文档:教程:基础教程:基础教程二

出自 Ogre3D 开放资源地带

跳转到: 导航, 搜索

本章翻译由收费打工仔完成

目录

[隐藏]

先决条件

本教程假定你已经拥有了 c++程序设计的知识，并且已经安装和编译了一个 Ogre 的应用程序（如果你在设置你的应用程序中有困难，请参考 this guide 获得更

详细的编译步骤）。这个教程同时也是建立在上一章基础上的，因此默认你已经了解了上个教程的内容。

简介

在这篇教程里，我会向您介绍几个新的构架，同时还会补充一些过去的内容。这篇教程主要介绍如何使用灯光对象以及如何产生阴影。我们还会稍微了解一下摄像机的用法。当你看完这篇教程以后，你应该试着慢慢的添加一些代码到你自己的工程里，然后看看结果。你可以参考这个教程里最终状态的源代码。如果你在写代码时遇到了困难，也可以和最终工程里的源文件对比一下。

从这里开始

像上一个教程一样，我们将使用一个先前建立的代码作为我们出发的起点。我们将增加两个方法到 TutorialApplication class 中：createViewport 和 createCamera。这两个方法我们已经在基类 ExampleApplication 中定义了，在这个教程中我们将看到摄像机和视口具体建立和使用。

为这个项目在编译器中创建一个工程，添加源文件包含这些代码：

```
#include "ExampleApplication.h"
class TutorialApplication  : public ExampleApplication
{
protected:
public:
    TutorialApplication()
    {
    }
    ~TutorialApplication()
    {
    }
protected:
    virtual void createCamera(void)
    {
    }

    virtual void createViewports(void)
    {
    }

    void createScene(void)
    {
        Entity *ent;
```

```
        Light *light;
    }
};

#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 occurred!", MB_OK | MB_ICONERROR | MB_TASKMODAL);
#else
        fprintf(stderr, "An exception has occurred:  %s\n",
                e.getFullDescription().c_str());
#endif
    }

    return 0;
}
```

如果你是在 Windows 下使用 OgreSDK 的，请确定添加“"[OgreSDK_DIRECTORY]\samples\include”目录到这个工程(ExampleApplication.h 文件所在的位置)除了标准包含以外。如果使用的是 Ogre 的源代码包，这个文件应该在“"[OgreSource_DIRECTORY]\Samples\Common\include”目录中。请确保你可以编译这段代码在进入下一阶段之前（我们并没有运行，因为他是会崩溃，直到随后给他添加更多的东西）。我们稍后将添加代码来完成这项工作。如果你有问题，请查看这个 Wiki 页获得设置你编译器的信息，如果仍然有问题请试着去看看帮助栏。

程序控制：用键盘上 WASD 来移动，用鼠标观察四周。Ese 键退出。

摄像机

Ogre 的摄像机

摄像机是用来观察我们所创建的场景的。摄像机是一个特殊的物体，她的工作方式有点类似于场景节点。摄像机可以设置位置进行偏移，滚动，倾斜操作和绑定到任意场景节点上。就像场景节点一样，摄像机的位置跟父节点有关（就好像尊重长辈一样）。对所有的移动和旋转来说，你可以把摄像机考虑成场景节点。

对于 Ogre 摄像机来说，有一点跟你想象不一样的是同一个时间内你只能使用一台摄像机（当前）。这就是说，我们不能创建其中一个摄像机去观察场景的一部分，而用第二个去观察场景的另一部分，并且仅通过打开或关闭某一个摄像机就可以来呈现我们想要的一部分场景。不过要完成这一步有一个替代方式就是通过一些创建场景节点来扮演“摄像机句柄”。这些场景节点简单的参加到场景中指向摄像机可能要捕捉的位置。当只需要展示一部分场景时，摄像机只需要简单的把自己绑定到适当的节点上就可以了。我们会在帧监听教程里重温这项技术。

