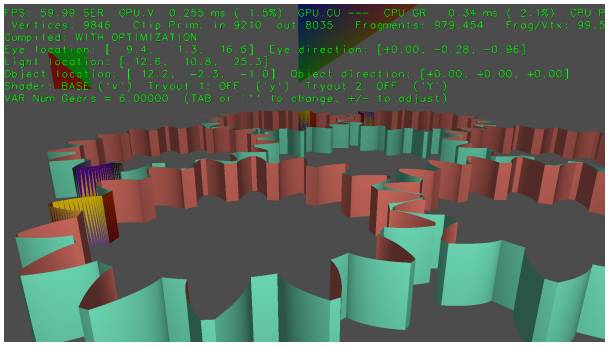
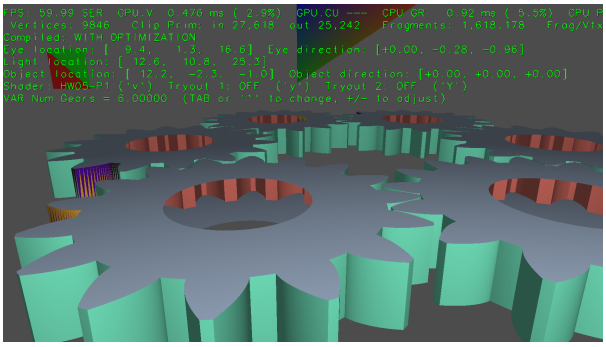
LSU EE 4702-1 **Homework 5** **Due: 28 Nov 2022, 23:59 CST**

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 look like the screenshot below on left. This shows involute gears, which are much better than the crude gears in the midterm exam. The screenshot on the right is taken from a correct solution to Problem 1.





User Interface

Press `Ctrl=` to increase the size of the green text and `Ctrl-` to decrease the size. Press `F12` to generate a screenshot. The screenshot will be written to file `hw05.png` or `hw05-debug.png`.

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, press `e` to move the eye (which is what the arrow keys do when the program starts), and press `b` to move the gears around.

Assignment-Specific User Interface

The scene can be reset by pressing `1`. Pressing `2` and `3` set up scenes, but those two haven't been tested. Pressing `p` will stop and start the gears from spinning. The gears can be rendered using three different shaders (each using their own pipeline), *Base*, *HW05-P1*, and *HW05-P2*. Pressing `v` (upper or lower-case) cycles through the shaders.

The number of teeth in the gears can be changed by adjusting `VAR Num Teeth`, and the number of gears can be changed by adjusting `Num Gears`. When Problem 2 is correctly solved rendering a large number of gears will be more efficient.

Code Generation and Debug Support

The compiler generates two versions of the code, `hw05` and `hw05-debug`. Use `hw05` to measure performance, but use `hw05-debug` for debugging. The `hw05-debug` version is compiled with optimization turned off. You are strongly encouraged to run `hw05-debug` under the GNU debugger, `gdb`. See the material under "Running and Debugging the Assignment" on the course procedures page.

To help you debug your code and experiment in one way or another, the user interface lets you change variables. In this assignment these include the Boolean variables `opt_tryout1` and `opt_tryout2`, and floating-point variable `opt_tryoutf`, which are available both in CPU code and in shader code. 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` and `Y` toggle the value of the Boolean variables. Their values are shown in the green text. The `VAR` mechanism can be used to change `opt_tryoutf`.

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. Next, the green text shows whether frames are being prepared one at a time (serially), SER, or the steps in preparing a frame are being overlapped, OVR. In overlap mode commands for one frame are recorded while commands for a prior frame may be executing. For assignments early in the semester the mode will be kept at SER.

GPU.V shows how long the GPU spends updating the frame buffer (per frame), GPU.CU shows the execution of CUDA code per frame. CUDA code is 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 Vulkan 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`.

For this assignment rendering is done by rasterization. (Other assignments will use ray tracing.) For rasterization 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 Overview

Routine `World::render` is called each time the frame buffer is to be prepared. It is similar to other render routines used in class, the parts relevant to this assignment will be discussed here.

All shaders use buffer set `bset_gear_strip`, in which vertices are arranged for a triangle strip. The gear shape consists of cylindrical portions called *lands*, and also includes *faces*, the part of a tooth that makes contact with a tooth on another gear. The shape of a face is an *involute* of a circle. With this face shape the point of contact between two gears follows a straight line.

