Lecture 4
Getting Started with ITK!

Goals for this lecture

- Compile, compile, compile
  - Learn how to use SVN & CMake
  - Build ITK
  - Compile several programs that use ITK
- Find documentation online
- Learn the quirks (if any) of the system you choose to use
**Getting help**

- Email your instructor
- Join the insight-users mailing list; instructions are at http://www.itk.org

**Assignments**

- Collaboration is **encouraged**; unless told otherwise, feel free to discuss assignments with other students
- But... *please* submit your own code - don’t copy and paste stuff from friends
- More so than other classes, you will be learning techniques that translate directly to the real world - don’t cheat yourself
Grading of assignments

- Grading criteria:
  - Does it accomplish the specified task?
  - Is it well commented? Follow the “6 month rule” - if you leave for 6 months, you should be able to pick up where you left off.
  - Many/most assignments will be divided into sections, with each section pass-fail.
  - We may give opportunities to fix “stupid” problems before final judgment is passed.

Assignments, cont.

- Please interpret due dates as absolute, unless told otherwise
- Really
- We’re happy to spend time helping you debug code, but not at 11 pm the day before the assignment is due
Computer requirements: recommended

- Your own computer is preferable
  - Cluster machines should also work
  - Please be aware that ITK can consume a lot of disk space during the build process
- Windows 7 through 10; with Visual Studio 2010, 2012, or 2015; Python ≥ 3.6 (x64)
  - Suggest Win10x64 with VS 2015 (V14) and Anaconda
- OS X Yosemite through High Sierra, X-Code, Python 3.6 (x64)
- Recent Linux (e.g. Ubuntu, RHEL, or CentOS), x64, with gcc ≥ 4.8, Python ≥ 3.6

What are the TA and I using?

- TA: Windows 10 x64 (preferred by TA):
  - Anaconda Python ≥ 3.6 (preferred by TA)
  - Visual Studio
- TA: Linux (Ubuntu, multiple versions):
  - Anaconda Python ≥ 3.6
  - gcc ≥ 4.9.2
- Instructor: Mac OS X El Capitan (10.11.6):
  - Anaconda Python 3.6.2
  - Xcode 8.2.1
**Alternative usable computer configurations**

- Any platform supported by ITK (Mac, Linux, etc.)
- **If there are problems, you will have to work with us to get your code working on one of our machines.**
  - Try having us check your code before it is due.
- If the grader’s computer can’t run your code, you will have a short (but reasonable) period of time to fix it after he emails you that your code appears broken (along with what errors he got).
  - If you are trying to make things work, but have many things to “fix,” then more time may be granted.
  - For final projects, we may decide to let you show us your code running on your own machine, on a case-by-case basis.

---

**What is ITK?**

- To clarify, ITK is a source-code/library *toolkit*
  - It doesn’t “do” anything
  - You can’t “run” it
  - There isn’t an itk.exe file
- Typically, you use ITK in conjunction with other toolkits to handle visualization and GUI interaction
So, what’s it good for?

- ITK code is easy to add to existing C++ code
  - Also Python, C#, Java, ...
- It provides a variety of flexible data containers, and ways of processing / analyzing them
- You can do a lot in only a few lines of code
- Once you get used to it, it’s easy to use (gasp!)

What we assume you can do

- Understand C++ and/or Python syntax
  - Standard flow control such as for, do, calling functions, etc.
  - Classes
  - Inheritance
  - For C++: Pointers, dereferencing, passing by reference
- Work comfortably in the operating system of your choice, using the compiler or Python environment of your choice, running programs from the command line (i.e. terminal)
You may have not…

- Used revision control using SVN (or git)
- Engaged in collaborative programming
- Written C++ code that builds on multiple platforms
- Used cross-platform make software
  - (CMake or Jam, for example)
- Designed software using a data-flow architecture, worried about smart pointers, etc.

Revision control with SVN

- Revision control software allows you to store incremental changes to software
- You will be expected to use SVN to manage your homework assignments
  - SVN is like CVS, but better
  - SVN is simpler than git (git is better for larger projects)
- I encourage you to use revision control on your code outside of this class as well - it’s a good habit to develop
SVN terms

- Server - what it sounds like
- Module - a group of files that can be accessed on the server
- User - each module has associated users, with varying levels of access (read only, read/write, etc.).

SVN terms, cont.

- Checkout - Download a fresh copy of a module from the server to your computer
- Update - Sync your copy of a module with the server copy; much faster than a checkout
- Commit - Merge changes made to your local copy with the server
SVN setup

- The SVN server for this course will be:
  - https://biglab.ri.cmu.edu/svn/miia19/{your user name}
- You will each have a module, based on your email; you will get email about this in a week or two.
- Only you and the instructor/TA will have access to this module

SVN setup, cont.

- GUI wrappers for SVN
  - Windows: Tortoise SVN
    - http://tortoisesvn.net/
  - Mac: svnX
    - http://code.google.com/p/svnx/
  - Windows, Mac, and Linux (cross-platform):
    - RapidSVN: http://rapidsvn.org/download/release/0.12.1/
    - SmartSVN: http://www.wandisco.com/subversion/download
- Command line works fine too, but may be more awkward if you’re used to GUI’s
Cross platform (C++) development

- ITK builds on a large combination of operating systems and platforms
- For C++, each compiler has it’s own input format: Makefiles, workspaces, etc.
- Q: How can you possibly coordinate builds on different platforms?

The answer: CMake

- Cross platform tool to manage the build process
- Simplifies the build process
- Auto-configuration
- Easy access to external libraries
- Used by several other open source projects

www.cmake.org
**CMake is:**

- Required to build native (C++) ITK
- Cross-platform project generator
- Often simpler than particular environments
- Text as input
- Project file as output:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Project Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Visual Studio Solution</td>
</tr>
<tr>
<td>UNIX</td>
<td>Makefile</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>Xcode project or Makefile</td>
</tr>
</tbody>
</table>

**How CMake runs**

- Write a [CMakeLists.txt](#) file describing your project in CMake’s language
- Run CMake to generate an appropriate makefile/project/workspace for your compiler
- Compile as you normally would
How CMake runs, cont.

- This is not unlike the configure-make process you may be familiar with from various Unix systems
- But... it works with many compilers
- CMakeLists.txt files are easy to perform revision control on

CMakeLists.txt syntax

- Comment lines indicated with #
- Look at examples in ITK
- Simple example:

```makefile
cmake_minimum_required(VERSION 3.10)
# Make sure the user’s CMake is recent enough

# Give this project a name:
PROJECT(cool_demo)

# The command-line executable “demo”
# is built from “demo_code.cxx” and
# must be linked with the ITK libraries
ADD_EXECUTABLE(demo demo_code.cxx)
TARGET_LINK_LIBRARIES(demo ${ITK_LIBRARIES})
```
Full Example of CMakeLists.txt

```cmake
CMAKE_MINIMUM_REQUIRED(VERSION 3.10)
# CMake policies sometimes change, so it is
# recommended that new projects require the
# latest CMake version

PROJECT(cool_demo)

# Find ITK:
FIND_PACKAGE(ITK REQUIRED)
INCLUDE(${ITK_USE_FILE})

ADD_EXECUTABLE(demo demo_code.cxx)
TARGET_LINK_LIBRARIES(demo ${ITK_LIBRARIES})
```

Steps to get started with C++ ITK & SimpleITK

- Pay Attention
- This is important for everyone for HW2
- If you are compiling C++ SimpleITK, you do not need to compile ITK separately
  - Compiling C++ SimpleITK using the “SuperBuild” approach automatically downloads and compiles ITK for us—we’ll explain more shortly

- I suggest you make a directory to hold all your source code and build trees for this class, e.g. c:\mimia
- Windows WARNING: Do not use long directory names on Windows, or place the directory in your Documents folder, or else the total path names will become too long for Windows and building (Simple)ITK will fail.
Step 0 – Don’t panic!

- There is *substantial* documentation on everything I’m going to present here, and vastly more about things that we will never cover in this course
- [https://simpleitk.readthedocs.io/](https://simpleitk.readthedocs.io/)
- [http://www.itk.org/ITK/help/documentation.html](http://www.itk.org/ITK/help/documentation.html)
- Download a copy of the *ITK Software Guide*

---

Step 1 - Install CMake

- Check if CMake ≥ 3.10 is already installed on your computer.
- If not, ...
- Download and install the latest stable binary distribution of CMake from:
Step 2 - Install (Simple)ITK

- Note: If you are going to compile SimpleITK (we are), then do not install ITK directly
  - SimpleITK’s “SuperBuild” will compile & install ITK for you
  - ...Following a procedure similar to that for ITK

- Create a directory to hold the source and build for both SimpleITK and ITK
- Open a command prompt and go to the new directory
- Download the latest version of SimpleITK using git:
  - `git clone --recursive http:// itk.org/SimpleITK.git`

In source vs. out source builds

**Source Tree**

- ITK
  - Common
  - Algorithms
  - BasicFilter
  - Numerics
  - IO

**Binary Tree**

- ITKb
  - Common
  - Algorithms
  - BasicFilter
  - Numerics
  - IO

Recommended!
Why use two trees?

- Keeps your C++ source and binary code separate
- Minimizes the amount of damage you can do to your SVN tree
- We suggest that you build SimpleITK in a new folder you create named SITKBin

WARNING: Because ITK is downloaded automatically by building SimpleITK, both the source and the build directories for ITK will be found inside the SITKBin directory

Where Will Everything Be?

- After building the SimpleITK SuperBuild as instructed, your important stuff will be here:
  - SimpleITK Source:
    - SimpleITK
  - SimpleITK Build:
    - SITKBin
  - SimpleITK C++ Libraries (one or both of these two):
    - SITKBin\lib\Release OR Debug may be present
    - SITKBin\SimpleITK-Build\lib\Release OR Debug
  - ITK Source:
    - SITKBin\ITK
  - ITK C++ Build:
    - SITKBin\ITK-build
  - ITK C++ Libraries:
    - SITKBin\ITK-build\lib\Release OR Debug may be present
Configuring a SimpleITK Build: Easy Start

- Run CMake

- Select the SOURCE directory:
  - \SimpleITK\SuperBuild

- Select the BINARY directory:
  - \SITKBin

- Press the “Configure” button

CMake: Choosing a Compiler

CMake may ask to choose which compiler (“generator”) you want to use for compiling:

- On Windows, choose your version of Visual Studio
- On OS X, choose Xcode (recommended) or “Unix Makefiles”
- On Linux, choose an appropriate Makefiles or Ninja option
Configure - Easy Start, cont.

