## 1 引言

****Library描述****

bulletphysics是一种专业的使用便携式C ++编写的开源collisiondetection，rigidbody和软体动力学库。 bulletphysics Library主要设计用于游戏，视觉效果和机器人模拟。 Library根据ZLib许可证免费商业使用。   
主要特点

* 离散和连续的collisiondetection，包括射线和凸起扫描测试。 碰撞形状包括凹凸网格和所有的基本图元
* 最大坐标6自由度rigid body（btRigidBody）连接 constraints（btTypedConstraint）以及广义坐标多体 （btMultiBody）使用由动力装置连接的关节运动算法。
* 快速稳定的Rigid BodyDynamicsconstraints求解器，车辆动力学特征控制器和滑块，铰链，通用6DOF和锥形扭转constraints
* 软体动力学用于布，绳和可变形体积与双向相互作用 rigid body，包括constraints支持
* 在Zlib许可证下开源C ++代码，并在所有平台上免费使用任何商业用途 包括PLAYSTATION3，XBox   
  360，Wii，PC，Linux，Mac OSX，Android和iPhone
* Maya Dynamica插件，Blender集成，本机二进制串行化 示例如何导入URDF，Wavefront.obj和Quake .bsp文件。
* 许多示例说明如何使用SDK。 所有的example很容易浏览 OpenGL 3示例浏览器。 每个示例也可以编译没有图形。
* 快速启动指南，Doxygen文档，维基和论坛补充了示例。联系和支持
* 支持和反馈公共论坛，请[http://bulletphysics.org](http://bulletphysics.org/" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)

## 2 编译系统，入门和新增功能

请参阅单独的BulletQuickstart.pdf指南。   
从bullet2.83起，有一个单独的快速入门指南。 这个快速入门指南包括   
新SDK版本的变化和新功能，以及如何构建项目符号   
物理SDK和示例。   
你可以找到 bullet/doc/BulletQuickstart.pdf 本快速入门指南 。

## 3 Library概况

****介绍****   
物理引擎的主要任务是执行collisiondetection，resolvecollisions 等constraints，并提供更新的世界变换1中的所有对象。 本章将给出一个Rigid Body Dynamics流程的概述以及由所有组件共享的基本数据类型和数学库。   
****软件设计****   
bullet被设计为可定制和模块化。 开发者可以

* 仅使用collisiondetection组件
* 使用Rigid BodyDynamics部件，而不需要软体动力学部件
* 仅使用库的小部分，并以多种方式扩展库
* 选择使用单个精度或双精度版本的库
* 使用自定义内存分配器，连接自己的性能分析器或调试hook

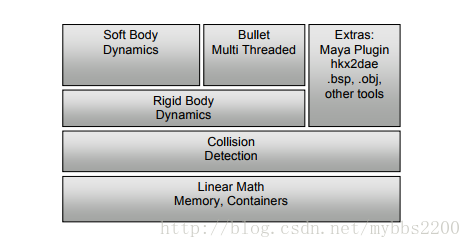
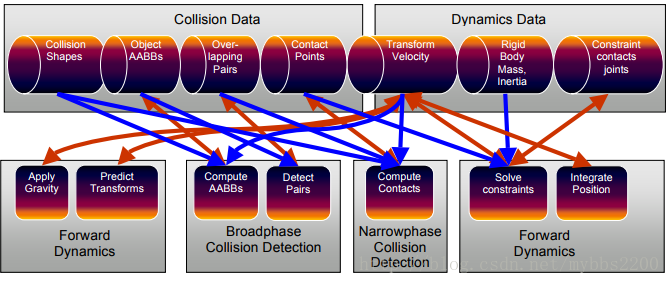
主要组成部分组织如下：

1. 线性数学
2. 内存，容器
3. 碰撞
4. 检测
5. rigid body
6. 动力学
7. 软体
8. 动力学
9. bullet多线程

额外：

1. maya插件
2. hkx2dae
3. .bsp，.obj，

其他工具

****rigid body物理管道****   
在详细介绍之前，下图显示了最重要的数据结构   
   
bullet物理管道的计算阶段。 这个管道从左到右执行，通过应用重力开始，并通过位置整合和更新世界变换结束。   
   
整个物理管道计算及其数据结构在Bullet中由dynamics world表示。 在dynamicsworld中进行“stepSimulation”的时候，所有上述阶段被执行。 默认的dynamicsworld实现是btDiscreteDynamicsWorld。   
bullet让开发人员明确地选择dynamicsworld的几个属性，如宽幅collisiondetection，窄带collisiondetection（调度器）和constraints求解器。   
****集成概述****   
如果要在自己的3D应用程序中使用Bullet，最好遵循HelloWorld中的步骤   
Demo位于Bullet /examples / HelloWorld。 简而言之：

* 创建一个btDiscreteDynamicsWorld或btSoftRigidDynamicsWorld

这些从btDynamicsWorld 继承出来的类 ，让你可以自己管理更高级的接口和物理对象和constraints。 它还实现了每帧的所有对象的信息更新。

* 创建一个btRigidBody并将其添加到btDynamicsWorld

为了构建一个btRigidBody或btCollisionObject，您需要提供：

* ****质量****（动态的 可移动对象为正值 >0，静态对象为0）
* ****CollisionShape**** 如Box，Sphere，Cone，ConvexHull或TriangleMesh
* ****材料性质**** 如:摩擦力和恢复力

每帧进行模拟更新各个物体的信息：

* stepSimulation

调用dynamicsworld的stepSimulation。 这个btDiscreteDynamicsWorld类自动通过执行内部插值而不是小于一帧的单步模拟来得出可变时间的步长。它使用60赫兹的内部固定时间步长。 （就是说计算大于60帧的模拟时，例如90帧中的30帧是通过插值计算出来的）stepSimulation将执行碰撞检测和物理模拟。 通过btMotionState的setWorldTransform的调用来更新可活动对象的世界变换。

基本物理数据类型和数学库的基本数据类型，存储器管理和容器位于 bullet/SRC / LinearMath。

* btScalar

一个 btScalar就是一个浮点数的声明。 为了允许编译单浮点精度和双精度的库，我们设置btScalar数据type。 默认情况下，btScalar是float的类型定义。 当然它也可以定义为 double   
BT\_USE\_DOUBLE\_PRECISION这个宏定义就是来说明当前的浮点数类型，它位于你的编译宏定义，或者在文件bullet/src/ LinearMath / btScalar.h的目录顶部。

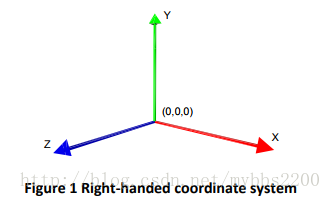
* btVector3

可以使用btVector3来表示3D位置和向量。 btVector3具有3个标量 x，y，z。   
但是，考虑到齐次和SIMD兼容性原因还有第 4 个未使用的w成员。 许多操作可以在一个 btVector3 中来执行 ，如加法减法和获取矢量的长度。

* btQuaternion和btMatrix3x3

三维方向和旋转可以使用btQuaternion或btMatrix3x3中的任何一个来表示 。

* btTransform

btTransform是位置和旋转的组合。 它可以用来进行从一个坐标空间到另一个坐标空间的点和向量的转换。 不允许缩放或剪切。   
*bullet使用右手坐标系：*   
   
图1右手坐标系

btTransformUtil，btAabbUtil用于变换计算和提供AABB计算等常见的实用功能。   
****内存管理，对齐，容器****   
通常重要的数据是16字节对齐的，例如在使用SIMD或DMA在Cell SPU传输时要求如此。 Bullet提供了用来处理对齐的默认内存分配器，开发人员也可以提供自己的内存分配器。bullet使用中的所有内存分配有：

* btAlignedAlloc，允许指定大小和对齐
* btAlignedFree，释放由btAlignedAlloc分配的内存。

要重写默认的内存分配器，您可以选择：

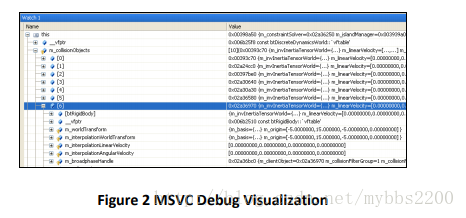
* btAlignedAllocSetCustom当您的自定义分配器不支持对齐时，将使用
* btAlignedAllocSetCustomAligned可以用来设置自定义对齐的内存分配器。

为了确保结构或类将自动对齐，您可以使用此宏：

* ATTRIBUTE\_ALIGNED16（type）variablename 创建一个16字节对齐的变量

通常我们需要维护一个对象的数组。原本Bullet库使用了STL 的std ::vector数据结构的数组，但为了便携性和兼容性原因，我们切换到我们自己的数组类。

* btAlignedObjectArray非常相似的std::vector。 它使用对齐的分配器保证对齐。它具有使用快速排序或堆排序对数组进行排序的方法。

如果要启用MicrosoftVisual Studio的调试器进行btAlignedObjectArray和   
btVector3的可视化调试，需要按照bullet/MSVC / autoexp\_ext.txt的说明进行配置   
   
图2 MSVC调试可视化   
****时间和性能分析****   
为了找到性能瓶颈，Bullet使用宏来进行分层性能测量。

* btClock使用微秒精度来测量时间。
* BT\_PROFILE（SECTION\_NAME）标志着一个分析部分的开始。
* CProfileManager :: dumpAll（）;转储分层的性能输出到console。在进行模拟之后才能调用它。
* CProfileIterator是一个类，可以让你分析树的遍历。

请注意，这里的分析器不使用内存分配器，因此您可能希望在任何的情况下禁用它来检查内存泄漏，或者在创建软件的最终版本时检查。   
这个分析功能可以通过在限定的宏 #defineBT\_NO\_PROFILE 1 这里被关掉。   
Bullet / src / LinearMath / btQuickProf.h   
****调试图****   
可视化的调试仿真数据结构可以对我们很有帮助。 例如，可视化后允许您验证物理模拟数据与图形数据匹配。 还会缩小问题的范围，如constraints的坏帧和极限值出现。   
btIDebugDraw是用于调试绘图接口类。 继承出自己的类和实现virtual drawLine和其他方法。   
使用setDebugDrawer这个接口可以将您的自定义调试hook分配给 dynamicsworld。   
然后，您可以通过设置调试hook的模式来选择绘制特定的调试功能：

animationWorld-> getDebugDrawer（） - > setDebugMode（debugMode）;

* 1

您可以通过调用这个接口来调用调试图

world-> debugDrawWorld（）;

* 1

以下是一些调试模式   
•btIDebugDraw :: DBG\_DrawWireframe   
•btIDebugDraw :: DBG\_DrawAabb   
•btIDebugDraw :: DBG\_DrawConstraints   
•btIDebugDraw :: DBG\_DrawConstraintLimits   
默认情况下，所有对象都可以在给定的调试模式下进行可视化，但是在使用很多对象时这样做会造成杂乱无章的显示。 您可以禁用特定对象的调试图

int f = objects-> getCollisionFlags（）;

ob-> setCollisionFlags（f| btCollisionObject :: CF\_DISABLE\_VISUALIZE\_OBJECT）;

* 1
* 2

## 4 bulletcollision detection

****collision detection****   
collision detection提供最近点（距离和渗透）的算法和加速度的结构来查询和射线和凸起扫描的测试。 主要数据结构如下：

* btCollisionObject是具有世界变换物体和collisionshape。
* btCollisionShape描述了一种碰撞对象的collisionshape，例如方块，球体，凸包或三角网。   
  单个collisionshape可以在多个之间共享碰撞对象。
* btGhostObject是一个特殊的btCollisionObject，用于快速查询定位有效碰撞
* btCollisionWorld存储所有btCollisionObjects，并提供了一个接口来执行查询。

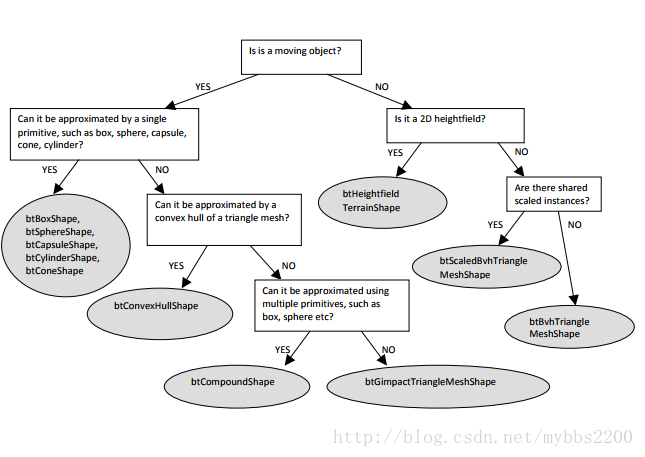
宽幅collision detection提供了加速结构，可快速排除基于轴对齐的边界框（AABB）重叠的物体对。 这里有几种不同的宽幅加速结构可用：

* btDbvtBroadphase采用了快速动态包围体积层次基于AABB树
* btAxisSweep3和bt32BitAxisSweep3实现增量3D扫描裁剪
* btSimpleBroadphase是粗鲁的参考实现。它是缓慢的，但容易理解并进行调试和测试更先进的broadphase时很有用。

broadphase从一对高速缓存重叠对中进行添加和删除。重叠对是保持一段时间后才删除的，并可以缓存信息，如之前交互的constraints 能量，可以用于“warmstarting”：使用以前的解决方案可以更快速度的求解constraints的靠拢情况。   
每个对的碰撞调度迭代，遍历基于所述匹配的碰撞算法都涉及对象的类型，同时会执行该碰撞算法计算交互点。

* btPersistentManifold是一个交互点缓存，用来存储对于给定的一对对象的交互点

## collision shape

bullet支持众多不同的collision shape，并且可以添加自定义的shape。 为了最好的性能和质量的需要，选择适合你目标的collisionshape是很重要的。   
下面的图表可以帮助作出决定：   
   
****ConvexPrimitives****   
最原始的形状是围绕着他们的局部坐标系的一个 原点：

* btBoxShape：方块通过其侧面的一半程度（半长）所定义
* btSphereShape：球形状由半径定义
* btCapsuleShape：绕Y轴胶囊。此外还有btCapsuleShapeX/ Z
* btCylinderShape：绕Y轴圆柱。此外还有btCylinderShapeX/ Z。
* btConeShape：锥绕Y轴。此外还有btConeShapeX/ Z。
* btMultiSphereShape：多个球体的凸包，可以被用来创建一个胶囊（有2个球）或其它凸形状。

