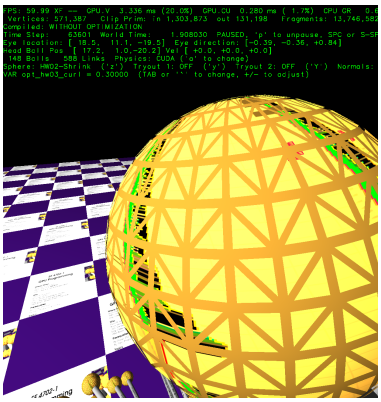
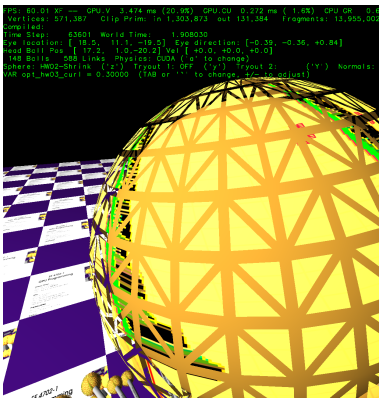
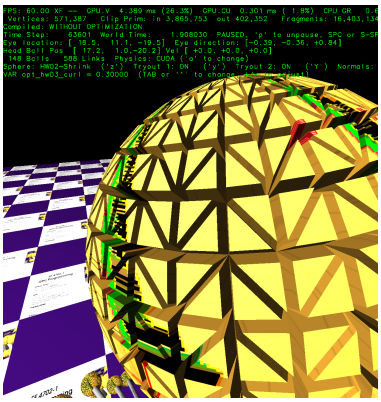
LSU EE 4702-1 **Homework 3** **Due: 17 November 2021**

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 scene from the solution to Homework 2, the one showing the silly tree with shrunken triangles. The code includes the Homework 2 solution. In this assignment the shrunken (but not the curled triangles) are to be given depth, as shown in the screenshots below.

Non-Assignment-Specific User Interface

Press digits `1` through `4` to initialize different scenes, the program starts with scene 1. Scene 1 starts with the balls arranged in the tree-like form.

Press `p` to pause the simulation.

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. Using the `Shift` modifier increases the amount of motion, using the `Ctrl` modifier reduces the amount of motion. Use `Home` and `End` to rotate the eye up and down, use `Insert` and `Delete` to rotate the eye to the sides. 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. These variables control things like light intensity and options needed for this assignment. The variable currently affected by the `+` and `-` keys is shown in the bottom line of green text next to VAR. Pressing `Tab` cycles forward through the different variables.

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

Assignment-Specific User Interface

The sphere can be rendered by four different shaders, `Normal`, `HW03-Shrink`, `HW03-Curl`, and `True`. The shader being used is shown to the right of `Sphere` in the green text. To cycle through the shaders (except for `True`) press `z`. To toggle between the `True` shader and some other press `Z`.

The amount of curling or shrinkage is specified by variable `opt_hw03_curl`. (This variable should affect curling and shrinkage.) This can be modified using the UI, look for that variable name to the right of VAR.

Pressing `n` toggles between computing sphere lighting based on triangle normals, `TRI`, and sphere normals, `SPHERE`. When solving the problems it might help to rendering using triangle normals so that you can see the triangle boundaries. It might also be helpful to adjust light intensity to make effects more visible.

Graphics and Performance Investigation Options

The user interface can be used to toggle various rendering options and for generating a screenshot.

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.

Pressing `F12` will write a screenshot to file with a name like `hw03-00.png`. The number at the end will start at `00` and will be incremented for each screenshot *within a run*. When the code is run again the count starts at zero, so be sure to rename files that you want to keep.

The rendering of shadows is toggled by `o` and the rendering of reflections it toggled by `r`. Their state is shown in the green text next to Effect:. Pressing `n` will toggle how surface normals are computed for tessellated spheres, the possibilities are to use the triangle normal or the sphere normal. The use of the triangle normals makes it easier to see the triangles from which the sphere was tessellated.

