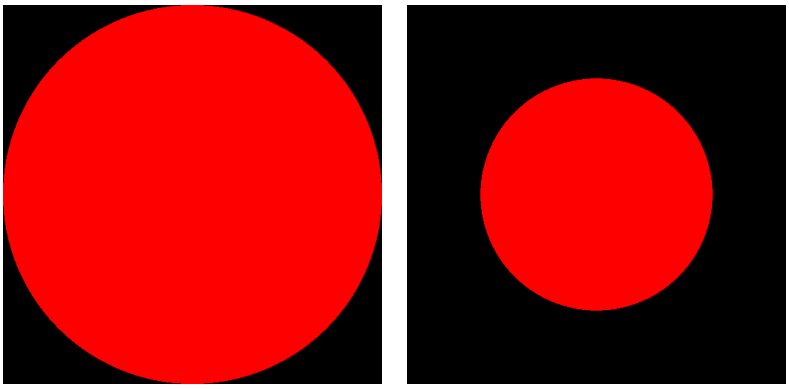
Imatge Sintètica - Assignment 2

Ricardo Marques

2017/2018



(a) Orthographic view (b) Perspective view

Figure 1: Result of ray-sphere intersection tests using an orthographic camera (left) and a perspective camera (right). The pixels with red color correspond to pixels for which the camera ray found an intersection with an object of the scene (in this case, a sphere). For the black pixels, no intersection was detected.

1 Introduction

To complete this assignment you need the framework provided in the previous class (i.e., RTIS - Ray Tracer for "Imatge Sintètica"). **If you do not have** this framewok yet, go to the aula global of this course, download the program RTIS, compile it and execute it.

RTIS is composed of a set of classes which will be the base of your first ray tracer. In Assignment 1, we have made extensive use of the classes Vector3D, Matrix4x4 and Film, with which you should be now familiar. In this assignment we will work with some new classes:

- The *Ray* class which implements the notion of ray.
- The *Shape* abstract class and the *Sphere* class which is derived from the *Shape* class
- The *Camera* abstract class and the two classes derived from this parent class: *Orthographic* and *Perspective* which implement the two projective camera models presented in Lecture 2.

Please take a look at the interface of these classes (defined in the header files). This will give you an idea of the main operations you can make with them.

Note that **some of these classes have incomplete methods** (functions). Your task consists of using the knowledge acquired in Lecture 2 to complete the provided skeleton.

2 Objective

At the end of this assignment your ray tracer should be able to produce two images as shown in Fig. 1. To reach this goal you will have to **complete the skeleton of the functions corresponding to Assignment 2** in RTIS. The rest of this assignment will guide you through this task by giving you a set of exercises which will allow you to progressively complete the missing parts of the RTIS. The main steps are the following:

1. Complete the Sphere class, making it able to determine if a given ray intersects a sphere or not.
2. Complete the orthographic and perspective camera classes, making them able to generate camera rays.
3. Synthesize your first ray-tracing image by writing a ray tracing loop which generates a camera ray for each pixel, tests the intersection of that ray with a sphere placed in front of the camera, and paints the pixel in red if an intersection is detected (otherwise the pixel is painted in black).

3 Using the Equation Solver

3.1 Context

The EqSolver class allows you to create objects capable of solving linear and quadratic equations. This class will thus be useful when computing ray-sphere intersections which, as you saw in the theory class, require solving a quadratic equation. When calling the *rootQuadEq()* method of this class passing as arguments the coefficients of a polynomial of second degree, this method returns false if there are no real roots (solutions) to the equation, and true otherwise. In case there exist solutions to the equation, an auxiliary structure of type *rootValues* (which you can find defined in the file eqsolver.h) is used to store the information about the number of solutions, and their value.

3.2 Your Task

Instantiate an object of the EqSolver class and verify (by printing the result to the screen) that:

$$5x^2 + 6x + 1 = 0 \Leftrightarrow x = -0.2 \quad \text{or} \quad x = -1$$

and that the equation

$$5x^2 + 2x + 1 = 0$$

has no real roots.

4 Completing the Sphere Class

4.1 Context

Computing the intersection of rays with objects is an essential operation in ray tracing. In Lecture 2 we studied the ray-sphere intersection and we noted that, for a sphere centered at $(0, 0, 0)$, the ray-sphere intersection equation is given by:

$$(o_x + t\,d_x)^2 + (o_y + t\,d_y)^2 + (o_z + t\,d_z)^2 = r^2 \tag{1}$$

where:

- $(o_x, o_y, o_z) = \mathbf{o}$ is the **ray origin**;
- $(d_x, d_y, d_z) = \mathbf{d}$ is the **ray direction**;
- r is the **sphere radius**;
- and t is a **scalar**.

If the ray direction $\mathbf{d}$ is normalized and if the ray intersects the sphere, then t gives the distance traversed by the ray until the intersection occurs. Moreover, from Lecture 2, we know that Eq. (1) can be expressed as a quadratic equation of the form:

$$a\,t^2 + b\,t + c = 0 \tag{2}$$

4.2 Your Task

Your **first task** is to complete the method *transformRay()* of the *Matrix4x4* class. This method receives a ray and uses the transformation contained in the *Matrix4x4* object to transform the ray accordingly. **Hint**: a ray is defined by a point $\mathbf{o}$ (specifying the origin) and a vector $\mathbf{d}$ (specifying its direction). Consequently the methods *transformPoint()* and *transformVector()* already implemented in the *Matrix4x4* class can be useful to implement the *transformRay()* method.

Once you have completed the first task, perform the following steps which will help you completing the missing parts of the Sphere class:

1. Develop Eq. (1) and write in the form of Eq. (2). Find the expression of the terms a, b and c.
2. Use the result computed above to complete the method Sphere::rayIntersectP(). This method receives a ray in world coordinates and returns true if the ray intersects the sphere, and false otherwise.
3. In the function *completeSphereClassExercise()*, create a sphere of radius 1 centered at point $\mathbf{p} = (0, 0, 3)$. **Note that** for this you will have to create a transformation matrix (which you can call *objectToWorld*) which contains a translation $\Delta = (0, 0, 3)$, and pass it as argument to the *Sphere* constructor. The role of this transformation matrix is to tell RTIS how to pass from local object coordinates to global world coordinates, which is equivalent to telling RTIS how to place the object in the scene.
4. Verify the sphere radius and location by printing the sphere to screen using the stream insertion operator <<.

5. Create a set of rays and, using the stream insertion operator <<, verify the result of their intersection test with the sphere.

   - Example 1: a ray given by $\mathbf{r} = (0,0,0) + (0,0,1)\,t$ should intersect the sphere.
   - Example 2: a ray given by $\mathbf{r} = (0,0,0) + (0,1,0)\,t$ should not intersect the sphere.

5 Camera Ray Generation & Ray Tracing Loop

5.1 Context

Another essential operation in ray tracing is the **generation of camera rays**, which are used to determine the visible points from the eye position. Recall that in Lecture 2 we have seen that this operation is **simpler if performed in camera space** instead of world space. Also recall that, according to our convention, in camera space all cameras are assumed to be located at $(0,0,0)$, "looking" in the +z direction $(0,0,1)$. **Note that** once the ray has been generated in camera space, it must then be transformed to world coordinates.

5.2 Your Task

The function *generateRay(const Vector3D sample)* of both the orthographic and the perspective cameras is incomplete. This function should receive a point in the image plane (in normalized device coordinates) and return a ray which passes through that image point. Currently, all it does is to convert the point in the image plane to camera coordinates.

Do the following task first for the Orthographic camera, and then for the Perspective camera.

1. Fill the function *generateRay(const Vector3D sample)* so that it returns a ray *in world coordinates* which passes though the image point passed as argument.
2. Using a double loop for image traversal, perform the following steps:
   - Generate a ray passing through the center of the pixel.
   - Test the intersection of the generated ray with a sphere of radius 1 centered at $(0,0,3)$.
   - If the ray intersects the sphere, then paint the pixel in red (1, 0, 0).
   - Otherwise, paint the pixel in black (0, 0, 0).
   - Compare your result with Fig. 1.