CS7.302: Computer Graphics

Assignment 1, [100 points]

Deadline: 21st January 2024

Welcome to your first assignment. The focus of this assignment is to make the ray-object intersections more robust and efficient. The scenes needed for this assignment are in the **Assignment 1** directory in the scenes repository. Please pull the latest version on your devices. Also, remember to pull the latest code, as there may have been a few bug fixes.

Important Note: Please do not release your code for this and any future assignments publicly, including on public GitHub Repository. Please also do not share the code with other students who may take this course in the future. We ask this to avoid plagiarism in future versions of the course. We are going to enforce this strictly.

1 Camera co-ordinate system [20 points]

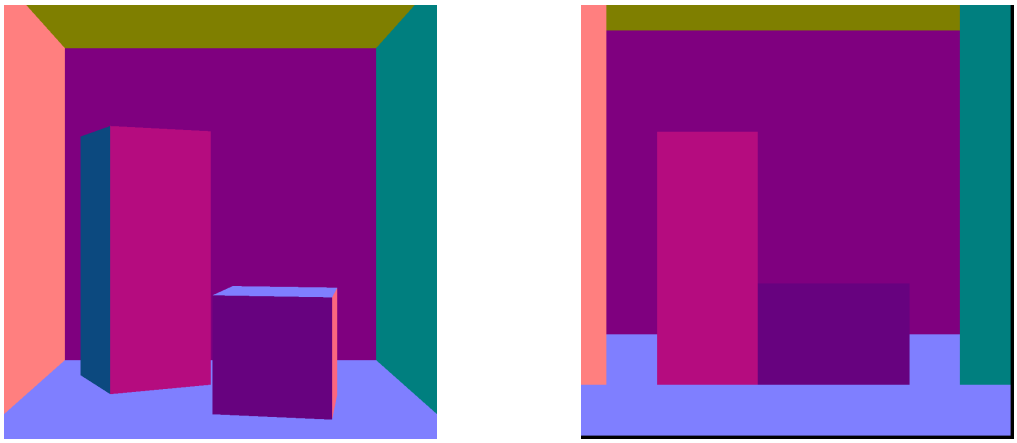

Figure 1: Renderings of **correct.json** (Left) and **incorrect.json** (Right)

You have been provided with two scenes, **correct.json** and **incorrect.json**. Both scenes share the same composition, i.e., the relative positions of the objects and the camera are identical. However, in the **incorrect.json** scene, all of the objects and the camera are shifted by 1000000 units along all X, Y, and Z coordinates. The rendering of both scenes should produce identical results, but this is not the case (see fig. 1).

Your task is to identify the cause of the difference in the rendered results and then modify the code to fix this issue. Note that you cannot change the **.obj** or **.json** files. Your modifications should enhance the code's robustness to such scenes and not be specific to the given scene.

2 Acceleration Structure (AS): Basic Bounding Volume Hierarchy (BVH) [80 points]

In the base implementation given to you, the ray-object intersection tests are implemented naïvely by iterating over all of the objects (listing 1) and then iterating over all triangles (listing 2) of the object and finding the closest intersection point. The runtime of this algorithm increases linearly ($\mathcal{O}(n)$) with the number of triangles in the scene, making the method unscalable for a large number of triangles.

```
Interaction Scene::rayIntersect(Ray&
    ray)
{
    Interaction siFinal;

    for (auto& surface : this->
    surfaces) {
        Interaction si = surface.
    rayIntersect(ray);
        if (si.t <= ray.t) {
            siFinal = si;
            ray.t = si.t;
        }
    }

    return siFinal;
}
```

Listing 1: scene.cpp

```
Interaction Surface::rayIntersect(Ray
    ray)
{
    Interaction siFinal;
    float tmin = ray.t;

    for (auto face : this->indices) {

        ...

        Interaction si = this->
    rayTriangleIntersect(ray, p1, p2,
    p3, n);
        if (si.t <= tmin && si.
    didIntersect) {
            siFinal = si;
            tmin = si.t;
        }
    }

    return siFinal;
}
```

Listing 2: surface.cpp

Figure 2: Current implementation of ray-object intersection test

The idea of Bounding Volume Hierarchy (BVH) is to avoid this linear search and instead build a binary tree over triangles. Doing so brings the runtime to logarithmic in the number of triangles of the scene ($\mathcal{O}(\log(n))$).

In this question, you are required to implement a BVH. Implement each sub-question in order (2.1, 2.2, 2.3). Each question progressively builds towards a simple BVH, each building upon the previous one.

2.1 Axis Aligned Bounding Boxes on Surfaces [20 points]

In this question, we will use axis-aligned bounding boxes (AABBs) to accelerate the intersection tests. Modify the code to do the following things:

- Create Axis-Aligned Bounding Boxes (AABBs) for each object and write a function to test whether a given ray intersects a given AABB.
- Iterate over all the bounding boxes of objects in the scene and test intersection with them.

- If the ray intersects the bounding box, iterate over all triangles of the object to find the closest intersection.

2.2 BVH on bounding boxes [40 points]

We will accelerate the intersection tests in this question by building a Bounding Volume Hierarchy (BVH) over the AABBs. Modify the code to do the following:

- Write code to create a BVH (see section 4) on the bounding boxes.
- Traverse the BVH recursively.
- If the ray intersects the AABB at the leaf node, iterate over all triangles of the object to find the closest intersection.

2.3 BVH on triangles (Two-Level BVH) [20 points]

Finally, we will build a two-level BVH to optimize the intersection queries further. Modify the code to do the following. We will retain the top-level BVH that was created in the previous question.

- Build a separate BVH for each object.
- Traverse the top-level BVH recursively.
- If the ray intersects the bounding box at the leaf node, then traverse the BVH of that object to find the closest intersection point.

3 Submission

You need to submit the modifed code along with a report as follows:

3.1 Question 1

Write the cause of the discrepancy in the renderings and describe your fix for the same in the report. Modify the original code and show that it works on the **incorrect.json** scene, during evaluation by TAs and as a rendered image in the report.

3.2 Question 2

Add a command line parameter which will allow you to choose the method for performing ray-object intersections. The renderer should be invoked as follows:

```
./render <scene_path> <output_path> <intersection_variant>
```

The `intersection_variant` will be an integer from $0 - 3$ where each integer is defined as follows:

- 0: Naïve intersection
- 1: AABB intersections (section 2.1)
- 2: BVH on AABB (section 2.2)
- 3: BVH on Triangles (section 2.3)

Test all of the intersection variants on the scenes provided in the **Assignment 1/Question 2** directory in the scenes github repo. The report should contain the timings for all variants of the ray-intersection tests as well as the rendered images obtained with them for each of the scene.

Create a zip of the modified code along with the report and submit it on Moodle. The name of the zip should be `<roll_number.zip>`.

4 Resources

We recommend you go through these resources:

- Building a BVH (With code): Jacco Bikker's blog on BVH
- PBRT-v3: Chapter 4, Section 4.3