Image Registration
Lecture 6: Software Tools

Prof. Charlene Tsai

Overview of Course Software

- Insight Toolkit (ITK)
  - NLM funded project to develop open-source medical image processing, segmentation and registration libraries
  - Consortium of 3 companies and 3 universities with 6 more universities as subcontractors
- VXL (Vision X Libraries), where X is a place-holder for names such as “image”, “numerics”
  - Developed as volunteer effort by computer vision researchers at GE and several universities

Combining the Toolkits

- Exploiting ITK
  - 3D representations, processing algorithms, etc.
  - Intensity-based registration
  - Extensive examples and documentation
- Subsets of VXL
  - Significant, novel feature-based algorithms
  - Base library for robust estimation and other feature-based representations
- Working together
  - Some shared libraries already exist (numerics)
  - Code can be compiled and linked, so that objects and functions from ITK and VXL can work together.
  - BUT, do need to learn parts of two libraries

3

Moving Forward In This Course

- Lectures 7-10
  - ITK and intensity-based registration
- Lectures 11-15
  - Rensselaer registration library for feature-based registration
- Homework exercises will explore both toolkits

4

ITK by the Numbers

- March 2000
  - First code check-in
- 1000
  - # of nightly builds
- 21
  - # of platforms (software + hardware)
- 700
  - # of classes
- 1600
  - # of files with code
- 400K
  - # of lines of code

5

ITK by the Numbers

- 35K
  - # of lines of examples
- 150K
  - # of lines of Applications
- 50
  - unique developers
- 500
  - # of users subscribed to the mailing-list
- 300
  - # of emails posted monthly to the users-list

6

What takes to maintain a Project like ITK?

- What can go wrong with a project of this size?
- What is needed?
  - Synchronization of code
  - Testing of modules
  - Continuous multi-platform maintenance
  - Development of user guide

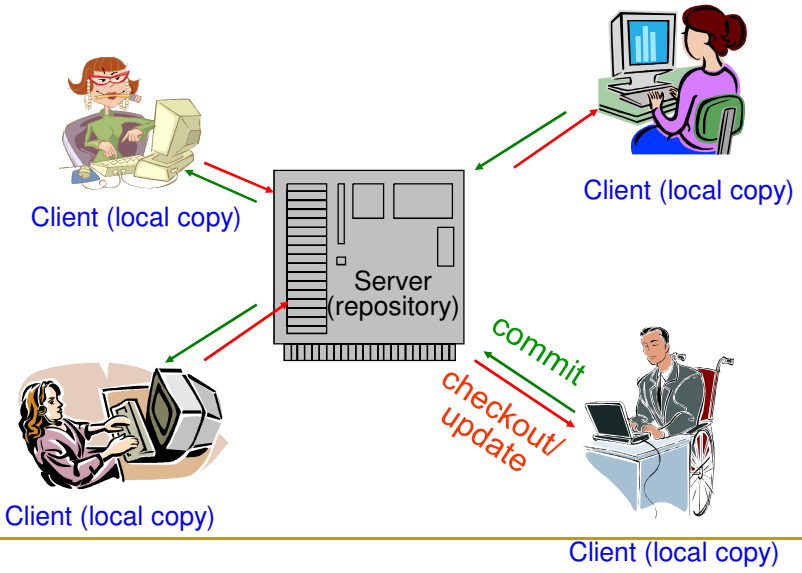
7

Development Tools and Procedures

- SVN
  - An open-source revision control system.
  - Keeping changes to software.
  - Vital for multiple developers.
- Doxygen
  - documentation system.
- DART (Dashboard)
  - Regression Testing System.
  - Providing feedback to developers who broke the compilation and testing.
- CMake
  - Cross-platform make system.
  - Generating appropriate makefiles (Unix) or workspaces (Windows).

8

What is SVN?



9

What is good about SVN?

- Its client-server access method lets developers access the latest code from anywhere there's an Internet connection.
- Its unreserved check-out model to version control avoids artificial conflicts common with the exclusive check-out model.
  - SVN is cautious and will merge automatically only as long as the changes aren't made to the same lines of code. If SVN can't safely resolve the changes, the developer will have to merge them manually.
- Its client tools are available on most platforms.

10

Download the Software

- Checking out vxl from sourceforge

  http://vxl.sourceforge.net/

  You can either download the latest official release, or check out from the svn repository.
- Checking out itk from Insight

  http://www.itk.org/ITK/resources/software.html

11

Doxygen

- Doxygen is a documentation system for C++, C, Java, etc..
- It can generate an on-line documentation browser Configurable:
  - extracting the code structure from undocumented source files.
  - visualizing the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams,
- This is very useful to quickly find your way in large source distributions.
- Example: http://www.itk.org/Doxygen314/html/index.html

12

DART (Dashboard)

15 Files Changed by 1 Authors as of 2003-11-17 07:00 GMT

Nightly Builds

Site	Build Name	Update	Cfg	Build		Test				Build Date
				Error	Warn	NotRun	Fail	Pass	NA	
pre.-vision.-cs.-rpi.-edu	FreeBSD-gcc-2.-95	15	0	0	1	73	0	0	6	Mon Nov 17 04:00:08 EST 2003
re.-vision.-cs.-rpi.-edu	FreeBSD-gcc-3.-0.-4-with-fresh-vxl	15	0	0	33	0	4	69	6	Mon Nov 17 05:01:22 EST 2003
pre.-vision.-cs.-rpi.-edu	FreeBSD-gcc-3.-3	15	0	0	121	73	0	0	6	Mon Nov 17 03:07:41 EST 2003
night.-vision.-cs.-rpi.-edu	Win2k-DotNET-Debug	15	0	0	57	0	5	68	6	Mon Nov 17 4:02:43 AM EST 2003
SUPERVISION	WinNT-vs6-Debug	0	0	32	90	19	2	33	25	Mon Nov 17 03:33:51 EST 2003

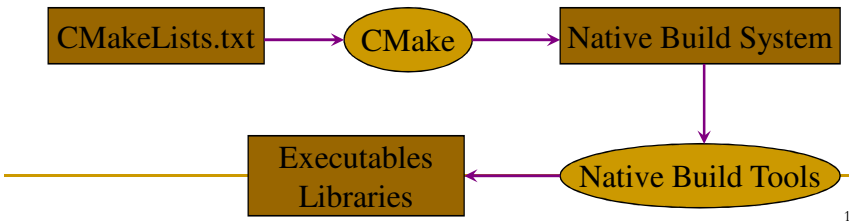
Continuous Builds

Site	Build Name	Update	Cfg	Build		Test				Build Date
				Error	Warn	NotRun	Fail	Pass	NA	
pre.-vision.-cs.-rpi.-edu	FreeBSD-4.-8-RELEASE-c++	1	0	0	0	73	0	0	6	Mon Nov 17 02:40:25 EST 2003

13

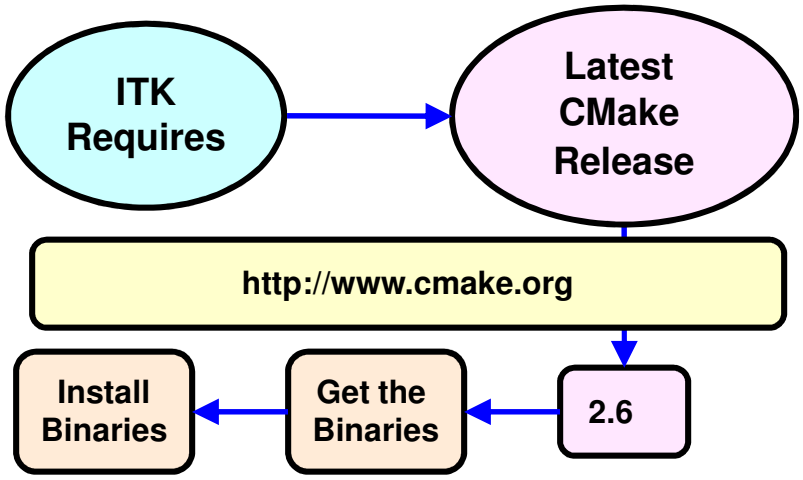
CMake

- Most slides by courtesy of Brad King from Kitware.
- The rest of the lecture is on CMake
- Build-System Generator
  - Provides single-sourcing for build systems
  - Knowledge of many platforms and tools
  - Users configure builds through a GUI



