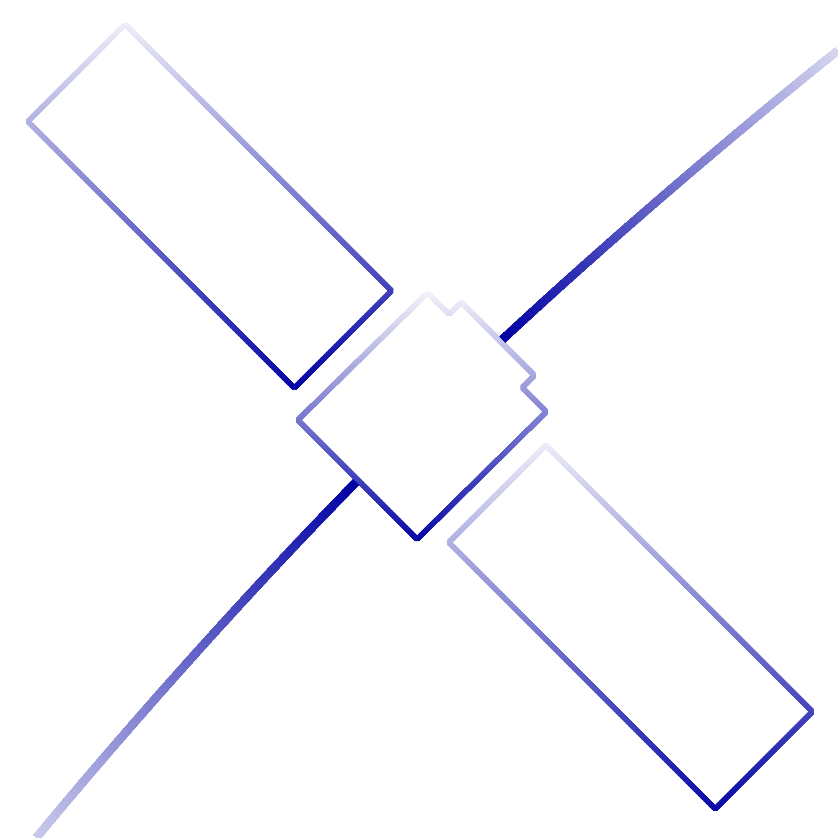


COMPASS

Developer’s Guide

Embry-Riddle Aeronautical University

Daytona Beach, FL

Table of Contents

1. Project Conventions
2. Editing Procedures
3. Deploying the Project
4. Resources

TODO:

Add QT and OpenGL Tutorials? Not like, *tutorial* tutorials, but have an outline of how they work and important information to know about each (and links to tutorials).

Project Conventions:

* Use Spaces instead of Tabs
  + QT replaces Tabs with spaces automatically
  + Notepad++ Setting: **settings**>**preferences**>**language**>**Replace by Space**
  + Convert Tabs to Spaces in Notepad++: **Edit**>**Blank Operations>TAB to Space**
* Use Standard Header Comment

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Computational Photometry Analyzer for Small Satellites

\* Copyright ©, 2017 Embry-Riddle Aeronautical University

\*

\* File: filename.h

\*

\* @author John Doe

\* @version 8/11/2017

\*

\* Include a description of this file here.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* Use the prefix “m\_” for all member variables.
* Comment Everything everywhere.
* Keep a change log with each addition to the program and add it to the master change log when uploading to GitHub.

TODO: Example change Log entry. Create format.

* Make code as modular as possible. Keeps things organized and easy to find.
* Avoid using QT libraries where possible. (Example: Use STD::String instead of QString)
* Use “Camel Case” variable naming style (i.e. compound words, capitalize beginning of each word but not the first word. Example: myEventHandler)
* Use descriptive variable names

Procedures for editing COMPASS:

1. Download and install Git: <https://git-scm.com/downloads>

Download and install QT Creator Open Source: <https://www.qt.io/download-open-source/?hsCtaTracking=f977210e-de67-475f-a32b-65cec207fd03%7Cd62710cd-e1db-46aa-8d4d-2f1c1ffdacea>

1. Clone the COMPASS repository onto your machine using the Git Command Terminal

- CMD: git clone <https://github.com/henry-valentine/COMPASS>

2. Configure and build the project in QT

- Open COMPASS.pro in QT

- On the selected default compiler kit, go to the Details tab and deselect *profile* and *debug*

- Select the default compiler kit and continue

- Change the build directory by going to **Projects>General>Build Directory**

- Click **Browse** and select <Your Repo Copy Directory>/COMPASS/build

3. Create a new branch from master to add your changes

- Branch should be titled according to what is being added

- CMD: git branch origin <Branch Name>

4. Checkout branch and make all changes

- CMD: git checkout <Branch Name>

- Add all changes to the program that you intend to make and return to step 5.

