Week 1

Ray tracing is the easiest and probably the most general technique for visualising 3D models. It is easy to create many sophisticated lighting effects using ray tracing, but ray tracing is still slower than rasterization. Hence ray tracing is used primarily for rendering of photo realistic images in applications where image quality rather than a high frame rate is the primary concern. The purpose of the first three exercises is to help you learn how a ray tracer works. A ray tracing framework is available on CampusNet which provides a coding infrastructure such that you only need to implement the essential parts.

Learning Objectives

- Implement ray casting (tracing rays from eye to first surface intersection).
- Use the pinhole camera model for generating rays in a digital scene.
- Compute intersection of rays with three-dimensional primitive objects (planes, triangles, spheres).
- Compute shading of diffuse surfaces using Kepler's inverse square law and Lambert's cosine law.

Getting Started

First of all, you need to get the framework up and running. And you need to familiarise yourself with the math library and material file format used in the framework. Go through the following steps to get started.

- Compile the framework and take a screenshot of the rasterized preview of the default scene. To compile the `raytrace` project, you must provide a path to the include directory of the OptiX installation on your local machine.[1]
- Familiarise yourself with the OptiX math library. The framework uses this library, which follows the CUDA[2] and Cg[3] syntax, to enable a smooth transition to GPU ray tracing. The documentation of the library is unfortunately still rather fragmented. The documentation available from NVIDIA is compiled in the note `OptiX_MathRef.pdf` on CampusNet.
- Find out how the default scene is defined by investigating the file `RenderEngine.cpp`. Note that material properties are loaded from a Wavefront MTL file.[4] Familiarise yourself with this file format. In addition, note that many rendering parameters are set in the `RenderEngine` class constructor and that the comments in the function `keyboard` document the user interface of the program.

Ray Casting

In this first exercise, your job is to implement the very basics of ray tracing which are ray generation, ray-object intersection, and shading of diffuse surfaces. This simple, non-recursive, visible-surface ray tracing is typically referred to as ray casting.

- Generate rays using a pinhole camera model.[5] Do this by completing the camera implementation in the file `Camera.h` of the `raytrace` project. You need to implement the following three functions: `set`, `get_ray_dir`, and `get_ray`.

[1] Right click the project name in the Solution Explorer window and choose Properties. Go to Configuration Properties → C/C++ → Additional Include Directories and insert the path.

[2] http://www.nvidia.com/cuda/

[3] http://developer.nvidia.com/Cg and http://developer.nvidia.com/CgTutorial

[4] http://paulbourke.net/dataformats/mtl/

[5] See the short lecture note of nearly the same title for more details. It is available on CampusNet.

- Implement ray-plane and ray-triangle intersection.[6] Do this by implementing the function `intersect` in the files `Plane.cpp` and `Triangle.cpp`, and optionally (but recommended) the function `intersect_triangle` in `Triangle.cpp`.
- Implement conditional acceptance of an intersection. For efficiency, we both need a function that finds the closest intersection and one that checks for any intersection. Planes are handled separately. Implement the following functions: `closest_hit`, `any_hit`, `closest_plane`, and `any_plane` in the file `Accelerator.cpp`.
- Implement the loop that computes all pixels. See the pseudocode (fig. 15.56) in the note about visible-surface ray tracing. Do this by implementing the function `compute_pixel` in the file `RayCaster.cpp` and the function `render` in the file `RenderEngine.cpp`. Press 'r' on the keyboard to check that the plane, the triangle, and the blue background colour seen in the preview also appear when you ray trace the default scene.
- Implement ray-sphere intersection (see Section 1.4.2 of **B**). Do this by implementing the function `intersect` in the file `Sphere.cpp`. Check that the sphere in the preview also appears in the ray traced version. Press 'b' on the keyboard to store the render result in the file `out.png`.
- Shade the diffuse surfaces according to the point light in the scene. Do this by implementing the function `sample` in the file `PointLight.cpp` and the function `shade` in the file `Lambertian.cpp`. Use Kepler's inverse square law of radiation and Lambert's cosine law. You only need to trace shadow rays if you have time. This will be part of next week's exercises. Store the render result.

Week 1 Deliverables

Renderings of the default scene (e.g. preview, flat reflectance shading, and shading of diffuse objects). Compare preview and ray casting using the default flat reflectance shading to ensure that ray generation and ray-object intersection work as expected. Include relevant code and render log (render time and likewise). Please copy everything into a document and upload at CampusNet under Assignments. Consider these weekly deliverables to be your lab journal.

Reading Material

The curriculum for Week 1 is (37 pages)

B Sections 1.3–1.4.4. *Vectors in computer graphics* and *Rays in computer graphics*.

B Section 5.1.3. *Camera or eye or view coordinate system*.

N1 Section 15.10. *Visible-surface ray tracing*.
Foley, J. D., van Dam, A., Feiner, S. K., and Hughes, J. F. *Computer Graphics: Principles and Practice*, second edition in C, Addison-Wesley, 1996. Reprinted with corrections, July 1997.

- Frisvad, J. R. *Ray Generation Using a Pinhole Camera Model*. Lecture Note, Technical University of Denmark, July 2011.
- Frisvad, J. R. *Ray-Triangle Intersection*. Lecture Note, Technical University of Denmark, July 2011.

Supplementary reading material:

- Glassner, A. S. An Overview of Ray Tracing. In *An Introduction to Ray Tracing*, Chapter 1, Morgan Kaufmann, 1989.

JRF 2011

[6] See the short lecture note with the title "Ray-Triangle Intersection". It is available on CampusNet.