Display of Performance-Related Data

The top green text line shows performance in various ways. The number to the right of FPS shows the frame rate, the number of frames completed per second. On some displays 60 is the target frame rate and anything significantly lower than that indicates mediocre performance.

GPU.V 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 some assignments, but not this one and so the time should be shown as `---`. On some of 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 recording graphics commands (or whatever it does in the callback installed by `vh.cbs_cmd_record.push_back`. CPU PH is the amount of time that the CPU spends on physics or whatever it does in the callback installed by the call to `vh.display_cb_set`.

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). The next line indicates whether the code was compiled with optimization. Use the version without optimization for debugging and the version with optimization for performance measurements.

The `+` and `-` keys can be used to change the value of certain variables. These variables specify things such as the light intensity, sphere radius, and variables that will be 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. To locate variables which can be set this way, and to see how they were set, search for `variable_control.insert` in the assignment file.

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 Vulkan validation 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.

When Vulkan validation is on (which is currently in both debug and normal versions) helpful error and warning messages will be printed about misuse or abuse of the Vulkan API. These will appear on the terminal window (which might be a gdb session) from which `hw03-debug` was started. Currently, the code is set so that all warnings are fatal (except for a select few, which won't generate messages).

To help you debug your code and experiment in one way or another, the user interface lets you change variables. In most assignments these include the variables `opt_tryout1`, `opt_tryout2`, `opt_tryout3`, and `opt_tryoutf`. You can use these variables in your code (for example, `if ( opt_tryout1 ) { x += 5; }`) to help debug, to help familiarize yourself with how the code works, or to experiment with new ideas. Keys `y`, `Y`, and `Z` toggle the value of host Boolean variables `opt_tryout1`, `opt_tryout2`, and `opt_tryout3`. The user interface can also be used to modify host floating-point variable `opt_tryoutf` using the `Tab`, `+`, and `-` keys, see the previous section.

Instanced Draws, and the Assignment Sphere Instance Shaders

The code in the assignment package can render the sphere two ways, using the true sphere shader, which is not a part of this assignment (but still can be run), and the instanced shaders which are.

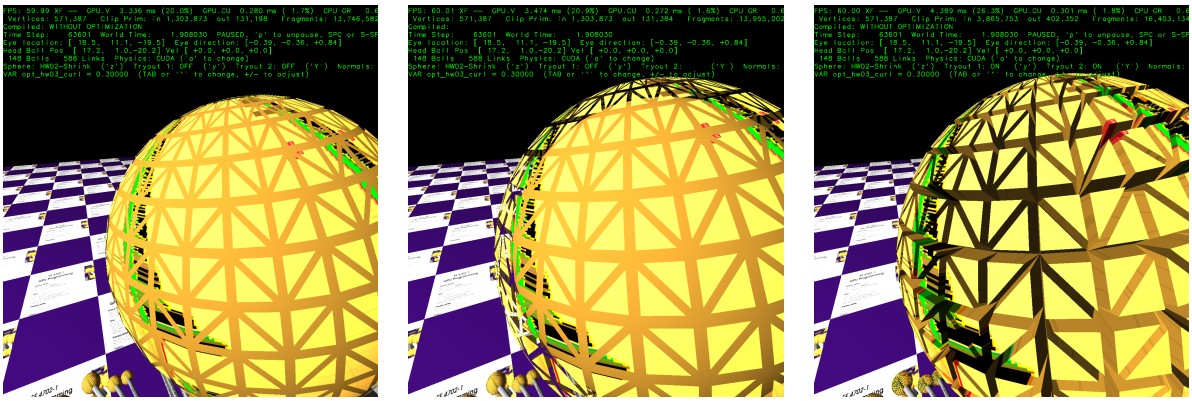
The sphere rendering that was described in `demo-09-shader.cc` rendered a single sphere. Recall that an input buffer set was prepared holding coordinates of the surface of the sphere in an order appropriate for triangle strips. The coordinates are for a sphere with its center at the origin and of radius one. A transformation matrix was used to scale and move the sphere to wherever we like. In homework 1 multiple spheres are rendered by recording multiple draws each with the same sphere buffer set but with different transformation matrices.

