Simple OpenGL Test Framework

<http://www.codeproject.com/Articles/33877/Simple-OpenGL-Test-Framework>

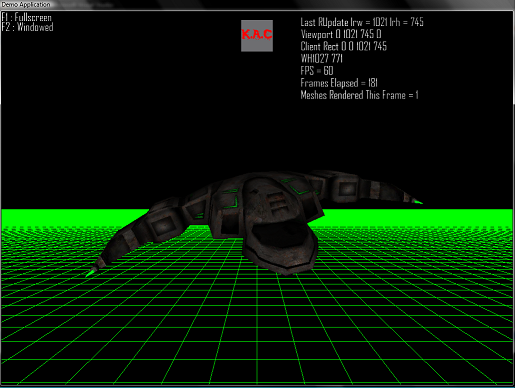


[**Kristian Carazo**](http://www.codeproject.com/script/Membership/View.aspx?mid=3885313), 5 Mar 2009 [CPOL](http://www.codeproject.com/info/cpol10.aspx)

|  |  |
| --- | --- |
| http://codeproject.cachefly.net/script/Ratings/Images/stars-fill-md.png  http://codeproject.cachefly.net/script/Ratings/Images/stars-empty-md.png | 4.94 (13 votes) |
|  | |  |  |  | | --- | --- | --- | | Rate: | [vote 1vote 2vote 3vote 4vote 5](http://www.codeproject.com/Articles/33877/Simple-OpenGL-Test-Framework) |  | |

A simple OpenGL framework for fast prototyping of OpenGL and game applications

* [**Download demo project - 2.64 MB**](http://www.codeproject.com/KB/openGL/OpenGLTestFrmwk/GLDemoApp_demo.zip)



Introduction

This is a simple framework for prototyping OpenGL and game applications. It provides a quick way to get up and running with all of the most commonly used tools already built in.

Included in the framework are the following features:

* Collada Loader
* OpenGL Renderer
* FreeType Font Rendering
* TGA Texture Loading
* Frame-rate Counting
* Full-screen/Windowed Support

Included inside of the zip file are the following compiled third party libraries.

* Collada DOM
* GLEW
* FreeType

Using the Code

In order to use the framework, simply unzip *GLDemoApp\_demo.zip*. Open up the project with Visual Studio 2005. There are known issues when working with Visual Studio 2008, you will need to rebuild the Collada DOM libraries with Visual Studio 2008.

Before using the framework, please take a moment to observe the directory structure. At the top level, there should be three directories:

* "*3rdParty*" - Contains all of the third party library and header files used in the framework
* "*bin*" - This is where all the executable files go after building the solutions; inside of this directory are these sub-directories: *release* and *resources*
* "*GLDemoApp*" - This is where all of the source files and *\*.sln* files are contained

The main point of interest in the code is contained inside of *demo.cpp*. Inside this file, there are three functions:

Hide   Shrink http://www.codeproject.com/images/arrow-up-16.png   Copy Code

void InitDemo()

{

RenderInit();

CColladaLoader loader;

g\_pTestColladaMesh = loader.LoadColladaFile(BuildResourcePath("test.dae"));

g\_uiTextureId = RenderLoadTexture(BuildResourcePath("test.tga"));

}

bool RunDemo()

{

static float32 fRotation = 0.0f;

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glPushMatrix();

gluLookAt(0.0f, 0.0f, 10.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);

RenderGroundGrid(-4.0f);

glPushMatrix();

glRotatef(fRotation, 1.0f, 1.0f, 1.0f);

RenderMesh(\*g\_pTestColladaMesh);

glPopMatrix();

glPushMatrix();

glPointSize(10.0f);

glBegin(GL\_POINTS);

glColor3f(1.0f, 1.0f, 1.0f);

glVertex3f(0.0f, 0.0f, 0.0f);

glEnd();

glPopMatrix();

glPopMatrix();

RenderText(&g\_defaultFont, "F1 : Fullscreen", Vector2f(0.0f, 20.0f));

RenderText(&g\_defaultFont, "F2 : Windowed", Vector2f(0.0f, 40.0f));

RenderBegin2dMode();

RenderTexture(Vector2f(RenderGetOrthoWidth() \* .5f, 50.0f), g\_uiTextureId, 0.0f);

RenderEnd2dMode();

if(GetAsyncKeyState(VK\_F1))

{

sRenderResolution sRes;

sRes.nWidth = RenderGetDesiredWidth();

sRes.nHeight = RenderGetDesiredHeight();

sRes.nBitDepth = RenderGetBitDepth();

RenderSetResolution(sRes, true);

}else if(GetAsyncKeyState(VK\_F2))

{

sRenderResolution sRes;

sRes.nWidth = RenderGetDesiredWidth();

sRes.nHeight = RenderGetDesiredHeight();

sRes.nBitDepth = RenderGetBitDepth();

RenderSetResolution(sRes, false);

}

fRotation += .1f;

if(fRotation > 360.0f)

fRotation = 0.0f;

RenderStats();

RenderSwapBuffers();

return (GetAsyncKeyState(VK\_ESCAPE) & 0xFFFF) ? false : true;

}

void EndDemo()

{

RenderReleaseTexture(g\_uiTextureId);

