UNIVERSITÄT DES SAARLANDES
Prof. Dr.-Ing. Philipp Slusallek
Computer Graphics Group
Stefan Lemme (lemme@cg.uni-saarland.de)

12. November 2018

Introduction to Computer Graphics
Assignment 3

Submission deadline for the exercises: 19. November 2018

The paper copies for the theoretical parts of the assignments will be collected at the beginning of the lecture on the due date. The programming parts must instead be marked as release before the beginning of the lecture on the due date.
The code submitted for the programming part of the assignments is required to reproduce the provided reference images. The submission ought to tag the respective commit in your group's git repository as well as attach the mandatory generated images to the release. The submission should also contain a creative image show-casing all extra-credit features that have been implemented.
The projects are expected to compile and work out of the box. A successful build by Drone CI is a good indicator. If it fails to do so on our Windows or Linux computers, we will try to do so on the CIP-pool students' lab as a "fallback".
To pass the course you need for every assignment at least 50% of the points.

3.1 Tree building (20 Points)

Suppose that we have devised a subdivision scheme where each node of a hierarchical acceleration structure is constructed in linear time with respect to the number of objects provided as input to the node. Considering a perfect tree with constant arity, derive an expression of the asymptotic complexity of constructing the whole tree as a function of the number of objects and the arity over which the hierarchy is being built.

3.2 Bounding boxes (8 Points)

Geometrical objects can have various complex shapes. When reasoning about them (intersecting with a ray, transforming, using in a bigger structure, etc...) it is often useful to reduce it to a simple shape, and access the precise data only when needed. One common shape is an axis-aligned bounding box.
In this assignment we ask you to:

- Implement a bounding box class (`BBox`)
- Compute a bounding box for each primitive (`Primitive::getBounds()`).

A `BBox` is an axis-aligned bounding box, defined by two of its opposing corners `min` and `max` . For each dimension i, $min_i \leq max_i$, otherwise the box is "negative" and no ray intersection will succeed. A bounding box is used to define bounds of a set of objects.
A bounding box is extensible by other points and bounding boxes. `BBox::extend` should enlarge the box to encapsulate the argument object.
Moreover, a bounding box can be intersected by a ray. `BBox::intersect` should return two values t_1, t_2, which when plugged into the input ray equation $o + dt$ should give the enter and exit points of the ray/box intersection. $t_1 > t_2$ indicates that the ray missed the box. Negative values of t are permitted.
In addition, `BBox` provides the following functionality:

- `empty()` static function creates a new empty `BBox` . Intersecting an empty box should always fail and when an empty box is extended by another object, it should match the bounds of that object.

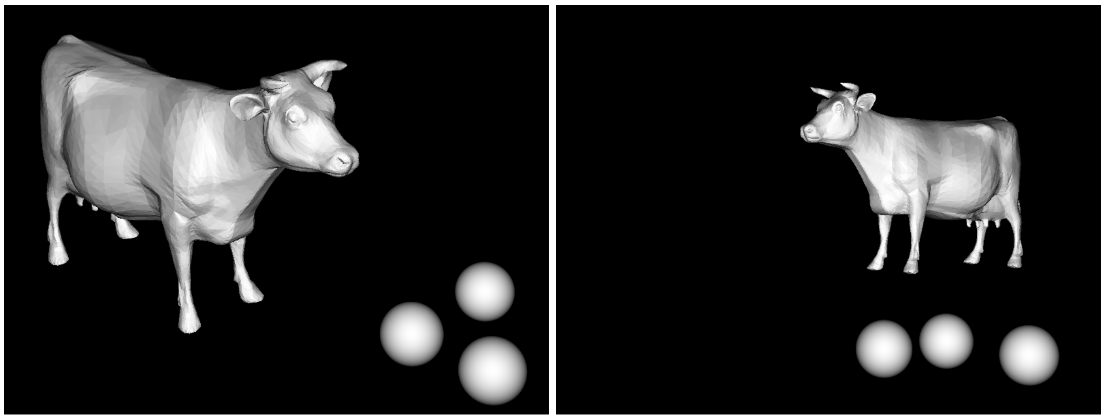

Figure 1: The cow model used in this assignment.

- `full ()` static function creates a "full" `BBox` , spanning the whole space. Intersecting the full box should always succeed.
- `isUnbound ()` checks if the box is unbound in at least one dimension (e.g. the `Box ::full()` is unbound)

Each primitive should now compute a bounding box which encapsulates all the geometry it represents. For simple `Solid` s the box can be computed analytically. For `Groups` you need to extend the box by all the members of the group.
If for some primitives (e.g. quadric solid) the exact bounds are hard to compute, you may compute a bigger, conservative bounding box. However, for `AABox` , `Sphere` , `Triangle` , `Quad` we want exact bounds as otherwise you may face performance drops.

3.3 Acceleration Structures (40 + 40 Points)

We are providing you an obj file loader (Wavefront format), and a simple cow scene as a model example. The scene contains over 34000 triangles. Trying to intersect each of them with every ray you shoot can be time consuming. For that reason your task is to implement a BVH acceleration structure.
The `BVH` is a derivative of `Group` class.
You will need to implement two major things:

- **The builder**. You can assume that once all the primitives are added into your group, `rebuildIndex ()` will be called exactly once. At that point your structure should be build in a top-down recursive manner.

  Since any geometry can be added to your class, you should not assume that you have primitives of only one kind (e.g. triangles). Instead, you should rely on the bounding boxes of the primitives.

  In each step you will need to find a split plane using a simple split-in-the-middle heuristic, along the longest dimension. Then, your primitives should be distributed between the left and right children. Once the children are created, you need to recursively perform the same operation on each of them. You can terminate your recursion when you have less than 3 primitives in your node.

- **The intersection**. When a ray intersects your structure, you should not try to intersect the ray with every primitive contained in the group. Instead, the ray should traverse the tree structure, ignore the nodes that are completely missed and intersect only primitives when a leaf node is reached. Moreover, when a primitive is finally hit, all intersections which are performed behind the hit point can be ignored (`previousBestDistance` parameter of the intersection function can be of help to control this)

If the structure is working as intended, it should take less than a minute (probably just few seconds on stronger machines) to render the cow.

3.4 Surface Area Heuristic (15 + 5 Points)*

Incorporate the Surface Area Heuristic when building your BVH. Compare the rendering time between the split-in-the-middle and SAH.
Find other interesting and free obj models on the Internet and try to render them.
Include the produced images of your models and the rendering times with the submission. We do not need the obj files themselves.