# Detailed Rigid Body Simulation with Extended Position Based Dynamics

Matthias Müller，Miles Macklin，Nuttapong Chentanez，Stefan Jeschke1，Tae-Yong Kim

**摘要**

我们提出了一种刚体模拟方法，通过使用准显式积分方案，可以解决小的时间和空间细节，是无条件稳定的。传统刚体模拟器线性化约束，因为它们在速度水平上操作或隐式求解运动方程，从而冻结了多次迭代的约束方向。我们的方法总是适用于最近的约束方向。这使我们能够跟踪物体在曲面几何上碰撞的高速运动，减少约束的数量，增加模拟的鲁棒性，并简化求解器的公式。在本文中，我们提供了实现一个完整的刚体求解器的所有细节，它能够处理接触，各种关节类型和与软物体的相互作用。

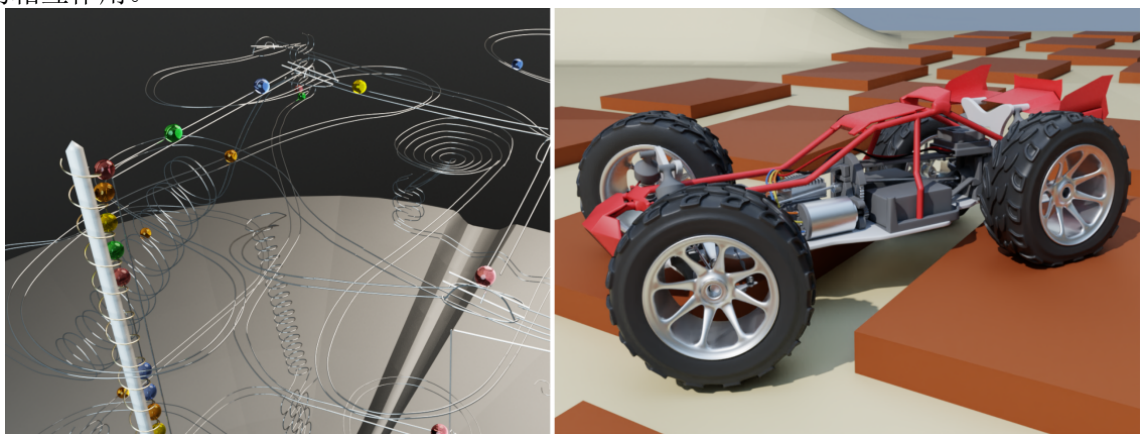

图 1: 效果图

**关键词：**刚体模拟；软体模拟；基于位置动力学

## 1 引言

刚体模拟是每个游戏引擎的核心，在计算机生成的电影特效中起着重要作用。刚体模拟的核心是接触和关节处理。目前有两种常用的碰撞处理方法，即 Penalty 方法和基于脉冲的方法。

Penalty 方法利用穿透物体产生的力进行分离。这些方法很少在游戏和电影中使用，因为需要大的力和小的时间步长来使碰撞的物体看起来刚性。

然而最流行的方法是使用脉冲，Mirtich 和 Canny 等[1]在 90 年代中期为图形和游戏中基于脉冲的刚体模拟奠定了基础。Hecker[2]将这些概念介绍给了游戏开发者 Baraff[3]到计算机图形社区。在这里，通过施加脉冲，速度在撞击时立即改变而不是施加由力引起的加速度。从概念上讲, 这些方法适用于直接忽略加速度层的速度。

模拟具有动态变化的位置和方向的物体是一个非线性问题。然而，冻结一个空间构型并求解速度会导致一个线性方程组。接触产生不等式约束，所以一般来说，必须解决线性互补问题 (LCP)，顾名思义，它仍然是线性的。速度空间可以看作是当前构型下空间状态非线性空间的切线空间。因此，在速度的线性空间内工作比直接处理位置和方向更方便。然而，这种方法的主要缺点之一是漂移问题，因为速度求解器看不到位置误差。现有的引擎可以通过各种方法解决这个问题，比如引入额外的力或约束。

基于位置的动力学 (PBD)[4-5]通过直接处理位置来解决这个问题。速度是求解后得到的，即时间步长结束时的构型和开始时的构型之间的差值。

PBD 主要用于模拟约束粒子系统来模拟布料和柔软的物体，直到 Macklin 等人[6]设计了一种处理流体的方法。这使得同一小组可以在基于位置的框架[7]中开发基于粒子的统一求解器。他们使用形状匹配[8]的思想将刚体模拟为刚性连接粒子的集合。然而，形状匹配的成本随着粒子数量的增加而增加，脉冲传播速度慢，处理关节困难。一种更有效的方法是将 PBD 扩展到粒子之外，通过引入旋转状态将刚体模拟为单个实体。Deul 等人[9]在 PBD 的位置框架中阐述了这类刚体动力学。

对于全局求解器的迭代，处理速度问题或线性化位置问题都有一个共同的方面: 它们冻结线性求解时间的约束方向，即在几个迭代中。在这种情况下，接触点必须作为局部平面，库仑摩擦锥作为多面体。此外，三维约束 (如附着) 会产生三个约束方程。相比之下，基于高斯-赛德尔等方法求解局部接触问题的算法允许接触几何形状在每次迭代中发生变化。该方法已用于平滑各向同性摩擦模型[10-11]。我们扩展了这种方法，也允许接触法向几何形状改变每次迭代

原 PBD 方法采用非线性投影高斯塞德尔 (NPGS) 方法求解非线性位置方程。NPGS 方法与线性化方程的常规或投影高斯-塞德尔 (PGS) 方法有本质区别。图 2显示了这种差异。关键是在每个单独的约束解决后，位置立即更新。这样，PBD 直接处理非线性问题，提高了鲁棒性和精度。圆形摩擦锥或与弯曲物体的碰撞很容易处理。我们不再将联系人存储为对一对对象的引用，以及产生接触平面的静态法线，而是仅存储引用并在每个特定联系人的单独求解之前重新计算法线。一种成本较低的方法是像传统方法一样存储局部接触几何，但使用高阶近似。

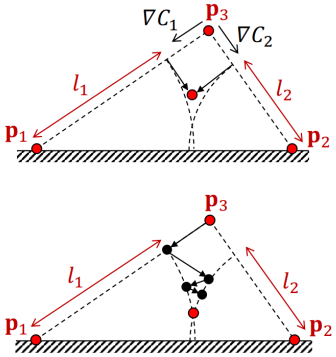


图 2: NPGS 与 PGS

考虑到这些优势，我们的目标是扩展 PBD，使其能够实现完全成熟的刚体引擎，但又不牺牲其简单性。我们将在允许在实时效果的级别上上提供所有算法。

然而，PBD 有非物理的名声，并且基于以下考虑过于简化:

1. 它不使用物理量和单位
2. 刚度具有迭代性和时间步长依赖性
3. 这个积分在物理上并不准确
4. 它取决于约束处理的顺序
5. 它依赖于网格镶嵌
6. 它收敛得很慢

幸运的是，所有这些问题最近都得到了解决，结果是我们提出的求解器是其他方法的有力竞争者，我们将在结果部分中展示

前三个问题已在 XPBD[12]中得到解决，扩展 PBD (XPBD) 对原始方法进行了小扩展，使刚度与迭代计数和物理单位的时间步长无关，并允许测量力和扭矩。作者还表明 XPBD 是隐式欧拉积分的近似。XPBD 的一个重要特点是采用了柔度，柔度是刚度的逆。这意味着通过将遵从性设置为零，我们可以轻松地以健壮的方式处理无限严格的约束。在这种情况下，XPBD 回落到 PBD。

从 Gauss-Seidel 方法返回的解通常依赖于约束解的顺序。在某些情况下，这种顺序依赖是有价值的，例如控制错误传播。然而，这种依赖关系可以通过使用 Jacobi 或对称连续过松弛来消除 (SSOR 迭代)。PBD 还可以处理基于常规有限元本构模型的连续介质力学约束[12]。这减轻了网格依赖刚度的问题。

最后，关于慢收敛的关注最近在[13]中得到了解决。通过用子步骤代替求解器迭代，高斯-赛德尔方法和雅可比方法在收敛性方面成为全局求解器的竞争对手。子步骤与每个子步骤的一次 NPGS 迭代相结合，产生的方法在计算上看起来几乎与显式积分步骤相同，但由于柔性的使用，具有无条件稳定的优点。我们称之为准显式方法。

在 Small Step[13]中描述的关于子步骤的发现是令人惊讶的。子步骤不只是减少时间步长。重要的思想是每帧的模拟时间负担，在实时应用程序中通常是恒定的，由子步骤的数量乘以每个子步骤的求解器迭代数给出。一种极端的选择是只使用一个子步骤，把所有的时间预算都花在高精度求解方程上。另一个极端是使用尽可能多的子步骤，只使用一次迭代来近似求解方程。令人惊讶的是，我们将在这篇论文中证明，就模拟的准确性而言，最好的选择是选择子步骤的最大数量，每个迭代一次。子步骤步进不仅是精度的最佳选择，它还揭示了使用大时间步进时错过的高频时间细节。子步骤步进还显著提高了能量守恒，并通过隧道降低了错过碰撞的几率。

由于使用一次迭代是精度方面的最佳选择，子步骤的数量可以从时间预算中得出，由于我们使用 XPBD，它允许使用真实的物理量，因此不需要调优任何参数。由于 XPBD 的无条件稳定性，也不需要为了稳定性而调整时间步长。

---

## 2 相关工作

刚体模拟在计算机图形学中有着悠久的历史。对于该领域的全面概述，我们建议读者参考 Bender 等人调查[14]。它涵盖了自 Baraff 的技术报告[15]以来在该领域发表的大部分重要工作。我们已经在引言中提到了大部分与我们的方法密切相关的具体工作。在这里，我们在这个列表中添加了更多的方法。

### 2.1 处理变化的接触法线

Xu 等人[16]提出了一种基于半隐式积分的仿真方法，他们分析接触梯度与 Penanlt 公式的关系，通过符号高斯消去，提高了系统的稳定性，从而可以处理时间变化的接触面。

为了提高处理接触的保真度，Wang 等人[17]通过将物体视为刚性变形对象，预先计算空间和方向变化的恢复系数，并解决代理接触问题。使用这些数据可以在模拟过程中获得更真实的弹跳行为。

### 2.2 基于位置的隐式时间离散和约束

基于位置的隐式时间离散和约束已被用于计算机辅助设计（CAD）和多体动力学软件，如 ADAMS 和 MBDyn[7,18-19] 。他们最常使用的是接触的 Penalty 模型，需要仔细调整参数，并且不允许完全硬的

接触响应。离线多体动力学软件也可能使用高阶积分方案，如二阶隐式欧拉（BDF2），提高了自由飞行物体的模拟精度。然而，对于非光滑轨迹，即刚体模拟中的典型情况，我们发现高阶积分可能产生虚假和不可预测的碰撞响应。这促使我们使用小时间步长和基于互补性的接触模型。

### 2.3 刚体与软体的耦合

Galvezden 等人[20]用运动关节连接刚体和可变形体系统来模拟非光滑动力学。所得到的接触问题用混合增广拉格朗日方法来表述。运动方程采用非光滑广义 u 时间积分格式进行积分。

### 2.4 形状匹配

为了处理刚体，我们用方向信息增强粒子。Müller 等人[21]利用这一思想通过形状匹配来稳定软物体的模拟。

后来 Umetani 等人[22]利用相同的思想来模拟基于位置的弹性杆

### 2.5 基于位置的动力学

与我们的方法最密切相关的方法是 Deul 等[9]。他们在 PBD 的位置框架下提出了刚体动力学公式。然而，他们的想法表达得有些模糊。一个例子是关于关节的简短段落，它只讨论了一种关节类型，而没有讨论关节极限——刚体发动机的核心特征。

## 3 本文方法

### 3.1 本文方法概述

对于采用拉格朗日视角的基于位置动力学的方法来说，其粒子一般仅仅使用位置 $x$, 速度 $v$ 以及质量 $m$ 描述，但是对于有限体积的刚体也包括一些角量，如：

1. 单位四元数描述它的方向（$q \in R^4, |q| = 1$）

2. 角速度 $\omega \in R^3$

3. 惯性张量 $I \in R^{3\times3}$

其中角速度向量可以分为单位选择轴和标量角速度：

$$\omega = \omega \cdot n_{rot}$$

惯性张量 $I$ 是与转动量中质量相对应的量。

因此，可以得到如下的伪代码 1：

**Procedure 1** Position Based Rigid Body Simulation

**while** *simulating* **do**
  CollectCollisionPairs();
  $h \leftarrow \Delta t / numSubsteps$;
  **for** *numSubsteps* **do**
    **for** *n bodies and particles* **do**
      $x_{prev} \leftarrow x$;
      $v \leftarrow v + h f_{ext}/m$;
      $x \leftarrow x + hv$;

