中国科学技术大学
University of Science and Technology of China

本 科 毕 业 论 文

题　目 布料实时仿真技术研究

英　文 **Research on Real-time Cloth Simulation**

题　目 **Technology**

院　系 少年班学院

姓　名 程诗涵　学　号 **PB19000216**

导　师 刘利刚教授

日　期 **2023 年 6 月 2 日**

摘　要

布料的实时仿真一直是计算机图形学中的热门研究问题。已有的方法主要分为基于物理的仿真（Pysical Based Simulation）和其余非物理的仿真方法两大类。诚然，基于物理仿真的方法能够在真实性上做到极好的还原，但是随着场景的复杂度上升，很难同时满足实时性。非物理方法虽总能保证实时，在真实性上则有欠缺。本文探究了前沿的布料仿真技术，概括设计了一套仿真方法。该仿真方法一方面可以弥补非物理仿真方法真实感不足的短板，另一方面进一步的提升了实时性。

本文首先介绍了基于位置的动态变化数值算法 PBD（Position-Based Dynamics）和其真实感改进算法 XPBD（Extended Position-Based Dynamics）。其次，本文总结整理了仿真算法的本质，将非物理布料结算步骤归结为多限制的优化问题。针对该优化问题，我们设计了两个优化方向的加速方法。针对雅可比迭代更新方法（Jacobi Iteration），本文采用了切比雪夫加速法；针对高斯-塞达尔迭代加速方法（Gauss Seidel Iteration），则采用了基于图聚类的 GPU 加速法。为了比较不同仿真算法的算法性能，本文一方面构建不同的测试样例定性的分析算法仿真效果；另一方面设计了两套衡量标准定量的对不同算法性能机型测试。

接着我们详细论述了算法的实现步骤，包括场景的构建，引擎的实现细节，评价的指标构建等等。

针对不同加速方法的实验结果，本文对加速算法的性能进行了探讨，并研究了可以进一步深入加速的方向。

关键词：布料仿真；实时仿真；数值迭代

ABSTRACT

Real-time cloth simulation has always been a hot research topic in computer graphics. Existing methods can be mainly classified into two categories: physical-based simulation and other non-physical-based simulation methods. While physical-based simulation methods achieve excellent realism, it becomes challenging to maintain real-time performance as the complexity of the scene increases. Non-physical methods, on the other hand, can ensure real-time performance but lack in realism. This paper explores cutting-edge cloth simulation technology and presents a comprehensive design of a simulation method. This simulation method aims to address the shortcomings of non-physical-based simulation methods in terms of realism while further enhancing real-time performance.

Firstly, this paper introduces the Position-Based Dynamics algorithm (PBD) and its realism-enhancing algorithm, Extended Position-Based Dynamics (XPBD). Secondly, the paper summarizes the essence of simulation algorithms and categorizes the non-physical cloth settlement steps as a multi-constraint optimization problem. To address this optimization problem, two acceleration methods are designed. The Chebyshev acceleration method is employed for the Jacobi Iteration, while the Graph Clustering GPU acceleration method is used for the Gauss Seidel Iteration. In order to compare the performance of different simulation algorithms, this paper constructs various test cases for qualitative analysis of algorithm simulation results. Additionally, two sets of quantitative evaluation criteria are designed to assess the performance of different algorithms.

Furthermore, the paper provides a detailed discussion on the implementation steps of the algorithms, including scene construction, engine implementation details, and the development of evaluation metrics.

Based on the experimental results of different acceleration methods, this paper discusses the performance of the acceleration algorithms and explores directions for further acceleration.

Key Words: Cloth Simulation, Real-time Simulation, Numerical Iteration

目 录

第一章　绪论

第一节　引言

仿真是计算机图形学中的一个重要分支，通过更新每一帧 3D 物体的顶点位置和速度并绘制，我们可以利用计算机预测 3D 物体之间的运动与交互。总的来讲，仿真界主要关注三个指标，即仿真场景的规模、仿真的速度和仿真的真实性。而这三个指标往往相互掣肘。在电影行业的仿真中，为了规模和真实性我们往往会舍弃实时性，但是在游戏行业，甚至是新型崛起的元宇宙中，实时性对于用户体验是不可或缺的[1] [2]。

图 1.1　真实感布料仿真。可以看到局部放大以后，仿真的布料表现出褶皱的复杂物理性质。（图片来源文献[3]）

布料仿真，作为仿真中的一个子方向，有着广阔的应用前景，如实时虚拟试穿等。但另一方面，布料的褶皱等外在表现又对真实感有更高的要求。这就对布料的实时仿真算法定下了更严格的标准，即如何在保证一定的真实感的前提下，进一步的提高仿真速度。在这里我们先简要回顾一下布料仿真的两个组成部分。

第一个部分是布料解算，即每一帧，布料解算器根据布料自身的约束以及上一帧计算的各个顶点的位置和速度，更新这一帧布料的各顶点位置和速度的物

理量。第二个部分是碰撞处理，由于布料中广泛出现自交，以及与其他物体的交互，需要引入碰撞处理避免穿模现象的发生。可以从这两个部分的介绍中看出，布料解算的速度决定了布料仿真的上界。

对于布料解算，则分为物理仿真方法和非物理仿真方法。前者通过离散物理公式和 PDE，逐帧计算每一帧顶点的速度和位置。后者则或者采取数值优化的方法，或者采取数据驱动的方法。由于物理仿真算法中总是涉及到大规模稀疏矩阵的求解，布料解算很难在实时的要求下完成。与之相反，非物理仿真算法通过各种近似或着使用空间换取时间，总是能达成实时的目标，因此，现代游戏中，总是利用非物理仿真算法，其中基于位置的动态变化数值算法（PBD）算法应用最为广泛。

然而，PBD 方法在布料的真实感仿真效果上仍然有所欠缺，而且布料的物理性质容易受到迭代次数的影响。因此本文调研并采用了 XPBD 方法。在不影响布料的实时解算的前提下，提高了布料仿真的真实感，并提高了布料物理性质相对于迭代次数的稳定性。

另一方面，对于大规模场景，即使是 PBD，也不能够满足实时性的需求，这对进一步的加速算法提出了需求，事实上，PBD 方法可以解构为一个多约束的迭代优化问题。多约束迭代优化问题有雅可比和高斯两种迭代求解的思路，区别在于是每一个约束求解后直接更新还是等待所有的约束全部完成求解后再一次性对所有变量更新。针对于此，本文提出了两种加速的优化迭代算法，Chebyshev 和图聚类 GPU 加速方法。

算法 1.1 Jacobi Iteration Pseudocode

input : initial guess $x^{(0)}$ to the solution, diagonal dominant matrix A, right-hand side vector b, convergence criterion
output: solution when convergence is reached

```
1  k ← 0
2  while convergence not reached do
3      for i ← 1 to n do
4          σ = 0
5          for j ← 1 to n do
6              if j ≠ i then
7                  σ ← σ + a_ij x_j^k
8              end
9              x_i^{k+1} = (b_i − σ)/a_ii
10         end
11     end
12     increment k
13 end
```

第一种算法（Chebyshev）可作用于多约束优化问题中的雅可比迭代。原来

算法 1.2 Gauss–Seidel Iteration Pseudocode

input : initial guess $x^{(0)}$ to the solution, diagonal dominant matrix A, right-hand side vector b, convergence criterion
output: solution when convergence is reached

```
Choose an initial guess x to the solution
while convergence not reached do
    for i ← 1 to n do
        σ = 0
        for j ← 1 to n do
            if j ≠ i then
                σ ← σ + a_ij x_j
            end
            x_i = (b_i − σ)/a_ii
        end
    end
    check if convergence is reached
end
```

的雅可比迭代累计下每一个变量在多约束中需要改变的值，在所有迭代结束后一次更新。Chebyshev 加速方法则通过将本帧需要的更新值与前几帧的更新值做加权平均，达到了加速收敛的目的。

第二种算法（图聚类 GPU 加速算法）则作用于多约束问题中的高斯迭代。高斯-塞德尔迭代对于单约束立即更新涉及变量的值。虽然高斯迭代比雅可比算法的迭代算法更快，但是由于多约束问题中约束之间涉及到的变量更新有重叠，因此不能直接使用 GPU 进行加速。基于此，本文使用了一种图聚类的 GPU 加速算法，实现了 GPU 加速的高斯迭代算法。

综上所述，本文的主要贡献可以概括为：

1. 调研布料的实时真实感仿真，使用 XPBD 算法保证布料仿真实时的同时仍保持稳定的物理属性；
2. 调研布料的进一步仿真加速，分别对于雅可比和高斯两种迭代算法对布料解算流程进行加速；
3. 提出了一套布料仿真加速效果的衡量标准，可以进一步拓展测试各种前沿的布料仿真加速算法。

第二节　相关工作

一、物理仿真方法

对于物理仿真方法而言，一般采用质点-弹簧模型。即在相邻的布料顶点之间装上弹簧，将布料运动过程中涉及到的物理方程离散化到布料三角网格中。此

时布料形变时内部复杂的的力的相互作用，被简化离散为一个个的弹簧对布料顶点的作用，让彼此独立的顶点得以通过弹簧链接进整个系统，从而实现相互影响。虽然说有很多显式求解算法，如 Baraff 等人[4]的显示欧拉积分方法。但是该方法由于算法中引入的人为仿真量衰减因素受到诟病，后续又有 Kharevych 等人[5]提出的辛欧拉积分方法和 Stern 等人[6]等人提出的半隐式积分方法。虽然说他们的工作克服了一些显示仿真方法带来的问题，但是依然无法根除显示仿真中的不稳定。下面，我们来介绍一下布料仿真中隐式仿真的形式。

不加赘述的，可以把布料的物理仿真问题转换成一个优化问题：

$$\boldsymbol{x}^{t+\Delta t} = \arg\min_{\boldsymbol{x}} F(\boldsymbol{x}), \quad F(\boldsymbol{x}) = \frac{1}{2\Delta t^2}(\boldsymbol{x}-\bar{\boldsymbol{x}})^T \boldsymbol{M}(\boldsymbol{x}-\bar{\boldsymbol{x}}) + E(\boldsymbol{x}) \tag{1.1}$$

