# 第一章 导言

## 1.1 关于

Box2D是一个用于游戏的2D刚体仿真库。程序员可以在他们的游戏里使用它，它可以使物体的运动更加可信，让世界看起来更具交互性。从游戏引擎的视角来看，物理引擎就是一个程序性动画（procedural animation）的系统，不是由动画师去移动你的物体，你可以让牛顿来做导演。

Box2D 是用可移植的C++来写成的。引擎中定义的大部分类型都有b2前缀，希望这能消除它和你游戏引擎之间的命名冲突。

## 1.2 必备条件

在此，我假定你已经熟悉了基本的物理学概念，例如质量，力，力矩和冲量。如果没有，请先考虑读一下Chris Hecker和David Baraff（google这些名字）的那些教程，你不需要了解得非常细致，但他们可以使你很好地了解一些基本概念，以便你使用Box2D。Wikipedia也是一个极好的物理和数学知识的获取源，在某些方面它可能比google更有用，因为它的内容经过了精心的整理。

这不是必要的，但如果你好奇Box2D内部是如何工作的，你可以看一下box2d.org中download部分的教程文档。

因为Box2D是使用C++写成的，所以你应该具备C++ 程序设计的经验，Box2D不应该成为你的第一个C++程序项目。你应该已经能熟练地编译，链接和调试了。

## 1.3 关于本手册

本手册介绍了Box2D API的主要部分，但并没有面面俱到。我们鼓励你通过研究包含在Box2D中的测试程序学到更多的东西。Box2D代码注释的格式是[Doxygen](http://baike.baidu.com/link?url=5zvQxFjclLyp6ZVclHWZe5xOgrjYqwqekvGkfkuIxjeEcyjjTMmol5HHoPwHYSdkP-FHavCiMmuLF6kdSnTERK)，你可以很容易地创建一个超文本的API文档。

只有在新版本发布时本手册才会更新，因此源代码中的版本可能会过时。

## 1.4 反馈和错误提交

如果你有问题或反馈，请在论坛留言，论坛也是进行交流讨论的好地方。

Box2D使用Google代码项目进行留言管理，它可以保证你的留言不会丢失在论坛某处。

你可以在<http://code.google.com/p/box2d/>提交程序错误和功能请求。

如果你能提供足够的细节，则能更好地保证程序的修复，最好能提供测试程序重现错误，在本文档的后面你可以了解什么是测试程序。

## 1.5 核心概念

Box2D 中有一些基本的对象，这里我们先做一个简要的定义，在随后的文档里会有更详细的描述。

### 形状（shape）

形状是一个2D几何物体，例如一个圆形或多边形。

### 刚体（rigid body）

一个十分坚硬的物质，它上面的任何两点之间的距离都是完全不变的。它们就像钻石那样坚硬。在后面的讨论中，我们用物体（body）来代替刚体。

### 定制器（fixture）

一个定制器可以将一个形状绑定到一个物体上，还可以添加诸如密度、摩擦因数和恢复系数之类的本身属性。一个定制器将形状加入到碰撞系统中（粗略检测过程broad-phase），这样就可以和其他形状发生碰撞。

### 约束（constraint）

一个约束（constraint）就是消除物体自由度的物理连接。一个2D物体有3个自由度（2个平移坐标和一个旋转坐标）。如果我们把一个物体钉在墙上（像摆锤那样），那我们就把它约束到了墙上。这样，此物体就只能绕着这个钉子旋转，所以这个约束消除了它2个自由度。