14

Downloading CMake



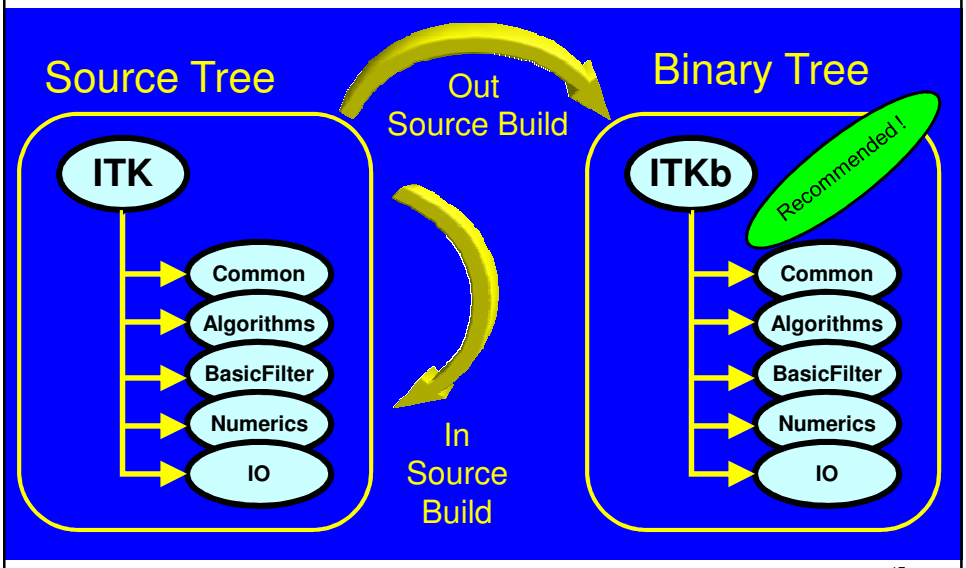
15

Source and Build Trees

- The *Source Tree* contains:
  - CMake input files (CMakeLists.txt)
  - Program source files (hello.cxx)
- The *Binary Tree* (build tree) contains:
  - Native build system files (hello.dsp)
  - Program libraries and executables (hello.exe)
- Source and Binary trees may be:
  - In the same directory (*in-source* build)
  - In different directories (*out-of-source* build)
    - Many binary trees (of different compilers, maybe) can share the same source tree

16

Example: Configuring ITK



17

The CMake Cache

- Represents build configuration
- Populated by CMake code
- Stored in CMakeCache.txt at top of build
- Entries have a type to help the GUI
- Holds global information for CMake code
- Updated by CMake configuration phase

18

Command Line Usage

- Can be used from scripts
- Set current-working-directory to binary tree
- Pass path to source tree as first argument
- Use -G to select build system generator
- Use -D to set cache variables

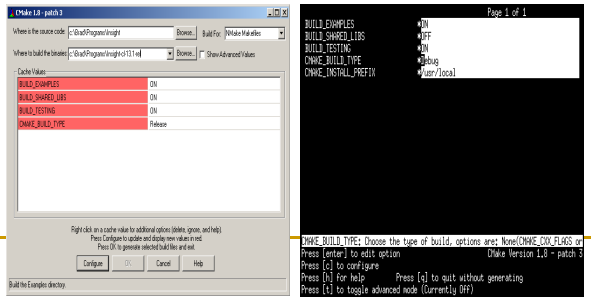
```
$ cd Foo-msvc-6
$ cmake ../Foo -G"Visual Studio 6" -DBAR:BOOL=1
```

19

GUI Usage

- Edit cache entries to configure the build
- Use configure button after a change
- Use OK (generate) button when finished

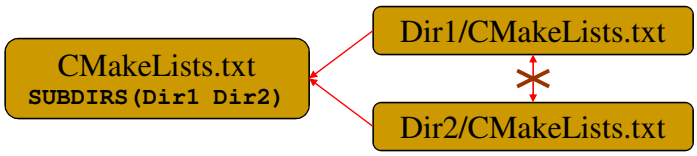
CMakeSetup **ccmake**



20

Source Tree Structure

- Every directory has a CMakeLists.txt file
- Subdirectories specified by SUBDIRS
- Directories depend only on parents
- A subset of commands are inherited



21

Writing CMakeLists.txt Files

- CMake language evolved while in use
- Scripting language with simple syntax
  - Comments `# Comment ends at a newline`
  - Commands `COMMAND(arg1 arg2 ...)`
  - Lists `A;B;C # Semicolon-separated`
  - Variables `${VAR}`
  - Control structures `IF(CONDITION)`
- Processed during CMake configure phase

22

Commands

- Simple syntax:

```
COMMAND(ARG "ARG WITH SPACES"
        ${A_LIST} "${A_STRING}")
```

- Each command must start on its own line
- Variable references are replaced by values
- Lists in unquoted arguments are expanded
- Argument meanings defined by command

```
TARGET_LINK_LIBRARIES(myTarget lib1 lib2)
FIND_LIBRARY(MY_LIB NAMES my1 my2
                    PATHS /foo /bar)
```

23

Variables

- Named by C-style identifier
- Value is always a string
- No associated type
- Initialized by CMake cache entries
- Assigned through commands like SET
- Referenced by ${VAR} (only one level)

```
SET(A_LIST ${A_LIST} foo)
SET(A_STRING "${A_STRING} bar")
```

24

Control Structures

- IF

```
IF(CONDITION)
  MESSAGE("Yes")
ELSE(CONDITION)
  MESSAGE("No")
ENDIF(CONDITION)
```

- FOREACH

```
FOREACH(c A B C)
  MESSAGE("${c}: ${${c}}")
ENDFOREACH(c)
```

- MACRO

```
MACRO(MY_MACRO arg1 arg2)
  SET(${arg1} "${${arg2}}")
ENDMACRO(MY_MACRO)
MY_MACRO(A B)
```

25

A Typical Project

CMakeLists.txt

```
PROJECT(FOO)
SUBDIRS(Foo Bar Executable)
```

Foo/CMakeLists.txt

```
ADD_LIBRARY(foo foo1.cxx foo2.cxx)
```

Bar/CMakeLists.txt

```
ADD_LIBRARY(bar bar1.cxx bar2.cxx)
TARGET_LINK_LIBRARIES(bar foo)
```

Executable/CMakeLists.txt

```
ADD_EXECUTABLE(zot zot1.cxx zot2.cxx)
TARGET_LINK_LIBRARIES(zot bar)
```

26

Developer Documentation

- Command-line documentation:
  - Run "`cmake --help`" for summary
  - Run "`cmake --help` *COMMAND*" for detailed help with a specific listfile command
  - Try "`cmake --help IF`"
- Online documentation:
  - http://www.cmake.org/HTML/Documentation.html
- Mastering CMake
  - Published by Kitware, Inc.
  - ISBN 1-930934-09-2

27

Editing CMake Code

- EMACS mode for CMake
  - `cmake-mode.el` located in CMake/Docs directory
  - Provides highlighting and indentation
  - Use this code in your .emacs file:

```
(setq load-path (cons "/path/to/cmake-mode" load-path))
(require 'cmake-mode)
(setq auto-mode-alist
      (append '(("CMakeLists.txt" . cmake-mode)
                ("\\.cmake$" . cmake-mode))
              auto-mode-alist))
```

- VIM mode is also available

28

Building ITK & VXL Together

- Build VXL using any configuration
- Run CMakeSetup to build ITK
  - Click “Configure”
  - Turn on “Show Advanced Values”
  - Set ITK_USE_SYSTEM_VXL to ON
  - Click “Configure”
  - Set VXL_DIR to point at VXL build tree

29

Using ITK & VXL Together

- Import ITK using code like this:

```
FIND_PACKAGE(ITK)
IF(ITK_FOUND)
  INCLUDE(${ITK_USE_FILE})
  IF(NOT ITK_USE_SYSTEM_VXL)
    MESSAGE("Need an ITK with ITK_USE_SYSTEM_VXL ON.")
  ENDIF(NOT ITK_USE_SYSTEM_VXL)
ELSE(ITK_FOUND)
  MESSAGE(FATAL_ERROR "Set ITK_DIR")
ENDIF(ITK_FOUND)
```

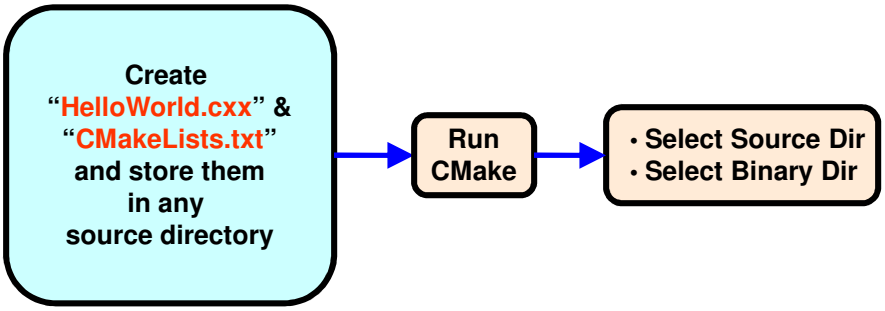
30

Starting Your Own Project

- Create a clean new directory
- Write a CMakeLists.txt file
- Write a simple .cxx file
- Configure with CMake
- Build
- Run

