**碰撞检测 上**

回想上一篇文章中我们拿方盒下落的粒子介绍了整个刚体碰撞的过程，在盒子跟地面接触的那一帧，不可避免的会发生重叠(Overlap)，为了处理这种情况，我们需要知道重叠的接触点，重叠距离才能把盒子抬上来。那么该如何知道是否发生重叠，又是怎么通过计算知道重叠相关的数据的呢？这篇文章将会给出答案。

在介绍算法之前，我们先来看下物理引擎都有哪些基本的几何类型。

在chaos中， 比较关键的几种体类型是：

球体Sphere、方盒Box、平面Plane、胶囊体Capsule、凸包Convex、三角网格TriangleMesh、高度场HeightField。这些类型基本可以囊括游戏中需要的各种形体。

针对这些形体之间配对组合，计算碰撞的算法也不尽相同。通过排列组合，我们可以知道需要21种算法。别被21这个数字给唬住了，大部分的算法其实都差不多，我们只需要了解几个核心算法就足够了。

复杂是由简单演化来的，正如几何中的点构成线，线构成面；一维变二维，二维变三维一样。碰撞算法也可以从点、线、面出发，计算出体相关的量。

先从最简单的开始

算法一：

计算空间中两点间的最小距离

我们设A，B两个点，则距离是

用向量的方法表示就是

通过这个算法，我们就可以计算球体A跟球体B之间的距离。在空间中，球体就是有一定厚度的点，所以我们计算出两个球心的距离，然后减去两个球的半径，就能得出两个球体间的距离，如果该距离小于0，则说明两个球体发生的重叠，重叠深度Phi就是该距离，重叠法向量Normal是两球心的连线*Normal = A - B*，重叠位置Location默认为球体A表面的接触点（因为是A跟B碰撞，反之亦然）。*重叠位置 = 球心坐标 – 球半径 \* 重叠法向量;*

21个算法被我们轻松减去了一个，是不是很简单？

算法二：

计算空间中点到平面的最小距离

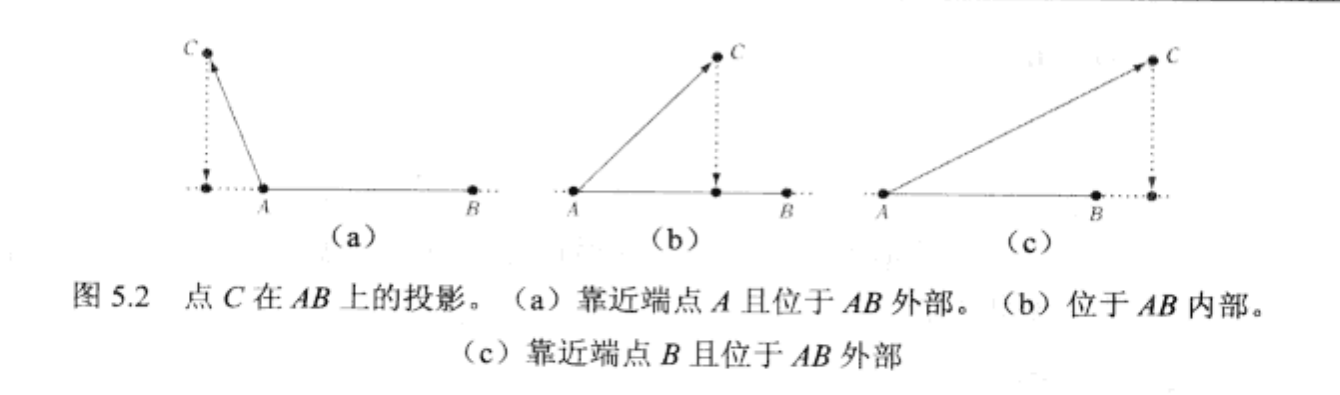
设 空间中的点是P，平面的单位法向量**n**，平面上某一个点是Q

则有 (点乘)

通过这个算法，我们可以计算球体跟平面或者平面跟球体之间的距离，还是前面的方法，空间中球体可以看作点。计算出的距离L减去球体的半径，就能得出球体跟平面的距离，如果该距离小于0，则发生重叠，重叠深度Phi是该距离，重叠法向量是平面的法向量，重叠位置是球体表面的接触点。*重叠位置 = 球心坐标 – 球半径 \* 重叠法向量*

