

# 文档: 教程: 基础教程: 基础教程五

出自 OGRE3D 开放资源地带

跳转到: [导航](#), [搜索](#)

## 目录

- [1 先决条件](#)
- [2 介绍](#)
- [3 从这开始](#)
- [4 简单的缓冲输入](#)
  - [4.1 介绍](#)
  - [4.2 键盘监听界面](#)
  - [4.3 鼠标监听界面](#)
- [5 代码](#)
  - [5.1 TutorialFrameListener 的构造函数](#)
  - [5.2 变量](#)
  - [5.3 TutorialFrameListener 构造函数](#)
  - [5.4 键盘绑定](#)
  - [5.5 鼠标绑定](#)
- [6 其他输入系统](#)

## 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。

## 介绍

在这一课里，你将学会 OIS 的带缓冲的输入，这不同于上节课中无缓冲的输入。而且在这节课中，当鼠标键盘的事件发生时，我们会立即处理它，不会像上节课那样每一帧只处理一次。请注意这里只是对缓冲输入的一个简单介绍，而不是完整的如何使用 OIS 的教程。若想了解更多内容，请查阅相关的 OIS 使用教程。

你可以找到这节课的源代码。在你学习本课的时候，你应该逐个地将这些代码添加到你自己的工程里去，然后构建并观察结果。

## 从这开始

本课是基于上一课的，但我们将改变输入的方式。既然功能基本上是相同的，我们就使用上次的 TutorialApplication 类，但我们会从 TutorialFrameListener 开始。用你喜欢的编译器创建一个工程，再添加一个包含如下代码的源文件：

```
#include "ExampleApplication.h"

class TutorialFrameListener : public ExampleFrameListener, public
OIS::MouseListener, public OIS::KeyListener
{
public:
    TutorialFrameListener(RenderWindow* win, Camera* cam,
SceneManger *sceneMgr)
        : ExampleFrameListener(win, cam, true, true)
    {
    }

    bool frameStarted(const FrameEvent &evt)
    {
        if(mMouse)
            mMouse->capture();
        if(mKeyboard)
            mKeyboard->capture();
        return mContinue;
    }

    // MouseListener
    bool mouseMoved(const OIS::MouseEvent &e) { return true; }
    bool mousePressed(const OIS::MouseEvent &e, OIS::MouseButtonID id)
{ return true; }
    bool mouseReleased(const OIS::MouseEvent &e, OIS::MouseButtonID
id) { return true; }

    // KeyListener
    bool keyPressed(const OIS::KeyEvent &e) { return true; }
    bool keyReleased(const OIS::KeyEvent &e) { return true; }
protected:
    Real mRotate;          // The rotate constant
    Real mMove;            // The movement constant

    SceneManger *mSceneMgr; // The current SceneManger
    SceneNode *mCamNode;    // The SceneNode the camera is currently
attached to
}
```

```

        bool mContinue;           // Whether to continue rendering or not
        Vector3 mDirection;       // Value to move in the correct direction
    };

class TutorialApplication : public ExampleApplication
{
public:
    void createCamera(void)
    {
        // create camera, but leave at default position
        mCamera = mSceneMgr->createCamera("PlayerCam");
        mCamera->setNearClipDistance(5);
    }

    void createScene(void)
    {
        mSceneMgr->setAmbientLight(ColourValue(0.25, 0.25, 0.25));

        // add the ninja
        Entity *ent = mSceneMgr->createEntity("Ninja", "ninja.mesh");
        SceneNode *node = mSceneMgr->getRootSceneNode()-
>createChildSceneNode("NinjaNode");
        node->attachObject(ent);

        // create the light
        Light *light = mSceneMgr->createLight("Light1");
        light->setType(Light::LT_POINT);
        light->setPosition(Vector3(250, 150, 250));
        light->setDiffuseColour(ColourValue::White);
        light->setSpecularColour(ColourValue::White);

        // Create the scene node
        node = mSceneMgr->getRootSceneNode()-
>createChildSceneNode("CamNode1", Vector3(-400, 200, 400));

        // Make it look towards the ninja
        node->yaw(Degree(-45));

        // Create the pitch node
        node = node->createChildSceneNode("PitchNode1");
        node->attachObject(mCamera);

        // create the second camera node/pitch node