其中 $\frac{1}{2\Delta t^2}(\boldsymbol{x}-\bar{\boldsymbol{x}})^T \boldsymbol{M}(\boldsymbol{x}-\bar{\boldsymbol{x}})$ 为布料系统的动能，$E(\boldsymbol{x})$ 布料系统的势能（这里势能只和顶点的位置有关），$\boldsymbol{M}$ 是顶点的质量矩阵。由于往往 $F(\boldsymbol{x})$ 是非线性函数，常用的方法是一些数值的迭代解法，它们的形式是：

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \alpha^{k+1}(\boldsymbol{A}^{k+1})^{-1}\frac{\partial F(x^k)}{\partial \boldsymbol{x}} \tag{1.2}$$

其中 $\boldsymbol{A}$ 矩阵的选取可以简单设为 Hessian 矩阵或者单位矩阵，此时即对应简单的 Newton 迭代法和梯度下降法。Liu 等人[7]提出可以将 $\boldsymbol{A}$ 设置成一个简单的常值矩阵，该矩阵只与布料结点之间的拓扑结构有关，因此在迭代过程中不会发生变化，可以进行预分解以加快仿真速度。后来 Bouaziz 等人[8] 归纳总结了这套方法为投影动力学。当然，也可以通过对矩阵 $\boldsymbol{A}$ 进行简化近似加速，如 Wang[9]使用对角矩阵近似 $\boldsymbol{A}$ 并使用 Chebyshev 加速算法；Fratarcangeli 等人[10]使用下三角矩阵近似 $\boldsymbol{A}$。除此之外，Peng 等人[11] 也提出了 Anderson 加速法进一步加速上述优化问题的求解。物理仿真算法虽然可以做到高真实感的布料仿真，但是由于涉及到大规模稀疏矩阵的求逆问题，在大规模的场景中很难做到实时，这也导致了现代工业引擎基本不采取纯物理仿真方法制作实时仿真项目。

二、非物理仿真方法

非物理仿真方法引入的动机正是解决物理仿真算法中实时性不足的问题。在这里我们不介绍数据驱动的算法，而是介绍基于位置的动态变化数值算法（PBD）。它首先由 Müller 等人[12]提出。与从 PDE 出发的物理仿真算法不同，PBD 的算法思想在于定义布料结点之间的多个约束，算法每次迭代只需要尽可能满足这些约束即可。这套方法有非常多的后续推广与改进，如 Macklin 等

人[13]提出的 XPBD，在保证了实时性的前提下可以对真实感进行进一步的提升。再比如说 Macklin 等人[14] 后续提出的小步长的进一步加速算法。

需要说明的是，PBD 算法并不是由物理算法推导而来，因此它有一些缺陷，如迭代次数对布料的物理性质具有很大的影响（软硬程度）。但是即使如此，Bouaziz 等人[8]指出其实 PBD 是前面物理仿真方法中介绍的投影算法（Projection Dynamics）方法的一种特例，因此可以预见的是 PBD 算法的收敛解，布料的物理性质应该像隐式仿真那样，并不取决于迭代次数。

三、图聚类与 GPU 加速方法

Fratarcangeli 等人[10]使用图聚类[15] 算法判定 XPBD 中哪些约束相互独立，从而使用 GPU 进行分块并行求解。染色的数量决定了所有约束被划分的类数，

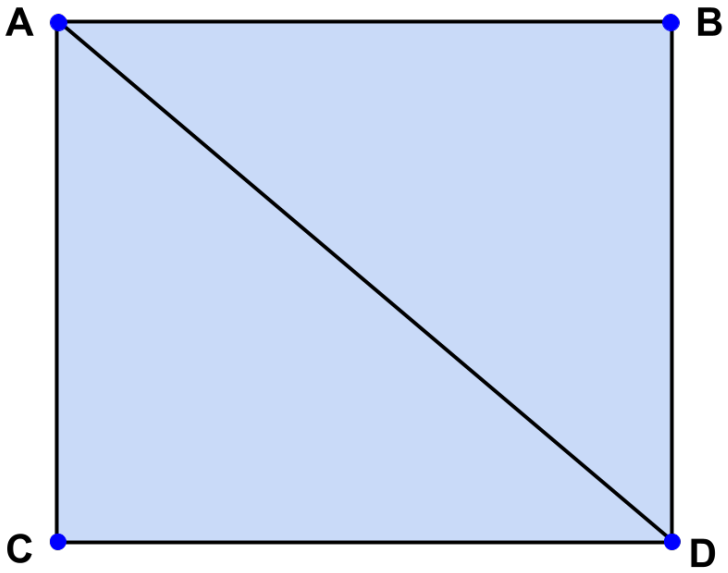


图 1.2　两个面片的约束示意图。AB,AC,AD 约束相互掣肘，如果直接并行这三个约束会导致对结点 A 位置的更新时出现读写冲突。

在完成划分后再依次求解。Fratarcangeli 等人[16] 发现可以引入虚拟的约束和虚拟的结点以减少染色数量。虽然这样进一步加速算法，但是布料仿真的物理性质也会随之变化。Macklin 等人[17]和 Ton-That 等人[18]提出可以将邻居结点聚成超节点以减少约束图的复杂性，从而进一步减少染色数量提高仿真速度。本文也探究了这种染色的 GPU 加速算法，并提出了进一步加速的展望。

第三节　章节安排

第一章主要介绍了实时布料仿真的必要性、一些前沿的布料仿真算法和本工作的研究动机与相关工作。

第二章主要介绍了针对实时布料的真实感与仿真速度两个尺度，设计的三种改进算法综述。

第三章主要介绍了实验引擎的各种实现细节与评价指标的构建。

第四章主要给出了五套算法改进效果定性与定量比较。

第五章主要总结了采用的改进算法的动机与效果并且对未来工作做出展望。

第二章　算法介绍

第一节　XPBD 算法介绍

首先先简要回顾一下 PBD 算法，具体推导可以参考 Bender 等人[19]的综述文章。正如我们在第一章所论述的，PBD 实际上是 Projection Dynamics 的特例。因此，PBD 算法的流程可以被简要概述为一个半隐式积分紧跟多个约束投影。投影步骤是使用每个约束函数的局部线性化与质量加权修正。PBD 约束求解器的主要步骤是计算每个约束带来的位置修正量：

$$\Delta \boldsymbol{x} = k_j s_j \boldsymbol{M}^{-1} \nabla C_j(x_i) \tag{2.1}$$

其中角标 i 注明迭代次数，j 是约束角标，$k \in [0,1]$ 是作用于每个约束修正的约束刚度。缩放系数 s 由每一个约束的 Newton 迭代迭代中给出：

$$s_j = \frac{-C_j(x_i)}{\nabla C_j \boldsymbol{M}^{-1} \nabla C_j^T} \tag{2.2}$$

可是简单的引入 k 来缩放位置变化带来了副作用——约束刚度（布料的软硬物理性质）现在依赖于时间步长和执行的约束投影的数量。Müller 等人[12] 尝试使用指数放缩刚度系数的方法解决这个问题。但是该解决方式仍然没有把时间步长纳入考量，而且在多约束问题中也没有收敛到良定解。

这些 PBD 的弊端启发了 XPBD 的提出，通过引入弹性势能重新定义 PBD 约束，来解决时间步长和迭代次数影响模拟结果刚性表现的问题。由 Newton 第二定律：

$$\boldsymbol{M}\boldsymbol{x} = \boldsymbol{f}_{int}(\boldsymbol{x}) + \boldsymbol{f}_{ext}(\boldsymbol{x}) \tag{2.3}$$

其中 $\boldsymbol{M} \in \mathbb{R}^{n \times n}$ 为对角质量矩阵，$\boldsymbol{f}_{int} \in \mathbb{R}^n$ 和 $\boldsymbol{f}_{ext}$ 分别为所受内力和外力，$\boldsymbol{x} \in \mathbb{R}^{\mathrm{n}}$ 为位置变量。其中 $n = 3|V|$。而内力又可以用定义的能量 $U(x)$ 的梯度来刻画：

$$U(\boldsymbol{x}) = \frac{1}{2} C(\boldsymbol{x})^T \alpha^{-1} C(\boldsymbol{x}) \tag{2.4}$$

$$\boldsymbol{f}_{int}(\boldsymbol{x}) = -\nabla U(\boldsymbol{x})^T = -\nabla C(\boldsymbol{x})^T \alpha^{-1} C(\boldsymbol{x}) \tag{2.5}$$

其中 $C(\boldsymbol{x}) = [C_1(\boldsymbol{x}), C_2(\boldsymbol{x}), \cdots C_m(\boldsymbol{x})]^T \in \mathbb{R}^m$ 是 m 维的约束向量，$\alpha \in \mathbb{R}^{m \times m}$ 是对角遵从矩阵。

通过一些对时间的有限元离散，可以通过引入 Lagrange 乘子将方程2.3重新改写。这里

$$\lambda = -\tilde{\alpha}^{-1}C(x) \tag{2.6}$$

其中 $\tilde{\alpha} = \frac{\alpha}{\Delta t^2}$，$\Delta t$ 为时间步长。在运动学方程中代入 λ，我们得到离散的运动方程：

$$\boldsymbol{M}(x^{n+1} - \tilde{x}) - \nabla C(x^{n+1})^T \lambda^{n+1} = 0 \tag{2.7}$$

$$C(x^{n+1}) + \tilde{\alpha}\lambda^{n+1} = 0 \tag{2.8}$$

其中 $\tilde{x^n} = 2x^n - x^{n-1} = x^n + \Delta t v^n$。最终再在每一帧同时更新 λ 和 x 的值即可：

$$\begin{aligned} \lambda^{n+1} &= \lambda^n + \Delta\lambda \\ x^{i+1} &= x^i + \Delta x \end{aligned} \tag{2.9}$$

