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 un-
modified. It should initially show a string of balls com-
pressed between two points, the first and last balls are
fixed in place. The balls quickly expand and eventu-
ally will form an arc-like shape. The screenshot to the
right, taken when the simulation was paused, shows
the balls uncompressing, with pointy markers showing
local axes, and a green ball showing the approximate
place where the lowermost part of the string will be
once the balls come to rest.

User Interface
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). Pressing `Shift` and an arrow key
will move by a larger distance (than if `Shift` were not
pressed) and pressing `Ctrl` and an arrow key will move
by a smaller distance. The eye can be aimed up and
down by pressing `Home` and `End`, and the eye can be
rotated by pressing `Insert` and `Delete`.

    Pressing `p` will pause and un-pause the simulation. While the simulation is paused time can
be advanced by one frame by pressing the space bar, and by one time step by pressing `Shift` and
the space bar.

    Press digits `1` through `4` to initialize different scenes, the program starts with scene 1. In the
unmodified code scenes 1, 2, and 3 are identical: they start with the balls arranged in a line, and
separated by springs at less than the relaxed distance. Once this assignment is solved scenes 1, 2,
and 3 will behave differently.

    Pressing `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 `Ctrl=` to increase the size of the green text and `Ctrl-` to decrease the size.

    The `+` and `-` keys can be used to change the value of certain variables. These variables specify
things such as the light intensity, spring constant, 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` cycles through the different variables.

Code Generation and Debug Support
The compiler generates two versions of the code, `hw01` and `hw01-debug`. Use `hw01` to measure
performance, but use `hw01-debug` for debugging. The `hw01-debug` version is compiled with opti-
mization turned off and with OpenGL error checking turned on. You are strongly encouraged to
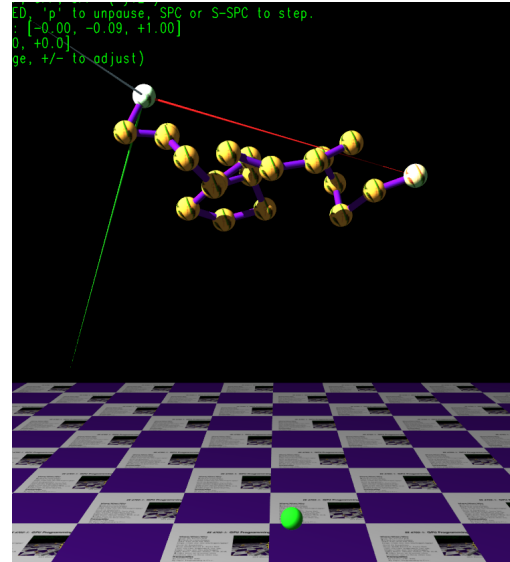
run `hw01-debug` under the GNU debugger, `gdb`. See the material under "Running and Debugging the Assignment" on the course procedures page.

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` using the `Tab`, `+`, and `-` keys, see the previous section. These variables are intended for debugging and trying things out.

Problem 1: Scene 1 starts with the balls arranged in a line and the first and last balls locked in place. The distance between adjacent balls is set to less than the spring's relaxed distance and so the balls are quickly pushed away from each other. They will eventually come to rest in an arc. The goal of this problem is to modify the setup routine so that the initial position of the balls are chosen such that the balls are closer to equilibrium when the scene starts.

In the ideal solution to this problem the balls are arranged to form a *catenary*, the name given to the curve made by an ideal cable with two ends fixed in space. A catenary is close in shape to a parabola.

This problem will be solved with a much cruder approximation: two straight lines. The code in `ball_setup_hw01` chooses a first and last position for the balls, `first_pos` and `last_pos`, denote these P_f and P_l. Let n denote the value of `chain_length`, the number of balls, and let l_r denote the value of `distance_relaxed`. The setup routine computes `nadir_pos`, denote that P_n.

Call $L = (n-1)l_r$ the (relaxed) length of the chain. Assume for a moment that the chain were an un-stretchable string, with the ends still fixed at P_f and P_l. Take a pencil and pull the string in some direction until it is taut. (That is, until the string forms a straight line from P_f to the pencil tip and P_l to the pencil tip.) Draw a dot. Pull in some other direction and draw a dot. Repeat ∞ times. Perhaps you've guessed that shape drawn this way is an ellipse. Points P_f and P_l are the foci. Since an ellipse is defined on a plane, we need a normal. The normal used is parallel to the xz plain. The point P_n is on this ellipse, and it is chosen so that it is the point on the ellipse with the minimum y value.

To help understanding these concepts the location of P_f, P_l, P_n, and the axes in the ellipse's local space can be visualized. To do so pause the simulation (using `p`) in scene 1, 2, or 3. (Switch scenes using `1`, `2`, etc.) The ellipse local x, y, and z axes are shown by red, green, and blue needles. Positions P_f, P_l, and P_f are shown by white, red, and green featureless spheres (which may be hidden by balls). See the screenshot to the right and the code below `hw01.markers_show` in `ball_setup_hw01` to see which colors are assigned.

P_n is chosen such that $\|\overrightarrow{P_f P_n}\| + \|\overrightarrow{P_n P_l}\| = L$. This means that the first several balls can be arranged on the line from P_f to P_n and the remaining balls can be arranged on the line from P_n to P_l, while keeping the distance between adjacent balls to l_r with possibly one exception. The exception occurs when no ball is exactly at P_n. In the exception case consider the two balls closest to P_n. The distance from the first of these balls, to P_n to the second of these balls will be l_r, but the distance between these two balls will be less than l_r.

2
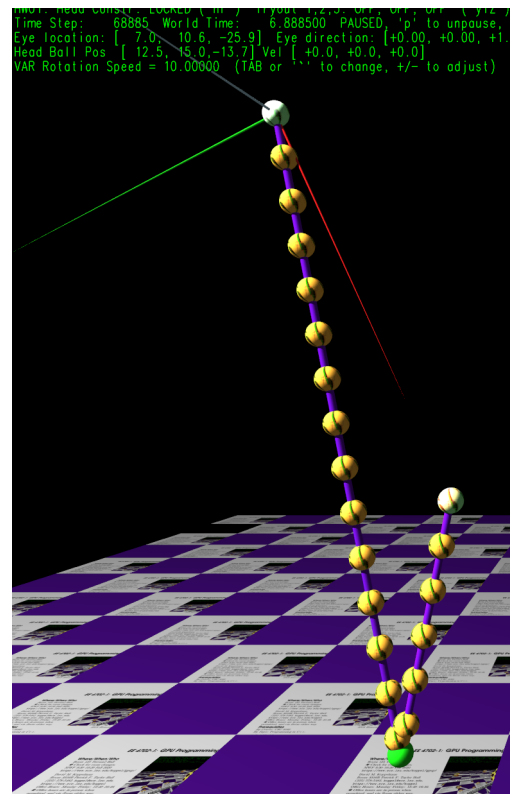
(*a*) Modify the code in `ball_setup_hw01` so that the balls are arranged in a straight line starting from `first_pos` reaching exactly or close to `nadir_pos`, and ending at `last_pos`. The distance between adjacent balls must be `distance_relaxed` except for the exception condition described above.

No ball should be placed below `nadir_pos`. The first ball must be at `first_pos` and the last ball must be at `last_pos`. See the screenshot above.

If this is solved correctly the two balls near `last_pos` should twitch (because they are closer than their relaxed distance). Initially the two lowest balls should be near the green marker (the location of `nadir_pos`) and the white and red spherical markers should not be visible.

To help debug your solution pause the simulation using `p`, then start the scene, `1`. Compare your placement of the balls to the marker balls.

(*b*) In this problem that twitching when a scene is started will be fixed. The solution is not ideal from a physical model point-of-view, but it makes a decent question. Modify the code so that each spring can have its own relaxed distance. Do so by adding a relaxed distance member to `Ball` and using it appropriately. The code in `ball_setup_hw01` must set this new member to `distance_relaxed` for all balls except for one. That one ball is the one that's closer to its neighbor. Set it to the appropriate distance if `opt_special_dist_relaxed` is true, otherwise set it to `distance_relaxed`. Modify the code in `time_step_cpu` so that the per-ball `distance_relaxed` values will be used. Part of this problem is deciding whether the value of `distance_relaxed` set for a ball applies to its connection to its predecessor in the `balls` array or to its connection with its successor.

Use scene 2 for this problem. In scene 2 (and 3) `opt_special_dist_relaxed` is true.

If this is solved directly scene 2 should start without a sudden twitch. Instead the balls will gently come to rest in the correct place.

(*c*) Modify the code in `ball_setup_hw01` so that when `opt_spin` is true (which is the case in scene 3) the balls are given an initial velocity so that they rotate rigidly around the line defined by `first_pos` and `last_pos`. To do this set the ball velocity. Note that balls closer to the line will have a lower velocity.

If this is solved correctly the balls will swing around the axis while generally keeping their shape.