Imatge Sintètica - Assignment 4

Ricardo Marques

2017/2018

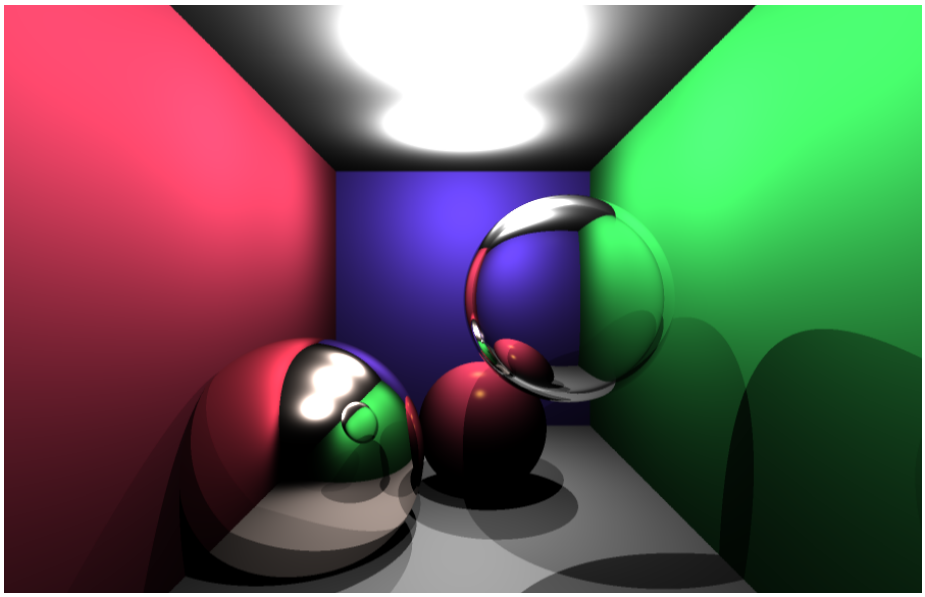

Figure 1: Cornell Box with diffuse, glossy, mirror and glass materials.

1 Introduction

In assignment 3 you have implemented new features in your ray tracer (such as the Phong material, light sources and the direct shader). These features allowed you to synthesize your first ray tracing image in which the lighting conditions of the scene were effectively taken into account to produce the final result. In this assignment, you will continue adding new features to your ray tracer, by applying the knowledge acquired in lecture 4, i.e., ray-plane intersections, perfect specular reflections, and perfect specular transmissions. At the end of this assignment your ray tracer should be able to produce an image similar to the one shown in Fig. 1.

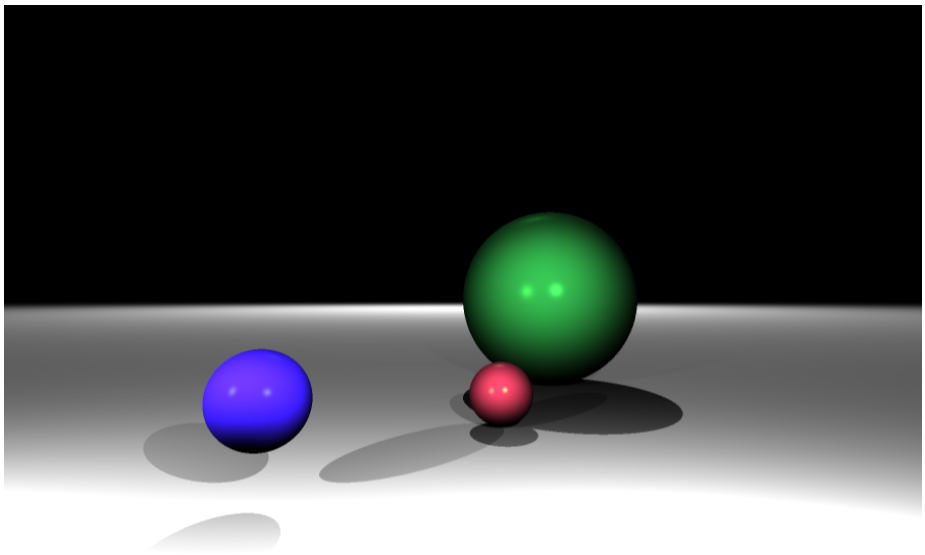

Figure 2: Direct illumination of shperes above an infinite plane.

2 The Infinite Plane Class

2.1 Context

So far we have only worked with a single geometric shape: the sphere. However, to make more realistic scenes, we need to be able to render other shapes rather than spheres. To this end, we will equip RTIS V2 with a new shape called an infinite plane.

Recall that, as seen in lecture 4, a **plane** is fully defined by a **point** $\mathbf{p}_0$ and a **normal n** which is perpendicular to the plane. Also recall that the **ray-plane intersection** method seen in lecture 4 is performed in **world coordinates**. For this reason, there is no local frame associated with the plane shape, and:

- the Shape class members *objectToWorld* and *worldToObject* are both set to the **identity transform**;
- the point $\mathbf{p}_0$ and the normal **n** defining the plane are stored in **world coordinates**;
- there is **no need to transform the coordinate space of the ray** when performing the ray-plane intersection, given that everything is expressed in world coordinates.

2.2 Your Task

Go to the site of the course in **Aula Global** and **download** the files infiniteplane.h and infiniteplane.cpp. After reading the interface of the class *InfinitePlane*, implement the methods *rayIntersect()* and *rayIntersectP()* of the same class.

Once the implementation is terminated, test your brand new *InfinitePlane* class in the following way:

1. Place an **infinite plane under the spheres** of the scene constructed by the function *buildSceneSphere()*. The result should look roughly similar to Fig. 2. Move the positions of the light sources and verify that the shadows are correctly cast.

2. Download the function buildSceneCornellBox() from Aula Global and use it to synthesize an image similar to that of Fig. 3. Given that you do not have yet an implementation of the *Mirror* and *Transmissive* materials you will get a compilation error. To solve it you can provisionally substitute the lines:

```
Material *transmissive = new Transmissive(1.1, Vector3D(1));
Material *mirror       = new Mirror(Vector3D(1, 0.9, 0.85));
```

by the lines:

```
Material *transmissive = new Phong( Vector3D(1, 1, 0.2),
                                    Vector3D(1, 1, 0.2),
                                    20 );
Material *mirror       = new Phong( Vector3D(0.0, 0.9, 0.9),
                                    Vector3D(0.1, 0.9, 0.9),
                                    50 );
```

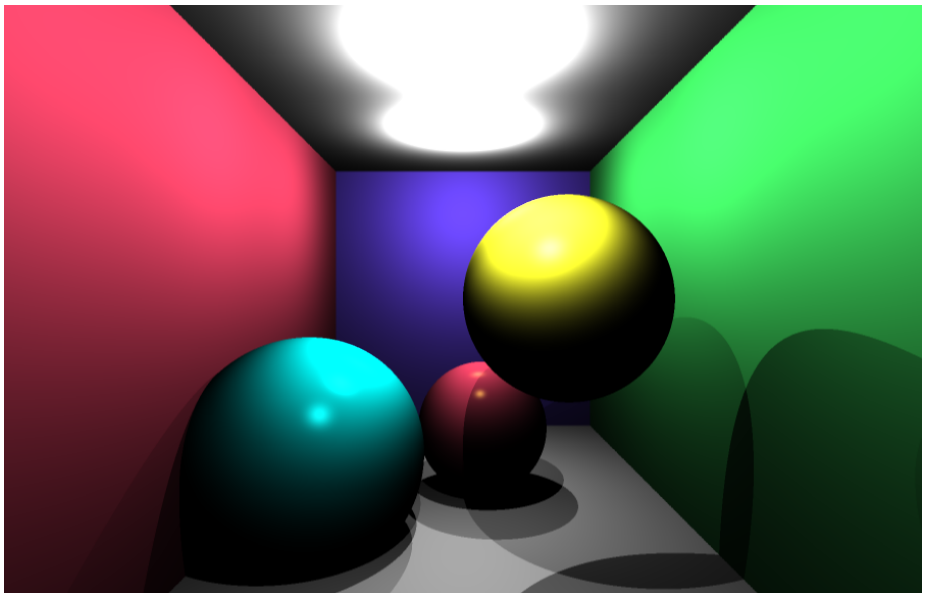

Figure 3: Cornell Box with diffuse and glossy materials only.

3 Enhancing the Direct Shader

3.1 Context

The classes *Mirror* and *Transmissive* implement two types of materials seen in lecture 4: perfect specular materials (e.g., a mirror, which *reflects* the incident light in the perfect reflection direction), and transmissive materials (e.g., glass, which *refracts* and/or *reflects* the incident light according to the Snell's law). Such materials, allow us to generate images with a richer set of lighting effects such as the one shown in Fig. 1, where diffuse, glossy and perfect specular reflections can be appreciated, as well as perfect specular transmissions.

3.2 Your Task 1 (Compulsory)

1. Attribute the mirror material to two of the spheres of the Cornell Box scene.

2. Change the implementation of the direct shader so that it can deal with perfect specular reflections. To this end, make use of the following functions:

   - *hasSpecular()*: function of type boolean. Returns true if the material has perfect specular reflections, and false otherwise
   - *hasDiffuseOrGlossy()*: function of type boolean. Returns true if the material has glossy or diffuse reflections at the surface, and false otherwise.

3.3 Your Task 2 (Optional)

1. Attribute the transmissive material to one of the spheres of the Cornell Box scene.

2. Change the implementation of the direct shader so that it deals with perfect specular transmissions. To this end, make use of the following function:

   - *hasTransmission()*: function of type boolean. Returns true if the material has perfect specular transmission, and false otherwise.

3.4 Your Task 3 (Optional)

1. Implement the ray-triangle intersection.

2. Place some triangles in the Cornell Box scene.