创建一个摄像机

我们将通过替换默认的 ExampleApplication 方法来创建摄像机。

找到 TutorialApplication::createCamera 方法。首先，我们将创建摄像机。场景管理器里有自带的摄像机。我们就以此来创建我们的摄像机。添加下面的代码到创建摄像机方法里：

```
    // 创建摄像机
mCamera = mSceneMgr->createCamera("PlayerCam");
```

这将创建一个叫"PlayerCam"的摄像机。如果你不打算保留它的指针，那么你可以调用场景管理器的方法 getCamera 通过传递它的名字来获取它。

接下来，我们继续设置摄像机的初始位置和朝向。我们将设置摄像机朝向坐标原点，所以我们需要设置一个恰当的 Z 轴距离。添加下面的代码到刚才的后面：

```
    //设置摄像机位置和方向
mCamera->setPosition(Vector3(0,10,500));
mCamera->lookAt(Vector3(0,0,0));
```

lookAt 方法是灵活的。你能够设置它面向任何你想要的角度。场景节点也有同样的方法，能够通过此方法在任何情况下，很容易的达到你想要的方向。最后，我设置一个 5 单位距离的近距离裁剪。通过设置摄像机的裁剪距离，来规定该距离内的事物是你看不到的。设置近距离裁剪后，你能够在离实体很近的时候，很

容易的透过实体看到场景。还有一种很微小的情况就是，当你和某个物体很近时，这个物体将填满你的整个视野。同样，你也可以设置远距离裁剪。这个值的设置能够让引擎不去渲染远于它的物体。这主要在你需要远距离渲染大场景的时候，用来增加帧速。添加如下代码，以实现近距离裁剪：

```
mCamera->setNearClipDistance(5);
```

设置远距离裁剪和上面一样，只是方法名是这个：setFarClipDistance。当然，在这个例子里，你没必要设置远距离裁剪。

视口

Ogre 的视口

当你开始处理多个摄像机时，视口类的概念就变得对你很有用了。我提出这个主题的原因是，我认为了解当渲染一个场景时 Ogre 是如何决定使用哪个摄像机的是非常重要的。在同一个时间内 Ogre 中可能有多个场景管理器在运行。也有可能把场景分为多个区域，然后用分开的摄像机去渲染场景中不同的区域（例如：思考一下控制台游戏中两个人的观察）。然而我们可能去做这些事情的时候不用考虑他们是如何被做到的，这是高级教程的内容。

为了了解 Ogre 是怎样渲染场景的，考虑一下 Ogre 中建立的这些东西：摄像机，场景管理器，和渲染窗口。渲染窗口我们还没有涉及到，但是她是所有物体呈现的最基本的窗口。场景管理器实体创建摄像机去观察场景。你必须告诉渲染窗口哪些摄像机去呈现场景，哪部分窗口被渲染进来。你告诉渲染窗口的摄像机所呈现的区域就是视口，并且这里只有一个视口实体。

在这篇指南中我们将介绍怎样注册摄像机来创建视口。我们可以通过运用这个视口实体来设置正在渲染的场景的背景色。

建立视口

我们将重点关注 ExampleApplication 里视口的建立，因此找到 TutorialApplication::createViewports 的成员函数。建立视口的方法通常是简单的调用渲染窗口中 addViewport 函数，把她提供给正在使用的摄像机。ExampleApplication 类已经通过我们的渲染窗口移到 mWindow 类里面了，因此添加这些代码：

```
// Create one viewport, entire window
Viewport* vp = mWindow->addViewport(mCamera);
```

现在我们拥有自己的视口了，我们可以用他做什么呢？答案是：不多。最重要的事情我们可以做的是调用 setBackgroundColour 方法给背景设置任意我们选择的颜色。因为在这章中我们要处理光照因此我们把背景设置为黑色。

```
vp->setBackgroundColour(ColourValue(0,0,0));
```

