# 第二章 Hello Box2D

在Box2D压缩包中包含一个Hello World项目，这个程序创建了一个大的地面盒和一个小的动态盒。代码中没有包含任何图形，你只能看到在一个控制台中输出的文本。

这个示例可以学习如何运行Box2D。

## 2.1 创建一个世界

每个Box2D程序都将从一个世界对象b2World的创建开始，这个对象是一个管理内存，对象和模拟的中心。你可以在堆、栈或数据区域中创建物理世界。

创建一个Box2D世界很简单。首先定义一个重力加速度矢量。

b2Vec2 gravity(0.0f, -10.0f);

现在开始创建世界对象。

b2World world(gravity);

有了物理世界，下面在里面添加一些东西。

## 2.2 创建一个地面盒

物体通常由以下步骤来创建：

1. 使用位置（position），阻尼（damping）等定义一个物体
2. 使用世界对象创建物体
3. 使用形状，摩擦，密度等定义定制器
4. 在物体上创建定制器

第1步，我们创建地面。要创建它我们需要一个物体定义（body definition），通过物体定义我们来指定地面体的初始位置。

b2BodyDef groundBodyDef;

groundBodyDef.position.Set(0.0f, -10.0f);

第2步，将物体定义传给世界对象来创建地面。世界对象并不保存到物体定义的引用。地面是作为静态物体（static body）创建的，静态物体之间并没有碰撞，它们是固定的。

b2Body\* groundBody = world.CreateBody(&groundBodyDef);

第3步，我们创建一个地面的多边形定义。我们使用SetAsBox快捷方式把地面多边形设置为一个盒子（矩形）形状，盒子的中点就位于父物体的原点上。

b2PolygonShape groundBox;

groundBox.SetAsBox(50.0f, 10.0f);

SetAsBox函数的参数为半宽度和半高度，这样的话，地面盒就是100个单位宽（x轴）以及20个单位高（y轴）。Box2D已被调整使用米，千克和秒来作单位，所以你可以用米来考虑长度。通常在使用真实世界物体的大小尺寸的情况下Box2D工作得最好。例如，木桶通常高1米。由于浮点数算法的限制，使用Box2D模拟冰川或灰尘的运动并不是一个好主意。

在第4步中，我们在地面体上创建形状定制器，以完成地面体。这里有个快捷方式，因为我们无需指定定制器的默认属性，所以可以直接将形状传递给物体，而无需首先创建一个定制器的定义。后面的内容会介绍如何使用定制器定义设置属性。第2个参数是以kg/m2为单位的形状密度。静止物体的质量定义为0，所以无需使用密度。

groundBody->CreateFixture(&groundBox, 0.0f);

Box2D并不保存到形状的引用。它把数据拷贝到b2Body结构中。

**注意：**每个定制器都必须有一个父物体，即使它是静态的。然而你可以把所有静态定制器都依附于单个静态物体之上。

当使用定制器将形状附加到物体上时，形状的坐标会变为物体的本地坐标。这样，当物体移动时，形状也会跟着移动。定制器的世界变换继承自父物体，它本身并没有独立于物体的变换，所以不要在物体周围移动形状，也不支持移动或改变连接到物体上的形状，道理很简单：具有可变形状的物体不是刚体，而Box2D是一个刚体模拟引擎。Box2D中很多假设都是基于刚体模型的，这个规则不容破坏。

## 2.3 创建一个动态物体

现在我们已经有了一个地面体，我们可以使用同样的方法来创建一个动态物体。除了尺寸之外的主要区别是——我们必须为动态物体设置质量性质。

首先我们用CreateBody创建物体。物体默认是静态的，所以我们必须在构造时设置b2BodyType，这样才能变为动态物体。

b2BodyDef bodyDef;

bodyDef.type = b2\_dynamicBody;

bodyDef.position.Set(0.0f, 4.0f);

b2Body\* body = world.CreateBody(&bodyDef);

**注意**：如果你需要让物体在力的作用下移动，就必须将物体类型设置为b2\_dynamicBody。

然后创建多边形形状，使用定制器定义将它进行连接。首先创建矩形形状：

b2PolygonShape dynamicBox;

dynamicBox.SetAsBox(1.0f, 1.0f);

然后使用这个形状创建定制器定义。注意密度设置为1，而默认值为0。形状的摩擦因数设置为0.3。

b2FixtureDef fixtureDef;

fixtureDef.shape = &dynamicBox;

fixtureDef.density = 1.0f;

fixtureDef.friction = 0.3f;

**注意**：一个动态物体至少得有一个密度不为零的定制器，否则物体的行为会变得很怪异。

有了定制器定义才能创建定制器。这一步会自动更新物体的质量，你可以在一个物体上添加任意多个定制器，每个定制器都会对总质量作出贡献。

body->CreateFixture(&fixtureDef);

这就是初始化过程。现在我们已经准备好开始模拟了。

## 2.4 模拟（Box2D的）世界

我们已经初始化好了地面盒和一个动态盒。现在是让牛顿接手的时刻了。我们只有少数几个问题需要考虑。

Box2D中有一些数学代码构成的积分器（integrator），积分器在离散的时间点上模拟物理方程，它将与游戏动画循环一同运行。所以我们需要为Box2D选取一个时间步进，通常来说游戏物理引擎需要至少60Hz的速度，也就是1/60秒的时间步进。你可以使用更大的时间步进，但是你必须更加小心地为你的世界调整定义。我们也不喜欢时间步进变化得太大，一个变化的时间步进会导致变化的结果，会让调试变得困难，所以不要把时间步进关联到帧频（除非你真的必须这样做）。下面就是时间步进的设定：

float32 timeStep = 1.0f / 60.0f;

除了积分器之外，Box2D中还有约束解析器（constraint solver）。约束解析器用于解决模拟中的所有约束，一次一个。单个的约束会被完美的求解，然而当我们求解一个约束的时候，我们就会稍微耽误另一个。要得到良好的解，我们需要迭代所有约束多次。

在约束解析器中包含两个过程：速度求解过程和位置求解过程。在速度求解过程中，解析器计算冲量用于物体的正确运动。在位置求解过程中，解析器会调整物体的位置避免重叠和关节脱落。每个过程都有自己的迭代数量。而且，在位置求解过程中，如果修正很小，这个过程会提早结束迭代。

Box2D建议的速度求解过程的迭代次数是8次，而位置求解过程为3次。你可以按自己的喜好去调整这个数，但要记得它是速度与精度之间的平衡。更少的迭代会增加性能并降低精度，同样地，更多的迭代会减少性能但提高模拟质量。在简单的例子中，我们无需很多迭代，因此选择下列迭代次数：

int32 velocityIterations = 6;

int32 positionIterations = 2;

**注意**时间步进和迭代数是完全无关的。一个迭代并不是一个子步骤。一次迭代就是在时间步进之中的单次遍历所有约束，你可以在单个时间步内多次遍历约束。

现在我们可以开始模拟循环了，在游戏中模拟循环应该并入游戏循环。每次循环你都应该调用b2World::Step，通常调用一次就够了，这取决于帧频以及物理时间步进。

这个Hello World程序设计得非常简单，所以它没有图形输出。代码会打印出动态物体的位置以及旋转角度。下面就是模拟1秒钟内60个时间步进的循环：

for (int32 i = 0; i < 60; ++i)

{

world.Step(timeStep, velocityIterations, positionIterations);

b2Vec2 position = body->GetPosition();

float32 angle = body->GetAngle();

printf("%4.2f %4.2f %4.2f\n", position.x, position.y, angle);

}

输出文本表示盒子下落并最终静止在地面上，如下文所示：

0.00 4.00 0.00

0.00 3.99 0.00

0.00 3.98 0.00

...

0.00 1.25 0.00

0.00 1.13 0.00

0.00 1.01 0.00

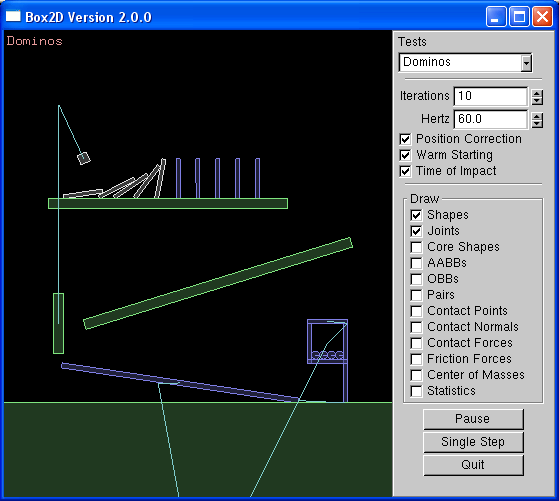
## 2.5 清除

当一个世界对象超出它的作用域，或通过指针将其delete时，所有物体、定制器和关节的内存都会被释放。这样做能提高性能。然而，你应该将物体，定制器或关节的指针都清零，因为它们已经无效了。

## 2.6 关于Testbed

一旦你理解了HelloWorld例子，你应该开始看Box2D的testbed了。testbed是一个单元测试框架以及演示环境，这是一些它的特点：

* 可移动和缩放的摄像机
* 鼠标拣选动态物体的形状
* 可扩展的测试集
* 通过图形界面选择测试，调整参数，以及调试绘图选项
* 暂停和单步模拟
* 文字渲染



在testbed中有许多Box2D的测试用例，以及框架本身的实例。我鼓励你通过研究和修改它来学习Box2D。

**注意**：testbed是使用[freeglut](http://en.wikipedia.org/wiki/Freeglut)和[GLUI](http://en.wikipedia.org/wiki/GLUI)写成的，testbed本身并不是Box2D库的一部分。Box2D本身对于渲染是无知的，就像HelloWorld例子一样，使用Box2D并不一定需要渲染器。