

【物理模拟】PBD算法详解

原创

beidou111

已于 2022-07-30 22:58:00 修改

阅读量5.1k

收藏 36

点赞数 9

版权

分类专栏:

计算机图形学

文章标签:

算法

基于物理的动画



计算机图形学 专栏收录该内容

12 订阅

41 篇文章

订阅专栏

参考：

Matthia Muller的十分钟物理（他就是PBD算法的发明者）
<https://matthias-research.github.io/pages/tenMinutePhysics/>

原理

PBD的算法主体分为三步：

1. 根据外力更新粒子速度位置，无需考虑粒子间关系
2. 求解约束，使粒子满足粒子间关系
3. 更新粒子的位置，并反向更新速度

由于第3步是先求出粒子位置，再反求速度的。因此它是基于位置的方法，故被称为position based dynamics。

PBD是纯粒子法。在PBD的世界中，只有粒子。其他的一切（三角面、四面体等）都是辅助性的。

```
while simulating
  for all particles i
     $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t \mathbf{g}$ 
     $\mathbf{p}_i \leftarrow \mathbf{x}_i$ 
     $\mathbf{x}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$ 

  for all constraints C
    solve(C,  $\Delta t$ )

  for all particles i
     $\mathbf{v}_i \leftarrow (\mathbf{x}_i - \mathbf{p}_i) / \Delta t$ 
```

```
solve(C,  $\Delta t$ ):

  for all particles i of C
    compute  $\Delta \mathbf{x}_i$ 
     $\mathbf{x}_i \leftarrow \mathbf{x}_i + \Delta \mathbf{x}_i$ 
```

1. 根据外力更新粒子速度位置，无需考虑粒子间关系

如图，先根据外力更新速度（这里只考虑了重力）

然后把旧的位置存到p中

最后利用速度更新位置

（如果有碰撞，也在这里处理）

整个过程粒子无需考虑相互关系。

```
for all particles i
   $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t \mathbf{g}$ 
   $\mathbf{p}_i \leftarrow \mathbf{x}_i$ 
   $\mathbf{x}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$ 
```