      $q_{prev} \leftarrow q$;
      $\omega \leftarrow \omega + hI^{-1}(\tau_{ext} - (\omega \times (I\omega)))$;
      $q \leftarrow q + h\frac{1}{2}[\omega_x.\omega_y, \omega_z, 0]q$;
      $q \leftarrow q/|q|$;
    **end**
    **for** *numPosIters* **do**
      SolvePosition($x_1, ..., x_n, q_1, ..., q_n$);
    **end**
    **for** *n bodies and particles* **do**
      $v \leftarrow (x - x_{prev})/h$;
      $\Delta q \leftarrow q q_{prev}^{-1}$;
      $\omega \leftarrow 2[\Delta q_x, \Delta q_y, \Delta q_z]/h$;
      $\omega \leftarrow \Delta q_w \geqslant 0 ? \omega : -\omega$;
    **end**
    SolveVelocities($v_1, ..., v_n, \omega_1, ..., \omega_n$);
  **end**
**end**

### 3.2 核心投影操作

对主循环的修改非常简单。具有挑战性的部分是求解器的扩展，以处理位置框架中有限大小的物体之间的约束。幸运的是，我们只需要两个基本操作。求解任意关节、处理接触或刚体与软体耦合的任务都可以单独建立在这些操作之上，如图 3所示：

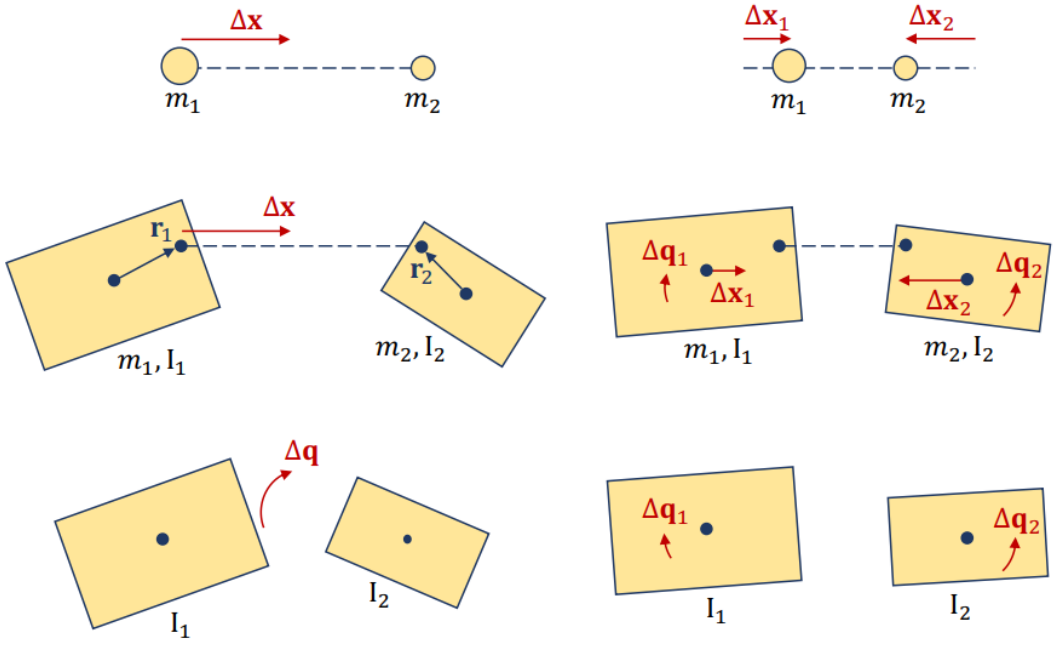


图 3: 投影操作

最上面一行显示了两个粒子之间的距离约束作为参考。对粒子上进行位置修正向量 $\Delta x$，这将会产生位置修正 $\Delta x_1$ 和 $\Delta x_2$, 其与粒子的质量的逆成比例，从而使线动量和角动量守恒。

### 3.2.1 位置约束

图 3中间一行是是第一个核心的投影操作：一对刚体上的点 $r_1$ 以及 $r_2$ 进行位置修正 $\Delta x$，这将会产生对质心位置应用位置修正 $\Delta x_1$ 和 $\Delta x_2$ 以及对刚体应用旋转修正 $\Delta q_1$ 和 $\Delta q_2$，其与他们的质量的逆以及转动惯量的逆的组合成比例。

为了在点 $r_1$ 以及 $r_2$ 进行位置修正 $\Delta x$, 我们首先将其分为方向 $n$ 以及大小 $c$。大小 $c$ 与 PBD 中的约束函数的值有关。然后我们计算两个广义质量逆：

$$w_1 \leftarrow \frac{1}{m_1} + (\mathbf{r}_1 \times \mathbf{n})^T \mathbf{I}_1^{-1} (\mathbf{r}_1 \times \mathbf{n})$$
$$w_2 \leftarrow \frac{1}{m_2} + (\mathbf{r}_2 \times \mathbf{n})^T \mathbf{I}_2^{-1} (\mathbf{r}_2 \times \mathbf{n})$$

在 XPBD 之后，我们可以更新拉格郎日乘子：

$$\Delta\lambda \leftarrow \frac{-c - \tilde{\alpha}\lambda}{w_1 + w_2 + \tilde{\alpha}}$$
$$\lambda \leftarrow \lambda + \Delta\lambda$$

其中 $\tilde{\alpha} = \alpha/h^2$ 和 $\alpha$ 是柔度约束。每个符合要求的约束都会存储一个拉格朗日乘子 $\lambda$，在迭代求解开始之前设置为零。

柔性对应的是刚性的逆并且单位是米/牛顿（meters/Newtow）。我们允许设置柔性 $\alpha = 0$ 来模拟无限刚性约束。

设置位置冲量 $p = \Delta\lambda n$, 我们在每个约束求解后立即更新物体的状态：

$$\mathbf{x}_1 \leftarrow \mathbf{x}_1 + \mathbf{p}/m_1$$
$$\mathbf{x}_2 \leftarrow \mathbf{x}_2 - \mathbf{p}/m_2$$
$$\mathbf{q}_1 \leftarrow \mathbf{q}_1 + \frac{1}{2}\left[\mathbf{I}_1^{-1}(\mathbf{r}_1 \times \mathbf{p}), 0\right]\mathbf{q}_1$$
$$\mathbf{q}_2 \leftarrow \mathbf{q}_2 - \frac{1}{2}\left[\mathbf{I}_2^{-1}(\mathbf{r}_2 \times \mathbf{p}), 0\right]\mathbf{q}_2$$