注意颜色值分别是红绿蓝三种颜色的值他们的参数在 0 和 1 之间。最后也是最重要的事情是我们需要设置摄像机的纵宽比。如果你用的是一些非标准全窗口视口，而未能设置这个的话结果会出现一个非常奇怪的场景图。我们立刻设置她尽管我们使用的是默认的纵宽比：

```
// Alter the camera aspect ratio to match the viewport
mCamera->setAspectRatio(Real(vp->getActualWidth()) /
Real(vp->getActualHeight()));
```

这就是所有的我们对视口的简单运用。

这时你应该首先去编译一下然后运行这个应用程序，尽管什么也呈现不了只是黑的场景（用 Esc 来退出）。确保你可以运行这个程序不会崩溃再继续。

光照和阴影

Ogre 支持的阴影类型

Ogre currently supports three types of Shadows: Ogre 目前支持 3 种类型的阴影（当前版本 1.4.x 目前已经支持四种阴影类型）：

1. 调制纹理阴影 (SHADOWTYPE_TEXTURE_MODULATIVE) - 是这 3 种中最节省资源的。她创建一个阴影投射者的黑与白渲染到纹理，然后用于场景中。
2. 调制模板阴影 (SHADOWTYPE_STENCIL_MODULATIVE) - 这项技术是在所有的非透明体被渲染到场景以后再渲染所有的阴影体来调制阴影，她耗费资源没有加成模板阴影强，但是也不是非常精确。
3. 加成模板阴影 (SHADOWTYPE_STENCIL_ADDITIVE) - 这项技术是渲染每一个光源作为分离的部分附加到场景中。 对显卡来说这是比较麻烦的，因为在场景中每一个附加的光源需要一个附加渲染通路。

Ogre 不支持软阴影作为引擎的一部分。如果你想用软阴影，你需要写你自己的顶点和片段语言。注意这只是一个简单的介绍。Ogre 手册完整描述了阴影以及他们的用法。

在 Ogre 中使用阴影

用阴影在 Ogre 中相对来说是比较简单的。SceneManager 类中有一个 setShadowTechnique 成员函数，我们可以使用她设置我们想要的阴影类型。这样只要你创建了一个实体，调用 setCastShadows 方法来设置这个实体是否投射阴影。我们现在将设置环境光全变暗，然后设置阴影类型。找到 TutorialApplication::createScene 成员函数添加如下代码：

```
mSceneMgr->setAmbientLight(ColourValue(0, 0, 0));
mSceneMgr->setShadowTechnique(SHADOWTYPE_STENCIL_ADDITIVE);
```

现在场景管理器运用的是加成模板阴影。让我们创建一个实体然后让他投射阴影。

```
ent = mSceneMgr->createEntity("Ninja", "ninja.mesh");
ent->setCastShadows(true);

mSceneMgr->getRootSceneNode()->createChildSceneNode()->attachObject(ent);
```

另外，ninja.mesh 已经通过 ExampleApplication 为我们预载入了。我们还需要一些东西来安放 Ninja（就是需要一些东西来接收她投射的阴影）。为了做到这个我们需要为此创建一个平面。这并不意味着我们需要用到 MeshManager 指南，但是我将复习一些更基础的东西因为我们要用她来创建一个平面。一开始我们需要定义平面实体本身，她需要提供一个法线和一个到原点的距离。（举个例子）我们可以用平面来组织世界几何体的一部分，在这种情况下我们需要指定一些东西与原点之间的距离不为 0. 现在我们仅仅想要一个平面把 y 正半轴作为她的法线（这意味着我想让她面朝上），与原点的距离为 0:

```
Plane plane(Vector3::UNIT_Y, 0);
```

现在我们需要注册这个平面以便于我们可以把她运用到我们的应用程序中。MeshManager 类了解所有读到我们应用程序中的模型（例如：她了解我们用到的 robot.mesh 和 ninja.mesh）。createPlane 成员函数接收一个平面定义通过参数制造的一个模型。这些是注册平面所用到的：

```
MeshManager::getSingleton().createPlane("ground",
ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME, plane,
1500,1500,20,20,true,1,5,5,Vector3::UNIT_Z);
```

