[**【Irrlicht Engine笔记】test7-Collision**](http://www.cnblogs.com/ducky/archive/2010/10/29/1864600.html)

/\*\* 例7 Collision

我们在这里会解释两个函数:   
自动碰撞检测：通过爬楼梯或滑动在三维世界中移动。  
手动检测节点并用光纤选择一个三角形。  
在这个例子中，我们将从摄像机中发射光速，当然你可以任意选择。

开始部分，我们从例2中借鉴了一些代码来载入quake3地图。  
我们将在这个地图上水平移动行走，并选择三角形。  
此外，我们设置一个动画器来进行三角形采集。  
下列代码开启引擎及载入模型。  
\*/  
#include <irrlicht.h>  
#include "driverChoice.h"

using namespace irr;

#ifdef \_MSC\_VER  
#pragma comment(lib, "Irrlicht.lib")  
#endif

enum  
{  
//我用这个ISceneNode ID来表示场景节点，而并非由getSceneNodeAndCollisionPointFromRay()  
ID\_IsNotPickable = 0,

//我用ISceneNode ID中的这个标志位标识该场景节点说明他可以被光线选择。  
IDFlag\_IsPickable = 1 << 0,

// 我用ISceneNode ID中的这个标志位标识该场景节点说明此节点可高亮。  
//在这个例子中，人类模型可以高亮，而地图模型不能。  
IDFlag\_IsHighlightable = 1 << 1  
};

int main()  
{  
// 询问用户  
video::E\_DRIVER\_TYPE driverType=driverChoiceConsole();  
if (driverType==video::EDT\_COUNT)  
   return 1;

//创建设备

IrrlichtDevice \*device =  
   createDevice(driverType, core::dimension2d<u32>(640, 480), 16, false);

if (device == 0)  
   return 1; // 创建失败

//获取各管理器指针  
video::IVideoDriver\* driver = device->getVideoDriver();  
scene::ISceneManager\* smgr = device->getSceneManager();

//载入地图模型：获取文件系统指针在来载入http://www.cnblogs.com/media/map-20kdm2.pk3  
//    PK3 文件简单地是一个改名的.ZIP文件。  
//    它通常包含一检查和防止文件的开头和再压或者额外地图的加法到演示版本。  
//    PK3文件包含地图、纹理、武器、声音和剧本等。  
device->getFileSystem()->addZipFileArchive("http://www.cnblogs.com/media/map-20kdm2.pk3");

//载入贴图  
//此时系统会自动在http://www.cnblogs.com/media/map-20kdm2.pk3内找到该文件  
scene::IAnimatedMesh\* q3levelmesh = smgr->getMesh("20kdm2.bsp");  
scene::IMeshSceneNode\* q3node = 0;

// 在这里载入模型，但没有设置光线  
/\*  
virtual ISceneNode\* irr::scene::ISceneManager::addOctTreeSceneNode ( IMesh \* mesh,   
                                                                          ISceneNode \* parent = 0,   
                                                                          s32 id = -1,   
                                                                          s32 minimalPolysPerNode = 256,   
                                                                          bool alsoAddIfMeshPointerZero = false     
                                                                         )

\*/  
if (q3levelmesh)  
   q3node = smgr->addOctreeSceneNode(q3levelmesh->getMesh(0), 0, IDFlag\_IsPickable);

/\*  
到目前为止都很好，我们已经装入了像教程2一样载入了quake3模型。  
现在会有些不同了。我们创建一个三角形选择器。  
三角形选择器是一个类，可以用来在场景中选择三角形来做不同的操作。  
三角形选择器有多种类型，均可由ISceneManager创建。  
在这个例子中，我们创建一个八叉树三角形选择器，  
它利用建立八叉树来维护场景内的三角形来减少输出。  
这对像Quake 3级别的大型地图来说是非常有必要的。  
在我们创建了三角形选择器之后，我们将它附加到quake3节点上。  
这并不是必须的，但这样做的话，我们不需要为选择器额外做什么了，  
例如我们不需要它，可以丢弃它。  
\*/

scene::ITriangleSelector\* selector = 0;

if (q3node)  
{  
   q3node->setPosition(core::vector3df(-1350,-130,-1400));

   selector = smgr->createOctreeTriangleSelector(  
     q3node->getMesh(), q3node, 128);  
   q3node->setTriangleSelector(selector);  
   //我们对选择器的操作还没有结束，所以不要在这里丢弃它。  
}

/\*  
在例2中我们向节点中添加一个始终指向个人的摄像机，这样我们可以观察quake 3地图中人物的移动。  
但这次我们像摄像机添加一个特别的动画：碰撞检测动画。  
这个动画可以让添加它的节点在移动的时候不避免穿墙等行为，并为节点添加了动力。  
我们唯一要告诉动画的是我们希望的虚拟世界是什么样子，有多大的场景节点，多少重力要提供等。  
碰撞响应动画添加后，我们不需要做其他事情，系统会自动提供。  
下面是其余部分的碰撞检测代码。  
并请注意另一个很酷的功能：碰撞响应动画也可以附加到所有其他节点的场景，不仅相机。  
也可以和其他动画混合使用。这样，碰撞检测和响应，在Irrlicht引擎是很容易。

现在我们来看看createCollisionResponseAnimator()的具体参数。  
createCollisionResponseAnimator ( ITriangleSelector \* world,   
                                    ISceneNode \* sceneNode,   
                                    const core::vector3df & ellipsoidRadius = core::vector3df(30, 60, 30),   
                                    const core::vector3df & gravityPerSecond = core::vector3df(0,-10.0f, 0),   
                                    const core::vector3df & ellipsoidTranslation = core::vector3df(0, 0, 0),   
                                    f32 slidingValue = 0.0005f     
                                    )

                      ITriangleSelector \* world:是三角选择器，它指定世界如何完成碰撞检测。  
                                     from API:三角选择器掌控世界中所有可能碰撞的三角形。  
                      可通过ISceneManager::createTriangleSelector()创建。

                ISceneNode \* sceneNode:是场景节点，它是由碰撞检测影响的对象，在我们的情况下，它是相机。  
                                     from API:SceneNode将被操控。在我们为节点添加动画器后，节点将不允许穿墙并受重力影响。

        const core::vector3df & ellipsoidRadius:定义的对象是多大，它是一个椭球的半径。  
                                              如果你尝试将它减小，可在移动时更靠近墙壁。  
                                     from API:默认为core::vector3df(0,-10.0f, 0).  
                      椭球半径用于完成碰撞检测。  
                      如果你拥有一个场景节点，您不确定多大的半径应，  
                你可以使用下面的代码，以确保它正常运行：  
                    const core::aabbox3d<f32>& box = yourSceneNode->getBoundingBox();  
                    core::vector3df radius = box.MaxEdge - box.getCenter();

    const core::vector3df & gravityPerSecond:接下来的参数是重力的方向和速度。  
                                              我们将其设置为（0，-10，0），它接近于现实的重力，假设我们的单位是米。  
             你可以将它设置为（0,0,0）禁用的重力。  
           from API:设置重力环境作为加速，单位/(s\*s)。  
                   如果你的单位，相当于米，则core::vector3df(0,-10.0f,0)将给予约现实的重力。  
                   可以通过它设置为core::vector3df(0,0,0)来禁用重力。

   const core::vector3df & ellipsoidTranslation:如果没有这一点，与它进行碰撞检测是将围绕椭圆相机，摄像机将在椭球的中间。  
                                                 但作为人类，我们习惯眼睛在身体上方，  
             来检测我们与世界产生的碰撞，而不是在它的中间。  
             因此，我们把场景节点向上移了50个单位。  
      
就是这样，碰撞检测开始咯。  
\*/  
    //当重力为(0, -10, 0)时可设置jumpSpeed为3.f以产生一个较真实的跳跃。  
/\*                              ^在移动时因重力产生的场景节点下落时需要jumpSpeed来确定下落速度  
     irr::scene::ISceneManager::addCameraSceneNodeFPS ( ISceneNode \* parent = 0,   
                                                            f32 rotateSpeed = 100.0f,   
                                                            f32 moveSpeed = .5f,   
                                                            s32 id = -1,   
                                                            SKeyMap \* keyMapArray = 0,   
                                                            s32 keyMapSize = 0,   
                                                            bool noVerticalMovement = false,   
                                                            f32 jumpSpeed = 0.f     
                                                            )

