DigiPen Institute of Technology

CS300

Assignment 2: Implementing Phong Illumination Model

Due date:

- Week 7, midnight
- For late submission, please refer the course outline document.

Description

Add the following functionality to your first programming assignment:

1. Write code to load, compile, link and install shaders
   - Shader code MUST be loaded from external file, i.e. you must NOT embed the shader code within your 'C/C++' code.
   - The "Reload Shaders" button should be used to load, compile, link and install GLSL Shaders on-the-fly by the user.
   - Compile and install the shader code to be used for geometry rendering. Should the shader/program fail to compile or link, you must print out the error messages to the console or the main window. We recommend creating a log-window at the bottom of the screen to display all messages.
   - The error message MUST specify which shader file(s) is (/are) causing the problem.
   - **Hint:** Refer the "ShaderManager" class in the framework – a program object contains a vertex and a fragment shader.
2. Write vertex and fragment shaders to implement Phong illumination model with support for point, spot and directional light source type. We will implement the model using three different shaders.
   - **Phong Lighting:** This is the shader that is partially provided with the framework. Lighting computations are done on the vertex level, and only the final color is interpolated to the fragments. This is the implementation of the OpenGL fixed-function pipeline.
   - **Phong Shading:** This is the shading implementation of the model where the lighting computations are implemented in the fragment shader
     - In vertex shader:
       - Transform the geometry vertices and normal from the model space to the projection space.

        - Save view space vertex and normal and pass them as output to fragment shaders.
        - Calculate all other required "out" variables for the lighting calculation and pass them to the fragment shader.
    - In fragment shader:
        - Setup the uniforms for the application to pass in lighting properties and other settings.
        - Use the values of the appropriate "in" variables in the phong equation, if applicable. Note that these "in" variables MUST match the "out" variables from the vertex shader.
        - Implement the phong lighting equations explained in class.
  - **Blinn Shading**: The variation to the Phong model where we do not have to calculate the reflection vector (expensive part of the computation). Instead, we use the half-vector between the light vector and view vector, and combine it with the surface normal for the specular computation.
3. Must be able to support at least **8 active lights** simultaneously.
  - Implement the following lighting scenarios and provide capability to toggle between them using suitable menu actions:
    - All lights have the same color parameters and type (position/directional/spot).
    - All lights have different color parameters and type (position/directional/spot).
    - **Implement a third scenario that mixes the lights and their types. For example, you can have half of your active lights positional and the other half spotlights. Arrange them in a creative combination to get cool lighting effects.**
4. Material properties: We will implement complex material properties with the help of textures to encode the diffuse and specular material coefficients. This is an important part of this assignment, so make sure you understand how to setup textures and sampler objects in GLSL. Ignore the texture coordinates (if present) from the OBJ files. The texture coordinates will be computed per frame on the GPU (in the fragment shader) using methods discussed in-class.
  - Load the provided texture map to OpenGL and setup sampler for accessing texels in the provided texture.
    - Diffuse: metal_roof_diff_512x512.tga
    - Specular: metal_roof_spec_512x512.tga
  - The material shininess (the power for the specular calculation) is stored in the specular map. The specular map is stored as a three-channel image. Use the RED channel as the reference for the shininess values. For e.g. if the RGB values in the specular map are (167,167,167), use 167 as the shininess value.

- If texture mapping is turned off, use dark gray (0.25, 0.25, 0.25, 1) for the ambient, diffuse and specular material and black (0, 0, 0, 0) for the emissive material.
- You will generate the texture coordinates for the models "by hand" using the formulae for spherical, cylindrical and planar (6-sided) texture mapping. This process can be done on the vertex shader for Phong Lighting and fragment shader for Phong & Blinn Shading.
- For the purpose of assignment 2 – consider
  - the vertex position (normalized to the bounding box), and
  - the vertex normal (normalized and converted to range [0,1]) as the Texture Entity.

