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

19.11.2021

Computer Graphics WS 20/21
Assignment 2

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.

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** in your given time slot. To keep presentation time short, make sure that the last commit contains the original scene file which generates the results shown below.

2.1 Ray-Surface Intersection Part II (5 Points)

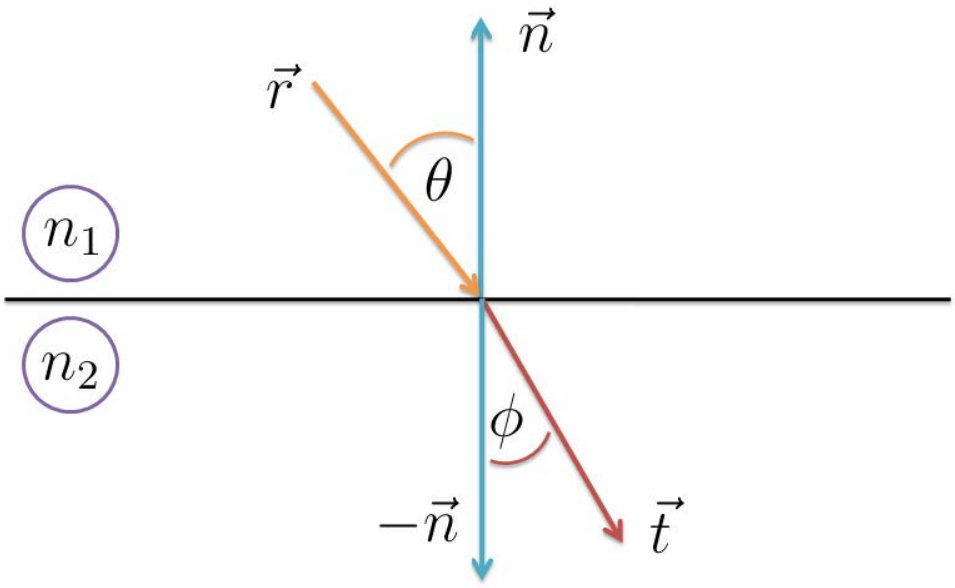
Implement `Box::intersect` in `primitive/box.cpp` using what you have learned in the lecture.

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 the missing part in `primitive/infiniteplane.cpp`, `primitive/sphere.cpp`, `primitive/box.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 η_1 and η_2 according to Snell's Law. The angles θ and ϕ have to be calculated. Take the total internal reflection into account. When does it occur?
Derive the formula and use this to implement `RefractionShader::shade` in `shader/refractionshader.cpp`. Write your derived formula as a comment in the source code. A shader can send out a secondary ray using: `scene.traceRay(ray);`

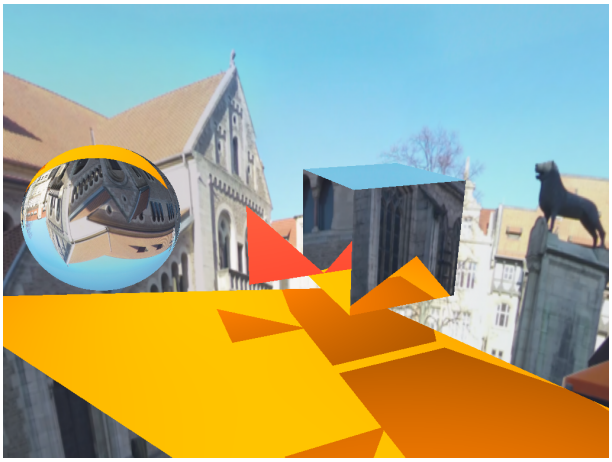


2.4 Mirror Shader (20 Points)

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` in `shader/mirrorshader.cpp`. Write your derived formula as a comment in the source code.

2.5 Shadow Ray and Point Lights (40 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(Scene const & scene, Ray & ray)` in `shader/simpleshadowshader.cpp` and `PointLight::illuminate(Scene const & scene, Ray & ray)` in `light/pointlight.cpp`. Implement the missing parts. A point light in our implementation should be a light source, which radiates light in all directions with quadratic intensity falloff. In other words, let d_p be the distance of an unoccluded point p to the point light, i the intensity of the point light and c the color of the point light, then the intensity (or "color") c_p reaching p is $c_p = \frac{i}{d_p^2} c$.



2.6 *Useful Stuff*

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

- CMake `https://cmake.org/`
- `http://www.khanacademy.org/#LinearAlgebra`. Introductory videos to linear algebra concepts like the dot and cross product.
- Realistic Ray Tracing, *Peter Shirly*.
- 3D Modelling: Blender `https://www.blender.org/` or 3ds Max Free Student Version `http://www.autodesk.com/education/free-software/3ds-max`