文档:教程:基础教程:基础教程七

出自 Ogre3D 开放资源地带

跳转到: 导航, 搜索

目录

先决条件

本教程是在假设你已经拥有 c++编程基础并能够成功建立和编译 Ogre 程序（如果你设置程序方面还存在问题，请参阅 SettingUpAnApplication 来获取详细信息）。本教程建立在之前的初学者教程的基础上，并且假设你已经学习了它们。

介绍

在这一课里，我们将一起来探索如何在 Ogre 里使用 CEGUI（一个嵌入的 GUI 系统）。学完本课以后，你应该能够往你的 Ogre 应用程序里添加基本的 CEGUI 功能了。注意：本课并不会全面地教你如何使用 CEGUI，本课只是一个入门。若有更深入的问题需要帮助，请直接去它们的主页。

你能在这里找到本课的代码。在你学习本课时，你应该逐个地往你的工程里添加代码，编译后观察它的效果。

从这开始

最初的代码

在这节课里，你将用预定义的基础代码作为起点。只要你学过之前的课程，你应该对它们已经很熟悉了。用你喜欢的编译器创建一个工程，然后新建一个源文件包含如下代码：

```
#include "ExampleApplication.h"
#include <CEGUI/CEGUI.h>
#include <OIS/OIS.h>
#include <OgreCEGUIRenderer.h>

class TutorialListener : public ExampleFrameListener, public
OIS::MouseListener, public OIS::KeyListener
{
public:
   TutorialListener(RenderWindow* win, Camera* cam)
       : ExampleFrameListener(win, cam, true, true)
   {
      mContinue=true;
      mMouse->setEventCallback(this);
      mKeyboard->setEventCallback(this);
   } // CEGUIDemoListener

   bool frameStarted(const FrameEvent &evt)
   {
       mKeyboard->capture();
       mMouse->capture();

       return mContinue && !mKeyboard->isKeyDown(OIS::KC_ESCAPE);
   }

   bool quit(const CEGUI::EventArgs &e)
   {
       mContinue = false;
       return true;
   }

   // MouseListener
   bool mouseMoved(const OIS::MouseEvent &arg)
```

```
    {
        return true;
    }

    bool mousePressed(const OIS::MouseEvent &arg, OIS::MouseButtonID
id)
    {
        return true;
    }

    bool mouseReleased(const OIS::MouseEvent &arg, OIS::MouseButtonID
id)
    {
        return true;
    }

    // KeyListener
    bool keyPressed(const OIS::KeyEvent &arg)
    {
        return true;
    }

    bool keyReleased(const OIS::KeyEvent &arg)
    {
        return true;
    }

private:
    bool mContinue;
};


class CEGUIDemoApplication : public ExampleApplication
{
public:
    CEGUIDemoApplication()
        : mSystem(0), mRenderer(0)
    {
    }

    ~CEGUIDemoApplication()
    {
        if (mSystem)
            delete mSystem;
```

```
        if (mRenderer)
            delete mRenderer;
    }
protected:
  CEGUI::System *mSystem;
  CEGUI::OgreCEGUIRenderer *mRenderer;

    void createScene(void)
    {
    }

    void createFrameListener(void)
    {
        mFrameListener= new TutorialListener(mWindow, mCamera);
        mFrameListener->showDebugOverlay(true);
        mRoot->addFrameListener(mFrameListener);
    }

};

#if 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
    CEGUIDemoApplication app;

    try {
        app.go();
    } catch(Exception& e) {
#if OGRE_PLATFORM == OGRE_PLATFORM_WIN32
        MessageBoxA(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;
}
```

确保你能编译并运行这些代码。目前这个程序除了显示一个黑屏（按 ESC 退出）什么都不能做。如果你在编译时遇到连接器错误，请保证 CEGUIBase_d.lib、OgreGUIRenderer_d.lib 已经添加到了连接器的输入里（debug 模式的，如果是 release 模式，去掉_d 后缀）。