算法三：

计算空间中点到线段的距离（注意这里是线段而不是直线）



设空间中的点C，线段AB，则有三种情况需要考虑，如图中所示，我们将空间按照线段AB划分成3个区域，用三种颜色标识，如果C在线段AB的左侧，则A就是离C最近的点，距离就是线段AC的长度，如果在AB内部，则距离是C到直线AB的距离，如果在AB的右侧，则B是离C最近的点，距离是BC的长度。

判断C点在左侧可以通过

C点在右侧可以通过

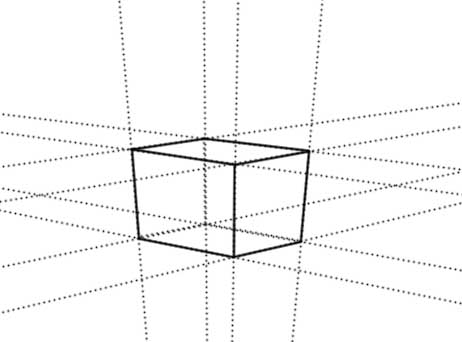
其他情况则是在AB内部

这个算法可以用来计算球体跟胶囊体之间的距离，是不是感觉很奇怪？胶囊体怎么会跟这个算法扯上关系的。我们可以拿球体作类比，球体是有厚度的点，那胶囊体就是有厚度的线段，胶囊体表面上的点到胶囊体内线段的距离相同。有了这个概念，就不难计算球体和胶囊体的碰撞信息了。点到线段的距离减去球的半径跟胶囊体的半径，结果如果小于0，就会发生重叠，其他重叠信息跟球体和球体的计算大致相同，这里不再赘述。

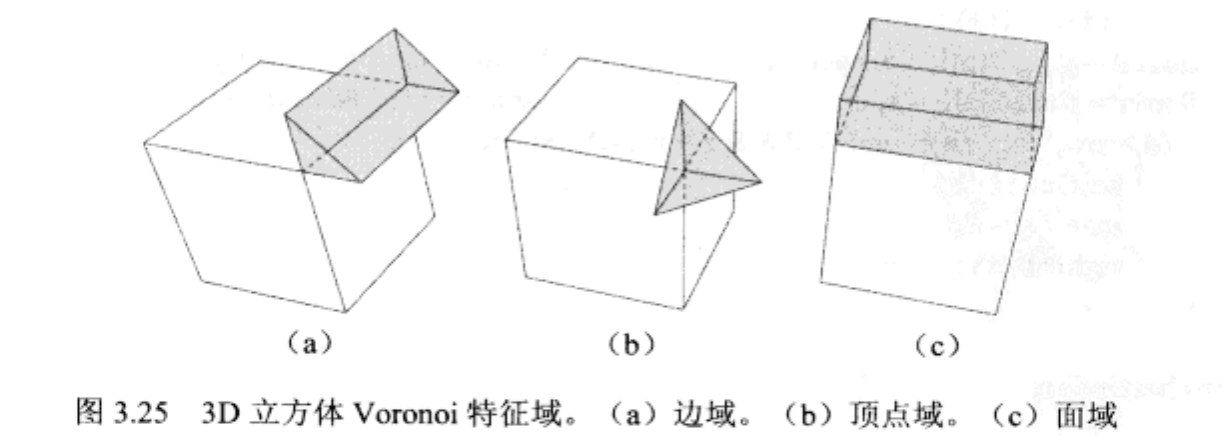
算法四：

计算空间中点到方盒的最小距离

我们按照图中所示的方式将空间按照方盒划分



空间将会划分成21份，简单记忆的话，8条边对应8个区域(对应下图中的a)，6个顶点分别对应6个区域(对应下图中的b)，6个面分别对应6个区域(对应下图中的c)，最后一个区域是在方盒内部。



如果空间中的点在6个顶点对应的区域内的话，距离最近的点就是所在区域对应的顶点。如果在6个面对应的区域的话，距离最近的点是该点在对应平面上的投影。如果在8条边对应的区域的话，距离最近的点是该点在对应边上的投影。而如果在方盒内部，需要比较该点到方盒6个面的距离，距离最小的就是最近距离，该点在应面上的投影就是最近点。

