\*创建者：履霜坚冰  @7 a\* v0 U  x" w  
\*说明：简单地英译汉，更多细节还望大家补充，如果愿意，请留下你的论坛ID，方便咨询9 k+ `# E5 C, }\* ^. m  
----  
4 g" U4 X0 G, b4 j6 b. R4 L7 W. N2 H  
本文还可以在：WIKI-  首页-->关于Irrlicht --->第三方Irrlicht例子 中找到  
\*原文地址:[http://www.irrlicht3d.org/wiki/i ... cCollisionDetection](http://www.irrlicht3d.org/wiki/index.php?n=Main.BasicCollisionDetection). ]( O% z4 [7 \_$ r0 R  
牛顿物理引擎：<http://www.newtondynamics.com/>  
  
这个教程教我们使用牛顿物理引擎在Irrlicht中进行两个物体间简单的碰撞检测   
\*首先，我们需要把Irrlicht引擎和Newton引擎以及它们的命名空间添加到我们的项目中. v- A4 x( \_5 B" I( j  
----  c, \_1 S" d2 E$ ^( }2 U S  
#include <irrlicht.h>  
#include <newton.h>0 Q9 J4 B C' }8 I5 z  
  
#ifdef WIN32  
#pragma comment(lib,"irrlicht.lib")  
#pragma comment(lib,"newton.lib")  
#endif/ o- Y8 u5 U  r9 d  
" k% p: {! h( \_) {  B1 h  
using namespace irr;" L\* G  |5 \_) ?( a- k2 V  
using namespace core;\* T7 e6 X2 B3 h  
using namespace scene;  
using namespace video;   
----  
\*不要忘了将Newton.dll和Irrlicht.dll拷贝到你的工程目录下。' H: t, W9 /( }! I: r8 t\* @) /( /" s' o  
  
  
\*我们需要一个结构体用来包含 IMeshSceneNode和NewtonCollision. NewtonCollision包含了碰撞的数学运算并且能够对Irrlicht中的两个IMesh进行比较.但它不含任何的位置,旋转和缩放信息.为了使用这些我们需要一个NewtonBody ,它在Newton引擎中就和Irrlicht中的IMeshSceneNode 一样,但是我们不在这里使用他们.  
  
  
\*用于碰撞检测的结构体. g( h3 y0 Y1 e8 w7 \_3 F  
----  
struct SObject0 [: ]2 x$ [8 ~  
{ 1 x5 O: j3 o" z- y  
\*Irrlicht的场景节点$ w+ i2 c" g. Z  i\* c. b' P\* B  
IMeshSceneNode \*irr\_node;  
% |, N8 g  t O7 m6 `0 O0 O  
6 Y! p- |7 q4 {. d! L  
//Newton 的碰撞对象  
NewtonCollision \*nwtn\_collision; " V5 T6 ]. v1 S: A. `  
}; ( A- U6 q- F! u  
----  
  
\*现在,我们需要写一个用于从IMesh对象中创建一个NewtonCollision 对象的函数.为了实现这个目标,我们只需要从IMesh中拷贝所有的顶点信息到我们的碰撞对象中即可.为了创建NewtonCollisions 我们还需要传一个指针给一个NewtonWorld对象,我们使用Newton引擎的每一样东西,都需要一个NewtonWorld.因为它描述了对象所存储的位置,它可以通过简单地调用NewtonCreate(NULL,NULL)函数创建.3 U9 J2 e" U- {+ ?4 c  
  
//从irrlicht mesh中创建NewtonCollision的函数  
----; D9 C\* Q: ^# U; W  
NewtonCollision \*CreateCollisionFromMesh(NewtonWorld \*nWorld, IMesh \*irr\_mesh )8 h9 ^2 j8 a! y8 |1 h, A- b2 L  
{/ T- B8 ^0 H\* /( G7 k  
\*//Mesh Buffer数量  
int nMeshBuffer = 0;  
IMeshBuffer \*mesh\_buffer = NULL;' Z/ [5 ^" [. O' o  
\*//用于存储顶点的数组  
float \*vertices;   
  
u32 nVertices = 0;! l0 n6 ]$ `/ /  
  
\*//取得顶点数量  
for( nMeshBuffer=0 ; nMeshBuffer < irr\_mesh->getMeshBufferCount() ; nMeshBuffer++ )  
{   
nVertices += irr\_mesh->getMeshBuffer(nMeshBuffer)->getVertexCount(); ' q9 b! i! H+ W  
}  
  