简单介绍

CEGUI 是一个功能全面的 GUI 库，它能够植入像 Ogre 这样的 3D 应用程序（当然也支持纯 DirectX 和 OpenGL）。就像 Ogre 只是一个图像库一样（不做声音、物理等其它的事情），CEGUI 只是一个 GUI 库，意味着既不自己做渲染，也不与任何鼠标键盘事件挂钩。实际上，为了让 CEGUI 能渲染，你必须提供一个渲染器给它（包含在 SDK 的 OgreGUIRenderer 库）。而为了让它能够理解鼠标键盘事件，你必须手工地把它们注入系统。这也许一开始看起来觉得痛苦，但其实只需要一丁点代码就能实现。这样，你就可以对渲染和输入进行全面控制，CEGUI 根本不会防碍你。

CEGUI 有很多的方面，有很多你不熟悉的诡异的地方（就算你以前使用过 GUI 系统）。在接下来的学习中，我将给你一一道来。

与 Ogre 结合

初始化 CEGUI

在前面的教程里，我们已经学习了如何启动 CEGUI，所以在这第一个部分我们不会深究。找到 createScene 函数并添加如下代码：

```
        mRenderer = new CEGUI::OgreCEGUIRenderer(mWindow,
Ogre::RENDER_QUEUE_OVERLAY, false, 3000, mSceneMgr);
        mSystem = new CEGUI::System(mRenderer);
```

好，CEGUI 已经初始化了，我们选择它将使用的皮肤。CEGUI 是高度自定义的，它允许你通过更换皮肤来定义应用程序的"look and feel"。我们将不涉及如何给这个库换皮肤，所以如果你想了解得更多，咨询 CEGUI 的网站。以下的代码选择了一个皮肤：

```
CEGUI::SchemeManager::getSingleton().loadScheme((CEGUI::utf8*)"Tahare
zLookSkin.scheme");
```

默认的 Ogre 安装不提供其它的皮肤给你，但如果你是从 CEGUI 网址上下载的，你就能找到几个其它的选择（或者你自己就可以制作）。接下来我们要做的事件就是设置默认鼠标指针图像和默认字体：

```
mSystem->setDefaultMouseCursor((CEGUI::utf8*)"TaharezLook",
(CEGUI::utf8*)"MouseArrow");
mSystem->setDefaultFont((CEGUI::utf8*)"BlueHighway-12");
```

在整个教程里，我们都会使用 CEGUI 来显示鼠标指针，就算我们没有使用这个 GUI 库的其它功能。用其它的 GUI 库来渲染鼠标，或者直接用 Ogre 来创建你自己的鼠标，也是可以的（后者要麻烦一点）。如果你只为了鼠标指针而使用 CEGUI，并担心内存的使用以及游戏所占的硬盘空间，你可以从这些选择中找一个来代替 CEGUI。

最后请注意，在最后一段代码里我们设置了默认的鼠标指针，但我们并没有像后面的教程那样直接使用 MouseCursor::setImage 函数来设置。这是因为在这一课里，我们总是在 CEGUI 窗口上面（虽然它可能是不可见的），而设置默认鼠标指针将会使得它是我们选择的图像。如果我们直接设置鼠标指针，而不设置默认的，则每当经过当 CEGUI 窗口时，鼠标指针都是不可见的（在本课里总是这种情况 ）。另一方面，如果你不显示任何 CEGUI 窗口，设置默认的鼠标指标将不产生任何效果，后面的教程就是这种情况。那样的话，调用 MouseCursor::setImage 才能在应用程序里显示指针。

注入键盘事件

CEGUI 不会自己处理输入，它不读取鼠标的移动和键盘的输入。相反地，它依赖用户把按键鼠标事件注入到系统。接下来我们要做的就是处理这些事件。若你正使用 CEGUI，你需要以缓冲模式运行鼠标键盘，这样你就能直接获取事件并在它们发生时注入。找到 keyPressed 函数，并添加代码：

```
CEGUI::System *sys = CEGUI::System::getSingletonPtr();
sys->injectKeyDown(arg.key);
sys->injectChar(arg.text);
```

