mathbf{q}_t + h\mathbf{v}_t + h^2\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}$;
for $k = 0 \ldots K-1$ **do**
  for each constraint c **do**
    $\mathbf{p}_c$ = Local_Solve($c, \mathbf{q}^{(k)}$);
  $\hat{\mathbf{q}}^{(k+1)} \leftarrow$ Jacobi_Solve($\mathbf{s}_t, \mathbf{q}^{(k)}, \mathbf{p}_1, \mathbf{p}_2, \ldots$);
  if $k < S$ **then** $\omega_{k+1} \leftarrow 1$;
  if $k = S$ **then** $\omega_{k+1} \leftarrow 2/(2-\rho^2)$;
  if $k > S$ **then** $\omega_{k+1} \leftarrow 4/(4-\rho^2\omega_k)$;
  $\mathbf{q}^{(k+1)} = \omega_{k+1}\left(\gamma\left(\hat{\mathbf{q}}^{(k+1)} - \mathbf{q}^{(k)}\right) + \mathbf{q}^{(k)} - \mathbf{q}^{(k-1)}\right) + \mathbf{q}^{(k-1)}$;
$\mathbf{q}_{t+1} \leftarrow \mathbf{q}^{(K)}$;
$\mathbf{v}_{t+1} \leftarrow (\mathbf{q}_{t+1} - \mathbf{q}_t)/h$;

图 12 chebyshev

Chebyshev 加速方法还需要谱半径 ρ，设估计的谱半径为 $\hat{\rho}$，当 $\hat{\rho} = 0$ 时，这个方法退化为不加速的迭代法，当 $0 < \hat{\rho} < \rho$ 时，随着 $\hat{\rho}$ 的增大收敛速度提升，一旦 $\rho < \hat{\rho}$，会产生震荡，因此需要准确估计谱半径大小。

由于 PD 有类似于线性系统的收敛性，把 ρ 作为每个问题中的常数，通过预模拟以估计 ρ。首先将 ρ 初始化为 $\left\|\mathbf{e}^{(K)}\right\|_2 / \left\|\mathbf{e}^{(K-1)}\right\|_2$，其中误差被定义为 $\mathrm{e}^{(k)} = \nabla\epsilon(\mathbf{q}^{(k)})$，然后多次调整 ρ 并进行仿真，找到收敛速度最大的 ρ。

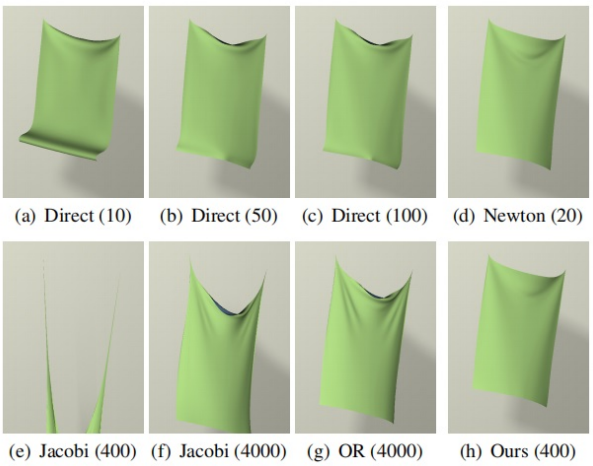

图 13 Chebyshev 加速对比

如 (c) 所示直接求解器在 100 次迭代后仍存在卷边伪影，同时 Jacobi 求解器收敛缓慢，而 (h) 中经过 Chebyshev 加速的 Jacobi 求解器与牛顿模拟方法高度相似。

7 XPBD[5]

PBD 算法中时间步长和迭代次数决定了物体的刚性，在复杂的系统中，由于参数之间的耦合，难以单独改变一个物体的刚性。在这篇文章中，提出了拓展的 PBD 算法（XPBD），显式引入弹性势能，从而通过弹性势能来确定物体刚性。

