Irrlicht Engine 中文学习指南

收集整理：By ISclub 翻译：FK_Duzhi （感谢他）

日期：2009 年 6 月 11 日星期四

前言：

为了学习 3D 网络游戏开发，尝试过几种经典免费开源的引擎:TV3D,Irrlicht（简称 Irr）,Ogre 等等；TV3D 目前最新的版本 V6.5 已经可以很好的支持 VB6,VC++,Delphi,.NET 等语言，特点是新手学习速度快，门槛低；缺陷是基于 ActiveX 控件，需要在运行前注册 RunTime DLL。

Irrlicht 采用 C++封装的 3D 引擎，是一款轻量级的 3D 引擎，引擎本身精简强悍，功能也齐全，并且有自己专用的场景编辑器；但扩充功能不易，对于新手来说，也是难得的入门快，学习快的引擎。

Ogre 是一款开源的 3D 渲染引擎，它非常强大，并且支持插件等形式来完成新的功能，这点优于 Irrlicht，但入门不易。

本文翻译，以及来源载录信息来自：FK_Duzhi 大侠。

欢迎您从 http://irrlicht.sourceforge.net 下载最新版的 Irrlicht 引擎

1：SDK 目录结构总述
当您下载 Irr 引擎 SDK 解压缩之后，你会看到以下文件夹：
\bin 完全编译好的 Irrlicht,DLL 以及一些编译好的 Demo，你可以通过这些例子来获取 Irr 的运行状态。（Windows Only）

\doc Irr 引擎的一些文档。
\examples 使用 C++编写的例子，告诉用户如何使用 Irr 引擎。
\examples.net 使用.NET 语言编写的例子，告诉用户如何使用 Irr 引擎。
\exporters 一些 Irr 引擎用到的辅助库和工具。
(注：部分版本没有此文件夹，该文件夹内容在 source 中，包括 Jpeglib,libpng,MacOSX,zlib 等)

\include 引擎包含的头文件。
\lib 当你使用 Irr 引擎时需要链接的库文件。
\media Demo 程序需要的一些图形图象和声音素材。
\source Irr 引擎的源代码，必要的话，你可以对其进行编译和 Debug 以获得新的 lib。
\tools 引擎的一些有用的工具。(包含源文件)

2：如何启用引擎
在 Windows 环境下启动引擎的话，你需要去\bin\Win32-VisualStudio 目录下运行程序。或者你也可以运行一个叫 Demo.exe 的程序，它会显示一些 Irr 引擎有意思的功能。
在开始我们自己的程序编写之前，你最好看看\examples 目录下的例子，那里也有一个.html 格式的文件对例子进行说明，以方便你的理解。(编者注：您也可以到这里获取一些例子相关的帮助)
在 Linix 环境下，Irr 引擎是一个静态库。你可以使用 source/Irrlicht 中的 Makefile 文件来生成一个编译好的引擎。之后，你应当可以生成/examples 中的例子程序了吧。之后你可以立刻运行这些例子了。

3：SDK 编译需求
你可以使用下列编译器之一来运行 Irr 应用程序。不过，其他编译器可能也能完美运行，但是我没有做过测试。

- gcc3.3
- gcc 3.4
- gcc 4.0.3
- Visual Studio 6.0
- Visual Studio.NET 2003 (7.1)
- Visual Studio 2005 (8.0)
- Code::Blocks 1.0 + gcc 或 Visio Studio Toolkit
- DevC++ 5.0 + gcc

如果你不想使用编译好的引擎库，你可以自行使用源代码进行编译，你需要以下支持：

- Windows 环境下：

Windows 平台 SDK（一般 IDE 里已经包含）
DirectX 9 SDK(可选，进行 D3D9 支持)

DirectX 8 SDK(可选，进行 D3D8 支持)
l Linux 环境下：
Xserver 以及其包含文件
OpenGL 支持库(可选，包括 libGL 和 libGLU)
GLX + XF86 视频扩展库 (可选)

4：版本注意
1： 您可以在 changes.txt 中查看引擎版本的更改信息。
2： 请注意，那些材质纹理 3D 模型等素材的版权依旧属于其原作者，而不受本引擎版权声明影响。

5：声明
这个声明是基于 Zlib/libpng 的。假若您在项目中使用了 Irr 引擎，您有权不提及它，但是还是希望您能来封感谢信。(笃志注：外国人这点就是好)
请注意，Irr 引擎是基于 JPEG 库，Zlib，libpng 这些库之上的，那就意味着，如果您在自己项目中使用了 Irr 引擎，您必须在项目文件中提到你使用了这些库，并且声明感谢。当然，若您愿意声明感谢 Irr 引擎的话自然更好。更详细的声明信息建议您看 jpeglib 和 zlib 中的 ReadMe 文件。

Irr 引擎许可声明
禁止使用本引擎进行任何不法行为，违反者后果自负。
在基于下列三项要求前提下，本引擎允许任何人进行随意使用，包括制作商业软件，同样也允许你随意修改源代码。

1：若您在项目中使用了本引擎，您可以在项目说明中声明感谢，不过，您也可以不做感谢声明,但是，本引擎文件来源说明绝对不允许修改删除。
2：若您进行源代码修改，请务必做出明显标注。
3：本份声明不允许被修改或删除。

6：联系方式

如果你有问题或者建议，欢迎来访 Irr 官方网页：http://irrlicht.sourceforge.net
在官方网页您将会看到 Irr 主题的论坛，补丁，例子和其他一些帮助。

如果您想加入引擎开发组，请发一封邮件至 Nikolaus Gebhardt: irrlicht@users.sourceforge.net
一部分朋友协助进行了部分引擎的编写和修改工作，请在这里查看完整的作者名表。
http://irrlicht.sourceforge.net/author.html

7：相关工具

为了使您在项目或游戏中使用 Irr 引擎更加方便简单，我们提供了一些外部工具。这些工具和库是完全独立于 Irr 引擎的，但是这些工具是完全针对 Irr 引擎兼容的
。
1：irrKlang 一个 2D 和 3D 音效引擎
irrKlang 引擎是一个免费的 2D 和 3D 音效引擎库，它提供 WAV，MP3，OGG，MOD,XM，IT，S3M 等一些音频格式的播放支持。它是由 C++编写的，下面是它的一些特性：

它能够与 Irr 引擎的完美结合，因为它使用了一些 Irr 简单的 API，
但是又可以完全独立于 Irr 引擎使用。
跨平台。
支持 2D 和 3D 的流音频播放。
支持回声，失真，多普勒等多种 2D3D 音频特效。
支持多种音频格式：wav, mp3, ogg, mod, xm, it, s3m …等等。
高级资源管理和资源探测。
提供了具有可扩展型的音频解码器和 plugin 系统。
免费。
更详细的信息，您可以从这里获得 http://www.ambiera.com/irrklang

2：irrEdit 一个 3D 地图编辑器
irrEdit 是一个免费的 3D 地图编辑器。它能够进行场景曲线编辑，世界场景编辑，粒子系统设计，Mesh 查看等功能。它的文件保存格式为.irr 格式，Irr 引擎能够支持。它的特征包括：

它包括一个高质量的地图产生器
它包括一个粒子系统编辑器
编辑器中还包括有一个自由完善的脚本系统
它包括动画器和数据编辑器
免费

因为它是使用 Irr 引擎编写的，所以它能够读取所有 Irr 引擎所支持的格式。
另外，它能够将一切文件保存输出为 COLLADA 文件格式。
更详细的信息，您可以从这里获得 http://www.ambiera.com/irredit

3：irrXML 一个 C++编写的最快的开源 xml 解析器
irrXML 是一个 C++编写的最快的开源 xml 解析器。它已经嵌入在 Irr 引擎中了，例如我们在读取一个.irr 或 COLLADA 格式文件时就是使用它。它非常快，又是轻量级代码，而且不浪费一点内存，是做游戏时很好的一个库。

如果你的项目不使用 Irr 引擎，你又需要一个快速的 XML 解析器，建议你使用这个库。它的使用声明和 zlib 一样。更详细的信息，你可以从这里获得 http://www.ambiera.com/irrxml

8：一些常见问题和解答

1：Irrlicht 引擎是什么？
Irr 引擎是一个高性能，跨平台，开源的 3D 引擎。它能够帮助你进行实时 3D 程序的开发。它的主要目标是：方便使用，运行速度快，可扩展，线程安全。
这个引擎是一个非常灵活的引擎，可以使用它编写许多不同的应用程序。例如：复杂的 3D 仿真应用程序，第一和第三人称室内或室外射击游戏，策略游戏，2D 游戏等等。

2：我能够在不开源的商业程序中使用这个引擎吗？
你当然可以。Irr 引擎是免费的，而且完全自由的。您可以随意的编辑修改它，但是你必须标注出你的修改部分。这个引擎是基于 Zlib 的版权声明的，既非 GPL 也非 LGPL。更详细的情况，您可以查看内部的声明。

3：我可以用 Irr 引擎做一款游戏吗？
当然可以，而且这会很简单。
但是，请注意：Irr 不是一款游戏引擎，它仅仅进行了图象处理。做为一款游戏，你需要更多的部分，例如声音输出，网络和物理系统，也需要一些游戏相关的编辑器。您当然可以选择自己制作这些编辑器，但是 Irr 引擎已经有一些现成的编辑器和库可以提供给您使用，例如 irrKlang, irrEdit。更详细的情况，您可以查看相关的说明。

4：我可以为引擎做一些贡献吗？
如果你编写了自己的场景节点,图片和 Mesh 读取器，或者对引擎做出了更多的扩展，请在 http://irrlicht.sourceforge.net/phpBB2/index.php 给我们来封论坛短信，我将会查阅您的代码，并会考虑将它发布到 Irr 主页，甚至加入到引擎 sdk 之中。
在例子 http://irrlicht.sourceforge.net/tutorials.html 这页将会告诉你如何对引擎进行扩展(笃志注：并未发现相关资料啊--难道所谓的扩展就是写几个 Demo?)如果你想加入我们的开发组，请先尝试习惯我们的代码规范。(笃志注：--并未给直接的联系方式)

5：现在有很多其他的 3D 引擎可以使用，请问 Irr 引擎相对于其他 3D 引擎有什么优势？
容易使用。Irr 引擎把所有常规的事情和绚丽的特效都已经做好了，您不需要花几天的时间去学习文档。例如：启动引擎，读取并显示一个 Quake3 的地图仅需要调用引擎 6 个函数，代码行数不超过 10 行，有兴趣的话，您可以查看 sdk 中的例子。

灵活性强。虽然引擎很方便使用，但是你能够很容易的修改它。

极其稳定。大部分实时程序在用户进行意外操作时会严重当机。但是 Irr 引擎不同，它会打印出一个警告信息，而应用程序大部分情况会继续运行。

快如闪电。因为 3D 引擎的常见问题在于速度，而影响速度的主要原因在于大量的内置导入解析器，而 Irr 引擎可以直接的进行文件导入而没有使用任何转换器，节省了大量的编译时间。它支持下列格式(.3ds, .md2, .obj, .pk3, .ms3d, .bsp, .x, .bmp, .tga, .jpg, .psd, .pcx 等等)

平台独立性。Irr 引擎可以在众多平台上使用。

独立 API。因为有很多种设备类型，Irr 引擎支持多种 API，当前来说是支持 OpenGL,D3D8 和 D3D9 设备，一个空设备和两种软件模拟设备。甚至，在渲染的时候也允许进行设备类型的更换。

不依赖其他库。Irr 引擎 sdk 不需要其他库的支持。例如，即使用户系统中没有 DirectX,引擎也能顺利运行。

完善的文档。大量的例子和 Demo，并且提供了一些你在 3D 程序中常用的工具，于是你无需新建项目来做这些工作。它包括（GUI 系统工具，Font 生成工具..等）

开源。这个引擎是完全免费并且开源的。你可以随意的修改它，但是你必须标注出你的修改部分。这个引擎是基于 Zlib 的版权声明的，既非 GPL 也非 LGPL。

6：为什么这款引擎这么快？
很多朋友给我写信说：他们使用了一些 3D 引擎，而 Irr 引擎是最快的，为什么这款引擎会这么快呢？实话说，我也不知道明确的原因，但是有以下这些可能：

Irr 引擎没有象其他大部分引擎那样使用保守的拣选运算，而新式的数学拣选更加适合现代的 3D 硬件。它默认的是使用 16 位纹理图，引擎在解析和显示这些图片时候会比其他引擎快些。

Irr 引擎没有使用其他比较慢的库，大部分事情都在引擎核心中实现的。

7：这个引擎能够渲染多少个面？
无限。OK，是接近无限。真正的限制在你的硬件。所以，根据你的需要尽管在程序中加入大量的模型面吧，只要你的游戏不卡。有些朋友问我，为什么教学例子中定义最大面数为 10000，原因是我没有必要去做一个巨大的 Quake3 场景，弄的 SDK 下载包大的很。如果你想知道引擎能够支持渲染多少个面，那么你换个场景素材，自己看吧。

8：我的代码里出了些问题，能帮我解决下吗？我在论坛中没有获得解答。
抱歉，我没有时间为你的应用程序 Debug。很多朋友把他们的代码给我让我来解决，但是我没有时间去为每一个来信的朋友解决问题。大部分情况下都是他们自己的 C++代码原因而非引擎的错误。请打开你的 Debugger 用它去寻找你的程序中的错误。如果你不会使用 Debugger,那么请在每一行代码前执行一句命令行打印代码，你就会找到是哪一行导致当机了。

而你在论坛中获得不了帮助的大部分原因是你没有详细的描述你的问题情况。仅仅帖了 100 行代码然后写句“我这里有错，请帮我解决”这样是不行的。嘿，你是个程序员，应该最少 60%时间用在程序 Debug 上。这个是我 Debug 所花的时间，其他程序员或多或少，但我想都差不多。当你的程序出现错误后，请尽量自己去解决，你将会比问别人找到 bug 而获益更多。

9：“Irrlicht”是什么意思？
Irrlicht 是德语中的一种会发光，会飞行的神话动物。它通常生活在沼泽旁。我想，正确的英语翻译应该是”will-o-the-wisp”。同样，Irrlicht 也可以分割成两个德语单词，“irr”意思是“mad(疯狂的)”,”Licht”意思是”light(灯光)”。Irr 引擎发源于奥地利（德国南部），尽管

非德语用户可能不知道它的发音，我却认为“Irrlicht”是个好名字。

因为很多用户问我这个词如何发音，我专门上传了一个声音文件，你可以听到一个女孩子说”Irrlicht Engine”的声音，请点 http://irrlicht.sourceforge.net/download/irrlichtengine.wav

10：我能够加入 Irr 引擎开发团队吗？

基本上不行。抱歉，我喜欢小型的 Irr 引擎开发团队，因为这样可以保证每个计划简单的得到执行。

但是，这也有例外。如果你对 Irr 引擎提供了一个 强大/有用/突破性 的增强功能，而且愿意加入团队，我们也会欢迎。例如：你能够使 Irr 引擎在 Mac 平台上顺利运行(现在已经可以了，但仅仅是一个 Demo 可以)。部分朋友提供了 Irr 引擎一小部分的修改，但是他们没有通过版本控制器来获得引擎的同步，我们也不能让他加入。

不过现在又改了：如果你创建了一个模块并且按照 Irr 引擎编写规范(下文有)，并且愿意长期维护并分享它，你可以加入本小组，你只需要将你的代码发给 niko 就可以了。

请注意，每月总有 4-5 个朋友希望加入 Irr 引擎开发团队，我实在难以选择，因为我们总希望团队尽可能的保持简单而工作快速起来。

11：代码规范是什么？
如果你制作了一个 Irr 引擎的补丁，而且希望它加入到引擎中，请遵循以下原则：
除非特别特别特别必要，尽可能的不要依靠其他库
别用 STL，请使用 Irr 引擎中内置的容器类。
别用平台相关函数，必要的话请使用 os::命名空间内的函数。
请谨记，Irr 引擎是可以运行于其他平台和编译器下的。
请不要使用特殊的编译器扩展，尽量使用标准 C++
请使用 Irr 引擎中的类型定义，例如 f32, u32
请尽量不要修改现有的 Irr 接口。除非极其必要而且正确。另外，建议您应当下载最新版本的 Irr 引擎。若你依旧有这种修改需要，最好和 niko 先商谈一下。
尽量使用现有的 Irr 代码风格。

1：接口以 I 开头，实现以 C 开头，仅保存数据的结构体以 S 开头。

2：函数尽量以小写字母开头，变量尽量以大写字母开头。

3：接口的公共函数，其参数应当给默认值。

4：增加一个函数或修改一个函数，应当按照 doxygen 规范给予注释。如果你对 Irr 引擎进行了大量的修改，请在 changes.txt 中说明。

5：使用 C++风格的编码习惯，别使用 Java 代码习惯，例如
For(foo)

```
{
        //TODO:
}
    For(foo){
        // TODO:
    }
```

前一种就比后一种好。

6：上面的都是比较重要的规则，但是还有一些其他的小规则。例如，尽量使用命名空间，换行和 tab 键，使你的代码看起来好看一些。
多检查代码可能出错的地方，预防当机。例如，对一些参数进行判断，检查指针是否为空。另外，发生错误时，输出一个错误 log，用其他方式使程序继续运行下去。这些工作除非在软件模拟格式下都是必须做好的。
以上就是基本的代码规范了。

12：没有 CVS 吗？
Irrlicht 是由一个 subVersion 服务器做主机的，所有 CVS 服务器的数据都是老版本的。
如果你依然希望获得最新版本的 CVS，
请通过:pserver:anonymous@cvs.sourceforge.net:/cvsroot/irrlicht ，
模块名是”Irrlicht”,更详细的配置请看这里 http://sourceforge.net/cvs/?group_id=74339

13：我如何链接 SubVersion 服务器？
你能够通过 https://irrlicht.svn.sourceforge.net/svnroot/irrlicht/trunk 进行访问。

14：下一个版本引擎将有什么功能？
请查看开发主页 http://irrlicht.sourceforge.net/development.html 和我的博客 http://www.irrlicht3d.org 。我会在上面时时更新最新的开发新闻。你或许能在其中发现下一版本引擎的的预告说明。

15：下一个版本 Irr 引擎什么时候发布？
等它被完成的时候。这个引擎仅仅是一个开源引擎，我从中没有获得一分钱，所以我不能做出最终期限。一般 1-2 个月我会发布一个新的小版本更动，可能是从 0.4 到 0.5。平时我会提前 1-2 星期发布公告并贴出一些截屏和新的特征说明。所以请不要再给我来信询问这个问题，实际上，大部分时候我自己也不知道这个问题的答案。

评价：

虽然 Irr 本身还有一些缺陷和功能上的不足，例如 3D 部分功能并不强大，编辑器比较缺乏，部分封装也比较死板，灵活性缺乏。以致于其适用性总受到一定制约，然而这并不能掩饰它的长处：使用起来绝对简单。

这些例子却相当的显示出 Irr 引擎的特点，简洁，明了，针对性强。非常适合 Irr 新手学习。本套例子适用范围不是 3D 菜鸟，例子内部对很多的 3D 名词和概念并没有做出充分的解释。它适用于本身已经对 3D 有一定了解的游戏程序员学习。

学习这套引擎，你还需要一个 Irr 引擎库以及例子中的素材，这些在 Irr 官方都有，而且是随源代码一同提供的，无需下载。

另外你还需要自己去微软官方下载一份 DXSDK（或者用 OpenGL 也可，若使用 BurningVideo 则极其影响速度了）。

学习 Irr 引擎的话，最好是从这些例子入手，比起直接看源代码或 API 文档有效的多，
学习的时候，不妨多尝试更改里面一些函数的参数并查看结果，会使你对函数的理解更加深刻。

(例子中并没有详细解释 API 参数，但是因为很简单，应当 3D 程序员可以做到一目了然)
现在我已翻译完毕 Irr 引擎自带的 16 个例子，接下来我对其进行简单的统计分析。

01.HelloWorld

这个例子简单的告诉我们 Irr 引擎的一般使用步骤：

1)包含一个 irrlicht.h 头文件，若需要控制台，则再包含 iostream，根据平台以及消息机制的需求，你可以再包含 windows.h。

2)设置命名空间，所有 Irr 引擎文件都在 irr 命名空间之下被定义，irr 命名空间之下又有五大模块，每个模块有自己独立的命名空间，分别为：

Core ---内部是一些引擎核心类，包括各种数据结构，自定义结构类型。
Gui ---内部是一些常用的图形用户接口类，实现了各种常用控件
Io ---内部是一些输入输出，xml,zip,ini 文件读取写入等操作接口
Scene ---内部主要负责场景管理，包括场景节点，摄象机，粒子系统，Mesh，公告版，灯光，动画器，天空盒，地形等绝大部分的 3D 功能。

在 Irr 的思想中，这些东西都归属于一个一个的场景节点，统一由 sceneManager 管理。在其中又有一个特殊的命名空间 irr::scene::quake3,这个命名空间内的类和函数负责对 quake3 格式的文件进行特殊的完善处理。video---内部主要是负责对视频驱动的设置，2D 和 3D 渲染都在这里实现。包括了纹理，材质，灯光，图片，顶点等渲染属性的控制。

3)通知编译器链接 irr 库文件。

在以上预处理步骤完成后，我们在 main 中需要做的步骤如下：
1)创建设备
2)获取场景管理器，GUI 环境，视频设备的指针，使用他们进行渲染控制。
3)之后在 beginScene 和 endScene 中进行 DrawAll
4)释放设备
就这样，一个最基本的完整的基于 Irr 引擎的程序就完成了。

02.Quake3Map

这个例子仅简单的介绍了下如何 Load 一个 Quake3 的场景，其中没有什么特殊的概念。
只需注意一下 Irr 的思想就可以，将场景和摄象机都做为场景节点来处理。
最后，提供了用户选择设备类型的功能，并显示了 FPS。

03.CustomSceneNode

这个例子实用性更低了，感觉是为了 3D 编程入门用户使用的，它告诉了我们如何自己定义创建一个场景节点。之前的 Quake3Map 是从文件中读取的各种模型信息，而这里，我们自己手工定义顶点，颜色等信息，并将其做为一个场景节点加载到 Irr 程序中去。由于实际开发中几乎不会手工定义这些东西，所以实用性相当低，仅值得简单了解。

