Advance Graphic Report
Assignment 2
Ibrahim Mushtaq

This assignment explores faking global illumination through the environment and generating values at each vertex, given the environment data. There is not light source or anything that can simulate light except for cube map data.

Part 1:

In part one, we were task to implement reflection and refraction at each single pixel. The shader code that does this:

```
const float Eta = airRefract/waterRefract;
const float R0 = ((airRefract - waterRefract) * (airRefract - waterRefract)) /
((airRefract + waterRefract) * (airRefract + waterRefract));

void main() {
    vec3 N =normalize(normal);
    vec3 I = normalize(position - Eye);

    vec3 R = reflect(I,N);
    vec3 Rf = refract(I,N, Eta);

    vec4 reflectionColor = texture( tex, normalize(R));
    vec4 refractionColor = texture( tex, normalize(Rf) );

    float fresColor = R0 + (1.0 - R0) * pow( (1.0 - dot( -I, N ) ), 10.0);

    gl_FragColor = mix(reflectionColor,refractionColor,fresColor);
}
```

Where airRefract and waterRefract are the refractive index. Eta is the division between the two index, and R0 is Schlick's Approximation where R0 is the reflection coefficient. The code will calculate the reflect value and the refract value, and with the Fresnel coefficient fresColor, we can then add all the value together to get a sphere that is mirror like as you can see in Figure 1.

Figure 1 Reflection and Refraction

Part 2

In part two, this was relatively easy to implement since, we needed to decrease image size to 512*512 and blur it. However, when we blur the image, what seemed to be a ball with no seams, now clearly shows it with seams as you can see in Figure 2.

Figure 2 Blur image with seams visible

This is due to this code

```
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
```

Which linearly make the cube map and thus does not interpolate the two seams properly when its blurred.

Part 3

In part 3, we would need to fix this by, using uniform sampling strategy based on sampling directions close to the normal. The shader code that does this is:

```
float L = radius;
float theta = Theta;
vec3 newNormal = vec3(normal.x, normal.z, normal.y);
vec3 U = (cross(newNormal, normal));
vec3 W = (cross(U, normal));

vec3 V = ((L * cos(theta)* U) + (L*sin(theta)*W));
vec3 NV = normalize(normal+ V);
```

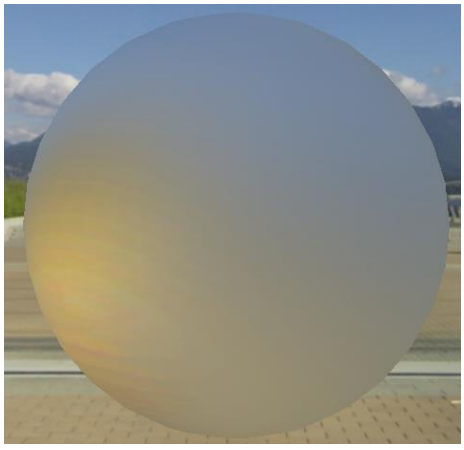

Figure 3 Uniform Sampling

and would produce this sphere that is almost like a frosted material, as seen in figure 3. Changing the theta value will change the directionality of the cube map. Changing the radius will change the number of sample points for each vector. However, the higher the value for the radius, the less interpolated the value are. For example, keeping theta at PI and radius at 0.09, would give a much better average picture, and you can go behind the sphere and see the outlines of the buildings.

Remarks

I have used a library in my solution. Its is called ImGui, https://github.com/ocornut/imgui, which is an immediate mode graphic GUI. This allows me to test my shader codes through the GUI. I have made the GUI with instructions that you can use, as well as picking different part of the assignment as well.

I am also using irradiance.vs and irradiance.fs as my main shaders.