An instanced draw does something like that using a single draw command. The Vulkan draw command, `vkCmdDraw` and the equivalent Vulkan hpp draw member function `CommandBuffer::draw` (See Vulkan Chapter 21.3) take an `instanceCount` argument that specifies the number of instances to draw. So far in this class `instanceCount` has been set to 1, but when using our Sphere Instanced shaders the `instanceCount` will be set to the number of spheres we want to render. The code preparing buffer sets and recording the instanced draw is not part of this assignment. But for those who are curious, the Vulkan record draw calls are made in `VPipeline::record_draw_general` in file `include/vutil-pipeline.h`. Code using the course helpers might call `pipe.record_draw_instanced(cb, bufset, n_instances)`. For the homework assignment spheres `record_draw_instanced` is called in not-well-organized `Sphere::render_bunch_render` in file `shapes.h`. Remember, the host-side code discussed above is not part of the assignment. (Shader code is, which is discussed below.)

Suppose `instanceCount` is set to 2. Then the buffer set will be streamed into the pipeline inputs twice. The first time OpenGL Shading Language variable `gl_InstanceIndex` will be set to 0, the second time it will be set to 1. It is up to the shader code to use the value of `gl_InstanceIndex` to retrieve, in the case of our spheres, the location, radius, and orientation of the sphere. That is done by the vertex shader `vs_main_instances_sphere` in file `hw03-shdr.cc` (and other files). It reads information from three storage buffers, `sphere_pos_rad`, `sphere_rot`, and `sphere_color`, and uses these to compute clip- and eye-space coordinates, as well as texture coordinates. (Note that it would be possible to replace `sphere_pos_rad` and `sphere_rot` by a single transformation matrix.) The subsequent shader stages are no different than in an un-instanced draw.

In this assignment a geometry shader is also used to apply the special affects described in the problems. In the unmodified assignment the geometry shaders, `gs_main_shrink` and `gs_main_curl`, just omit an ordinary triangle.

Problem 1: In the Homework 2 solution the shrunken triangle was very close to the sphere surface, see the screenshot below on the left. In the unmodified shader code for this assignment the triangle is further away from the sphere, see the center screenshot. Modify `gs_main_shrink` so that the shrunken triangle is connected to the sphere by trapezoidal surfaces colored as described below. The shape made by the shrunken triangle, the original triangle, and the new faces should be a pyramid with the top cut off (a frustum). See the screenshot below on the right.



The new faces are formed using vertices of the original triangle and the shrunken one. (There is no need to render the original triangle.) The new faces should use a darker color and texture coordinates should be applied so that it appears that whatever texture color appears at a point on the edge of the shrunken triangle is used on the path down toward the sphere surface. See the screenshot. (This is an easy thing to do.)

Be sure to supply the correct normals for the faces. The faces are flat, not curved.

Let c denote the value of `opt_hw03_curl`. Remember that $c \in [0, 1]$. As with Homework 2, the size of the shrunken triangle should be proportional to $(1 - c)$, so that for $c = 0$ the shrunken triangle is the same size as the original. For this assignment c should also change the distance of the shrunken triangle from the original sphere surface. Let $d(c)$ denote the distance of the shrunken triangle from the sphere computed in terms of c. In your code choose $d(c)$ such that the volume bounded by the shrunken triangle, faces, and original triangle does not change. That is, when c gets larger $d(c)$ should get larger such that the volume does not change.

Note that unlike Homework 2, the fragment shader uses the same color for the front and back of primitives.

Problem 2: Sorry, no Problem 2. Trying to keep this short. Would have asked that the curl be given thickness too. A potential problem is exceeding the limit on the number of output floats of the geometry shader (1024 on my system). So as part of this unasked question, the instanced draw would have to be re-done such that each instance (sphere) was rendered multiple times. The first time the upper part of the curl would be rendered, the second time the edges. Some other time.