04.Movement

这个例子相当重要，它增加了一个用户输入处理的功能。这个功能的实现实际上很简单，如下步骤：

1)自定义一个类，继承于 IEventReceiver 类，并实现其中的函数 OnEvent(sEvent)
2)根据 sEvent 参数值进行不同的逻辑处理
3)生成这个事件处理类的对象，在创建设备的时候将其做为参数传给设备就 OK 了。
本例中另外一个要点就是，为场景节点设置一个 Animator，很遗憾，一直到现在，我依旧无法正确的对
Animator 这个单词进行翻译，它可以实现旋转，移动等多种令模型生动起来的功能，使用起来也很简单，如下：
1)使用场景管理器创建一个 Animator，注意它有不同的类型，可以实现不同的功能。
2)将它和一个场景节点捆绑起来，之后场景节点就实现了该 Animator 所具有的功能。
3)值得注意的是，当 Animator 被绑定之后，它就可以立即释放了，所以个人猜测在绑定的时候，场景管理器
对 Animator 创建了一个副本，将副本给了目标场景节点，而非这个 Animator 本身。

05.UserInterface

这个例子也比较重要，它演示了 GUI 的使用方法。
这个使用方法是相当容易的，甚至到了一种太简单的地步，就是你简单的使用 GUI 环境指针 add 就 OK 了。
具体的请见代码，很容易理解。值得注意的是，GUI 在创建的时候，需要提供一个 ID 编号，这样的话，在
消息接收器中，就可以根据 GUI 的 ID 编号来获知是哪个 GUI 应当处理的事件了，很简单，

很高效！

06.2DGraphics

这个例子中演示了如何使用 Irr 引擎实现 2D 图形渲染，并不大重要。
代码中没有什么难理解之处，不过 Irr 支持的图片格式可真不少，某志不再在此列举了，有兴趣的可以看看 Irr 文档。
2D 部分 Irr 引擎明显没有做为重点来做，都是比较底层的功能，真正拿它来做 2D 游戏的话，还需要用户进一步封装。

07.Collision

这是一个相当重要的例子，它告诉了我们如何使用 Irr 引擎进行碰撞检测和鼠标拣选。
Irr 引擎中的碰撞检测是依靠碰撞 Animator 的，方法如下：
使用场景管理器创建两个碰撞 Animator，将他们分别绑定到要进行碰撞的两个目标对象上，就 OK 了。注意的是，碰撞 Animator(检测器)参数设置会比较麻烦，需要好好看下 SDK。
这个例子的第二要点就是鼠标拣选，虽然作者演示了两种鼠标拣选模式，一种基于三角面的拣选，一种基于碰撞盒的拣选，

实际上区别不大，都是通过场景管理器 getSceneCollisionManager 获取碰撞管理器来进行处理，由于封装的很好，

所以不会太难。（若无法理解这两中拣选的区别，请运行例子测试）

08.SpecialFx

这也是个比较重要的例子，主要演示了几种常用特效的使用，包括水面，移动光源，粒子系统，动态阴影。
实际上这些特效在 Irr 中有很方便的封装，你只要使用场景管理器创建一个相应的 SceneNode，再设置其参数就可以了。
在这里，你会更加明确的感受到 Irr 作者对场景节点的喜好了，一切都是场景节点！包括阴影也照样。具体的实现还请看例子，例如水面的双层纹理效果，粒子发射器的参数分别什么作用，还需要用户简单的更改调试查看。

09.MeshViewer

实际上，这个例子是一个让用户休息一下的例子，其中没有新的知识点，只是将 GUI 和图形渲染之间进行了一些衔接罢了，对 GUI 的使用还有问题的朋友可以进去看看。想这么点代码就做一个 MeshViewer 还是艰难了点吧，虽然我承认 Irr 封装的够强悍。

10.Shaders

这个例子告诉了我们如何使用 Irr 引擎实现 Shader. Irr 引擎既支持高级 Shader 语言,也支持 asm，也允许从文件中进行 Shader 的读取。但是操作难度不大，步骤如下：

1)创建一个 Shader 回调类继承 IShaderConstantSetCallBack 实现其中的 OnSetConstanes 函数，每桢材质渲染前，Irr 设备都会调用这个函数。
2)通过设备指针获取 GPU 服务对象指针
3)通过 GPU 服务对象指针去创建 Shader 材质,并将 Shader 回调类的对象做为参数传入。

例：MyShaderCallBack* mc = new MyShaderCallBack();
newMaterialType1 = gpu->addHighLevelShaderMaterialFromFiles(vsFileName, psFileName, mc,...);
这样就 OK 了，真的，没见过比这更简单的设置 Shader 了。

11.PerPixelLighting

所谓的 PerPixelLighting 逐象素光照，个人感觉就是凹凸贴图的另一个名词而已。
这个例子中实现凹凸贴图，使用了两种方式，法线映射和视差映射。这两种映射依旧是两个函数的调用而已，当然，这需要硬件支持 Shader，也需要我们明白一点凹凸帖图的原理：模型上贴两层纹理，一层颜色纹理用来普通的显示，一层高度纹理，我们通过高度纹理图来生成一个切线图，进行渲染就可以实现有立体感的凹凸贴图了。
这里需要注意的是 createMeshWithTangents 是将一个高度图转换为切线图的函数，它的内部对纹理图进行的复制，所以，最后记得要 drop 掉。
另外，在本例中增加了雾特效，这个实现本身相当容易，不过请记住打开每个场景的雾效开关，默认它是关闭着的。

12.TerrainRendering

此例子实现地形渲染，没有什么要点在其中的，只是作者在告诉用户，Irr 引擎使用了地形 Lod 和纹理混合的功能罢了。

13.RenderToTexture

本例子实现动态纹理渲染，如果用户了解镜面渲染原理的话，本例子也没有什么要点可言，只是进行两次渲染实现个动态渲染纹理而已，而且，这里 Irr 引擎封装不足，完全可以进一步改进。

14.Win32Window

这是个很有趣的例子，在一个 Win32 窗口内运行 Irr 程序，请注意例子中提示的两种消息接收方式，在特殊条件下该例子比较有用。Irr 支持用户使用 Win32 自身的消息分发系统，也允许继承自身的 OnEvent 进行消息处理，这两种方法都有效，注意，后一种方式必须 Device->Run(),详细情况请见例子。

15.LoadIrrFile

这是一个脑残的例子，不知道作者在想些什么，一个 loadScene(XX.irr) 把我们打发了，可能主要目的是向我们推荐他的 irrEdit 吧，我们拿记事本打开.irr 文件，会发现，那不过是 xml 格式的一套信息而已，类似于 Direct 的.x 对模型的格式定义一样，.irr 是对场景的一个格式定义。

16.Quake3MapShader

这个例子和 02.Quake3Map 基本上一致，不同之处在于两点：
1：本例加了个截屏功能。
2：在 02 例子的场景之上，本例添加了一些 Shader。

阅读本例需要我们对 quake3 的地图文件格式有一定的了解，即使你想去了解也没有关系，你只需要明白，Quake3 地图是将地图上的物件（如桌子，灯座，甚至灯光）都寄存在相关

材质中就 OK 了。本例中大量使用 irr::scene::quake 中的函数，对 quake 格式没兴趣的朋友可以不看。
简单的 Irr 例子介绍就这样了,之后我会继续翻译 Irr 相关文档说明以及 API 文档，并将其发布在我的博客之中，欢迎来访指教。（欺骗群众，2007 的话，2009 年都没兑现!）

Example 代码学习参考

（注意：一些老代码在新的 SDK 下可能执行有问题，具体需要手工修正，请注意!）

Irrlicht 之 01.HelloWorld

```
/*
这份指南告诉你如何配置 IDE 环境来使用鬼火引擎，以及如何使用它来写一份简单的 HelloWorld。
这个程序将告诉我们如何简单的使用 VideoDriver 视频驱动,GUIEnvironmentGUI 环境和 SceneManager 场景管理器。
使用这个引擎，我们首先需要包含 irrlicht.h 这个头文件，它包含在鬼火引擎 SDK 的\include 中。
首先让编译器找到这个头文件，我们应当在项目中指定该文件夹路径。这个过程和你所使用的 IDE 环境不同而不同。
我这里只介绍一下 VC6.0 和.NET 的设置方式。
点工具->选项->设置，添加其头文件和库文件路径。这样的话你就可以顺利的在鬼火引擎上进行开发。
*/
#include <irrlicht.h>
#include <windows.h>
/*
在鬼火引擎中，所有的代码都被命名空间 irr 约束着。所以如果你想调用引擎中的任何一个类，你必须在类名前
这样写 irr::ClassName，例如，你要使用 IrrlichtDevice 类，你就这么写 irr::IrrlichtDevice。
或者偷懒，告诉编译器使用命名空间.using namespace irr;
*/
using namespace irr;
/*
下面是五个子命名空间。你会在文档中的 NamespaceList 中找到他们。
就象上面的命名空间一样，如果你想偷懒，直接告诉编译器好了。
*/
using namespace core;
using namespace scene;
using namespace video;
using namespace io;
using namespace gui;
/*
```

```
我们应该包含一个 Irrlicht.Dll 文件，我们也可以在编译器设置，或者直接这里写一个声明。
*/
#ifdef _IRR_WINDOWS_
#pragma comment(lib, "Irrlicht.lib")
#endif

/*
int main()是一个主函数，我们在所有平台中都可以这么写。在 Windows 平台下，我们也可以使用
WinMain 函数，但是那样的话，我们将失去一个控制台。
*/
int main()
{
/*
引擎函数中最重要的函数就是“CreateDevice”创建设备了。鬼火引擎做一切事情的根对象就是这个鬼火设备了。
CreateDevice 有七个参数。
1：设备类型 deviceType：
可以是空设备，软件模拟设备，第三方渲染设备，D3D8，D3D9，OpenGL。在例子里我们使用的是 EDT_SOFTWARE
的软件模拟方式，但是你可以拿 EDT_BURNINGSVIDEO, EDT_NULL, EDT_DIRECT3D8 , EDT_DIRECT3D9,或 EDT_OPENGL
进行测试。
2：窗口宽高
3：窗口色深
4：是否全屏
5：我们绘制阴影时是否使用摸版缓冲区
6：是否开启同步，注意的是只有全屏时才有效
7：事件接收器的对象，也就是对鼠标消息，键盘消息，用户信息，GUI 消息等一系列消息的回调处理函数指针。
    我们这里不加处理的话，设置为 0。
*/
IrrlichtDevice *device =
    createDevice( video::EDT_DIRECT3D9, dimension2d<s32>(1024, 768), 32,
     false, false, false, 0);
/*
设置窗口标题文字。
注意，这里有个 L 在字符串之前，这代表鬼火引擎绘制文字时使用的是宽字节
*/
device->setWindowCaption(L"Hello World! - Irrlicht Engine Demo");
/*
获取视频设备，场景管理器和用户图形环境的指针并存储起来。
*/
IVideoDriver* driver = device->getVideoDriver();
```

```
ISceneManager* smgr = device->getSceneManager();
IGUIEnvironment* guienv = device->getGUIEnvironment();
/*
使用用户图形环境设备绘制一个 Label。
*/
guienv->addStaticText(L"Hello World! This is the Irrlicht Software renderer!",
    rect<int>(10,10,260,22), true);
/*
我们读取一个模型文件并且显示它。
步骤很简单，只需要使用场景管理器 getMesh()读取 mesh 信息.再创建一个场景节点
来显示这个 mesh 就可以了。
Irr 引擎同时还支持.obj（Maya 文件）.bsp（Quake3 地图文件）.ms3d（Milshape 文件）。
另外，Quake2 模型的导入支持也可以在网上找到。BrianCollins 写的。
*/
IAnimatedMesh* mesh = smgr->getMesh("../../media/sydney.md2");
IAnimatedMeshSceneNode* node = smgr->addAnimatedMeshSceneNode( mesh );
/*
接下来我们让 Mesh 看起来更漂亮一些，我们对其材质环境进行一些设置：
1：我们在这里不需要使用动态光影，所以先关闭灯光效果,若打开了将会是一团黑影
2：然后我们设置桢循环，保证我们的模型动作。
3：最后我们在 Mesh 上贴一层纹理，若是没纹理的话，Mesh 上将会填充一种颜色。
*/
if (node)
{
    node->setMaterialFlag(EMF_LIGHTING, false);
    node->setMD2Animation ( scene::EMAT_STAND );
    node->setMaterialTexture( 0, driver->getTexture("../../media/sydney.bmp") );
}
/*
设置摄象头的 3D 空间位置(0,30,-40)以及观察目标点(0,5,0)
*/
smgr->addCameraSceneNode(0, vector3df(0,30,-40), vector3df(0,5,0));
/*
好了，现在我们设置好了场景，还是绘制吧。
首先我们创建一个死循环，除非我们按下 ALT+F4 导致设备不再运行，这个循环就会持续下
去。
*/
while(device->run())
{
    /*
    在 beginScene()和 endScene()中间写下需要绘制的代码。
    其中，beginScene 会将屏幕以指定的颜色和深度缓冲清空。
    之后我们使用场景管理器和 GUI 环境去进行内存中的绘制。
    最后 endScene 会这些东西绘制到屏幕上。
```

```
    */
    driver->beginScene(true, true, SColor(255,100,101,140));
    smgr->drawAll();
    guienv->drawAll();
    driver->endScene();
}
/*
当我们一切都 OK 了之后，我们再释放 Irr 设备就可以了。
我们需要注意的是，前面的视频设备，场景管理器，GUI 环境这些仅仅是进行了获取，
并非"Create"，所以我们仅需要删除 Irr 设备，它是 Create 出来的。
我们->drop()函数将其释放。
更详细的信息可以查看 Irr 官方的说明。
http://irrlicht.sourceforge.net//docu/classirr_1_1IUnknown.html#a3
*/
device->drop();
return 0;
}
```

Irrlicht 之 02.LoadQuake3Map

```
/*
这份指南将为你演示如何向引擎中载入一张 Quake3 地图文件，并且告诉你如何
最便捷优化的创建一个场景节点渲染，以及创建一个用户控制的摄象机。
让我们象上一个 HelloWorld 例子一样做吧：首先包含 Irr 引擎头文件。
另外包含一个 iostream 头为了方便用户的控制台输入。
*/
#include <irrlicht.h>
#include <iostream>
/*
类似于写 HelloWorld 例子之前需要做的准备一样，在 Irrlicht 引擎中，一切
函数，类命名都是在 irr 命名空间内的。我们依旧要告诉编辑器我们现在使用
的函数应当在 irr 命名空间内寻找。它有五个子命名空间，Core,Scene,Video,
Io,Gui.与 HelloWorld 不同的是，我们这里没有为五个子空间分别指定命名空间
的通知，因为这样做的话，在下面的代码中，你将更容易获知每个函数到底是属于
哪个命名空间内的。当然，你也可以加上 using namespace XX;尽随你意了。
*/
using namespace irr;
/*
同样，为了可以使用 Irrlicht.DLL 文件，我们需要链接一个 Irrlicht.lib 文件，我们需要
进行项目设置，或者在代码中进行一次链接声明。
*/
#pragma comment(lib, "Irrlicht.lib")
/*
OK，我们开始。main()入口
*/
int main()
{
/*
类似 HelloWorld 例子，我们通过 CreateDevice 创建一个 Irr 设备，不过在这里我们允许
用户进行硬件加速设备的选择。其中软件模拟进行一次巨大的 Q3 场景的加载将会相当慢，
不过为了进行演示有这样一个功能，我们也把它列做选项了。
*/
video::E_DRIVER_TYPE driverType;
printf("Please select the driver you want for this example:\n"\
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
    " (f) NullDevice\n (otherKey) exit\n\n");
char i;
std::cin >> i;
switch(i)
{
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
```

```
    case 'c': driverType = video::EDT_OPENGL;      break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO;break;
    case 'f': driverType = video::EDT_NULL;          break;
    default: return 1;
}
// 创建 Irr 设备
IrrlichtDevice *device =
    createDevice(driverType, core::dimension2d<s32>(640, 480));
if (device == 0)
    return 1;
/*
获取一个视频驱动和场景管理的指针。
*/
video::IVideoDriver* driver = device->getVideoDriver();
scene::ISceneManager* smgr = device->getSceneManager();
/*
为了显示 QUAKE3 的地图，我们首先需要读取它。
Quake3 地图被打包在.pk3 文件中，所以我们的文件系统需要加载.pk3 包文件，
在我们加载它之后，我们还需要从包文件中对其进行读取。
*/
device->getFileSystem()->addZipFileArchive("../../media/map-20kdm2.pk3");
/*
现在我们可以通过调用 getMesh()函数来进行 Mesh 的读取。我们获得了一个动画 Mesh
IAnimatedMesh 的指针。然而我们可能有疑问，Quake3 地图并非一个动画，我们为什么要
使用 IAnimatedMesh 动画 Mesh 呢？我们先研究下 Quake3 的地图，它是由一个巨大的模型
以及一些贴图文件组成的。我们可以理解为，它是由一个动画组成，而这个动画仅有一桢，
所以我们获得动画的第一桢 getMesh(0)(其中 0 就是指定桢数)，然后使用它创建一个八叉树
场景节点。
八叉树的作用是对场景渲染进行优化，就是仅仅渲染摄象机所见的场景，这个请自行
查看 3D 渲染相关书籍。
相对于八叉树场景节点的另一种加载方式就是直接创建一个 AnimatedMeshSceneNode，
动画 Mesh 场景节点，但是这样做的话就不会进行优化的拣选，它会一次性加载绘制所有的
场景。
在下面的代码里，我两种类型都写了，你可以切换着进行尝试一下。
值得注意的是八叉树场景的适用范围一般是大型的室外场景加载。
*/
scene::IAnimatedMesh* mesh = smgr->getMesh("20kdm2.bsp");
scene::ISceneNode* node = 0;

if (mesh)
    /*  下面两行可更替进行编译  */
    //node = smgr->addOctTreeSceneNode(mesh->getMesh(0), 0, -1, 128);
    node = smgr->addAnimatedMeshSceneNode(mesh);
```

```
/*
我们将世界矩阵的原点进行一些平移以适应这个模型。
*/
if (node)
    node->setPosition(core::vector3df(-1300,-144,-1249));
/*
现在我们仅仅需要一个摄象机去观察这张地图。这次我们设计一个可用户控制的
灵活摄象机。在 Irr 引擎中有许多不同类型的摄象机：例如，Maya 摄象机就类似于
Maya 软件中的摄象机控制，左键按下可进行旋转，两键按下就可以进行缩放，
右键按下就可以进行移动。假如我们想创建这样操作方式的摄象机，那么只要
addCameraSceneNodeMaya()就可以了。而我们现在需要设计的摄象机则是类似于
标准 FPS 的控制设定，所以我们调用 addCameraSceneNodeFPS()函数来创建。
*/
smgr->addCameraSceneNodeFPS();
/*
鼠标的图标没必要显示出来，所以我们设置其属性为隐藏。
*/
device->getCursorControl()->setVisible(false);
/*
我们做完了所有的事情，现在我们开始绘制它吧。我们还需要在窗口的标题上
显示当前的 FPS。
if (device->isWindowActive()) 这一行代码是可选的，但是为了预防由于
切换活动窗口而导致引擎渲染桢速率显示不正确，还是加上吧。
*/
int lastFPS = -1;
while(device->run())
if (device->isWindowActive())
{
    driver->beginScene(true, true, video::SColor(0,200,200,200));
    smgr->drawAll();
    driver->endScene();
    int fps = driver->getFPS();
    if (lastFPS != fps)
    {
      core::stringw str = L"Irrlicht Engine - Quake 3 Map example [";
      str += driver->getName();
      str += "] FPS:";
      str += fps;
      device->setWindowCaption(str.c_str());
      lastFPS = fps;
    }
}
/*
最后，记得销毁 Irr 设备。
```

```
*/
device->drop();
return 0;
}
```

