CS371 Tools Overview

Updated September 13, 2012

In CS371 you'll use a development environment similar to what you would encounter in professional development. It comprises a C++ build system, revision control, a debugger, documentation tools, profiling tools, and many software libraries. Most industry developers use commercial, visual integrated development environments (IDEs) like Visual Studio. In class we favor command-line open source tools . Learning these tools may help you understand the fundamentals better than the visual environments. What you learn with these tools is directly applicable to the visual environments, and they are always available to you for future courses and work environments because they are that are cross-platform and freely available.

This document briefly introduces the software development environment for CS371. It is intended as an introduction and quick reference guide. Refer to the online manuals and guides [van Heesch 2010; Roberts 2009; Collins-Sussman et al. 2008; McGuire 2012], OS X man pages, and built-in help commands for more detailed information. I'm intentionally telling how to find information rather than giving you the information directly so that you will learn to work with reference materials and external resources.

Contents

1 Subversion **2**
- 1.1 Revision Control . . . 2
- 1.2 Commands . . . 3
- 1.3 Starting Each Week . . . 4

2 iCompile **5**
- 2.1 Directory Organization . . . 5

3 Coordinate System **6**
- 3.1 3D . . . 6
- 3.2 World and Object Space . . . 6
- 3.3 Rotations . . . 6
- 3.4 2D . . . 6
- 3.5 Units . . . 6

4 Doxygen **7**
- 4.1 Markup . . . 7
- 4.2 Style . . . 7
- 4.3 Links . . . 8
- 4.4 Equations . . . 8

5 G3D **9**

6 Working from Home **10**

1 Subversion

1.1 Revision Control

Subversion is a **revision control system**. Revision control maintains a server-side **repository** (i.e. database) of the files in your project. You can **check out** (i.e., download) a copy of these files to your local machine, into what is often called a **workspace**. You then develop with the local copy and **commit** your changes back to the server, which merges your changes into the files already there. Commits usually occur at the end of your programming session or after completing some milestone. Because commits merge files, you can modify your program on multiple computers and you individual changes will be integrated at the server. Multiple programmers can also modify files from the project simultaneously and independently, and then rely on the merge to integrate them. Once you have a workspace, you can also **update** it by merging any changes from the server side made since check out time into your workspace. Most software today, both in research and industry, is developed using revision control to manage project files. That is because of the many advantages it offers, including:

1. History–you can jump back to the state your project had at any previous commit point. This is particularly useful if some new change introduced a bug or accidentally removed a component.

2. Asynchronous development–multiple developers can work on the same code base without tightly coordinating.

3. Multi-computer development–you can use the fast local disk for work and rely on revision control for moving files between computers, rather than explicit copying which is prone to error.

Revision control has drawbacks as well. To avoid these, adopt the following practices:

1. Always add new files to the system as soon you create them. Adding does not commit.

2. Always commit before leaving a machine, and then update to see if you forgot to add new files.

3. Update and build every time you sit down at a computer. This will alert you if the build is broken before you make new changes.

4. Always run `svn status` in your project root before you log out to make sure you checked everything important in.

5. Work in small increments, committing frequently.

6. Only commit working builds. Use `if (false)` or comments to temporarily disable broken code if you have to end your programming session at a specific time.

7. Avoid editing the same files, and especially the same methods, simultaneously with your partner. The system cannot merge changes to the same line of code and changes within the same method are likely to merge but risk incorrect semantics.

8. Never copy or move directories that are under revision control.

9. Never modify the `.svn` directories.

10. Never add generated files (e.g., executables, generated documentation) to the repository.

11. Avoid adding large binary files (e.g., 20 MB movies, PSD files), and especially avoid changing such large binary files because Subversion cannot merge these, so they consume tremendous server space and slow down the system.

1.2 Commands

You will access Subversion through the `svn` command-line program. Issue subversion commands by running `svn` with arguments specifying the operation you would like to perform and any options that command requires. The major commands that you will use are:

`svn co` *source-URL*

`svn update`

`svn add` *filename*

`svn commit -m "` *log message* `"`

`svn export [--force]` *source-URL* *dest-dir*

`svn status`

To tell Subversion to ignore a file, use:

`svn propset svn:ignore` *file-pattern* *containing-dir*

For, example,

```
svn propset svn:ignore log.txt data-files
```

Refer to the Subversion manual [Collins-Sussman et al. 2008] or use the `svn help` command for other useful commands and for the details of these.

When you commit you must specify a log message. Make this a one sentence description of your changes. These will help you if you need to revert a change and will help your partner (in future projects) to understand what new code has come in with an update.

You can combine `svn` with other Unix commands for some powerful effects. For example, to recursively descend through your source tree and add all files to Subversion except for generated and temporary ones, use the command (all on one line):

```
find .  | sed -e "/\/build\//d" -e "/\/build$/d" -e "/.svn/d"
    -e "/\/tmp$/d" -e "/\/tmp\//d" -e "/\/\.ice-tmp$/d"
     -e "/\/\.ice-tmp\//d" -e "/~$/d"  -e "/\/log.txt$/d"
     -e "/\/g3d-license.txt$/d" -e "/^\.$/d" | xargs svn add
```

1.3 Starting Each Week

For each project I will create a Subversion module for you. This will either have your username or an assigned team name in the directory name.

Your workspace will initially be an empty directory. For most projects, you'll quickly fill this by copying your solution (or another student's) from the previous week. You can't just copy the directory structure of another project directly because Subversion maintains its state in subdirectories named `.svn`. If you copy a `.svn` subdirectory, you will corrupt the state of your working copy. Copying would also bring along generated files like executables that you don't want.

Use the Subversion `export` operation to export a previous solution from the server and strip its revision control data. You can then check this back in as a different project. If your username was `ewilliams` and you were working on Project 1, the commands for this process would be:

```
cd /local-scratch
svn co svn://graphics-svn.cs.williams.edu/371/1-Meshes/meshes-ewilliams meshes-ewilliams
svn export --force svn://graphics-svn.cs.williams.edu/371/0-Cubes/ewilliams-cubes meshes-ewilliams
cd meshes-ewilliams
svn add *
svn commit -m "Exported from previous week"
```

2 iCompile

iCompile is an automated build system for C++ on Linux and OS X. It provides similar functionality to tools like Make, MSBuild, and Ant. What makes iCompile unique is that it generally requires no configuration. You just run `icompile` with no options in the root directory of your project and it automatically determines dependencies, directories, and compiler and linker options and builds your program. You can also use it to build documentation, shared and static libraries, and standalone OS X distributions (.dmg files).

iCompile is implemented as an open source Python script that is installed as part of the G3D distribution. Run `icompile --help` to see a full list of options. Some of the most common are:

`--run [... args ...]` If compilation succeeds, run the program. Arguments can be passed on after the run flag.

`--gdb [... args ...]` If compilation succeeds, run the program under a debugger.

`--clean` Delete all generated files.

`--doc` Generate documentation from Doxygen markup.

`--opt` Build an optimized executable.

You can customize iCompile's behavior by editing `~/.icompile` and the project's `ice.txt` file.

2.1 Directory Organization

iCompile can work with almost any directory structure. However, it treats certain directory names specially to support common development needs. For CS371, I want you to take advantage of this by structuring all of your projects with the following subdirectories:

(project root)	The `mainpage.dox` and `journal.dox` files
`source`	All of your source code, divided among `.h` and `.cpp` files
`data-files`	Any runtime data required by your program that is *not* also in the G3D data distribution.
`doc-files`	Any data required for your report or documentation, such as images and videos.
`journal`	Any data required for your development journal that isn't also needed for your general documentation. This usually contains a list of dated screenshots and isn't used on the first project.
`graveyard`	Files that you want to keep around for your own reference but do not want me to evaluate or the build scripts to process.

You **must** use the exact naming scheme described here, including capitalization, to ensure that the scripts I use to process the projects work correctly. The naming scheme is part of the specification for each project and you will lose points for varying from it!

3 Coordinate System

Every 3D system imposes its own coordinate system conventions. These are arbitrary–everything that you'll learn in this course works equally well in any coordinate system, and it is straightforward to convert between them.

3.1 3D

In the 3D coordinate system used in this course, the x-axis increases to the East, the y-axis increases vertically upwards, and the z-axis increases to the South.

This is a **right handed** coordinate system. If you point your right hand in the direction of the x-axis and curl your fingers towards the y-axis (which necessitates having your palm upwards), then your thumb will be pointing along the z-axis. This works for any cyclic rotation of the order of axes, e.g., x-y-z has the same relationship as y-z-x.

3.2 World and Object Space

We distinguish between the absolute **world-space** (a.k.a. global) coordinates in which we will define the entire world (a.k.a. **scene**) and the relative **object-space** (a.k.a. body-space, local) coordinates used to define parts of an object relative to the reference frame of that object. For example, I might position a chessboard relative to the center of the scene, but the pieces on the board relative to the board itself.

By convention we will generally define object space coordinate systems in a common way. For objects that have a clear "top," we will make their object space y-axis point upward. For objects that have a natural "facing" direction, such as cars and people, we will define their object space z-axis to point out their back and the x-axis to point to their right. Thus objects look along their negative z-axis.

3.3 Rotations

The canonical rotations about the x-, y- and z-axes are called **pitch**, **yaw**, and **roll**. These also follow a right hand rule: if you point your thumb in the direction of the axis of rotation, your fingers curl in the direction of increasing angular measure.

3.4 2D

In the 2D coordinate system used in this course for images and the screen, the origin is at the upper-left corner. The x-axis increases to the right and the y-axis increases downward. The reading discusses the historical origin if this coordinate system.

Image space coordinates are sometimes expressed in pixel side-lengths, e.g., position (100, 120) on a 1920×1080 image. At other times they are in normalized so-called **texture coordinates**, in which (1, 1) is the lower-right corner of the image regardless of its resolution or aspect ratio. Texture coordinates are often expressed using the variables (u, v) or (s, t) to distinguish them.

3.5 Units

We use SI units (e.g., meters, seconds, Joules, Watts), which include radians as the unit of planar angle measure.

4 Doxygen

Write entry point (class, method, function, macro, enum, typedef) documentation and final report as **Doxygen** comments inside your C++ header (`.h`) files and standalone `.dox` files, all stored in the `source` directory.

Put images and other files referenced from your documentation in the `doc-files` subdirectory. iCompile will copy them when you build documentation.

Like HTML and LaTeX, Doxygen is a markup language that you use to *edit* a document. To actually *view* the document, you must compile it. To compile the document, execute the command `doxygen` with no arguments in the directory containing a file named `Doxyfile`. The Doxyfile that you will use for all projects is provided on the course webpage. You never need to modify it, although you may if you wish.

The following is a brief overview of some of the features of Doxygen. Read the manual [van Heesch 2010] for full details. **Sharing markup tips and helping classmates with formatting is one way to earn class participation, so please collaborate on this and let me know at the end of your report if you gave or received assistance.**

4.1 Markup

Doxygen comments begin with `/**` and end with `*/`. They apply to the entry point immediately following the comment. Only the markup in your header files and in .dox will affect your generated documentation.

An example of how to document a class is:

```
/** Represents a direction and magnitude in 3D. */
class Vector3 {
public:
    /** Distance along the x-axis. */
    float x;

    ...

    /** Magnitude of the vector. */
    float length() const;
};
```

You may have exactly one `\mainpage` markup command throughout your program. This declares that the containing comment forms the `index.html` page that will be your report. Put this in a .dox file in the source directory, e.g.,

```
/**
\mainpage

<b>Project 0: Cubes</b>
<br>Ephram Williams

\section outline Code Outline
App::onInit loads the scene...

\htmlonly
<center><img src="architecture.png" width=100></center>
\endhtmlonly

\thumbnail{result1.jpg}
...

*/
```

4.2 Style

Doxygen markup commands begin with a backslash. Some useful ones are `\sa`, `\brief`, `\param`, `\author`, and `\return`. Doxygen also allows creation of nested lists using leading dashes and hash marks, and some HTML commands work as well. You can escape to raw HTML by creating a `\htmlonly`...`\endhtmlonly` block. See the manual for more markup commands.

4.3 Links

Doxygen will automatically hyperlink URLs and the names of entry points (e.g., methods, functions, classes, and variables) in your project. Make sure to check these links in your report–mispellings and incorrect capitalization can break them. Compare the G3D header source code and the generated documentation for a page, and remember that you can mine the G3D source for examples of how to achieve specific effects.

4.4 Equations

Within Doxygen comments, you can format standalone equations using LaTex markup inside blocks bracketed by `\f[` and `\f]`. For inline equations, use `\f$` to both begin and end the block. As an example, the following Doxygen source:

```
\f[ \int_{0}^{2\pi} \!\! \int_{0}^{\pi/2} \cos \theta ~ d\theta ~ d\phi = \pi \f]
```

Embeds this equation in your document:

$$\int_0^{2\pi} \int_0^{\pi/2} \cos\theta \, d\theta \, d\phi = \pi$$

I recommend Andrew Roberts' LaTex math tutorial [Roberts 2009] if you are unfamiliar with LaTeX.

If your LaTeX code contains an error, Doxygen may cache the erroneous result, which makes it hard to debug. When you suspect that this is happening, use `icompile --clean` to clear the cache.

5 G3D

The **G3D Innovation Engine** is an open source C++ library for 3D graphics on Windows, Linux, and OS X. It is used in commercial games, research papers, military simulators, and university courses. G3D supports hardware accelerated real-time rendering using OpenGL, off-line rendering like ray tracing, and general purpose computation on GPUs.

No 3D developer programs directly on the C++ standard library and OpenGL or DirectX. They are at too low of a level and don't provide necessary facilities such as scene management, image I/O, GUIs, and platform abstraction. Instead, programmers adopt "engines" packaged as libraries that provide those features.

G3D is similar to the 3D engines that you would find in a film or game company, but it has been tailored for research and education. In particular, G3D has a modular design that allows you to replace components with ones that you built yourself, and because the full source code is available it provides about 200k lines of sample code (in addition to the samples that are in the documentation).

See the latest version of the G3D manual [McGuire 2012] for detailed information about the library.

6 Working from Home

I only support working on the department Mac computers in TCL 216 and the Special Purpose Lab using Emacs, gdb, and g++/iCompile with G3D and the libraries it includes.

However, you are *permitted* to use any development tools (such as Xcode), computer (such as your own laptop), or operating system (such as Windows) in this course. Beware that if you run into trouble, I'm probably going to tell you to use the CS department computing environment.

G3D 9.00 beta for Windows / Visual Studio 2010 and OS X / gcc is available from the G3D Subversion server (which is different than the course subversion server). See `http://g3d.sf.net` for information. Make sure that you use the 9.00 top of tree built from source, not the public release 9.00 beta binaries. Installation and use instructions are included with the library

The Visual Studio 2010 Express IDE for Windows is a free download from Microsoft. The OS X developer tools including gcc are a free download from Apple.

The course subversion server is available outside the department and from off campus. Beware that deadline timestamps are based on the server's clock, not your client machine's clock.

G3D Windows and OS X are 100% compatible. For my own research I move the same code between Windows and OS X on a daily basis. So you should be able to move fluidly between IDEs and operating systems on the same project.

References

COLLINS-SUSSMAN, B., FITZPATRICK, B. W., AND PILATO, C. M. 2008. Subversion complete reference. In *Version Control with Subversion*. O'Reilly, ch. 9. `http://svnbook.red-bean.com/en/1.5/svn.ref.html`. 1, 3

MCGUIRE, M., Ed. 2012. *The G3D 9.00 beta Manual*. September. `http://graphics.cs.williams.edu/courses/cs371/f12/G3D/manual`. 1, 9

ROBERTS, A., 2009. Getting to grips with Latex - Mathematics, December. `http://www.andy-roberts.net/misc/latex/latextutorial9.html` and `http://www.andy-roberts.net/misc/latex/latextutorial10.html`. 1, 8

VAN HEESCH, D., 2010. Doxygen 1.7.1 manual. `http://www.stack.nl/~dimitri/doxygen/manual.html`. 1, 7

Index

.dox file, 7
.svn, 3

check out, 2
commit, 2
coordinate system, 6

doc-files, 7
Doxyfile, 7
Doxygen, 7

G3D, 9, 10

header file, 7
HTML, 7

iCompile, 5, 7

LaTeX, 7

object-space, 6

pitch, 6

repository, 2
revision control system, 2
right handed, 6
roll, 6

scene, 6
Subversion, 2

texture coordinates, 6

update, 2

Visual Studio, 1, 10

Windows, 10
workspace, 2
world-space, 6

Xcode, 10

yaw, 6