5.1 A simple speed up (20 Points)

You have noticed that your raytracer has become significantly slower due to the large number of primitives added by the mesh loader. In order to speed up computation a little, we want to introduce bounding boxes around our loaded objects. This way we can quickly check if a ray intersects the bounding box and only then test it against all the primitives in the object.

Write a new class ObjModel in primitive/objmodel.h and primitive/objmodel.cpp, which can load an .obj-File using Scene::loadObj and compute its axis-aligned bounding box using an instance of primitive/box.cpp. In ObjModel::intersect check for intersection with the bounding box and if a hit has been detected, check all primitives in the model for an intersection.

5.2 Phong Shader (20 Points)

Implement a Phong shader in shader/phongshader.cpp. Use the definition of the original model, not its physically plausible extension from the lecture.

5.3 Cook-Torrance Shader (40 Points)

Implement the Cook-Torrance shader using what you have learned in the lecture. Look at shader/cooktorrance.cpp and implement the missing parts.

Use Schlick’s approximation for the Fresnel term and Beckmann’s distribution as the distribution function.

If you have implemented everything correctly, your result should look like this:
5.4 Smooth Triangles (20 Points)

You may have noticed that each vertex in `Scene::addObj` gets assigned a normal vector, if it is present in the `.obj`-file. This normal can be used for smooth shading. This removes the faceted look from our models.

Instead of returning the plane normal of a triangle return the interpolated normal vector of the intersection point in `Triangle::intersect`. Default back to the usual behavior, if one of the normal vectors of the triangle is not set (i.e is the zero vector). Use the barycentric coordinates $u$ and $v$ for interpolation and remember that an interpolated normal is not necessarily normalized.

If everything is implemented correctly, your result should look like this: