# Reflectance Models

Professor : Dr. Michael Manzke

Student name: Yuzhou Shao

Student id: 19322035

Youtube Demonstration Link: https://youtu.be/BaxBP4zB6kk

Gouraud Shading and Phong Shading on Phong Reflection Model are implemented in this Project.

Gouraud Shading Phong Shading

# Background:

The Phong model describes the interaction of light with a surface, in terms of the properties of the surface and the nature of the incident light.

The Phong model reflected light in terms of a diffuse and specular component together with an ambient term. The intensity of a point on a surface is taken to be the linear combination of these three components.

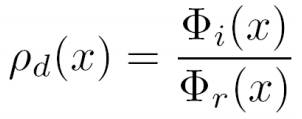
## Lambertian Illumination

We can use the cosine rule to implement shading of Lambertian or diffuse surfaces.

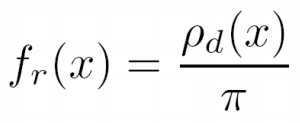
Diffuse surfaces reflect light in all directions equally:

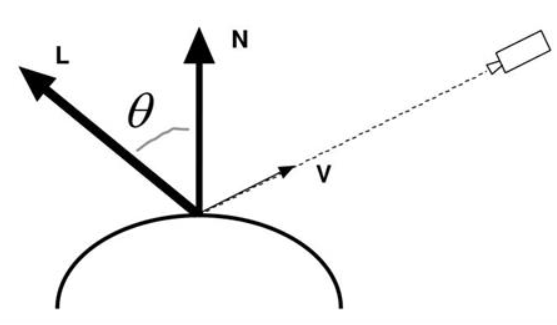
BRDF is a constant with respect to reflected direction

Surface may be characterized by a reflectance rd rather than a BRDF:

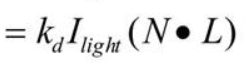


The reflectance gives the ratio of the total reflected power to the total incident power





## Lambertian Law

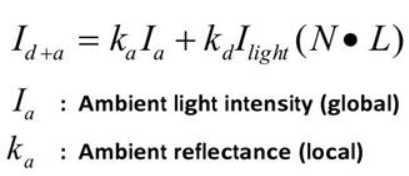
: Light Source Intensity

: Surface reflectance coefficient between 0 with 1.

: Light/Normal angle

## Ambient Light

CG started using the Lambertian model and then added more terms as extra effects



This is diffuse illumination + simple ambient light term, a trick to account for a background light level caused by multiple reflections from all objects in the scene.

## Specular Reflection:

One function that approximates specular falloff is called the Phong illumination model



: Angle between reflected light R and viewer V

: Specular reflectance

: Rate of specular falloff

## Put all together:



Combining ambient, diffuse, and specular illumination

For multiple light sources

1. Repeat the diffuse and specular calculations for each light source
2. Add the components from all light sources
3. The ambient term contributes only once

The different reflectance coefficients can differ

1. Simple “metal”: Ka and Kb share material color, ks is white
2. Simple “plastic”: Ks also include material color

When cosine is negative, light term is 0.

## Difference between Gouraud Shading and Phong Shading

Gouraud Shading:

Applying the phong model on a subset of surface points and interpolates the intesnsity of the remaining points on the surface.

Phong Shading:

A normal vector is linearly interpolated across the surface of the polygon from the polygon’s vertex normal. Phong shading is more computationally expensive than Gouraud shading since the reflection model must be computed at each pixel instead of at each vertex.

## During Implementing:

Gouraud shading is per-vertex color computation, in other words, the vertex shader is supposed to determine a color for each vertex and pass the color as an out variable to the fragment shader as an in varying variable, it is interpolated across the fragments. Therefore, it should give the smooth shading.

Fragment

Shader

Calculate Color

Vertex

Shader

Color gl\_FragColor color

Frame

Buffer

## The Gouraud Shading vertex shader:

out vec3 vertex\_color;

void main(){

vec3 v = vec3(gl\_ModelViewMatrix \* gl\_Vertex);

vec3 N = normalize(gl\_NormalMatrix \* gl\_Normal);

vec3 L = normalize(gl\_LightSource[0].position.xyz - v);

vec3 E = normalize(-v); // we are in Eye Coordinates, so EyePos is (0,0,0)

vec3 R = normalize(-reflect(L,N));

//calculate Ambient Term:

vec4 Iamb = gl\_FrontLightProduct[0].ambient;

//calculate Diffuse Term:

vec4 Idiff = gl\_FrontLightProduct[0].diffuse \* max(dot(N,L), 0.0);

// calculate Specular Term:

vec4 Ispec = gl\_FrontLightProduct[0].specular

\* pow(max(dot(R,E),0.0),0.3\*gl\_FrontMaterial.shininess);

vertex\_color = gl\_FrontLightModelProduct.sceneColor + Iamb + Idiff + Ispec;

gl\_Position = gl\_ModelViewProjectionMatrix \* gl\_Vertex;

}

## The fragment shader:

in vec3 vertex\_color;

void main (void)

{

gl\_FragColor = vertex\_color;

}

However, Phong shading is a per-fragment color computation. The vertex shader provides the normal and position data as out variables to the fragment shader. The fragment shader then interpolates the variables and computers the color.

Calculate Color

Fragment

Shader

Vertex

Shader

Normal Data gl\_FragColor color

Frame

Buffer

Position Data

## The Phong Shading vertex shader:

varying vec3 N;

varying vec3 v;

void main(void)

{

v = vec3(gl\_ModelViewMatrix \* gl\_Vertex);

N = normalize(gl\_NormalMatrix \* gl\_Normal);

gl\_Position = gl\_ModelViewProjectionMatrix \* gl\_Vertex;

}

## The Phong Shading fragment shader:

varying vec3 N;

varying vec3 v;

void main (void)

{

vec3 L = normalize(gl\_LightSource[0].position.xyz - v);

vec3 E = normalize(-v); // we are in Eye Coordinates, so EyePos is (0,0,0)

vec3 R = normalize(-reflect(L,N));

//calculate Ambient Term:

vec4 Iamb = gl\_FrontLightProduct[0].ambient;

//calculate Diffuse Term:

vec4 Idiff = gl\_FrontLightProduct[0].diffuse \* max(dot(N,L), 0.0);

// calculate Specular Term:

vec4 Ispec = gl\_FrontLightProduct[0].specular

\* pow(max(dot(R,E),0.0),0.3\*gl\_FrontMaterial.shininess);

// write Total Color:

gl\_FragColor = gl\_FrontLightModelProduct.sceneColor + Iamb + Idiff + Ispec;

}

Finally, the project was implemented successfully.

Thank you for reading my report patiently.

Referenced from:

[Cs7055 – Real-time Rendering (blackboard.com)](https://tcd.blackboard.com/bbcswebdav/pid-1769555-dt-content-rid-10548990_1/courses/CS7GV3-A-SEM202-202021/CS7GV3-2021-02-LocalIlluminationModels.pdf)

[Surface Reflection Model](http://web.cs.wpi.edu/~emmanuel/courses/cs563/S05/projects/surface_reflection_losasso.pdf)

[LearnOpenGL - Basic Lighting](https://learnopengl.com/Lighting/Basic-Lighting)

[Gouraud-Shading-and-Phong-Shading](https://github.com/ruange/Gouraud-Shading-and-Phong-Shading)