Irrlicht 之 03.CustomSceneNode

```
/*
这个例子是为更高级一些的开发者设计的，如果你仅仅是简单的了解 Irr 引擎，
请优先看看其他的例子。
这个例子里演示了如何手工创建（而非从文件加载）一个场景节点，并在引擎
中将其进行显示。首先，我们需要一个手工创建的场景节点。如果你想实现一个
渲染技巧，Irr 引擎并不支持的时候，你就必须自己手工创建了。
例如，你想在一个大场景地形节点中增加一个户内场景的入口时，就可以自己
手工创建节点了。在创建这些自定义节点时，你会发现你能够很轻松的对 Irr 引擎
进行延伸，使其适合你的需求。
在这个例子以至今后的例子中我依旧保持我的简朴风格：保证代码的简洁，
并将一切代码都仅写在一个.cpp 源文件中。
最开始，依旧是包含头文件以及链接库，声明命名空间。
*/
#include <irrlicht.h>
#include <iostream>
using namespace irr;
#pragma comment(lib, "Irrlicht.lib")
/*
从这里开始进入了例子中最重要的部分：
我们自己定义的场景节点类。为了简单，我们不象前文所说，在巨大的场景地形
中增加一个户内场景入口了，我们仅简单的做一个 4 个顶点连接出 3D 物体，
虽然这个显的很简单，实际上两者原理是一样的，我们也仅仅负责将 3D 物绘制出来，
并不做其他逻辑。
当我们自己定制的场景节点插入 Irr 引擎场景吧。
首先我们将自定义场景节点继承于 ISceneNode 接口，并重载其一些函数。
*/
class CSampleSceneNode : public scene::ISceneNode
{
/*
首先，我们声明一些成员变量，开辟一些空间来存储数据。包括：
1 个包围盒，4 个顶点，以及 3D 物体的材质。
*/
core::aabbox3d<f32> Box;
video::S3DVertex Vertices[4];
video::SMaterial Material;
```

```
/*
构造函数的参数包含了：场景节点的父节点指针，场景管理器的指针，
以及一个场景节点的 ID 号。
在构造函数内部，我们调用了父类的构造函数，设置了一些我们绘制场景节点
和创建 3D 物体的材质和顶点等属性。
*/
public:
CSampleSceneNode(scene::ISceneNode* parent, scene::ISceneManager* mgr, s32 id)
    : scene::ISceneNode(parent, mgr, id)
{
    Material.Wireframe = false;
    Material.Lighting = false;
    Vertices[0] = video::S3DVertex(0,0,10, 1,1,0, video::SColor(255,0,255,255), 0, 1);
    Vertices[1] = video::S3DVertex(10,0,-10, 1,0,0, video::SColor(255,255,0,255), 1, 1);
    Vertices[2] = video::S3DVertex(0,20,0, 0,1,1, video::SColor(255,255,255,0), 1, 0);
    Vertices[3] = video::S3DVertex(-10,0,-10, 0,0,1, video::SColor(255,0,255,0), 0, 0);
/*
Irr 引擎需要你定义的场景节点的包围盒。它将会使用这个包围盒进行一些自动
的剪裁等工作。因此我们需要使用 3D 物体的 4 个顶点来创建他。
如果你不希望引擎使用包围盒进行自动剪裁，或者不想创建包围盒，你可以这么写
AutomaticCullingEnabled = false;它可以帮你关闭包围盒。
*/
    Box.reset(Vertices[0].Pos);
    for (s32 i=1; i<4; ++i)
      Box.addInternalPoint(Vertices[i].Pos);
}

/*
在绘制之前，Irr 的场景管理器会调用每个场景节点的 OnRegisterSceneNode()方法。
如果你想场景能被自动加载，那么你就将它注册到场景管理器中，它就会在场景管理器
进行 Render 的时候被自动调用绘制。
那么引擎为什么不默认的将全部场景节点都注册进去呢？
这是因为渲染顺序不好决定的原因，个别时间还是需要用户进行特殊的设置。
例如，在一般的场景节点被渲染的时候，深度缓冲区的阴影需要在其他所有场景节点
渲染之后进行绘制， 或者象摄象机，光线场景节点这些却需要在其他普通场景节点
渲染之前进行渲染。
不过我们在这里仅需要普通渲染，所以可以直接将其注册到场景管理器中去。
如果我们需要提前或拖后进行渲染的话，可以这么写
SceneManager->registerNodeForRendering(this, SNRT_LIGHT_AND_CAMERA);
这样就是告诉场景管理器，我们这次注册的场景节点，是和灯光和摄象机一样需要
提前渲染的。
值得庆幸的是在本场景节点被注册的同时，它的所有子节点也会默认的得到注册。
*/
virtual void OnRegisterSceneNode()
```

```
{
    if (IsVisible)
      SceneManager->registerNodeForRendering(this);
    ISceneNode::OnRegisterSceneNode();
}
/*
在 render()方法中，最有趣的事情发生了：场景节点开始渲染它们自己了。
我们重载这个了这个函数进行 3D 物体的绘制工作。
In the render() method most of the interesting stuff happenes: The
Scene node renders itself. We override this method and draw the
tetraeder.
*/
virtual void render()
{
    u16 indices[] = { 0,2,3, 2,1,3, 1,0,3, 2,0,1 };
    video::IVideoDriver* driver = SceneManager->getVideoDriver();
    driver->setMaterial(Material);
    driver->setTransform(video::ETS_WORLD, AbsoluteTransformation);
    driver->drawIndexedTriangleList(&Vertices[0], 4, &indices[0], 4);
}
/*
我们在重载 ISence 之外，至少还是新创建了三个额外的方法。
GetBoundingBox()可以获得场景节点的包围盒
GetMaterialCount()返回了这个场景节点的材质数量。(我们这个 3D 物体仅一套材质)
getMaterial()返回了指定编号的材质。(因为在这里我们只有一个材质，
我们可以 return Material;这样写下去，并且在本代码中，该函数的参数也
不应当大于 0，我们只有一个材质嘛，编号最大当然是 0 咯)
*/
virtual const core::aabbox3d<f32>& getBoundingBox() const
{
    return Box;
}
virtual u32 getMaterialCount()
{
    return 1;
}
virtual video::SMaterial& getMaterial(u32 i)
{
    return Material;
}
};
/*
就这样，我们自定义的场景节点就 OK 了，现在开始我们运行引擎，
来创建这个场景节点和一个摄象机，再看看结果。
```

```
*/
int main()
{
// 首先允许用户选择设备类型
video::E_DRIVER_TYPE driverType;
printf("Please select the driver you want for this example:\n"\
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
    " (f) NullDevice\n (otherKey) exit\n\n");
char i;
std::cin >> i;
switch(i)
{
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
    case 'c': driverType = video::EDT_OPENGL;    break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO; break;
    case 'f': driverType = video::EDT_NULL;      break;
    default: return 0;
}
// 创建一个 Irr 设备
IrrlichtDevice *device =
    createDevice(driverType, core::dimension2d<s32>(640, 480), 16, false);

if (device == 0)
    return 1;
// 获取视频驱动和场景管理器指针，并创建摄象机。
device->setWindowCaption(L"Custom Scene Node - Irrlicht Engine Demo");
video::IVideoDriver* driver = device->getVideoDriver();
scene::ISceneManager* smgr = device->getSceneManager();
smgr->addCameraSceneNode(0, core::vector3df(0,-40,0), core::vector3df(0,0,0));
/*
创建我们的场景节点，注意！在我们创建完它之后立刻就释放掉了 drop()。
这并没有错误，是合法的。因为我们的场景管理器已经控制了它.
当然，你怕不安全还需要用，非要在代码最后（设备释放之前）
再 drop()它，也没有什么问题。
*/
CSampleSceneNode *myNode =
    new CSampleSceneNode(smgr->getRootSceneNode(), smgr, 666);
myNode->drop();
/*
现在你应该知道了如何创建一个我们自定义的场景节点了吧，并且让它象其他的
场景节点一样被渲染绘制。为了让画面更有意思一些，我们为场景节点加了个旋转动画器①。
```

让我们的 3D 物体所在场景节点旋转起来吧。
①笃志注：意译为动画器，原文是 RotationAnimator 鼓舞者。
*/
```
scene::ISceneNodeAnimator* anim =
    smgr->createRotationAnimator(core::vector3df(0.8f, 0, 0.8f));
myNode->addAnimator(anim);
anim->drop();
/*
开始进行绘制，并显示 FPS
*/
u32 frames=0;
while(device->run())
{
    driver->beginScene(true, true, video::SColor(0,100,100,100));
    smgr->drawAll();
    driver->endScene();
    if (++frames==100)
    {
      core::stringw str = L"Irrlicht Engine [";
      str += driver->getName();
      str += L"] FPS: ";
      str += (s32)driver->getFPS();
      device->setWindowCaption(str.c_str());
      frames=0;
    }
}
/*
释放设备，例子结束
*/
device->drop();

return 0;
}
```

Irrlicht 之 04.Movement

```
/*
这个例子告诉我们如何使场景节点动起来。
SceneNodeAnimators 的最基本概念就是能够使用键盘让场景节点移动起来
并且还能够显示。
一如之前，包含头文件，链接库，使用 irr 命名空间。
*/
#include <irrlicht.h>
#include <iostream>
using namespace irr;
#pragma comment(lib, "Irrlicht.lib")
/*
在这个例子中，我们的一个目标是使用键盘控制场景节点的移动。
当我们按键的时候，场景节点需要有移动，所以我们需要保存一个场景节点的指针。
另一个指针是 Irr 设备的指针，因为这次我们要为场景节点做个事件接收器，
还需要设置一个灵活摄象机，所以我们需要保存它。
*/
scene::ISceneNode* node = 0;
IrrlichtDevice* device = 0;

/*
为了获取用户鼠标键盘输入事件，以及 GUI 的各类事件(例如，XX 按钮被按下)并
对其进行相应的处理，我们需要创建一个 EventReceiver 的实体对象，它必须
继承于 IEventReciver 类。对于这个接口类，我们仅需要重载其一个方法：OnEvent();
在引擎中有上列事件消息时，该方法会被自动调用。
在这个例子里，我们仅写出按下 W，S 时的场景节点处理。
*/
class MyEventReceiver : public IEventReceiver
{
public:
virtual bool OnEvent(SEvent event)
{
    /*
    当我们松开 W 或 S 键时，我们将获取场景节点的位置，并且使其在 Y 轴上进行略微的偏
移。
    所以，如果你按 W，节点将向上移动一点，你按 S，节点将向下移动一点。
    */
    if (node != 0 && event.EventType == irr::EET_KEY_INPUT_EVENT&&
      !event.KeyInput.PressedDown)
    {
      switch(event.KeyInput.Key)
      {
      case KEY_KEY_W:
      case KEY_KEY_S:
```

```
        {
          core::vector3df v = node->getPosition();
          v.Y += event.KeyInput.Key == KEY_KEY_W ? 2.0f : -2.0f;
          node->setPosition(v);
        }
        return true;
      }
    }
    return false;
}
};

/*
事件接收器已经完成。现在我们可以创建 Irr 设备并且显示场景节点吧。
我们同时再做一些其他的场景节点，用来比较表现场景之间的不同之处
*/
int main()
{
// 让用户选择设备类型
video::E_DRIVER_TYPE driverType = video::EDT_DIRECT3D9;
printf("Please select the driver you want for this example:\n"\
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
    " (f) NullDevice\n (otherKey) exit\n\n");
char i;
std::cin >> i;
switch(i)
{
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
    case 'c': driverType = video::EDT_OPENGL;    break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO;break;
    case 'f': driverType = video::EDT_NULL;      break;
    default: return 0;
}
// 创建设备
MyEventReceiver receiver;
IrrlichtDevice* device = createDevice( driverType, core::dimension2d<s32>(640, 480),
    16, false, false, false, &receiver);
if (device == 0)
    return 1;

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

```
scene::ISceneManager* smgr = device->getSceneManager();

/*
创建一个场景节点，这个场景节点是准备提供给我们去进行事件处理的。
我们创建一个球形的节点，这种简单的几何节点是无需导入模型或自定义顶点属性的。
节点创建完毕后，我们设置节点的位置和纹理。
因为我们的场景中不需要动态光照，所以我们屏蔽掉光照。（否则模型会变黑）
*/
node = smgr->addSphereSceneNode();
node->setPosition(core::vector3df(0,0,30));
node->setMaterialTexture(0, driver->getTexture("../../media/wall.bmp"));
node->setMaterialFlag(video::EMF_LIGHTING, false);

/*
现在我们创建一些其他的节点，并为其设置一个场景节点动画器。
场景节点动画器可以依附于各种场景节点，例如 Mesh,公告版，灯光，摄象机等都可以。
它的作用是对场景节点的一些属性进行修改调整。
它不仅能够更改场景节点的位置，还可以更改场景节点中的一个对象纹理等操作。
我们创建一个立方体场景节点，并且对其安排一个圆形轨道的场景节点动画器，
这样我们就实现了让立方体场景节点围着我们的球形场景节点旋转的功能。
*/
scene::ISceneNode* n = smgr->addCubeSceneNode();
if (n)
{
    n->setMaterialTexture(0, driver->getTexture("../../media/t351sml.jpg"));
    n->setMaterialFlag(video::EMF_LIGHTING, false);
    scene::ISceneNodeAnimator* anim =
      smgr->createFlyCircleAnimator(core::vector3df(0,0,30), 20.0f);
    n->addAnimator(anim);
    anim->drop();
}
/*
最后我们再增加一个场景节点，用它来显示一个 md2 格式的模型，
我们用一个场景节点动画器来绑定他，使其在两点间跑动。
*/
scene::IAnimatedMeshSceneNode*                    anms                    =
smgr->addAnimatedMeshSceneNode(smgr->getMesh("../../media/sydney.md2"));
if (anms)
{
    scene::ISceneNodeAnimator* anim =
      smgr->createFlyStraightAnimator(core::vector3df(100,0,60),
      core::vector3df(-100,0,60), 2500, true);
    anms->addAnimator(anim);
    anim->drop();
```

```
    /*
    为了让模型看起来完整，我们设置场景节点动画器让模型在两桢之间进行循环，之后设置了
    模型的骨骼动画速度和纹理，并将模型以及其动作反转了 180 度(原模型动作是面朝右走，
    旋转后模型动作是面朝左走，符合人物移动路径了)。
    之后我们设置 setMD2Animation()函数来替代之前的设置浈循环速率和设置动画速度来进行动画控制。
    但是需要注意的是 setMD2Animation()函数仅仅适用于 MD2 格式文件。
    另外需要说明的是，虽然您知道了如何考试播放动画，但我还是建议您不要尝试将每秒的
    动画桢速率开的过高……请珍惜 GPU。
    */
    anms->setMaterialFlag(video::EMF_LIGHTING, false);
    anms->setFrameLoop(160, 183);
    anms->setAnimationSpeed(40);
    anms->setMD2Animation(scene::EMAT_RUN);
    anms->setRotation(core::vector3df(0,180.0f,0));
    anms->setMaterialTexture(0, driver->getTexture("../../media/sydney.bmp"));
}

/*
为了能够看到场景的全貌，我们创建了第一人称射击风格的摄象机
并且关闭了鼠标图标
*/
scene::ICameraSceneNode * cam = smgr->addCameraSceneNodeFPS(0, 100.0f, 100.0f);
device->getCursorControl()->setVisible(false);
/*
用 GUIManager 增加了一个 Irr 的 Logo 图。
*/
device->getGUIEnvironment()->addImage(
    driver->getTexture("../../media/irrlichtlogoalpha2.tga"),
    core::position2d<s32>(10,10));
/*
我们完成了所有的事情，现在我们开始绘制吧。我们同样的，在程序窗口标题上
绘制 FPS。
*/
int lastFPS = -1;
while(device->run())
{
    driver->beginScene(true, true, video::SColor(255,113,113,133));
    // 这里绘制的 3D 场景
    smgr->drawAll();
    // 这里绘制的 GUI 环境
    device->getGUIEnvironment()->drawAll();
```

```
    driver->endScene();
    int fps = driver->getFPS();
    if (lastFPS != fps)
    {
      core::stringw tmp(L"Movement Example - Irrlicht Engine [");
      tmp += driver->getName();
      tmp += L"] fps: ";
      tmp += fps;
      device->setWindowCaption(tmp.c_str());
      lastFPS = fps;
    }
}
/*
最后，释放 Irr 设备。
*/
device->drop();

return 0;
}
```

Irrlicht 之 05.UserInterface

```
/*
这个例子告诉我们如何为 Irr 引擎创建一个用户图形界面。
它将告诉我们如何去创建一个窗口，按钮，滚动条，静态字体和列表框。
象以往一样，包含头文件，说明命名空间，链接库。
我们全局变量中保存一个 Irr 设备指针，一个变量(用来存储新创建的窗口坐标数值)
一个 ListBox 的指针。
This tutorial shows how to use the built in User Interface of
the Irrlicht Engine. It will give a brief overview and show
how to create and use windows, buttons, scroll bars, static
texts and list boxes.
*/
#include <irrlicht.h>
#include <iostream>
using namespace irr;
using namespace core;
using namespace scene;
using namespace video;
using namespace io;
using namespace gui;
#ifdef _IRR_WINDOWS_
#pragma comment(lib, "Irrlicht.lib")
#endif

IrrlichtDevice *device = 0;
s32 cnt = 0;
IGUIListBox* listbox = 0;

/*
事件接收器不仅仅接受键盘和鼠标的输入消息，也处理 GUI 事件消息。
例如：按钮按下消息，列表选择消息等等。为了处理这些 GUI 消息，
我们在这里创建了一个事件接收器。
假如我们接收到一个 GUI 事件，我们先从中获得 GUI 控件的 ID，
再获取 GUI 环境的指针，就可以进行处理了。
具体代码如下：
*/
class MyEventReceiver : public IEventReceiver
{
public:
virtual bool OnEvent(SEvent event)
{
    if (event.EventType == EET_GUI_EVENT)
    {
      s32 id = event.GUIEvent.Caller->getID();
```

```
IGUIEnvironment* env = device->getGUIEnvironment();
switch(event.GUIEvent.EventType)
{
/*
如果一个滚动条移动了其位置，我们先判断此滚动条是否我们创建的，
if(id == 104)，若是的话，则我们改变 GUI 环境的透明度。
改变 GUI 环境透明度实际上很容易做到：GUI 环境有一个皮肤对象，
这个对象管理着所有的 GUI 环境的颜色设置，我们简单的遍历了皮肤
里的所有颜色，并改变他们的 Alpha 值。
注：控件的 ID 在创建时会需要传入。
*/
case EGET_SCROLL_BAR_CHANGED:
 if (id == 104)
 {
  s32 pos = ((IGUIScrollBar*)event.GUIEvent.Caller)->getPos();

  for (u32 i=0; i<EGDC_COUNT ; ++i)
  {
   SColor col = env->getSkin()->getColor((EGUI_DEFAULT_COLOR)i);
   col.setAlpha(pos);
   env->getSkin()->setColor((EGUI_DEFAULT_COLOR)i, col);
  }

 }
 break;
/*
如果有按钮按下的消息，我们先判断其 ID 到底是哪个按钮。
若是 ID 为 101 的按纽，我们就关闭引擎。
若是 ID 为 102 的按钮，我们就创建一个小窗口，并且在 ListBox 中添加一行文字，
同时在窗口上加一个静态 Label.
若是 ID 为 103 的按钮，我们就创建一个打开的文件对话框。
同时在 ListBox 中添加一行文字。
注：控件的 ID 在创建时会需要传入。
*/
case EGET_BUTTON_CLICKED:
 if (id == 101)
 {
  device->closeDevice();
  return true;
 }
 if (id == 102)
 {
  listbox->addItem(L"Window created");
  cnt += 30;
```

```
        if (cnt > 200)
          cnt = 0;
        IGUIWindow* window = env->addWindow(
          rect<s32>(100 + cnt, 100 + cnt, 300 + cnt, 200 + cnt),
          false, // modal?
          L"Test window");
        env->addStaticText(L"Please close me",
          rect<s32>(35,35,140,50),
          true, //是否粗体字
          false, // wordwrap?
          window);
        return true;
      }
      if (id == 103)
      {
        listbox->addItem(L"File open");
        env->addFileOpenDialog(L"Please choose a file.");
        return true;
      }
      break;
    default:
      break;
    }
  }
  return false;
}
};

/*
好了，现在开始另外一块有意思的部分。
*/
int main()
{
// 让用户选择设备类型

video::E_DRIVER_TYPE driverType;
printf("Please select the driver you want for this example:\n"\
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
    " (f) NullDevice\n (otherKey) exit\n\n");
char i;
std::cin >> i;
switch(i)
{
```

```
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
    case 'c': driverType = video::EDT_OPENGL;    break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO;break;
    case 'f': driverType = video::EDT_NULL;    break;
    default: return 1;
}
device = createDevice(driverType, core::dimension2d<s32>(640, 480));
if (device == 0)
    return 1;
/*
Irr 设备创建完了，现在我们设置事件处理器。同时保存下设备的 GUI 环境指针。
*/
MyEventReceiver receiver;
device->setEventReceiver(&receiver);
device->setWindowCaption(L"Irrlicht Engine - User Interface Demo");
video::IVideoDriver* driver = device->getVideoDriver();
IGUIEnvironment* env = device->getGUIEnvironment();
/*
为了使字体看起来更好看一些，我们读取一个外部的字体库,
并且将其设置为 GUI 环境皮肤的默认字体。
但是为了按钮看起来与众不同，所以我们将按钮上的文字特殊化，
使按钮上的文字依旧使用内嵌的标准字体。
*/
IGUISkin* skin = env->getSkin();
IGUIFont* font = env->getFont("../../media/fontcourier.bmp");
if (font)
    skin->setFont(font);
skin->setFont(env->getBuiltInFont(), EGDF_BUTTON);
/*
我们增加三个按钮，第一个用来关闭引擎，第二个用来创建一个窗口，第三个用来打
开一个文件对话框。在这里设置 Button 的 ID 以便事件处理器的判别处理。
*/
env->addButton(rect<s32>(10,210,110,210 + 32), 0, 101, L"Quit", L"Exits Program");
env->addButton(rect<s32>(10,250,110,250 + 32), 0, 102, L"New Window", L"Launches a new
Window");
env->addButton(rect<s32>(10,290,110,290 + 32), 0, 103, L"File Open", L"Opens a file");
/*
现在，我们增加一个静态文字 Label 个一个滚动条，用它来修改 GUI 环境的透明度。
我们设置好滚动条的最大值为 255(最大色深值)，之后我们再创建一个 ListBox
*/
env->addStaticText(L"Transparent Control:", rect<s32>(150,20,350,40), true);
IGUIScrollBar* scrollbar = env->addScrollBar(true, rect<s32>(150, 45, 350, 60), 0, 104);
```

```
scrollbar->setMax(255);
// 设置滚动条位置
scrollbar->setPos(env->getSkin()->getColor(EGDC_WINDOW).getAlpha());
env->addStaticText(L"Logging ListBox:", rect<s32>(50,80,250,100), true);
listbox = env->addListBox(rect<s32>(50, 110, 250, 180));
env->addEditBox(L"Editable Text", rect<s32>(350, 80, 550, 100));
// 增加一个 Irr 引擎 Logo
env->addImage(driver->getTexture("../../media/irrlichtlogo2.png"),
      position2d<int>(10,10));

/*
全部设置 OK 了，现在我们来进行绘制
*/
while(device->run() && driver)
if (device->isWindowActive())
{
    driver->beginScene(true, true, SColor(0,200,200,200));
    env->drawAll();

    driver->endScene();
}
device->drop();
return 0;
}
```

irrlicht 之 06.2DGraphics

```
/*
这个例子告诉我们使用 Irr 引擎中的 2D 图形。
如何去绘制 2D 图片，闪色精灵以及不同的字体。
如果你想使用 Irr 引擎做一个 2D 游戏的话，或者你想为自己的 3D 游戏
绘制一个很酷的界面的时候，你会发现本例子很有用。
按照往常，我们先包含头文件，使用命名空间，链接 Lib 库。
*/
#include <irrlicht.h>
#include <iostream>
using namespace irr;
#pragma comment(lib, "Irrlicht.lib")

int main()
{
// 我们让用户选择设备类型
video::E_DRIVER_TYPE driverType;
printf("Please select the driver you want for this example:\n"\
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
    " (f) NullDevice\n (otherKey) exit\n\n");

char i;
std::cin >> i;
switch(i)
{
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
    case 'c': driverType = video::EDT_OPENGL;    break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO;break;
    case 'f': driverType = video::EDT_NULL;      break;
    default: return 0;
}
// 创建 Irr 设备
IrrlichtDevice *device = createDevice(driverType,
    core::dimension2d<s32>(512, 384));
if (device == 0)
    return 1;
device->setWindowCaption(L"Irrlicht Engine - 2D Graphics Demo");
video::IVideoDriver* driver = device->getVideoDriver();
/*
本例子中的所有的 2D 图形都是整理在一个叫 2ddemo.bmp 图形素材里面的。
因为我们想绘制一个变色的精灵，我们需要先把素材读入内存，然后告诉引擎，
```

我们想对该素材中的一部分进行变色处理。
我们在例子中，并没有直接告诉引擎图形素材的背景色是什么颜色，而是告诉
引擎，让它自己去图形素材的 0,0 点去找那个颜色，而那种颜色就是背景色，予以处理。
当然，你也可以自己指定背景色。这样写

