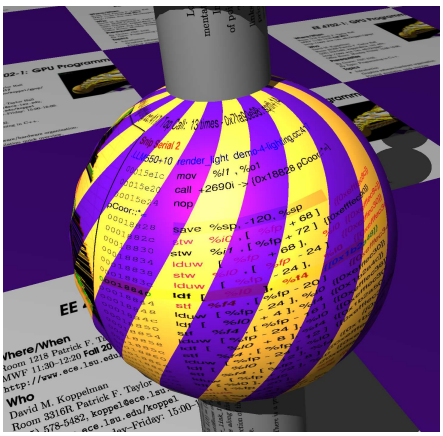
LSU EE 4702-1 **Homework 4** **Due: 6 November 2017**

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 initially show a column of beads swaying back and forth. The illustration to the right shows Scene 1 after this assignment is completed.



Scene Interaction and User Interface

Pressing `h` (head) will release one end (to be precise, the ball at one end) and pressing `t` (tail) will release the other end. (Actually, those keys toggle the fixed-in-space status of their respective balls.)

Pressing digits `1` through `4` will initialize different scenes, the program starts with scene 1.

The scenes differ in the number of objects, which include spheres, links, and the platform (which for this assignment we'll consider one object). The rendering of objects by type can be toggled on and off by pressing `!`, `@`, `#`, for spheres, links, and the platform. See the green text line starting with Hide. The scene includes shadow and mirror effects.

These can be toggled by pressing `o` and `r`, their state is shown next to Effect. Pressing `n` toggles between using a single normal for each triangle in a sphere (making the location of the triangles obvious), and using the interpolated normals associated with the triangles (making the surface appearance much more sphere-like).

Display of Performance-Related Data

The top green text line shows performance. XF shows the number of display updates per frame buffer update. An ideal number is 1. A 2 means that two display updates were done in the time needed to update the frame buffer, presumably because the code could not update the frame buffer fast enough. GPU.GL shows how long the GPU spends updating the frame buffer (per frame), GPU.CU shows how long the computational accelerator takes per frame. The computational accelerator computes physics in this assignment. On the lab computers the computational accelerator GPU is different than the one performing graphics. CPU GR is the amount of time that the CPU spends on graphics, and CPU PH is the amount of time that the CPU spends on physics.

The second line, the one starting with Vertices, shows the number of items being sent down the rendering pipeline per frame. Clip Prim shows the number of primitives before clipping (in) and after clipping (out).

User Interface for Navigation and Interaction

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 first 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 `Ctrl+` increases the size of the green text and `Ctrl+` decreases it. 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. Those who want to increase the spring constant to the point that the scene explodes may be disappointed to learn that there is a protection mechanism that increases the mass of balls when the spring constant is high enough to make the system go unstable, such balls turn red.

Look at the comments in the file hw04.cc and hw04-shdr.cc for documentation on other keys.

Switching Between Shaders

The program can use three different sets of shaders for the spheres, SPHERE, PROBLEM 0 (render_p1), and PROBLEM 1 (render_p1). Pressing v switches between them.

Description of Instanced Rendering in render_p1 and vs_main_p0

The code in render_p1 contains code adapted from the solution to Homework 2. The routine computes the coordinates of a spiral slice of a sphere and places it in a buffer object. There are two differences with Homework 2. First, the w component of a spiral slice coordinate is set to the value of θ with which it was computed. The vertex shader uses this to compute texture coordinates. Second, unlike Homework 2, the slice goes from pole to pole (as opposed to pole to equator as in Homework 2).

The routine then puts information about the balls in three buffer objects, pos_rad, rotations, and colors. Each element of buffer object pos_rad is a pCoor (vec4 in shader code). The x, y, and z members hold the world-space coordinate of the ball center and the w member holds its radius. The rotations buffer object holds the ball orientation (as a 4×4 transformation matrix) and colors holds the ball color. The render_p1 routine binds these buffer objects to OpenGL array buffers, so that they can be accessed in shader code.

Next, the routine installs either the *Problem 0* shaders, vs_main_p0, gs_main_p0, and fs_main_p0 or the *Problem 1* shaders, vs_main_p1, gs_main_p1, and fs_main_p1, both sets of which are in file hw04-shdr.cc. It then starts an instanced rendering pass using slice coordinates as vertex shader inputs.

Your solutions should be placed in the Problem 1 shaders. Leave the Problem 0 shaders unchanged or use them for minor experiments.

Instanced Rendering

In an instanced rendering pass a set of vertices, $\mathcal{V}$, is sent through the rendering pipeline I times, where I is an integer. In render_p1 the OpenGL command glDrawArraysInstanced starts the instanced rendering pass, the last argument to this command is the number of instances. The first time that $\mathcal{V}$ is sent through the rendering pipeline vertex shader built in variable gl_InstanceID will be set to 0, the second time gl_InstanceID will be set to 1, etc.

The code in vs_main_p0 uses gl_InstanceID to retrieve the ball's position and orientation and uses these to convert vertex shader input gl_Vertex, the sphere normal in the sphere local coordinate space, into an object-space coordinate, vertex_o. It then performs eye- and clip-space conversions on vertex_o.

The code in vs_main_p0 computes texture coordinates based upon the sphere's local y coordinate and the θ value provided. Study vs_main_p0 to understand how that works.

Debugging Help

Boolean variables opt_tryout1 and opt_tryout2 can be used for debugging on the CPU. Pressing y and Y toggles their state. In the shader code use tryout.x and tryout.y for these two variables.

To see the difference between your code and the original code issue the Emacs command Ctrl x v = or from the shell the command git diff.

Problem 1: Modify the code in vs_main_p1, gs_main_p1, fs_main_p1, and possibly render_p1 so that it renders the entire sphere by using the geometry shader to emit opt_spirals tri-

angles, for values ≤ 20. As in Homework 2, color odd spirals purple. For your convenience `color_lsu_spirit_purple` is defined in the shader file.

- Use one geometry shader invocation to generate triangles for all spirals.
- Be sure to generate approximately correct texture coordinates.
- The code must be reasonably efficient. Avoid trigonometric operations if these can be pre-computed and re-used.
- Color alternate spirals purple (with texture applied).
- **Don't forget to update the triangle layout.**