‍

/\*

      在此例中，我将会告诉大家如何使用IRR引擎进行鼠标拣选和碰撞检测。

      我将会说明3种方法：与3D世界的场景节点碰撞检测，普通的三角形拣选，以及普通的场景节点拣选

      首先，我们先打开02.Quake3Map的例子，读取一个Q3的地图，我们在这张地图上进行移动

      (即增加了与场景节点之间的碰撞检测)并且对这个地图进行其三角形的拣选(进行了鼠标拣选)。

      另外我们增加了三个模型，目的是实现我们的场景节点拣选。

      下面的简单代码我不再加注释，若不理解，请认真看例子02.Quake3Map

\*/

#include <irrlicht.h>

using namespace irr;

using namespace core;

using namespace scene;

using namespace video;

using namespace io;

using namespace gui;

#pragma comment(lib,"irrlicht.lib")

void main()

{

// 创建irr设备

IrrlichtDevice\* device=createDevice(EDT\_OPENGL,dimension2d<u32>(640,480),16,false);

ISceneManager\* smgr=device->getSceneManager();

IVideoDriver\* driver=device->getVideoDriver();

device->setWindowCaption(L"7.Collision - Lauthrul");

//读取Quake 3地形

device->getFileSystem()->addZipFileArchive("../../../media/chiropteraDM.pk3");

IAnimatedMesh\* mesh=smgr->getMesh("chiropteraDM.bsp");

ISceneNode\* qk3node=smgr->addOctTreeSceneNode(mesh->getMesh(0));

qk3node->setPosition(vector3df( 0,  - 30,  0));

/\*

      到现在为止，我们读取了一个Quake3的地形。现在，我们来增加一些与例子02不同的一些东西：

      我们创建一个三角形选择器。TriangleSelector(三角形选择器)是一个类对象，它能够从

      场景节点中拣选出其中的三角形。

      拿碰撞检测来说吧，实际上就是有很多被场景管理器创建出的三角形选择器共同实现的。

      在本例中，我们创建了一个八叉树三角型选择器，它在巨大的场景中常被使用，能够

      有效又高效的进行碰撞检测。

      在我们创建过三角型选择器之后，我们把它绑定在一个Q3场景节点上。当然我们也可以不进行

      绑定，但是若绑定后，我们就可以不再对其进行复杂的管理了，例如，它会随场景一起释放掉，

      否则，我们就需要手动drop掉。

      \*/

//三角形选择器

ITriangleSelector\* selector=smgr->createOctreeTriangleSelector(mesh->getMesh(0),qk3node);

qk3node->setTriangleSelector(selector);

selector->drop();

/\*

      我们象例子02一样在场景中增加了一个FPS类型的摄象机。但是，我们这次为摄象机增加了一个

      Animator动画器：使它成为一个能碰撞检测的摄象机。这样的话摄象机就会与场景节点做一些处理，

      使它不会穿越墙壁和地表。

      想做这样的一个摄象机，我们仅需要告诉Animator动画器一些场景节点的属性，这个场景节点多大，

      重力多大，设备完毕后再将Animator动画器与摄象机绑定就OK了，不需要再做其他的。一切都由

      引擎做完，在这之下的代码目的仅仅是进行拣选的代码而已了。

      请注意另一个很酷的特性：这个碰撞检测动画器能够与非摄象机的其他的所有场景节点进行绑定。

      同样，一个摄象机上也允许绑定多个动画器。正因为这个特性的寻在，碰撞检测在Irr引擎中的实现

      是非常容易做到的。

      现在我们介绍一下createCollisionResponseAnimator()函数的各个参数。

      第一个参数是已经指定了三角型选择器。第二个参数是进行碰撞检测的场景节点，在我们这个例子里面，

      这个需要碰撞检测的场景节点是摄象机对象。第三个参数是进行检测的对象的大小，它是一个椭球状

      的包围体，我们可以设置它的三围。第四个参数是重力的方向和强度。你若是设置为0,0,0则代表无重力。

      最后一个参数是一个偏移：默认的时候摄象机是在椭球包围体中间，但是人类眼睛的话，是在身体上方，

      所以我们应当设置一个偏移量，使摄象机处于包围体的上方。现在就OK了，碰撞检测设置OK了。

      \*/

//碰撞检测的摄像机，最后一个参数设为true，作用是禁止垂直方向的移动

ICameraSceneNode\* camera=smgr->addCameraSceneNodeFPS(0,100,0.5,-1,0,0,true);

camera->setPosition(vector3df( -50, 0,-250));

//设置碰撞检测参数

//第一个参数是已经指定了的三角型选择器。

//第二个参数是进行碰撞检测的场景节点，这里是摄象机对象。

//第三个参数是进行检测的对象的大小，它是一个椭球状的包围体，可以设置它的三围。

//第四个参数是重力的方向和强度。若设置为(0, 0, 0)，则代表无重力。

//最后一个参数是一个偏移：默认时摄象机在椭球包围体中间，但人类眼睛在身体上方，所以应当设置一个偏移量，使摄象机处于包围体的上方。

ISceneNodeAnimator\* anim=smgr->createCollisionResponseAnimator(selector,camera,vector3df(30,60,30),vector3df(0,0,0),vector3df(0,0,0));

camera->addAnimator(anim);

anim->drop();