XPBD 没有在每次迭代时使用全局线性求解来求解 $\Delta\lambda$ 和 Δx，而是使用高斯-塞达尔局部求解器，该求解器投影每个约束 $C_j(x)$，依次更新它们相关的拉格朗日乘子和位置变化量：

$$\Delta x = \boldsymbol{M}^{-1}\nabla C(x_i)^T \Delta\lambda \tag{2.10}$$

同时我们也可以计算出 Gauss-Seidel 求解器中 λ 的更新量

$$\Delta\lambda_j = \frac{-C_j(x_i) - \tilde{\alpha}_j\lambda_{ij}}{\nabla C_j \boldsymbol{M}^{-1}\nabla C_j^T + \tilde{\alpha}_j} \tag{2.11}$$

综上，可以写出 XPBD 算法的伪码：

算法 2.1 XPBD Pseudocode

input : x^t, v^t, mesh information
output: x^{t+1}, v^{t+1}

$h \leftarrow \frac{\Delta t}{numSubsteps}$
for $s = 1, \cdots numSubsteps$ **do**
  $\lambda \leftarrow 0$
  $\boldsymbol{x} \leftarrow \boldsymbol{x}^t + h\boldsymbol{v}^t + h^2\boldsymbol{M}^{-1}\boldsymbol{f}_{ext}(\boldsymbol{x})$
  $\tilde{\alpha} \leftarrow \frac{1}{h^2}\alpha$
  for $j = 1, \cdots m$ **do**
    $\boldsymbol{A} \leftarrow [\nabla C_j(x)\boldsymbol{M}^{-1}\nabla C_j(x)^T + \tilde{\alpha}_j]$
    $\Delta\lambda_j \leftarrow -\boldsymbol{A}^{-1}(C_j(\boldsymbol{x})) + \tilde{\alpha}_j\lambda_j)$
    $\lambda_j \leftarrow \lambda_j + \Delta\lambda_j$
    $\boldsymbol{x} \leftarrow \boldsymbol{x} + \Delta\boldsymbol{x}$
  end
  $\boldsymbol{v}^{t+1} \leftarrow \frac{\boldsymbol{x}-\boldsymbol{x}^t}{h}$
  $\boldsymbol{x}^{t+1} \leftarrow \boldsymbol{x}$
end

本文在第三章中展示了 XPBD 算法相较于 PBD 算法的优势，总之，XPBD 算法的引入克服了 PBD 算法固有的布料物理性质受到迭代次数等非物理量的控制的缺点，在保证实时性的同时增加了仿真的真实感。

第二节　Chebyshev 加速法介绍

Wang[9]提出了 Chebyshev 加速法在投影动力学中的加速应用。虽然射影动力学从本质上讲是非线性的，但它的收敛行为类似于求解线性系统的迭代方法。正因为如此，我们可以估计“谱半径”，并使用它在 Chebyshev 方法中加速收敛至少一个数量级，且该方法同时适用于雅可比求解器和高斯-塞达尔求解器。

实际上，虽然除了 Chebyshev 加速法外还有许多其他的加速方法，如 Liu 等人[20] 提出的 LBFGS 加速方法，但是在这些方法中只有 Chebyshev 方法适用于 GPU 加速，这是因为 GPU 不擅长处理点积，而 Chebyshev 方法中不存在点积，Wang[9] 甚至断言 Chebshev+Jacobi Iteration 可以得到最好的加速效果。

可以从第一章的论述看出，PBD 算法是投影动力学算法的一种特例，因此 PBD 算法也可以自然的使用 Chebyshev 加速。也正如我们之前对 Chebyshev 介绍的那样，Chebyshev 加速法其实就是在雅可比更新中，把本帧的更新量与上一帧（上两帧）的更新量做了一个加权平均：

$$q_i^{k+1} = q_i^k + \alpha \sum_c \Delta q_{i,c} \tag{2.12}$$

其中 i 是顶点编号，$\Delta q_{i,c}$ 是约束 c 投影到顶点 i 上的位移修正量。α 是松弛系数。值得注意的是，使用 Chebyshev 方法虽然可以加速算法的收敛，但是如果松弛系数选择不合适的话有可能会导致算法发散，这一点可以在第四章的算法结果中看出来。

算法2.2中我们给出 PBD+Chebyshev 的伪码。虽然 Chebyshev 方法可以在合适的松弛系数下加快算法收敛速度，但是其适用对象还是雅可比迭代，对于高斯-塞达尔迭代方法，Chebyshev 加速效果有限。

最后，为了更好的比较松弛系数对于 Chebyshev 加速法的影响，我们把算法2.2中的 ρ 和 γ 设置成相同的值。由于对于 Chebyshev 加速法，涉及到的迭代更新式形如：

$$y^{(k+1)} = w_{k+1}(\boldsymbol{B}^{-1}(\boldsymbol{C}y^{(k)+b}) - y^{(k-1)}) + y^{(k-1)} \tag{2.13}$$

由于 Chebyshev 加速法成立要求 $\boldsymbol{B}^{-1}\boldsymbol{C}$ 的特征值都是实数且 $\rho(\boldsymbol{B}^{-1}\boldsymbol{C}) < 1$，因此

算法 2.2 Cheby Pseudocode

input : q^t, v^t, λ^t denote the positions of mesh vertices and velocities in time t respectively. λ^t is the lagrange multiplier in time t. ρ denotes the spectral radius.
output: $q^{t+1}, v^{t+1}, \lambda^{t+1}$

$q^0 \leftarrow s_t \leftarrow q_t + hv_t + h^2 M^{-1} f_{ext}$
for $k = 1, \cdots K-1$ **do**
  for *each constraint c* **do**
    $p_c \leftarrow Local\_Solve(c, q^{(k)}, \lambda^{(k)})$
  end
  $\hat{q}^{(k+1)} \leftarrow Jacobi\_Solve(s_t, q^{(k),p_1,p_2,\cdots,\lambda_1,\lambda_2,\cdots})$
  if $k < S$ **then**
    $w_{k=1} \leftarrow 1$
  end
  if $k = S$ **then**
    $w_{k+1} \leftarrow 2/(2-\rho^2)$
  end
  if $k > S$ **then**
    $w_{k+1} \leftarrow 4/(4-\rho^2 w_k)$
  end
  $q^{(k+1)} = w_{k+1}(\gamma(\hat{q}^{(k+1)} - q^k) + q^k - q^{(k-1)}) + q^{(k-1)}$
  $\lambda^{(k+1)} = w_{k+1}(\gamma(\hat{\lambda}^{(k+1)} - \lambda^k) + \lambda^k - \lambda^{(k-1)}) + \lambda^{(k-1)}$
end
$q_{t+1} \leftarrow q^{(K)}$
$\lambda_{t+1} \leftarrow \lambda^{(K)}$
$v_{t+1} \leftarrow (q_{t+1} - q_t)/h$

如 Wang[9] 中所述，对于雅可比更新迭代，一般设置 $\rho \in [0.6, 0.9]$。

在第四章我们会对不同的松弛系数 (ρ) 的选值对加速的影响做出定性与定量的分析。

第三节　图聚类 GPU 加速法

下面我们介绍基于高斯-塞达尔迭代算法的图聚类加速算法。由于高斯-塞达尔求解器循环迭代中的约束投影直接根据 x 的当前值更新 x，由于每个约束之间涉及到的变量有重叠，因此我们不能简单地并行化算法。然而，每个约束梯度 $\nabla C_j(x)$ 都是关于结点稀疏的。两个非重叠的约束可以以任意顺序更新 x。

下面我们详细阐述一下如何利用约束梯度关于结点的稀疏性来设计图聚类的 GPU 并行算法。首先定义：

$$\mathscr{C}_j = \left\{ i | \frac{\partial C_j}{\partial x_i} \neq 0, \forall i \in [1, n] \right\} \tag{2.14}$$

换言之,$\mathscr{C}_j$ 定义了约束 C_j 的自由度。因此，为了并行化整个高斯-塞达尔迭代更新过程，需要构建约束的划分 $\mathscr{P}$，其中每个划分 $\mathscr{P}_c \subset [1, m], c \in [1, |\mathscr{P}|]$, 每一个约束的划分集中包含的约束彼此之间相互独立。即 $\mathscr{C}_j \cap \mathscr{C}_k = \varnothing$, 其中 $\mathscr{C}_j$ 和 $\mathscr{C}_k$

和 $\mathscr{P}$ 中的两个约束划分。接着只需要对约束划分中的每个约束并行，在对所有的划分串形即可，依照这个思想，可以把第一节中介绍的 XPBD 算法2.1改写为:

算法 2.3 GPU XPBD Pseudocode

```
for c ∈ 𝒫 do
    for j = in𝒫_c do
        A ← [∇C_j(x) M^{-1} ∇C_j(x)^T + α̃_j]
        Δλ_j ← −A^{-1}(C_j(x) + α̃_j λ_j)
        Δx ← M^{-1} ∇C_j(x)^T Δλ_j  λ_j ← λ_j + Δλ_j
        x ← x + Δx
    end
end
```

确定并行算法后需要处理的问题就是如何进行染色。首先，由于染色数量越少算法运行速度提高的越多，因此需要先行估计一下给定网格的染色下界。在这里我们不加证明的给出答案：染色数量的下界是约束图中最小生成环的维度。

对于任意给定的一个 3D 网格, 确定最小染色数的一种染色方案到目前位置仍然是 NP Hard 问题，Ton-That 等人[18]提出了一种基于贪婪算法的染色方案。为了方便起见，在我们的实验中，并没有涉及到不规则的布料模型，所有的不了模型均为被三角网格离散的正方形网格（在下一章中会进一步说明）。因此对于这种规整的网格结构，可以直接给出一个达到染色下界的染色方案。如图，一共使用八种颜色对约束进行染色，由于规整的网格约束图中最小生成图维度确实为8，因此该染色方案达到染色数量的理论下界。后续只需要对每一种颜色的约束串行即可。同时可以指出，这种看似简单的染色方案实际上符合并行计算中的合并存取范式：即相邻线程访问相邻位置的内存数据的时候存取效率最高。对于贪婪算法而言，虽然总是可以找到一种接近最佳的染色方案，但是并不能保证染色方案适配于合并存取范式，这也正是笔者在未来工作中想探究的一点——如何设计染色方案使得这种染色并行方案符合合并存取范式。

