Chapter IX
Lighting
Phong Lighting Model

- Lighting refers to the techniques handling the interaction between light sources and objects.
- The most popular lighting method is based on the Phong model. It is widely adopted in commercial games and lays foundations of various advanced lighting techniques.
- The Phong model is composed of four terms:
  - diffuse
  - specular
  - ambient
  - emissive
There can be various light types such as point, spot, and directional light sources. For now let’s take the simplest, the directional light source, where the light vector \( l \) is constant for a scene. The diffuse term is based on Lambert's law. Reflections from ideally diffuse surfaces (Lambertian surfaces) are scattered with equal intensity in all directions. So, the amount of perceived reflection is independent of the view direction, and is just proportional to the amount of incoming light.
Suppose a white light \((1,1,1)\). If an object lit by the light appears yellow, it means that the object reflects R and G and absorbs B. We can easily implement this kind of filtering through material parameter, i.e., if it is \((1,1,0)\), then \((1,1,1) \odot (1,1,0) = (1,1,0)\) where \(\odot\) is component-wise multiplication.

The diffuse term: \( \max(n \cdot l, 0) s_d \odot m_d \)

In general, the texture provides \(m_d\).
The specular term is used to make a surface look shiny via *highlights*, and it requires *view vector* ($v$) and *reflection vector* ($r$) in addition to *light vector* ($l$).

**Phong Lighting Model - Specular Term**

- Computing the reflection vector
  
  - $s = n \cos \theta - l$
  - $s = r - n \cos \theta$
  - $r = 2n \cos \theta - l$
  - $= 2n (n \cdot l) - l$
Phong Lighting Model - Specular Term (cont’d)

- Whereas the diffuse term is view-independent, the specular term is highly view-dependent.
  - For a perfectly shiny surface, the highlight at \( p \) is visible only when \( \rho \) equals 0.
  - For a surface that is not perfectly shiny, the maximum highlight occurs when \( \rho \) equals 0, but falls off sharply as \( \rho \) increases.
  - The rapid fall-off of highlights is often approximated by \((r \cdot v)^{sh}\), where \( sh \) denotes shininess.

- The specular term: \( (\max(r \cdot v, 0))^{sh} s \times m_s \)
- Unlike \( m_d \), \( m_s \) is usually a gray-scale value rather than an RGB color. It enables the highlight on the surface to end up being the color of the light source.
Phong Lighting Model – Ambient and Emissive Terms

- The ambient light describes the light reflected from the various objects in the scene, i.e., it accounts for *indirect lighting*.

- As the ambient light has bounced around so much in the scene, it arrives at a surface point from all directions, and reflections from the surface point are also scattered with equal intensity in all directions.

- The last term of the Phong model is the emissive term $m_e$ that describes the amount of light emitted by a surface itself.
**Phong Lighting Model**

- The Phong model sums the four terms!!

\[
\text{max}(n \cdot l, 0)s_d \otimes m_d + (\text{max}(r \cdot v, 0))^{sh}s_s \otimes m_s + s_a \otimes m_a + m_e
\]

- (a) diffuse
- (b) specular
- (c) ambient
- (d) emissive
- (e) sum
**Vertex Shader (revisited)**

- Recall our first vertex shader presented in Chapter 6.

```glsl
#version 300 es

uniform mat4 worldMat, viewMat, projMat;

layout(location = 0) in vec3 position;
layout(location = 1) in vec3 normal;
layout(location = 2) in vec2 texCoord;

out vec3 v_normal;
out vec2 v_texCoord;

void main() {
    gl_Position = projMat * viewMat * worldMat * vec4(position, 1.0);
    v_normal = normal;
    v_texCoord = texCoord;
}
```
The inputs to the fragment shader

- Inputs: The per-vertex output variables produced by the vertex shader are interpolated to determine the per-fragment ones.
- Uniforms to be used by the fragment shader.
- Samplers = Textures.

The output: one or more fragment colors that are passed to the output merger.
Fragment Shader

- A simplest fragment shader
  
  ```
  uniform sampler2D s_tex0;

  in vec2 v_texCoord;
  layout(location = 0) out vec4 outColor;

  void main() {
    outColor = texture(s_tex0, v_texCoord);
  }
  ```

- The fragment shader declares a uniform variable, `s_tex0`, of type `sampler2D`, which represents a texture map.

- Our first vertex shader simply outputs 2D texture coordinates without modification. The per-vertex texture coordinates were interpolated so that the per-fragment texture coordinates, `v_texCoord`, are now passed to the fragment shader and used for fetching the texture, `s_tex0`.

- A built-in function `texture` accesses the texture and returns a `vec4` representing the color fetched from the texture. (The fragment shader does not accept another variable, `v_normal`, which will be used later though.)
Texturing alone is never enough to make an object look realistic. Let’s illuminate the object!