****CompoundShapes****   
多个凹凸形状可被组合成一个复合或化合物的形状，那就是btCompoundShape。这是合成出来的凸子部分，称为子形状的凹状。每个子形状都有自己相对于btCompoundShape的本地偏差变换。使用凸包的集合近似凹形状，并将它们存储在一个btCompoundShape这是一个好主意。您可以使用一个实用的方法调整重心

btCompoundShape:: calculatePrincipalAxisTransform。

* 1

****Convex HullShapes****   
bullet支持多种方式来表示的凸三角形网格。最简单的方法是创建一个btConvexHullShape并传递顶点数组。在某些情况下，图形网格包含了过多的顶点直接作用于btConvexHullShape。在这种情况下，应该尽量减少顶点数量。   
****ConcaveTriangle Meshes****   
对于静态的世界环境中，以代表静态三角形非常有效的方式网格是使用btBvhTriangleMeshShape。该collisionshape从btTriangleMesh或btStridingMeshInterface构建一个静态加速结构。而不是在运行时建立树，它是也是一个可序列化并保存到硬件的二叉树。详见*example/ ConcaveDemo*如何保存和加载这btOptimizedBvh的树加速结构。当你有相同的几个实例三角形网格，但有不同的比例，你可以重用实例btBvhTriangleMeshShape或btScaledBvhTriangleMeshShape。该btBvhTriangleMeshShape可以存储   
多个网格部分。它在一个32位的结构里建立一个三角形索引和索引部分，保留10个byte用于部分ID和剩余的22位三角形索引。如果你需要超过2万个三角形，任一分裂的三角形网孔分成多个子网格，或去改变默认的#defineMAX\_NUM\_PARTS\_IN\_BITS在*SRC\ BulletCollision \ BroadphaseCollision \ btQuantizedBvh.h*文件中。

****ConvexDecomposition****

理想情况下，凹网格应仅用于静态的艺术品。否则，它的凸包应该是使用btConvexHullShape进行网格传递。如果单个凸形状是不够详细来描述形状的，多个凸部可以被组合成称为复合对象btCompoundShape。凸   
分解可以被用于将凹网格分解成若干凸的部分。查看*Demos/ConvexDecompositionDemo* 做ConvexDecomposition的一种程序计算的自动方式。

****Height field****

bullet提供了一种用于平板2D凹地形的特殊支持   
btHeightfieldTerrainShape。见*example/TerrainDemo*展示其用法。

****btStaticPlaneShape****

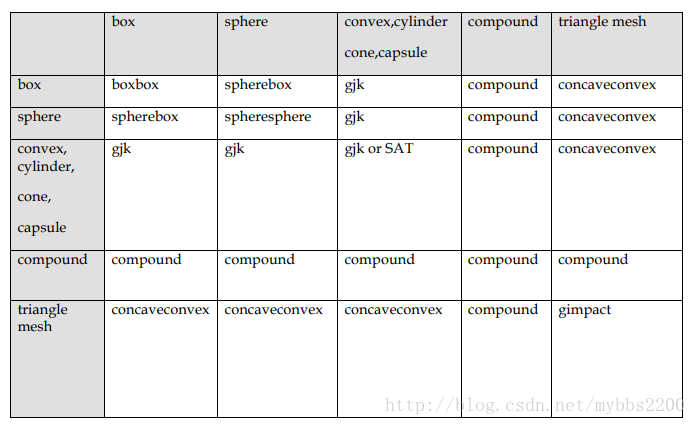
顾名思义， btStaticPlaneShape可以代表无限平面或半空间。这个形状只能用于静态的，非移动的物体。这种形状被引入主要是为了做Demos。

****Scaling ofCollision Shapes****

一些collision shape都可以应用局部缩放。 使用btCollisionShape:: setScaling（Vector3）。非均匀缩放可以每个轴使用不同的缩放值，它可用于btBoxShape，btMultiSphereShape，btConvexShape，btTriangleMeshShape。如果想要均匀缩放，使用x值作用于所有轴，可作用于btSphereShape。*注意，非均匀缩放的球体可以通过创建btMultiSphereShape的1个球体然后再使用。*和以前一样，提到btScaledBvhTriangleMeshShape允许实例化btBvhTriangleMeshShape使用不同的非均匀缩放因子。所述btUniformScalingShape允许实例化不同尺度的凸形，并减少内存量。

****CollisionMargin****   
bullet使用一个小的CollisionMargin作用于collision shapes以提高collision detection的性能和可靠性。****最好不要修改默认的CollisionMargin，如果你不使用正值：零余量可能会带来问题****。默认情况下，该碰撞边距CollisionMargin设置为0.04，这是4厘米，如果你的单位是米（推荐）。   
****边缘****取决于物体的collisionshape有不同的含义。一般的碰撞余量将扩大对象来定义边缘。这将创建一个小的差距。为了弥补这一点，一些形状会减去这个余量来产生实际的大小。例如，btBoxShape减去CollisionMargin   
的一半。对于btSphereShape，整个半径都是CollisionMargin所以没有误差会产生。不要重写球体的CollisionMargin。对于凸壳，圆柱和圆锥，所述余量被添加到对象的宽度，所以会产生间隙，除非你调整图形啮合碰撞体积。对于凸包来说，是通过缩小对象边缘来除去由引入的间隙的。见*example/Importers/ ImportBsp*为这种牛逼技术的应用。

****CollisionMatrix****   
对于每一对类型的形状，bullet将通过使用调度程序分配一定的碰撞算法。   
默认情况下，整个矩阵充满了以下算法。需要注意的是一些凸的多面体，圆柱体，锥体和囊等GJK兼容图元。（GJK代表*Gilbert,Johnson and Keerthi* 发明凸起的距离计算算法的人） 它是与EPA组合为渗透深度计算。EPA是ExpandingPolythope Algorithm算法。bullet都有自己的免费实现GJK和EPA。

   
注册自定义collision shape和算法   
用户可以注册一个自定义的collisiondetection算法，并通过使用btDispatcher :: registerCollisionAlgorithm重写在这种碰撞矩阵中的任何条目。请看*examples/ UserCollisionAlgorithm* 为一个example，它演示注册一个SphereSphere碰撞算法。

## 5 Collision Filtering (selective collisions)碰撞过滤（选择性碰撞）

bullet提供了三种简单的方法，以确保只有特定的对象相互碰撞：masks，broadphasefilter callbacks和nearcallbacks。值得注意的是，基于masks的碰撞选择是经常使用的。总之，如果masks满足你的用途，就使用它们; 他们有更好的表现和很多更易于使用。   
****当然，不要只是因为性能可能会更好一点就试图将鞋拔子一样的东西放到基于masks的选择系统，该系统显然不适合。****

****使用masks过滤碰撞（Filteringcollisions using masks）****   
bullet支持按位掩码来决定的这个物体是否应该与其他东西碰撞，或接收碰撞。

int myGroup = 1;int collideMask = 4;

world->addCollisionObject(object,myGroup,collideMask);

