Program 1: ray tracing, due 9/23

Due: 11:59pm, Friday September 23

Goal: In this assignment you will practice the ray casting methods that we are discussing.

Submission: Submit your assignment using Wolfware Classic. The submit locker is now available. Click on "Submit Assignments" next to "Grade Book".

BASIC GRADING:

- *10% Part 0:* properly turned in assignment
- *50% Part 1:* using ray casting, render the colored spheres in the input file without lighting
- *40% Part 2:* color the spheres with Blinn-Phong illumination.
- *Participation:* Receive participation credit (outside of this assignment) for posting images of your progress, good or bad, on the class forum!

Note that prior parts need not remain functional if later parts are completed well.

General:

You will only render spheres in this assignment, which are described in an input file. We will test your program using several different input files, so it would be wise to test your program with several such files. The input files describe an array of spheres using JSON. An example input file resides at https://ncsucgclass.github.io/prog1/spheres.json. When you turn in your program, you should *use this URL in hardcode as the location of the input spheres file* — it will always be there. While testing, you should use a different URL referencing a file that you can manipulate, so that you can test multiple sphere files. Note that browser security makes loading local files difficult, so we encourage you to access any input files with HTTP GET requests.

We provide a small shell in which you can build your code. You can run the shell here, and see its code here. The shell shows how to draw pixels without using WebGL, and how to parse the input spheres.json file. It contains three drawing functions: one that merely draws random pixels, one that loads the sphere file and draws orthographic projections of them using canvas draw functions, and one that loads the sphere file and renders some random pixels in them. The last is probably closest to what you must produce for this program.

All vertex locations should be described in world coordinates, meaning they do not require any transformation. Locate the eye at (0.5,0.5,-0.5), with a view up vector of [0 1 0] and a look at vector of [0 0 1]. Locate the window a distance of 0.5 from the eye, and make it a 1x1 square normal to the look at vector and centered at (0.5,0.5,0), and parallel to the view up vector. With this scheme, you can assume that everything in the world is in view if it is located in a 1x1x1 box with one corner at the origin, and another at (1,1,1). Put a white (1,1,1) (for ambient, diffuse and specular) light at location (2,4,-0.5).

You should code the core of this assignment yourself. You may not use others' code to determine the location of pixels in the world, to do ray-sphere intersection, or to color a pixel. You may use math libraries you find, but you must credit them in comments. You may recommend libraries to one another, speak freely with one another about your code or theirs, but you may never directly provide any code to another student. If you are ever uncertain if what

you are contemplating is permissible, simply ask me or the TA.

Part 0: Properly turned in assignment
Remember that 10% of your assignment grade is for correctly submitting your work! For more information about how to correctly submit, see this page on the class website.

Part 1: Using ray casting, render unlit, colored spheres
Use ray casting to render unlit spheres, with every pixel in each sphere having the unmodified diffuse color of that sphere (e.g, if the diffuse color of a sphere is (1,0,0), every pixel in it should be red). You will have to test for depth, to ensure that each sphere is correctly colored. You should see flat circles, turning slightly ellipsoidal at the edge of the view due to perspective.

Part 2: Using ray casting, render lit spheres
Now you will have to perform a local Blinn-Phong lighting calculation at each intersection. As you perform that lighting calculation, don't forget to normalize your vectors, and recalculate the local light vector, since it is different at every pixel. You should now see spheres with depth made clear by illumination, in the same locations and with the same silhouettes as in part 1.

EXTRA CREDIT GRADING:
- *2.5%* arbitrarily sized images (and interface windows)
- *2.5%* arbitrary viewing setups
- *2.5%* off-axis and rectangular projections
- *2.5%* multiple lights at arbitrary locations
- *5%* shadows during ray casting
- *10%* render triangles

Other extra credit is possible with instructor approval. You must note any extra credit in your readme.md file, otherwise you will likely not receive credit for it.

Extra credit: Arbitrarily sized images and viewports
Accept a new canvas (viewport) width and height through your UI. Size your canvas to match, and change your ray casting interpolation to match. This should effect every part of your assignment.

Extra credit: Support arbitrary viewing setups
Accept new eye location, view up and look at vectors through your UI. Reorient the window to be normal to the new look at vector and centered around the new eye. Render the scene with these viewing parameters. Note that with bad viewing parameters, you will not see the model. This should affect every part of your assignment.

Extra credit: Support off-axis and rectangular projections
Accept new window parameters through your UI (relative to the viewing coordinates described by the look at and up vectors, these will be two X values (left, right) and two Y values (top, bottom)). Adjust your ray casting interpolation to this new windows. Render the scene with these new projection parameters. Note that if you also perform the arbitrary viewing extra credit, these coordinates may not be in world space! Also with bad projection parameters, you will not see the model. This should affect every part of your assignment.

Extra credit: Multiple and arbitrarily located lights
Read in an additional *lights.json* file that contains an array of objects describing light location and color. Note that these lights will have distinct ambient, diffuse and specular colors. Render the scene with these lights. During illumination, you will have to sum the colors all the lights. You can find an example lights.json file here. Assume that the input lights file will always reside at this URL when you turn in your code.

Extra credit: Detect shadows during ray casting
When performing lighting during ray casting, shoot an additional ray toward the light to decide if only ambient light reaches the intersection. If you also support multiple lights, make sure to shoot a ray at each light! This should only affect the last part of your assignment.

Extra credit: Render triangles
Read in an additional *triangles.json* file that describes multiple vertices, multiple triangles connecting them, and one material (set of reflectivity coefficients) to use with them all. Read in and render these triangles in addition to the input spheres. You will have to perform ray-triangle intersection, and must code this yourself. You can find an example triangles.json file here. Assume that the input triangles file will always reside at this URL when you turn in your code.
Posted by Benjamin Watson at 3:22 AM
Labels: announcements, homework, raycasting, raytracing