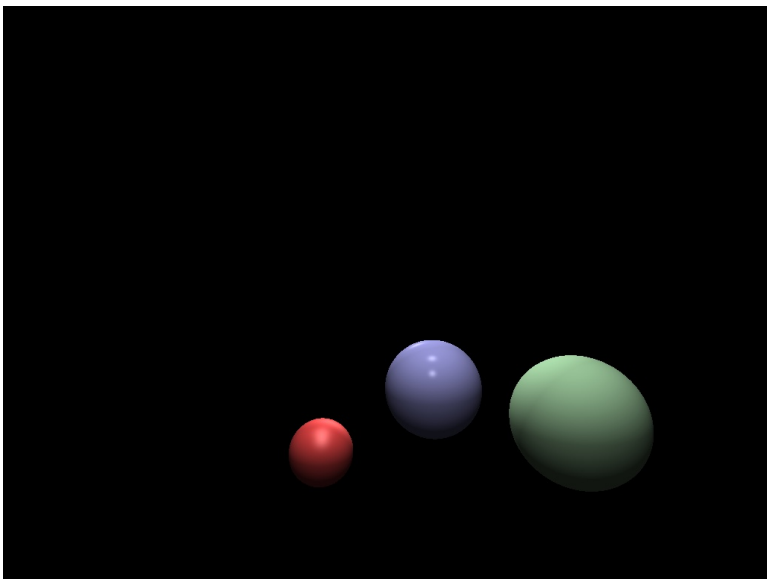
Università
della
Svizzera
italiana

Faculty
of Informatics

Computer Graphics (Fall 2021)

Assignment 2: Phong lighting model

September 29, 2021



Your main task in this assignment will be to implement Phong lighting model to compute the above image. You will be given an updated framework with clearly marked places for implementing the required functionality. The updated framework is also the solution for the previous assignment.

Updated framework

We updated the framework by:

- adding *Material.h* file, which contains the structure for describing all the parameters of the Phong lighting model,
- extending the definition of the *Object* class by adding variable `material` to store the material information as well as functions `getMaterial()` and `setMaterial()`,
- adding a new constructor for the *Sphere* class, which takes as an argument the material structure,
- adding a new class *Light*, to represent a point light source, i.e., its position and the intensity,
- declaring additional variables for lights, i.e., array of point light sources, `lights`, and the intensity of the ambient light, `ambient_light`,
- declaring function `PhongModel` for implementing the lighting model.

The code is commented. Before continuing with the assignment, please familiarize yourself with the updates and check the code for comments indicating the places which should be modified.

Exercise 1 [10 points]

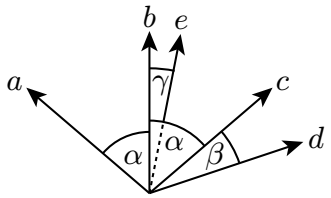
Implement the Phong lighting model as indicated by the comments in the code. Modify the scene definition such that it includes the following objects:

	Blue sphere	Red sphere	Green sphere
center	$(1.0, -2.0, 8.0)$	$(-1.0, -2.5, 6.0)$	$(3.0, -2.0, 6.0)$
radius	1.0	0.5	1.0
ρ_a	$(0.07, 0.07, 0.1)$	$(0.01, 0.03, 0.03)$	$(0.07, 0.09, 0.07)$
ρ_d	$(0.7, 0.7, 1.0)$	$(1.0, 0.3, 0.3)$	$(0.7, 0.9, 0.7)$
ρ_s	$(0.6, 0.6, 0.6)$	$(0.5, 0.5, 0.5)$	$(0.0, 0.0, 0.0)$
k	100.0	10.0	0.0

and three point light sources with positions at $(0.0, 26.0, 5.0)$, $(0.0, 1.0, 12.0)$, $(0.0, 5.0, 1.0)$, each emitting the light with intensity $(0.4, 0.4, 0.4)$. Upon completion of this exercise, you should be able to produce an image same as the one on top of this document.
In this exercise, you can use the in-built function `glm:reflect()` to compute reflected direction. If you do so, please read the documentation of this function and pay attention to the orientation of the directions you provide as an input to this function.

Exercise 2 [5 points]

Consider the configuration of planar vectors shown on the right. Let α be the angle between the unit vectors a and b and let $c = 2b\langle a, b\rangle - a$, so that the angle between b and c is also α. We further consider the unit vector d and denote the angle between c and d by β. We finally compute the *halfway-vector* $e = \frac{a+d}{\|a+d\|}$ and denote the angle between b and e by γ. Show that $\gamma = \beta/2$. Note that d may also lie on the other side of c, i.e. between b and c or even between a and b, so you need to distinguish these different cases.



Bonus Exercise 3 [2 points]

Imagine it is night and that you see a full moon in the sky. Ignoring the shading artifacts caused by craters, it appears as a white disk with constant brightness (like the red sphere from the first assignment), rather than a sphere shaded in different intensities. What could be the reason for this?

Bonus Exercise 4 [5 points]

Use the solution to the Exercise 1 to create a short (2 seconds) animation where one of the light sources moves in a circle above the three spheres. To be able to localize the light sources, visualize them with small spheres. One way of modifying the code to generate an animation is to introduce an argument for the program, which can be set from the command line (Terminal) and is the time stamp or simply frame index. Using this additional argument, the raytracer can position the light source in the desired location and write the output to a unique file which also contains the frame index, e.g., `result_0.ppm`, `result_1.ppm`,... You can write a short bash script to then render all the frames. Now, you only have to compile all the frames into the video sequence. One way of doing it is to use in macOS `sips` function from the terminal to first convert all the PPM files to PNG, for example, `sips -s format png result.ppm --out result.png`. Again you can use a bash script to convert all the files at once. Once you have PNG files that correspond to individual frames, you can load them as a sequence in QuickTimePlayer using *File → Open Image Sequence*. QuickTimePlayer will allow you to save the sequence as a video file which you can later play. Alternatively, you can use FFmpeg[1].

[1] https://ffmpeg.org

Submission

You should submit one ZIP-file via iCorsi containing:

- readme file with information which exercises you solved, the authors of the solutions, and explanation of encountered problems, if any,
- one modified *main.cpp* file for exercises 1,
- a PDF or image file containing solution to exercises 2 and 3,
- if you solve also the Exercise 4, please also include the video file together with a separate *main.cpp* which solves this exercise.

Solutions must be returned on October 6, 2021 via iCorsi3