ETH zürich

computer graphics laboratory

Prof. Markus Gross

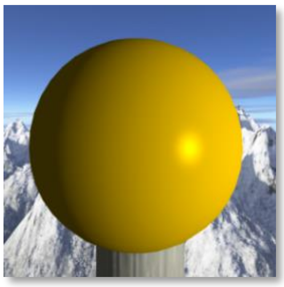
Visual Computing
Exercise 10: Lighting and Shading

Hand-out Date: 2nd December, 2022

Goals

- Understand the Phong reflection model
- Learn how to use textures in a fragment shader
- Implement reflection mapping and normal mapping

Exercise 1) Phong Reflection Model



The Phong reflection model computes a diffuse term, which is independent of the position of the viewer, and a viewer-dependent specular term. In this exercise, you compute these two reflection terms in a fragment shader.

(a) Phong Material Shader

Implement the function `computeDiffuseSpecIntens` in `phong.glsl` which computes the amount of diffuse and specular light, given the following quantities in world coordinates: position of the pixel being rendered, normal vector at the pixel, position of the camera, and position of the light source.

Exercise 2) Reflective Material

Materials with a perfectly reflective component can be simulated efficiently under the assumption that the light being reflected originates at infinity. Under this assumption, a cube map can be used as a lookup table, storing for each direction the light coming from that direction.

(a) Texture Lookup

Copy your implementation of `computeDiffuseSpecIntens` to `texturereflection.glslf`. Then implement the missing code to read the texture color (`colorTex`) from the texture sampler `txtColor` using the `texture2D` command.

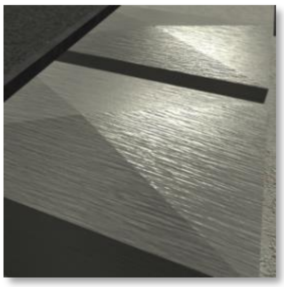
(b) Reflection Shader

Implement reflection mapping in `texturereflection.glslf` by reflecting the camera-to-pixel vector at the normal and using the reflected direction for a texture lookup into the cube map (`txtEnvMap`). Use the `textureCube` command to read the texture value.

Optionally, improve realism by computing the reflection term (`reflTerm`), approximating the so-called *Fresnel* term. This term controls the amount of reflection. When looking straight at the object (in the direction of the surface normal vector), there should be little reflection. When looking at an oblique angle (perpendicular to the normal), there should be most reflection.

Exercise 3) Normal Mapping Material

(a) New Material Class



In `AdvancedMaterial.h` and `AdvancedMaterial.cpp`, create a new material class named `TextureNormalMaterial`. As a starting point, you can create a copy of the existing `TextureReflectionMaterial` material. The material should have a method `setNormalMap` to set the 2D texture containing the normal map. In `MaterialWindow::LoadMaterials`, create an instance of the material and add it to the collection of available materials using `addMaterial`. Edit `ETHTower::draw` to retrieve the material from the 'world' and draw the models using the material.

(b) Normal Mapping Shader

The framework for the normal mapping shaders already exists in `texturenormal.glslv` and `texturenormal.glslf`. Implement the vertex shader's main method. In the fragment shader, insert the Phong shading code from Exercise 1 to get proper illumination. Fill in the missing code for the computation of the perturbed normal vector. Each (r, g, b) pixel of the normal map is interpreted as a vector $(n_t, n_b, n_n)^T$ defining the direction of the (perturbed) normal vector in the local (tangent, bitangent, normal) coordinate system. Note that the color values $(r, g, b) \in [0, 1]$ must be scaled to $[-1, 1]$ range to get$(n_t, n_b, n_n)^T$.

Exercise 4) Skybox

The geometry for the skybox object (class Skybox) consists of a single triangle (see method `buildGeometry`). Have a look at the shaders for the skybox (files `skybox.glslv` and `skybox.glslf`) and try to explain how (and why) this works:

- Why is the single triangle always visible, filling the whole screen independently of where the camera looks at?
- Why does it still look like the 'skybox triangle' is an object that surrounds the entire scene?

ETH zürich

computer graphics laboratory

- Why does the rendered sky not change when translating the camera without rotating it?