CS 3451 - Computer Graphics **Fall 2025**

Assignment 4: Lighting and Shading

You have created your mesh models, but you may not yet know how to put colors onto its surface. This assignment is designed to instruct you on implementing realistic lighting effects on these models. Additionally, you will acquire skills in GLSL programming throughout this process.

1 Reading

To effectively implement this assignment, it's crucial to understand the mathematical foundations of lighting and shading algorithms. I highly recommend studying the course slides for a solid grasp of these concepts before coding. Additionally, specific book chapters and tutorials have been curated below for a deeper understanding of GLSL syntax. These resources are essential for mastering GLSL programming and will equip you with the skills to implement complex shading models and dynamic lighting effects in GLSL shaders.

- Course Slides on Lighting and Shading
- Tiger Book Chapter 10
- https://learnopengl.com/Getting-started/Shaders
- https://thebookofshaders.com/

2 Starter Code

Make sure to execute 'git pull' from our git repository to ensure you have the most recent updates. After pulling the latest code, make sure to run the setup script to recreate the project files (for Windows: run `.\scripts\setup.bat`; for Mac or Linux: run `./scripts/setup.sh`). The starter code for this assignment, found under `assignments/a4/`, includes `main.cpp`, `a4_vert.vert` and `a4_frag.frag`. Upon successful compilation and execution, you should observe the silhouette of a bunny together with two spheres at the center of the window, all colored black, as depicted in Fig. 1. We highly recommend you read through the provided starter code, primarily focusing on the fragment shader at `a4_frag.frag`. You may first review the code's structure thoroughly, paying special attention to the variables declared at the beginning and their respective types and mathematical meanings. Also, before commencing your coding efforts, ensure you understand the seven steps outlined within the `main()` function. This preparatory review is essential for successfully implementing your coding tasks.

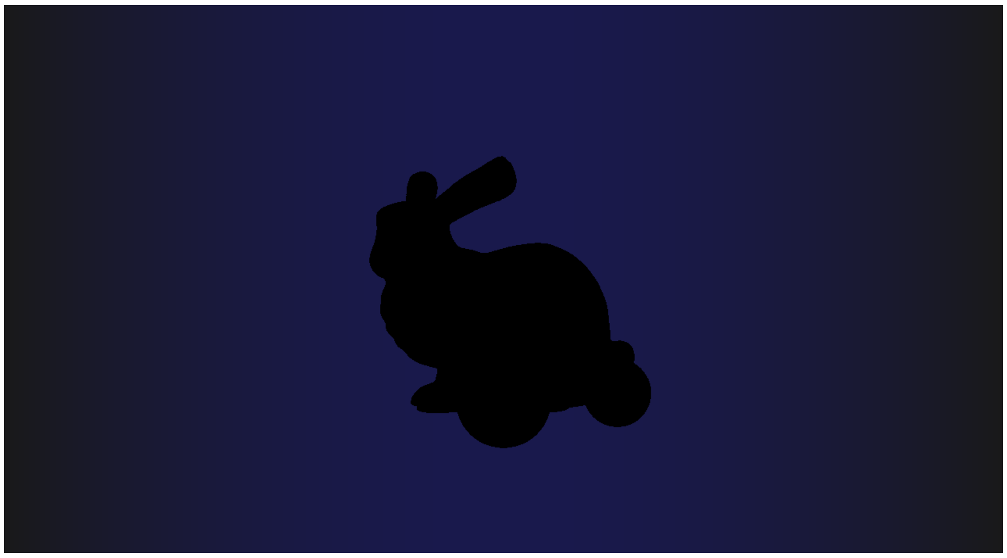

Figure 1: The default display of our scene setup with a bunny and two spheres.

3 Implementation Tasks

You will implement various shading techniques, including normal visualization, ambient shading, Lambertian shading, and Phong shading, and introduce dynamic lighting by manipulating light source properties over time. Additionally, you will explore the application of multiple light sources and develop a customized lighting effect to enhance the visual realism of 3D objects. Next, we describe the seven implementation tasks with expected intermediate results.