另外，我现在不想探究怎样用 MeshManager 的细节（如果你想准确的了解每个参数的作用，请参考 API）。基本上我们已经注册了我们的平面尺寸为 1500×1500，新模型名字叫“ground”。现在，我们可以通过模型创建一个实体把她放置到场景中：

```
ent = mSceneMgr->createEntity("GroundEntity", "ground");
```

```
mSceneMgr->getRootSceneNode()->createChildSceneNode()->attachObject(ent);
```

漂亮吧？哈！对 ground 来说在结束之前我们还有两件事情要做。首先是告诉场景管理器我们不想把她设置成投射阴影体直到她被用做阴影体。第二件事是我们需要在上面贴纹理。我们的 robot 和 ninja 模型已经拥有了为他们定义的材质脚本。当我们手动去创建 ground 模型的时候，我们不需要指定用什么纹理。我们将用"Examples/Rockwall""Examples/Rockwall"材质脚本，Ogre 包含以下这个例子：

```
 ent->setMaterialName("Examples/Rockwall");
ent->setCastShadows(false);
```

现在场景中我已经有一个 Ninja 和 ground，让我们来编译和运行这个程序。我们等待...什么也没有！！发生什么事了？上一个指南中我们明明添加了 Robots 而且她也被呈现出来了。原因是 Ninja 没有露出来，因为场景中的环境光设置的是全黑。因此让我们添加光源看看会发生什么。

故障排除

一些人报道在用 GCC 和 code::blocks 时他们的程序片段中遇到一些问题。如果收到类似这样的错误：

variable 'vtable for Ogre::MeshPtr' can't be auto-imported. Please read the documentation for ld's --enable-auto-import for details.

You'll need to add the following to your linker options: 你需要添加如下的东西添加到你连接器选项中：

-Wl,--enable-runtime-pseudo-reloc

如果用了 codeblocks::ide 点击 project name->build options->linker options，在那设置一下。

光源种类

Ogre 提供了三种光源类型。

1. 点光源 (LT_POINT) - 从一个点发出光放射到四周。

2. 聚光源 (LT_SPOTLIGHT) - 严格来说聚光灯工作有点像手电筒。 你在某个地方设置了灯光的起点，然后灯光射向一个方向。你可以告诉光源内锥角度大小和外锥角度大小（你知道手电筒内部会明亮一些。）
3. 有向光 (LT_DIRECTIONAL) - 有向光模拟非常遥远地方的灯光，在场景中从某个方向照射物体。如果你想要一个夜晚的场景你就需要模拟月光。你可以通过为场景设置环境光来做到这个，但是这并非是真实场景的模拟因为月光并不平等地照亮所有物体（日光也不是）。一种方法解决这个是设置一个有向的光源然后指向一个方向，月光就会显示出来。

光源有一个范围属性描述光看上去的样子。两个更重要的属性是光的漫反射颜色和镜面反射颜色(diffuse and specular color)。每一个材质脚本定义有多少漫反射和镜面反射光线由材质反射来，稍后我们将学习如果控制她。

建立光源

在 Ogre 中创建一个光源需要调用场景管理器中的 createLight 方法然后提供光源的名称，就跟我们创建实体和摄像机一样，光源仅仅有两个方法 setPosition 和 setDirection（而且并不是像旋转，偏移，滚动一样有全套的移动函数）。如果你需要移动光源（例如创建一个光源跟随角色），你需要把光源绑定到场景节点上。

因此，让我们从基础的点光源开始吧。首先我们要创建光源设置她的类型和她的位置：

```
 Light* light = mSceneMgr->createLight("Light1");
light->setType(Light::LT_POINT);
light->setPosition(Vector3(0, 150, 250));
```

