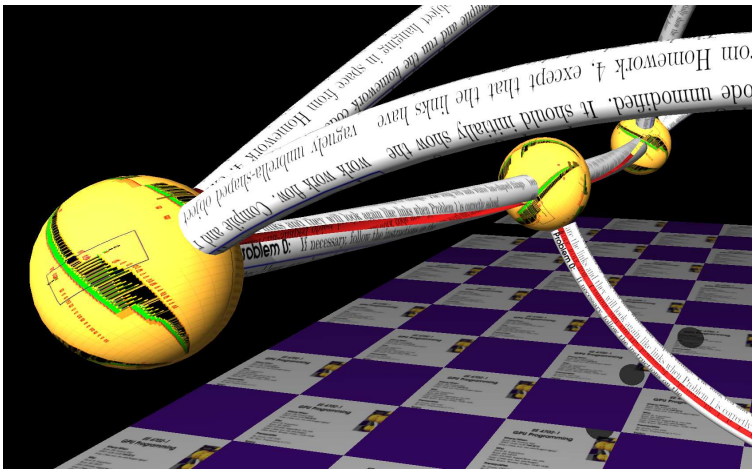
LSU EE 4702-1 **Homework 5** **Due: 17 October 2016**

Problem 0: If necessary, follow the instructions on the `http://www.ece.lsu.edu/koppel/gpup/proc.html` page for account setup and programming homework work flow. For this assignment only edit file `hw05-shdr-links.cc`. Compile and run the homework code unmodified. It should initially show the vaguely umbrella-shaped object hanging in space from Homework 4, except that the links have been replaced by long red and white fan-shaped things. The long red and white fan-shaped things are the links and they will look again like links when Problem 1 is correctly solved.

Pressing `v` will cycle through three different sets of shaders. The shader set that is being used is shown in the penultimate line of green text. Shader set PLAIN is a conventional set of shaders, and is there for comparison purposes. Shader set SET 1 is comprised of vertex shader `vs_main`, geometry shader `gs_main_1`, and fragment shader `fs_main_1`, all in file `hw05-shdr-links.cc`, and is used for Problem 1. Shader set SET 2 is comprised of vertex shader `vs_main`, geometry shader `gs_main_2`, and fragment shader `fs_main_2`, also in file `hw05-shdr-links.cc`, and is used for Problem 2.

The screenshot on the right shows what the object should look like after Problem 1 is solved correctly.



Press digits `1` through `2` to initialize different scenes, the program starts with scene 2. Scene 1 shows a balls connected in a rectangular spiral. *Promptly report any problems.*

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.

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).

When using the arrow and other keys to move the eye, light, or ball using `Shift` will move by a 5× greater amount and using `Ctrl` will move by one $\frac{1}{5}$ the amount than the motion without either modifier.

Look at the comments in the file `hw05.cc` for documentation on other keys.

Problem 1: One flaw in Homeworks 3 and 4 was in the use of triangle strips. Recall that a link was rendered in segments, and that each segment was roughly a cylinder. The link would have been rendered correctly if each individual segment were one triangle strip, but instead the code rendered the entire link as a triangle strip, resulting in two extra triangles at the beginning of each segment. The flaw persists in the Homework 5 code and is exasperated because the links are no longer supposed to be complete cylinders, instead there should be a gap revealing the red insides of the link. See the screenshot above.

An inefficient solution would be to perform a rendering pass for each segment. Or one might use an OpenGL draw command that can re-start strips periodically. But in this assignment we'll

fix the problem another way: by having the geometry shader recognize that a triangle should not be rendered and abandoning it.

On the CPU side in routine `render_link_1` the value of variable `theta` has been placed into the w component of the vertex coordinate. (I'll pause for a moment while you think about this.) Hopefully some of you realized that it's the w component that explains the long red and white fan-like things, since the w acts like a scale factor.

Modify the Set 1 shaders so that the value of the w component, `theta`, is used to determine whether a triangle should be rendered, and if not to discard the triangle. And make whatever other changes are needed so that the links look like links again.

