文档:教程:基础教程:基础教程三

出自 Ogre3D 开放资源地带

跳转到: 导航, 搜索

目录

先决条件

简介

在这篇教程里，我们将会一起探索 OGRE 中的天空，地面和雾化处理。通过这篇教程，您应该明白天空盒(Skybox)，天空穹(Skydome)和天空面(Skyplane)的用法和区别。您还会了解不同种类的雾化效果，以及它们的使用方法。

从这里开始

和以往一样，我们将使用已经写好的代码作为模版。在您的编译器里创建一个新的工程，然后把下面的代码添加进去：

```
#include "ExampleApplication.h"

class TutorialApplication : public ExampleApplication
{
protected:
public:
    TutorialApplication()
    {
    }

    ~TutorialApplication()
    {
    }
protected:
    void chooseSceneManager(void)
    {
    }

    void createScene(void)
    {
    }
};

#if OGRE_PLATFORM == PLATFORM_WIN32 || OGRE_PLATFORM ==
OGRE_PLATFORM_WIN32
#define WIN32_LEAN_AND_MEAN
#include "windows.h"

INT WINAPI WinMain( HINSTANCE hInst, HINSTANCE, LPSTR strCmdLine,
INT )
#else
int main(int argc, char **argv)
#endif
{
    // Create application object
    TutorialApplication app;

    try {
        app.go();
    } catch( Exception& e ) {
#if OGRE_PLATFORM == PLATFORM_WIN32 || OGRE_PLATFORM ==
OGRE_PLATFORM_WIN32
        MessageBox( NULL, e.getFullDescription().c_str(), "An
exception has occured!", MB_OK | MB_ICONERROR | MB_TASKMODAL);
```

```
#else
        fprintf(stderr, "An exception has occured: %s\n",
                e.getFullDescription().c_str());
#endif
    }

    return 0;
}
```

假如您能够成功编译这段代码，您在运行的时候可以用 WASD 键移动，鼠标来转镜头，ESC 键来退出程序。

根对象和场景管理器的创建

根对象

首先我们需要让 OGRE 创建一个地面。我们必须将场景管理器设为地面的场景管理器，而不是 ExampleApplication 里默认的那个。将下面的代码添加到 chooseSceneManager 函数里：

```
        mSceneMgr = mRoot->createSceneManager(ST_EXTERIOR_CLOSE);
```

根对象是 OGRE 的核心对象，您可以在这里看到 OGRE 对象的 UML 图。到目前为止，您已经见到过除了 RenderSystem 以外大部分的对象了。在上面的代码中，我们告诉根节点我们需要一个 ST_EXTERIOR_CLOSE 类型的场景管理器。紧接着，根对象会要求场景枚举器查找您要的场景管理器然后将它返回。

等你的程序设置好了之后，你几乎不会再接触根对象了，你也不会直接调用场景枚举器。

场景管理器的创建

我想先讲一下场景管理器的创建和管理，以免您在后面产生误会。场景管理器不是单例，您想创建多少就创建多少。与场景节点/灯光等不同的是，您可以直接使用 "new SceneManager()" 这种语句来直接创建它（而不必使用 Root 的 createSceneManager 方法，但你的确应该这么用）。你可以有多个场景管理器，来同时装载多个独立的几何体和实体。你可以通过重建视口的方式在任何时间交换这些场景管理器（这部分内容详见中级教程 4）或者使用多视口方法同时显示多个场景管理器（这会在高级教程中有所涉及）。

我们为什么使用 createSceneManager 函数，而不是手动地创建我们的场景管理器对象呢？当我们使用单一场景管理器时，Ogre 的插件系统给我们以巨大的灵活性。在 SceneType 枚举类型中仅定义了几种场景类型。直到现在，ExampleApplication 例程还一直使用 ST_GENERIC 作为我们的场景管理器。你也许认为这里用的是作为基础的 SceneManager 类，但是只要你没有修改

plugins.config 文件，那么你用的就不是它。如果你使用了 OctreeSceneManager 插件的话，OctreeSceneManager 会将自身注册为 ST_GENERIC 类型，并且覆盖作为基础的 SceneManager 类。OctreeSceneManager 使用了拣选不可见对象的系统，因此它比标准的 SceneManager 要快）。如果你从 plugins.cfg 文件中去除了 OctreeSceneManager 插件，那么当你请求 ST_GENERIC 类型时，你才*可能*会使用基础的 SceneManager，也许会根据你的插件设置使用更好的场景管理器，这就是 Ogre 系统的优美所在。