```
driver->makeColorKeyTexture(images, video::SColor(0,0,0,0))
```

这样的话，引擎就会把图形的黑色的点全部设置为透明(以黑色为背景色抠除了)
makeColorKeyTexture()这个函数，实际上就是指定图形的背景色，予以透明处理。

```
*/
video::ITexture* images = driver->getTexture("../../media/2ddemo.bmp");
driver->makeColorKeyTexture(images, core::position2d<s32>(0,0));

/*
```

我们想绘制两种不同风格的字体，所以我们先将这两种字体都读取进来，其中，
第一种字体是引擎的内置字体。第二种是我们从外部字体库读取的。
同样，我们定义了两个矩形，这两个矩形指定了纹理图片中两个小精灵的位置。
（译者注：实际上这俩矩形坐标就是指明了图片中的那俩个带小怪物动作的方块位置）

```
*/
gui::IGUIFont* font = device->getGUIEnvironment()->getBuiltInFont();
gui::IGUIFont*                                font2                                =
device->getGUIEnvironment()->getFont("../../media/fonthaettenschweiler.bmp");
core::rect<s32> imp1(349,15,385,78);
core::rect<s32> imp2(387,15,423,78);

/*
```

所有先前事情准备好了，现在我们开始在渲染循环中开始绘制所有的一切。
在 beginScene 和 endScene 中调用这些绘制工作，使他们在每桢中绘制显示。
本例中，我们仅进行 2D 图形渲染，但是同时也是允许进行 3D 图形渲染的，我们
在本例中不再做演示了。你若想实验一下的话，就绘制一些 3D 图形或者
设置一个场景管理器读取一个场景节点素材进行绘制吧。

```
*/
while(device->run() && driver)
{
    if (device->isWindowActive())
    {
      u32 time = device->getTimer()->getTime();
      driver->beginScene(true, true, video::SColor(0,120,102,136));
      /*
```

首先，我们绘制 3 个精灵，我们已经使用 makeColorKeyTexture()函数进行透明
背景的设置。
现在开始 2D 图形的绘制，draw2DImage 函数的最后一个参数是是否打开 Alpha 通道混合.
因为我们的第三张图需要实现变色，所以我们打开了该功能。
(实际上前两个背景和精灵的绘制时可以关闭该选项，和白色的混合是无意义的)
倒数第二个参数定义了精灵的混合颜色。（该颜色会与精灵在图片中的颜色进行混合）

所以我们现在可以实现颜色变换的功能了，只到将倒数第二个参数设置为
随时间改变，我们就获得了变色的精灵。
*/

```
// 绘制背景
driver->draw2DImage(images, core::position2d<s32>(50,50),
  core::rect<s32>(0,0,342,224), 0,
  video::SColor(255,255,255,255), true);
// 绘制一个不变色的精灵，因为我们设置混合颜色为白色，当白色和
// 精灵原本的橘红色混合后，颜色不变。
driver->draw2DImage(images, core::position2d<s32>(164,125),
  (time/500 % 2) ? imp1 : imp2, 0,
  video::SColor(255,255,255,255), true);
// 绘制一个随时间变色的精灵
driver->draw2DImage(images, core::position2d<s32>(270,105),
  (time/500 % 2) ? imp1 : imp2, 0,
  video::SColor(255,(time) % 255,255,255), true);
/*
简单的绘制文字
*/
// 绘制文字
if (font)
font->draw(L"This demo shows that Irrlicht is also capable of drawing 2D graphics.",
  core::rect<s32>(130,10,300,50),
  video::SColor(255,255,255,255));
if (font2)
font2->draw(L"Also mixing with 3d graphics is possible.",
  core::rect<s32>(130,20,300,60),
  video::SColor(255,time % 255,time % 255,255));
/*
```

最后，我们绘制一个 Irrlicht 引擎的 logo，并且在鼠标指针图标上绘制了一个半透明 2D 矩形。

```
*/
// draw logo
driver->draw2DImage(images, core::position2d<s32>(10,10),
  core::rect<s32>(354,87,442,118));
// draw transparent rect under cursor
core::position2d<s32> m = device->getCursorControl()->getPosition();
driver->draw2DRectangle(video::SColor(100,255,255,255),
  core::rect<s32>(m.X-20, m.Y-20, m.X+20, m.Y+20));
driver->endScene();
  }
}
/*
```

这样就结束了，希望不会理解起来太难。
*/
device->drop();
return 0;
}

irrlicht 之 07.Collision

```
/*
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>
#include <iostream>
using namespace irr;
#pragma comment(lib, "Irrlicht.lib")

int main()
{
// 用户进行设备选择
video::E_DRIVER_TYPE driverType;
printf("Please select the driver you want for this example:\n"\
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
    " (f) NullDevice\n (otherKey) exit\n\n");
char i;
std::cin >> i;
switch(i)
{
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
    case 'c': driverType = video::EDT_OPENGL;    break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO;break;
    case 'f': driverType = video::EDT_NULL;      break;
```

```
    default: return 0;
}
// 创建 irr 设备
IrrlichtDevice *device =
    createDevice(driverType, core::dimension2d<s32>(640, 480), 16, false);

if (device == 0)
    return 1;
video::IVideoDriver* driver = device->getVideoDriver();
scene::ISceneManager* smgr = device->getSceneManager();

device->getFileSystem()->addZipFileArchive("../../media/map-20kdm2.pk3");

scene::IAnimatedMesh* q3levelmesh = smgr->getMesh("20kdm2.bsp");
scene::ISceneNode* q3node = 0;

if (q3levelmesh)
    q3node = smgr->addOctTreeSceneNode(q3levelmesh->getMesh(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。
*/
scene::ITriangleSelector* selector = 0;

if (q3node)
{
    q3node->setPosition(core::vector3df(-1350,-130,-1400));
    selector = smgr->createOctTreeTriangleSelector(q3levelmesh->getMesh(0), q3node, 128);
    q3node->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。
*/
```
scene::ICameraSceneNode* camera =
    smgr->addCameraSceneNodeFPS(0, 100.0f, 300.0f, -1, 0, 0, true);
camera->setPosition(core::vector3df(-100,50,-150));
scene::ISceneNodeAnimator* anim = smgr->createCollisionResponseAnimator(
    selector, camera, core::vector3df(30,50,30),
    core::vector3df(0,-3,0),
    core::vector3df(0,50,0));
camera->addAnimator(anim);
anim->drop();
/*
```
因为碰撞检测不是 Irr 引擎的主要研究部分。我将在一会告诉大家如何去实现两种不同的鼠标拣选，但是在此之前，我们再场景加一些东西。我需要三个角色用来协助我的拣选测试，还需要给它们一点动态光照亮他们。另外，我们在鼠标处绘制一个公告版，让它去替代鼠标吧。
*/
```
// 屏蔽鼠标指针
device->getCursorControl()->setVisible(false);
// 增加一个公告版场景节点
```

```
scene::IBillboardSceneNode * bill = smgr->addBillboardSceneNode();
bill->setMaterialType(video::EMT_TRANSPARENT_ADD_COLOR );
bill->setMaterialTexture(0, driver->getTexture("../../media/particle.bmp"));
bill->setMaterialFlag(video::EMF_LIGHTING, false);
bill->setMaterialFlag(video::EMF_ZBUFFER, false);
bill->setSize(core::dimension2d<f32>(20.0f, 20.0f));
// 增加了三个活生生的仙子角色
video::SMaterial material;
material.Textures[0] = driver->getTexture("../../media/faerie2.bmp");
material.Lighting = true;
scene::IAnimatedMeshSceneNode* node = 0;
scene::IAnimatedMesh* faerie = smgr->getMesh("../../media/faerie.md2");
if (faerie)
{
    node = smgr->addAnimatedMeshSceneNode(faerie);
    node->setPosition(core::vector3df(-70,0,-90));
    node->setMD2Animation(scene::EMAT_RUN);
    node->getMaterial(0) = material;
    node = smgr->addAnimatedMeshSceneNode(faerie);
    node->setPosition(core::vector3df(-70,0,-30));
    node->setMD2Animation(scene::EMAT_SALUTE);
    node->getMaterial(0) = material;
    node = smgr->addAnimatedMeshSceneNode(faerie);
    node->setPosition(core::vector3df(-70,0,-60));
    node->setMD2Animation(scene::EMAT_JUMP);
    node->getMaterial(0) = material;
}
material.Textures[0] = 0;
material.Lighting = false;
// 增加一个光源
smgr->addLightSceneNode(0, core::vector3df(-60,100,400),
    video::SColorf(1.0f,1.0f,1.0f,1.0f),
    2000.0f);

/*
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。
*/

scene::ISceneNode* selectedSceneNode = 0;
scene::ISceneNode* lastSelectedSceneNode = 0;

int lastFPS = -1;
```

```
while(device->run())
if (device->isWindowActive())
{
    driver->beginScene(true, true, 0);
    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)
    */
    core::line3d<f32> line;
    line.start = camera->getPosition();
    line.end = line.start + (camera->getTarget() - line.start).normalize() * 1000.0f;
    core::vector3df intersection;
    core::triangle3df tri;
    if (smgr->getSceneCollisionManager()->getCollisionPoint(
      line, selector, intersection, tri))
    {
      bill->setPosition(intersection);

      driver->setTransform(video::ETS_WORLD, core::matrix4());
      driver->setMaterial(material);
      driver->draw3DTriangle(tri, video::SColor(0,255,0,0));
    }

    /*
    另外一个 Irr 引擎支持的拣选是基于场景节点包围盒的拣选。每一个场景节点在创建时都必定
    有自己的包围盒，所以我们可以很快的获取摄象机拣选的那个场景节点。
    好，我们再次设置一个碰撞管理器，当我们选择到一个非公告版和地图的场景节点时，让它材质高亮。
    */
    selectedSceneNode =
smgr->getSceneCollisionManager()->getSceneNodeFromCameraBB(camera);
    if (lastSelectedSceneNode)
      lastSelectedSceneNode->setMaterialFlag(video::EMF_LIGHTING, true);
    if (selectedSceneNode == q3node || selectedSceneNode == bill)
      selectedSceneNode = 0;
```

```
    if (selectedSceneNode)
      selectedSceneNode->setMaterialFlag(video::EMF_LIGHTING, false);
    lastSelectedSceneNode = selectedSceneNode;

    /*
    这样，拣选的任务就完成了，我们需要做的就是 EndScene 了。
    */
    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;
}
```

Irrlicht 之 08.SpecialFx

/*
这个例子告诉我们如何去做一些特效。我会告诉大家如何去使用深度缓冲进行阴影计算，粒子系统，
公告版，动态光照以及制作水表场景节点。
我们依旧象之前的例子一样开始，但是注意，因为我们想为角色创建一个动态光影，
所以这次我们在创建设备的时候 CreateDevice()应当把最后一个参数是否开启深度缓冲这项设置为“打开”。
如果你嫌进行动态光影计算之后渲染太慢了，你就设置它为 false 关闭。Irr 引擎会自动检查你的显卡是否支持深度缓冲，如果你的电脑硬件不支持深度缓冲，它会自动将你关闭掉它。
*/

```
#include <irrlicht.h>
#include <iostream>
using namespace irr;
#pragma comment(lib, "Irrlicht.lib")
int main()
{
// 询问用户是否要使用动态阴影
char i;
printf("Please press 'y' if you want to use realtime shadows.\n");
std::cin >> i;
bool shadows = (i == 'y');
// 询问用户希望创建的设备类型
video::E_DRIVER_TYPE driverType;
printf("Please select the driver you want for this example:\n"\
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
    " (f) NullDevice\n (otherKey) exit\n\n");
std::cin >> i;
switch(i)
{
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
    case 'c': driverType = video::EDT_OPENGL;    break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO;break;
    case 'f': driverType = video::EDT_NULL;      break;
    default: return 1;
}

IrrlichtDevice *device =
    createDevice(driverType, core::dimension2d<s32>(640, 480),
    16, false, shadows);
if (device == 0)
```

```
    return 1;
video::IVideoDriver* driver = device->getVideoDriver();
scene::ISceneManager* smgr = device->getSceneManager();
/*
在本例中，我们读取一个.3ds 文件。这是我使用 Anim08or 创建的一个小房子模型，
因为 Irr 引擎不支持.an8 格式，于是我把它导出为.3ds 格式。我的 3D 美术实在很糟糕，
所以我制作的纹理贴图看起来肯定会不太漂亮。幸好我是一个程序员，而非美术人员，
更幸运的是 Irr 引擎可以帮我提供一个很绚的材质贴图，方法是：只要使用 Mesh 管理器
为 Mesh 创建一个平面材质贴图就可以了。如果你想看我用 Anim08or 做的采制贴图的话，
那只要把 makePlanarTextureMapping()这行注释掉就可以了。我同样不会讲述如何使用
Anim8or
创建材质，因为 Anim8or 中有我非常不喜欢的灯光设置。
*/
scene::IAnimatedMesh* mesh = smgr->getMesh(
    "../../media/room.3ds");
smgr->getMeshManipulator()->makePlanarTextureMapping(
    mesh->getMesh(0), 0.004f);
scene::ISceneNode* node = 0;
node = smgr->addAnimatedMeshSceneNode(mesh);
node->setMaterialTexture(0, driver->getTexture("../../media/wall.jpg"));
node->getMaterial(0).SpecularColor.set(0,0,0,0);
/*
现在，我们开始制作第一个特效：生动的水。
它是这样工作的：将一个 Mesh 制作成水表场景节点,然后使它象水面一样起波纹。
如果我们让这个场景节点用一个象 EMT_REFLECTION_2_LAYER 这样的优秀的材质，水表看起
来就很酷了。
象导入 Mesh 一样，我们创建一个丘陵平面状的 Mesh，用它来做为水面。
之后我们为水面设置两层纹理，一层做为水表，一层做为水底物体透视纹理。
最后我们为水面设置一个很优秀的材质使它看起来更象一些。
（译者注：用户在阅读该部分时，请多进行参数修改调试以了解各函数作用）
*/

mesh = smgr->addHillPlaneMesh("myHill",
    core::dimension2d<f32>(20,20),
    core::dimension2d<s32>(40,40), 0, 0,
    core::dimension2d<f32>(0,0),
    core::dimension2d<f32>(10,10));
node = smgr->addWaterSurfaceSceneNode(mesh->getMesh(0), 3.0f, 300.0f, 30.0f);
node->setPosition(core::vector3df(0,7,0));
node->setMaterialTexture(0, driver->getTexture("../../media/stones.jpg"));
node->setMaterialTexture(1, driver->getTexture("../../media/water.jpg"));
node->setMaterialType(video::EMT_REFLECTION_2_LAYER);
/*
第二个特效是非常简单基础的，我敢打赌你在一些 Irr 引擎例子中一定见过这样的情景：
```

一个透明公告板和一个动态光线结合起来实现一个动态的灯光。我们创建一个灯光场景节点，使它旋转飞行。并为它绑定一个公告板场景节点，用公告版来表示当前灯光光源所在地。
*/
```
// 创建灯光
node = smgr->addLightSceneNode(0, core::vector3df(0,0,0),
    video::SColorf(1.0f, 0.6f, 0.7f, 1.0f), 600.0f);
scene::ISceneNodeAnimator* anim = 0;
anim = smgr->createFlyCircleAnimator (core::vector3df(0,150,0),250.0f);
node->addAnimator(anim);
anim->drop();
// 在灯光上绑定一个公告版以表示光源点
node = smgr->addBillboardSceneNode(node, core::dimension2d<f32>(50, 50));
node->setMaterialFlag(video::EMF_LIGHTING, false);
node->setMaterialType(video::EMT_TRANSPARENT_ADD_COLOR);
node->setMaterialTexture(0, driver->getTexture("../../media/particlewhite.bmp"));
```
/*
下一个特效就更加有意思了：它就是粒子系统。Irr 引擎中的粒子系统很容易扩展而且非常容易使用。
我们可以创建一个粒子系统场景节点，然后在其中创建一个粒子发射器，用它来发射粒子。粒子发射器有一些通用而且灵活的参数，例如粒子的方向，个数，颜色等。而且我们还有不同的
粒子发射器可供选择，例如点状粒子发射器会让粒子从一个点进行发射。如果 Irr 引擎中的粒子
发射器类型不能满足你的需要，你当然也可以很容易创建一个你所需要的发射器。只要你继承
IParticleEmitter 接口实现一个自定义类就可以了。在这里例子中我们创建一个盒子状的粒子发射器，
就是说粒子在一个盒子形状内随机生成。创建函数的参数分别定义了发射器盒子属性，粒子方向，
每秒生成粒子的个数范围，粒子颜色，粒子存活时间等。
因为把所有属性都放置在粒子发射器系统中，会使粒子发射器创建时令人厌烦。
所以在粒子飞行的时候，也有些粒子属性可以被加入粒子系统中。例如，风向，重力等。
在这个例子中我们加入了一个影响器，让它对粒子进行印象，它改变了粒子的颜色属性：使粒子颜色拥有了淡出的属性。就象粒子发射器一样，你可以创建自己的粒子影响器类型，只要你
继承一个 IParticleAffector 的类，并实现其 AddAffector()函数就 OK 了。

在我们为粒子设置一个优秀的材质之后，我们就获得了一个很逼真的火焰效果了。
经过调整材质，纹理，粒子发射器和粒子影响器的参数，我们就很轻松的可以创建出烟雾，雨水，爆炸，雪这种效果了。
*/
```
// 创建一个粒子系统
scene::IParticleSystemSceneNode* ps = 0;
ps = smgr->addParticleSystemSceneNode(false);
```

```
ps->setPosition(core::vector3df(-70,60,40));
ps->setScale(core::vector3df(2,2,2));
ps->setParticleSize(core::dimension2d<f32>(20.0f, 20.0f));
scene::IParticleEmitter* em = ps->createBoxEmitter(
    core::aabbox3d<f32>(-7,0,-7,7,1,7),
    core::vector3df(0.0f,0.06f,0.0f),
    80,100,
    video::SColor(0,255,255,255), video::SColor(0,255,255,255),
    800,2000);
ps->setEmitter(em);
em->drop();
scene::IParticleAffector* paf =
    ps->createFadeOutParticleAffector();
ps->addAffector(paf);
paf->drop();
ps->setMaterialFlag(video::EMF_LIGHTING, false);
ps->setMaterialTexture(0, driver->getTexture("../../media/fire.bmp"));
ps->setMaterialType(video::EMT_TRANSPARENT_VERTEX_ALPHA);
/*
我们最后一个特效就是，我们为一个动画模型设置一个动态阴影。
我们读取一个 DirectX 的.x 格式模型。并将它放置到我们的世界中。之后我们需要简单的
调用 addShadowVolumeSceneNode() 函数，我们全局所有的阴影都可以调用
ISceneManager::setShadowColor()
来设置它们的颜色，瞧，这就是我们的动态阴影。
因为我们的角色模型有点小，我们调用 setScale()函数使它看起来大一些。
又因为我们的角色模型被一个动态光线照着，我们需要将其法线进行格式化，保证光照计算
的正确性。
如果一个动态光线照射下的模型缩放比例不是(1, 1, 1)的话，就必须做这一步。
否则的话，你就会发现由于法线的缩放而导致阴影也变的过亮或者过暗。
*/
// 为场景增加一个动画模型
mesh = smgr->getMesh("../../media/dwarf.x");
scene::IAnimatedMeshSceneNode* anode = 0;
anode = smgr->addAnimatedMeshSceneNode(mesh);
anode->setPosition(core::vector3df(-50,20,-60));
anode->setAnimationSpeed(15);
// 增加一个阴影场景节点
anode->addShadowVolumeSceneNode();
smgr->setShadowColor(video::SColor(150,0,0,0));
// 把模型进行一下放大，并且将其法线格式化，以保证正确的光照计算。
anode->setScale(core::vector3df(2,2,2));
anode->setMaterialFlag(video::EMF_NORMALIZE_NORMALS, true);
/*
最后我们简单的将一些绘制出来，就 OK 了
```

```
*/
scene::ICameraSceneNode* camera = smgr->addCameraSceneNodeFPS();
camera->setPosition(core::vector3df(-50,50,-150));
// 关闭鼠标显示
device->getCursorControl()->setVisible(false);

int lastFPS = -1;
while(device->run())
if (device->isWindowActive())
{
    driver->beginScene(true, true, 0);
    smgr->drawAll();
    driver->endScene();
    int fps = driver->getFPS();
    if (lastFPS != fps)
    {
      core::stringw str = L"Irrlicht Engine - SpecialFX example [";
      str += driver->getName();
      str += "] FPS:";
      str += fps;
      device->setWindowCaption(str.c_str());
      lastFPS = fps;
    }
}
device->drop();
return 0;
}
```

