Illumination and Shading
Illumination (Lighting)

- Model the interaction of light with surface points to determine their final color and brightness
- OpenGL computes illumination at vertices
Apply the lighting model at a set of points across the entire surface
Illumination Model

- The governing principles for computing the illumination
- A illumination model usually considers:
  - Light attributes (light intensity, color, position, direction, shape)
  - Object surface attributes (color, reflectivity, transparency, etc)
  - Interaction among lights and objects (object orientation)
  - Interaction between objects and eye (viewing dir.)
Illumination Calculation

- **Local illumination:** only consider the light, the observer position, and the object material properties

  ![Diagram of light and object]

  \[ \theta \]

- **Example:** OpenGL
Global illumination: take into account the interaction of light from all the surfaces in the scene

Example: Ray Tracing (CIS681)
Basic Light Sources

- **Point light**
- **Directional light**
- **Spot light**

Light intensity can be independent or dependent of the distance between object and the light source.
Simple local illumination

- The model used by OpenGL – consider three types of light contribution to compute the final illumination of an object
  - Ambient
  - Diffuse
  - Specular

- Final illumination of a point (vertex) = ambient + diffuse + specular
Ambient light contribution

- Ambient light (background light): the light that is scattered by the environment
- A very simple approximation of global illumination

- Independent of the light position, object orientation, observer’s position or orientation – ambient light has no direction (Radiosity is the calculation of ambient light)
Ambient lighting example
Each light source has an ambient light contribution \(( I_a )\)

Different objects can reflect different amounts of ambient (different ambient reflection coefficient \( K_a \), \( 0 \leq K_a \leq 1 \))

So the amount of ambient light that can be seen from an object is:

\[
\text{Ambient} = I_a \times K_a
\]
Diffuse light contribution

- Diffuse light: The illumination that a surface receives from a light source and reflects equally in all directions.

It does not matter where the eye is.
Diffuse lighting example
Diffuse light calculation

- Need to decide how much light the object point receive from the light source – based on Lambert’s Law
Lambert’s law: the radiant energy $D$ that a small surface patch receives from a light source is:

$$D = I \cdot \cos(\theta)$$

$I$: light intensity

$\theta$: angle between the light vector and the surface normal

light vector (vector from object to light)

$N$: surface normal
Diffuse light calculation (3)

- Like the ambient light case, different objects can reflect different amount of diffuse light (different diffuse reflection coefficient $K_d$, $0 \leq K_d \leq 1$)

- So, the amount of diffuse light that can be seen is:

$$\text{Diffuse} = K_d \times I \times \cos(\theta)$$

$$\cos(\theta) = N \cdot L$$
Specular light contribution

- The bright spot on the object
- The result of total reflection of the incident light in a concentrate region

See nothing!
Specular light example
Specular light calculation

How much reflection you can see depends on where you are.

The only position the eye can see specular from P if the object has an ideal reflection surface.

But for a non-perfect surface you will still see specular highlight when you move a little bit away from the idea reflection direction.

When $\phi$ is small, you see more specular highlight.
Specular light calculation (2)

- Phong lighting model

\[
\text{specular} = K_s \times I \times \cos^n(\phi)
\]

- \(K_s\): specular reflection coefficient
- \(N\): surface normal at \(P\)
- \(I\): light intensity
- \(\phi\): angle between \(V\) and \(R\)

\(\cos(\phi)\): the larger is \(n\), the smaller is the \(\cos\) value
\(\cos(\theta) = R \cdot V\)
Specular light calculation (3)

- The effect of ‘n’ in the phong model

n = 10
n = 90

n = 30
n = 270
Put it all together

- Illumination from a light:
  \[ \text{Illum} = \text{ambient} + \text{diffuse} + \text{specular} \]
  \[ = K_a * I + K_d * I * (N.L) + K_s * I * (R.V)^n \]

- If there are N lights

  Total illumination for a point \( P = \sum \text{Illum} \) \( (N.H) \)

- Some more terms to be added (in OpenGL):
  - Self emission
  - Global ambient
  - Light distance attenuation and spot light effect
Lighting in OpenGL

- Adopt Phong lighting model (specular) plus diffuse and ambient lights
  - Lighting is computed at vertices
    - Interpolate across surface (Gouraud/smooth shading) OR
    - Use a constant illumination (get it from one of the vertices)

- Setting up OpenGL Lighting:
  - Light Properties
  - Enable/Disable lighting
  - Surface material properties
  - Provide correct surface normals
  - Light model properties
Light Properties

Properties:
- Colors / Position and type / attenuation

`glLightfv(light, property, value)`

1. (1) constant: specify which light you want to set the property
   example: `GL_LIGHT0`, `GL_LIGHT1`, `GL_LIGHT2` … you can create multiple lights (OpenGL allows at least 8 lights)

2. (2) constant: specify which light property you want to set the value
   example: `GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR`, `GL_POSITION`
   (check the red book for more)

3. (3) The value you want to set to the property
Define colors and position a light

```c
GLfloat light_ambient[] = {0.0, 0.0, 0.0, 1.0};
GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_position[] = {0.0, 0.0, 1.0, 1.0};

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

What if I set the Position to (0,0,1,0)?
Types of lights

- OpenGL supports two types of lights
  - Local light (point light)
  - Infinite light (directional light)
- Determined by the light positions you provide
  - $w = 0$: infinite light source (faster)
  - $w \neq 0$: point light – position = ($x/w, y/w, z/w$)

