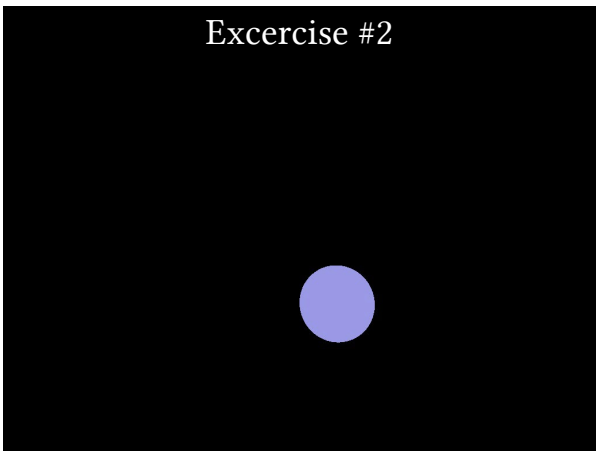
Università della Svizzera italiana

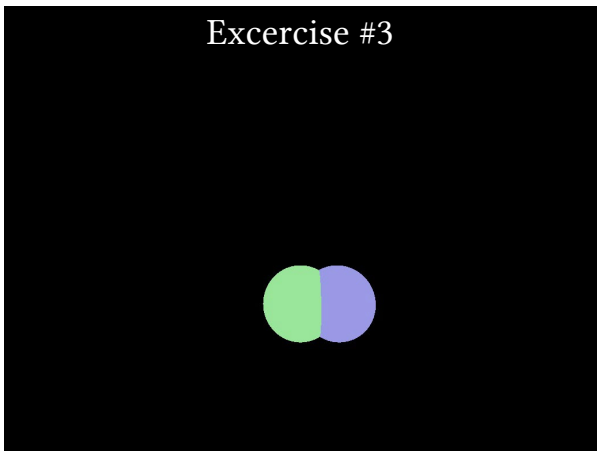
Faculty of Informatics

Computer Graphics (Fall 2022)

Assignment 1: Ray-sphere intersection

September 21, 2022





Your main task in this assignment will be to implement raytracing a sphere. You will be given a framework with some functionality already implemented. You will need to add a routine for going over all pixels of the image, defining the ray, and intersecting it with a scene. Finally, you will have to determine the color of each pixel depending on the distance to the intersection points with different objects.

Exercise 1 [3 points]

Before implementing your first raytracer, let's do some math. You are given two vectors $\mathbf{x} = (\sqrt{2}, 1, 0)^T$, $\mathbf{y} = (1, 1, 1)^T$, and matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \\ -1 & -3 & -3 \end{pmatrix}$$

- Task 1 (1 points): Compute the cosine of the angle between vectors $\mathbf{x}$ and $\mathbf{y}$.
- Task 2 (1 points): Compute vector $\mathbf{z}$ that is the vector perpendicular to vectors $\mathbf{x}$ and $\mathbf{y}$. Because there is many vectors that fulfill this requirement, report the one that additionally has magnitude equal 1, i.e., it is normalized.
- Task 3 (1 point): Compute vector $\mathbf{u}$ defined as $\mathbf{u} = A\mathbf{z}$

Exercise 2 [2 points]

Consider a tasks of raytracing a sphere with center at $c = (1, 1, 1)^T$ and radius $r = \sqrt{2}/2$. Assume that the camera is located at the origin of the coordinate system. Let l be the line that passes through the camera location and the center of the sphere, and d is the direction of a ray that intersect the sphere exactly at one location. Compute the angle between d and l.

Exercise 3 [8 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 using command:

```
./a.out
```

Running the code will create an image *result.ppm* in the same directory. When all the parts of the assignment are implemented correctly, the generated image should look like the one in the top-left of this document, i.e., violet 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 (e.g., XCode). 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 4 [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 shown in the top-right of the document, i.e., two intersecting spheres. 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.

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

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,
- a PDF or image file containing solution to exercises 1 and 2,
- one modified *main.cpp* file for exercises 3 and 4.

Solutions must be returned on September 29, 2022 via iCorsi3