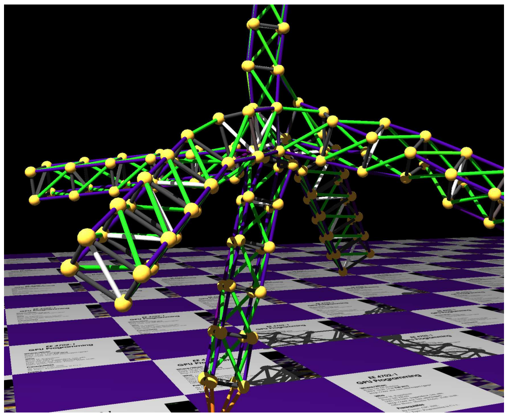
# LSU EE 4702-1 Homework 2 Due: 22 September 2015



**Problem 0:** Follow the instruction on the `http://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 show a swinging string of beads. Press digits `1` through `3` to initialize different scenes, the program starts with scene 1. The illustration to the right is from scene 3 generated with a solution to Problem 1. *Promptly report any problems.*

Unlike the code used for Homework 1, in this version when you manually move the head ball motion is smooth. That is, it doesn't jump to it's new location in one time step, rather it moves smoothly over 250 ms. This will help with Problem 1. Another change is in the collision resolution with the platform. In the Homework 1 code and some of the classroom demos, a ball colliding with the platform would have the $y$ component of its velocity flipped and its position snapped to $y = 0$. In the unmodified Homework 2 code virtual springs are used to compute separation forces for balls dropping beneath the platform.

Use the arrow keys, PageUp, and PageDown to move around the scene. Use key `h` to toggle between the first (head) ball being locked in place and free. Use key `t` to do the same for the last (tail) ball. Press (lower-case) `b` and then use the arrow and page keys to move the first ball around. Press `l` to move the light around and `e` to move the eye.

Look at the comments in the file `hw02.cc` for documentation on other keys. One fun thing to do is to lock both the first and last ball, move the head ball until the spring is stretched tight, then release one of the balls. Press `p` to pause, then the space bar to single step. *Note: There is nothing to turn in for this first problem.*

**Problem 1:** The balls and links code used in Homework 1 and classroom demos simulated indestructible links. In this problem modify the Homework 2 code so that links can snap. The illustration at the top of this assignment shows a screenshot from a solution.

First, define two variables in `My_Piece_Of_The_World`, one indicating a stressed threshold and one indicating a snap threshold. These might refer to a ratio of current length to relaxed length. If a link exceeds the snap threshold set the `snapped` member of `Link` to `true`. Modify the time step routine so that forces for snapped links are ignored. (The snapped links are not removed from the links list.)

If the link length is between the stressed threshold and snap threshold then proportionally change the color of the link to red. `Link` has two color members `natural_color` and `color`. `Link::natural_color` is set when a link is constructed and should not be changed. `Link::color` should be set based upon link stress. If the link is not stressed set `color` to `natural_color`. If it is stressed set `color` to `(1-s) * natural_color + s * red`, where $0 \leq s \leq 1$ is the stress level. (Determining a value for `s` is part of the problem.)

You can test your code by freezing both the head and tail balls and then using the keyboard to move the head ball, stretching the object.

**Problem 2:** Pressing `v` will cycle through three views: Showing only the balls and links (the SKEL), showing only triangles (the SKIN), and showing both. In the unmodified code the triangles will be scattered around the trusses. Modify the code so that triangles are rendered on the surface of the trusses, and optionally other appropriate places, such as the caps of the top.

The code for actually telling OpenGL to render the triangles should be placed in `render_my_piece`, which is near the bottom of the file. It currently contains placeholder code which results in the scattered triangles.

A solution to this problem will require new code in `World::make_truss` to identify which balls form the surface and adds them to arrays, perhaps in `My_Piece_Of_The_Wolrd`. For example, you might add new arrays and fill them with Ball pointers that are in the correct order for rendering.

Modify `render_my_piece` and other routines so that the triangles are placed on the surface, and so that normals are set correctly. This code might use the arrays that you prepared in `make_truss`.

Note that the arrays need to save pointers to Balls, not coordinates, because the balls move and so any coordinates copied during truss construction would result in a skin that doesn't move.

Try to render the skin using strips, rather than individual triangles or other individual primitives (hint).

**Problem 3:** The unmodified code does not simulate friction along the platform. Modify it so that it does. The time step routine uses a virtual spring to compute a separation force for any ball moving under the platform. Use this separation force to compute a friction force, and apply the friction force in the opposite direction along the platform.