DVA338 Fundamentals of Computer Graphics

Assignment 2

1 Shader programming

Current graphics hardware supports the execution of several types of user-defined programs such as vertex, fragment, geometry, tessellation, and compute shaders. This assignment will give you some practical experience in writing your own shaders using the OpenGL Shading Language (GLSL). In this assignment you will focus on vertex and fragment shaders, along with a brief introduction to geometry shaders.

1.0 Loading the Shaders from Text Files

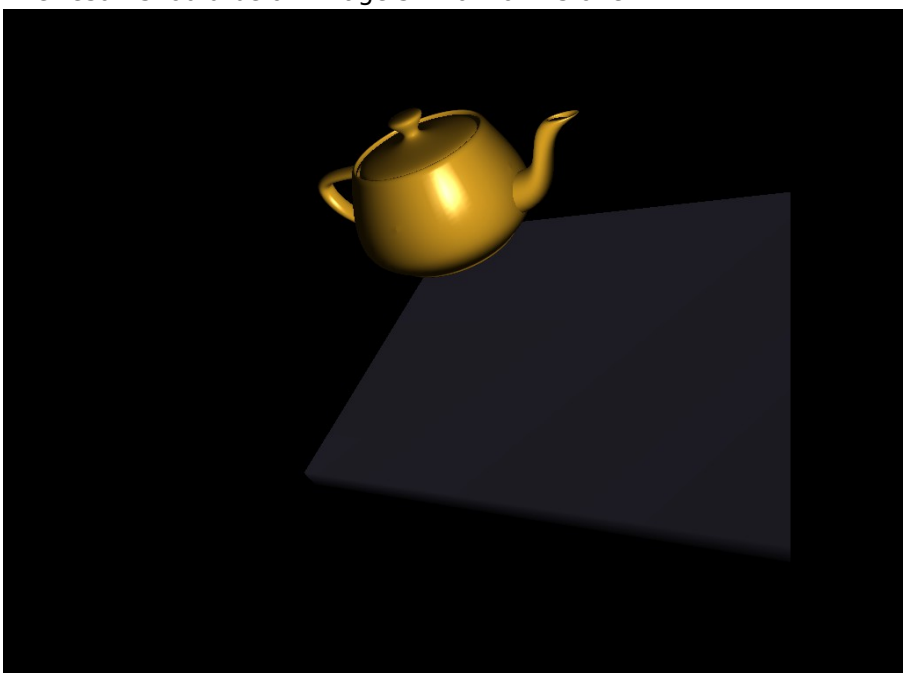
The startup code comes with basic a basic vertex shader and a basic fragment shader. The GLSL code for these shaders is defined in the "`shaders.h`" file as string arrays and then later used in the "`main.cpp`" file (look at the `init` function in `main.cpp` and follow the call to `prepareShaderProgram` function).
Of course this is not a good way of doing this. **Your first task** is to implement a system to load the shader code from text files to related string arrays. Then type in the provided shader code into text files and load them into string arrays before calling the prepareShaderProgram. This will make the rest of the assignment much easier. Additionally, you are advised to extend the shader compilation so that it outputs compilation information. This will help you during shader debugging.
For more information about how to do this, see e.g. https://www.khronos.org/opengl/wiki/Example/GLSL_Shader_Compile_Error_Testing

