Foundations of Raytracing

Introduction

- Based on defined camera, raytracing is inherently providing **perspective or orthographic projection**
- Raytracing, foundamentally, **solves visibility problem**: two points are visible to each other if line segment that joins them does not intersect any obstacle.
- Using raytracing, we solve **light transport**
- Raytracing is coupled with **shading** to calculate color of visible objects.

Raytracing: generating rays

- Ray is fundamental element of raytracing. It is used for finding visible objects from camera as well as later in light transport.
- To generate ray, we use camera description in a 3D scene, e.g., pinhole camera.
  - Camera defines eye (aperture) position as well as film plane position, size and distance from aperture. Film plane is raster image: array of pixels
- Rays are generated by starting from eye and passing through the center of each pixel in the film plane.
  - It is possible, and in practice always desired, to generate multiple rays for each pixel. We will discuss this later, but it is important to note that large portion of the scene is actually represented by only one pixel. Since pixel can represent only one color, it is important to use multiple rays to obtain the color which is the most representative for that part of the scene covered by the pixel.
- Generated rays are called **camera/primary rays**. These rays will be used to compute the visible objects for current camera position. This method can be called **ray-casting**

<IMAGE: GENERATING RAYS>

Raytracing: generating rays algorithm

- TODO

Camera rays: testing for intersections

- Camera rays are "sent" into the 3D scene
  - They are tested for intersection with each object in 3D scene. In other words, we loop over all 3D objects and test them for intersection with camera ray.
  - Note that in this step, we are interested in a shape of 3D object and that material will be used later. Therefore, decoupling material and shape of 3D object is useful.
- Two possibilities are:
  - Ray intersects an object
  - Ray doesn't intersects anything → intersect background
-

Testing object intersections

- In lesson on objects in 3D scene, we have seen that various shape representations exist for 3D objects.
  - Parametric representations (e.g., simple objects; spheres), mesh, curves, curved surfaces, subivision surfaces, voxels, SDFs, etc.
- Objects which shapes are represented parametrically can be tested analytically
  - Very often, you will see sample scenes rendered with raytracing containing planes and spheres!
- For other objects, ray-shape intersection must be defined for each representation separately.
  - For example, different intersection test is performed for triangle meshes, quad meshes and curved surfaces, etc.
- Alternative solution (which is taken by almost all professional rendering software) is to convert each shape representation to same internal representation which is used for rendering. This internal representation is in almost all cases triangulated mesh.
  - Why triangulated mesh
- https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-overview/ray-tracing-rendering-technique-overview

Intersection tests

- Plane
- Sphere
- Triangle

Testing intersections algorithm

- TODO

Object intersection shading

- Once intersection with object is found, we need to calculate color and intensity (appearance) of that point. Method for calculating this is called **shading**.
-

Overview of raytracing