

Integrating OGRE with wxWidgets

Introduction

While OGRE does offer an integrated cross-platform GUI subsystem, there are cases where something more comprehensive and mature is desirable. This tutorial describes how to embed OGRE into a wxWidgets application in Windows with Visual Studio. Here, wxWidgets was chosen for its maturity, simplicity and liberal licensing model. A reader seeking to integrate OGRE with some other GUI framework (possibly on some other platform) may still be able to glean some useful information from this document, as a similar approach could prove effective under most other frameworks.

The code presented here currently is only known to work with DirectX 9 on Windows using Microsoft Visual Studio .NET 7.1, using OGRE 0.14 and wxWidgets 2.4.2. The window handle mechanism of Windows is currently the only well-supported means of linking OGRE with external GUI frameworks. Should other platforms come to be supported, this tutorial could be extended to full cross-platform functionality. As wxWidgets is cross-platform, the changes would be minor. This tutorial assumes some familiarity with the basics of both OGRE and wxWidgets. There is an abundance of introductory-level documentation available for beginners.

High-Level Overview

All the classes defined by the code can be categorized as belonging either with OGRE, or with wxWidgets. The tutorial will skip between the two categories as necessary, but the provided example code is organized in such a way as to make the distinction as explicit as possible. The four source files can be arranged as follows:

	wxWidgets	OGRE
Code	wxTest.cpp	OgreApp.cpp
Header	wxTest.h	OgreApp.h

In a full-scale application, more files can be expected in each of the categories, but the distinction would remain clear. The following UML diagram illustrates the relationships between the three classes defined by the code:

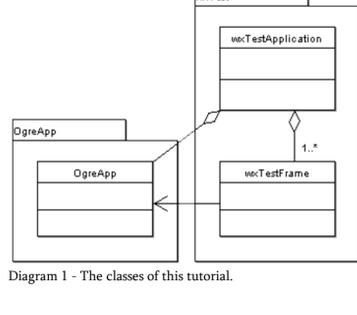


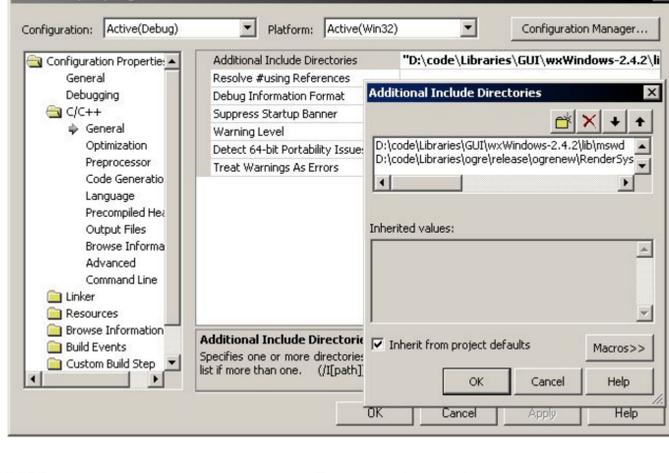
Diagram 1 - The classes of this tutorial.

The stylistic principle that one header/code file pair should correspond to one class declaration/definition was violated deliberately here to preserve symmetry. In practice, the application and frame classes would appear in files of their own. It was necessitated by the domineering GUI design paradigm to subordinate the OGRE application class to the wxWidgets one.

The overall structure of the application was inspired by the wxWidgets 'Minimal' sample, as well as the OGRE 'ExampleApplication' (sans the 'ExampleFrameListener' – for obvious reasons).

Configuration

The Visual Studio solution file (wxTest.sln) provided assumes that the wxWidgets, OGRE and DirectX library and include directories are correctly configured in your Visual Studio installation. The solution also defines some additional include directories, which must be set in the property pages of the project, thus:



NB: wxWidgets and OGRE both have custom memory managers. The two collide, causing memory leaks and mysterious crashes. After some experimentation and investigation, no satisfactory solution with both enabled was found. Therefore, OGRE's memory manager must be disabled for the example code's Debug builds to work. To do this, edit /ogrenew/OgreMain/include/OgreConfig.h and set the preprocessor directive named OGRE_DEBUG_MEMORY_MANAGER to 0. The OGRE libraries must be rebuilt whenever OgreConfig.h has been changed.

