Interactive Computer Graphics Coursework – Task 6 (*assessed)

Bernhard Kainz

December 18, 2018

Task 6a: Texture*

Given that an object has defined *uv* texture coordinates, the texturing of an object can by done automatically in hardware. Simple texture coordinates can be generated automatically by OpenGL using spherical, cubical, cylindrical, etc. mapping. However, *uv* texture coordinates for more complex objects are usually generated by an artist, *e.g.*, for computer games using specialised tools.

Your task is to apply your own texture to the test objects. You can use the texture management capabilities of the framework and define a 2D texture sampler `sampler2D` object as `uniform` varible in the *per-polygon* fragment shader. The required textures can then be set in the *User uniforms* and downloaded from CATE.

Note that `FragmentIn.texCoord` is a `vec4`. However, to access the correct pixel position in texture using the glsl function `texture(...)`, you only need the first two components of this vector, *i.e.*, `frag.texCoords.st`.

Now apply Phong illumination from Task 3 to the result of the texture lookup. The result should look like Figure 1b.

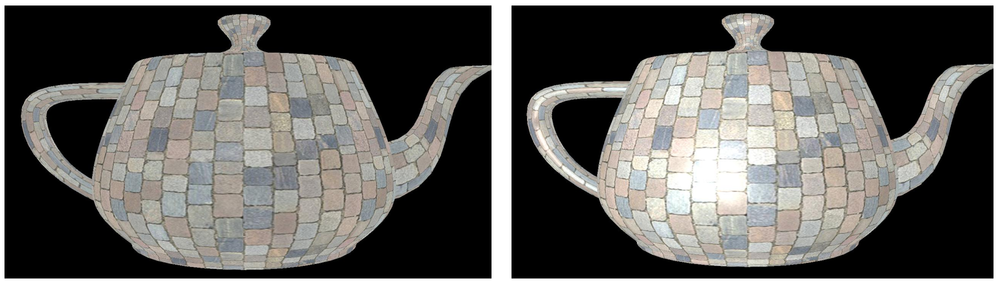
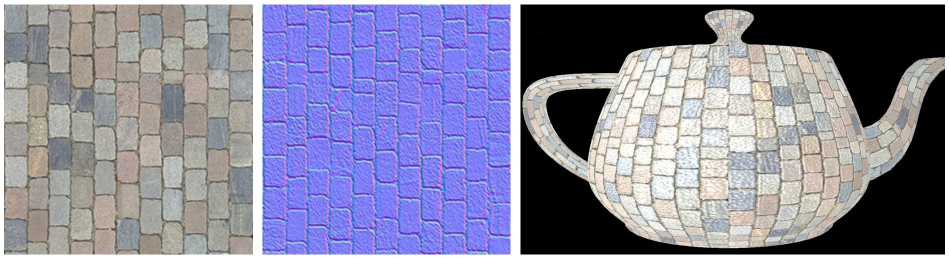

(a) texture and ambient only (b) texture and Phong

Figure 1: Textured and Phong shaded teapot.

Task 6b: Bump mapping*

Bump mapping can be used to reduce geometric complexity why generating the impression of highly tessellated surfaces. The idea of bump mapping is simply to use another lookup texture which encodes surface normals instead of RGB colour values. The normals are still encoded as RGB values but can be interpreted during the illumination step as surface normal instead of the real, from the vertex shader coming, interpolated normal.

Your task is to use a second texture sampler in the *per-polygon* fragment shader and to use one of the provided normal maps as additional input texture. Use the sampled normals in the Phong illumination model. The result should look like shown in Figure 2(c). Figure 2 shows also the used texture and normal map.



(a) texture (b) normal map (c) Bump mapping

Figure 2: Bump mapping textures and result.

Task 6c: Render to Texture*

For this task you will use the render-to-texture 2D fragment shader instead of the per-polygon shaders. These shaders are applied to a screen aligned quad that is rendered in front of the camera. The quad is textured with the scene you have been working with so far and serves as an intermediate representation to allow image-based operations.

The framework renders the scene first into a so called framebuffer. A framebuffer is basically a texture image similar to the one that you used in the previous exercise. However, this object has the additional capability to capture the output of your render window. This function is currently one of the most important ones in applied Computer Graphics because many different image procession algorithms can be applied to this 2D texture image as post processing step.

The render-to-texture 2D fragment shader and render-to-texture 2D vertex shader are available in the editor and act in their plain version as pass-through shader for the screen-aligned textured quad.

Your task is to extend the render-to-texture 2D fragment shader, so that it produces a simple blur effect.

Simple blur can be achieved by sampling the available texture in the direction towards the image center. In this example we work with normalized texture coordinates, which means for GLSL, that every position within the input texture is encoded within [0.0 1.0]. Therefore the image center is located at $c = (0.5, 0.5)$ and the vector to the image center from any position p can be calculated by $\vec{p} = c - p$. By accumulating color values from the input texture tex parallel to the normalized $\vec{p}$ you can define a blurred color value for the current pixel according to it's distance d to the current pixel position p:

$$rgb_{blur} = \frac{1}{n}\sum_{i=0}^{n}(tex(p + \vec{p} * d_i)), \tag{1}$$

where d can be limited to a maximum range d_{max} and sampled within this range by fixed distances s_i. Therefore,

$$d_i = s_i * d_{max}. \tag{2}$$

You should use the following $n = 12$ factors s_i to determine your samples within d_{max}:

```
float s[12] =
float[](-0.10568,-0.07568,-0.042158,
-0.02458,-0.01987456,-0.0112458,
0.0112458,0.01987456,0.02458,
0.042158,0.07568,0.10568);
```

When defining $d_{max} = 1.0$ and, the resulting scene should look similar to Figure 3.

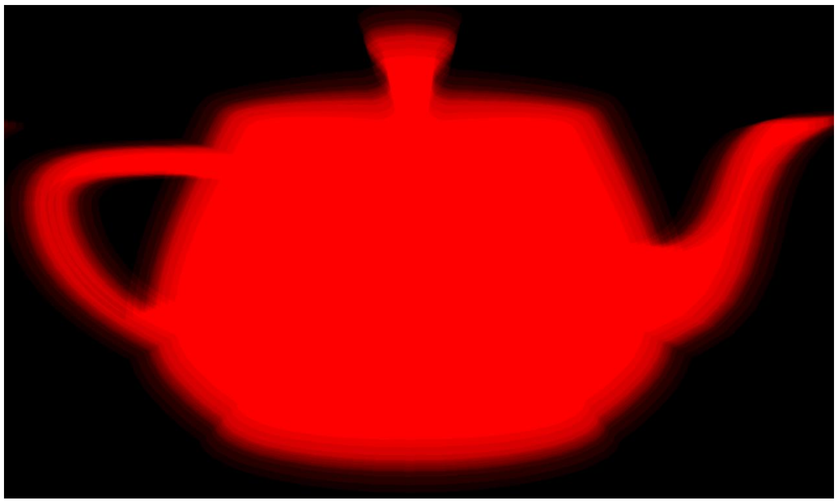


Figure 3: Very simple radial blur effect.

HAVE A LOT OF FUN!!