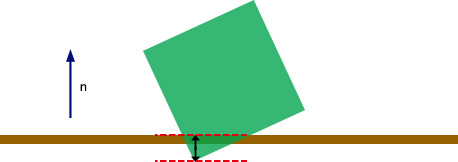
在上篇文章中，我们成功推导出了刚体碰撞后运动的公式。但是，仔细想一下的话，我们仍然没有解决刚体之间穿透的问题。如下图



刚体虽然在碰撞后会反弹，但反弹距离跟刚体穿透距离无关，并不一定能分开两个刚体。而且，如果两个刚体的相对速度是0，也就是相对静止，根据牛顿碰撞定律，反弹速度也是0，这种情况下，就更不可能用反弹来解决穿透问题了。所以，我们需要一种更直接的方式将两个刚体分开，该方法就是本篇文章将要讲解的主要内容——基于PBD的约束求解。

在人们想出用约束来解决穿透问题之前，还有很多别的方法，比如罚函数法，直观的解释可以想象在两个刚体重叠处放一个弹簧，这样在发生穿透时，弹簧可以把两个刚体弹开，类似的方法还有force based method和impulse based method，这些方法都会模拟一种力，有点像反弹力，但又不等同于真实的反弹力。因为产生穿透的根本原因是计算机企图用线性离散运动来模拟高阶连续世界导致的，是一种技术上误差，就算没有这些误差，反弹力也是真实存在的，用模拟力的方式来消除这种位置上的误差，虽然有些曲线救国的意思，但好在遵守了牛顿运动法则，不会违反动量守恒和能量守恒定律。缺点也是比较明显的，无法及时修正误差，修正也不准确，经常会在正确的道路上左右横跳。本文介绍的方法完美解决了这些问题，基于PBD（position based dynamic）的约束求解。

约束可以简单理解为对物体运动的限制，一个任意运动的物体，我们可以限制它的移动速度，可以限制它的移动范围，可以限制它的移动时间等等。这些限制当然也可以用经典牛顿力学来实现，但与其创造各种力来限制位置、速度等，还不如直接限制来的简单精确。所以约束在分析力学中使用非常广泛。

根据不同的条件可以将约束划分为不同的类型，在分析力学中，约束被划分为可解约束、不可解约束，稳定约束、不稳定约束等等。而在游戏引擎领域，我们可以简单的将约束划分为相等约束equality/bilateral和不相等约束inequality/unilateral。根据约束限制内容的不同，还可以有距离约束、形状约束、体积约束、碰撞约束等等。所以，从某种角度来说，约束可以用来解决物理引擎中的大部分情况，其重要性有些类似于渲染中的shader。

写成公式的大概类似于

或者

我们举一个简单的例子，两个小球之间有一杆子相连，则两个小球之间的距离被限制为杆子的长度。设、分别是两个小球的中心坐标，和是两个小球的半径，细线长度是，则有式子

就是一个简单的约束方程。它可以表示两个小球之间的连接关系，而不用去考虑杆子的拉力。

这是一个位置约束，我们可以定义位置约束的一般形式，设是由一系列点组成的向量，则这些点之间的约束关系可以整合成一个约束

position based dynamic的思想是直接修改坐标位置来满足约束。所以当出现误差时，我们可以设的修改量为，得到方程

对一阶展开(参考泰勒展开)。得到

|  |  |  |
| --- | --- | --- |
|  |  | **Eq. 1** |

是约束方程的梯度，梯度就是由一堆偏导数组成的向量。因为是一个维向量(n个点，每个点是3维坐标)，所以也是一个维向量。导数的模表示变化率，梯度的模也能表示变化率，而且其方向是变化率最大的方向。

PBD的论文中指出，如果点沿着方向变化的话，动量和角动量是守恒的。原文中并没有给出明确的证明，描述也不太清楚。下面给出我自己的粗浅理解，是一个描述一堆点之间位置关系的约束，既然位置关系已经确定，我们就可以在多维空间中(维)构造出明确的形状(论文中叫rigid body modes)。这样描述可能比较抽象，大家可以参考描述梯度的等高线图，那是在2维空间中构造出的形状，扩展到多维空间同理。假设点的质量相同，如果按照梯度方向进行变化的话，相当于对形状进行等比例的缩放或者扩大。如果将这堆点的变化相加的话，结果会是0

