CS33400/ECE30834: Assignment #2 - GPU it!
Basic Lighting and Normals Calculation

Out: February 7, 2025

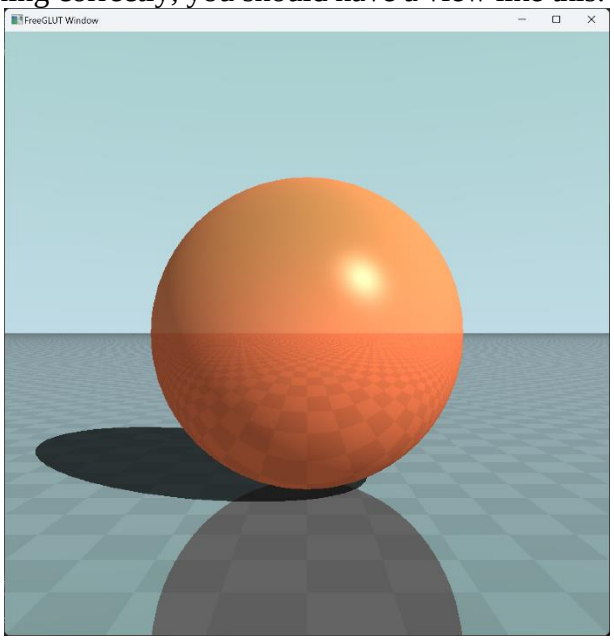
Due: February 21, 2025

Objective:
This assignment is designed to give you some experience with GPU ray-tracing using fragment shaders.

Summary:
Before getting started, as the main contents are implemented in GPU code, a time-saving trick is to dynamically load the GPU shader code into your running program. So you will be guided to apply this trick as a warm-up.

For the main content, you are given a scene with a sphere, a ground plane, a sky background and a point light source. You will first have to compute surface normals for the sphere and plane. Then you need to implement the shading function that defines the surface color using Phong shading model. You will learn how to render second-order lighting effect reflection as a first step to understanding ray tracing. A bonus task is to render another second-order lighting effect shadow using ray tracing.

If you implement everything correctly, you should have a view like this:



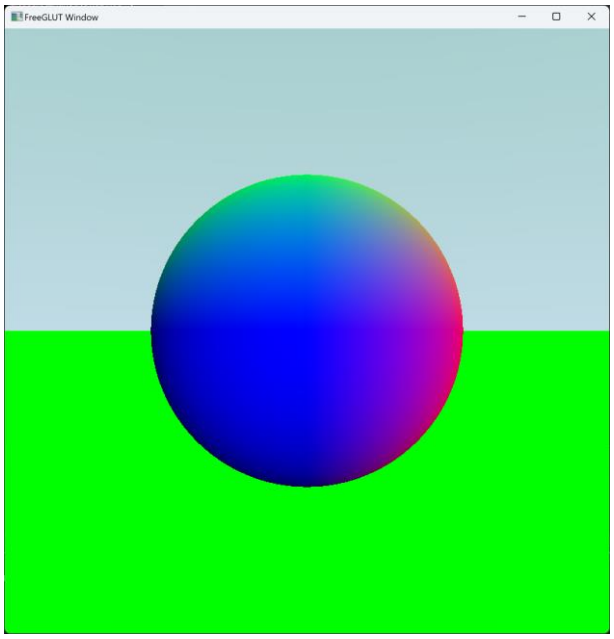
Specifics:

1. Warm-up (20%)
   Start with the template from the course website, similar to previous assignments. Previous assignments are C++ codes, any changes need to be recompiled and then run the executive in CPU again. In this assignment, most of the codes are in the `shaders/f.glsl` file, which is the shading language and is executed in GPU. GLSL programming language is very similar to C. You can define struct, but cannot define class. You can define functions, global variables, etc. There is no need to rely on glm library to use data structures like vec2, vec3, vec4, mat3, mat4. Those data structures are built-in in GLSL.

   We implement a small trick here for efficiently developing GPU code: "hot-update" the GPU code. To achieve this, you must add one line of code in the main.cpp L-123. Later on, you just need to press r, and your modification in `shaders/f.glsl` can be compiled on-the-fly and executed automatically without re-opening your program if your `shaders/f.glsl` does not have a grammar error. To test the feature, after implementing the TODO 1, try add one line of code "outCol = vec3(1.0, 0.0, 0.0);" after "outCol = shading(ro, rd, intersect);" in the `shaders/f.glsl` file. You should see a red screen when you press r without re-opening the executive.
   And then you can delete the new line, and press r to see if it changes back.

2. Normal Calculation (10%)
   Lighting models require knowledge of the surface's normal direction to produce realistic effects. In previous assignments, surface normals were read directly from the mesh files, but in this assignment, you need to manually compute the normal for the sphere in `shaders/f.glsl`.
   In TODO 2, "**pos**" is the surface point on the sphere. The global vec4 variable "**sphere**" defines the sphere shape. The xyz component of "**sphere**" is the sphere center position. The w component of "**sphere**" is the radius of the sphere. Given the surface point and sphere center, you should be able to calculate the normal easily.