弹性势能场为：

$$U(\mathbf{x}) = \frac{1}{2}\mathbf{C}(\mathbf{x})^T\alpha^{-1}\mathbf{C}(\mathbf{x}) \tag{41}$$

由此所产生的弹性力为：

$$\mathbf{f}_{elastic} = -\nabla_{\mathbf{x}}U^T = -\nabla\mathbf{C}^T\alpha^{-1}\mathbf{C} \tag{42}$$

引入拉格朗日乘子将其分解得到：

$$\lambda_{elastic} = -\tilde{\alpha}^{-1}\mathbf{C}(\mathbf{x}) \tag{43}$$

其中 $\tilde{\alpha} = \frac{\alpha}{(\Delta t)^2}$

得到离散的运动方程：

$$\begin{aligned} &\mathrm{M}(\mathbf{x}^{n+1} - \tilde{\mathbf{x}}) - \nabla\mathbf{C}(\mathbf{x}^{n+1})^T\lambda^{n+1} = \mathrm{O} \\ &\mathbf{C}(\mathbf{x}^{n+1}) + \tilde{\alpha}\lambda^{n+1} = \mathbf{0} \end{aligned} \tag{44}$$

将两式分别记为 g 和 h，需要找到 x 和 λ 满足：

$$\begin{aligned} &\mathbf{g}(\mathbf{x}, \lambda) = \mathbf{0} \\ &\mathbf{h}(\mathbf{x}, \lambda) = \mathbf{0} \end{aligned} \tag{45}$$

得到以下线性牛顿子问题：

$$\begin{bmatrix} \mathbf{K} & -\nabla\mathbf{C}^T(\mathbf{x}_i) \\ \nabla\mathbf{C}(\mathbf{x}_i) & \tilde{\boldsymbol{\alpha}} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \mathbf{g}(\mathbf{x}_i, \boldsymbol{\lambda}_i) \\ \mathbf{h}(\mathbf{x}_i, \boldsymbol{\lambda}_i) \end{bmatrix} \tag{46}$$

现在使用两个近似方法以简化计算的实现，假设 $\mathbf{K} \approx \mathbf{M}$，并且 $\mathbf{g}(\mathbf{x}_i, \boldsymbol{\lambda}_i) \approx 0$：

$$\begin{bmatrix} \mathbf{M} & -\nabla\mathbf{C}^T(\mathbf{x}_i) \\ \nabla\mathbf{C}(\mathbf{x}_i) & \tilde{\boldsymbol{\alpha}} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} 0 \\ \mathbf{h}(\mathbf{x}_i, \boldsymbol{\lambda}_i) \end{bmatrix} \tag{47}$$

最终得到新的迭代方程：

$$\begin{aligned} \Delta\mathbf{x} &= \mathbf{M}^{-1}\nabla\mathrm{C}(\mathbf{x}_i)^T\Delta\lambda \\ \Delta\lambda_j &= \frac{-C_j(\mathbf{x}_i) - \bar{\alpha}_j\lambda_{ij}}{\nabla C_j \mathrm{M}^{-1}\nabla(C_j^T + \tilde{\alpha}_j} \end{aligned} \tag{48}$$

XPBD 的算法流程如下

Algorithm 1 XPBD simulation loop

predict position $\tilde{\mathbf{x}} \Leftarrow \mathbf{x}^n + \Delta t\mathbf{v}^n + \Delta t^2\mathbf{M}^{-1}\mathbf{f}_{ext}(\mathbf{x}^n)$
initialize solve $\mathbf{x}_0 \Leftarrow \tilde{\mathbf{x}}$
initialize multipliers $\boldsymbol{\lambda}_0 \Leftarrow \mathbf{0}$
while $i < solverIterations$ **do**
  for all constraints **do**
    compute $\Delta\lambda$ using Eq (18)
    compute $\Delta\mathbf{x}$ using Eq (17)
    update $\lambda_{i+1} \Leftarrow \lambda_i + \Delta\lambda$
    update $\mathbf{x}_{i+1} \Leftarrow \mathbf{x}_i + \Delta\mathbf{x}$
  end for
  $i \Leftarrow i + 1$
end while
update positions $\mathbf{x}^{n+1} \Leftarrow \mathbf{x}_i$
update velocities $\mathbf{v}^{n+1} \Leftarrow \frac{1}{\Delta t}(\mathbf{x}^{n+1} - \mathbf{x}^n)$

