Prof. G. Zachmann
R. Fischer
A. Mühlenbrock

University of Bremen
School of Computer Science
CGVR Group
April 21, 2022

Summer Semester 2022

Assignment on Advanced Computer Graphics - Sheet 1

Due Date 11.05.2022

Exercise 1 (Raytracing Basics, *10 Credits*)

In this exercise, you should complete our raytracing framework. To do that, you have to implement some functions in this framework. The framework is available for download on our website.

First, please take some time to become familiar with the structure of the framework. The hints in the tutorial class, but also the comments in the code should help you. Basically, the framework consists of four essential components:

- **GUI:** The GUI is based on Qt. In order to compile the framework you will need a version of the Qt-framework to be installed on your PC, i.e., Qt 5 (https://download.qt.io/archive/qt/5.15/). In the window on the right side, the GUI shows the resulting image that is generated via raytracing. The window on the left side shows an OpenGL rendering of the scene. You can use it for debugging. However, OpenGL and Qt are not necessary for the understanding of the raytracer; therefore, you should simply ignore this part of the code!
- **XML-Parser:** The program reads 3D scenes in a simple XML format. We use a freely available XML library for parsing. You will find a pre-compiled version for windows in the zip-file. If you use Linux or Apple, you have to compile the library from the sources. You can find a Makefile in the xmlParser sub-folder.

  The XML file format should be almost self-explanatory. However, you don't have to create scenes by your own, you can simply use the pre-defined example scenes (objects.xml, glass-spheres.xml, metal-spheres.xml or steinbach.xml).
- **Mathematical Helper Classes:** You will find some simple template-classes to simplify mathematical computations:
  - `VectorT`: template for n-dimensional vector computations.
  - `MatrixT`: template for quadratic n×n matrices.
  - `Matrix33T`: specialisation of `MatrixT` for 3×3 matrices.
  - `ColorT`: template for RGB color computations.
- **Raytracer:** The basic components of the raytracer are concentrated in the Raytracer class:
  - `Raytracer::render()`: generates a ray for each pixel.
  - `Raytracer::traceRay()`: traces the ray through the scene.
  - `Raytracer::shade()`: This function should compute the local (Phong) shading for a point in the scene. Moreover, it should test whether or not the point is in the shadow.

  The PinholeCamera class implements a simple pinhole camera. The most important function is `PinholeCamera::generateRay()`. It generates a ray from the eye point through the pixel `(x,y)` in the image plane.

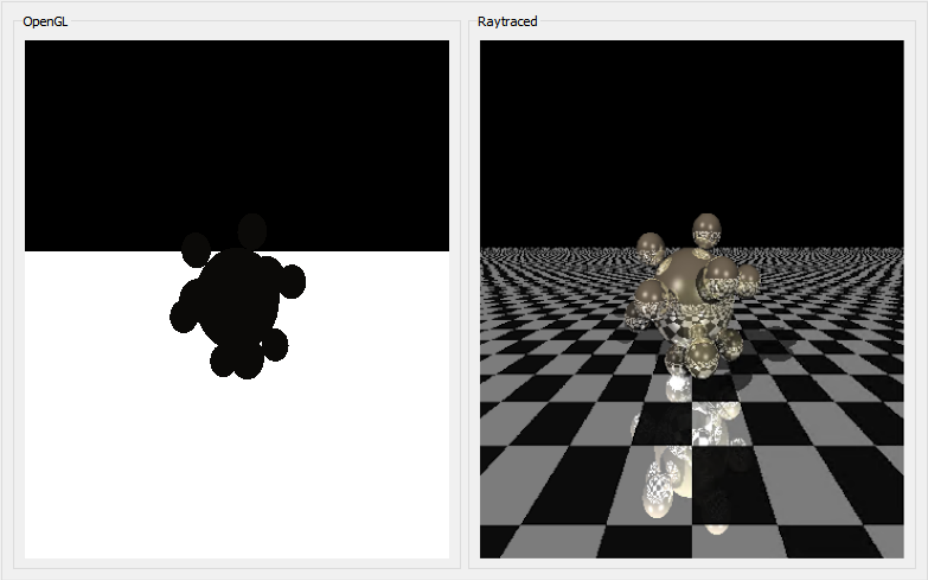

Figure 1: Raytracing: metal-spheres.xml

The `Ray` class represents such a ray. Basically, rays are defined by their starting point and direction. Moreover, the class includes functions to compute the reflected and refracted rays.

All classes for entities in the scene (geometry, materials, light sources) are derived from their respective virtual base classes:

- `Surface`: virtual base class for geometric objects. All derived classes implement their own `intersect()` function that computes the intersection between the objects and a ray. The derived classes `Plane`, `Sphere` and `Checkerboard` are fully implemented.
- `Shader`: virtual base class for materials. The derived class `PhongShader` is fully implemented. The function `PhongShader::shade()` computes the local Phong model.
- `Light`: Virtual base class for light sources. The derived classes `PointLight` and `DirectionalLight` are fully implemented.

Your tasks:

a) Implement the function `Raytracer::shade()` in the file `Raytracer.cpp`. In this function, you should test whether or not the pixel is in the shadow. If the pixel is not in the shadow, you have to set its color to the appropriate (Phong) lighting.

b) In order to realize a real raytracer, we have to trace also reflected rays. To do that, you should implement the function `Ray::reflectedRay()` in `Ray.cpp`.

Moreover, you have to use this function in `Raytracer::traceRay()` to recursively compute the reflected color (that has to be added to the current color value).

c) Finally, we also want to include refractions into our raytracer. Therefore, you have to implement the function `Ray::refractedRay()` in `Ray.cpp`. It should, depending on the refraction parameter of the material, return the refracted ray.

As in the previous part, you also have to trace the refracted ray recursively through the scene in `Raytracer::traceRay()` to compute the appropriate amount of color.

Exercise 2 (Meta-Ball Raytracing, *10 Credits*)

In this assignment you are going to implement the intersection test for Metaballs in our raytracer. To do that, extend the function `Metaball::intersect()` in `Metaball.cpp` with the intersection test. As a potential-function you should use the exponential function shown in the lecture:

$$p(r) = e^{-br^2}$$

For the blending use the sum defined on the slide in the lecture:

$$F(\mathbf{x}) = \sum_{i=1}^{n} a_i \; p(\| \mathbf{x} - \mathbf{x}_i \|)$$

The following equation therefore defines the isosurface:

$$K = \{\mathbf{x} \mid F(\mathbf{x}) - \tau = 0\}$$

The assignment is comprised from the following sub-tasks:

a) Implement the potential function and sum thereof in `Metaball::F()`. All the points comprising the metaball are stored in `Metaball::m_balls`. They contain their bandwidth b_i, weight a_i and position x_i. The isovalue τ is stored in `Metaball::m_isovalue`.

b) Implement the intersection test with the metaball in `Metaball::intersect()` and compute the normal at the intersection point with the method from the lecture slides. This can be done with brute-force sampling along the ray.

Hints:

- For guidance how to implement an intersection test have a look into `Sphere.cpp`.
- Start with the scene in `meta-balls-simple.xml`. It only contains a single metaball which allows you to use the calculation for the normal of a simple sphere as a starting point.

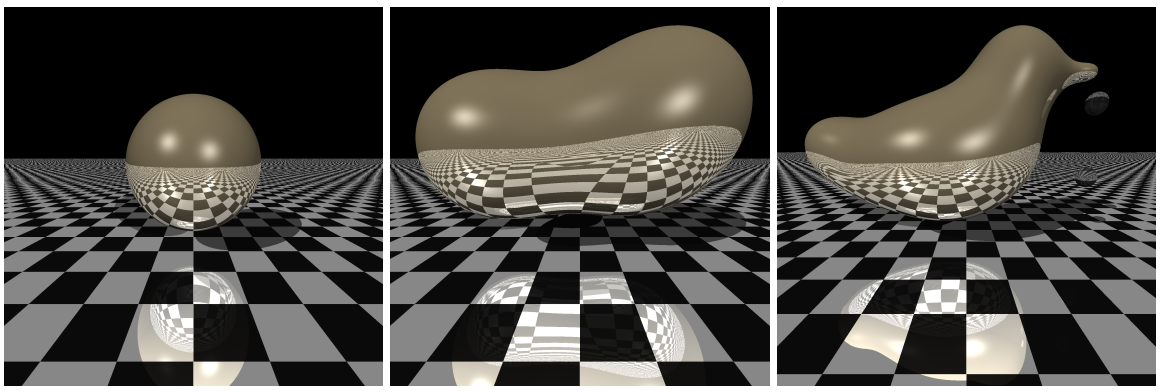

Figure 2: This is how the metaball scenes should look like. From left to right: `meta-balls-simple.xml`, `meta-balls.xml` and `meta-balls-complex.xml`.