Step 1: Visualize Normal Vectors as Color Implement the `shading_normal` function in the fragment shader to convert normal vectors to color. The data pipeline passes the normal vectors into the fragment shader from the vertex shader as a global variable `vtx_normal`. This variable can be used in any function declared in the fragment shader, as demonstrated in the first line of code in the `shading_normal` function. Your task for this step involves **mapping the components of the input normal vector from a range of [-1, 1] to [0, 1]** as the returned vector for visualization purposes. If everything is implemented correctly, you should be able to see a colorful bunny, as shown in Figure 2.

Step 2: Ambient Shading Next, you will develop the `shading_ambient` function to emulate the impact of ambient lighting on an object. Ambient shading represents the most fundamental form of lighting, predicated on the assumption that light is uniformly dispersed across the environment. The mathematical formula for this model is presented as follows:

$$\boldsymbol{L}_{ambient} = k_a \boldsymbol{I}_a, \tag{1}$$

where both k_a and $\boldsymbol{I}_a$ are with the type of `vec3`, indicating the intensity of the `R`, `G`, `B` components respectively. Here, k_a is introduced as a `uniform` variable transferred from the CPU (for insight on how this variable is passed from the CPU to the GPU, refer to the function `Create_Bunny_Scene()` within `main.cpp`, akin to the transfer of other uniform variables like k_d, k_s, and *shininess*, which

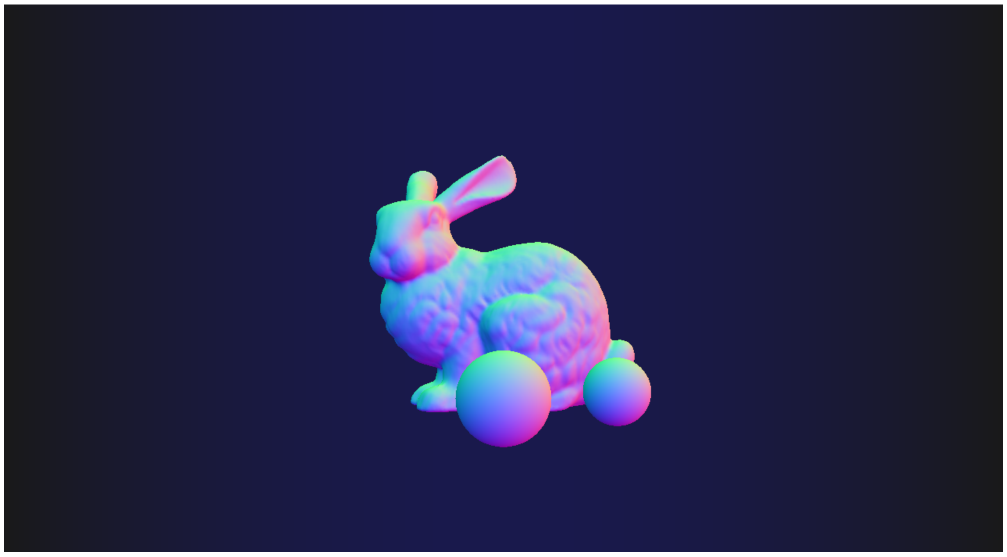
Figure 2: Visualizing normal vectors as colors on object surface.

you will employ in subsequent steps). The operation is executed component-wise, facilitated directly by the `*` operator applied between two `vec3` variables. A correct implementation will result in a rendering as depicted in Figure 3, showcasing each object in a distinct color. These colors are dim due to the small ambient component we set in the light source.

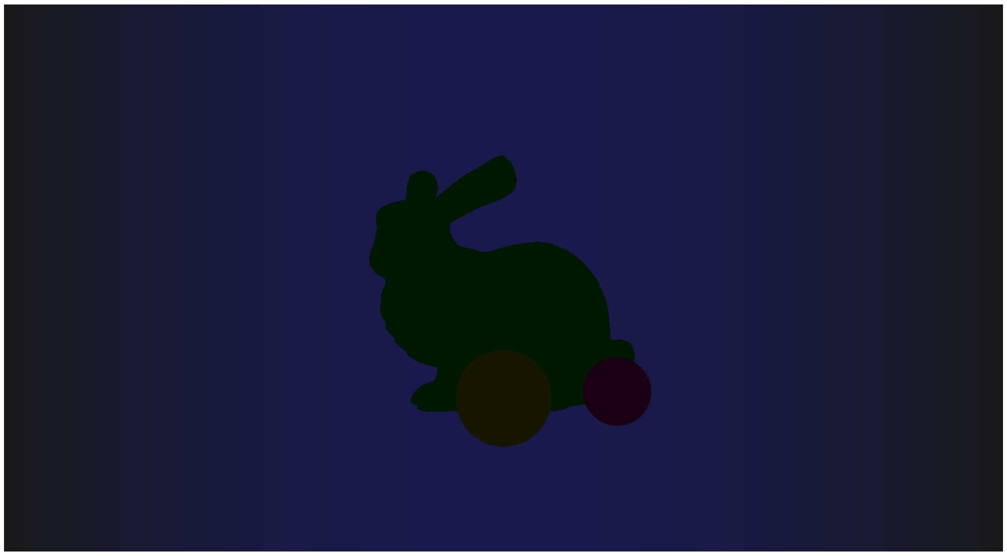
Figure 3: Ambient shading

Step 3: Lambertian Shading Next, you will implement the `shading_lambertian` function to apply the Lambertian shading model. This model simulates the diffuse reflection of light from a rough surface, where the intensity of the reflected light is directly proportional to the cosine of the angle between the light source direction and the surface normal. The mathematical formula for

Lambertian model is presented as follows:

$$\boldsymbol{L}_{Lambertian} = k_a \boldsymbol{I}_a + k_d \boldsymbol{I}_d \max(0, \boldsymbol{l} \cdot \boldsymbol{n}), \tag{2}$$

with k_d and $\boldsymbol{I}_d$ indicating the material diffuse property and light diffuse intensity (each as a `vec3`), and $\boldsymbol{l}$ and $\boldsymbol{n}$ specifying the light direction (pointing from the surface to the light source) and surface normal direction separately. You may refer to our course slides for a detailed illustration of the geometric picture of these vectors. Both $\boldsymbol{l}$ and $\boldsymbol{n}$ are **normalized unit vectors**, which you might want to pay special attention to in your implementation. Again, the multiplication between k_a and $\boldsymbol{I}_a$ are carried out in a component-wise way. After your implementation, you should be able to get a diffusive surface, as shown in Figure 4.