图 14 XPBD 算法

8 多重网络[6]

多重网格法是迭代方法的一种，可用于求解线性方程组。

一般的迭代法在求解线性方程组 $A\phi = b$ 基于固定的步长进行迭代得到 ϕ^n，对残差 $r = b - A\phi^n$ 进行傅里叶变换可以把它们分解成一系列不同频率的残差，在迭代中对于频率较高的残差消除的较快，对频率低的残差消除的慢。可以通过增大步长来等价地提高频率，从而消除低频误差，但步长大时精度较低，因此可以在不同的步长上交替进行，在保持精度的同时加快误差消除，这被称为多重网格法。

先在细网格上迭代使得频率大的误差衰减，剩下误差中频率较小的部分；再将误差中频率较小的部分作为方程组的右侧，在粗网格上迭代，使得频率较小的误差也能快速地衰减；最后为了保证精度，将粗网格上迭代的结果和细网格上的结果加总之后，作为初始猜测值再在细网格上迭代。如此循环往复。因为先由细到粗，后由粗到细，网格转换的路径形似 V 字，所以该方法被称为 V-Cycle Multigrid。除此之外还有 F-cycle multigrid 和 W-cycle multigrid。他们的基本思想相同。

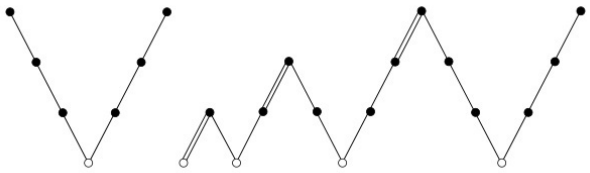

图 15 multigrid

图中黑色圆点代表平滑，白色圆点代表精确求解器，\和/代表限制和插值，//代表高阶插值，限制和插值用于粗细网格结果之间的映射。

非线性 FMG(full multigrid) 的工作结构如下：

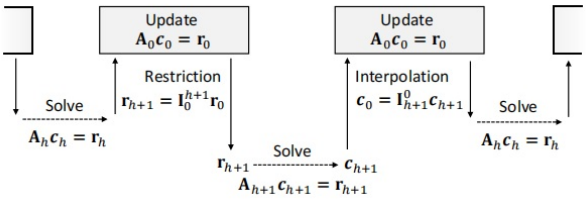

图 16 非线性 FMG

利用多重网格，可以加速布料仿真中迭代法求解线性方程收敛的速度。

9 基于深度学习的服装 CG 生成[7]

衣物的生成通常通过物理模拟来实现，通过设置不同的物理参数，可以生成不同材质、形状的衣服。但是，为一个 CG 来生成衣服需要对于每一帧动作指定对应的物理参数，而这个参数和动作，前后帧都是有关系的。

为了减少参数设置的工作量，需要在关键帧之间自动生成对应的衣服，但关键帧之间的运动可能非常复杂，衣服的生成无法直接通过插值得到，因此通过深度学习的方式找到衣服所对应的潜在空间，从而将衣物应用到不同的动作上。

