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)

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

9. NOVEMBER 2020

GRAPHISCHE DATENVERARBEITUNG
ASSIGNMENT 1

Submission deadline for the exercises: 16. 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.

1.1 Primary Ray-Generation for a Perspective Camera Model (written, 30 Points)

A *Perspective Camera Model* can be defined by the following parameters:

- Camera origin (center of projection) $\mathbf{o}$
- Viewing direction of the camera $\mathbf{d}$
- Up-vector $\mathbf{u}$
- Focal length f which is the distance to the image plane in pixels
- Image resolution $resX \times resY$

Given the above camera description, derive the ray direction $\mathbf{d_r}$ for given pixel coordinates x, y (e.g. $128.5, 5.5$ through the center of the pixel). Pixels are squared and the projection plane is perpendicular to the viewing direction $\mathbf{d}$. Please incorporate all the variables that are listed above (except $\mathbf{o}$).

1.2 Ray-Surface Intersection (written, 30 Points)

Given a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ with origin $\mathbf{o} = (o_x, o_y, o_z)$ and direction $\mathbf{d} = (d_x, d_y, d_z)$, derive the equations to compute the parameter t for the intersection point(s) of the ray and the following surfaces:

a) An infinite plane $(\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} = 0$ through point $\mathbf{a} = (a_x, a_y, a_z)$ with surface normal $\mathbf{n} = (n_x, n_y, n_z)$, where any point $\mathbf{p} = (x, y, z)$ that satisfies the equation lies on the surface.

b) An axis aligned bounding box represented by the vectors $\mathbf{min} = (x_{min}, y_{min}, z_{min})$ and $\mathbf{max} = (x_{max}, y_{max}, z_{max})$. Compute the values of t for which the ray intersects the bounding box and find the nearest one.

c) A triangle represented by three vertices $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$. If the intersection is inside the triangle, find the corresponding value t. Use either the method presented in the lecture or find another one which is suitable for a later implementation.

1.3 Implementation of a Minimal Ray Tracing System (40 Points)

A basic ray tracing system consists of these parts:

- Primary ray generation to generate the rays cast from a virtual camera into the scene,
- ray tracing to find either the closest intersection of a ray with the scene or to check the visibility,
- shading to calculate the color.

In this exercise you will build a minimal ray tracing system by implementing these three tasks. You are provided with a basic framework intended for running it the same way as the test framework of exercise00. When you build and start this framework without modifications, it will do some simplified ray tracing and print some console output.

In this exercise your task is to implement the following features (watch out for the `TODO`s in the code):

a) **Bounding Box Computation**. Compute the axis aligned bounding box of the Test.ra2 scene.

b) **Ray Tracing**. Implement intersection routines for triangle and bounding box intersection (see `rtStructs.h`).

- there is a simple parallel projection camera implemented with which you can visually debug the intersection routines (see `cam.h`).

c) **Primary Ray Generation**. Implement your camera model you have derived in 1.1.

d) Use diffuse shading to shade the pixels.