现在我们已经创建好了光源，我们可以设置她的漫射色和镜面色了。让我们设置为红色：

```
 light->setDiffuseColour(1.0, 0.0, 0.0);
light->setSpecularColour(1.0, 0.0, 0.0);
```

好的现在编译和运行这个应用程序。成功了！我们现在可以看见 Ninja 和她投射的影子了。确定你也可以从前面，侧面都可以看到她。有个要注意的事是你看不见光源。你看见产生的光事实上不是光照实体本身。许多 Ogre 指南里都添加了一个实体展示光是从哪发射出来的。如果你在应用程序中使用灯光时遇到了困难，你应该考虑创建这些东西这样你可以准确知道光源的位置。

接着让我们试试有向光。注意为什么 ninja 前面有一些倾斜的黑色？让我们添加一些黄色的有向光射向她的前面。设置光源和颜色的步骤和设置点光源是一样的：

```
 light = mSceneMgr->createLight("Light3");
light->setType(Light::LT_DIRECTIONAL);
light->setDiffuseColour(ColourValue(.25, .25, 0));
light->setSpecularColour(ColourValue(.25, .25, 0));
```

因为有向光是从无穷远的距离射来的，因此我们不需要设置她的位置，仅仅是设置她的方向。我们设置光的方向是 z 正半轴和 y 负半轴（类似从 ninja 的 45 度角正前方射过来）：

```
light->setDirection(Vector3( 0, -1, 1 ));
```

编译并运行这个应用程序。在场景中我们已经有两个影子了，因为有向光很微弱，因此影子也比较弱。最后让我们来试试聚光灯。我们将创建一个蓝聚光灯：

```
 light = mSceneMgr->createLight("Light2");
light->setType(Light::LT_SPOTLIGHT);
light->setDiffuseColour(0, 0, 1.0);
light->setSpecularColour(0, 0, 1.0);
```

我们仍然需要设置位置和方向来确定聚光源。我们将创建一个聚光源盘旋在 Ninja 的右侧，然后直射下来：

```
light->setDirection(-1, -1, 0);
light->setPosition(Vector3(300, 300, 0));
```

聚光源还允许我们指定光束的宽度。考虑一下手电筒的光束。中心的内光束比周围光束明亮一些。我们可以设置这两种宽度通过调用 setSpotlightRange 方法。

```
light->setSpotlightRange(Degree(35), Degree(50));
```

编译运行她，紫色的 Ninja……好恐怖！

尝试要做的事情

不同的阴影类型

这个 demo 里我们设置的阴影类型仅仅是 SHADOWTYPE_STENCIL_ADDITIVE（加成模板阴影）。尝试设置成其他两种阴影看看会发生什么。还有一些其他的与阴影相关的方法在场景管理器类中。试试他们然后看看你能得到什么。

光源衰减

当离光源越来越远的时候，光源定义了一个 setAttenuation 方法允许你控制定义光怎样渐弱。添加一个函数来调用点光源设置衰减为不同的值，看看会怎样影响光？

SceneManager::setAmbientLight

实验一下这个方法 setAmbientLight，在 mSceneMgr 中。

视口背景色

在 createViewports 方法中改变默认的 ColourValue 值。在这里改变颜色也许并不是合适的，但是可以让你了解一下如何改变她。

Camera::setFarClipDistance

在 createCamera 中我们设置的摄像机距离很近。添加一个函数调用 setFarClipDistance，设置参数为 500，分别设置模板阴影开启和关闭并移动摄像机来观察 Ninja。注意到速度变慢没？

注意：你需要设置 mSceneMgr->setShadowUseInfiniteFarPlane(false)，否则你将得到一些奇怪的影子（看看这里）

平面

在本指南中我们并没有涉及太多关于平面的内容（她不是本文的重点）。在后续章节中我们会重新提到她，但是现在你应该了解了解 createPlane 方法试着去用用这些函数。

完整来源

如果有困难，可以看看源代码