得到系统对象之后，我们做两件事。第一件事是把 Key Down 事件注入到 CEGUI。第二件是注入一个实际按下的字符。正确地注入字符是非常重要的，因为如果使用的是非英语键盘，只凭注入 KeyDown 事件并不能总是取到想要的结果。记住 injectChar 是支持 Unicode 的。

现在我们给系统注入 KeyUp 事件。找到 keyReleased 函数，添加以下代码：

```
CEGUI::System::getSingleton().injectKeyUp(arg.key);
```

注意，我们这里不需要注入字符 up 事件，KeyUp 使用就可以了。

转变和注入鼠标事件

现在我们完成了键盘输入的处理，下面我们来看看鼠标输入。然而我们有个小问题需要说一下。当我们注入 KepUp 和 KepDown 事件到 CEGUI 里时，我们不必转换键码。OIS 和 CEGUI 为键盘输入使用相同的键码。但鼠标按钮却不是这样。在按下鼠标按钮注入到 CEGUI 时，我们需要写一个函数来把 OIS 的按钮 ID 转换为 CEGUI 的按钮 ID。在你代码顶部附近，TutorialListener 类前面，添加以下函数：

```
CEGUI::MouseButton convertButton(OIS::MouseButtonID buttonID)
{
    switch (buttonID)
    {
    case OIS::MB_Left:
        return CEGUI::LeftButton;

    case OIS::MB_Right:
        return CEGUI::RightButton;

    case OIS::MB_Middle:
        return CEGUI::MiddleButton;

    default:
        return CEGUI::LeftButton;
    }
}
```

现在我们准备注入鼠标事件。找到 mousePressed 函数，并添加以下代码：

```
CEGUI::System::getSingleton().injectMouseButtonDown(convertButton(id)
);
```

这代码的意思应该比较清楚了。我们转换传入的 ID，再把结果传给 CEGUI。找到 mouseReleased 函数，并添加这么一行：

```
CEGUI::System::getSingleton().injectMouseButtonUp(convertButton(id));
```

最后，我们往 CEGUI 里注入鼠标移动。The CEGUI::System 对象有一个 injectMouseMove 方法，它需要鼠标的相对运行作为参数。OIS::mouseMoved 处理者在 state.X.rel 和 the state.Y.rel 变量里给我们提供了这些相对运动。找到 mouseMoved 方法，并添加以下代码：

```
        CEGUI::System::getSingleton().injectMouseMove(arg.state.X.rel,
arg.state.Y.rel);
```

好了！CEGUI 设置完成了，可以接收鼠标和键盘事件了。

窗口、表单、组件

介绍

CEGUI 与其它多数 GUI 系统有所不同。在 CEGUI 里，所有能显示出来的东西都是 CEGUI::Window 类的一个子类，而且一个 window 可以有任意数量的子 window。这样会导致一些奇怪的事情发生。你能在一个按钮里面放置另外一个按钮，虽然实际上不会这么干。我提这些的原因是，当你正寻找放在应用程序里的某个特殊的小部件时，它们都被称作 Windows，而且可以通过访问 Windows 的函数来访问。

CEGUI 最常用的用法是，你不必在代码里创建每一个单独的对象。而你可以通过一个像 CEGUI Layout Editor 这样的编辑器，来为你的程序创建一个 GUI 布局。根据你的喜好，放置你的窗口、按钮以及其它部件到屏幕上之后，编辑器会把布局保存到一个文本文件里。你就可以之后加载这个布局到 GUI sheet 里面（它也是 CEGUI::Window 的一个子类）。

最后，要知道 CEGUI 包含大量的小部件供你的程序使用。我们在这里不去涉及，所以如果你决定使用 CEGUI，最好还是去它们的网站了解更多的信息。

载入表单

在 CEGUI 里载入一个表单(sheet)是非常容易的事。WindowManager 类提供了一个"loadWindowLayout"函数来加载表单，并把它放入 CEGUI::Window 对象。然后你可以通过 CEGUI::System::setGUISheet 来显示它。我们在本课不会使用它，但如果不给你介绍一下使用它的例子，我觉得我很失职。别把这些添加到代码里（如果添加了，看见结果后删除掉）：

```
        // Do not add this to the program
        CEGUI::Window* sheet =
CEGUI::WindowManager::getSingleton().loadWindowLayout((CEGUI::utf8*)"
ogregui.layout");
        mSystem->setGUISheet(sheet);
```

