Prof. Dr. Leif Kobbelt
Dr. Jan Möbius
Patrick Schmidt, Moritz Ibing

Visual Computing Institute | RWTH AACHEN UNIVERSITY

Basic Techniques in Computer Graphics

Assignment 7

Date Published: November 26th 2019, Date Due: December 3rd 2019

- All assignments (programming and text) have to be completed in teams of 3–4 students. Teams with fewer than 3 or more than 4 students will receive no points.
- Hand in **one solution per team per assignment**.
- Every team must work independently. Teams with identical solutions will receive no points.
- Solutions are due 14:30 on December 3rd 2019. Late submissions will receive zero points. No exceptions!
- Instructions for **programming assignments**:
  - Download the solution template (a zip archive) through the Moodle course room.
  - Complete the solution.
  - Prepare a new zip archive containing your solution. It must contain exactly those files that you changed. **Only change those files you are explicitly asked to change in the task description.** The directory layout must be the same as in the archive you downloaded.
  - Upload your zip archive through Moodle before the deadline. Use the Moodle group submission feature. Only in the first week (when Moodle groups have not been created yet), list all members of your group in the file `assignmentXX/MEMBERS.txt`. Remember, only one submission per group.
  - Your solution must compile and run correctly **on our lab computers** using the exact same `Makefile` provided to you. Do not include additional libraries and do not change code outside of the specified sections. If it does not compile on our machines, you will receive no points.
- Instructions for **text assignments**:
  - Prepare your solution as a single pdf file per group. Submissions on paper will not be accepted.
  - If you write your solution by hand, write neatly! Anything we cannot decipher will receive zero points. No exceptions!
  - Add the names and student ID numbers of all team members to every pdf.
  - Unless explicitly asked otherwise, always justify your answer.
  - Be concise!
  - Submit your solution via Moodle, together with your coding submission.

Prof. Dr. Leif Kobbelt
Dr. Jan Möbius
Patrick Schmidt, Moritz Ibing

Visual Computing Institute | RWTH AACHEN UNIVERSITY

Exercise 1 Transformations and Normals [12 Points]

Consider a triangle with the three vertices $\mathbf{p_1} = (8, -10, 2)^T$, $\mathbf{p_2} = (-4, -10, -2)^T$ and $\mathbf{p_3} = (6, 4, 6)^T$. The triangle is first rotated by $45°$ around the y-axis, then scaled by $(1, 4, 16)$ (in x-, y-, and z-direction) and finally translated by 10 units along the x-axis.

(a) [4 Point]

Derive the transformation matrix for the points of the triangle.
Hint: $\cos(45°) = \sin(45°) = 1/\sqrt{2}$

(b) [2 Point]

As seen in the lecture, every 3D affine transformation can be represented by a 4×4 matrix using extended coordinates. For which kinds of three-dimensional affine transformations can the corresponding transformation matrices be used as-is (without any modifications) to also transform normals? You can assume that normals are explicitly normalized after they are transformed. Explain your answer.

(c) [6 Point]

Derive a matrix that correctly transforms the normal of the triangle. Compute the normal of the triangle before and after the transformation. You have to normalize again after the transformation.

Exercise 2 Local Lighting [8 Points]

You are visiting the International Space Station. You take a look out of a window and are quite impressed by the view of the moon—especially by that of point $\mathbf{p} = (1, -7, -6)^T$ (in some helio-centric coordinate system not specified further) on its surface. At that point the moon has a normal vector $\mathbf{n} = (0, 1, 0)^T$. Your current position is $\mathbf{v} = (4, -4, -11)^T$ and the sun is of course situated in the origin, accepted to be a point light source emitting light of color $C_l = (1, 0.75, 0)$—the only one in the universe.

According to one of your fellow passengers, the surface of the moon has a shininess of 1 and reflection properties specified by the material matrices $\alpha_d = diag(0.75, 0.75, 0.5)$ for diffuse reflection and $\alpha_{sp} = diag(0.5, 0.5, 1.0)$ for specular reflection. Ambient light is not present.

(a) [5 Points]

According to the Phong lighting model (ignoring attenuation effects), in which color do you perceive point p?