1.1 Gouraud shading

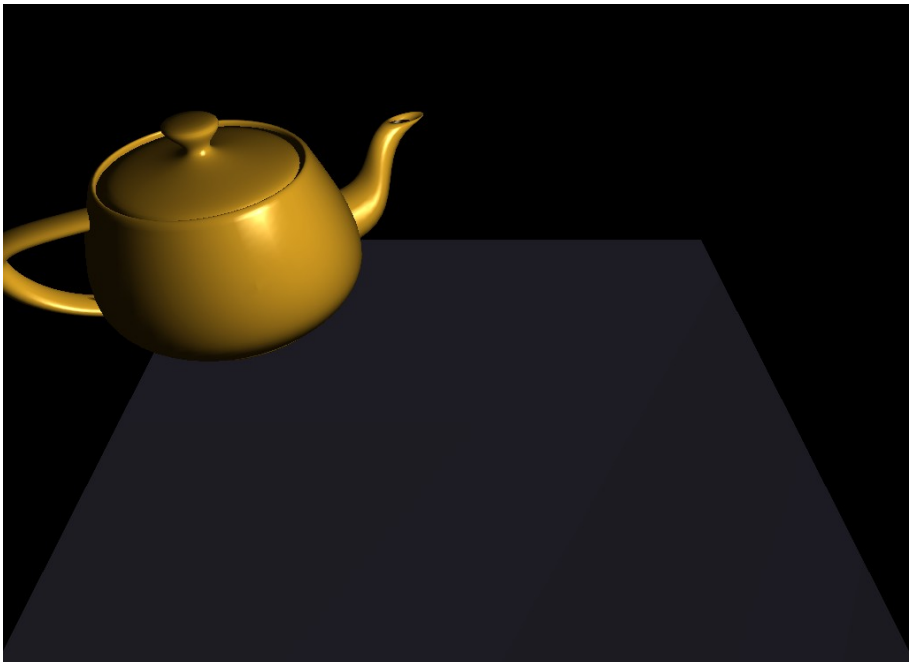
Implement a shader program that uses the Phong Reflection Model to find the light intensity at polygon vertices, and then uses Gouraud shading over the triangle surfaces. You need to add suitable parameters to describe a point light source in the scene as well as material descriptions for your models.
Checking your work:
Make the following scene:
Light:
Pos: (x,y,z) = (10,10,10)
ambient(r,g,b) = (0.2,0.2,0.2)
diffuse(r,g,b) = (1.0, 1.0, 1.0)
specular(r,g,b) = (1.0, 1.0, 1.0)
Camera:
Pos (x,y,z) = (20,20,30) Rotation (x,y,z) = (-45, 30, 0) degrees
FOV: 60 degrees

Objects:
Teapot – Default position and scaling *(uniform scaling should be 2.0, old versions of lab skeleton may have other values, if so please change it)*, but rotated -90 degrees around x axis.
Use the following for material:
Ambient: (0.329412f, 0.223529f, 0.027451f)
Diffuse: (0.780392f, 0.568627f, 0.113725f)
Specular: (0.992157f, 0.941176f, 0.807843f)
Shininess: 27.8974f

Cube:
Default, but non-uniform scaling (x,y,z) = (4.0,0.2,4.0)
Use the following material:
Ambient: (0.05375f, 0.05f, 0.06625f)
Diffuse: (0.18275f, 0.17f, 0.22525f)
Specular: (0.332741f, 0.328634f, 0.346435f)
Shininess: 38.4f
For other examples of materials to play around with, see e.g. http://devernay.free.fr/cours/opengl/materials.html
The result should be an image similar to this one:

