Interactive Computer Graphics
Coursework – Task 3 (*assessed)

December 13, 2020

Task 3: Illumination and Shading

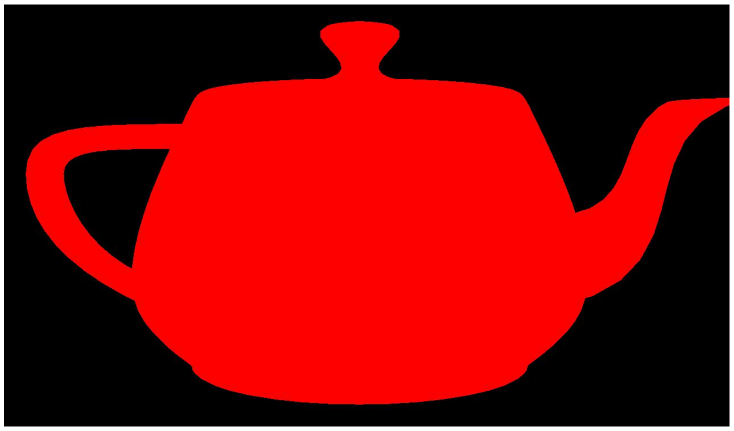


Figure 1: Default scene shown by the framework: an unshaded teapod model.

In this exercise you will learn how to use vertex and fragment shaders for vertex-wise and pixel-wise scene illumination. We use the *per-polygon* vertex and fragment shaders for this task (the two in the left-most tabs). The framework shows you an unshaded read teapot model per default as shown in Figure 1. At the end of this task you will be able to program different shading models as shown in Figure 2.

Task 3a: Per Vertex Gouraud shading*

For this exercise you will need to edit the *per-polygon* vertex and fragment shaders. This shader performs operations on scene vertices in object space. It is already provided by the editor.

In this exercise you will implement *Gouraud shading* as discussed during the lecture. Gouraud shading is an interpolation scheme for the illumination based on the viewer's, the vertex' and the light position within the scene.

In the *per-polygon* vertex shader you need to define basic interfaces to the following fragment shader to pass on specific information about the currently processed vertex. You can do this by defining in both shaders global `varying <type> <name>` variables. Note that these variables will be passed through the rasterization stage and when accessed in the fragment shader they will contain the interpolated result of the initially assigned per-vertex value.

`gl_Position` in the vertex shader and `gl_FragColor` in the fragment shader is the only output that is expected to be set by the GLSL compiler. Everything else can be freely defined. `mMatrix` is the ModelView matrix and `pMatrix` is the projection matrix. These are updated by the main program using `uniform` variables. Please not that previous versions of OpenGL and GLSL had intrinsic variables for values like the ModelView and Projection matrix. Modern OpenGL is freely programmable and defines the use of the intrinsic as deprecated. Many examples on the internet might use old style OpenGL and intrinsic. Also WebGL still uses this older style but we try to emulate modern OpenGL with the *attach to* mechanism.

You can define light source and material properties through uniform variables. e.g.,

```
uniform vec4 ambient;
uniform vec4 diffuse;
uniform vec4 specular;
uniform float shininess;
```

The light source is part of the scene. For our purpose a headlight is sufficient, i.e., the light may move with the camera. Try to make a light source that is part of the scene so that the highlight stays at a constant location on the object when you move the camera. Please submit the headlight version.

To convert a vector from `vec4` to `vec3`, you can either cast the vector, e.g., `vec3 vec = vec3(lightPosition_camSpace),`, or access the vector elements individually

```
vec3 vec = someVector.xyz;
```

`float x = someVector.x;` etc..

Your task for the exercise is to pass a color value to the fragment shader and set `gl_FragColor` **according to *Gouraud shading.*** Use the provided *Utah Teapot* model for testing. (Model tab → Mesh → Teapot).

Note that you only need to define one color per vertex. The interpolation between these vertices's is done by the rendering pipeline.

The result should look similar to Figure 2a. Note the illumination problems at the border of the specular highlight.

Task 3b: Per Pixel Phong shading*

In this part you will implement *Phong shading* as discussed during the lecture. Phong shading is an interpolation scheme for the illumination based on interpolated normal vectors for each fragment instead of interpolated colors as done for the previous task . The shading effect depends on the viewer's, the fragment's and the light position. In contrast to Exercise , this exercise operates directly on fragments and needs therefore most of the extension of the *per-polygon* fragment shader.

You can use the same interface definitions as provided in Exercise . However, note that in the fragment shader you get an input fragment instead of an input vertex.

Your task for exercise is to redefine `gl_FragColor` according to *Phong shading.* Use the provided *Utah Teapot* model for testing. (Model tab → Mesh → Teapot).

You can again access the light source position as in Task 3a.

If you test your programme, the result should look similar to Figure 2b. Note that the quality of the specular highlight is better than in Figure 2a.

Task 3c: Per Pixel Toon shading*

In this part you will implement *Toon shading*. Toon shading is a simple lighting scheme, which allows you to achieve effects similar to hand drawn cartoons. This exercise also operates directly on fragments and needs therefore most of the extension in the *per-polygon* fragment shader. You can use preprocessor definitions as common in C-like language to switch between the shading types.

A toon shader can be defined per fragment as done in equation 1 and equation 2.

$$I_f = \frac{l}{||l||} \cdot \frac{n}{||n||} \tag{1}$$

$$I = \begin{cases} (0.8, 0.8, 0.8, 1.0), & \text{if } I_f > 0.98 \\ (0.8, 0.4, 0.4, 1.0), & \text{if } I_f > 0.5 \text{ and } I_f <= 0.98 \\ (0.6, 0.2, 0.2, 1.0), & \text{if } I_f > 0.25 \text{ and } I_f <= 0.5 \\ (0.1, 0.1, 0.1, 1.0), & \text{if } else \end{cases} \tag{2}$$

Your task for exercise is to redefine `gl_FragColor` according to *Toon shading*. Use the provided *Utah Teapot* model for testing. (Model tab → Mesh → Teapot).

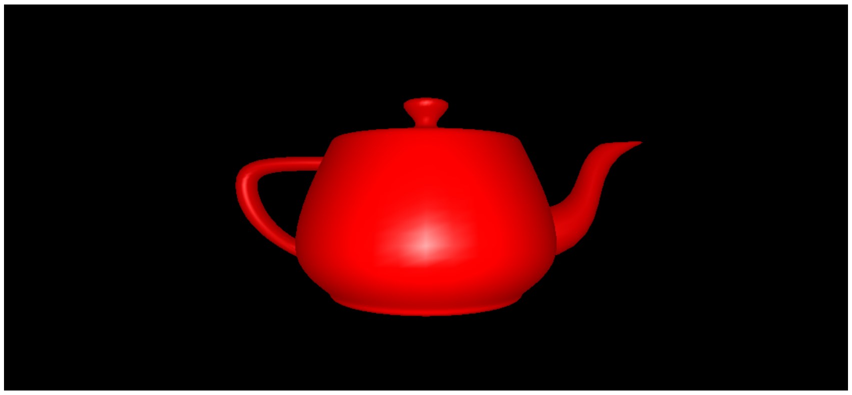
The final result should look similar to Figure 2c.

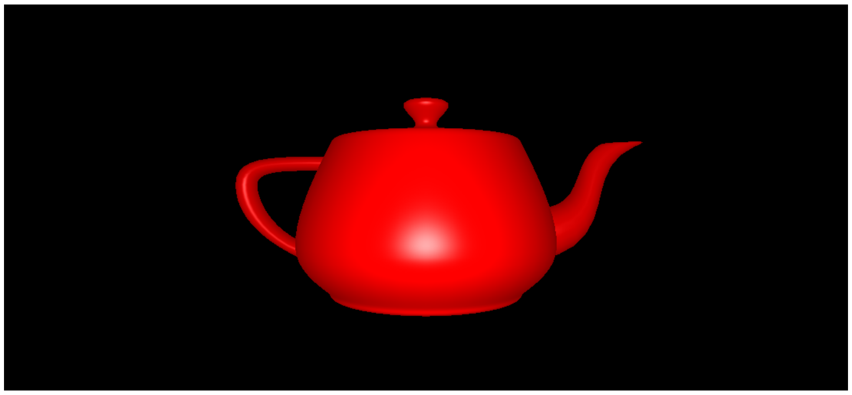
Task 3d: Blinn-Phong (not assessed)

Phong shading is not the most efficient way to approximate illumination. Blinn-Phong is a more efficent modification of Phong shading using halfway-vectors.

Your task for exercise is to redefine `gl_FragColor` according to *Blinn-Phong shading* as discussed during the lecture. Use the provided *Utah Teapot* model for testing. (Model tab → Mesh → Teapot) Figure 3 shows and example output.

HAVE A LOT OF FUN!!


(a) Gouraud shading


(b) Phong shading

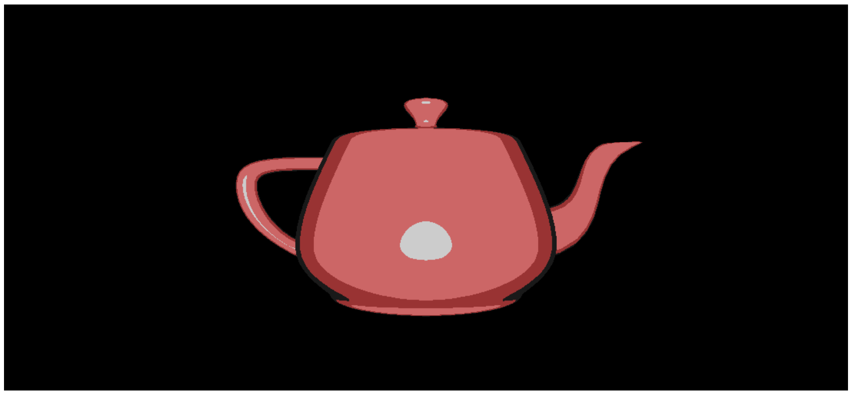

(c) Toon shading

Figure 2: Results of Task3a-c at identity view matrix with camera z location at -37 and a light intensity of 500.0. ambient = (0.5,0,0,1), diffuse = (1,0,0,1), specular = (1,1,1,1), shininess=10

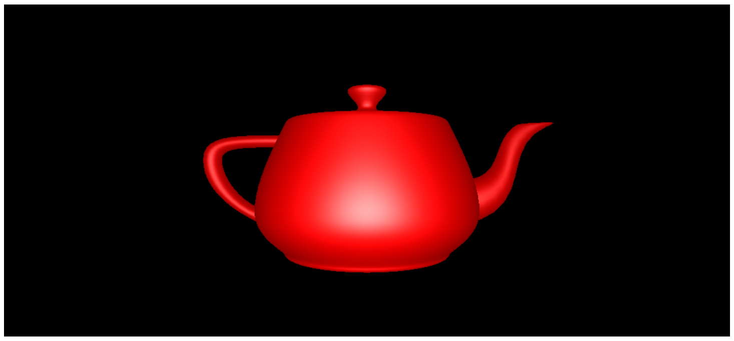


Figure 3: Result for Task 3b: Blinn-Phong shading at identity view matrix with camera z location at -37 and a light intensity of 500.0. ambient = (0.5,0,0,1), diffuse = (1,0,0,1), specular = (1,1,1,1), shininess=10