Adding Color

- Sometimes light or surfaces are colored
- Treat R, G and B components separately
- Illumination equation goes from:
  \[ \text{Illum} = \text{ambient} + \text{diffuse} + \text{specular} \]
  \[ = K_a \times I + K_d \times I \times (N \cdot L) + K_s \times I \times (R \cdot V) \]
- To:
  \[ \text{Illum}_r = K_{a_r} \times I_r + K_{d_r} \times I_r \times (N \cdot L) + K_{s_r} \times I_r \times (R \cdot V) \]
  \[ \text{Illum}_g = K_{a_g} \times I_g + K_{d_g} \times I_g \times (N \cdot L) + K_{s_g} \times I_g \times (R \cdot V) \]
  \[ \text{Illum}_b = K_{a_b} \times I_b + K_{d_b} \times I_b \times (N \cdot L) + K_{s_b} \times I_b \times (R \cdot V) \]

### Adding Color

<table>
<thead>
<tr>
<th>Material</th>
<th>Ambient $K_a$</th>
<th>Diffuse $K_d$</th>
<th>Specular $K_s$</th>
<th>Exponent, n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black plastic</td>
<td>0.0</td>
<td>0.01</td>
<td>0.5</td>
<td>32</td>
</tr>
<tr>
<td>0.0</td>
<td>0.01</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brass</td>
<td>0.329412</td>
<td>0.780392</td>
<td>0.992157</td>
<td>27.8974</td>
</tr>
<tr>
<td>0.235292</td>
<td>0.568627</td>
<td>0.941176</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.274911</td>
<td>0.113728</td>
<td>0.807843</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Polished Silver</td>
<td>0.23125</td>
<td>0.2775</td>
<td>0.773911</td>
<td>89.6</td>
</tr>
<tr>
<td>0.23125</td>
<td>0.2775</td>
<td>0.773911</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.2775</td>
<td>0.773911</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.17, Hill, courtesy of McReynolds and Blythe

### Lighting in OpenGL

- Adopt Phong lighting model
  - specular + diffuse + ambient lights
  - Lighting is computed at vertices
    - Interpolate across surface (Gouraud/smooth shading)
    - 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
  
  \[ \text{glLightfv(light, property, value)} \]

  1. Constant: specify which light you want to set the property
     E.g: GL_LIGHT0, GL_LIGHT1, GL_LIGHT2 ... you can create multiple lights (OpenGL allows at least 8 lights)
  2. Constant: specify which light property you want to set the value
     E.g: GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_POSITION
     (check the red book for more)
  3. The value you want to set to the property

Property Example

- Define colors and position a light

  \[
  \begin{align*}
  \text{GLfloat light_ambient[]} &= \{0.0, 0.0, 0.0, 1.0\}; \\
  \text{GLfloat light_diffuse[]} &= \{1.0, 1.0, 1.0, 1.0\}; \\
  \text{GLfloat light_specular[]} &= \{1.0, 1.0, 1.0, 1.0\}; \\
  \text{GLfloat light_position[]} &= \{0.0, 0.0, 1.0, 1.0\}; \\
  \end{align*}
  \]

  What if I set 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

  \[
  \begin{align*}
  w &= 0: \text{infinite light source (faster)} \\
  w \neq 0: \text{point light - position} = (x/w, y/w, z/w)
  \end{align*}
  \]

Turning on the lights

- Turn on the power (for all the lights)

  \[
  \begin{align*}
  \text{glEnable(GL_LIGHTING);} \\
  \text{glDisable(GL_LIGHTING);} \\
  \end{align*}
  \]

- Flip each light's switch

  \[
  \begin{align*}
  \text{glEnable(GL_LIGHTn)} (n = 0,1,2,...)
  \end{align*}
  \]
Controlling light position

- Modelview matrix affects a light’s position
- Two options:
  - Option a:
    - Treat light like vertex
    - Do pushMatrix, translate, rotate, ...`glLightfv` position, popMatrix
    - Then call `gluLookat`
    - Light moves independently of camera
  - Option b:
    - Load identity matrix in modelview matrix
    - Call `glLightfv` then call `gluLookat`
    - Light appears at the eye (like a miner’s lamp)

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)
  `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 assign to the property

Material Example

- Define ambient/diffuse/specular reflection and shininess

  ```
  GLfloat mat_amb_diff[] = {1.0, 0.5, 0.8, 1.0};  // refl. coeff.
  GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};  // refl. coeff.
  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
  ```
  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 (mesh) polygon

Flat shading

- Only use one vertex normal and material property to compute the color for the polygon
  - Benefit: fast to compute
  - Used when:
    - Polygon is small enough
    - Light source is far away (why?)
    - 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

Side view of a polygonal surface

Smooth shading

- Fix the mach band effect – remove edge 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 in OpenGL)

Gouraud Shading

- The smooth shading algorithm used in OpenGL
  - glShadeModel(GL_SMOOTH)
- Lighting is calculated for each of the polygon vertices
- Colors are interpolated for interior pixels
Gouraud Shading

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

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

Gouraud Shading

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

\[
Ca = \text{lerp}(C_1, C_2) \quad Cb = \text{lerp}(C_1, C_3)
\]

Gouraud Shading Problem

- Lighting in the polygon interior can be inaccurate

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

Gouraud Shading Problem

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

\[
x = \frac{x}{x+y} \cdot v_1 + \frac{y}{x+y} \cdot v_2
\]
Phong Shading

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

References

- Hill, chapter 8