CMSC 23700 **Introduction to Computer Graphics** **Handout 2**
Winter 2008 **January 9**

Lab tips

This handout provides an introduction to some of the tools you will use to complete the programming projects.

Getting Started

You will need an account on the CS machines (this is different from your harper account). If you do not already have one, you can request one at

`www.cs.uchicago.edu/info/services/account_request`

We recommend that you use the Intel Macs in the Mac Lab for your project. These machines support OpenGL 2.0 and have good-quality graphics cards. Apple's Developer tools also include useful tools for debugging and optimizing your OpenGL applications.

It may be possible to use the Mac Lab linux machines too, but we have not yet verified that they support a recent enough version of OpenGL.

Using OpenGL and GLUT functions in your C programs

In order to use OpenGL and GLUT functions in your program you will need to include the appropriate header files. The file `glut.h` header file includes the OpenGL header files (`gl.h` and `glu.h`), so it is the only one you will need to include. Unfortunately, Linux and MacOS X differ in where they put the `glut.h` file. The following bit of preprocessor code will allow your program to compile on both platforms:

```
#if defined(__APPLE__) && defined(__MACH__)
#  include <GLUT/glut.h>
#else
#  include <GL/glut.h>
#endif
```

Compiling under MacOS X

MacOS X also uses **gcc** as its default C compiler. Apple uses a different set of linking flags from Linux. To link an OpenGL program, you need the following linker flags:

```
-framework GLUT -framework OpenGL -framework Foundation
```

Apple provides an IDE (called **Xcode**) that you may use to develop your project, but please include a makefile in you submissions.

Compiling under Linux

On Linux systems, you should use the default version of **gcc** (version 4.1), which accessed using the path `/usr/bin/gcc`. To compile and link an OpenGL program under Linux, you must use the following linking options:

```
-lglut -lGL -lGLU -lm
```

Doxygen

For this course, it is not sufficient to just write working code, you code should be well organized and documented. To help with the latter, we will use the Doxygen tool (`doxygen.org`) to generate documentation from source code. Specifically, we expect you to document each file, data structure, and function, as well as other significant definitions (*e.g.*, macros). You will be expected to hand in a hardcopy of your documentation as part of the project submission.

Makefiles

For each of your projects, you should include a makefile in your submission. We will provide a skeleton makefile for you, but you are responsible for maintaining it. For a simple project, such as Project 0, that contains only a single source file, the following makefile will suffice:

```
SHELL =                 /bin/sh

ifeq ($(shell uname -s),Darwin)
  CC = cc -std=gnu99
  LDFLAGS = -framework GLUT -framework OpenGL -framework Foundation
else
  CC = gcc -std=gnu99
  LDFLAGS = -lglut -lGL -lGLU -lm
endif

project0:        main.c
        $(CC) $(CFLAGS) -o project0  main.c $(LDFLAGS)

clean:
        rm -rf project0
```

This makefile works on both Linux and MacOS X by setting the LDFLAGS make variable based on the host OS. If you have not used **make** before, you should take a look at the documentation. Information about make is available at `www.gnu.org/software/make` and online documentation can be found at `www.gnu.org/manual/make/html_chapter/make.html`.

Gforge

We are using a new system to keep track of projects called gforge. A server has been set up with hostname `cmsc23700-gforge.cs.uchicago.edu`. You can access it using your web browser at that address.

Before you can have a project, you need to register yourself as a user. Do this by pointing your web browser to `http://cmsc23700-gforge.cs.uchicago.edu/` and clicking on the link "New Account" on the top right corner of the page. Follow the directions (you only need to fill in the starred fields) and submit the form. In a few seconds you should receive an email with a link to confirm your registration. Click on it and log in and you should have an active account. When this is complete, email me at `jriehl@cs.uchicago.edu` with the account name you just set up.

Using Subversion

Once a project is created for you, it will have a Subversion repository on the server. You are expected to keep the source code of your projects in the repository. To *checkout* a copy of a project called foo, run the following command:

```
svn checkout svn://cmsc23700-gforge.cs.uchicago.edu/foo
```

On your first checkout, you should be prompted for your password. It will assume you are using the username of the account executing svn. If your gforge username is different than this name, just press enter on the password prompt and it will then ask you for your username first and then your password. If everything checks out, a directory called `foo` will be created in the current directory. All the files related to your project should live in this directory.

Now suppose you create a file called `main.c` in your `foo` directory. In order for Subversion to keep track of it, it needs to be added to the repository. You do this using the following command:

```
svn add main.c
```

You should see a message like:

```
A         main.c
```

This command records the fact that `main.c` has been added to the repository, but the file will only be added when you commit your changes. To do so, type the following command:

```
svn commit
```

to add the file permanently to the repository. You will be prompted to enter a log message in an editor. You can avoid editors altogether by typing your log message on the command line with the `-m` flag:

```
svn commit -m "added files"
```

After you have entered your message, you will see a message like the following:

```
Adding         main.c
Transmitting file data .
Committed revision 1.
```

Changes you make to your files are recorded in the repository every time you do a `svn commit`.

Before you make changes to your files, you can ensure that you have a current version, by running `svn update`. This fact is not of tremendous significance for individual projects, but matters when more that one person can modify the same files.

Not all the files in your project directory need to be in the repository. For example, you should not put your executable files in the repository — these can always be recreated (hopefully!) by compiling the source.

The "`svn diff`" command is for comparing differences between versions. If no files (or options) are specified, all working files are compared to their last committed versions, otherwise only the specified files are compared. There are also flags to compare other versions, see the man pages or the online manual for details.

Useful resources

There are links to some useful Computer Graphics resources on the course web page at

`www.classes.cs.uchicago.edu/archive/2008/winter/23700/`

Information about make is available at `www.gnu.org/software/make/` and online documentation can be found at `www.gnu.org/manual/make/html_chapter/make.html`.

The Doxygen home page is at `http://doxygen.org` and it includes an online manual.

The Subversion home page is at `http://subversion.tigris.org/`. Official documentation is at `http://svnbook.red-bean.com/`.