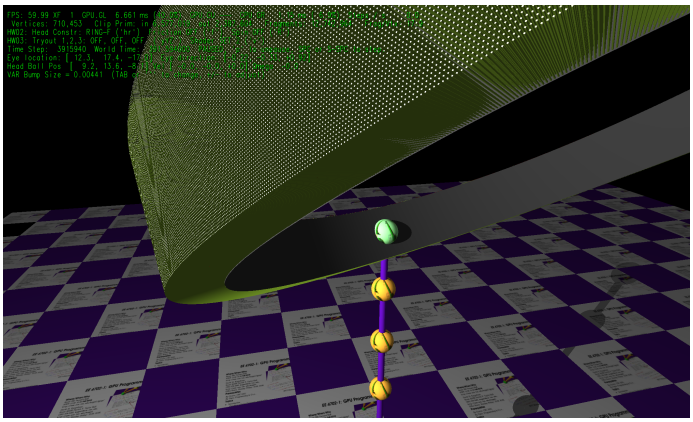
LSU EE 4702-1 **Homework 3** **Due: 1 November 2019**

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 green-colored protrusions with tan-colored tops, based on the solution to Homework 2. You may notice that in the screenshot to the right there are a large number of protrusions.



If the Homework 2 solution were used to render so many protrusions the frame rate would be very low, but as can be seen (perhaps after zooming) the code generating the screenshot above can keep up (the frame rate is about 60 FPS and there are no skipped frames [XF is 1]). That code is the solution to Problem 2 in this assignment, in which the protrusions are generated using shaders.

As you may remember one possible reason the code is slow is because protrusion geometry is re-computed each frame. That was true in Homework 2, but in the code for this assignment the geometry is only re-computed when something changes. Therefore performance problems are only due to the overhead of sending vertices to the rendering pipeline one-by-one (rather than using arrays).

Assignment-Specific User Interface

The code in this assignment uses one of three shaders, shown to the right of Shader on the HW03 line of the green text. The shader in use can be changed by pressing `v`. Shader FIXED uses the OpenGL default shaders, as was used in previous assignments. Shader STRIP-PLUS is for use in Problem 1. When STRIP-PLUS is in use the protrusions are rendered using a single triangle strip. They won't look good until Problem 1 is solved. When shader POINTS is used the protrusions are rendered using primitive type `GL_POINTS` in a `glDrawArrays` rendering pass. Code for this will be completed as part of Problem 2. The shaders are in file `hw03-shdr.cc`. STRIP-PLUS uses vertex shader `vs_strip_plus` and geometry shader `gs_strip_plus`, while POINTS uses vertex shader `vs_points` and geometry shader `gs_points`. Both use the same fragment shader (which is not part of this assignment).

To change the protrusion size modify variable Bump Size as shown to the right of VAR.

Previous Assignment 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.

Code Generation and Debug Support

The compiler generates two versions of the code, `hw03` and `hw03-debug`. Use `hw03` to measure performance, but use `hw03-debug` for debugging. The `hw03-debug` version is compiled with optimization turned off and with OpenGL error checking turned on. You are strongly encouraged to run `hw03-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`. and corresponding shader variables `tryout.x`, `tryout.y`, and `tryout.z`. 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: One frustration with triangle strips is that in order to start a new strip you either need to start a new rendering pass (which slows things down) or else you need to duplicate both the vertex that ends the old strip and starts the new strip (which is inelegant). Fortunately this frustration will be in the past once this problem is solved!

When shader STRIP-PLUS is used the protrusions will all be rendered using a single triangle strip. The shader is already written so that only the normal of the third (provoking) vertex is used for lighting.

Modify the `_strip_plus` shaders in file `hw03-shdr.cc` and the code in `World::render_cylinder` in file `hw03.cc` so that a new strip will be started when needed without having to start a new rendering pass. Signal the start of a new strip by sending something down the rendering pipeline. For example, you might set the normal of the first two vertices of a strip to the zero vector.

Problem 2: The `_points` shaders are to be used in a `glDrawArrays` rendering pass with input primitive point and with no inputs to the vertex shaders other than `gl_VertexID`.

Modify the `_points` shaders so that they generate the protrusions themselves using only `gl_VertexID` and uniform variables. As a convenience some useful uniform variables are set, but you'll need to set others to complete the problem. See the code in `render_cylinder`.

When this problem is successfully solved Bump Size (bump was the old name of protrusion) can be set small without impacting performance).

- Do not source any arrays into the rendering pipeline.
- Delete existing code in the `points` shaders that's not needed.
- Don't forget to modify the interface blocks (both vertex shader output and geometry shader input).
- Make sure that normals are correctly computed. Check normals by switching between shaders and comparing the shading of the edges.