\*/  
scene::ICameraSceneNode\* camera =  
   smgr->addCameraSceneNodeFPS(0, 100.0f, .3f, ID\_IsNotPickable, 0, 0, true, 3.f);  
camera->setPosition(core::vector3df(50,50,-60));  
camera->setTarget(core::vector3df(-70,30,-60));

if (selector)  
{  
   scene::ISceneNodeAnimator\* anim = smgr->createCollisionResponseAnimator(  
    selector, camera, core::vector3df(30,50,30),  
    core::vector3df(0,-10,0), core::vector3df(0,30,0));  
   selector->drop(); //selector一旦被赋予某节点，我们即可丢弃它。  
   camera->addAnimator(anim);  
   anim->drop(); //同样，动画器也是用完即弃。  
}

/\*  
现在我们创建三个可挑选的动画角色，一个动态灯光来照亮他们，  
还有一个广告牌可以用来表示我们所选择的三角形。  
首先，我们给鼠标赋予红色，并用广告牌来显示让我们可以看到它。  
\*/  
device->getCursorControl()->setVisible(false);

//添加广告牌  
scene::IBillboardSceneNode \* bill = smgr->addBillboardSceneNode();  
bill->setMaterialType(video::EMT\_TRANSPARENT\_ADD\_COLOR );  
bill->setMaterialTexture(0, driver->getTexture("http://www.cnblogs.com/media/particle.bmp"));  
bill->setMaterialFlag(video::EMF\_LIGHTING, false);  
bill->setMaterialFlag(video::EMF\_ZBUFFER, false);  
bill->setSize(core::dimension2d<f32>(20.0f, 20.0f));  
bill->setID(ID\_IsNotPickable); // This ensures that we don't accidentally ray-pick it

/\*   
给三个人物添加动画器，我们可以通过光线选择他们的三角形。  
他们都缓慢的变化一边看清三角形的选择。  
\*/  
scene::IAnimatedMeshSceneNode\* node = 0;

//添加MD2节点，它使用点缓冲动画。(vertex-based animation"?)  
node = smgr->addAnimatedMeshSceneNode(smgr->getMesh("http://www.cnblogs.com/media/faerie.md2"),  
       0, IDFlag\_IsPickable | IDFlag\_IsHighlightable);  
node->setPosition(core::vector3df(-70,-15,-120));  
node->setScale(core::vector3df(2, 2, 2));  
node->setMD2Animation(scene::EMAT\_POINT);  
node->setAnimationSpeed(20.f);  
video::SMaterial material;  
material.setTexture(0, driver->getTexture("http://www.cnblogs.com/media/faerie2.bmp"));  
material.Lighting = true;  
material.NormalizeNormals = true;  
node->getMaterial(0) = material;

//现在为他创建一个三角形选择器。选择器应该知道被关联到哪个节点，并可以自行更新。  
selector = smgr->createTriangleSelector(node);  
node->setTriangleSelector(selector);  
selector->drop(); // 选择器使用完后可丢弃。

//这种.x文件包含骨骼动画，但不包含蒙皮。  
node = smgr->addAnimatedMeshSceneNode(smgr->getMesh("http://www.cnblogs.com/media/dwarf.x"),  
       0, IDFlag\_IsPickable | IDFlag\_IsHighlightable);  
node->setPosition(core::vector3df(-70,-66,0));  
node->setRotation(core::vector3df(0,-90,0));  
node->setAnimationSpeed(20.f);  
selector = smgr->createTriangleSelector(node);  
node->setTriangleSelector(selector);  
selector->drop();

// 这B3D文件使用骨骼蒙皮动画。  
node = smgr->addAnimatedMeshSceneNode(smgr->getMesh("http://www.cnblogs.com/media/ninja.b3d"),  
       0, IDFlag\_IsPickable | IDFlag\_IsHighlightable);  
node->setScale(core::vector3df(10, 10, 10));  
node->setPosition(core::vector3df(-70,-66,-60));  
node->setRotation(core::vector3df(0,90,0));  
node->setAnimationSpeed(10.f);  
node->getMaterial(0).NormalizeNormals = true;

//同上  
selector = smgr->createTriangleSelector(node);  
node->setTriangleSelector(selector);  
selector->drop();

material.setTexture(0, 0);  
material.Lighting = false;

//添加灯光，这样为选择的节点不会完全黑暗。  
scene::ILightSceneNode \* light = smgr->addLightSceneNode(0, core::vector3df(-60,100,400),  
   video::SColorf(1.0f,1.0f,1.0f,1.0f), 600.0f);  
light->setID(ID\_IsNotPickable); // Make it an invalid target for selection.

//记录highlighted节点  
scene::ISceneNode\* highlightedSceneNode = 0;  
scene::ISceneCollisionManager\* collMan = smgr->getSceneCollisionManager();  
int lastFPS = -1;

//用线框表示出选择器选中的唯一的三角形。  
material.Wireframe=true;

while(device->run())  
if (device->isWindowActive())  
{  
   driver->beginScene(true, true, 0);  
   smgr->drawAll();

   //不照亮任何当前已经设置为highlighted的节点  
   if (highlightedSceneNode)  
   {  
    highlightedSceneNode->setMaterialFlag(video::EMF\_LIGHTING, true);  
    highlightedSceneNode = 0;  
   }

   /\*  
   在这个例子中，所有焦点的产生都从光源出发到距离其1000单位内。  
   你可以很容易的修改它来检验到这一点。  
   \*/  
   //TODO:  
   //All intersections in this example are done with a ray cast out from the camera to  
   // a distance of 1000. You can easily modify this to check (e.g.) a bullet  
   // trajectory or a sword's position, or create a ray from a mouse click position using  
   // ISceneCollisionManager::getRayFromScreenCoordinates()  
   core::line3d<f32> ray;  
   ray.start = camera->getPosition();  
   ray.end = ray.start + (camera->getTarget() - ray.start).normalize() \* 1000.0f;

   //记录当前与网格或模型碰撞的点  
   core::vector3df intersection;  
   // 用于显示被选中的三角形的边缘  
   core::triangle3df hitTriangle;

   /\*  
   这个方法的调用可以做到你需要的所有效果（如显示光线，选择的待有碰撞检测的节点中的三角形等）  
   它检测到最近的顶点或三角形，并返回包含此顶点的场景节点。  
   Irrlicht提供其他类型的选择器，包括光线/三角形选择器，光线/和模型选择器。  
   椭圆/三角形选择器，和一些附加的帮助相关方法。

   我们来看看ISceneCollisionManager方法  
   virtual ISceneNode\* irr::scene::ISceneCollisionManager::getSceneNodeAndCollisionPointFromRay   
                                                 ( core::line3df ray,  
                  core::vector3df & outCollisionPoint,  
               core::triangle3df & outTriangle,  
               s32 idBitMask = 0,  
               ISceneNode \* collisionRootNode = 0,  
               bool noDebugObjects = false  
               )   
                       core::line3df ray:用于碰撞检测的光线。  
     core::vector3df & outCollisionPoint:如果检测受到干扰，这里将存放最近一次检测到的碰撞点。  
         core::triangle3df & outTriangle:如果检测受到干扰，这里将存放最近一次检测到碰撞的三角形。  
                 s32 idBitMask idBitMask:只有节点的id匹配至少此掩码中的一位才被测试。  
                             若掩码为0，则测试所有节点。  
    ISceneNode \* collisionRootNode:需检测的所有场景节点的根节点。  
                                   若为0，默认检查整个现场。  
         bool noDebugObjects:若为true，测试debug节点（默认为false）。  
   \*/  
   scene::ISceneNode \* selectedSceneNode =  
    collMan->getSceneNodeAndCollisionPointFromRay(  
      ray,  
      intersection,  
      hitTriangle,  
      IDFlag\_IsPickable,  
      0);

   //如果光线与节点碰撞，则将广告版移动到碰撞位置显示三角形边缘。  
   if(selectedSceneNode)  
   {  
    bill->setPosition(intersection);

    //在进行渲染之前我们要进行一些矩阵变换操作。  
    driver->setTransform(video::ETS\_WORLD, core::matrix4());  
    driver->setMaterial(material);  
    driver->draw3DTriangle(hitTriangle, video::SColor(0,255,0,0));

    //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。  
    if((selectedSceneNode->getID() & IDFlag\_IsHighlightable) == IDFlag\_IsHighlightable)  
    {  
     highlightedSceneNode = selectedSceneNode;  
     //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。  
     highlightedSceneNode->setMaterialFlag(video::EMF\_LIGHTING, false);  
    }  
   }

   //我们结束了场景节点的绘制  
   driver->endScene();

   int fps = driver->getFPS();

   if (lastFPS != fps)  
   {  
    core::stringw str = L"Collision detection example - Irrlicht Engine [";  
    str += driver->getName();  
    str += "] FPS:";  
    str += fps;

    device->setWindowCaption(str.c\_str());  
    lastFPS = fps;  
   }  
}

device->drop();

return 0;  
}