

CSC 4356 / ME 4573 Interactive Computer Graphics

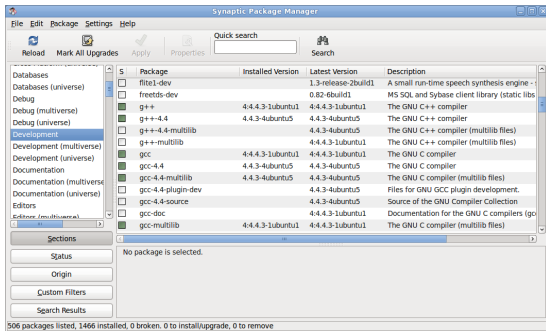
Project Management

OpenGL project with GLUT and GLEW

The next few slides give gcc, GLUT, and GLEW setup procedures for Linux, Mac OS X, and Windows.

- This setup need only be done once at the beginning of the semester.
- We'll not be using a specific IDE. You may use whatever text editor you prefer.

Getting set up with Linux



Your system package manager has everything.
Install gcc, glut-dev, libglew-dev. Dependencies will follow.

Getting set up with Mac OS X



Xcode 4.0+ can be downloaded from the App Store.

GLUT and GLEW for Mac OS X

GLUT is already installed.

The most trouble-free solution is to install the **MacPorts** package manager. This will add a command port to your terminal.

```
sudo port install glew
```

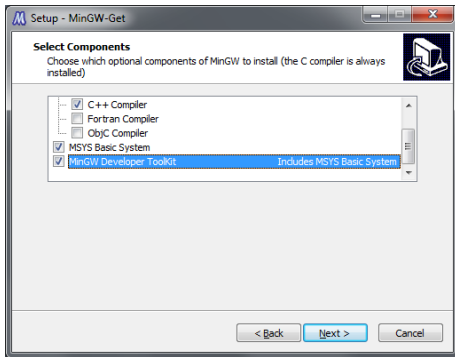
This will be especially valuable in the future, as we add capabilities and require new libraries.

Getting set up with Windows

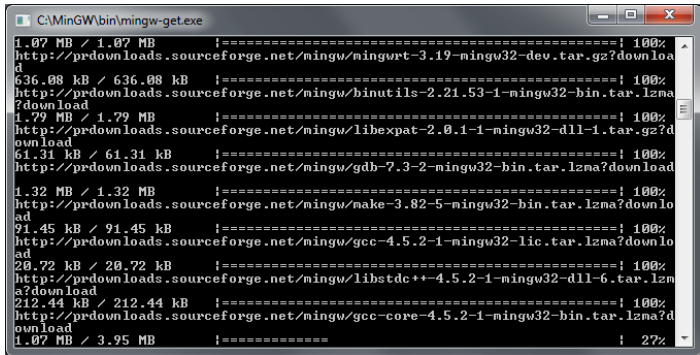
Download and install **MinGW**, the GNU compiler toolkit for Windows, using the “Automated MinGW Installer.”

If you prefer MS Visual Studio, download it from **Microsoft DreamSpark**. These notes won't cover it, however.

MinGW Setup



Select the C++ compiler, MSYS, and the Developer Toolkit.



```
C:\MinGW\bin\mingw-get.exe
1.07 MB / 1.07 MB |=====| 100%
http://prdownloads.sourceforge.net/mingw/mingwrt-3.19-mingw32-dev.tar.gz?download
636.08 kB / 636.08 kB |=====| 100%
http://prdownloads.sourceforge.net/mingw/binutils-2.21.53-1-mingw32-bin.tar.lzma?download
1.79 MB / 1.79 MB |=====| 100%
http://prdownloads.sourceforge.net/mingw/libexpat-2.0.1-1-mingw32-dll-1.tar.gz?download
61.31 kB / 61.31 kB |=====| 100%
http://prdownloads.sourceforge.net/mingw/gdb-7.3-2-mingw32-bin.tar.lzma?download
1.32 MB / 1.32 MB |=====| 100%
http://prdownloads.sourceforge.net/mingw/make-3.82-5-mingw32-bin.tar.lzma?download
91.45 kB / 91.45 kB |=====| 100%
http://prdownloads.sourceforge.net/mingw/gcc-4.5.2-1-mingw32-lic.tar.lzma?download
20.72 kB / 20.72 kB |=====| 100%
http://prdownloads.sourceforge.net/mingw/libstdc++-4.5.2-1-mingw32-dll-6.tar.lzma?download
212.44 kB / 212.44 kB |=====| 100%
http://prdownloads.sourceforge.net/mingw/gcc-core-4.5.2-1-mingw32-bin.tar.lzma?download
1.07 MB / 3.95 MB |=====| 27%
```