迄今为止我们只用了 Root 对象最基本的一些方法来创建场景管理器。实际上，我们可以使用一个字符串而不是 SceneType 枚举来请求一个场景管理器，Ogre 灵活的 SceneManager 工厂允许你你定义任意数量的场景管理器类型并在需要的时候创建和销毁它们。例如我们使用了一个插件，允许你创建叫做 "FooSceneManager"类型的场景管理器，我们这样来创建一个：

```
 // do not add this to the project
mSceneMgr = mRoot->createSceneManager("FooSceneManager");
```

这会创建这样一个场景管理器并给它一个默认名称，虽然我在这个教程里不会这么用，你始终应该使用 createSceneManager 的第二个参数来給场景管理器命名。例如我们要创建两个不同名称的场景管理器可以这么做：

```
// do not add this to the project
mSceneMgr1 = mRoot->createSceneManager("FooSceneManager",
"foo");
mSceneMgr2 = mRoot->createSceneManager("FooSceneManager",
"bar");
```

通过给场景管理器命名，我们就不再需要再保留指针来跟踪它们。Root 对象会替我们搞定，之后你要使"foo"和"bar"场景管理器的时候，这样做就行了：

```
// do not add this to the project
SceneManager *foo = mRoot->getSceneManager("foo");
SceneManager *bar = mRoot->getSceneManager("bar");
```

当你彻底用不再使用一个场景管理器时，使用 Root 的 destroySceneManager 来删除它释放内存。

尽管我们不会通过教程来讲解，你同样可以通过继承 SceneManagerFactory 类定义一个自己的 SceneManager 工厂。如果你创建了自己的场景管理器，或者想在你的程序使用一个标准的场景管理器之前对它做一些修改（比如创建摄像机，创建光照，加载几何体等等），这将会非常有用。

地形

在场景中添加地面

现在我们已经搞清楚了根对象和场景管理器，是时间来真正创建一个地面了（当然是在 OGRE 里）。场景管理器的基类定义了一个叫做 setWorldGeometry 的方法，使派生类便于创建场景。当使用地面场景管理器(TerrainSceneManager)时，它需要一个文件名来加载地面属性。在 createScene 函数中添加这行代码：

```
mSceneMgr->setWorldGeometry( "terrain.cfg" );
```

编译运行你的程序，稍微移动一下镜头就可以看到 OGRE 生成的地面了，简单吧？

"terrain.cfg" 文件

terrain.cfg 文件包含了许多生成地面的选项，和以往一样，我只会讲一些基础的东西，更详细的内容可以在/*这里*/找到。需要注意的是，地面场景管理器设计的时候包含了分页的功能，只不过还没有被实现。不过 OGRE 有一个插件目前可以实现这个功能：Paging Scene Manager。

地面场景管理器使用高度图(Heightmap)来生成地面。你可以通过更改 Heightmap.image 的参数来改变高度图。你可以通过更改 WorldTexture 来更改地面的帖图。你还可以通过更改 DetailTexture 来使地面看得更逼真。你可以在/Media/materials/textures 里找到 terrain.cfg 默认的几个图片。

照亮地面

在上篇教程里我们刚学过如何使用灯光和阴影，不幸的是他们很难用在地面上。使用已经处理过光照效果的帖图会比真的光照容易许多。在雾化处理那一节，我们还会讨论如何使用“伪黑暗”。假如你想用真正的光照效果，你应该考虑一下 Paging Scene Manager，他有很好的光照支持。

天空

OGRE 提供三种天空：天空盒，天空穹和天空面。我们会逐个学习他们，现在你必须在 chooseSceneManager 函数中添加以下代码:

```
ResourceGroupManager::getSingleton().initialiseAllResourceGroups();
```

