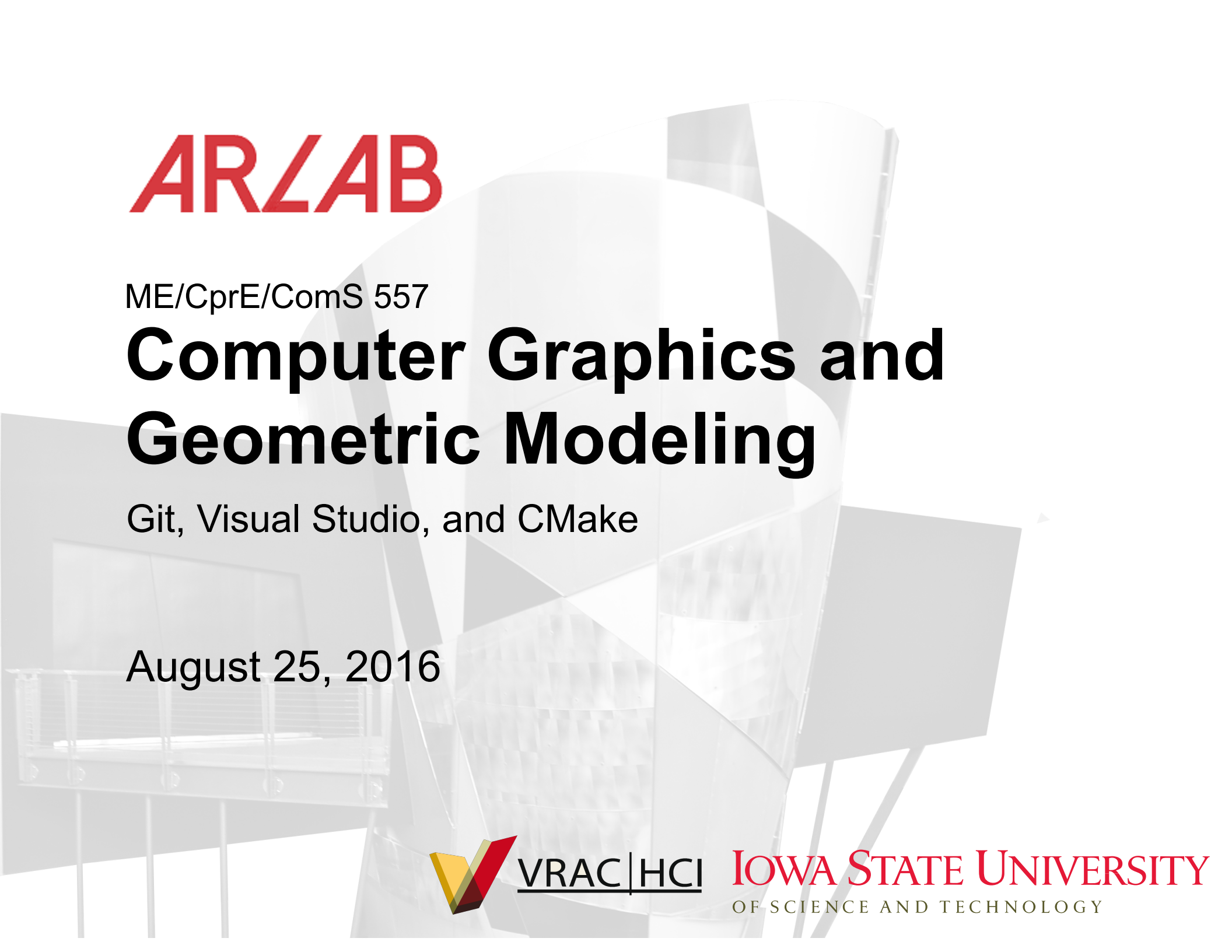


ME/CprE/ComS 557

Computer Graphics and Geometric Modeling

Git, Visual Studio, and CMake

August 25, 2016

VRAC|HCI

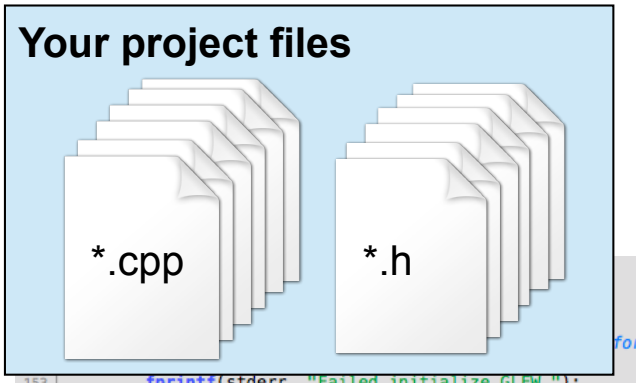
IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Content

ARLAB

- Compiler, source files, and frameworks
- Introduction to git
- Visual Studio and Visual Studio Solutions / Projects
- CMake
- Installing OpenGL (GLEW, GLM, GLFW)
  - Windows
  - Mac OS X with Homebrew
- Homework: test project

Software Development

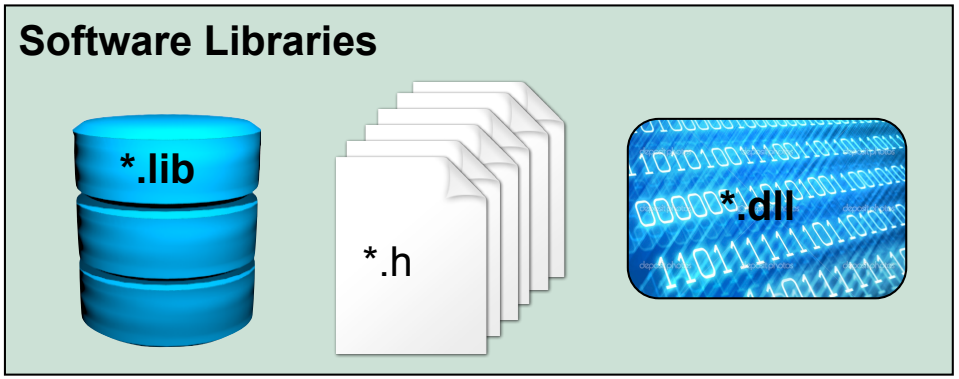


with C/C++

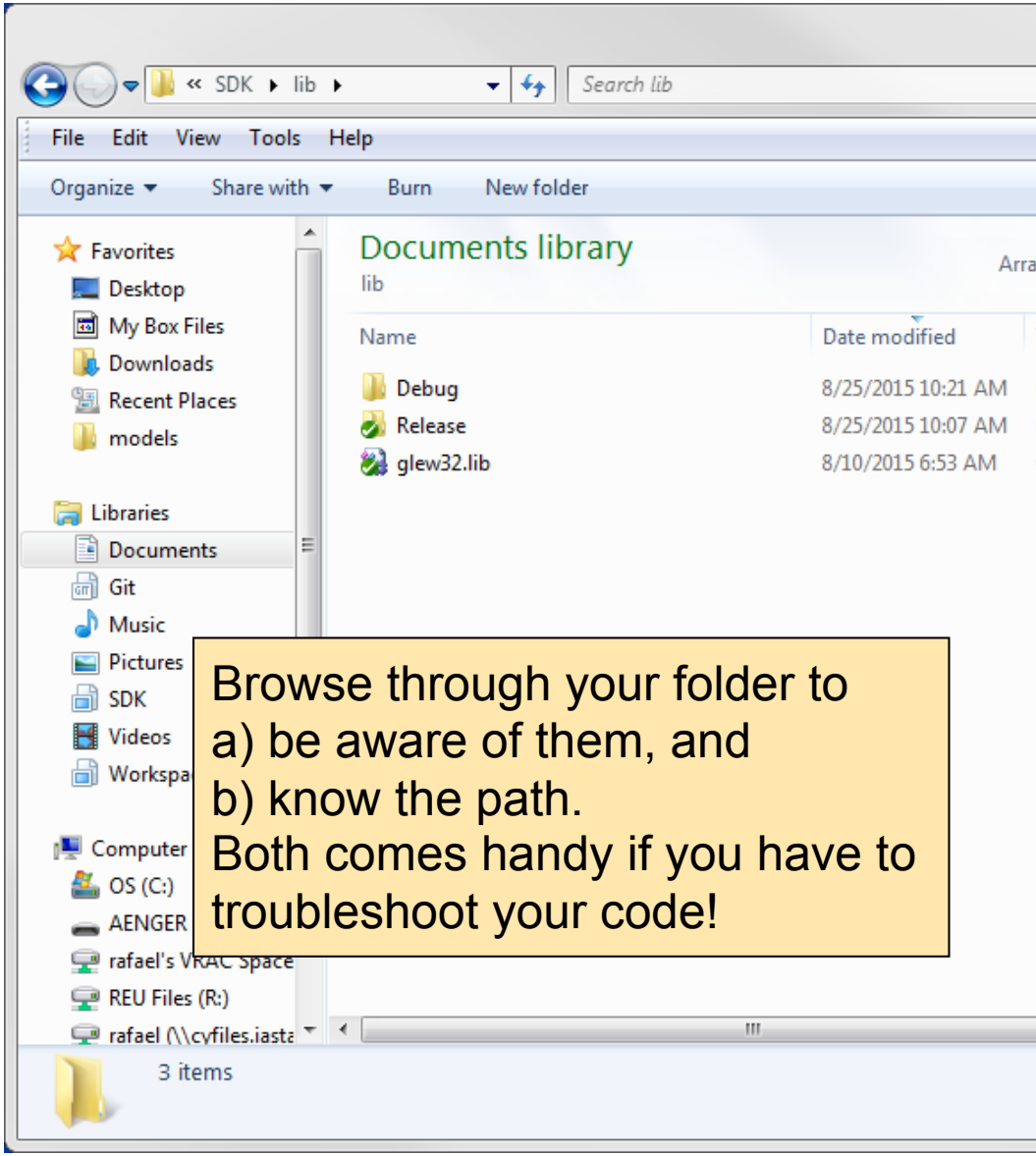
- .h - header files: keep all the declarations
- .cpp - implementation files: keep all the definitions

```
for any reason, print it out to STDERR.
153         fprintf(stderr, "Failed initialize GLFW.");
154         exit(EXIT_FAILURE);
155     }
156
157     // Set the error callback, as mentioned above.
158     glfwSetErrorCallback(error_callback);
159
160     // Set up OpenGL options.
161     // Use OpenGL verion 4.1,
162     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
163     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);
164     // GLFW_OPENGL_FORWARD_COMPAT specifies whether the OpenGL context should be forward-compatible, i.e. one where all functionality
165     glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
166     // Indicate we only want the newest core profile, rather than using backwards compatible and deprecated features.
167     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
168     // Make the window resize-able.
169     glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
170
171     // Create a window to put our stuff in.
172     GLFWwindow* window = glfwCreateWindow(800, 600, "Hello OpenGL", NULL, NULL);
173
174     // If the window fails to be created, print out the error, clean up GLFW and exit the program.
175     if(!window) {
176         fprintf(stderr, "Failed to create GLFW window.");
177         glfwTerminate();
178         exit(EXIT_FAILURE);
179     }
180
181     // Use the window as the current context (everything that's drawn will be place in this window).
182     glfwMakeContextCurrent(window);
183
184     // Set the keyboard callback so that when we press ESC, it knows what to do.
185     glfwSetKeyCallback(window, key_callback);
186
187     printf("OpenGL version supported by this platform (%s): \n", glGetString(GL_VERSION));
188
189     // Makes sure all extensions will be exposed in GLEW and initialize GLEW.
190     glewExperimental = GL_TRUE;
191     glewInit();
192
193     // Shaders is the next part of our program. Notice that we use version 410 core. This has to match our version of OpenGL we are u
194
195     // Vertex shader source code. This draws the vertices in our window. We have 3 vertices since we're drawing an triangle.
196     // Each vertex is represented by a vector of size 4 (x, y, z, w) coordinates.
```

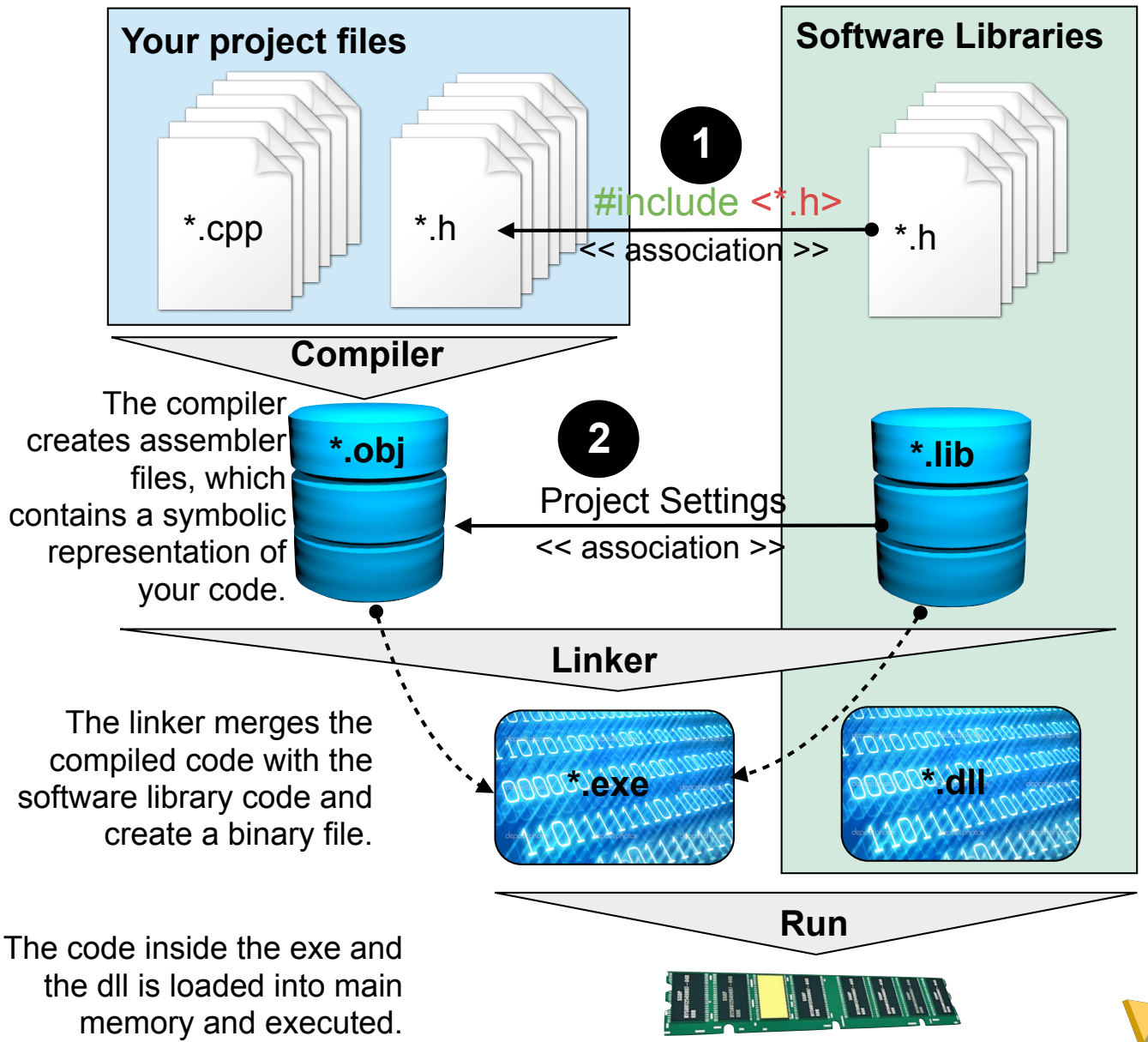
Software Libraries and their Files



- Software libraries provide APIs to additional functions for your project.
- .h files: contain the declaration of the function
- .lib files: contain the function signature and entry point + memory information.
- .dll files: contain the executable machine code
- Example: GLEW library uses glew32.lib
  GLFW library file: glfw.lib
  GLM is a "header only" programming packages, and provides only .h files.



C/C++ Compiler



Every software project consists of two set of code: your own code and code from software libraries.

Your project code incorporates a set of cpp-files and header files.

The software library incorporates a set of header files, a library (multiple library files), and a binary file (dll), which contains the executables.

C/C++ code is generated in two steps.

First, a compiler compiles your project files and generates object files (obj). The contain assembler code. During this step, your code needs to know all the libraries and the provided function. This association is established using the #include command in your header files. The obj files contain a symbolic link to each library function.

Secondly, the Linker merges the generated obj files to one binary file. During this process, the Linker searches the lib files for the binary code, related to the symbolic links.

The result is an executable file containing machine code.

During program start, the machine code from the exe and the dll are loaded into computer's main memory. Thus, the program runs.

ARLAB

Software Tools

Software Tools

ARLAB

- Microsoft Visual Studio or XCode (Mac OS X)
- Windows: Microsoft Windows SDK
- CMake
- Git repository (GitHub, Bitbucket)
- SourceTree or GitHub
- OpenGL
  - GLEW
  - GLM
- GLSL

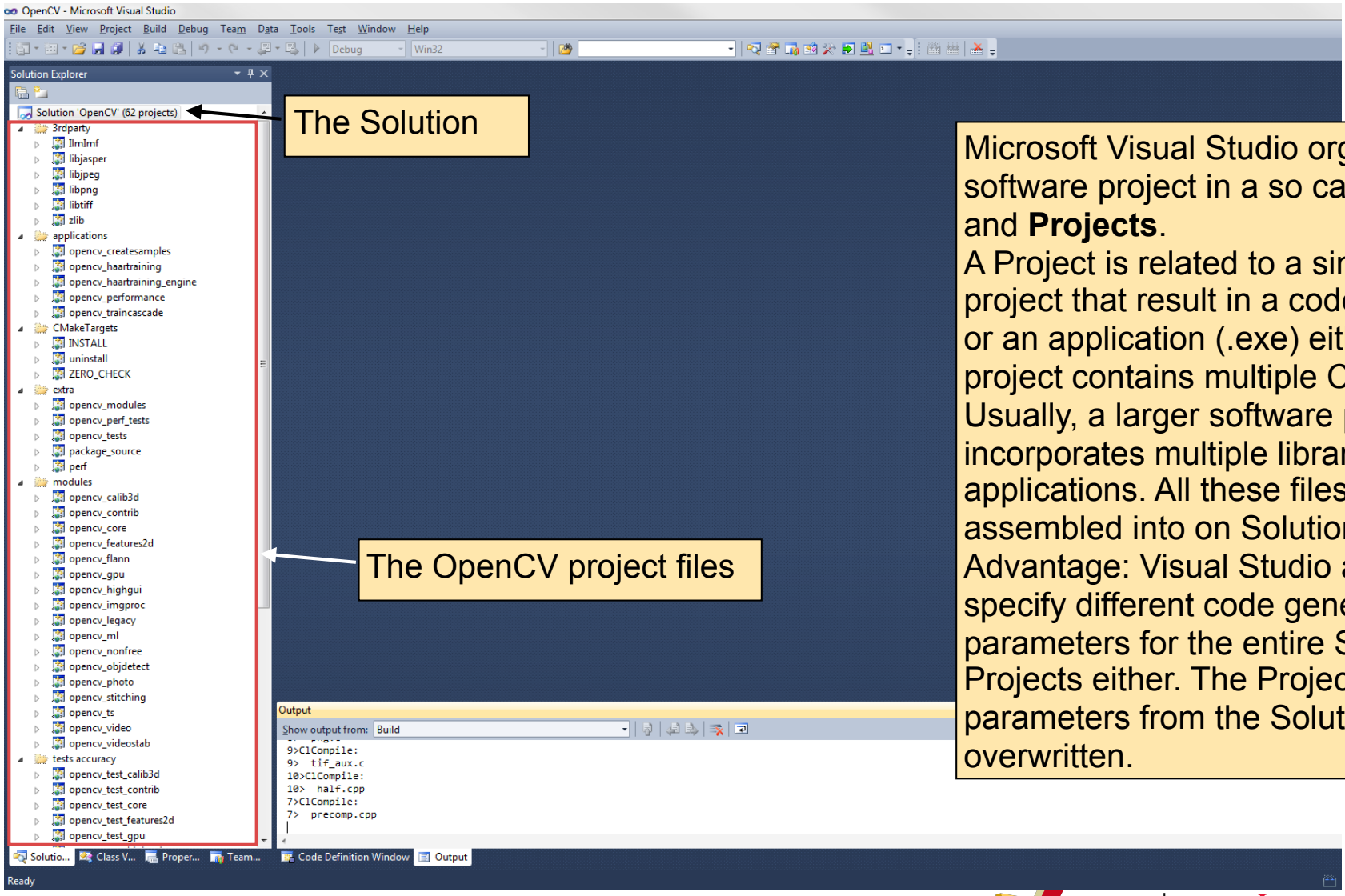
ARLAB

Visual Studio

Microsoft Visual Studio

ARLAB

MS Visual Studio organization



Microsoft Visual Studio organize a software project in a so called **Solution** and **Projects**.
A Project is related to a single code project that result in a code library (.lib) or an application (.exe) either. Each project contains multiple C/C++ files. Usually, a larger software project incorporates multiple library files and applications. All these files are assembled into on Solution.
Advantage: Visual Studio allows to specify different code generation parameters for the entire Solution or the Projects either. The Projects inherit the parameters from the Solution if not overwritten.

Microsoft Visual Studio main view

Microsoft Visual Studio

ARLAB

Visual Studio Express

Visual Studio Express editions provide free tools to develop applications for a specific platform, such as Windows Universal Platform applications, web sites, and Windows desktop applications.

New edition available

Visual Studio Community has all the features of Express and more, **and is still free** for individual developers, open source projects, academic research, education, and small professional teams.

Download Community 2015

Learn more >

https://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx

Microsoft Visual Studio

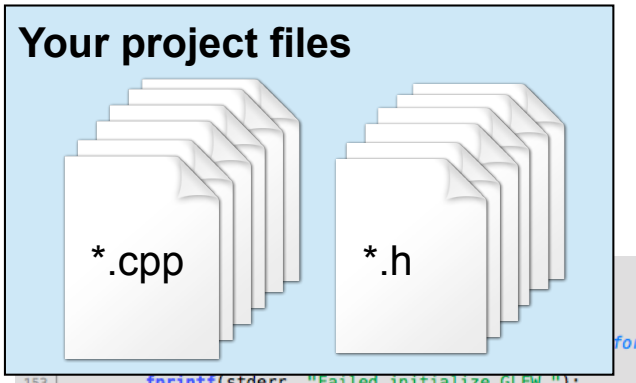
ARLAB

More later.....

ARLAB

Git

Back to Files with Code



with C/C++

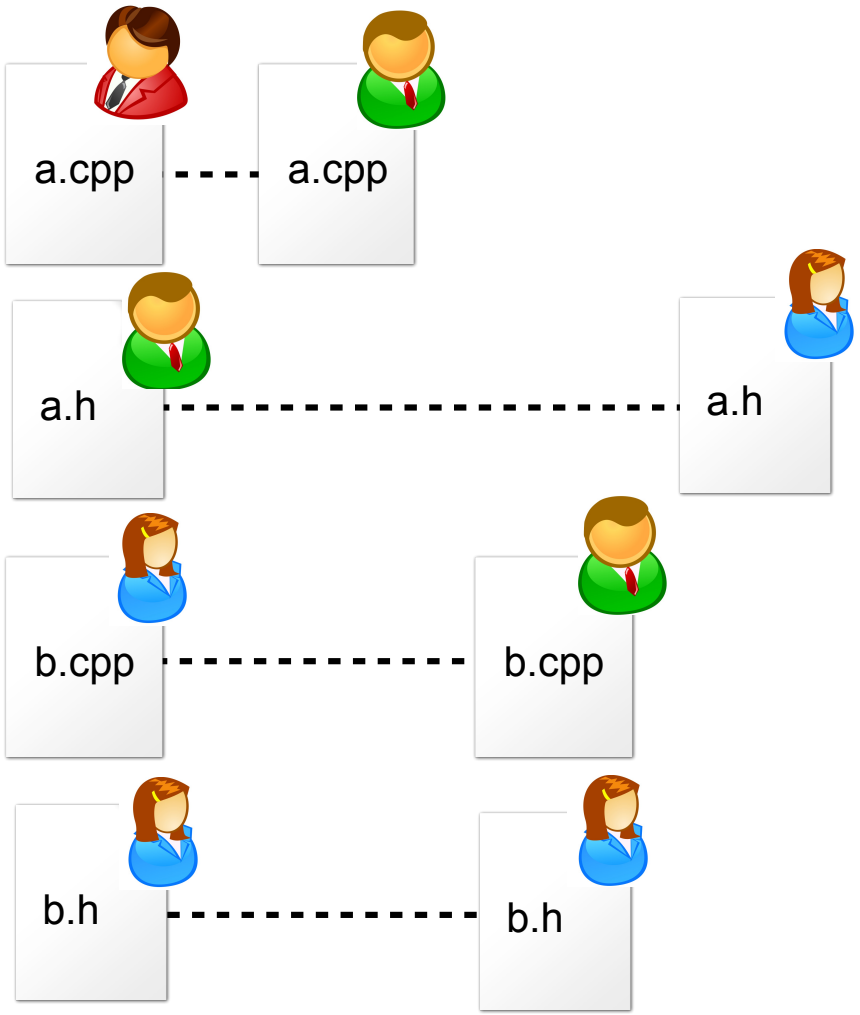
- .h - header files: keep all the declarations
- .cpp - implementation files: keep all the definitions

```
                                                                    for any reason, print it out to STDERR.
153         fprintf(stderr, "Failed initialize GLFW.");
154         exit(EXIT_FAILURE);
155     }
156
157     // Set the error callback, as mentioned above.
158     glfwSetErrorCallback(error_callback);
159
160     // Set up OpenGL options.
161     // Use OpenGL verion 4.1,
162     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
163     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);
164     // GLFW_OPENGL_FORWARD_COMPAT specifies whether the OpenGL context should be forward-compatible, i.e. one where all functionality
165     glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
166     // Indicate we only want the newest core profile, rather than using backwards compatible and deprecated features.
167     glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
168     // Make the window resize-able.
169     glfwWindowHint(GLFW_RESIZABLE, GL_FALSE);
170
171     // Create a window to put our stuff in.
172     GLFWwindow* window = glfwCreateWindow(800, 600, "Hello OpenGL", NULL, NULL);
173
174     // If the window fails to be created, print out the error, clean up GLFW and exit the program.
175     if(!window) {
176         fprintf(stderr, "Failed to create GLFW window.");
177         glfwTerminate();
178         exit(EXIT_FAILURE);
179     }
180
181     // Use the window as the current context (everything that's drawn will be place in this window).
182     glfwMakeContextCurrent(window);
183
184     // Set the keyboard callback so that when we press ESC, it knows what to do.
185     glfwSetKeyCallback(window, key_callback);
186
187     printf("OpenGL version supported by this platform (%s): \n", glGetString(GL_VERSION));
188
189     // Makes sure all extensions will be exposed in GLEW and initialize GLEW.
190     glewExperimental = GL_TRUE;
191     glewInit();
192
193     // Shaders is the next part of our program. Notice that we use version 410 core. This has to match our version of OpenGL we are u
194
195     // Vertex shader source code. This draws the vertices in our window. We have 3 vertices since we're drawing an triangle.
196     // Each vertex is represented by a vector of size 4 (x, y, z, w) coordinates.
```

Code Management - The Problem

ARLAB

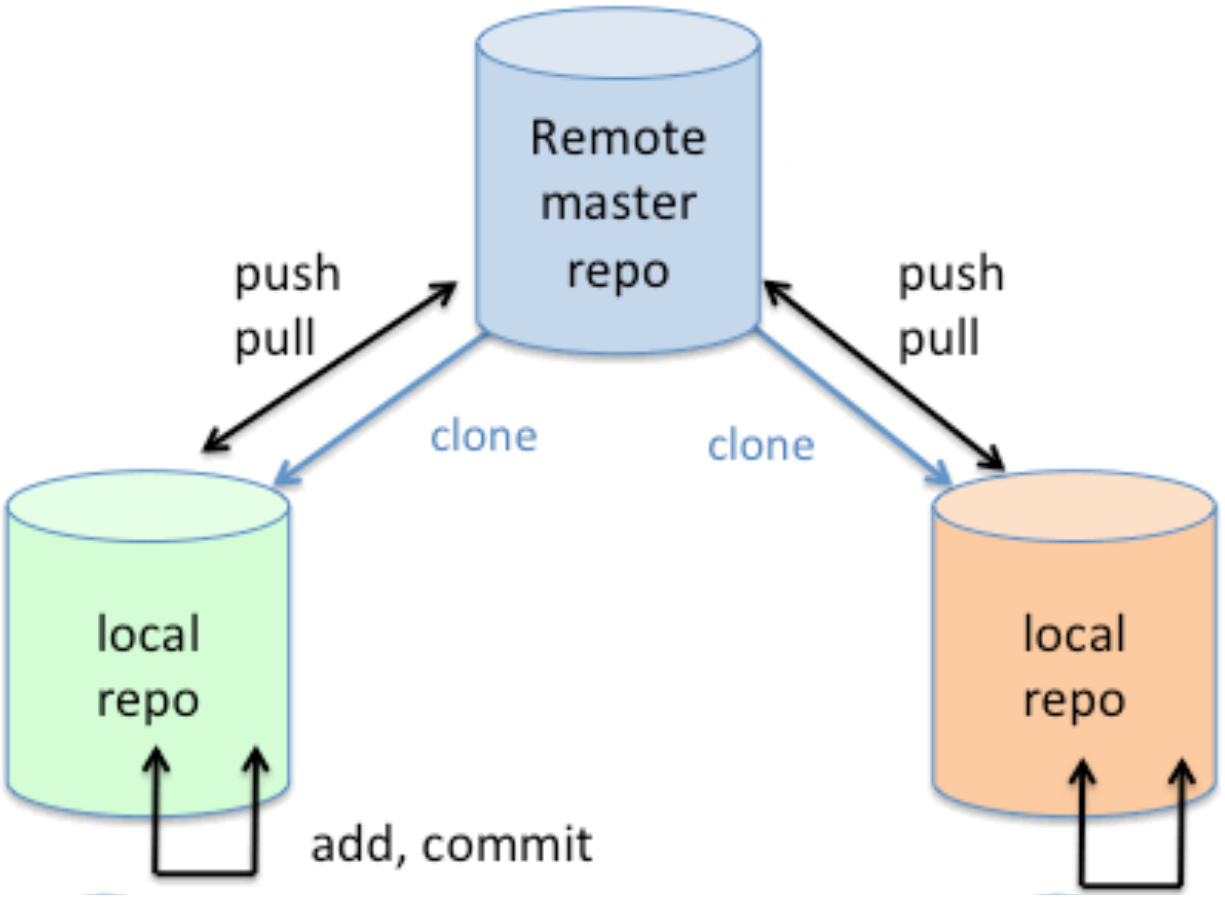
Multiple programmers develop code



Git is a file version control system for software development.

