Santiago Bonada, 100525252

Assignment Two: Faking Global Illuminaton on the GPU

General

The fragment shader for the background is always back.fs; only one vertex shader is used for all parts and that's general.vs. The shaders that get used are specified in the *main()* function within viewer.cpp.

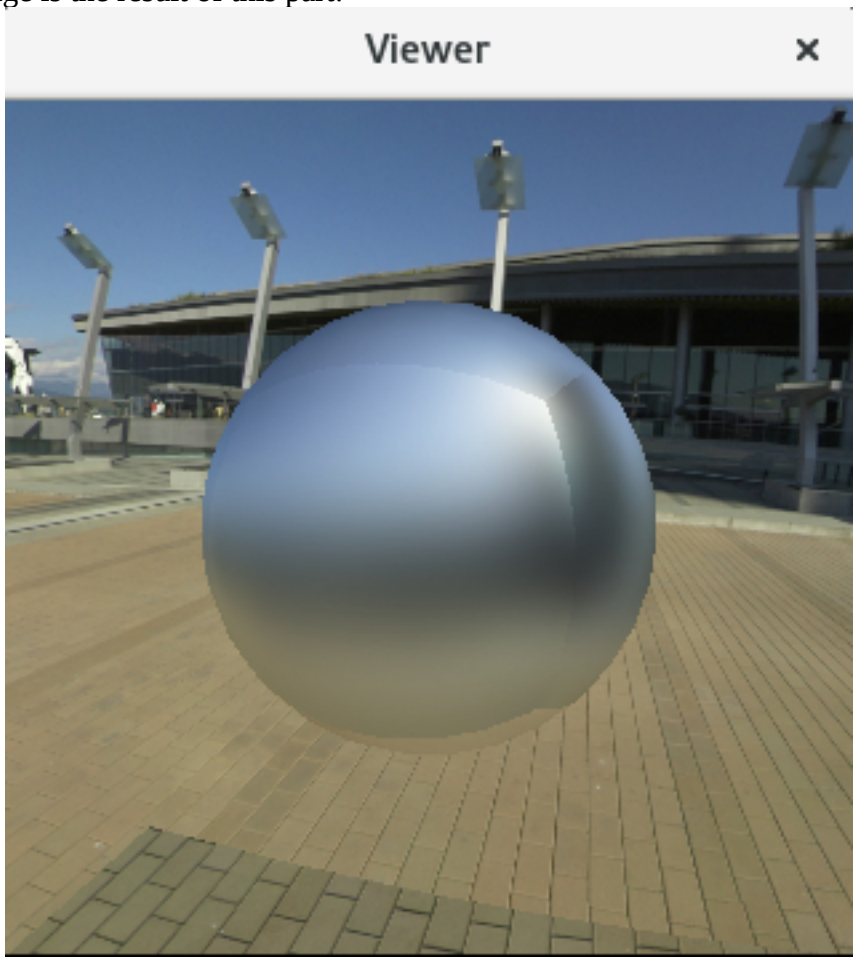
Reflection and Refraction

All of the code which handles this part is contained within the fragment shader refract.fs. It contains two functions, one for computing the R0 parameter in the Schlick approximation, s*chlick_R0()* which takes as input the index of refraction of the medium, and another function which computes the Schlick approximation, s*chlick_approx()* which computes the contribution of the reflection component and refraction component. The eye position is used as the incident vector to the Schlick approximation. After sampling the texture with the reflection and refraction vectors, we set the output colour by calling the GLSL *mix()* function with the reflection and refraction colour components, and using the value returned by the Schlick approximation as the mixing parameter.
The following image is the result:

Diffuse Reflection – Part One

A "diffuse" version of the cubemap was created by applying a gaussian blur to all the cubemap images and scaling down from 2048x2048 to 512x512, as was also detailed in the assignment. This diffuse version is stored in vcc_diffuse/. The fragment shader for this part is in diffuse_one.fs, and it's very

simple, all it does is sample the cubemap using the normal vector and outputs that as the colour. Most of the work done on this section was in the C++ code. Notably, in the VAO_loadCubeMap() function defined in VAO.cpp, the function is mostly a copy of of the lab05 code for loading and storing the cubemap. Main difference is that the index of the texture is given to OpenGL based on the function parameter “tid” added to GL_TEXTURE0, which in this program is 0 for the foreground sphere and 1 for the background sphere (so it becomes GL_TEXTURE0 and GL_TEXTURE1 respectively). The display function, *displayFunc()* in viewer.cpp, is also pretty similar to previous ones from the lab, difference is that there’s a loop that goes through the meshes to draw, and the texture for each mesh is binded before drawing. The irradiance map is specified for the foreground sphere within the *init()* function in viewer.cpp, where the second argument to the *VAO_loadCubeMap()* is the directory to get the irradiance map from (which would be vcc_diffuse).
The following image is the result of this part:



Diffuse Reflection – Part Two
The code for this part was mostly done in the fragment shader diffuse_two.fs. I used the uniform sampling method to average the hemisphere around the normal at a fragment. First I constructed a vector orthogonal to the normal by taking the cross product of the normal with the normal vector that has two components switched around; this vector is U. Then I get another vector orthogonal to both the normal and U by taking the cross product of U and the normal; this vector is W. Then I initialize the output colour to (0,0,0,0) and perform a double for loop to sum the samples, after which they are averaged. The two for loops vary the theta and phi variables. The formula to get the samples is the one detailed in the assignment – Lcos(theta)U + Lsin(phi)W. The iterations for each for loop, and the size of the interval of angles, is determined by the uniform *samples* which I currently have at 15. The total samples taken is thus 15*15. The range of angles is given by the uniform *angles*, currently set to 2*PI,

and the radius of the hemisphere that samples are taken from is given by the uniform *radius*, currently set to 0.1. These uniforms are set in the *displayFunc()* in viewer.cpp, which is the only further modification to that file.
I chose the mentioned values as they seemed to give the best results, additional sample size only slowed the program down further with no significant increase in quality.
The following image is the final output of the program: