CS371 Tools Overview

Updated September 11, 2014

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 2014; Roberts 2009; Collins-Sussman et al. 2008; McGuire et al. 2014], 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.

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 this course, 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 at home, for future courses, and in future work environments because they are cross-platform and freely available. You are welcome to use any tools that you wish on CS371 projects, including Xcode and Visual Studio, although you must keep all code on our Subversion server because that is how I monitor and evaluate your projects. We also use some content editing software. There is no credible open source version available for this, so there we use commercial tools.

Our supported OS X tools in the CS TCL216 and TCL312b labs and the recommended, but unsupported alternatives, on Windows are:

OS X Tool	Version	**Windows Tool**	Version
Mac OS X	10.9.4	Windows	7 or 8.1
G3D	10 beta (svn head)	G3D	10 beta (svn head)
bash	3.2.51(1)	CMD	
svn	1.7	TortoiseSVN	1.8.8 (svn 1.8.10)
clang++ (llvm)	4.1 (3.4svn)	Microsoft Visual Studio	2012 (express is ok) 64-bit
icompile	0.5.19	”	”
emacs	24.3.1	”	”
lldb	310.2.37	”	”
doxygen	1.8.7	doxygen	1.8.7
latex	TeX Live 2014	MiKTeX	5345
python3	3.4.1	python3	3.4.1
Photoshop	CreativeCloud	Photoshop	CreativeCloud
iMovie	10.0.4	Windows Movie Maker	2012

I've tested the above tools on our projects. In most cases, any newer version of the tools is also likely to work. G3D is not officially supported on Linux, so I don't recommend using that operating system. Room TCL217 has all of the above software except for Photoshop. It is occasionally useful to have 3DS Max installed if working on a personal machine. This is free for students from Autodesk, but is not installed on CS machines.

Contents

1 Configuring your Development Environment

1.1 Environment Variables

Every time that you open Terminal, the bash script `~/.bash_profile` executes. Ensure that yours[1] contains the following lines to include the locally installed copy of the G3D Innovation Engine and iCompile:

```
G3D10=/usr/mac-cs-local/share/cs371/G3D
export INCLUDE=$G3D10/include
export LIBRARY=$G3D10/lib
export PATH=$G3D10/bin:/opt/local/bin:/usr/local/texlive/2014/bin/x86_64-darwin:$PATH
export G3D10DATA=$G3D10/data

# Don't paste the slashes on the next line. They are there because it is a
# single long line that you should concatenate together in your editor
export ICE_EXTRA_SOURCE=$G3D10/source/GLG3D.lib/source:\
$G3D10/source/GLG3D.lib/include/GLG3D:$G3D10/source/G3D.lib/source:\
$G3D10/source/G3D.lib/include/G3D:$ICE_EXTRA_SOURCE

# Optionally, set your prompt to show the current directory
PS1="\w -> "

# Optionally set permissions
umask 077
```

It also must have the executable bit set (`chmod u+x .bash_profile`). You should set your default file permissions (via umask) to make your files readable only by you. Otherwise, the files that you create on the local scratch disk will be visible to everyone who logs in.

On Windows, you must set the `G3D10DATA` environment variable (search for "environment variable" on your local machine and you'll find the relevant control panel dialog) and add the G3D bin (or build/bin if building G3D yourself) directory to your `PATH` variable so that programs can find the DLLs. You must then set the Visual Studio include directories as described in the G3D documentation.

Changes to environment variables do not take effect until you restart your shell and applications because they are inherited at launch from the parent process's environment. Logging out is not necessary.

1.2 Editor

Our coding conventions mandate **four-space indenting and no tabs**. This is really important. If you don't follow this, when someone (such as the professor) else checks out your code and looks at it, it will become unreadable because the lines will appear randomly indented. In emacs, put the following lines in your `~/.emacs` file for proper tabs:

```
(setq-default indent-tabs-mode nil)
(setq tab-width 4)
(setq c-basic-offset 4)
```

[1] I could have automatically configured your login environment and provided scripts for everything else. I'm instead telling you how to set it up yourself because you're now learning to be a sophisticated programmer. You need to know your way around the details of an operating system, and when a script or command does something for you, you should figure out what it is doing and how it works. That will enable you to diagnose problems quickly and to automate tedious tasks in the future. In the first project lab report, I will ask you what each of these lines does and why they are needed.