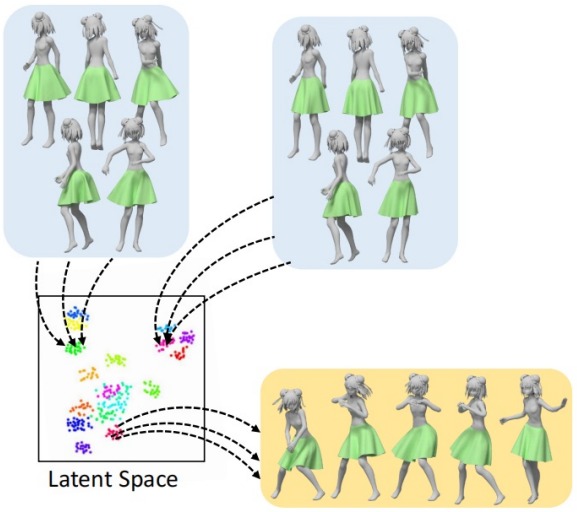

图 17 潜在空间

一个潜在空间上的点可以在不同的动作下对应于不同的衣物。

首先使用多层感知机 (MLP) 对服装网格进行编码与解码，将其进行降维

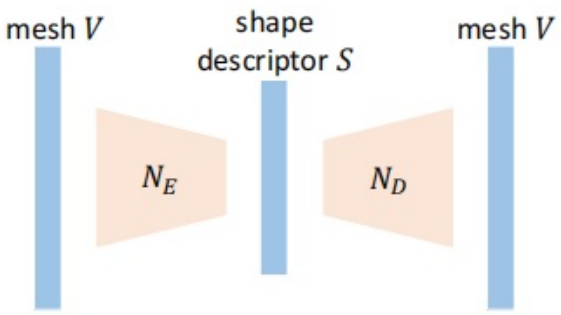

图 18 多层感知机

它的损失函数为：

$$||V-N_D(N_E(V))||_2+\omega\cdot||\Delta(V)-\Delta(N_D(N_E(V)))|_2 \tag{49}$$

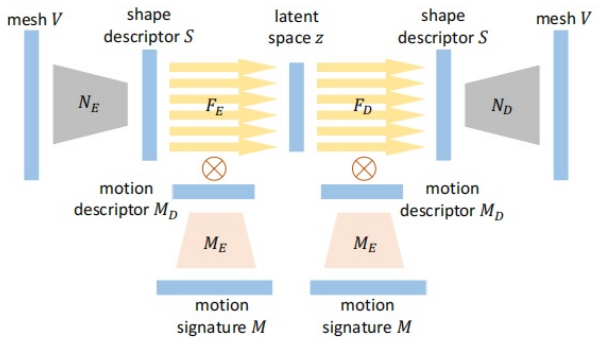

图 19 auto-encode

通过自动编码器进行服装网格与形状描述符之间的编码与解码，运动标签与运动描述符之间的编码与解码，将服装描述符和运动描述符降维到潜在空间中，其中运动描述符作为形状描述符到潜在空间的编码器的子网权重，再通过新的运动描述符重新变换回服装网络。

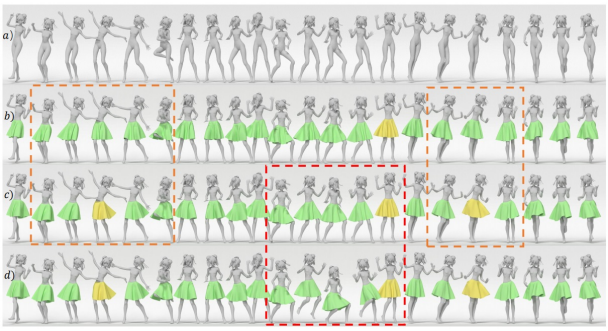

图 20 animation

这是一个半自动的服装动画创作工作流程。给定一个身体动画序列 (a)，艺术家选择一个关键帧（用黄色标记），并输入该帧 (b) 的服装形状，系统学习该设计的内在属性，并自动将该设计传播到其他帧（用绿色标记）。艺术家可以通过插入新的关键帧（用黄色标记）和编辑关键帧的服装形状来进一步调整服装动画 (c)。然后，通过链接关键帧与合理的过渡（用橙色框突出显示），可以自动更新动画序列。工作流程还允许艺术家在构图过程中修改身体动画（用红色框标记），因为我们的系统可以从底层的身体运动推断出服装的形状 (d).

10 Incremental Potential Contact[8]

为了处理仿真中产生的碰撞，可以将碰撞视为一种约束，从而转化为优化问题。一般的隐式欧拉优化目标为：

$$\begin{aligned} E(x,x^t,v^t) &= \frac{1}{2}(x-\hat{x})^T\mathcal{M}(x-\hat{x}) - h^2x^Tf_d + h^2\Psi(x) \\ \hat{x} &= x^t + hv^t + h^2M^{-1}f_e \\ x^{t+1} &= \arg\min_x E(x,x^t,v^t), \quad x^\tau \in \mathcal{A} \end{aligned} \tag{50}$$

其中 $\mathcal{A}$ 为不发生碰撞的空间区域

IPC 方法将约束条件融合到目标函数中：

$$\arg\min_x E(x) = \frac{1}{2}\|x-x^*\|_M^2+h^2\Psi(x)+\kappa\sum_{k\in C} b(d_k(x)) \tag{51}$$

其中 b 是一个障碍函数，当 $0 < d < \hat{d}$ 时靠的越近会受到越大的阻力

$$b(d_k(x)) = \begin{cases} -(d-\hat{d})^2 \ln(\frac{d}{\hat{d}}), & 0 < d < \hat{d} \\ 0, & d \geqslant \hat{d} \end{cases} \tag{52}$$

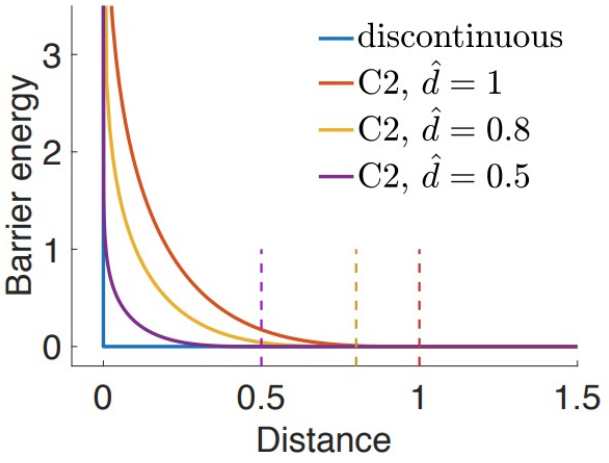

图 21 障碍势能

11 高分辨率布料褶皱模拟[9]

传统的布料仿真使用非结构化的三角形网络，但这会带来巨大的内存开销和计算成本，在较高分辨率下三角形网络的灵活性不再重要，可以使用规则网络，提高内存访问的效率。

在高分辨率下模拟布料褶皱的流程如下

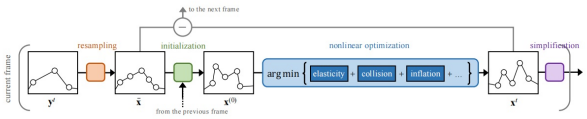

图 22 褶皱模拟流程

输入一个非结构化的网络，通过一个规则网络重新采样，然后运行一个初始化步骤来估计网格上的褶皱，通过非线性优化来生成褶皱的细节，最终将褶皱简化以便下一步处理。

为了提升收敛速度，使用 GPU 来并行处理，可以在不重叠的 GPU 线程块中使用共享内存，这是由网格分区所自然创建的。但在每次迭代中每个块都需要将其部分从全局内存加载到共享内存中，如果块能够覆盖全部顶点，那就可以一次性加载共享内存，在顶点较多时，如果我们只为同一个不重叠的 GPU 块中的线程运行内部迭代，就可以重用只加载一次的共享内存数据。

利用 GPU 加速的块式梯度下降算法如下

ALGORITHM 1: A Block-Based Descent Method

Input: The original mesh $\bar{\mathbf{x}}$, the block width W, the number of outer iterations K, the number of inner iterations L.

$\mathbf{x}^{(0)} \leftarrow$Initialize($\bar{\mathbf{x}}$, ...);
for $k = 0...K-1$ **do**
  for *each non-overlapping thread block B* **do**
    __shared__ X[W+4][W+4];
    Global_to_Share_Transfer(X, $\mathbf{x}^{(k)}$, B, W);
    __syncthreads();
    $\mathbf{x}_B^{[0]} \leftarrow$ X[2:W+1][2:W+1];
    for $l = 0...L-1$ **do**
      $\mathbf{x}_B^{[l+1]} \leftarrow \mathbf{x}_B^{[l]} - \alpha \mathbf{P}_B^{-1}(\mathbf{X})\mathbf{g}_B(\mathbf{X})$;
      X[2:W+1][2:W+1] $\leftarrow \mathbf{x}_B^{[l+1]}$;
      __syncthreads();
    end
    $\text{temp}_B \leftarrow$ X[2:W+1][2:W+1];
  end
  $\mathbf{x}^{(k+1)} \leftarrow$ temp;
end
return $\mathbf{x}^{(K)}$;

图 23 gpu

12 总结

布料是一种在形变时会产生弹性势能的物体，在布料仿真时需要计算形变时弹性势能对状态的影响，通常仅考虑相邻顶点之间的弹性势能。布料仿真的模型较为单一，因此近十年的研究方向主要在于如何设计新的算法或是采用一些优化策略提升仿真速度与效果，在此基础上提出了 PD、XPBD、IPC 等方法构造不同的优化目标以取得更好的效果，以及应用数值计算方法（如块式梯度下降、SOR、Chebyshev 加速）以加速原有的算法，还有基于深度学习半自动生成服装 CG 的新的研究方向。

参考文献

[1] Liu T , Bargteil A W , O'Brien J F , et al. Fast Simulation of Mass-Spring Systems[J]. ACM Transactions on Graphics, 2013, 32(6):209:1-7.

[2] Sofien, Bouaziz, Sebastian, et al. Projective dynamics: fusing constraint projections for fast simulation[J]. ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2014, 2014, 33(4).

[3] Macklin M , Mueller M , Chentanez N , et al. Unified particle physics for real-time applications[J]. Acm Transactions on Graphics, 2014, 33(4CD):1-12.

[4] Wang H . A Chebyshev Semi-Iterative Approach for Accelerating Projective and Position-based Dynamics[J]. ACM Transactions on Graphics, 2015, 34(6cd):246.1-246.9.

[5] Macklin M , M Müller, Chentanez N . XPBD: Position-Based Simulation of Compliant Constrained Dynamics[C]// Motion In Games 2016. ACM, 2016.

[6] Wang Z , Wu L , Fratarcangeli M , et al. Parallel Multigrid for Nonlinear Cloth Simulation[J]. Computer Graphics Forum, 2018, 37(7):131-141.

[7] Wang T Y , Shao T , Fu K , et al. Learning an intrinsic garment space for interactive authoring of garment animation[J]. ACM Transactions on Graphics, 2019, 38(6):1-12.

[8] Li M , Ferguson Z , Schneider T , et al. Incremental Potential Contact: Intersection-and Inversion-free, Large-Deformation Dynamics[J]. ACM Transactions on Graphics, 2020, 39(4):20.

[9] Wang H . GPU-based simulation of cloth wrinkles at sub-millimeter levels[J]. ACM Transactions on Graphics, 2021, 40(4):1-14.

Cloth Simulation: A Decade of Acceleration

Chen Wang

(University of Science and Technology of China, School of Computer Science and Technology, PB20111696)

Abstract: This article mainly introduces the common methods of cloth simulation and the cutting-edge research of cloth simulation in the past ten years, including performance optimization of existing algorithms, new simulation algorithms, and parallel implementation of algorithms on GPU, and summarizes the current frontier direction of cloth simulation.

Key words：Cloth Simulation