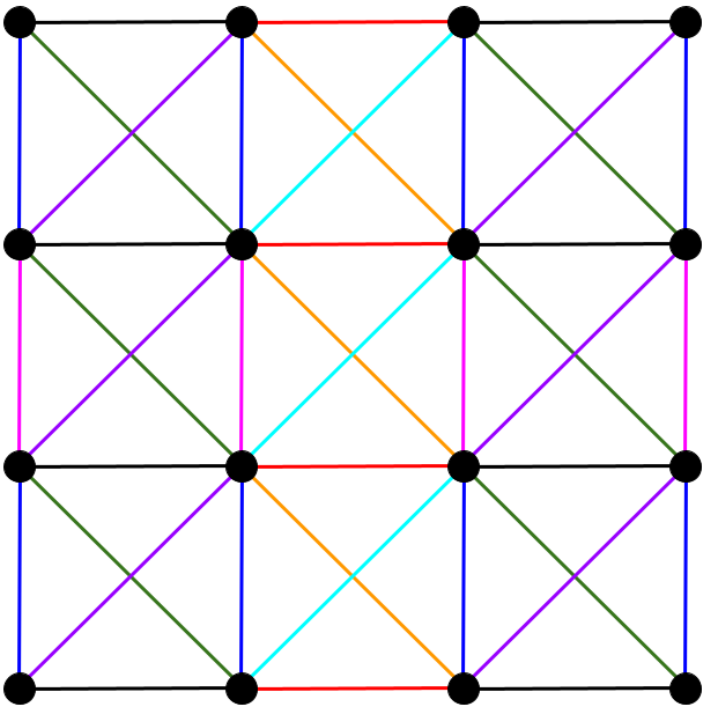


图 2.1　4×4 规整网格的染色示例。本示例种染色方法手工确定，约束一共被染成八类，此时的染色算法符合合并存取范式。

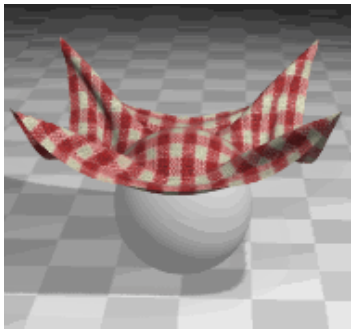
第三章　算法实现

第一节　仿真引擎

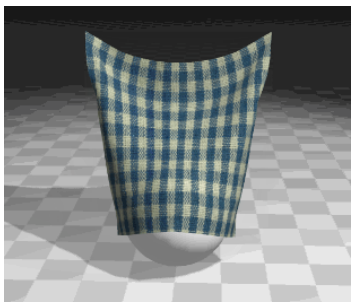
本文使用 Velvet 引擎[①]进行算法实验。这是因为：

- Velvet 是完全开源的引擎；
- Velvet 支持 GPU 加速且已经有现成的 GPU 加速代码供学习和参考；
- Velvet 引擎满足轻量化框架的需求，适合代码的修改。

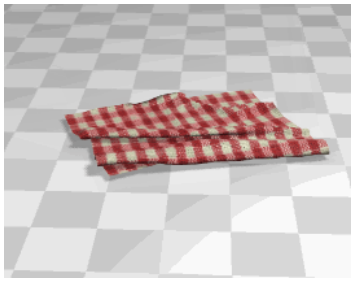
Velvet 引擎首先实现了 GPU 加速的雅可比迭代的 PBD 算法，除了布料解算以外，也解决了布料的碰撞和基础的渲染，具体的实现方法读者可以自行阅览。Velvet 引擎也构造了数个场景测试算法性能。Attach 场景为布料和球的交互，测试布料与刚体的碰撞处理，Self Collision 测试了布料的自交处理，Mutiple Object 测试了不同布料之间的碰撞处理。



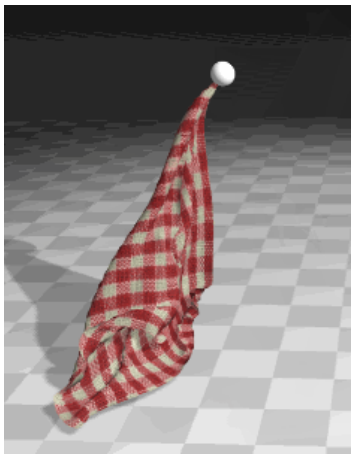
(a) 场景一, Attach



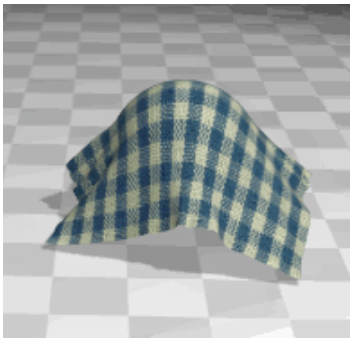
(b) 场景二, SDF Collision



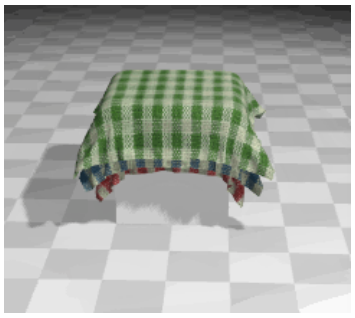
(c) 场景三, Self Collision



(d) 场景四, Swirl



(e) 场景五, Friction



(f) 场景六, Multiple Objects

图 3.1　Velvet 初始自带的六个场景，可以看到 Velvet 支持布料与刚体用户的交互，同时又已经实现的布料自交和布料相交处理。

[①]https://github.com/vitalight/Velvet

本次研究我们仿真的模型全部是规整的二维布料，可以指出的是我们的算法其实可以从二维布料仿真扩展到更加复杂的软体仿真，只需要把三角网格的有限元离散该成四面体的有限元离散，就可以仿真橡胶、金属、皮革等材质的复杂物体。

第二节　约束建立

由第一章中介绍，PBD 算法实质上是退化的投影学方法，其关约束的定义可以直接参考相关论文[8] [12]对于二维网格来说，即拉伸约束和折叠约束，下面我们详细定义一下这两种约束：

1. 拉伸约束

正如我们前文所述，二维网格中使用的是质点弹簧模型，在布料形变的过程中，顶点之间的弹簧发生伸缩，但是约束总是希望顶点之间的弹簧恢复原长。

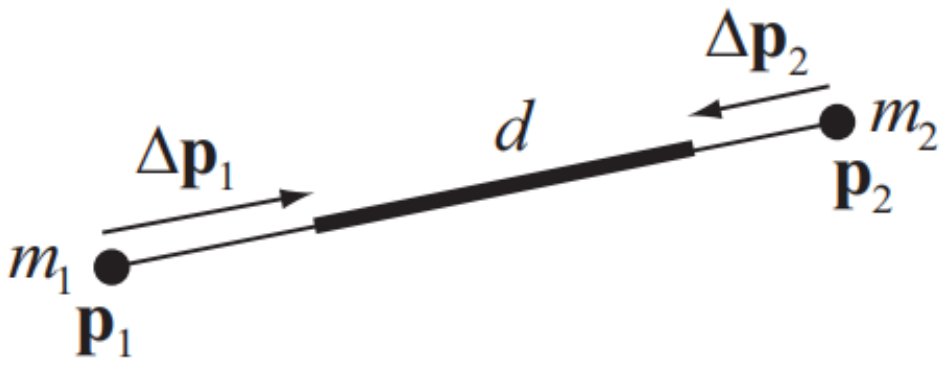


图 3.2　P_1P_2 拉伸约束示意图。（图片来源文献[12]）

这里以结点 P_1 和 P_2 举例，拉伸约束即为 $C(P_1,P_2)=|P_1-P_2|-d$，其中 d 为结点 P_1 和 P_2 之间的弹簧原长。直接计算该拉伸约束关于 P_1 和 P_2 的导数可以得到 $\nabla_{P_1}C(P_1,P_2)=\boldsymbol{n},\nabla_{P_2}C(P_1,P_2)=-\boldsymbol{n}$, 其中 $\boldsymbol{n}=\frac{P_1-P_2}{|P_1-P_2|}$。同时根据计算可知：

$$s=\frac{C(P_1,\cdots,P_n)}{\sum_j w_j|\nabla_{P_j}C(P_1,\cdots P_n)|^2} \tag{3.1}$$

其中 w_i 为相应的结点 i 的质量的倒数 $w_i=\frac{1}{m_i}$, 在本例中 $s=\frac{|P_1-P_2|-d}{w_1+w_2}$。据

此，可以计算得到顶点 P_1 和 P_2 的投影修正量分别为

$$\begin{aligned} \Delta P_1 &= -\frac{w_1}{w_1+w_2}(|P_1-P_2|-d)\frac{P_1-P_2}{|P_1-P_2|} \\ \Delta P_2 &= +\frac{w_2}{w_1+w_2}(|P_1-P_2|-d)\frac{P_1-P_2}{|P_1-P_2|} \end{aligned} \tag{3.2}$$

