OmniMap API OpenGL Integration Guide

Version 2.0.3

Last updated: August 15, 2008

IMMERSIVE PROJECTION DESIGN
THE ELUMENATI
www.elumenati.com

Contents

.....1
OmniMap API 1
OpenGL Integration Guide.....1
Overview.....3
Intended Audience.....3
Developer Requirements.....3
Assumptions.....3
OmniMap API Components.....4
OmniMap API Architecture.....5
Demo Program Case Study.....6

Please send any questions or comments to support@elumenati.com.

Overview

This document is a guide for integrating the OmniMap Geometry Correction Library API into existing OpenGL applications. The document will provide an overview of the library's architecture, and how it fits into an existing OpenGL application, followed by a case study of the steps required to integrate the OmniMap API into a sample OpenGL program.

For more information, see the *OmniMap API Configuration Guide* and the *OmniMap API Class Documentation* included with the OmniMap installer.

Intended Audience

This document is written for programmers interested in implementing geometry correction within their real-time OpenGL application for use with the Elumenati OmniFocus range of products.

Developer Requirements

- Microsoft WindowsXP
- GPU supporting OpenGL 2.0
- Visual Studio 2005

Assumptions

This document assumes the developer has installed the OmniMap API from the installer program *OmniMap.msi*,and used the default location for the installation. The default installation location for the OmniMap API is:

C:\Program Files\Elumenati\OmniMap

In this document, we will refer to this folder as <*InstallationDir*>. So if you have chosen to install the OmniMap API in a location other than the default location, <*InstallationDir*> refers to that location.

OmniMap API Components

The OmniMap API is shipped with documentation, the example program referred to in this document, C++ header files, lib files and corresponding DLLs. The documentation includes this document, the API Configuration document, and detailed class documentation in HTML format.

The class documentation can be found in:

<InstallationDir>/docs/classdocs

The header files are found in:

<InstallationDir>/include

The library files can be found in:

<InstallationDir>/lib

The DLL's can be found in:

<InstallationDir>/bin

The example program can be found in:

<InstallationDir>/examples

OmniMap API Architecture

The OmniMap API is designed to be easily integrated into existing OpenGL applications. The main class that provides the dome rendering functionality is called *OmniMap*. The *OmniMap object* is constructed with arguments for the width and height of the final output:

```
OmniMap omniMapObj = OmniMap::OmniMap(width, height);
```

To generate a scene for a dome, the OmniMap class needs to call the application's rendering function 3 or 4 times with a modified viewing frustum and viewing angle. The rotation and viewing angles of each channel are dependent upon the dome configuration specified in the Lua script files *omnimap_startup.lua*, and *omnimap_user_edit.lua*. Each of these render passes is managed by the *OmniMapChannelBase* class. In the application where the rendering of a single frame is done, a call is made to the *OmniMap* object telling the *OmniMap* object which function to call for rendering a single channel. That call looks like:

```
omniMapObj->ForEachChannel(renderFunction);
```

where *renderFunction* is a pointer to a function that takes a pointer to an *OmniMapChannelBase* object as an argument:

```
void renderFunction(OmniMapChannelBase *channel);
```

ForEachChannel will call the *renderFunction* once for each channel. Prior to making that call, the OpenGL projection matrix has been set to the projection required to render the channel, and the OpenGL Modelview matrix has been set to the correct rotation to render the channel. The application programmer can access these values through the *OmniMapChannelBase* class interface. As each channel is rendered, the resulting image is placed into an OpenGL texture.

When all channels have been rendered, the application calls:

```
omniMapObj->PostRender()
```

This method will composite the channels and spherically project them onto the display.

Demo Program Case Study

This section describes what was done to create the project file for the Demo project located in:

<*InstallationDir*>*examples\demo\demo_dome_enabled\Source*

1. Open Visual Studio Workspace File

1.1 Open the Visual Studio Workspace for the dome enabled Demo program by double-clicking on “Demo.sln” in this directory.

2. Add OmniMap include Directory.

2.1 Using the properties dialog for the Demo project, we first added the include path for the OmniMap API include files to the property:

Configuration Properties

C/C++

General

Additional Include Directories

That path is :

<*InstallationDir*>*include*

In the project file, it is referenced relative to the Source directory. It is refererenced as ../../../../include.

3. Add OmniMap API Directory

3.1 Using the same properties dialog, we then added the library path for the OmniMap API library files to the property:

Configuration Properties

Linker

General

Additional Library Directories

That path is :

<*InstallationDir*>*lib*

In the project file, it is referenced relative to the Source directory. It is referenced as ../../../../lib.

