CS33400/ECE30834: Assignment #2 - GPU it!

Basic Lighting, Materials, and Introductory Ray Tracing

Out: September 18, 2026 **Due:** October 2, 2026

Objective

This assignment is designed to give you experience with GPU programming in a fragment shader, surface-normal computation, and local lighting. The main required component is a Phong-style lighting model with ambient, diffuse, and specular terms. You will also implement a deliberately simplified one-bounce reflection effect as an introduction to ray-tracing ideas. An optional bonus task explores sphere-to-sphere reflection.

The provided scene contains **two fixed spheres**, an infinite ground plane, a sky background, and a moving point light. The two spheres have radius 1 and remain fixed throughout the assignment so that lighting and reflection changes are easier to observe.

The template now uses **GLFW** for window creation and input handling. Most assignment work remains in `shaders/f.glsl`.

Summary of Tasks

Part	Task	Weight
1	Shader hot reload	20%
2	Sphere normal calculation	10%
3	Phong shading + reflection-direction formula	40%
4	Two-sphere material setup	10%
5	Simplified one-bounce reflection	20%
6	Sphere-to-sphere reflection	Bonus 10%

1. Warm-up: Shader Hot Reload (20%)

Start with the GLFW template from the course website. As in previous assignments, first make sure the provided project compiles and runs on your development machine.

Most of the code you will modify in this assignment is in `shaders/f.glsl`. Rebuilding the entire C++ project after every shader edit is unnecessary. In `main.cpp`, complete **TODO 1** so that pressing the `R` key recompiles/reloads the shader program while the application is running.

To test your implementation, temporarily add the following line after the final shading result is assigned in `f.glsl`:

```
outCol = vec3(1.0, 0.0, 0.0);
```

Save the shader and press `R`. The window should become red without restarting the executable. Remove the temporary line and press `R` again to restore the scene.

2. Normal Calculation (10%)

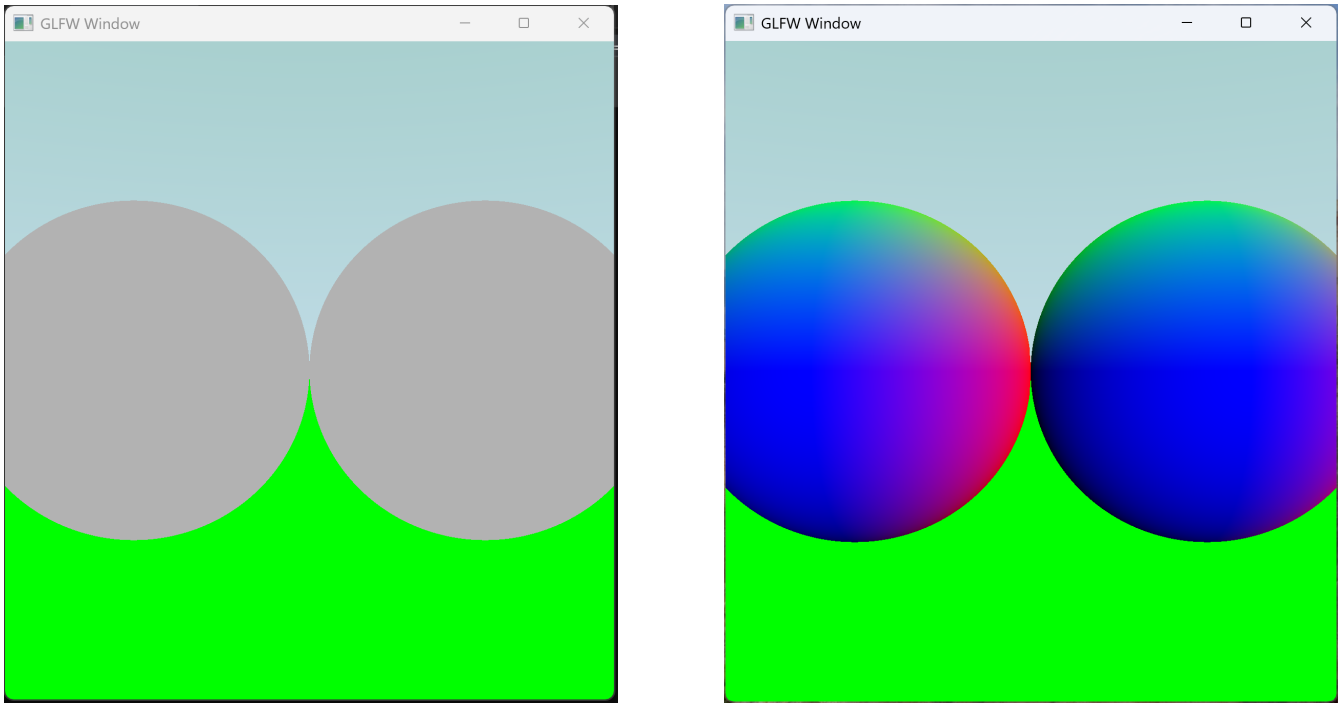
Lighting models require the surface normal at each intersection point. The ray-sphere intersection routines are already provided. Your task is to compute the normal for a point `pos` on each sphere in `f.glsl`.