- Keeps track of all code changes
- Identifies conflicts when different users change the same code
- Automatically merges code when no conflicts exist.
- etc.

Architecture

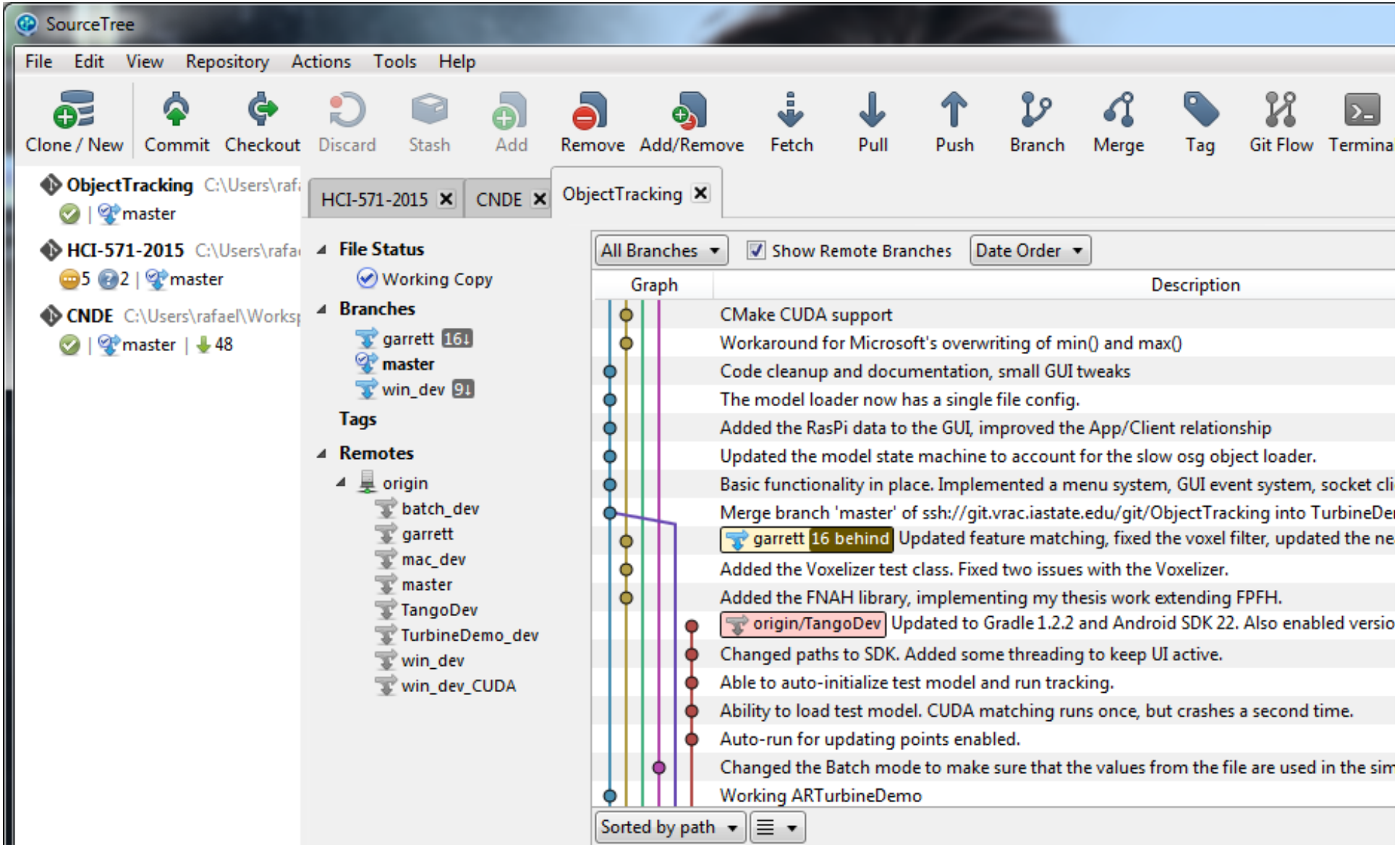


- Git works with a server-based architecture.
- All users have access to code using a remote repository
- Everybody develops using his/her local repository.

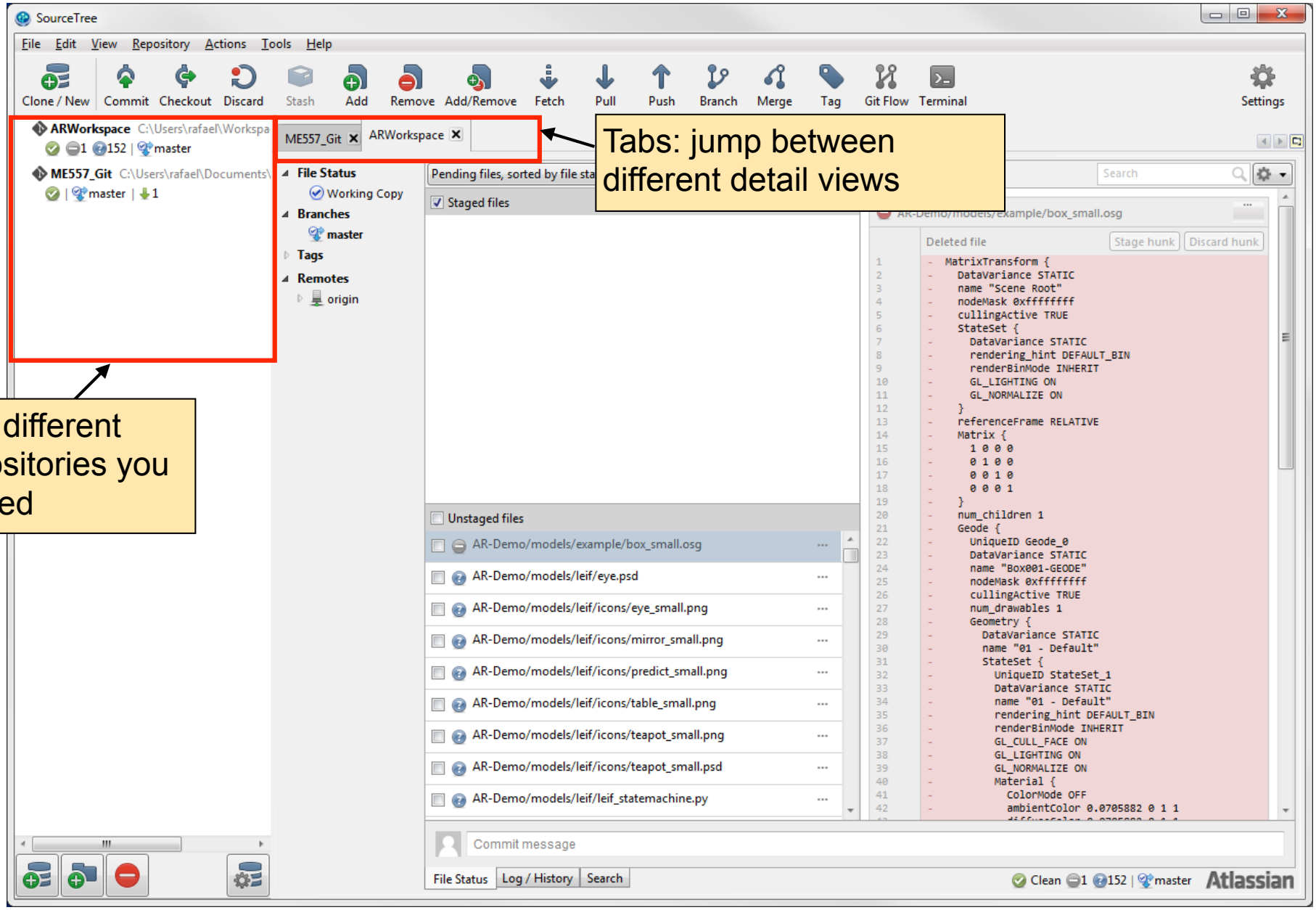
- **clone: the first time you get a repository from the server!**

Repository structure of a typical git-system

SourceTree

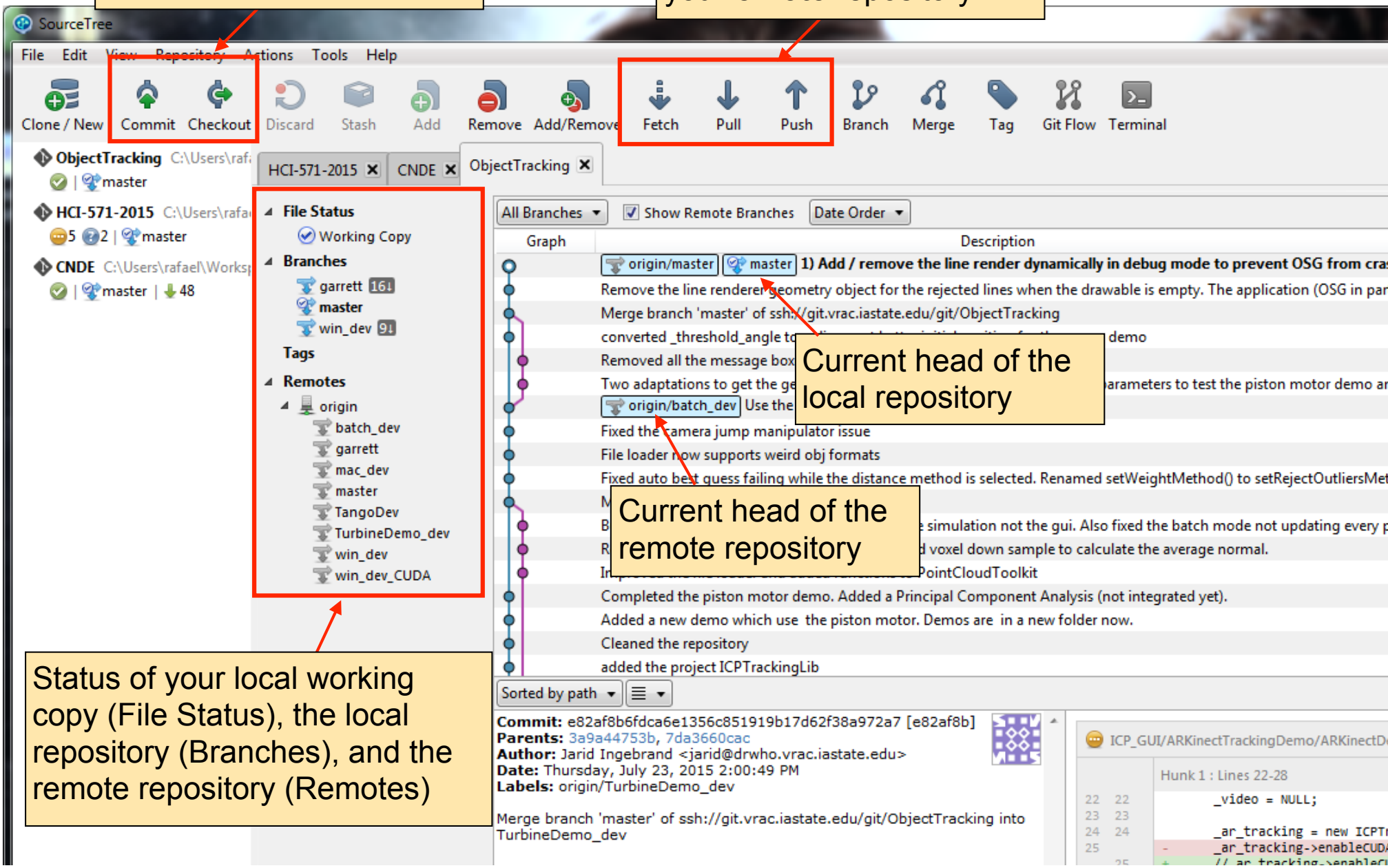

SourceTree
File Edit View Repository Actions Tools Help
Clone / New
Commit
Checkout
Discard
Stash
Add
Remove
Add/Remove
Fetch
Pull
Push
Branch
Merge
Tag
Git Flow
ObjectTracking C:\Users\raf
master
HCI-571-2015 C:\Users\rafa
5 2 | master
CNDE C:\Users\rafael\Works
master | 48
HCI-571-2015
CNDE
ObjectTracking
File Status
Working Copy
Branches
garrett 16↓
master
win_dev 9↓
Tags
Remotes
origin
batch_dev
garrett
mac_dev
master
TangoDev
TurbineDemo_dev
win_dev
win_dev_CUDA
All Branches
Show Remote Branches
Date Order
Graph
Description
CMake CUDA support
Workaround for Microsoft's overwriting of min() and max()
Code cleanup and documentation, small GUI tweaks
The model loader now has a single file config.
Added the RasPi data to the GUI, improved the App/Client relationship
Updated the model state machine to account for the slow osg object loader.
Basic functionality in place. Implemented a menu system, GUI event system, socket cli
Merge branch 'master' of ssh://git.vrac.iastate.edu/git/ObjectTracking into TurbineDe
garrett 16 behind Updated feature matching, fixed the voxel filter, updated the ne
Added the Voxelizer test class. Fixed two issues with the Voxelizer.
Added the FNAH library, implementing my thesis work extending FPFH.
origin/TangoDev Updated to Gradle 1.2.2 and Android SDK 22. Also enabled versio
Changed paths to SDK. Added some threading to keep UI active.
Able to auto-initialize test model and run tracking.
Ability to load test model. CUDA matching runs once, but crashes a second time.
Auto-run for updating points enabled.
Changed the Batch mode to make sure that the values from the file are used in the sim
Working ARTurbineDemo
Sorted by path

Source Tree



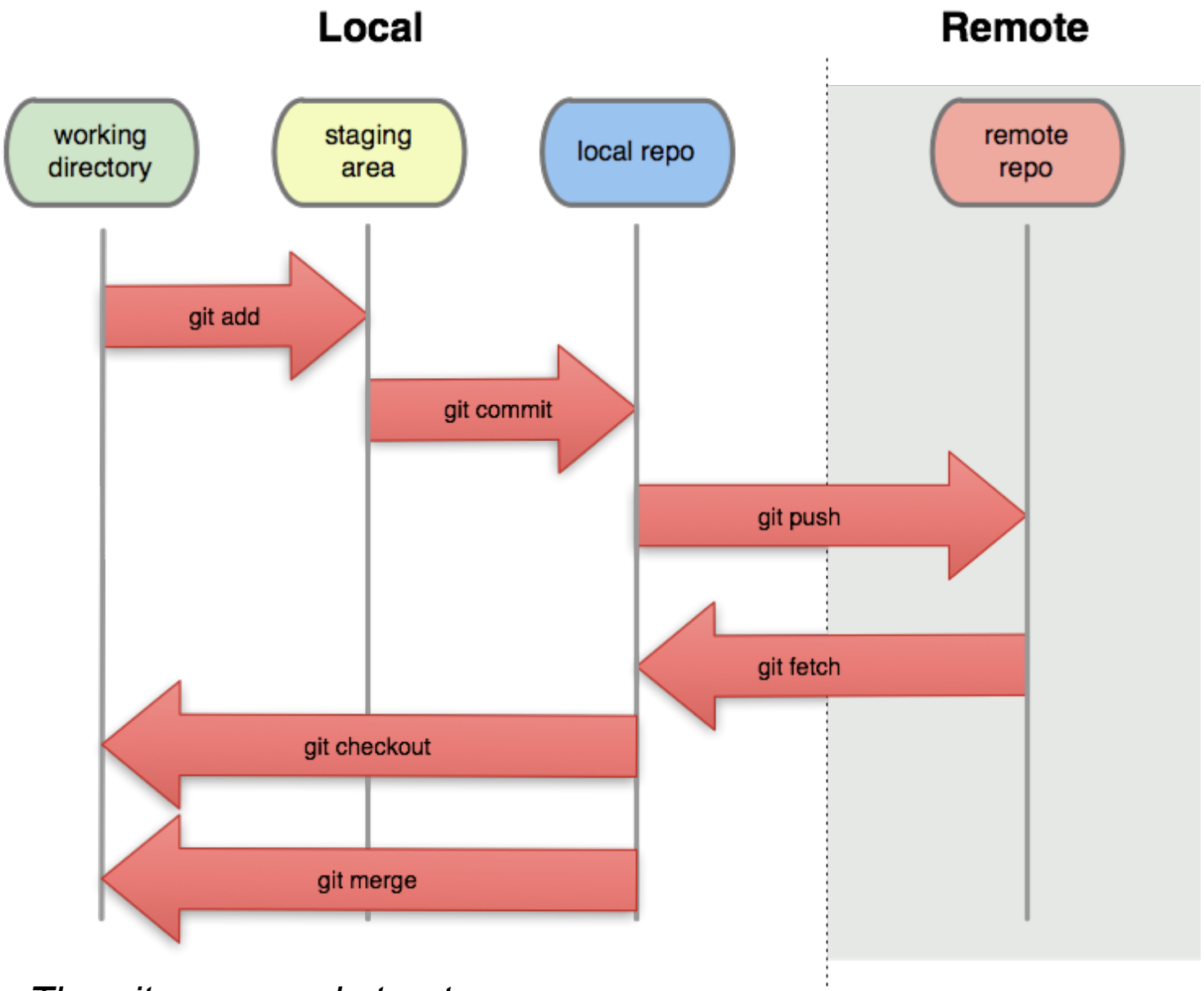
The different repositories you cloned

SourceTree


To "copy" code to and from you **local** repository
To "copy" code to and from you **remote** repository
SourceTree
File Edit View Repository Actions Tools Help
Clone / New
Commit
Checkout
Discard
Stash
Add
Remove
Add/Remove
Fetch
Pull
Push
Branch
Merge
Tag
Git Flow
Terminal
ObjectTracking C:\Users\raf
master
HCI-571-2015 C:\Users\rafa
5 2 | master
CNDE C:\Users\rafael\Works
master | 48
HCI-571-2015
CNDE
ObjectTracking
File Status
Working Copy
Branches
garrett 16
master
win_dev 9
Tags
Remotes
origin
batch_dev
garrett
mac_dev
master
TangoDev
TurbineDemo_dev
win_dev
win_dev_CUDA
All Branches
Show Remote Branches
Date Order
Graph
Description
origin/master
master
1) Add / remove the line render dynamically in debug mode to prevent OSG from cra
Remove the line renderer geometry object for the rejected lines when the drawable is empty. The application (OSG in par
Merge branch 'master' of ssh://git.vrac.iastate.edu/git/ObjectTracking
converted _threshold_angle to
demo
Removed all the message box
Two adaptations to get the ge
arameters to test the piston motor demo ar
origin/batch_dev
Use the
Fixed the camera jump manipulator issue
File loader now supports weird obj formats
Fixed auto best guess failing while the distance method is selected. Renamed setWeightMethod() to setRejectOutliersMet
simulation not the gui. Also fixed the batch mode not updating every p
d voxel down sample to calculate the average normal.
PointCloudToolkit
Completed the piston motor demo. Added a Principal Component Analysis (not integrated yet).
Added a new demo which use the piston motor. Demos are in a new folder now.
Cleaned the repository
added the project ICPTrackingLib
Current head of the local repository
Current head of the remote repository
Sorted by path
Commit: e82af8b6fdca6e1356c851919b17d62f38a972a7 [e82af8b]
Parents: 3a9a44753b, 7da3660cac
Author: Jarid Ingebrand <jarid@drwho.vrac.iastate.edu>
Date: Thursday, July 23, 2015 2:00:49 PM
Labels: origin/TurbineDemo_dev
Merge branch 'master' of ssh://git.vrac.iastate.edu/git/ObjectTracking into TurbineDemo_dev
ICP_GUI/ARKinectTrackingDemo/ARKinectD
Hunk 1 : Lines 22-28
_video = NULL;
_ar_tracking = new ICPT
_ar_tracking->enableCUD
Status of your local working copy (File Status), the local repository (Branches), and the remote repository (Remotes)

Git Command / Functions

ARLAB



The git command structure

Remote:

- your server, GitHub, Bitbucket
- Every member of the team has access

Local:

- your computer
- only you have access
- not secured on the server.

Terminology

ARLAB

- Head: the latest code version on your remote repository
- Base: the code version on the server at which you started to develop (pull)
- Working copy: the local working copy of all files you pulled.
- Staged files: files you edited and marked to commit and push to the local and remote repository.
- Branches: your local version of a remote repository. Note, git allows to start a new development branch, separated from the main branch.
- Remotes: the remote repositories.

Most favorite gotcha: remote repositories and local repositories do not need to share the same name. If you work in a team and you use different branches to develop code, keep track of the names.

Students should try the next things at home.

I usually do not explain all of the following slides.

I just browse quickly thtough all of them

ARLAB

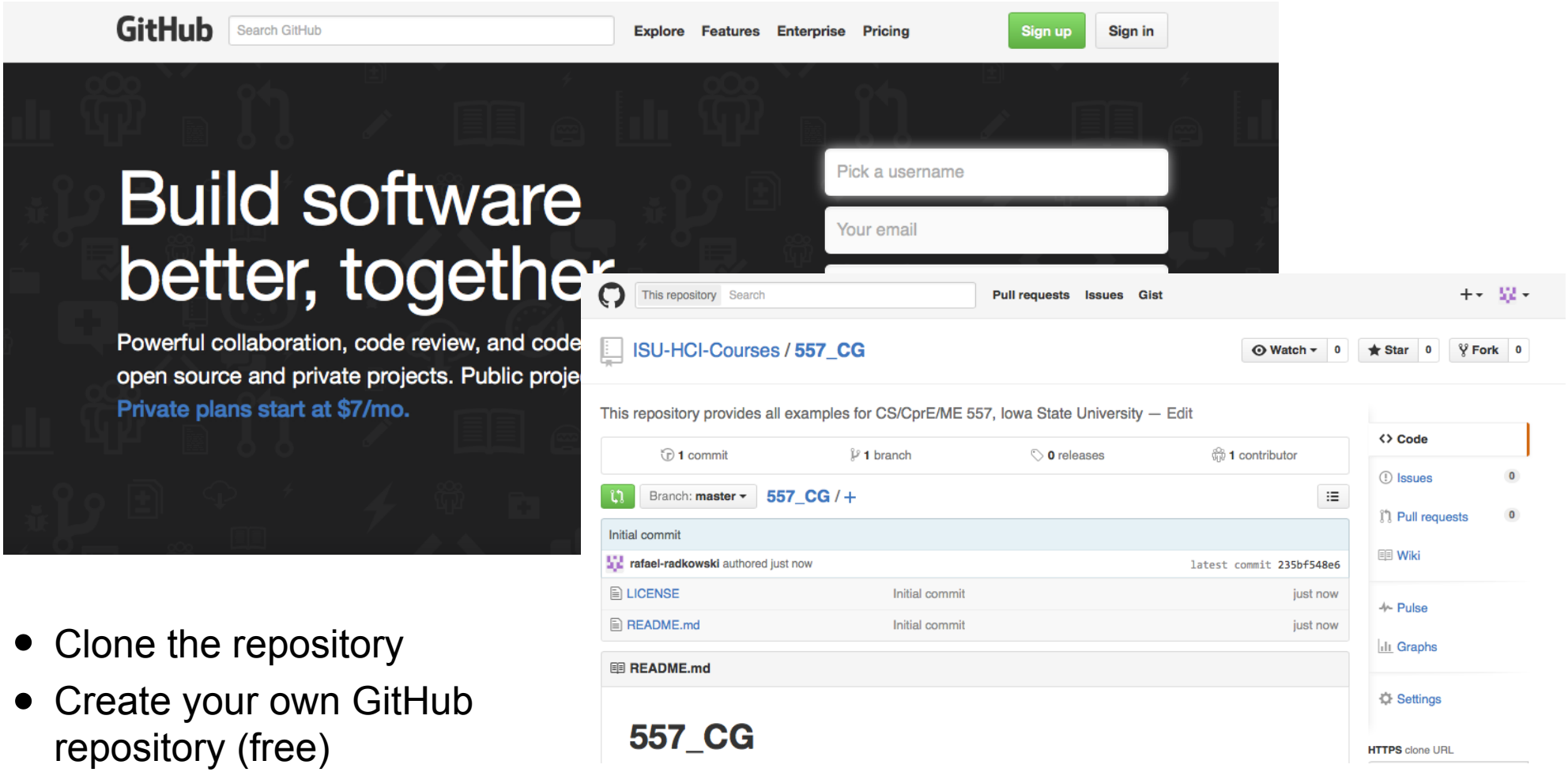
Example:
Clone the source code repository for this course

https://github.com/rafael-radkowski/HCI-557-CG.git

GitHub

All examples for this class can be cloned from a git repository hosted by GitHub

https://github.com/ISU-HCI-Courses/557_CG

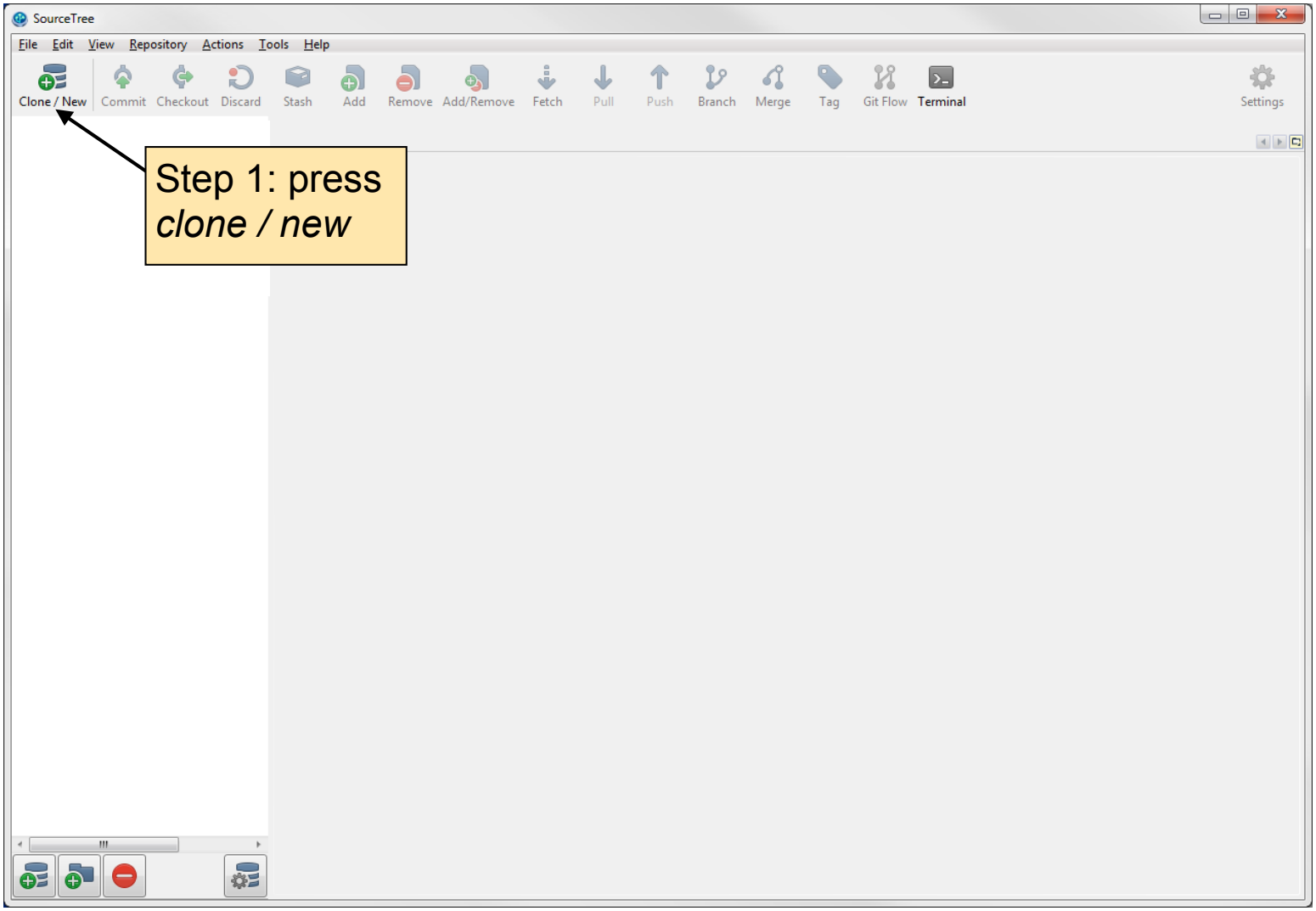


- Clone the repository
- Create your own GitHub repository (free)

Clone a Repository

ARLAB

Let's consider you need to clone a remote repository.