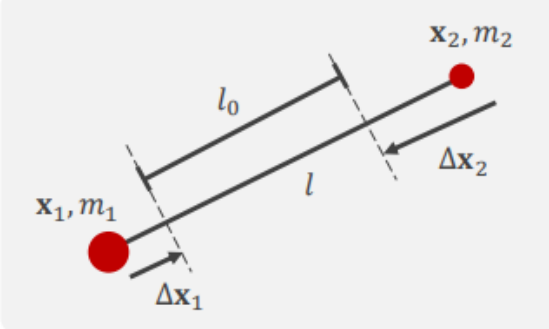
2. 求解约束，使粒子满足粒子间关系

我们分两种情况：一种是最简单的二维情况，一种是三维情况。

我们先从简单的二维情况来：

二维情况：一个弹簧

我们只考虑一个弹簧的约束



这个弹簧只有两个质点和中间的一个边。质点具有位置和质量，边具有



beidou111

关注

👍 9

💬

🌟 3

随意移动两个粒子，弹簧目前不处于原长。

因此，弹簧要回到原长。

弹簧回到原长，这就是这个系统的约束。

$$C(x1,x2) = |x1 - x2| - L0$$

其中x1, x2分别是粒子1和粒子2的位置。L0是原长。

如何让系统满足约束呢？

所谓的满足约束，就是让约束误差等于0。

上面这个式子中的C，就是约束的误差（有时候约束和约束误差这两个词混用）。

通过迭代，让误差趋于0，那就是求解过程。

如何让其趋于0？

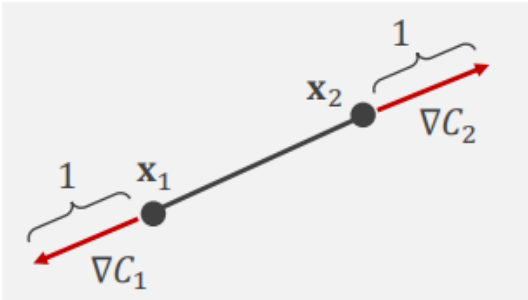
那就是求梯度，然后向着梯度减小的方向移动（即所谓的梯度下降思想）。

实际上，从另一个角度来看，梯度就是一维的导数在高维的推广。让导数等于0的位置（即驻点），就是原函数最小化的位置。

于是我们就找C的梯度。

$$\nabla C1 = (x1 - x2)/|x1 - x2| \nabla C2 = (x2 - x1)/|x2 - x1|$$

我们无需那样严谨地去推导数学公式，然后给出C梯度的表达式。我们直接从物理意义上来理解C梯度。那就是让C函数上涨最快的方向。在这个例子中，也就是x1与x2相对的方向。因此我们给出上面的公式。而且我们这里是归一化了的，只求出一个方向。



$$\nabla C_1 = \frac{x_1 - x_2}{|x_1 - x_2|}$$
$$\nabla C_2 = \frac{x_2 - x_1}{|x_2 - x_1|}$$

那么大小是多少呢？我们给出如下公式

$$\lambda = \frac{-C}{w_1|\nabla C_1|^2 + w_2|\nabla C_2|^2 + \cdots + w_n|\nabla C_n|^2} = \frac{-(l - l_0)}{w_1 \cdot 1 + w_2 \cdot 1}$$
$$\Delta x_1 = \lambda w_1 \nabla C_1 = -\frac{w_1}{w_1 + w_2} (l - l_0) \frac{x_2 - x_1}{|x_2 - x_1|}$$

大小是由系数lambda和质量倒数w决定的。其中lambda的正式名称叫做拉格朗日乘数。

那么，我们求解出来了gradC，因此就求解出来了粒子间的相互关系。因此，也就知道粒子为了满足相互关系，该向哪里移动。因此给出了dx（如上图）。这个移动的方向，就是最小化约束误差的方向，就是负梯度的方向（因此lambda分母有个负号）。

三维情况：一个四面体

对于三维，我们期望的约束是四面体的体积保持原体积。

其约束误差就是

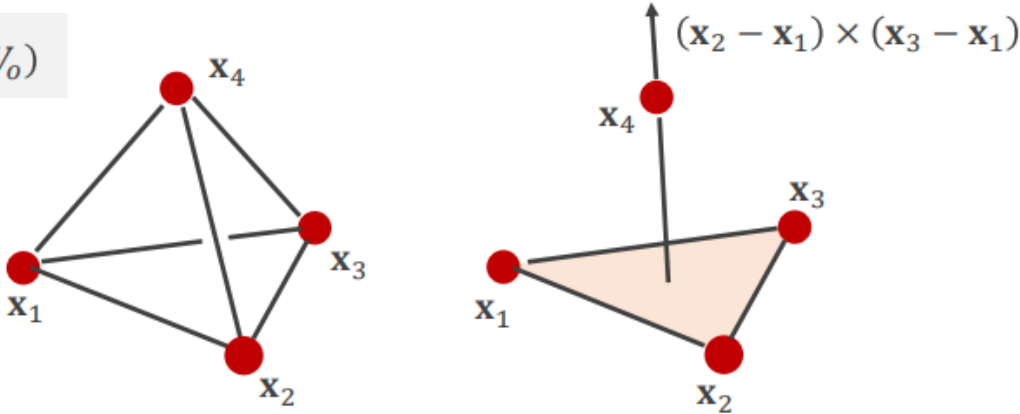
$$C = (V - V0)$$

而四面体的体积公式是

$$\frac{1}{6} [(x2 - x1) \times (x3 - x1)] \cdot (x4 - x1)$$

先叉乘求出底面积（叉乘得到的是菱形面积，还要除以2），然后再乘以高度（点乘会消除非垂直部分），再乘以1/3(四面体公式中本来的1/3)

$$C = 6(V - V_o)$$



$$C = 6(V - V_o) = [(\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)] \cdot (\mathbf{x}_4 - \mathbf{x}_1) - 6V_o$$