2. 折叠约束

与拉伸约束不同，折叠约束限制了公共边的两个三角形形成的二面角，使其尽可能在仿真过程中保持初始角度。

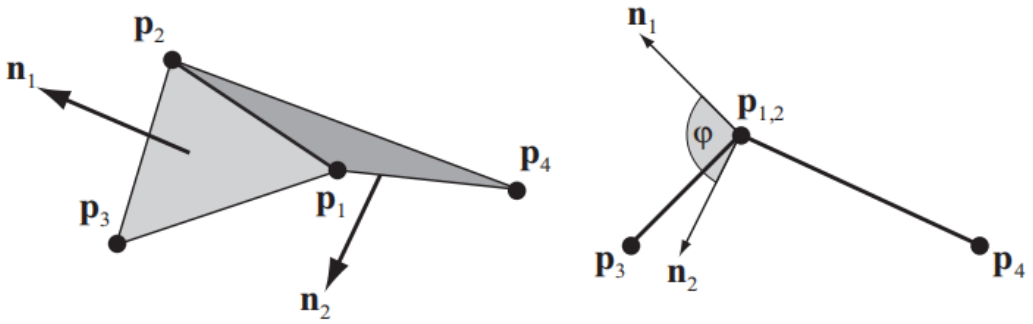


图 3.3　折叠约束示意图，三角面片 $P_1P_2P_3$ 和 $P_1P_2P_4$ 公用边 P_1P_2。（图片来源[12]）

如图3.3所示，三角面 $P_1P_2P_3$ 与三角面 $P_4P_5P_6$ 共享边 P_1P_2，假设对应面的单位法向量为 n_1 和 n_2，且在仿真开始，两个面片的夹角为 φ_0, 则对应的 z 折叠约束定义为：

$$C(P_1,P_2,P_3,P_4) = arccos(n_1 \cdot n_2) - \varphi_0 \tag{3.3}$$

折叠约束与直接对 P_3 与 P_4 之间的距离做约束的好处在于，由于约束表达式中完全不包含边的长度，因此定义的折叠与拉伸约束无关。这样，用户可以指定低拉伸刚度但抗弯曲性高的布料。

3. 连接约束

在 PBD 算法中，将顶点附加到静态或运动学对象是相当简单的。顶点的位置被简单地设置为静态目标的位置，或在每个时间步长中进行更新，以符合运动学物体的位置。为了确保包含此顶点的其他约束不移动它的位置，其逆质量 w_i 被设置为零。

第三节　评价指标构建

由前文论述可知，本文主要关注两个仿真指标，即在实时仿真的基础上，仿真是否更加具有真实感（对应的是 XPBD 算法的改进）；在 XPBD 算法的基础上，

能否进一步加快仿真速度（对应 Chebyshev 加速法和基于图染色的 GPU 加速算法）。

1. 仿真真实感

对于真实性仿真指标来说，如果需要定量的给出一个评价指标，需要先实现一套基于物理 PDE 的实现方法，例如之前提及的牛顿迭代法。在本文中，作者采取了一套更加简单的定量的衡量方法。

Macklin 等人[13]指出两个关键参数对模拟的效果会产生直接影响，一个是时间步长 Δt，另一个是解算迭代次数 iteration times。具体直观表现是 Δt 越小，迭代次数越大其模拟的直观表现为越硬，反之表现的越软绵绵的效果。基于这个观察，Macklin 等人[13]构建了一个悬挂布料的场景，该场景控制每一步时间步长不变，改变解算迭代次数，观察在不同迭代次数下 PBD 和 XPBD 的物理性质的变化。

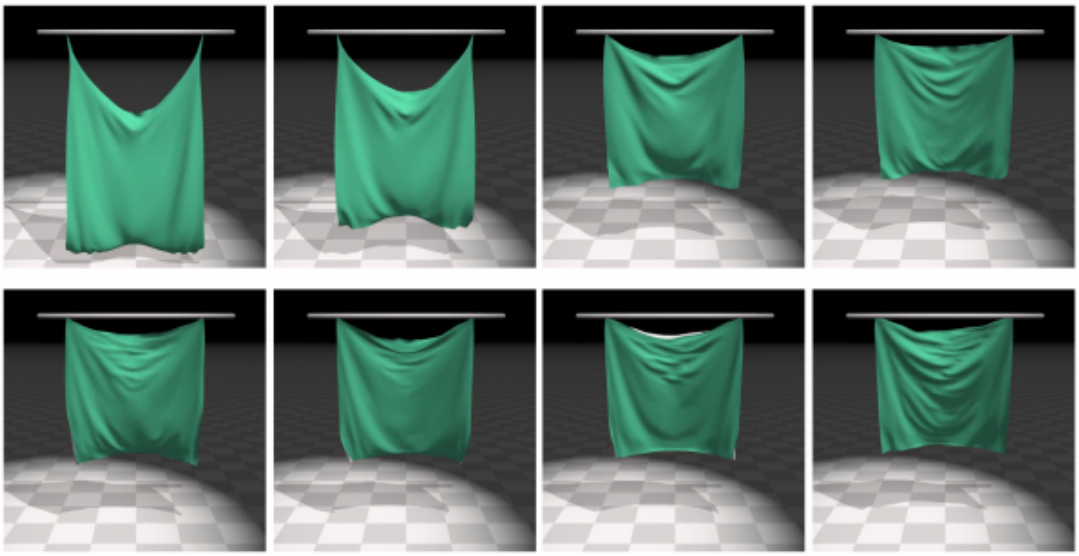


图 3.4　悬挂的布料, 分别迭代 20、40、80、160 次（从左到右）。上面一行是 PBD 的实现结果，下面一行是 XPBD 的实现结果。（图片来源文献[13]）

可以从图3.4中看出 PBD 的物理性质以非线性的方式依赖于迭代计数，而 XPBD 的物理性质定性的来看不随迭代次数发生变化。本文中也复刻了该实验并进行定性分析，具体的结果见第四章中的实验结果。

2. 仿真速度

对于仿真速度，最简单的评价指标就是仿真所用的时间。Velvet 引擎使用物理帧 (Physical Frame) 对此进行刻画，其中一帧的时间为 $\frac{1}{60}s$。对于仿真速度，我们分别采取了 Wang[9] 和 Ton-That 等人[18]构建的两套衡量方法。首先是基于 XPBD 方程推到的一套衡量指标，从方程2.8中可以看出布

料解算的每一次迭代都是尽可能使得 $h(x,\lambda)=C(x^{n+1})+\tilde{\alpha}\lambda^{n+1}$ 等于 0。因此可以通过观测 $h(x,\lambda)$ 的绝对值来衡量算法的收敛速度。

另一种更加简单的评价指标是通过观测仿真过程中的能量衰减速度。这是因为在没有其他外力做功的系统中，由于空气阻力的存在，最终系统一定会处于能量最低的状态。本文中构造的外力不做功的场景即以横挂的布料，在不加中立加速度的前提下，初始时刻给布料正中心的结点一个向下的初速度，由于 XPBD 算法中有衰减系数，即随着时间流逝，结点速度会自然衰减且没有重力做功。因此，在算法稳定的时候，可以遇见布料的弹性势能为 0，此时布料处于初始时刻水平悬挂的状态。在这个场景下，只需要观测 $E=\sum_{i,j}|\frac{k_{i,j}}{2}|P_i-P_j|-d_{i,j}|^2$ 随物理帧数的变化即可。

第四节　Velvet 引擎实现

Velvet 引擎并不是简单的实现了 GPU 加速的 PBD 算法，它采用了许多篇其他论文的改进思路，并且进行了一些值得学习的 CUDA 代码优化，笔者将简要介绍。算法3.1给出 Velvet 引擎实现的 PBD 算法的伪码

下面说明一下 Velvet 引擎中 PBD 代码相较于原始的 PBD 代码的改进之处：

1. 在主循环之前先调用 CollideSDF 函数

   运动物体（仿真或用户输入等控制的物体）可能移动得非常快（比如说用户剧烈的拉拽布料结点），如果结点在 PBD 循环中与它们碰撞，结点将以非常大的速度结束并产生不稳定的行为（大振动等）。因此，我们采用了碰撞预处理（即 CollideSDF）。这个预处理方式直接更新结点的位置 x_i 因此不会影响结点的速度。

2. 在实际 PBD 主循环中不调用 SolveBending 函数

   虽然在前面一节的分析中，我们指出约束中包括折叠约束这一项，但是根据 Velvet 开发者的经验，加入折叠约束并不能很好的适配与雅可比布料解算。使用小的折叠约束系数使得算法难以收敛，并且会导致结点的聚集或振动。使用大的折叠约束系数对整体视觉结果影响不大。此外，弯曲约束比其他约束计算更昂贵。

3. 稳定的结点碰撞滤波

   在基于位置的动力学中，结点碰撞是实现布自碰撞的一个简单方法。然而，结点的大小可能很难选择。小的结点尺寸会导致布的几何形状中出

算法 3.1 Velvet PBD Pseudocode

input : x^t, v^t,other hyperparameters
output: x^{t+1}, v^{t+1}

```
CollideSDF()
for substep = 0, ··· numSubsteps do
    PredictPositions()                  // Predict particle positions
    FindNeighborsBySpatialHash()     // Use spatial hashing to find
     particle neighbors
                                                // Handle Collisions
    CollideParticles()       // Collide particles with neighboring
     particles
    CollideSDF()     // Collide particles with Signed Distance
     Fields
                                                // Solve constraints
    for iteration = 0, ··· numiterations do
        SolveStretch()  // Stretch constraint keeps the distance
         between two particles
        SolveAttachment()      // Attachment constraint keeps the
         distance between a particle and a 3D position
        // SolveBending()      // Bending constraints doesn't work
         well on GPU
        ApplyDeltas()   // Apply position deltas accumulated in
         previous constraint solving
    end
    Finalize()                  // Update velocities and positions
end
ComputeNormals()       // Compute vertex normals for rendering
```

现空隙，从而导致结点隧道和凹凸不平的表面。大的结点尺寸使布料视觉效果过厚，相邻的结点可能不断碰撞，这可能会导致错误的弯曲或折叠。

在本项目中，我们使用一个相对较大的结点尺寸（1.5 倍弹簧静止长度），并采用碰撞过滤技术来忽略处理最初接近的结点之间的碰撞。

实现碰撞过滤的一个初始的方法是存储所有由弹簧连接的一对结点（拉伸约束）。但这种数据结构在 GPU 上的查询效率很低。

我们的方法是将初始结点的位置存储在一个内存缓冲区 initialPositions 中。在寻找结点邻居的阶段，我们可以通过使用 initialPositions 检查结点的初始距离，轻松地确定结点初始位置是否接近。

