General Rules and Instructions

Plagiarism Note and Late Policy

Copying code (either from other students or from external sources) is strictly prohibited! We will be using automatic anti-plagiarism tools, and any violation of this rule will lead to expulsion from the class. Late submissions will generally not be accepted. In case of serious illness or emergency, please notify Julian and provide a relevant medical certificate.

Provided Libraries

For each assignment, you will use the geometry processing library `libigl`, which includes implementations of many of the algorithms presented in class. The `libigl` library includes a set of tutorials, an introduction to which can be found in `tutorial/tutorial.html`. You are advised to look over the relevant tutorials before starting the implementation for the assignments; you are also encouraged to examine the source code of all the library functions that you use in your code to see how they were implemented.

To simplify compilation, we will use `libigl` as a header-only library (note that, if you prefer, you can compile it into a set of static libraries for faster builds at your own risk—this can be brittle on some platforms). This way, all you need to do to install `libigl` is to clone it somewhere and point your `$LIBIGL_ROOT` there. If you don't want to deal with environment variables, `libigl` can also be found if you clone it into your assignment directory's parent directory. All dependencies of `libigl` are included as git submodules, so please follow the installation instructions below carefully (note the '–recursive' clone).

No libraries apart from `libigl` are permitted unless permission is granted in advance.

Installing CMake and LIBIGL

Before we can begin, you must install CMake, the system `libigl` uses for cross-platform builds. If you are using Linux or macOS, I recommend installing it with a package manager instead of the CMake download page. E.g. on Debian/Ubuntu: `sudo apt-get install cmake` or with MacPorts on macOS: `sudo port install cmake`.

Next, choose an installation location for `libigl`. As mentioned above, if you clone it into the parent directory of your assignment repositories, each assignment's CMake scripts will find it automatically. If you place it elsewhere, you will need to set the `$LIBIGL_ROOT` environment variable to point to the cloned `libigl` directory.

Finally, clone `libigl` and all of its dependencies:
`git clone --recursive https://github.com/libigl/libigl.git`. The `--recursive` is needed to bring in the dependencies, which are git submodules.

Cloning and Building Each Assignment

We will provide a repository for each assignment with some visualization and user interface code to get you started so that you can focus on implementing the algorithms presented in class. Included in each assignment repository is a CMake build system that should enable you to easily compile your code (assuming you set up `libigl` properly as described above).

For each assignment, you will need to do the following:

(1) Clone the assignment repository, e.g. by typing in a terminal:
`git clone https://github.com/NYUGP17/Assignment_1`
(2) Create a directory called `build` in the cloned assignment repository, e.g.:
`cd Assignment_1; mkdir build`
(3) Use CMake to generate the Makefiles needed for compilation inside the `build/` directory:
`cd build; cmake -DCMAKE_BUILD_TYPE=Release ../`
(4) Compile and run the executable, e.g.: `make && ./assignment_1 <some_mesh_file>`

If you encounter problems, please create an issue on the assignment's repository on GitHub.

Submitting Your Work

You will submit your finished work using our class' GitHub organization. When you finish each assignment, you will

- Create a **private** repository in https://github.com/NYUGP17 called **AssignmentN_USER**, where `USER` is your github username that you entered in the class survey and `N` is the assignment number.
- Push your code to the repository:
`git push https://github.com/NYUGP17/Assignment1_USER`

Please follow this naming convention precisely, since we will use scripts to automatically collect and timestamp your work.

Grading. The code will be evaluated on Linux, so even if you develop on Windows/macOS, your work will only be accepted if it compiles and runs on Linux. In order to receive a grade, your code **must** compile using steps (2)-(4) of the compilation instructions above. To help you check this, we have included a Travis-CI script in each repository. This will automatically report whether the build is successful each time you push changes to GitHub. However, you must first create an account with Travis-CI and give it access to your repository; see `Assignment1/Readme.md` for more information.

To ensure fairness of your grade, you will be asked to briefly present your work to Julian. You will have 3-4 minutes to demo your submission and explain in some detail what has been implemented. You will also be asked to report encountered problems and how you tried to solve them, and point to the locations in the code where you implemented the various key parts of the assignments. The scheduled demo date will be listed at the top of each assignment.