Foundations of Raytracing

Introduction

- Based on defined camera, ray-tracing-based rendering is inherently providing **perspective or orthographic projection** in rendered image.
- Raytracing, foundamentally, **solves visibility problem**: two points are visible to each other if line segment that joins them does not intersect any obstacle.
  - First, we need to determine which objects are visible from camera
  - Secondly, we need to determine color and intensity of visible objects. This is solved using **shading** which relies on **light transport** to obtain incoming light to surface.

Raytracing: generating rays

- Ray is fundamental element of raytracing. It is used for finding visible objects from camera as well as later in shading, that is, 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. Later, once intersection is determined material will be used during the shading step. 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 and for this representation, the conversion method is called **tessellation**

Tessellation and triangulated mesh.

- Conversion of almost any type of surface to a triangulated mesh is well researched and feasible.
- Triangulated mesh is also basic rendering primitive for rasterization-based renderers as we will see and graphics hardware is adapted to working with triangles.
- Triangles are necessary co-planar which makes various computations, such as ray-triangle, much easier
- Lot of research was devoted to efficient computation of ray-triangle
- For triangles we can easily compute barycentric coordinates which are essential to shading.
- Tessellation process can be done after modeling of shape is done and when exporting takes place or it can be done during rendering.

Intersection tests

- Plane
- Sphere
- Triangle

Testing intersections (triangulated meshes) algorithm

- TODO
-

Notes on testing intersections

- Time to render a scene is directly proportional to the number of triangles in the scene.
- Raytracing-based rendering, in shading step, has to test more triangles in the scene as compared to rasterizer where many of triangles can be discarded if not visible from camera (e.g., back faces of object – back face culling).
- Optimization: not all triangles have to be tested for intersections. If certain parts of the scene are not relevant for current ray, they can be skipped
  - Acceleration datastructures

Testing intersections: practical tip

- Same algorithm can be used for any kind of tracing ray into the scene since only information is current ray and objects in the scene
  - Camera ray for determining what is visible from camera
  - Light transport for gathering light
- Therefore, this part of the code can be encapsulated

Testing intersections: multiple hits

- Imagine scene where we have several layers of the objects so that behind one object, there is another one.
- Tracing ray in this case intersects multiple objects
- When determining what is visible from camera, we take the first intersection: one closest to the camera
- Note that based on material, objects behind will influence (transparent objects) or be simply discarded (opaque objects).
  - This computation is performed in shading step. This is important element to take in account - we need to trace additional rays for computing incoming light to intersected point not only from surfaces facing the intersected point – but also surfaces which are behind.

Testing intersections: multiple hits 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**.
- **Shading takes in account material and shape of the objects and couples it with light to compute color and intensity of intersected point.**
- Important information about the shape are: intersection position surface normal in intersected point, texture coordinates in intersected points, surface derivatives in intersection point.

Shading

- https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-overview/ray-tracing-rendering-technique-overview