irrlicht 之 09.MeshViewer

```
/*
这个例子告诉我们如何使用引擎去创建一个更加复杂的应用程序。我们使用 Irr 的场景管理器和用户界面 API
创建一个简单的 Mesh 查看器。这个例子告诉我们如何去创建按钮，窗口，工具栏，菜单，下拉列表，编辑框，
图象，消息窗口，天空盒，以及如何去使用 Irr 引擎中完善的 XML 阅读器进行解析一个 XML 文件。
我们依旧象其他的例子一样：包含所有必要的头文件，为编译器一个 commment 命令要求它链接正确的 lib.
之后我们定义了一些全局变量。同样的，我们还使用了两个"using namespace"声明来减少麻烦。
在本例中，我们使用了大量的 GUI 命名空间内的函数。
*/
#include <irrlicht.h>
#include <iostream>

using namespace irr;
using namespace gui;
#pragma comment(lib, "Irrlicht.lib")

IrrlichtDevice *Device = 0;
core::stringc StartUpModelFile;
core::stringw MessageText;
core::stringw Caption;
scene::IAnimatedMeshSceneNode* Model = 0;
scene::ISceneNode* SkyBox = 0;
scene::ICameraSceneNode* Camera[2] = { 0, 0};
/*
切换两种不同的摄象机
*/
void setActiveCamera ( scene::ICameraSceneNode* newActive )
{
if ( 0 == Device )
    return;
scene::ICameraSceneNode* active = Device->getSceneManager()->getActiveCamera ();
newActive->setInputReceiverEnabled ( true );
Device->getSceneManager()->setActiveCamera ( newActive );
}
/*
下面的三个函数做了几个 Mesh 查看器的功能。
第一个 showAboutText()函数简单的显示了一个消息框，其标题和内容都在 XML 文件中进行动态读取。
这个文本将在启动时候被存储在 MessageText，Caption 两个变量中。
```

```
*/
void showAboutText()
{
// 创建一个模态消息窗口，其中信息从 XML 文件中读取
Device->getGUIEnvironment()->addMessageBox(
    Caption.c_str(), MessageText.c_str());
}

/*
第二个函数是 loadModel()，它能读取一个模型并且使用场景管理器 addAnimatedMeshSceneNode 再将它显示出来。
这并不复杂是吧。当然，在模型无法读取时，会显示一个简单的警告消息窗口。
*/
void loadModel(const c8* fn)
{
// 根据文件格式进行区分处理
core::stringc filename ( fn );
core::stringc extension;
// 获取 extension 的后缀名
core::getFileNameExtension ( extension, filename );
// 把 extension 变成小写字母
extension.make_lower();
// 支持下列格式的纹理文件
if ( extension == ".jpg" ||
      extension == ".png" ||
      extension == ".tga" ||
      extension == ".pcx" ||
      extension == ".psd" ||
      extension == ".bmp"
    )
{
    video::ITexture * texture = Device->getVideoDriver()->getTexture( filename.c_str() );
    if ( texture && Model )
    {
     // 重新加载纹理
     Device->getVideoDriver()->removeTexture ( texture );
     texture = Device->getVideoDriver()->getTexture( filename.c_str() );
     Model->setMaterialTexture ( 0, texture );
    }
    return;
}
// 如果是一个压缩文件，则将它放置到文件管理器系统中进行处理
if ( extension == ".pk3" ||
      extension == ".zip"
```

```
    )
{
    Device->getFileSystem()->addZipFileArchive( filename.c_str () );
    return;
}
// 将模型读取到引擎中
if (Model)
    Model->remove();
Model = 0;
scene::IAnimatedMesh* m = Device->getSceneManager()->getMesh( filename.c_str() );
if (!m)
{
    // 如果模型 Load 失败，则弹出警告窗口
    if (StartUpModelFile != filename)
      Device->getGUIEnvironment()->addMessageBox(
      Caption.c_str(), L"The model could not be loaded. " \
      L"Maybe it is not a supported file format.");
    return;
}
// 设置默认材质参数
Model = Device->getSceneManager()->addAnimatedMeshSceneNode(m);
Model->setMaterialFlag(video::EMF_LIGHTING, false);
// Model->setMaterialFlag(video::EMF_BACK_FACE_CULLING, false);
Model->setDebugDataVisible(scene::EDS_OFF);
Model->setAnimationSpeed(30);
}

/*
最后，第三个函数创建了一个工具栏窗口。在这个简单的 Mesh 查看器中，
这个工具栏窗口仅仅包含一个制表子窗口，在其上包含三个改变模型缩放显示的编辑框。
*/
void createToolBox()
{
IGUIEnvironment* env = Device->getGUIEnvironment();
IGUIElement* root = env->getRootGUIElement();
IGUIElement* e = root->getElementFromId(5000, true);
// 为防止重复创建工具箱窗口，所以先进行安全删除
if (e) e->remove();
// 创建工具箱窗口
IGUIWindow* wnd = env->addWindow(core::rect<s32>(600,25,800,480),
    false, L"Toolset", 0, 5000);
// 创建制表窗口以及其中的制表
IGUITabControl* tab = env->addTabControl(
    core::rect<s32>(2,20,800-602,480-7), wnd, true, true);
```

```
IGUITab* t1 = tab->addTab(L"Scale");
IGUITab* t2 = tab->addTab(L"Test");
// 在一号制表上增加三个输入框和一个按钮
env->addEditBox(L"1.0", core::rect<s32>(40,50,130,70), true, t1, 901);
env->addEditBox(L"1.0", core::rect<s32>(40,80,130,100), true, t1, 902);
env->addEditBox(L"1.0", core::rect<s32>(40,110,130,130), true, t1, 903);
env->addButton(core::rect<s32>(10,150,100,190), t1, 1101, L"set");
// 增加一个无效的选择框
env->addCheckBox(true, core::rect<s32>(10,220,200,240), t1, -1, L"Senseless Checkbox");
// 增加一个透明度滚动条以及说明文字
env->addStaticText(L"Transparent Control:", core::rect<s32>(10,240,150,260), true, false, t1);
IGUIScrollBar* scrollbar = env->addScrollBar(true, core::rect<s32>(10,260,150,275), t1, 104);
scrollbar->setMax(255);
// 将 Irr 引擎 Logo 放置在最前方，因为，此时它可能被新创建的工具箱窗口遮挡住了。
// 译者注：作者原意是如上所说，将 Irr 引擎 Logo 移到顶层，防止被新建工具箱遮盖，然而其参数设置错误，
//          666 并非 Irr 引擎 Logo 所在窗口 ID 编号，所以，实际上该效果没有被实现。
root->bringToFront(root->getElementFromId(666, true));
}

/*
为了获取 GUI 管理器的所有消息，我们需要创建一个事件接收器，这件事情很容易，如果有事件发生了，
根据事件呼叫者 ID 和事件类型进行处理就可以了。例如，一个 ID 为 1000 的 Menu 控件被选择到了，
则开启一个文件选择框。
*/
class MyEventReceiver : public IEventReceiver
{
public:
virtual bool OnEvent(SEvent event)
{
    // Esc 键进行摄象机控制权更变
    if (event.EventType == EET_KEY_INPUT_EVENT &&
      event.KeyInput.Key == irr::KEY_ESCAPE &&
      event.KeyInput.PressedDown == false)
    {
      if ( Device )
      {
        scene::ICameraSceneNode * camera = Device->getSceneManager()->getActiveCamera ();
        if ( camera )
        {
          camera->setInputReceiverEnabled ( !camera->isInputReceiverEnabled() );
        }
```

```
        return true;
      }
    }
    if (event.EventType == EET_GUI_EVENT)
    {
      s32 id = event.GUIEvent.Caller->getID();
      IGUIEnvironment* env = Device->getGUIEnvironment();
      switch(event.GUIEvent.EventType)
      {
      case EGET_MENU_ITEM_SELECTED:
        {
          // 如果菜单中一个控件被点击
          IGUIContextMenu* menu = (IGUIContextMenu*)event.GUIEvent.Caller;
          s32 id = menu->getItemCommandId(menu->getSelectedItem());

          switch(id)
          {
          case 100: // File -> Open Model
            env->addFileOpenDialog(L"Please select a model file to open");
            break;
          case 101: // File -> Set Model Archive
            env->addFileOpenDialog(L"Please select your game archive/directory");
            break;
          case 200: // File -> Quit
            Device->closeDevice();
            break;
          case 300: // View -> Skybox
            SkyBox->setVisible(!SkyBox->isVisible());
            break;
          case 400: // View -> Debug Information
            if (Model)
              Model->setDebugDataVisible(Model->isDebugDataVisible() ? scene::EDS_OFF :
scene::EDS_FULL);
            break;
          case 500: // Help->About
            showAboutText();
            break;
          case 610: // View -> Material -> Solid
            if (Model)
              Model->setMaterialType(video::EMT_SOLID);
            break;
          case 620: // View -> Material -> Transparent
            if (Model)
              Model->setMaterialType(video::EMT_TRANSPARENT_ADD_COLOR);
```

```
      break;
    case 630: // View -> Material -> Reflection
      if (Model)
        Model->setMaterialType(video::EMT_SPHERE_MAP);
      break;
    case 1000: // Camera -> Maya Style
      setActiveCamera ( Camera[0] );
      break;
    case 1100: // Camera -> First Person
      setActiveCamera ( Camera[1] );
      break;
    }
                          break;
  }
 case EGET_FILE_SELECTED:
  {
   // 在文件展开对话框中进行选择，读取一个模型文件
   IGUIFileOpenDialog* dialog =
     (IGUIFileOpenDialog*)event.GUIEvent.Caller;
   loadModel(core::stringc(dialog->getFilename()).c_str());
  }
 case EGET_SCROLL_BAR_CHANGED:
  // 控制 UI 的皮肤透明度
  if (id == 104)
  {
   s32 pos = ((IGUIScrollBar*)event.GUIEvent.Caller)->getPos();
   for (s32 i=0; i<irr::gui::EGDC_COUNT ; ++i)
   {
     video::SColor col = env->getSkin()->getColor((EGUI_DEFAULT_COLOR)i);
     col.setAlpha(pos);
     env->getSkin()->setColor((EGUI_DEFAULT_COLOR)i, col);
   }
  }
  break;
 case EGET_COMBO_BOX_CHANGED:
  // 控制反锯齿/纹理过滤模式
  if (id == 108)
  {
   s32 pos = ((IGUIComboBox*)event.GUIEvent.Caller)->getSelected();
   switch (pos)
   {
     case 0:
     if (Model)
     {
```

```
        Model->setMaterialFlag(video::EMF_BILINEAR_FILTER, false);
        Model->setMaterialFlag(video::EMF_TRILINEAR_FILTER, false);
        Model->setMaterialFlag(video::EMF_ANISOTROPIC_FILTER, false);
      }
      break;
      case 1:
      if (Model)
      {
        Model->setMaterialFlag(video::EMF_BILINEAR_FILTER, true);
        Model->setMaterialFlag(video::EMF_TRILINEAR_FILTER, false);
      }
      break;
      case 2:
      if (Model)
      {
        Model->setMaterialFlag(video::EMF_BILINEAR_FILTER, false);
        Model->setMaterialFlag(video::EMF_TRILINEAR_FILTER, true);
      }
      break;
      case 3:
      if (Model)
      {
        Model->setMaterialFlag(video::EMF_ANISOTROPIC_FILTER, true);
      }
      break;
      case 4:
      if (Model)
      {
        Model->setMaterialFlag(video::EMF_ANISOTROPIC_FILTER, false);
      }
      break;
    }
  }
  break;
case EGET_BUTTON_CLICKED:
  switch(id)
  {
  case 1101:     // “set”按纽
    {
      // 设置缩放比例
      gui::IGUIElement* root = env->getRootGUIElement();
      core::vector3df scale;
      core::stringc s;
      s = root->getElementFromId(901, true)->getText();
```

```
            scale.X = (f32)atof(s.c_str());
            s = root->getElementFromId(902, true)->getText();
            scale.Y = (f32)atof(s.c_str());
            s = root->getElementFromId(903, true)->getText();
            scale.Z = (f32)atof(s.c_str());
            if (Model)
              Model->setScale(scale);
          }
          break;
        case 1102: // 菜单上的快捷按钮 Open a model
          env->addFileOpenDialog(L"Please select a model file to open");
          break;
        case 1103: // 菜单上的快捷按钮 Open Help
          showAboutText();
          break;
        case 1104: // 菜单上的快捷按钮 Open ToolSet
          createToolBox();
          break;
        case 1105: // 菜单上的快捷按钮 OSet module archive
          env->addFileOpenDialog(L"Please select your game archive/directory");
          break;
        }
        break;
      }
    }
    return false;
}
};

/*
最简单的工作已经做完了，我们仅需要创建一个 Irr 设备，再创建所有的按钮，菜单，工具拦就 OK 了。
我们象平常一样开动引擎，首先创建一个 Irr 设备。为了让我们的应用程序获取到消息，我们必须把
自己的事件接收器作为一个参数传入。
#ifdef WIN32 这个预处理命令不是必要的，但是我包含了这句，目的是使我们的例子能够在非 Windows
平台上也顺利使用 DirectX.
译者注：本例中并未包含#ifdef WIN32 该句，另外作者在本例中错误单词也较多，怀疑是喝高了或是心情不好吧。
如你所见，例子中使用了一个不常见的函数 IrrlichtDevice::setResizeAble(). 这个函数允许渲染窗口被拉伸缩短，这点在 Mesh 浏览器中是非常有用的。
*/
int main()
```

```
{
// 让用户选择设备类型
video::E_DRIVER_TYPE driverType = video::EDT_DIRECT3D8;
printf("Please select the driver you want for this example:\n"\
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
    " (f) NullDevice\n (otherKey) exit\n\n");
char key;
std::cin >> key;
switch(key)
{
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
    case 'c': driverType = video::EDT_OPENGL;      break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO;break;
    case 'f': driverType = video::EDT_NULL;          break;
    default: return 1;
}
// 创建设备，如果失败则直接返回。
MyEventReceiver receiver;
Device = createDevice(driverType, core::dimension2d<s32>(800, 600),
    16, false, false, false, &receiver);
if (Device == 0)
    return 1;
Device->setResizeAble(true);
Device->setWindowCaption(L"Irrlicht Engine - Loading...");
video::IVideoDriver* driver = Device->getVideoDriver();
IGUIEnvironment* env = Device->getGUIEnvironment();
scene::ISceneManager* smgr = Device->getSceneManager();
driver->setTextureCreationFlag(video::ETCF_ALWAYS_32_BIT, true);
smgr->addLightSceneNode();
smgr->addLightSceneNode(0, core::vector3df(50,-50,100), video::SColorf(1.0f,1.0f,1.0f),20000);
// 设置我们的媒体目录，方便搜索
Device->getFileSystem()->addFolderFileArchive ( "../../media/" );
/*
    下一步就是读取配置文件了。它以 XML 格式存放着，恩，就象这样子：
    <?xml version="1.0"?>
    <config>
      <startUpModel file="some filename" />
      <messageText caption="Irrlicht Engine Mesh Viewer">
        Hello!
      </messageText>
    </config>
```

```
    我们需要在 StartUpModelFile，MessageText，Caption 这些全局变量中的数据，
    现在我们开始使用 Irr 引擎对 XML 文件进行解析读取
*/
// 从 XML 文件中读取配置
io::IXMLReader* xml = Device->getFileSystem()->createXMLReader(
    "config.xml");
while(xml && xml->read())
{
    switch(xml->getNodeType())
    {
    case io::EXN_TEXT:
     // 在这个 XML 文件中，只有唯一的正文信息，就是 messageText
     MessageText = xml->getNodeData();
     break;
    case io::EXN_ELEMENT:
     {
      if (core::stringw("startUpModel") == xml->getNodeName())
       StartUpModelFile = xml->getAttributeValue(L"file");
      else
      if (core::stringw("messageText") == xml->getNodeName())
       Caption = xml->getAttributeValue(L"caption");
     }
     break;
    }
}
if (xml)
    xml->drop(); // 别忘记最后需要删除 XML 读取器
/*
    这些并不难是吧，现在我们将要设置一个好看的字体，并且创建一个菜单。
    我们必须为每个菜单项目创建一个子菜单，例如 menu->addItem(L"File", -1, true, true);
    这样我们就创建了一个名叫 File 的子菜单，其 ID 为-1 表示使用默认 ID 编号规则，其余
的参数表意为
    这个子菜单控件是否可用，最后一个参数是说它本身也有自己的子控件，子菜单
    能够被 menu->getSubMenu(0)这个函数获取，因为"File"这个子菜单的编号为 0。
    注意：子菜单的默认编号规则是从 0 开始递增的。
*/
// 设置一个好看的字体
IGUISkin* skin = env->getSkin();
IGUIFont* font = env->getFont("fonthaettenschweiler.bmp");
if (font)
    skin->setFont(font);
// 创建菜单
gui::IGUIContextMenu* menu = env->addMenu();
menu->addItem(L"File", -1, true, true);
```

```
menu->addItem(L"View", -1, true, true);
menu->addItem(L"Camera", -1, true, true);
menu->addItem(L"Help", -1, true, true);
gui::IGUIContextMenu* submenu;
submenu = menu->getSubMenu(0);
submenu->addItem(L"Open Model File & Texture...", 100);
submenu->addItem(L"Set Model Archive...", 101);
submenu->addSeparator();
submenu->addItem(L"Quit", 200);
submenu = menu->getSubMenu(1);
submenu->addItem(L"toggle sky box visibility", 300);
submenu->addItem(L"toggle model debug information", 400);
submenu->addItem(L"model material", -1, true, true );
submenu = submenu->getSubMenu(2);
submenu->addItem(L"Solid", 610);
submenu->addItem(L"Transparent", 620);
submenu->addItem(L"Reflection", 630);
submenu = menu->getSubMenu(2);
submenu->addItem(L"Maya Style", 1000);
submenu->addItem(L"First Person", 1100);
submenu = menu->getSubMenu(3);
submenu->addItem(L"About", 500);
/*
    在菜单之下，我们需要一个工具栏。这个工具栏应该是一些进行贴图的按钮组成。
    另外我们创建一个下拉选框，恩，其中的选项就没有什么意义啦。
*/
// 创建工具栏
gui::IGUIToolBar* bar = env->addToolBar();
video::ITexture* image = driver->getTexture("open.png");
bar->addButton(1102, 0, L"Open a model",image, 0, false, true);
image = driver->getTexture("tools.png");
bar->addButton(1104, 0, L"Open Toolset",image, 0, false, true);
image = driver->getTexture("zip.png");
bar->addButton(1105, 0, L"Set Model Archive",image, 0, false, true);
image = driver->getTexture("help.png");
bar->addButton(1103, 0, L"Open Help", image, 0, false, true);
// 创建一个没有意义文字选项的下拉选框
gui::IGUIComboBox* box = env->addComboBox(core::rect<s32>(250,4,350,23), bar, 108);
box->addItem(L"No filtering");
box->addItem(L"Bilinear");
box->addItem(L"Trilinear");
box->addItem(L"Anisotropic");
box->addItem(L"Isotropic");
/*
```

为了使编辑器看起来更好一点，我们设置了编辑器 GUI 的 Aplha 透明度，让它默认全不透明。

并把制表窗口显示出来.另外，我们创建了一个显示当前桢率的文本。最后，

改变一下窗口标题文字。

```
*/
// 设置编辑器的 GUI 部分为不透明
for (s32 i=0; i<gui::EGDC_COUNT ; ++i)
{
    video::SColor col = env->getSkin()->getColor((gui::EGUI_DEFAULT_COLOR)i);
    col.setAlpha(255);
    env->getSkin()->setColor((gui::EGUI_DEFAULT_COLOR)i, col);
}
// 创建一个制表窗口并将其显示
createToolBox();
// 创建 FPS 显示文字
IGUIStaticText* fpstext = env->addStaticText(L"", core::rect<s32>(400,4,570,23), true, false, bar);
// 设置窗口标题
Caption += " - [";
Caption += driver->getName();
Caption += "]";
Device->setWindowCaption(Caption.c_str());
/*
    这基本是上程序的全部了，我们在程序开始时把 About 帮助窗口显示出来，
    再读入一个默认的模型，再创建一个天空盒子，一个用户控制的摄象机，
    使程序看起来更加生动有趣一些。最后，在一个标准的绘制循环中一切被绘制出来。
*/
// 显示 About 这个帮助窗口
showAboutText();
// 读取一个默认的模型
loadModel(StartUpModelFile.c_str());
// 增加一个天空盒
SkyBox = smgr->addSkyBoxSceneNode(
    driver->getTexture("irrlicht2_up.jpg"),
    driver->getTexture("irrlicht2_dn.jpg"),
    driver->getTexture("irrlicht2_lf.jpg"),
    driver->getTexture("irrlicht2_rt.jpg"),
    driver->getTexture("irrlicht2_ft.jpg"),
    driver->getTexture("irrlicht2_bk.jpg"));
// 增加一个摄象机场景结点
Camera[0] = smgr->addCameraSceneNodeMaya();
Camera[1] = smgr->addCameraSceneNodeFPS();
setActiveCamera ( Camera[0] );
// 读取 Irr 引擎 Logo
IGUIImage *img =
```

```
    env->addImage(driver->getTexture("irrlichtlogo2.png"),
      core::position2d<s32>(10, driver->getScreenSize().Height - 64));
// 将 Logo 锁定到屏幕的左下方
img->setAlignment(EGUIA_UPPERLEFT, EGUIA_UPPERLEFT, EGUIA_LOWERRIGHT,
EGUIA_LOWERRIGHT);
// 绘制一切
while(Device->run() && driver)
{
    if (Device->isWindowActive())
    {
      driver->beginScene(true, true, video::SColor(150,50,50,50));
      smgr->drawAll();
      env->drawAll();
      driver->endScene();
      core::stringw str(L"FPS: ");
      str.append(core::stringw(driver->getFPS()));
      str += L" Tris: ";
      str.append(core::stringw(driver->getPrimitiveCountDrawn()));
      fpstext->setText(str.c_str());
    }
    else
      Device->yield();
}
Device->drop();
return 0;
}
```