判断点所在区域的算法可以看作是算法三的延伸

首先判断点在方盒内区域最简单，剩下的方盒顶点对应的区域可以看作是顶点的3个邻边的单侧区域的交集，边对应的区域看作是两个单侧区域的交集，剩下的就是面对应的区域了。算法比较复杂，这里就不详细展开了。

这个算法可以用来计算球体跟方盒之间的距离。

算法五：

计算空间中线段到线段的最小距离

读到这里，应该能猜到这是用来计算胶囊体跟胶囊体的碰撞检测的。因为该算法比较复杂，而且使用频率比较低，这里仅简单介绍下算法思想

要计算线段和线段的距离需要先计算直线和直线的距离，设线段 P1P2和Q1Q2，计算俩条直线的叉乘并归一化后得到法向量 ，由P1与该法向量形成一个平面，则P1P2在该平面上，Q1Q2与该平面平行。将Q1Q2投影到该平面上，然后与P1P2相交，计算该交点，就是P1P2上的与Q1Q2的最近点。同理，可以计算出Q1Q2上的与P1P2的最近点。两个点的距离就是最小距离。

如果将直线改成线段的话，还需要考虑该最近点是否超出了线段范围。如果超出了就缩回到最近的顶点，然后计算该顶点到另一条直线的距离。如果两条线上最近点都超了，缩回操作需要计算两次。具体算法可以参考《实时碰撞检测算法技术》 ，chaos中的实现在SegmentDistToSegmentSafe中

总结一下，通过上面五个算法，我们可以计算

球体-球体、球体-平面、球体-胶囊体、球体-方盒、胶囊体-胶囊体之间的碰撞。

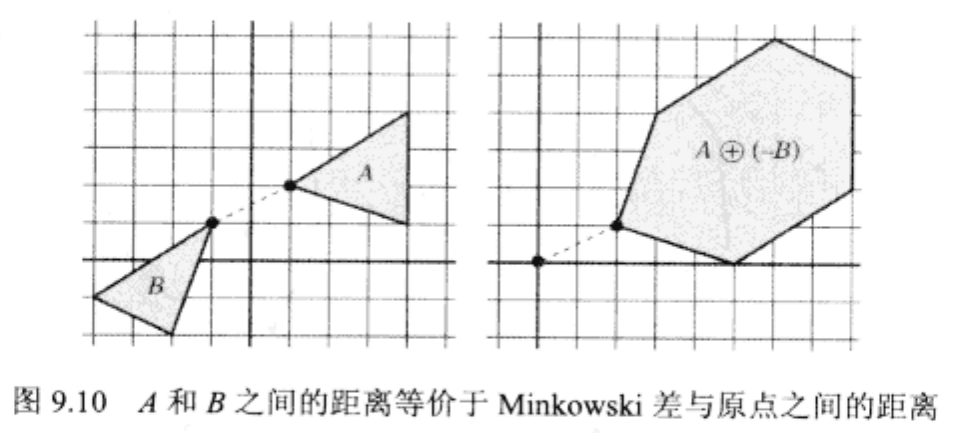
当然，以上算法还可以解决更多问题。比如：

方盒-平面，可以简化为算法二（点和平面最小距离）的问题，分别计算方盒的8个顶点与平面计算最小距离（注意这里的距离是点乘，有正负之分），这些距离的最小值就是方盒和平面的重叠距离。

胶囊体-平面，可以简化为计算两个点到平面的最小距离，并取最小值

除了上面列举的这些组合情况，其他组合相对来说都比较复杂，好在我们有统一的解决方案来处理这些问题。那就是大名鼎鼎的GJK（Gilbert-Johnson-Keerthi）算法，它可以用来计算两个凸体间的最小距离。这里的凸体区别于凸包，可以看作是任意数量的点构成的凸形状，所以，从某种意义上来说，点、线段、三角形、四面体、凸包等都可以算作凸体。因此，该算法几乎可以解决我们上面列举的所有情况。

GJK算法比较关键的一点是需要计算两个形状之间的Minkowski差，Minkowski差可以定义为 A，也就是形状A中的任意一点与形状B中的任意一点的差组合成的形状。该形状有条重要的性质，它与原点的最小距离就是A与B的最小距离，而当原点在该形状内时，A和B发生重叠。

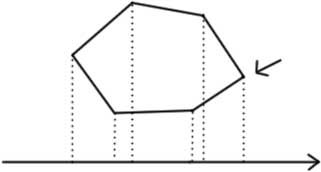


所以问题就变成了计算原点到Minkowski差形状的最小距离。但实际在工程中，我们并不会计算一个新的Minkowski差形状，这会增加不小的内存开销。这里不会介绍原版的GJK算法，想研究可以参考《实时碰撞检测算法技术》，我们这里介绍chaos使用的方法：

1.首先定义一个初始方向，通常是x轴方向

然后分别计算形状A在该方向的支撑点以及形状B在该方向反向上的支撑点

*支撑点就是，找出形状中所有顶点在该方向的投影点，如果该方向作为坐标轴，具有最大值的投影点所对应的顶点就是支撑点。说起来可能比较抽象，画个2D图就明白了。*

**

*在GJK算法中，一般通过计算点的坐标和方向的点乘来比较大小得到支撑点。当顶点比较多时，可以使用爬山法来优化遍历，但会引入顶点间的连接信息，增加内存。Chaos没有使用该方法可能是考虑到了这方面因素。*

2.算出A和B的支撑点后，计算这两个点的差，该差值就是Minkowski差形状中的第一个点。根据前面所介绍的，我们需要计算Minkowski差形状到原点的最小距离，因为现在只有一个点，我们可以通过算法一解决，并得到最近点也就是这个点。

3.然后以该点到原点的矢量方向作为新的方向进行第二轮计算A和B的支撑点。重复之前的操作，得到新的支撑点差值加入Minkowski差形状中，这时候Minkowski差形状中有两个点，构成了线段，我们用算法二计算点到线段的最小距离来计算距离原点的最近点。

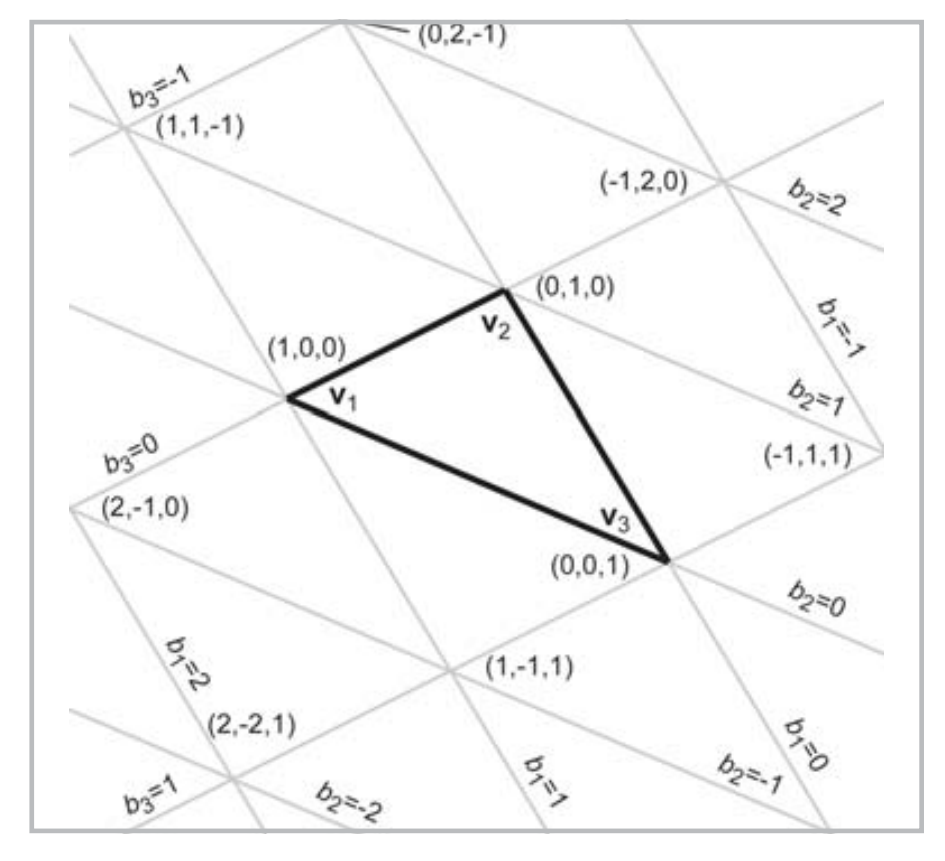
4.依然是以新计算的最近点作为新的方向，重复之前操作，计算出第三个支撑点差值。这时候Minkowski差形状就变成了三角形，如何计算点到三角形的最小距离呢？