注意在第二主体上的更新中我们采用负号。

在处理每个约束之后我们立即更新可以防止 overshooting，这也是 PBD 健壮性的原因之一。同时这就产生了一个非线性的高斯-赛德尔投影解。当然我们可以使用雅可比（Jacobi）解用于并行实现或消除对约束投影顺序的依赖，但是代价是收敛速度比较慢。

求解后，沿约束作用的力可以由下面的公式得到：

$$f = \lambda n/h^2$$

对于连接一个定义为粒子系统的软体以及一个刚体是非常简单的，因为每个物体能够用一个粒子表示，通过设置 $w \leftarrow m^{-1}$ 并且忽略方向的更新。

### 3.2.2 角度约束

图 3 最下面一行上图表示的第二个核心的投影操作：一对刚体进行旋转修正——使他们的方向对齐，这将会产生旋转修正 $\Delta q_1$ 以及 $\Delta q_2$, 其与他们的的转动惯量的逆成比例，而质心不受这旋转修正的影响。

对于关节，我们需要约束两个物体相互方向的能力，因为我们需要上述的旋转修正。为了应用该旋转旋转修正 $\Delta q \in R^3$, 我们将其分为方向 $n$，表示为旋转轴，以及它的大小 $\theta$，表示为旋转角度。所以广义质量逆是：

$$w_1 \leftarrow \mathbf{n}^T \mathbf{I}_1^{-1} \mathbf{n}$$

$$w_2 \leftarrow \mathbf{n}^T \mathbf{I}_2^{-1} \mathbf{n}$$

XPBD 的更新和上面的位置约束一样，只不过用角度代替了距离：

$$\Delta\lambda \leftarrow \frac{-\theta - \tilde{\alpha}\lambda}{w_1 + w_2 + \tilde{\alpha}}$$

$$\lambda \leftarrow \lambda + \Delta\lambda$$

这里，修正只影响方向：

$$\mathbf{q}_1 \leftarrow \mathbf{q}_1 + \frac{1}{2}\left[\mathbf{I}_1^{-1}\left(\mathbf{r}_1 \times \mathbf{p}\right), 0\right]\mathbf{q}_1$$

$$\mathbf{q}_2 \leftarrow \mathbf{q}_2 - \frac{1}{2}\left[\mathbf{I}_2^{-1}\left(\mathbf{r}_2 \times \mathbf{p}\right), 0\right]\mathbf{q}_2$$

惯性张量 $I$ 取决于物体的实际方向。因此必须在每个约束投影之后更新。相反，我们在计算上述表达式之前，我们将 $n, r$ 以及 $p$ 投射到物体的静息状态上（the rest state of bodies）。

对于关节来说，接触点 $r$ 通常定义为静止状态（静息）。此外，我们在静止（静息）状态下旋转物体，使惯性张量变为对角线，这简化了上面的表达式，并允许将张量存储为一个向量。

与上面类似，我们可以推导出施加的力矩：

$$\tau = \lambda n / h^2$$

## 3.3 关节处理

我们现在描述如何使用上述两个修正操作处理各种关节。关节连接成对的物体，冰限制物体的相对位置和旋转程度。这对于刚体引擎来说十分重要。

### 3.3.1 旋转自由度

对于使两个物体相互对齐的关节，我们计算角度修正如下：

$$q = q_1 q_2^{-1}$$

$$\Delta q_{fixed} = 2(q_x, q_y, q_z)$$

为了建立的更一般的关节，我们必须分别在物体上都定义一个接触点 $\bar{r}$ 以及一组相互垂直的单位轴 $[\bar{a}, \bar{b}, \bar{c}]$，他们首先被转化到世界向量 $r$ 和 $[a, b, c]$

对于链接关节，我们希望轴 $a_1$ 和 $a_2$ 对齐，因此我们应用：

$$\Delta q_{hinge} = a_1 \times a_2$$

为了驱动链接关节到一个特定角度 $\alpha$，我们绕 $a_1$ 旋转 $b_1$ 一个角度 $\alpha$ 得到 $b_{target}$，然后应用：

$$\Delta q_{target} = b_{target} \times b_2$$

相关的柔度 $\alpha$ 控制约束的刚性。有一个目标角度约束，我们可以通过更新目标角度 $\alpha \leftarrow \alpha + h$ 在每一个子步（substep）创建一个速度驱动的马达，其中 $v$ 是马达的目标速度以及反应了柔性的大小。

处理关节限制是刚体引擎中的关键一部分。对于旋转自由度来说，这相当于限制了关节的角度。

我们使用伪代码 2定义的通用过程，它将两个物体的轴 $n_1$ 和 $n_2$ 限制在使用一个共同的旋转轴 $n$ 的角度范围 $[\alpha, \beta]$ 之间。

**Procedure 2** Handling joint and limits

**Input:** $n, n_1, n_2, \alpha, \beta$
$\phi \leftarrow (n_1 \times n_2) \cdot n$
**if** $n_1 \cdot n_2 < 0$ **then**
  $\phi \leftarrow \pi - \phi$
**end**
**if** $\phi > \pi$ **then**
  $\phi \leftarrow \phi - 2\pi$
**end**
**if** $\phi < -\pi$ **then**
  $\phi \leftarrow \phi + 2\pi$
**end**
**if** $\phi < \alpha$ *or* $\phi > \beta$ **then**
  $\phi \leftarrow clamp(\phi, \alpha, \beta)$;
  $n_1 \leftarrow rot(n, \phi)n_1$;
  Apply($\Delta q_{limit} = n_1 \times n_2$);
**end**

对于使用共同旋转轴 $a_1 = a_2$ 的铰链关节，我们使用 $[n, n_1, n_2] = [a_1, b_1, b_2]$。

对于球状关节（也叫做球窝接头），我们必须能够对轴 $a_2$ 相对于 $a_1$ 的运动之中区分出摆动（swing）和扭转（Twist）限制。为了限制摆动，我们使用 $[n, n_1, n_2] = [a_1 \times a_2, a_1, a_2]$。同时扭转必须与摆动解耦，我们通过以下轴进行实现：

$$n \leftarrow (a_1 + a_2)/(|a_1 + a_2|)$$
$$n_1 \leftarrow b_1 - (n \cdot b_1)n;\ n_1 \leftarrow |n_1|$$
$$n_2 \leftarrow b_2 - (n \cdot b_2)n;\ n_2 \leftarrow |n_2|$$