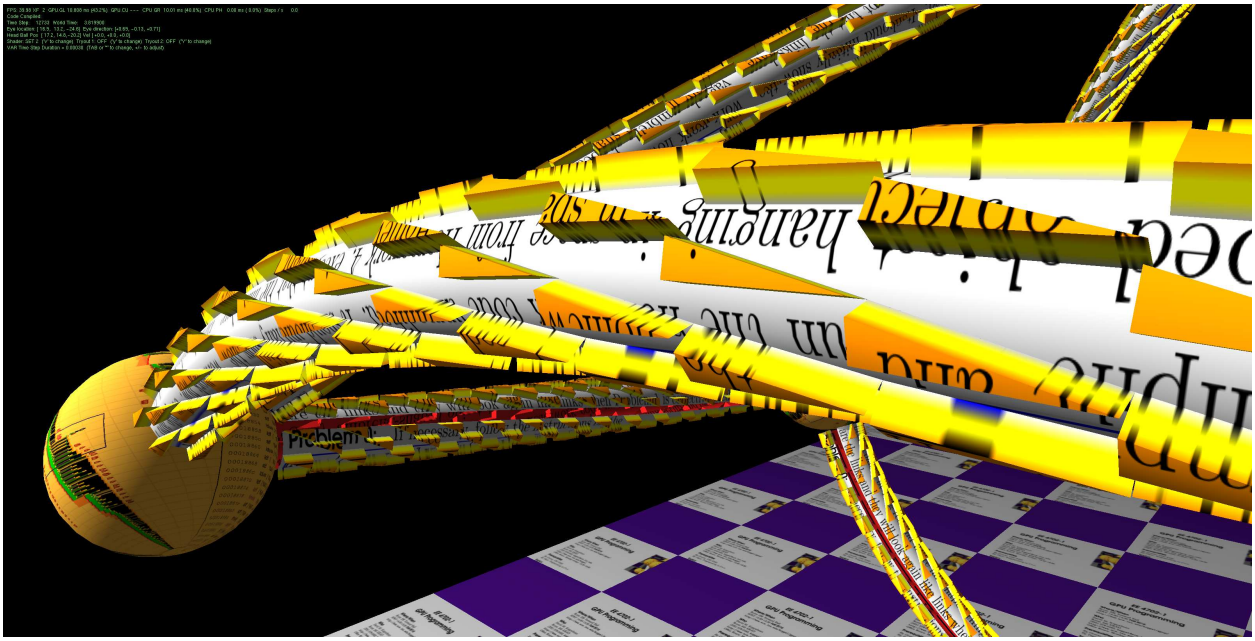
A solution to this problem will require the following: declaring a variable to communicate the value of theta between the vertex shader and geometry shader, modifying `vs_main_1` to extract theta from the vertex coordinate and then making sure that the correct vertex coordinate is used, and modifying `gs_main_1` so that it does not emit any triangles if theta is zero.

This triangle-strip issue is fixed using an alternative approach in the shaders for Problem 2. You might want to inspect that code for hints.

Note that packing unrelated information (θ) into the w component vertex coordinate is bad from the point-of-view of code readability however because GPUs handle four-element vectors very efficiently it might result in better performance than using a separate variable for θ.

There is another problem on the next page!

Problem 2: Modify the Set 2 shaders so that some triangles making up the link appear raised and colored as in the screenshot below. Details are given in the subproblems.



(*a*) Modify the shaders so that some triangles that make up the surface of the link are raised above the surface `thickness` units (where `thickness` is a uniform variable) with additional *side* primitives added to connect the raised triangle to the link surface. The raised triangles are the solid orange triangles in the screenshot above and the side triangles are colored orange/yellow/white. It is not important which triangles are raised, so long as some raised triangles don't share edges with other raised triangles.

Try to render the side triangles using a triangle strip. If done properly, a triangle strip will make this part easier, but it will make specifying the proper normals more difficult.

(*b*) Use the value of `color_top` for the color of the raised triangle, `color_bottom` for the link surface, and a blend of `color_side` and the other two colors for the side primitives. The color of the side primitives starting at the raised triangle and going towards the link surface should shift gradually from `color_top` (orange in the screenshot) to `color_side` (yellow in the screenshot), and then from `color_side` to `color_bottom` (light gray in the screenshot). The color should be completely `color_side` from 33% to 66% of the way down. (The colors should still be lighted.)

(*c*) The normals for the raised triangle should be the same as the normals that would be used on the link surface. The normals for the side triangles must be their geometric normals, so that the side triangles appear flat and the corners are sharp, not rounded.

(*d*) Render the text (texture) on the side triangles so that as one goes from the raised triangle to the link surface the text fades, completely fading halfway down, then darkening again. (See the screenshot.)