Middle East Technical University Department of Computer Engineering

CENG 477

Introduction to Computer Graphics

Fall '2019-2020

Assignment 1 - Ray Tracing

Due Date: 29 October 2019 Tuesday, 23:55

Objectives

Ray tracing is a fundamental rendering algorithm. It is commonly used in applications in which the quality of the images are more important than the time it takes to create them such as architectural simulations and animations. You are going to implement a very basic ray tracer simulating the propagation of the light in vacuum in this assignment.

Specifications

- A template for implementation of a basic ray tracer is provided to you. This template consists of header files of the classes you will need, declarations of the public methods and public variables, implementation of an XML parser, and a ppm file saver. Explanation of the files are not provided here. Take a look at the files before starting, read the comments carefully, and try to understand how you can use the provided utilites. You are allowed to add your own variables and methods to only *private* area of the classes, not to *public* area. The assignment is implementable in this way, you do not actually need to add more public variables and methods to the classes.
- Your program will take a scene as XML file as command line input. You should save the output image as a ppm file. Parsing the XML file, and a method to save the image as ppm file is implemented for you. Some sample scene files are provided to you in *inputs* directory. Correct outputs of those scenes are also provided in *outputs* directory.
- 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.
- 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.
- You should use Blinn-Phong shading model for the specular shading computations.
- You will implement two types of light sources: ambient and point. Although there will be only one ambient light, there may be multiple point light sources. The intensity of these lights will be given as (r, g, b) triplets.

- 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 rgb value given in the XML file) and $E(d)$ is the irradiance at a distance of d from the light source.

Scene File

The scene file will be formatted as a XML file. In this file, there may be different number 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.

Explanations for each XML tag are provided below:

- **BackgroundColor:** Specifies rgb value 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-intersections 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:** Defines the coordinates of the camera.
  - **Gaze:** Defines 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:** Defines the up vector for the camera.
  - **NearPlane:** Defines the coordinates of the image plane with left, right, bottom, top parameters.
  - **NearDistance:** Defines the distance to the image plane to the camera.
  - **ImageResolution:** Defines the resolution of the image as width, and height.
  - **ImageName:** Defines the name of the output file.

    Cameras defined in this assignment 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:** Defined by an intensity rgb triplet. This is the amount of light receive by each object even if the object is in shadow.
- **PointLight:** Defined by a position (triplet as x, y, z coordinates) and intensity (rgb triplet).
- **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 *MirrorReflectance* parameters.
- **VertexData:** Each line contains a vertex whose x, y, and z coordinates are given as floating point values, respectively.
- **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 (Note that vertices are 1-based, i.e., the index of the first vertex in the *VertexData* field is 1 not 0.). 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 you 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 (Note that vertices are 1-based, i.e., the index of the first vertex in the *VertexData* field is 1 not 0.). Radius attribute is the radius of the sphere.
- **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. That is, each line is represented by three integer vertex ids given in counter-clockwise order (Note that vertices are 1-based, i.e., the index of the first vertex in the *VertexData* field is 1 not 0.). Material attribute represents the material id of the mesh.

You can open the sample XML files given to you with a text editor to see and understand the scene format better.

Regulations

- **Programming Language:** The programming language for this assignment is C/C++ (You can use C++11).
- **External Libraries:** You can use external library such as Eigen for matrix/vector algebra, like adding, subtracting, multiplying (both cross and dot product) vectors. If you do so, update *Vector3f* variables inside of the files according to the ones your external library uses.
- **Class Interfaces:** The classes provided to you has public variables and methods. You have to implement thos methods. You can define you own variables and methods only as *private* area of the classes. You are not allowed to add new variables and methods to *public* area of the classes. Otherwise, you will lose considerable amount of points.
- **Grouping:** You can team-up with another student. There is a *members.txt* file among the files provided to you. Write your student ids into that file. Only one member of the team may upload the assignment, upload of the assignment by two of the members of the team is not necessary.
- **Submission:** Submission will be done via OdtuClass.

- **Late Policy:** You can submit your codes up to 3 days late. Each late day will be deduced from the total 7 days 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 and still get 0, you cannot claim back your late days.

- **Bonus Grades:**

  - You can get 10 points bonus if your ray tracer is among the fastest N ray tracers to be determined by our benchmarks. Note that your outputs should be correct to be eligible for this bonus.
  - You can get 10 points bonus if you create and share interesting XML scene files. These should not be simple scenes as we ahave already shared such scenes with you. Interesting scenes to get bonus will be determined by the course instructors and assistants.

- **Evaluation:** Your codes will be evaluated on department Inek machines. Be careful about this if you are implementing your homework in other operating systems such as Windows or macOS. Although some simple scenes are provided to you with the correct outputs, other scenes with different camera positions may be used while evaluating your code.

  A makefile is provided to you to compile the code, the output of the makefile is named as *raytracer.out*. While evaluating your assignment, your code will be compiled only with the following command:

  $make

  After that, a scene will be given to your program as a XML file as follows:

  $./raytracer inputs/input01.xml

  Your program saves the output(s) as ppm file(s) (Whose implementation is done) in the same directory with the source code files.

  Stick with the evaluation specification, if the codes you will provide will not be compiled, your grade will be zero. Even if you make it work during the objection stage, you will lose points.

- **Cheating:** We have zero tolerance 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 part code are strictly forbidden.