4. 使用 Long Range Attachment 技术解决布料过弹性问题

在将 CPU 算法升级为使用 GPU 加速的雅可比布料解算算法过程中，会遇到布料视觉上变得过于具有弹性的问题实际上这是因为使用雅可比算法的收敛速度更慢。虽然可以通过增加 substeps 数量和每次迭代的数量解决这个问题，正如前文所论述的，随着 substeps 和每次迭代的数量增加，布料的物理性质会变得弹性更弱，但是如此一来，每一帧的布料解算的计算

开销增大，算法表现就会下降。

Kim 等人[21]提出了一种更加简单且低计算成本的解决方案。在原本的 Solve Attachment 步骤中，只需要单独修改连接点的位置、即修改算法使得所有布料的顶点都与连接点保持一定的距离。在 Velvet 引擎的实现中，只需要简单的将 attachDistances 引入到连接约束即可。

5. 原子操作更新

在许多 GPU 程序中会调用原子操作用于避免线程冲突，开发者使用了一个小技巧来提升原子操作的表现。矢量的 x,y,z 分量的原子更新是根据重排值来重排顺序的。这避免了 $\frac{2}{3}$ 的可能性让多个线程在同一时间更新同一内存位置。重新排序的顺序可以任意选择，这里开发者利用了被操作的两个结点编号：

算法 3.2 Atomic Reorder

input : Address of the particles, index of the particle to be changed, value of the particle added on the position, reorder sequence
output: New address

```
1 r1 ← reorder%3
2 r2 ← (reorder + 1)%3
3 r3 ← (reorder + 2)%3
4 atomicAdd(&(address[index].x) + r1, val[r1])
5 atomicAdd(&(address[index].x) + r2, val[r2])
6 atomicAdd(&(address[index].x) + r3, val[r3])
```

6. 空间哈希中的邻居哈希技术

空间哈希的计算成本很高，但我们不需要每次做结点碰撞时都更新空间哈希。邻居信息可以存储在一个邻接数组中并多次使用。在这个项目中，我们使用一个整数参数 interleavedHash 来控制在重新计算之前重复使用多少次邻居。邻居数组的结构是，neighbors[i + numMaxNeighborsPerParticle * j] 存储结点 i 的第 j 个邻居。这里没有把邻居结构设置成 neighbors[i * numMaxNeighborsPerParticle + j]，因为这不符合合并存取范式。

7. 结点渲染

虽然结点渲染容易调试，但是即使使用实时渲染方法，渲染超过 10 万个结点也会变得相当慢（在开发者的笔记本电脑上使用 RTX2060，5 万个结点也无法达到 60FPS）。一个更有效的方法是 Green 等人[22]介绍的屏幕渲染。虽然渲染不具有物理正确性，但是更加高效。

第四章　算法结果

在第三节中我们介绍了理想的 XPBD 算法相较于 PBD 算法的改进，本章将从定性和定量的角度分别对 XPBD，Chebyshev 加算法和基于图聚类的 GPU 加速方法对经典 PBD 算法的改进做出分析。测试电脑使用的配置：

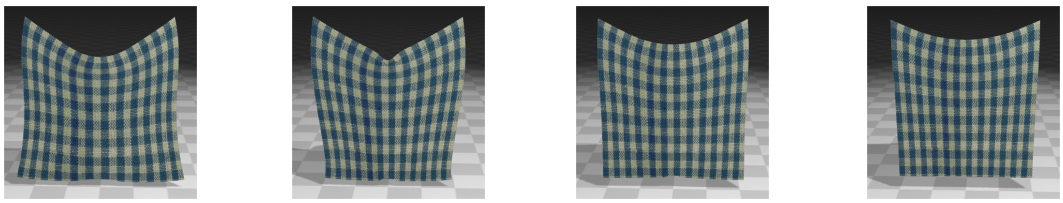
1. 硬件
    - CPU: Intel Core i7 8^{th} Gen
    - GPU: Nvidia GeForce MX250
2. 环境
    - 编译环境：Visual Studio 2019，C++ 17，C++ 17
    - 窗口显示与交互：imgui[core, opengl3-binding, glfw-binding], glfw3, glad
    - 输出：fmt
    - 3D 数学库：glm
    - 3D 模型加载：assimp

后文中，我们一共比对了五套算法的各项表现，这五套算法包括：

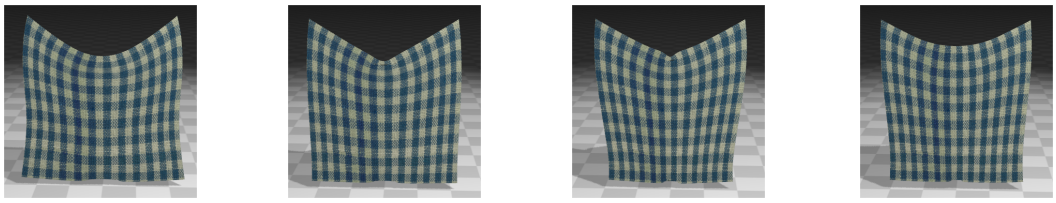
1. Velvet 引擎实现的改进版本的 GPU 加速 PBD 算法
2. 在 Velvet 实现基础上增加拉格朗日乘子的 GPU 加速的 XPBD 算法
3. 使用 Chebyshev 加速布料结算的 XPBD 算法（雅可比迭代，此时 Chebyshev 松弛系数设置为 0.6）
4. 使用 Chebyshev 加速布料结算的 XPBD 算法（雅可比迭代，此时 Chebyshev 松弛系数设置为 0.9）
5. 使用图染色算法加速的布料解算的 XPBD 算法（高斯-塞达尔迭代，使用图2.1的染色方法）

第一节　XPBD VS PBD

图3.4中我们展示了 XPBD 原论文中比较的悬挂的布料在不同迭代次数的物理性质变化。本文作者也效仿原文作者构造了一个类似的场景用于测试 XPBD 相较于 PBD 的改进。可以从图4.1(a)到图4.1(h)中看出，虽然笔者实现的结果中，PBD（第一行）与 XPBD（第二行）相比，布料的物理性质（弹性）确实随着布料迭代次数变化更为剧烈，但是并没像图3.4 中那样形成如此鲜明的对比，笔者认为约束刚度的设置差异：在 XPBD[13]一文中，作者故意使用了最低的约束刚



(a) PBD, 20 次迭代 (b) PBD, 40 次迭代 (c) PBD, 80 次迭代 (d) PBD, 160 次迭代



(e) XPBD, 20 次迭代 (f) XPBD, 40 次迭代 (g) XPBD, 80 次迭代 (h) XPBD, 160 次迭代

图 4.1 本文复现的 XPBD 的布料悬挂场景，PBD 与 XPBD 的结果对比，第一行为 PBD 在 20,40,80,160 次解算迭代的结果；第二行为 XPBD 在对应次数的解算迭代的结果，布料为 64×64，一共有 16512 个约束。

度，即 $k = 0.01$。

最后我们再比较一下 PBD 和 XPBD 的运行速度：

迭代次数	20	40	80	160
PBD	166.62FPS	55.52FPS	22.22FPS	5.55FPS
XPBD	157.32FPS	55.85FPS	20.80FPS	5.55FPS
相差	5.58%	0.59%	6.39 %	0.00%

表 4.1 在 16512 个约束下，PBD 与 XPBD 运行速度比较。

从表格5.1中可以看出，虽然 XPBD 相较于 PBD 的运行速度变慢了，但是并没有超过 6%，由此可以得出结论，XPBD 并不会影响到 PBD 的实时性。因此我么可以得出结论：XPBD 在合适的约束刚度下确实可以在不影响实时性的前提下提高算法的真实感。

第二节 Chebyshev 加速法

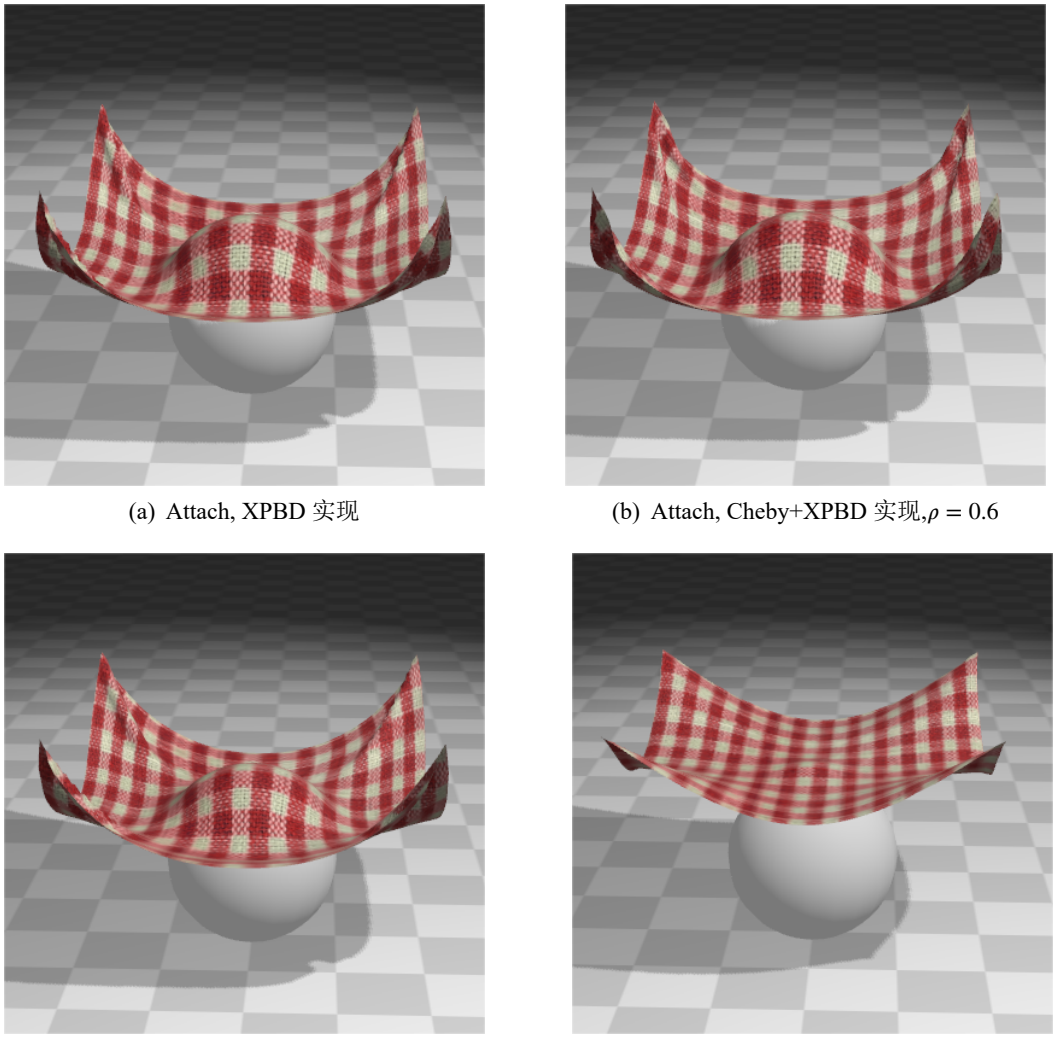
在本节中我们会定性的给出 Chebyshev 加速法的改进结果，定量的结果可以参考下一节中利用第三章第三节中介绍的 $h(x, \lambda)$ 和能量衰减速度做出的具体分析。据我们在公式2.13的论述可知，松弛系数会影响到 Chebyshev 加速法的加速表现。图4.2(a)到图4.2(d)展示了 XPBD, Chebyshev 加速法（$\rho = 0.6$）, Chebyshev 加速法（$\rho = 0.9$），图染色 GPU 加速法在 Attach 场景中的表现。