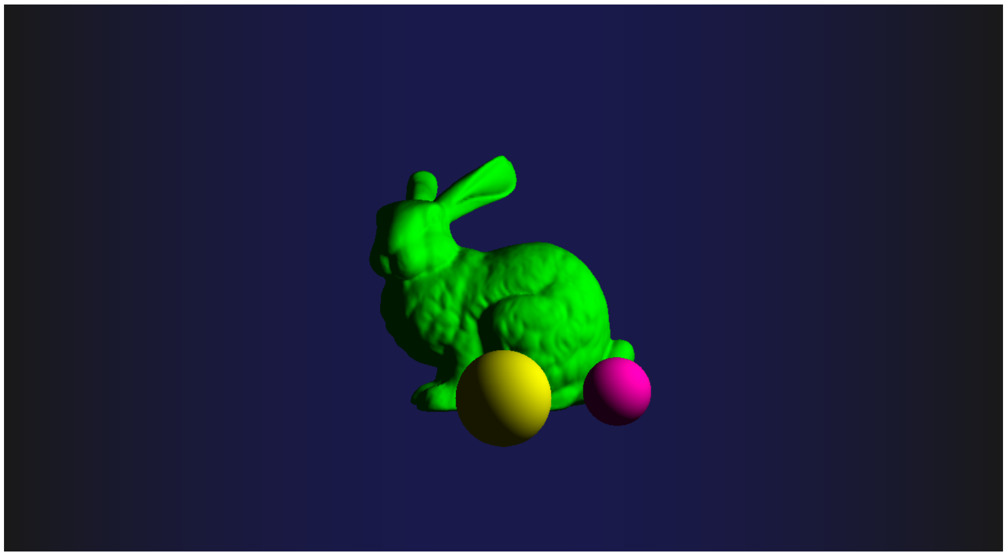

Figure 4: Lambertian shading

Step 4: Phong shading Your next task is to develop the `shading_phong` function to incorporate the Phong shading model. This model includes ambient, diffuse, and specular reflections, providing a more realistic representation of how light interacts with shining surfaces. In addition to the previous implementation, the Phong shading model requires calculating vectors representing the viewer's direction, the light source direction, and the reflection direction. The mathematical formula for the Phong model is presented as follows:

$$\boldsymbol{L}_{Phong} = k_a \boldsymbol{I}_a + k_d \boldsymbol{I}_d \max(0, \boldsymbol{l} \cdot \boldsymbol{n}) + k_s \boldsymbol{I}_s \max(0, \boldsymbol{v} \cdot \boldsymbol{r})^p, \tag{3}$$

with k_s and $\boldsymbol{I}_s$ indicating the material specular property and light specular intensity (each as a `vec3`), $\boldsymbol{v}$ representing the direction from the object's surface point to the eye position, $\boldsymbol{r}$ representing the reflection direction from the light source, and p representing the power index controlling the shininess of the highlight region. You may refer to our course slides for a detailed illustration of the geometric picture of these vectors. All $\boldsymbol{v}$, $\boldsymbol{l}$ and $\boldsymbol{r}$ are **normalized unit vectors**. You might want to consider using the GLSL `reflect()` function to calculate the reflected vector. If everything is implemented correctly, you should be able to see the following effect as shown in Figure 5. Move the camera around to feel the light flow on the object's surface. Specifically, notice the highlights on the surface of each object and pay attention to the size of each highlight region and their relation to the value of `shininess` set in `main.cpp`.

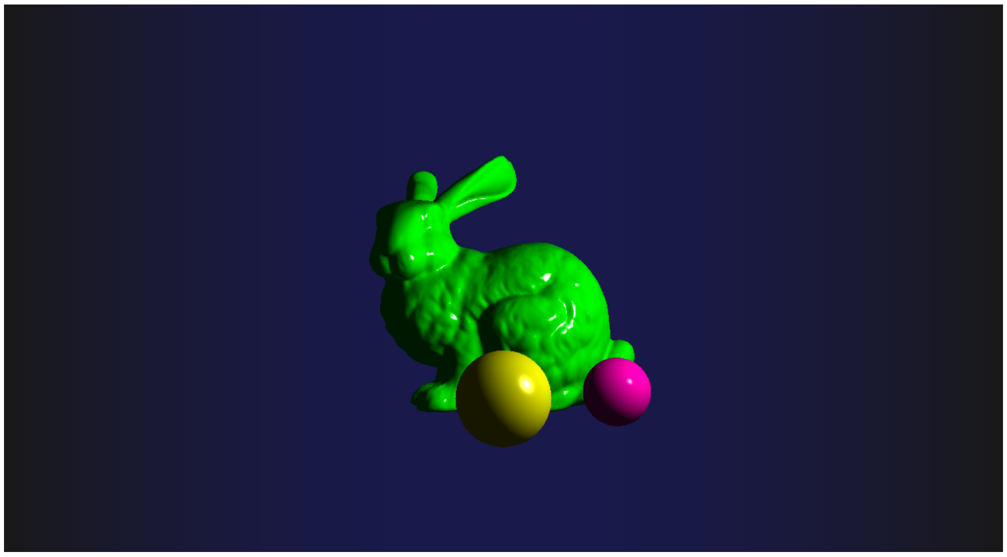
Figure 5: Phong shading

Step 5: Multiple Lights Next, you will extend the shading model to support multiple light sources. Declare an additional point light source with specified properties and modify the `main()` function in the fragment shader to include its contribution. This step demonstrates the cumulative effect of multiple lights on the appearance of objects. Specifically, the additional light source is **a point light, with its position in (-3, 1, 3), ambient coefficient (0.05, 0.02, 0.03), and both diffuse and specular coefficients (0.4, 0.2, 0.3).** Your task is to declare a light struct with the above specifications, call the `shade_phong` you have implemented in the previous step, and add the colors calculated from the two light sources together for the final fragment color.

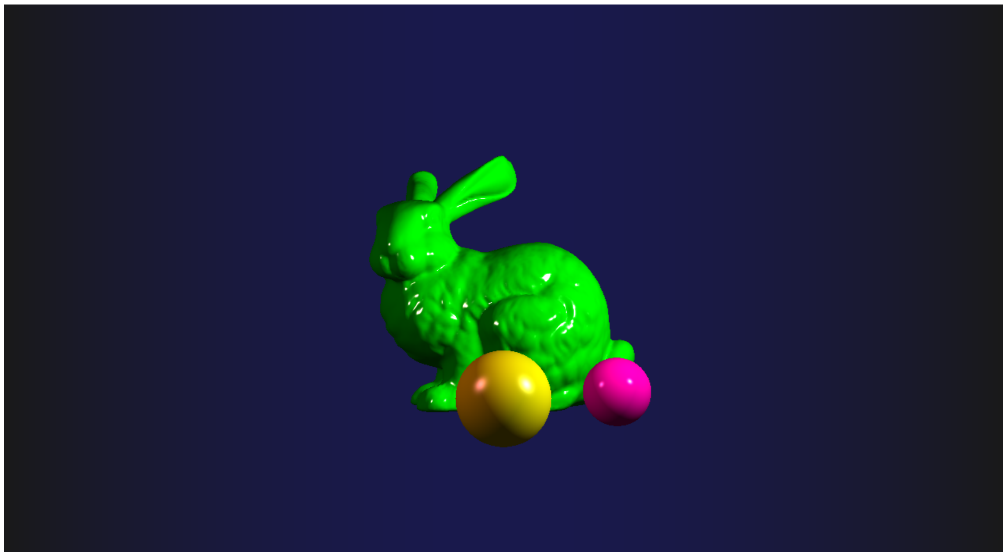
Figure 6: Phong shading with two lights

Step 6: Dynamic Light Source Your task is to implement the `get_spinny_light` function, which aims to animate a dynamic light source rotating around the Z axis. This entails modifying the transformation matrix to facilitate the desired rotation angle (which we have already defined for you in the starter code), effectively simulating the temporal movement of the light source and its resultant impact on object shading. It's important to remember that GLSL stores matrices in **column-major** order, meaning that elements are arranged by columns first. Consequently, when specifying matrix elements in what appears to be a row-major format, you are essentially defining the **transpose** of the intended matrix. **Press `p` to start the animation** after you implement the rotation matrix. With everything implemented correctly, you should see beautiful light flow moving around the object's surface (we took a screenshot of one frame as below in Figure 7).