The spheres are represented as `vec4` values:

- `sphere1.xyz` and `sphere2.xyz`: sphere centers,
- `sphere1.w` and `sphere2.w`: sphere radii.

Because the spheres have different centers, make sure the normal for each intersection is computed using the center of the sphere that was actually hit.

Before TODO 2 is completed, the template uses a placeholder normal. After TODO 2 is correct, raw normals are displayed directly as RGB values. The plane normal is $(0, 1, 0)$ and therefore appears green.



(a) Starter / placeholder normals.

(b) After TODO 2: correct sphere normals.

Figure 1: Normal-calculation stage.

3. Phong Shading (40%)

After the surface normals are correct, implement local Phong-style illumination in **TODO 3**. Lighting calculations are performed per fragment in `f.glsl`. Each material contains ambient, diffuse, and specular coefficients together with a shininess exponent.

For a surface point P with unit normal N, let

$$L = \frac{\text{light} - P}{\|\text{light} - P\|}.$$

Ambient. The ambient contribution is a constant approximation to indirect illumination and does not

depend on the light or viewing direction.

Diffuse. The diffuse term depends on the angle between the *light direction* and the surface normal:

$$d = \max(N \cdot L, 0).$$

Do not use the viewing direction for the diffuse term.

Specular. The specular term produces a view-dependent highlight. In the provided code, you will use the reflection helper from TODO 3.5 and compare the reflected direction with the appropriate viewing/light direction. Be consistent about direction signs.

TODO 3.5: Reflection Direction

Implement the reflection-direction formula yourself rather than using GLSL's built-in `reflect()` function:

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

where I is the incoming direction and N is the unit surface normal.

At the end of this stage, both spheres use the provided neutral/default material so that the effect of the lighting computation is easy to inspect.

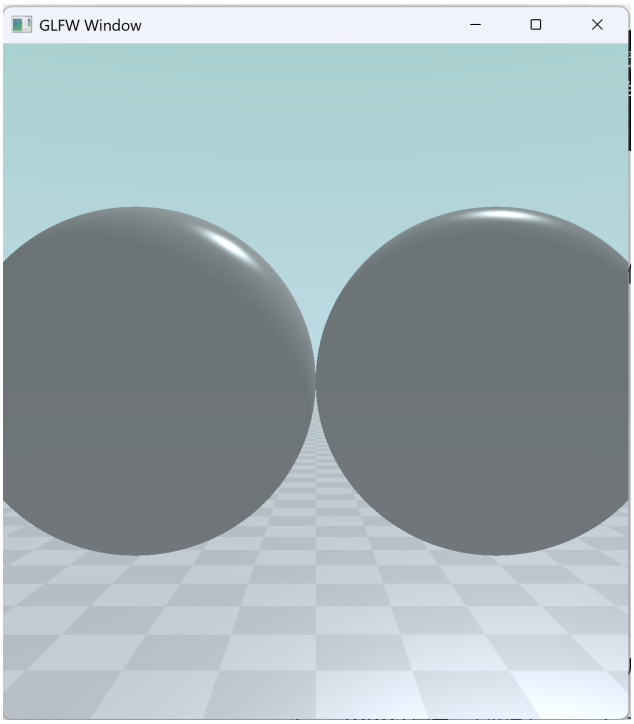


Figure 2: After TODO 3/3.5: neutral-material Phong shading with diffuse and specular terms.