如果考虑点的质量的话，则有

根据上面的式子可以推测出形状的重心没有发生变化。重心不变，则整体的动量和角动量也都不会发生变化，相当于没有受到外力作用。其内部的各个点无论如何运动，都会互相抵消掉，维持一个整体的平衡。

根据上面描述，跟同向，给出一个，则有以下关系

所有点一起考虑的话，有

带入**Eq. 1**得到

|  |  |  |
| --- | --- | --- |
|  |  | **Eq. 2** |

回到刚体穿透的问题，针对两个接触点存在约束

则

是带符号穿透距离，则

同理旋转的变化量是

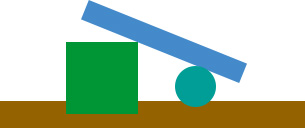
再根据公式

计算出修正后的位置和朝向

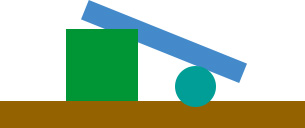
约束得到满足，两个刚体也成功被分开了

如果说场景里面有多个刚体互相之间接触，这样就会同时存在多个约束，比如

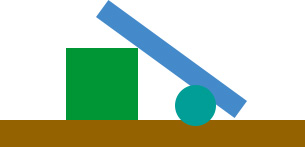
、、，当我们满足了其中一个约束后，往往会破坏另外一个约束，这时候就得赶紧去修复，这样把所有约束修复完一遍之后，并不能确保所有约束都会满足。误差就像病毒一样在各个约束之间蔓延，我们只能再重新修复一遍，拆东墙补西墙，直到误差被消磨殆尽。这样不断的重复迭代修复约束的方法虽然看起来比较笨，但胜在简单高效。目前大多数引擎包括chaos都使用了这种方法。下面的图示介绍了这个迭代过程



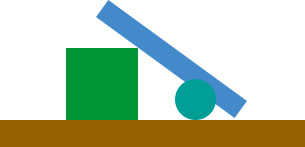
解决绿色盒子和褐色地面穿透



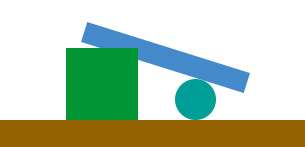
解决绿色盒子和蓝色盒子穿透



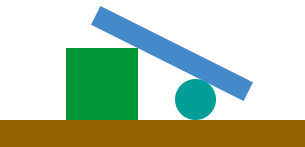
解决青色小球和褐色地面穿透



解决蓝色盒子和青色小球穿透



解决绿色盒子和蓝色盒子穿透



实际上迭代的次数要多得多，图示只是简化说明

以上是对解多个约束迭代法比较直观的解释，下面我们从数学角度来研究这个问题

从公式**Eq. 2**开始，我们假设不再是单个约束，而是由个约束组成的向量。

对应的标量也变成了向量

是梯度向量，对应的多约束版本是矩阵，该矩阵叫做雅可比矩阵，用字母表示

依然是质量矩阵

公式**Eq. 2**就变成了

已知和，求解，这就变成了一个类似求解线性方程组的问题。

有很多方法可以求解该方程组，在数值分析这门课程中，提供了很多解法，比如直接求解法、牛顿迭代法、高斯-赛德尔Gauss-Seidel迭代法等等。具体解法这里不作分析。值得一提的是，前面介绍的拆东墙补西墙的方法其本质就是Gauss-Seidel迭代法，因为每次求得的约束解会直接生效(修改位置和朝向)，代入下一个约束求解中，所以Gauss-Seidel迭代要比牛顿法收敛快一些。但是，从另一方面考虑，因为每次约束求解需要依赖其他约束的结果，所以该方法并不适合并行计算，在GPU上，通常不会采取该方法求解。GPU运行物理引擎是个很好的趋势，大规模的粒子效果，可以释放自然元素水、风、火等纯物理演算的技能效果，想想都让人兴奋。但是，因为物理跟gameplay关系紧密，需要CPU和GPU的频繁交互，这会极大的影响两端的运行效果。所以，目前大部分物理引擎包括chaos都只能在CPU端运行，开启新的物理线程来加快运行效率。