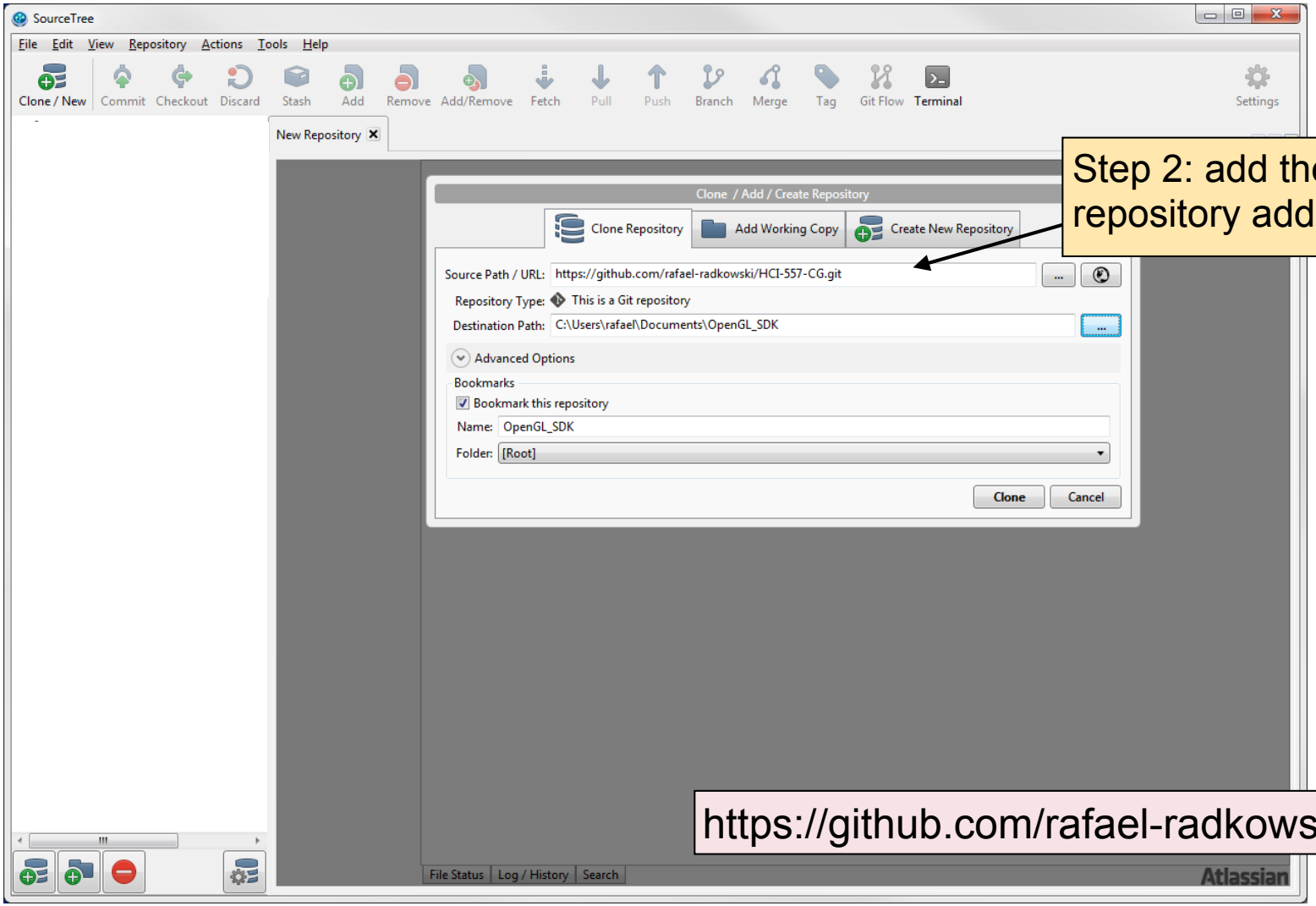


SourceTree main view

Clone a Repository

ARLAB

Let's consider you need to clone a remote repository.



Step 2: add the git-repository address

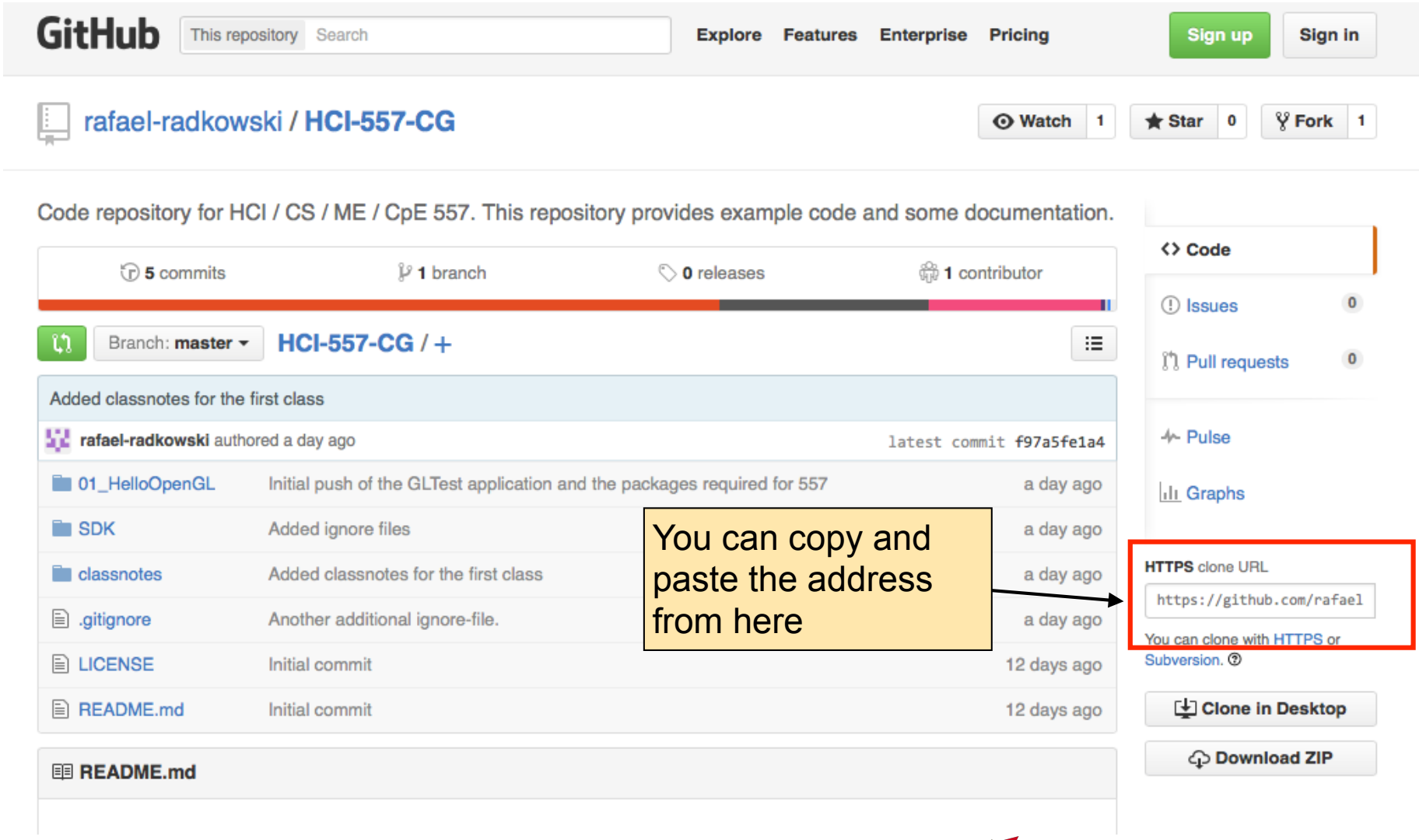
https://github.com/rafael-radkowski/HCI-557-CG.git

SourceTree main view

GitHub

https://github.com/rafael-radkowski/HCI-557-CG

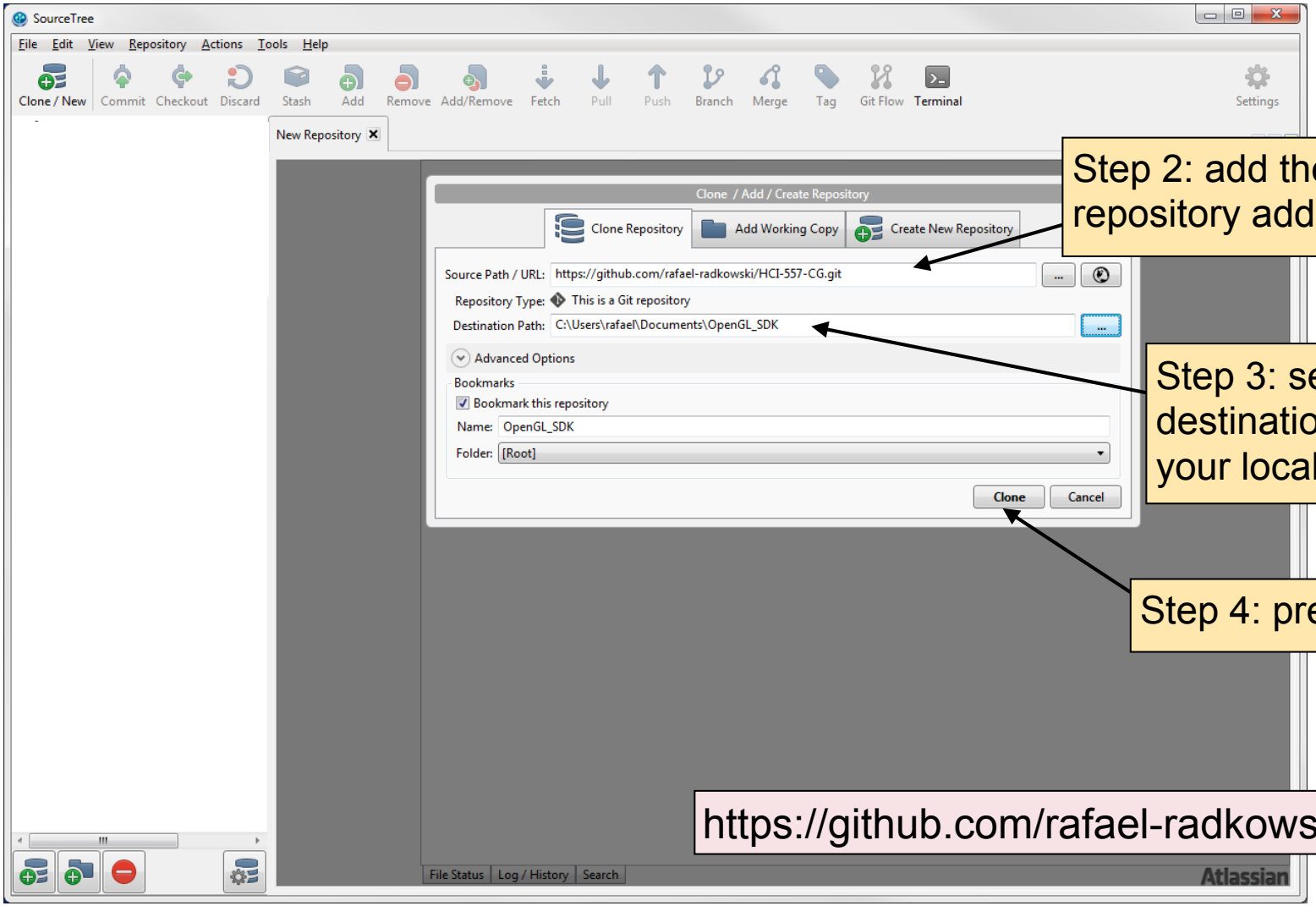
I posted a message on blackboard with the git address and the web page address.



Clone a Repository

ARLAB

Let's consider you need to clone a remote repository.



Step 2: add the git-repository address

Step 3: select a destination folder on your local hard drive.

Step 4: press *clone*

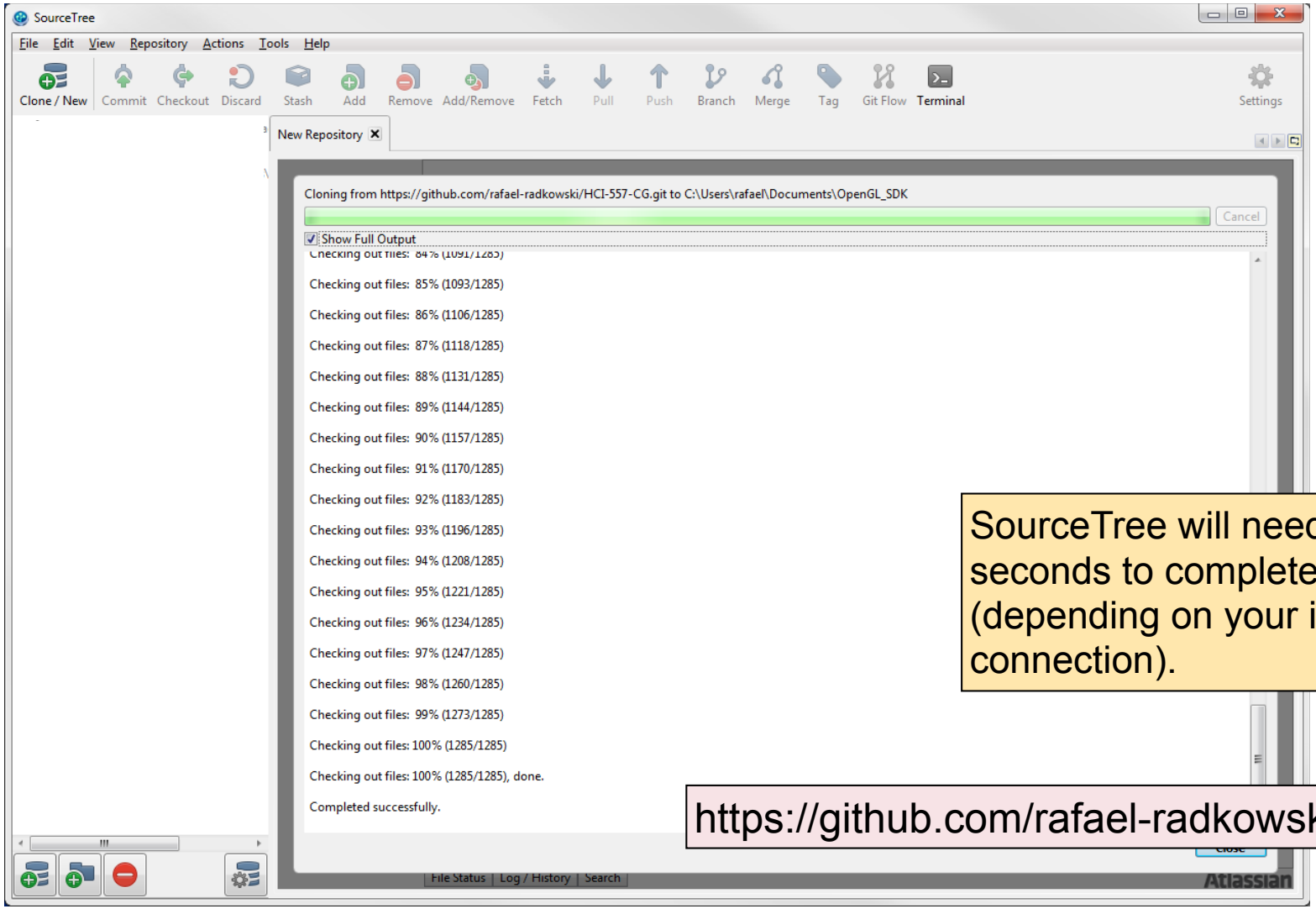
https://github.com/rafael-radkowski/HCI-557-CG.git

SourceTree main view

Clone a Repository

ARLAB

Let's consider you need to clone a remote repository.

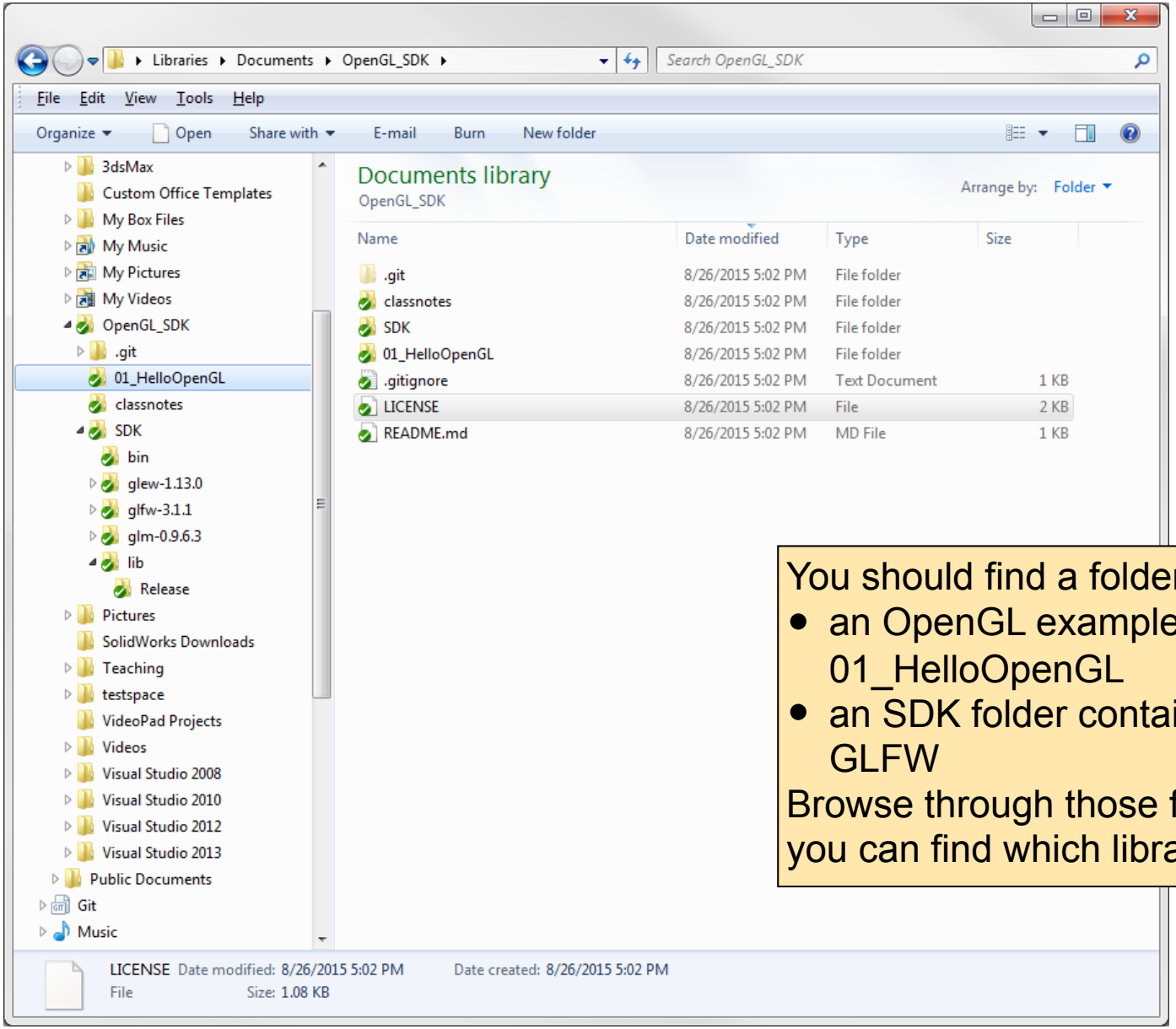


SourceTree will need a few seconds to complete this process (depending on your internet connection).

https://github.com/rafael-radkowski/HCI-557-CG.git

SourceTree main view

Clone a Repository



You should find a folder with

- an OpenGL example program 01_HelloOpenGL
- an SDK folder containing GLEW, GLM, and GLFW

Browse through those folders to learn where you can find which libraries.

Continue here

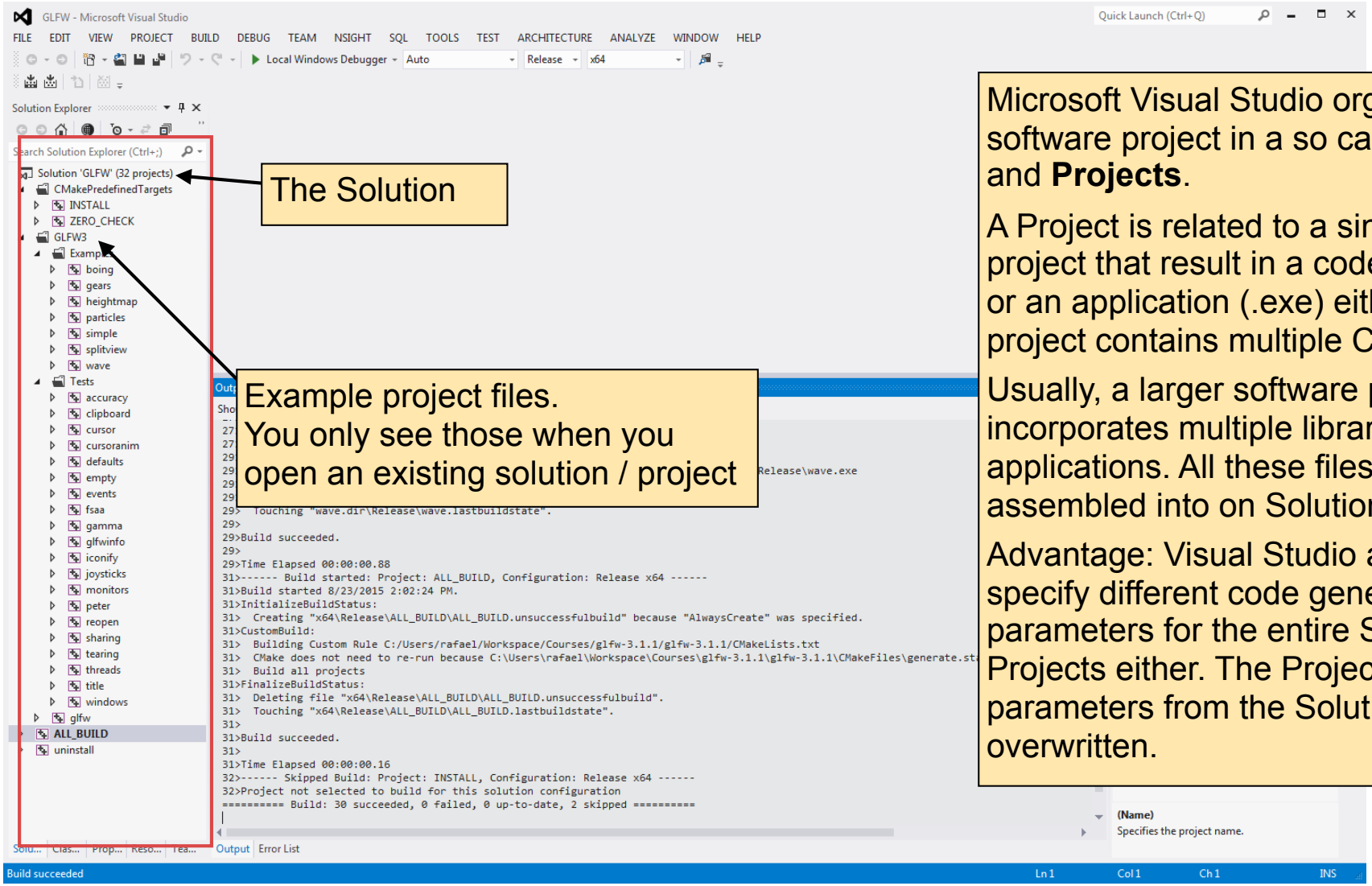
ARLAB

Visual Studio

Microsoft Visual Studio

ARLAB

MS Visual Studio organization



Microsoft Visual Studio organize a software project in a so called **Solution** and **Projects**.

A Project is related to a single code project that result in a code library (.lib) or an application (.exe) either. Each project contains multiple C/C++ files.

Usually, a larger software project incorporates multiple library files and applications. All these files are assembled into on Solution.

Advantage: Visual Studio allows to specify different code generation parameters for the entire Solution or the Projects either. The Projects inherit the parameters from the Solution if not overwritten.

Microsoft Visual Studio main view

The distinction on the next slide is imperative

Many students had problems with this in the past, to determine when to use git and when to use the "manual" project set up.

Point this out.

I usually tell the student that I explain this in detail so they can work with compiler error messages and can distinguish linker errors from compiler errors.

Solution / Project Creation

ARLAB

Before you can start to develop code, you have to create a solution along with a project.

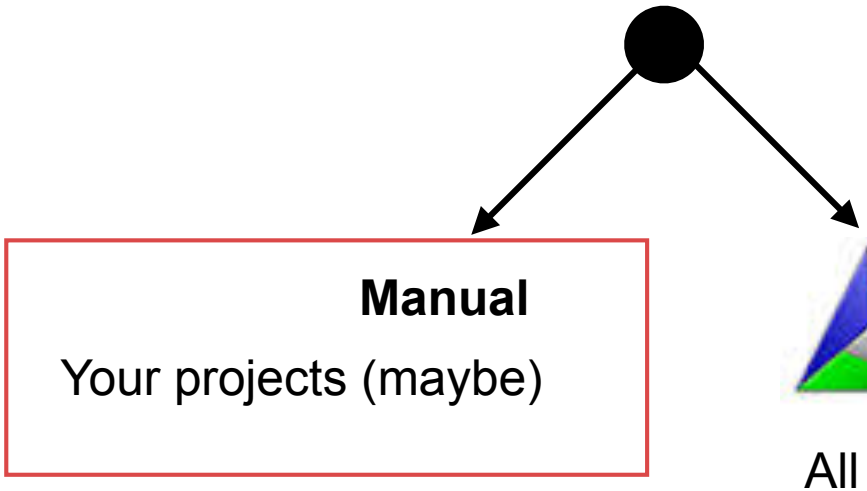
Two ways:

Starting point:

I would like to create a project

or

create a solution / project for existing code.



Manual

Your projects (maybe)

CMake

All 557 examples provide cmake files!

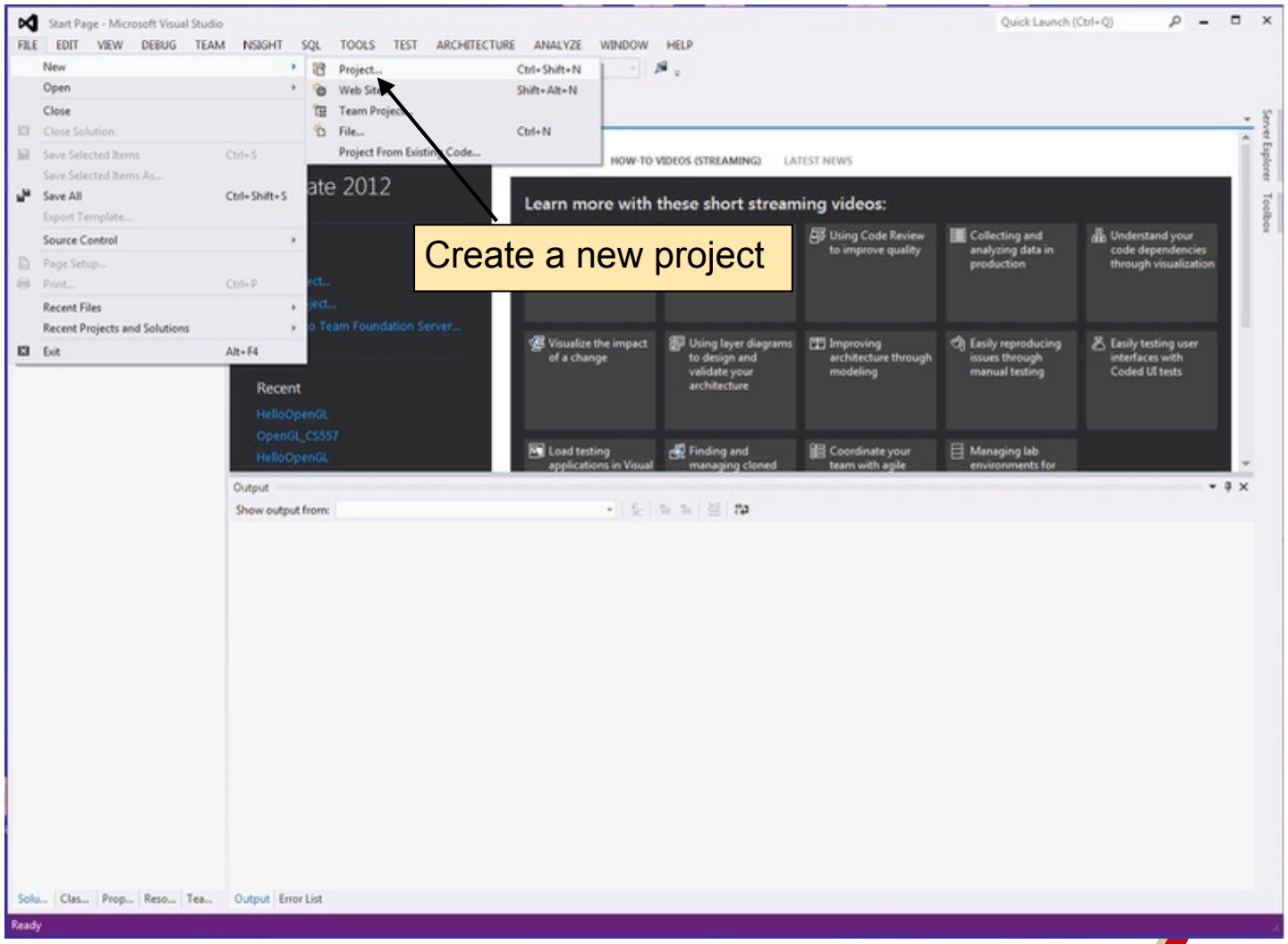
ARLAB

Next slides:
Manual preparation of an "empty" project

Create a Project

ARLAB

1. Go to File -> New -> Project to create a new code project. The solution is created automatically.



Microsoft Visual Studio main view

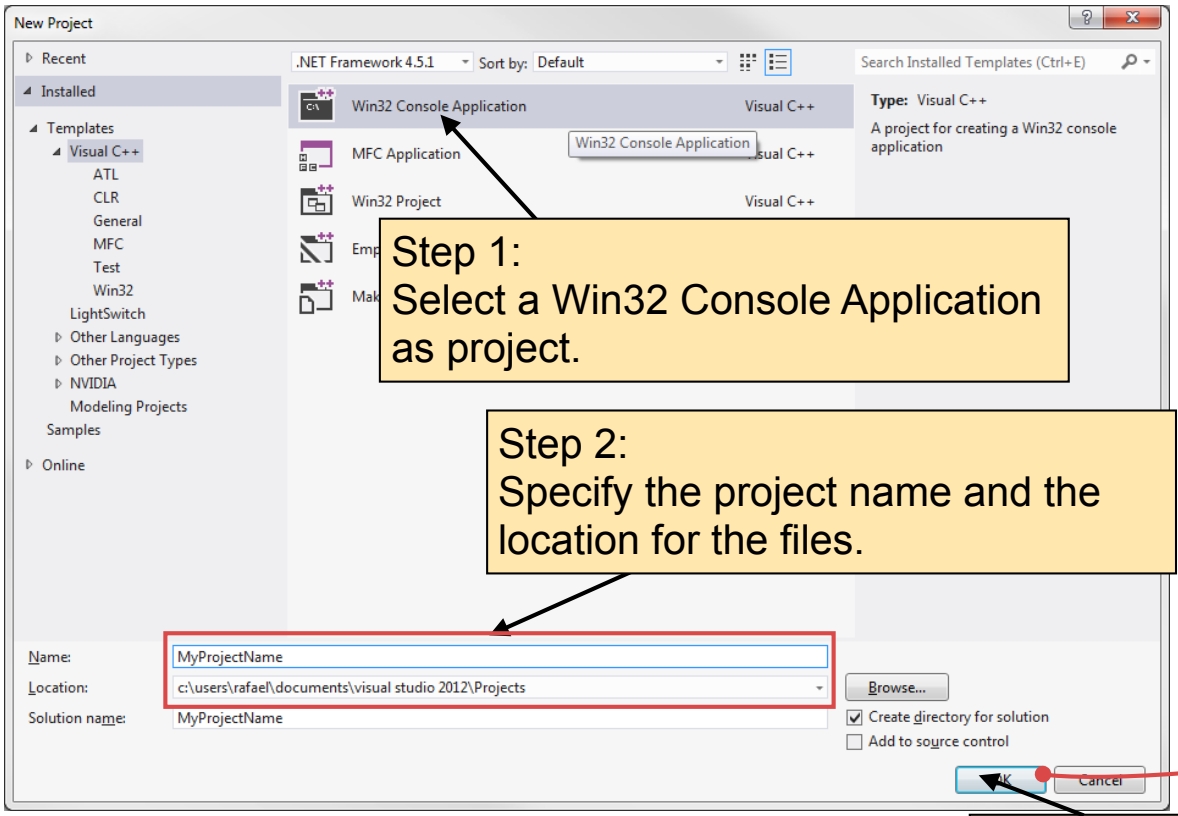
VRAC|HCI

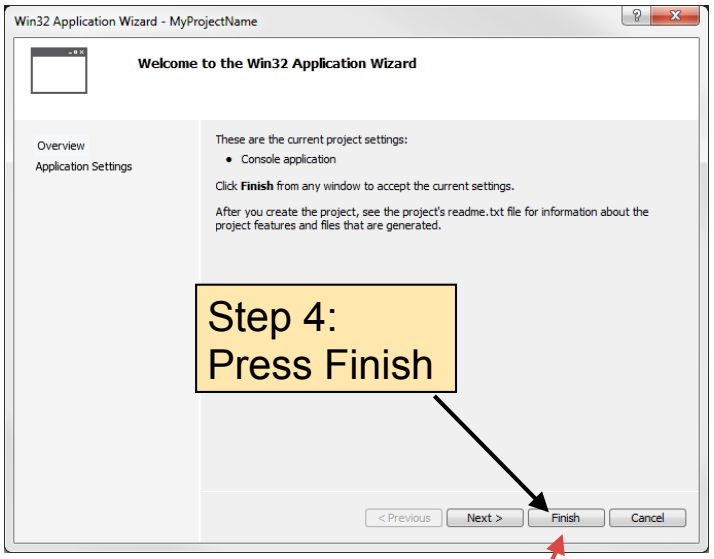
IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Create a Project

ARLAB

1. A project selection window pops up.
   Step 1: Select a Visual C++ -> Win32 -> Win32 Console Application.
   Step 2: Specify a project name and a project location.





A new window opens after OK was pressed.

Microsoft Visual Studio project selection window

Project Preparation

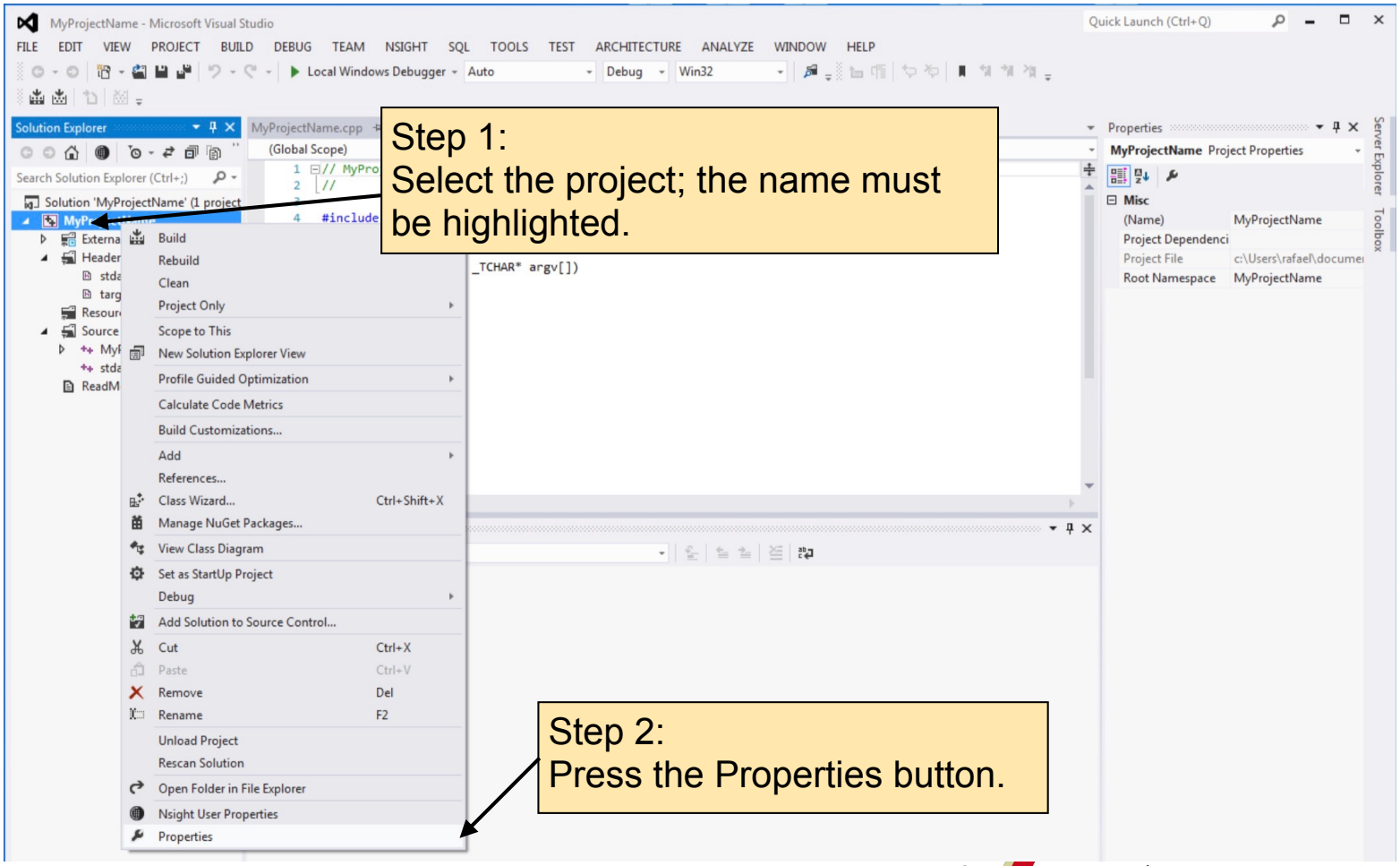
ARLAB

2

The project must know the libraries and header files you like to use

Step 1: Select the solution (highlighted)

Step 2: Press the Properties button



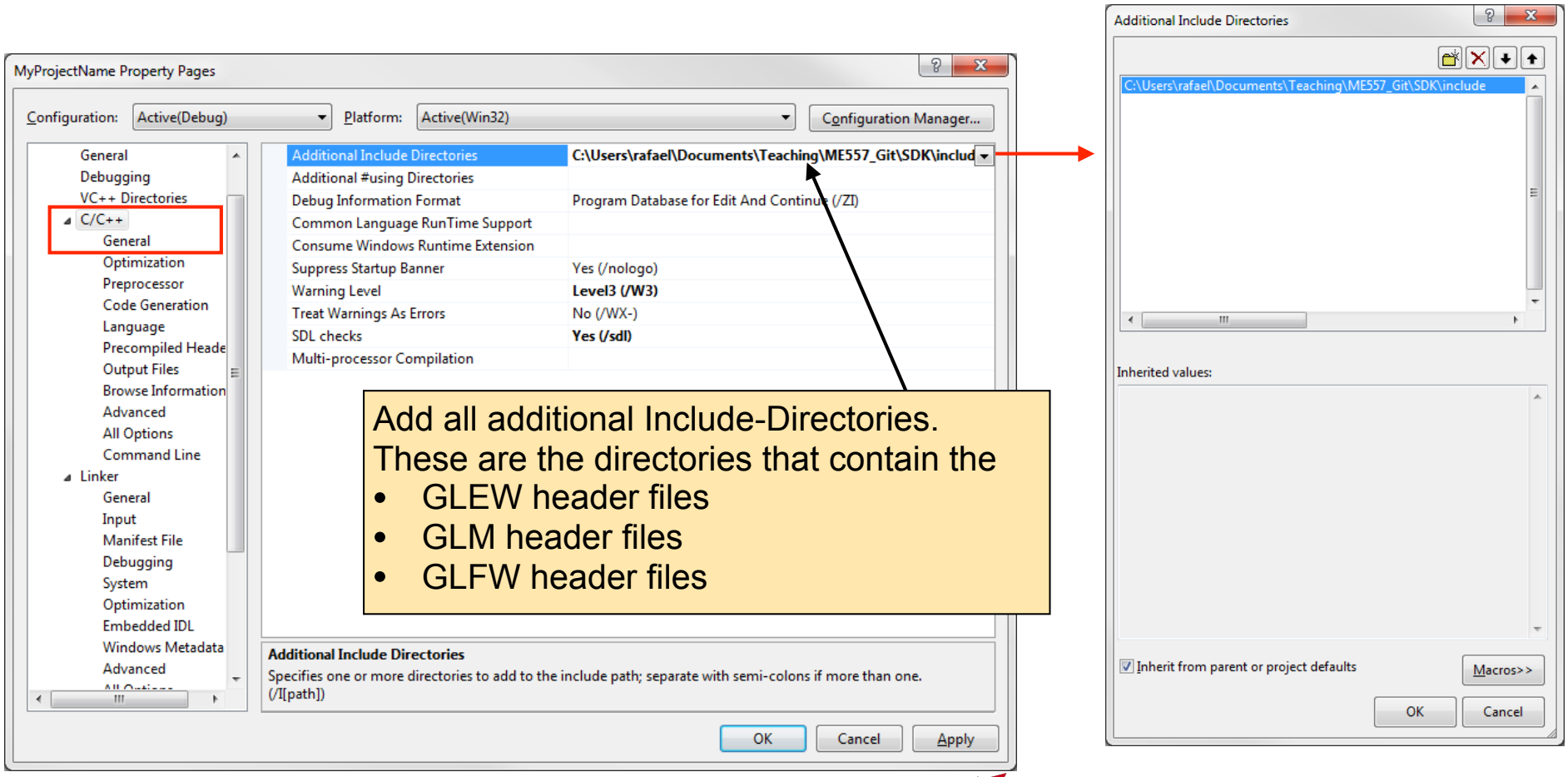
Microsoft Visual Studio main view

Prepare the Compiler

ARLAB

2. Set up the compiler means, tell the compiler where to find the header files of the framework you use (GLEW, GLM, GLFW in our case).

   Go to: *Configuration Properties -> C/C++ -> General* and add the directory.



Microsoft Visual Studio - Project Property Window

VRAC | HCI

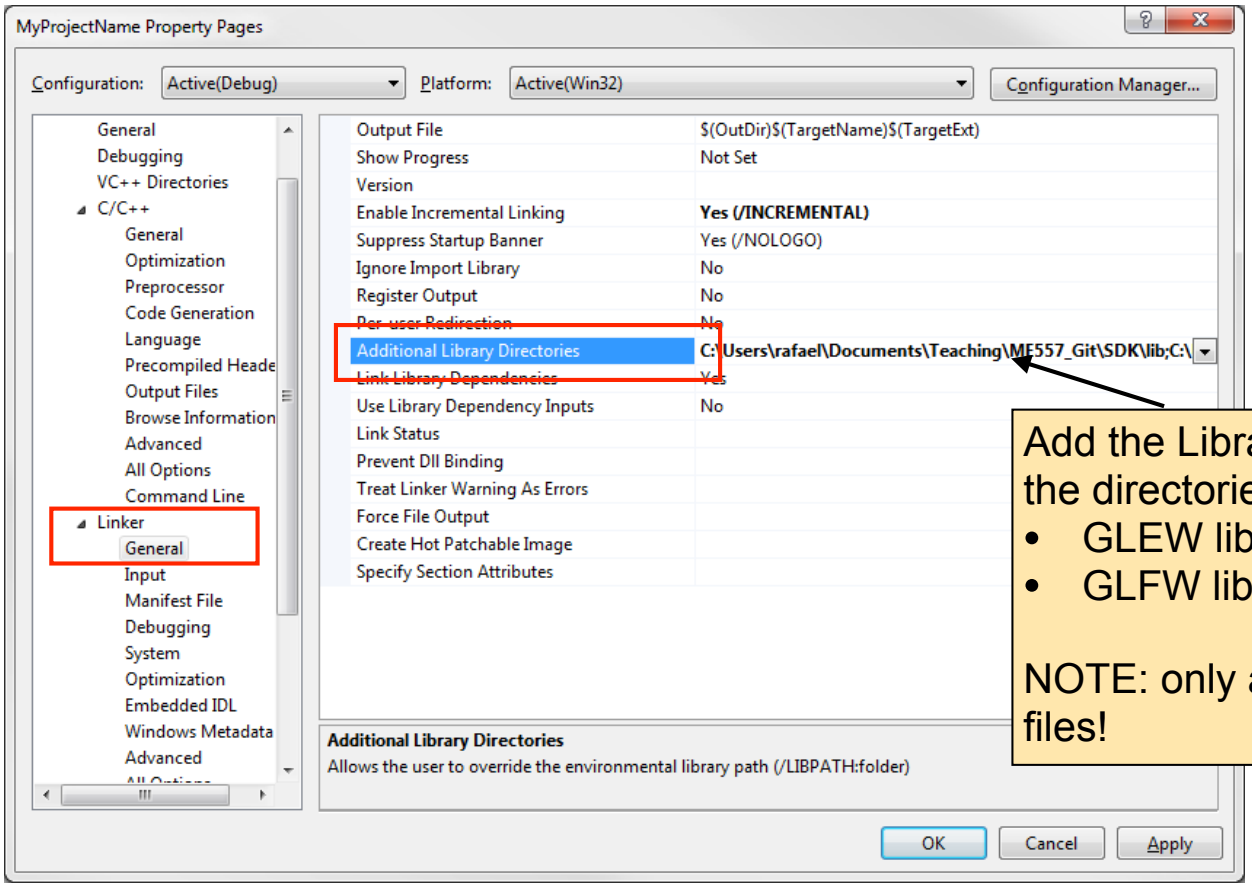
IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Setup the Linker

ARLAB

3 Setup the linker means, tell the linker where to find the library files and which library files you want to use.

Go to: *Configuration Properties -> Linker -> General* and add the directory.



Add the Library directories. These are the directories that contain the

- GLEW library file glew32.lib
- GLFW library file glfw.lib

NOTE: only add the path and not the files!

Microsoft Visual Studio - Project Property Window

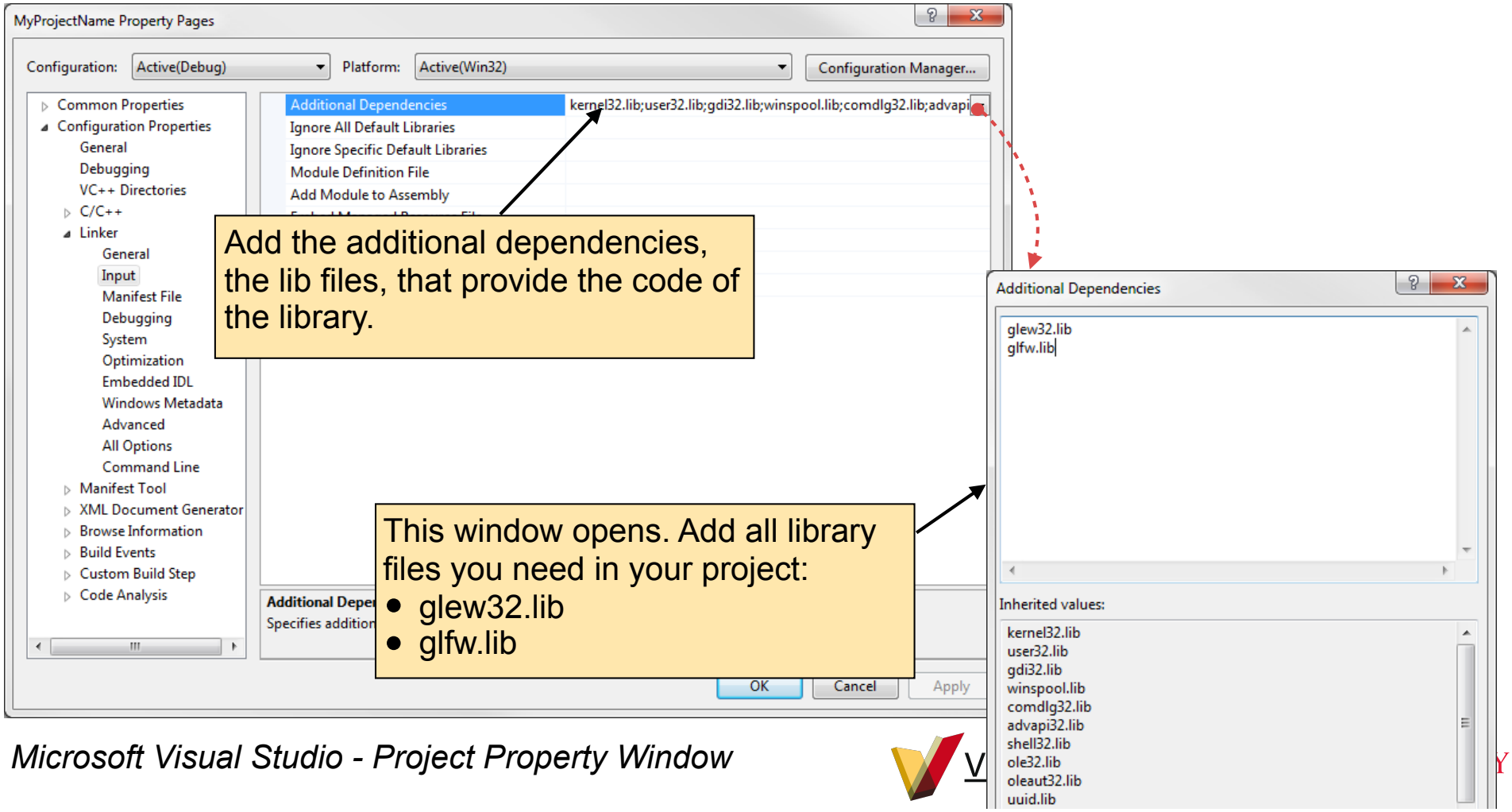
VRAC|HCI IOWA STATE UNIVERSITY OF SCIENCE AND TECHNOLOGY

Add Libraries

ARLAB

3 Setup the linker means, tell the linker where to find the library files and which library files you want to use.

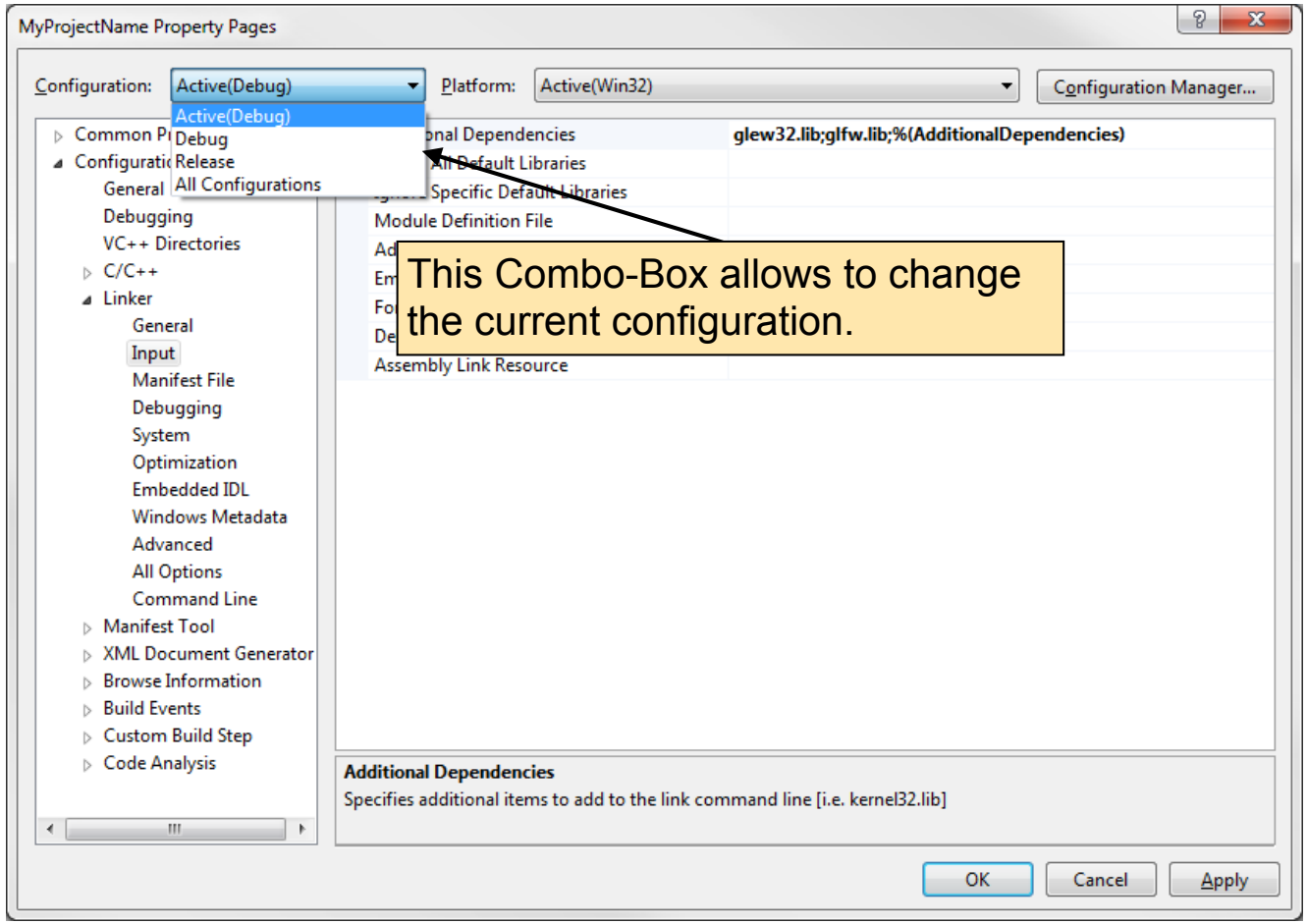
Go to: *Configuration Properties -> Linker -> General* and add the directory.



Microsoft Visual Studio - Project Property Window

Debug and Release

There are different configurations you can generate an executable. **Debug** and **Release** are the default configurations. You need to repeat those settings for all configurations.



Microsoft Visual Studio - Project Property Window

Debug:

To compiler will add an additional logical layer between your code and the machine code to allow you to retrieve memory information during runtime.

Debug versions are always slow!!!

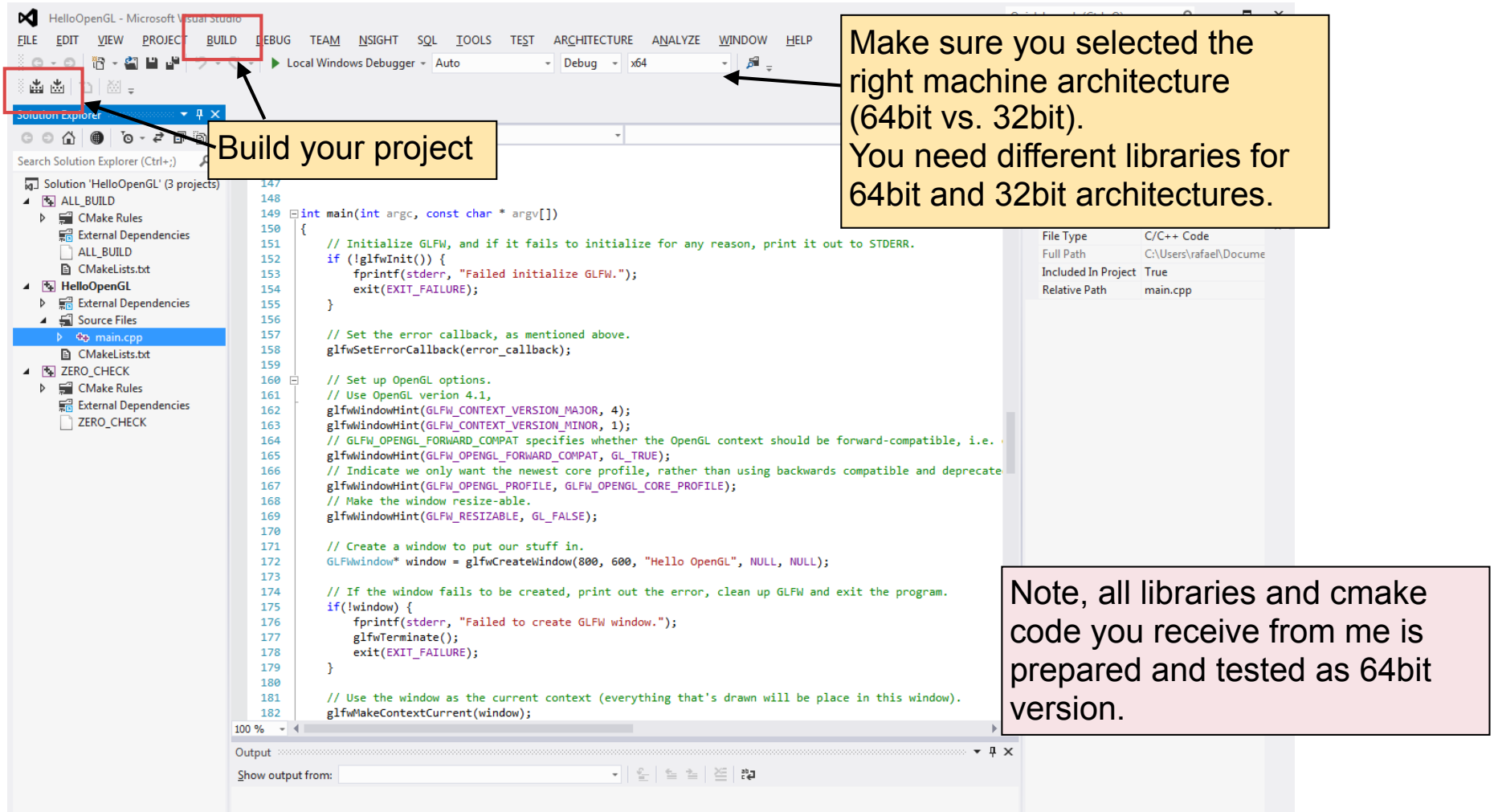
Release:

The code is smaller and faster but you cannot read any variable information from your memory.