5. Delete your build information before committing changes

- Delete COMPASS/COMPASS.pro.user

- Delete all files in COMPASS/build *except* for “ui\_CpsWindow.h”

6. Add changes to the branch

- CMD: git add -A

7. Commit changes to the branch and include a message describing the changes

- CMD: git commit -m "I changed stuff"

8. Make sure your copy of the repository is up-to-date (VERY IMPORTANT)

- CMD: git checkout master

- CMD: git pull

- If any files were updated after running the pull command, you need to merge master into your branch.

- checkout your branch again

- CMD: git checkout <Branch Name>

- Merge master into your branch

- CMD: git merge master

- You may end up with a merge conflict and the merge will fail

- If there is a conflict, find the conflict and resolve it (Ask if you're not sure)

- Add and commit changes

9. Once your version is up to date, push your branch up to GitHub to be reviewed

- Check out your branch

- CMD: git checkout <Branch Name>

- Push it up

- CMD: git push

10. Create a pull request so your changes are reviewed

- Go to repository on GitHub <https://github.com/henry-valentine/COMPASS>

- Select your branch

- Click Green Button (Compare, Review, Create Pull Request)

- Green Button at the top (Create Pull Request)

- Add a title and comments to the pull request describing what changes or additions were made

- Click Send Pull Request

11. Have a peer review your pull request, do not merge the pull request yourself.

\*\*Create Write-Ups for significant contributions and add them to the Software-Design document. (e.g. the brightness model)

Helpful Tips:

Accidently Overwrite Files on Local Repository before adding them to GitHub:

- CMD: git fsck −−lost-found

When promoting a Widget to a user defined class in QT Designer, make sure you add the entire path for the include file. E.g. include/cpsGUI/cpsGraph.h

Style Sheets are modified in the UI editor, not the included qss style sheet.

**Installing OGRE:**

Prerequisites:

1. Download a prebuilt SDK here: <http://www.ogre3d.org/download/sdk>
2. Download and install CMake <https://cmake.org/download/>
3. Download and install Visual Studio for C++

Installing Ogre:

1. Open “BuildingOgre.md” from the OGRE SDK download file.
2. Follow the directions….
3. Complete the “Installing Ogre” section in order to copy the required libraries and headers into a clean location named “sdk.”
4. Place the contents of this folder inside the *dependencies/OGRE* directory. This directory should be in the COMPASS folder and may need to be created.

Integrating Ogre with QT:

1. Add the following lines to the .pro file before the SOURCES and HEADERS section:

*#* *#*

*#* *OGRE* *CONFIGURATION* *#*

*#* *#*

INCLUDEPATH += dependencies/OGRE/*include*/OGRE

INCLUDEPATH += dependencies/OGRE/*include*/OGRE/Overlay

LIBS += \

-L dependencies\OGRE\lib\OgreMain.lib \

-L dependencies\OGRE\lib\OgreOverlay.lib

**Setting Up an OGRE Application with Visual Studio**

The following instructions are based on these online tutorials.

1. <http://wiki.ogre3d.org/Setting+Up+An+Application+-+Visual+Studio>
2. <https://ogrecave.github.io/ogre/api/1.10/setup.html>
3. <https://github.com/OGRECave/ogre/blob/master/BuildingOgre.md>

Some modifications were required in order to get it working properly. Follow the instructions provided here.

