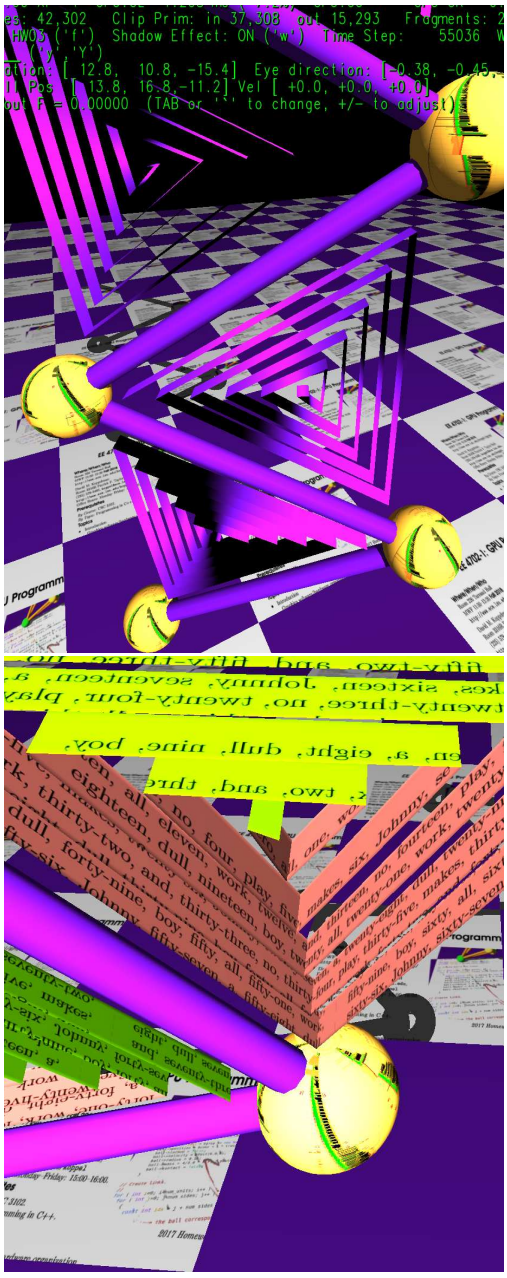
LSU EE 4702-1 **Homework 3** **Due: 11 October 2018**



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 the scene from the solution to Homework 2, in which balls are connected by links in a zig-zag pattern, and the triangles formed by each adjacent group of three balls are filled with a strip folded into a triangular spiral (the louvers). See the screenshot to the upper right (press `f` twice to see that coloring). The screenshot to the lower right is from a completely solved assignment.

Non-Assignment-Specific User Interface

Pressing `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 the fixed-in-space status 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. The other scenes are not used in this assignment.

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 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).

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.

Look at the comments in the file `hw03.cc` for documentation on other keys.

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 was compiled with optimization turned off and with OpenGL error checking turned on.

Keys `y` and `Y` toggle the value of host Boolean variables `opt_tryout_1` and `opt_tryout_2` and corresponding shader variables `tryout.x` and `tryout.y`. 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.

Shaders and Rendering Passes

The spiral triangle can be rendered using three different sets of shaders, Fixed, Plain, and HW03. The shader in use is shown in the green text near the upper left. Pressing `f` cycles through the shaders. The code in routine `hw03_render` sets up and uses these shaders. For the `Fixed` and `Plain` shaders it will emit each triangular spiral using many triangle strips, the method used in Homework 2; see the code near the comment "Small Strips Rendering." For the `HW03` shader each triangular spiral is rendered using one long triangle strip, see the code near the comment "Large Strip Rendering." By now everyone should see that the Large Strip Rendering code is more efficient since it uses fewer rendering passes (thus suffering less overhead setting up a rendering passes) and because it sends fewer vertices into the rendering pipeline.

Physics Model and Graphics

The physics model in the Homework 3 code is the same as the one used in Homework 2, in which springs are arranged to maintain the balls in a zig-zag pattern.

Pressing `w` will toggle shadows on and off. The triangular spirals do not cast shadows, and shadows are not part of this assignment.

Problem 1: When run with the Fixed shader the triangular spiral is colored green (okay, chartreuse) on the outside and salmon on the inside. These colors are obtained by setting the material front and back properties. With the HW03 shaders the color is a purple gradient, see the screen shot at the beginning of the assignment. The gradient is due to the interpolation of the normals from one fold of the triangle strip to the next.