When editing `.Any`, `.pix`, `glsl`, and `.vrt` files in emacs, you can receive syntax highlighting and proper indenting by entering C++ mode. You can force a file to open in C++ mode by ensuring that the following string appears on the first line (e.g., in a comment): `-*- c++ -*-`.

In Visual Studio 2012, you can set indenting to four spaces from Tools → Options → Text Editor → C/C++ → Tabs. To enable syntax highlighting, at Tools → Options → Text Edito → File Extensions, associate the following (one at a time) with Microsoft Visual C++: any vrt pix geo glsl. There are some templates and hooks for the Visual Studio debugger found in G3D's `bin` directory that are also very helpful.

1.3 G3D

When installing G3D on **your own computer**, I recommend checking it out from SourceForge and building yourself instead of installing a public release. Building locally will give you a copy that you can update throughout the semester and will give you proper symbol files for debugging. Check it out via the command:

```
svn checkout https://svn.code.sf.net/p/g3d/code/G3D10 G3D10
```

(*note the https://...not svn:// for G3D*), and then typing `buildg3d update` to run the Python script that creates the library, data, and documentation. You can svn update and then run this build command again at any time for an incremental build. These commands are the same on all platforms.

1.4 Testing

Your environment is correctly configured when you can copy the G3D "starter" project and compile and run it.

On OS X, simply make a new directory for your project, enter that directory, and run `icompile --run`. It will prompt you to create a new project...agree, and choose the G3D starter template. On Windows, copy `samples/starter` into your own directory and load it in Visual Studio as a new solution, not from `G3D.sln`. Select the 64-bit Debug configuration and then press F5 to compile and run.

On all platforms, running `doxygen` with no arguments in your project directory should generate entry point and journal documentation for your project.

2 Coding Conventions

Coding conventions are important because we're going to be sharing a lot of code, so we need it to fit together cleanly and clearly. They're also important because a large portion of your evaluation for each assignment is how readable your code is (to the professor).

In CS371, we follow the coding conventions of the G3D Innovation Engine, which are fairly standard in the field: four-space indenting, Capitalized CamelCase class names; lower-case camelCase variable and method names; ALL CAPS enumeration values, constants, and macros; no single-line `if`, `for` or other blocks without {curly braces}; and the prefix `m_` on private member variables (the last is optional, but really convenient).

Indent variable declarations, including method arguments, to form clean columns. These are much easier to scan and to edit with block commands.

Use Doxgen /** ... */ C comments for method documentation in header files and single-line // C++ comments for everything else. Write code that is easy to understand and read. Use comments sparingly and mostly to explain algorithms and weird cases–if your code is well-written and has good variable names and structure, it should not require comments.

Prefer pre-increment `++i` to post increment `i++` because otherwise you'll do something really expensive with a C++ iterator by accident some day. Use G3D's data structures `Array, String, Table,` etc., that are tuned for performance, or the std standard library's ones when there's something obscure (like a priority queue) that you require.

Look at G3D's source code for lots of examples.

3 Subversion

3.1 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.

3.2 Can I use Git, Mercurial, Perforce, AccuRev, Plastic SCM, etc.?

No. Mercurial is superior to Subversion, but is not supported by the CS department. The others are much harder to use for small team projects. Even if you disagree, they aren't supported by the department either. So much as mentioning Git in particular to the professor is grounds for automatically failing the course.

3.3 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 [-u]`

To tell Subversion to ignore a file, use:

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

For, example,

```
svn propset svn:ignore log.txt g3d-license.txt
```

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.

Before you log off of your computer and end a programming session, run `svn -u status` to verify that you have no modified (M) or unknown (?) files. If you run `svn -v status`, then it will confirm all of the files that are in the repository as well.

3.4 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 with group `eph`, then the commands for this process would be:

```
cd /Volumes/scratch
mkdir ewilliams
cd ewilliams
svn co svn://graphics-svn.cs.williams.edu/cs371-projects/1-Meshes/meshes-ewilliams meshes-eph
svn export --force svn://graphics-svn.cs.williams.edu/cs371-projects/0-Cubes/cubes-ewilliams meshes-eph
cd meshes-ewilliams
svn add *
svn commit -m "Exported from previous week"
```

4 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.

`--lldb [... 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.

4.1 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:

`source`	All of your source code, divided among `.h` and `.cpp` files, and the source for your report and overview documentation in `.dox` files.
`data-files`	Any runtime data required by your program that is *not* also in the G3D data distribution.
`journal`	Your development log, including images and videos. Press F4 or F6 in G3D to automatically save screenshots and videos to here, with a log entry.
`doc-files`	Any data required for your report, such as images and videos. Do *not* put the `.dox` files here.
`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!

5 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.

5.1 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.

5.2 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.

5.3 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.

5.4 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.

5.5 Units

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

6 lldb

lldb is the llvm debugger. You can launch it from iCompile using `icompile --lldb` or from the command line directly. Below are the most common commands that I use when debugging. The GDB to LLDB reference contains more exotic ones should you require them. lldb has a complete Python interpreter within it, so if you're familiar with that language you can use it to run very sophisticated command sets on breakpoints or just interact using Python syntax instead of debugger-specific syntax.

Run (or restart) the program	r
See the call stack (backtrace)	bt
Change the current stack frame to the third back from the top of the stack	f 3
Show all variables and function arguments in the current frame	fr v
Show the value of variable name in the current stack frame	p name
Set a breakpoint at all functions named "main"	b main
Set a breakpoint in file main.cpp at line 10	b main.cpp:10
List all breakpoints	breakpoint list
Delete breakpoint #2 from the list	breakpoint delete 2
Step over the next expression (don't recurse into it)	s
Step into the next expression (recurse into function calls)	n
Continue execution until the end of the current call frame	finish
Continue execution until the next breakpoint or assertion	c

7 Profiling

Focus on making your code correct and clear before worrying about performance. C++ and G3D are both very efficient, and if your program is well-designed it will naturally perform well. About halfway through the course performance will become more of a concern because we'll be working with large assets, and at that point you may wish to have a tool to explore your program's behavior more rigorously.

There are a few profilers available that will work well with G3D. The first is built right into the engine and is accessible from the Developer Window (push the clock icon). This profiles both CPU and GPU code surrounded by the `BEGIN_PROFILER_EVENT` and `END_PROFILER_EVENT` macros or GPU shaders launched by `LAUNCH_SHADER`.

On OS X, you may also use the Apple OpenGL Profiler. On Windows, the free Very Sleepy CPU profiler and the free NVIDIA Nsight GPU profiler are very good and both work with G3D. Nsight also works as a GPU debugger, allowing single stepping through shaders and inspecting draw calls.

8 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 2014] for full details, and see the examples in the G3D starter project's default `journal.dox` file. **Sharing markup tips and helping classmates with formatting is one way to earn class engagement points, so please collaborate on this and let me know at the end of your report if you gave or received assistance.**

8.1 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

...

*/
```

8.2 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.

8.3 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.

8.4 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.

9 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. The GPU libraries also change too frequently. For example, OpenGL will go through two complete overhauls in the next two years–any details that you learn about that library today would be useless when you next tried to use it.

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 250k lines of sample code (in addition to the samples that are in the documentation). It is very fast, but where a choice must be made, it always chooses flexibility over performance.

One way in which G3D is very fast is that it can load assets in many formats. This makes it easy to incorporate content from the extensive data library or that you find online into your projects. The **G3D viewer** tool allows you to preview most content that G3D can load, including PNG, JPG, BMP, and TGA images and IFS, 3DS, OBJ, Collada (dae), and FBX 3D models. On OS X, you can launch the viewer from the command line using `viewer`, and on Windows it is in the `bin` directory. Once launched, you can drag content onto it. For 3D models, use the W, A, S, and D keys to translate the camera, and the mouse + right button to rotate the camera (you can also use a game pad such as an Xbox360 controller). Pushing F3 gives information about the model and clicking on it identifies individual meshes.

See the latest version of the G3D manual [McGuire et al. 2014] for detailed information about the library.

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, 7

MCGUIRE, M., MARA, M., AND OTHERS. 2014. *The G3D 10.00 beta Manual*. September. 1, 15

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, 14

VAN HEESCH, D., 2014. Doxygen 1.8.2 manual. http://www.stack.nl/~dimitri/doxygen/manual.html. 1, 13