Assignment 2 report

Giulia Benvenuto

April 23, 2022

Environment:

- Windows 11
- Visual studio 2022 17.1

1 Ray Tracing: Triangle

I implemented the function called "**intersect_triangle**" which checks whether there are any intersections between a ray and a triangle. The implementation is similar to the one done for the parallelogram, only the verified conditions change.

2 Ex.1: Triangle Mesh Ray Tracing

I decided to implement the second method, in this case the function traverses the BVH tree and tests the intersections between the ray and the triangles at the leaf nodes.

The function checks whether a node is a leaf. The Node struct has four values. The value "*triangle*" is -1 for internal nodes so the function checks if this value is different from -1 (if it is different, it is a leaf node). Otherwise if there is an intersection with a box, the function traverses recursively the BVH Tree while it gets to a triangle inside a leaf node that intersects with the ray.

3 Ex.2: AABB Trees

I decided to build the AABB Tree for triangles using the **Top-Down Construction** method.

The function called "**AABBTree**", at first computes the centroids of all the triangles in the input mesh, then stores inside a vector called "*tIndices*" the indices of all the triangles and at the end it calls the recursive function called "**AABBTree_build**" which builds the AABB Tree.

The function "**AABBTree_build**" checks if the size of the vector which contains the indices of the triangles, if the size is 1, it means that there is only one triangle so it computes the root box using this triangle, it sets the root node equal to the root box, pushes it at the end of the vector "*node*" and then returns the value nodes.size() - 1 as root.

Otherwise if the vector size is bigger than 1, the function searches the biggest box which contains all the vertices, computes the longest axis of this box and then sorts the centroids along the axis found.

Then, the function performs the following steps:

- Splits the input set of triangles into two sets S1 and S2.
- Recursively builds the subtree T1 corresponding to S1.
- Recursively builds the subtree T2 corresponding to S2.
- Updates the box of the current node by merging boxes of the root of T1 and T2.
- Returns the new root node R.

4 Notes

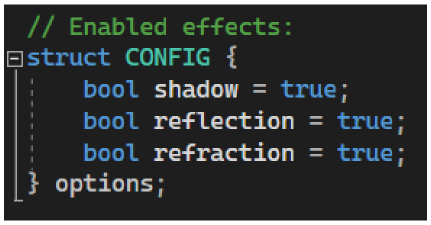
Since in this case shadow, reflection and refraction effects are not necessary I decided to implement a struct called "*options*", it contains three boolean values set to false. I used them inside the effects conditions in order to disable them. I also decided to remove the implementation for the depth of field, which I already implemented in the Assignment 2 which I submitted.

I made these changes to increase the speed of ray tracing.

With these changes ray tracing of all .off files is possible.

5 Results

5.1 Cube



```
// Enabled effects:
struct CONFIG {
    bool shadow = true;
    bool reflection = true;
    bool refraction = true;
} options;
```

Figure 1: Options enabled.

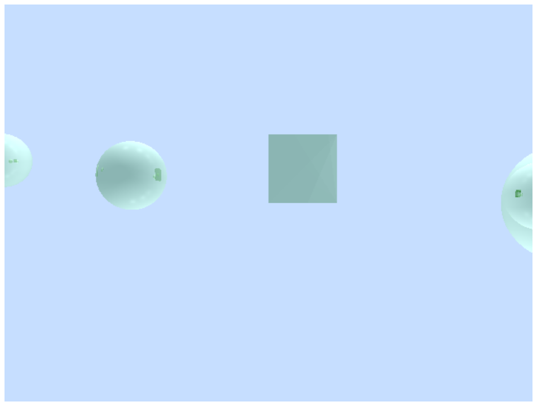
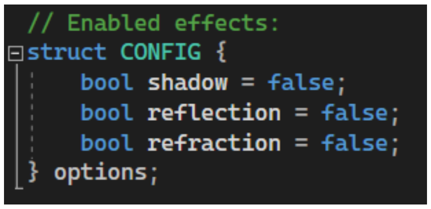


Figure 2: Cube

5.2 Dodecahedron



```
// Enabled effects:
struct CONFIG {
    bool shadow = false;
    bool reflection = false;
    bool refraction = false;
} options;
```

Figure 3: Options disabled.

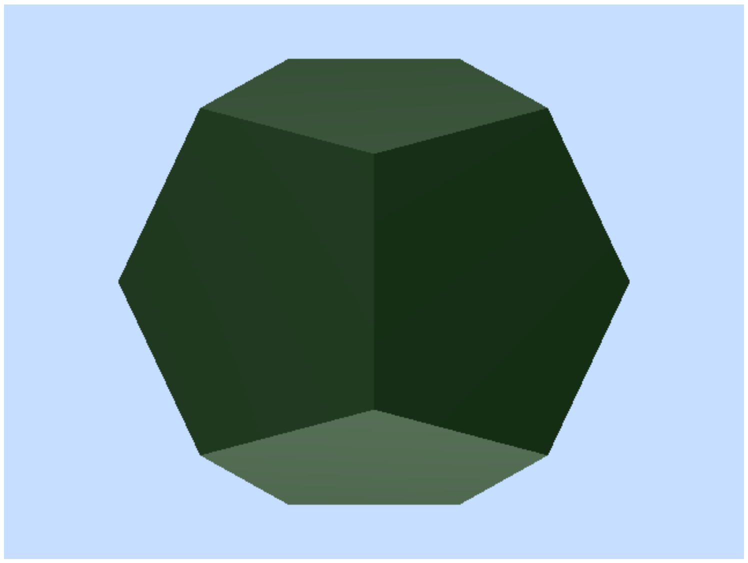


Figure 4: Options disabled.

5.3 Bunny

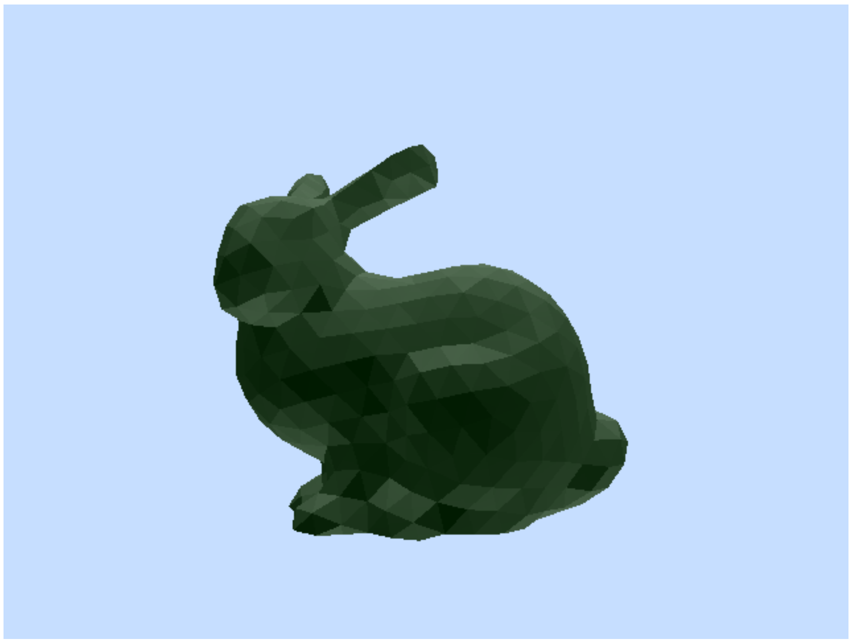


Figure 5: Options disabled.

5.4 Dragon

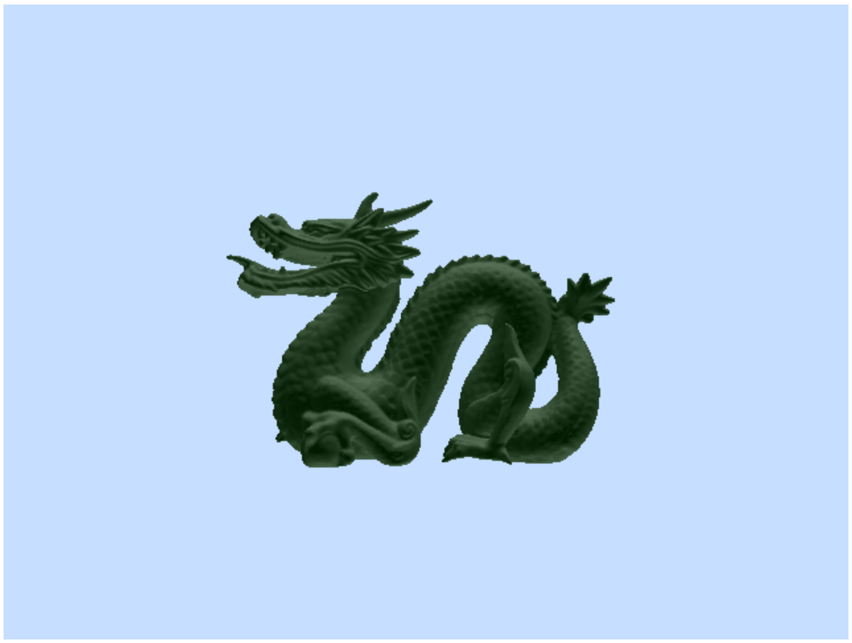


Figure 6: Options disabled.