Like all OGRE applications, the demo depends upon the OGRE DLL's. These must be copied into the /Debug and /Release subdirectories (or your Windows System directory) before the Demo will run.

The example imports the OGRE configuration parameters blindly from ogre.cfg, therefore it is liable to get very confused and consequently to die if the parameters are not suitable for the hardware on which the example is being run. It is recommended you edit ogre.cfg before attempting to execute the demo. In particular, the Rendering Device should be changed to whichever is your DirectX 9 primary rendering device (unless, of course, it happens to be a RADEON 9600 SERIES). An easy way to achieve this is to copy over the appropriate line of the configuration file generated by some other OGRE application which uses the OGRE configuration dialog. The OGRE samples work nicely.

Instructions for using the Demo

The view can be controlled by dragging the mouse cursor in the window while holding down some combination of the left and right mouse buttons. The effects of the combinations are as follows:

- Left Only – Looks around.
- Right Only – Pans the view.
- Left and Right – Zooms in and out.

The wxTestApplication class

The wxTestApplication class declaration appears in wxTest.h:

```

class wxTestApplication : public wxApp
{
private:
    OgreApp *mOgreApp;
    wxTestFrame *mFrame;

public:
    virtual bool OnInit();
    virtual int OnExit();
};
  
```

It is just a typical example of a wxWidgets application class. Note that it has wxTestFrame and OgreApp fields. The interaction between these is where the actual embedding is done. The OnInit() method is defined as follows:

```

bool wxTestApplication::OnInit()
{
    // Make the frame
    mFrame = new wxTestFrame(T("wxWidgets/OGRE Integration Demo"), wxPoint(100, 100), wxSize(640, 480));

    // Show it as the top window
    mFrame->Show(TRUE);
    SetTopWindow(mFrame);

    // Make an OGRE application object.
    mOgreApp = new OgreApp();

    // Initialize with the window handle.
    HWND h = (HWND)(mFrame->GetHandle());
    mOgreApp->Init(h);

    // Tell the frame about its OGRE app.
    mFrame->SetOgreApp(mOgreApp);

    // Update!
    mOgreApp->Update();

    // All clear!
    return TRUE;
}
  
```

A frame is created and set to display as the top-level window. Next, an OgreApp object is created, and initialized with the (platform dependent) window handle of the frame. Once all the initialization steps are done, the first frame is rendered through a call to OgreApp::Update(). The OnExit() method simply deletes the OgreApp object.

The OgreApp class

The Init() method of the OgreApp class is where the window handle is actually used to embed the OGRE D3D9RenderSystem into the wxTestFrame. This method will be dissected step-by-step:

```

void OgreApp::Init(HWND handle)
{
    // Make the root
    mRoot = new Root();

    try
    {
        // Set parameters of render system (window size, etc.)
        mRoot->restoreConfig();
    }
}
  
```

The parameters for the render system must be configured with care. In this example, an ogre.cfg file was prepared (see the section on Configuration) and "restored". An alternative to force the relevant parameters through more method calls.

```

// Render system.
mRSys = mRoot->getRenderSystem();
D3D9RenderSystem *rsys = (D3D9RenderSystem *)mRSys;
rsys->SetExternalWindowHandle(handle);
  
```

The OGRE D3D9RenderSystem class provides a method for setting a window handle. It is vital that this method get called only once the configuration parameters have been set (this was done in the previous step), and before calling initialise() on the Root object.

```

// Root and Scene.
mWnd = mRoot->initialise(true);
mScene = mRoot->getSceneManager(ST_GENERIC);
  
```

Now the Root is initialised and a SceneManager obtained.

```

// Create the OGRE scene as normal (lights, resources, skybox, entities, etc.)
.
.
.
  
