Modeling and Solving Physical Constraints

HTTPS://GAMEDEVELOPMENT.TUTSPLUS.COM/TUTORIALS/MODELLING-AND-SOLVING-PHYSICAL-CONSTRAINTS--GAMEDEV-12578

约束是对物理建模对象的限制 在模拟中。通常，一个对象有着6个自由度，代表着他在模拟世界中移动与旋转的能力。通过以一些方法限制这些自由度，我们可以达到许多有趣且引人注意的效果。

随着现代计算机的CPU变得越来越强力，算力(Computations)可以被用来建模和解决许多有趣的物理情况。约束是一个可用于产生该样情况的广义且数学化支持的方法。我们每个人都可以感谢 Erin Catto 因为他在该主题上的初稿论文: Iterative Dynamics with Temporal Coherence。我将会在本文中引用该文，所以我建议您在继续下文之前至少先看一看，甚至是出于对Erin的尊敬。

// 文章地址为 [https://code.google.com/p/box2d/downloads/detail?name=GDC2005_ErinCatto.zip&can=2&q=]

词汇表

在谈到物理引擎时有一些常用的术语。为了清楚起见，这是一个用于参考的短词汇表 在阅读本文中。:

RigidBody：一个在物理引擎中模拟的对象。该对象假设为全硬的，例如像钻石。大部分情况下，刚体之间互相反弹 滑动，并构成物理世界。

Degree of Freedom 自由度：6个用于表示刚体于世界中的位置与朝向的标量值中的其中一个。三个标量值用于表示位置，(以及另外)三个标量值表示朝向。

Constraint 约束：置于刚体的1+个的自由度上的限制。几乎所有的约束都是成对的，这将意味着其总是一前一后地影响着两个刚体。

Joint 关节：关节是成对的约束，尽管约束的类型不是解互穿(interpenetration)约束。所有其他(非解互穿)类型的约束都称为关节。 // 成对关节约束 (顺便地 非接触约束

Contact Constraint：非为关节的成对的约束，即解互穿约束。 // 接触约束 (非关节约束

Contact Normal：一个用于解决解互穿约束的方向，通常于碰撞检测中确定。

// 注释: interpenetration-constraint 解互穿约束 本文中意为 接触约束 contact-constraint

预求

为了充分运用本文，您需要持有少数一些先决条件。然而，通常读者仍然可以享受多数内容，即使对以下几点有一些基本了解会更好：

- 代数
- 微积分
- 矢量积分
- 中到高等的程序技能 (为了写出程序)

约束的类型

由于约束限制着自由度，因此让我们看看同时使用着6个自由度的刚体：

https://youtu.be/z5FBESRLOt8
[高速位移与旋转着的一个盒子]

上面的刚体使用了(其中)三个自由度以在世界中变换(translate)自己，而后三个自由度则是用于不断地改变着朝向。

现在让我们看看几个不同的示例 - 约束到底是什么。一个最令人熟悉的约束是 防止两个刚体互相穿透/交叠的约束。这种类型的约束只在两个物体互相穿透时工作(有效)，并于随后驱使两个物体分隔。一旦该解互穿约束有效(生效)，将很容易看出来 刚体的自由度被以那样的方法被限制和被影响并产生出一个有趣的结果 (该结果为两个物体可形成互相碰撞)：

https://youtu.be/_Ziuj50JTD0
[两个物体发生碰撞 并于随后弹走]

约束的Hello World将无疑是 距离约束 - 位于两个刚体上的两个点 互相以一个精确的距离形成约束。可以想象一个无质量的杆子将两个点连接在一起，且该杆子不会被拉长或压缩：

https://youtu.be/2zCta1mXuF8
[一个杆子分别地连接着两个物体上的某个点 在这种情况下两个物体的运动]

许多类型的约束有着各种各样的有趣行为，包括：摩擦，prismatic, revolute, weld, angle, 及更多。

门外约束方程

通常(式)，约束为一个等于某值(通常为0)的标量方程。

Equation1

$$C(l_1, a_1, l_2, a_2) = 0$$

在 eq1 中的 l 和 a 项是我自己的标记：l 为线性(linear)，而 a 为角性(angular)。下标 1 和 2 表示两个在该约束中的对象。如你所见，线性和角性输入向了一个约束方程，且每个(输入)都必须为一个标量值。