5. Light(s) properties:
   - Define the following variables in your 'C/C++' source and pass them into the shader as **uniform** variables:
     - The number of active lights
     - Light type (point, spot or directional)
     - Global ambient color (same for all lights)
     - Ambient, diffuse and specular colors (can change per light)
     - Position for point and spot (one for each light)
     - Direction for spot and directional (one for each light)
     - Inner and outer angle for spot light (same for all spot lights)
     - Spot falloff (same for all spot lights)
     - Distance attenuation coefficients (same for all lights)
     - Atmospheric (fog) color
     - Initial values:
       - Active light count = 1
       - Light source type = point
       - Light global ambient color = dark grey (0.2, 0.2, 0.2, 1)
       - Light ambient color = black (0, 0, 0, 0)
       - Light diffuse color = light grey (0.8, 0.8, 0.8, 1)
       - Light specular color = white (1, 1, 1, 1)
       - Spot light inner angle = 15 degree and outer angle = 30 degree as measured from the spot light direction vector
       - Spot light falloff = 1
       - Distance attenuation coefficients (c0, c1, c2) = (1, 0.1, 0)
       - Fog color to whatever the background clear color is
       - Fog near at 50% of camera far clip distance
       - Fog far at camera far clip distance
6. Scene setup:

- Render a 2d plane aligned with the **xz-plane** and 5 units under the lowest point in the OBJ file, i.e. the plane should be centered at a location `(0, ymin-5, 0)`, where ymin defines the minimum y-value in the OBJ file, with plane normal `(0, 1, 0).` The plane extents should be from (-10 to 10) in x and z directions respectively.
  - Scale the model such that the maximum extent of it's bounding box is from (-1 to 1) in the appropriate axis. In other words, find the maximum axis of bounding box (x, y or z) and rescale that axis to (-1 to 1). Apply the SAME scale factor to other axes to keep the aspect ratio of the model consistent.
- Place the light(s) equally spaced along a circle centered at `(0, ymax, 0)` with diameter of 1.5 times the diagonal of the model bounding box in the xz-plane; Let the lights rotate slowly (~90 deg/sec) around the y-axis.
- If light type is spot or directional, have them look at the **object center**, i.e. the light(s) follows the object center.
- Apply the shader to both the object and the plane.
- Render a small sphere (radius = 0.25) to represent the light(s) (**do not apply the shader on these spheres**.)
- Each sphere should be colored with the diffuse color of the light (**do not apply lighting to the spheres.)**

7. Have inputs to:
   - Select OBJ file to be rendered.
   - Toggle texture mapping on/off. Note the material setting you need to use when texture mapping is off (check #3 above).
   - Select the number of lights to be used in the scene.
   - Select the light types. Default light type should be a point light.
   - Toggle to pause/start the light 'rotation'.

Note

1. Add the following code in your first assignment
   - ShaderManager class to manage switching of shaders. This class should support a vertex shader and a fragment shader pair.
2. DO NOT use OpenGL half vector. Compute the half-vector yourself.
3. **DO NOT use OpenGL built-in lighting functionality and structures to pass in lighting properties. Use textures and uniform variable blocks to access the light and material properties.**
4. Make sure all information is in correct space.
5. **DEBUG tip: To check values of a certain variable in the fragment shader, you can set your fragment color to that variable so that you can visualize the variable**

value. You might need to do some transformation to bring the variables value to a [0, 1] range first.

6. DO NOT use GLSL's 'reflect' function for your specular calculation, i.e. write one yourself.

Extra-credit functionality:

As extra-credit worth **20 points**, implement the following functionality:

(a) Compute a circular path around the model with radius 10 units around the scaled model. Add a sine function to modulate the height of the points along the path. (10 points)

(b) Move your camera about this path such that it always points to the center of the object (use a dynamically changing model-view-matrix to achieve this) (10 points)

Assignment Submission Guideline

Please refer to the syllabus for assignment submission guideline. Failure to the submission guidelines correctly might cause you to lose points.

Appendix: Shading Models Primer

(Assume the notation as described in the course notes)

Phong Lighting:

Basic Idea: Perform the lighting calculations per vertex and interpolate the color values to individual fragments.