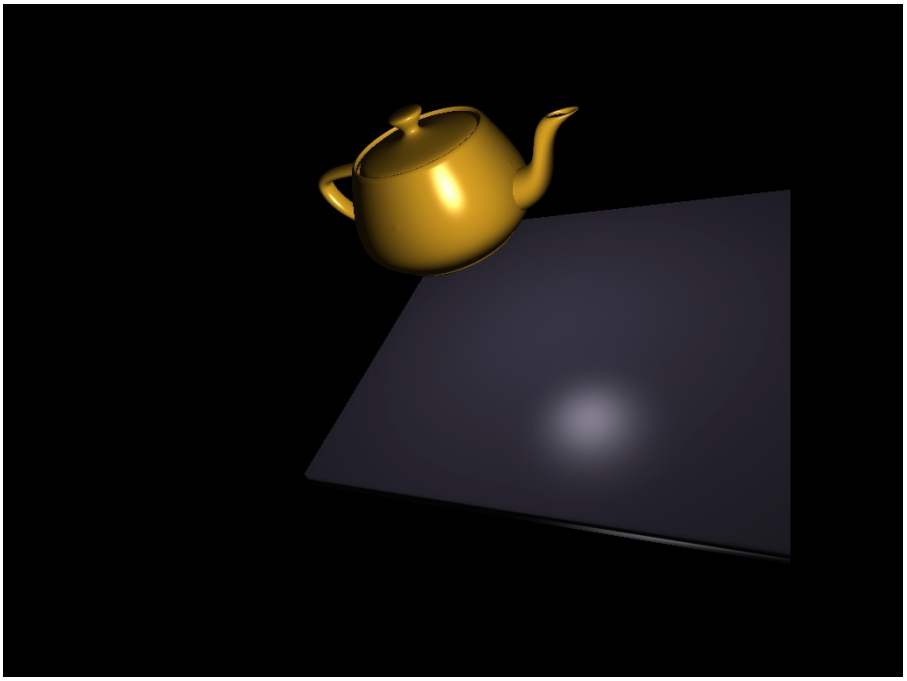


Note: If you experience issues with the plate overlapping the teapot's bottom, you should check your handling of the depth buffer. If that does not solve it, here's an alterative scene to use:
Camera:
pos(x,y,z)=(10, 15, 20) rot(x,y,z)=(-45, 0, 0) FOV=60
Teapot:
Translation(x,y,z)=(3, 2, 5) Rotation(x,y,z)=(-90,0,30)
Light – same as standard scene above

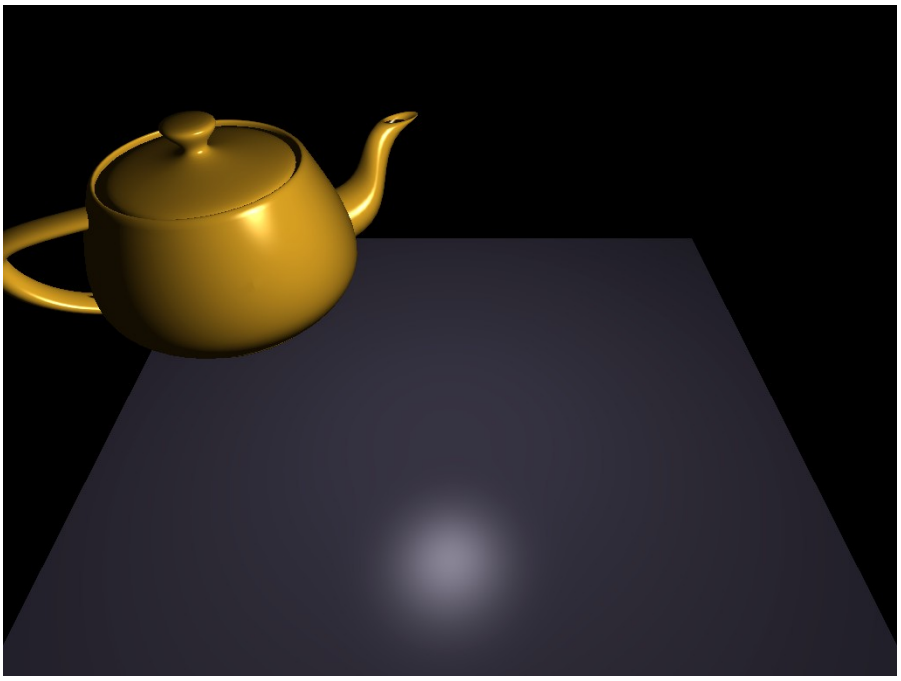


1.2 Phong shading

Write a shader program that accomplishes Phong Shading, i.e., vertex normals are interpolated over the faces. In this way, the Phong Reflection Model can be used for each pixel. Show a demo of your solution illustrating that nice looking specular highlights can appear also on large polygons. Compare the result from this shader to the corresponding result using Gouraud shading. Finally, make it possible for the user to select the shader program to be used, interactively (i.e. during runtime).
Checking your work: Using the same scene as above, we can now see that using phong shading, the large polygon making up the cube side still has some specular highlights.



And the alternative scene:

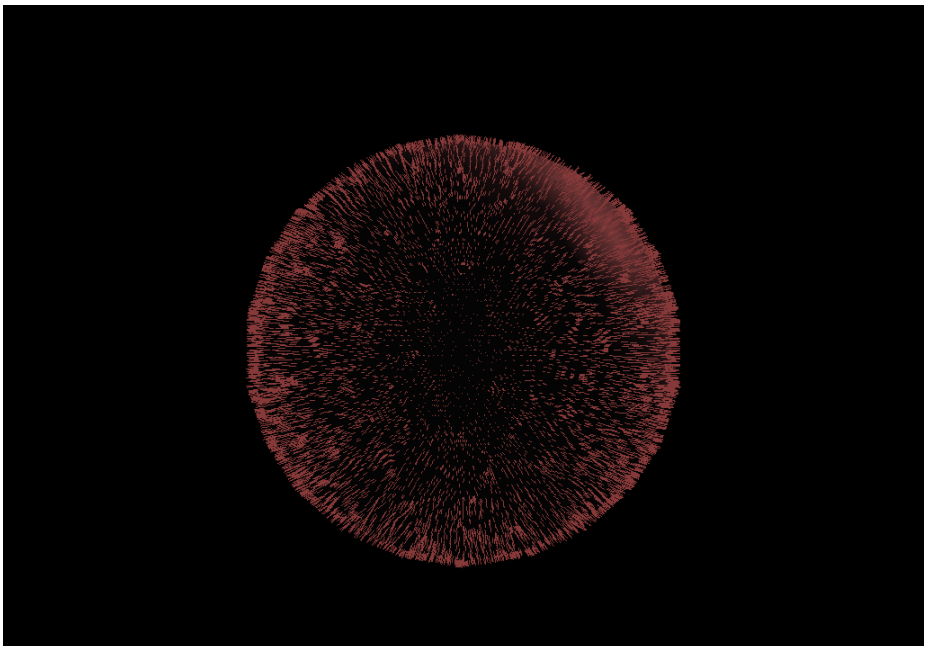


1.3 Geometry fur shader

Write a shader program that uses the geometry shader to add additional lines coming out of each surface, making a rudimentary rendering of fur. Then use this program **together with one of the previous ones** to do a two-pass rendering of a furry model.

As an example, using the default camera (0,0,20, no rotation) and the lightsource from above, the image below shows a sphere placed at (0,0,17) with a uniform scaling of 1 *(in older versions of lab skeleton, reduce default from 12 to 1)* and the following material:
{ { 0.05375f, 0.05f, 0.06625f } , /* Mat_amb */
{ 0.5f, 0.22525f, 0.22525f }, /* Mat_diff */
{ 0.5f, 0.22525f, 0.22525f }, /* Mat_spec */
38.4f } /* Mat_shininess */
The lines are uniformly colored using the color (0.5,0.2,0.2).

Note: Depending on how you approach this assignment, your results may vary. This is OK, the main point is that you get a feeling of how the geometry shader works, and how to combine different shader programs to get interesting effects. **That being said, please try to get something fur-like. A single line on each triangle/vertex is not fur.**



1.4 Extra: Multiple light sources

Extend your previous shader programs by adding support for at least two point light sources.

1.5 Extra: Cartoon shading

Implement a simple cartoon shader, which gives a 3D object a kind of 2D look by only using a few levels of color for the shading. Make it possible for the user to choose at run-time if your cartoon shader, your Phong shader, or the Gouraud shader should be used. Add support for at least two point light sources in your cartoon shader program.

1.6 Extra: Flat shading

Design a shader program that emphasize the actual faceted look of the polygon meshes by using the true mathematical polygon normal in the lighting computations, rather than constructed vertex normals.

1.7 Extra: Morphing

Write a shader program that renders a morphing model, that is, a model trans-formed smoothly from a source model to a target model, and back again. Note that you only need to care about morphing between triangle models which have same number of vertices, triangles, and mesh topology.

1.8 Extra: Mapping methods

Write a shader program that utilize texture mapping techniques to accomplish various rendering effects. For example, implement one of the following popular techniques: environment mapping, bump mapping, or shadow mapping. Texture mapping can also be used to render more realistic fur than what you did in Assignment 1.3.

1.9 Extra: Anisotropic shading

Implement a reflection model for anisotropic shading in a shader program. Try to capture the streaky appearance of the highlights as seen for example on objects made of brushed metals. Demonstrate the effect of your shader program on several different polygonal models, and compared the results with the results produced by your Phong shader.

1.10 Extra: One-pass wireframe rendering

Write a shader program to accomplish high-quality wire frame rendering of polygon models with hidden line removal. See the one-page article Single-pass Wireframe Rendering by Baerentzen et al. published in ACM SIGGRAPH 2006 Sketches for more information on how this can be done.