Setup will download and install many packages.

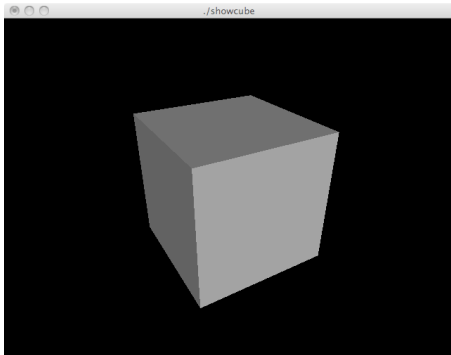
Use whatever text editor you prefer to edit code.

Use *All Programs / MinGW / MinGW Shell* to build.

GLUT and GLEW for MinGW

Stay tuned for the Windows PITA 2011.

Building Projects



We begin with a simple project that renders a cube.



cube.h



glut.h



glew.h



gl.h



main.c



cube.c



showcube



main.o



cube.o



libglut.a

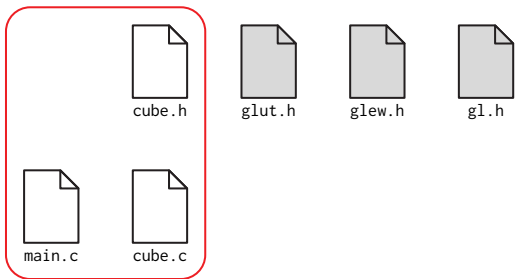


libGLEW.a

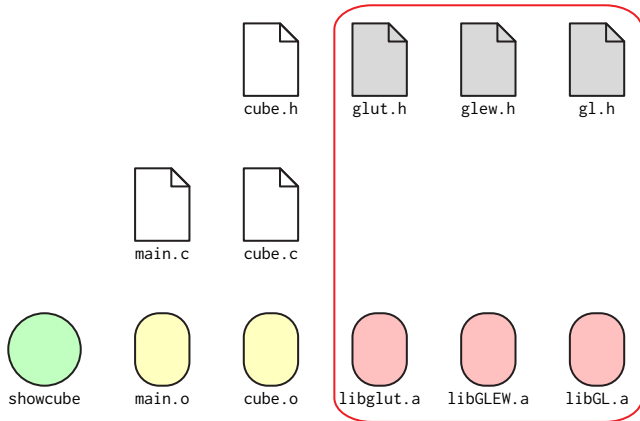


libGL.a

These are all of the files involved in the showcube example.



These files contain the C code that you create.



These are *system* files, installed globally, read-only.



cube.h



glut.h



glew.h



gl.h



main.c



cube.c



showcube



main.o



cube.o



libglut.a

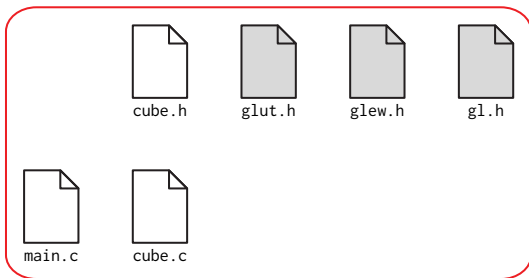


libGLEW.a



libGL.a

These are the *output* generated by the build process.



