Middle East Technical University | Department of Computer Engineering

CENG 477

Introduction to Computer Graphics

Fall '2018-2019
Assignment 1 - Ray Tracing

Due date: November 4, 2018, Sunday, 23:59

1 Objectives

Ray tracing is a fundamental rendering algorithm. It is commonly used for animations and architectural simulations, in which the quality of the created images is more important than the time it takes to create them.

In this assignment, you are going to implement a basic ray tracer that simulates the propagation of light in the real world.

Keywords: *ray tracing, light propagation, geometric optics, ray-object intersections, surface shading*

2 Specifications

1. You should name your executable as "raytracer".
2. Your executable will take an XML scene file as argument (e.g. "scene.xml"). A parser will be given to you, so that you do not have to worry about parsing the file yourself. The format of the file will be explained in the next section. You should be able to run your executable via command "./raytracer scene.xml".
3. You will save the resulting images in the PPM format. A PPM writer will be given to you so you don't have to worry about writing this file yourself. Interested readers can find the details of the PPM format at: `http://netpbm.sourceforge.net/doc/ppm.html`
4. The scene file may contain multiple camera configurations. You should render as many images as the number of cameras. The output filenames for each camera is also specified in the XML file.
5. You will have at most 15 minutes to render scenes for each input file on inek machines. Programs exceeding this limit will be killed and will be assumed to have produced no image.

6. You should use Blinn-Phong shading model for the specular shading computations.

7. You will implement two types of light sources: point and ambient. There may be multiple point light sources and a single ambient light. The intensity values of these lights will be given as (R, G, B) triplets that are not restricted to the [0, 255] range (however, they cannot be negative as negative light does not make sense). Any pixel color value that is calculated by shading computations and is greater than 255 must be clamped to 255 and rounded to the nearest integer before writing it to the output PPM file.

8. Point lights will be defined by their intensity (power per unit solid angle). The irradiance due to such a light source falls off as inversely proportional to the squared distance from the light source. To simulate this effect, you must compute the irradiance at a distance of d from a point light as:

$$E(d) = \frac{I}{d^2}$$

where I is the original light intensity (a triplet of RGB values given in the XML file) and $E(d)$ is the irradiance at a distance of d from the light source.

3 Scene File

The scene file will be formatted as an XML file (see Section 7). In this file, there may be different numbers of materials, vertices, triangles, spheres, lights, and cameras. Each of these are defined by a unique integer id. The ids for each type of element will start from one and increase sequentially. Also notice that, in the XML file:

· Every number represented by X, Y and Z is a floating point number.

· Every number represented by R, G, B, and N is an integer.

Explanations for each XML tag are provided below:

- **BackgroundColor:** Specifies the R, G, B values of the background. If a ray sent through a pixel does not hit any object, the pixel will be set to this color.

- **ShadowRayEpsilon:** When a ray hits an object, you are going to send a shadow ray from the intersection point to each point light source to decide whether the hit point is in shadow or not. Due to floating point precision errors, sometimes the shadow ray hits the same object even if it should not. Therefore, you must use this small ShadowRayEpsilon value, which is a floating point number, to move the intersection point a bit further so that the shadow ray does not intersect the same object again. Note that ShadowRayEpsilon value can also be used to avoid self-interactions while casting reflection rays from the interaction point.

- **MaxRecursionDepth:** Specifies how many bounces the ray makes off of mirror-like objects. Applicable only when a material has nonzero MirrorReflectance value. Primary rays are assumed to start with zero bounce count.

- **Camera:**

  ·**Position** parameters define the coordinates of the camera.
  ·**Gaze** parameters define the direction that the camera is looking at. You must assume that the Gaze vector of the camera is always perpendicular to the image plane.
  ·**Up** parameters define the up vector of the camera.
  ·**NearPlane** attribute defines the coordinates of the image plane with Left, Right, Bottom, Top floating point parameters, respectively.
  ·**NearDistance** defines the distance of the image plane to the camera.
  ·**ImageResolution** defines the resolution of the image with Width and Height integer parameters, respectively.
  ·**ImageName** defines the name of the output file.

  Cameras defined in this homework will be right-handed. The mapping of Up and Gaze vectors to the camera terminology used in the course slides is given as:

$$\begin{aligned} Up &= v \\ Gaze &= -w \\ u &= v \times w \end{aligned}$$

- **AmbientLight:** is defined by just an X, Y, Z intensity triplet. This is the amount of light received by each object even when the object is in shadow.

- **PointLight:** is defined by a position and an intensity, which are all floating point numbers.

- **Material:** A material can be defined with ambient, diffuse, specular, and mirror reflectance properties for each color channel. Values are floats between 0.0 and 1.0.
  PhongExponent defines the specularity exponent in Blinn-Phong shading.
  MirrorReflectance represents the degree of the mirrorness of the material. If this value is not zero, you must cast new rays and scale the resulting color value with the MirrorReflectance parameters.

- **VertexData:** Each line contains a vertex whose x, y, and z coordinates are given as floating point values, respectively.

- **Mesh:** Each mesh is composed of several faces. A face is actually a triangle which contains three vertices.
  When defining a mesh, each line in Faces attribute defines a triangle. So, each line is represented by three integer vertex id given in counter-clockwise order (see Triangle explanation below). Material attribute represents the material ID of the mesh.