* 1
* 2
* 3

在broadphase collision detection重叠对被添加到一对高速缓存期间，仅当masks的组中的其他对象可以被匹配（在needsBroadphaseCollision）

bool collides = (proxy0->m\_collisionFilterGroup &proxy1->m\_collisionFilterMask) != 0;collides = collides && (proxy1->m\_collisionFilterGroup &proxy0->m\_collisionFilterMask);

* 1
* 2

如果你在masks中需要放置非常多的类型，超过了给你的可用对象32bits，或者一些碰撞是基于其他因素来启用或禁用的，那么有几种方法可以注册回调到自定义的逻辑的实现，来实现只传递是你想要的那些碰撞：

****Filtering Collisions Using a Broadphase Filter Callback（过滤碰撞 使用Broadphase过滤回调）****   
其中一个有效的方法就是注册一个broadphase过滤器回调。此回调实现在碰撞管道中非常早期的阶段，然后防止碰撞对的产生。

struct YourOwnFilterCallback : public btOverlapFilterCallback

{

// return true when pairs need collision

virtual bool needBroadphaseCollision(btBroadphaseProxy\*proxy0,btBroadphaseProxy\* proxy1) const

{

bool collides = (proxy0->m\_collisionFilterGroup &proxy1->m\_collisionFilterMask) != 0;

collides = collides && (proxy1->m\_collisionFilterGroup &proxy0->m\_collisionFilterMask);

//add some additional logic here that modified 'collides'

return collides;

}

};

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11

然后建立这个类的一个对象，并使用注册此回调：

btOverlapFilterCallback \* filterCallback = new YourOwnFilterCallback();

dynamicsWorld->getPairCache()->setOverlapFilterCallback(filterCallback);

* 1
* 2

****Filtering Collisions Using a Custom NearCallback（过滤碰撞 使用自定义NearCallback）****   
另一个回调可以在narrowphase期间被注册，当所有的碰撞对都由broadphase产生后。该btCollisionDispatcher :: dispatchAllCollisionPairs对于每一对都调用此narrowphasenear callback的’btCollisionDispatcher:: needsCollision’中进行测试。您可以自定义此nearcallback：

void MyNearCallback(btBroadphasePair& collisionPair,

btCollisionDispatcher& dispatcher, btDispatcherInfo& dispatchInfo) {

// Do your collision logic here

// Only dispatch the Bullet collision information if you want the physics tocontinue

dispatcher.defaultNearCallback(collisionPair, dispatcher, dispatchInfo);

}

mDispatcher->setNearCallback(MyNearCallback);

* 1
* 2
* 3
* 4
* 5
* 6
* 7

****Deriving your own class from btCollisionDispatcher（从btCollisionDispatcher派生自己的类）****   
想要在碰撞调度中更细腻的进行控制，你可以从btCollisionDispatcher派生您自己的类并重写下列一种或多种方法：

virtual bool needsCollision(btCollisionObject\* body0,btCollisionObject\*body1);virtual bool needsResponse(btCollisionObject\* body0,btCollisionObject\*body1);virtual void dispatchAllCollisionPairs(btOverlappingPairCache\*pairCache,const

btDispatcherInfo& dispatchInfo,btDispatcher\* dispatcher) ;

* 1
* 2
* 3
* 4

## 6 Rigid BodyDynamics

****介绍****   
Rigid Body Dynamics是在collisiondetection模块的基础上实现的。它增加了力量，质量，惯性，速度和（约束）constraints。

btRigidBody用于模拟单6自由度运动的物体。btRigidBody从btCollisionObject继承来，所以它继承了其世界变换，摩擦和回归力并增加了线性速度和角速度。

btTypedConstraint为rigidbody的constraints的基类，包括   
btHingeConstraint，btPoint2PointConstraint，btConeTwistConstraint，   
btSliderConstraint和btGeneric6DOFconstraint。

btDiscreteDynamicsWorld是UserCollisionAlgorithmbtCollisionWorld，并且是rigidbody和constraints的容器。它提供了stepSimulation。

btMultiBody是RoyFeatherstone所讨论的使用广义（或简化）坐标的刚体层次结构的替代表示，使用铰接体算法。   
树形层次结构以固定或浮动的基础和子体（也称为链接）通过关节连接：1-DOF旋转关节（类似于btRigidBody的btHingeConstraint），1-DOF棱柱形关节（类似于btSliderConstraint）

****请注意，btMultiBody最近在BulletPhysics SDK中引入，并且仍在修改中。**** 在本文中，仅讨论基于btRigidBody和btTypedConstraints的最大坐标。 在将来的修订中，将添加一个关于btMultiBody的章节。 现在，如果您对btMultiBody感兴趣，请参阅示例浏览器及其*examples/MultiBodyand examples/ImportURDF*。

****Static, Dynamic and Kinematic Rigid Bodies****   
有3种不同类型的bullet对象：   
•动态（移动）rigid body   
- 正质量   
- 每个模拟支架的动态更新了自己的世界变换

•（静态的刚体）rigid body   
- 零质量   
- 不能移动，但只是发生碰撞

•（主动刚体）Kinematic rigidbodies   
- 零个质量   
- 可以由用户设置动画，但只会有一个双向互动：动态对象将被推开，但没有从动态物体而来的影响

所有这些都需要被添加到dynamicsworld。rigidbody可以分配一个collisionshape。这个形状可被用来计算质量分布，也称为惯性张量。

****Center of mass World Transform****

在bullet中，世界变换rigid body的总是其质量中心，并在这个基础还定义了其用于惯性的本地帧。本地惯性张量取决于物体的形状，然后btCollisionShape类提供计算局部惯性的方法，并赋予了质量。这个世界变换必须是rigid body的变换，这意味着它不应该包含缩放，剪切等，如果你想要一个对象进行缩放，可以缩放collision shape。其他的变换，例如剪切，如果必要的话可以应用（烘焙）成三角形网格的顶点。   
在这种情况下，collision shape不与质量中心对齐变换，它可以相应的被转移。   
对于这一点，你可以使用一个btCompoundShape，并使用孩子的变换来转移孩子的碰撞形状。

****什么是MotionState？****

MotionStates是bullet中的一种方式，帮你做所有的困难工作，如世界对象到被模拟到程序的渲染里的变换。   
在大多数情况下，你的游戏循环将迭代你每帧的渲染前要模拟的对象。对于每一个对象，你会更新渲染物体与物理身体的位置。bullet使用一种叫MotionStates为您节省这方面的努力。   
有MotionStates多个其他的好处：

* 参与计算的只是运动物体（已经移动的）和它周围的环境， 如果它是不动的，一点也不会更新渲染对象的每一帧的位置。
* 您需要仅仅做渲染的东西在其中。他们可以很有效的通知你一个物体已移动，并且通知代码需要进行更新。
* 插值通常只在屏幕上看到的东西的背景下才有意义。bullet通过MotionStates管理物体位置的插值。
* 您可以跟踪图形对象和质心之间的变换的转变。
* 它们很容易

****Interpolation（插值）****

bullet知道如何插值身体的运动比较适合你。如前所述，插值是通过MotionStates处理的。如果试图通过btCollisionObject :: getWorldTransform或btRigidBody :: getCenterOfMassTransform查询一个body的位置，它将返回在最后结束一刻的物理位置。这在很多情况是有用的，但对于渲染你偶尔会想要一些插值，bullet会在传递给setWorldTransform之前内插所述body的变换值。   
如果你想要一个身体的非内插位置（这将是最后一个物理时刻末尾计算的位置），请使用btRigidBody:: getWorldTransform（）直接查询body。