天空盒

天空盒实际上是一个包含了场景里所有对象的巨型立方体。耳闻不如眼见，将下面代码添加到 createScene 里，自己体会什么是天空盒吧：

```
mSceneMgr->setSkyBox( true, "Examples/SpaceSkyBox" );
```

编译运行你的程序。怎么样，效果不错吧？(你可以更改高像素的贴图使它看起来更逼真。) 当我们调用 setSkyBox 的时候我们可以设置几个有用的参数。第一个代表是否启用天空盒。假如你什么时候想要取消天空盒你只须调用 mSceneMgr->setSkyBox(false, ""); 第二个参数是天空盒使用的材质脚本。

setSkyBox 的第三个和第四个参数并不是十分重要。第三个参数设定了天空盒与摄像机的距离，第四个参数决定了天空盒是在其他对象之前渲染还是其他对象之后。我们来试试将第三个参数从默认的 5000 改为一个很小的值然后看看有什么效果：

```
mSceneMgr->setSkyBox( true, "Examples/SpaceSkyBox", 10 );
```

什么都没改变！这是因为第四个参数的默认值是 true，也就是说天空盒会最先渲染，其它的东西会渲染到天空盒之上，从而使天空盒看起来像是在背后。（注意你不应该让第三个参数小于摄像机的近剪切面距离，否则他会不可见！）其实天空盒并不应该最先渲染，因为你会渲染他的全部。假如你最后渲染他，OGRE 只会渲染可见的部分，从而提高一定的运行速度。所以，我们来试一下最后渲染天空盒：

```
mSceneMgr->setSkyBox( true, "Examples/SpaceSkyBox", 5000,
false );
```

现在它看起来好像一样没有任何改变，但是天空盒的不可见的部分不会被渲染。有一点要注意，当你把天空盒设置得很近时，部分的几何会被裁剪掉，试一下这个：

```
mSceneMgr->setSkyBox( true, "Examples/SpaceSkyBox", 100,
false );
```

看到了么？地面看上去穿过了天空。这肯定不是你想要的结果（假如你还正常的话）。假如你要在你的程序中使用天空盒你必须决定如何使用它。当你最后渲染天空盒是，程序的运行速度会提高一些，但是你必须小心它不要遮住你的几何。总的来说，将第二个参数之后的所有东西都设为默认是最安全的。

天空穹

天空穹和天空盒很相像，你需要使用 setSkyDome 来创建天空穹。他会创建一个包含了场景中所有对象的巨型立方体，最大的区别是贴图被用球体的方法投影到立方体上。你看到的其实还是一个立方体，只不过它的贴图看上去像贴到了一个球体上一样。这种天空有一个很大的缺陷，就是立方体的下面没有任何贴图，所有你必须要有一个类似地面的东西来遮盖下面。

OGRE 的示例贴图将让你很明显地看到这个缺陷。删掉 setSkyBox 然后在 createScene 中添加下面代码：

```
mSceneMgr->setSkyDome( true, "Examples/CloudySky", 5, 8 );
```

当你运行时，将镜头移动到地面的中间，然后是镜头几乎和地面平行。按 R 键切换到网格状态，你会发现你看到的仍然是一个立方体（没有底的），但是那些云彩看起来却像覆盖在球体上一样。（注意云彩的变换是在"Examples/CloudySky"脚本里定义的，并不是所有的天空穹都有这样的效果。）

天空穹的前两个参数和天空盒一样，第三个参数是天空穹的弯度。OGRE 的 API 建议使用 2 到 65 之间的数值，降低第三个参数会有更好的距离感，增加第三个参数会降低扭曲度从而达到更平滑的效果。试着将它设为 2 然后再设为 65 来观察它们的区别，下面截图可以很好的展示天空的弯曲效果，这是当第三个参数为 2 时的效果：

http://www.idleengineer.net/images/beginner03_2.png

这是当第三个参数为 64 时的效果：

http://www.idleengineer.net/images/beginner03_64.png

第四个参数是贴图重复的次数，你需要根据你贴图的大小来设置它。但是注意这个参数是 Real 类型(浮点数)的而不是整形。第五第六个参数分别是距离和渲染顺序，与天空盒的相同。

天空面

天空面与前两种天空有相当大的区别。我们用一个平面来替代正方体。（注意下面所有的天空面的属性都是尽量靠地面中间）删掉 createScene 里所有天空穹的代码。我们要先创建一个平面，然后是他朝下。setSkyPlane 与前两个函数不同的是他没有距离参数。相应的，我们在 Plane 的 d 变量里设置它的距离。

```
Plane plane;
plane.d = 1000;
plane.normal = Vector3::NEGATIVE_UNIT_Y;
```

现在我们定义了平面，接下来就可以创建天空面了。注意第四个参数是天空面的大小（在这里是 1500x1500 个单位）第五个参数是他重复的次数：

```
mSceneMgr->setSkyPlane( true, plane, "Examples/SpaceSkyPlane", 1500, 75 );
```

编译运行你的程序。咱们创建的天空面有两个缺陷。第一个是由于贴图像素低，重复的时候不好看。你自己画一个高像素边缘处理好的贴图就行了。最主要的缺陷是当你注视水平线的时候，你能看到天空面的结尾。即使你有一个好的贴图，他看起来并不会很逼真假如你能看到天空的结尾。天空面的这种用法只能用在盆地或者四周都是墙的场景。不过他对显卡的要求会比天空盒和天空穹要低。

幸运的是天空面的功能还不仅是这些。第六个参数跟天空盒和天空穹的渲染顺序很相似。第七个参数允许你设置天空面的弯曲度，这样一来平面就变成弧形的了。同时我们还需要设置 x 和 y 的线段数量（天空面是一个巨大的正方形，但是假如我们想让他弯曲那么它就要变成许多小正方形）。第八个和第九个参数就是 x 和 y 的线段数量了：

```
        mSceneMgr->setSkyPlane( true, plane, "Examples/SpaceSkyPlane",
1500, 50, true, 1.5f, 150, 150 );
```

编译运行你的程序。现在这个天空面好看多了，你还可以用云彩的那个材质：

```
        mSceneMgr->setSkyPlane( true, plane, "Examples/CloudySky",
1500, 40, true, 1.5f, 150, 150  );
```

编译运行你的程序。云彩的移动以及他重复的方法比天空穹还差一点，尤其是当你接近地面的边缘往水平线的方向看去时。

还有一点，你可以通过 `mSceneMgr->setSkyPlane( false, Plane(), "" );` 来取消天空面。

用哪个好?

究竟用哪种天空完全取决于你的程序。假如你需要看到周围所有的东西，包括负 y 的方向，那你可能只有天空盒这个选择了。假如你有一个地面或者类似地板的东西可以遮住负 y 方向，那么使用天空穹会有一个更真实的效果。假如你看不到海平线（比如一个峡谷，一个监狱，或者城堡中间的院子），天空面会有一个很好的效果同时只有很低的 GPU 使用量。我们会在下一节中来解释用天空面的最重要的一个原因，是因为它能和云雾很好地结合在一起。

这些只是建议。当你自己写程序的时候应该把这几种都试试然后看看哪种最好再来决定最终用哪个。

雾化效果

入门

雾化效果在 OGRE 里非常简单。但是在你使用它之前要注意，当你使用地面管理器(TerrainSceneManager)的时候，一定要在调用 setWorldGeometry 函数之前

调用 setFog 函数。(别的场景管理器里没关系的)。根据你调用这两个函数的顺序，不同的顶点程序会被调用。(注意在 OGRE 1.0.0 里有一个 Bug，当你使用正确顺序调用这些函数的时候，指数雾化不会被渲染。这个 Bug 在 1.0.1 的时候就已经被修复了)。

在我们开始之前，清除掉 createScene 函数里除了 setWorldGeometry 之外的所有内容。

你需要知道关于设置雾化效果的很重要的一点是他并不像你想象的那样在空的地方创建“雾”的实体。实际上雾只是当你观看物体时的一个滤镜。当你面前没有任何物体时，你是看不见雾的。事实上，你只会看到视口(viewport)的背景色。所以想要使雾逼真，我们必须将视口的背景色设成雾的颜色。