### 接触约束（contact constraint）

一个防止刚体穿透，以及用于模拟摩擦（friction）和恢复（restitution）的特殊约束。你无需手动创建一个接触约束，它们会自动被Box2D创建。

### 关节（joint）

它是一种用于把两个或多个物体固定到一起的约束。Box2D支持的关节类型有：旋转，棱柱，距离等等。某些关节可以支持限制（limits）和马达（motors）。

### 关节限制（joint limit）

一个关节限制（joint limit）限定了一个关节的运动范围。例如人类的胳膊肘只能做某一范围角度的运动。

### 关节马达（joint motor）

一个关节马达（joint motor）能依照关节的自由度来驱动所连接的物体。例如，你可以使用一个马达来驱动一个肘的旋转。

### 世界（world）

一个物理世界就是物体，定制器和约束相互作用的集合。Box2D支持创建多个世界，但这通常是不必要的。

### 解析器（solver）

物理世界包含一个解析器，它用来推进时间并解析接触和关节约束。Box2D解析器是一个高性能的迭代运算程序，如果有N个约束，则它会依次进行N次求解运算。

### 连续碰撞检测（continuous collision）

解析器使用离散的时间步进对物体进行处理，如果没有这种机制则会导致隧道（tunneling）现象的发生。



Box2D包含专门的算法解决隧道现象。首先，碰撞算法可以对两个物体的运动进行插值运算，由此找到第一次发生碰撞的时间（first time of impact，TOI）。然后，有一个次步骤解析器用来将物体移动到第一次碰撞的位置并进行碰撞处理。

### 模块

Box2D由三个模块组成：Common，Collision和Dynamics。Common模块的代码用于分配（allocation），数学和设置。Collision模块的代码定义形状，粗略检测（a broad-phase）和碰撞方法/查询。Dynamics模块提供了模拟世界，物体，定制器和关节。



### 单位

Box2D使用浮点数，所以必须使用一些公差来保证它正常工作。这些公差已经被调整得适合米-千克-秒（MKS）单位。尤其是，Box2D被调整得能良好地处理0.1到10米之间的移动物体。这意味着从罐头盒到公共汽车大小的形状都能良好地工作，静止形状大到50米也不会出现任何问题。

作为一个2D物理引擎，如果能使用像素作为单位是很诱人的。很不幸，那将导致不良模拟，也可能会造成古怪的行为。一个200像素长的物体在Box2D看来就有45层建筑那么大。想象一下使用一个被调整好模拟玩偶和木桶的引擎去模拟高楼大厦的运动。那并不有趣。

**注意**：Box2D已被调整至MKS单位。移动物体的尺寸大约应该保持在0.1到10米之间。你可能需要一些缩放系统来渲染你的场景和物体。Box2D中的例子是使用OpenGL的视口来变换的。**千万别使用像素**。

可以将Box2D物体想象为一个移动的公告板（billboard），billboard的移动是以米为单位的，但你可以通过一个缩放因子转换为像素坐标，然后就可以使用像素坐标放置sprite了。

Box2D使用弧度制。物体的旋转以弧度方式存储，可以无限增加。若角度变得太大时，最好考虑重置物体的角度（使用b2Body::SetAngle）。

### 工厂和定义（Factories and Definitions）

快速内存管理在Box2D API的设计中担当了一个中心角色。所以当你创建一个b2Body或一个b2Joint的时候，你需要调用b2World的工厂函数。

以下是创建函数：

b2Body\* b2World::CreateBody(const b2BodyDef\* def)

b2Joint\* b2World::CreateJoint(const b2JointDef\* def)

以下是对应的销毁函数：

void b2World::DestroyBody(b2Body\* body)

void b2World::DestroyJoint(b2Joint\* joint)

当你创建一个物体或关节的时候，你需要提供一个定义（definition）。这些定义包含了创建物体或关节的所有相关信息。通过这样的方法，我们就能预防构造错误，使函数参数的数量较少，提供有意义的默认参数，并减少访问器（accessor）的数量。

因为定制器（形状）必须有父物体，所以b2Body上有创建和销毁定制器（形状）的工厂方法：

b2Fixture\* b2Body:: CreateFixture(const b2FixtureDef\* def)

void b2Body::DestroyFixture(b2Fixture\* fixture)

还有一个快捷方式根据形状和密度直接创建定制器。

b2Fixture\* b2Body::CreateFixture(const b2Shape\* shape, float32 density)

工厂并不保留到定义的引用，所以你可以在栈上创建定义，并在临时资源中保存它们。