Middle East Technical University | Department of Computer Engineering

CENG 477

Introduction to Computer Graphics

Fall '2025-2026
Assignment 1 - Ray Tracing
(v.1.0)

Due date: November 2, 2025, 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 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 Section 3. 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 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. Note that *horse_and_mug* scene is exempt from the 30-minute rule as it has a high object count, and it is optional to render.

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 values of these lights will be given as (R, G, B) color triplets that are not restricted to [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 distance d from the light source.

9. **Back-face culling** is a method used to accelerate the ray - scene intersections by not computing intersections with triangles whose normals are pointing away from the camera. Its implementation is simple and done by calculating the dot product of the ray direction with the normal vector of the triangle. If the sign of the result is positive, then that triangle is ignored. Note that shadow rays should not use back-face culling. In this homework, back-face culling implementation is optional.

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. Only applicable for primary rays sent through pixels.

- **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 from the hit point in the direction of the hit point's normal vector so that the shadow ray does not intersect with the same object again. Note that ShadowRayEpsilon value can also be used to avoid self-intersections while casting reflection rays from the intersection point.

- **MaxRecursionDepth:** Specifies how many bounces the ray makes off of mirror-like objects. Applicable only when a material is of type mirror. 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:

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

- **AmbientLight:** is defined by just an X, Y, Z radiance triplet. This is the amount of light received by each object even when the object is in shadow. Color channel order of this triplet is RGB.

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

- **Material:** A material can be defined with ambient, diffuse, specular, and mirror reflectance properties for each color channel. The values are floats between 0.0 and 1.0, and color channel order is RGB.
  PhongExponent defines the specularity exponent in Blinn-Phong shading.
  MirrorReflectance represents the degree of the mirrorness of the material. If the material is of type mirror, you must cast new rays and scale the resulting color value with the MirrorReflectance parameters. The attribute *type="mirror"* is provided only for the materials which has non-zero MirrorReflectance.

- **VertexData:** Each line contains a vertex whose x, y, and z coordinates are given as floating point values, respectively. The first vertex's ID is 1.

- **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. Therefore, each line is composed of three integer vertex IDs 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.

- **Plane:** A plane is represented by Material, Center, and Normal attributes. Material is the material ID. Center is the integer vertex ID of a point lying on the plane. Normal is a 3D vector giving the plane's outward-facing normal (it need not be unit; it will be normalized during parsing/shading). The plane is infinite.

- **Cylinder:** A finite right circular cylinder is represented by Material, Center, Axis, Radius, and Height attributes. Material is the material ID. Center is the integer vertex ID of the cylinder's geometric center (midpoint along its axis). Axis is a 3D vector pointing from the center toward the top cap (it need not be unit; it will be normalized internally). Radius is the cylinder radius, and Height is the distance between the two caps (the cylinder extends $\frac{Height}{2}$ in both directions along the axis from Center).

Ray–cylinder intersection concept: We treat the cylinder using its natural frame: a center C, an axis direction $\mathbf{A}$ (normalized), a radius r, and a height h (caps at $\pm h/2$ along $\mathbf{A}$). Given a ray $O + t\mathbf{D}$, we split motion into two pieces: the part *along* the axis and the part *perpendicular* to it.

- **Side test (infinite tube).** Project the ray onto the plane perpendicular to $\mathbf{A}$. In that 2D view, the cylinder becomes a circle of radius r. Intersect the projected ray with this circle (a simple quadratic). This yields up to two candidate distances t where the ray would hit the *infinite* tube. For each candidate, check the hit point's axial coordinate (distance along $\mathbf{A}$) and keep it only if it lies within the finite height interval $[-h/2, h/2]$. Those that pass are "lateral" hits.

- **Cap tests (two disks).** Intersect the ray with the two planes orthogonal to $\mathbf{A}$ at the top and bottom of the cylinder ($y = \pm h/2$ along $\mathbf{A}$). For each plane hit, check whether the hit point is inside the circular cap (distance to the cap center $\leq r$).

- **Choose the earliest valid hit.** Among all valid candidates (side and caps), pick the smallest positive t (in front of the camera) and return it together with the part tag.

- For additional background, you can benefit from NYU Media Lab and Wikipedia.

4 Hints & Tips

1. Start early. It takes time to get a ray tracer up and running.
2. You may use the -O3 option while compiling your code for optimization. This itself will provide a huge performance improvement.
3. 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.
4. 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).
5. 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).
6. 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 on ODTUClass.

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 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++. You also must use gcc/g++ for the compiler. Any C++ version can be used as long as the code works on Inek machines. 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. **Changing the Code Template:** We provide you a code template. You are free to edit, rename or delete any file from the given code template as long as you comply with the submission rules below. However, keep in mind that any error introduced by the changes you made is your responsibility.

3. **Additional Libraries:** If you are planning to use any library other than *(i)* the standard library of the language, *(ii)* pthread, *(iii)* the parser and the writeppm function in the template code, please first ask about it on ODTUClass and get a confirmation, as any unapproved library usage may cause a penalty during grading.

4. **Submission:** Submission will be done via ODTUClass. To submit, you must select an option in "HW Groups" page in ODTUClass according to the criteria in the Groups section below. 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 following commands (scene.xml will be provided by us during grading):

   tar -xf raytracer.tar.gz
   make
   ./raytracer scene.xml

   Any error in these steps will cause a 10 points penalty over 100 points!!

5. **Groups:** You can team-up with another student. To form a group, both students must select the same group with 2 people capacity in "HW Groups" page on ODTUClass. If you want to work on the homework by yourself, in the same page select any "Individual Work" option which only have 1 person capacity. A selection must be made to upload the homework.

6. **Previous Submissions:** Our aim in homework is to establish an environment where students contribute equally. If you have previously submitted homework in past semesters with a grade exceeding 50 but wish to approach the current assignments entirely from scratch, that is acceptable to us, otherwise we regret to inform you that we will not be able to accept joint submissions for the current semester for corresponding assignment. In such cases, partners are required to complete the corresponding homework individually. The key point here is to avoid submitting work that closely resembles assignments from previous semesters. If both you and your partner commit to completing the homework without incorporating any parts from previous submissions, then there is no issue with collaborating as partners.

7. **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. You must e-mail the assistants if you want your submission not to be evaluated (and therefore preserve your late day credits).

8. **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.

9. **Forum:** Any updates/corrections and discussions regarding the homework will be on ODTU-Class. You should check it on a daily basis.

10. **Evaluation:** Your codes will be evaluated on Inek machines using various scene files, including but not limited to the examples provided. A subset of the given test cases will be used in grading (not necessarily all of them). We will not use automated grading; outputs will be inspected visually, so tiny pixel-value differences will not cause point loss. Rendering all evaluated scenes correctly within the time limit will earn 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" [type="mirror"]>
                        <AmbientReflectance>X Y Z</AmbientReflectance>
                        <DiffuseReflectance>X Y Z</DiffuseReflectance>
                        <SpecularReflectance>X Y Z</SpecularReflectance>
                        <MirrorReflectance>X Y Z</MirrorReflectance>
                        <PhongExponent>X</PhongExponent>
                </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>
```