31

Example on Using ITK – Hello World



32

CMakeLists.txt

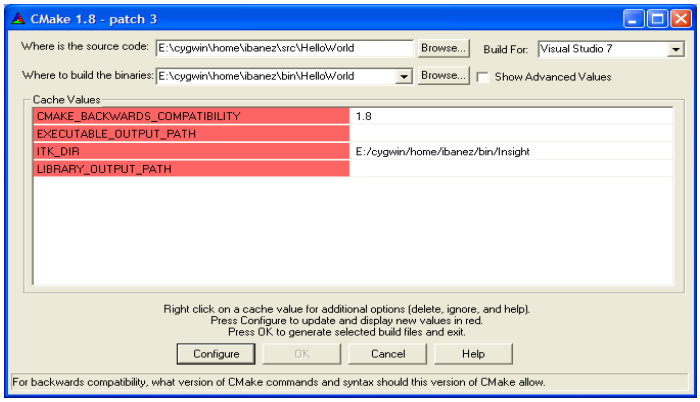
```
PROJECT( myProject )

FIND_PACKAGE ( ITK )
IF ( ITK_FOUND )
      INCLUDE( ${ITK_USE_FILE} )
ELSE( ITK_FOUND )
      MESSAGE(FATAL_ERROR
            "Cannot build without ITK.  Please set ITK_DIR.")
ENDIF( ITK_FOUND )

ADD_EXECUTABLE( HelloWorld   HellowWorld.cxx )
TARGET_LINK_LIBRARIES ( HelloWorld ITKCommon   ITKIO)

#ADD_EXECUTABLE( foo   foo.cxx )
#TARGET_LINK_LIBRARIES ( foo ITKCommon   ITKIO)
```

HellowWorld.cxx

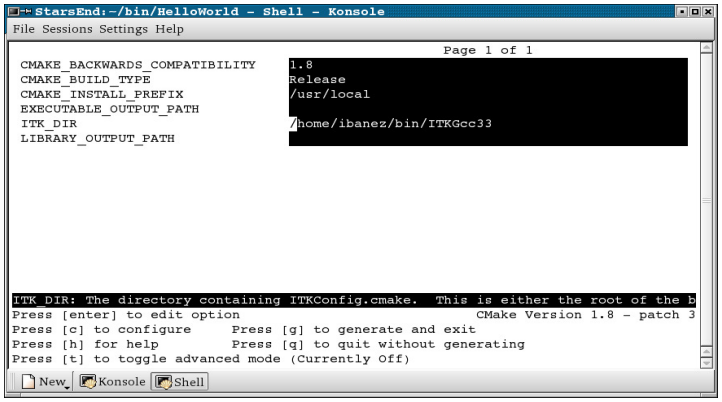
```
#include "itkImage.h"
#include <iostream>

int main( int argc, char **argv ) {
  typedef itk::Image<unsigned short,3>      ImageType;

  ImageType::Pointer image = ImageType::New();

  std::cout<< “ITK Hello World!” << std::endl;

  return 0;
}
```

34

Configure CMake



Configure CMake



Configure CMake

- Accept the default in CMAKE_BACKBARD_COMPATIBILITY
- Leave empty EXECUTABLE_OUTPUT_PATH
- Leave empty LIBRARY_OUTPUT_PATH
- Set ITK_DIR to the binary directory where ITK was built

37

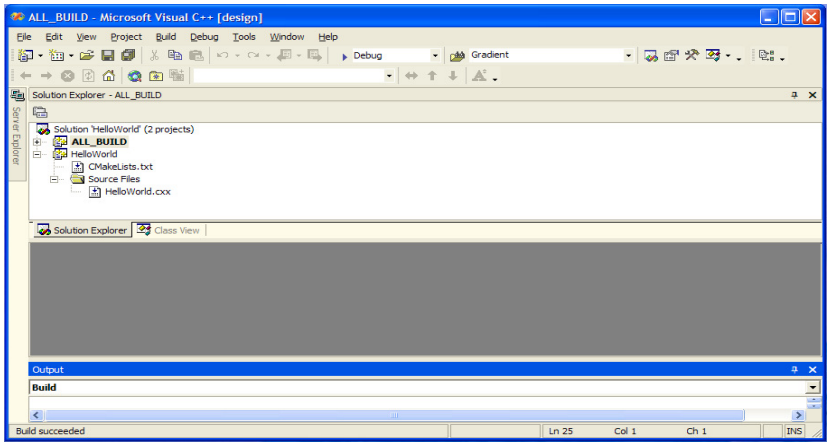
Building the myProject Project

- Open myProject.dsw (or .sln) generated by CMake
- Select ALL_BUILD project
- Build it

…It will take about 3 seconds …

38

Building the myProject Project



39

Execute the Excutables

- Locate the file HelloWorld.exe
- Run it…
- It should produce the message:

  ITK Hello World !

40