irrlicht 之 10.Shaders

```
/*
这个例子将告诉我们如何使用引擎使用 D3D8，D3D9，OPenGL 中的 Shader 以及
如何使用这些 Shader 创建新的材质类型。例子中同样告诉了我们如何关闭纹理 Mipmap 以及
如何使用一个文字型场景节点。
这个例子并不详细解释 Shaders 如何工作，如果你想研究它内部的实现原理，我推荐
你去读一下 D3D 或者 OPenGl 文档或者相关书籍来深入了解 Shader。
首先，我们需要象之前的例子一样包含一些文件，设置命名空间这些。
*/
#include <irrlicht.h>
#include <iostream>

using namespace irr;
#pragma comment(lib, "Irrlicht.lib")

/*
因为我们在本例中想使用一些有趣的 Shaders，所以我先为它们设置一些数据使 Shaders 能计算出更美丽的颜色。
在本例中，我们使用一些简单的顶点 Shader,它将会根据摄象机的位置计算顶点的颜色。
为了实现这些功能，我们需要为 Shader 定义以下数据：转换法线的逆世界矩阵，转换坐标的剪切矩阵，
为了计算光照角度和颜色而需要的摄象机位置和物体世界矩阵中的位置。
为了在每一桢中都为 Shader 提供这些信息，我们不得不继承 IShaderConstantSetCallBack 接口类并
实现其中唯一的函数 OnSetConstanes().这个函数将会在每次设置材质的时候被调用。
IMaterialRendererServices 接口类中的 setVertexShaderConstant()函数用来设置 Shaders 所需要的数据。
如果用户选择使用高级着色语言代替本例中的汇编语言，你必须将变量名做为一个参数传入。
*/
IrrlichtDevice* device = 0;
bool UseHighLevelShaders = false;
class MyShaderCallBack : public video::IShaderConstantSetCallBack
{
public:
virtual void OnSetConstants(video::IMaterialRendererServices* services, s32 userData)
{
    video::IVideoDriver* driver = services->getVideoDriver();
    // 设置逆向世界矩阵
    // 如果我们使用高级着色（用户可以在程序开始时进行选择），我们在此必须设置其名称。
    core::matrix4 invWorld = driver->getTransform(video::ETS_WORLD);
    invWorld.makeInverse();
    if (UseHighLevelShaders)
```

```
    services->setVertexShaderConstant("mInvWorld", invWorld.pointer(), 16);
  else
    services->setVertexShaderConstant(invWorld.pointer(), 0, 4);
  // 设置剪切矩阵
  core::matrix4 worldViewProj;
  worldViewProj = driver->getTransform(video::ETS_PROJECTION);
  worldViewProj *= driver->getTransform(video::ETS_VIEW);
  worldViewProj *= driver->getTransform(video::ETS_WORLD);
  if (UseHighLevelShaders)
    services->setVertexShaderConstant("mWorldViewProj", worldViewProj.pointer(), 16);
  else
    services->setVertexShaderConstant(worldViewProj.pointer(), 4, 4);
  // 设置摄象机位置
  core::vector3df pos = device->getSceneManager()->
    getActiveCamera()->getAbsolutePosition();
  if (UseHighLevelShaders)
    services->setVertexShaderConstant("mLightPos", reinterpret_cast<f32*>(&pos), 3);
  else
    services->setVertexShaderConstant(reinterpret_cast<f32*>(&pos), 8, 1);
  // 设置灯光颜色
  video::SColorf col(0.0f,1.0f,1.0f,0.0f);
  if (UseHighLevelShaders)
    services->setVertexShaderConstant("mLightColor", reinterpret_cast<f32*>(&col), 4);
  else
    services->setVertexShaderConstant(reinterpret_cast<f32*>(&col), 9, 1);
  // 设置世界转换矩阵

  core::matrix4 world = driver->getTransform(video::ETS_WORLD);
  world = world.getTransposed();
  if (UseHighLevelShaders)
    services->setVertexShaderConstant("mTransWorld", world.pointer(), 16);
  else
    services->setVertexShaderConstant(world.pointer(), 10, 4);
}
};
/*
下面几行代码就象之前的例子一样，开始运行引擎，我们加一个命名行控制，用来
询问工具是否愿意开启高级着色（当然，用户选择的软件必须支持高级着色才给予选项）。
*/
int main()
{
// 让用户选择设备类型
video::E_DRIVER_TYPE driverType = video::EDT_DIRECT3D9;
printf("Please select the driver you want for this example:\n"\
```

```
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
    " (f) NullDevice\n (otherKey) exit\n\n");
char i;
std::cin >> i;
switch(i)
{
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
    case 'c': driverType = video::EDT_OPENGL;    break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO;break;
    case 'f': driverType = video::EDT_NULL;      break;
    default: return 1;
}
// 如果用户选择的设备类型支持高级着色，则允许用户进行选择是否开启高级着色
if (driverType == video::EDT_DIRECT3D9 ||
    driverType == video::EDT_OPENGL)
{
    printf("Please press 'y' if you want to use high level shaders.\n");
    std::cin >> i;
    if (i == 'y')
      UseHighLevelShaders = true;
}
// 创建设备
device = createDevice(driverType, core::dimension2d<s32>(640, 480));
if (device == 0)
    return 1;
video::IVideoDriver* driver = device->getVideoDriver();
scene::ISceneManager* smgr = device->getSceneManager();
gui::IGUIEnvironment* gui = device->getGUIEnvironment();
/*
现在进入更加有趣的部分。
如果我们使用 D3D，我们需要使用顶点 Shader 或象素 Shader 程序，如果我们使用 OpenGL，则
我们需要使用 ARB fragment 和顶点程序。（注：由于译者不熟悉 OpenGL 高层应用，ARB fragment 不做翻译）
我为不同的设备写了不同的文件，分别是 d3d8.ps,d3d8.vs,d3d9.psd3d9.vs,opengl.ps 和 opengl.vs。
注意：没必要象例子中这样把 Shaders 写到文件中。你同样可以把这些 Shaders 写到 cpp 文件中，不过，
如果这样做的话，你将需要将下面的 addShaderMaterialFromFiles() 函数改为 addShaderMaterial()
*/
```

```
c8* vsFileName = 0; // VertexShader 文件名
c8* psFileName = 0; // PixelShader 文件名
switch(driverType)
{
case video::EDT_DIRECT3D8:
    psFileName = "../../media/d3d8.psh";
    vsFileName = "../../media/d3d8.vsh";
    break;
case video::EDT_DIRECT3D9:
    if (UseHighLevelShaders)
    {
      psFileName = "../../media/d3d9.hlsl";
      vsFileName = psFileName; // VertexShader， PixelShader 都在同一文件中
    }
    else
    {
      psFileName = "../../media/d3d9.psh";
      vsFileName = "../../media/d3d9.vsh";
    }
    break;
case video::EDT_OPENGL:
    if (UseHighLevelShaders)
    {
      psFileName = "../../media/opengl.frag";
      vsFileName = "../../media/opengl.vert";
    }
    else
    {
      psFileName = "../../media/opengl.psh";
      vsFileName = "../../media/opengl.vsh";
    }
    break;
}
/*
另外，我们检测硬件支持我们需要的 Shaders。如果不支持，我们就设置该类 Shader 文件为空，
就不再进行该类 Shader 显示了,但不影响另一种 Shader 的显示。所以在本例中你可能看不到 PixelShader,但一般可以看到 VertexShader.
*/
if (!driver->queryFeature(video::EVDF_PIXEL_SHADER_1_1) &&
    !driver->queryFeature(video::EVDF_ARB_FRAGMENT_PROGRAM_1))
{
    device->getLogger()->log("WARNING: Pixel shaders disabled "\
```

```
        "because of missing driver/hardware support.");
    psFileName = 0;
}

if (!driver->queryFeature(video::EVDF_VERTEX_SHADER_1_1) &&
    !driver->queryFeature(video::EVDF_ARB_VERTEX_PROGRAM_1))
{
    device->getLogger()->log("WARNING: Vertex shaders disabled "\
        "because of missing driver/hardware support.");
    vsFileName = 0;
}
/*
OK,现在我们开始创建一个新的材质。
你可以从之前的例子中可以获知，若要在 Irr 引擎中改变一个材质类型，我们可以通过修改
材质结构中的材质类型值来达到相应的效果。而这个值仅是一个简单的 32 位数值，例如
Video::EMT_SOLID.
所以我们仅需要引擎来创建一些新的数值，然后我们将它拿去做参数进行修改就可以了。
现在我们获得 IGPUProgrammingServices 的指针，通过这个指针我们调用
addShaderMaterialFromFiles()，
而这个函数会为我们返回一个新的 32 位数值。这样就 OK 了，我现在说一下这个函数的参
数：
首先，包含顶点 Shader 象素 Shader 代码的文件，如果你不从文件中读取，使用的是
addShaderMaterial()
函数，你就不需要填写文件名，你仅需要填写 Shader 函数代码的函数名即可。
第三个参数是一个 IShaderConstantSetCallBack 类的指针，这个类在本例前面实现过。如果你
不想设置
这个参数，设为 0 就可以了。
最后一个参数告诉引擎哪个材质适合作为基层材质。为了证明这点，我们创建了两个不同基
层材质的材质层，
一个使用 EMT_SOLID（不透明的）另一个使用 EMT_TRANSPARENT_ADD_COLOR（透明+颜色）。
*/
// 创建材质
video::IGPUProgrammingServices* gpu = driver->getGPUProgrammingServices();
s32 newMaterialType1 = 0;
s32 newMaterialType2 = 0;
if (gpu)
{
    MyShaderCallBack* mc = new MyShaderCallBack();
    // 如果用户想使用高层或低层 Shader,则创建不同的 Shaders。
    if (UseHighLevelShaders)
    {
     //创建高级着色材质（HLSL 或者 GLSL）
     newMaterialType1 = gpu->addHighLevelShaderMaterialFromFiles(
      vsFileName, "vertexMain", video::EVST_VS_1_1,
```

```
        psFileName, "pixelMain", video::EPST_PS_1_1,
        mc, video::EMT_SOLID);
      newMaterialType2 = gpu->addHighLevelShaderMaterialFromFiles(
        vsFileName, "vertexMain", video::EVST_VS_1_1,
        psFileName, "pixelMain", video::EPST_PS_1_1,
        mc, video::EMT_TRANSPARENT_ADD_COLOR);
    }
    else
    {
      // 创建低级着色材质（asm 或 arb_asm）
      newMaterialType1 = gpu->addShaderMaterialFromFiles(vsFileName,
        psFileName, mc, video::EMT_SOLID);
      newMaterialType2 = gpu->addShaderMaterialFromFiles(vsFileName,
        psFileName, mc, video::EMT_TRANSPARENT_ADD_COLOR);
    }
    mc->drop();
}
/*
现在我们测试这些材质。我们创建一个测试用的立方体，并将我们创建的材质设置上去。
另外，我们为立方体增加一个文字场景节点，一个旋转器，使它看起来更加生动有趣。
*/
// 创建测试场景节点 1，我们将创建的材质 1 给予这个场景节点的立方体。
scene::ISceneNode* node = smgr->addCubeSceneNode(50);
node->setPosition(core::vector3df(0,0,0));
node->setMaterialTexture(0, driver->getTexture("../../media/wall.bmp"));
node->setMaterialFlag(video::EMF_LIGHTING, false);
node->setMaterialType((video::E_MATERIAL_TYPE)newMaterialType1);
smgr->addTextSceneNode(gui->getBuiltInFont(),
      L"PS & VS & EMT_SOLID",
      video::SColor(255,255,255,255), node);
scene::ISceneNodeAnimator* anim = smgr->createRotationAnimator(
      core::vector3df(0,0.3f,0));
node->addAnimator(anim);
anim->drop();
/*
创建第二个场景节点，依旧是个立方体，但是使用我们创建的第二种类型的材质。
*/
node = smgr->addCubeSceneNode(50);
node->setPosition(core::vector3df(0,-10,50));
node->setMaterialTexture(0, driver->getTexture("../../media/wall.bmp"));
node->setMaterialFlag(video::EMF_LIGHTING, false);
node->setMaterialType((video::E_MATERIAL_TYPE)newMaterialType2);
smgr->addTextSceneNode(gui->getBuiltInFont(),
      L"PS & VS & EMT_TRANSPARENT",
```

```
    video::SColor(255,255,255,255), node);
anim = smgr->createRotationAnimator(core::vector3df(0,0.3f,0));
node->addAnimator(anim);
anim->drop();
/*
现在我们创建第三个场景接点，但是我们并不对其设置 Shader，目的是与前两个 Shader 立
方体进行比较。
*/

node = smgr->addCubeSceneNode(50);
node->setPosition(core::vector3df(0,50,25));
node->setMaterialTexture(0, driver->getTexture("../../media/wall.bmp"));
node->setMaterialFlag(video::EMF_LIGHTING, false);
smgr->addTextSceneNode(gui->getBuiltInFont(), L"NO SHADER",
    video::SColor(255,255,255,255), node);
/*
最后，我们为场景添加一个天空盒和一个用户控制的摄象机。
因为我们的天空盒不需要使用 Mipmap，所以，我们关闭了 Mipmap.
*/
// 增加一个好看的天空盒
driver->setTextureCreationFlag(video::ETCF_CREATE_MIP_MAPS, false);
smgr->addSkyBoxSceneNode(
    driver->getTexture("../../media/irrlicht2_up.jpg"),
    driver->getTexture("../../media/irrlicht2_dn.jpg"),
    driver->getTexture("../../media/irrlicht2_lf.jpg"),
    driver->getTexture("../../media/irrlicht2_rt.jpg"),
    driver->getTexture("../../media/irrlicht2_ft.jpg"),
    driver->getTexture("../../media/irrlicht2_bk.jpg"));
driver->setTextureCreationFlag(video::ETCF_CREATE_MIP_MAPS, true);
// 增加一个摄象机场景接点，另外屏蔽鼠标图标
scene::ICameraSceneNode* cam = smgr->addCameraSceneNodeFPS(0, 100.0f, 100.0f);
cam->setPosition(core::vector3df(-100,50,100));
cam->setTarget(core::vector3df(0,0,0));
device->getCursorControl()->setVisible(false);
/*
现在进行绘制就 OK 了
*/
int lastFPS = -1;
while(device->run())
    if (device->isWindowActive())
{
    driver->beginScene(true, true, video::SColor(255,0,0,0));
    smgr->drawAll();
    driver->endScene();
```

```
    int fps = driver->getFPS();
    if (lastFPS != fps)
    {
      core::stringw str = L"Irrlicht Engine - Vertex and pixel shader example [";
      str += driver->getName();
      str += "] FPS:";
      str += fps;
      device->setWindowCaption(str.c_str());
      lastFPS = fps;
    }
}
device->drop();

return 0;
}
```

irrlicht 之 11.PerPixelLighting

```
/*
这个例子告诉我们如何使用 Irr 引擎中更加复杂的技术：逐象素光照处理的 法线映射和视差映射。
另外我们还需要使用 雾 以及 运动的粒子系统。别害怕！在 Irr 引擎中，你做这些工作，不需要太多的 Shaders 经验。
注意：因为我们使用视差映射，我们需要使用 32 位纹理。
首先，我们象以往一样，包含 Irr 头文件，定义库链接，定义命名空间。
*/
#include <irrlicht.h>
#include <iostream>

using namespace irr;
#pragma comment(lib, "Irrlicht.lib")

/*
这个例子中，我们需要一个事件接收器，因为我们准备为用户提供三个不同类型的材质风格，这需要用户交互，获得他们的输入。另外，事件接受器还将创建一些小的 GUI，用这些 GUI 来
显示用户现在选择的哪种材质。在这个类中没有什么特殊的东西，所以你在详细研究的时候可以跳过它。
*/
class MyEventReceiver : public IEventReceiver
{
public:
MyEventReceiver(scene::ISceneNode* room,
    gui::IGUIEnvironment* env, video::IVideoDriver* driver)
{
    // 存储一些指针以方便我们更改它的绘制模式
    Room = room;
    Driver = driver;
    // 设置一个漂亮的字体
    gui::IGUISkin* skin = env->getSkin();
    gui::IGUIFont* font = env->getFont("../../media/fonthaettenschweiler.bmp");
    if (font)
      skin->setFont(font);
    // 增加一个窗口和一个 ListBox 列表单
    gui::IGUIWindow* window = env->addWindow(
      core::rect<s32>(460,375,630,470), false, L"Use 'E' + 'R' to change");
    ListBox = env->addListBox(
      core::rect<s32>(2,22,165,88), window);
    ListBox->addItem(L"Diffuse");
    ListBox->addItem(L"Bump mapping");
    ListBox->addItem(L"Parallax mapping");
```

```
    ListBox->setSelected(1);
    // 当硬件不支持 Shaders 时创建错误提示信息
    ProblemText = env->addStaticText(
     L"Your hardware or this renderer is not able to use the "\
     L"needed shaders for this material. Using fall back materials.",
     core::rect<s32>(150,20,470,80));
    ProblemText->setOverrideColor(video::SColor(100,255,255,255));
    // 设置初始材质（若硬件支持的话，我们优先显示视差映射）
    video::IMaterialRenderer* renderer =
     Driver->getMaterialRenderer(video::EMT_PARALLAX_MAP_SOLID);
    if (renderer && renderer->getRenderCapability() == 0)
     ListBox->setSelected(2);
    // 根据用户在列表单中的选择设置不同的材质
    setMaterial();
}
bool OnEvent(SEvent event)
{
    // 如果用户按下 E 或 R
    if (event.EventType == irr::EET_KEY_INPUT_EVENT &&
     !event.KeyInput.PressedDown && Room && ListBox)
    {
     // 更改列表单中的子控件选项
     int sel = ListBox->getSelected();
     if (event.KeyInput.Key == irr::KEY_KEY_R)
      ++sel;
     else
     if (event.KeyInput.Key == irr::KEY_KEY_E)
      --sel;
     else
      return false;
     if (sel > 2) sel = 0;
     if (sel < 0) sel = 2;
     ListBox->setSelected(sel);

     // 根据用户在列表单中的选择设置不同的材质
     setMaterial();
    }
    return false;
}
private:
// 根据 ListBox 列表单设置房子 Mesh 的材质
void setMaterial()
{
    video::E_MATERIAL_TYPE type = video::EMT_SOLID;
```

```
    // 更改材质设置
    switch(ListBox->getSelected())
    {
    case 0: type = video::EMT_SOLID;
      break;
    case 1: type = video::EMT_NORMAL_MAP_SOLID;
      break;
    case 2: type = video::EMT_PARALLAX_MAP_SOLID;
      break;
    }
    Room->setMaterialType(type);
    /*
    我们先简单检测一下硬件是否能够完善的进行这种材质渲染方式。
    如果材质不能够被 100%正确显示，我们将汇报一个错误，然后使用差一级别的材质，
    最少也要让用户知道使用这种法线映射或视差映射能使画面更加漂亮。
    */
    video::IMaterialRenderer* renderer = Driver->getMaterialRenderer(type);
    // 当硬件不支持，则进行错误汇报。
    if (!renderer || renderer->getRenderCapability() != 0)
      ProblemText->setVisible(true);
    else
      ProblemText->setVisible(false);
}
private:
gui::IGUIStaticText* ProblemText;
gui::IGUIListBox* ListBox;
scene::ISceneNode* Room;
video::IVideoDriver* Driver;
};

/*
现在我们创建一个 Irr 设备并且建立好场景
*/
int main()
{
// 让用户选择设备类型
video::E_DRIVER_TYPE driverType = video::EDT_DIRECT3D9;
printf("Please select the driver you want for this example:\n"\
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
    " (f) NullDevice\n (otherKey) exit\n\n");
char i;
std::cin >> i;
switch(i)
```

```
{
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
    case 'c': driverType = video::EDT_OPENGL;    break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO;break;
    case 'f': driverType = video::EDT_NULL;    break;
    default: return 0;
}
// 创建设备
IrrlichtDevice* device = createDevice(driverType, core::dimension2d<s32>(640, 480));
if (device == 0)
    return 1;

/*
在我们开始进行有趣的材质渲染设置之前，我们做一些简单的事情：
1：存贮一些引擎重要部分的指针，（图形驱动器，场景管理器，GUI 环境）。
2：在窗口增加一个 Irr 引擎 Logo
3：增加一个用户控制 FPS 风格的摄象机
*/
video::IVideoDriver* driver = device->getVideoDriver();
scene::ISceneManager* smgr = device->getSceneManager();
gui::IGUIEnvironment* env = device->getGUIEnvironment();
driver->setTextureCreationFlag(video::ETCF_ALWAYS_32_BIT, true);
// 增加一个 Irr 的 Logo
env->addImage(driver->getTexture("../../media/irrlichtlogo2.png"),
    core::position2d<s32>(10,10));

// 增加摄象机
scene::ICameraSceneNode* camera =
    smgr->addCameraSceneNodeFPS(0,100.0f,300.0f);
camera->setPosition(core::vector3df(-200,200,-200));
// 屏蔽鼠标图标显示
device->getCursorControl()->setVisible(false);

/*
因为我们希望整个场景看起来更加恐怖一些，所以我们增加一些雾。
增加雾的方法是 IVideoDriver::setFog()。通过这个函数你可以设置不同类型的雾。
在本例中，我们使用象素雾，因为它更适合本例中的材质风格。
请注意，你必须在每个你需要雾特效影响的场景节点中开启雾特效，默认的它是为关闭的。
*/
driver->setFog(video::SColor(0,138,125,81), true, 250, 1000, 0, true);
/*
我们读取一个 a.3ds 文件中的 Mesh，那是我使用 Anim8or 软件建立的一个房子模型。
```

