TU BRAUNSCHWEIG
Prof. Dr.-Ing. Marcus Magnor
Institut für Computergraphik
Contact: cgg@cg.cs.tu-bs.de

October 23, 2019

# Computer Graphics WS 19/20
## Assignment 1

The exercises will take place in room G40 in Mühlenpfordtstrasse 23. Your y-account is sufficient to login and access all tools. `Ctrl+Alt+T` gives you a terminal and `g++` is your GNU C++ compiler. This project offers a `CMake` configuration to generate and executable.

Throughout the course you will implement your own minimal raytracer. In each exercise you will extend your raytracer a little further. To make the task easier, you are provided with a basic raytracing framework so that you just have to **fill in** the missing core parts. You may use your own computer to solve the exercises, but your final program **must** run on the machines in the CIP pool.

Each week you must complete the assignments and hand in your *commented* source code for the practical tasks, as well as your solutions to the theoretical tasks (with drawings/formulas). Please use different colors in your drawings and also make sure that formulas are recognizable in your source code. Be prepared to present the completed assignments on **Friday, 9:45**.
To keep presentation time short, make sure that the last commit contains the original scene file which generates the results shown below.

---

At the start, your raytracer consists of only three simple parts:

- **Primary Ray Generation** for generating the rays to be cast from a virtual camera into the scene.
- **Ray Tracing** for finding the (closest) intersection of a ray with the scene to be rendered.
- **Shading** for calculating the *color* of the ray.

To begin, create an account on our git `git.cg.cs.tu-bs.de` and tell me your account name so I can give you access rights Have a look at the ray tracing framework in repository `WS1920` and its C++ classes:

- The framework is structured into the components `Camera`, `Light`, `Primitive`, `Renderer`, `Scene`, and `Shader`. Each has a base class of the same name, as well as multiple child classes that we will be developing over the course of this semester.
- A `Ray` is defined by its origin (`Vector3d`), direction (`Vector3d`), and length (`float`).
- The `Scene` holds all the geometry in the form of `Primitives`. Each type of `Primitive` has a virtual method `Primitive::intersect(Ray & ray)`, which has to be implemented by you.
- The abstract base class `Camera` handles camera parameters. For each derived class, e.g. a perspective or orthogonal camera, the pure virtual method `Camera::castRay(float x, float y)` has to be implemented. Here, `x` and `y` specify the relative position in the camera frustum.
- In the class `SimpleRenderer` you will have to implement the function `SimpleRenderer::renderImage(Scene const& scene, Camera const& camera, int width, in height)` . This function calculates the images aspect ratio and casts a ray for each pixel.

Before implementing anything read through the presented classes and `ex1.cpp`.