这样就把表单设置成显示了。你能在后面取回这个表单，用 System::getGUISheet 方法。通过调用 setGUISheet，你能够无缝地交换 GUI 表单（但你必须要保证拥有指向当前表单的指针，如果你打算再换回来）。

手动创建部件

就像我前面所说的，大多数你使用 CEGUI 的时候，你使用的是由编辑器生成的 GUI 表单。然而有时候，你需要手动地创建一个部件以呈现在屏幕上。在这个例子中，我们将添加一个 Quit 按钮，我们稍后会给它添加功能。因为当本课结束时，我们不只会添加一个 Quit 按钮到屏幕上，我们要创建一个默认的 CEGUI::Window 来包含所有我们要创建的部件。将以下代码添加到 createScene 函数末尾：

```
        CEGUI::WindowManager *win =
CEGUI::WindowManager::getSingletonPtr();
      CEGUI::Window *sheet = win->createWindow("DefaultGUISheet",
"CEGUIDemo/Sheet");
```

这里使用 WindowManager 创建了一个被称为"CEGUIDemo/Sheet"的 "DefaultGUISheet"。尽管我们可能为表单取任何名称，但用层次结构的方法来命名它是非常常见的（推荐），比如 “SomeApp/MainMenu/Submenu3/CancelButton”。然后我们要做的事情就是，创建一个 Quit 按钮，并设置它的大小：

```
      CEGUI::Window *quit = win->createWindow("TaharezLook/Button",
"CEGUIDemo/QuitButton");
      quit->setText("Quit");
      quit->setSize(CEGUI::UVector2(CEGUI::UDim(0.15, 0),
CEGUI::UDim(0.05, 0)));
```

这个的含义比较隐晦。CEGUI 里的大小和方位使用了一种“统一化的尺度”。当设置大小的时候，你必须创建一个 UDim 对象来告诉它到底是什么尺寸。第一个参数是这个对象相比于其父亲的相对大小。第二个参数才是对象的绝对大小（像素表示）。必须注意的是，你只能给 UDim 设置两种参数的一种，另一种必须是 0。所以在这里，我们创建了一个宽度为父亲的 15%且高度为 5%的按钮。如果我们想要把它设置成 20×5 像素，就要两个 UDim 的第二参数分别设置为 20 和 5。

我们要做的最后一件事情是，把 Quit 按钮粘合到我们创建的表单上，并把它设成当前系统的 GUI 表单：

```
      sheet->addChildWindow(quit);
      mSystem->setGUISheet(sheet);
```

现在如果你编译并运行了你的程序，你将会在屏幕左上角看到一个 Quit 按钮，但当你点击它时不会有任何反应。

事件

CEGUI 里的事件是非常灵活的。它不是实现一个接收事件的接口，而是使用一种回调机制，把任何公共函数绑定成一个事件处理器(通过适当的方法签名)。

不幸的是，这也意味着注册事件会比 Ogre 更麻烦。我们现在要为处理 Quit 按钮的点击事件进行注册。这样的话，我们首先需要一个指向 Quit 按钮的指针。找到 TutorialListener 的构造器，并添加以下代码：

```
	CEGUI::WindowManager *wmgr = 
CEGUI::WindowManager::getSingletonPtr();
	CEGUI::Window *quit = wmgr-
>getWindow((CEGUI::utf8*)"CEGUIDemo/QuitButton");
```

现在我们取得了指向按钮的指针，我们来订阅这个点击事件。CEGUI 里的所有部件都有它们所支持的事件集，而且都是以“Event”开头。这里是订阅事件的代码：

```
	quit->subscribeEvent(CEGUI::PushButton::EventClicked,
	CEGUI::Event::Subscriber(&TutorialListener::quit, this));
```

subscribeEvent 的第一参数是事件本身。第二个参数是一个 Event::Subscriber 对象。当创建 Subcriber 对象时，我们传入的第一个东西是指向处理事件函数的指针(注意&符号给出了函数的指针)。我们传经 subscriber 的第二个参数是处理这个事件的 TutorialListener 对象(就是“this”对象)。好了!我们的 TutorialListener::quit 函数(已经定义了)将会处理鼠标点击并终止程序。