- Disable each of these:
  - BUILD_EXAMPLES
  - BUILD_TESTS
  - WRAP_*, e.g. WRAP_JAVA, WRAP_LUA, etc.

- To read special microscopy-format images
  - Click the “Add Entry” button
  - Fill in the dialog as shown
Configuring and Generating

- After you change an option or options you will need to “configure” CMake again
- If the generate button (“OK” under Windows) is not presented, you definitely need to click the “configure” button again
- If any of the options are highlighted in red, you need to click the “configure” button again
- When done, click either “Generate” or “OK
  - Generating is usually very fast

Build ITK

- You are now ready to actually compile C++ SimpleITK and ITK
- In CMake, click “Open Project”
  - This will open up Visual Studio or Xcode, etc. for your SimpleITK project
- Visual Studio:
  - Build→Build Solution
- Xcode:
  - Product→Build
- It will probably take somewhere between 30-300 minutes, but your time may vary a lot
Verify the Build

SimpleITK Libraries will be found in:
SITK_BINARY / { SimpleITK-Build / } lib / { Debug, Release}

ITK Libraries will be found in:
SITK_BINARY / ITK-build / bin / { Debug, Release}

Alternative Approach:
Building with Makefiles and gcc

- WARNING: Do not follow these instructions if you already followed the previous CMake instructions.

- Order of operations is the same
- Differences
  - Can only configure for a debug OR a release build
  - Run the cmake executable, with one of these:
    - DCMAKE_BUILD_TYPE=DEBUG
    - DCMAKE_BUILD_TYPE=RELEASE
  - cmake uses a curses TUI, “identical” to the GUI
  - Run make instead of Visual Studio
  - Think of CMake as replacing the “./configure” step you may be used to
Alternative Approach: Building with Makefiles and gcc

Start in the directory containing SimpleITK

```bash
mkdir SITKBin
cd SITKBin
ccmake -DCMAKE_BUILD_TYPE=RELEASE
   -Module_SCIFIO=ON
   ../SimpleITK/SuperBuild
```

Edit CMake options
Reconfigure if needed

```bash
make
```

Now what?

- At this point, you have a bunch of source code and a bunch of compiled libraries
- As mentioned earlier, you don’t yet have anything executable
Reminder: What Will Be Where

- SimpleITK Source:
  - ../SimpleITK
- SimpleITK Build:
  - ../SimpleITK
- SimpleITK C++ Libraries (one or both of these two):
  - ../SimpleITK/lib/{Release OR Debug may be present}
  - ../SimpleITK-build/lib/{Release OR Debug}

- ITK Source:
  - ../ITK
- ITK C++ Build:
  - ../ITK-build
- ITK C++ Libraries:
  - ../ITK-build/lib/{Release OR Debug may be present}

Building an application

- ITK comes with a simple application you can build in order to test the ITK libraries “out of source” (i.e. not built inside ITK)
- It can be found in:
  
  SITKBin/ITK/Examples/Installation
How to build HelloWorld

- Copy & rename the *Installation* directory somewhere outside of the Insight directory
- Run CMake on **HelloWorld**
  - Remember the source/binary distinction and use **HelloWorldBin** as your build location
- CMake should automatically find ITK
  - if not, edit the **ITK_DIR** option

How to build HelloWorld, cont.

- Once CMake is happy, generate the makefile/project for your compiler
- Build HelloWorld
- Give it a try
More examples

- You can turn on ITK’s *Examples* option in CMake, which will build all of the examples for you.
- Or... you can copy the examples out-of-source and build them like you did HelloWorld
- These examples link into ITK Software Guide; read the chapter, poke the code and see what happens...

C++ Workflow thoughts

You should get used to the idea of:
1. Writing some code
2. Writing a CMakeLists.txt file
3. Running CMake
4. Building your code
5. Rinse, repeat
An aside: how to use ITK with existing C/C++ applications

- Your existing app may not use CMake
- In this case, you need to link to the ITK libraries explicitly and include the appropriate source directories
- This isn’t hard, but it may take some trial and error to discover everything you need
- You don’t need to worry about this in the context of this class

ITK Documentation

- Most of the ITK documentation is generated automatically from source comments using Doxygen
- Please familiarize yourself with the various means of navigating the Doxygen documentation online, e.g. selecting a Module and then selecting a Class
- Also, don’t forget about:
  - Download a copy of the ITK Software Guide
  - https://simpleitk.readthedocs.io/
Assignment 2...

- Will be assigned next week.
  - Please do NOT start until we have it ready
- Follows this pattern of CMake & Compilation
- Don’t procrastinate!
- If you’re going to have compiler problems, they will show up early in the course, so don’t procrastinate this assignment.