Computer Graphics WS 16/17
Assignment 2

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.

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. Your group must present the completed assignments on each Friday, 9:45.

2.1 Environment Map (20 Points)

In environment mapping, the scene is surrounded by a closed three dimensional surface onto which the environment is projected. Reflected rays are traced from the object, hit the surface and then index onto the map. For a given direction vector \(r\) (\(\text{ray.direction}\)) calculate the mapping from the polar coordinates \((\phi, \theta)\) to the texture coordinates \((u, v)\). Keep in mind that \(\phi = \arccos(r_y)\) and \(\theta = \arctan2(r_z, r_x)\).

Have a look at \texttt{Scene::traceRay(Ray * const)} in \texttt{scene/scene.cpp} and implement the missing part.
2.2 Primitive Normals (10 Points)

The normal vector, often simply called the “normal” to a surface is a vector which is perpendicular to the surface at a given point. Implement `normalFromRay(Ray const& ray)` in `primitive/infiniteplane.cpp`, `primitive/sphere.cpp` and `primitive/triangle.cpp`. You will need this for all of the following shaders.

2.3 Refraction (25 Points)

Derive the new direction \( \vec{t} \) of a ray \( \vec{r} \) that hits the surface with normal \( \vec{n} \) between the two media with index \( \eta_1 \) and \( \eta_2 \) according to Snell’s Law. The angles \( \theta \) and \( \phi \) have to be calculated. Take the total internal reflection into account. When does it occur?

![Diagram of refraction](image)

Derive the formula and use this to implement `RefractionShader::shade(Ray * ray)` in `shader/refractionshader.cpp`. Write your derived formula as a comment in the source code. A shader can send out a secondary ray using: `this->parentScene->traceRay(ray);`

2.4 Mirror Shader (20 Points)

Given a ray \( r(t) = \vec{o} + t \cdot \vec{d} \) which hits a reflective surface at \( t = t_{hit} \), the surface has the normal \( \vec{n} \) at the hit point. Derive the formula for the reflection ray and use this to implement `MirrorShader::shade(Ray * ray)` in `shader/mirrorshader.cpp`. Write your derived formula as a comment in the source code.

2.5 Shadow Ray (20 Points)

If a ray hits an object, we want to know if that point on the object is in a shadow. A secondary ray, called a “shadow ray”, is sent from the object to each light source. If this shadow ray intersects another object before it hits the light source, then the point is in the shadow.

Have a look at `SimpleShadowShader::shade(Ray * ray)` in `shader/simpleshadowshader.cpp` and implement the missing part. Make sure to first have a look at the implementation of `PointLight::illuminate(Ray *ray)` in `light/pointlight.cpp`. 
2.6 Lambert Shader (25 Points)

Create a new LambertShader using what you have learned in the lecture. Take a look at shader/lambertshader.h and shader/lambertshader.cpp.

If you implemented everything correctly, you should get the following results for the test scene:

![Test Scene Image]

2.7 Useful Stuff

Have a look at the following links. They may help you solving the single Tasks.

- Qt Creator https://www.qt.io/ide/
- Realistic Ray Tracing, Peter Shirly.