Building *THE* libraries

Prepare for the journey

I've had all sorts of problems building *THE* for the different platforms. I've done what I can to make this process less painful. There are pre-built binaries for each of the platforms. I encourage you to use these, as they will make the process easier. If you are the unlucky one that needs to rebuild the prebuilt libraries there is a more comprehensive section below. Otherwise the instructions assume you'll use the pre-built libraries.

I'll explain the versions I used for each of the packages. Many of the libraries are perfectly forward compatible. Unfortunately I've found that other libraries are not. For those that I noticed have trouble I'll make notes.

In numbered instructions, command line instructions are italicized, just copy past them to your console.

Windows

> *The original instructions state that you'll need the MinGW environment to install. I personally prefer Visual Studio, and will write these instructions with that in mind. If you choose to use MinGW, please refer back to the original build-from-scratch-w32.txt.*

CRCNS Environment Variable

Much of the tools references a CRCNS environment variable. It uses local paths from there to find all of the code, include files, and prebuilt libraries. Goto System Properties->Environment Variables. Create a new variable CRCNS and set it to your trunk directory. This directory should look as follows:

trunk (or alternate name depending on checkout)

- bin
- code
- docs
- include
- lib

Setting up Visual Studio

Oftentimes throughout building *THE* you may need to use a command line compiler. Visual Studio has one called *nmake*. In order to use it they have a batch file that will set up your environment. It is found at:

C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\vcvarsall.bat

To build a 64bit version pass in the x64 flag such as vcvarsall.bat x64

When using the IDE. Sometimes you will have a project that says Win32... To convert this solution to x64 do the following.

1. Click on Win32 drop down box.
2. Click Configuration Manager...
3. Click Win32->New
4. Select x64 from top drop down box.

5. Click ok.

You are now building in 64 bit.

Libraries for *THE*

Library	Pre-built	Required
Cg	Yes	Yes
Glew	Yes	Yes
itk	No	Yes
Boost	Yes (Only necessary libraries)	Yes
Qt	No	Yes
FLTK	No	No

Building Qt

Version: 4.4.3

The installer that Qt provides for windows is great. It comes with just about everything one needs. Unfortunately we need just a little bit more. So we'll have to build some things from scratch.

Note: Don't be fooled by Qt. Qt implies that the open source version can't be built using Visual Studio. They sell commercial versions that integrate with Qt. This doesn't mean the open source version can't be built by Visual Studio, it indeed can be. In fact I found that Qt has bugs and the build segfaults on the MinGW compiler (to my glee I assure you). The Visual Studio build was the only way to get Qt to build for Windows.

Use the installer to get all the things you need. Don't install MinGW (unless you want to). Go ahead and set up the Visual Studio environment (as shown above). Once it is setup...

1. Navigate to your qt directory.
2. *configure -release -static -qt-zlib -qt-gif -qt-libpng -qt-libmng -qt-libjpeg*
3. *nmake sub-src sub-tools*

Both the configure step and the make step take a fair bit of time. Without fail it will crash. Every time I've built it (and every version) it fails. Qt claims its a compiler issue. But it on mingw it segfaults so you're stuck.

Luckily the things that crash are not things that we need. Using Visual Studio...

1. Open the projects.sln (Created for you by Qt with the previous steps).
2. Make sure that Release x64 is selected.
3. Build using the IDE.

You don't have to build all of the projects. A few will fail, but none seem necessary to our projects.

Congratulations! One tool down.

Setting Up Qt Environment
Go to System Properties->Environment Variables:
Set the following variables...

QTDIR
C:\Qt\4.4.3   (or wherever you installed it)

Append to...
Path
%QT_DIR%\bin; %QT_DIR%\plugins\imageformats

Building Qt Projects
Unfortunately not all of the projects use cmake (though they really all should). The common alternative is using Qt projects. Qt projects are all contained in a .pro file. Below is an example of the current version of iris.pro

```
CONFIG +=   console
HEADERS =   IrisGui.h
SOURCES =   IrisGui.cpp iris.cpp
FORMS =   iris.ui

target.path =   myqtapp
sources.files =   $$SOURCES $$HEADERS $$RESOURCES $$FORMS *.pro
sources.path = .
INSTALLS = target  sources
TEMPLATE = app
```

You'll notice its a fairly simple file that contains information about how it needs to be built. To make a Visual Studio project out of this .pro file you can use qmake. Since we just built this, let's use it!

Building iris
1. Open up the command line.
2. Navigate to %CRCNS%\code\iris
3. *Create a folder for your project such as "vs"*
4. Use the command:
   qmake -t vcapp ..\iris.pro

Step three is italicized since its optional. Visual studio creates a fair bit of temporary data, so its nice to have that data in a separate directory. You're welcome not to. If you don't ../iris.pro will be changed to iris.pro.

Now using Visual Studio you can open the iris Visual Studio Project file. If you build Qt correctly, this will now build without a problem. Congratulations! You're one step further to getting *THE* built.

Cmake
Cmake is a cross platform tool for building. It allows you to pick your compiler and it creates a project file for that specific compiler. It supports both 32 bit and 64 bit Visual Studio 2008.

It can be downloaded here: http://www.cmake.org/

There are no specific needs for this tool.

Building itk

Note: Building itk takes a very long time. Plan for at least 2-3 hours of build time.

WARNING:
The Insight Toolkit has been patched by Paul. Because of this it is NOT wise to use the most up to date version of the Insight Toolkit. I have tried the latest itk and came across build errors
Instead use this one: http://superb-west.dl.sourceforge.net/sourceforge/itk/InsightToolkit-3.2.0.tar.gz

Download the above link and place in %CRCNS\bin. I suggest renaming it to itk. As of this writing the cmake files specifically look for an itk folder and an itkbin folder. I recommend naming them that way.

The good news is itk comes with a Cmake project file. But before we get started we need to patch itk.

Patching itk

Copy %CRCNS%\include\itk-patch to %CRCNS*\itk\

That's it. If you're using a newer version doing the above will stomp newer code changes. This can cause it to fail building. If you need a newer version I recommend trying to merge the changes instead. You may have some success.

Building itk

1. Run Cmake
2. Point binaries to %CRCNS%\bin\itk
3. Point source to %CRCNS%\itk

Turn off...
BUILD_EXAMPLES
BUILD_SHARED_LIBS – *This is important since we want static libraries. It won't build them with this turned on.*
BUILD_TESTING

Testing and examples are not needed.

Create a new project. Select Visual Studio 2008 64 bit. Make sure it is 64 bit. Mixing 32 bit and 64 bit does not work.

Configuring will take a while. You should have no errors. You can then click okay. You will have a new project built in: %CRCNS%\bin\itkbin. There should be a solution file named ITK.sln.

Double check to make sure that Release and x64 are selected. Click build, and go home. This build is super long, and Visual Studio will take all 4 cores compiling it.

Congratulations! Two tools down. And the long ones at that.

Building *THE*

Building *ir-tools*

Building ir-tweak

Comprehensive

Boost

FFTW

glew

CG