TU Clausthal

GPU Programming **WiSe16/17**
Prof. Thorsten Grosch, Johannes Jendersie

Exercise 3: Shader Effects

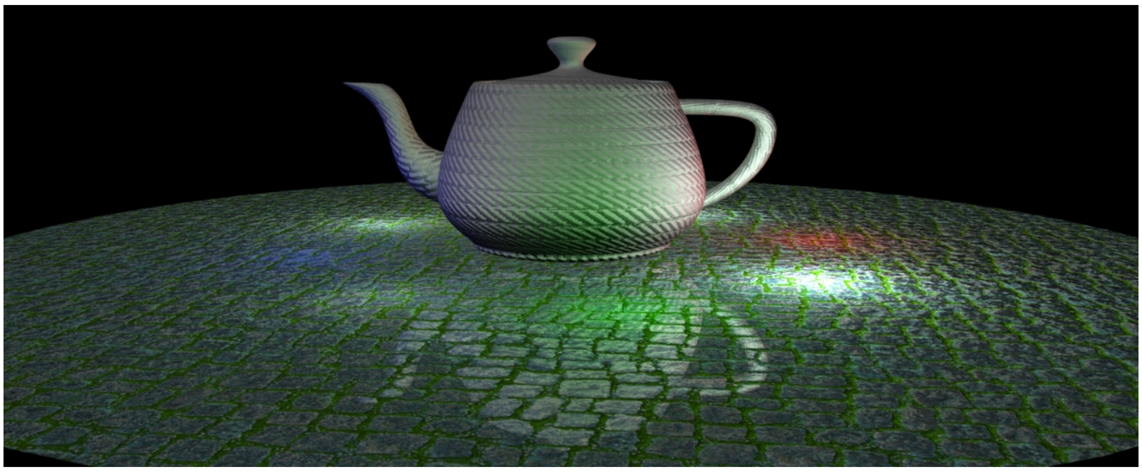

Figure 1: Final scene with normal mapping, multiple light sources and a reflection of the teapot on the ground.

Task 1: Transformation shader **1P**

On the first run the entire scene will look broken, because the OpenGL-default state has the z-test disabled. Enable z-test and backface culling in `objectShadingWithSwirlPipe`.

To get in contact with the first shader we want to do a transformation on the teapot. Rotate the vertices in *`swirl.vert`* by a time varying angle. The 2D rotation matrix

$$\begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \tag{1}$$

will probably help you.

Task 2: Lighting **1P**

There are eight animated point light sources which are already given through an uniform buffer. However, lighting direction and intensity are currently not computed in *`shading.frag`*. Fill in these two computations to complete the light sources.

Task 3: Normal Mapping **1P**

Normal mapping is a technique which adds local geometry details based on a texture. The texture provides normals which must be transformed into world space using the tangent frame at the pixel's position.

The tangent space is already provided in *`swirl.vert`* and *`simple.vert`*, but not passed through. Add the output declarations and pass the transformed tangent and bitangent vectors into them. Also, add the two vectors as inputs to the fragment stage *shading.frag*. Once the tangent and bitangent are available in the pixel shader you can implement the normal mapping itself: Fetch a local normal from the normal texture and transform it into that tangent space.

Task 4: Stencil Buffer Reflections 4P

To mirror the scene we simply render it again with a reflected transformation matrix. However, to combine everything properly there are some additional steps. The ground plane must be drawn into the stencil buffer, to make sure the reflection does not appear outside that plane.

Therefore, setup `setStencilPipe` such that the Stencil buffer is set to 1 and that there are no depth or color writes. Use this pipeline to draw the ground plane, then render the scene reflected using `objectShadingWithSwirlMaskedPipe`. You will not see anything, because the final ground plane rendering will overwrite the results again. To solve that change the `planeShadingPipe`'s blending state as described in the source code.

Theory

Be prepared to answer the following questions.

Q1 How is shader loading working? What is a 'program' in OpenGL?

Q2 What is `layout(location = x)` for in the vertex input declarations? What is it for at uniform variable declarations?

Q3 What is `layout(binding = x)` for?

Q4 What is culling?

Q5 How is alpha blending working? What is the blend equation?

Q6 How is the tangent space defined per triangle and vertex?

Q7 The tangent space per pixel is not orthogonal. Why is that so and how can the problem be solved?

Q8 What is the Stencil buffer?