4. Setting Materials (10%)

A material controls how a surface responds to lighting. The provided `Material` structure stores:

- k_a: ambient color,
- k_d: diffuse color,

- k_s: specular-highlight color,
- n: shininess exponent.

For TODO 4, assign **different materials to the two spheres** so that their diffuse colors and specular highlights are clearly distinguishable under the same moving light. The supplied reference configuration uses an orange sphere and a blue sphere with different specular/shininess settings. You may use the supplied values or choose reasonable alternatives, but the two materials should remain visibly different.

For reference, one possible configuration is:

```
// Sphere 1 (orange)
ka = kd = vec3(1.0, 0.5, 0.31);
ks = vec3(0.5);
n  = 32.0;

// Sphere 2 (blue)
ka = kd = vec3(0.15, 0.35, 0.80);
ks = vec3(0.90);
n  = 100.0;
```

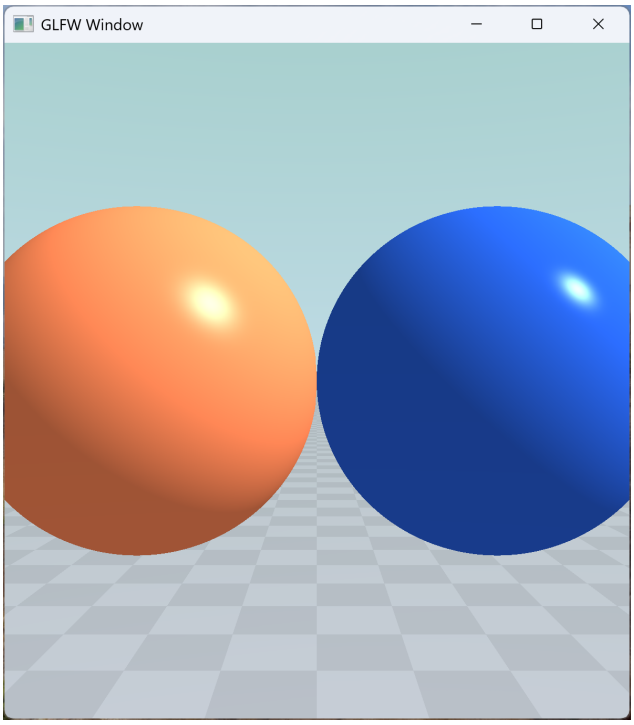


Figure 3: After TODO 4: two distinct Phong materials under the same point light.

5. Simplified Specular Reflection (20%)

This part introduces a single secondary ray. When a primary camera ray hits a surface, compute the reflected direction and trace one additional ray through the scene.

The required version intentionally uses a simplified reflection model so that the assignment remains focused on lighting fundamentals:

- A reflection ray launched from a **sphere** uses the provided `ray_scene_intersect_skip_spheres()` helper. If it would hit a sphere, the helper continues along the same direction until the ray reaches the ground or sky. Thus, sphere-to-sphere mirror reflections are intentionally omitted in the required portion.
- A reflection ray launched from the **ground** uses the normal scene-intersection routine and may therefore see either sphere.
- Use a small positive offset along the reflected direction when launching the secondary ray to reduce self-intersection artifacts.
- As in the original assignment, use the secondary intersection result to modify the simplified reflection color in the final shading computation.

Only one secondary ray is required; recursive/multiple-bounce reflection is not part of the required assignment.