The correct implementation should have a similar view:

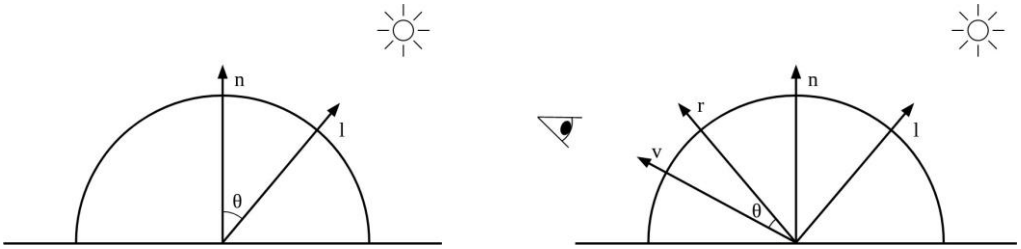


3. Phong Shading (40%)
   After completing the normal calculation, you can begin implementing the Phong illumination model. Lighting calculations happen in the fragment shader, where you can access all needed lighting and material information. Each light source affects the scene with three components: ambient, diffuse, and specular, described below. Lights have an associated light color, which modulates each of the below components, and a light position, which determines the position of the light. Additionally, the object in the scene has its own color, which is multiplied by the contribution of each light, as well as material properties that determine the relative strength of each of the below components.

   Ambient (a). Ambient light approximates indirect lighting, where light rays bounce around the scene before striking the object. In practice, this is implemented as a simple additive term that does not depend on light or view directions.

   Diffuse (d). Diffuse lighting approximates the amount of light that hits the surface based on the angle between the surface normal and the incident light vector. The more closely aligned the normal vector and the light direction are, the brighter the surface. This is modeled by the cosine of the angle between these two vectors.

   Specular (s). Specular highlighting occurs when light is reflected off the surface directly toward the viewer. The incident light direction is reflected across the surface normal, and the cosine of the angle between the viewing direction and the reflected light direction determines the brightness of the highlight.



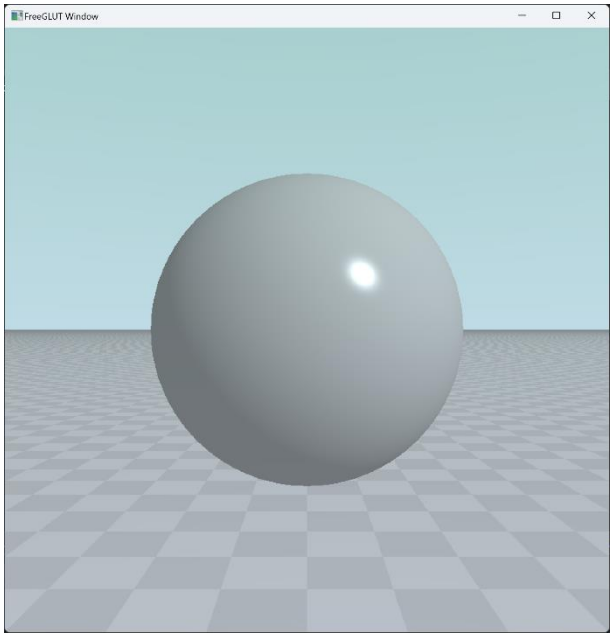
Diffuse Specular

Write your implementation in the fragment shader, `shaders/f.glsl` TODO 3 and 3.5. Note, TODO 3 is about shading, TODO 3.5 is the specific implementation of reflection. GLSL has a built-in function called reflect, which should not be used in your homework. The reflection equation is:

$$R = I - 2(N \cdot I)N$$

Where I is the incoming vector, the N is the surface normal.

The correct implementation should have this view:



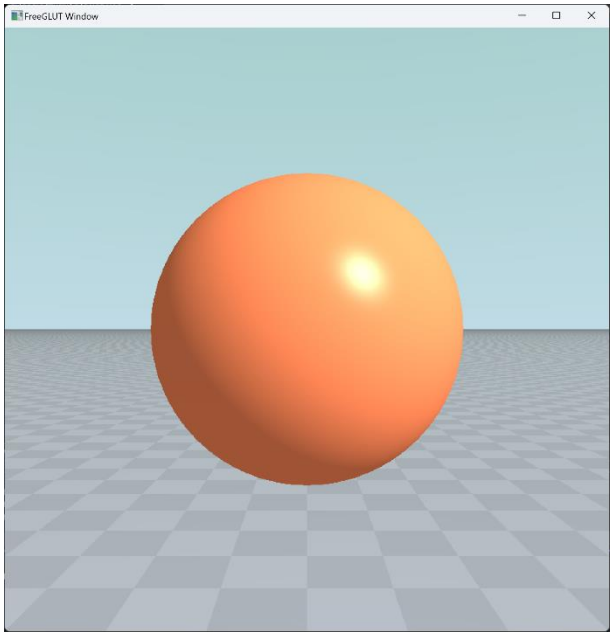
4. Setting Material (10%)
   To simulate realistic interactions between light and objects, we define material properties for each surface. We have provided default material for each object.

