Middle East Technical University | Department of Computer Engineering

CENG 477

Introduction to Computer Graphics

Fall '2016-2017

Assignment 1 - Ray Tracing

Due date: 31 October 2016, Monday, 23:59

1 Objectives

In this assignment, you are going to implement some parts of a ray tracer that simulates the propogation of light in empty space. You do not need to write the whole program, you will be given several files for convention and you should fill them according to the regulations. The scope of the ray tracer will be limited to what you have learned in the class. You must use C/C++ in your implementations. ***Most of the important parts that needs to be considered when doing homework have been written as comments above of the related functions.***

Keywords: *ray tracing, light propagation, geometric optics, ray intersections, shadings*

2 Specifications

1. You should name your executable as "raytracer".

2. Your executable will take two txt files as argument: one that describes the virtual environment and one that describes the cameras as argument (e.g. "sample_scene.txt sample_camera.txt"). The format of these files will be explained in the next section. You should be able to run your executable via command "./raytracer sample_scene.txt sample_camera.txt".

3. The camera description file may contain multiple camera configurations. You should render as many images as the count of cameras. The output filenames for each camera is also specified in the camera input file.

4. 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

5. You will have at most 30 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. Therefore, if your program cannot produce any output at most 30 minutes, your program will get a 0 from that test case.

6. 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. The intensity values may be arbitrarily large positive numbers; i.e. not restricted to the [0, 255] range. As a result of shading computations if you find a color value greater than 255 you must clamp that value to 255 (also you must never find negative values).

7. The intensity value of a point light source falls of as inversely proportional to the distance from the light source. This decline in intensity is called light attenuation. You must attenuate all color channels of the light source as:

$$I_p(d) = \frac{I}{4\pi d^2} \tag{1}$$

where I is the original light intensity (a triplet of (R, G, B) values given in the scene input file) and Ip(d) is the intensity at distance d from the light source.

3 Camera File

A sample template of a camera file looks like the following:

- **Camera Count** Num
- **Camera "Cid"** id
  - **Position** X Y Z
  - **Gaze** X Y Z
  - **Up** X Y Z
  - **ImagePlane** Left Right Bottom Top Distance HorRes VerRes
  - **OutputName** imagename.ppm

1. Camera Count
Count of the cameras that will be used in the scene can be obtained from this line.

2. Camera
Camera with the **Cid** is defined in this attribute. **Position** parameters define the X, Y, Z coordinates of the camera. **Gaze** parameters define the direction that the camera is looking at, **Up** parameters together define the up vector of the camera. **ImagePlane** attribute defines: the coordinates of the image plane in **Left**, **Right**, **Bottom**, **Top** parameters; distance of the image plane to the camera in **Distance** parameter; and the resolution of the final image in **HorRes** and **VerRes** parameters. All values are floats except **HorRes** and **VerRes**. These two are integers. You may assume that the **Gaze** vector of the camera is always perpendicular to the image plane. **OutputName** is a string which represents the name of the image file that you must produce.

4 Scene File

A sample template of a scene file looks like the following:

- **Ray Reflection Count** Num
- **Background Color** R G B
- **Ambient Light** R G B
- **Point Light Count** Num
    - **Position** X Y Z
    - **Intensity** R G B
- **Material Count** Num
- **Material "Mid"** id
    - **Ambient** R G B
    - **Diffuse** R G B
    - **Specular** R G B SpecExp
    - **Reflectance** R G B
- **Vertex Count** Num
- **Vertex Data**
    - **Coordinates** X Y Z
- **Model Count** Num
- **Mesh "Mid"** id
    - **Triangle Count** Num
    - **Material Id** Mid
    - **Vertices** Vid1 Vid2 Vid3
- **Sphere "Sid"** id
    - **Material Id** Mid
    - **Radius** R
    - **Center** Vid1

1. Ray Reflection Count
Specifies how many bounces the ray makes off of mirror-like objects. Applicable only when a material has non zero reflectance value.

2. Background Color
Specifies the R, G, B values of the background. If a ray through a pixel does not hit any object, the pixel will be of this color.

3. Ambient Light
R, G, B intensity parameters of the ambient light as floats. This is the amount of light received by each object even when it is in shadow.

4. Point Light Count
Specifies how many point lights are in the scene file. Position X, Y, Z and Intensity R, G, B values for the light source. All values are floats. The numbers of Position and Intensity lines are equal to the number of point lights.

5. Material Count
Specifies how many materials are in the scene file.

5. Material
Material with id Mid is defined with ambient, diffuse, specular, and reflection properties for each color channel. Values are floats between 0.0 and 1.0. SpecExp defines the specularity exponent in Phong shading. Reflectance represent 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 specified parameters.

6. Vertex Count
Specifies how many vertices are in the scene file.

7. Vertex
X,Y,Z float coordinates of the vertex.

8. Model Count
Specifies how many models are in the scene file.

9. Mesh
First line indicates how many vertices are in the mesh model. Vertices of triangles are are given in counter-clockwise order for computing normals. MaterialId specifies the material of the mesh. All values are integers.

10. Sphere
Material ID of the center of the sphere is Mid. Radius is its radius in float. Center is the specific vertex that has an id called Vid1.

5 Hints & Tips

1. Start early!
2. There are several files that is needed to be filled by you. For example; You should fill the function called **Intersect()** for both ***Triangle.h*** and ***Sphere.h*** files for finding every ray intersection. Also, you should use **ReadScene()** function inside of ***Scene.h*** for parsing the scene. In addition, you should render each camera on the function called **Render()** in file ***Camera.h***.
3. 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 10 points out of 100.
4. You may use -O2 or -O3 option while compiling your code for optimization.
5. Try to pre-compute anything that would be used multiple times and save these results somewhere. For example, you can pre-compute the normals of the triangles and save it in your triangle class when you read the scene file.
6. We will not create 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 will be tested please ask this in the newsgroup.
7. 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).
8. If you see generally correct but noisy results (black dots), it is most likely that you are falling victim to 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).
9. We will evaluate your results visually. Therefore, if you have subtle differences (numerically different but visually imperceivable) in the resulting images that will not be a problem.

6 Regulations

1. **Programming Language:** C/C++
2. **Late Submission:** You can submit your codes up to 3 days late. Each late day will incur a penalty of 10 points. After 3 days, you will get 0.
3. **Cheating: We have zero tolerance policy for cheating**. There is no teaming up for computer graphics homeworks. People involved in cheating will be punished according to the university regulations and will get 0. You can discuss algorithmic choices, but sharing code between each other 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 a "part" of the code from somewhere/somebody else - this is also cheating. Please be aware that there are "very advanced tools" that detect if two codes are similar. So please don't think you can get away with by changing a code obtained from another source.

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

5. **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 "raytracer" and should be able to be run using command "./raytracer scene_name.txt camera_name.txt". If your makefile doesn't work (or you don't have one) and therefore we will have to manually try to compile your code - there will be an automatic penalty of 10 points.

6. **Evaluation:** Your codes will be evaluated based on several input files including, but not limited to the test cases given to you as example on Department Inek Machines! Rendering all scenes correctly within the time limit will get you 100 points. The fastest 10 ray tracers will be awarded 10 points bonus. The speed will be measured in a complex scene to avoid noise in measurements. To be eligible for the speed bonus, the produced images must first be correct.