RenderReleaseMesh(&g\_pTestColladaMesh);

RenderDestroy();

}

It's pretty easy to see what is going on in here. InitDemo() is called from *gldemoapp.cpp* inside the winmainfunction right before the main app loop is entered. Inside the InitDemo() function, you should add any initialization code that you need for your application. Keep in mind that this function is only called once during the lifetime of the application. By default, inside the InitDemo() function, there is some code that loads a Collada file, and a texture file. You can simply remove most of this code except for RenderInit(), which does all of the OpenGL initialization stuff. The BuildResourcePath() function builds a path relative to the current directory from which the app is being run, one directory up and inside a folder named "*resources*". Look over the directory structure, and you will see that BuildResourcePath() always supplies a path that is "*ACTUALPATH\\bin\\resources\\yourfilename.ext*".

Next is the RunDemo() function. Inside it, you should put the bulk of your code. This function is called repeatedly during the execution of the application. It is important to note that this function returns a bool. This indicates whether the application should continue to run, or exit immediately. As you can see, there are several draw calls already inside this function. These are here simply for demonstration, and can be safely removed. Although, you should keep glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT) and RenderSwapBuffers().

Finally, the EndDemo() function is called when the application is about to terminate its execution. This is where most clean up code should be placed.

Rendering Functions

The framework provides many utilities that should make using OpenGL a lot easier. Some of the rendering and utility functions which you may find useful are listed here:

* Hide   Copy Code

RenderLoadTexture(const std::string& strFilePath)

Loads a TGA texture from disk and returns the OpenGL texture object ID. This ID can be used in calls toglBindTexture.

* Hide   Copy Code

RenderGroundGrid(float32 fYPos)

This function draws a grid on the ground at the given Y-coordinate position.

* Hide   Copy Code

RenderText(CFTFont\* pFont, const char\* strText, Vector2f& vtPosition, float32 fScale, const sRGBColor& rgbColor)

This function displays text using the font provided as the first parameter. By default, inside RenderInit(), there is a font loaded, and it can be accessed by using the global variable g\_defaultFont. For example, if you want to display some text, you could do the following...

Hide   Copy Code

RenderText(&g\_defaultFont, "Your cool text here", Vector2f(0.0f, 20.0f));

* Hide   Copy Code

RenderQueryResolutions(std::list<srenderresolution />& lstResolution)

Retrieves a list of available resolutions on the current display.

* Hide   Copy Code

RenderSetResolution(sRenderResolution& rRes, bool bFullScreen)

Sets the current resolution for the OpenGL window. Additionally, it can switch to full-screen by passing "true" in the second parameter. For example, to set the resolution and switch to full-screen, you could do the following...

Hide   Copy Code

sRenderResolution rRes;

rRes.nWidth = 640;

rRes.nHeight = 480;

rRes.nBitDepth = 32;

RenderSetResolution(rRes, true);

* Hide   Copy Code

RenderBegin2dMode()

Using this function will switch the current projection matrix to be orthographic. It is important to note that the orthographic resolution is independent of the perspective resolution.

* Hide   Copy Code

RenderEnd2dMode()

Use this function when you are done drawing in orthographic mode. The following snippet would draw a loaded texture at 50,50:

Hide   Copy Code

RenderBegin2dMode();

RenderTexture(50.0f, 50.0f), g\_uiTextureId, 0.0f);

RenderEnd2dMode();

* Hide   Copy Code

std::string ReadTextFileToBuffer(const std::string& strFileName)

This function will load a file from disk and put its contents in the returned buffer. This is useful for loading shaders.

Utility Classes

In addition to the above functions, there are several classes which you will find useful when working with the framework:

* CFTFont- This class is a wrapper for the FreeType library. It will load a given FreeType font and create the proper display lists for rendering. You can use an instance of this class as a parameter for RenderText.

Hide   Copy Code

CFTFont coolFont;

coolFont.Init(BuildResourcePath("MyAwesomeFont.TTF"), 18);

RenderText(&coolFont, "Loaded the font!", Vector2f(0.0f, 20.0f));

* CColladaLoader- This is a wrapper class for the Collada DOM library. This will load a Collada mesh file and store its constants into an instance of sMesh. Once the file has been loaded and processed, the instance returned from the load function can be used in calls to void RenderMesh(const sMesh& pMesh). The loader will load textures as long as they are TGA format. Please keep in mind that a lot of extra data provided by Collada will be discarded once it is loaded.

Hide   Copy Code

CColladaLoader loader;

sMesh\* pMyMesh = loader.LoadColladaFile(BuildResourcePath("colladaTest.dae"));

RenderMesh(\*pMyMesh);

*//Once you are done with the mesh call*

*//*

RenderReleaseMesh(&pMyMesh);

Additional Notes

Feel free to look around in the source files provided. There are many helper functions at your disposal that should make your life easier. If you run into any issues running a compiled executable, be sure to double check what your "Working Directory" is set to (should be "*..\bin\debug*" for debug builds and "*..\bin\release*" for release builds). This option can be found under "Debugging" in the Solution Properties window.

If you have any questions or comments about the code, please feel free to contact me at rebelcoder at gmail.com.

History

* 4th March, 2009: Initial post
* 4th March, 2009: Changes made to article