4. Add OmniMap library dependency.

4.1 Finally, using the same properties dialog, we added the OmniMap library to the list of libraries to link with the Demo application. That property is:

Configuration Properties

Linker

Input

Additional Dependencies

That library is *OmniMap.lib* for the Release configuration, and *OmniMapD.lib* for the Debug configuration.

5. Edit "Basecode.cpp"

5.1 The changes to the code are shown in the *example program in <InstallationDir>\examples\demo\demo_dome_enabled\Source*

The changes made for OmniMap integration are commented and surrouned by

#if defined(USE_OMNIMAP)

#endif

5.2 The changes include the following:

5.2.1 Add *Omnimap* header file

5.2.2 Add *glew* header file

5.2.3 Nest *render()* function - or *display()* function – inside the *fun()* function. The *fun()* funcion will be called once for each OmniMap render channel. The application programmer is required to call the following methods in the sequence shown here:

```
// Push matrices, set projection, model view matrix
// enable frame buffer object for this channel's rendering
chan->beginRenderToChannel();
// The application's rendering code
Render();
// Pop matrices, disable frame buffer object
chan->endRenderToChannel();
```

The rest of the code shown in the example is for demonstration purposes, but is not required.

5.2.4 Update *quit()* and *resize()* functions.

5.2.5 Update the render function (or display code) to insure that all rendering calls happen inside the render function AND that all scene updates happen outside the render function.

5.2.6 Set up the render channel.

5.2.7 Replace the display code in the main loop with the Omnimap display code.

5.2.8 Delete Fullscreen message box – if present

5.2.9 Create the Omnimap object

5.2.10 Compile and Run.

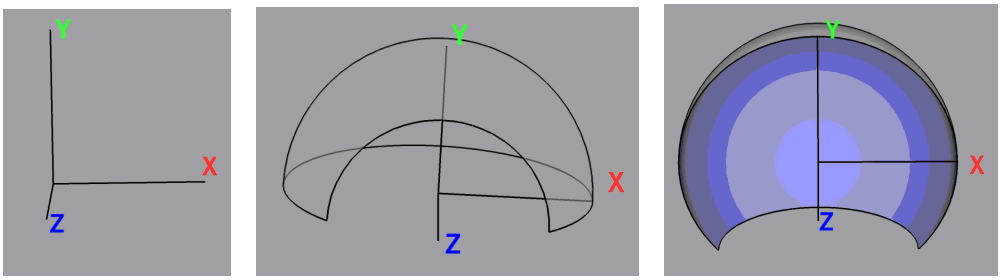
6. Set Omnimap Parameters

6.1 For details on setting OmniMap parameters, see the OmniMap Configuration Guide. OminMap parameters control where the projector is located, the direction of the projector, the orientation of the display and other configurable parameters. The parameters are set using Lua scripts. The default location of the configuration files is a directory called *OmniMapConfig*, located in the working directory of the program. The *OmniMap* library looks for a file called *OmniMapConfig/omnimap_startup.lua.* The location or name of that file can be changed using the 3rd argument to the *OmniMap* class constructor. There are two other LUA files referred to by *OmniMapConfig/omnimap_startup.lua* in the *OmniMapConfif* directory: *omnimap_user_edit.lua*, and *omnimap_dome_wiz_ai.lua*. If these files are not in the directory called *OmniMapConfig*, then the reference to them must be modified in *OmniMapConfig/omnimap_startup.lua*. The lines to be changed are:

```
Dofile("OmniMapConfig/omnimap_user_edit.lua")
Dofile("OmniMapConfig/omnimap_dome_wiz_ai.lua")
```

The parameters to the *dofile* call must be the correct paths to the two files.

6.2 Omnimap Coordinate System: Right Hand Rule where the X-vector is to the RIGHT, the Y-vector is UP and the Z-vector is AWAY from the user's gaze. The images below show the coord system in both a horizontal and vertical dome.



6.3 Parameters like dome style (horizontal, vertical), number of render channels and projector and audience position are set in the Lua (ASCII) file *omnimap_startup.lua* in the <*InstallationDir*>\bin*OmniMapConfig* directory.

6.4 The most common parameters however have been assembled into more the file *omnimap_user_edit.lua* in the same directory.

6.5 Run the program. Make sure that the directory containing the OmniMap DLLs is in your path. The directory is <*InstallationDir*>*bin.*