Setting Up OpenGL, GLM, and FLTK

Description:
In this lab, your task is to set up your development environment for future labs and assignments that use OpenGL, the GLM library, and FLTK.

OpenGL is the standard (3D) graphics library. Although there are other graphics libraries out there (e.g. Vulcan, DirectX or Direct3D by Microsoft, Metal by Apple, and Mantle by AMD), OpenGL remains the most common and most popular API for implementing 3D graphics applications. For convenience, we will be using the GLM library (GLM stands for OpenGL Mathematics) to do all of our linear algebra. The library will help keep the code readable when manipulating matrices, vectors, points, and such.

Finally, we will be using FLTK (version 1.3.4) in conjunction with OpenGL. FLTK is a cross-platform, lightweight graphical user interface (GUI) library that supports OpenGL canvas. Since FLTK is cross-platform, you should be able to use your own favorite operating system and windows environment. What a cross-platform toolkit means is that you will write your GUI code once, and you can compile the code in Windows, Mac, and whatever operating system that FLTK supports. The resulting compiled program will be native to the operating system (whereas Java is also a cross-platform language, but Java applications are run in a virtual machine).

Personally, I use a Windows machine at home and at work. I have therefore written the initial support code and demo applications in Windows (using Visual Studio 2017 Enterprise edition). For this class, we will support Windows and Mac development environments. If you are using some version of Unix, I assume that “you know what you're doing.” Luckily, this setup only needs to happen once. As soon as you get things to run in today's lab, you should be good to go for the rest of this semester.

Your Task:

- You will need to set up your development environment.
- You will need to download and install FLTK.
- You will need to download and install GLM.
- You will need to show that you can run the example.cpp
- Once you get the example program running, your task is to add a new slider that:
  - Controls the line width when rendering in wire frame mode
  - The range of the possible values should go from 1 to 10
  - The default value should be 1
- You will be working alone in this lab.
- When you are done, show your work to myself or Matt!

Files Given:

example.cpp – a simple test program that uses OpenGL, GLM, and FLTK.

Compiling: (On Windows)

- Download and install Visual Studio 2019.
    - This application should be free to you. You can find it at: https://visualstudio.microsoft.com/. Download the 2019 Community edition (which is free).
    - The download and installation process could take a little while…
    - On the "workload" tab, select "Desktop development with C++".
    - When running Visual Studio for the first time and when asked about layout options, choose C++ development from the pull down menu.
- Installing FLTK
    - Download and install FLTK. You can find it at: https://www.fltk.org/software.php. Download the latest version, version 1.3.5. (Note that I'm still using 1.3.4, but from the change log I don't think much has changed).
    - After unzipping and un-tarring the file, go into the folder "ide -> VisualC2010" and double click on "fltk.sln". This will launch your Visual Studio.
    - Go to "Build -> Build Solutions"
        - Be Mindful that you are compiling "Debug" in "x86" (or"Win32")
    - In the menu bar area, change from "Debug" to "Release".
    - Again, go to "Build -> Build Solutions"
    - The above two steps will build both the debug and the release versions of the FLTK libraries.
    - (Optional) You can right click on any of the projects, select "Set as StartUp Project", and then click on the green right arrow in the menu (or go to "Debug -> Start Debugging"). Try it out with the application "cube"!
- Setting up headers and libraries
    - Go back to the folder where you have unzipped FLTK.
    - Copy the "FL" folder (all of it) into "C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\VS\include"
    - Go into the lib folder, copy the files: fltk.lib, fltkd.lib, fltkgl.lib, fltkgld.lib" into "C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\VS\lib\x86"
- Installing GLM
    - GLM is a "header only" library (i.e. there is nothing to compile). The only thing you need to make sure of is that the downloaded header files are put in the right place.
    - Download GLM from https://github.com/g-truc/glm/tags
    - The version I downloaded was 0.9.9.3, but you should download the latest (stable) version.

    - After unzipping the files, copy the entire "glm" folder (that is inside the parent 'glm' folder) into into "C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\VS\include"
    - You should notice the "FL" folder in the same directory as well.
- Compiling and Running example.cpp
    - Start Visual Studio. At the startup screen, select "continue without code" (which is a tiny link at the bottom right of the screen)
    - go to "File -> new -> Project"
    - Select "Empty Project". Fill in the requested info.
    - Right click on the project name, select "Add -> New Item -> C++ File"
    - Copy the contents of example.cpp to the new file
    - Right click on the project name, select "Properties"
    - Make sure that you are using the "Debug" configuration with X86 (NOT X64):
        - Go to "C/C++"
        - Select Preprocessor
        - Replace the "Preprocessor Definitions" with the following: _CRT_SECURE_NO_DEPRECATE;WIN32;_DEBUG;_WINDOWS; %(PreprocessorDefinitions)
        - Go to "Linker"
        - Select "Input"
        - Replace the "Additional Dependencies" with the following: fltkd.lib;fltkgld.lib;glu32.lib;opengl32.lib;comctl32.lib;%(AdditionalDependencies)
        - Now try running the program
    - Now switch to "Release" configuration with X86 (NOT X64):
        - Repeat the previous steps, but for the "Preprocessor Definitions", use the following:
        - _CRT_SECURE_NO_DEPRECATE;WIN32;NDEBUG;_WINDOWS; WIN32_LEAN_AND_MEAN;VC_EXTRA_LEAN;WIN32_EXTRA_LEAN;%(PreprocessorDefinitions)
        - For the linker, repeat the same, but use the "non-debug" versions of the FLTK libraries (note that the non-debug versions don't have the letter d in the names): fltk.lib;fltkgl.lib;glu32.lib;opengl32.lib;comctl32.lib;%(AdditionalDependencies)
        - Now try running the program
    - Note that if you go back to where you have set up your project, you will find a "Release" and a "Debug" folder. If you go into these folders, you will find an executable (.exe) file. You can double-click on them and run these applications as a native Windows desktop app.

Compiling: (On Mac using Terminal or xCode)
FLTK can be installed easily by using one of the following package managers: MacPorts or Homebrew.

Thereafter you can install FLTK with the corresponding commands for:

```
MacPorts: 'sudo port install fltk'
Homebrew: 'brew install fltk
```

FLTK provides a neat script named "fltk-config" that can provide all the flags needed to build FLTK applications using the same flags that were used to build the library itself. Architecture flags (e.g., -arch i386) used to build the library, though, are not provided by the fltk-config script. This allows the script to build universal libraries and to produce applications of any architecture from them. Running "fltk-config" without arguments will print a list of options. The easiest call to compile an FLTK application from a single source file is:

```
fltk-config --compile myProgram.cxx
```

However, we will be building fltk applications using openGL through fltk and the default "easy" call will not work. You must include the –openGL option as so:

```
fltk-config --use-gl --compile myProgram.cxx
```

When run, the script will perform a compile command similar to:

```
clang++ -I/usr/local/Cellar/fltk/1.3.4-2/include -D_LARGEFILE_SOURCE -
D_LARGEFILE64_SOURCE -D_THREAD_SAFE -D_REENTRANT -o 'example'
'example.cpp' /usr/local/Cellar/fltk/1.3.4-2/lib/libfltk_gl.a -framework
OpenGL /usr/local/Cellar/fltk/1.3.4-2/lib/libfltk.a -lpthread -framework
Cocoa
```

If you wish to use xCode for program creation, this information is very important. Under "<Project Name>" → Build Settings → Linking → Other Linker Flags add:

```
/usr/local/Cellar/fltk/1.3.4-2/lib/libfltk_gl.a -framework OpenGL
/usr/local/Cellar/fltk/1.3.4-2/lib/libfltk.a -lpthread -framework Cocoa
```

And under "<Project Name>" → Build Settings → Search Paths → Header Search Paths add:

```
/usr/local/Cellar/fltk/1.3.4-2/include
```

Once these are added to the project, you should be able to compile/run the lab code without issue.

Installation of glm can be accomplished using MacPorts or HomeBrew as well.

```
MacPorts: 'sudo port install glm'
```

Homebrew: 'brew install glm

As noted in the Windows section, GLM is a “header only” library (i.e. there is nothing to compile). In order to use glm within xCode, you must add the directory path of the installed library to “<Project Name>” → Build Settings → Search Paths → Header Search Paths. If installed by Homebrew, the path will look similar to:

/usr/local/Cellar/glm/0.9.9.3/include/glm

Once added, you should be able to include any of the associated files directly.