DVA338 Fundamentals of Computer Graphics

Assignment 3

1 Shader programming

Current graphics hardware supports the execution of several types of user-defi ned 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). We will focus mainly on vertex and fragment shaders.

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.

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 which shader program that is used interactively.

1.3 Multiple light sources

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

1.4 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.

1.5 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.

1.6 Extra: Morphing

Write a shader program that renders a morphing model, that is, a model transformed 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.

1.7 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.

1.8 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.

1.9 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.