Add some Code

ARLAB

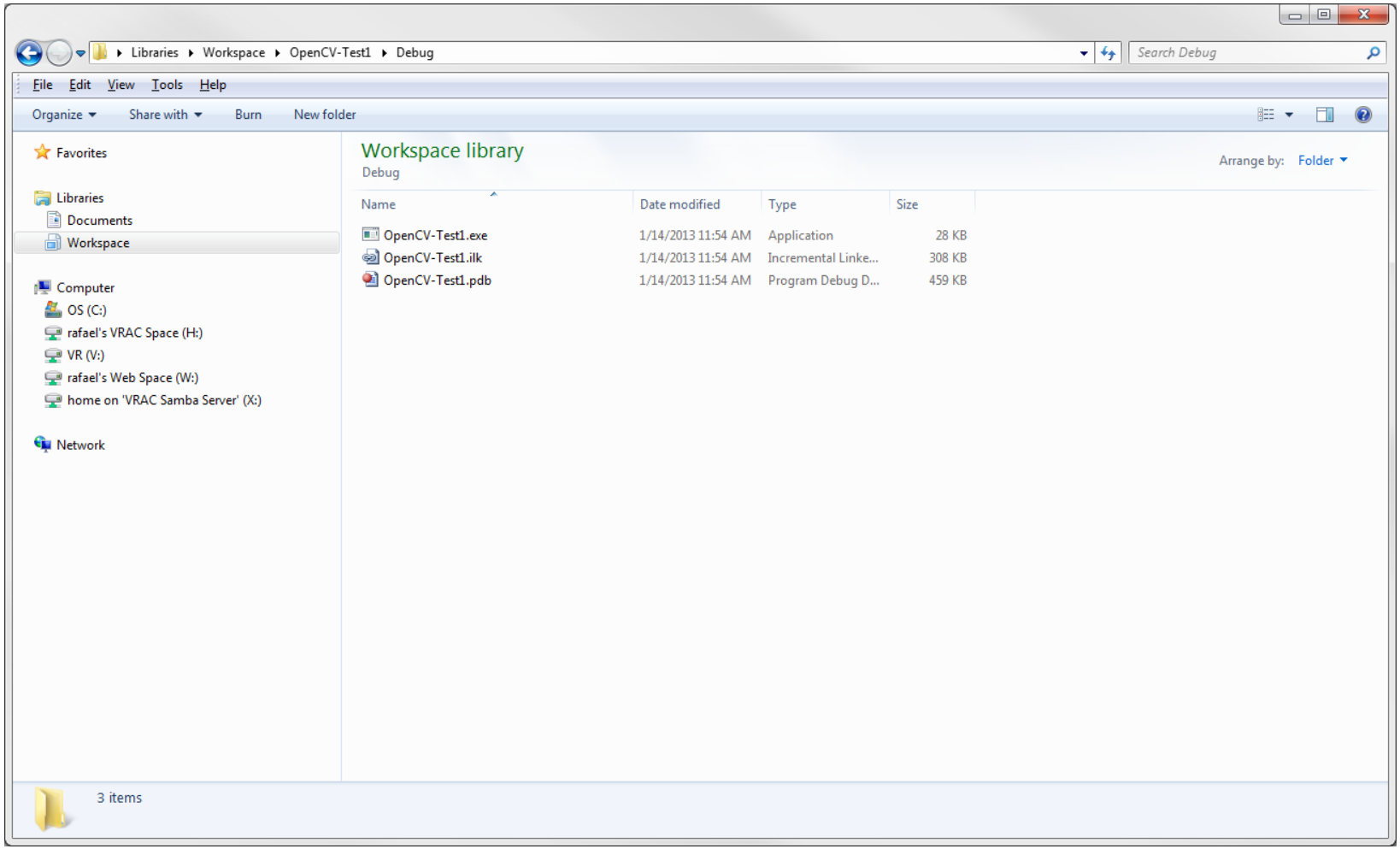
4 Add the example code to your program file and build the code.



Microsoft Visual Studio main view

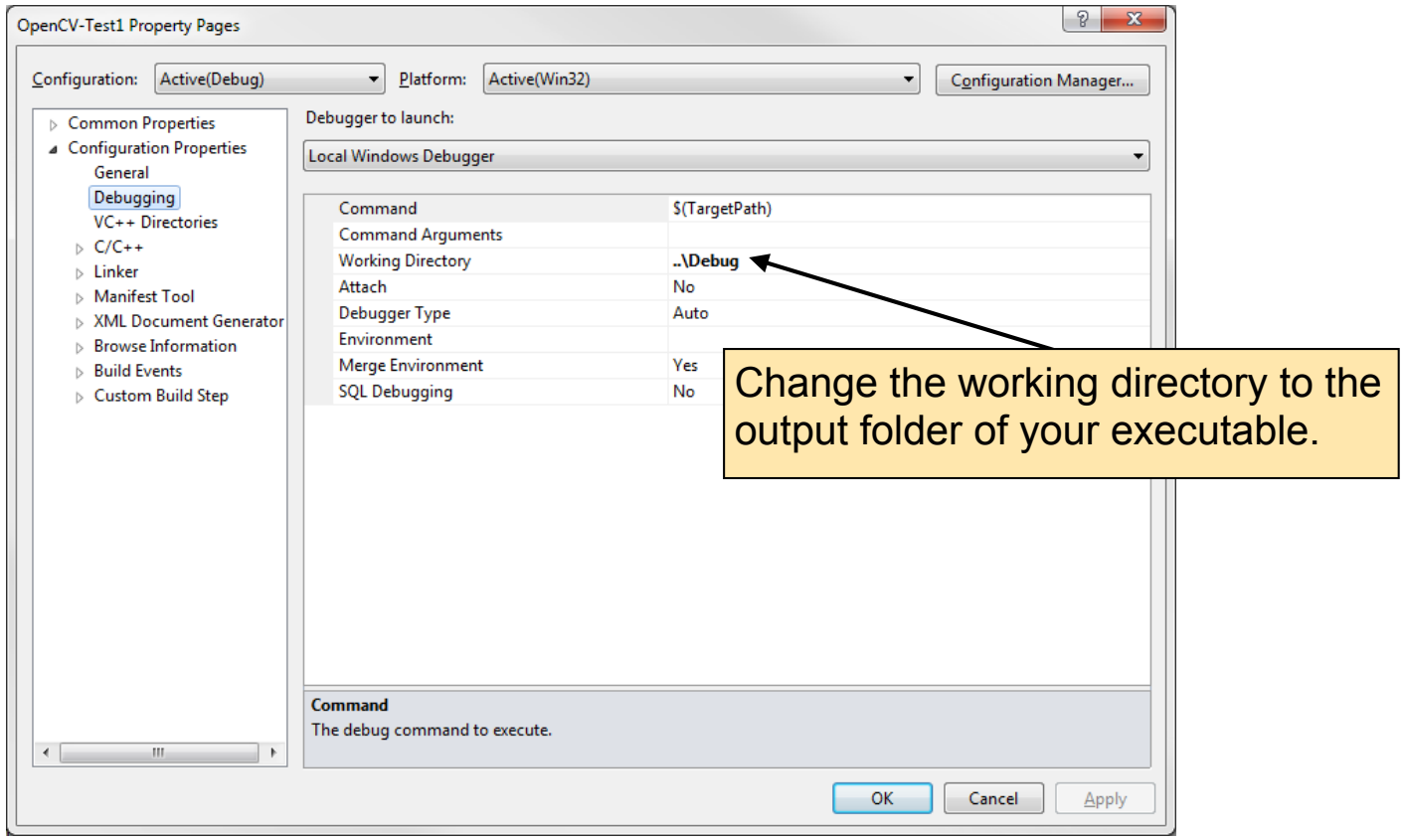
Output

The output of the build process is an *.exe file, where * is the project name by default.



Working Directory

The working directory defines the path from which MS Visual Studio start your executable. Specify the **executable output directory** as working path. By default, MS Visual Studio uses the solution path. This is not helpful because - by default- the working path is the executable path when you start a program from the window explorer. Changing the path restores the usual Windows behavior.



Microsoft Visual Studio - Project Property Window

ARLAB

CMake

Solution / Project Creation

ARLAB

Before you can start to develop code, you have to create a solution along with a project.

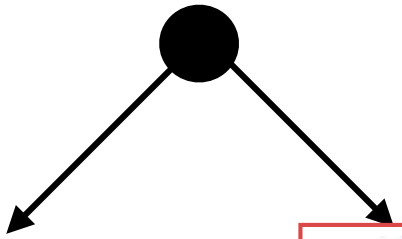
Two ways:

Starting point:

I would like to create a project

or

create a solution / project for existing code.



Manual

Your projects (maybe)

CMake

All 557 examples provide cmake files!

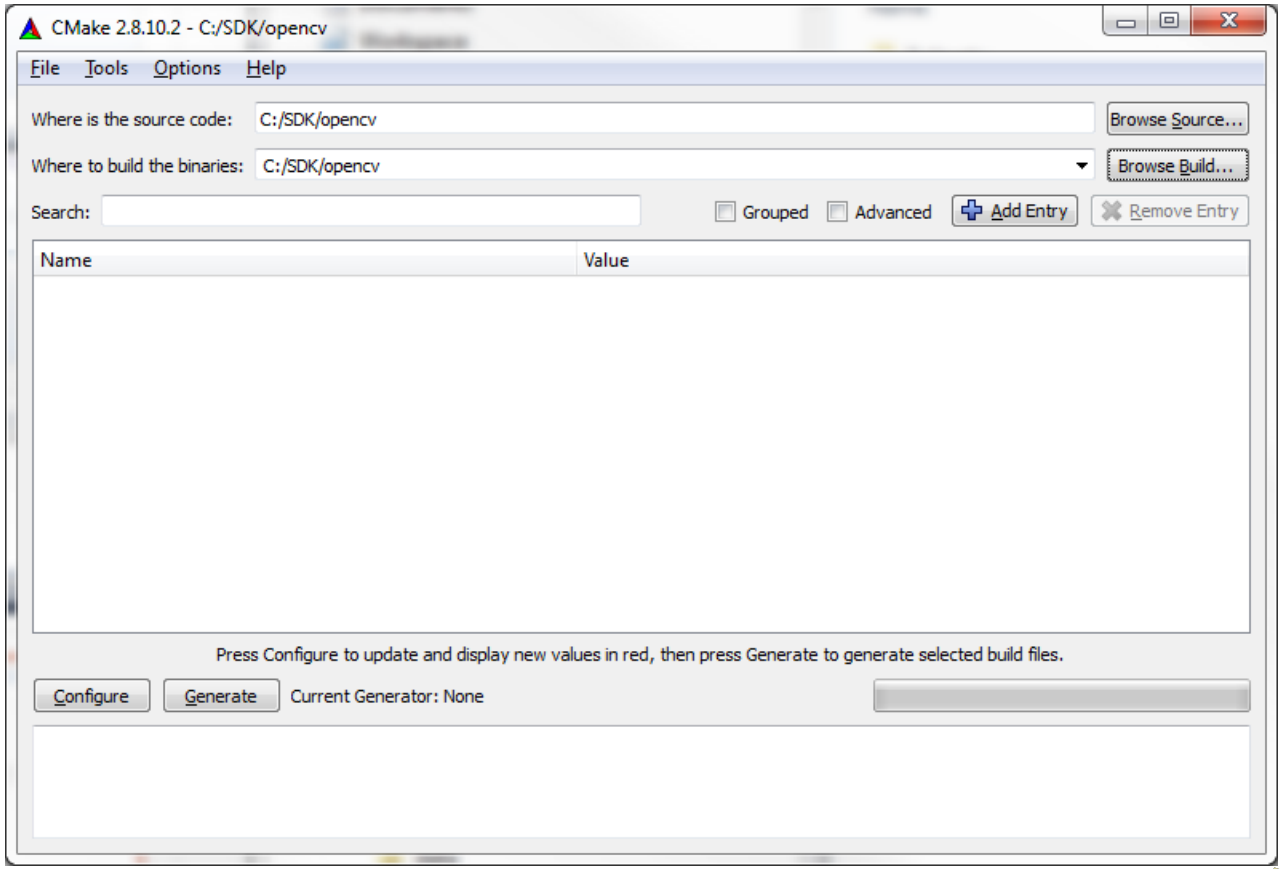
ARLAB

CMake
"Automatic" preparation of a project file / solution

CMake

ARLAB

CMake is a cross-platform, open-source build system, designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files. CMake generates native makefiles and workspaces that can be used in the compiler environment of your choice.



Main Window of CMake

Download the latest version!

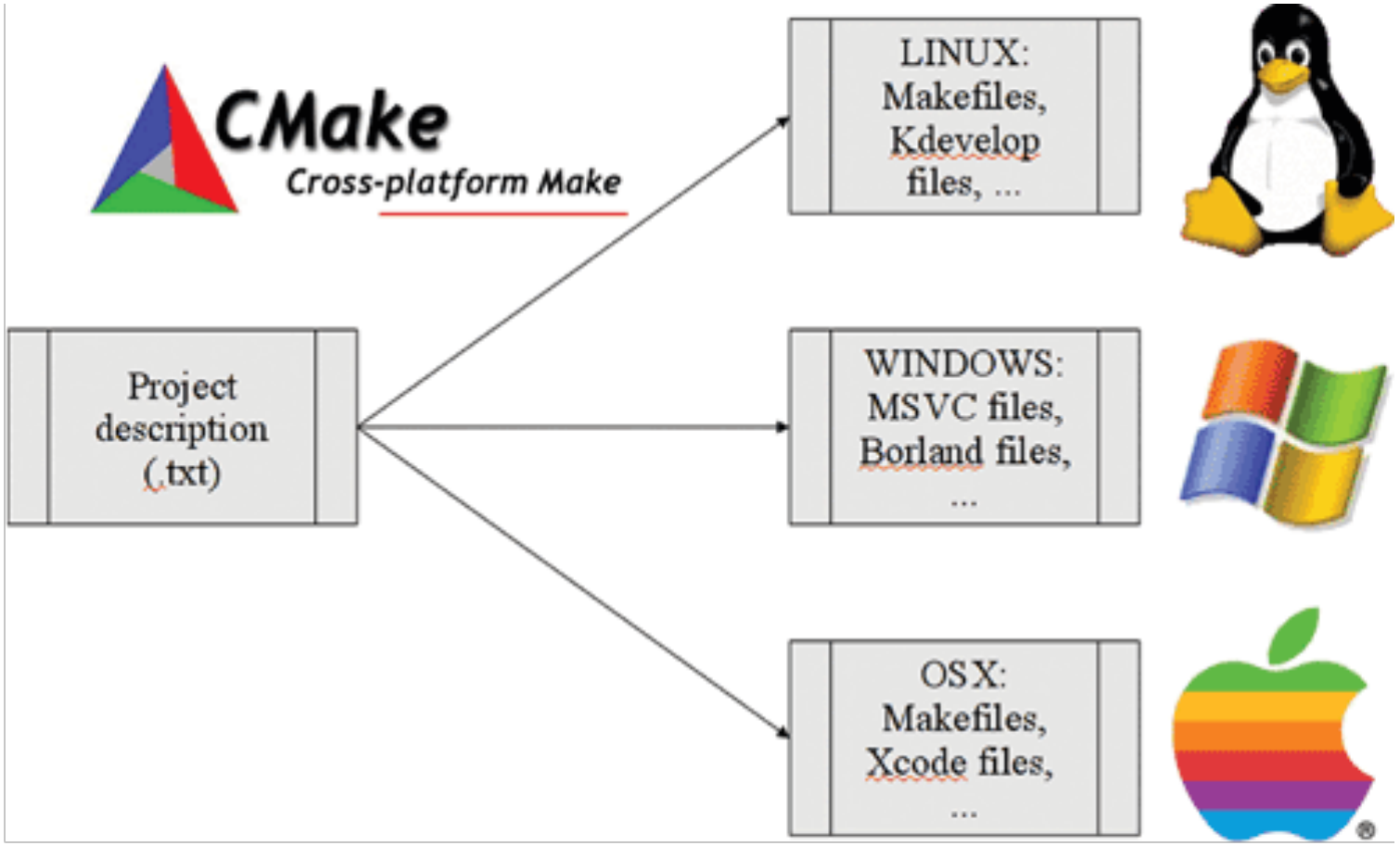
http://www.cmake.org

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

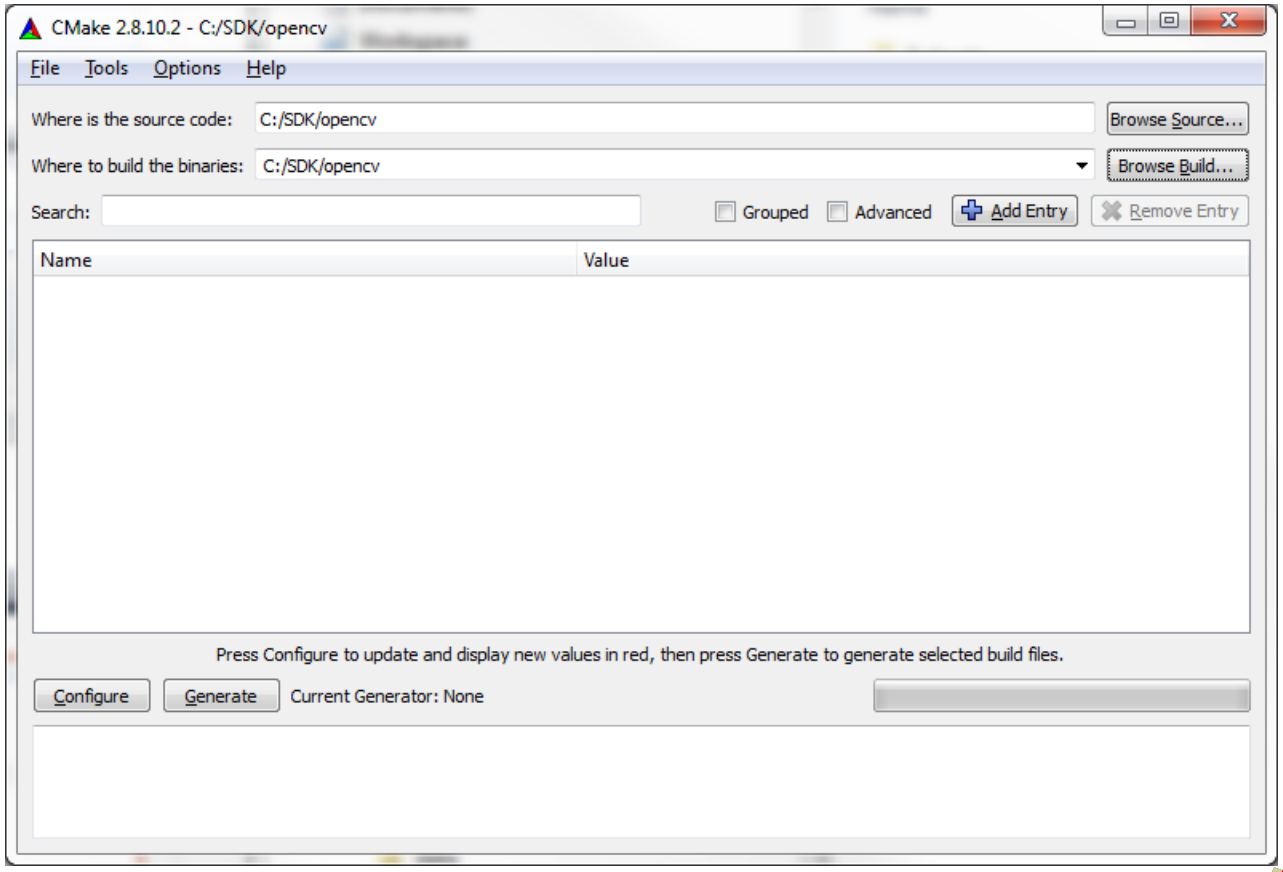
CMake Basic Idea

ARLAB



CMake

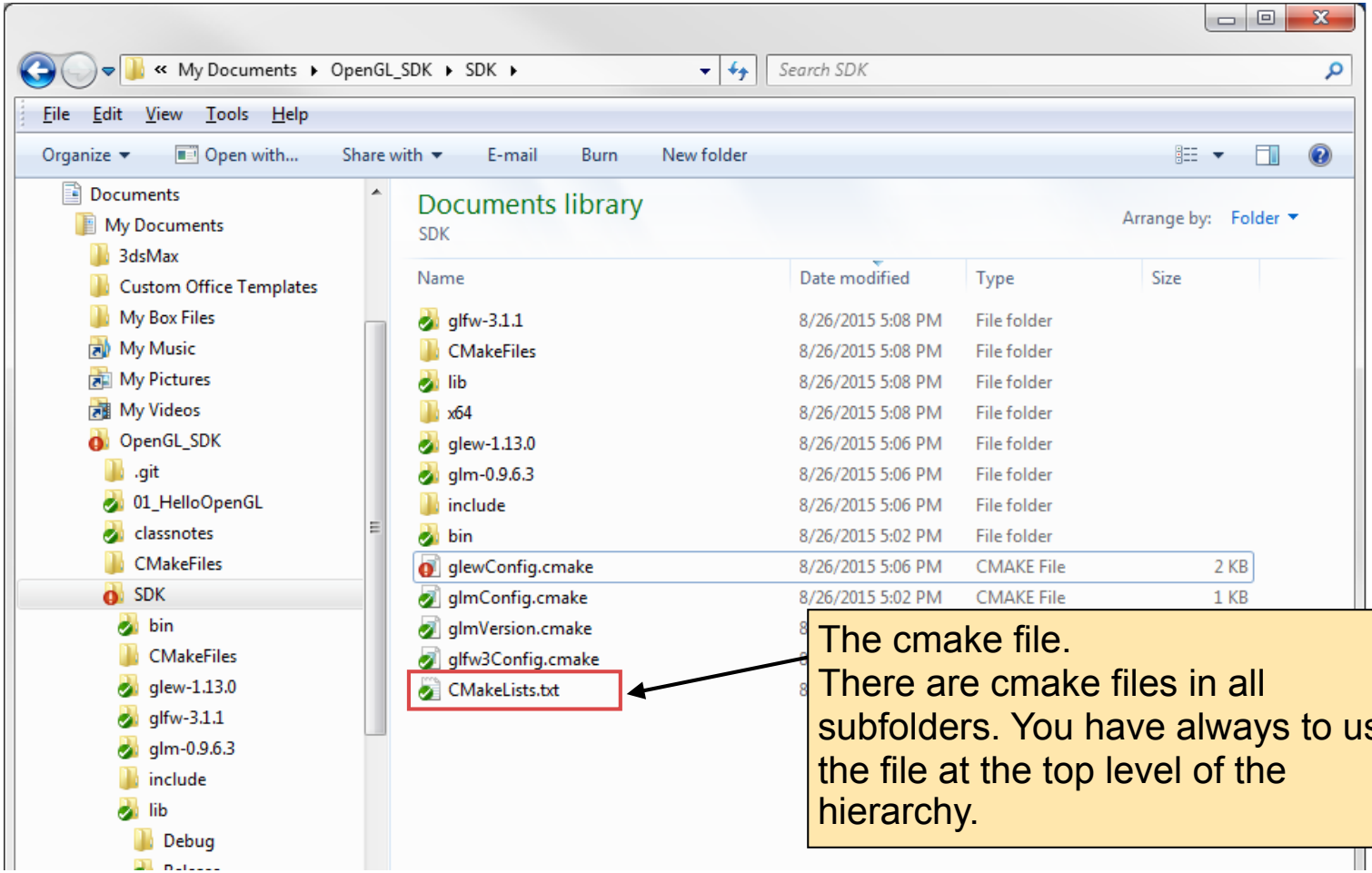
CMake is a cross-platform, open-source build system, designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files. CMake generates native makefiles and workspaces that can be used in the compiler environment of your choice.



Main Window of CMake

CMake Project Description

A CMake project is always defined in one or several files with the filename **CMakeLists.txt**. A code project solution / project file can be generate with CMake when this file is available.



CMake Project Files

ARLAB

H:\Teaching\02_Fall 2013\ME 557\10_Virtual_Trackball\CMakeLists.txt - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

CMakeLists.txt

```
1  # Main cmake file
2
3  cmake_minimum_required(VERSION 2.6)
4
5  # Main project name is ARMaker
6  project(Virtual_Trackball)
7
8  # Find packages
9  FIND_PACKAGE(OpenGL)
10 FIND_PACKAGE(GLUT)
11
12 # Include dirs
13 include_directories(${OPENGL_INCLUDE_DIR})
14 include_directories(${GLUT_INCLUDE_DIR})
15
16
17 # Add all files to the configuration
18 file(GLOB Virtual_Trackball_SRC
19     "*.h"
20     "*.cpp"
21 )
22
23 # Create an executable
24 add_executable( Virtual_Trackball ${Virtual_Trackball_SRC} )
25
26
27 # Add libraries
28 if(WIN32 AND NOT APPLE)
29     target_link_libraries(Virtual_Trackball ${OPENGL_LIBRARIES} ${GLUT_LIBRARIES} )
30 else()
31     target_link_libraries(Virtual_Trackball ${OPENGL_LIBRARIES} ${GLUT_LIBRARIES} )
32 endif()
33
34 # Set the project architecture for this example
35 set( PROJECT_ARCH "x86" )
36
```

CMAKEFILE length : 752 lines : 36 Ln : 1 Col : 1 Sel : 0 | 0 UNIX ANSI as UTF-8 INS

Project name

Programming SDKs

Include directories

Code files. In this case: all files

Adds an executable to a project

Add libraries and library paths.

Example CMakeLists.txt file: the file contains script commands that tell CMake how to setup the project

Not part of this course!

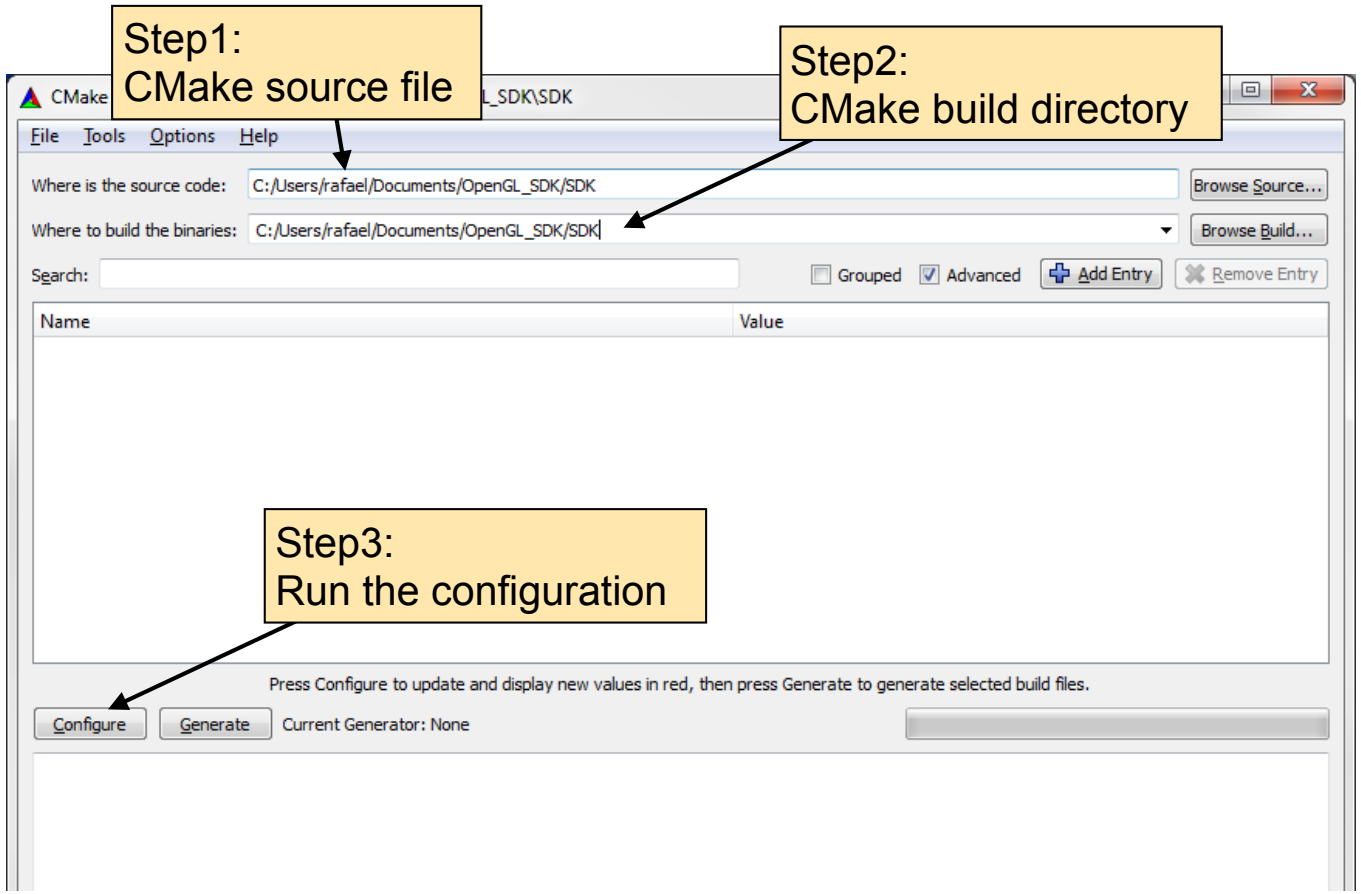
Just to get you an idea about the content when you open those files!

Create OpenCV project files (1/2)

Step 1: set the source code file; the directory containing the file *CMakeList.txt*.

Step 2: set the build directory; take the same.

Step 3: run the configuration by pressing the *Configure*-Button two times (the second times you press it, all error will be highlighted red).

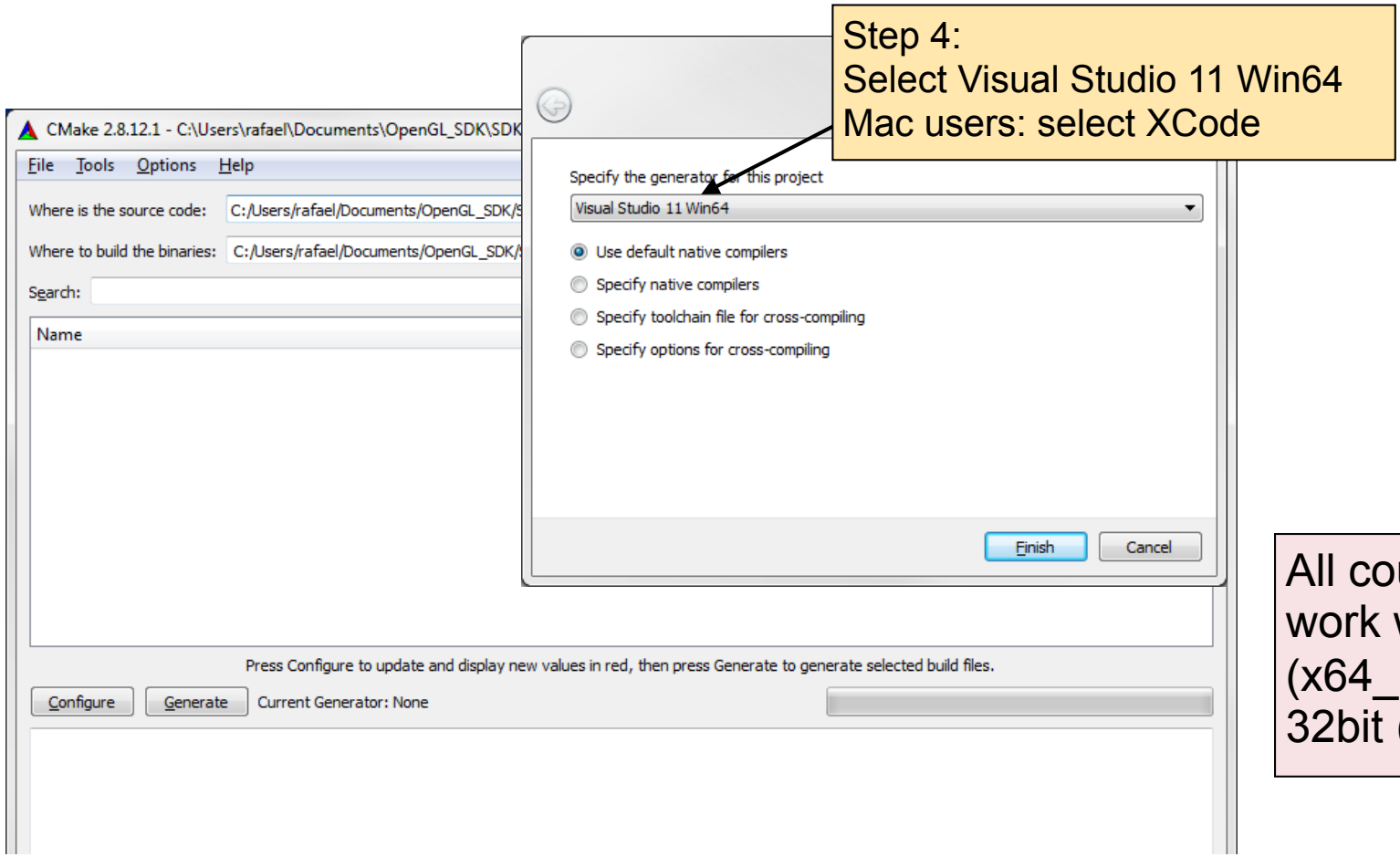


Main Window of CMake

Create OpenCV project files (1/2)

Step 4: specify the generator for this project; select ***Visual Studio 11 Win64***.

Note: Visual Studio 2012 is Visual Studio Version 11



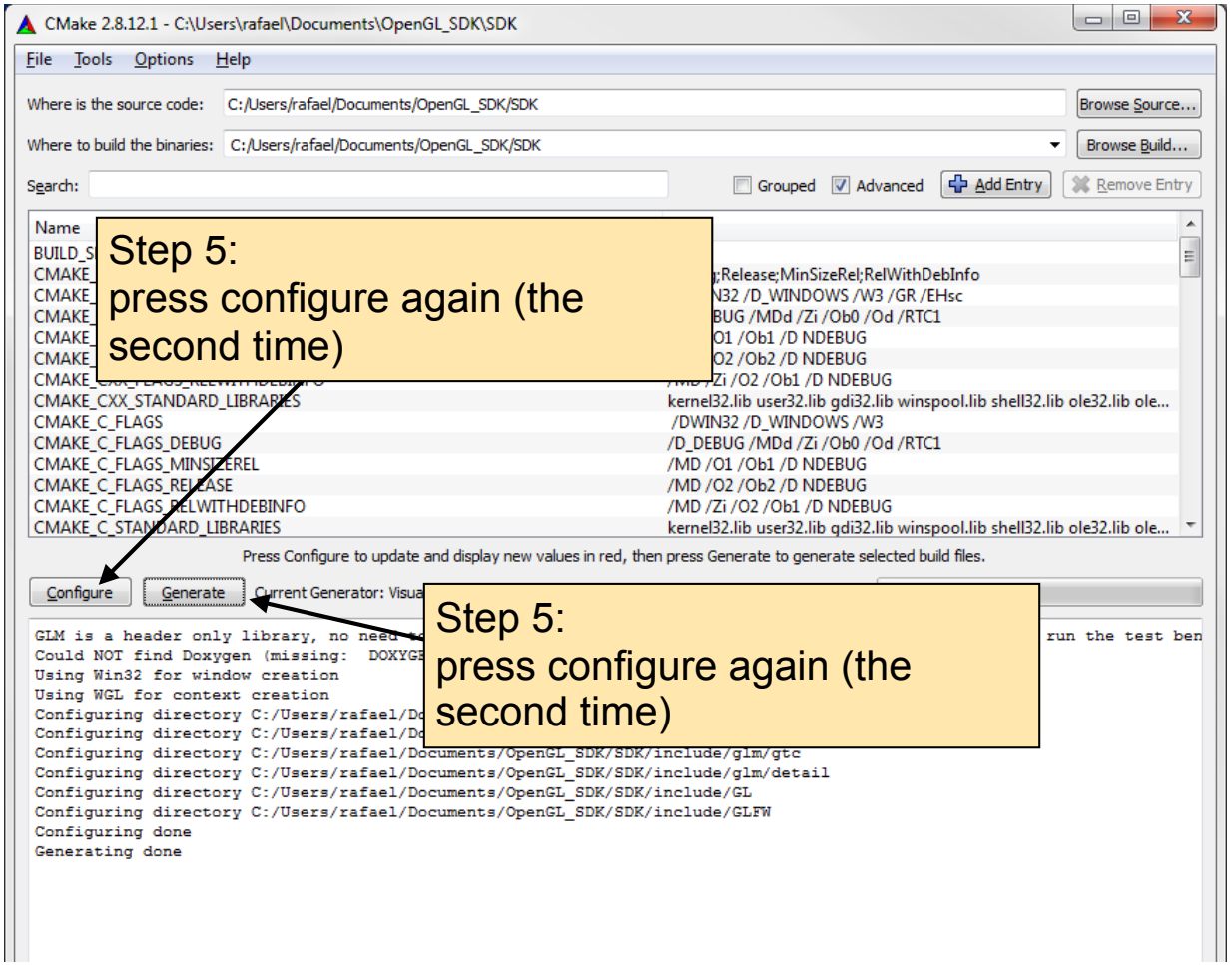
Main Window of CMake

All course example work with 64bit (x64_86) and not with 32bit (i386)

Create OpenCV project files (1/2)

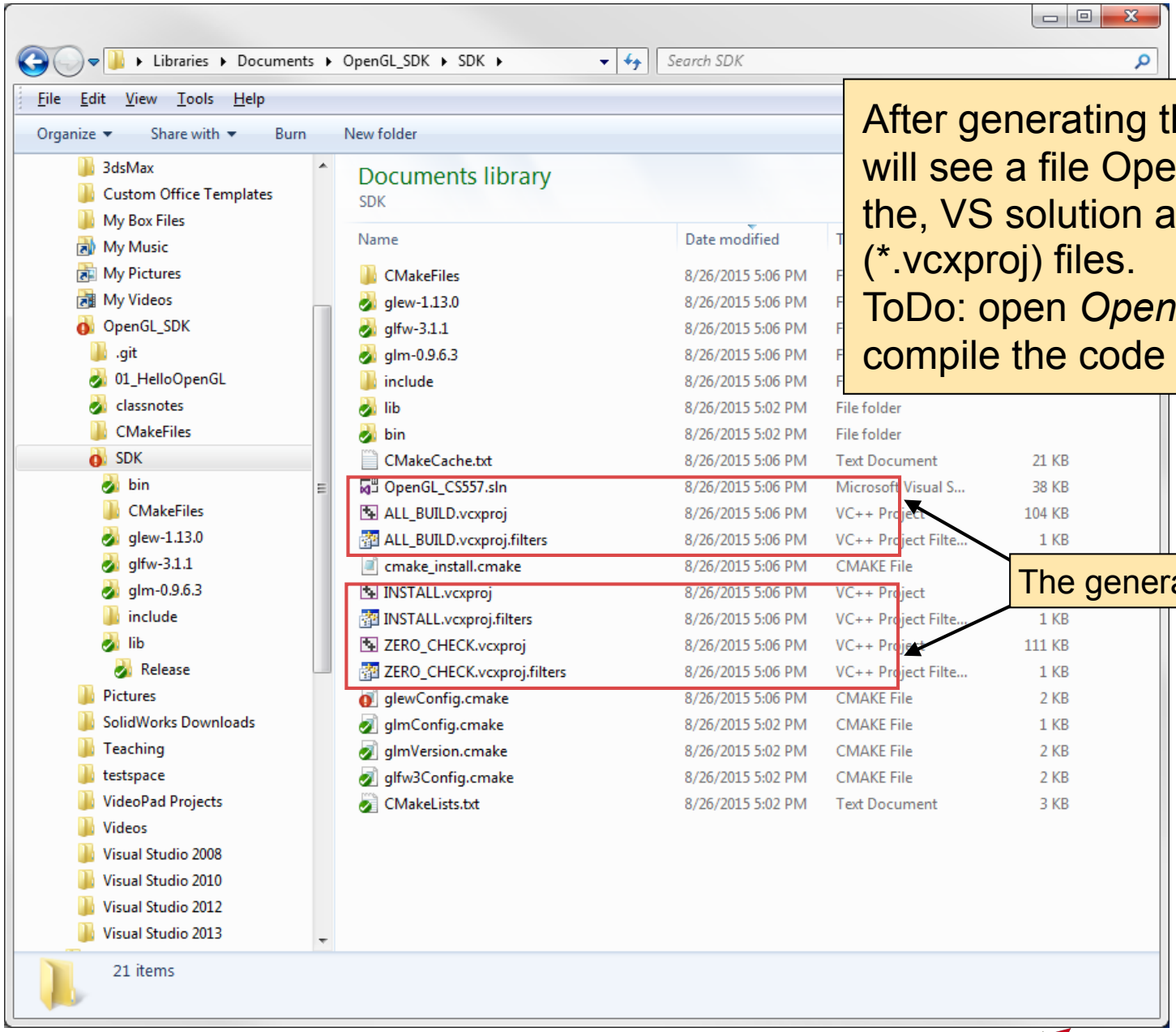
Step 5: press configure a second time.

Step 6: generate the Visual Studio Project files by pressing Generate



Main Window of CMake

SDK-Folder with GLEW, etc.



After generating the project files, you will see a file OpenGL_CS557.sln the, VS solution and multiple project (*.vcxproj) files.
ToDo: open *OpenGL_CS557.sln* and compile the code

The generated files

The SDK folder for this course

ARLAB

Install the OpenGL libraries GLEW, GLFW, and GLM

- Compile GLFW

ARLAB

OpenGL (Graphics Library) is a cross-language application programming interface (API) **specification** for rendering of 2D and 3D computer graphics. Every API (or function call) interacts with the Graphics Processing Unit (GPU) to achieve rendering.

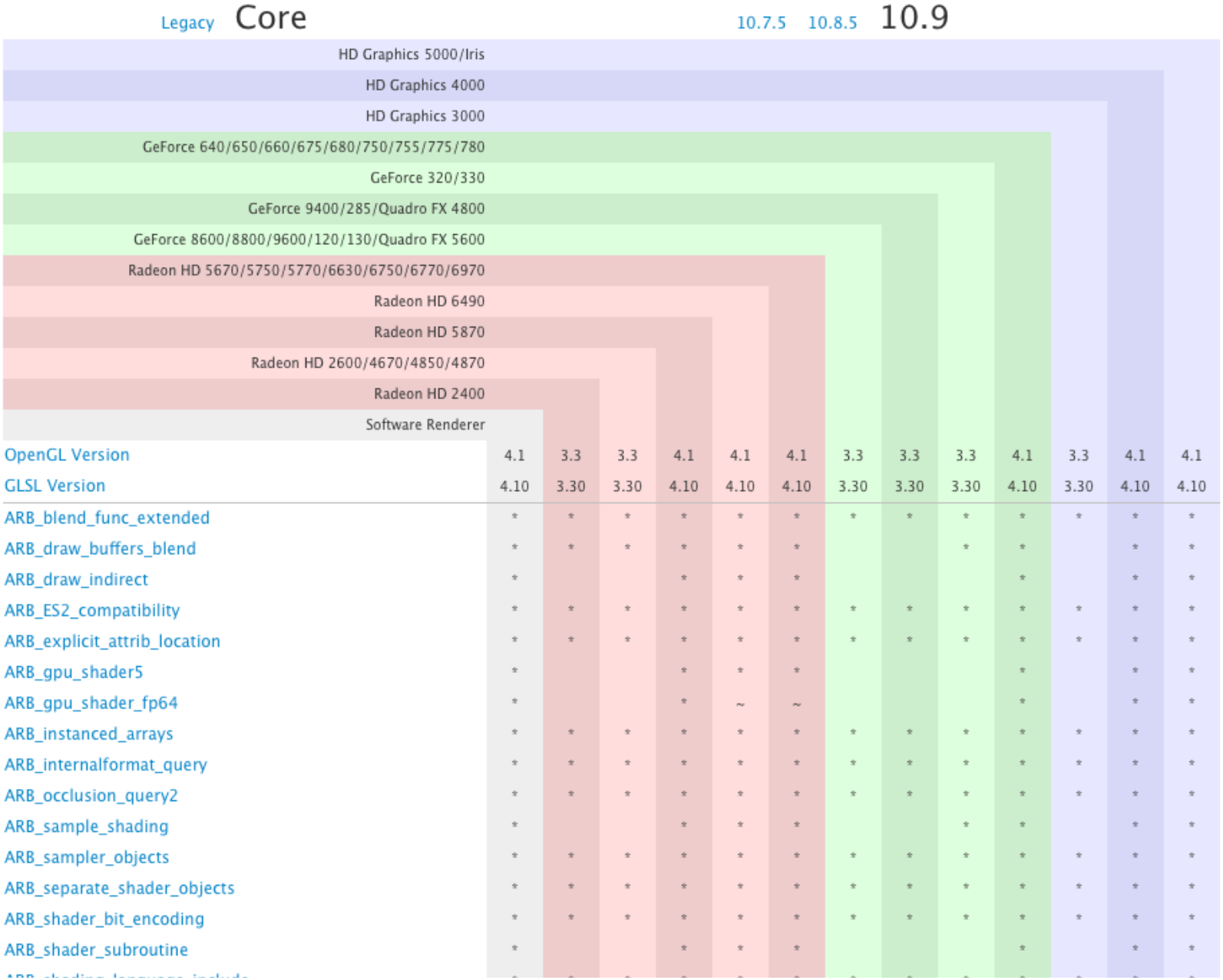
OpenGL was developed by Silicon Graphics Inc. The first version was released in 1992. Today, it is managed by the non-profit consortium Khronos Group.

Latest release OpenGL V4.5, Release on August 11, 2014

- About 150 distinct commands to specify the objects and operations needed to produce interactive three-dimensional applications.
- Geometric primitives like points, lines, and polygons.
- Implemented in C\C++ (Wrappers for Java and other languages are also available).
- No window control system
- OpenGL installation depends on the vendor
  - Windows (1.1, frozen and not supported by MS)
  - Mac OS X (4.1, OS X Maverick)

https://www.opengl.org

OpenGL Functionality



GLEW:
The OpenGL Extension Wrangler Library

ARLAB

http://glew.sourceforge.net

GLEW is an OpenGL implementation which provides access to the latest OpenGL driver functions

Latest Release: 1.13.0

Download
Usage
Building
Installation
Source Generation
Credits & Copyright
Change Log
GitHub
Project Page
Bug Tracker

Last Update: 08-10-15

OpenGL
GitHub
SOURCEFORGE.NET

The OpenGL Extension Wrangler Library

The OpenGL Extension Wrangler Library (GLEW) is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform. OpenGL core and extension functionality is exposed in a single header file. GLEW has been tested on a variety of operating systems, including Windows, Linux, Mac OS X, FreeBSD, Irix, and Solaris.

Downloads

GLEW is distributed as source and precompiled binaries.
The latest release is 1.13.0[08-10-15]:

Source	ZIP | TGZ
Binaries	Windows 32-bit and 64-bit

An up-to-date copy is also available using git:

- github
  `git clone https://github.com/nigels-com/glew.git glew`
- Sourceforge
  `git clone git://git.code.sf.net/p/glew/code glew`

Unsupported snapshots are also available:

- glew-20150805.tgz
- glew-20150124.tgz

Supported Extensions

The latest release contains support for OpenGL 4.5 and the following extensions:

- OpenGL extensions

ARLAB

"OpenGL Mathematics (GLM) is a header only C++ mathematics library for graphics software based on the OpenGL Shading Language (GLSL) specifications"
http://glm.g-truc.net/0.9.7/index.html

Provides the basic data structures for computer graphics such as:

- Vectors
- Matrices
- Quaternions

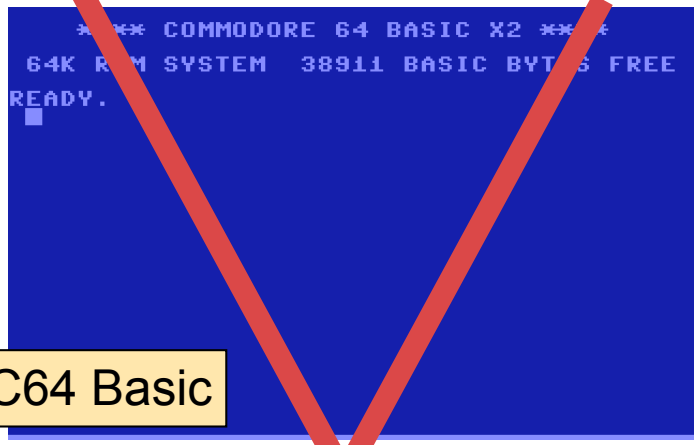
and graphics-related operations

- matrix transformation
- inverse
- projections
- ...

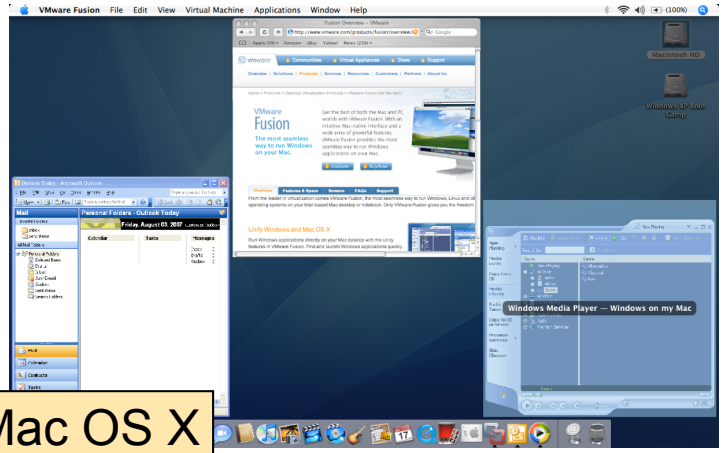
GLFW

ARLAB

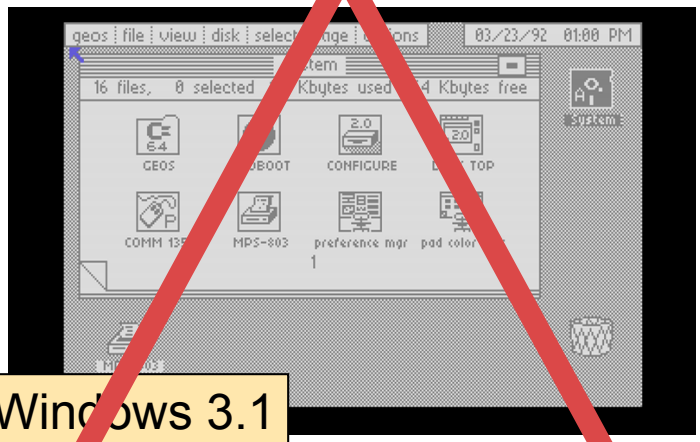
GLFW is a library that creates OpenGL window context on different platforms such as MS Windows, and Apple Mac OS X with no effort. It can also manage keyboard/ mouse events.



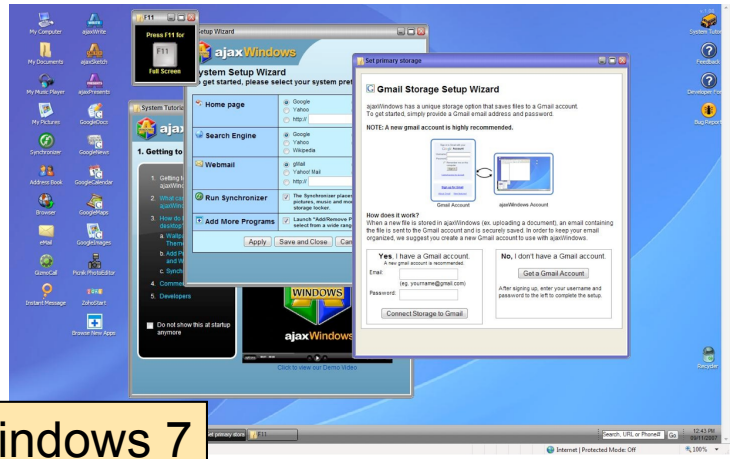
C64 Basic



Mac OS X



Windows 3.1



Windows 7

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

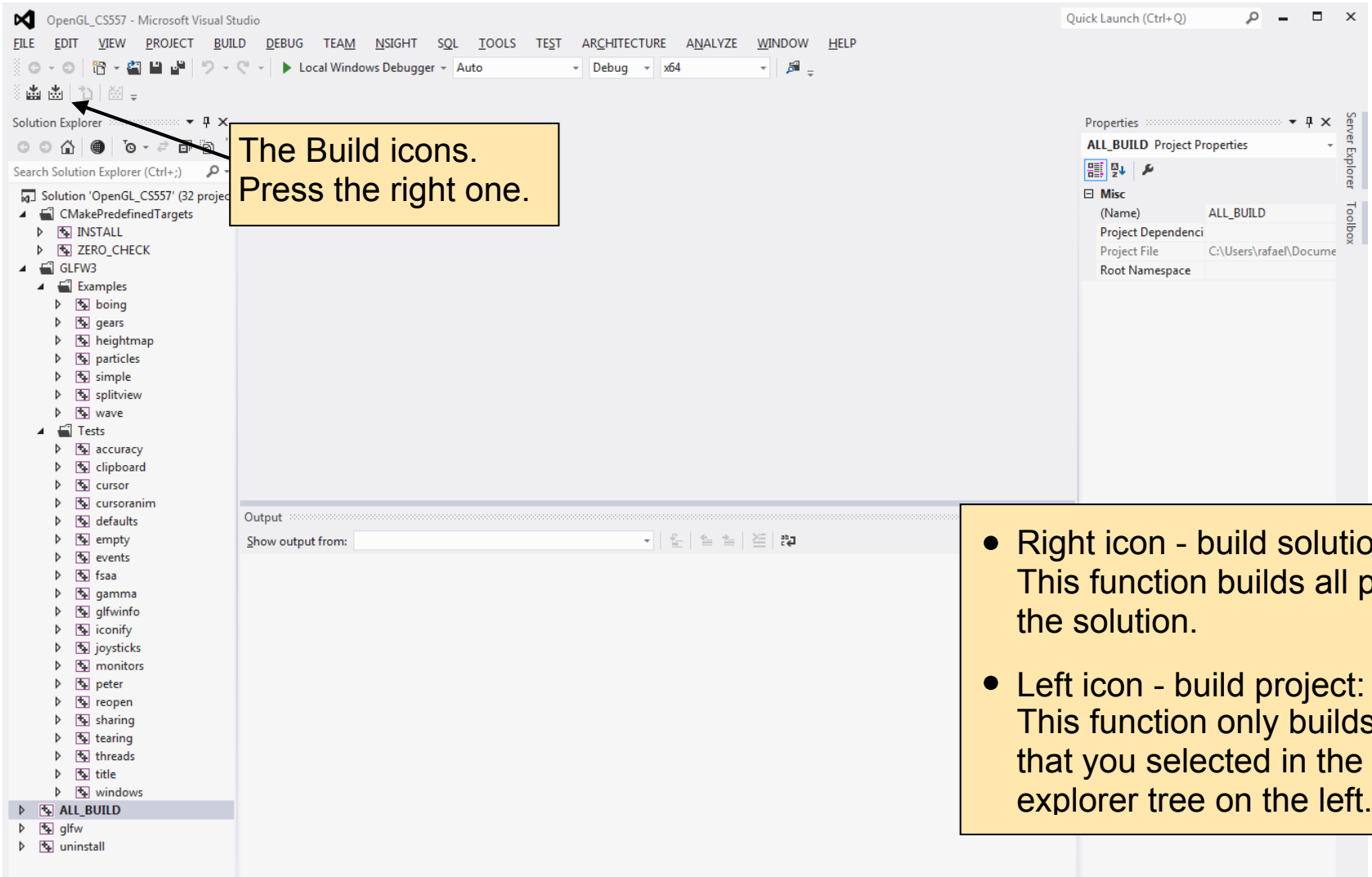
ARLAB

"GLFW is an Open Source, multi-platform library for OpenGL, OpenGL ES and Vulkan development on the desktop. It provides a simple API for creating windows, contexts and surfaces, receiving input and events." (http://www.glfw.org)

- Gives you a window and OpenGL context with just two function calls
- Support for OpenGL, OpenGL ES, Vulkan and related options, flags and extensions
- Support for multiple windows, multiple monitors, high-DPI and gamma ramps
- Support for keyboard, mouse, gamepad, time and window event input, via polling or callbacks
- Comes with guides, a tutorial, reference documentation, examples and test programs

Build the GLFW

Open OpenGL_CS557.sln and compile the code



- Right icon - build solution:
  This function builds all projects in the solution.
- Left icon - build project:
  This function only builds the project that you selected in the solution explorer tree on the left.

Mac OSX Users can use Homebrew

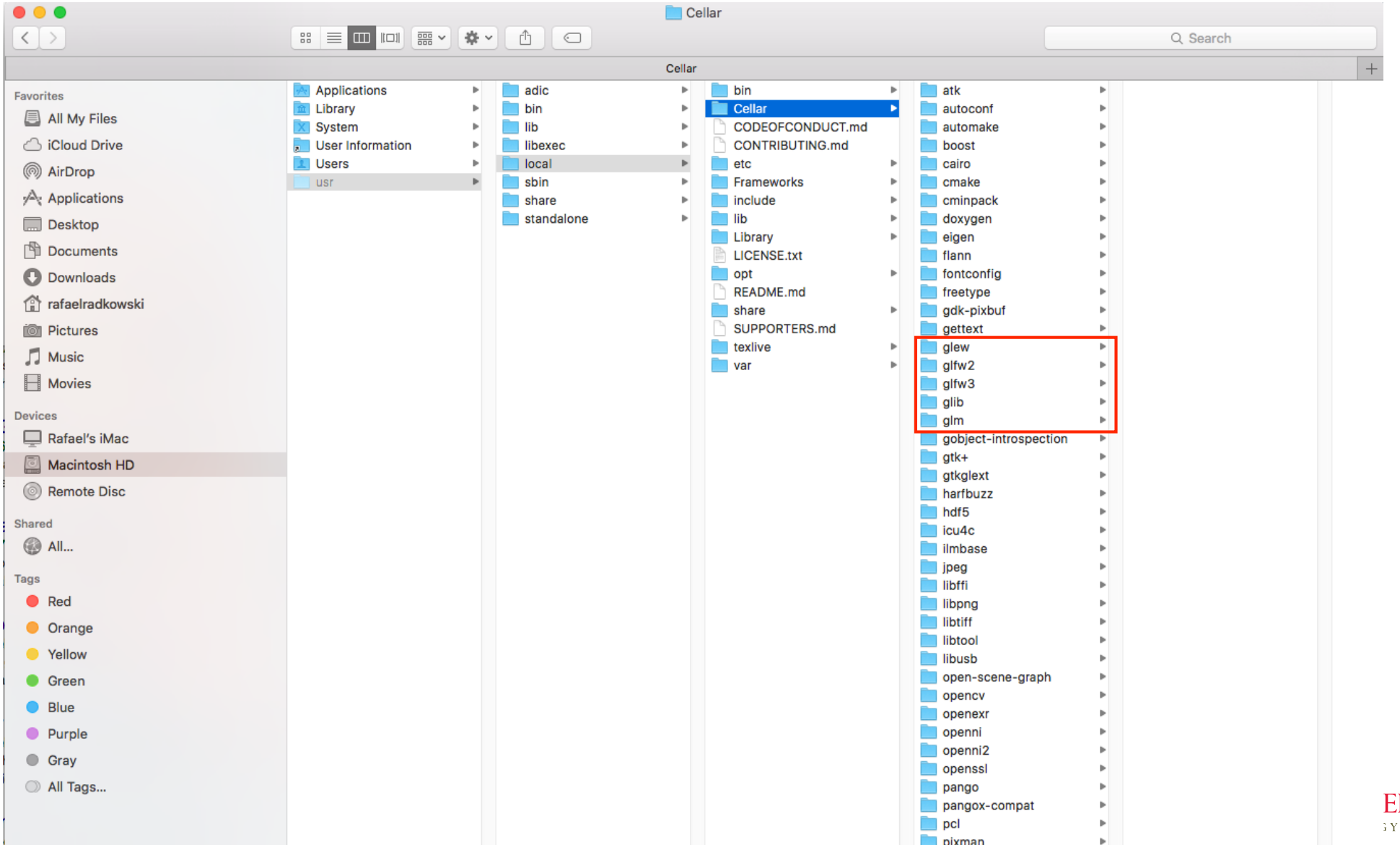
1. Open a terminal window

2. Install GLM (in your terminal, type and press enter): brew install glm
3. Install GLFW (in your terminal, type and press enter): brew install glfw3
4. Install GLEW (in your terminal, type and press enter): brew install glew

Warning: brew does not support the pre-release of Mac OS X El Captain!

Installation

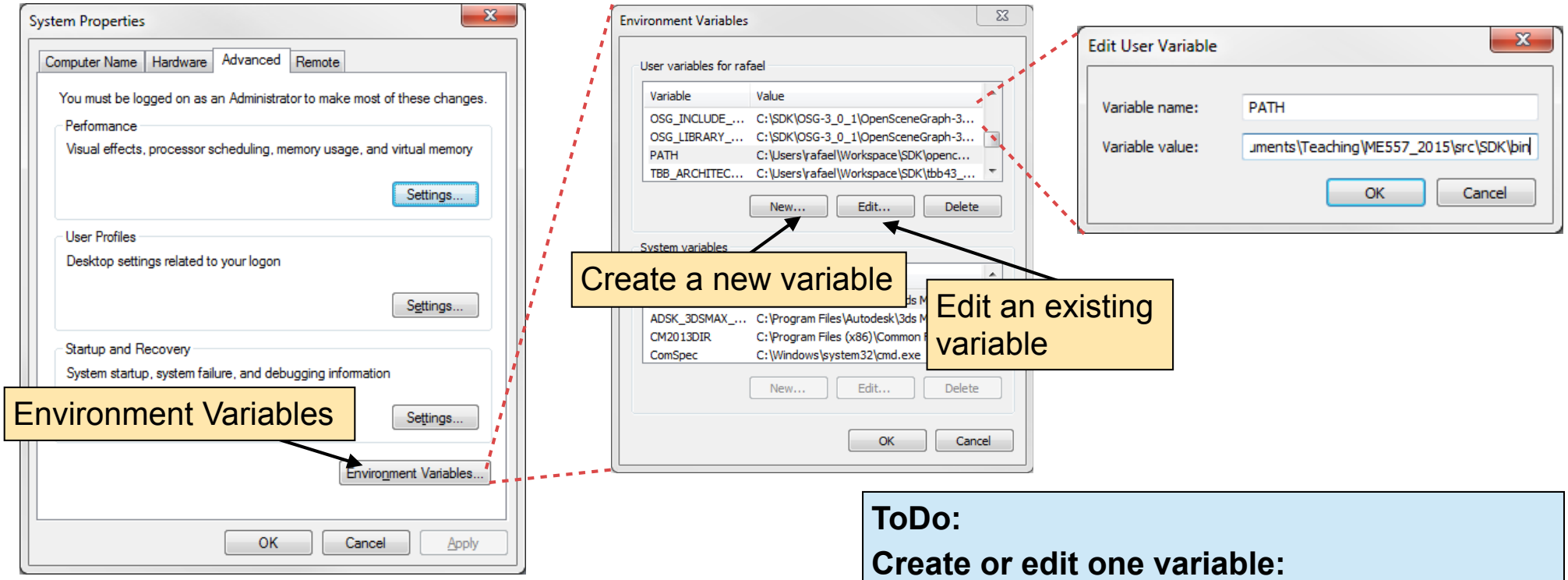
brew installs all files in /usr/local/Cellar



Setup Environment Variables

Environment variables help Windows and other programs to find files, folders, and programs by using a unique name.

Control Panel -> System -> Advanced System Settings -> Environment Variables



Windows System Properties Window

ToDo:

Create or edit one variable:

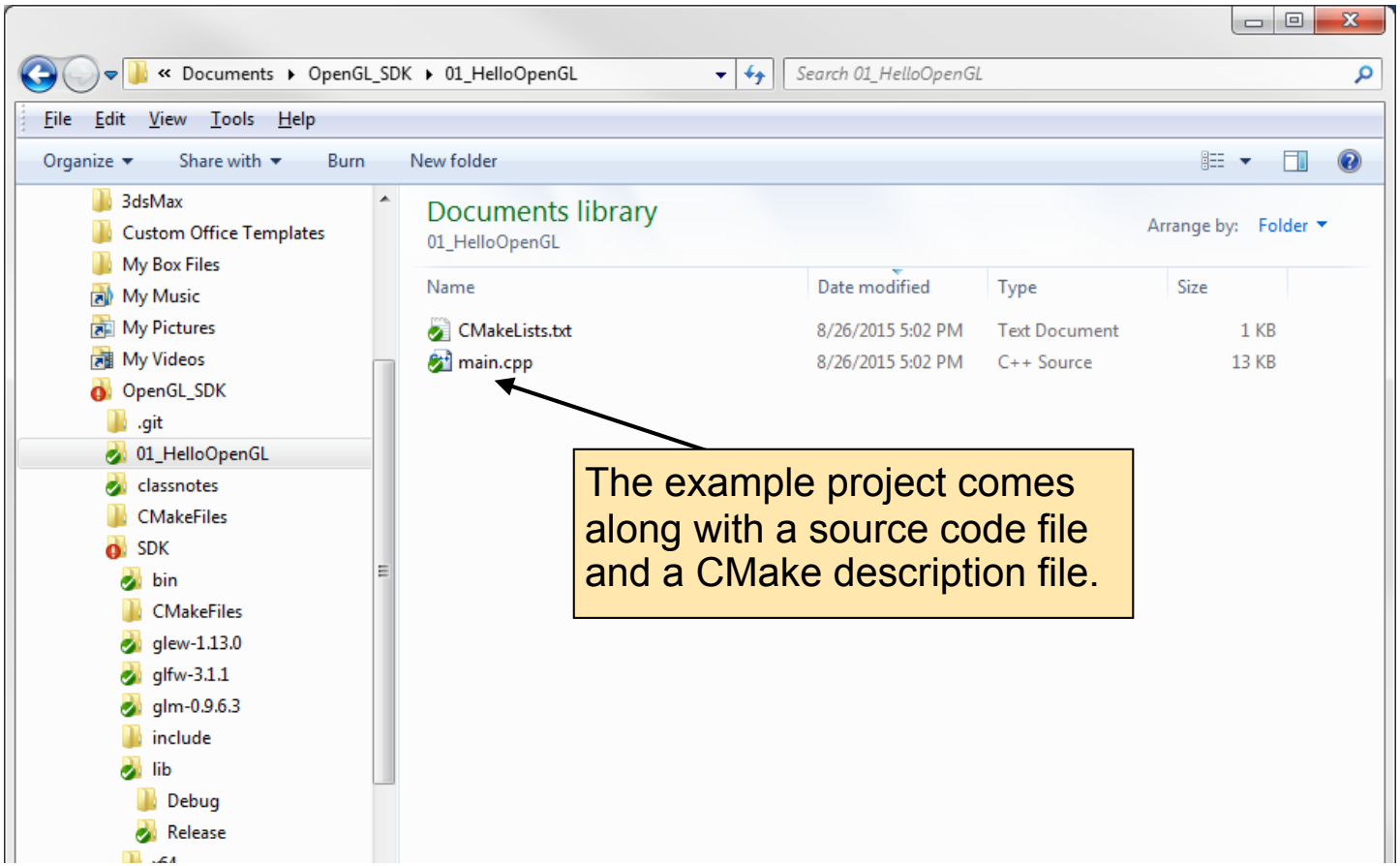
- Name: PATH
- Value: the path to the folder with **glew32.dll**

ARLAB

Demo Code HelloOpenGL

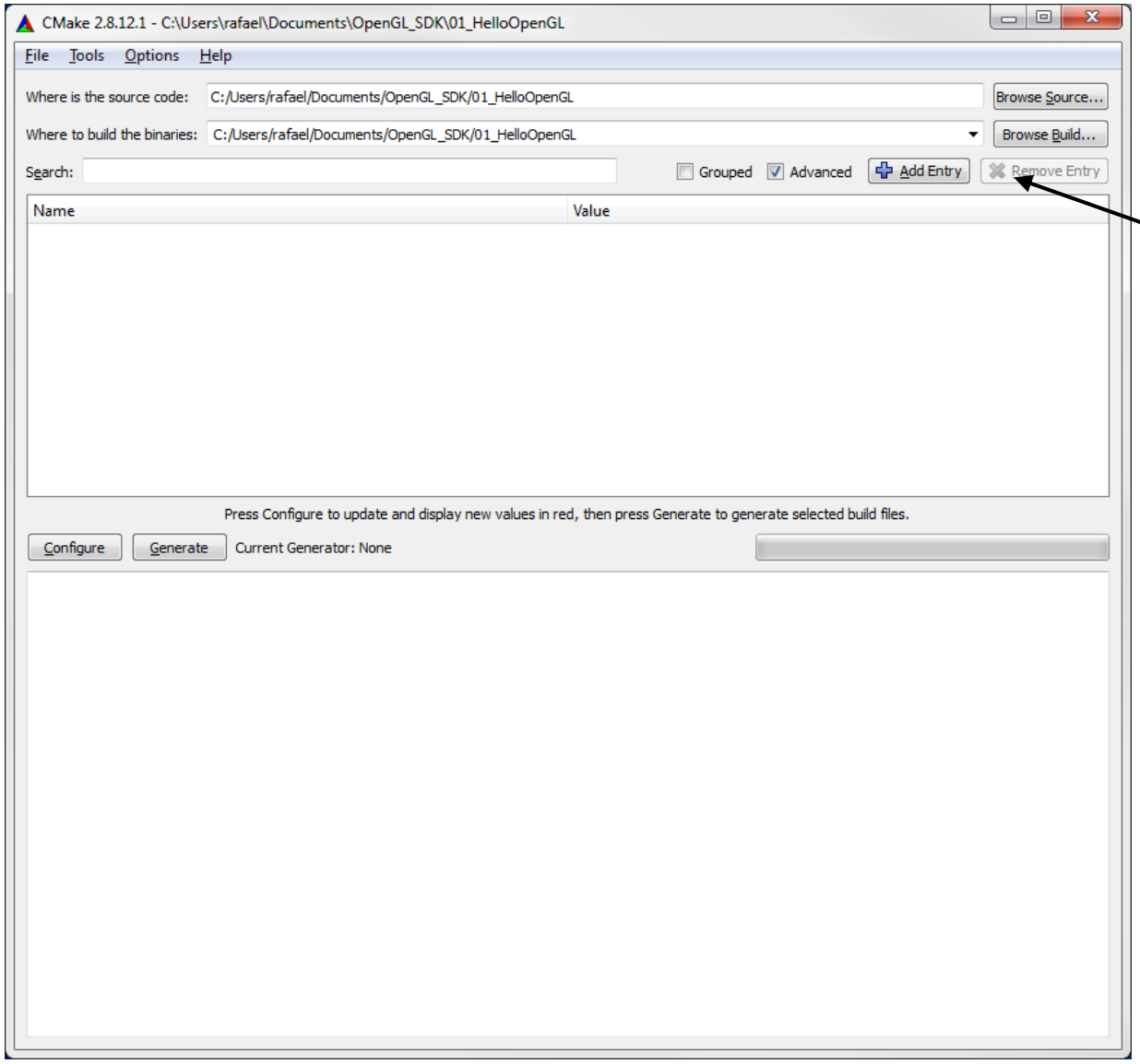
Hello OpenGL

Your homework: download (git), compile, and run the example project 01_Hello_OpenGL



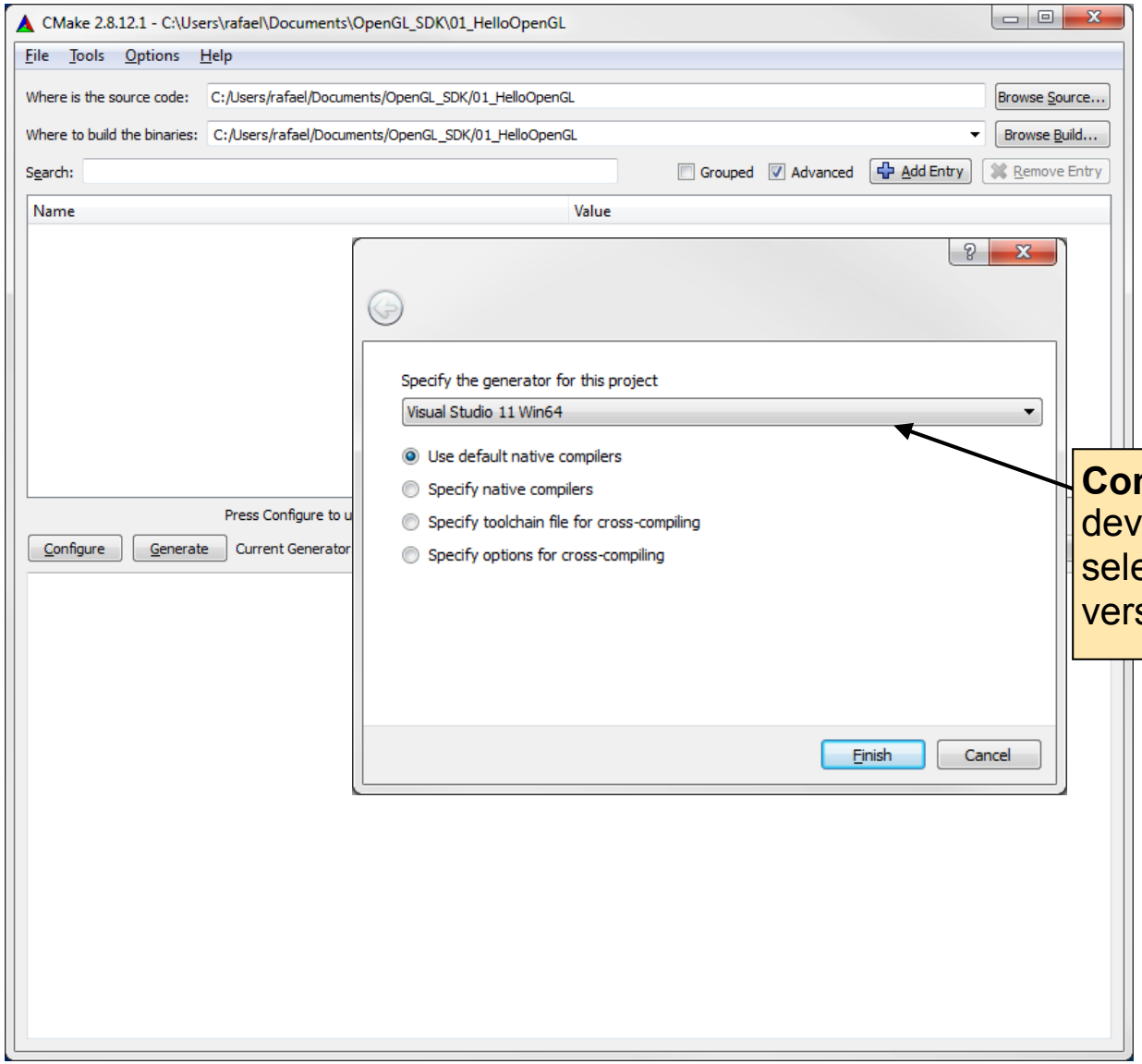
Windows Explorer

Generate the Project Files with CMake



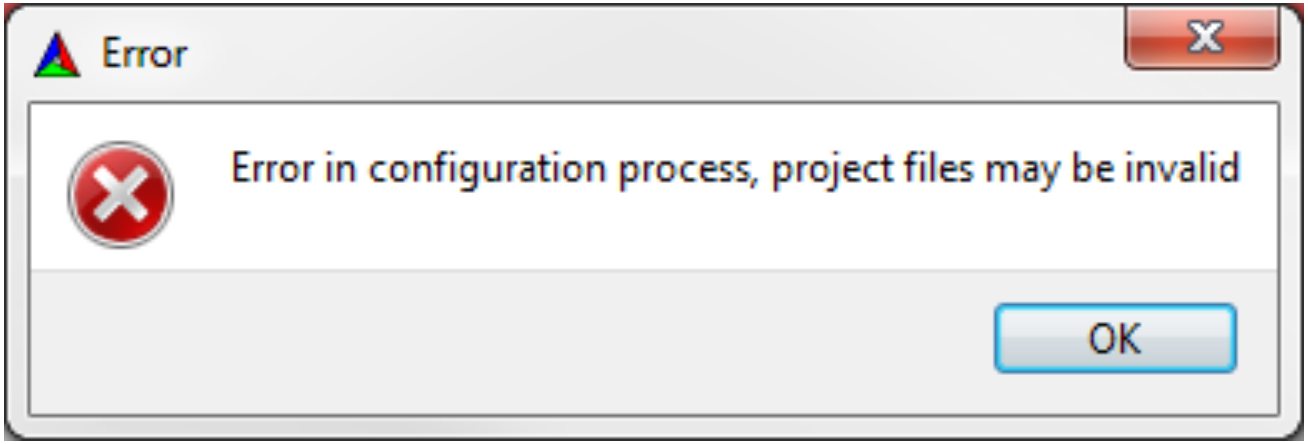
The same procedure we used before.
Select the CMakeLists.txt file and the destination folder.

Generate the Project Files with CMake

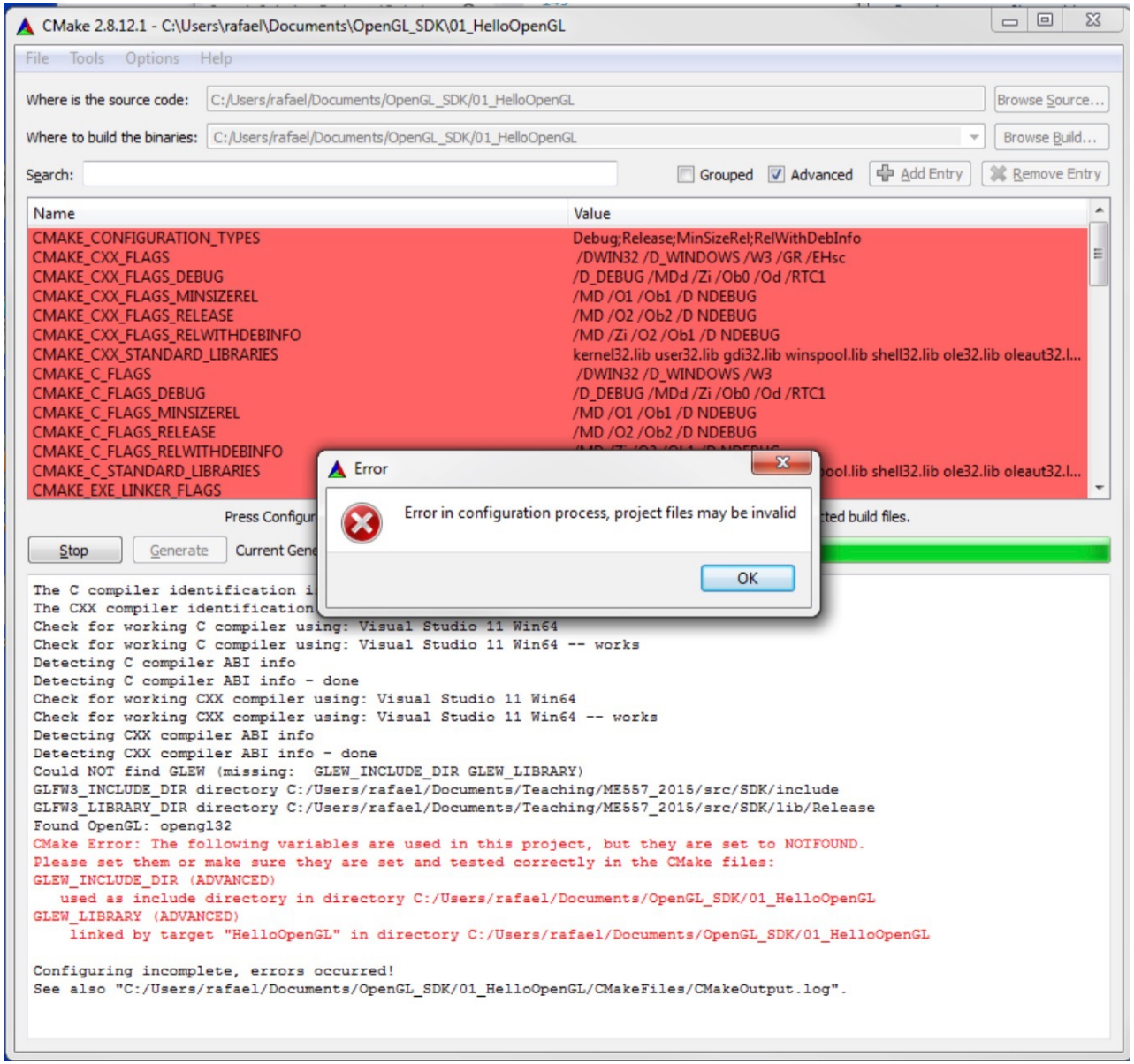


Configure the correct development environment and select the correct architecture version!

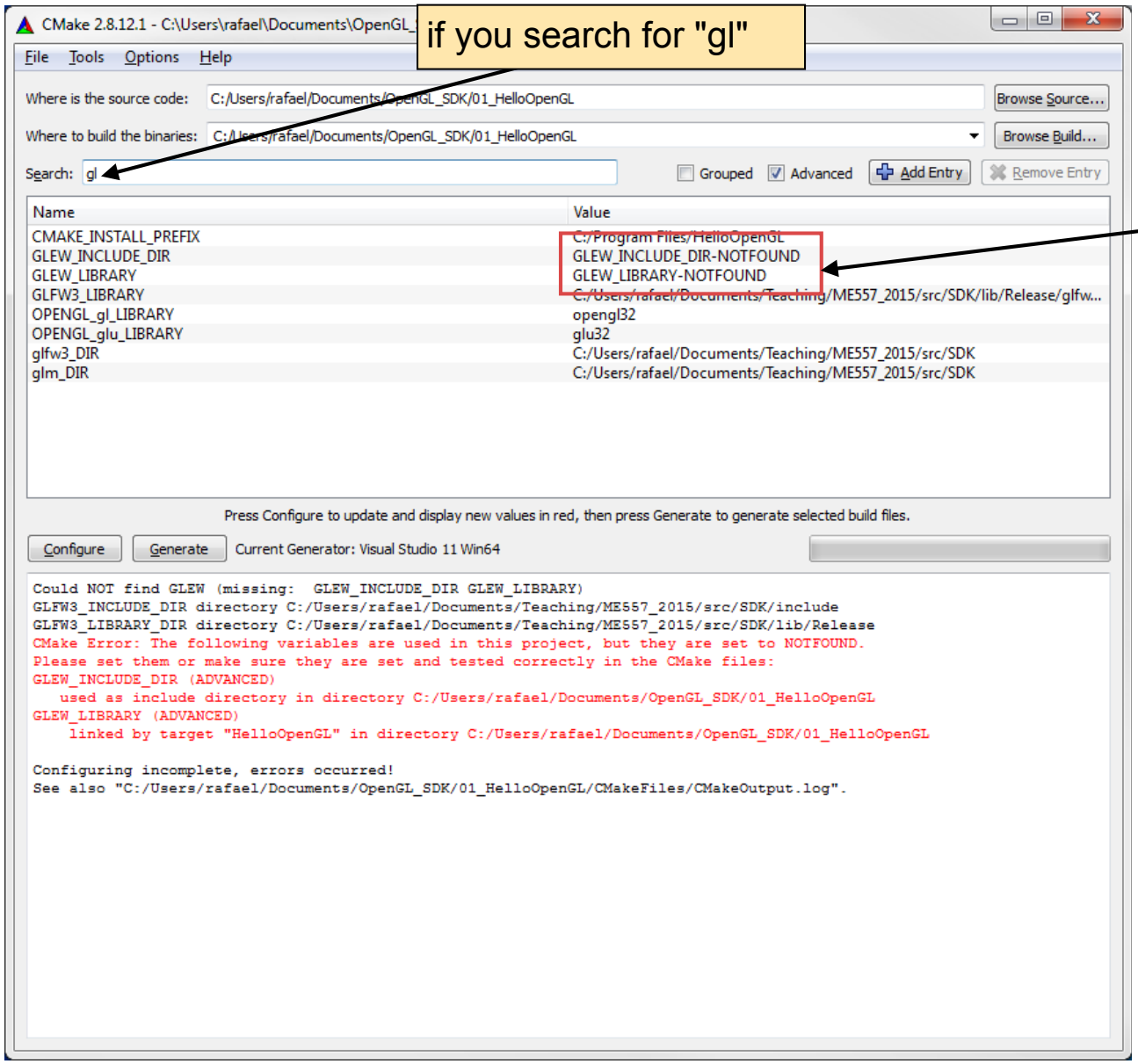
And now?


Error
Error in configuration process, project files may be invalid
OK

More Errors


CMake 2.8.12.1 - C:\Users\rafael\Documents\OpenGL_SDK\01_HelloOpenGL
File Tools Options Help
Where is the source code: C:/Users/rafael/Documents/OpenGL_SDK/01_HelloOpenGL
Browse Source...
Where to build the binaries: C:/Users/rafael/Documents/OpenGL_SDK/01_HelloOpenGL
Browse Build...
Search:
Grouped
Advanced
Add Entry
Remove Entry
Name
Value
CMAKE_CONFIGURATION_TYPES Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_CXX_FLAGS /DWIN32 /D_WINDOWS /W3 /GR /EHsc
CMAKE_CXX_FLAGS_DEBUG /D_DEBUG /MDd /Zi /Ob0 /Od /RTC1
CMAKE_CXX_FLAGS_MINSIZEREL /MD /O1 /Ob1 /D NDEBUG
CMAKE_CXX_FLAGS_RELEASE /MD /O2 /Ob2 /D NDEBUG
CMAKE_CXX_FLAGS_RELWITHDEBINFO /MD /Zi /O2 /Ob1 /D NDEBUG
CMAKE_CXX_STANDARD_LIBRARIES kernel32.lib user32.lib gdi32.lib winspool.lib shell32.lib ole32.lib oleaut32.l...
CMAKE_C_FLAGS /DWIN32 /D_WINDOWS /W3
CMAKE_C_FLAGS_DEBUG /D_DEBUG /MDd /Zi /Ob0 /Od /RTC1
CMAKE_C_FLAGS_MINSIZEREL /MD /O1 /Ob1 /D NDEBUG
CMAKE_C_FLAGS_RELEASE /MD /O2 /Ob2 /D NDEBUG
CMAKE_C_FLAGS_RELWITHDEBINFO
CMAKE_C_STANDARD_LIBRARIES
CMAKE_EXE_LINKER_FLAGS
Error
Error in configuration process, project files may be invalid
OK
Press Configur
ted build files.
Stop
Generate
Current Gene
The C compiler identification i
The CXX compiler identification
Check for working C compiler using: Visual Studio 11 Win64
Check for working C compiler using: Visual Studio 11 Win64 -- works
Detecting C compiler ABI info
Detecting C compiler ABI info - done
Check for working CXX compiler using: Visual Studio 11 Win64
Check for working CXX compiler using: Visual Studio 11 Win64 -- works
Detecting CXX compiler ABI info
Detecting CXX compiler ABI info - done
Could NOT find GLEW (missing: GLEW_INCLUDE_DIR GLEW_LIBRARY)
GLFW3_INCLUDE_DIR directory C:/Users/rafael/Documents/Teaching/ME557_2015/src/SDK/include
GLFW3_LIBRARY_DIR directory C:/Users/rafael/Documents/Teaching/ME557_2015/src/SDK/lib/Release
Found OpenGL: opengl32
CMake Error: The following variables are used in this project, but they are set to NOTFOUND.
Please set them or make sure they are set and tested correctly in the CMake files:
GLEW_INCLUDE_DIR (ADVANCED)
used as include directory in directory C:/Users/rafael/Documents/OpenGL_SDK/01_HelloOpenGL
GLEW_LIBRARY (ADVANCED)
linked by target "HelloOpenGL" in directory C:/Users/rafael/Documents/OpenGL_SDK/01_HelloOpenGL
Configuring incomplete, errors occurred!
See also "C:/Users/rafael/Documents/OpenGL_SDK/01_HelloOpenGL/CMakeFiles/CMakeOutput.log".

