Lab Project: Computer Graphics
Building a basic ray tracer
SS2017

Haralambi TODOROV

August 4, 2017

Advisor: Prof. Dr.-Ing. Matthias Teschner

1 Introduction

The main objective of the *Lab project: Computer Graphics* was to make the author familiar with the basic concepts of a ray tracer by implementing one. This lab report aims to present the gathered knowledge.

1.1 Report overview

The report is split in five chapters with each chapter explaining a certain part of the implemented ray tracer. The ordering of the chapters try to follow the implementation order the author followed during the project.

Current chapter (1) is an introductory one giving details about the organisation of this report, motivation about why is ray tracing an important rendering technique and where does it find application nowadays along with some information on where do the roots of ray tracing come from and the basic idea of the ray tracing algorithm.

Chapter (2) deals with one of the fundamental parts of a ray tracer, the ray-object intersection test. It starts with the geometric definition of a half-line (ray), goes on with the different type of geometries the ray tracer supports (spheres, triangles, axis-aligned boxes and triangulated meshes) and how the half-line intersects with these geometries.

Chapter (3) is concerned with how the camera and the image plane are modelled in the ray tracer. It explains the virtual pinhole camera model, how the image plane is mapped within the 3D scene and the two supported camera projections - orthographic and perspective along with their configurable parameters. The chapter goes on with how are rays shot from the image plane and how can one reduce aliasing artefacts by

using half-jittered sampling technique. It finishes with the use of gamma correction to enhance the visual appearance of the generated images.

Chapter (4) deals with visual appearence of objects rendered in a scene. It begins by explaining how light sources are modeled, goes on with the concept behind the Phong illumination model and the physical motivation behind its components, how shadows are ray traced and lastly it presents reflective and refracitve materials and how these type of materials are ray traced.

Chapter (5) deals with transformations, explaining the motivation behind the use of homogeneous notation, the different type of supported transformations on objects, light sources and cameras and why is the inverse view transformation useful in a ray tracer.

Chapter (6) is concerned with how one can reduce the computation time per generated image by introducing acceleration structures. Firstly explaining the concept behind axis-aligned bounding boxes and the benefits they bring, followed by the a more advanced acceleration method, the uniform grid.

1.2 Why is ray tracing important

Ray tracing is one of the rendering techniques capable of producing images with a high degree of visual realism and which can also naturally incorporate physically motivated visual effects such as reflections, refractions, caustics, soft shadows and others. This advantage makes the technique very attractive to movie and commercial studios, automotive industry (see figure 1a) as well as architectural design studios (see figure 1b) to simulate realistic illumination.

(a) ”Another R8” by Filip Sadlon rendered using Blender Render

(b) ”Mies Van Der Rohe Farnsworth House” by Alessandro Prodan using Mental Ray

Figure 1: Ray tracing used for visualisations by different industries

Although one can produce very stunning imagery with a ray tracing based render engine, this comes at a great computation cost, e.g. a frame from a recent computer generated Pixar’s movie takes between three and eight hours to render. [Goo14]
Though great computation times, major animation studios seems nowadays to be fond of ray tracing and make switches from scanline-based rendering approaches such as

"REYES" which were proved stable and fast over the years to ray tracing. [Pix17] That points to the demands in the entertainment industry for more physically accurate imagery, but also pushes the boundaries of research in ray tracing and its computational efficiency. [ENSB13]

1.3 The roots of ray tracing

The first ray tracing algorithm was introduced by Arthur Appel in 1968 [App68], which idea was to shoot rays from the eye (camera), one per pixel, and find the closest object blocking the path of that ray. Using the material properties and the effect of the lights in the scene, this algorithm can determine the shading of objects.
The next notable contribution in ray tracing was made by Turner Whitted in 1979 [Whi79], who introduced a technique to compute shadows as well as the idea of recursive ray tracing to handle reflective and refractive materials. (see figure 2).
Other major contributions in the scene of ray tracing were made by Robert Cook in 1984 [CPC84] and James Kajiya in 1986 [Kaj86] introducing distributed ray tracing and the Rendering equation respectively. Because the accompanying ray tracer does not make extensive use of these concepts, they won't be discussed in detail.

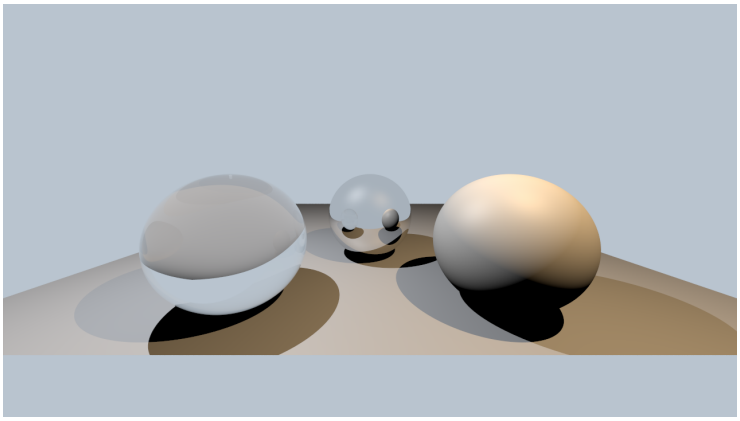

Figure 2: A Whitted-like scene with reflective (back) and refractive (front left) spheres rendered in the accompanying ray tracer

1.4 Basics of the ray tracing algorithm

For the following explanation we make the assumption we have a camera with perspective projection. The algorithm then works by tracing a path from an imaginary eye (camera) through each pixel in a virtual image plane (in front of the camera) and calculating the radiance of the object visible through it.
Typically, each ray must be tested for intersection with some subset of all the objects in the scene. Once the nearest object has been identified, the algorithm will estimate the incoming light at the point of intersection, examine the material properties of the

object, and combine this information to calculate the final color of the pixel (see figure 3).

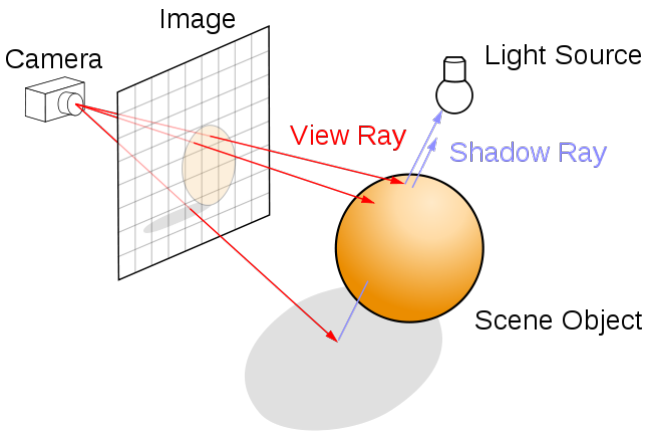

Figure 3: Image depicting the concept behind the ray tracing algorithm. Wikipedia

2 Ray-object intersection tests

To explore ray-object intersections, one have to define first what is an intersection. An intersection is a point(s) along a half-line (from now on I'll use the word *ray* meaning half-line) on which the given ray intersects a certain object. Mostly one is interested in the nearest intersection point along a ray.
To find an intersection points between a ray and an object, one needs two definitions, one for the ray and one for the object which the ray intersects with. First comes the definition for a ray, since a ray is needed for all type of ray-object intersections.

2.1 Ray

A ray is geometrically defined using two vectors: a point o, which is called the *origin* and a normal vector $\hat{d}$, which is gives the *direction* of this ray. To be able to give the location of an intersection point p along this ray, one introduces a scalar value t, which is the distance between the ray's origin o and the intersection point p we're looking for. One is mainly interested in intersection points "in front" of a ray, so the values for t should be positive. The equation for the intersection point we'll be using from now on looks like: [SH14]

$$p(t) = o + t\hat{d} \tag{1}$$

Within the implemented ray tracer a ray has five data members: origin point, direction, inverse of the the direction $(1/\hat{d})$, sign of the ray's direction and a type component.

The author has decided not to have an intersection point data member stored within the ray for memory efficiency. Depending on the scene a big part of the rays could not have any intersection points at all.
The inverse of the ray's direction and sign of the ray's direction data members are used to optimize the performance of the ray-axis-aligned bounding box intersection test. Details will be provided on the following section on the topic.
The type component is used for distinguishing between different type of rays and printing out a statistic for a given rendered scene. One distinguishes between following ray types in the implemented ray tracer: primary, shadow, reflection and refraction rays.

3 Cameras and the image plane

4 Shading

5 Transformations

6 Acceleration structures

7 Answers to Definitions

References

[App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*, pages 37–45, 1968.

[CPC84] Robert L. Cook, Thomas K. Porter, and Loren C. Carpenter. Distributed ray tracing. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1984, Minneapolis, Minnesota, USA, July 23-27, 1984*, pages 137–145, 1984.

[ENSB13] Christian Eisenacher, Gregory Nichols, Andrew Selle, and Brent Burley. Sorted deferred shading for production path tracing. In *Proceedings of the Eurographics Symposium on Rendering*, EGSR '13, pages 125–132, Aire-la-Ville, Switzerland, Switzerland, 2013. Eurographics Association.

[Goo14] Craig Good. How long does it take to render a Pixar film?, 2014.

[Kaj86] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1986, Dallas, Texas, USA, August 18-22, 1986*, pages 143–150, 1986.

[Pix17] Pixar. About RIS, 2017.

[SH14] Kevin Suffern and Helen H. Hu. *Ray Tracing from the Ground Up*. A. K. Peters, Ltd., Natick, MA, USA, 2nd edition, 2014.

[Whi79] Turner Whitted. An improved illumination model for shaded display. In *Proceedings of the 6th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1979, Chicago, Illinois, USA, August 8-10, 1979*, page 14, 1979.