这就引出了算法六

计算点到三角形的最小距离

可以像算法四那样把空间划分成不同的区域，三角形的3个顶点和3条边分别对应一个区域，另外还有三角形内部区域。在3个顶点区域的话最近的就是对应的顶点，在3条边区域的话最近点就是在该边上的投影。在三角形内部区域的话，需要分别计算该点到3条边的距离，然后取最小值。

虽然依次计算投影是可行的方法，但还有更高效的方法。那就是利用重心坐标。先将该点投影到三角形所在的平面，然后计算其重心坐标。通过观察下图的重心坐标划分空间的方式，可以发现利用重心坐标可以很容易的确定点在哪个区域中。然后重心坐标还有个重要的性质，其解释为面积比。如下图三角形ABC和点X到平面ABC的投影点D，点D重心坐标的一个值就是三角形DBC和三角形ABC的比值，，如果ABC面积是0.5\*BC\*AE，DBC的面积是0.5\*BC\*DE，则其面积比就是DE/AE，而DE就是点D到边BC的投影，利用这个性质，我们只需要计算出AE，就可以得到DE，有了DE之后，因为D是X到平面ABC的投影，计算X到边BC的距离也将不是难事。

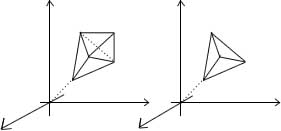


得到点到三角形的最近点之后，我们继续回到GJK算法，将该最近点作为新的方向重复之前的步骤，又会得到新的支撑点差，将其加入Minkowski差形状中，Minkowski差形状就变成了一个四面体。

计算点到四面体的最小距离引出了算法七

该算法可以看做是点到三角形最小距离的延伸，分别计算点到4个三角形的最小距离，然后再取最小值。具体实现细节这里不再展开。

我们经过不断的重复，Minkowski差形状依次变成了点、线段、三角形、四面体。如果再继续下去，形状会变得更加复杂，好在，四面体作为基础的形状细分粒度已经足够我们解决问题。如果顶点数量超过4个，就可以剔除掉距离原点最远的那个点，至于哪个点距离最远，在前面计算最近点的那些算法里就可以顺便找出来。为了方便理解，可以观察下图，会发现最远的那个点是可有可无的。



判断循环结束的方法也很简单，我们每次重复都会得到新的最小距离，如果新得到的距离约等于上一次的，则说明上一次已经找到了最小距离，这时候再退出循环就可以了，但要注意相应的最近点要取上一次的，法向量则取这一次的。

GJK算法只能解决凸体之间的碰撞，然而三角网格TriangleMesh却并不一定是凸体，虽然这样，我们仍然可以使用GJK算法，办法就是分别计算三角网格中每个三角形跟其他形状的最小距离，并找出最小值。这个算法需要遍历所有三角形，但TriangleMesh通常是由3D建模软件制作的，有成百上千个三角形。每个三角形都要用GJK算法，计算复杂度可想而知。为了加速这个检测过程，通常会预先为三角网格生成BVH（层次包围体树 Bounding Volume Hierarchy），这样算法复杂度就从O(n)变成了O(logn)。

类似BVH来加速检测的方法有很多，这类方法一般是由树状结构来划分场景，把待检测对象根据空间位置进行分组管理，每次检测只处理同组中的对象就可以了。在碰撞检测中，这类方法不光能用来加速三角网格等复杂形状，还能用于加速世界场景中刚体与刚体间的碰撞检测。Chaos在这块有比较复杂的实现，我们下一篇文章将会详细介绍。

最后，再总结一些没有涉及到的技术点

1. 上文中提到的形状类型有一种还没有讲到，高度场HeightField
2. 在代码实现中为了避免数值精度的问题，通常会增加待检测形状的厚度，相当于套了一层薄膜，这样能有效避免误检测。
3. 本文介绍的算法是计算具体重叠信息的算法，很多情况并不需要这些信息，比如Raycast和sweep，针对不同的情况，算法也会有些许不同。