编译并运行你的程序，调试一下看看。

注意，我们可以创建任意数量的函数来为 CEGUI 处理事件。对它们的唯一约束是，返回值必须是一个布尔型，而且必须带有类型为"const CEGUI::EventArgs &"的唯一参数。关于事件的更多信息(以及如何退订事件)，请查阅 CEGUI 的网站。

渲染到纹理

我们能用 CEGUI 来做更有趣的事，其中之一就是创建一个纹理窗口的渲染器。这让我们能够创建一个可以直接渲染到 CEGUI 部件的视口。这样的话，我们首先要创建一个让我们看见的场景。在 createScene 函数底部添加如下代码：

```
	mSceneMgr->setAmbientLight(ColourValue(1, 1, 1));
	mSceneMgr->setSkyDome(true, "Examples/CloudySky", 5, 8);
	Entity* ogreHead = mSceneMgr->createEntity("Head", 
"ogrehead.mesh");
	SceneNode* headNode = mSceneMgr->getRootSceneNode()-
>createChildSceneNode(Vector3(0, 0, -300));
	headNode->attachObject(ogreHead);
```

现在我们创建好了一个渲染纹理。RenderSystem 对象提供了一种渲染到纹理的功能。我们用 RenderSystem::createRenderTexture 函数创建一个纹理。在这个程序里，我们创建一个 512 x 512 的纹理：

```
    RenderTexture *tex = mRoot->getRenderSystem()-
>createRenderTexture("RttTex", 512, 512, TEX_TYPE_2D, PF_R8G8B8);
```

UPDATE: The createRenderTexture() method appears to have been deprecated in the trunk. So instead of the above line use this:

```
    RenderTexture *tex= mRoot->getTextureManager()->createManual
                                ("RttTex", "Default", TEX_TYPE_2D,
512, 512, 0, PF_R8G8B8, TU_RENDERTARGET)
                                 ->getBuffer()->getRenderTarget();
```

关于这个函数，请查阅 API 参考来获取更多的信息。接下来我们还要创建一个摄像机和一个视口，以便察看我们创建的场景。注意，我们改变了一些视口设置，包括关闭 Overlays。这是非常重要的，否则你会看见 CEGUI 与 Ogre 在小窗口里面重叠在一起。

```
    Camera *cam = mSceneMgr->createCamera("RttCam");
    cam->setPosition(100, -100, -400);
    cam->lookAt(0, 0, -300);
    Viewport *v = tex->addViewport(cam);
    v->setOverlaysEnabled(false);
    v->setClearEveryFrame(true);
    v->setBackgroundColour(ColourValue::Black);
```

注意，我们把视口添加给了纹理（而我们通常是把视口添加到渲染窗口）。好了，我们已经创建了我们的场景和纹理，现在我们需要把它们植入 CEGUI。你能用任何 Ogre 纹理来创建 CEGUI::Texture，通过调用 OgreCEGUIRenderer::createTexture 函数：

```
    CEGUI::Texture *cTex = mRenderer-
>createTexture((CEGUI::utf8*)"RttTex");
```

不幸的是，这里比较复杂。在 CEGUI 里，你从来不处理单个的纹理或单个的图像。CEGUI 处理的是图像集，而非单个的图像。当你试着定义皮肤的 look and feel 时，处理一整套图像是很有用的（比如，看一看 SDK 里 media 目录 TaharezLook.tga 的图像是什么样子）。然后，就算你只要定义单个图像，你必须为它创建一个图像集。下面我们就是做这个事情：

```
    CEGUI::Imageset *imageSet =
CEGUI::ImagesetManager::getSingleton().createImageset((CEGUI::utf8*)"
RttImageset", cTex);
```

```
imageSet->defineImage((CEGUI::utf8*)"RttImage",
    CEGUI::Point(0.0f, 0.0f),
    CEGUI::Size(cTex->getWidth(), cTex->getHeight()),
    CEGUI::Point(0.0f,0.0f));
```

第一行用我们提供给它的纹理，创建了一个图像集（称为"RttImageset"）。第二行（调用 defineImage），指定了第一个也是唯一一个图像，它被称为"RttImage"，而且它的大小与我们提供的 cTex 纹理大小一样。最后我们还要创建一个 StaticImage 小部件，来收容这个渲染纹理。第一部分与创建其它窗口没有什么不同：

```
CEGUI::Window *si = win-
>createWindow((CEGUI::utf8*)"TaharezLook/StaticImage", "RTTWindow");
si->setSize(CEGUI::UVector2(CEGUI::UDim(0.5f, 0),
CEGUI::UDim(0.4f, 0)));
si->setPosition(CEGUI::UVector2(CEGUI::UDim(0.5f, 0),
CEGUI::UDim(0, 0)));
```

现在我们要指定 StaticImage 部件将要显示的图像。再一次，由于 CEGUI 总是处理图像集而不是单独的图像，我们必须从图像集里得到图像名称，并显示它：

```
si->setProperty("Image",
CEGUI::PropertyHelper::imageToString(&imageSet-
>getImage((CEGUI::utf8*)"RttImage")));
```

这似乎是我们把一个纹理包装进了图像集，然后再解包出来，我们实际上就是这么做的。在 CEGUI 里操作图像并不是这个库中最简单最直接的事件。最后，我们把 StaticImage 部件添加到先前创建的 GUI 表单里去：

```
sheet->addChildWindow(si);
```

现在我们完成了。编译并运行这个应用程序。

结论

其它的选择

CEGUI 并不是完美无缺的，它确定不是适用于每一个人。除了 CEGUI，这里还有一些其它的选择：

Navi Right Brain Games GUI QuickGUI BetaGUI

更多的信息

你还能够在其它的地方找到关于 CEGUI 的更多信息。

Practical Application - Something With A Bit More Meat - A more in-depth tutorial than the one here. CEGUI's Official Tutorials The CEGUI Website

--Xiao7cn 11:13 2008 年 3 月 1 日 (CST)Xiao7cn(FriedChicken)翻译

原文
上一章节：基础教程六
下一章节：基础教程八
目录

查看

- 页面
- 讨论
- 源码
- 历史

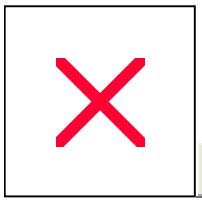
个人工具

- 登录 / 创建账户

导航

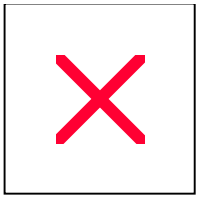
- 首页
- 社区
- 当前事件
- 最近更改
- 随机页面
- 帮助

搜索



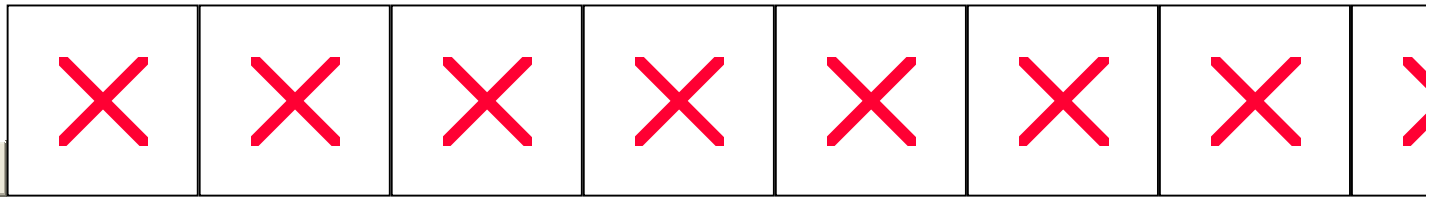
进入 搜索

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

- 这页的最后修订在 2009 年 4 月 3 日（星期五）09:27。
- 本页面已经被浏览 1,227 次。
- 隐私政策
- 关于 Ogre3D 开放资源地带
- 沪 ICP 备 09049564 号