和 SpecialFX 那个例子中的模型是一样的，哈哈，你可能会想起那个例子吧，
我不是一个优秀的模型设计师，所以我简单的把这个模型加了个糟糕的纹理，
但是我可以使用 IMeshManipulator::makePlanarTextureMapping()这个函数把
纹理变的漂亮起来。
*/
```
scene::IAnimatedMesh* roomMesh = smgr->getMesh(
    "../../media/room.3ds");
scene::ISceneNode* room = 0;
if (roomMesh)
{
    smgr->getMeshManipulator()->makePlanarTextureMapping(
      roomMesh->getMesh(0), 0.003f);
    /*
    现在开始我们的第一个有意思的事情：如果我顺利的读取模型 Mesh，我们现在就为它
    增加一个纹理。因为我想房子看起来更加真实，所以我们需要为纹理材质做一点事情。
    不同于往常简单的读取一个彩色纹理那么简单，我们读取一个灰白高度图。

    这个高度图使用图象设备的 makeNormalMapTexture()函数去创建成法线图。第二个参数
    是说明高度图中的图素高度差，如果你把这个值设置大一些的话，你看墙壁会更有凹凸
感。
    除了法线图外，再为房子贴上一层普通的纹理图就可以了。
    */
    video::ITexture* colorMap = driver->getTexture("../../media/rockwall.bmp");
    video::ITexture* normalMap = driver->getTexture("../../media/rockwall_height.bmp");

    driver->makeNormalMapTexture(normalMap, 9.0f);
    /*
    但是仅仅设置了法线纹理和颜色纹理还是不够的。我们还需要对材质信息进行一些设置，
    使每个顶点拥有切线信息。因为我们懒得去计算这些信息，所以我们让 Irr 帮我们来做这
些工作。
    这就是我为什么调用 IMeshManipulator::createMeshWithTangents()这个函数的原因。
    这个函数创建了一个 Mesh 的 Copy，并为其计算了切线信息。
    在做完这个之后，我们使用这个 MeshCopy 做了一个简单场景节点，设置它的颜色，纹
理以及一些材质属性。
    注意，我们这里设置了 EMF_FOG_ENABLE 为 true,这样我们为房间开启了雾化效果。
    */
    scene::IMesh* tangentMesh = smgr->getMeshManipulator()->createMeshWithTangents(
      roomMesh->getMesh(0));
    room = smgr->addMeshSceneNode(tangentMesh);
    room->setMaterialTexture(0, colorMap);
    room->setMaterialTexture(1, normalMap);
    room->getMaterial(0).SpecularColor.set(0,0,0,0);
    room->setMaterialFlag(video::EMF_FOG_ENABLE, true);
    room->setMaterialType(video::EMT_PARALLAX_MAP_SOLID);
```

```
    // 调整视差特效的高度
    room->getMaterial(0).MaterialTypeParam = 0.035f;
    // 因为这个 Mesh 是我们创建的 Copy 版，之后我们也用不到，所以我们要删除它。
    tangentMesh->drop();
}
/*
在我们为房间增加了一个逐象素光照（法线映射和视差映射）效果之后，我们在其中
设置了一个圆球体，并为其设置了相同的材质，但是我们设置它为半透明的。
另外，为了使圆球看起来更象我们的地球，我们让旋转起来。
这些程序和我们之前的程序差不多，不同的是我们读取的 mesh 文件中已经包含了颜色纹理，
所以我们没必要再设置它的颜色纹理了。但是因为这个球体有点小了，所以我们把它放大一
些。
*/
// 增加一个地球球体。
scene::IAnimatedMesh* earthMesh = smgr->getMesh("../../media/earth.x");
if (earthMesh)
{
    // 获取 Mesh 操控器，之后我们将使用它进行一些操作
    scene::IMeshManipulator *manipulator = smgr->getMeshManipulator();
    // 为 Mesh 创建一个带切线信息的 Copy
    scene::IMesh* tangentSphereMesh =
      manipulator->createMeshWithTangents(earthMesh->getMesh(0));
    // 设置 Mesh 所有顶点的 Alpha 透明度为 80
    manipulator->setVertexColorAlpha(tangentSphereMesh, 80);

    // 放大模型
    core::matrix4 m;
    m.setScale ( core::vector3df(50,50,50) );
    manipulator->transformMesh( tangentSphereMesh, m );
    scene::ISceneNode *sphere = smgr->addMeshSceneNode(tangentSphereMesh);
    sphere->setPosition(core::vector3df(-70,130,45));
    // 读取高度图，并用它生成法线图
    video::ITexture* earthNormalMap = driver->getTexture("../../media/earthbump.bmp");
    driver->makeNormalMapTexture(earthNormalMap, 20.0f);
    sphere->setMaterialTexture(1, earthNormalMap);
    // 调整材质的属性
    sphere->setMaterialFlag(video::EMF_FOG_ENABLE, true);
    sphere->setMaterialType(video::EMT_NORMAL_MAP_TRANSPARENT_VERTEX_ALPHA);
    // 增加一个旋转控制器
    scene::ISceneNodeAnimator* anim =
      smgr->createRotationAnimator(core::vector3df(0,0.1f,0));
    sphere->addAnimator(anim);
    anim->drop();
    // 删除 Mesh 的 Copy
```

```
    tangentSphereMesh->drop();
}
/*
逐象素光照材质只有在移动的灯光效果下才会看起来很真实很酷，所以我们添加一些移动的光源。
但是因为仅仅有移动光源的话，看起来会很让人困惑郁闷，所以我们为每个光源添加一个公告版，
来告诉用户我们的光源位置。另外，我还为其中一个光源设置了一个完整的粒子系统，让它看起来象有尾巴一样。
好，我们开始设置第一个光照，我们设置它为红色，而且不加粒子特效。
*/
// 添加第一个光照（接近红色）
scene::ILightSceneNode* light1 =
    smgr->addLightSceneNode(0, core::vector3df(0,0,0),
    video::SColorf(0.5f, 1.0f, 0.5f, 0.0f), 200.0f);

// 为其增加一个飞行控制器
scene::ISceneNodeAnimator* anim =
    smgr->createFlyCircleAnimator (core::vector3df(50,300,0),190.0f, -0.003f);
light1->addAnimator(anim);
anim->drop();
// 再为其设置一个公告板
scene::ISceneNode* bill =
    smgr->addBillboardSceneNode(light1, core::dimension2d<f32>(60, 60));
bill->setMaterialFlag(video::EMF_LIGHTING, false);
bill->setMaterialType(video::EMT_TRANSPARENT_ADD_COLOR);
bill->setMaterialTexture(0, driver->getTexture("../../media/particlered.bmp"));
/*
一样的，我们设置另一个光照，不同的是，我们需要对它添加一个粒子系统。
因为光源是移动的，粒子系统需要对其进行跟随。如果想知道粒子系统如何使用，请看一下 SpecialFx 的例子。
或许你会发现，我们仅仅添加了两个灯光，我们这么简单处理的原因是：在低版本的 ps1.1 和 vs1.1
硬件环境下，不允许我们使用更多的灯光。当然，你可以为场景再增加一个灯光，但是它就不会参加
对墙壁阴影的计算。当然，在更高级的 Pixel/Vertex Shader 中这个问题将会改善解决。
*/
// 增加 2 号灯光（灰色）
scene::ISceneNode* light2 =
    smgr->addLightSceneNode(0, core::vector3df(0,0,0),
    video::SColorf(1.0f, 0.2f, 0.2f, 0.0f), 200.0f);
// 为 2 号灯光设置一个飞行控制器
anim = smgr->createFlyCircleAnimator (core::vector3df(0,150,0),200.0f, 0.001f, core::vector3df(0.2f, 0.9f, 0.f));
```

```
light2->addAnimator(anim);
anim->drop();
// 为灯光绑定一个公告版
bill = smgr->addBillboardSceneNode(light2, core::dimension2d<f32>(120, 120));
bill->setMaterialFlag(video::EMF_LIGHTING, false);
bill->setMaterialType(video::EMT_TRANSPARENT_ADD_COLOR);
bill->setMaterialTexture(0, driver->getTexture("../../media/particlewhite.bmp"));
// 增加一个粒子系统场景节点
scene::IParticleSystemSceneNode* ps =
    smgr->addParticleSystemSceneNode(false, light2);
ps->setParticleSize(core::dimension2d<f32>(30.0f, 40.0f));
// 设置粒子发射器
scene::IParticleEmitter* em = ps->createBoxEmitter(
    core::aabbox3d<f32>(-3,0,-3,3,1,3),
    core::vector3df(0.0f,0.03f,0.0f),
    80,100,
    video::SColor(0,255,255,255), video::SColor(0,255,255,255),
    400,1100);
ps->setEmitter(em);
em->drop();
// 为粒子系统节点设置一个淡出特效器
scene::IParticleAffector* paf = ps->createFadeOutParticleAffector();
ps->addAffector(paf);
paf->drop();
// 调整一些材质的设置
ps->setMaterialFlag(video::EMF_LIGHTING, false);
ps->setMaterialTexture(0, driver->getTexture("../../media/fireball.bmp"));
ps->setMaterialType(video::EMT_TRANSPARENT_VERTEX_ALPHA);

MyEventReceiver receiver(room, env, driver);
device->setEventReceiver(&receiver);

/*
最后，绘制一切，这样就完成了。
*/
int lastFPS = -1;
while(device->run())
if (device->isWindowActive())
{
    driver->beginScene(true, true, 0);
    smgr->drawAll();
    env->drawAll();
    driver->endScene();
    int fps = driver->getFPS();
```

```
    if (lastFPS != fps)
    {
      core::stringw str = L"Per pixel lighting example - Irrlicht Engine [";
      str += driver->getName();
      str += "] FPS:";
      str += fps;
      device->setWindowCaption(str.c_str());
      lastFPS = fps;
    }
}
device->drop();

return 0;
}
```

irrlicht 之 12.TerrainRendering

```
/*
这个例子将简单的告诉我们如何使用 Irr 引擎的地形渲染，以及如何使用地形渲染三角形选
择器进行
摄象机与地形之间碰撞检测。
请注意，Irr 引擎中的地形渲染是基于 Spintz 先生的 GeoMipMapSceneNode，非常感谢他。
我们将使用小型的高度图来生成大型的地图场景，所以我们不得不使用---LOD 技术。
这个例子的开头依旧没什么特殊的，我们包含 Irr 头文件，设置好命名空间，库链接后，再
设置一个
事件接收器来监听用户输入，如果用户按下“W”键，则地形切换成线型网状，若用户按下
“D”键，
则在标准绘制和细节绘制之间进行切换。
（译者注：EMT_SOLID，最基本的标准绘制，即简单的将纹理贴在模型上。EMT_DETAIL_MAP，
是双层纹理
的渲染模式，它将第一层纹理简单贴在模型上之后，将第二层纹理进行放大缩小后混合于第
一层之上，
以达到更好的渲染效果，这种模式通常在大型地形渲染时使用）
*/
#include <irrlicht.h>
#include <iostream>
using namespace irr;
#pragma comment(lib, "Irrlicht.lib")

class MyEventReceiver : public IEventReceiver
{
public:
MyEventReceiver(scene::ISceneNode* terrain)
```

```
{
    // 为了方便改变地形渲染模式，我们保存一下地形场景节点的指针
    Terrain = terrain;
}
bool OnEvent(SEvent event)
{
    // 如果用户按下 W 或者 D 键
    if (event.EventType == irr::EET_KEY_INPUT_EVENT && !event.KeyInput.PressedDown)
    {
      switch (event.KeyInput.Key)
      {
      case irr::KEY_KEY_W:
        // 切换成线状地形渲染
        Terrain->setMaterialFlag(video::EMF_WIREFRAME, !Terrain->getMaterial(0).Wireframe);
        Terrain->setMaterialFlag(video::EMF_POINTCLOUD, false);
        return true;
      case irr::KEY_KEY_P:
        // 切换为点状地形渲染
        Terrain->setMaterialFlag(video::EMF_POINTCLOUD, !Terrain->getMaterial(0).PointCloud);
        Terrain->setMaterialFlag(video::EMF_WIREFRAME, false);
        return true;
      case irr::KEY_KEY_D:
        // 普通标准绘制和细节绘制中进行切换
        Terrain->setMaterialType(
          Terrain->getMaterial(0).MaterialType == video::EMT_SOLID ?
          video::EMT_DETAIL_MAP : video::EMT_SOLID);
        return true;
      }
    }
    return false;
}
private:
scene::ISceneNode* Terrain;
};

/*
类似于其他例子一样的主函数。
*/
int main()
{
// 让用户选择设备类型
video::E_DRIVER_TYPE driverType = video::EDT_DIRECT3D9;
printf("Please select the driver you want for this example:\n"\
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
```

```
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
    " (f) NullDevice\n (otherKey) exit\n\n");
char i;
std::cin >> i;
switch(i)
{
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
    case 'c': driverType = video::EDT_OPENGL;    break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO;break;
    case 'f': driverType = video::EDT_NULL;      break;
    default: return 1;
}
// 创建设备
IrrlichtDevice* device = createDevice(driverType, core::dimension2d<s32>(640, 480));
if (device == 0)
    return 1;

/*
首先，我们为场景增加一些标准元素：一个 Irr 引擎的 Logo,一个小的帮助界面，
一个用户控制摄象机，并且屏蔽鼠标图标。
*/
video::IVideoDriver* driver = device->getVideoDriver();
scene::ISceneManager* smgr = device->getSceneManager();
gui::IGUIEnvironment* env = device->getGUIEnvironment();
driver->setTextureCreationFlag(video::ETCF_ALWAYS_32_BIT, true);
// 增加一个 Logo
env->addImage(driver->getTexture("../../media/irrlichtlogo2.png"),
    core::position2d<s32>(10,10));
// 更换一个字体
env->getSkin()->setFont(env->getFont("../../media/fontlucida.png"));
// 增加一个帮助界面
gui::IGUIStaticText* text = env->addStaticText(
    L"Press 'W' to change wireframe mode\nPress 'D' to toggle detail map",
    core::rect<s32>(10,440,250,475), true, true, 0, -1, true);
// 增加一个摄象机
scene::ICameraSceneNode* camera =
    smgr->addCameraSceneNodeFPS(0,100.0f,1200.f);
camera->setPosition(core::vector3df(1900*2,255*2,3700*2));
camera->setTarget(core::vector3df(2397*2,343*2,2700*2));
camera->setFarValue(12000.0f);
// 屏蔽鼠标图标
device->getCursorControl()->setVisible(false);
```

```
/*
现在开始，我们开始渲染地形场景节点：我们和创建别的场景节点类似，
使用 ISceneManager::addTerrainSceneNode()来创建它。第一个参数是我们使用的高度图的
文件名称。高度图实际上就是一个简单的灰度纹理图，地形渲染器通过它创建 3D 地形。
为了使用地形看起来更大一些，我们创建时对其进行放大。
又因为我们场景中暂时不需要灯光，所以我们关闭了灯光。
之后我们设置 terrain-texture.jpg 作为地图的第一层纹理，而 detailmap3.jpg 做为第二层纹理：
最后，我们保持第一层纹理大小不变，第二层纹理放大了 20 倍才进行渲染。
*/
// 增加一个场景节点
scene::ITerrainSceneNode* terrain = smgr->addTerrainSceneNode(
    "../../media/terrain-heightmap.bmp",
    0,                  // 父节点
    -1,                 // 节点 ID 编号
    core::vector3df(0.f, 0.f, 0.f),     // 节点坐标
    core::vector3df(0.f, 0.f, 0.f),     // 旋转矩阵
    core::vector3df(40.f, 4.4f, 40.f),  // 缩放矩阵
    video::SColor ( 255, 255, 255, 255 ), // 顶点颜色
    5,                  // LOD 最大程度
    scene::ETPS_17,             // 地形块大小
    4                   // 平滑度
    );
// 译者注：地形块大小是与细节地形渲染紧密相关的，详情请参看引擎源码中的
ETerrainElements.h 文件。
terrain->setMaterialFlag(video::EMF_LIGHTING, false);
terrain->setMaterialTexture(0, driver->getTexture("../../media/terrain-texture.jpg"));
terrain->setMaterialTexture(1, driver->getTexture("../../media/detailmap3.jpg"));

terrain->setMaterialType(video::EMT_DETAIL_MAP);
terrain->scaleTexture(1.0f, 20.0f);
/*
为了能与地形进行碰撞检测，我们需要创建一个三角碰撞检测器。
如果你想知道碰撞检测器如何运作，请看碰撞检测的例子。
我们创建了一个摄象机，并为它创建了一个碰撞器，这样，我们的摄象机与地形之间
就会有碰撞处理，摄象机于是就不会移动出地表了。
*/
// 为地形创建一个三角型碰撞检测
scene::ITriangleSelector* selector
    = smgr->createTerrainTriangleSelector(terrain, 0);
terrain->setTriangleSelector(selector);
selector->drop();
// 为摄象机捆绑一个碰撞检测器
scene::ISceneNodeAnimator* anim = smgr->createCollisionResponseAnimator(
    selector, camera, core::vector3df(60,100,60),
```

```
    core::vector3df(0,0,0),
    core::vector3df(0,50,0));
camera->addAnimator(anim);
anim->drop();
/*
为了使用户能够在不同的模式之间进行切换，我们创建了一个事件接收器的实例，
另外我们如其他例子一样创建了一个天空盒。
*/
// 创建一个事件接收器
MyEventReceiver receiver(terrain);
device->setEventReceiver(&receiver);
    // 创建天空盒
driver->setTextureCreationFlag(video::ETCF_CREATE_MIP_MAPS, false);
smgr->addSkyBoxSceneNode(
    driver->getTexture("../../media/irrlicht2_up.jpg"),
    driver->getTexture("../../media/irrlicht2_dn.jpg"),
    driver->getTexture("../../media/irrlicht2_lf.jpg"),
    driver->getTexture("../../media/irrlicht2_rt.jpg"),
    driver->getTexture("../../media/irrlicht2_ft.jpg"),
    driver->getTexture("../../media/irrlicht2_bk.jpg"));
driver->setTextureCreationFlag(video::ETCF_CREATE_MIP_MAPS, true);

/*
OK 了，接下来绘制一切，你应该明白了如何在 Irr 引擎中绘制地形了吧。
*/
int lastFPS = -1;
while(device->run())
if (device->isWindowActive())
{
    driver->beginScene(true, true, 0 );
    smgr->drawAll();
    env->drawAll();
    driver->endScene();
    // 在窗口标题中显示当前桢数
    int fps = driver->getFPS();
    if (lastFPS != fps)
    {
      core::stringw str = L"Terrain Renderer - Irrlicht Engine [";
      str += driver->getName();
      str += "] FPS:";
      str += fps;
      // 同时，显示当前摄象机所在位置的地形高度
      str += " Height: ";
      str += terrain->getHeight(camera->getAbsolutePosition().X,
```

```
camera->getAbsolutePosition().Z);
        device->setWindowCaption(str.c_str());
        lastFPS = fps;
      }
}
device->drop();

return 0;
}
```

irrlicht 之 13.RenderToTexture

```
/*
这个例子告诉我们如何去使用 Irr 引擎实时渲染一个纹理。实时渲染纹理是一个制作优美特效的手段。
另外，这个例子告诉我们如何渲染镜面。
开始依旧和之前一样，包含头文件，让用户定义渲染设备，创建 Irr 设备。
*/
#include <irrlicht.h>
#include <iostream>
using namespace irr;
#pragma comment(lib, "Irrlicht.lib")
int main()
{
// 让用户选择设备类型
video::E_DRIVER_TYPE driverType = video::EDT_DIRECT3D9;
printf("Please select the driver you want for this example:\n"\
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
    " (f) NullDevice\n (otherKey) exit\n\n");
char i;
std::cin >> i;
switch(i)
{
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
    case 'c': driverType = video::EDT_OPENGL;    break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO;break;
    case 'f': driverType = video::EDT_NULL;      break;
    default: return 1;
}
```

```
// 创建设备，若失败则直接返回
IrrlichtDevice *device =
    createDevice(driverType, core::dimension2d<s32>(640, 480),
    16, false, false);
if (device == 0)
    return 1;
video::IVideoDriver* driver = device->getVideoDriver();
scene::ISceneManager* smgr = device->getSceneManager();
gui::IGUIEnvironment* env = device->getGUIEnvironment();

/*
现在我们读取一个将被显示的 Mesh.这个 Mesh 我们依旧使用常见的.md2 格式模型。
不同之处在于：我们设置模型自发光为 20，而一般默认值为 0。
这样的话，如果动态光照开启就可以进行镜面渲染。而这个自发光值也将会影响镜面渲染。
*/
// 读取并显示一个 Mesh
scene::IAnimatedMeshSceneNode* fairy = smgr->addAnimatedMeshSceneNode(
    smgr->getMesh("../../media/faerie.md2"));
if (fairy)
{
    // 设置纹理
    fairy->setMaterialTexture(0, driver->getTexture("../../media/faerie2.bmp"));
    // 开启动态光照
    fairy->setMaterialFlag(video::EMF_LIGHTING, true);
    // 设置自发光强度大小（即镜面渲染中的亮度大小）
    fairy->getMaterial(0).Shininess = 20.0f;
    fairy->setPosition(core::vector3df(-10,0,-100));
    fairy->setMD2Animation ( scene::EMAT_STAND );
}

/*
为了在模型上实现镜面映射渲染，我们在场景中设置一个动态光照。我们将它设置在模型旁
边。
另外，为了使模型看起来不太黑，我们设置环境光为灰色。
*/
// 增加一个场景光照
scene::ILightSceneNode* light = smgr->addLightSceneNode(0,
    core::vector3df(-15,5,-105), video::SColorf(1.0f, 1.0f, 1.0f));
// 设置环境光
smgr->setAmbientLight(video::SColor(0,160,160,160));

/*
下面的将是一些标准步骤：添加一个用户控制摄象机，隐藏鼠标，增加一个测试立方体，使
它旋转起来等
```

```
*/
// 添加一个 FPS 摄象机场景节点
scene::ICameraSceneNode* fpsCamera = smgr->addCameraSceneNodeFPS();
fpsCamera->setPosition(core::vector3df(-50,50,-150));
// 隐藏鼠标图标显示
device->getCursorControl()->setVisible(false);
// 创建一个测试用的立方体
scene::ISceneNode* test = smgr->addCubeSceneNode(60);
// 使立方体旋转起来，并且设置一些光照属性
scene::ISceneNodeAnimator* anim = smgr->createRotationAnimator(
    core::vector3df(0.3f, 0.3f,0));
test->setPosition(core::vector3df(-100,0,-100));
// 关闭动态光照
test->setMaterialFlag(video::EMF_LIGHTING, false);
test->addAnimator(anim);
anim->drop();
// 设置窗口标题
device->setWindowCaption(L"Irrlicht Engine - Render to Texture and Specular Highlights example");

/*
为了获取渲染后的纹理，我们先需要一个纹理对象。它和普通的纹理不同，我们需要先使用IVideoDriver::createRenderTargetTexture()创建它并且指定纹理的大小。
请注意：这里不允许使用比桢缓存尺寸还大的纹理尺寸。
因为我们不打算以用户摄象机做为镜面摄象机，所以我们创建了一个新的摄象机，由它来获取渲染出的纹理。
（译者注：镜面渲染的原理是，由一个摄象机点拍摄当前渲染桢，把这个拍摄到的结果做为一个
纹理帖到一个模型上）
*/
// 创建渲染目标
video::ITexture* rt = 0;
scene::ICameraSceneNode* fixedCam = 0;
if (driver->queryFeature(video::EVDF_RENDER_TO_TARGET))
{

    rt = driver->createRenderTargetTexture(core::dimension2d<s32>(256,256));
    // 为渲染目标设置材质属性
    test->setMaterialTexture(0, rt);
    // 增加一个固定摄象机
    fixedCam = smgr->addCameraSceneNode(0, core::vector3df(10,10,-80),
      core::vector3df(-10,10,-100));
}
else
```

```
{
    // 报错信息
    gui::IGUISkin* skin = env->getSkin();
    gui::IGUIFont* font = env->getFont("../../media/fonthaettenschweiler.bmp");
    if (font)
     skin->setFont(font);
    gui::IGUIStaticText* text = env->addStaticText(
     L"Your hardware or this renderer is not able to use the "\
     L"render to texture feature. RTT Disabled.",
     core::rect<s32>(150,20,470,60));
    text->setOverrideColor(video::SColor(100,255,255,255));
}

/*
接近结束了，现在我们需要进行绘制了。每桢，我们绘制场景两次。
绘制第一次是为了方便镜面摄象机进行渲染截取，此时我们就可以获得渲染目标纹理了。但是请
注意，此时我们需要设置测试方块不可见，因为我们的动态镜面纹理中不应当出现渲染立方体本身。
恩，我希望这样讲不会太复杂。
*/
int lastFPS = -1;
while(device->run())
if (device->isWindowActive())
{
    driver->beginScene(true, true, 0);
    if (rt)
    {
     // 绘制渲染纹理中的场景

     // 设置渲染目标的纹理
     driver->setRenderTarget(rt, true, true, video::SColor(0,0,0,255));
     // 设置立方体不可见，并设置镜面混合摄象机为活动摄象机
     test->setVisible(false);
     smgr->setActiveCamera(fixedCam);
     // 绘制整个场景并将其绘制到渲染缓冲中去
     smgr->drawAll();
     // set back old render target
     driver->setRenderTarget(0);
     // 设置立方体为可见，并更换回 FPS 摄象机为活动摄象机
     test->setVisible(true);
     smgr->setActiveCamera(fpsCamera);
    }
```

```
    // 绘制第二次场景
    smgr->drawAll();
    env->drawAll();
    driver->endScene();
    // 窗口标题显示帧数
    int fps = driver->getFPS();
    if (lastFPS != fps)
    {
      core::stringw str = L"Irrlicht Engine - Render to Texture and Specular Highlights example";
      str += " FPS:";
      str += fps;
      device->setWindowCaption(str.c_str());
      lastFPS = fps;
    }
}
if (rt)
    rt->drop(); // 释放渲染对象
device->drop(); // 释放设备
return 0;
}
```