//屏蔽鼠标指针

device->getCursorControl()->setVisible(false);

// 增加一个公告版场景节点

IBillboardSceneNode \* board=smgr->addBillboardSceneNode();

board->setMaterialType(EMT\_TRANSPARENT\_ADD\_COLOR);

board->setMaterialTexture(0,driver->getTexture("../../../media/particle.bmp"));

board->setMaterialFlag(EMF\_LIGHTING,false);

board->setMaterialFlag(EMF\_ZBUFFER,false);

board->setSize(dimension2d < f32 > (20.0f, 20.0f));

// 增加了三个仙子角色

SMaterial material;

material.setTexture(0,driver->getTexture("../../../media/faerie2.bmp"));

material.Lighting=true;//开启光照（后面增加了一个光源）

IAnimatedMesh\* faerie=smgr->getMesh("../../../media/faerie.md2");

//仙子一

IAnimatedMeshSceneNode\* node=smgr->addAnimatedMeshSceneNode(faerie);

node->setPosition(vector3df( -150, 0,  -200));

node->setMD2Animation(EMAT\_RUN);//跑动

node->getMaterial(0) = material;

//仙子二

node = smgr->addAnimatedMeshSceneNode(faerie);

node->setPosition(vector3df( -150, 0,  -170));

node->setMD2Animation(EMAT\_SALUTE);//敬礼

node->getMaterial(0) = material;

//仙子三

node = smgr->addAnimatedMeshSceneNode(faerie);

node->setPosition(vector3df(-150, 0,  -140));

node->setMD2Animation(EMAT\_JUMP);//跳跃

node->getMaterial(0) = material;

//设置光源

smgr->addLightSceneNode(0,vector3df(-150, 0,  0),SColorf(1,1,1),500,-1);

/\*

      为了不使例子变的太复杂，我在绘制循环中进行鼠标拣选。

      另外我们创建了两个指针分别用来存储：当前的场景节点和最后选择的场景节点。

      那么，我们开始循环。

      \*/

ISceneNode \*selectedSceneNode = 0;

ISceneNode \*lastSelectedSceneNode = 0;

while(device->run())

{

if(device->isWindowActive())

{

driver->beginScene();

smgr->drawAll();

/\*

在我们使用场景管理器DrawAll()了之后，我们开始做第一个鼠标拣选：我们们想知道我们鼠标

在当前世界中选择了哪一个三角面。另外，我们想知道我们鼠标当前位置。

那么，我们创建一个3D的拣选线，它从我们的摄象机出发到我们的摄象机观察点。然后我们

告诉碰撞管理器，如果我们这条线和场景中的一个三角面碰撞了，那么我们就描绘这个三角面，

并且设置我们的公告版位置在交点处。（此时的公告版位置与鼠标类似，然而会多一个深度概念）

\*/

//基于三角面的拣选

//所有的交点都是由摄像机投射到距离1000f远的位置完成的

//你可以据此轻易地修改程序，用于如子弹轨道、刀剑的位置的检测

//或者用 ISceneCollisionManager::getRayFromScreenCoordinates()

//创建一条从摄像机到鼠标点击点的拣选线

line3df line;

line.start=camera->getPosition();

line.end=line.start + (camera->getTarget()-line.start).normalize()\*1000.0f;

vector3df collisionPoint;//跟踪目前的交叉点

triangle3df tri;//保存选中的三角形

//getSceneNodeAndCollisionPointFromRay函数返回距拣选线起点最近的节点

//如果没有检测到碰撞，返回0

selectedSceneNode=smgr->getSceneCollisionManager()->getSceneNodeAndCollisionPointFromRay(line, collisionPoint, tri);

if(selectedSceneNode)

{

board->setPosition(collisionPoint);

//在做渲染之前，我们需要做一下矩阵变换操作

driver->setTransform(ETS\_WORLD, matrix4());//世界矩阵变换

material.setTexture(0,0);

material.Wireframe=true;

material.Lighting = false;

driver->setMaterial(material);

driver->draw3DTriangle(tri,SColor(100,255,255,0));

}

//我们可以检测标志位看是否节点被选中需要凸显，

//动画节点可以凸显，但地图不需要。

//在这种情况下，凸显出节点即为把这个节点的照明关闭。

//这意味着将与全亮度绘制。

selectedSceneNode = smgr->getSceneCollisionManager()->getSceneNodeFromCameraBB(camera);

if (lastSelectedSceneNode)

  lastSelectedSceneNode->setMaterialFlag(EMF\_LIGHTING, true);

if (selectedSceneNode == qk3node || selectedSceneNode == board)

  selectedSceneNode = 0;

if (selectedSceneNode)

  selectedSceneNode->setMaterialFlag(EMF\_LIGHTING, false);

lastSelectedSceneNode = selectedSceneNode;

driver->endScene();

}

}

driver->drop();

}