限制可以通过设置 $\alpha > 0$ 变得柔软。

### 3.3.2 位置自由度

处理位置自由度更简单，我们首先计算位置偏移 $\Delta r = r_2 - r_1$。设置 $\Delta x = \Delta r$ 从而不分离的情况下连接物体，这是关节的典型情况。使用 $\alpha > 0$ 允许一个零静息长度的弹簧。我们能够定义分离距离的上限 $d_{max}$ 使其变得更灵活。在这种情况下，我们只有当 $|\Delta r| > d_{max}$ 才应用修正：

$$\Delta x = \frac{\Delta r}{|\Delta r|}(|\Delta r| - d_{max})$$

我们也可以允许物体在边界内沿轴的一个子集移动来放松固定的附着。为此，我们从 $\Delta x$ 开始。

对于第一个轴 $a_1$, 我们计算在其上的投影位移 $a = \Delta r \cdot a_1$。如果 $a < a_{min}$, 我们添加 $a_1(a - a_{min})$ 到修正向量上，如果 $a > a_{max}$, 我们添加的是 $a_1(a - a_{max})$。在应用最后的修正向量之前，我们对所有轴和极限都这样做。这样，所有限制都用了一个单一的约束投影来处理。

将除了第一个轴外的所有限制设为零，这将模拟了一个移动关节（prismatic joints）。

对于一个机器人，我们可能想要驱动关节到一个特定的偏移量。通过将 $d_{max}$ 替换成 $d_{target}$，兵无条件地应用校正实现了这一点。

选择柔性 $\alpha = \frac{1}{f}(|\Delta r| - d_{target})$ 来应用一个力 $f$.

关节处理显示了非线性高斯-赛德尔方法在位置层上的优势。单边约束只需在某些条件成立时应用修正即可。而且，校正总是与当前的偏移和误差保持一致。此外，在线性化求解器中，附着是用一个约束来处理的，而不是三个约束。

### 3.4 处理接触与摩擦

为了减少计算花费，我们使用 AABB 树型结构在每一个时间步中（time step）收集潜在的碰撞对，而不是每一个子步（sub-step）。我们通过一个距离 $k\Delta t v_{body}$ 扩展 AABB，其中 $k \geqslant 1$ 是一个安全的乘数，考虑了时间步中的潜在加速度。在我们的例子中，使用 $k = 2$。

在每个子步中，我们迭代所有碰撞对检查实际的碰撞。如果碰撞发生，我们计算当前的接触法线以及两个物体的局部的接触点 $r_1$ 和 $r_2$。我们同时初始化两个拉格朗日乘数用于法向以及切线的力 $\lambda_n$ 和 $\lambda_t$ 为零。为了在位置上处理接触求解，我们计算两个物体基于当前状态以及在子步积分之前的接触位置：

$$p_1 = x_1 + q_1 r_1$$

$$p_2 = x_2 + q_2 r_2$$

$$\overline{p}_1 = x_{1,prve} + q_{1,prev} r_1$$

$$\overline{p}_2 = x_{2,prve} + q_{2,prev} r_2$$

其中四元数与向量的乘积指的是使用四元数旋转向量。

当前计算出的穿透为 $d = (p_1 - p_2) \cdot n$, 如果 $d \leqslant 0$，我们跳过该接触。

非线性高斯-塞德尔解算器让我们通过简单地检查每个约束的基础来处理互补性条件。如果两个物体正在穿透，我们使用 $\alpha = 0$ 和 $\lambda_n$ 并应用 $\Delta x = dn$。

为了处理静摩擦，我们计算在接触点的相对运动以及它的切向分量：

$$\Delta p = (p_1 - \overline{p}_1) - (p_2 - \overline{p}_2)$$

$$\Delta p_t = \Delta p - (\Delta p \cdot n)n$$

静摩擦力阻止了在接触点的切向运动，如果 $\Delta p_t = 0$ 就会出现这种情况。因此，为了强制执行静摩擦，我们在接触点上应用 $\Delta x = \Delta p_t$，其中 $\alpha = 0$，但只有在 $\lambda_t < \mu_s \lambda_n$，其中 $\mu_s$ 是静摩擦系数。如果两个物体有不同的系数，我们使用 $\mu = (\mu_1 + \mu_1)/2$。另一个选择是取最大值或最小值。

### 3.5 速度集

PBD 在位置求解之后更新速度并且立即进行下一子步。然而，处理动摩擦以及恢复，我们附加了一个速度解，如算法 2 所示。在这里，我们对所有的接触点进行一次迭代，并更新新的速度。

对于每个接触对，我们在接触点上计算对应的法线和切线速度：

$$v \leftarrow (v_1 + \omega_1 \times r_1) - (v_2 + \omega_2 \times r_2)$$

$$v_n \leftarrow n \cdot v$$

$$v_t \leftarrow v - nv_n$$

通过计算速度对摩擦力进行显式积分：

$$\Delta v \leftarrow -\frac{v_t}{|v_t|} min(h\mu_d|f_n|, |v_t|)$$

其中 $\mu_d$ 是动摩擦系数，$f_n = \lambda_b/h^2$ 是法向力。这种更新对应与动态库伦摩擦力的明确应用。

与高斯-赛德尔更新相联系的显式形式使我们能够使这一步无条件地稳定! 这个最小值保证了速度修正的大小永远不会超过速度本身的大小。

我们还可以利用速度通道来施加关节阻尼：

$$\Delta v \leftarrow (v_2 - v_1) min(\mu_{lin} h, 1)$$

$$\Delta \omega \leftarrow (\omega_2 - \omega_1) min(\mu_{ang} h, 1)$$

根据推导，在位置 $r_1$ 和 $r_2$ 应用速度更新 $\Delta v$ 是通过以下步骤实现的：

$$\mathbf{p} = \frac{\Delta \mathbf{v}}{w_1 + w_2}$$

$$\mathbf{v}_1 \leftarrow \mathbf{v}_1 + \mathbf{p}/m_1$$

$$\mathbf{v}_2 \leftarrow \mathbf{v}_2 - \mathbf{p}/m_2$$

$$\omega_1 \leftarrow \omega_1 + \mathbf{I}_1^{-1} (\mathbf{r}_1 \times \mathbf{p})$$

$$\omega_2 \leftarrow \omega_2 - \mathbf{I}_2^{-1} (\mathbf{r}_2 \times \mathbf{p})$$

为了处理恢复，我们需要 $\overline{v}_n$，即 PBD 速度更新之前的法线速度。我们通过对更新前的速度应用之前的公式计算出法线速度。考虑到恢复系数 $e$，我们希望在接触出的法线速度是 $-e\overline{v}_n$, 通过应用：

$$\Delta v \leftarrow n(-v_n + min(-e\overline{v}_n, 0))$$

我们减去当前速度 $v_n$ 并将将其换成反射速度 $-e\overline{v}_n$，这样确保了产生的速度指向碰撞法线方向。

为了避免颤抖，如果 $|v_n|$ 非常小，我们设 $e = 0$. 我们使用一个阈值 $|v_n| \leqslant 2|g|h$，其中 $g$ 是重力。这个值相当于预测步骤因重力加速度而增加的速度的两倍。

这一步骤也缓解了 PBD 的一个重要问题。PBD 的常规速度更新步骤产生的速度，只有在最后一个时间步骤中没有发生碰撞时才有意义。否则，他们只是反映了穿透深度，而穿透深度又取决于轨迹的离散性。另外，如果物体是在重叠状态下产生的，PBD 会产生较大的分离速度。上述公式消除了除碰撞外的派生速度，并考虑到了恢复系数的前一个时间步长的速度来代替它。在物体最初重叠的情况下，这个速度为零。

# 4 复现细节

## 4.1 细节

全部代码是自己实现的，没有参考相关源代码。代码使用 taichi 语言框架实现。

论文算法的核心部分是对两个核心投影操作进行实现，在 Section3.2中有具体细节的描述。

论文中另一个核心的部分是给出了对于关节处理的所有细节，从而能够模拟出任意的关节，并且

也给出接触后处理碰撞以及摩擦等细节。

根据论文的推导，将刚体模拟的约束大致分为了三类：

1. 距离约束：最大距离 $x$, 约束作用方向 $n$，在两个相互作用的物体上的作用点 $r_1, r_2$。

2. 碰撞约束：一种特殊的距离约束, 约束的柔性为 d, 当物体发生穿透时候才作用，在 Section 3.4上有碰撞的处理的具体描述。

3. 关节约束：关节约束可以分为三类固定关节，铰接关节，球状关节，在 Section3.3上有具体细节描述。

因此，整个求解器大致思路如下面的伪代码 3所示：（伪代码中 for 循环均是并行化）

**Procedure 3** XPBD Solver With Rigid Simulation

perform collision detection using $x^n, v^n$;
$\Delta t_s \Leftarrow \dfrac{\Delta t_f}{n_{steps}}$;
**while** $n < n_{steps}$ **do**
  **for** *all constraints* **do**
    $\lambda \leftarrow 0$
  **end**
  **for** *n bodies and particles* **do**
    $x_{prev} \leftarrow x$;
    $v \leftarrow v + hf_{ext}/m$;
    $x \leftarrow x + hv$;

    $q_{prev} \leftarrow q$;
    $\omega \leftarrow \omega + hI^{-1}(\tau_{ext} - (\omega \times (I\omega)))$;
    $q \leftarrow q + h\dfrac{1}{2}[\omega_x.\omega_y, \omega_z, 0]q$;
    $q \leftarrow q/|q|$;
  **end**
  **for** *all constraints* **do**
    According to the current constraint type to solve(Distance,Collision,Joints);
    compute $\Delta\lambda$ using $\Delta\lambda \leftarrow \dfrac{-c}{w_1 + w_2 + \tilde{\alpha}}$; ▷ $\Delta\lambda$ 计算公式有修改
    compute impulse $p$ using $p = \Delta\lambda n$;
    select to complete the core project operator(Position,Angle); ▷ See section 3.2
    update the current state including position $x$ and Quaterion $q$
  **end**
  **for** *n bodies and particles* **do**
    $v \leftarrow (x - x_{prev})/h$;
    $\Delta q \leftarrow q q_{prev}^{-1}$;
    $\omega \leftarrow 2[\Delta q_x, \Delta q_y, \Delta q_z]/h$;
    $\omega \leftarrow \Delta q_w \geqslant 0 ? \omega : -\omega$;
  **end**
  SolveVelocities($v_1, ..., v_n, \omega_1, ..., \omega_n$); ▷ See section 3.5
  $n \Leftarrow n + 1$;
**end**
**for** *all constraints* **do**
  compute force $f = \lambda n/h^2$ or compute moment of force $\tau = \lambda n/h^2$
**end**

同时根据论文中的推导，我们可以计算出每个约束沿着约束作用方向的的力以及力矩。

### 4.2 实验环境搭建

本次复现使用了 taichi 语言框架，借助 taichi 多平台多后端的能力，仅需要将 Github 仓库中的 requirements.txt 的 python 环境安装后。即可在 windows,linux,mac 上运行。