irrlicht 之 14.Win32Window

```
// 这个例子仅可在 Windows 环境下运行，它可以证明 Irr 可以在一个 Win32 窗口内运行。
#include <irrlicht.h>
#ifndef _IRR_WINDOWS_
#error Windows only example
#else
#include <windows.h> // 这个例子仅可运行于 Windows 环境下
using namespace irr;
#pragma comment(lib, "irrlicht.lib")
HWND hOKButton;
HWND hWnd;
static LRESULT CALLBACK CustomWndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
switch (message)
{
case WM_COMMAND:
    {
      HWND hwndCtl = (HWND)lParam;
      int code = HIWORD(wParam);
      if (hwndCtl == hOKButton)
      {
        DestroyWindow(hWnd);
        PostQuitMessage(0);
        return 0;
      }
    }
    break;
case WM_DESTROY:
    PostQuitMessage(0);
    return 0;
}
return DefWindowProc(hWnd, message, wParam, lParam);
}
int main()
//int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hpre, LPSTR cmd, int cc)
{
HINSTANCE hInstance = 0;
const char* Win32ClassName = "CIrrlichtWindowsTestDialog";
WNDCLASSEX wcex;
wcex.cbSize        = sizeof(WNDCLASSEX);
wcex.style         = CS_HREDRAW | CS_VREDRAW;
wcex.lpfnWndProc = (WNDPROC)CustomWndProc;
wcex.cbClsExtra      = 0;
```

```
wcex.cbWndExtra     = DLGWINDOWEXTRA;
wcex.hInstance      = hInstance;
wcex.hIcon          = NULL;
wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW);
wcex.lpszMenuName = 0;
wcex.lpszClassName = Win32ClassName;
wcex.hIconSm        = 0;
RegisterClassEx(&wcex);
DWORD style = WS_SYSMENU | WS_BORDER | WS_CAPTION |
    WS_CLIPCHILDREN | WS_CLIPSIBLINGS | WS_MAXIMIZEBOX | WS_MINIMIZEBOX |
WS_SIZEBOX;
int windowWidth = 440;
int windowHeight = 380;
hWnd = CreateWindow( Win32ClassName, "Irrlicht Win32 window example",
    style, 100, 100, windowWidth, windowHeight,
    NULL, NULL, hInstance, NULL);
RECT clientRect;
GetClientRect(hWnd, &clientRect);
windowWidth = clientRect.right;
windowHeight = clientRect.bottom;
// 创建 OK 按钮
hOKButton = CreateWindow("BUTTON", "OK - Close", WS_CHILD | WS_VISIBLE | BS_TEXT,
    windowWidth - 160, windowHeight - 40, 150, 30, hWnd, NULL, hInstance, NULL);
// 创建一些静态文字
CreateWindow("STATIC", "This is Irrlicht running inside a standard Win32 window.\n"\
    "Also mixing with MFC and .NET Windows.Forms is possible.",
    WS_CHILD | WS_VISIBLE, 20, 20, 400, 40, hWnd, NULL, hInstance, NULL);
// 创建一个 BUTTON 子窗口以提供 Irr 使用
HWND hIrrlichtWindow = CreateWindow("BUTTON", "", WS_CHILD | WS_VISIBLE |
BS_OWNERDRAW,
     50, 80, 320, 220, hWnd, NULL, hInstance, NULL);
// 在 BUTTON 子窗口中创建一个 Irr 设备
irr::SIrrlichtCreationParameters param;
param.WindowId = reinterpret_cast<void*>(hIrrlichtWindow);
param.DriverType = video::EDT_OPENGL;
irr::IrrlichtDevice* device = irr::createDeviceEx(param);
// 设置一个简单的 3D 场景
irr::scene::ISceneManager* smgr = device->getSceneManager();
video::IVideoDriver* driver = device->getVideoDriver();
scene::ICameraSceneNode* cam = smgr->addCameraSceneNode();
cam->setTarget(core::vector3df(0,0,0));
scene::ISceneNodeAnimator* anim = smgr->createFlyCircleAnimator(core::vector3df(0,15,0),
30.0f);
```

```
cam->addAnimator(anim);
anim->drop();
scene::ISceneNode* cube = smgr->addCubeSceneNode(20);
cube->setMaterialTexture(0, driver->getTexture("../../media/wall.bmp"));
cube->setMaterialTexture(1, driver->getTexture("../../media/water.jpg"));
cube->setMaterialFlag( video::EMF_LIGHTING, false );
cube->setMaterialType( video::EMT_REFLECTION_2_LAYER );
smgr->addSkyBoxSceneNode(
driver->getTexture("../../media/irrlicht2_up.jpg"),
driver->getTexture("../../media/irrlicht2_dn.jpg"),
driver->getTexture("../../media/irrlicht2_lf.jpg"),
driver->getTexture("../../media/irrlicht2_rt.jpg"),
driver->getTexture("../../media/irrlicht2_ft.jpg"),
driver->getTexture("../../media/irrlicht2_bk.jpg"));
// 显示 Windows 窗口
ShowWindow(hWnd , SW_SHOW);
UpdateWindow(hWnd);
/*
你可以简单的使用 GetMessage，DispatchMessage 等消息循环。然后在其中调用 Device->Run()
的话 Irr 引擎就可以获取 Windows 的消息分发了。
但是如果你不调用 Device->run()的话，你就无法获取 Windows 分发的用户输入信息。那么
你只能
自己使用 DirectInput 等东西来获取。
*/
while (device->run())
{
    driver->beginScene(true, true, 0);
    smgr->drawAll();
    driver->endScene();
}
// 如果你没有调用 Device->run()你就必须这样自己获取消息了，代码如下
/*
MSG msg;
while (true)
{
    if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
    {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
      if (msg.message == WM_QUIT)
        break;
    }
    // advance virtual time
```

```
    device->getTimer()->tick();
    // 绘制渲染引擎场景
    driver->beginScene(true, true, 0);
    smgr->drawAll();
    driver->endScene();
}*/
device->closeDevice();
device->drop();
return 0;
}
#endif // if windows
```

irrlicht 之 15.LoadIrrFile

```
/*
译者注：研究本例者请务必查看 example.irr 文件。
从 1.1 版本开始，Irr 引擎就支持使用.irr 文件来读取和保存一个完整的场景。
实际上，这个 irr 文件就是一个 xml 格式的配置文件，我们可以通过记事本打开它看看。
不过有一个更好的专门的编辑器，叫 IrrEdit,请从 http://www.ambiera.com/irredit
这里进行下载。这个编辑器甚至可以进行肩负起场景编辑器和粒子编辑器的功能。
不过我们不讲述编辑器，本例我们来演示一下如何使用.irr 文件。
首先，我们创建一个窗口和 Irr 设备。
*/
#include <irrlicht.h>
#include <iostream>
using namespace irr;
#pragma comment(lib, "Irrlicht.lib")
int main()
{
// 让用户创建设备
video::E_DRIVER_TYPE driverType;
printf("Please select the driver you want for this example:\n"\
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
```

```
    " (f) NullDevice\n (otherKey) exit\n\n");
char i;
std::cin >> i;
switch(i)
{
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
    case 'c': driverType = video::EDT_OPENGL;    break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO;break;
    case 'f': driverType = video::EDT_NULL;      break;
    default: return 1;
}
IrrlichtDevice* device =
    createDevice(driverType, core::dimension2d<s32>(640, 480));
if (device == 0)
    return 1;
device->setWindowCaption(L"Load .irr file example");
video::IVideoDriver* driver = device->getVideoDriver();
scene::ISceneManager* smgr = device->getSceneManager();
/*
现在我们读取一个.irr 文件。irr 文件能够存储整个场景的材质，纹理，粒子系统等信息。
同时也可以存储场景中的所有场景接点和用户数据。
我们为了使例子简单些，我简单的读取一下场景。想获取更多信息的话，可以看文档中
关于 ISceneManager::loadScene 和 ISceneManager::saveScene 的说明。
我们这里仅读取显示一个场景，所以仅需要使用 loadScene()函数。
*/
// 读取一个场景
smgr->loadScene("../../media/example.irr");
/*
这样就 OK 了，完成了！我们仅需要再增加一个摄象机，绘制场景就行了
*/
//增加一个 FPS 摄象机
smgr->addCameraSceneNodeFPS();
// 绘制一切
int lastFPS = -1;
while(device->run())
if (device->isWindowActive())
{
    driver->beginScene(true, true, video::SColor(0,200,200,200));
    smgr->drawAll();
    driver->endScene();
    int fps = driver->getFPS();
    if (lastFPS != fps)
```

```
    {
      core::stringw str = L"Load Irrlicht File example - Irrlicht Engine [";
      str += driver->getName();
      str += "] FPS:";
      str += fps;
      device->setWindowCaption(str.c_str());
      lastFPS = fps;
    }
}
device->drop();

return 0;
}
```

irrlicht 之 16.Quake3MapShader

```
/*
这个例子告诉我们如何使用引擎读取一个 Quake3 形式的压缩地图，如何优化场景节点的渲染，
以及如何创建一个用户控制摄象机。
（译者注：此段注释怀疑作者并未认真处理，仅是从第二个例子简单复制粘贴过来，所以，本文中真正的独到之处：截屏技术 以及 各场景 Shader 的实现 却未在此说明）
首先，象其他例子一样，我们包含 Irr 头文件，并要求用户创建了一个设备。
*/
#include <irrlicht.h>
#include <iostream>

/*
类似于写 HelloWorld 例子之前需要做的准备一样，在 Irrlicht 引擎中，一切
函数，类命名都是在 irr 命名空间内的。我们依旧要告诉编辑器我们现在使用
的函数应当在 irr 命名空间内寻找。它有五个子命名空间，Core,Scene,Video,
Io,Gui.与 HelloWorld 不同的是，我们这里没有为五个子空间分别指定命名空间
的通知，因为这样做的话，在下面的代码中，你将更容易获知每个函数到底是属于
哪个命名空间内的。当然，你也可以加上 using namespace XX;尽随你意了。
*/
using namespace irr;
```

```
using namespace scene;
/*
同样，为了可以使用 Irrlicht.DLL 文件，我们需要链接一个 Irrlicht.lib 文件，我们需要
进行项目设置，或者在代码中进行一次链接声明。
*/
#pragma comment(lib, "Irrlicht.lib")

// 创建一个截屏类
class CScreenShotFactory : public IEventReceiver
{
public:
CScreenShotFactory( IrrlichtDevice *device, const c8 * templateName )
{
    // 存储设备指针，之后我需要用到它。
    Device = device;
    // 从 0 开始编号
    Number = 0;
    Filename.reserve ( 256 );
    FilenameTemplate = templateName;
}
bool OnEvent(SEvent event)
{
    // 如果用户按下 F9 键，则截屏并保存为 jpg 文件
    if (event.EventType == EET_KEY_INPUT_EVENT &&
      event.KeyInput.Key == KEY_F9 &&
      event.KeyInput.PressedDown == false)
    {
      video::IImage* image = Device->getVideoDriver()->createScreenShot();
      if (image)
      {
        sprintf ( (c8*) Filename.c_str() ,
            "%s_shot%04d.jpg",
            FilenameTemplate.c_str (),
            Number++
          );
        Device->getVideoDriver()->writeImageToFile(image, Filename.c_str(), 85 );
        image->drop();
      }
    }
    return false;
}
private:
IrrlichtDevice *Device;
u32 Number;
```

```
core::stringc Filename;
core::stringc FilenameTemplate;
};

/*
OK，开始
*/
int IRRCALLCONV main(int argc, char* argv[])
{
/*
类似 HelloWorld 例子，我们通过 CreateDevice 创建一个 Irr 设备，不过在这里我们允许
用户进行硬件加速设备的选择。其中软件模拟进行一次巨大的 Q3 场景的加载将会相当慢，
不过为了进行演示有这样一个功能，我们也把它列做选项了。
*/
// 让用户选择设备类型
video::E_DRIVER_TYPE driverType;
printf("Please select the driver you want for this example:\n"\
    " (a) Direct3D 9.0c\n (b) Direct3D 8.1\n (c) OpenGL 1.5\n"\
    " (d) Software Renderer\n (e) Burning's Software Renderer\n"\
    " (f) NullDevice\n (otherKey) exit\n\n");
char i;
std::cin >> i;
switch(i)
{
    case 'a': driverType = video::EDT_DIRECT3D9;break;
    case 'b': driverType = video::EDT_DIRECT3D8;break;
    case 'c': driverType = video::EDT_OPENGL;    break;
    case 'd': driverType = video::EDT_SOFTWARE; break;
    case 'e': driverType = video::EDT_BURNINGSVIDEO;break;
    case 'f': driverType = video::EDT_NULL;      break;
    default: return 1;
}
// 创建设备
core::dimension2di videoDim ( 800,600 );
IrrlichtDevice *device = createDevice(driverType, videoDim, 32, false );
if (device == 0)
    return 1;

/*
获取一个视频驱动和场景管理的指针。
*/
video::IVideoDriver* driver = device->getVideoDriver();
scene::ISceneManager* smgr = device->getSceneManager();
// 设置我们的资源目录工作夹
```

```
device->getFileSystem()->addFolderFileArchive("../../media/");
/*
为了显示 QUAKE3 的地图，我们首先需要读取它。
Quake3 地图被打包在.pk3 文件中，所以我们的文件系统需要加载.pk3 包文件，
在我们加载它之后，我们还需要从包文件中对其进行读取。
*/
device->getFileSystem()->addZipFileArchive("../../media/map-20kdm2.pk3");

/*
现在我们可以通过调用 getMesh()函数来进行 Mesh 的读取。我们获得了一个动画 Mesh
IAnimatedMesh 的指针。然而我们可能有疑问，Quake3 地图并非一个动画，我们为什么要
使用 IAnimatedMesh 动画 Mesh 呢？我们先研究下 Quake3 的地图，它是由一个巨大的模型
以及一些贴图文件组成的。我们可以理解为，它是由一个动画组成，而这个动画仅有一桢，
所以我们获得动画的第一桢 getMesh(0)(其中 0 就是指定桢数)，然后使用它创建一个八叉树
场景节点。
八叉树的作用是对场景渲染进行优化，就是仅仅渲染摄象机所见的场景，这个请自行
查看 3D 渲染相关书籍。
相对于八叉树场景节点的另一种加载方式就是直接创建一个 AnimatedMeshSceneNode，
动画 Mesh 场景节点，但是这样做的话就不会进行优化的拣选，它会一次性加载绘制所有的
场景。
在下面的代码里，我两种类型都写了，你可以切换着进行尝试一下。
值得注意的是八叉树场景的适用范围一般是大型的室外场景加载。
*/
scene::IQ3LevelMesh* mesh = (scene::IQ3LevelMesh*) smgr->getMesh("maps/20kdm2.bsp");
// 为了能够截图，我们设置一个事件接收器
CScreenShotFactory screenshotFactory ( device, "20kdm2" );
device->setEventReceiver ( &screenshotFactory );

/*
更改 Mesh 的结构类型，以获取更快的渲染速度
（译者注；此处作者原文意为创建一个几何体以获取更快的速度，这明显是错误的，
跟踪带源码中可获知，getMesh()中的参数是说明需要获取 Mesh 类型，而从一个 Mesh 中
重新获取本 Mesh,而修改了 Mesh 结构类型,必然进行的是优化结构的操作,并无创建工作）
*/
scene::ISceneNode* node = 0;
if ( mesh )
{
    scene::IMesh *geometry = mesh->getMesh(quake3::E_Q3_MESH_GEOMETRY );
    //node = smgr->addOctTreeSceneNode(geometry, 0, -1, 128);
    node = smgr->addMeshSceneNode ( geometry );
}
/*
现在，为每个场景节点创建自己的 Shader。这些 Shaders 目标存储在 QuakeMesh 场景
quake3::E_Q3_MESH_ITEMS 中，Shaders 的 ID 存储在材质参数中。
```

（译者注：此处就是与 Irr 第二个例子不同之处，它将场景中的 MeshItems 单独进行了渲染，这些 MeshItems 在本例中
表现为各个灯盏以及上面的火焰等一些附属物件）
*/
```
if ( mesh )
{
    // 这些额外的 Mesh 会非常庞大
    scene::IMesh * additional_mesh = mesh->getMesh ( quake3::E_Q3_MESH_ITEMS );
    for ( u32 i = 0; i!= additional_mesh->getMeshBufferCount (); ++i )
    {
     IMeshBuffer *meshBuffer = additional_mesh->getMeshBuffer ( i );
     const video::SMaterial &material = meshBuffer->getMaterial();
     // Shaders 索引值保存在材质参数中
     s32 shaderIndex = (s32) material.MaterialTypeParam2;
     // 普通的附加 Mesh 可以不需要额外支持的进行渲染，但是火焰 Shader 不行，它需要特殊支持
     const quake3::SShader *shader = mesh->getShader ( shaderIndex );
     if ( 0 == shader )
     {
      continue;
     }
     //core::stringc szTemp = NULL;
     //quake3::dumpShader ( szTemp, shader );
     // 通过管理器对每个 MeshBuffer 提供一个正确的 Shader，将 Shader 绑定入场景 Mesh
     smgr->addQuake3SceneNode ( meshBuffer, shader );
    }
    // 原始的 Mesh 数据已经不需要了
    mesh->releaseMesh ( quake3::E_Q3_MESH_ITEMS );
}
/*
现在我们仅仅需要一个摄象机去观察这张地图。这次我们设计一个可用户控制的
灵活摄象机。在 Irr 引擎中有许多不同类型的摄象机：例如，Maya 摄象机就类似于
Maya 软件中的摄象机控制，左键按下可进行旋转，两键按下就可以进行缩放，
右键按下就可以进行移动。假如我们想创建这样操作方式的摄象机，那么只要
addCameraSceneNodeMaya()就可以了。而我们现在需要设计的摄象机则是类似于
标准 FPS 的控制设定，所以我们调用 addCameraSceneNodeFPS()函数来创建。
*/
scene::ICameraSceneNode* camera = smgr->addCameraSceneNodeFPS();
/*
我们还需要一个不错的初始观察点，实际上，这些点在 Quake3 地图中是定义好了的，在配置中这些点的索引文字是
“info_player_deathmatch”，我们找到这些点之后，随机选择一个点做为摄象机初始点。
*/
if ( mesh )
```

```
{
    const quake3::tQ3EntityList &entityList = mesh->getEntityList ();
    quake3::SEntity search;
    search.name = "info_player_deathmatch";
    s32 index = entityList.binary_search_const ( search );
    if ( index >= 0 )
    {
      const quake3::SVarGroup *group;
      s32 notEndList;
      do
      {
        group = entityList[ index ].getGroup(1);
        u32 parsepos = 0;
        core::vector3df pos = quake3::getAsVector3df ( group->get ( "origin" ), parsepos );
        parsepos = 0;
        f32 angle = quake3::getAsFloat ( group->get ( "angle"), parsepos );
        core::vector3df target ( 0.f, 0.f, 1.f );
        target.rotateXZBy ( angle, core::vector3df () );
        camera->setPosition ( pos );
        camera->setTarget ( pos + target );
        index += 1;
        notEndList = ( index < (s32) entityList.size () &&
              entityList[index].name == search.name &&
              (device->getTimer()->getRealTime() >> 3 ) & 1
            );
      } while ( notEndList );
    }
}
/*
屏蔽鼠标图标
*/
device->getCursorControl()->setVisible(false);
// 读取一个 Irr 引擎 LOGO
gui::IGUIEnvironment* env = device->getGUIEnvironment();
env->addImage(driver->getTexture("irrlichtlogo2.png"),core::position2d<s32>(10, 10));
// 根据设备不同添加不同的设备 Logo
core::position2di pos ( videoDim.Width - 128, videoDim.Height - 64 );
switch ( driverType )
{
    case video::EDT_BURNINGSVIDEO:
      env->addImage(driver->getTexture("burninglogo.png"),pos );
      break;
    case video::EDT_OPENGL:
      env->addImage(driver->getTexture("opengllogo.png"),pos );
```

```
        break;
    case video::EDT_DIRECT3D8:
    case video::EDT_DIRECT3D9:
        env->addImage(driver->getTexture("directxlogo.png"),pos );
        break;
}
/*
我们做完了所有的事情，现在我们开始绘制它吧。我们还需要在窗口的标题上
显示当前的 FPS。
if (device->isWindowActive()) 这一行代码是可选的，但是为了预防由于
切换活动窗口而导致引擎渲染桢速率显示不正确，还是加上吧。
*/
int lastFPS = -1;
while(device->run())
if (device->isWindowActive())
{
    driver->beginScene(true, true, video::SColor(255,20,20,40));
    smgr->drawAll();
    env->drawAll();
    driver->endScene();
    int fps = driver->getFPS();
    //if (lastFPS != fps)
    {
        io::IAttributes * attr = smgr->getParameters();
        s32 calls = attr->getAttributeAsInt ( "calls" );
        s32 culled = attr->getAttributeAsInt ( "culled" );
        core::stringw str = L"Q3 [";
        str += driver->getName();
        str += "] FPS:";
        str += fps;
        str += " Cull:";
        str += calls;
        str += "/";
        str += culled;
        device->setWindowCaption(str.c_str());
        lastFPS = fps;
    }
}
/*
最后，删除渲染设备
*/
device->drop();
return 0;
}
```