1 }9 k1 I' ~4 K5 [  x: `8 ?  
5 H2 A; a( c( g# W$ h" i5 p L  
\*//创建一个顶点缓冲  C8 ?8 s9 D+ ~8 Z' b+ F  
vertices = new float[nVertices \* 3];. B; V `; |% B; L1 f  
u32 tmpCounter = 0;6 Z0 v# g# N3 X7 \_6 \_9 {  
  
% Y/ p; n# k6 G6 e  
\*//取得(irr\_)mesh buffers并拷贝面顶点% ]9 X' ]# h4 O" I1 Y) s P0 {  
for( nMeshBuffer=0 ; nMeshBuffer < irr\_mesh->getMeshBufferCount() ; nMeshBuffer++ )  
{   
  
mesh\_buffer = irr\_mesh->getMeshBuffer(nMeshBuffer);  
  
  
\*//取得顶点和索引的指针  
S3DVertex \*S3vertices = (S3DVertex\*)mesh\_buffer->getVertices();  
~( l. m. ?' w6 Z7 J- V9 B. {  
\* N, / Q" g/ I6 n  
7 c) t' p, W! I1 ]/ X. J  
\*//从MESH拷贝顶点到BUFFER  
for(int i=0; i<mesh\_buffer->getVertexCount(); i++)  
{ 7 a" d, N4 Z1 b! j; m) {  
/ d' f! `\* e6 j3 a: Z5 O' w  
vertices[tmpCounter++] = S3vertices.Pos.X;' }: T% r\* n! ~7 r5 h h; n! T. ]& dvertices[tmpCounter++] = S3vertices.Pos.Y;1 T+ ^' @4 ~4 H5 yvertices[tmpCounter++] = S3vertices.Pos.Z;4 ]) z% X2 u Z: v; J}# z# Z7 x# @8 z# B# h( d}0 F# w i0 R$ b7 S; o' j\*//创建一个Newton碰撞对象! o" Y# r/ W, oNewtonCollision \*collision\_obj = NewtonCreateConvexHull(nWorld,nVertices,vertices,sizeof(float)\*3,NULL);\*//删除顶点4 E7 i: C4 r# O3 |6 ldelete [] vertices;{ return collision\_obj; }6 R8 g% g0 ]8 Z0 V9 q- I$ {----\*NewtonCreateConvexHull创建一个动态的NewtonCollision对象.我们也可以用NewtonCreateTreeCollision函数创建树优化(tree optimized )的NewtonCollisions对象.优化的对象对大模型来说是很好的,因为它们会更快.不同于前一个函数,NewtonCollisions 被创建为一个静态对象,并且我们不能对两个树优化的对象作出碰撞检测反应.. D) c! o7 S o2 s: `3 w\*为了创建一个树优化NewtonCollision 对象,我们需要拷贝所以的IMESH面,而不是顶点.  m" p6 /! j\* h2 g) Q9 }5 y\* c, H- Z9 C8 Y( ^\*用于创建树优化NewtonCollision对象的函数.! ^6 e/ ?- A' F: b-----NewtonCollision \*CreateTreeCollisionFromMesh(NewtonWorld \*nWorld, IMesh \*irr\_mesh )- D8 k  /, z8 h; Z( X# K \_{8 N' Y. s/ G\* |! r\*创建一个新的树优化MESHNewtonCollision \*collision\_obj = NewtonCreateTreeCollision(nWorld ,NULL);3 ?0 @; ?  W+ N( {6 b, T\* F+ U2 t\*//开始碰撞模型的构造NewtonTreeCollisionBeginBuild(collision\_obj);$ w\* W3 N; w, e- J\*// 模型BUFFER数量7 m) p# m/ ^( h: q, ^# Yint nMeshBuffer = 0;. o( S, `  J6 ?: B; A\*//  顶点索引int v\_index[3] = {0,0,0};0 j6 ~5 ^; W- bIMeshBuffer \*mesh\_buffer = NULL;\*//数组用于存放3顶点0 I2 f) z3 m" j! |7 D l( E4 Lfloat array[9];7 M% D) e$ w: S, j# O! t+ M; b) y\* A  C\* ?0 s% ~$ K\*取得(irr\_)mesh buffers并拷贝三角面到碰撞模型for( nMeshBuffer=0 ; nMeshBuffer < irr\_mesh->getMeshBufferCount() ; nMeshBuffer++ ); x6 c- ~5 P0 O  N. L{8 P0 R( I\* k1 u' f3 I+ |9 g5 cmesh\_buffer = irr\_mesh->getMeshBuffer(nMeshBuffer);$ u# d. z; U a0 G2 D/ M, m" P$ Z; U\*取得顶点和索引指针S3DVertex \*vertices = (S3DVertex\*)mesh\_buffer->getVertices();u16 \*indices = mesh\_buffer->getIndices();2 y0 e$ C  u\* P# u- Y6 C. f( c: p) k8 b6 S- j\*填充碰撞模型( }8 a) o; b2 ^3 }. w, a9 Rfor(int i=0; i<mesh\_buffer->getIndexCount(); i+=3)2 B: L" {) I! g/ X4 s{v\_index[0] = indices[ i ];v\_index[1] = indices[i+1];! @/ x1 R0 Y) n5 D/ }. xv\_index[2] = indices[i+2];, m2 o m+ a) F; J# \_4 `2 m\*第一个顶点; c' S1 j( w7 ]array[0] = vertices[ v\_index[0] ].Pos.X;array[1] = vertices[ v\_index[0] ].Pos.Y;/ f9 c9 K) E4 I7 g/ @; oarray[2] = vertices[ v\_index[0] ].Pos.Z;\* G9 S: G: j9 ?9 d\*第二个顶点+ ^5 x5 U' \_$ l5 F1 Oarray[3] = vertices[ v\_index[1] ].Pos.X;array[4] = vertices[ v\_index[1] ].Pos.Y;+ @3 Y) J1 h1 L  c  [+ s) U, karray[5] = vertices[ v\_index[1] ].Pos.Z;\*第三个顶点5 w7 r8 j8 z Zarray[6] = vertices[ v\_index[2] ].Pos.X;H6 K2 Q' L. i- @% h, a. ?array[7] = vertices[ v\_index[2] ].Pos.Y;/ q, ]3 n3 ]" z: j1 w( larray[8] = vertices[ v\_index[2] ].Pos.Z;\*增加新的面到碰撞模型/ U8 v# G/ K! /  q$ ONewtonTreeCollisionAddFace(collision\_obj, //将要增加面的碰撞模型3, //数组中的顶点数量(float\*)array, //顶点数组指针3\*sizeof(float),//顶点大小1); //面的ID! h# ]1 M- K! [; n! `# n\* n}\* k/ Q" N$ ^! P0 T3 V; \_# E}5 q8 A8 o: f5 N: [$ b5 ]\*结束碰撞模型的构造,*设置*用于优化的第1个和第2个参数) |" a+ L4 m, s2 K1 ~" pNewtonTreeCollisionEndBuild(collision\_obj,0);' ` }5 y3 p% X  \_& n7 b+ w/ W' Q6 t) r\* f7 f }) I( Qreturn collision\_obj;+ j9 ]# q4 H, e6 V0 i1 u. q}8 u7 o0 s, Q5 j+ \_2 e----' w3 P/ t\* e" w\*前面的两个函数可以改良.通过调用IMeshBuffer::getVertexType()来确保我们使用了正确的数据和标准的顶点格式.我们需要的最后一个函数是CheckForCollision.它将决定我们的SObjects 是否横穿其它物体.如果它横穿了,返回TRUE.为了检测两个NewtonCollisions 对象间的碰撞,我们使用NewtonCollisionCollide. T: m5 [- W% s\*检测两个SObject的碰撞----bool CheckForCollision(NewtonWorld \*nWorld, SObject \*obj\_A, SObject \*obj\_B)$ J# t' ^. P2 x1 c1 C" |/ u{% k) ?9 w\* Y8 `/ k% S# w//Matrix to store irr\_node position\*存储irr\_node位置的矩阵matrix4 mat\_A,mat\_B;/ D  I2 n) N0 A( t$ ]$ ~  `1 m0 K7 |" E( Z5 e7 V( C2 [//Copy position\*复制位置1 t$ H7 x  u4 G  /  Lmat\_A.makeIdentity();mat\_B.makeIdentity();  u( f" y8 I; m2 W8 z9 c# z( y2 Y. v% F  bmat\_A.setTranslation(obj\_A->irr\_node->getPosition());mat\_B.setTranslation(obj\_B->irr\_node->getPosition());  