```c
GLfloat light_position[] = {x,y,z,w};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```
Turning on the lights

- Turn on the power (for all the lights)
  - glEnable(GL_LIGHTING);
  - glDisable(GL_LIGHTING);

- Flip each light’s switch
  - glEnable(GL_LIGHTn) (n = 0,1,2,...)
Controlling light position

- Modelview matrix affects a light’s position
- You can specify the position relative to:
  - Eye space: the highlight remains in the same position relative to the eye
    - call glLightfv() before gluLookAt()
  - World space: a light’s position/direction appears fixed in the scene
    - Call glLightfv() after gluLookAt()

- See Nat Robin’s Demo
  http://www.xmission.com/~nate/tutors.html
Material Properties

- The color and surface properties of a material (dull, shiny, etc)
- How much the surface reflects the incident lights (ambient/diffuse/specular reflection coefficients)

```c
glMaterialfv(face, property, value)
```

- **Face**: material property for which face (e.g. GL_FRONT, GL_BACK, GL_FRONT_AND_BACK)
- **Property**: what material property you want to set (e.g. GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_SHININESS, GL_EMISSION, etc)
- **Value**: the value you can to assign to the property
Material Example

- Define ambient/diffuse/specular reflection and shininess

```c
GLfloat mat_amb_diff[] = {1.0, 0.5, 0.8, 1.0};
GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat shininess[] = {5.0};  // (range: dull 0 – very shiny 128)

glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, mat_amb_diff);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, shininess);
```
Global light properties

`glLightModelfv(property, value)`

- **Enable two sided lighting**
  - property = `GL_LIGHT_MODEL_TWO_SIDE`
  - value = `GL_TRUE` (`GL_FALSE` if you don’t want two sided lighting)

- **Global ambient color**
  - Property = `GL_LIGHT_MODEL_AMBIENT`
  - Value = (red, green, blue, 1.0);

- Check the red book for others
Surface Normals

- Correct normals are essential for correct lighting
- Associate a normal to each vertex

```c
glBegin(…)
  glNormal3f(x,y,z)
  glVertex3f(x,y,z)
  …
  glEnd()
```

- The normals you provide need to have a unit length
  - You can use `glEnable(GL_NORMALIZE)` to have OpenGL normalize all the normals
Lighting revisit

- Where is lighting performed in the graphics pipeline?
Polygon shading model

- Flat shading – compute lighting once and assign the color to the whole polygon
Flat shading

- Only use one vertex (usually the first one) normal and material property to compute the color for the polygon
- Benefit: fast to compute
- It is used when:
  - The polygon is small enough
  - The light source is far away (why?)
  - The eye is very far away (why?)
- OpenGL command: `glShadeModel(GL_FLAT)`
Mach Band Effect

- Flat shading suffers from "mach band effect"
- Mach band effect – human eyes accentuate the discontinuity at the boundary

perceived intensity

Side view of a polygonal surface
Smooth shading

- Reduce the mach band effect – remove value discontinuity
- Compute lighting for more points on each face

Flat shading  →  smooth shading
Smooth shading

- Two popular methods:
  - Gouraud shading (used by OpenGL)
  - Phong shading (better specular highlight, not supported by OpenGL)
Gouraud Shading (1)

- The smooth shading algorithm used in OpenGL
  \text{glShadeModel(GL\_SMOOTH)}
- Lighting is calculated for each of the polygon vertices
- Colors are interpolated for interior pixels
Gouraud Shading (2)

- Per-vertex lighting calculation
- Normal is needed for each vertex
- Per-vertex normal can be computed by averaging the adjacent face normals

\[ n = \frac{(n_1 + n_2 + n_3 + n_4)}{4.0} \]
Gouraud Shading (3)

- Compute vertex illumination (color) before the projection transformation
- Shade interior pixels: color interpolation (normals are not needed)

\[ C_a = \text{lerp}(C_1, C_2) \]
\[ C_b = \text{lerp}(C_1, C_3) \]
\[ \text{Lerp}(C_a, C_b) \]

for all scanlines

* lerp: linear interpolation
Gouraud Shading (4)

- Linear interpolation

\[ x = \frac{a}{a+b} \cdot v_2 + \frac{b}{a+b} \cdot v_1 \]

- Interpolate triangle color: use y distance to interpolate the two end points in the scanline, and use x distance to interpolate interior pixel colors
Gouraud Shading Problem

- Lighting in the polygon interior can be inaccurate
Gouraud Shading Problem

- Lighting in the polygon interior can be inaccurate
Phong Shading

- Instead of color interpolation, we calculate lighting for each pixel inside the polygon (per pixel lighting)
- We need to have normals for all the pixels - not provided by the user
- Phong shading algorithm interpolates the normals and compute lighting during rasterization (need to map the normal back to world or eye space though - WHY?)
Phong Shading (2)

- Normal interpolation

\[
\begin{align*}
na &= \text{lerp}(n_1, n_2) \\
\text{lerp}(na, nb) &= \text{lerp}(n_1, n_3)
\end{align*}
\]

- Slow – not supported by OpenGL and most of the graphics hardware