CNG 477
Introduction to Computer Graphics
Homework 1 – Ray Tracing
Instructor: Prof. Dr. Yusuf Sahillioğlu
Duration: 3 weeks

1. Homework Definition

Ray tracing is a rendering technique for generating an image by simulating propagation of light in the real world. It is used in the applications in which real-time generation of the images is not necessary, such as animations.

In this homework, you are asked to implement a basic ray tracer you have studied in the lecture which reads the scene from a text file and outputs the scene as an image. The details of the homework is given below.

2. Specifications & Regulations

a. You must use C++ (You can use C++ 11) programming language to implement the ray tracer and your code will be tested in Linux environment. Please see Submission Section for submission details.

b. You will read the input from a .txt file named as `'input.txt'` and write the rendered scene as an image to a .ppm file named as `'output.ppm'`. Please see Input & Output Section for the details of this issue.

c. Point lights will be defined by their intensity. The intensity due to such a light source falls off as inversely proportional to the squared distance from the light source. To simulate this effect, you should attenuate the intensity of point light as,

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

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

d. You must use *Blinn-Phong shading model* for the specular shading computations.

e. For each input file, 15 minutes will be given to your programs to produce the output. When your program exceeds this time limit, you will get zero from that input case.

f. We may test your implementations with input files other than the provided inputs, and/or changing the position of the camera in the input files provided to you. Thus, make sure you test your code carefully.

g. You can refer to Hints Section for the tips which make your way easier. Note that using those tips are not mandatory, they are given to ease you implementation.

h. This is an individual homework. 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 you 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.

3. Input & Output

a. You will read the scene from the file named as `'input.txt'` which will be placed in the directory of your source code. In this file, there will be tags starting with '#' character and the related values with the tag will be listed under it. The list of tags and their descriptions are listed below:

#BackgroundColor: Specifies RGB values of the background. If any ray does not hit any object, the color of the associated pixel should have this value.
- Three integer values between 0 and 255 will be given under the tag corresponding to R, G, and B channels, respectively.
- There will be only one of this tag in the scene file.

#MaxRecursionDepth: Specifies the number of bounces the ray makes off of mirror-like objects. Applicable only when a material has nonzero mirror reflectance value. Primary rays are assumed to start with zero bounce count.
- An integer value will be given under the tag specifying the value.
- There will be only one of this tag in the scene file.

#ShadowRayEpsilon: When a ray hits an object, you are going to send a ray from the intersection point to each point light source. Due to floating point precision errors, sometimes the ray hits the same object even if should not. Therefore, you should use this value which is a floating point number, to move the intersection point a bit further so that it does not intersect the same object again.
- A floating point number will be given under the tag specifying the value.
- There will be only one of this tag in the scene file.

#Camera: Parameters defining the camera will be given under this tag.
- In the first line under the tag, position parameters of the camera will be given as three floating point numbers, specifying x, y, z coordinates, respectively.
- In the second line under the tag, *gaze* parameters will be given. These will define the *gaze vector* which is the direction that the camera is looking at. *Gaze* parameters will consist of three floating point numbers specifying x, y, z values of the *gaze vector*.
- In the third line under the tag, up parameters will be given. These will define the *up vector* of the camera. Up parameters will be three floating point numbers specifying x, y, z values of the *up vector*.

- In the fourth line under the tag, *near plane* attributes will be listed. These will consist of four floating point numbers specifying *left*, *right*, *bottom*, and *top* parameters, respectively.
- In the fifth line under the tag, *near distance* will be given. This value is the distance between the image plane and the camera. The value will consist of one floating point number.
- In the sixth line under the tag, *resolution* of the output image will be given. These will consist of two integer values specifying the number of pixels in the *width* and *height*, respectively.
- There will be only one of this tag in the scene file.

#Material: A material will be defined with *ambient*, *diffuse*, *specular*, and *mirror reflectance* values for each color channel in this homework.
- In the first line under the tag, index of the material will be given.
- In the second line under the tag, *ambient reflectance* will be given. It will be specified as three floating point numbers each between 0 and 1 representing R, G, and B channels, respectively.
- In the third line under the tag, *diffuse reflectance* will be given. It will be specified as three floating point numbers each between 0 and 1 representing R, G, and B channels, respectively.
- In the fourth line under the tag, *specular reflectance* will be given. It will be specified as three floating point numbers each between 0 and 1 representing R, G, and B channels, respectively.
- In the fifth line under the tag, *phong exponent* for the specular reflectance will be given. It will be specified as a floating point number.
- In the sixth line under the tag, *mirror reflectance* will be given. It will be specified as three floating point numbers each between 0 and 1 representing R, G, and B channels, respectively.
- There may be multiple material tags each introducing another material. You can use material indexes to differentiate between them, same material index will not be defined for more than one material.

#AmbientLight: *Ambient light* value will be given as three floating point numbers under the tag, corresponding to R, G, and B channels, respectively.
- There will be only one of this tag in the scene file.

#PointLight: A *point light* will be defined with its position and intensity in this homework.
- In the first line under the tag, index of the point light will be given.
- In the second line under the tag, position of the light will be given as three floating point numbers corresponding to x, y, and z coordinates, respectively.
- In the third line under the tagi intensity of the light will be given. It will consist of three floating point numbers for R, G, and B channels, respectively.
- There may be multiple point lights in the scene. You have to take into account all of them while rendering. You can use their indexes to differentiate between them, same index will not be defined for more than one point light source.