$$\nabla_4 C = (\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)$$

而约束误差的梯度是什么呢？我们仍然跳过数学推导，从物理意义上解释。

梯度即函数增长最快的方向，也就是体积增长最快的方向。对于某个粒子来说，哪个方向让体积增长最快呢？

那就是垂直于底面的方向。

而垂直与底面的方向，就是底面的三角形其中两个边叉乘的方向（叉乘满足右手定则）。

因此

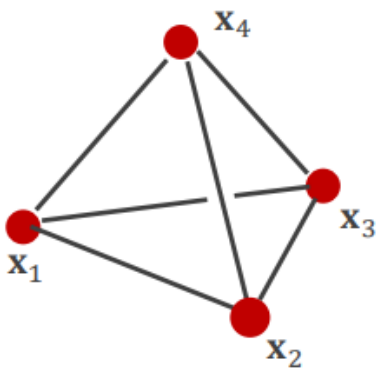
$$gradC4 = (x2 - x1) \times (x3 - x1)$$

这就是粒子4所对应约束的梯度。

那么大小如何确定呢？

仍然利用拉格朗日乘数lambda。

Solve



Right hand rule

$$\nabla_1 C = (\mathbf{x}_4 - \mathbf{x}_2) \times (\mathbf{x}_3 - \mathbf{x}_2)$$

$$\nabla_2 C = (\mathbf{x}_3 - \mathbf{x}_1) \times (\mathbf{x}_4 - \mathbf{x}_1)$$

$$\nabla_3 C = (\mathbf{x}_4 - \mathbf{x}_1) \times (\mathbf{x}_2 - \mathbf{x}_1)$$

$$\nabla_4 C = (\mathbf{x}_2 - \mathbf{x}_1) \times (\mathbf{x}_3 - \mathbf{x}_1)$$

$$\lambda = \frac{-6(V - V_o)}{w_1 |\nabla C_1|^2 + w_2 |\nabla C_2|^2 + w_3 |\nabla C_3|^2 + w_4 |\nabla C_4|^2}$$

$$\Delta \mathbf{x}_i = \lambda w_i \nabla C_i$$

值得注意的是：

求解一个三维弹性物体，我们必须同时满足弹簧约束和体积约束。体积约束保证弹性体不发生体积的膨胀和缩小，而弹簧约束保证物体上的每个质点回到原位（也就是最终弹性体恢复原状）。

代码

我们参考的是

<https://matthias-research.github.io/pages/tenMinutePhysics/>

中第10讲的代码

数据结构：

存储在class SoftBody内。SoftBody可多次实例化以添加多个物体。

simulate函数为算法的主体，它分为三步：

1. preSolve
2. solve
3. postSolve

preSolve

```
1 for all particles:
2     vel[i] += gravity * dt
```



beidou111

关注

👍 9



3

```
3   prevPos[i] = pos[i]
4   pos[i] += vel[i] *dt
5   collision处理: 当y<0时把pos挪到0
```

solve

又被分为两个部分:

1. solveEdge
2. solveVolume

也就对应上面说的弹簧约束和体积约束。

solveEdge

```
1   for i in all edges:
2       alpha = C/dt^2
3       grad = pos0- pos1
4       grad归一化
5       s = - (L - L0)/(1/m + alpha)
6       pos0[i] += grad * (s/m)
7       pos1[i] += grad * (-s/m)
```

solveVolume

```
1   for i in all tets:
2       alpha = C/dt^2
3       for j in 4:
4           temp0 = pos1 - pos0
5           temp1 = pos2 - pos0
6           grad[j] = (1/6) * tem0.cross(temp1)
7   w = sum((1/m) * ||grad[j]||^2 )
8   s = -(V - V0) / (w + alpha)
9   for j in 4:
10      pos[i] = grad[j] * (s/m)
```

postSolve

```
1   for i in all particles:
2       vel[i] = (pos[i] - prevPos[i])/dt
```

