一、Irrlicht引擎简介

Irrlicht引擎是一个用[C++](http://baike.baidu.com/view/824.htm)编写的高性能实时3D引擎。该引擎支持底层图形接口Direct3D和OpenGL，并且自带了软件渲染的实现。Irrlicht引擎还提供了诸如动态阴影，[粒子系统](http://baike.baidu.com/view/85873.htm)，角色动画，室内和室外技术以及碰撞检测等功能特性。

Irrlicht是一个德国神话故事中的一种动物的名字，它能够发光和飞翔，可以在大部分的沼泽地附近发现它。单词"Irrlicht"是两个德国单词（"irr"意思是疯狂的；而"Licht"意思是光）的组合。在英语中，它被译为"鬼火"。

二、Irrlicht引擎的组成结构

Irrlicht引擎共分为五部分：

1）Core

该部分由一些容器类及数学库组成，如string、vector等。

对应的名字空间为：namespace irr::core

2）Scene

该部分主要负责三维场景的绘制及管理，包括场景节点，摄像机，粒子系统、mesh 资源，公告板，灯光，动画器，天空体，地形等。

Irrlicht的场景中的所有的东西都是场景节点，统一由场景管理器来管理。

对应的名字空间为：namespace irr::scene

3）Video

该部分主要负责图片纹理的载入及管理，包括纹理，材质，灯光，图片，顶点等渲 染属性的控制。

对应的名字空间为：namespace irr::video

4）GUI

该部分包括了一些二维GUI控件

对应的名字空间为：namespace irr::gui

5）FileSystem

该部分负责文件系统的读写。

对应的名字空间为：namespace irr::io

三、Irrlicht编程步骤

1、获取设备指针

在编写任何一个Irrlicht程序时，首选需要做的就是获取设备指针

IrrlichtDevice  \*device ：

 video::E\_DRIVER\_TYPE driver\_type = irr::video::EDT\_OPENGL;

 core::dimension2d<s32> screen\_resolution = core::dimension2d<s32>(1280, 800);

 u32 color\_depth = 32;

 bool is\_full\_screen = true;

 IrrlichtDevice\* device = irr::createDevice(driver\_type, screen\_resolution, color\_depth, is\_full\_screen);

driver\_type为驱动类型，可以选择OPENGL，DX8 或 DX9。其余几个参数分别为分辨率，颜色 深度，是否全屏。

获得 device 指针以后，就可以得到属于该 device 的四大块功能：

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

scene::ISceneManager\* scene\_mgr = device->getSceneManager();

gui::IGUIEnvironment\* gui\_env = device->getGUIEnvironment();

io::IFileSystem\* file\_system = device->getFileSystem();

2、创建3D场景

创建3D场景有三个步骤：

(1) 通过场景管理器scene\_mgr添加3D物体：

 // 为场景添加一个立方体，边长100

 scene::ISceneNode\* cube = scene\_mgr->addCubeSceneNode(100.0f);

(2) 为3D物体贴上纹理：

 // 载入纹理

 video::ITexture\* tex = driver->getTexture("box.jpg");

 // 将纹理附加到立方体上

 cube->setMaterialTexture(0, tex);

 // 将纹理EMF\_LIGHTING属性设为false

 // 表示该纹理现实与光源无关，即为图片自身颜色

 cube->setMaterialFlag(video::EMF\_LIGHTING, false);

 // 使纹理支持半透明，半透明效果与图片相同

 cube->setMaterialType(video::EMT\_TRANSPARENT\_ALPHA\_CHANNEL);

(3) 添加Camera，使物体可见：

 // 添加一个相机，在(700,700,-700)位置，往(0,0,0)位置拍摄。

 scene::ICameraSceneNode\* camera = scene\_mgr->addCameraSceneNode(0,

     core::vector3df(700,700,-700), core::vector3df(0,0,0));

3、进入主循环

通过 driver 载入图片和 texture，通过 scene\_mgr 为3d场景添加 irrlicht 内置支持的3D对象等工作后，即可进入主循环，主循环结束时，释放 device，程序结束。其中 beginScene 的参数 SColor(alpha, r, g, b) 为背景色：

1 while (device->run())

2 {

3     if (device->isWindowActive())

4     {

5         driver->beginScene(true, true, video::SColor(0, 0, 0, 0));

6         scene\_mgr->drawAll();

7         gui\_env->drawAll();

8         driver->endScene();

9     }

10 }

11 device->drop();

值得一提的是 drop() 函数。Irrlicht中大部分类都继承自一个 IReferenceCounted 的接口，类似智能指针。Irrlicht 中的惯例是不使用 delete 删除对象，而调用该接口的 drop() 函数。在添加对象的引用时，调用 grap() 函数。

四、Irrlicht引擎之场景中的对象结构

3D对象在场景中被组织成一个树形结构，在该树中，仅有一个根节点。当在程序的主循环中调用scene\_mgr->drawAll()时，系统会从场景的树形结构的根节点开始递归绘制所有的节点对象。

每一个节点对象都维护了一个“动画效果”ISceneNodeAnimator列表：

core::list<ISceneNodeAnimator\*> Animators;

在绘制每一个节点对象之前，都会将其所维护的所有动画效果应用到该节点上。

“动画效果”的原理是，当节点每次被绘制之前，均先根据其维护的动画对象Animators来计算出该节点的位置、大小、纹理。这样，当一个节点不断被绘制时，就产生了动画效果。

五、Irrlicht引擎之3D对象运动原理

所谓运动，实际上是计算机在不停地绘制场景，每绘制一次称之为一帧。 当各帧中物体的位置或外观有所变化，那么它就动起来了。 在irrlicht中，绘制一帧是在run循环中完成的：

1 while (device->run())

2 {

3     if (device->isWindowActive())

4     {

5         driver->beginScene(true, true, video::SColor(0, 0, 0, 0));

6         scene\_mgr->drawAll();  // 绘制一帧

7         driver->endScene();

8     }

9 }

10 device->drop();

我们所要做的，就是在 drawAll() 函数中，更新物体的位置及大小等属性，那么场景就动起来了。

在Irrlicht中，所有与运动相关的一切，都与 scene::ISecenNodeAnimator 这个接口相关。凡是实现这个接口的类实例，都可以通过 addAnimator() 函数加入到 ISceneNode 所维护的animators列表中。

SceneManager 的drawAll() 函数在渲染(render)场景前，会调用其OnAnimate() 函数。这个函数是递归的，以保证加入场景中的每个 SceneNode 都会被调用。 在OnAnimate() 函数中， SceneNode 的每一个 ISecenNodeAnimator 的 animateNode() 函数都会被调用，以更新 SceneNode 的位置、大小或纹理等属性。

其具体的调用顺序如下：

1. SceneManager           --> drawAll()                 绘制一帧画面

2. ISceneNode             --> OnAnimate()              SceneNode运动

3. ISceneNodeAnimator --> animateNode(ISceneNode)      实现运动的具体函数

4. SceneManager           --> render()                  渲染

可见，只要实现 ISecenNodeAnimator 接口，并加入到 SceneNode 中，就能够让该SceneNode 运动起来。而至于具体如何运动，完全可由自己来定义。

六、Irrlicht引擎之消息传递原理

Irrlicht引擎中消息的传递是从device->run()开始的，首先由具体平台的操作系统接口将接收到的用户消息打包成Irrlicht的SEvent结构（这一步打包过程是在device->run()中完成的），再由postEventFromUser()将该消息依次传递给UserReceiver, GUI 和 3D Scene。

Irrlicht 中所有处理消息的类都必须实现 IEventReciever 接口。UserReceiver 就是在CreateDevice函数中指定的一个 IEventReceiver。 也就是说，消息处理的优先级为 UserReceiver > GUI > 3D Scene。

如果在程序中，没有指定UserReceiver和GUI，消息就会直接交给SceneManager。

SceneManager 的消息，也由 ISceneManager 的 postEventFromUser() 传递。 这个函数的实现如下：

1 bool CSceneManager::postEventFromUser(const SEvent& event)

2 {

3     bool ret = false;

4     ICameraSceneNode\* cam = getActiveCamera();

5     if (cam)

6         ret = cam->OnEvent(event);

7

8     \_IRR\_IMPLEMENT\_MANAGED\_MARSHALLING\_BUGFIX;

9     return ret;

10 }

也就是说，只有当前Active(有效)的 ICameraSceneNode 才会接收到消息。ICameraSceneNode 会检测ISceneNodeAnimator 的isEventReceiverEnabled(), 如果为真则调用其OnEvent 函数。

整理消息传递的机制如下：

1. IrrlichtDevice           --> run()                      搜集消息并打包

2. IrrlichtDevice           --> postEventFromUser()    传递消息到userReceiver     GUI 和 Scene

3. ISceneManager         --> postEventFromUser()       传递消息到CameraNode

4. ICameraSceneNode     --> onEvent()                  调用 Animator 的onEvent

5. ISceneNodeAnimator    --> onEvent()  // if enabled      响应消息。

可见，实现 ISceneNodeAnimator 虽然可以 SceneNode 动起来，但只有在 ICameraSceneNode 上，才能够接收和处理消息。