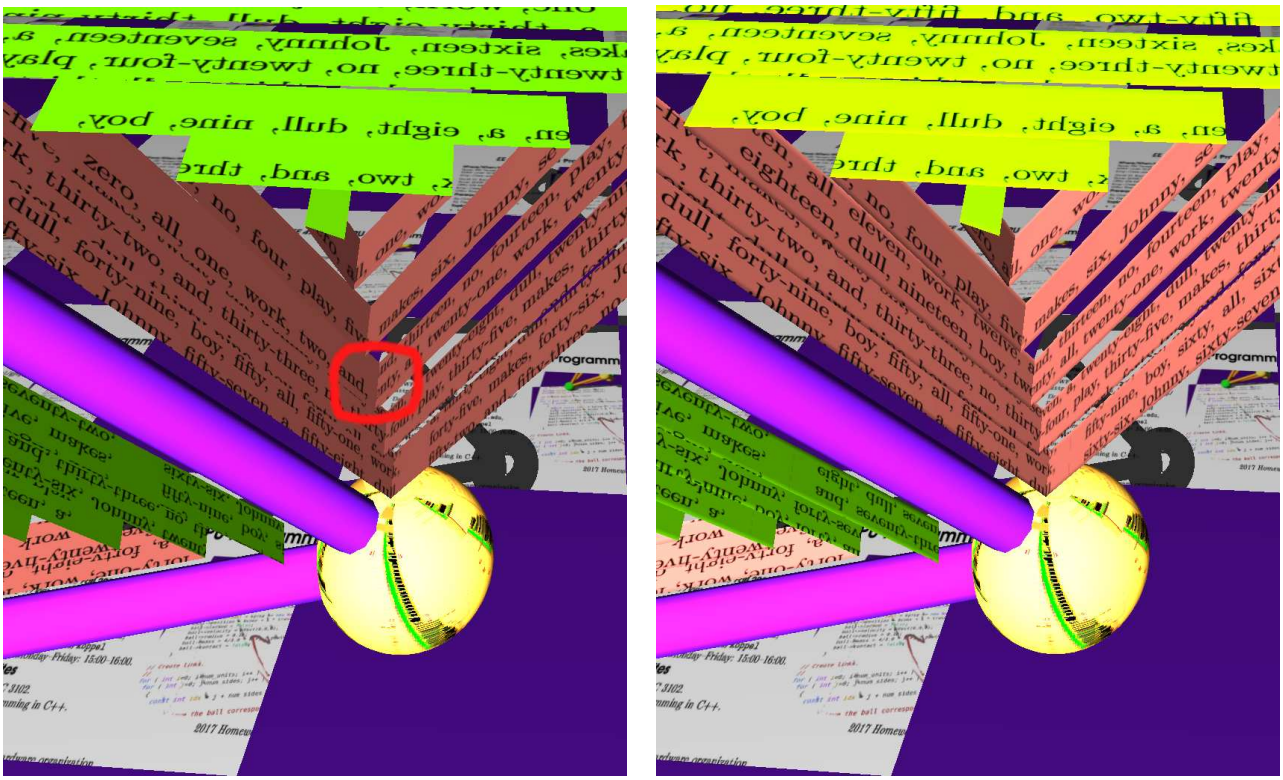
(*a*) Modify the `hw03` vertex and fragment shader related code in `hw03-shdr.cc` so that the correct normal is used to compute lighting. *Hint: A correct solution requires modifying a declaration in the shader code. That's one word to be inserted.*

(*b*) Modify the `hw03` vertex and fragment shaders in `hw03-shdr.cc` so that the front and back colors are used when the `hw03` shaders are active. The front and back colors are set by `glMaterialfv` calls in `hw03_render`.

- Use the OpenGL Shading Language documentation to find the built-in names of the uniform variables holding these values. (Look for the Built-In Variables chapter.)
- **Do not** declare your own variables for the front and back colors. Ordinarily declaring your own variables for material properties would be a good thing in this situation because it avoids the use of deprecated functionality. However the point of this problem is familiarization with OGSL documentation.
- Don't needlessly increase the amount of data sent between shaders.

Problem 2: Notice that in routine `hw03_render` (in `hw03.cc`) a texture unit is set up with a texture object named `texid_spiral_image`. Modify code in `hw03_render` and the `hw03` shaders in `hw03-shdr.cc` so that the texture is applied to the spiral so that the start of the text on the texture image is at the center of the spiral and so that text can be continuously read following the spiral from the center on outward. The text includes consecutive integers starting at zero.

The screenshot to the lower right shows the texture correctly applied (and with other parts of this assignment solved). The screenshot to the lower left shows a flaw. This problem can be solved two ways, the first will receive only partial credit, a correct solution to the second will receive full credit.



For both solutions:

- The size of the text must be the same on all parts of the spiral. That is, the inner segments **should not** show squished text and the aspect ratio should be consistent.

- It should be possible to continuously read the text starting from the spiral center. That is, it should be as though the texture image was cut into horizontal strips and these strips were glued together to make one long strip, which was folded into the spiral. The text at the end of the spiral does not need to reach the text at the end of the texture image.

(*a*) **For Partial Credit:** Modify the code in `hw03_render` in and around the "Small Strips Rendering" block to emit texture coordinates that will result in the texture appearing approximately as described above. In limiting changes to `hw03_render` this way flaws such as the one circled in red above are unavoidable. **There is no need to solve this part if the subproblem below is solved.**

(*b*) Modify the code in `hw03_render` in and around the "Large Strip Rendering" block and in the `hw03` shaders in file `hw03-shdr.cc` so that the texture appears as described above. The texture coordinates or equivalent emitted for this problem can also be used to solve the next problem. For full credit there should be no repetition and no gaps (except for the end of the spiral not reaching the end of the texture image).

Problem 3: Modify the code in `hw03_render` in and around the "Large Strip Rendering" block and in the `hw03` shaders in file `hw03-shdr.cc` so that the edges of the strip appear rounded, like the edges of a metal tape measure. The screenshot on the upper right has the rounding effect, the one on the upper left does not have the rounding effect.

Do so by rotating the normal used for lighting away from the center of the spiral. Perform the rotation by combining the normal currently used for lighting with the triangle normal (`nz` in

`hw03_render`). The blending should be done near the edges of the strip, elsewhere just use the existing normal for lighting.

- There is starter code in the fragment shader to solve this problem. Finish it.
- Make sure that the normal is computed correctly. The more closely the normal points to the light, the more brightly lit the object. Pay attention to the shading in the screenshot for the correct solution.
- Do not waste data bandwidth delivering the triangle normal to the shaders.
- Consider using the texture coordinates from the previous problem to detect distance from the strip edge.