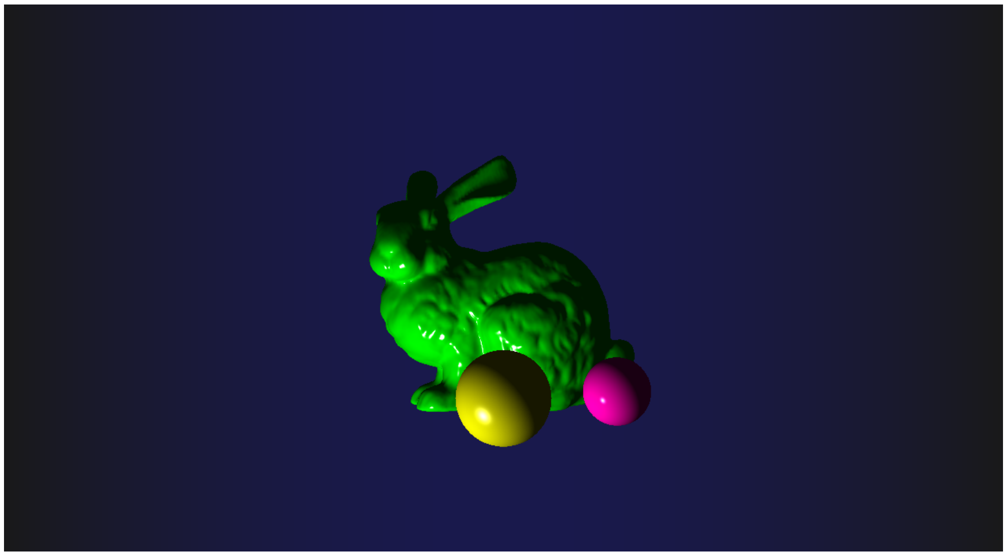

Figure 7: Phong shading with a dynamic light source

4 Creative Expression

The application of lighting and shading techniques offers considerable creative latitude in crafting images and animations that approach the complexity and realism of actual scenes. You can create visually appealing objects by integrating these methods with geometric models you've previously developed. In the Creative Expression section for this assignment, you will practice placing new mesh objects into the scene and setting lights for them. Your tasks for this part are twofold: (1) You want to add your customized mesh objects and specify their transform (which you should be familiar with from your previous assignment) and material properties in the function of `Create_Shining_Scene()` in `main.cpp` by mimicking `Create_Bunny_Scene()`. Pay special attention to the values of the shading material properties (i.e., k_a, k_s, k_d and *shininess*) we used for the bunny example. (2) You want to adjust (or add) light sources within the fragment shader `a4_frag.frag` to create realistic lighting and shading effects for your objects. You are encouraged to combine the techniques you acquired from previous tasks to achieve a visually appealing rendering of shining objects. Look for inspiration from previous works (check `students_work_a4_2024.pdf` on Canvas).

The theme of this Creative Expression is **Born to Shine**!

5 Submission

Submit the following components for evaluation:

- Your source code `main.cpp` and `a4_frag.frag`;
- Screenshots demonstrating the correctness of your implementation for Steps 1-5;
- A video demonstrating the correctness of your implementation for Step 6 (dynamic lighting);
- A video or a few screenshots from different camera angles demonstrating your customized shining object scene;
- A concise paragraph that provides a technical explanation of your implementation for the customized scene.

6 Grading

This assignment is worth a total of 8 points, with the grading criteria outlined as follows:

1. Technical contribution (7 pts): The core of the grading is based on the correct implementation of lighting and shading models. The distribution of points is as follows:
   - **Step 1-2 (normal and ambient):** 1 point
   - **Step 3 (Lambertian):** 2 points
   - **Step 4 (Phong):** 2 points
   - **Step 5 (multiple lights):** 1 points
   - **Step 6 (dynamic light):** 1 points
2. Creative expression (**Step 7**) (1 point): This aspect focuses on your ability to create new shining objects by applying lighting models to 3D mesh objects.

7 Sharing your Work

You are encouraged to share your graphical work with the class. If you want to do so, please upload your image to the Ed post **A4 Gallery: Born to Shine!**. This is an excellent opportunity to engage with your peers and gain recognition for your work. We look forward to seeing you shine with your excellent renderings!