Textures

The framework has been updated to support textures. Every `Material` can now have the following textures assigned to it:

```
struct Textures {
    Texture albedo{};
    Texture normal{};
    Texture roughness{};
    Texture displacement{};
} textures{};
```

We will use them in the following exercises to describe the color of our material (using the `albedo`), describe their small-scale surface orientation using the `normal` map, describe their micro-scale surface orientation with the GGX microfacet distribution parameterized by the `roughness`, resulting in glossy appearance, and we describe a small-scale displacement of the geometry towards its geometric normal using `displacement` maps.

7.1 Normal (Bump) Mapping (25 + 25 = 50 Points)

Bump mapping describes a technique where we use a `normal` texture to model detailed surface normals on a coarse mesh. Every pixel in the normal map describes how to rotate the normal in its tangent space before computing the shading.
In order to correctly apply the normal from the texture, we first need to compute the local tangent s and bi-tangent t for each triangle based on its texture coordinates.
The following equalities are our starting point:

$$x_2 - x_1 = (u_2 - u_1) \cdot s + (x_2 - x_1) \cdot t$$
$$x_3 - x_1 = (u_3 - u_1) \cdot s + (x_3 - x_1) \cdot t,$$

where x_1, x_2, x_3 are the triangle points, and u and v are the texture coordinates at each point.
We can solve these for s and t to get the following:

$$s = \frac{(x_2 - x_1) \cdot (v_3 - v_1) - (x_3 - x_1) \cdot (v_2 - v_1)}{(u_2 - u_1) \cdot (v_3 - v_1) - (v_2 - v_1) \cdot (u_3 - u_1)},$$
$$t = \frac{(x_3 - x_1) \cdot (u_2 - u_1) - (x_2 - x_1) \cdot (u_3 - u_1)}{(u_2 - u_1) \cdot (v_3 - v_1) - (v_2 - v_1) \cdot (u_3 - u_1)}.$$

Note that both share the same denominator.

a) Implement the computation of the tangent and bi-tangent vector in the geometry shader. Your inputs are the texture coordinates `vsTexCoords` and world space positions `vsWSPosition` from the vertex shader (or tesselation evaluation shader – more on that in the next task).

You're already given a somewhat working dummy implementation which places the tangent in the xz-plane. This works for some cases, but might not always be consistent with the orientation the texture expects.

b) Once you have your tangent and bi-tangent vectors, lookup the normal map texture `normalMap` at the given texture coordinates, re-scale its colors from the $[0, 1]$ interval to $[-1, 1]$ and compute the normal as $s \cdot r + t \cdot g + n \cdot b$. Normalize the result.

You should now see shadows matching the small-scale geometry simulated by the normal map.

7.2 Displacement Mapping (25 Points)

With displacement mapping, we can not just change the shading normal, but can instead move the actual geometry of our 3D models into specific directions. Here, we will only use a scalar displacement map to move geometry towards the surface normal.
To get some detailed results, we need more detailed geometry than our input meshes would usually provide. To get more detail on-demand, we can use tesselation.
The necessary tesselation control and tesselation evaluation shaders are already implemented. You only have to add a few final steps in the geometry shader.

a) Lookup the displacement from the displacement map texture `displacementMap` and subtract 0.5 (to allow for negative displacements). Scale with the `uniform float displacementScale` and the `normal` and add the resulting displacement vector to the world space position `wsPosition`. Project the result using the view projection matrix `vp` and assign it to the `gl_Position`.

7.3 Shadow Maps (25 Points)

The final big feature to implement this week is shadow mapping. For shadow mapping, we need two render passes: One for computing the shadow map from the light's point of view and one for computing the final image.
When rendering the shadow map, all we care about is the depth buffer (or z-buffer). It tells us for every point in the image, how far away it is from the light source, if we use the light source as our camera position. During the final render pass, we can, for every point in the scene, again project into the view of the light source and compare the depth to the value stored in the depth buffer we computed before.

For simplicity, we don't apply any displacement mapping in this render pass, so shadows on displaced geometry might be lacking a bit. For point lights, one would also usually render an entire cube map to get shadows in all directions. We keep things a bit simpler and use a projective camera looking down just below the scene's center. This way, we will never get shadows on the ceiling, but that is ok for now.

OpenGL will already do the shadow map computation for us if we use the special shadow sampler for the shadow map: `uniform sampler2DShadow shadowMap`. All points that are further away than the positions stored in the depth buffer are occluded and, thus, the shadow sampler returns black. If there is no occluder in the way, it returns white.

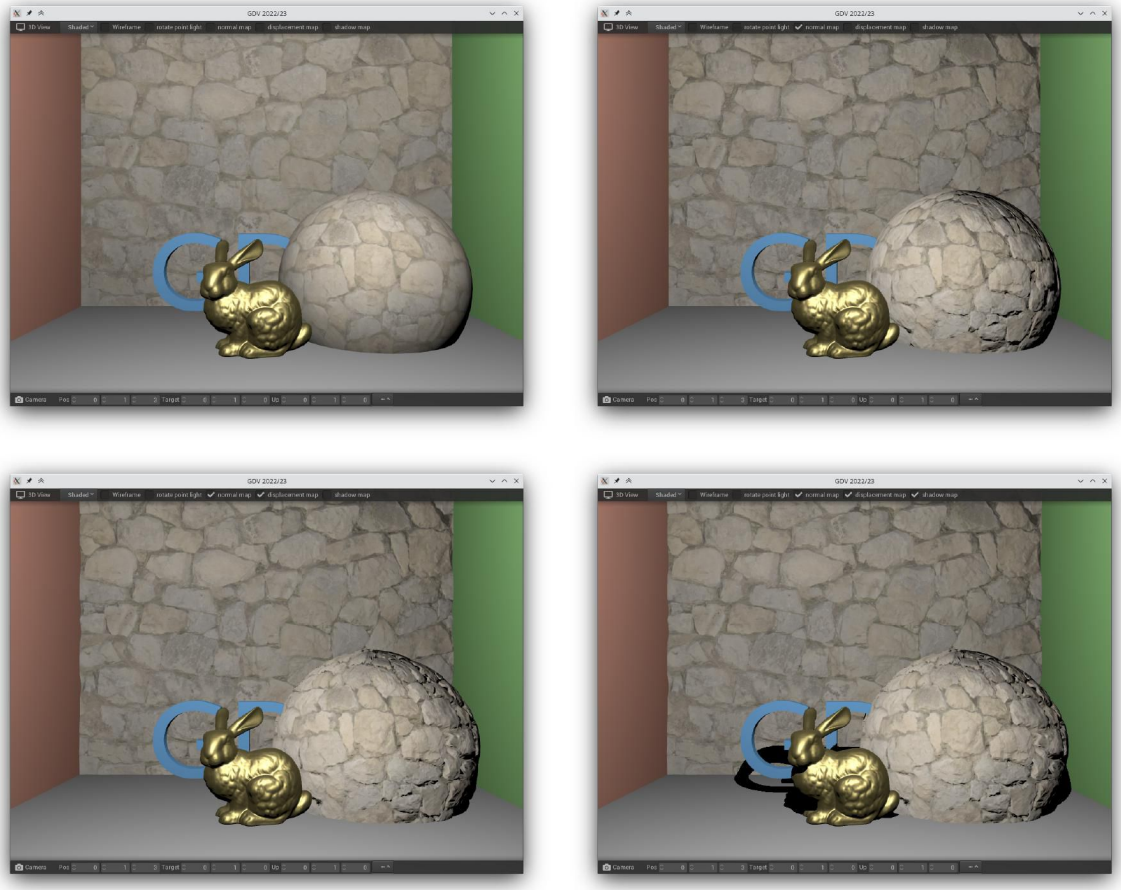

Figure 1: Base version, normal mapping, displacement mapping and shadow mapping.

As input to the fragment shader, you're already getting the position in the light's space. For your lookup, make sure to divide by its w component, or your shadows might be missing or appear projected.
Also scale the depth for your comparison with $1 - \varepsilon$ (`1.0f-epsilon`) to avoid self-shadowing. You additionally have to account for the different coordinate systems used for the viewport ($[-1, 1]^2$) and the texture coordinates ($[0, 1]^2$).

a) Implement the fragment shader part of shadow mapping:

Do the shadow texture lookup at the correct coordinate with an appropriate depth value. Make sure to ignore cases where the depth value is beyond the light space's far plane.

For points outside the light's view, we're already clamping the shadow map to a white border.

You can see the effects in the final program in Figure 1. This time, you have to implement shader code. Instead of wrapping everything into a single `.cpp` file, the shader code is in the folder `src/shaders`. Submit (only) the entire `shaders` folder when you are done with the exercises.