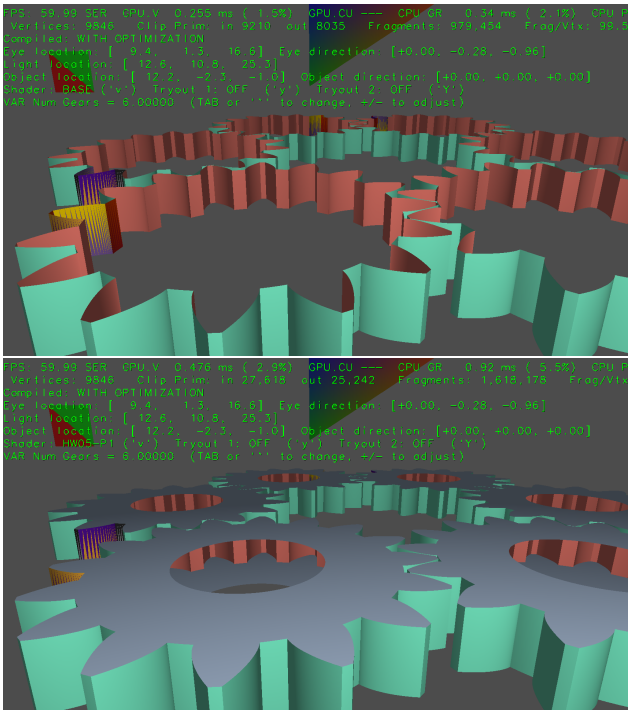
Recall that in Homework 4 the surface normal was eliminated from the buffer set so as to reduce the amount of data sent. Here the surface normal is back and for a reason. The normal is for the approximated surface (cylinder or involute) not of a triangle.

The buffer set holds the coordinates of a gear in a local coordinate space. In that space the gear axis is aligned with the z, with the center of the gear at $x = 0, y = 0$. Array `buf_gear_xforms` holds a transformation matrix for each gear that moves it to the desired position. The gears are arranged in a circle and are positioned so that the teeth mesh. Gear rotation is done using matrix `rot`, computed in the render routine. For the Base and Problem 1 shaders, `rot` is and should be used to update the set of transformation matrices sent to the shaders. (That's done in the `ds_set` member function, see Problem 2.) For the Problem 2 shader it is used to update the transformation matrices, but shouldn't be.

In this assignment there is no need to change the code that prepares the buffer set.

Some notes on how to work with the `VPipeline`, `VVertex_Buffer_Set`, and the buffer types can be found in this under-construction Vulkan course library note set.

Problem 1: The gears in the upper screenshot consists only of *faces* and *lands*, but not the sides of the gears. The sides are visible in the lower screenshot.



Modify the shaders in the Problem 1 area of `hw05-shdr.cc` and code in `hw05.cc` so that the pipeline `pipe_gear_prob1` renders the sides of the gears, with the radius of the hole set to `gear_info.radius_inner * 0.5`. The color should be set to color set 6 (light steel blue).

It should come as no surprise that the sides will be emitted in the geometry shader. Most of the solution should be in the shader code. Do not change the vertex shader inputs. It will be necessary send over some data as uniforms, `Uni_Misc` can be used.

It might be helpful to look at built-in variable `gl_VertexIndex`, available in the vertex shader. For the first vertex its value is zero, for the second its value is one, and so on. Because of the way the gear is constructed, odd values of `gl_VertexIndex` will be on one side of the gear (perhaps facing up), and even values will be on the other.

- Do not change the vertex shader inputs.
- Set the radius of the hole to `gear_info.radius_inner * 0.5`.
- Set the color to index 6.
- Be sure to correctly compute the normal.

Problem 2: As pointed out in an earlier assignment, performing multiple draws can be inefficient, this is the case for the Problem 1 pipeline because a draw is performed for each element in `buf_gears_xforms`:

```
case HW_Shader_Prob1:
  for ( auto m: buf_gears_xforms )
    {
      pipe_gear_prob1.ds_set( transform * m * rot );
      pipe_gear_prob1.record_draw(cb, bset_gear_strip );
    }
  break;

case HW_Shader_Prob2:
  pipe_gear_prob2.ds_set( transform * buf_gears_xforms[0] * rot ); // This is wrong.
  pipe_gear_prob2.record_draw_instanced( cb, bset_gear_strip, opt_n_gears );
  break;
```

The Problem 2 pipeline is rendered using an instanced draw. In an instanced draw the vertices (the ones in `bset_gear_strip` in the example above) are sent to the pipeline multiple times. In the example above they are sent `opt_n_gears` times (once for each gear). In the vertex shader the value of built-in variable `gl_InstanceIndex` is set to 0 for the first gear (or instance), 1 for the second gear, and so on.

```
void vs_main_prob2() {
  int inst = gl_InstanceIndex;
  vertex_c = ut.clip_from_object * in_vertex_o;
```

For your solving convenience, the contents of `buf_gears_xforms` has been sent to the GPU and made available as a storage buffer named `gears_xform`. That storage buffer is only sent over when the position of the gears changes. The storage buffer **is not sent over** for gear rotation and it should not be sent over for rotation, so don't change that.

The unmodified code will seem to show just one gear when using the Problem 2 shader. It is actually showing `opt_n_gears` gears, but they are all in the same position, which is not what we want. Modify the code in the Problem 2 section of `hw05-shdr.cc` and in `hw05.cc` so that the instanced draw works correctly (showing the gears in the same place as the other shaders) and is reasonable efficient. There is no need to include the sides that were completed for Problem 1.

- Modify the code so that the instanced draw works.
- The gears should turn.
- Do not send over `buf_gears_xforms` just to update rotation.
- If solved correctly CPU time should go down, especially with a large number of gears.