# CS541 Project 1

## Synopsis

Implement Phong lighting and Phong shading using GLSL, the OpenGL shader language.

## Instructions

This project serves as an introduction to GLSL shader programming. You are to implement Phong lighting and Phong shading as follows. Phong **shading** is implemented by calculating Phong **lighting** at every pixel using values interpolated from vertices to every pixel. (This is in contrast to Gouraud shading, where the Phong lighting is calculated only at vertices and the resulting color is interpolated to each pixel.) The calculations are to be done in the eye-space, with some of the calculations done in a vertex shader and some in the pixel shader, and communication between the two done via varying variables.

The Phong lighting calculation requires three vectors:

- The normal to the surface
- The direction toward the light
- The direction toward the eye

All three vectors should be

- Calculated at each vertex in the eye-space,
- Passed to the pixel shader via GLSL varying variables,
- Normalized in the pixel shader to calculate Phong Lighting at each pixel.

The details of each step are shown in the next two sections.

## Vertex Shader

Inputs:

- **gl_Vertex:** The vertex in modeling coordinates.
- **gl_Normal:** The normal in modeling coordinates.
- **gl_ModelViewMatrix:** The matrix to transform vertices to eye-space
- **gl_NormalMatrix:** The matrix to transform normals to eye-space.
- **E=(0,0,0):** The eye's position in eye-space.
- **gl_LightSource[0].position:** The light's position in eye-space

Outputs/calculations:

- gl_Position = ftransform(); // or gl_ModelViewProjectionMatrix*gl_Vertex
- pixel position as a local variable **V**=(gl_ModelViewMatrix * gl_Vertex).xyz
- varying variable **normal** as gl_NormalMatrix*gl_Normal
- varying variable **eye** as E-V
- varying variable **light** as gl_LightSource[0].position.xyz – V

## Pixel (or fragment) shader

Inputs:

- varying variables **normal, eye, light** interpolated by the rasterization pipeline. (Don't forget to scale these to unit length.)
- gl_LightModel.ambient: Scene ambient light.
- gl_LightSource[0]:
  - ambient: Light's contribution to ambient light
  - diffuse: Diffuse version of $I_i$
  - specular: Specular version of $I_i$
- gl_FrontMaterial:
  - emission: $I_e$,
  - ambient: $K_a$,
  - diffuse: $K_d$,
  - specular $K_s$
  - shininess: $n$

Output/calculations:

- gl_FragColor = ...

## The framework

The framework provides all the features necessary to engage in shader programming including some partially written dummy shaders.

The framework:

- Opens a window, creates a scene, and draws it to the window.
- Reads in, compiles, links, and uses a shader program. The initial shader program is nearly useless, but provides a starting point for your shader programming.
- Provides mouse and keyboard interaction to manipulate the scene. The left mouse button rotates the scene, the middle wheel zooms in/out, and the right button pans the scene. In addition, the 'm' key toggles between two models (sphere and torus)/ the 's' key toggles the shader programs on/off, the 'q' key exits, and the digits 0..9 are communicated directly to the shader program via a uniform variable named mode.

## How to submit

Use Moodle to submit a single zip file of your project.

- Create a zip file containing only your *.vert and *.frag files and nothing else. (If, for some reason, you made any changes to the framework, then also include any changed **source** files.)
- Log onto the Moodle at "https://distance.digipen.edu/", follow the course web page link CS541, and find the project submission link "CS541 Project 1" . Use the input field, and the "Browse" and "Upload this file" buttons to upload your zip file.

## Grading Basis

Grading will be on a 100 point basis, with the following distribution:

- **Accuracy:** 80%, judgment based on looking at the results on the screen.
- **Code:** 20%, judgment based on looking at the code for proper implementation and reasonable commenting practices.