`showcube`



`main.o`



`cube.o`



`libglut.a`



`libGLEW.a`



`libGL.a`

These files are human-readable *C source* code.



cube.h



glut.h



glew.h



gl.h



main.c



cube.c



showcube



main.o



cube.o



libglut.a

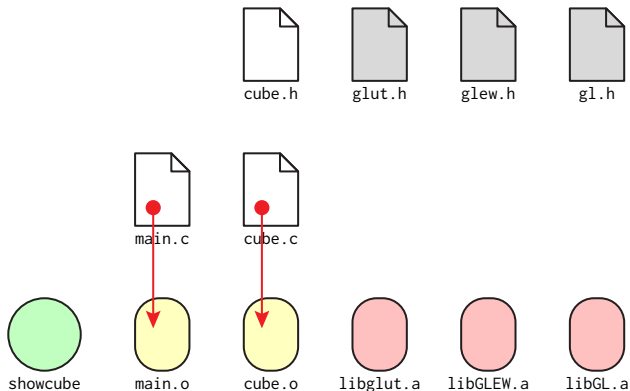


libGLEW.a

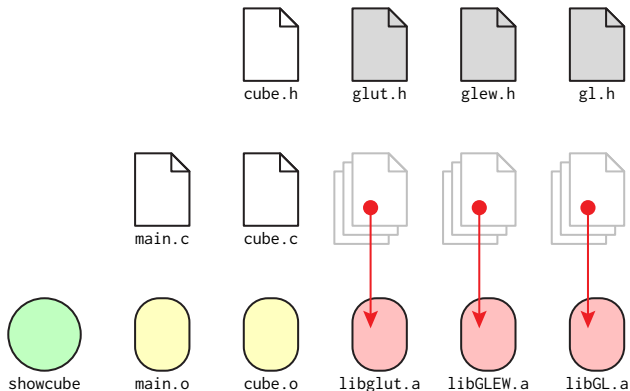


libGL.a

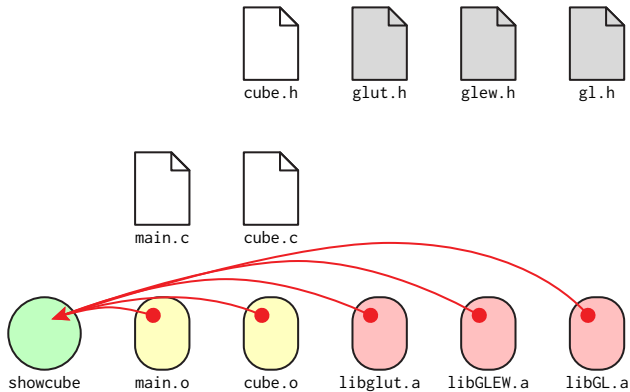
These files are machine-readable *binary* code.



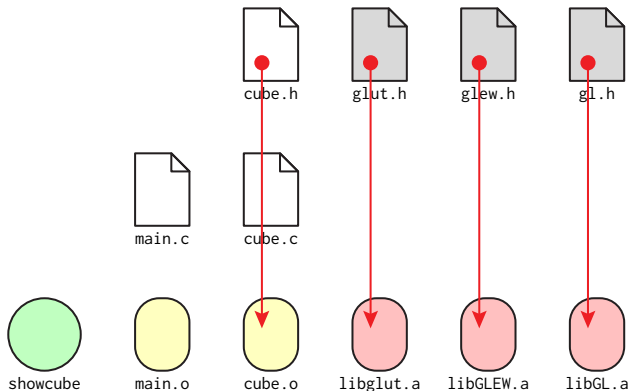
Each C source file is compiled to a binary *object* file.



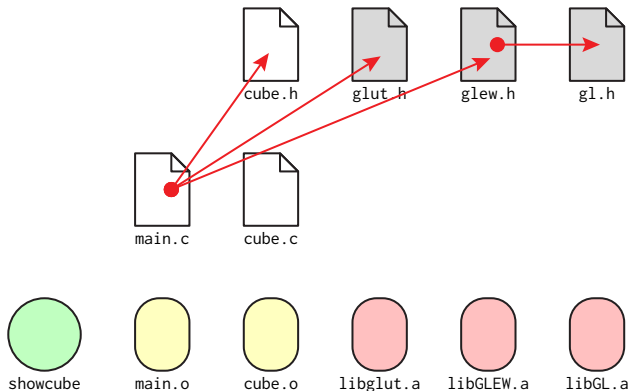
The libraries were produced in the same way, and *archived*.



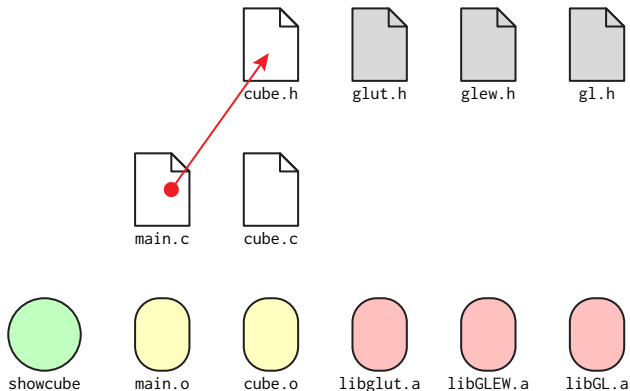
These objects are *linked* giving the final *executable*.



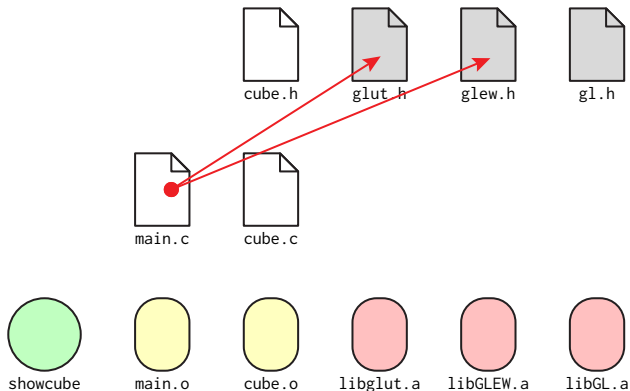
A header is C source listing the contents of a binary object.



Any source that *uses* an object must *include* the header.

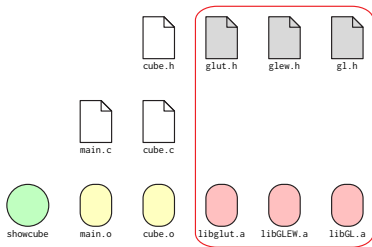


Local headers: `#include "cube.h"`



System headers: `#include <glut.h>`

System files are installed in global, read-only locations where the compiler knows to find them.



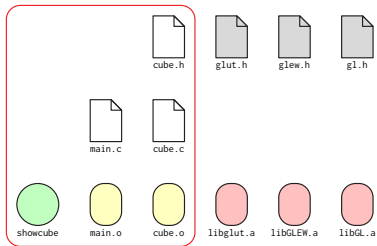
System headers in...

- `/usr/include`
- `/usr/local/include`

System libraries in...

- `/usr/lib`
- `/usr/local/lib`

Local files, including your sources and generated objects, may be wherever you want to put them.



But do be organized and keep each project in a *separate* directory.

Using gcc

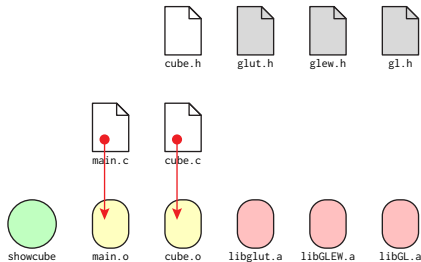
For now, we'll use gcc, the GNU C Compiler, by invoking it on the command line.

- Windows: *All Programs / MinGW / MinGW Shell.*
- Mac OS X: *Applications / Utilities / Terminal.*
- Linux: *Terminal Emulator* *

* Which goes by different names in different distributions.

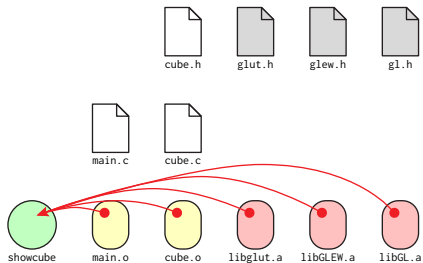
Invoke gcc with -c to compile source into object.

```
gcc -c main.c  
gcc -c cube.c
```



Invoke gcc with -o to link objects to executable.

```
gcc -o showcube main.o cube.o -lglut -lGLEW -lGL
```



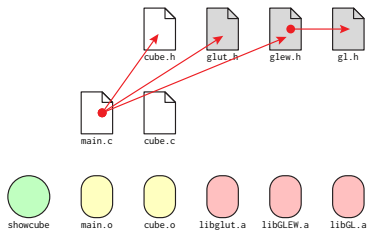
Note, there is a discrepancy between objects and libraries...

```
gcc -o showcube main.o cube.o -lglut -lGLEW -lGL
```

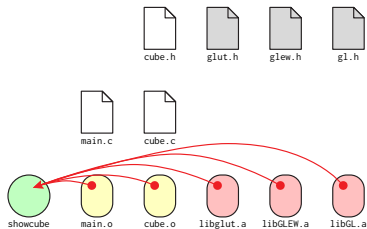
System libraries are listed using the `-l` argument.

This allows the compiler to flexibly search for and select an appropriate version of the library.

Don't forget, to use a library you must *both*



include the header



and link the library.

Automating the Build

It is tedious and annoying to type these compilation commands repeatedly.

We can automate this using an intelligent tool called make.

The following slides show several examples of automatic build definitions, of increasing complexity.

This file is called Makefile. It contains a recipe for each step of the build process.

```
showcube : main.o cube.o
    gcc -o showcube main.o cube.o -lglut -lGLEW -lGL

main.o : main.c
    gcc -c main.c

cube.o : cube.c
    gcc -c cube.c
```


Each recipe has exactly the same form.

Target File : *Dependent Files*
Command

The *Command* performs some action upon the *Dependent Files* to produce the *Target File*.

make is careful to ensure that the *Command* is executed *only* for those *Target Files* whose *Dependent Files* have been modified.

Note that the two C source files are built in exactly the same fashion. We can write a single, generic recipe for all such cases.

```
showcube : main.o cube.o
    gcc -o showcube main.o cube.o -lglut -lGLEW -lGL

%.o : %.c
    gcc -c $<
```

The `$<` token is a special variable that gives the dependents.

Variables allow us to generalize the recipes and eliminate repetition.

```
CC= gcc
EXEC= showcube
OBJS= main.o cube.o
LIBS= -lglut -lGLEW -lGL

$(EXEC) : $(OBJS)
    $(CC) -o $(EXEC) $(OBJS) $(LIBS)

%.o : %.c
    $(CC) -c $<
```

Targets may perform tasks other than building. Here, `clean` removes all generated files, leaving a clean source directory.

```
CC= gcc
EXEC= showcube
OBJS= main.o cube.o
LIBS= -lglut -lGLEW -lGL

$(EXEC) : $(OBJS)
    $(CC) -o $(EXEC) $(OBJS) $(LIBS)

%.o : %.c
    $(CC) -c $<

clean :
    rm $(EXEC) $(OBJS)
```

Configurability

Different OSs often require different build parameters.

```
# Linux
# LIBS= -lGLEW -lglut -lGL

# Mac OS X
# LIBS= -lGLEW -framework GLUT -framework OpenGL

# Windows
# LIBS= -lglew32 -lglut -lopengl32 -lwinmm -lgdi32
```

Remove the comment marker to enable one of these.

Complex Build Logic

Advanced features of make allow for complex flow control. This example automatically determines the current OS and builds the software accordingly.

```
ifeq      ($(shell uname), Linux)
    LIBS= -lglut -lGLEW -lGL
else ifeq ($(shell uname), Darwin)
    LIBS= -lGLEW -framework GLUT -framework OpenGL
else
    LIBS= -lglut -lglew32 -lopengl32 -lwinmm -lgdi32
endif
```

(Don't bother with this unless you really want to.)

Coming up...

We get to the point: Computer Graphics.