代码

拷贝自tenMinutePhysics

```
preSolve(dt, gravity)
{
    for (var i = 0; i < this.numParticles; i++) {
        if (this.invMass[i] == 0.0)
            continue;
        vecAdd(this.vel,i, gravity,0, dt);
        vecCopy(this.prevPos,i, this.pos,i);
        vecAdd(this.pos,i, this.vel,i, dt);
        var y = this.pos[3 * i + 1];
        if (y < 0.0) {
            vecCopy(this.pos,i, this.prevPos,i);
            this.pos[3 * i + 1] = 0.0;
        }
    }
}
```



beidou111

关注

👍 9



3

```

solveEdges(compliance, dt) {
    var alpha = compliance / dt /dt;

    for (var i = 0; i < this.edgeLengths.length; i++) {
        var id0 = this.edgeIds[2 * i];
        var id1 = this.edgeIds[2 * i + 1];
        var w0 = this.invMass[id0];
        var w1 = this.invMass[id1];
        var w = w0 + w1;
        if (w == 0.0)
            continue;

        vecSetDiff(this.grads,0, this.pos,id0, this.pos,id1);
        var len = Math.sqrt(vecLengthSquared(this.grads,0));
        if (len == 0.0)
            continue;
        vecScale(this.grads,0, 1.0 / len);
        var restLen = this.edgeLengths[i];
        var C = len - restLen;
        var s = -C / (w + alpha);
        vecAdd(this.pos,id0, this.grads,0, s * w0);
        vecAdd(this.pos,id1, this.grads,0, -s * w1);
    }
}

solveVolumes(compliance, dt) {
    var alpha = compliance / dt /dt;

    for (var i = 0; i < this.numTets; i++) {
        var w = 0.0;

        for (var j = 0; j < 4; j++) {
            var id0 = this.tetIds[4 * i + this.volIdOrder[j][0]];
            var id1 = this.tetIds[4 * i + this.volIdOrder[j][1]];
            var id2 = this.tetIds[4 * i + this.volIdOrder[j][2]];

            vecSetDiff(this.temp,0, this.pos,id1, this.pos,id0);
            vecSetDiff(this.temp,1, this.pos,id2, this.pos,id0);
            vecSetCross(this.grads,j, this.temp,0, this.temp,1);
            vecScale(this.grads,j, 1.0/6.0);

            w += this.invMass[this.tetIds[4 * i + j]] * vecLengthSquared(this.grads,j);
        }
        if (w == 0.0)
            continue;

        var vol = this.getTetVolume(i);
        var restVol = this.restVol[i];
        var C = vol - restVol;
        var s = -C / (w + alpha);

        for (var j = 0; j < 4; j++) {
            var id = this.tetIds[4 * i + j];
            vecAdd(this.pos,id, this.grads,j, s * this.invMass[id])
        }
    }
}

```




```
postSolve(dt)
{
    for (var i = 0; i < this.numParticles; i++) {
        if (this.invMass[i] == 0.0)
            continue;
        vecSetDiff(this.vel,i, this.pos,i, this.prevPos,i, 1.0 / dt);
    }
    this.updateMeshes();
}
```

文章知识点与官方知识档案匹配，可进一步学习相关知识

算法技能树 首页 概览 60564 人正在系统学习中

20天，从大厂被刷到拿到另一个大厂的Offer，空窗期半年的小伙伴做对了什么？

广告

讨论一个问题，大厂或者背景实力好的人，是否还需要培训？ 我们用实例来证明，越是按我们“常规”理解所谓厉害的人，他们对自己增...

Vellum —— 简介

4-9

三,PBD算法 手动实现PBD的拉伸 一,介绍 Vellum是一个解算模拟框架,使用更高级的PBD(XPBD,extended position based dynamics),是2nd Order Integration(传统的PBD使用的是1nd Order Integrat...

Bullet 布料仿真的底层算法分析_布料仿真算法

4-17

3.1 PBD算法 先给结论,Bullet布料仿真建模用的是有限元方法(Finite Element Method, FEM)的底层算法是基于位置约束的动力学算法(Position-Based Dynamics,PBD),这个在PyBullet Quickstart Gui...

unity给头发添加物理_基于PBD算法的布料和头发的实时模拟 (Unity Demo)

weixin_39736047的博客

1423

什么是PBD算法PBD算法的全称是Position-Base-Dynamic，顾名思义，它是一种计算机视觉方面的动态模拟算法，主要用于各种需要实时模拟的场景。相比起过去的基于真实物理规则的模拟来说...

PID算法的解析 热门推荐

往事撩人醉的博客

8万+

LZ以前有个小小的理想，就是让手边的MCU自己“思考”起来，写出真正带算法的程序。 前段时间做一个比赛项目的过程中，对经典、实用的PID算法有了一点点自己的理解，就写了这些，与大家分...