```

```

        node = mSceneMgr->getRootSceneNode()-
>createChildSceneNode("CamNode2", Vector3(0, 200, 400));
        node = node->createChildSceneNode("PitchNode2");
    }

    void createFrameListener(void)
    {
        // Create the FrameListener
        mFrameListener = new TutorialFrameListener(mWindow, mCamera,
mSceneMgr);
        mRoot->addFrameListener(mFrameListener);

        // Show the frame stats overlay
        mFrameListener->showDebugOverlay(true);
    }
};

#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。不要试着去运行程序，因为我们还没有定义键盘的行为。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](#)。

注意：如果你没有把你的工程设置成使用 Ansi C++，并收到关于 MessageBox 的错误信息，你可能需要把对 MessageBox 的引用改成 MessageBoxA。

程序的控制与上一课的相同。

## 简单的缓冲输入

### 介绍

在上一次课里，我们使用的是无缓冲的输入，也就是说，在每一帧里我们查询 OIS::Keyboard 和 OIS::Mouse 实例的状态，以判断它们是否被按下。而缓冲输入使用了一个 listener 接口，以便在事件发生时通知你的程序。比如，当一个键被按下时，会触发一个 KeyListener::keyPressed 事件，而当这个键被释放（不再按下）时，KeyListener::keyReleased 事件被触发给所有已注册的 KeyListener 类。这些能用在追踪按键的时间，或判断按键在上一帧中是否没有被按下。

通过 OIS::JoystickListener 接口，OIS 也支持无缓冲的操纵杆事件，但在本课我们不会涉及到。

关于 OIS 的监听系统有一点要注意的是，对于每一个 Keyboard, Mouse, Joystick 对象只能有一个监听器。这样是为了简单（也为了速度）。多次调用 setEventCallback 函数（后面会讲到）的结果是只有最后一次注册的监听器才得到事件消息。如果你有多个对象需要获得 Key, Mouse, 或 Joystick 事件，你只有自己写一个消息分发。还有，千万记得在 frameStarted 方法里调用 Keyboard::capture 和 Mouse::capture。OIS 不会使用线程（或其它玩意儿）来确定键盘鼠标状态，所以你必须指明什么时候去获取输入。

### 键盘监听界面

OIS 的 KeyListener 接口提供了两个纯虚函数。第一个是 keyPressed 函数（每次按下某个键时调用它），还有一个是 keyReleased（每次离开某个键时调用它）。传入这些函数的参数是一个 KeyEvent，它包含被按下/释放的按键的键码。

## 鼠标监听界面

MouseListener 接口比 KeyListener 接口要稍微复杂一些。它包含查看何时鼠标键被按下/释放的函数： `MouseListener::mousePressed` 和 `MouseListener::mouseReleased`。它还包含一个 `mouseMoved` 函数，当鼠标移动时调用它。这些函数都接收一个 `MouseEvent` 对象，在 "state" 变量里保存着当前鼠标的状态。

需要注意的是，`MouseState` 对象即包含了鼠标移动的相对 XY 坐标（即，从上一次调用 `MouseListener::mouseMoved` 开始，它所移动的距离），还包含了绝对 XY 坐标（即，屏幕上的准确位置）。

## 代码

### TutorialFrameListener 的构造函数

在我们开始修改 `TutorialFrameListener` 之前，请注意我们对 `TutorialFrameListener` 类做了两处大的改变：

```
class TutorialFrameListener : public ExampleFrameListener, public
OIS::MouseListener, public OIS::KeyListener 我们继承了 OIS 的
MouseListener 和 KeyListener 类，这样我们才能从它们那里接收事件。注
意，OIS 的 MouseListener 处理鼠标按钮事件，也处理鼠标移动事件。
```

同样，`ExampleFrameListener` 的构造函数也有变化：

```
    : ExampleFrameListener(win, cam, true, true)
```

这个 "true, true" 参数指明了我们将要使用带缓冲的键盘鼠标输入。我们接下来会深入介绍如何手动地设置 OIS，以及 OGRE 的其它部分。

## 变量

上一课的变量在这里有些改变。我移除了 `mToggle` 和 `mMouseDown` 变量（再也用不到了）。我还添加了一些其它的：

```
Real mRotate;           // 旋转常量
Real mMove;             // 运动常量
SceneManager *mSceneMgr; // 当前的场景管理器
SceneNode *mCamNode;    // 当前摄像机附着的场景节点

bool mContinue;         // 是否要继续渲染
Vector3 mDirection;     // 指向正确的移动方向
```

mRotate, mMove, mSceneMgr, 以及 mCamNode 与上一课的相同（虽然我们会改变 mRotate 的值，因为它有不同的用处）。mContinue 变量是 frameStarted 方法的返回值。当 mContinue 为 false 的时候，程序退出。mDirection 变量指定了在每一个帧里我们如何移动摄像机节点。

## TutorialFrameListener 构造函数

在构造函数里，我们像在上一课那样初始化一些变量，并把 mContinue 设成 true。添加如下代码到 TutorialFrameListener 的构造函数里：

```
// Populate the camera and scene manager containers
mCamNode = cam->getParentSceneNode();
mSceneMgr = sceneMgr;

// 设置旋转和移动速度
mRotate = 0.13;
mMove = 250;

