UNIVERSITÄT TÜBINGEN
Prof. Dr.-Ing. Hendrik P.A. Lensch
Lehrstuhl Computergrafik
Dennis Bukenberger (dennis.bukenberger@uni-tuebingen.de)
Zabiha Kehribar (zabiha.kehribar@student.uni-tuebingen.de)

16. November 2020

Graphische Datenverarbeitung
Assignment 2

Submission deadline for the exercises: 23. November 2020 6.00 am

Source Code Solutions

- Upload **only** the source code files listed in the description in Ilias.
- Upload them one by one and **don't** zip them.

The source code must run on `cgpool120[0-7].informatik.uni-tuebingen.de` by extracting your submitted `.tar.gz` file to a certain folder and running `scons`. You can log onto this machine via ssh by using your WSI account. From outside the WSI network, you may have to use a ssh gateway, e.g. `cgcontact.informatik.uni-tuebingen.de`
Attention: Do not use `cgcontact` for working, but only for ssh-ing to the `cgpool` machines.

The framework you get for completion already compiles and runs as requested above and you only have to modify source and header files - no files have to be created.

Written Solutions

Written solutions have to be submitted digitally as one PDF file via Ilias.

2.1 Blender (0 Points)

It will be convenient to use a 3D modeling software for future assignments. Make yourself familiar with *blender*, a freely available 3D animation program (`www.blender.org`).
In the framework for this assignment you will find an `io_mesh_ra2` folder. Copy this folder into your `~/.config/blender/2.79/scripts/addons/` directory (in Windows you may use the `scripts` folder in the blender installation directory).
After calling blender, you must enable the exporter add-on at `File` → `User Preferences` → `Add-Ons` → `Import-Export RA2 exporter`. Be careful: user preferences are stored per blender file. Pressing the `Safe As Default` button in the user preferences dialog overwrites the default blender file for initialization with the currently openend blender file including the current scene and all other settings.
To get familiar with how to export data from blender, it might be a good idea to have a look at the supplied `io_mesh_ra2/ra2_export.py` file. The `__init__.py` file contains the registration of the exporter as plugin and in blender's GUI.

2.2 Implementation of a BVH (0 + 20 + 40 + 40 = 100 Points)

Download the new framework from the course website. It contains a sample solution of the last assignment and implements the base methods for this task. So far, the ray tracer implementation has used no acceleration structure for reducing the number of ray/primitive intersections. This was simple to implement and worked relatively well. Unfortunately, this is of course not practical for larger scenes. As such,

you need a data structure to speed up the process of finding the first hit of a ray with the primitives. Therefore, in this exercise you will implement a BVH (Bounding Volume Hierarchy) construction and traversal. Proceed as follows:

a) Have a look at the files `src/bvh.{h,cpp}` in the framework. Especially the `struct Node` and `struct BVH` are important. Keep in mind that a BVH is an object list partitioning method (as opposed to spatial hierarchies as the kd-tree). This means that the object list is split up and assigned to leaf nodes in the tree without reference duplication. To enable efficient sorting without shifting too much memory, only the index array `Node::indices` is sorted. So to access the first triangle in a given leaf node, you would use `bvh->tris[bvh->indices[node->triIndex]]`.

b) Implement a `BVH::findSplitPlane(...)` method, which should find a location of the splitting plane according to the current bounding box. As simple heuristic use the middle of the longest side of the current bounding box.

c) Implement the method `BVH::buildBVH(int nodeIndex, int triIndex, int numTris, int depth)`, which should set up the BVH by subdividing the nodes and putting the primitives into the corresponding child nodes. As soon as you have reached a maximum depth, or you have less than a minimum number of primitives (e.g. 4), stop subdividing and generate a leaf node. Otherwise, split your current tree node into two child nodes and process them recursively. The subdivision is started with a list of all primitives and an initial recursion depth of 0. Put the primitives based on the `findSplitPlane(...)` method into the appropriate child node. The tree should only contain the primitives in its leaf nodes.

d) For traversal implement the `BVH::intersect(const Ray &ray)` method. Use a recursive or an iterative algorithm. For leaf nodes intersect the ray with each primitive contained by the node to find the intersection.

When your implementation was successful, you should encounter a big increase in performance.

Instead of optimizing too much, rather concentrate on a stable, bug-free implementation.

If you are stuck come by and ask questions!