```

Now we can create the scene as normal. Here we create the lights, skybox, environment, entities, etc. This example just defines a skybox with a Ninja walking in place, floating in thin air.

```

// All set up, activate.
mWnd->setActive(true);

// Ready now.
mReady = true;
  
```

Finally, the window is set to active and a flag is set to true, indicating that this OgreApp is ready to start receiving messages.

```

}
catch(Ogre::Exception &e)
{
    String s = "OgreApp::Init() - Exception:\n" + e.getFullDescription() + "\n";
    LogManager::getSingleton().logMessage(s, LML_CRITICAL);
    wxMessageBox(e.getFullDescription().c_str(), "Exception!", wxICON_EXCLAMATION);

    // Clean up.
    if(mRoot)
        delete mRoot;
}
  
```

This code just attempts to handle any possible exception gracefully, by writing to the log and showing a message box, then cleaning whatever might be left of OGRE by deleting the Root, if there is any.

Since the continuous OGRE rendering loop is being suppressed in favour of an event-driven model as is customary with GUI applications, we must define a method for rendering a single frame, to be called whenever necessary. That is the function of the Update() method, shown here:

```

void OgreApp::Update()
{
    if(mReady)
    {
        mRoot->renderOneFrame();
    }
}
  
```

In a full-scale application, should one have more than one render target to manage, the set of targets that renderOneFrame() updates can be controlled through setAutoUpdate() calls. Alternatively, one could exercise more direct control through calls to RenderWindow::update(), although renderOneFrame() may be preferable for its ability to prioritize render targets.

In cases where continuous animation without user intervention is required, such as the walking of the ninja in the demonstration, simply register a timer event to be called periodically. The Tick() method here merely updates the ninja's animation:

```

void OgreApp::Tick(Real t)
{
    // Tell the Ninja.
    mScene->getEntity("Ninja Boy")->getAnimationState("Walk")->addTime(t);
}
  
```

The wxTestFrame class

Most of the methods of this class deal with responses to events. The OgreApp::Update() method is often called at the end of responses to these events. In general, the OgreApp does not update() itself, the wxTestFrame tells it to update() at the end of a response to an event. This is to prevent cases where the display may be updated several times, which would be detrimental to performance.

The only non-trivial event handler in the demonstration code is OnMouse(). The view responds to the user dragging with the mouse while holding down certain combinations of mouse buttons. It is shown here:

```

void wxTestFrame::OnMouse(wxMouseEvent &e)
{
    // Camera controls
    if(e.Dragging() && mOgreAppPtr)
    {
        // Compute deltas
        long delta_x = e.m_x - mMouseX,
        delta_y = e.m_y - mMouseY;

        // Dolly, orient or pan?
        if(e.m_leftDown && e.m_rightDown)
            mOgreAppPtr->MoveView(0.0f, 0.0f, delta_y);
        else if(e.m_leftDown)
            mOgreAppPtr->RotateView(delta_x*2, delta_y);
        else if(e.m_rightDown)
            mOgreAppPtr->MoveView((Real)(-delta_x), (Real)delta_y, 0.0f);
    }

    // Save mouse position (for computing deltas for dragging)
    mMouseX = e.m_x;
    mMouseY = e.m_y;

    // Tell OGRE to redraw.
    if(mOgreAppPtr)
        mOgreAppPtr->Update();
}
  
```

The view is moved or rotated according to the relative movements of the mouse, then Update() is called to refresh the display.

Conclusion

It turned out not to be too painful to integrate wxWidgets with OGRE after all. OpenGL support, and then fully cross-platform GUIs are tantalizingly close, but out of reach in OGRE 0.14. This tutorial may be used and distributed freely, but may not be modified without written permission of the author, Dieter Buys. The associated demo code may be used, distributed and modified freely provided that credit always goes to the original author. The author welcomes any comments, suggestions or requests and can be contacted by email at:

dieter@telus.net.