让我们回一步看向(上面提到的)距离约束。距离约束想 驱使两个物体上的两个锚点之间的距离等于某个标量值：

Equation2

$$C(l_1, a_1, l_2, a_2) = \frac{1}{2}[(P_2 - P_1)^2 - L^2] = 0$$

// 注释 1/2概念上无意义 无平方根为简化运算,效果一样 锚点为本地坐标0

L 为连接两个物体的杆子的长度；P_1 和 P_2 分别为两个物体的位置。

在当前形式中，该约束为一个位置方程。这样的位置方程是非线性[http://en.wikipedia.org/wiki/Nonlinear_system]的，这会使得求解他变得异常困难。一个求解该方程的方法为 导出位置约束(相对于时间) 并使用速度约束。得出的速度方程是线性的，使得求解变得可能。求解结果可以随后被积分回位置性形式。

通常，速度约束的形式为：

Equation3

$$\dot{C}(l_1, a_1, l_2, a_2) = 0$$

eq3 中 C 上的点指 C 相对于时间的一级导数。在进行物理商讨时，这是常见的表示法。
在导数之中，一个新项 J 通过 链规则[http://en.wikipedia.org/wiki/Chain_rule] 出现了：

Equation4

$$\dot{C}(l_1, a_1, l_2, a_2) = JV = 0$$

C 的时间导数 创建了一个速度矢量 和一个Jacobian[http://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant]。该Jacobian为一个1x6矩阵 其中包含着 与每个自由度相对应的标量值(?系数)。对于一个成对的约束，Jacobian通常将包含12个元素(??)(以足够容下物体A与B的 l 和 a 项)。

约束系统可以形成关节(Joint)。关节可以包含许多以各种方式限制自由度的约束。在这种情况下，Jacobian将会是一个矩阵 其行数为系统中有效(active)约束数。
// 这里的系统可能是指 “一系列”

Jacobian是“用手”离线导出的。一旦Jacobian被取得，去算和使用Jacobian的程序代码则可于随后被创建出来。如你从 eq4 中所见，速度 V 被从笛卡尔空间变换至了约束空间。这是重要的，因为在约束空间 原点是已知的。实际上，任何目标都可为已知。这意味着 可以导出任何约束 以产生一个可以将力从笛卡尔空间变换至约束空间的Jacobian。

在约束空间中，给出一个目标标量，其方程可以朝向目标移动或远离目标移动。
解决方案可以被轻松的在约束空间中取得 以将刚体当前的状态 移向目标状态。
求解结果可以随后被从约束空间变换回笛卡尔空间。像这样：

Equation5

$$F = \lambda J^T$$

F 是一个于笛卡尔空间中的力，J^T 是Jacobian的逆(转置)。lambda 为一个标量乘数。

// 注释：对于正交矩阵，其逆等效于其转置矩阵

设想Jacobian为一个速度向量，其每一行本身为一个向量(2D中两个标量值，3D中三个标量值)：

Equation6

$$J = \begin{bmatrix} l_1 \\ a_1 \\ l_2 \\ a_2 \end{bmatrix}$$

// 或许实际上这个J是实际J的转置 。。？ 只是原作者忘记或懒得说明这一点细节。。

用J乘以V 数学上将涉及到矩阵乘法。然而 大多数元素为零，这就是为什么我们将Jacobian视为向量。这将允许我们 定义用于计算 JV 的运算操作，如eq4中所示。

Equation7

$$JV = \begin{bmatrix} l_1 & a_1 & l_2 & a_2 \end{bmatrix} \begin{bmatrix} v_1 \\ \omega_1 \\ v_2 \\ \omega_2 \end{bmatrix}$$