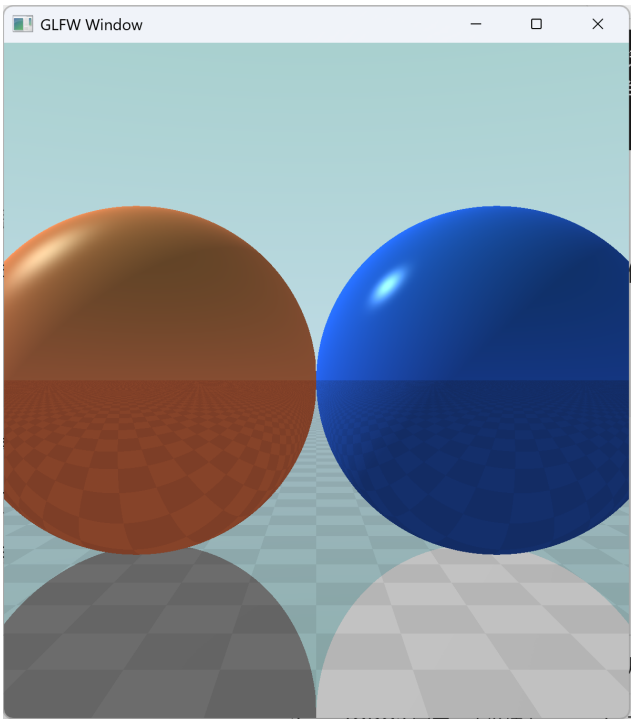


Figure 4: After TODO 5: simplified one-bounce reflections. Sphere reflections show the ground/sky, while the ground can reflect the spheres.

6. Sphere-to-Sphere Reflection (Bonus 10%)

The required TODO 5 deliberately ignores sphere intersections when tracing a reflection ray from a sphere. For this bonus, remove that simplification for the sphere-to-sphere case.

If a reflected ray from one sphere hits the other sphere:

1. keep the secondary sphere intersection rather than skipping it,
2. compute the secondary hit's **local Phong appearance** using its position, normal, and material,
3. combine that secondary color with the primary sphere's local appearance.

Only **one** sphere-to-sphere reflection bounce is required:

$$\text{camera} \rightarrow \text{sphere A} \rightarrow \text{sphere B} \rightarrow \text{stop.}$$

Do not recursively trace another mirror ray from sphere B. A simple fixed blend between the primary local color and secondary-sphere color is sufficient; the reference solution uses a reflection weight of 0.35.

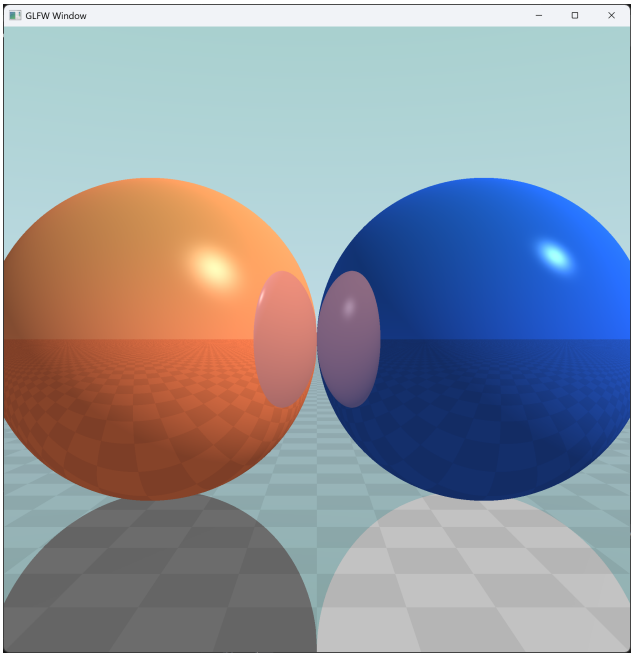


Figure 5: Bonus: one-bounce sphere-to-sphere reflection.

Turn-in

Submit a ZIP archive through Brightspace containing your complete project, including project/build files, source code, shaders, and a precompiled executable for your development platform.

It is your responsibility to verify that your submission is complete and dated before the deadline.

For grading, the project may be compiled on Linux and run from the terminal, with Visual Studio as a fallback. Avoid unnecessary platform-specific code. If your submission does not compile in the grading environment, it may receive no credit for functionality that cannot be evaluated.

If you have questions, please use the course discussion board. Start early and good luck!