****So how do I use one?****

MotionStates是在bullet两个地方使用。   
第一个是身体首次创建时。 当身体进入模拟时，Bullet抓取身体的初始位置，当身体进入模拟时，Bullet调用getWorldTransform并引用该变量，希望您使用变换信息填充该变量。Bullet还会在运动体上调用getWorldTransform。 请参阅下面的部分，在模拟期间的第一次更新之后，Bullet将获取物体的MotionState来驱动这个物体。Bullet使用body的transform来调用setWorldTransform来适当的更新对象。要实现一个这样的功能，只需继承btMotionState并重写getWorldTransform和setWorldTransform。

****DefaultMotionState****

虽然建议，没有必要推导从btMotionState到自己的motionstate接口。bullet提供默认motionstate，您可以使用这个。简单地用它构建一个你的body默认的变换：

btDefaultMotionState\* ms =new btDefaultMotionState();

* 1

有一个OGRE3D运动状态在附录的example。

****Kinematic Bodies****

如果您打算去应用动画或移动一个静态对象，你应该将它们标记为可运动的。同时为他们禁用 睡眠/停用 在动画过程中。这意味着bullet dynamicsworld将每个模拟帧从btMotionState得到一个新的worldtransform。

body->setCollisionFlags( body->getCollisionFlags() |

btCollisionObject::CF\_KINEMATIC\_OBJECT);

body->setActivationState(DISABLE\_DEACTIVATION);

* 1
* 2
* 3

如果您使用的是Kinematic Bodies，然后getWorldTransform每模拟步都被调用。 这意味着你的Kinematic body的motion state应该有一个机制（如主动动画）来推动当前位置的运动body进入motionstate。

****Simulationframes and interpolation frames（仿真帧和插值帧）****

默认情况下，Bullet物理模拟运行在60赫兹（0.01666）的内部固定帧率。 游戏或应用程序可能具有不同的或甚至可变的帧速率。 为了将应用程序与模拟帧率分离，自动内插方法被嵌入到步进模拟中：当应用程序增量时间小于内部固定时间步长时，Bullet将内插世界变换，并将内插的世界变量发送到btMotionState，而不执行物理模拟。 如果应用程序的时间步长大于60赫兹，则在每个“stepSimulation”调用期间可以执行多于1个仿真步骤。 通过传递最大值作为第二个参数，用户可以约束模拟步骤的最大数量。

当创建rigid body，他们将检索从btMotionState初始worldtransform，   
使用btMotionState :: getWorldTransform。当模拟运行时，使用   
stepSimulation，新worldtransform是使用主动更新的rigid body   
btMotionState :: setWorldTransform。   
动态rigid body有正质量，并且它们的运动是由模拟来确定。静态的和kinematicrigid body具有零质量。静态对象不应该由用户来移动。

翻译于 [http://blog.csdn.net/mybbs2200/article/details/70141765](http://blog.csdn.net/mybbs2200/article/details/70141765" \t "http://blog.csdn.net/mybbs2200/article/details/_blank) 请支持或者回复发现的错误，以供修正。

## 7Constraints

bullet实施了一些约束。参见*examples/ConstraintDemo* 为它们每个约束做得示例。所有btRaycastVehicleconstraints都源自 btTypedConstraint。   
constraints之间的2个rigid body，他们中的至少一个需要是动态的。

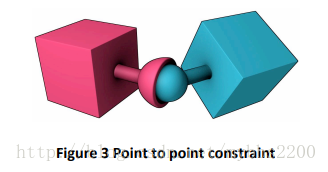
****Point toPoint Constraint****

点对点constraint 约束了位移，以便在2个rigid body在世界空间匹配。rigid body的A链可使用该constraint来连接。

btPoint2PointConstraint(btRigidBody& rbA,const btVector3& pivotInA);

btPoint2PointConstraint(btRigidBody& rbA,btRigidBody& rbB, constbtVector3& pivotInA,const btVector3& pivotInB);

* 1
* 2

   
图3///

****HingeConstraint(铰链约束)****

铰链constraints，转动关节约束了两个附加的角自由度，所以身体只能绕一个轴，就是所述的铰链轴。代表门或车轮绕一个轴旋转时这可能是有用的。用户可指定用于铰链的约束。

btHingeConstraint(btRigidBody& rbA,const btTransform& rbAFrame, booluseReferenceFrameA = false);

btHingeConstraint(btRigidBody& rbA,const btVector3&pivotInA,btVector3& axisInA, bool useReferenceFrameA = false);

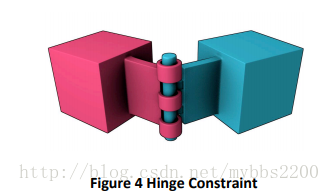
btHingeConstraint(btRigidBody& rbA,btRigidBody& rbB, constbtVector3& pivotInA,const btVector3&

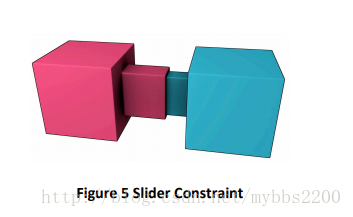
pivotInB, btVector3& axisInA,btVector3& axisInB, booluseReferenceFrameA = false);

btHingeConstraint(btRigidBody& rbA,btRigidBody& rbB, constbtTransform& rbAFrame, const btTransform& rbBFrame, bool

useReferenceFrameA = false);

* 1
* 2
* 3
* 4
* 5
* 6

   
图4///tu铰链constraint

****SliderConstraint（滑块constraint）****   
滑块constraint使身体绕一个轴和沿此轴btSliderConstraint(btRigidBody& rbA, btRigidBody& rbB, constbtTransform& frameInA, const btTransform& frameInB ,booluseLinearReferenceFrameA);   
5///滑块constraint   
****Cone TwistConstraint（锥扭constraint）****   
要创建一个人偶，Cone TwistConstraint用来做四肢像上臂非常有用的。这是一个特殊的点对点constraints，增加了锥体和扭轴的约束。在x - 轴用作扭转轴线。

btConeTwistConstraint(btRigidBody& rbA,const btTransform& rbAFrame);

btConeTwistConstraint(btRigidBody& rbA,btRigidBody& rbB,constbtTransform& rbAFrame, const btTransform& rbBFrame);

* 1
* 2

****Generic 6Dof Constraint（通用6自由度constraint）****

这个通用constraints可以模拟各种标准的约束，通过配置各6自由度（DOF）的。前3个自由度轴是线性轴，其表示rigid body的位移，和后者3个自由度轴代表角运动。每个轴既可以锁定，自由或约束。在建设一个新的btGeneric6DofSpring2Constraint时，所有轴都被锁定。事后这些轴可以被重新配置。需要注意的是几种组合，其中包括自由/被限制的角的自由度是不确定的。参考*Bullet/examples/Dof6SpringSetup.c*

btGeneric6DofConstraint(btRigidBody& rbA, btRigidBody& rbB, constbtTransform& frameInA, const btTransform& frameInB,booluseLinearReferenceFrameA);