The vertex shader for per-fragment lighting

```glsl
#version 300 es

uniform mat4 worldMat, viewMat, projMat;
uniform vec3 eyePos, lightPos;

layout(location = 0) in vec3 position;
layout(location = 1) in vec3 normal;
layout(location = 2) in vec2 texCoord;

out vec3 v_normal;
out vec2 v_texCoord;
out vec3 v_lightDir, v_viewDir;

void main() {
    gl_Position = projMat * viewMat * worldMat * vec4(position, 1.0);
    v_texCoord = texCoord;
    v_normal = mat3(worldMat) * normal; // uniform scaling

    vec3 posWS = (worldMat * vec4(position, 1.0)).xyz;
    v_lightDir = normalize(lightPos - posWS);
    v_viewDir = normalize(eyePos - posWS);
}
```
Per-fragment Lighting (cont’d)

- The fragment shader.

```glsl
#version 300 es
precision mediump float;

uniform sampler2D s_tex0;

uniform vec3 materialDiff, materialSpec, materialAmbi, materialEmit; // Md, Ms, Ma, Me
uniform float materialSh;
uniform vec3 sourceDiff, sourceSpec, sourceAmbi; // Sd, Ss, Sa

in vec3 v_normal;
in vec2 v_texCoord;
in vec3 v_lightDir, v_viewDir;

out vec4 fragColor;

struct Material {
  float sh; // shininess (specular power)
  vec3 diff, spec, ambi, emit; // material colors
};

struct Light {
  vec3 dir, diff, spec, ambi; // light direction and colors
};

max(n \cdot l, 0)s_d \otimes m_d + (max(r \cdot v, 0))^{sh}s_s \otimes m_s + s_a \otimes m_a + m_e
```
Per-fragment Lighting (cont’d)

- The fragment shader.

\[ \max(n \cdot l, 0)s_d \otimes m_d + \left(\max(r \cdot v, 0)\right)^{sh}s_s \otimes m_s + s_a \otimes m_a + m_e \]

```glsl
cvec3 phongLight(vec3 view, vec3 normal, Material M, Light S) {
    float diff = max(dot(normal, S.dir), 0.0);
    vec3 refl = 2.0 * normal * dot(normal, S.dir) - S.dir;
    float spec = 0.0;
    if (diff > 0.0) spec = pow(max(dot(refl, view), 0.0), M.sh);
    vec3 sum = vec3(0.0);
    sum += diff * S.diff * M.diff; // add diffuse term
    sum += spec * S.spec * M.spec; // add specular term
    sum += S.ambi * M.ambi + M.emit; // add ambient and emissive term
    return sum;
}

void main() {
    vec3 materialDiff = texture(s_tex0, v_texCoord).xyz;

    Material material = Material(materialSh, materialDiff, materialSpec,
        materialAmbi, materialEmit);
    Light source = Light(normalize(v.lightDir), sourceDiff, sourceSpec,
        sourceAmbi);

    vec3 color = phongLight(normalize(v.viewDir), normalize(v.normal),
        material, source);
    fragColor = vec4(color, 1.0);
}
```
Ivan Sutherland

- Kendall Station at Boston (MIT)

"Eliza," a computer program, is designed by MIT’s Professor Joseph Weizenbaum to simulate a psychotherapy session.

Ivan E. Sutherland, ’63, designs “Sketchpad” at MIT, the predecessor of computer graphics and design systems.
Ivan Sutherland (cont’d)

**Turing Award**

1966 A. J. Perlis  
1969 Marvin Minsky  
1971 John McCarthy  
1972 E.W. Dijkstra  
1975 Herbert A. Simon  
1977 John Backus  
1980 C. Antony R. Hoare  
1983 Ken Thompson  
1984 Niklaus Wirth  
1988 Ivan Sutherland  
1999 F. Brooks  
2002 R. Rivest, A. Shamir, L. Adleman

**Steven A. Coons Award**

1983 Ivan Sutherland  
1985 Pierre Bezier  
1987 Donald Greenberg  
1989 David Evans  
1991 Andries van Dam  
1993 Ed Catmull  
1995 Jose Luis Encarnacao  
1997 James Foley  
1999 James Blinn  
2001 Lance Williams  
...
Ivan Sutherland (cont’d)

- Ivan Sutherland's students
  - Edwin Catmull - founder of Pixar and now president of Walt Disney and Pixar Animation Studios
  - James H. Clark - founder of Silicon Graphics and Netscape
  - John Warnock - founder of Adobe
  - Nolan Bushnell - founder of Atari
  - Alan Kay - inventor of the Smalltalk language
  - Henri Gouraud - inventor of Gouraud shading algorithm (per-vertex lighting with Phong model)
  - Bui Tuong Phong - inventor of Phong shading algorithm (per-fragment lighting with Phong model)
Local vs. Global Illumination

- The lighting models are divided into two categories.
  - Local illumination considers only direct lighting in the sense that the illumination of a surface depends solely on the properties of the light sources and the surface materials. This has been dominant in real-time graphics.
  - In the real world, however, every surface receives light indirectly. (Even though a light source is invisible from a particular point of the scene, light can still be transferred to the point through reflections or refractions from other surfaces of the scene.) For indirect lighting, the global illumination (GI) model considers the scene objects as potential lighting sources.

- The cost for GI is often too high to permit interactivity. The rasterization-based architecture of GPU is more suitable for local illumination.
When a primary ray is shot and intersects an object, three secondary rays would be spawned: a shadow ray, a reflection ray, and a refraction ray.

Global Illumination – Ray Tracing
The refraction ray is computed using the Snell’s law.
Global Illumination – Ray Tracing (cont’d)