- **Triangle:** A triangle is represented by Material and Indices attributes. Material attribute represents the material ID. Indices are the integer vertex IDs of the vertices that construct the triangle. Vertices are given in counter-clockwise order, which is important when you want to calculate the normals of the triangles. Counter-clockwise order means that if you close your right-hand following the order of the vertices, your thumb points in the direction of the surface normal.

- **Sphere:** A sphere is represented by Material, Center, and Radius attributes. Material attribute represents the material ID. Center represents the vertex ID of the point which is the center of the sphere. Radius attribute is the radius of the sphere.

4 Hints & Tips

1. Start early. It takes time to get a ray tracer up and running.
2. You must provide a makefile to compile your program. This file must use g++ as the compiler. You are free to choose your compile options. Failing to provide a working makefile will have a penalty of 5 points out of 100.
3. You may use the -O3 option while compiling your code for optimization. This itself will provide a huge performance improvement.
4. Try to pre-compute anything that would be used multiple times and save these results. For example, you can pre-compute the normals of the triangles and save it in your triangle data structure when you read the input file.
5. If you see generally correct but noisy results (black dots), it is most likely that there is a floating point precision error (you may be checking for exact equality of two FP numbers instead of checking if they are within a small epsilon).
6. For debugging purposes, consider using low resolution images. Also it may be necessary to debug your code by tracing what happens for a single pixel (always simplify the problem when debugging).
7. We will not test your code with strange configurations such as the camera being inside a sphere, a point light being inside a sphere, or with objects in front of the image plane. If you doubt whether a special case in addition to these will be tested, you may ask this in the newsgroup.

5 Bonus

1. You can get 5 points bonus if your ray tracer is among the fastest N ray tracers to be determined by our benchmarks. Number N is to be determined experimentally but you can expect it to be a small number such as 3 or 5. Of course, your outputs should be correct to be eligible for this bonus.
2. You can get 5 points bonus if you create and share interesting XML scene files. These should not be very simple scenes as we already share such scenes with you. Interesting scenes to get bonus will be determined by the course instructors and assistants.

6 Regulations

1. **Programming Language:** C/C++. Only under special cases other PLs may be allowed (for instance being an Erasmus student and not having learned these languages in your program). Ask your instructor if in doubt.
2. **Late Submission:** You can submit your codes up to 3 days late. Each late day will be deduced from the total 7 credits for the semester. However, if you fail to submit even after 3 days, you will get 0 regardless of how many late credits you may have left. If you submit late and still get zero, you cannot claim back your late days. As unfortunately the COW

system does not have an “unsubmit” option, you must e-mail your assistants if you want your submission to not be evaluated (and therefore preserve your late day credits).

3. **Groups:** You can team-up with another student. But you must notify the assistants about who your partner is.

4. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations and will get 0 from the homework. You can discuss algorithmic choices, but sharing code between groups or using third party code is strictly forbidden.
   To prevent cheating in this homework, we also compare your codes with online ray tracers and previous years' student solutions. In case a match is found, this will also be considered as cheating. Even if you take only a “part” of the code from somewhere or somebody else, this is also cheating. Please be aware that there are “very advanced tools” that detect if two codes are similar.

5. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.

6. **Submission:** Submission will be done via COW. Create a "tar.gz" file named "raytracer.tar.gz" that contains all your source code files and a makefile. The executable should be named as "raytracer" and should be able to be run using the command "./raytracer scene_file_name.xml".

7. **Evaluation:** Your codes will be evaluated on Department Inek Machines; based on several input files including, but not limited to the test cases given to you as example. Rendering all scenes correctly within the time limit will get you 100 points.

7 Sample Scene File

```
<Scene>
        <BackgroundColor>R G B</BackgroundColor>
        <ShadowRayEpsilon>X</ShadowRayEpsilon>
        <MaxRecursionDepth>N</MaxRecursionDepth>
        <Cameras>
                <Camera id="Cid">
                        <Position>X Y Z</Position>
                        <Gaze>X Y Z</Gaze>
                        <Up> X Y Z </Up>
                        <NearPlane>Left Right Bottom Top</NearPlane>
                        <NearDistance>X</NearDistance>
                        <ImageResolution>Width Height</ImageResolution>
                        <ImageName>ImageName.ppm</ImageName>
                </Camera>
        </Cameras>
        <Lights>
                <AmbientLight>X Y Z</AmbientLight>
                <PointLight id="Lid">
                        <Position>X Y Z</Position>
                        <Intensity>X Y Z</Intensity>
                </PointLight>
        </Lights>
        <Materials>
                <Material id="Mid">
                        <AmbientReflectance>X Y Z</AmbientReflectance>
                        <DiffuseReflectance>X Y Z</DiffuseReflectance>
                        <SpecularReflectance>X Y Z</SpecularReflectance>
                        <PhongExponent>X</PhongExponent>
                        <MirrorReflectance>X Y Z</MirrorReflectance>
                </Material>
        </Materials>
        <VertexData>
                V1X V1Y V1Z
                V2X V2Y V2Z
                .....
        </VertexData>
        <Objects>
                <Mesh id="Meid">
                        <Material>N</Material>
                        <Faces>
                                F1_V1 F1_V2 F1_V3
                                F2_V1 F2_V2 F2_V3
                                .....
                        </Faces>
                </Mesh>
                <Triangle id="Tid">
                        <Material>N</Material>
                        <Indices>
                                V1 V2 V3
                        </Indices>
                </Triangle>
                <Sphere id="Sid">
                        <Material>N</Material>
                        <Center>N</Center>
                        <Radius>X</Radius>
                </Sphere>
        </Objects>
</Scene>
```