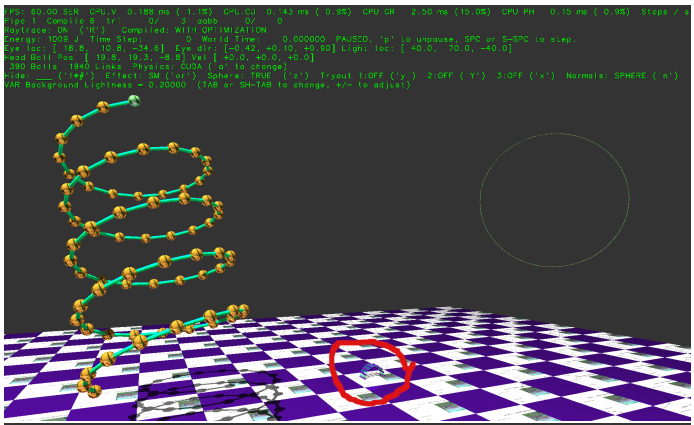
LSU EE 4702-1 **Homework 2** **Due: 25 September 2026**

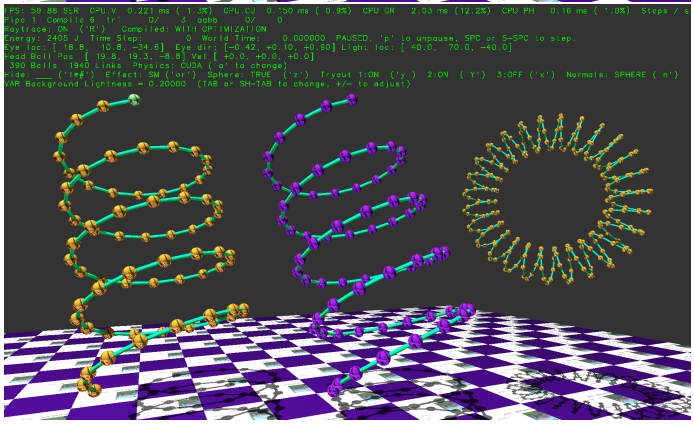
Formatted 11:43, 22 September 2026

All of the code for this assignment is in the course repo. HTMLized versions of the assignment file are at `https://www.ece.lsu.edu/koppel/gpup/2026/hw02.cc.html`.

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 three objects, balls forming a helix on the left, a ring on the right, and a small easy-to-miss helix lying on the platform. See the screenshot to the upper right. When the assignment is correctly solved there will be two similar helices (with gold and purple balls), plus a *toroidal helix* on the right. See the screenshot to the lower right. The purple helix is drawn by routines `hw02_prob1` and `spring_make2`. The toroidal helix is drawn by code in `hw02_prob2`.





Homework Overview

File `hw02.cc` contains several routines that draw helices. The easiest to understand is `World::spring_make`. It uses the parametric circle drawing approach used in class. Routine `spring_make2` avoids the need for computing trigonometry functions each iteration by instead using a transformation matrix, `global_space_delta_pos` to advance the current ball's coordinate, `pos_i`, to the coordinate of the next ball.

User Interface

The code can display several scenes, numbered 0 through 7. The currently displayed scene is identified in the penultimate line of green text. For this assignment use scene 7. Press [0] to select scene 0, [1] for scene 1, etc. Scene 3 shows a crude corona virus particle, Scene 4 shows a wheel, scene 5 a top (as in a spinning child's toy), scene 6 is a very crude parachute, and scene 7 is for this assignment.

Each time [7] is pressed the specifications for the displayed items will change. (That's to prevent students from ignoring call arguments.) Each time [Ctrl]-[7] is pressed the same specifications will be used, to help in debugging.

Press Ctrl = to increase the size of the green text and Ctrl - to decrease the size. Press F12 to generate a screenshot. The screenshot will be written to file `hw02.png` or `hw02-debug.png`. Press F10 to start recording a video, and press F10 to stop it. The video will be in file `hw02-1.webm` or `hw02-debug-1.webm`.

Initially the arrow keys, PageUp, and PageDown, can be used to move around the scene. Using the Shift modifier when pressing one of these keys increases the amount of motion, using the Ctrl modifier reduces the amount of motion. Use Home and End to rotate the eye up and down, use Insert and Delete to rotate the eye to the sides.

After pressing l the motion keys will move the light instead of the eye, after pressing b the motion keys will move the head ball around, and after pressing e the motion keys operate on the eye.

The simulation can be paused and resumed by pressing p or the space bar. Pressing the space bar while paused will advance the simulation by 1/30 s. Gravity can be toggled on and off by pressing g.

The + and - keys can be used to change the value of certain variables. These variables specify things such as the gravitational acceleration, dynamic friction, and variables that may be needed for this assignment.

The variable currently affected by the + and - keys is shown in the bottom line of green text. Pressing Tab and Shift-Tab cycles through the different variables. To locate variables which can be set this way, and to see how they were set, search for `variable_control.insert` in the assignment file.

Code Generation and Debug Support

The compiler generates an optimized version of the code, `hw02`, and a debug-able version of the code, `hw02-debug`. The `hw02-debug` version is compiled with optimization turned off, which makes it easier to debug. When needed, 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. **You must learn how to debug.** If not, you will be at a severe disadvantage *for the rest of your life, if that's how long you stubbornly resist learning!*

To help you debug your code and experiment in one way or another, the user interface lets you change *tryout* variables. There are three Boolean tryout variables, `opt_tryout1`, `opt_tryout2`, and `opt_tryout3` and one floating-point tryout variable `opt_tryoutf`. You can use these variables in your code (for example, `if ( opt_tryout1 ) { x += opt_tryoutf; }`) to help debug, to help familiarize yourself with how the code works, or to experiment with new ideas. Keys y, Y, x to toggle the values of the Boolean variables; their values are shown in the green text at the label `Tryout 1:` and `Tryout 2:`. The user interface can also be used to modify host floating-point variable `opt_tryoutf` using the Tab, +, and - keys, see the previous section.

Resources

A good reference for C++ is `https://en.cppreference.com/w/`. A past homework assignment that may be helpful is 2018 Homework 1. In particular look at the alternative solution to this assignment.

Collaboration Rules

Each student is expected to complete his or her own assignment. It is okay to work with other

students and to ask questions in order to get ideas on how to solve the problems or how to overcome some obstacle (be it a question of C++ syntax, interpreting error messages, how a part of the problem might be solved, etc.) It is also acceptable to seek out programming and graphics resources. It is okay to make use of AI LLM tools such as Claude and Copilot to generate sample code. (Do not assume LLM output is correct. Treat LLM output the same way one might treat legal advice given by a lawyer character in a movie: it may sound impressive, but it can range from sage advice to utter nonsense.)

After availing oneself to these resources **each student is expected to be able to complete the assignment alone**. Test questions will be based on homework questions and **the assumed time needed to complete the question will be for a student who had solved the homework assignment on which it was based**.

Students are expected to understand their own solutions. If you use something, such as a function, expression, or method of calculation found elsewhere, **make sure that you understand it**, don't just paste it blindly.

Student Expectations

To solve this assignment students are expected to avail themselves of references provided in class and on the Web site, such as for C++ programming and to seek out any additional help and resources that might be needed. (Of course this doesn't mean asking someone else to solve it for you.) Students are expected to experiment to learn how graphics work, and how to code C++ sequences. Experimentation might be done on past homework assignments. Students are also expected to learn what error messages mean by consulting documentation and by asking others (including Dr. Koppelman), and also to develop debugging skills. It is the students' responsibility to resolve frustrations and roadblocks quickly. (Just ask for help!)

This assignment cannot be solved by blindly pasting together code fragments found in class notes or past assignments. Solving the assignment is a multi-step learning process that takes effort, but one that also provides the satisfaction of progress and of developing skills and understanding.

Problem 1: Routine `World::spring_make2( m, n_balls )` constructs a helix consisting of `n_balls` balls connected to their neighbors with links. The ball coordinates are initially computed in a local space and then are transformed to global space using matrix `m`. In the local space the entire helix fits in a box of $x \pm 1$, $y \pm 1$, and z from 0 to 1. In the local space the helix has a radius of 1 and its axis starts at the origin and is along the z axis. The first ball is placed at local coordinate $\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ and the last ball will be about $\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$.

Routine `World::hw02_prob1(axis_top, axis_bottom, r, n_balls)` prepares a transformation matrix, `m`, and then calls `spring_make2` using `m` and `n_balls`. In the unmodified assignment `m` is an identity matrix. Because `m` is identity the second helix is shown in its local space, not in the intended global space. See the helix circled in red on the screenshot.

In `World::hw02_prob1` compute `m` so that the helix axis goes from coordinate `axis_top` to `axis_bottom` and the helix has a radius of `r`. If this is solved correctly the second helix should look exactly like the first one (including size and orientation) except for the ball colors.

☐ Compute `m` so that ☐ the two helices are identical (except for position and color).

☐ **Do not** change routine `spring_make2`.

▷ See routine `spring_make2` for examples of how to use transformation matrices.

▷ Solution will require a scale matrix, or should scale some other way.

☐ As always, code should be clear and reasonably efficient.

There's another problem on the next page.

Problem 2: In the unmodified code `World::hw02_prob2 (ring_top, ring_center, helix_vx, n_balls)` constructs a ring (circle) of `n_balls` balls around coordinate `ring_center`, passing through `ring_top`, and in a plane with normal `helix_vx`. Because the value of `n_balls` is large the balls and links are small and perhaps easy to overlook.

Modify `hw02_prob2` so that it constructs a *toroidal helix*. A toroidal helix is like a helix except that its axis forms a circle. The toroidal helix axis should be on the circle (ring) described above.

Consider the code from `spring_make` that computes a coordinate (`pos`) on a regular helix:

```
for ( int i=0; i<n_balls; i++ )
  {
    float theta = i * delta_theta;
    pCoor point_on_axis = axis_top + i * vz;
    pVect vector_from_axis_to_helix = cosf(theta) * vx + sinf(theta) * vy;
    pCoor pos = point_on_axis + vector_from_axis_to_helix;
```

Coordinate `pos` is computed by adding the coordinate of a point on the helix axis to a vector from that point to the helix.

For the toroidal helix `point_on_axis` should be a point on the circle, which should be easy to compute since the unmodified code already computes it. For the toroidal helix `vector_from_axis_to_helix` should be computed the same way as the regular one *except* that something must be done with `vx` and/or `vy` to account for rotation around the circle.

- ☐ Modify `hw02_prob2` so that it draws the toroidal helix.
- ☐ There should be about 9 balls per rotation around the helix.
- ☐ The distance between neighboring balls should be the same, including the first and last balls.
- ☐ Set `ball_rad` to $\frac{1}{5}$ the distance between balls or $r\Delta\theta/5$ where r is the helix radius and $\Delta\theta$ is for the helix, not the ring.
- ☐ As always, code should be clear and reasonably efficient.