#VertexList: Vertexes will be listed under this tag line by line. Each line will correspond to one vertex. That means, there will be three floating point numbers in each line

corresponding to x, y, and z values of the vertex, respectively. Note that vertex indices starts from 1, not from 0.

#Sphere: A sphere will be represented with a *material index*, coordinates of its center and its radius.
- In the first line under the tag, sphere index will be given.
- In the second line under the tag, material index of the sphere will be given. You will assume that the sphere is built by the material to which the index corresponds while rendering.
- In the third line under the tag, vertex id corresponding to the coordinates of the center of the sphere will be given as an integer. You will use vertex list under #VertexList tag to reach the coordinates of the center.
- In the fourth line under the tag, radius of the sphere will be defined as a floating point number.
- There may be multiple spheres in the scene. You have to take into account all of them while rendering. You can use their indexes to differentiate between them, same index will not be defined for more than one sphere.

#Triangle: A triangle will be represented with a material index and its vertex positions.
- In the first line under the tag, triangle index will be given.
- In the second line under the tag, material index of the triangle will be given. You will assume that the triangle is built by the material to which the index corresponds while rendering.
- In the third line under the tag, vertex indexes corresponding to vertexes of the triangles will be given as three integers. You will use coordinates of the vertexes listed under the tag #VertexList Note that the indices will be given in counter clockwise order from which you can extract the frontface and backface information of the triangle.
- There may be multiple triangles in the scene. You have to take into account all of them while rendering. You can use their indexes to differentiate between them, same index will not be defined for more than one triangle.

#Mesh: Meshes are composed of faces all of which will be triangles in this homework.
- In the first line under the tag, mesh index will be given.
- In the second line under the tag, material index of the mesh will be given. You will assume that the mesh is built by the material to which the index corresponds while rendering.
- Under the second line of the tag, vertex indices corresponding to the corners of the triangles composing the mesh will be listed. Indices for one triangle will be given in each line, which will consist of three integers. The indices will be given in counter clockwise order.
- There may be multiple meshes in the scene. You have to take into account all of them while rendering. You can use their indexes to differentiate between them, same index will not be defined for more than one mesh.

b. Structure of a typical input file will be as follows:

```
#BackgroundColor
r g b

#MaxRecursionDepth
d

#ShadowRayEpsilon
e

#Camera
x_cam y_cam z_cam
x_gaze y_gaze z_gaze
x_up y_up z_up
left right bottom top
distance
width height

#Material
id1
r_amb g_amb b_amb
r_dif g_dif b_dif
r_spec g_spec b_spec
spec_exp
r_mir g_mir b_mir

#Material
id2
. . .
. . .

#AmbientLight
r g b

#PointLight
id1
x y z
intensity_r intensity_g intensity_b

#PointLight
id2
. . .
. . .

#VertexList
x1 y1 z1
x2 y2 z2
. . .

#Sphere
```

```
id1
material_id
center_ind
r

#Sphere
id2
. . .
. . .

#Triangle
id1
material_id
ind1 ind2 ind3

#Triangle
id2
. . .
. . .

#Mesh
id1
material_id
ind11 ind12 ind13
ind21 ind22 ind23
. . .

#Mesh
id2
. . .
. . .
```

c. There will be at least one whitespace between the values given under the tags in the scene file.

d. You must output the image in .ppm format (which is a very basic format for which you can search the Internet to learn how to output an image in this format) named as `'output.ppm'` to the directiory where your source code files reside. Required resolution is provided under #Camera tag in the input file.

You are provided three sample input files which you can investigate and corresponding outputs.

4. Submission

a. Your submission should be a .zip, .rar, or .tar file which should include your source code and a `Makefile`. Please note that your source code will be compiled only with `make` command, thus do not forget to prepare a proper `Makefile`.

b. Name your submission file as hw1_[student1_id]_[student2_id], i. e., names of your submission files will be `hw1_eXXXXXXX_eYYYYYYY.zip` or `hw1_ eXXXXXXX_eYYYYYYY.rar` or `hw1_ eXXXXXXX_eYYYYYYY.tar`. You will work in groups of 2.

c. Send your submission files through ODTUClass.

d. After unzipping your submission file, your code will be compiled with the following command:

```
make
```

After that, the program will be run with the following command:

```
./Raytracer input.txt
```

That means your executable should be named as `Raytracer`. Make sure your code will work in this way.

e. Prepare a report file in pdf which includes i) screenshots of various stages/parameters of your execution, ii) screenshots of your final output results for the 3 input files in hw package, iii) encountered problems and interesting observations (optional).

f. Late submission is not allowed.

5. Hints

a. Implementation of a ray tracer is a cumbersome and time consuming work so start as earlier as possible.

b. You may use `-O3` option while compiling your code. It will reduce the rendering time considerably.

c. You can precompute the normals of the triangles while reading the input and save the results which will improve rendering time.

d. Stick with object oriented coding principles while implementing the homework so that your code will be less buggy and less complicated.

e. You can use *Eigen* library for linear algebra tasks.

f. Your code will not be tested with strange configurations such as a camera inside of a sphere or a point light inside of a sphere. Besides, input files will not include intersecting objects.

g. For questions related with the homework, send e-mail to ys@ceng.metu.edu.tr.