(b) [3 Points]

Now your position moved to $\mathbf{v}' = (4, -5, -11)^T$ and you want to calculate the color at $\mathbf{p}$ again. Unfortunately, in the meantime you forgot how to reflect vectors on planes, hence you cannot evaluate the specular term. Your fellow passenger reminds you that you could alternatively use the Blinn-Phong model that avoids the computation of the reflected light vector but comes up with a similarly plausible solution. In which color do you perceive $\mathbf{p}$ according to the simpler Blinn-Phong model?

Prof. Dr. Leif Kobbelt
Dr. Jan Möbius
Patrick Schmidt, Moritz Ibing

Visual Computing Institute
RWTH AACHEN UNIVERSITY

Exercise 3 Programming [20 Points]

In this week's programming exercise your task is to show us what you have learned about local lighting. File `assignment.cpp` already contains all the code you need to get the geometry and the necessary parameters onto the graphics card, ready to be rendered. The only thing that needs to be done is writing the shader programs for the scenes. The shader source files have file extension `*.vsh` (Vertex Shader) and `*.fsh` (Fragment Shader). The code you write in the Vertex Shader will be applied to every vertex in the scene, while the Fragment Shader is applied to every fragment generated by the rasterizer. The scene to be displayed contains a teapot mesh which is loaded onto the graphics card. Additionally to the vertices' positions, you also have vertex normals available that you are supposed to use for the color computation. Furthermore, the following variables and parameters are available for use in your shader code:

`aPosition`: The vertex's position in world-space coordinates.

`aNormal`: The vertex's normal in world-space coordinates.

`uProjectionMatrix`: The projection matrix.

`uModelViewMatrix`: The model-view matrix, transforming points to camera space.

`uLightPosition`: The light source's position in *camera space* coordinates.

`uLightColor`: The light source's color.

`uLightSpotLightFactor`: The spot light exponent for the light.

`uAmbientMaterial`: The ambient material matrix for the teapot.

`uDiffuseMaterial`: The diffuse material matrix for the teapot.

`uSpecularMaterial`: The specular material matrix for the teapot.

`uSpecularityExponent`: The specularity exponent for the teapot.

When the keys a, b, or c are pressed, the solution for the corresponding sub-exercise should get shown. `drawScene` gets called with 1, 2, or 3 as the `scene` parameter accordingly.
The final (Screen-Space) vertex positions that you will compute need to be written into the variable `gl_Position` within the Vertex Shader, the final color needs to be written into `oFragColor` within the Fragment Shader.

Both shaders are written in the OpenGL Shading Language, GLSL version 1.50 (OpenGL version 3.2). Its syntax is very close to c++. In in doubt, check the language specification[1]. Feel free to use the following GLSL functions in your shader code: `pow(), normalize(), length(), inverse()` and `transpose()`, or any other built-in function of GLSL 1.50. If needed, convert 3D vectors to 4D vectors with `vec4(myVec3, 0.0)` and vice-versa `vec3(myVec4)`, which just omits the fourth component.

Please do **not** rename the shader files! Only the following files should be modified:

- shader_a.vsh
- shader_a.fsh
- shader_b.vsh
- shader_b.fsh
- shader_c.vsh
- shader_c.fsh

[1] https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.1.50.pdf

(a) [10 Points]

`scene == 1`: Implement the *Phong-Blinn* lighting model that computes the color of each vertex of the mesh. Use the *Gouraud* model for the shading stage.

In the scene at hand there is only one single *point* light source having a position and a color. For simplicity reasons, assume that the attenuation factor is 1. We already transformed the points into the right space in order to do the calculations. You still need to do the same for the normals. Remember to bring the points into screen space in the end. The teapot's material properties are given as parameters (as described above).

Note: The incoming fragment shader parameters defined as outgoing parameters of a vertex shader are already bi-linearly interpolated using the fragment's barycentric coordinates with respect to its triangle. The resulting scene should look like the screenshot on the left in Fig. 1.

(b) [4 Point]

`scene == 2`: Assume the same setting as in the previous exercise but this time consider the light source to be a spot light. The spot light's direction should point from the light source to the *global* origin (also the teapot's center). The spot light exponent is given as a parameter. Your scene should now look like the screenshot on the right in Fig. 1.