这里，v 代表线性速度，ω (omega) 代表角速度。为了达到比全矩阵乘法更有效率的计算 eq7可以被记为几个点积和乘法：

Equation8

$$JV = l_1 \cdot v_1 + a_1 \cdot \omega_1 + l_2 \cdot v_2 + a_2 \cdot \omega_2$$

Jacobian可被设想为一个约束空间中的方向向量。该方向始终指向 需要完成最少工作的目标方向。因为该“方向”Jacobian是离线导出的，所以所有(剩下的)待解决的问题是 为了维持约束 所施加的力的幅度(magnitude)。该幅度称为 lambda。lambda又为 Lagrange Multiplier [http://en.wikipedia.org/wiki/Lagrange_multiplier] 拉格朗日乘数。我本人没有系统的学习过 Lagrangian Mechanics 拉格朗日力学，然而 若只是为了实现约束，对于 Lagrangian Mechanics 的学习是不必要的。(我可以保证这一点！)。可以使用约束求解器解算出lambda(稍后将对此详细介绍)。

求解雅可比矩阵(S)

在 Erin Catto 的论文中[本文开头的那一份]，有一个对于 手工导出Jacobians 的简单的大纲。
步骤为：

1. 从约束方程 C 开始
2. 计算时间导数 $\dot{C}$
3. 隔离出所有速度项
4. 通过审查定义 J

唯一困难的部分是计算导数，(然而)这可以通过实践来实现。通常，手导约束是比较难的，但随着时间的推移会变的更加容易。(而不是因为时间微分…?)

让我们导出一个有效的Jacobian 用于求解距离约束。我们可以用eq2从 Step1 开始。
(然而)这是一些 Step2 的细节：

Equation9

$$\dot{C} = (P_2 - P_1)(\dot{P}_2 - \dot{P}_1)$$

Equation10

$$\dot{C} = (P_2 - P_1)((v_2 + \omega_2 \times r_2) - (v_1 + \omega_1 \times r_1))$$

// 注释 w x r = v; r x v = w;
r_1 和 r_2 是分别在物体1和2上 从质心至锚点的向量。

下一步是隔离速度项。为此，我们将使用 标量三重积 定义：
// TripleProduct a · (b x c) == b · (c x a) == c · (a x b) https://en.wikipedia.org/wiki/Triple_product

Equation11

$$(P_2 - P_1) = d$$

Equation12

$$\dot{C} = (d \cdot v_2 + d \cdot \omega_2 \times r_2) - (d \cdot v_1 + d \cdot \omega_1 \times r_1)$$

Equation13

$$\dot{C} = (d \cdot v_2 + \omega_2 \cdot r_2 \times d) - (d \cdot v_1 + \omega_1 \cdot r_1 \times d)$$

最后一步是通过审查 定义Jacobian。为此，所有的速度项v和w的所有系数 都将被用作Jacobian的元素。因此：

Equation14

$$J = \begin{bmatrix} -d & -r_1 \times d & d & r_2 \times d \end{bmatrix}$$

更多雅可比矩阵

接触约束Contact-Constraint (解互穿约束Interpenetration-Constraint)
其中 n 为接触法向 (Contact-Normal)：

Equation15

$$J = [-n \quad -r_1 \times n \quad n \quad r_2 \times n]$$

摩擦力约束 (仅在互穿/交叠期间有效)
其中 t 为摩擦轴 (2D为一个轴，3D则有两个)：

(16)

$$J = [-t \quad -r_1 \times t \quad t \quad r_2 \times t]$$

求解约束

现在我们对约束有了一个(基本的)了解，我们可以看看如何求解他们。如之前所述，一旦Jacobian已被手工导出，我们(将)只需要去求解 lambda。隔离地解决单个约束是简单的，但同时解决多个约束则是难的，并且非常低效(计算上)。这就带来了一个问题，因为游戏和模拟可能会希望同时使用许多约束。

同时求解所有约束(全局求解)的另一种方法 是迭代(性的)求解约束。通过求解近似解，并将先前的解输入至方程中，我们则可 收敛求解(过程)。

一种这样的迭代性求解器为 Sequential Impulses 顺序脉冲，由Erin Catto所称。
顺序脉冲 与Projected Gauss seidel 投影高斯·赛德尔 [http://en.wikipedia.org/wiki/Gauss%E2%80%93Seidel_method] 非常相似。这个想法是一次一次地求解所有约束，多次地。该解决过程将会使其他约束不有效(不同时有效 因为是离散的分别解决)，但经过多次迭代后 每个约束都会收敛，并导致全局求解已达成！这很佳！迭代求解器速度(也)很快。