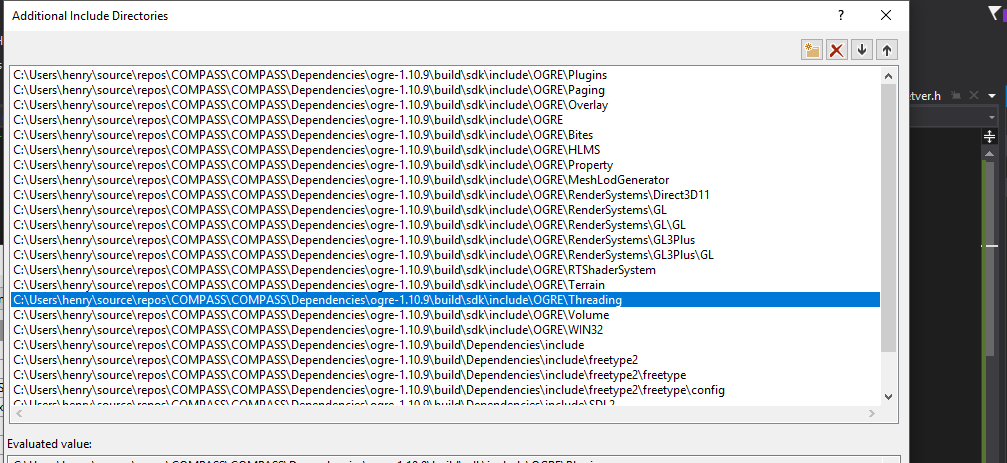
**Downloads**

1. Download OGRE source here: [http://www.ogre3d.org/download/sdk - Version 1.10.9](http://www.ogre3d.org/download/sdk%20-%20Version%201.10.9)
2. Download and install CMake: <https://cmake.org/download/>
3. Download and install Visual Studio for C++: <https://www.visualstudio.com/downloads/>

**Building OGRE**

1. Create a new solution in Visual Studio and navigate to the directory containing the source and header files. This directory will henceforth be referred to as <Project>
2. Create a new folder called “Dependencies” in this directory.
3. Place the contents of the OGRE source folder (From step 1 of Downloads) in a new folder under the Dependencies directory: “<Project>/Dependencies/ogre-1.10.9”.
4. Open the CMake GUI from the start menu.
5. In the field titled “Where is the source code:” enter the path to the OGRE source directory that you just created: “<Project>/Dependencies/ogre-1.10.9”
6. In the field titled “Where to build the binaries:” enter the following: “<Project>/Dependencies/ogre-1.10.9/build”
7. Press “Configure” at the bottom of the CMake window. You will be prompted to create the “<Project>/Dependencies/ogre-1.10.9/build” directory as it does not yet exist. Select “Yes.”
8. Select your installed version of Visual Studio (e.g. Visual Studio 15 2017) as the generator and click “Finish.”
9. This will configure the build files for OGRE. Once the configuration is complete, click Generate at the bottom of the CMake GUI.
10. Navigate to the directory you just created using CMake, “<Project>/Dependencies/ogre-1.10.9/build.” This should now be populated with new files and folders.
11. In this directory, find “OGRE.sln” and open it with the version of Visual Studio you used to generate the build files in CMake.
12. In Visual Studio, select the “Release” build configuration in the top center of the window.
13. Right click on the the “BUILD\_ ALL” project in the solution explorer on the far-right side of the window and select the “Build option.” This will build the release configuration of OGRE from source and will take several minutes.
14. Once the build successfully completes, select the “Debug” build configuration in the top center of the Visual Studio window.
15. Once again, right click “BUILD\_ ALL” and select “Build.” This will build the debug configuration of OGRE from source.
16. Once the second build is complete, right click the “INSTALL” project under the solution explorer and build this target under the Debug and Release configurations as was done for the “BUILD\_ ALL” target in the previous steps.
17. This should have created the following directory: “<Project>/Dependencies/ogre-1.10.9/build/sdk” which contains the files required to implement OGRE in your project.

**Configuring Visual Studio**

1. Open your project in Visual Studio.
2. Right-Click your project in the solution explorer on the right side of the window and select “Properties.”
3. In the top left corner of the pop-up window select “All Configurations” from the Configuration dropdown menu.
4. Under *Configuration* *Properties* 🡪 *General* change the “Character Set” field to “Use Multi-Byte Character Set”
5. Apply all changes. Then, set the Configuration to “Debug” (Configuration dropdown menu in top left of the properties window).
6. In the field titled “Target Name,” enter the following: $(ProjectName)\_d
7. Next, select the *Configuration* *Properties* 🡪 *Debugging* menu.
8. Change the Configuration to “All Configurations” once again.
9. Set the working directory field to “<Project>/Dependencies/ogre-1.10.9/build/sdk/bin”
10. Apply all changes. Then, change the Configuration to “Release.”
11. In the “Command” field, enter the following: “<Project>/Dependencies/ogre-1.10.9/build/sdk/bin/$(ProjectName).exe” and Apply changes.
12. Switch to the “Debug” configuration and enter the following in the “Command” field: “<Project>/Dependencies/ogre-1.10.9/build/sdk/bin/$(ProjectName)\_d.exe”
13. Next, go to *Configuration* *Properties* 🡪 *C/C++* 🡪 *General* and switch to “All Configurations”
14. Under the “Additional Include Directories” field select <Edit>
15. Another window will appear. Here, add all folders containing header files (example.h) from the following directories: “<Project>/Dependencies/ogre-1.10.9/build/sdk/bin/include” and “<Project>/Dependencies/ogre-1.10.9/build/Dependencies/include”
16. Go to *Configuration* *Properties* 🡪 *C/C++* 🡪 *Precompiled Headers* and select “Not Using Precompiled Headers” in the “Precompiled Header” field.
17. Go to *Configuration* *Properties* 🡪 *Linker* 🡪 *General.*
18. In the “Additional Library Directories” field, add all the folders containing library (example.lib) files from the following directories: “<Project>/Dependencies/ogre-1.10.9/build/sdk/bin/lib” and “<Project>/Dependencies/ogre-1.10.9/build/Dependencies/lib”
19. Go to *Configuration* *Properties* 🡪 *Linker* 🡪 *Input.*
20. Apply changes and switch to the “Release” configuration.
21. Under the “Additional Dependencies” field, type the name of each library (example.lib) file that was contained in the directories selected in step 18. IMPORTANT: Only include the libraries that DO NOT have the “\_d.lib” suffix. These are the Debug-specific library files and will be handled in the next step. The list should include the following for OGRE V1.10.9:

OgreBites.lib

OgreGLSupport.lib

OgreHLMS.lib

OgreMain.lib

OgreMeshLodGenerator.lib

OgreOverlay.lib

OgrePaging.lib

OgreProperty.lib

OgreRTShaderSystem.lib

OgreTerrain.lib

OgreVolume.lib

Plugin\_BSPSceneManager.lib

Plugin\_OctreeSceneManager.lib

Plugin\_OctreeZone.lib

Plugin\_ParticleFX.lib

Plugin\_PCZSceneManager.lib

RenderSystem\_Direct3D11.lib

RenderSystem\_GL.lib

RenderSystem\_GL3Plus.lib

freetype.lib

SDL2.lib

SDL2main.lib

zlib.lib

zlibstatic.lib

zzip.lib

1. Apply changes and switch to the “Debug” configuration.
2. Under the “Additional Dependencies” field, type the name of each library that DOES have the “\_d.lib” suffix, in addition to the library files contained in “<Project>/Dependencies/ogre-1.10.9/build/Dependencies/lib” which DO NOT have the “\_d.lib” suffix. For OGRE 1.10.9, this should consist of the following:

OgreBites\_d.lib

OgreGLSupport\_d.lib

OgreHLMS\_d.lib

OgreMain\_d.lib

OgreMeshLodGenerator\_d.lib

OgreOverlay\_d.lib

OgrePaging\_d.lib

OgreProperty\_d.lib

OgreRTShaderSystem\_d.lib

OgreTerrain\_d.lib

OgreVolume\_d.lib

Plugin\_BSPSceneManager\_d.lib

Plugin\_OctreeSceneManager\_d.lib

Plugin\_OctreeZone\_d.lib

Plugin\_ParticleFX\_d.lib

Plugin\_PCZSceneManager\_d.lib

RenderSystem\_Direct3D11\_d.lib

RenderSystem\_GL\_d.lib

RenderSystem\_GL3Plus\_d.lib

freetype.lib

SDL2.lib

SDL2main.lib

zlib.lib

zlibstatic.lib

zzip.lib

1. Next, go to *Configuration* *Properties* 🡪 *Build Events* 🡪 *Post-Build Events.*
2. Apply changes and switch to “All Configurations.”
3. Under the “Command Line” field, enter the following:

copy "$(OutDir)/$(TargetFileName)" "<Project>/Dependencies/ogre-1.10.9/build/sdk/bin"

1. Apply changes and click “OK.”
2. Visual Studio should now be configured to use the OGRE SDK in your project.
3. In order to Run your program in either Release or Debug mode, navigate to the directory: “<Project>/Dependencies/ogre-1.10.9/build/sdk/bin” and select ProjectName.exe or ProjectName\_d.exe respectively.

**Blender2Ogre**

https://github.com/OGRECave/blender2ogre

Note: Must add path to OgreXMLConverter.exe in config.py prior to first run. Follow install instructions from the README

Then export as an OGRE mesh from blender

How to Export and Use:

<https://www.youtube.com/watch?v=3Jcjz9iNHjI>

<https://www.youtube.com/watch?v=fddNclbNkts>

**Blender**

Good intro to modelling and texturing: <https://www.youtube.com/watch?v=9PJL0eAuZ_E>

Beginner Tutorials: <https://www.youtube.com/watch?v=yi87Dap_WOc&list=PLjEaoINr3zgHJVJF3T3CFUAZ6z11jKg6a>

**Ogre Notes**

Use resources.cfg to change file paths for resources