雾分两种：线性的和指数的。线性雾“线性的”增浓，而指数雾“指数的”增浓（每个距离单位雾的浓度都会比上一个单位增加得更多）。你自己看他们效果的差距比这样讲更容易理解，所以我们来看几个例子。

雾化类型

我们将看到的第一种雾的类型是线性的，这也是最容易理解的类型。在调用 setWorldGeometry 后首先要设置视口的背景颜色。我们可以用 createViewport 方法（象用在上一个指南）一样，但是有时我们并不需要调用视口。代码如下：

```
ColourValue fadeColour(0.9, 0.9, 0.9);
mWindow->getViewport(0)->setBackgroundColour(fadeColour);
```

如果视口不唯一的话，你可以用 getNumViewports 元函数来得到视口的数量然后不断的操作她，但是通常这种情况是很少见的（目前为止我们知道我们仅用了一个视口），我们可以直接得到视口。设置完背景色以后我们就可以创建雾了。记住，这段代码必须出现在 setWorldGeometry 被调用之前。

```
mSceneMgr->setFog(FOG_LINEAR, fadeColour, 0.0, 50, 500);
```

第一个参数设置雾的类型（这里我们设的是线性的）。第二个参数是我们用到的雾的颜色。第三个参数在线性雾里面是不用设置的。第四个和第五个参数是设置雾慢慢变浓的范围。这里我们从 50 开始结束于 500。这就意味着在摄像机的 0 到 50 个单位内是没有雾的。从 50 到 500 单位内雾慢慢线性变浓。在 500 单位以外全是雾了。编译运行这个应用程序。

另一种雾的类型是指数雾。和设置雾的起点和终点不一样的是，我们要设置雾的密度（第四个和第五个参数不需要设置）。代码如下：

```
mSceneMgr->setFog(FOG_EXP, fadeColour, 0.005);
```

编译运行这个应用程序。如果你用的是 DirectX 渲染器你会发现完全在雾的外面，在 setWorldGeometry 之前调用 setFog 来修正它。OpenGL 渲染器会按照文档里说的那样做，这能产生另一种生成雾的效果。这里还有一种更厉害的指数雾函数（与前一种比较起来，离摄像机越远它的雾更浓）。注意，如果你使用 FOG_EXP2 会得到更加浓密的雾。将上面的代码替换成下面的：

```
mSceneMgr->setFog(FOG_EXP2, fadeColour, 0.003);
```

再一次编译和运行这个程序。基本上 Ogre 提供的这三种雾函数是可以互换的。你应该都实验一下这些函数，来看看哪一种在你的应用程序里效果更好。

雾与天空面

当你试图将雾与天空盒/天空穹一起使用时，会遇到些有趣的问题。因为天空盒/天空穹是方形的，而雾效是以球形的方式工作的。清空 createScene 方法里的内容，如果巧妙地调整天空穹和雾的参数，我们就能明显地发现这个问题。

```
ColourValue fadeColour(0.9, 0.9, 0.9);
mSceneMgr->setFog(FOG_LINEAR, fadeColour, 0.0, 50, 515);
mWindow->getViewport(0)->setBackgroundColour(fadeColour);
mSceneMgr->setWorldGeometry("terrain.cfg");
mSceneMgr->setSkyDome(true, "Examples/CloudySky", 5, 8, 500);
```

编译运行程序。如果你转动摄像机，你会发现天空穹的某些部分会探出来（蓝色部分从边缘探出来，而不是中央）：

http://www.idleengineer.net/images/beginner03_fogbox.png

这显然不是我们想要的。另一种选择就是使用天空面。在 createScene 里用以下代码替换：

```
ColourValue fadeColour(0.9, 0.9, 0.9);
mSceneMgr->setFog(FOG_LINEAR, fadeColour, 0.0, 0, 130);
mWindow->getViewport(0)->setBackgroundColour(fadeColour);

Plane plane;
plane.d = 100;
plane.normal = Vector3::NEGATIVE_UNIT_Y;

mSceneMgr->setWorldGeometry("terrain.cfg");

mSceneMgr->setSkyPlane(true, plane, "Examples/CloudySky", 500,
20, true, 0.5, 150, 150);
```