下面是约定：

btVector3 lowerSliderLimit = btVector3(-10,0,0);

btVector3 hiSliderLimit = btVector3(10,0,0);

btGeneric6DofSpring2Constraint\* slider = newbtGeneric6DofSpring2Constraint(\*d6body0,\*fixedBody1,frameInA,frameInB);

slider->setLinearLowerLimit(lowerSliderLimit);

slider->setLinearUpperLimit(hiSliderLimit);

* 1
* 2
* 3
* 4
* 5

对于每个轴：   
• LOWERLIMIT == UPPERLIMIT - >轴被锁定。   
• LOWERLIMIT> UPPERLIMIT - >轴是自由的   
• LOWERLIMIT <UPPERLIMIT - >轴它在该范围内约束

建议使用btGeneric6DofSpring2Constraint，它比原有的btGeneric6Dof（Spring）Constraint有一些改进。

## 8 Actions:Vehicles & Character Controller(操作：车辆及人物控制器)

****Action接口****   
在某些情况下，处理物理管道内的一些定制的物理游戏的代码是非常有用的。   
虽然可以使用时钟回调，当需要更新很多个对象的时候，也可以从派生自定义类btActionInterface并调用btActionInterface::updateAction(btCollisionWorld\*world, btScalar deltaTime)来变得更方便的。 这里有内置的example，*btRaycastVehicle和btKinematicCharacterController*，正在使用该btActionInterace。

****RaycastVehicle（光线投射车辆）****

对于街机风格的车辆模拟，建议使用在btRaycastVehicle提供的简化bullet车辆模型。而不是模拟的每个车轮和底盘作为单独的rigidbody，由constraint连接，它使用一个简化模型。这种简化的模型有很多好处，而且是广泛用于商业的驾驶游戏的。   
整个车辆被表示为一个单一的rigidbody，底盘。这里的collisiondetection车轮近似由射线管型，和轮胎摩擦是一个基本的各向异性摩擦模型。   
见*src/BulletDynamics/Vehicle 和examples/ForkLiftDemo*了解更多详情，或检查bullet论坛。   
Kester Maddock在这里分享了bullet车辆模拟了一个有趣的文件：   
[http://tinyurl.com/ydfb7lm](http://tinyurl.com/ydfb7lm" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)   
****CharacterController（角色管理）****

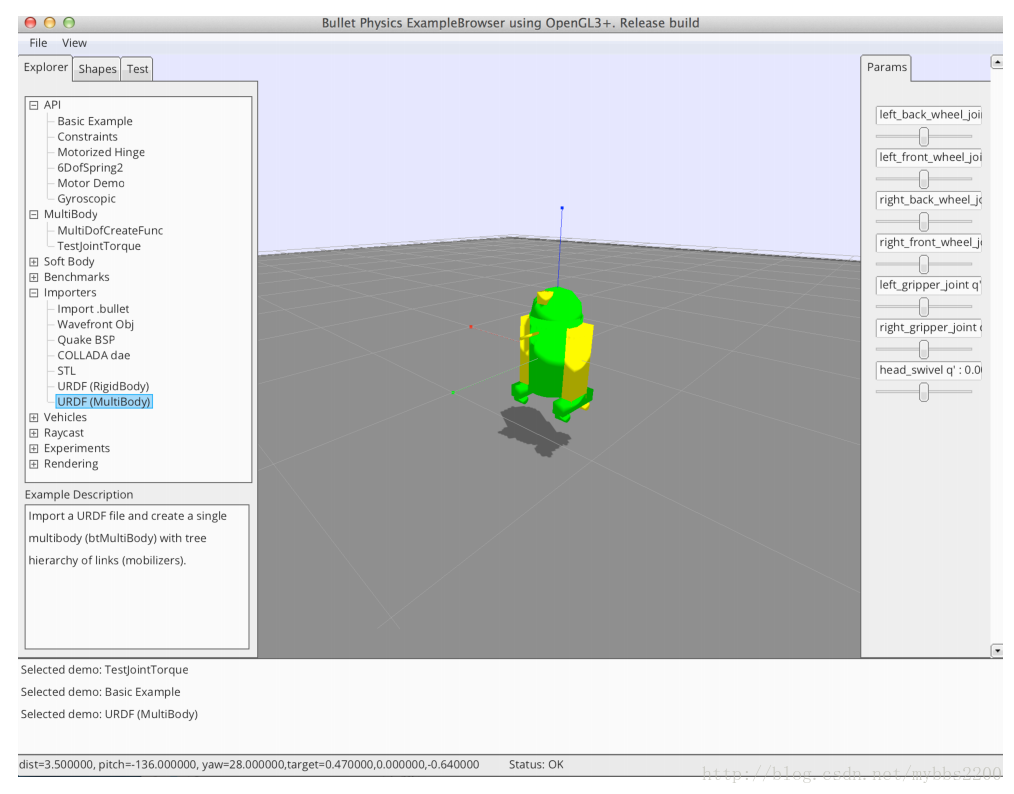
玩家或NPC字符可使用的胶囊形状，球形或其它形状来构造。避免旋转，可以设置“角因子”为零，这期间禁止角旋转作用碰撞和constraint等。见btRigidBody :: setAngularFactor。其他选项（不太推荐）包括用于向上轴逆惯性张量设定为零，或使用angular-only铰链constraint。   
还有一个实验（ btKinematicCharacterController 有几个悬而未决的问题。）btKinematicCharacterController作为一个example的非物理   
角色管理。它采用了btGhostShape进行碰撞查询来创建一个字符   
能爬楼梯，顺畅地滑动沿墙壁等。参见*src/BulletDynamics/Character和Demos/CharacterDemo*其用法。

## 9Soft BodyDynamics（柔体动力学）

///暂时翻译到这里，下面的特性用到时候继续翻译//   
初步文档   
介绍   
柔体动力学提供了绳子，布料模拟和体积柔体，在顶部   
现有Rigid BodyDynamics。有软，rigidbody之间的双向互动   
碰撞对象。   
• btSoftBody是主要的软体物体。它源自btCollisionObject。不一样   
rigid body，柔体没有一个单一的世界变换：指定每个节点/顶点   
在世界坐标。   
• btSoftRigidDynamicsWorld是软的，rigidbody和碰撞容器   
对象   
最好是从学习 的example/SoftBodyDemo如何使用柔软的身体仿真。   
下面是一个简单地说一些基本准则：   
从三角网格结构   
该btSoftBodyHelpers :: CreateFromTriMesh可以自动从创建柔体   
三角形网格。   
碰撞集群   
默认情况下，软机构执行的顶点（节点）和三角形（面）之间使用的collisiondetection。   
这需要密集的细分，否则可能碰撞被遗漏。一种改进的方法使用   
自动分解成凸变形的簇。为了使碰撞集群，使用方法：   
psb-> generateClusters（numSubdivisions）;   
//启用软体和rigid body之间簇碰撞   
psb-> m\_cfg.collisions   
+ =   
btSoftBody :: fCollision :: CL\_RS;   
//启用集群之间的碰撞柔软的身体和柔软的身体   
psb-> m\_cfg.collisions + = btSoftBody :: fCollision :: CL\_SS;   
该ExampleBrowser的软体有一个调试选项，以可视化的凸碰撞集群。

