Lighting and Shading

Outline
- Global and Local Illumination
- Normal Vectors
- Light Sources
- Phong Illumination Model
- Polygonal Shading
- Example

Global Illumination
- Ray tracing
- Radiosity
- Photon Mapping
- Follow light rays through a scene
- Accurate, but expensive (off-line)

Local Illumination
- Approximate model
- Local interaction between light, surface, viewer
- Phong model (this lecture): fast, supported in OpenGL
- GPU shaders
- Pixar Renderman (offline)

Light Sources
Phong Illumination Model
Normal Vectors
[Angel Ch. 5]
Local Illumination

- Approximate model
- Local interaction between light, surface, viewer
- Color determined only based on surface normal, relative camera position and relative light position
- What effects does this ignore?

Outline

- Global and Local Illumination
- Normal Vectors
- Light Sources
- Phong Illumination Model
- Polygonal Shading
- Example

Normal Vectors

- Must calculate and specify the normal vector
  - Even in OpenGL!
- Two examples: plane and sphere

Normals of a Plane, Method I

- Method I: given by \( ax + by + cz + d = 0 \)
- Let \( p_0 \) be a known point on the plane
- Let \( p \) be an arbitrary point on the plane
- Recall: \( u \times v \) orthogonal to \( u \) and \( v \)
- \( n_0 = (p_1 - p_0) \times (p_2 - p_0) \)
- Order of cross product determines orientation
- Normalize to \( n = n_0/|n_0| \)

Normals of a Plane, Method II

- Method II: plane given by \( p_0, p_1, p_2 \)
- Points must not be collinear
- Recall: \( u \times v \) orthogonal to \( u \) and \( v \)
- \( n_0 = (p_1 - p_0) \times (p_2 - p_0) \)
- Normalize to \( n = n_0/|n_0| \)

Normals of Sphere

- Implicit Equation \( f(x, y, z) = x^2 + y^2 + z^2 - 1 = 0 \)
- Vector form: \( f(p) = p \cdot p - 1 = 0 \)
- Normal given by gradient vector
  \[
  n_0 = \begin{bmatrix}
  \frac{\partial f}{\partial x} \\
  \frac{\partial f}{\partial y} \\
  \frac{\partial f}{\partial z}
  \end{bmatrix} = \begin{bmatrix}
  2x \\
  2y \\
  2z
  \end{bmatrix} = 2p
  \]
- Normalize \( n_0/|n_0| = 2p/2 = p \)
Reflected Vector
- Perfect reflection: angle of incident equals angle of reflection
- Also: $l$, $n$, and $r$ lie in the same plane
- Assume $|l| = |n| = 1$, guarantee $|r| = 1$
\[
\mathbf{l} \cdot \mathbf{n} = \cos(\theta) = \mathbf{n} \cdot \mathbf{r}
\]
\[
\mathbf{r} = \alpha \mathbf{l} + \beta \mathbf{n}
\]
Solution: $\alpha = -1$ and $\beta = 2 (\mathbf{l} \cdot \mathbf{n})$
\[
\mathbf{r} = 2 (\mathbf{l} \cdot \mathbf{n}) \mathbf{n} - \mathbf{l}
\]

Normals Transformed by Modelview Matrix
Modelview matrix $M$ (shear in this example)
Only keep linear transform in $M$ (discard any translation).

Normals Transformed by Modelview Matrix
When $M$ is rotation, $M = (M^*)^T$

Normals Transformed by Modelview Matrix (proof of $(M^*)^T$ transform)
Point $(x,y,z,w)$ is on a plane in 3D (homogeneous coordinates) if and only if
\[
a x + b y + c z + d w = 0, \text{ or } [a \ b \ c \ d] [x \ y \ z \ w]^T = 0.
\]
Now, let’s transform the plane by $M$.
Point $(x,y,z,w)$ is on the transformed plane if and only if
$M^T [x \ y \ z \ w]^T$ is on the original plane:
\[
[a \ b \ c \ d] M^T [x \ y \ z \ w]^T = 0.
\]
So, equation of transformed plane is
\[
[a' \ b' \ c' \ d'] [x \ y \ z \ w]^T = 0,
\]
for
\[
[a' \ b' \ c' \ d']^T = (M^*)^T [a \ b \ c \ d]^T.
\]

Outline
- Global and Local Illumination
- Normal Vectors
- Light Sources
- Phong Illumination Model
- Polygonal Shading
- Example

Light Sources and Material Properties
- Appearance depends on
  - Light sources, their locations and properties
  - Material (surface) properties:
    - Viewer position
Types of Light Sources

- Ambient light: no identifiable source or direction
- Point source: given only by point
- Distant light: given only by direction
- Spotlight: from source in direction
  - Cut-off angle defines a cone of light
  - Attenuation function (brighter in center)

Point Source

- Given by a point \( p_0 \)
- Light emitted equally in all directions
- Intensity decreases with square of distance

\[
I \propto \frac{1}{|p - p_0|^2}
\]

Limitations of Point Sources

- Shading and shadows inaccurate
- Example: penumbra (partial “soft” shadow)
- Similar problems with highlights
- Compensate with attenuation
  \[
  q = \text{distance} |p - p_0| \\
  a, b, c \text{ constants}
  \]
- Softens lighting
- Better with ray tracing
- Better with radiosity

Distant Light Source

- Given by a direction vector \([x \ y \ z]\)

Spotlight

- Light still emanates from point
- Cut-off by cone determined by angle \( \theta \)

Global Ambient Light

- Independent of light source
- Lights entire scene
- Computationally inexpensive
- Simply add \([G_R \ G_G \ G_B]\) to every pixel on every object
- Not very interesting on its own. A cheap hack to make the scene brighter.
Outline

- Global and Local Illumination
- Normal Vectors
- Light Sources
- Phong Illumination Model
- Polygonal Shading
- Example

Phong Illumination Model

- Calculate color for arbitrary point on surface
- Compromise between realism and efficiency
- Local computation (no visibility calculations)
- Basic inputs are material properties and I, n, v:
  - I = unit vector to light source
  - n = surface normal
  - v = unit vector to viewer
  - r = reflection of I at p
  (determined by I and n)

Phong Illumination Overview

2. Add contributions from each light source
3. Clamp the final result to [0, 1]

- Calculate each color channel (R,G,B) separately
- Light source contributions decomposed into
  - Ambient reflection
  - Diffuse reflection
  - Specular reflection
- Based on ambient, diffuse, and specular lighting and material properties

Ambient Reflection

\[ I_a = k_a L_a \]

- Intensity of ambient light is uniform at every point
- Ambient reflection coefficient \( k_a \geq 0 \)
- May be different for every surface and r,g,b
- Determines reflected fraction of ambient light
- \( L_a \) = ambient component of light source
  (can be set to different value for each light source)
- Note: \( L_a \) is not a physically meaningful quantity

Diffuse Reflection

- Diffuse reflector scatters light
- Assume equally all direction
- Called Lambertian surface
- Diffuse reflection coefficient \( k_d \geq 0 \)
- Angle of incoming light is important

Lambert's Law

Intensity depends on angle of incoming light.

\[ I = \frac{I_a}{d^2} \]

\[ \frac{d}{\cos \theta} = \frac{d}{d} \]

(a) (b)
Diffuse Light Intensity Depends On Angle Of Incoming Light

- Recall
  \[ l = \text{unit vector to light} \]
  \[ n = \text{unit surface normal} \]
  \[ \theta = \text{angle to normal} \]
  \[ \cos \theta = l \cdot n \]
  \[ I_d = k_d L_d (l \cdot n) \]

- With attenuation:
  \[ I_d = \frac{k_d L_d}{a + bq + cq^2} (l \cdot n) \]  
  \( q = \text{distance to light source,} \)  
  \( L_d = \text{diffuse component of light} \)

Specular Reflection

- Specular reflection coefficient \( k_s \geq 0 \)
- Shiny surfaces have high specular coefficient
- Used to model specular highlights
- Does not give the mirror effect (need other techniques)