由之前对 PBD(XPBD) 算法的分析，迭代次数越大，算法越收敛，布料表现出的物理性质越“硬”。从图4.2(a)到图4.2(d)定性分析可以得出结论：

- Chebyshev 加速法（雅可比迭代）在合理的松弛系数下与图染色的 GPU 加

速方法（高斯-塞德尔迭代）都可以加快 XPBD 算法的收敛速度。

- 松弛系数和 ρ 对 Chebyshev 加速法的表现有直接影响，在本例测试中，Chebyshev 加速法（$\rho = 0.6$）相较于 XPBD 并无加速；但是，Chebyshev 加速法（$\rho = 0.9$）相较于 XPBD 有明显加速。
- 图染色的 GPU 加速方法（高斯-塞德尔迭代）加速效果比 Chebyshev 加速法（雅可比迭代）要强。



(a) Attach, XPBD 实现　(b) Attach, Cheby+XPBD 实现,$\rho = 0.6$

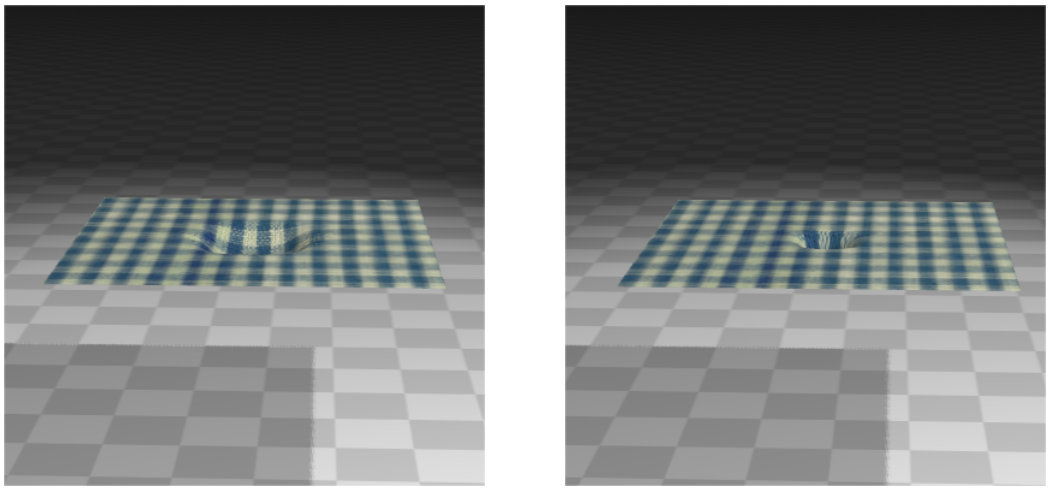
(c) Attach, Cheby+XPBD 实现,$\rho = 0.9$　(d) Attach, 图染色 GPU 实现

图 4.2　四种算法在 Attach 场景3.1(a) 的定性表现。第一行分别为 XPBD, Chebyshev 加速法（$\rho = 0.6$）的表现；第二行则分别为 Chebyshev 加速法（$\rho = 0.9$）和图染色 GPU 加速法的表现。四个算法 Numsubsteps 均设置成 2，Numiterations 均设置成 4，布料分辨率为 40×40，一共有 6480 个约束

第三节　图染色的 GPU 加速

本节笔者从定性和定量的两个角度对图染色的 GPU 加速法（高斯-塞德尔）迭代的加速效果做出了分析。首先，可以从图4.2(a)到图4.2(d)的对比中看出，基

于高斯-塞德尔迭代图染色的 GPU 加速方法比 Chebyshev 加速的基于雅可比迭代的算法收敛速度更快。除此场景外，笔者还构造了另一个简单的水平悬挂的布料场景用于说明问题。见图4.3(a)和图4.3(b). 可以从图4.3(a)与图4.3(b)看出，图染

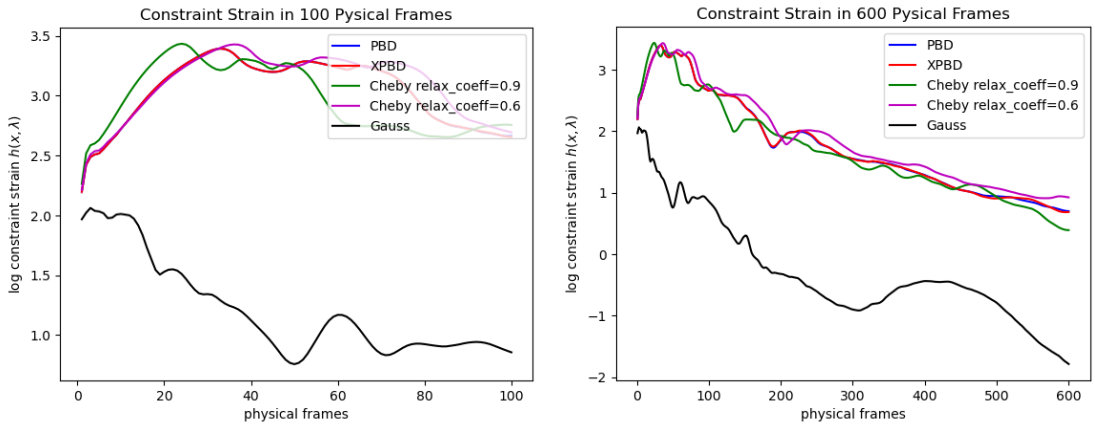


(a) Attach, 图染色 GPU 加速的 XPBD 实现　　(b) Attach, Cheby+XPBD 实现,$\rho = 0.9$

图 4.3　两种算法在水平悬挂场景的定性表现。左图和右图分别为图染色 GPU 加速法和 Chebyshev 加速法的表现。布料分辨率为 64×64，一共有 16512 个约束。

色 GPU 加速的 XPBD 算法中布料向下运动趋势的传播更加快，定性说明了图染色的 GPU 加速方法相较于 Chebyshev 加速法收敛速度更快。

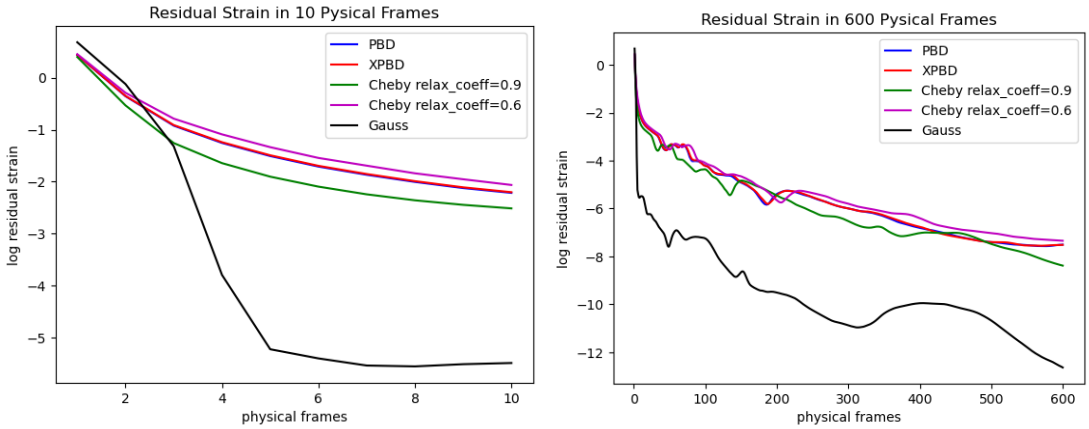
除了定性分析这几种加速算法的加速效果以外，我们将利用第三章第三节中设计的评价指标对本章引言中五种算法进行定量的进行对比，这里我们还是采用的定性分析中的水平悬挂布料的场景。（见图4.4(a)和图4.4(b)）首先是 $log(h(x,\lambda)) = log(C(x^{n+1}) + \tilde{\alpha}\lambda^{n+1})$ 的值随着 physical frame 的变化。