这样看起来就对了。如果我们往上看，就能看见天空（真实情况也是这样），而不会滑稽地从别处探出来。不管你是否使用弯曲，这都能解决当你往地平线上看的时候出现的不正常现象。

还有一种办法能让雾效不涉及整个天空，但需要对材质脚本进行修改。这已不是本课的内容，为了今后参考，它就是材质里禁用雾效的那个参数。

黑暗中的雾

当你设置雾的时候，可能不想使用天空，因为雾已经浓到你无法看见天空了。上面讲的雾效的一些技巧在某些场景非常有用。不把雾设置成明亮的颜色，而是设成非常暗，看看有什么效果。（注意，我们把天空面设置成离摄像机只有10个单位距离，在设置雾之前）：

```
        ColourValue fadeColour(0.1, 0.1, 0.1);
        mWindow->getViewport(0)->setBackgroundColour(fadeColour);
        mSceneMgr->setFog(FOG_LINEAR, fadeColour, 0.0, 10, 150);

        mSceneMgr->setWorldGeometry("terrain.cfg");

        Plane plane;
        plane.d = 10;
        plane.normal = Vector3::NEGATIVE_UNIT_Y;

        mSceneMgr->setSkyPlane(true, plane, "Examples/SpaceSkyPlane",
100, 45, true, 0.5, 150, 150);
```

编译运行程序。这就是我们得到的：

http://www.idleengineer.net/images/beginner03_darkness.png

不算太糟糕。当然，一旦你会使用光照，你就不必用这种技巧了。但这的确展示了雾效的灵活性，以及用这个引擎你能做一些有趣的事情。如果你正写一个第一人称游戏，使用黑色雾能够制造“失明”或“符咒”这样的特效。

原文
上一章节：基础教程二
下一章节：基础教程四
目录

7%A8%8B:%E5%9F%BA%E7%A1%80%E6%95%99%E7%A8%8B:%E5%9F%BA%E7%A1%80%E6%95%99%E7%A8%8B%E4%B8%89"

查看

- 页面
- 讨论
- 源码
- 历史

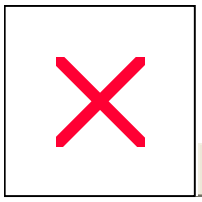
个人工具

- 登录 / 创建账户

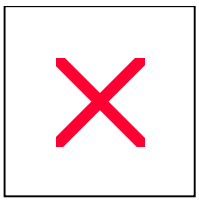
导航

- 首页
- 社区
- 当前事件
- 最近更改
- 随机页面
- 帮助

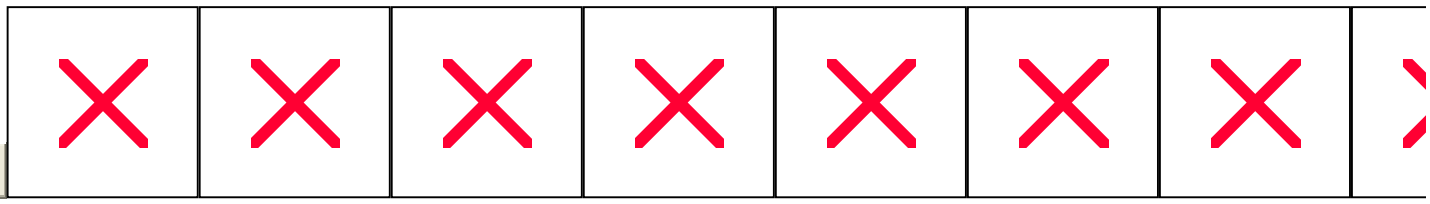
搜索


进入 搜索

Google **搜索**


our search terms

search form


earch

工具箱

- 链入页面
- 链出更改
- 特殊页面
- 可打印版
- 永久链接

Google Adsense