(c) [6 Points]

`scene == 3`: As you hopefully have discovered by now, Gouraud shading does not really generate realistically looking renderings due to interpolation artifacts that reveal the mesh's tesselation at some spots. Fortunately, you know everything about the *Phong shading* model that interpolates the normals rather than the colors for each fragment. So the lighting computation is performed for each fragment as compared to each vertex.

Implement the Phong shading model using the spot light from the previous scene. Similar to the previous scenes, assume the attenuation factor to be 1. Your scene should now look like the one shown in Fig. 2.

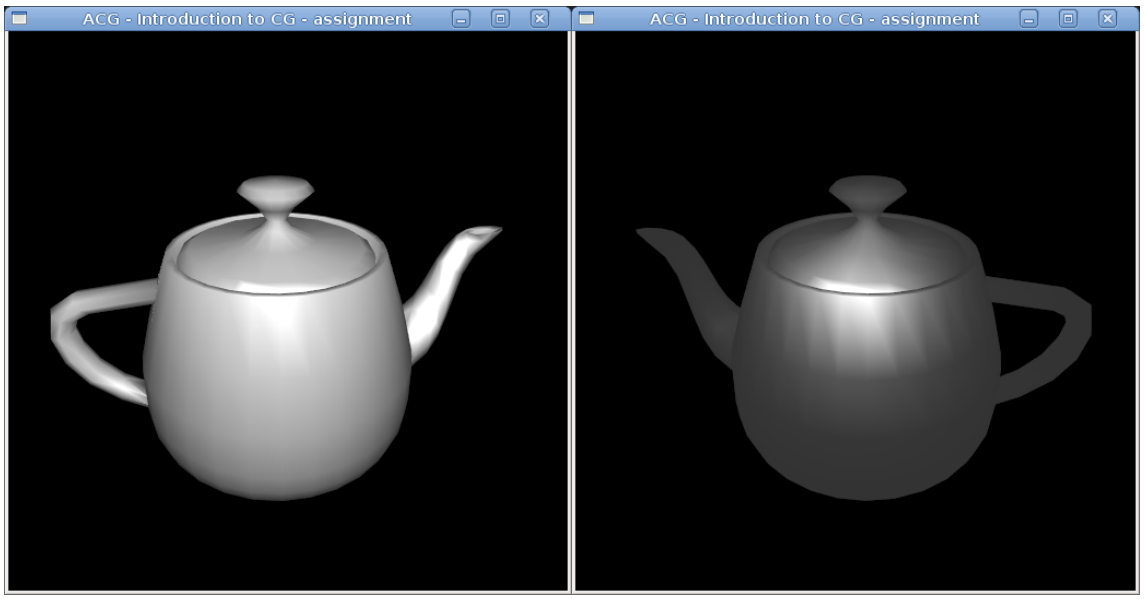

Figure 1: The teapot model lit with the Phong-Blinn local lighting model and Gouraud shading. Left: Scene 1. Right: Scene 2.

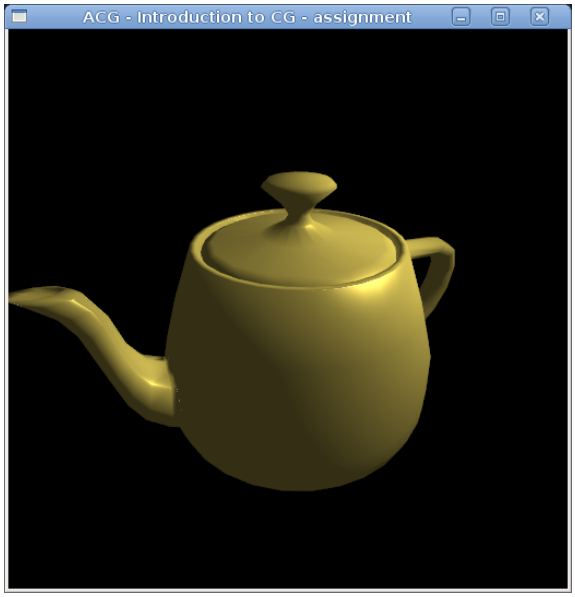

Figure 2: Scene 3. If you still see small tesselation artifacts around the handle or the spout of the teapot, don't be worried! They are due to discontinuities in the normal field. The rest of the mesh should appear as smoothly curved surfaces though.