(a) 前 100 帧定量分析　　(b) 前 600 帧定量分析

图 4.4　PBD,XPBD,Chebyshev 加速法（$\rho = 0.6$）, Chebyshev 加速法（$\rho = 0.9$），图染色 GPU 加速法前 100 帧和前 600 帧加速定量分析，其中评价指标为 $log(h(x,\lambda))$。

同理我们可以使用 residual strain 作为衡量指标对五种算法进行定量的对比。（见图4.5(a)和图4.5(b)）从上述两种评价指标的定量分析，可以在第二节定



(a) 前 10 帧定量分析　　(b) 前 600 帧定量分析

图 4.5　PBD,XPBD,Chebyshev 加速法（$\rho = 0.6$）, Chebyshev 加速法（$\rho = 0.9$），图染色 GPU 加速法前 10 帧和前 600 帧加速定量分析。其中评价指标为 $log(\sum_{i,j} |(p_i - p_j) - d_{i,j}|^2)$

性结论上进一步确定：

- PBD 与 XPBD 相比，并无明显的运行速度差异。
- Chebyshev 加速法（雅可比迭代）是否可以加快算法收敛速度取决于松弛系数的设置是否合理，在合理的松弛系数下 Chebyshev 加速法与图染色的 GPU 加速方法（高斯-塞德尔迭代）都可以加快 XPBD 算法的收敛速度。
- 图染色的 GPU 加速方法（高斯-塞德尔迭代）相较于 Chebyshev 加速法（雅可比迭代）可以更高效的加速算法收敛。

第五章　总结与展望

第一节　总结

PBD 作为现在主流游戏中集成的布料更新算法，有实时性的优点，同时也有缺乏真实性的缺点。因此作者在 PBD 的基础上，从真实感和运行速度两个方面探究了前沿的改进方法。在这里我们再次总结一下前沿改进算法的改进动机与改进效果：

- 为了提高 PBD 算法的真实感，减少 PBD 算法中布料物理性质迭代次数和迭代时间步长的敏感程度，我们采用了 XPBD 算法。从实验结果来看，XPBD 确实相较 PBD 提升了算法的真实感与稳定性，同时并没有影响布料解算的实时性。
- 为了进一步提高 XPBD 算法的收敛速度，我们采用了切比雪夫加速法，该算法作用于雅可比迭代。从实验结果来看，在算法的松弛系数选择合适的时候，确实可以提高算法的收敛速度，但是松弛系数的选择没有自动确定的方法，需要用户手动确定。
- 同样为了提高 XPBD 算法的收敛速度，而且为了克服雅可比算法不稳定和使用雅可比算法带来的过弹性问题，我们采用了基于图染色的 GPU 加速法（高斯-塞德尔解算器），该算法相较于实验中的算法具有最好的加速收敛效果。

综上，基于作者对布料的实时仿真技术探究，我们基于算法实现难度、鲁棒性、真实感和运行速度得出了如下五种算法的性能优劣比较：

算法	算法实现难度	算法鲁棒性	算法真实感	算法速度
物理仿真	★★★	★★★	★★★	★
PBD	★	★	★	★★
XPBD	★★	★★	★★	★★
XPBD+Chebyshev	★★★	★	★★	★★★
XPBD+ 图聚类	★★★★	★★	★★	★★★★★

表 5.1　各种算法在不同维度上的优劣比较。

第二节　展望

本文探究了前沿的布料实时仿真技术，在笔者实现这些技术的过程中发现了一些有趣的未来工作：

- 最优图染色问题。对于图染色的加速算法来说，一个符合合并存取范式同

时染色数量最少的染色方案能够最大性能挖掘出图染色加速方法对高斯-塞德尔迭代的加速优势。目前笔者只处理了规整的正方形布料，可以提前给出最佳的染色方案，但是对于不规整的布料模型，需要给出一套自动化算法确定最佳或者逼近最佳的染色方案。

- 多布料的碰撞问题。在 Velvet 引擎原本的封装中，多布料碰撞场景3.1(f)中三块布料重叠时并没有发现穿模现象，然而在笔者实现的图染色加速算法中出现了穿模现象，初步推测时由于布料解算收敛速度大幅提高，使得碰撞检测无法实时的处理场景中的碰撞所致。
- 布料与不规则物体的碰撞问题。Velvet 引擎中只提供了布料与一些基本 3D 物体的碰撞接口，如球体的碰撞接口。但在显示的布料仿真运算中，往往布料需要于不规则的几何体发生碰撞，如穿着裙子跳舞的女人，裙子就与人这个不规则几何体频繁发生碰撞。因此需要额外修改碰撞算法，如使用包围盒算法来解决这一问题。
- 布料的其他交互。目前 Velvet 只支持用户拉拽布料，但是布料还有诸如碎裂的复杂物理现象，这些需要未来另外调研相关算法并集成到 Velvet 处理。

参 考 文 献

[1] POWER P. Animated expressions: Expressive style in 3d computer graphic narrative animation[J]. Animation, 2009, 4(2): 107-129.

[2] MAUYAKUFA F T, PRADHAN A. An analysis on the role of computer graphics and animation in zimbabwean film industry[C]//Proceedings of the International Conference on Industrial Engineering and Operations Management. 2018: 686-693.

[3] WANG H. Gpu-based simulation of cloth wrinkles at submillimeter levels[J]. ACM Transactions on Graphics (TOG), 2021, 40(4): 1-14.

[4] BARAFF D, WITKIN A. Large steps in cloth simulation[C]//Proceedings of the 25th annual conference on Computer graphics and interactive techniques. 1998: 43-54.

[5] KHAREVYCH L, WEI W, TONG Y, et al. Geometric, variational integrators for computer animation[M]. Eurographics Association, 2006.

[6] STERN A, GRINSPUN E. Implicit-explicit variational integration of highly oscillatory problems[J]. Multiscale Modeling & Simulation, 2009, 7(4): 1779-1794.

[7] LIU T, BARGTEIL A W, O'BRIEN J F, et al. Fast simulation of mass-spring systems[J]. ACM Transactions on Graphics (TOG), 2013, 32(6): 1-7.

[8] BOUAZIZ S, MARTIN S, LIU T, et al. Projective dynamics: Fusing constraint projections for fast simulation[J]. ACM transactions on graphics (TOG), 2014, 33 (4): 1-11.

[9] WANG H. A chebyshev semi-iterative approach for accelerating projective and position-based dynamics[J]. ACM Transactions on Graphics (TOG), 2015, 34(6): 1-9.

[10] FRATARCANGELI M, TIBALDO V, PELLACINI F. Vivace: A practical gauss-seidel method for stable soft body dynamics[J]. ACM Transactions on Graphics (TOG), 2016, 35(6): 1-9.

[11] PENG Y, DENG B, ZHANG J, et al. Anderson acceleration for geometry optimization and physics simulation[J]. ACM Transactions on Graphics (TOG), 2018, 37(4): 1-14.

[12] MÜLLER M, HEIDELBERGER B, HENNIX M, et al. Position based dynamics [J]. Journal of Visual Communication and Image Representation, 2007, 18(2): 109-118.

[13] MACKLIN M, MÜLLER M, CHENTANEZ N. Xpbd: position-based simulation of compliant constrained dynamics[C]//Proceedings of the 9th International Conference on Motion in Games. 2016: 49-54.

[14] MACKLIN M, STOREY K, LU M, et al. Small steps in physics simulation[C]// Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation. 2019: 1-7.

[15] MATULA D W, MARBLE G, ISAACSON J D. Graph coloring algorithms[M]// Graph theory and computing. Elsevier, 1972: 109-122.

[16] FRATARCANGELI M, PELLACINI F. Scalable partitioning for parallel position based dynamics[C]//Computer Graphics Forum: Vol. 34. Wiley Online Library, 2015: 405-413.

[17] MACKLIN M, MULLER M. A constraint-based formulation of stable neo-hookean materials[C]//Proceedings of the 14th ACM SIGGRAPH Conference on Motion, Interaction and Games. 2021: 1-7.

[18] TON-THAT Q M, KRY P G, ANDREWS S. Parallel block neo-hookean xpbd using graph clustering[J]. Computers & Graphics, 2023, 110: 1-10.

[19] BENDER J, MÜLLER M, OTADUY M A, et al. A survey on position-based simulation methods in computer graphics[C]//Computer graphics forum: Vol. 33. Wiley Online Library, 2014: 228-251.

[20] LIU T, BOUAZIZ S, KAVAN L. Quasi-newton methods for real-time simulation of hyperelastic materials[J]. Acm Transactions on Graphics (TOG), 2017, 36(3): 1-16.

[21] KIM T Y, CHENTANEZ N, MÜLLER-FISCHER M. Long range attachments-a method to simulate inextensible clothing in computer games[C]//Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. 2012: 305-310.

[22] GREEN S. Screen space fluid rendering for games[C]//Proceedings for the Game Developers Conference. Moscone Center San Francisco, CA, 2010.

致　谢

感谢刘利刚教授对我学业和科研的悉心指导。师者，传道，授业，解惑。正是因为刘老师的计算机图形学的课，让我从学习数学而不知其用的迷茫中解脱出来，激发了我对计算机图形的兴趣。另一方面，刘老师也教会了我做事、科研的方法论，他是我科研的指路人，我受益良多。

感谢本科阶段其他教授的授课，没有他们的高质量课程，就没有我扎实的数理功底与对计算机编程的理解，无法攻克科研的难关。

感谢实验室的师兄们的帮助，他们作为科研的先行者，让我感受到了严谨的学术态度和火热的科研热情。也因为他们的不吝帮助，我少走了很多弯路。

感谢开源社区的贡献者，他们的开源代码让我受益匪浅，他山之石可以攻玉。未来我也会更多的在开源社区分享，尽绵薄之力回报社区。

感谢我的同学、父母、爱人，他们在我科研的低谷时期给我坚持下去的勇气，是我坚定的心灵靠山。

最后感谢本文的读者，希望读者能有所收获，祝顺安！

2023 年 6 月