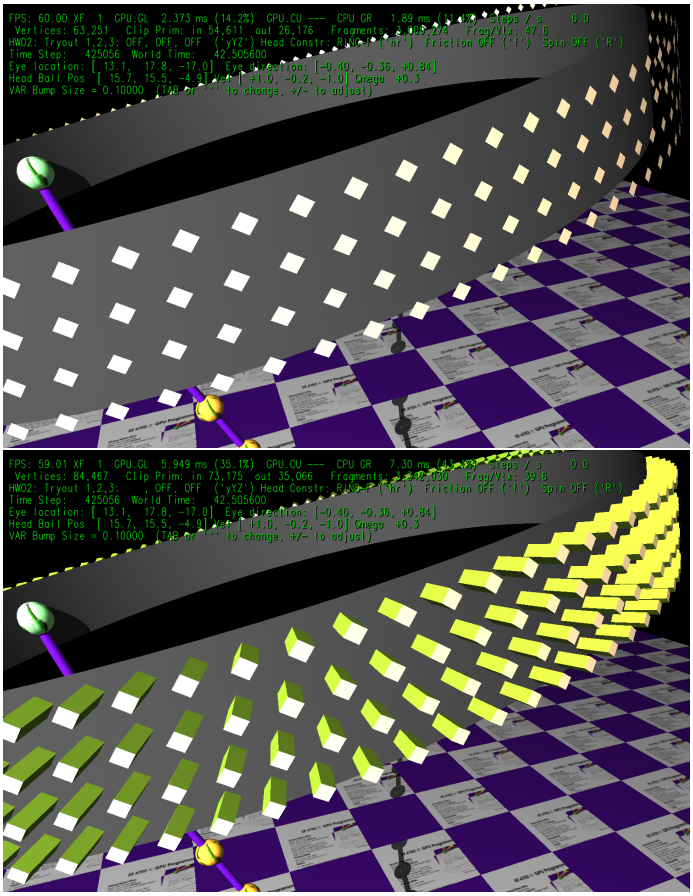
LSU EE 4702-1 **Homework 2** **Due: 2 October 2019**

Problem 0: Follow the instructions on the `https://www.ece.lsu.edu/koppel/gpup/proc.html` page for account setup and programming homework work flow. Compile and run the homework code unmodified. It should initially show a string of beads hanging from a ring. The ring is shown as a cylinder surrounded by tan-colored squares (or diamonds). See the screenshot on the upper right. The cylinder and the squares are rendered by code in routine `World::render_cylinder`, near the end of `hw02.cc`.



As you may have guessed, this code is based on the solution to Homework 1, and the user interface is similar. The goal in this assignment is to add sides to the squares so that they appear as blocks protruding from the cylinder surface, as in the screenshot on the lower right. The goal in the next assignment, Homework 3, will be to render these efficiently. (There will be a separate handout for Homework 3.)

Assignment-Specific Model and User Interface

When the simulation starts the head (first, zero) ball is attached to the ring and can slide freely along the ring. Pressing `r` (lower-case) will toggle between the head ball sliding freely and the head ball rotating along the ring at a fixed rate. The green text line starting with HW02: shows the state of the ball's attachment to the ring and of the ring itself. On this line the text to the right of Head Contr (head ball constraint) is RING-F if the ball is sliding freely, RING-A (animated) if the ball is rotating with the ring, FREE if the ball is not attached to anything, and LOCKED if the ball is fixed in space (does not change its position). The last two states can be obtained by pressing `h` (head). (Press `h` and `r` to familiarize yourself with the behavior, and be sure to read the next Common User Interface section which describes other available UI features, such as moving the viewer and pausing simulation.)

When in mode RING-A the rotation rate is determined by variable VAR Rotation rate (variable `hw01.omega` in the code). See the Common User Interface section below to see how to change that and other variables.

Pressing `f` toggles friction on and off, the state is shown to the right of Friction and by the color of the ring. When friction is turned on the ring color is yellowish, like sandpaper. Pressing `R` spins the ring.

Common User Interface

Press `h` (head) will grab or release one end (to be precise, the ball at one end) and pressing `t` (tail) will grab or release the other end. (Actually, those keys toggle between the `OC_Locked` and `OC_Free` constraint of their respective balls.)