   In our framework, a material is defined by the following properties:
   Ambient Color (ka): The color reflected under ambient lighting.
   Diffuse Color (kd): The color reflected under direct light.
   Specular Color (ks): The color of the specular highlight.
   Shininess (n): Controls the size and intensity of the specular highlight.
   These properties are grouped into a material struct for organization.

Then the lighting for a fragment is calculated as:

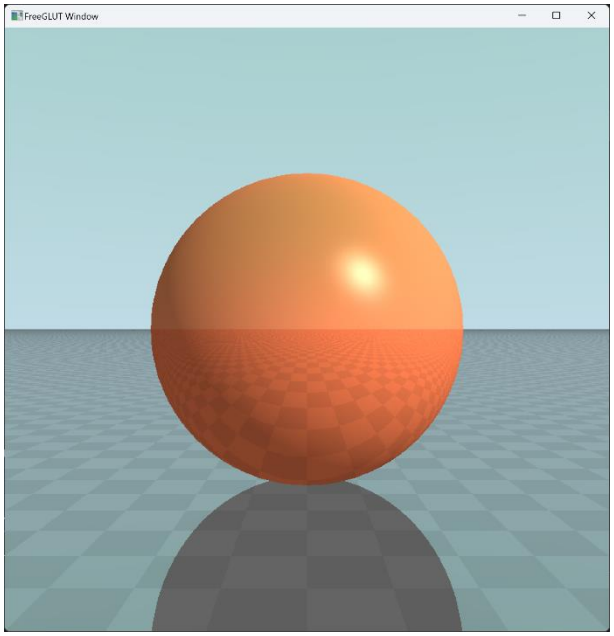
$$ret = (a \cdot ka + d \cdot kd + s^n \cdot ks) * light\_color$$

When submitting your file, you need to change the material property of the ball (you don't need to change the materials for ground and sky). We provide a set of parameters that you can use in your submitted file:
ka = (1.0, 0.5, 0.31)
kd = (1.0, 0.5, 0.31)
ks = (0.5, 0.5, 0.5)
n = 32
But we encourage you to test and submit in different values, (for example, make it "gold" or "pewter"), and have a better understanding how different parameters affects the lighting effects.
The correct implementation should have this view:



5. Specular Reflection (20%)
   Reflection is an effect when the ray hits an object, a flat surface will reflect the ray, and the reflected ray may hit some other objects in the scene, as introduced in the ray tracing lecture. Here we implement a very simple reflection effect to visualize the reflection effects. The idea is to shoot ray based on the specular reflection, use the pre-defined ray-scene intersection function to get the intersection result. And replace the light color in the previous Phong shading model to see the reflection effects.

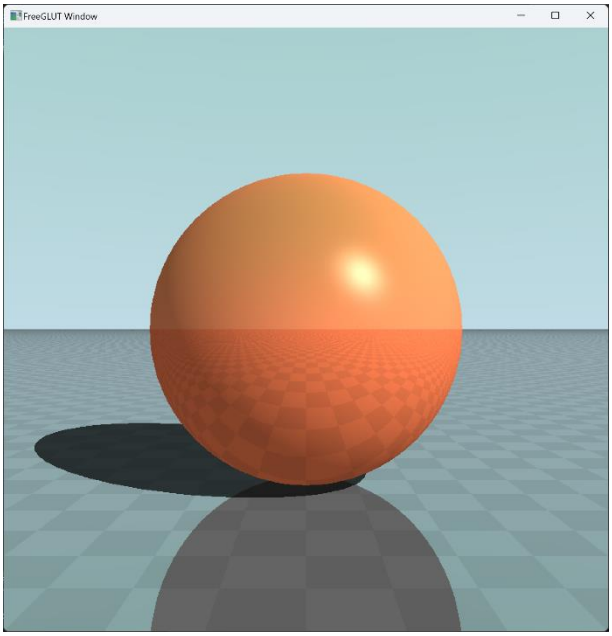
   The correct implementation should have this view:



6. Hard Shadow (Bonus 10%)
   A hard shadow is coming from geometry occlusion. To be specific, shadows will be cast when an occluder blocks the light rays. You are encouraged to implement this feature to make the scene look more realistic.

   Similar to the specular reflection effect, you should rely on ray tracing technique to render the hard shadow.

   The correct implementation should have this view:



Turn-in:
To give in the assignment, please use Brightspace. Give in a zip file with your complete project (project files, source code, and precompiled executable). The assignment is due BEFORE class on the due date. It is your responsibility to make sure the assignment is delivered/dated before it is due. If you wish to receive confirmation of receipt, please ask by email in advance.

Don’t wait until the last moment to hand in the assignment!

For grading, the program will be compiled on Linux and run from the terminal (with Visual Studio as a fallback – please try to avoid platform-specific code (e.g., don’t #include <windows.h>)), run without command line arguments, and the code will be inspected. If the program does not compile, zero points will be given. If you have more questions, please ask on Piazza!

Start early and good luck!