Middle East Technical University Department of Computer Engineering

# CENG 795

## Advanced Ray Tracing

Fall '2022-2023
Assignment 2 - Speed and Geometry
(v.1.0)

Due date: November 13, 2022, Sunday, 23:59

## 1 Objectives

The focus of this assignment is to add support for acceleration structures, geometric transformations, and instancing to your ray tracers. These features will help you to create and render much more complex scenes.

**Keywords:** *transformations, instancing, bounding volume hierarchy (BVH), kd-tree*

## 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 PNG format. You can use a library of your choice for saving PNG images.
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. There is no time limit for rendering the input scenes. However, if you implement an acceleration structure such as kd-tree or BVH correctly, it should take only a few seconds to render even the most complex scenes.
6. You should use Blinn-Phong shading model for the specular shading computations. Mirrors and dielectrics should obey Fresnel reflection rules. Dielectrics are assumed to be isotropic and should obey Beer's law.

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 <u>not</u> 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 PNG file. This step will be replaced by the application of a tone mapping operator in our later homeworks.

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. Experiment with enabling and disabling it in your ray tracers and report your time measurements in your blog post.

10. **Degenerate triangles** are those triangles whose at least two vertices coincide. The input files given to you should be free of such cases, but models downloaded from the Internet occasionally have this problem. You can put a check in your ray tracer either to detect or ignore such triangles (using such triangles usually produce NaN values).

# 3 Scene File

Please see HW1 for a detailed description of the scene file. In this section, only the elements introduced in this homework are explained.

- **Transformations:** All transformations are defined in this element group. Each of the transformations defined below has an id for referring to them later.

  **Translation** $\Delta x$, $\Delta y$, and $\Delta z$ values corresponding to translation amounts in each axis.

  **Scaling** $S_x$, $S_y$, and $S_z$ scaling factors along each axis.

  **Rotation** $\theta$, $x$, $y$, and $z$ values that define the angle of rotation in degrees and the rotation axis. Positive angles correspond to counter-clockwise rotations.

- **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. The color channel order of intensity is RGB.

- **Mesh:** The mesh definition now has an extra element called **<Transformations>** that defines which transformations will apply to this mesh and in which order. For example, if the value of this element is **r1 t1 s2 r2**, it means first apply rotation transformation with id equal to 1, then translation with id equal to 1, then scaling with id equal to 2 and then rotation with id equal to 2. If we represent it as a matrix multiplication, the composite transformation should look like this:

$$M = R_2 S_2 T_1 R_1, \tag{1}$$

  and the transformed coordinates of a vertex $v$ will be equal to:

$$v' = Mv \tag{2}$$

- **Triangle:** The same transformations element can be found under the triangle element as well.

- **Sphere:** The same transformations element can be found under the sphere element as well.

- **MeshInstance:** This is a new object that is introduced by this homework. A mesh instance is a copy of its base mesh with a potentially different material and transformation. A mesh instance can be instanced from another mesh instance as well and because of that ids of meshes and mesh instances must not be the same. A mesh instance does not have **<Faces>** element as it uses the same faces with its mother mesh. An example usage of this element is given below:

```
<MeshInstance id="3" baseMeshId="2" resetTransform="true">
    <Material>3</Material>
    <Transformations>t2 r1</Transformations>
</MeshInstance>
```

  This defines a mesh instance whose mother mesh (or mesh instance) is the object with id equal to 2. This instance is using material 2, which may be different from the base object's material. The **resetTransform** attribute, whose default value is **false**, indicates whether the defined transformation should apply on top of the transformations that are already defined for the base object. If its value is true, the defined transformation is the only one that should apply on the initial geometry. Otherwise, the new transformation should be applied on top of what is already defined for the base mesh. This can be represented as follows:

$$M_{instance} = \begin{cases} R_1 T_2 & \text{if resetTransform is true} \\ R_1 T_2 M_{base} & \text{if resetTransform is false (default)} \end{cases}, \tag{3}$$

  where $M_{base}$ is the composite transformation of the base object.

# 4 Hints & Tips

In addition to those for HW1, the following tips may be useful for this homework.

1. There are no special elements that are defined for acceleration structures. You are expected to implement a structure of your choice, with BVH being the recommended one.

2. As our goal is to speed up our ray tracers, adding multi-threading support could be quite beneficial at this point. There are different ways of implementing this. You can divide the image region into sub-regions whose count is equal to the number of threads. Each thread renders its own region. However, if some threads finish their jobs earlier than others, this may be sub-optimal. Alternatively, you can divide the image into a larger number of regions and assign threads to each region. Whenever a thread finishes processing its own region, it can get assigned to a new one. This would be somewhat more complex to implement but would result in a more efficient ray tracer.

# 5 Bonus

I will be more than happy to give bonus points to students who make important contributions such as new scenes, importers/exporters between our XML format and other standard file formats. Note that a Blender exporter[1], which exports Blender data to our XML format, was written by one of our previous students. You can use this for designing a scene in Blender and exporting it to our file format.

# 6 Regulations

1. **Programming Language:** C/C++ is the recommended language. However, other languages can be used if so desired. In the past, some some students used Rust or even Haskell for implementing their ray tracers.

2. **Changing the Sample Codes:** You are free to modify any sample code provided with this homework.

3. **Additional Libraries:** If you are planning to use any library other than *(i)* the standard library of the language, *(ii)* pthread, *(iii)* the XML parser, and the PNG libraries please first ask about it on ODTUClass and get a confirmation. Common sense rules apply: if a library implements a ray tracing concept that you should be implementing yourself, do not use it!

4. **Submission:** Submission will be done via ODTUClass. To submit, 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**

[1] `https://saksagan.ceng.metu.edu.tr/courses/ceng477/student/ceng477exporter.py`

**./raytracer scene.xml**

Any error in these steps will cause point penalty during grading.

5. **Late Submission:** You can submit your codes up to 3 days late. Each late day will cause a 10 point penalty.

6. **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. By the nature of this class, many past students make their ray tracers publicly available. You must refrain from using them at all costs.

7. **Forum:** Check the ODTUClass forum regularly for updates/discussions.

8. **Evaluation:** The basis of evaluation is your blog posts. Please try to create interesting and informative blog posts about your ray tracing adventures. You can check out various past blogs for inspiration. However, also expect your codes to be compiled and tested on some examples for verification purposes. So the images that you share in your blog post must directly correspond to your ray tracer outputs.