Advanced Topics in Computer Graphics I

Universität Bonn
Institut für Informatik II
April 28, 2016

Summer term 2016
Prof. Dr. Reinhard Klein
Alexander Dieckmann, Oliver Burghard, Sebastian Merzbach

Sheet R02 - Turner-Whitted-Raytracing

Solutions for the theoretical part via e-mail[1] or in written form by Tue, 03.05.2016, **2 p.m.** Points for the practical part will only be accredited after demonstration (whole group presence required) and discussion in the exercise on Wed, 04.05.2016 in the VR-Lab pool.

Assignment 1) Turner-Whitted-Raytracer *(8Pts)*

In this exercise, a raytracer based on the Turner-Whitted algorithm has to be implemented. A class framework can be found on the web page which will be the foundation for the remainder of the practical exercises of this lecture. Please read through this assignment sheet carefully and complete the framework to construct a working raytracer.

Please take your time to make yourself familiar with the provided framework. Besides the class diagram in Figure 1 and the comments on this sheet, detailed comments in the source code itself should help you to understand the individual classes and their methods.

The raytracer has to render a predefined scene and show the result in a Qt window. The result image can be saved to disc via a context menu command. Your task is to implement necessary functionality in some of the classes.

For this, the framework contains the following classes and methods:

- **GUI** The class `CAppWidget` implements the GUI using the Qt framework. Moreover, it constructs the scene. The main functionality is provided via a context menu. Knowledge of this class and the usage of the Qt framework is not necessary to understand the raytracer.
- **Math classes**
  - `CVectorT` Template for n-dimensional vector arithmetics
  - `CMatrixT` Template for quadratic nxn-matrices
  - `CMatrix33T` Specialization of `CMatrixT` for 3x3-matrices
  - `CColorT` Template for RGB-color arithmetics
- **Raytracer** The raytracer is based on virtual base classes for the scene definition (objects, materials, light sources) and contains classes for rays (`CRay`) and for the camera definition (`CPinholeCamera`). The necessary basic data types are defined in `types.h`.
  - `CSurface` Virtual base class for objects that can be intersected with a ray. For the derived classes `CPlane` (a plane defined by a point and a normal) and `CSurfaceList` (manages a list of objects) a full implementation is already provided. For the class `CSphere` the method `Intersect()` has to be implemented by you. We suggest to complete the theoretical exercise first.
  - `CShader` Virtual base class for materials. Materials will be discussed later in the lecture and thus, three simple classes have already been implemented for you.
  - `CLight` Virtual base class for light sources. The derived class `CDirectionalLight` is already implemented.

[1] Please send emails to *all* tutors and with subject: '[atcg] submission'.

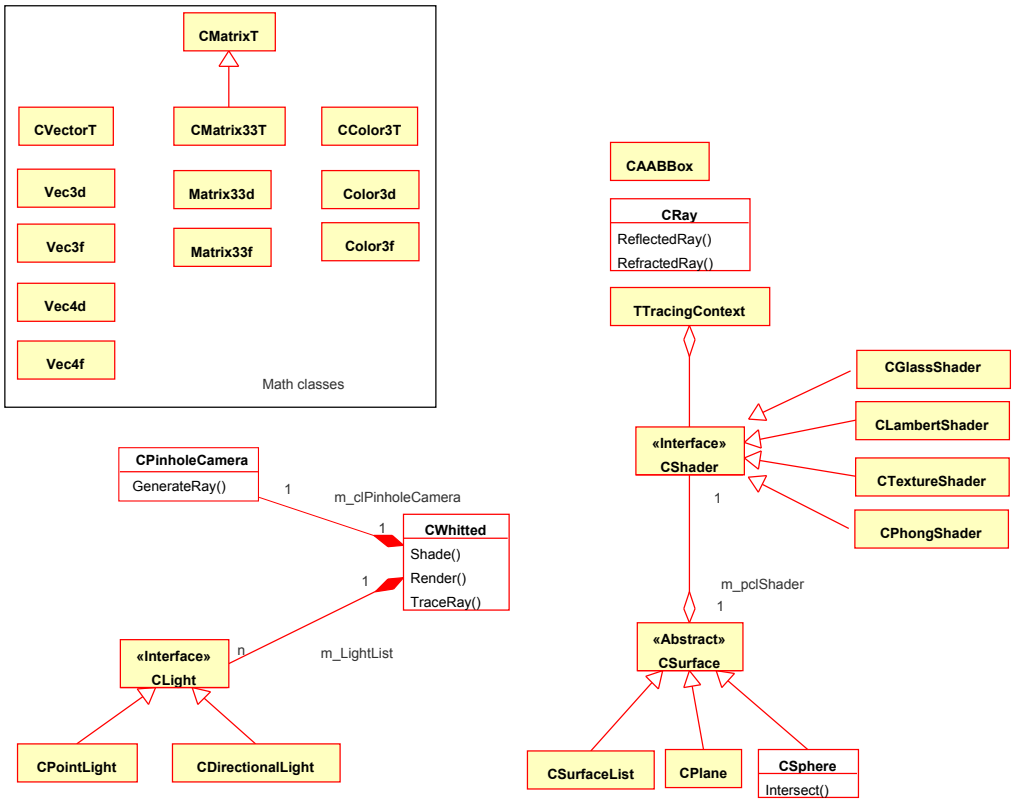

Figure 1: An overview of the classes of the raytracer. The shown methods have to be implemented.

- `CRay` Represents a ray by its origin and direction. You still have to implement the methods to compute reflected and refracted rays.
- `CPinholeCamera` A class that represents a simple pinhole camera. You have to implement the method that generates a ray through the eye point and a pixel (x,y).
- `CWhitted` Implements the raytracing algorithm itself. You have to fill the following methods:
  - `Render()` Renders the scene into a given QImage.
  - `TraceRay()` Implements the recursive raytracing itself. The ray has to be traced until the given maximum trace depth is reached and the final pixel color has to be calculated.
  - `Shade()` Evaluates the local lighting at a surface point. Relevant light sources are determined by tracing shadow rays to the light sources.

If you finished all tasks successfully your image should look like the one in Figure 2.

General comments:

- The methods you have to implement have detailed comments on necessary arguments and details.
- Make yourself familiar with the structure `TTracingContext` in `types.h`. It contains the most important arguments for the algorithm to reduce the size of the argument lists.
- Start by implementing a simple raycasting algorithm without shadows and recursion first. This way you can test your implementations of `CSphere` and `CPinholeCamera` as well as the main loop. Afterwards, complete the raytracer by adding shadow rays and recursion for reflection and refraction.

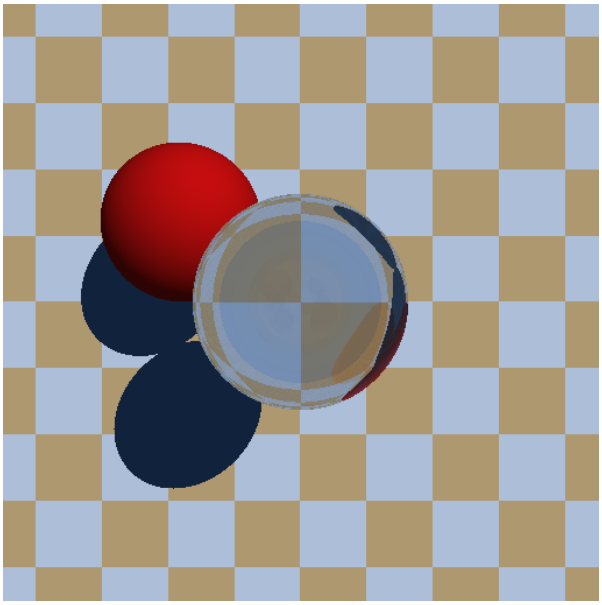

Figure 2: Reference image

Theoretical Assignment

Assignment 2) Ray-Sphere intersecion *(2Pts)*

Derive the formula for ray-sphere intersection formally. The ray-sphere intersection is used in the practical part of this exercise.

Assignment 3) Reflection Equation *(2.5 + 1 bonusPts)*

Consider the simple scene from last sheet as it is shown in Figure 3. Again, we're given the radiance of the sun $L_s = 20.045 \frac{MW}{m^2 sr}$ and the direction v having an angle of 45° to the table. You can again assume that the radiance is constant over the whole surface of the sun.

Rückstrahlvermögen

Now, assume that the table plate is perfectly diffuse with 50% albedo. A camera having a lens of 5cm diameter has been placed 1.2m above the center of the table looking exactly downwards. Calculate the irradiance and the radiant power at the camera lens assuming that everything else around the table is perfectly black. Furthermore, you can for simplicity assume that the table covers the same solid angle from all points of the lens and that the irradiance is constant over the camera lens. Use Maple (installed in the CIP pool) for solving necessary integrals.

For a rough approximation you will get 2.5 points and for a complete solution you will get an additional 1.5 bonus points.

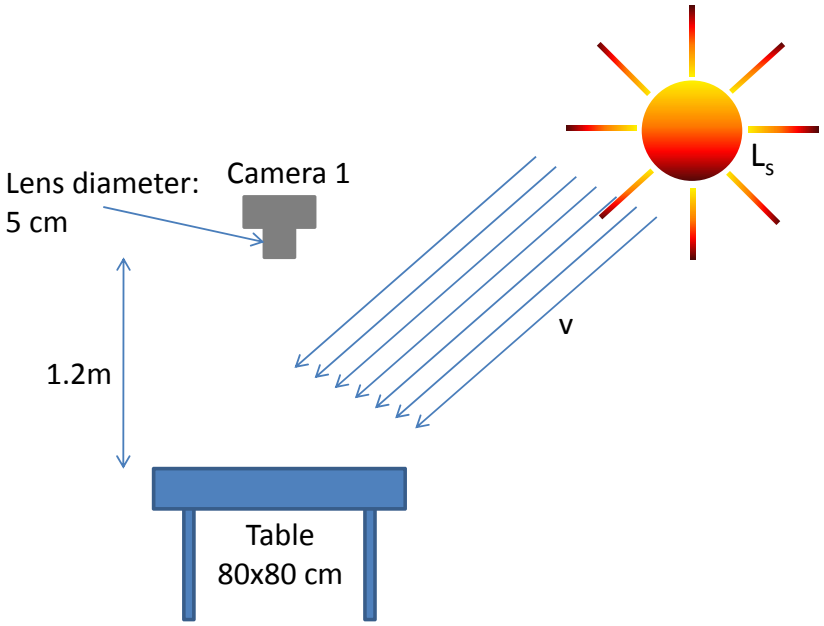

Figure 3: Toy example for radiometric quantities

Good luck !