// 继续渲染
mContinue = true;
```

在 ExampleFrameListener 的构造函数里已经取得了 OIS 的 mMouse 和 mKeyboard 对象。我们调用这些输入对象的 setEventCallback 方法，把 TutorialFrameListener 注册成一个监听器。

```
mMouse->setEventCallback(this);
mKeyboard->setEventCallback(this);
```

最后，我们还要把 mDirection 初始化成零向量（因为我们最开始不需要它动）：

```
mDirection = Vector3::ZERO;
```

## 键盘绑定

在我们深入之前，我们应该设置 Escape 键用来退出程序。找到 TutorialFrameListener::keyPressed 方法，每当键盘上一个键被按下时，都会调用这个方法并传入一个 KeyEvent 对象。我们能够通过这个对象的 "key" 变量来获取按键的键码(KC\_\*)。基于这个值，我们构造一个 switch，为绑定所有程序里用到的按钮。

```
bool keyPressed(const OIS::KeyEvent &e)
{
    switch (e.key)
    {
```

```

        case OIS::KC_ESCAPE:
            mContinue = false;
            break;
    }
    return true;
}

```

我们继续之前请保证以上都能编译和运行。

我们需要在 `switch` 语句里为其它按钮做绑定。首先我们要让用户按 1、2 键进行视口的切换。

代码基本上与上节课相同（需要包括进 `switch` 语句），除了不需要处理 `mToggle` 变量：

```

    case OIS::KC_1:
        mCamera->getParentSceneNode()->detachObject(mCamera);
        mCamNode = mSceneMgr->getSceneNode("CamNode1");
        mCamNode->attachObject(mCamera);
        break;

    case OIS::KC_2:
        mCamera->getParentSceneNode()->detachObject(mCamera);
        mCamNode = mSceneMgr->getSceneNode("CamNode2");
        mCamNode->attachObject(mCamera);
        break;

```

瞧！这比用临时变量来保存按键时间要干净多了。

接下来我们要添加键盘移动。每次用户按下移动按键，我们都要朝正确的方向加上或者减去 `mMove`：

```

    case OIS::KC_UP:
    case OIS::KC_W:
        mDirection.z -= mMove;
        break;

    case OIS::KC_DOWN:
    case OIS::KC_S:
        mDirection.z += mMove;
        break;

    case OIS::KC_LEFT:
    case OIS::KC_A:
        mDirection.x -= mMove;

```



```

        break;

case OIS::KC_RIGHT:
case OIS::KC_D:
    mDirection.x += mMove;
    break;

case OIS::KC_PGDOWN:
case OIS::KC_E:
    mDirection.y -= mMove;
    break;

case OIS::KC_PGUP:
case OIS::KC_Q:
    mDirection.y += mMove;
    break;

```

当按键被释放时，我们要立即取消 `mDirection` 向量上的移动。找到 `keyReleased` 方法，添加如下代码：

```

switch (e.key)
{
case OIS::KC_UP:
case OIS::KC_W:
    mDirection.z += mMove;
    break;

case OIS::KC_DOWN:
case OIS::KC_S:
    mDirection.z -= mMove;
    break;

case OIS::KC_LEFT:
case OIS::KC_A:
    mDirection.x += mMove;
    break;

case OIS::KC_RIGHT:
case OIS::KC_D:
    mDirection.x -= mMove;
    break;

case OIS::KC_PGDOWN:
case OIS::KC_E:
    mDirection.y += mMove;

```

```

        break;

    case OIS::KC_PGUP:
    case OIS::KC_Q:
        mDirection.y -= mMove;
        break;
} // switch
return true;

```

好了，我们能根据按键输入对 `mDirection` 进行更新了，我们还需要让它真正移动起来。下面的代码与上节课是一样的，添加到 `frameStarted` 函数里：

```

    mCamNode->translate(mDirection * evt.timeSinceLastFrame,
Node::TS_LOCAL);

```

编译并运行程序。我们现在已经有了带缓冲的输入来控制运动！

## 鼠标绑定

现在我们已经完成了键盘绑定，接下来轮到鼠标了。我们从点击鼠标左键来控制灯的开关开始。找到 `mousePressed` 函数并看看它的参数。用 `OIS`，我们可以访问 `MouseEvent` 和 `MouseButtonID`。我们用 `MouseButtonID` 作为 `switch` 条件，来确定按下的是哪个按钮。用下面的代码替换掉 `mousePressed` 函数里的：

```

    Light *light = mSceneMgr->getLight("Light1");
    switch (id)
    {
    case OIS::MB_Left:
        light->setVisible(! light->isVisible());
        break;
    }
    return true;

```

编译并运行。噢耶！成功了，剩下的事情就是绑定鼠标右键来进入鼠标观察模式。每当鼠标移动时我们都检查右键是否按下。如果是，我们基于相对运动来转动摄像机。通过传入函数的 `MouseEvent` 对象，我们能获取相对运动。它包含一个 `"state"` 变量，里面有鼠标的状态（是关于鼠标的详细信息）。

`MouseState::buttonDown` 告诉我们是否一个特定的按钮被按下，而 `"X"` 和 `"Y"` 变量告诉我们鼠标的相对运动。找到 `mouseMoved` 方法，用以下代码替换掉原来的：

```

    if (e.state.buttonDown(OIS::MB_Right))
    {
        mCamNode->yaw(Degree(-mRotate * e.state.X.rel),
Node::TS_WORLD);

```

```

        mCamNode->pitch(Degree(-mRotate * e.state.Y.rel),
Node::TS_LOCAL);
    }
    return true;

```

编译并运行程序。当鼠标右键被按下时，摄像机就以自由视角的模式工作。

## 其他输入系统

OIS 是非常棒的，应该满足你的应用程序的大部分需求。话说回来，你也有其它的选择。有的窗口系统像 `wxWidgets` 也许正是你想要的，人们已经成功地把它整合进了 `Ogre`。如果你不介意你的应用程序是特定平台的，你也可以使用标准的 Windows 消息系统，或者从 Linux GUI 工具集中选择一个。

你也可以试一试 SDL，它是一个跨平台的带窗口输入系统，并支持摇杆/手柄输入。我不能指导你用 `Ogre` 如何设置 `wx/gtk/qt` 这些（因为我自己也没搞过-\_-），但我有许多将 SDL 的摇杆/手柄输入整合进 `Ogre` 的成功经验。为了使 SDL 的操纵杆系统生效，通过调用 `SDL_Init` 和 `SDL_Quit`，把你的应用程序包裹起来（放在你的程序 `main` 函数里，或放在你的应用对象里）：

```

SDL_Init(SDL_INIT_JOYSTICK | SDL_INIT_NOPARACHUTE);
SDL_JoystickEventState(SDL_ENABLE);

app.go();

SDL_Quit();

```

为了设置 Joystick，调用 `SDL_JoystickOpen` 并传入操纵杆编号（你能通过 0、1、2... 来指定多个操纵杆）：

```

SDL_Joystick* mJoystick;
mJoystick = SDL_JoystickOpen(0);

if ( mJoystick == NULL )
    ; // error handling

```

如果 `JoystickOpen` 返回 `NULL`，表示打开 joystick 有问题。这基本上意味着你请求的摇杆不存在。使用 `SDL_NumJoysticks` 来确定连接在系统上的摇杆数量。当你结束之后，需要关闭摇杆：

```

SDL_JoystickClose(mJoystick);

```

为了使用摇杆，调用 `JoystickGetButton` 和 `JoystickGetAxis`。我个人使用的是 PS2 的手柄，所以我有四个轴向和 12 个按钮来玩。这是我的操纵代码：

```
SDL_JoystickUpdate();
```

```
    mTrans.z += evt.timeSinceLastFrame * mMoveAmount *  
    SDL_JoystickGetAxis(mJoystick, 1) / 32767;
```

```
    mTrans.x += evt.timeSinceLastFrame * mMoveAmount *  
    SDL_JoystickGetAxis(mJoystick, 0) / 32767;
```

```
    xRot -= evt.timeSinceLastFrame * mRotAmount *  
    SDL_JoystickGetAxis(mJoystick, 3) / 32767;
```

```
    yRot -= evt.timeSinceLastFrame * mRotAmount *  
    SDL_JoystickGetAxis(mJoystick, 2) / 32767;
```

mTrans 在后面要传入到摄像机的 SceneNode::translate 方法，xRot 传入到 SceneNode::yaw，而 yRot 传入到 SceneNode::pitch。注意，SDL\_JoystickGetAxis 返回一个从-32767 到 32767 的值，所以我得把它缩小到-1 到 1 的范围。这应该可以让你开始使用 SDL 摇杆输入了。若你想查看更多关于这个领域的信息，SDL joystick 的头文件是最好的文档。

如果你想要认真地在你的程序时使用 SDL，你应该查阅标准 SDL 文档。

---

[原文](#)

[上一章节：基础教程四](#)

[下一章节：基础教程六](#)

[目录](#)

取自

["http://ogre3d.cn/wiki/index.php?title=%E6%96%87%E6%A1%A3:%E6%95%99%E7%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%BA%94"](http://ogre3d.cn/wiki/index.php?title=%E6%96%87%E6%A1%A3:%E6%95%99%E7%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%BA%94)

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](#)

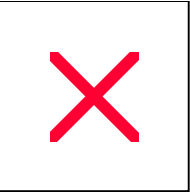
搜索



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 5 月 14 日（星期四）07:52。
- 本页面已经被浏览 1,344 次。
- [隐私政策](#)
- [关于 Ogre3D 开放资源地带](#)
- [沪 ICP 备 09049564 号](#)