Press digits `1` through `4` to initialize different scenes, the program starts with scene 1. Scene 1 starts with the balls arranged almost vertically. Scene 2 starts with the balls arranged horizontally, and they will start swinging. Each time scene 1 and 2 are initialized the ring is positioned to a new position. In scenes 3 and 4 the ring is always positioned the same way.

Press `Ctrl=` to increase the size of the green text and `Ctrl-` to decrease the size. Initially the arrow keys, PageUp, and PageDown can be used to move around the scene. Press (lower-case) `b` and then use the arrow and page keys to move the tail ball around. Press `l` to move the light around and `e` to move the eye (which is what the arrow keys do when the program starts).

The `+` and `-` keys can be used to change the value of certain variables to change things like the light intensity, spring constant, and variables needed for this assignment. The variable currently affected by the `+` and `-` keys is shown in the bottom line of green text. Pressing `Tab` cycles through the different variables.

Look at the comments in the file `hw02.cc` for documentation on other keys.

Code Generation and Debug Support

The compiler generates two versions of the code, `hw02` and `hw02-debug`. Use `hw02` to measure performance, but use `hw02-debug` for debugging. The `hw02-debug` version is compiled with optimization turned off and with OpenGL error checking turned on. You are strongly encouraged to run `hw02-debug` under the GNU debugger, `gdb`. See the material under "Running and Debugging the Assignment" on the course procedures page.

Keys `y`, `Y`, and `Z` toggle the value of host Boolean variables `opt_tryout1`, `opt_tryout2`, and `opt_tryout3`. The user interface can also be used to modify host floating-point variable `opt_tryoutf` and corresponding shader variable `tryoutf` using the `Tab`, `+`, and `-` keys, see the previous section. These variables are intended for debugging and trying things out.

There are problems on the next page.

Problem 1: Please submit the solution to this problem in writing or in the comments of your assignment. (Problem 2 is a computer assignment.) A question similar to this may be asked on the midterm exam. Each iteration of the loop below, taken from `World::render_cylinder`, correctly renders one diamond. Notice that there is one rendering pass per iteration.

```
for ( auto& e: info )
  {
    glBegin(GL_TRIANGLE_STRIP);
    glNormal3fv(e.nc); glVertex3fv(e.ctop);
    glNormal3fv(e.nl); glVertex3fv(e.cl);
    glNormal3fv(e.nr); glVertex3fv(e.cr);
    glNormal3fv(e.nc); glVertex3fv(e.cbot);
    glEnd();
  }
```

In contrast, the code below performs just one rendering pass. Assume that the contents of the `info` container is the same in the code above and below.

```
glBegin(GL_TRIANGLE_STRIP);
for ( auto& e: info )
  {
    glNormal3fv(e.nc); glVertex3fv(e.ctop);
    glNormal3fv(e.nl); glVertex3fv(e.cl);
    glNormal3fv(e.nr); glVertex3fv(e.cr);
    glNormal3fv(e.nc); glVertex3fv(e.cbot);
  }
glEnd();
```

(*a*) Explain why the second code fragment will not render the diamonds correctly. You should be able to sketch the result on paper.

(*b*) How could the code be modified to correctly render all the diamonds in one rendering pass? A correct solution requires changing one word and moving some code around. Feel free to use a deprecated primitive.

Problem 2: Modify the code in file `hw02.cc`, routine `World::render_cylinder` so that the tan-colored squares form the top of a block which touches the surface of the cylinder. To do so render the sides of these blocks and assign them color `color_olive_drab`, or some other color different than the squares. See the screenshots at the top of this assignment.

It will be helpful when solving this assignment to move the eye and light location around. The instructions above explain how to do so.

- Pay attention to efficiency. The code should not be less efficient than the code that's already there.
- Don't call `glColor` and `glNormal` more often than is necessary.
- Be sure to set the normal correctly. If the normal is set correctly adjacent faces will be shaded differently. If the same normal is used for adjacent faces then it will not be possible to tell where one stops and the other starts.

The easier part of this assignment is using `glVertex` calls in a triangles or triangle strip rendering pass to emit the coördinates of a block. See 2016 Homework 1 Problem 1a for an easy

example of how to use `glVertex` calls to draw shapes. (The code is in the repo, so you can look at the unsolved and solved versions.)

The more difficult parts are figuring out exactly what the code in `World::render_cylinder` is doing and then getting the geometry right. Try drawing a diagram of the cylinder and a diamond, and label it with the variables used in the code, such as `ctr`, `nc`, `ctop`, etc.