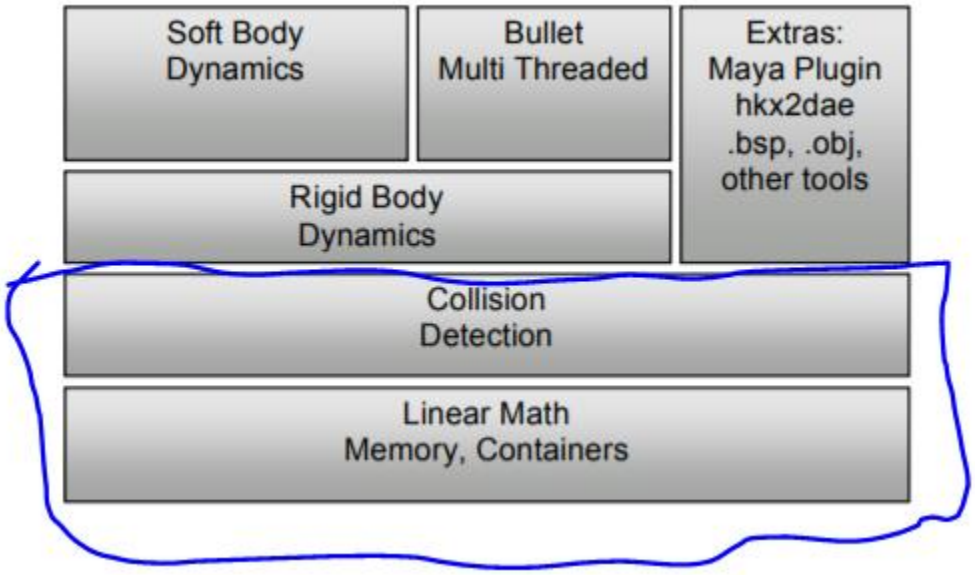
Bullet Collision 说明

本文档为 Bullet Engine 碰撞检测模块的使用说明。

一、Bullet Collision 框架

Bullet Engine 的整体模块架构如下图。其中 Linear Math 模块为提供最基础的向量矩阵等操作的模块，Bullet 中所有模块都会依赖 Linear Math 模块。Collision Detection 为碰撞检测模块，它在仅在 Linear Math 模块之上。



二、碰撞检测的基本步骤

通常，在包含多个可以发生碰撞的物体中，碰撞检测分为两个阶段：Broad Phase，Narrow Phase。

Broad Phase 通过空间划分的方法和包围盒，进行一个粗略的检测，排除大量不可能发生碰撞的物体对，留下一些可能发生碰撞的物体对。

Narrow Phase 再更精确地对可能发生碰撞的物体对进行检测，这个阶段根据可能发生碰撞的物体的形状不同，会需要不同的算法。同时这个阶段也是最耗时间的阶段。

三、主要的数据结构

btCollisionObject : 进行碰撞检测的物体的基类。包含物体的位置变换信息，和碰撞形状。

btCollisionShape：碰撞形状的基类。包含碰撞的形状信息。碰撞形状包括：方形，球形，凸多边形，以及普通的三角 mesh。应当尽量为物体定义简单的 CollisionShape 以加快检测效率。不同的物体可以共享同一个 CollisionShape，例如有多个物体都是球形，那么只需定义一个 btSphereShape，这几个 CollisionObject 共同拥有这个 btSphereShape 对象。尽量利用这一点来优化内存管理。

btGhoseObject：是 btCollisionObject 的一个特例，在局部的快速碰撞检测很有用。

btCollisionWorld：保存了所有的 CollisionObject，并提供了各种碰撞检测请求的接口。

Bullet Collision 的 broad phase 碰撞检测提供了一些加速结构，它们基于 AABB 重叠检测，来高效排除一些不可能发生碰撞的物体对。这些结构包括:

btDbvtBroadphase: 通过一个动态层次包围盒结构进行 broad phase 碰撞检测。

btAxisSweep3 or bt32BitAxisSSweep3: 实现了 3d sweep and prune 算法及其相关结构。

btSimpleBroadphase： 暴力法

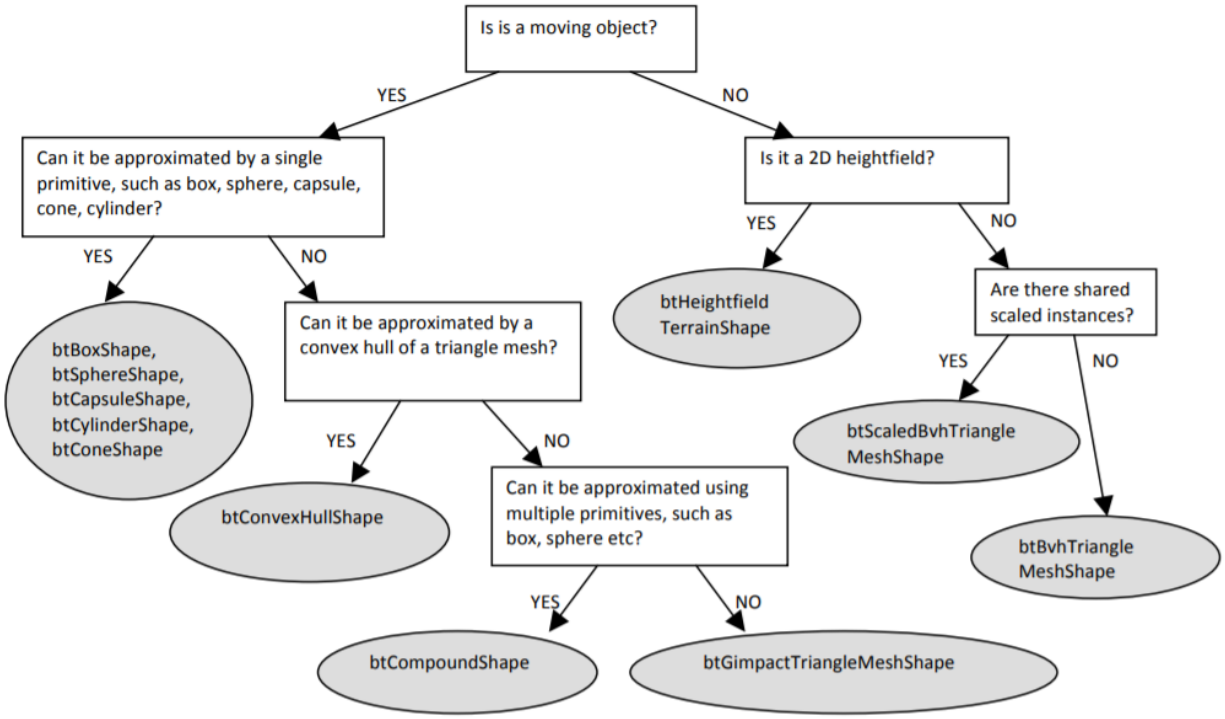
其它:

btTransform: 表示一个变换, 或者理解为一个位置和角度状态. 包含一个旋转和一个平移.

btPersistentManifold： 碰撞接触点结构，保存一对碰撞物体的碰撞点信息。

四、Collision Shape

Bullet Collision 中定义了大量不同的 Collision Shape。需要根据不同的情况选用。



简单几何体:

- ***btBoxShape***:
- ***btSphereShape***
- ***btCapsuleShape***: 沿着 y 轴的胶囊形状。同样还有沿着 x 轴和 z 轴的胶囊形状: btCapsuleShapeX, btCapsuleShapeZ
- ***btCylinderShape***: 沿着 y 轴的柱形。同样还有沿着 x 轴和 z 轴的柱形: btCylinderShapeX, btCylinderShapeZ
- ***btConeShape***: 沿着 y 轴的锥形。同样还有沿着 x 轴和 z 轴的锥形: btConeShapeX, btConeShapeZ
- ***btMultiSphereShape***: 由多个球体组成的凸包。例如两个球体的凸包可组成胶囊形状。

btConvexHullShape

用来表示一个凸几何体的三角网格。可直接将网格的顶点数据传给 btConvxHullShape，即可创建网格的碰撞模型。

btCompoundShape

多个凸的形状可以组合在一起成为一个大的组合形状。btCompoundShape 可以用来表示一些简单凹的形状。非凸的 Mesh 网格在进行碰撞检测时是比较复杂耗时的，如果它可以拆分成多个简单的凸几何体，能有效提高检测效率。

凹几何体

虽然 Bullet 支持使用 btBvhTriangleMeshShape 进行静态的凹几何体的碰撞检测，但建议使用几何体的凸包作为碰撞形状进行碰撞检测。如果单个凸包不能满足精度要求，那么尝试多个凸包组成一个 btConvexHullShape。

高度场地形

Bullet 支持高度场非凸地形的碰撞检测。使用 btHeightfieldTerrainShape.

碰撞检测分发

在 Narrow phase 阶段，根据检测的碰撞物体对的形状不同，bullet 使用 Dispatcher 来分发不同的检测算法。默认的检测算法如下表

	box	sphere	convex,cylinder cone,capsule	compound	triangle mesh
box	boxbox	spherebox	gjk	compound	concaveconvex
sphere	spherebox	spheresphere	gjk	compound	concaveconvex
convex, cylinder, cone, capsule	gjk	gjk	gjk or SAT	compound	concaveconvex
compound	compound	compound	compound	compound	compound
triangle mesh	concaveconvex	concaveconvex	concaveconvex	compound	gimpact

Bullet 支持自定义的检测算法，使用 btDispatcher::registerCollisionAlgorithm 进行注册替换上表中的任意检测算法。

四、具体接口

1、 btCollisionWorld

Bullet Collision 的外用常用接口几乎都定义在 btCollisionWorld 中.

1）btCollisionWorld ::rayTest

```
virtual void btCollisionWorld::rayTest ( const btVector3 &      rayFromWorld,
                                         const btVector3 &      rayToWorld,
                                         RayResultCallback &    resultCallback
                                       )                                    const [virtual]
```

对射线和场景中的所有物体进行光线投射检测。当检测成功后，会调用 *resultCallback* 进行处理。

输入：

- ***rayFromWorld***，射线的起始位置；
- ***rayToWorld***， 射线的终止位置，超出起始点和终点的范围不会被检测。
- ***resultCallBack***，检测成功后的回调类。***RayResultCallback*** 是回调类的虚基类，有一个 ***addSingleResult*** 虚函数，需要继承以实现自己的回调。

Bullet 中实现了用于找最近命中物体的 ***ClostestRayResultCallback***， 和 用于找所有命中的 ***AllHitsRayResultCallback***.

btCollisionWorld ::RayResultCallback:

rayTest 检测的回调接口：

Public Member Functions

virtual btScalar	addSingleResult (LocalRayResult &rayResult, bool normalInWorldSpace)=0
bool	hasHit () const
virtual bool	needsCollision (btBroadphaseProxy *proxy0) const
	RayResultCallback ()
virtual	~RayResultCallback ()

Public Attributes

btScalar	m_closestHitFraction
short int	m_collisionFilterGroup
short int	m_collisionFilterMask
btCollisionObject *	m_collisionObject
unsigned int	m_flags

主要接口函数为 ***addSingleResult***，一旦某次检测成功，那么这个函数就会被调用，以进行一些处理，如，***AllHitsRayResultCallback*** 和 ***ClostestRayResultCallback*** 分别实现了这个接口函数以记录下所有和射线相交的物体或者记录离投射点最近的相交物体等。

函数 ***needsCollision*** 一般会在 ***addSingleResult*** 之前被调用，一旦为真，则调用 ***addSingleResult*** 进行处理。这个函数已经有默认实现，没有特殊需求不需要重写。

2) btCollisionWorld::convexSweepTest

```
void btCollisionWorld::convexSweepTest ( const btConvexShape *   castShape,
                                         const btTransform &     from,
                                         const btTransform &     to,
                                         ConvexResultCallback &  resultCallback,
                                         btScalar                allowedCcdPenetration = btScalar(0.)
                                       )                                                  const
```

对凸几何体一次位移过程进行碰撞检测，检测这个位移过程中发生碰撞的情况。

输入：

- ***castShape,*** 检测的凸几何体；
- ***from***， 位移的起始位置；
- ***to***， 位移的终末位置；
- ***resultCallback,*** 检测成功后的回调类。它的类型 ***ConvexResultCallback*** 和 ***RayResultCallback*** 类似.

3）contactTest

```
void btCollisionWorld::contactTest ( btCollisionObject *        colObj,
                                     ContactResultCallback &  resultCallback
                                   )
```

某个物体与场景中所有其它物体进行碰撞检测。

4）contactPairTest

```
void btCollisionWorld::contactPairTest ( btCollisionObject *        colObjA,
                                         btCollisionObject *        colObjB,
                                         ContactResultCallback &  resultCallback
                                       )
```

两个物体之间的碰撞检测。接触点可能不只有一个

5）performDiscreteCollisionDetection

virtual void btCollisionWorld::performDiscreteCollisionDetection ( ) [virtual]

对场景中所有物体之间进行碰撞检测。检测的结果通过 ***btCollisionWorld*** 的 Dispatcher 获取。具体实例可看 Example2 项目的 demo2。

五. 示例

以所有物体的碰撞检测为例说明如何调用

(1) 创建碰撞检测所需的环境.这里有四个东西需要创建:

- btDefaultCollisionConfiguration, 用于管理配置 bullet collision 所需的内存等.
- btCollisionDispatcher, 管理碰撞的物体对,碰撞点,以及 Narrow phase 检测算法的分发等.
- btDbvtBroadphase, 是 broad phase 检测的类
- btCollisionWorld, 总的碰撞检测的管理和用户调用的类. 它的初始化需要指定上面三个对象.

```
///collision configuration contains default setup for memory, collision setup
btDefaultCollisionConfiguration* collisionConfiguration = new btDefaultCollisionConfiguration();

///use the default collision dispatcher. For parallel processing you can use a diffent dispatcher (see Extras/BulletMultiThreaded)
///collision dispatcher will dispatch narrow phase detection algorithm according to different kinds of collision shapes.
btCollisionDispatcher* dispatcher = new btCollisionDispatcher(collisionConfiguration);

///broad phase collision detector
btDbvtBroadphase* broadphase = new btDbvtBroadphase();

///collision world
btCollisionWorld* collisionWorld = new btCollisionWorld(dispatcher, broadphase, collisionConfiguration);
prender->collisionWorld = collisionWorld;
```

(2) 创建 collision object, 并添加到 collision world 里面.

- 需要为 object 先创建它的 collision shape
- 下面的例子中 Cube 继承了 btCollisionObject, 创建后为它指定它的 shape.

```
// box -------------------------
btBoxShape* box_shape = new btBoxShape(btVector3(0.5, 0.5, 0.5));
Cube* box = new Cube();
box->setCollisionShape(box_shape);
box->setSize(1.0, 1.0, 1.0);

btTransform& box_trans = box->getWorldTransform();
box_trans.setOrigin(btVector3(1.0, 0, -0.0));
box->setInitTransform(box_trans);

prender->addRenderObject(box);
collisionWorld->addCollisionObject(box);
```

(3) 在合适的地方调用碰撞检测函数:

```
// collision detection of all CollisionObjects
collisionWorld->performDiscreteCollisionDetection();
```

(4) 通过 dispatcher 获取碰撞点的信息

- 对于其它类型的碰撞检测, 主要是通过回调类来获取碰撞点等信息的.

```
// get total number of collision manifolds
// a persistent manifold is a contact point cache
int numManifolds = collisionWorld->getDispatcher()->getNumManifolds();

// get all persistent manifolds
for (int i = 0; i<numManifolds; i++)
{
    // get the i-th manifold
    btPersistentManifold* contactManifold = collisionWorld->getDispatcher()->getManifoldByIndexInternal(i);
```