运用力量身子一软   
有一个力施加到每个顶点（节点），或在一个单独的节点的方法：   
软体 - >addForce（常量btVector3＆forceVector）;   
软体 - >addForce（常量btVector3＆forceVector，INT节点）;   
柔软的身体constraints   
它是可以固定一个或多个顶点（节点），使其不能移动的：   
softbody-> setMass（节点，0.f）;   
或柔软的身体的一个或多个顶点附加到rigidbody：   
softbody-> appendAnchor（INT节点，btRigidBody\*rigid body，布尔   
disableCollisionBetweenLinkedBodies = FALSE）;   
它也可以连接两个软体使用constraints，看到bullet/Demos/软体。

## 10 bullet实施例的浏览器

bullet2.83介绍了基于OpenGL3+一个新的example浏览器，取代了之前的转运蛋白   
Demos。存在与对OpenGL2有限的支持的命令行选项：–opengl2   
示例浏览器在Windows，Linux和Mac OSX测试。它具有视网膜的支持，所以它看起来   
最好在Mac OSX。下面是截图：   
  
每个example也可以被编译独立显卡没有。查看   
example/ BasicDemo / main.cpp中如何做到这一点。

BSPDemos   
导入地震.bsp文件和刷转换成凸的对象。这种性能更好然后使用   
三角形。   
车辆Demos   
本Demos展示了使用内置的车辆。该轮由射线管型近似。 这个   
近似非常适用于快速移动的车辆。   
叉车Demos   
一个Demos，Demos如何使用constraints像铰链和滑动constraints打造叉车车辆。

## 11个高级低级技术Demos

碰撞接口技术Demos   
本Demos展示了如何使用bullet的collisiondetection没有动态。它使用   
btCollisionWorld类，并填补了这个将btCollisionObjects。的   
performDiscreteCollisionDetection方法被调用，并Demos了如何收集   
交互点。   
碰撞Demos   
这个Demos是越来越低的水平，那么先前的碰撞接口技术Demos。它直接使用   
btGJKPairDetector查询两个对象之间的最近点。   
用户碰撞算法   
显示如何注册一个处理碰撞自己的collisiondetection算法   
检测针对特定一对碰撞type。一个简单的球，球壳会取代默认GJK   
检测。   
GJK凸演员/扫描Demos   
该Demos展示如何进行碰撞对象之间的线性扫描和返回的时间   
影响。这可能是为了避免摄像头和角色控制贯穿有用。   
凸形状的连续碰撞   
使用连续collisiondetection冲击查询的时间，两个旋转和之间示出   
位移的对象。它采用了bullet的实施保守进步。

光线追踪Demos   
这显示collisionshape使用CCD射线投射的。它实现了光线追踪，可以   
准确地可视化collisionshape的隐式表示。这包括CollisionMargin，   
隐式对象，闵可夫斯基和以及其他形状是很难想象的凸壳   
除此以外。

单纯Demos   
这是一个非常低的水平的Demos测试GJK子距离算法的内部工作。 这个   
计算单工和原点，其被绘制为红色线之间的距离。单工   
含有1到4个点，该Demos示出了4点的情况下，一个四面体。沃罗诺伊单纯求解   
使用时，在他的collisiondetection书克里斯特•埃里克森描述。

## 12个创作工具和序列化

collision shapes，rigidbody和constraints可以在三维创作工具来创建和导出到   
那bullet可以读取的文件格式。

maya动感中插件

华特迪士尼动画工作室贡献了自己的- 房子maya插件作者bullet碰撞   
形状和rigid body开源。可以动感中在Maya地模拟rigid body动感中，   
它可以出口到bullet.bullet物理文件和COLLADA物理。最新的版本有初步   
对于布/软身体支撑。

有一个在bulletwiki页面了解更多信息。你可以下载的预编译的版本   
动感中从插件适用于Windows或Mac OSXhttp://bullet.googlecode.com。   
动感中的源代码库是下[http://dynamica.googlecode.com](http://dynamica.googlecode.com/" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)   
搅拌机   
开源3D制作套装软件Blender使用Bullet物理动画和它的内部游戏   
发动机。见[http://blender.org](http://blender.org/" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)   
搅拌机拥有自营出口COLLADA物理文件的选项。还有，可以直接读取项目   
从搅拌机.blend文件的任何信息，包括collisionshape，rigidbody和constraints   
信息。 见[http://gamekit.googlecode.com](http://gamekit.googlecode.com/" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)   
搅拌机2.57后来有出口直接从游戏引擎.bullet文件的选项。这个可以   
使用来完成 exportBulletFile（“name.bullet”）在Python命令   
PhysicsConstraints模块。   
四维影院，光波CORE，胡迪尼   
CINEMA 4D 11.5使用bullet的rigid body模拟，并有报告称光波CORE   
还计划利用bullet。

对于胡迪尼有一个DOP /插件，请参考   
[http://code.google.com/p/bullet-physics-solver/](http://code.google.com/p/bullet-physics-solver/" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)

序列化和bullet.bullet二进制格式   
bullet2.76起先后到dynamicsworld保存到一个二进制转储能力。保存   
对象和形状到缓冲区是内置的，因此没有额外的库是必需的。下面是一个example   
如何dynamicsworld保存到一个二进制文件.bullet：   
btDefaultSerializer \*串行=新btDefaultSerializer（）;   
dynamicsWorld->的serialize（串行）;   
FILE \*文件=的fopen（“testFile.bullet”，“WB”）;   
的fwrite（serializer->getBufferPointer（），serializer->getCurrentBufferSize（），如图1所示，文件）;   
FCLOSE（文件）;   
您可以按在大多数bulletexampleF3键保存“testFile.bullet”。您可以阅读.bullet   
使用btBulletWorldImporter作为实现文件   
bullet/example/Importers/ ImportBullet。   
约.bullet系列化Futher信息是在bullet在维基   
[http://bulletphysics.org/mediawiki-1.5.8/index.php/Bullet\_binary\_serialization](http://bulletphysics.org/mediawiki-1.5.8/index.php/Bullet_binary_serialization" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)

## 13个一般提示

避免过小和非常大的collisionshape   
为移动对象的最小对象的大小为约0.2单位，20厘米地球重力。 如果   
更小的物体或更大的重力被操纵，减少内部仿真频率   
因此，使用的第三个参数btDiscreteDynamicsWorld :: stepSimulation。通过   
默认情况下为60Hz。例如，模拟骰子掷（与9.8米的重力1厘米宽框/ S2）   
需要至少为300Hz的频率（1./300）。建议保持的最大尺寸   
移动物体较小然后约5单位/米。   
避免大的质量比（差异）   
当沉重的对象是一个很轻的物体上休息模拟变得不稳定。最好是保持   
质量周围1。这意味着一个罐和一个很轻的物体之间的精确相互作用是不现实的。   
结合多个静态三角形网格成一个   
许多小btBvhTriangleMeshShape污染broadphase。更好地结合起来他们。   
使用默认的内部固定时间步长   
bullet效果最好至少60赫兹（1/60秒）的固定时间步长内。   
对于安全性和稳定性，bullet都会自动细分可变时间步长为固定内部   
仿真子步骤，直至指定为第二个参数的子步骤的最大数目   
stepSimulation。当时间步长越小则内部分步骤，bullet将插值   
运动。   
此安全机构可通过使0作为子步骤的最大数目（第二被禁用   
参数stepSimulation）：内部时间步长和分步骤被禁用，而实际   
时间步长的模拟。不建议禁用此安全机制。   
对于布娃娃使用btConeTwistConstraint   
这是更好地建立一个布娃娃出来的btHingeConstraint和/或btConeTwistLimit的膝盖，   
肘部和手臂。   
不要设置CollisionMargin为零   
collision detection系统需要的性能和稳定性一定的余量。如果差距是明显的，   
请补偿图形表示。