移动端或游戏布料的解算

微小的鱼的博客

1271

主要介绍PBD算法，目标是移动端能实现布料的模拟与仿真，讲述布料模拟的基础理论原理，数学公式，PBD算法的详解；有Nvcloth、GPU等的实现方案。

Houdini Grain 学习笔记

weixin_30496431的博客

304

// Grain使用PBD算法，说白了就是先更新位置，再更新速度，Houddini MasterclassGrain中的案例有一些简单的实现（最基本的原理） 比如跟新位置（这里为了简化Jeff把粒子的pscale大小都看成...

unity 绳子模拟 Position Based Dynamics

hughsjk的博客

1148

unity 绳子模拟 Position Based Dynamics PBD采用几何的方式，通过先建立约束再对约束进行投影的方式来直接计算出位置和速度。这里的约束投影可以理解为通过数学公式计算出物体下一个时...

Everything about PBD：关于PBD的一切！

beidou111的博客

2316

原始PBD可以认为是alpha为0的系统，因此是刚度无穷大的系统，条件数比较大。但是它是能够产生三维的变形的。根据刘天添的fast simulation of mass-spring system 中的论证，XPBD相当于是...

谱聚类PBD算法 复杂网络

07-19

谱聚类PBD算法源码 基于优化模块度和谱原理 参考文献： Clustering algorithm for determining community structure in large networks [J] Physical Review E 74 1 2006 016107

PBD（Position Based Dynamics）学习笔记

dragonylee的博客

2305

符号说明 仿真物体包括 NNN 个节点和 MMM 个约束。 每个节点 $i \in [1, ..., N]$ 包含的参数有质量 m_i 、位置 \mathbf{x}_i 和速度 \mathbf{v}_i ； 每个约束 $j \in [1, ..., M]$ 包含的参数有质量 m_j 、位置 \mathbf{x}_j 和速度 \mathbf{v}_j 。

10分钟完成MySQL物理xtrabackup增量备份

weixin_33994444的博客

103

在生产环境中，我们一般采取mysqldump全库备份，但这对于大型数据库是不可用的，因为mysqldump是逻辑备份，备份和恢复的效率缓慢，根据个人经验给出一个临界值的话，50G以下的数据库...

PBD文件查看DW

08-24

查看PBD文件里面的datawindow语法，源码是PB9的，可以自己升级到更高版本，非常方便和实用

一款很好用的pbd反编译软件

10-07

一款很好用的PBD反编译软件,我试过，很不错。

PBD统一流体：在GPU上统一运行的PBD流体

02-06

这是在GPU上的Unity中运行的基于位置的动力学粒子流体模拟。 它使用与先前相同的数学方法，但修改了一些要在GPU上运行的内容。 最大的区别是如何处理查找相邻粒子。 这在GPU上更为复...

pbd反编译软件超级好用pb

08-28

前几天，我不小心把pb90的pbl删了，里面有好多写的新代码，还有公式，在网站找了一圈后，终于找到可以导出w窗口的软件，可是数据窗口不行，又找了一圈，终于找到导出的软件。很好用。

PBD计算机动画模拟教程.pdf

07-12

PBD计算机动画模拟教程.pdf

基于PBD和体积约束的薄壳弹性碰撞变形模拟

04-07

基于PBD和体积约束的薄壳弹性碰撞变形模拟

PBD加密测试.rar

03-18

能对抗目前 PB DeCompiler ,shudepb, pbkiller 3类反编译工具反编译 能加密PB5-PB12.5 任意版本的PB程序

Leetcode 41. 缺失的第一个正数和Leetcode 155. 最小栈 最新发布

2301_79614379的博客

1078

定义一个结构体，数组 data 用于存放推进的数据，数组 min 用于存放最小值，变量 top 存放最后进入的数在 data 的下标，变量 mintop 记录最小值在 min 中的下标。循环遍历数组 nums，如果该...

力扣---从中序与后序遍历序列构造二叉树

m0_71536793的博客

462

力扣 中序遍历 后序遍历 建树

python pbd 用法

04-04

Python PDB（Python Debugger）是一个交互式调试器，它允许程序员在程序执



beidou111

关注

9



3