LSU EE 4702-1 Homework 4 Due: 7 November 2014

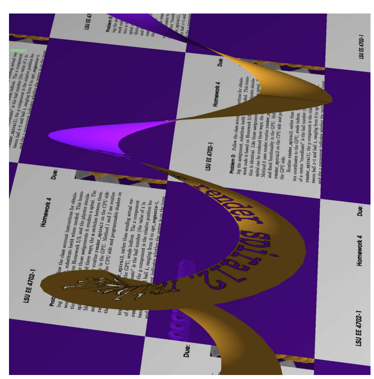
The solution code is in the repo in file name `hw04-shdr-sol.cc`. *If you pull and* `make` *you can run the solution by executing* `hw04sol`. *An html version is available at* `https://www.ece.lsu.edu/koppel/gpup/2014/hw04-shdr-sol.cc.html`. *Note that the git version is always the most up to date.*

Problem 0: Follow the class account instructions for obtaining the assignment, substitute hw04 where needed. This homework code is based on Homework 2/3, and the physics simulation is identical. Like those assignments it renders a spiral. The spiral can be rendered three ways, the `m` switches between them. Method 0 uses render routine `render_spiral1` on the CPU side and fixed functionality in the GPU. Method 1 and 2 use routine `render_spiral2` on the CPU side and programmable shaders on the GPU side.

Routine `render_spiral2`, rather than sending actual vertex coordinates to the GPU, sends indices. The x component of a vertex "coordinate" is the ball number (the value of `i` in `render_spiral1`), the y component is the relative position between ball `i-1` and ball `i`, ranging from 0 to `opt_segments-1`, and the z component indicates whether the vertex is on the inner or outer edge of the spiral. The code in vertex shader `vs_main` in file `hw04-shdr.cc` uses these values to compute the actual vertex coordinate and normal, and to compute the texture coordinates.

Notice that the indices sent by `render_spiral2` are not affected by the position of the balls. Therefore indices can be put in a buffer object and sent just once. (They do need to be re-sent if `opt_segments` changes.) The ball positions do need to be sent each frame, but that's much less data. The code in `render_spiral2` always uses vertex shader `vs_main` and fragment shader `fs_main`, but can activate either of two geometry shaders, `gs_main_simple` (for Method 1) and `gs_main_solution` (for Method 2). Geometry shader `gs_main_simple` makes no changes to the triangles it receives, and it should not be modified. Geometry shader `gs_main_solution` is initially the same as `gs_main_simple` but should be modified in one of the problems below.

The level of detail used for the spiral is controlled by variable `opt_segments`. Its value can be modified by first pressing the `TAB` key until "Number of segments per spiral" appears, and then pressing `+` and `-` to adjust its value. Check out and compile the code, and make sure that it runs correctly. **For this assignment** only modify file `hw04-shdr.cc`.



Problem 1: Notice that with Method 1 and 2 pale blue is used for both sides of the spiral. In this problem modify the fragment shader, `fs_main`, so the spiral appears like the one on the image to the right. Details are given in the subproblems. Changes to the fragment shader affect Methods 1 and 2, but not zero.

(a) Modify the fragment shader so that the front surface is gold and has the texture applied and so that the back surface is purple. Do this *without* using the color attribute (fragment shader input), instead rely on the material properties that are set in `render_spiral2`. Note that these material properties are uniform variables.

For the names of the predefined material property uniforms available in the fragment shader see Section 7.4.1 in the OpenGL Shading Language spec version 4.5. To determine whether a fragment is from the front or the back of a primitive see Section 7.1 in the spec.

(*b*) The surface of the spiral shows a draft this homework assignment handout, using black for the text and yellow for the background. Modify the fragment shader so that there is no spiral wherever the texture is black (or a dark color). In other words, modify it so that you can see through the spiral wherever the texture is black.

To determine if a texel is dark add up the red, green, and blue components and check if the sum is less than .05. If a texel is dark, then prevent the corresponding fragment from being written. To see how to prevent a fragment from being written see Section 6.4 in the OpenGL Shading Language spec version 4.5. (This section describes jump-like statements in the shading language.)

Problem 2: The spiral's appearance is a little unnatural because it is two-dimensional. Modify the shaders (except `gs_main_simple`) and surrounding code so that when Method 2 is active the spiral has some thickness. (See the illustration at the beginning of this assignment.)

Use geometry shader `gs_main_solution` to render multiple triangles as follows. Let V_1, V_2, and V_3 denote the vertices of a triangle at the input to the geometry shader. The geometry shader should emit $V_1V_2V_3$ (as it already does) and also $U_1U_2U_3$ where $U_i = V_i + \vec{d}$ and $\vec{d}$ is a vector of length 0.1 pointing from ball $i-1$ to i (the balls that the spiral segment is between). The geometry shader should also emit triangles for an inner and outer edge which joins the two spirals. The edge should be the gray color set in `render_spiral2`. See the illustration.

The solution to this problem will require modifying all three shaders, and modifying the shaders' interface blocks (such as `Data_to_GS`). **No changes** should be necessary in file `hw04.cc`. It's okay to modify this file to help in debugging your code, but the solution itself should only be in `hw04-shdr.cc`. If you believe otherwise, please contact the instructor. Divide work appropriately between the vertex shader and geometry shader.

Here are some things to watch out for:

- Be sure to adjust the maximum number of vertices specified for the geometry shader output. If this number is too low execution will end with an error.
- When you modify an interface block (such as `Data_to_GS`) be sure to modify the other interface block with the same name (one is a shader output, the other is a shader input).

Here are some debugging tips:

- Check for shader code compilation errors when your program starts. The errors are sent to `stdout` and should appear in the shell or in gdb, depending on how you started the program.
- If execution ends with an OpenGL error, you can get a more detailed error message by turning on OpenGL debugging. To do this change `false` to `true` in `popengl_helper.ogl_debug_set(false);` near the end of `hw04.cc`.
- Three UI-controlled variables are available for debugging. They are `debug_bool.x`, `debug_bool.y`, and `debug_float`. The Booleans can be toggled with `d` and `D`. The float can be adjusted by pressing `Tab` until "Debug Float" appears and then press `+` and `-` to adjust the value.