Programming Assignment #6

Due Nov 22 by 11:59pm **Points** 4
Submitting a text entry box or a file upload **Available** until Nov 26 at 11:59pm

Due: Tuesday November 22nd (at midnight).

Late Policy: Assignment submission will remain open for 4 additional calendar days after the formal deadline. Late submissions will be penalized by a maximum of 1 point on our 4-point scale (i.e. if you submit 4 days late and your assignment was otherwise deserving a "3", you might get a grade as low as a "2" due to this penalty). Some leniency might be exercised on this general rule based on how late the assignment was, and whether turning in an assignment was a repeating occurrence or a one-time event.

Synopsis: You get to write some shaders! You will experiment with the **shdr.bkcore.com (http://shdr.bkcore.com/)** interactive shader editor to create your own pair of vertex & fragment shader, and test it with the provided 3D geometric models

Learning Objectives: To familiarize yourselves with the GLSL shading language, the respective roles of the vertex & fragment shaders, exchanging information between them (via *varying* variables), and using the supplied attributes and uniforms. Also, experimenting with shading models incorporating diffuse and specular reflection.

Evaluation: Based on our 4-point grading scheme, as discussed in our introductory lecture. You get a check (“3”) if you turn in a viable, and complete submission (even if it just draws a rectangle like the example in the tutorial). “Above and beyond” grades (i.e. a "4") will be awarded for people who have crafted something particularly cool. As a general rule, no more than 1/3 of all assignments turned in (the very best ones, that is) will be considered for a "4" grade.

Collaboration policy: This is an assignment to be done individually. Code not written by you needs to include proper attribution (see **this post (https://canvas.wisc.edu/courses/320922/pages/collaboration-policy)** here). It is always ok to use code provided in our in-class examples as a starting point, but you need to add your own effort to raise those examples (or other sources) to what is asked by the programming assignment (i.e. finding some code on some online forum that does all the job for you that is needed to satisfy the assignment is not the intent, if you haven't added any of your own effort to it). If you use somebody else's code, make sure to clarify in your submission notes what **you** did, and what you repurposed from the external source; for this particular assignment, since you have been given some very intricate shaders as examples, it's important to explain what is the new thing that you added yourself, if your starting point was one of the shader examples discussed in class.

Hand-in: Electronic turn-in on Canvas. The **shdr.bkcore.com** (http://shdr.bkcore.com/) editor gives you a way to build a "link" that points to your implemented shaders: if you click on the "Share" button, you will get a (long!) URL that captures your entire program, including both the vertex shader and the fragment shader. Include this link BOTH in the text box of the Canvas submission, AND in a README file that you upload. Also, as a fail-safe, we recommend uploading the shaders as a text file. You can do that by clicking the "Download" button on top of the interface; this will give you a text source of both shaders, that you can save and upload to your submission.

Description

Your task is to write some shaders in GLSL; specifically *a pairs of shaders* – a fragment shader and an associated vertex shader.

This used to require building an application, getting some geometry, attaching the shader to it … but in the modern era, we can use the web! There are a few "shader sandbox" websites out there. Both are cool websites where you can type in shader code and see it applied to a model in real time. They let you try writing shaders with a minimum of fuss – just focus on writing the shader! They even helps you by giving you sample fragments of code.

http://shdr.bkcore.com/ (http://shdr.bkcore.com/) is a simpler one that we have used over the years in CS559, with reasonable success and a small number of hickups (that seem to change from year to year, but typically not particularly problematic – this year it appears that the only issue is that the URL "Shorten" utility in the "Share" button doesn't work anymore, since it depended on Google's URL shortener that is no longer operational). But it should be perfectly sufficient for your purposes.

The intent of this assignment is to make it relatively easy to get a "3", i.e. a *satisfactory* grade. In particular, to meet that standard your shaders need to meet the following requirements:

- **Requirement #1**. At least *some* of the shading/lighting computations need to happen in the fragment shader. That's to say, you should do something different than computing a per-vertex color and simply interpolating the color itself across each triangle by assigning it to a *varying* and directly using it in gl_FragColor as **this shader (http://goo.gl/ORVZrq)** does (again, to avoid confusion, this is what you should **not** do; perform some of the shading calculations in the fragment shader, like **this example (http://goo.gl/ooE6NL)** instead).
- **Requirement #2.** Implement a shading model that includes at least diffuse reflection or specular reflection (ideally both, but at least one of them).
- **Requirement #3.** Use the "time" uniform provided by shdr.bkcore.com to have *some* element of your scene change dynamically over time. You have many options how to do this … you can change the shape of the geometry being drawn, you can change the location of the light source (see **this example (http://goo.gl/2qVk59)**), you can change the color of the object or the color of the light, you can change the fragments being discarded, etc … **this example (http://goo.gl/gMWglO)** does several of these things at once!

As always, you are encouraged to try and exceed these requirements, and if you do well, you can compete for a "4" *above-and-beyond* grade. Here are some ideas (you should try multiple of these, to be more competitive for this grade!)

- Use multiple lights, maybe with changing colors, or changing locations.
- Make sure to incorporate both diffuse and specular reflection.
- Make the color of the object (the underlying "intrinsic color" of the object, that is, not just the apparent color due to lighting) vary across the spatial extent of the object. **Here (http://goo.gl/36jtDv) are (http://goo.gl/1tVj5D) some (http://tinyurl.com/y2y5sd8v)** examples.
- Make the geometry of the model being shown "stretch" or otherwise deform. You can do it by applying transforms to the vertex positions in the vertex shader. **This example (http://goo.gl/gMWglO)** takes this principle to the extreme!
- Use the "discard" functionality to create interesting/artistic transparency.