Specular Reflection

- Recall
  \[ v = \text{unit vector to camera} \]
  \[ r = \text{unit reflected vector} \]
  \[ \phi = \text{angle between} v \text{ and } r \]
  \[ \cos \phi = v \cdot r \]
  \[ I_s = k_s L_s (\cos \phi)^\alpha \]

- \( L_s \) is specular component of light
- \( \alpha \) is shininess coefficient
- Can add distance term as well

Shininess Coefficient

- \( I_s = k_s L_s (\cos \phi)^\alpha \)
- \( \alpha \) is the shininess coefficient
- Higher \( \alpha \) gives narrower curves

Summary of Phong Model

- Light components for each color:
  - Ambient \( (L_a) \), diffuse \( (L_d) \), specular \( (L_s) \)
- Material coefficients for each color:
  - Ambient \( (k_a) \), diffuse \( (k_d) \), specular \( (k_s) \)
- Distance \( q \) for surface point from light source
  \[ I = \frac{1}{a + bq + cq^2} (k_d L_d (l \cdot n) + k_s L_s (r \cdot v)^\alpha) + k_d L_d \]  
  \( I = \text{unit vector to light} \)  
  \( r = I \text{ reflected about} n \)  
  \( n = \text{surface normal} \)  
  \( v = \text{vector to viewer} \)

BRDF

- Bidirectional Reflection Distribution Function
- Must measure for real materials
- Isotropic vs. anisotropic
- Mathematically complex
- Implement in a fragment shader
Outline

• Global and Local Illumination
• Normal Vectors
• Light Sources
• Phong Illumination Model
• Polygonal Shading
• Example

Polygonal Shading

• Now we know vertex colors
  – either via OpenGL lighting,
  – or by setting directly via glColor3f if lighting disabled

• How do we shade the interior of the triangle?

Polygonal Shading

• Curved surfaces are approximated by polygons

• How do we shade?
  – Flat shading
  – Interpolative shading
  – Gouraud shading
  – Phong shading (different from Phong illumination!)

Flat Shading

• Shading constant across polygon

• Core profile: Use interpolation qualifiers in the fragment shader

• Compatibility profile: Enable with glShadeModel(GL_FLAT);

• Color of last vertex determines interior color

• Only suitable for very small polygons

Flat Shading Assessment

• Inexpensive to compute
• Appropriate for objects with flat faces
• Less pleasant for smooth surfaces

Interpolative Shading

• Interpolate color in interior

• Computed during scan conversion (rasterization)

• Core profile: enabled by default

• Compatibility profile: enable with glShadeModel(GL_SMOOTH);

• Much better than flat shading
• More expensive to calculate (but not a problem)
Gouraud Shading
Invented by Henri Gouraud, Univ. of Utah, 1971
- Special case of interpolative shading
- How do we calculate vertex normals for a polygonal surface? Gouraud:
  1. average all adjacent face normals
     \[ n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|} \]
  2. use \( n \) for Phong lighting
  3. interpolate vertex colors into the interior
- Requires knowledge about which faces share a vertex

Data Structures for Gouraud Shading
- Sometimes vertex normals can be computed directly (e.g. height field with uniform mesh)
- More generally, need data structure for mesh
- Key: which polygons meet at each vertex

Phong Shading (“per-pixel lighting”)
Invented by Bui Tuong Phong, Univ. of Utah, 1973
- At each pixel (as opposed to at each vertex):
  1. Interpolate normals (rather than colors)
  2. Apply Phong lighting to the interpolated normal
- Significantly more expensive
- Done off-line or in GPU shaders (not supported in OpenGL directly)

Phong Shading Results
Michael Gold, Nvidia

Outline
- Global and Local Illumination
- Normal Vectors
- Light Sources
- Phong Illumination Model
- Polygonal Shading
- Example

