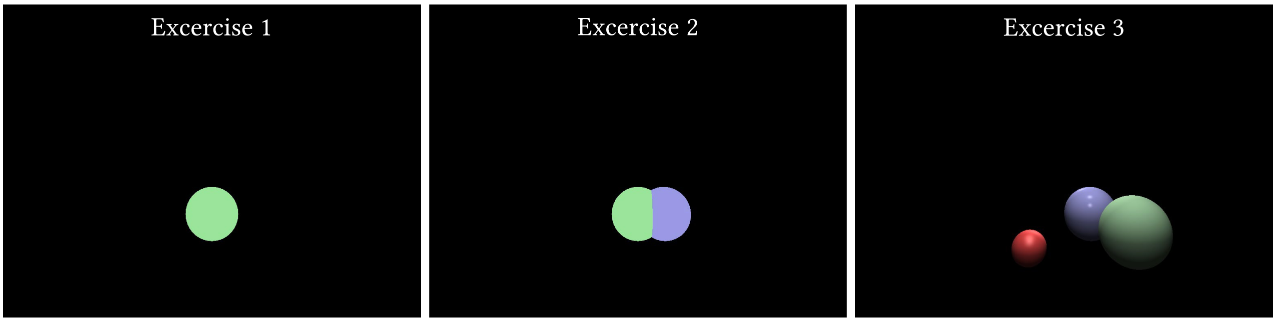
Università
della
Svizzera
italiana

Faculty
of Informatics

Computer Graphics

Assignment 1: Spheres with Phong lighting model



Your main task in this assignment will be implementing the raytracing of spheres with a simple lighting model. You will start by implementing a routine for going over all image pixels, defining the ray, and intersecting it with a scene. Next, you will have to determine the color of each pixel depending on the distance to the intersection points with different objects. Finally, you will implement the Phong lighting model. You will be given a framework containing the raytracer's overall structure and some simple functionalities.

Exercise 1 [7 points]

In this exercise, you are asked to implement your first raytracer capable of creating an image of a sphere. To this end, you need to:

- implement for each pixel of the image the ray creation,
- trace the ray to determine whether it intersects the sphere,
- color the pixel using the color of the sphere, if the ray intersect the sphere, or in black otherwise,
- make sure your code will work well also when the ray origin is not at point $(0, 0, 0)$.

The template code you downloaded with the assignment provides already the structure for the raytracer which we will be extending in the following assignments. Please familiarize yourself with the provided code and look into it for comments regarding the assignment. This time you will need to modify only *main.cpp* file. The places which you should edit are clearly marked in the code.
To compile the raytracer you can use *g++* compiler in the terminal by executing the following command:

```
g++ main.cpp
```

You can then run the code by executing the binary generated by the compiler.
Running the code will create an image *result.ppm* in the same directory. When all the parts of the exercise are implemented correctly, the generated image should contain an image of a green sphere.

In case of any questions please post them directly to the forum dedicated to this assignment on iCorsi. For solving this and next assignments you can also use any of the available development environments. However, keep in mind that the binaries and the output file may be created in a different directory depending on the project settings.

OpenGL Mathematics (GLM)

For our raytracer, we will be using GLM[1] library for mathematics. The full documentation you can find on its website or on iCorsi. Here, you are provided with useful snippets which will let you quickly start solving the assignment without reading the documentation. In fact, these are all you need to solve this assignment.

`glm::vec3 v = glm::vec3(x,y,z);`	declares a vector of three numbers `x`, `y`, and `z`
`glm::vec3(x)`	returns a vec3 with all components equal `x`
`glm::normalize(x)`	returns the normalized version of vector `x`
`glm::distance(p,q)`	computes distance between points `p` and `q`
`glm::dot(x,y)`	returns the value of the dot product between two vectors `x` and `y`

Exercise 2 [2 points]

Modify the `sceneDefinition` function and add another sphere with centre $c = (1.0, -2.0, 8.0)$, radius $r = 1$, and color $(0.6, 0.6, 0.9)$. Now, the raytracer should generate an image of two spheres; see the image at the top of the document. If this is not the case, there are potentially two problems:

- your ray-sphere intersection code does not work properly,
- the code does not correctly resolve occlusions based on the computed distance from the camera to the intersection point.

Play around with the order in which the two spheres are added in the code. No matter which order you use, the image should be the same.

Exercise 3 [6 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	$(1.0, -2.0, 8.0)$	$(-1.0, -2.5, 6.0)$	$(2.0, -2.0, 6.0)$
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 the final image for this assignment.

In this exercise, you can use the in-built function `glm:reflect()` to compute reflected direction. If you do so, make sure you read the documentation of this function and pay attention to the orientation of the directions you provide as an input to this function.

To solve this exercise, pay attention to the following elements of the framework we provide:

- *Material.h* file contains the structure for describing all the parameters of the Phong lighting model,
- the *Object* class contains a variable `material` to store the material information as well as functions `getMaterial()` and `setMaterial()`,
- the *Sphere* class has a constructor which takes as an argument the material structure,

[1] https://github.com/g-truc/glm

- the class *light* represents a point light source, i.e., its position and the intensity,
- variables for lights, i.e., the array of point light sources, `lights` and the intensity of the ambient light, `ambient_light`,
- the function `PhongModel` should implement the lighting model.

Submission

Your submission **must** contain one ZIP-file with:

- a readme file or a PDF document file with information about which exercises you solved, the authors of the solutions, and the explanation of encountered problems, if any,
- an image file, *result.ppm*, containing the final image you could render,
- a directory named *code* containing all the source code used to generate the image.

The source code, upon compilation, should generate the image identical to the submitted *result.ppm* file. Your code should compile by calling *g++ main.cpp*. The ZIP file name must be of a form *surname1_surname2.zip* for a team of two, and *surname.zip* for a single submission. Only one person from the team should submit the solution.

Solutions must be submitted via iCorsi by the indicated there deadline.