Assignment 7 Bounding Volume Hierarchy
CS180 \CS280 Fall 2022

Professor: Lingqi Yan

University of California, Santa Barbara

Assigned on Nov 22, 2022 (Tuesday)

Due at 11:59 PM on Dec 4, 2022 (Sunday)

Notes:

- Be sure to read the Programming and Collaboration Policy on course website.
- Any updates or correction will be posted on EdStem, so check there occasionally.
- You must do your own work independently.
- Read through this document carefully before posting questions on EdStem.
- Submit your assignment on GauchoSpace before the due date.

---

1 Overview

In the previous assignment, we implemented the basic functionality of ray tracing, namely ray-casting and ray-triangle intersection. To find an intersection between a ray and the scene, we looped over all the objects in the scene and tested if there was an intersection. This works fine when the scene contains a small number of objects. However, when we want to render a complex scene with a huge amount of objects, this naive approach will be very inefficient. This is why we need some acceleration structures to speed up our ray-intersection routine. There are several acceleration structures to use, in this class we focus on the bounding volume hierarchy(BVH), which is an object partition method. In this assignment you will implement the ray-bounding volume intersection and the BVH traversal.

The functions you should take from your previous assignment are:

- `Render()` in Renderer.cpp: Paste your ray generation code from last assignment here.
- `rayTriangleIntersect()` in Triangle.hpp: Paste your ray-triangle intersection function from the last assignment here. Update the Intersection information accordingly.

The functions you need to implement for this homework:

- `IntersectP(const Ray& ray, const Vector3f& invDir, const std::array<int, 3>& dirIsNeg)` in the Bounds3.hpp: This function implements the boundingbox - ray intersection, and by implementing the process in lecture slides, you can find if a ray is intersects with the current bounding box or not.
- `getIntersection(BVHBuildNode* node, const Ray ray)`in BVH.cpp: After constructing the BVH, we can use it to accelerate our intersection routine. This is a recursive function and you need to call the `Bounds3::IntersectP` function you finished here.

2 Getting started

The only dependency of the start code is CMake. It should be able to run on your own machine. Please download the project's skeleton code, and build the project by using the following commands:

```
$ mkdir build
$ cd ./build
$ cmake ..
$ make
```

After this, you should be able to run the given code by using **./Raytracing**. There are several updates of the code base. Some general introductions are:

- Material.hpp: We separate the material paraters of an object to a class, now each object instance holds their own material.
- Intersection.hpp: This is the structure contains the information of an intersection.
- Ray.hpp: The ray class, contains its origin, direction, transport time t and its range.
- Bounds3.hpp: The class for our axis-aligned bounding box. Each bounding box can be specified by two points, pMin and pMax (why?). The `Bounds3::Union` function combines two bounding boxes into a new one. Also, each object in the scene holds their own bouding box.
- BVH.hpp: The class of our BVH acceleration structure. The scene now holds a `BVHAccel` instance. And start from a root node, we can recursively build our BVH from the objects list of the scene. The BVH is indeed a binary tree of the `BVHBuildNode` struct.

3 Grading and Submission

Grading:

1. (5 points) Submission is in the correct format, include all necessary files. Code can compile and run.
2. (20 points) Bounds intersection:
   You successfully implement the ray-Bounding box intersection function.
3. (15 points) BVH traversal:
   You implement the ray-scene intersection routine correctly. Return the correct intersection.

Your results will look like this:

Submission:
After you finish the assignment, clean your project, remember to include the CMakeLists.txt in your folder, whether you modified it or not. Add a short report in the directory, write down your full name and perm number inside it. **Briefly describe what you have implemented in each function.** Then compress the entire folder and rename it with your name, like "Goksu.zip". Submit **ONLY ONE** .zip file on GauchoSpace.