Implementation: Under this lighting model, you will perform the lighting model calculations in the vertex shader and pass on the color values (as interpolated variables) to the fragment shaders.

Phong Shading:

Basic Idea: Implement the per-pixel Phong Illumination model.

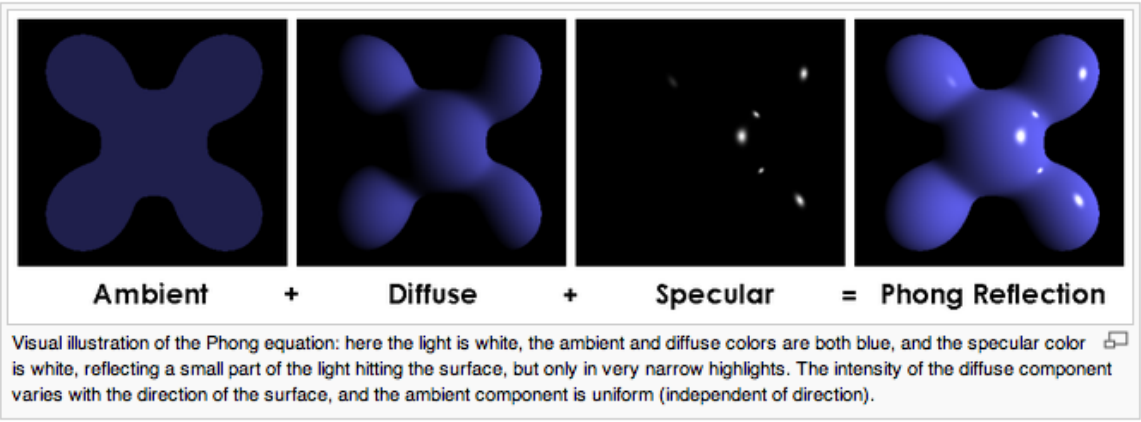
Implementation: Perform lighting calculations in the fragment shader while passing some data from the vertex shader to the fragment shader as varying variables.

Blinn Shading:

Basic Idea: Use the "half-vector" to perform specular highlight calculations instead of the per-pixel view-vector, to speed up the computations.

Implementation: Compute the half-vector for a vertex / triangle, and then use it in the fragment shader as the term H.

Also refer to http://en.wikipedia.org/wiki/Phong_reflection_model for details.



Visual illustration of the Phong equation: here the light is white, the ambient and diffuse colors are both blue, and the specular color is white, reflecting a small part of the light hitting the surface, but only in very narrow highlights. The intensity of the diffuse component varies with the direction of the surface, and the ambient component is uniform (independent of direction).

GRADING SHEET

Name of student: _____ **Total points obtained :** _____

Implementation Point	Grade	Points obtained	Comments
Shaders	**15%**		
Vertex shader compiles OK	**3**		**One point per shader**
Vertex shader links OK	**3**		
Fragment shader compiles OK	**3**		
Fragment shader links OK	**3**		
Shader program installed correctly and used to render geometry	**3**		
Material	**10%**		
Textures correctly loaded	**3**		
Samplers correctly defined and bound to textures	**3**		
Textures correctly sampled and used (diffuse texture for diffuse material component)	**3**		
Emissive component calculated correctly, if specified	**1**		
Phong Lighting Equation	**60%**		
Ambient term	**8**		
Diffuse term	**8**		
Specular term	**8**		
Distance attenuation	**8**		
Atmospheric attenuation/fog	**8**		
Spot light implementation	**10**		
Directional light	**10**		
Light value accumulation	**5%**		
Correct calculation of light intensity from multiple lights	**5**		
Miscellaneous	**10%**		
Correct README file	**2**		

with all entries	
Application compiles OK	**3**
Scene setup as described	**5**
Extra Credit	**20%**
Camera path computation OK	**5**
Camera path implementation looking at the object center	**5**
Roller coaster implementation with oscillating camera	**10**