### 4.3 界面分析与使用说明

按照目前的复现效果，提供了两个场景，具体运行参考仓库的 readme。

一个三摆场景, 如下图 4所示

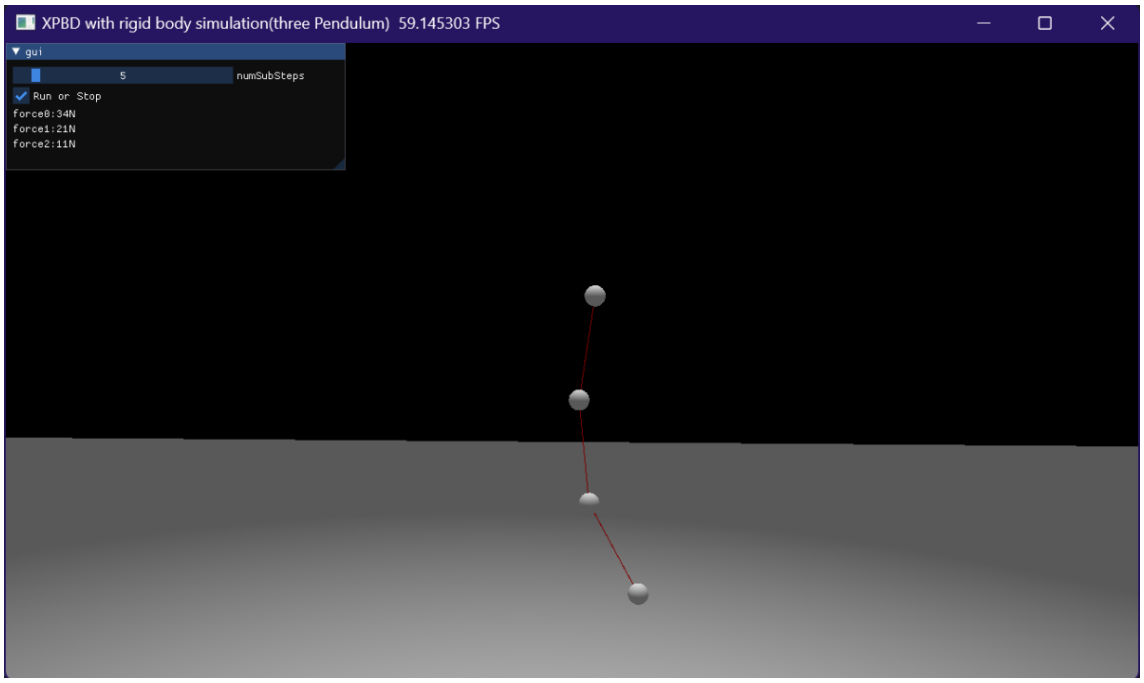


图 4: 三摆场景

一个盒子场景, 如下图 5所示

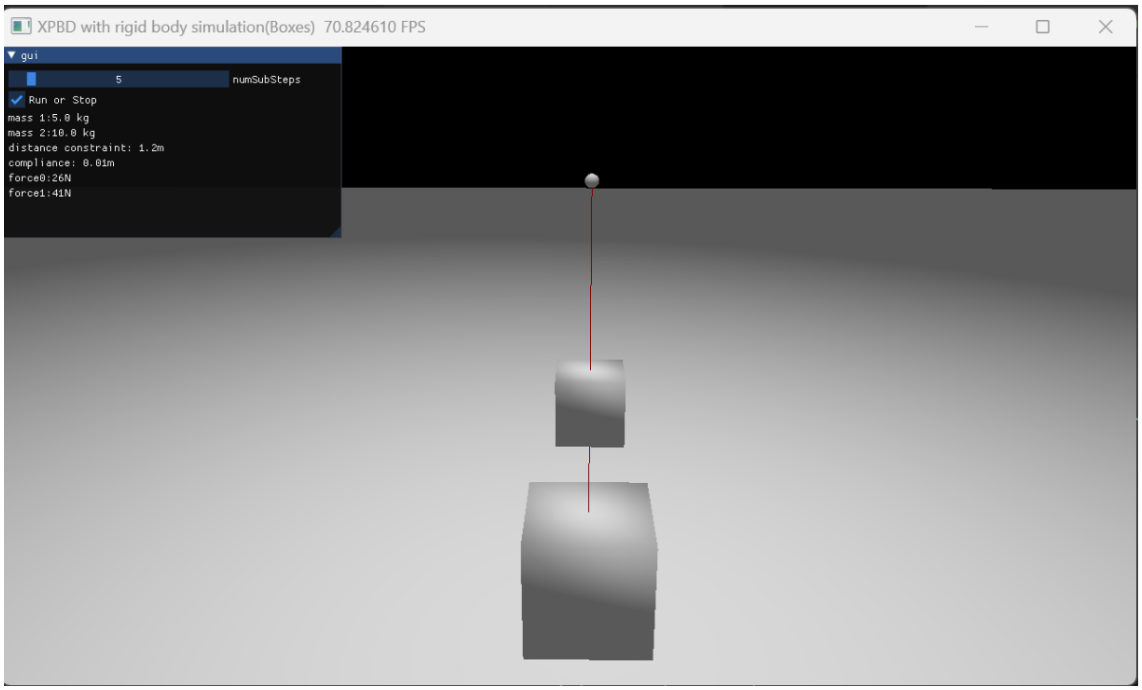


图 5: 盒子场景

### 4.4 创新点

对于在核心投影操作计算 XPBD 的拉格朗日乘子 $\lambda$, 在 Small Step[13]论文的描述上，在一个时间步长上采取子步骤迭代一次所有约束比一个时间步迭代多次约束可以显著得提高了能量守恒，所以在

实现约束求解时候我采用的是前者，并且根据其描述 XPBD 乘子的计算中，因此每次迭代约束时候总的拉格郎日乘子需要初始化为零，但是我们仅仅迭代一次，所以在 $\Delta\lambda$ 计算过程中，我们可以将分子上的 $\lambda$ 省去，减少冗余的计算，具体公式如下：

$$\Delta\lambda \leftarrow \frac{-c}{w_1 + w_2 + \tilde{\alpha}}$$

$$\lambda \leftarrow \lambda + \Delta\lambda$$

本次复现采取 taichi 语言编程框架，我们仅仅通过写一份 python 代码，就可以在多平台（win,linux,mac）以及多个后端（CPU,CUDA,Metal）运行，并且具有媲美使用相关后端实现的性能，本次复现也提供了 CPU 与 GPU 后端在 windows 平台的性能差异。

## 5 实验结果分析

本次复现的算法使用 taichi 语言框架，其不同场景在 CPU 以及 GPU 后端（CUDA）的性能结果如下表 1所示：

| Example | numSubSteps = 5 | numSubSteps = 10 |
|---|---|---|
| Boxes(CPU) | 80FPS | 60FPS |
| Boxes(GPU) | 43FPS | 30FPS |
| Three pendulum(CPU) | 60FPS | 48FPS |
| Three pendulum(GPU) | 35FPS | 24FPS |

表 1: 帧数对比（时间步长 =$1/60s$）

在图 4以及图 5的结果中可以看到，对于仅子步骤次数等于 5，求解器的算法的结果以及收敛，不仅可以到实时的效果，并且解决了 PBD 算法中的几个重要问题, 该求解器具有的性质如下：

1. 积分过程是物理的，可以推导出相应的作用力。

2. 约束的刚性不依赖于迭代次数以及时间步长。

3. 收敛快。

但是从上表 1中可以看到，在 GPU 并行的效果并不如在 CPU 并行的效果高，这也是本次算法的任务比较复杂，并且场景的量不大，对于 GPU 并行来说在传输数据以及 kernel 函数启动的花费也比更多，因此性能提升并不大甚至降低。三摆场景的具体性能分析如下图所示：

```
================================================================================
Kernel Profiler(count, default) @ X64
================================================================================
[      %     total   count |      min       avg       max    ] Kernel name
--------------------------------------------------------------------------------
[ 20.54%   1.357 s  16190x |    0.039     0.084     1.016 ms] InitConstraintLambda_c100_0_kernel_2_struct_for
[ 20.15%   1.331 s  16190x |    0.038     0.082     0.900 ms] SolveConstraint_c96_0_kernel_4_struct_for
[ 11.87%   0.784 s  16190x |    0.007     0.048     0.502 ms] preSolve_c92_0_kernel_5_struct_for
[ 11.45%   0.756 s  16190x |    0.007     0.047     0.647 ms] UpdateAfterSolve_c94_0_kernel_4_struct_for
```

图 6: CPU

```
================================================================================
Kernel Profiler(count, default) @ CUDA on NVIDIA GeForce RTX 3070 Laptop GPU
================================================================================
[      %     total   count |      min       avg       max    ] Kernel name
--------------------------------------------------------------------------------
[ 39.57%   3.056 s   3230x |    0.824     0.946    20.593 ms] SolveConstraint_c96_0_kernel_4_struct_for
[  5.47%   0.423 s   3600x |    0.006     0.117     2.396 ms] snode_reader_451_kernel_0_serial
[  2.40%   0.185 s   4800x |    0.010     0.039     0.671 ms] snode_reader_375_kernel_0_serial
[  2.34%   0.181 s   3600x |    0.010     0.050     1.921 ms] snode_reader_452_kernel_0_serial
[  2.22%   0.172 s   4800x |    0.010     0.036     1.013 ms] snode_writer_456_kernel_0_serial
[  2.19%   0.169 s   4800x |    0.010     0.035     0.315 ms] snode_reader_374_kernel_0_serial
[  1.92%   0.149 s   3600x |    0.010     0.041     2.243 ms] snode_reader_453_kernel_0_serial
```

图 7: GPU

# 6 总结与展望

本次复现实现了一个基于 XPBD 的刚体求解器，可以解决小的时间和空间细节，并且是无条件稳定。相当于 PBD 的刚体求解器，该求解器的刚性不依赖于迭代次数以及时间步长，积分是在物理上是准确的，可以推导出沿着约束作用方向的力以及力矩，同时求解器的收敛是非常快的。

在复现的过程中，不仅学习到物理模拟仿真的实现思路，一般是对一个非线性的方程组的求解，而基于位置的动力学采取于一般物理模拟仿真不同的思路，他从位置层出发，从而反推出速度等其他矢量的信息，但是 PBD 效果是非物理的，后续也提出了 XPBD，XPBD 解决了 PBD 非物理等的一系列问题；同时论文提到的算法，在 XPBD 的基础上引入了刚体运动学，也学习到了很多刚体运动学的知识。

目前的代码是有实现对应关节处理以及接触后碰撞和摩擦处理的算法，由于时间的关系以及碰撞的实现过程相对复杂，其代码测试并没有做，因此无法保证其正确性，后续有时间将其完善并提高场景进行测试。

对于物理模拟的约束的迭代求解，一般有两种方法，一个是高斯-赛德尔迭代方法（非线性的高斯-赛德尔迭代），一个是雅可比迭代方法，由于本次复现的论文是基于高斯-赛德尔方法，并且采用 taichi 语言框架，主要是想采用其自动并行的能力，但是对于高斯-赛德尔迭代并不适合并行，因为其的迭代过程是顺序依赖的，并行化后因为运行顺序不可预知，所以可能会导致求解不正确。由于时间关系，并没有实现并行化后的高斯-赛德尔迭代过程，后续考虑将其并行化。

# 参考文献


[1] MIRTICH B, CANNY J. Impulse-based simulation of rigid bodies[C]//Proceedings of the 1995 symposium on Interactive 3D graphics. 1995: 181-ff.

[2] HECKER C. Physics, part 4: The third dimension[J]. Game Developer Magazine, 1997: 15-26.

[3] BARAFF D. An introduction to physically based modeling: rigid body simulation I—unconstrained rigid body dynamics[J]. SIGGRAPH course notes, 1997, 82.

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

[5] STAM J. Nucleus: Towards a unified dynamics solver for computer graphics[C]//2009 11th IEEE International Conference on Computer-Aided Design and Computer Graphics. 2009: 1-11.

[6] MACKLIN M, MÜLLER M. Position based fluids[J]. ACM Transactions on Graphics (TOG), 2013, 32(4): 1-12.

[7] MACKLIN M, MÜLLER M, CHENTANEZ N, et al. Unified particle physics for real-time applications [J]. ACM Transactions on Graphics (TOG), 2014, 33(4): 1-12.

[8] MÜLLER M, HEIDELBERGER B, TESCHNER M, et al. Meshless deformations based on shape matching[J]. ACM transactions on graphics (TOG), 2005, 24(3): 471-478.

[9] DEUL C, CHARRIER P, BENDER J. Position-based rigid-body dynamics[J]. Computer Animation and Virtual Worlds, 2016, 27(2): 103-112.

[10] ERLEBEN K. Rigid body contact problems using proximal operators[C]//Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. 2017: 1-12.

[11] DAVIET G, BERTAILS-DESCOUBES F, BOISSIEUX L. A hybrid iterative solver for robustly capturing coulomb friction in hair dynamics[C]//Proceedings of the 2011 SIGGRAPH Asia Conference. 2011: 1-12.

[12] 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.

[13] 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.

[14] BENDER J, ERLEBEN K, TRINKLE J. Interactive simulation of rigid body dynamics in computer graphics[C]//Computer Graphics Forum: vol. 33: 1. 2014: 246-270.

[15] BARAFF D. Non-penetrating rigid body simulation[J]. State of the art reports, 1993.

[16] XU H, ZHAO Y, BARBIČ J. Implicit multibody penalty-baseddistributed contact[J]. IEEE transactions on visualization and computer graphics, 2014, 20(9): 1266-1279.

[17] WANG J H, SETALURI R, JAMES D L, et al. Bounce maps: an improved restitution model for real-time rigid-body impact.[J]. ACM Trans. Graph., 2017, 36(4): 150-1.

[18] ORLANDEA N, CHACE M A, CALAHAN D A. A sparsity-oriented approach to the dynamic analysis and design of mechanical systems—Part 1[J]., 1977.

[19] RYAN R. ADAMS—Multibody system analysis software[J]. Multibody systems handbook, 1990: 361-402.

[20] GALVEZ J, CAVALIERI F J, COSIMO A, et al. A nonsmooth frictional contact formulation for multi-body system dynamics[J]. International Journal for Numerical Methods in Engineering, 2020, 121(16): 3584-3609.

[21] MÜLLER M, CHENTANEZ N. Solid simulation with oriented particles[G]//ACM SIGGRAPH 2011 papers. 2011: 1-10.

[22] UMETANI N, SCHMIDT R, STAM J. Position-based elastic rods[G]//ACM SIGGRAPH 2014 Talks. 2014: 1-1.