Troubleshooting



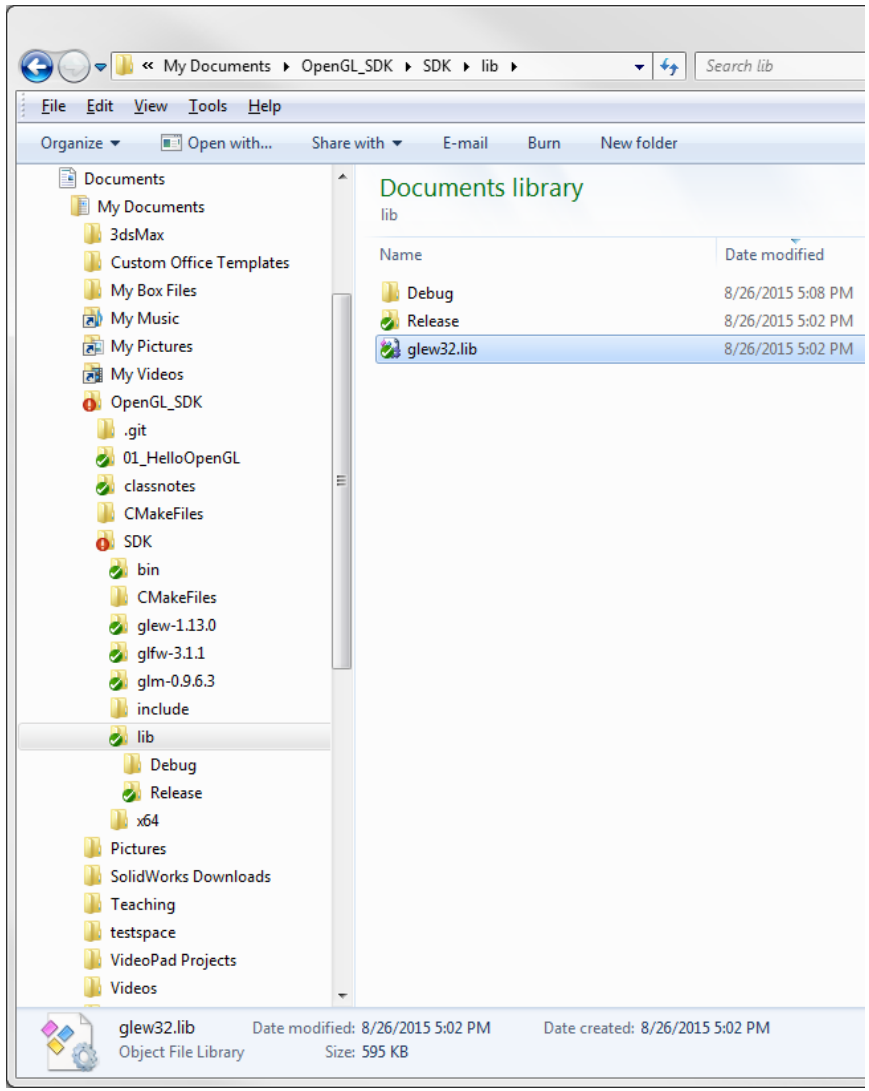
if you search for "gl"

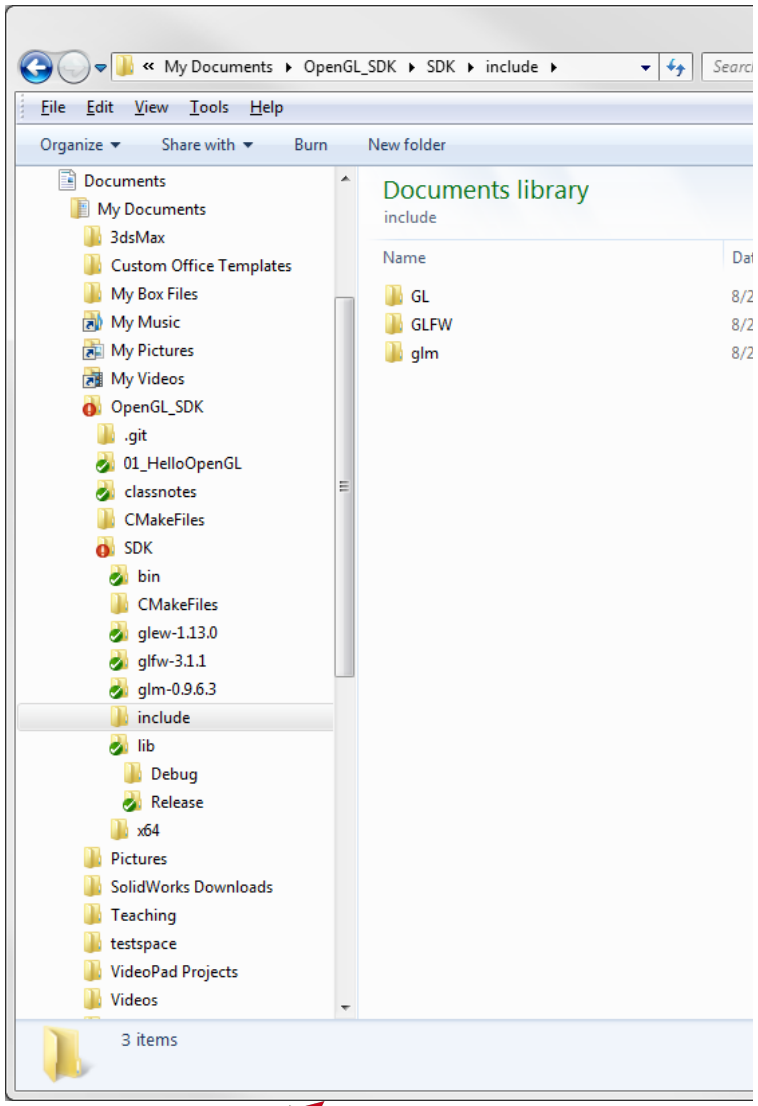
You should notice that CMake cannot find GLEW.

Solution: do it manually

Troubleshooting

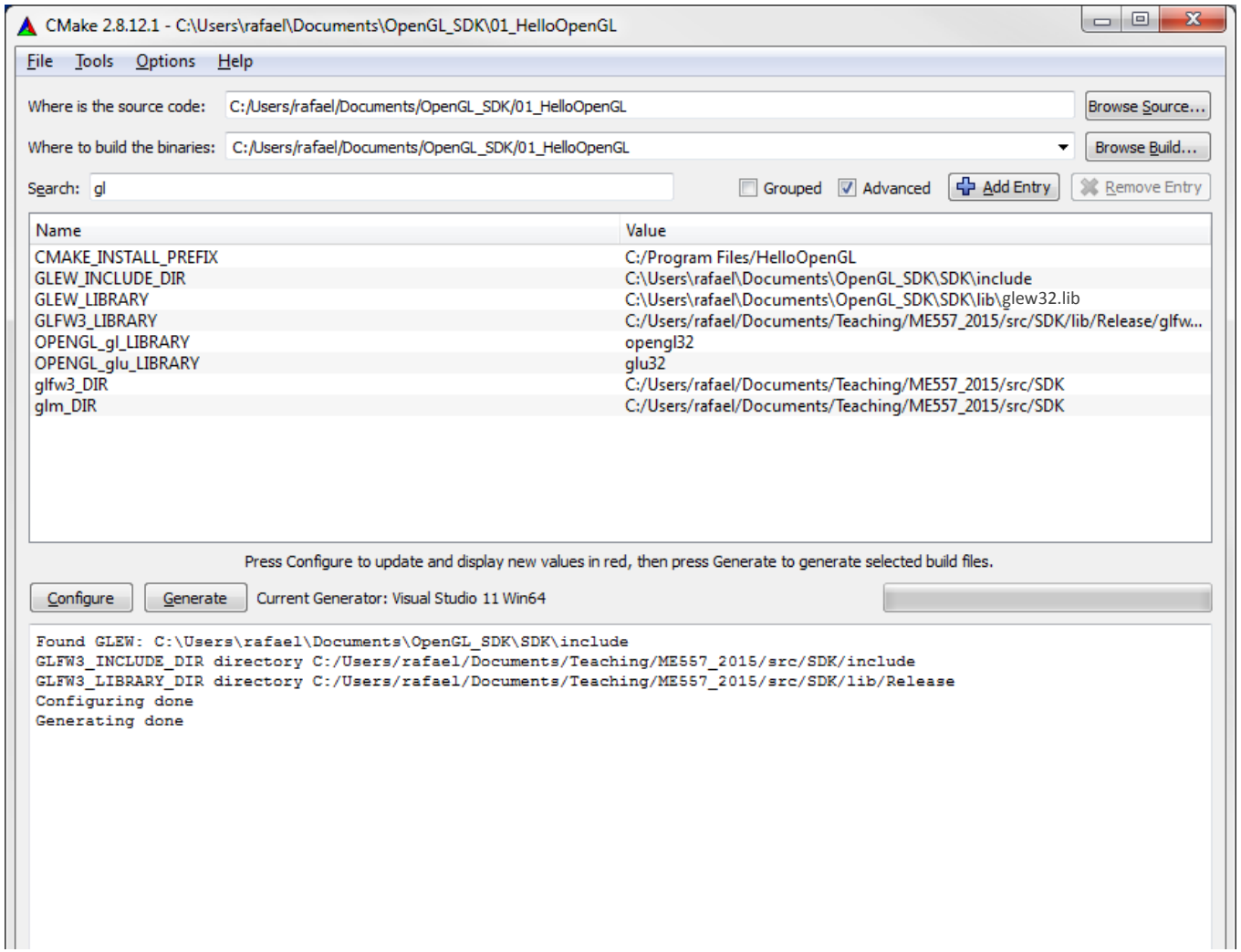
The glew32.lib file and the include-file path are missing!





Troubleshooting

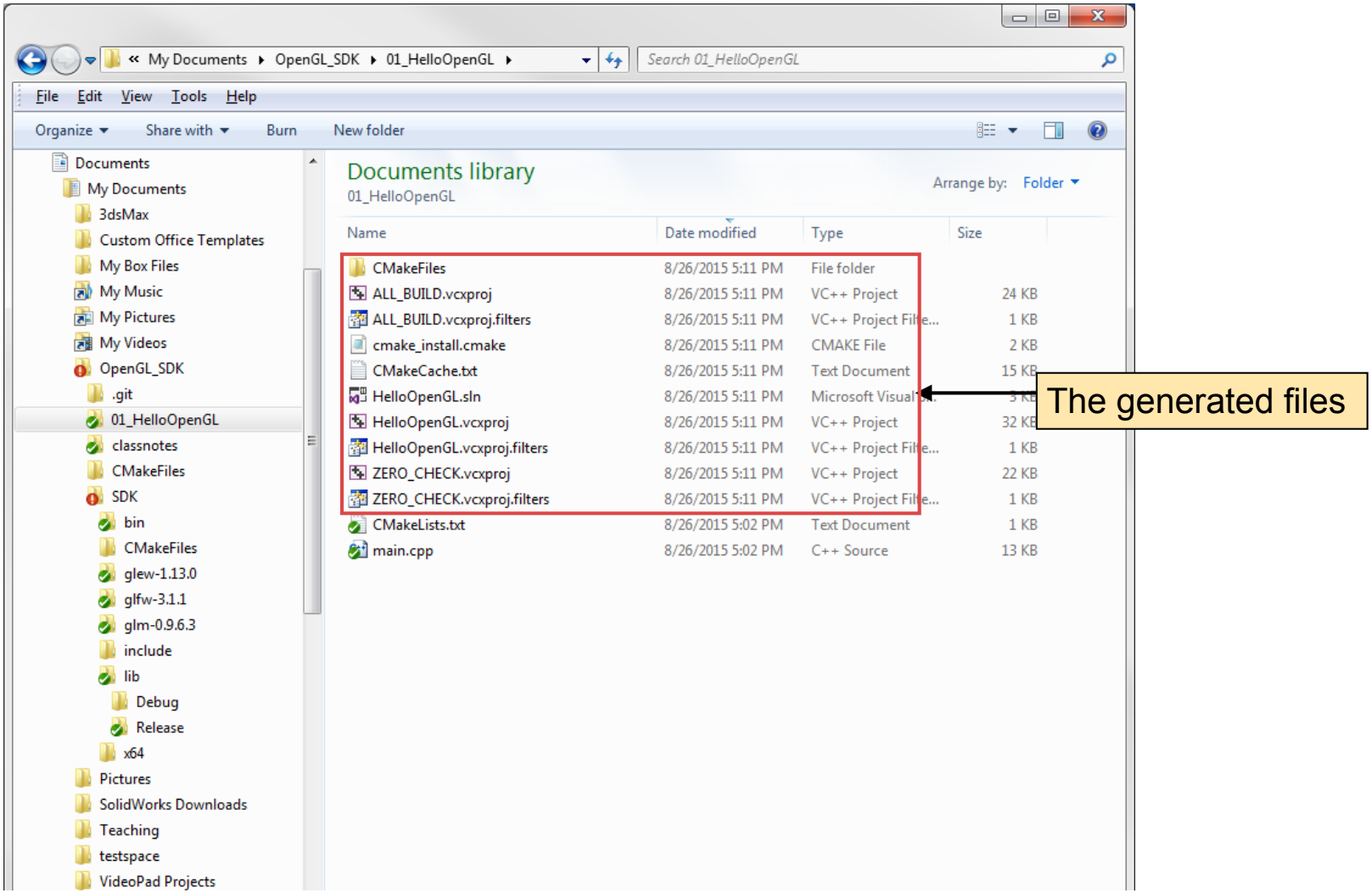
You can add them manually and restart.



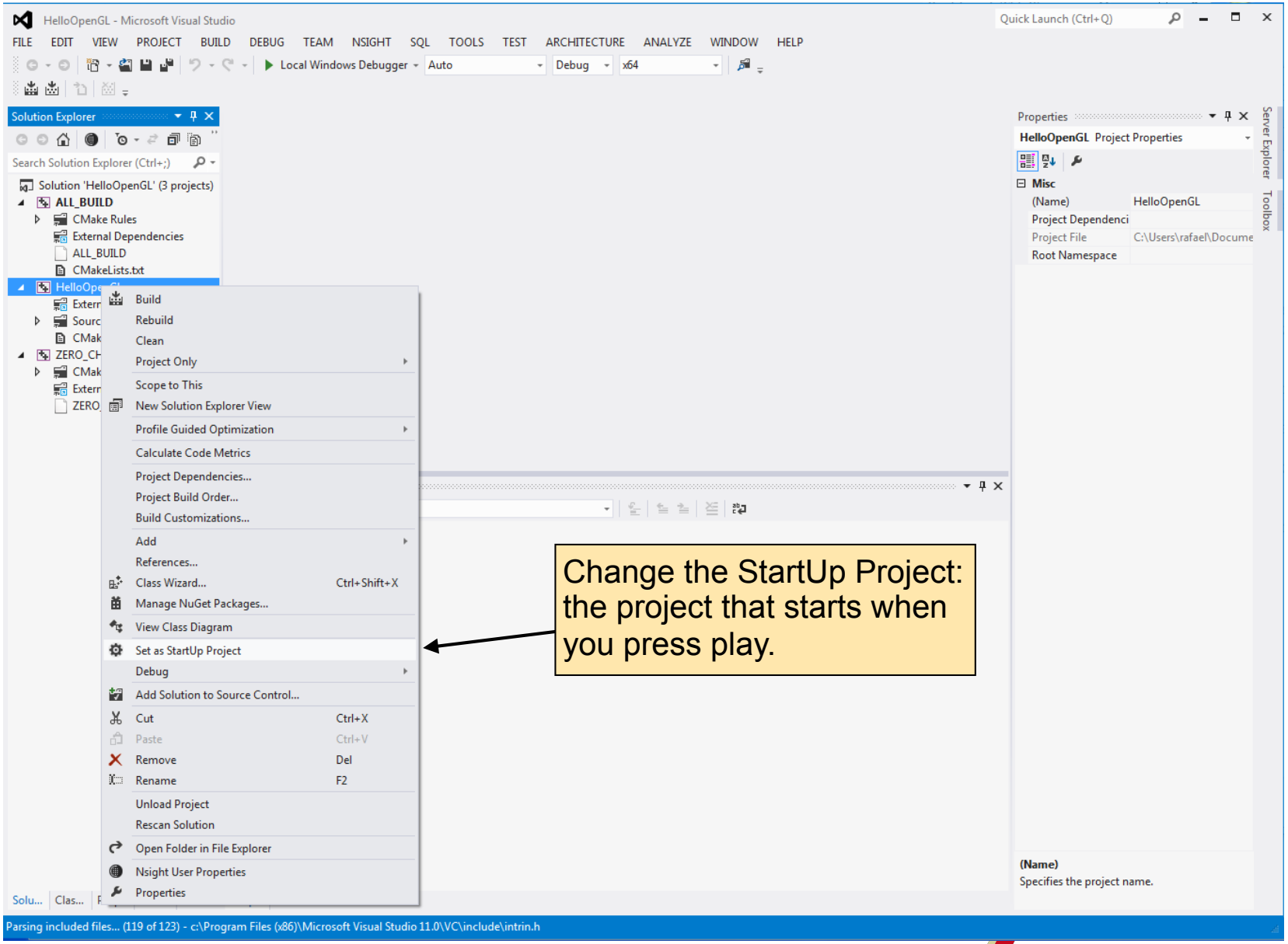
Result

ARLAB

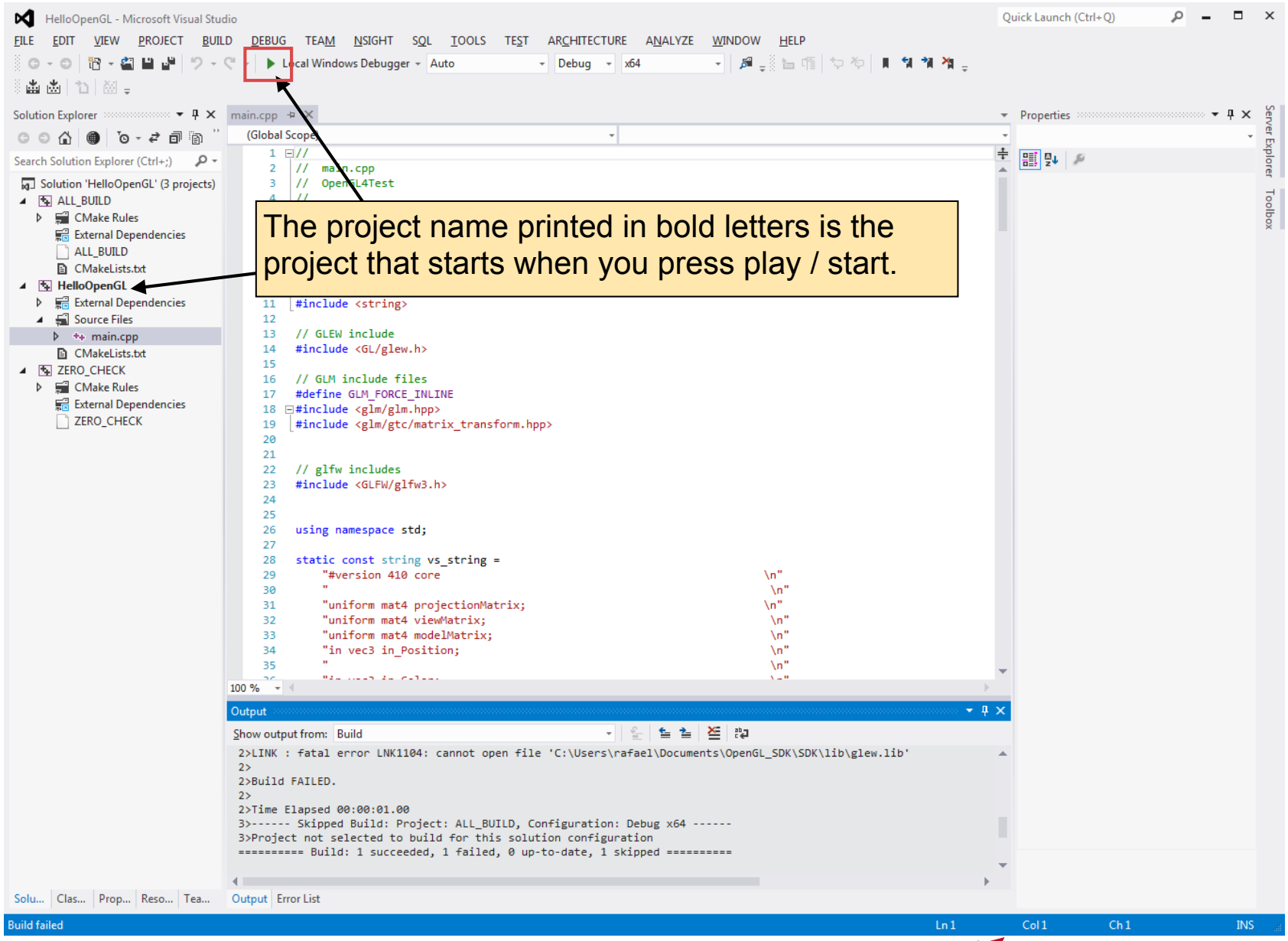
A solution file and the project files should be the outcome



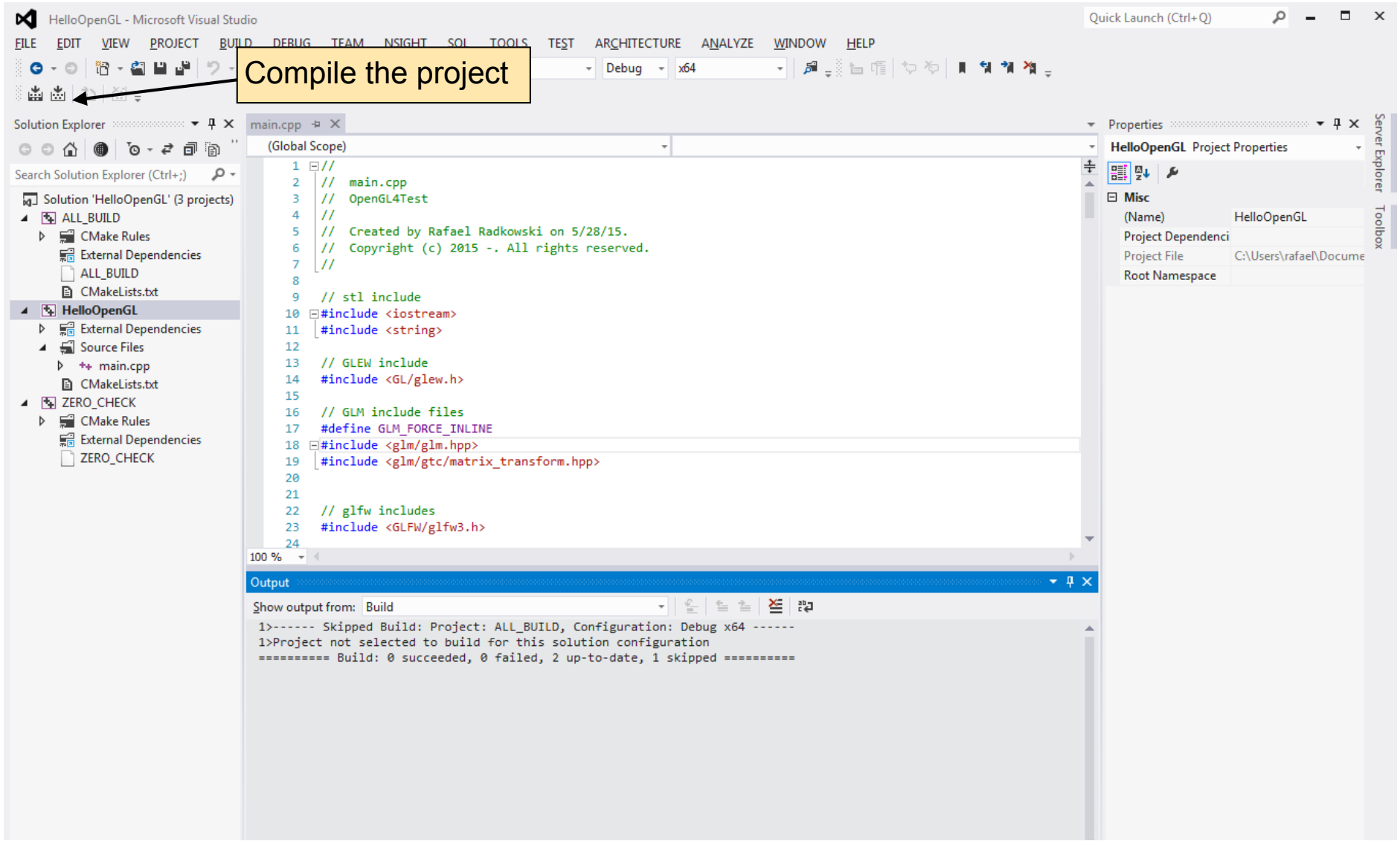
Change the StartUp Project



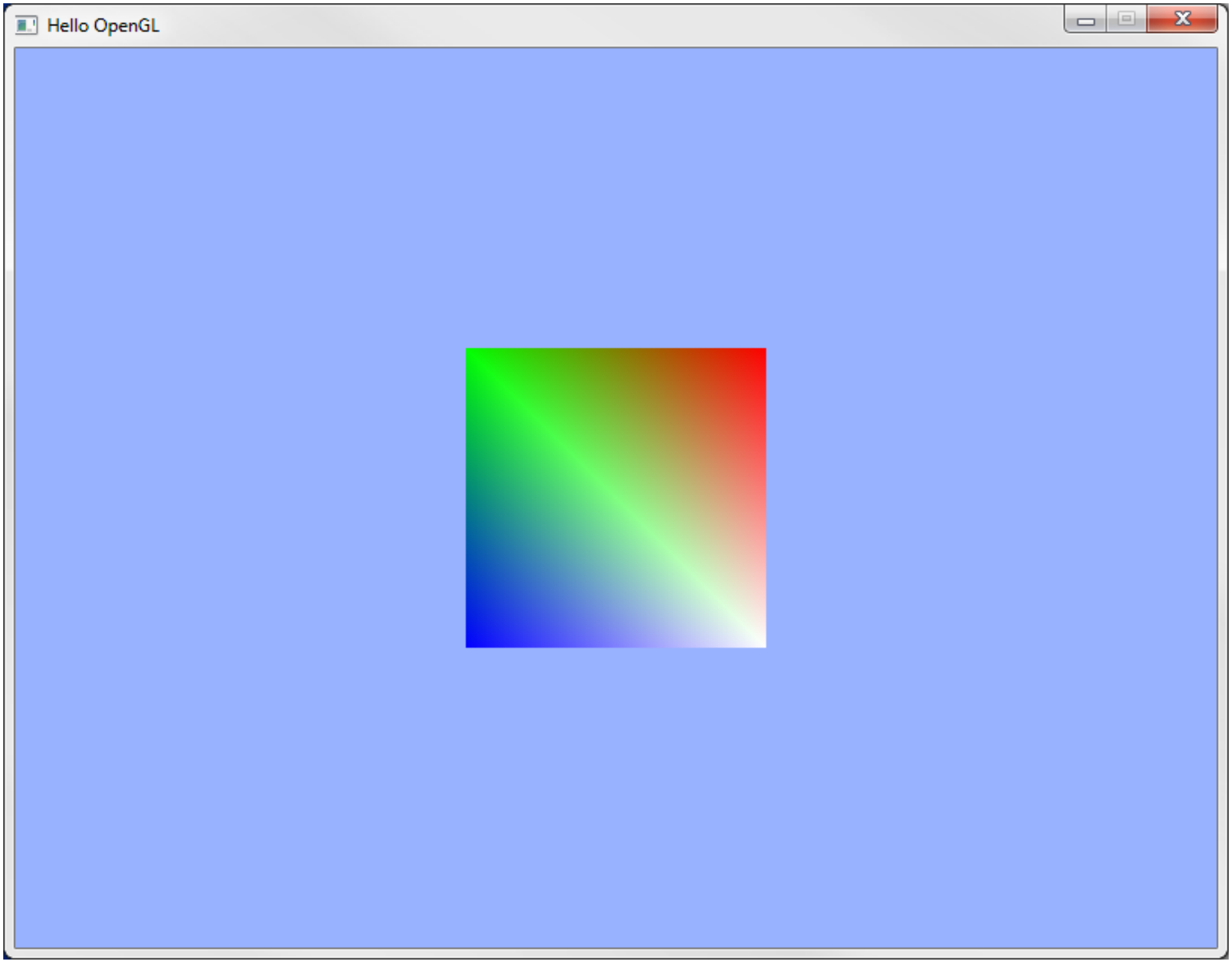
Run the Project



Compile first



Result



Thank you!

Questions

Rafael Radkowski, Ph.D.
Iowa State University
Virtual Reality Applications Center
1620 Howe Hall
Ames, Iowa 50011, USA
+1 515.294.5580

rafael@iastate.edu
http://arlab.me.iastate.edu

www.linkedin.com/in/rradkowski

ARLAB

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY