CS33400/ECE30834: Assignment #0 - Cook it!

OpenGL Warm-up (GLFW)

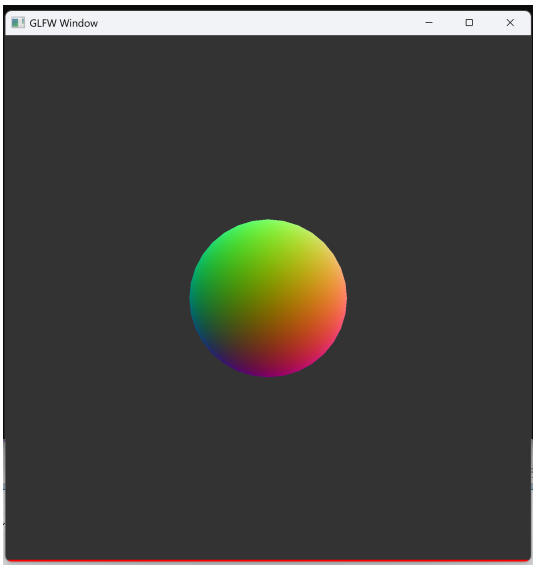
Out: August 28, 2026
Back/Due: September 4, 2026, 11:59 PM

Objective:

The objective of this assignment is a simple warm-up program to help set up your programming and graphics environment. This assignment will require you to set up a shell programming environment for this and future assignments, using OpenGL. We provide a template using **GLFW**. It is to your benefit to write the program modularly and with a clean setup so as to facilitate subsequent assignments. You have 1 week, but it should take you much less time. If you want to use a different framework, contact the TAs with your request (note that you still must use OpenGL within your framework).

Summary:

The assignment is to implement a program which simulates a bouncing ball. The program includes a bouncing ball and a bar (the ball will bounce when it hits the bar) inside an imaginary box. You may use whatever OpenGL commands you wish, but you will probably need to add some simple matrix math operations. You also need to design a simple physics engine to enable gravity so that the ball will fall like in the real world. You do not need to use 9.8 as the acceleration; just choose a value that makes the animation look smooth and visually plausible. The program also supports moving the bar and the ball using mouse clicks.



Specifics:

1. Start with the template from the course website. You may develop under Windows or Linux. The GLFW version is also intended to support MacOS. See the `README.txt` included with the template for details on compiling and running.

2. Compile and run the template. The image above shows the GLFW version. Everything is still because the physics engine and the refresh mechanism have not yet been added. **Read the code to understand how it works**, and then make the changes below.

3. Start with `glstate.cpp`. You will need to define some global variables indicating the status of the ball: the vertical position (it is already given as `yLoc`), velocity, radius, and acceleration of the ball. After your complete implementation, you may need to try modifying these values and find the best ones. It is suggested that you

start with small values (e.g., 0.0001 for the acceleration, but that also depends on how you design the physics and the refresh mechanism), since it makes debugging simpler. Then you should update the global values you defined in other appropriate places in the template.

4. Implement the physics engine inside `GLState::paintGL` in `glstate.cpp`. The function `paintGL` will be called every frame, so you need to update the ball's status: its vertical position (`yLoc`) and velocity.

   First, do the bouncing check: when the ball hits the bar (note that you have to take the ball's radius into consideration), its velocity should be reversed. Also update any other variables you think are necessary.

   Second, draw the updated ball. Call `glm::translate` and pass the updated `yLoc` so that the ball will be moved to its new position in the next frame. We use `objXform` to represent the transformation applied to the ball.

5. Implement the left-click feature inside `GLState::paintGL` in `glstate.cpp`. In `paintGL`, we use `lineXform` to represent the transformations applied to the bar. The initial position of the bar is saved in `GLState::initLinePos`, and the current position of the bar is saved as `linePos`. Similar to how the ball is moved, use `glm::translate` to move the bar to the place where the left click happens, and save the matrix in `lineXform`.

6. In `GLState::moveBall`, after the ball is moved, reset the velocity with the initial value you defined.

7. The template code does not automatically update the screen each frame. For a moving object, you will need to repeatedly tell the windowing framework to redraw the screen.

   - **GLFW version:** use the `idle` function and `elapsed` in `main.cpp` to trigger `display()` at a constant rate.

8. The code lines requiring your implementation are marked "TODO"; however, depending on your particular implementation there might be other places for you to change code. You are expected to add/modify the code as necessary to make sure the application runs smoothly.

Turn-in:

To give in the assignment, please use Brightspace. Give in a zip file with your complete project (project files, source code, and precompiled executable). The assignment is due **BEFORE** the end of the due day. It is your responsibility to make sure the assignment is delivered/dated before it is due. If you wish to receive confirmation of receipt, please ask by email in advance.

Do not wait until the last moment to hand in the assignment!

For grading, the program will be compiled on Linux and run from the terminal (with Visual Studio as a fallback—please try to avoid platform-specific code, e.g., do not `#include <windows.h>`), run without command-line arguments, and the code will be inspected. If the program does not compile, zero points will be given.

Good luck!