Phong Shader: Vertex Program
#version 150
in vec3 position;  // input vertex position and normal, in world-space
in vec3 normal;   
out vec3 viewPosition;  // vertex position and normal, in view-space
out vec3 viewNormal;  // these will be passed to fragment program (interpolated by hardware)
uniform mat4 modelViewMatrix;
uniform mat4 normalMatrix;
uniform mat4 projectionMatrix;
Phong Shader: Vertex Program

```cpp
void main()
{
    // view-space position of the vertex
    vec4 viewPosition4 = modelViewMatrix * vec4(position, 1.0f);
    viewPosition = viewPosition4.xyz;

    // final position in the normalized device coordinates space
    gl_Position = projectionMatrix * viewPosition4;
}
```

Phong Shader: Fragment Program

```cpp
void main()
{
    // camera is at (0,0,0) after the modelview transformation
    vec3 eyedir = normalize(vec3(0, 0, 0) - viewPosition);
    // reflected light direction
    vec3 reflectDir = -reflect(viewLightDirection, viewNormal);
    // Phong lighting
    float d = max(dot(viewLightDirection, viewNormal), 0.0f);
    float s = max(dot(reflectDir, eyedir), 0.0f);
    // compute the final color
    c = ka * La + d * kd * Ld + pow(s, alpha) * ks * Ls;
}
```

VBO Layout: positions and normals

```cpp
VBO
```

VAO code ("normal" shader variable)

```cpp
During initialization:
```

Upload the light direction vector to GPU

```cpp
void display()
{
    // get a handle to the program
    GLuint program = pipelineProgram->GetProgramHandle();
    openGLMatrix->LookAt(ex, ey, ez, fx, fy, fz, ux, uy, uz);
    float view[16];
    openGLMatrix->GetMatrix(view); // read the view matrix
    // get a handle to the pipeline program
    GLuint pipelineProgram = windowProgram->GetProgramHandle();
    // get a handle to the viewLightDirection shader variable
    GLuint h_viewLightDirection = glGetUniformLocation(program, "viewLightDirection");
    // bind the VBO "buffer" (must be previously created)
    glBindBuffer(GL_ARRAY_BUFFER, buffer);
    // set the layout of the "normal" attribute data
    glVertexAttribPointer(loc, 3, GL_FLOAT, normalized, stride, offset);
    glEnableVertexAttribArray(loc); // enable the "normal" attribute
    // get a handle to the "normal" shader variable
    GLuint loc = glGetAttribLocation(program, "normal");
    glEnableVertexAttribArray(loc); // enable the "normal" attribute
    vidBindBuffer(GL_ARRAY_BUFFER, buffer);
    // bind the VBO "buffer" (must be previously created)
    vidBindBuffer(GL_ARRAY_BUFFER, buffer);
    // bind the VAO
    glBindVertexArray(vao);
    // bind the VBO "buffer" (must be previously created)
    glBindBuffer(GL_ARRAY_BUFFER, buffer);
    // set the layout of the "normal" attribute data
    glVertexAttribPointer(loc, 3, GL_FLOAT, normalized, stride, offset);
    glEnableVertexAttribArray(loc); // enable the "normal" attribute
    ```
Upload the light direction vector to GPU

```c
float lightDirection[3] = { 0, 1, 0 }; // the "Sun" at noon
float viewLightDirection[3]; // light direction in the view space
// the following line is pseudo-code:
viewLightDirection = (view * float4(lightDirection, 0.0)).xyz;
// upload viewLightDirection to the GPU
glUniform3fv(h_viewLightDirection, 1, viewLightDirection);
// continue with model transformations
openGLMatrix->Translate(x, y, z);
...
renderBunny(); // render, via VAO
glutSwapBuffers();
```

Upload the normal matrix to GPU

```c
// in the display function:

// get a handle to the program
GLuint program = pipelineProgram->GetProgramHandle();
// get a handle to the normalMatrix shader variable
GLint h_normalMatrix =
    glGetUniformLocation(program, "normalMatrix");
float n[16];
matrix->SetMatrixMode(OpenGLMatrix::ModelView);
matrix->GetNormalMatrix(n); // get normal matrix
// upload n to the GPU
GLboolean isRowMajor = GL_FALSE;
glUniformMatrix4fv(h_normalMatrix, 1, isRowMajor, n);
```

Summary

- Global and Local Illumination
- Normal Vectors
- Light Sources
- Phong Illumination Model
- Polygonal Shading
- Example