使用小于100个顶点以凸起目   
最好是保持在一个顶点数量btConvexHullShape有限。这是为了更好的   
性能和过多的顶点可能会导致不稳定。使用btShapeHull实用程序简化   
凸包。   
避免巨大或者退化三角形三角形网格   
保持合理的三角形的大小，比如低于10个单位/米。此外退化三角形大   
可以更好地避免每一个侧面或接近零区域之间的尺寸比例。   
该分析功能btQuickProf绕过内存分配器   
如果有必要，检查内存泄漏时，或创建最终版本时禁用评测   
你的软件版本。的分析功能可以通过定义被关闭 的#define   
BT\_NO\_PROFILE 1在bullet/SRC / LinearMath / btQuickProf.h

## 高级主题

每个三角形摩擦和归还值

默认情况下，只有一个rigidbody一个摩擦值。您可以为每个形状或每个实现   
三角形摩擦对于更多的细节。见Demos/ ConcaveDemo如何设置每个三角形的摩擦。   
基本上，增加CF\_CUSTOM\_MATERIAL\_CALLBACK碰撞标志或rigid body，和   
注册一个全球性的材料回调函数。为了鉴定在网格三角形，既triangleID和   
网格的PARTID传递给材料回调。这符合跨越式的triangleId/ PARTID   
网接口。   
一个简单的方法是使用btMultimaterialTriangleMeshShape。见   
Demos/ MultiMaterialDemo的使用。   
其他MLCPconstraints求解器   
bullet使用其btSequentialImpulseConstraintSolver默认。您可以使用不同的   
constraints求解器，通过它传递到您的btDynamicsWorld的构造。这些替代   
MLCPconstraints求解器是bullet/src目录/ BulletDynamics/ MLCPSolvers。见的源代码   
example/辆/VehicleDemo如何使用不同的constraints求解。

定制摩擦模型   
如果您想为某些type的对象的不同摩擦模型，你可以注册一个摩擦   
在功能constraints求解某些身体type。此功能与高速缓存兼容   
友好的constraints解算器设置。   
见的#defineUSER\_DEFINED\_FRICTION\_MODEL在Demos/CcdPhysicsDemo.cpp。

## 14 并行使用的OpenCL

OpenCL的rigidbody和collisiondetection。   
我们从头开始实现rigid body和collisiondetection流水线运行使用100％   
OpenCL内核。它最适合于中高端独立GPU的台式机，如AMD 7970或更高版本或   
NVIDIA GTX 680或更高版本。   
一个简单的exampleOpenCL的，默认情况下在示例浏览器禁用。如果你有正确的GPU   
硬件和最新的最新的OpenCL驱动程序/编译器，你可以使用下面的命令行选项：   
–enable\_experimental\_opencl   
需要注意的是有很多原因的OpenCL内核失败，你将需要熟悉   
与OpenCL的处理这些问题。   
请参见单独的PDF文档中关于OpenCL的collisiondetection更多的背景   
和rigid body管道的bullet/docs文件夹。还有关于OpenCL的刚性书章   
身体管道通过CRC出版社“在视觉效果多线程”的一部分。这本书也可   
从亚马逊。

## 15 更多文档和参考

在线资源

在参观物理学的bullet网站[http://bulletphysics.org](http://bulletphysics.org/" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)的讨论论坛，一个wiki   
常见的是最新版本的问题，并提示和下载。维基百科页面   
列出了使用bullet在一些游戏和电影   
[http://en.wikipedia.org/wiki/Bullet\_(software](http://en.wikipedia.org/wiki/Bullet_(software" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)）   
创作工具   
•maya动感中的插件，bulletCOLLADA物理支撑位   
[http://dynamica.googlecode.com](http://dynamica.googlecode.com/" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)   
•搅拌机3D建模包括bullet和COLLADA物理支持：   
[http://www.blender.org](http://www.blender.org/" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)   
•COLLADA物理学标准：[http://www.khronos.org/collada](http://www.khronos.org/collada" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)   
图书

•实时collisiondetection，克里斯特埃里克松   
[http://www.realtimecollisiondetection.net/](http://www.realtimecollisiondetection.net/" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)   
bullet用来GJK所讨论的维诺单纯求解   
•collision detection的交互式3D环境，吉诺•范登卑尔根   
[http://www.dtecta.com](http://www.dtecta.com/" \t "http://blog.csdn.net/mybbs2200/article/details/_blank) 也为固体collisiondetection库网站   
讨论GJK等算法，了解是非常有用的bullet   
•基于物理的动画，肯尼Erleben   
[http://www.diku.dk/~kenny/](http://www.diku.dk/~kenny/" \t "http://blog.csdn.net/mybbs2200/article/details/_blank)   
非常有用的理解bullet动力和constraints   
•多线程的视觉效果   
讨论了bullet物理OpenCL的rigid body工作。

捐款和人   
Bullet物理库下的合作有许多专业的游戏开发活跃   
开发商，电影制片厂，以及学术界，学生和爱好者。   
主要作者和项目负责人是欧文•库曼斯，谁在索尼电脑项目的开工   
娱乐美国公司美国R＆D，然后在AdvancedMicro Devices公司，现在在谷歌。   
有些人认为贡献源代码，bullet：   
罗曼•波诺马廖夫，SCEA，constraints，CUDA和OpenCL研究   
约翰•麦卡彻，SCEA，光线投射，角色控制，几项改进   
纳森尔•普雷斯森，Havok的：bullet柔体动力学和EPA的最初作者   
吉诺•范登卑尔根，Dtecta：LinearMath类，各种collisiondetection思路   
克里斯特埃里克森，SCEA：维诺单纯求解   
菲尔•奈特，迪斯尼雪崩工作室：多平台兼容性，BVH系列化   
奥莱Kniemeyer，Maxon公司：各种常规补丁，btConvexHullComputer   
西蒙•霍布斯，SCEE：3D轴扫描和修剪和btPolyhedralContactClipping的部分   
皮尔•丁曼，NVIDIA：各项工作与分离轴测试，扫描和修剪   
德克•格雷戈里厄斯，因素5：讨论和协助约束   
埃里•卡托，暴雪：在连续的冲动累积冲动   
弗朗西斯科•利昂：GIMPACT凸凹凹的碰撞   
埃里克日照：果酱+ msvcgen构建系统（因为bullet2.76cmake的取代）   
史蒂夫•贝克：GPU物理和通用实施的改进   
杰伊•李，TrionWorld：双精度支持   
KleMiX，又名弗谢沃洛德Klementjev，托管版本，C＃端口XNA   
貂Svanfeldt，星风：平行constraints求解器和其他改进和优化   
马库斯•亨尼克斯，星风：btConeTwistConstaint等。   
阿瑟•谢克，尼古拉Candussi，劳伦斯•查，迪士尼动画：maya动感中的插件   
越来越多的人已经到bullet贡献，感谢大家对bullet论坛。