一旦完成了一项求解，为了实施约束 冲量即可被应用于 约束上的两个物体。

为求解单个约束，我们可以使用下面的方程：

Equation17

$$\lambda = \frac{-(JV + b)}{JM^{-1}J^T}$$

M^-1 为约束的质量；b 为一个微量(稍后说到)。

这是一个矩阵，其包含着 约束上的两个刚体 的质量逆和惯性矩逆。下面为约束的质量(逆)。注意 m^-1 为物体质量的逆，而 I^-1 为物体惯性矩的逆：

Equation18

$$M^{-1} = \begin{bmatrix} m_1^{-1} & 0 & 0 & 0 \\ 0 & I_1^{-1} & 0 & 0 \\ 0 & 0 & m_2^{-1} & 0 \\ 0 & 0 & 0 & I_2^{-1} \end{bmatrix}$$

即使 M^-1 理论上是一个矩阵，但请别真的让他成为那样 (大部分都是零！)。相比之下，使用聪明些的办法 做且仅做你实际上要做的运算。

JM^-1J^T 又成为 约束质量 Constraint Mass。该项只计算一次[?] 并在随后用于解算lambda。我们像这样用此样一个系列来计算他：

Equation19

$$JM^{-1}J^T = (l_1 \cdot l_1) * m_1^{-1} + (l_2 \cdot l_2) * m_2^{-1} + a_1 * (I_1^{-1} a_1) + a_2 * (I_2^{-1} a_2)$$

请注意为了计算eq17 您必须反转/逆掉eq19。[?]

以上信息是求解约束所需要的全部信息！为了实施约束 (我们)可以求出 一个于笛卡尔空间中的力 并于随后将其用于更新对象/物体的速度。请回看eq5。

Equation20

$$F = \lambda J^T$$

$$V_{final} = V_{initial} + m^{-1} * F$$

$$\therefore \begin{bmatrix} v_1 \\ \omega_1 \\ v_2 \\ \omega_2 \end{bmatrix} += \begin{bmatrix} m_1^{-1} & 0 & 0 & 0 \\ 0 & I_1^{-1} & 0 & 0 \\ 0 & 0 & m_2^{-1} & 0 \\ 0 & 0 & 0 & I_2^{-1} \end{bmatrix} \begin{bmatrix} \lambda * l_1 \\ \lambda * a_1 \\ \lambda * l_2 \\ \lambda * a_2 \end{bmatrix}$$

约束漂移

由于非线形位置方程的线形化，一些信息被丢失了。这将导致求解结果不会完全满足原始的位置方程，但会满足速度方程。该误差称为约束漂移(constraint drift)。(我们)可以认为该误差为 tangent line approximation 切线近似 [http://en.wikipedia.org/wiki/Linear_approximation] 的结果。

有几种不同的方法可以解决此误差，所有这些方法都可以近似出误差值 并进行某种形式的校正。(其中)最简单的(一种方法)为 Baumgarte。

Baumgarte 是约束空间中的一小份微量能量附加，并在前面的等式中记为 b 项。为了加上该偏差(bias)，这是eq4的一个修改版本：

Equation21

$$\dot{C} = JV + b = 0$$

为了计算Baumgarte项并将其作为一个偏差，我们必须审查原始的约束方程 并确定一种合适的方法以计算该偏差。Baumgarte 的式子为：

Equation22

$$JV = -\beta C$$

beta (Baumgarte项) 是一个可调的，无单位的，与仿真相关的因子。他通常在 0.1 至 0.3 之间。

为了计算偏差(bias)项，让我们看看 解互穿约束 等式eq15 在相对于时间导出前的形式。其中 n 为碰撞法向：

Equation23

$$C = [-x_1 \quad -r_1 \quad x_2 \quad r_2] \cdot \vec{n}$$

上式表示，C 的标量误差 是两个刚体之间的互穿深度。

结论

感谢 Erin Catto 及其关于约束求解的论文，使得我们有了一种方法 在速度项上 来解决位置约束。约束可引起许多有趣的行为，希望本文对许多人有用。与往常一样，请感觉着自由地发问或说些何子。

请参阅 Box2D Lite 以获取有关解决各种类型的2D约束的资源，及更深入地了解此处未涵盖的许多实现细节。

Randy Gaul

Hello! I am a Computer Science student studying at DigiPen IT. You can find more content by myself at RandyGaul.net!