P: I4 ~9 W- f% l; ?8 i/ econst int nContacts = 2;: J/ I- f' k. B# z8 ufloat contacts[3 \* nContacts];. X0 \_! l4 C2 c' /) K1 Ifloat normals[3 \* nContacts];0 [; B+ G( w% i+ y! bfloat penetration[ nContacts ];+ e$ Z% p) g8 x" M3 R8 U\*检测两个碰撞模型的碰撞\*返回连接的顶点数目int nHits = NewtonCollisionCollide( nWorld,nContacts,obj\_A->nwtn\_collision, (float\*)mat\_A[0],obj\_B->nwtn\_collision, (float\*)mat\_B[0],contacts,  B) Q7 S/ /) X. \_3 {" O: m$ c8 vnormals,1 e; S6 E) l2 y0 h) Apenetration);3 E  A3 B3 V8 f: J\* /! /9 q: D8 i. Y0 {) E$ a, ~\*if nHits > 0 表示碰撞. O7 E' A ^& w- k- t\* b8 [# jif( nHits > 0)return true;return false;}----1 @8 l  F' I5 M) S' [ R+ t$ `; s! J2 t1 o4 ^\*现在我们要做的唯一事情就是创建一个IrrlichtDevice 并且初始化我们的场景,我们将要创建两个SObjects并且像往常一样加载两个模型.首先我们把其中一个将放在中间,另一个从左移到右,反之亦然.如果CheckForCollision 返回true,我们将设置它的emessive颜色成红色来表示它们的接触,最后,我们需要清除所有的东西.\*为了运行这个*程序*,就需要两个模型,我使用Blender来创建一个icosphere 和一个立方体,并导出它们成.3ds格式,你也可以使用你自已的模型.# y- M" U, k  I% i+ u4 O7 s+ A\**推荐*在碰撞检测中你使用低模.而不是节点模型.这样容易看见.这样可以加速程序的速度.. |5 Q0 |2 i, l! o) M6 O' K: P `! r! W; W----' o\* p. M; m# `5 q( |; X; @2 \_int main(){5 B\* f7 p7 h4 C3 N\*创建irrlicht 设备IrrlichtDevice \*device = createDevice(EDT\_OPENGL, dimension2d<s32>(800, 600), 16,false, false, true, 0);+ P: m4 W% e/ r; i% o! q( @: x. w6 q4 C\*创建newton world. k3 A8 {# K; P3 ZNewtonWorld \*nWorld = NewtonCreate(NULL,NULL);$ W8 w" R\* P3 `$ m( B4 a; D Q: x0 k; d\*设置0用于准确的碰撞检测\*1到n用于加速程序NewtonSetSolverModel(nWorld,1);8 Q\* \_ h) \_2 M" /, f- ]) Q+ l9 /( b\*设置WINDOWS标题,注意这里使用了'L',Irrlicht引擎在显示文本的时候使用宽字符5 h3 @% |5 v- E0 h/ f4 [: i8 e. }device->setWindowCaption(L"Basic Collision Detection with Newton GD");  I\* }. A5 D7 }8 L8 j9 v1 b% e/ `4 O\*取得一个图形设备的指针,场景管理器和GUI,以至于我们不用总是写\*device->getVideoDriver(),\*device->getSceneManager()% c/ p% p( Q' ~7 G\*device->getGUIEnvironment().( Z1 /" A! Z$ X5 CIVideoDriver\* driver = device->getVideoDriver();" B4 `3 Q' Y2 Q+ f6 H# I0 V3 uISceneManager\* smgr = device->getSceneManager();$ Y$ a2 |$ A: z//IGUIEnvironment\* guienv = device->getGUIEnvironment();M& w  t7 T7 P  t! ~2 z1 v% j+ A3 v! G# L\*创建两个SObjectsSObject sphere;' L0 b F; h8 \_; g" m% jSObject cube;\*加载模型并创建一个irrlicht 节点sphere.irr\_node = smgr->addMeshSceneNode(smgr->getMesh("sphere.3ds")->getMesh(0));- w0 W, A, E( }7 /) f' w4 ~cube.irr\_node = smgr->addMeshSceneNode(smgr->getMesh("cube.3ds")->getMesh(0));sphere.irr\_node->getMaterial(0).EmissiveColor.set(255,0,255,5);; N5 /% ^/ s7 z2 xcube.irr\_node->getMaterial(0).EmissiveColor.set(255,0,255,255);1 G$ i% h  h3 ]  m2 O z) k2 p: Y\* /; F- K\*从irrlicht 创建一个碰撞模型sphere.nwtn\_collision = CreateCollisionFromMesh(nWorld,sphere.irr\_node->getMesh());5 M! `' f x\* b+ }cube.nwtn\_collision = CreateCollisionFromMesh(nWorld,cube.irr\_node->getMesh());# I6 v( m( z! x! i q" yJ\* x7 }" B2 v+ L" R& u& q. g1 s\*设置位置sphere.irr\_node->setPosition(vector3df(-5,0,0));cube.irr\_node->setPosition(vector3df(0,0,0));' A6 // |1 W+ {. J1 C8 s3 x2 u\*增加摄相机smgr->addCameraSceneNode(0, vector3df(0,1,-7), vector3df(0,0,0));+ } Y7 H: x& g7 u8 J4 @+ L\*好了,现在我们设置好了场景,让我们绘制所有的东西.\*我们用一个while()循环,直到设备不想再运行为止.当用户关闭容器或者用\*ALT+F4关闭窗口' P: t5 o+ A4 z8 a\*定义节点要移动的位置" @3 s2 c7 d6 T4 /; |# m. Wfloat vel=0.1f;$ P1 G) `; v8 ]  V8 y0 n$ w: u8 ]9 k6 \_6 y, P, awhile(device->run())X+ j3 ^! x/ T& u; E{driver->beginScene(true, true, SColor(0,200,200,200));smgr->drawAll();: h, z. @' x( /8 w8 M5 V- Ldriver->endScene();' a: \_. ?6 O0 k8 U0 Z) X6 r" a; ^- a/ U) ^! l+ o- Y+ G" V- [. a\*让我们移动这个圆sphere.irr\_node->setPosition(sphere.irr\_node->getPosition()+vector3df(1,0,0)\*vel);if(sphere.irr\_node->getPosition().X > 5.5f)8 a2 k+ e' j3 u" J2 D4 rvel = -0.05f;! f/ T7 D% o2 b4 c3 n9 Y( i, sif(sphere.irr\_node->getPosition().X < -5.5f)vel = 0.05f;\*检测碰撞并,如果发生了就设置为红色. o! M\* b+ L# U\* G( e7 Bif(CheckForCollision(nWorld,sphere,&cube))/ p1 k5 K: T" T) ]cube.irr\_node->getMaterial(0).EmissiveColor.set(255,255,0,0);else\* u( O! }. J/ /: g\*使用蓝色cube.irr\_node->getMaterial(0).EmissiveColor.set(255,0,0,255);}0 ^+ n9 t7 J" n; K1 ]\*如果你不再需要了,那我们需要释放newton碰撞模型NewtonReleaseCollision(nWorld, sphere.nwtn\_collision);9 G, K6 |\* D$ ?- d% /. u( s; M! eNewtonReleaseCollision(nWorld, cube.nwtn\_collision);4 ?0 l# E- h- x) b$ B\*销毁newton world1 n5 K+ \_/ E\* Z# }( j6 q; b! QNewtonDestroy(nWorld);. z5 u3 {$ D6 m' m/ H: Pdevice->drop();return 0;}$ t9 q# q' z+ w7 M$ {$ z- Y1 v0 R- Y( @9 ^5 @----. W% o. F: q1 Q\* F/ q\*NewtonReleaseCollision 销毁NewtonCollision 对象(类似于Irrlicht中的drop()).如果你不再需要NewtonCollision 对象,你需要调用它.\*NewtonDestroy 销毁一个NewtonWorld.如果我们不再需要物理*系统*,则要调用他.\*更多信息参见NewtonSDK文档原文:Xwan/ \_$ K\* c\* T% L9 c译:履霜坚冰# i$ k# U: n) r5 C