Illumination, Reflection, Shading

- Need to display surfaces in “natural” colors
  - Colors observed if we really saw the scene
- How do they get those colors?
- Observed Colors Depend on:
  - Light sources in scene
  - Material properties of object surfaces
  - How light interacts with those surfaces
    - Reflection, Transmission, Absorption
- Need an Illumination/Reflection model
Light Sources

- Approximate with two types:
  1. Ambient (non-directional, diffuse, background light)
     - Take as constant in the scene
     - Non-directional
     - Grossly approximates multiply-reflected light
     - “Global” reflection
  2. Light Sources
     - Approximate with a series of point sources
     - Directional

Interaction of Light with Surfaces

- Absorption
- Transmission
- Reflection
  - Diffuse
    - Nondirectional
    - Dull, chalky surfaces
    - No highlights
  - Specular
    - Directional
    - Mirror-like surfaces
    - Highlights
Material Properties

- Incident light is reflected to different degrees
  - Depends on physical (material) properties of reflecting surface
  - This gives intrinsic color to materials
  - Approximate by giving 3 diffuse reflection coefficients
    - Fractions of red, green blue reflected
    - $k_r, k_g, k_b$ ($0 \leq k \leq 1$)
      - 0 means no reflection in that color band
      - 1 means 100% reflection in that band

Phong Illumination/Reflection Model

- Assume all illumination comes from:
  - Ambient Light
  - Point sources
- Diffuse reflection of Ambient light
- Reflection from Point sources:
  - Some is reflected diffusely
  - Some is reflected specularly
Reflection of Ambient Light

- \( I = k_d I_a \)
  - \( I \) = intensity of ambient light reflected
  - \( k_d \) = diffuse/ambient reflection coefficient
    - Assume ambient light is reflected diffusely
  - Actually 3 values of \( k_d \):
    - \( k_r, k_g, k_b \)
    - Values give object its intrinsic color
  - (So this is really three equations)
  - \( I_a \) = Intensity of ambient light in scene
    - Could also have color dependence
    - But for simplicity we'll assume white lights
  - \( I_a, k_r, k_g, k_b \) are adjustable parameters

---

**Diffuse Reflection of a Point Source**

- \( L_i \) = Vector from reflecting point to light source
- \( N \) = Normal vector to surface at reflecting point
- \( I_i \) = Intrinsic intensity of the point source
- \( V \) = Vector from reflecting point to view point
- \( k_d \) = Diffuse reflection coefficient (0-1)

For perfectly diffuse surfaces the intensity of the reflected light is independent of \( V\).

The intensity of light reflected from a diffuse surface depends on the angle between \( N \) and \( L_i \).

\[
I = k_d I_i \cos(\theta)
\]

\[
I = k_d I_i \hat{N} \cdot \hat{L}_i
\]
Three color intensity equations:

\[ I(r,g,b) = Ambient + Point Diffuse + Point Specular \]

\[ I(r,g,b) = kd(r,g,b)I_a \]  (ambient)

\[ + lp*kd(r,g,b)\textbf{(N ∙ L)} \]  (diffuse from point source)

\[ + lp*ks(R ∙ V)^n \]  (specular from pt. Source)

It can be shown that \( R = 2\textbf{(N ∙ L)}\textbf{N} - \textbf{L} \)

where \( \textbf{N} \) and \( \textbf{L} \) are unit vectors

Note that specular term has no color dependency

(First approximation)

If viewer moves, specular term must be recomputed
Computing N and L

Assumes light source is at infinity

If $N \cdot L < 0$, no light received, so only use ambient light

CalcNormal(double v0[3], double v1[3], double v2[3], double n[3])
{
    // Form two vectors from the points v0, v1, v2.
    double a[3], b[3];  // Array elements 0,1,2 are x,y,z components
    // Calculate the cross product of the two vectors.
    double length = sqrt(n[0]*n[0]+n[1]*n[1]+n[2]*n[2]);
    n[0] = n[0] / length;  // Normalize
}
Intensity Computations

• For each polygon
  – Compute I(r), I(g), and I(b) from Phong Formula
  – Scale to Frame Buffer r,g,b values:

\[
\frac{FB(\text{color})}{FB\text{max}} = \frac{I(\text{color})}{I\text{max}}
\]
  • For True color, FBmax=255
  • Imax from formula with all dot products = 1 and maximum values of reflection coefficients

• Paint Polygon with resulting FB(color) values

This is Lambertian Flat Shading
  – All points on a polygon have same color intensity
  – Gives a faceted appearance to all (curved) surfaces

Rendering Process
(Flat Shading)

• 1. Set up polygon model data structures
  – Include information needed for subsequent shading
    • Object list, polygon list, vertex list, lighting (Ia,L,Lp), reflection properties (kr, kg, kb, ks, n)

• 2. Apply chain of transformations to model
  – For each vertex get (xv,yv,zv) and (xs,ys)

• 3. Do Back-Face Culling

• 4. Compute & store polygon colors (Phong model)

• 5. Apply Z-Buffer Algorithm and shade polygons
Data Structures for Flat-Shaded Polygon Mesh Rendering

1. Array of objects
2. Array of polygons
3. Lighting parameters
4. Viewing parameters

- Values could come from a scene file

1. Array of objects (e.g., for object i):

   Object[i].num_pts          // number of vertices in object
   Object[i].w_pts[num_pts]   // vertex 3D world coords
   Object[i].v_pts[num_pts]   // vertex 3D viewing coords
   Object[i].s_pts[num_pts]   // vertex 2D screen coords
   Object[i].num_polys       // number of polygons
   Object[i].polys[num_polys] // array of polygons

   // Diffuse reflection coefficients:
   Object[i].kr; Object[i].kg; Object[i].kb
   Object[i].ks       // Specular reflection coefficient
   Object[i].n        // Specular exponent

- This assumes that all faces of the object have the same reflection properties
2. Array of polygons (e.g., for polygon j):

- `polys[j].num_verts` // Number of vertices in polygon
- `polys[j].inds[num_verts]` // List of polygon vertices
- `polys[j].visibility` // Back-Face culling visibility
- `polys[j].Ired` // Red computed intensity
- `polys[j].Igreen` // Green computed intensity
- `polys[j].Iblue` // Blue computed intensity

- Alternative to storing color intensities: compute and store surface normals
  - `Polys[j].n[3]` // x,y,z components of surface normal
  - Compute color intensities later
  - Would facilitate interpolated smooth shading (see below)

3. Lighting Parameters:

- `Ia` // Ambient Light Intensity (Ia)
- `num_lights` // Number of light sources
- `Lx[k]`, `Ly[k]`, `Lz[k]` // World coordinates of kth light source
- `Ip[k]` // intensity of kth light source
Scene Description Files

- Viewing parameters ($\rho$, $\theta$, $\phi$, scrn_dist)
- Number of objects (num_objs)
- For each object:
  - File name of Generic Object Description File
  - x,y,z scaling factors to be applied to object (sx,sy,sz)
  - rotation angles to be applied to object ($\alpha_z, \alpha_y, \alpha_z$)
  - translation distances to be applied to object (tx,ty,tz)
- Position, Intensity of light sources (Lx,Ly,Lz,Ip)
- Intensity of ambient light (Ia)

Example Scene Description File

```
200, 1000, 45, 60    // scrn_dist, $\rho$, $\theta$, $\phi$
1                              // number of objects in scene
pyramid.des    // name of generic object description file
1.8, 1.0, 1.0    // sx, sy, sz scaling factors
0, 0, 0             // x, y, z, rotation angles
200, 0, 0         // x, y, z translation components
1                     // number of light sources
500, 500, 500, 100   // x,y,z & Intensity of light source
50                   // ambient light intensity
```
Generic Object Description Files

- For each object:
  - Number of points (num_pts)
  - For each point:
    - 3-D world coordinates of point (xw,yw,zw)
  - Number of polygons (num_polys)
  - For each polygon:
    - Number of vertices (num_verts)
    - List of polygon vertices (*inds)
  - Reflection properties:
    - Diffuse reflection coefficients (kr,kg, kb)
    - Specular reflection coefficient & exponent (ks,n)

Example Generic Object Description File

```
// pyramid.des file:

5, 5    // number of vertices and polygons
// World coordinates of pyramid vertices:
(0,0,0), (150,0,0), (150,150,0), (0,150,0), (75,75,150)
// Pyramid polygons:
3,(0,1,4), 3,(1,2,4), 3,(2,3,4), 3,(0,4,3), 4,(0,3,2,1)
0.2, 0.5, 0.9    // kr, kg, kb diffuse reflection coefficients
0.4                  // ks specular reflection coefficient
8                     // n specular exponent
```
Interpolated Shading

- To "fake" curved surfaces
- Easiest way--Gouraud shading:
  - Compute vertex intensities
  - Double Interpolate values across polygon
    - Should be done at same time as Z-Buffer interpolations
  - Gives a curved appearance to surfaces

Interpolated (Gouraud) Shading

1. Calculate the Vertex Normal as the average of the surface normals of the polygons surrounding the vertex.

\[
N_A = \frac{N_1 + N_2 + N_3 + N_4}{4}
\]

2. Calculate a Vertex Intensity for each vertex using the Phong model.

3. When scan converting the polygon, calculate the intensity of a pixel by double interpolation:

\[
l_a = I_1 + \frac{(I_2 - I_1)}{(y_2 - y_1)} \cdot (y_s - y_1)
\]

\[
l_b = I_1 + \frac{(I_4 - I_1)}{(y_4 - y_1)} \cdot (y_s - y_1)
\]

\[
l_s = l_a + \frac{(l_b - l_a)}{(x_b - x_a)} \cdot (x_s - x_a)
\]
Polygon Mesh (Z-Buffer hidden Surface removal + Flat Shading)

Polygon Mesh (Z-Buffer and Flat/Gouraud/Phong shading)
Phong Smooth Shading

- Interpolate the vertex normal vectors
  - Instead of the intensities
  - Means a Phong intensity calculation for each pixel on each polygon
  - Much more computationally intensive
  - But “catches” specular highlights that Gouraud misses
  - More realistic images

<table>
<thead>
<tr>
<th>Flat</th>
<th>Gouraud</th>
<th>Phong</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Flat Example" /></td>
<td><img src="image2.png" alt="Gouraud Example" /></td>
<td><img src="image3.png" alt="Phong Example" /></td>
</tr>
</tbody>
</table>
Polygon Mesh (no hidden surface removal)

Polygon Mesh (Back-Face Culling)
Polygon Mesh (Z-Buffer hidden Surface removal + Flat Shading)

Polygon Mesh (Z-Buffer and Flat/Gouraud/Phong shading)
Illumination & Reflection in OpenGL

- OpenGL Uses the Phong Illumination/Reflection Model

Final Phong Illumination/Reflection Model Result (Single White Light Source)

- Three color intensity equations:
  \[ I(r,g,b) = \text{Ambient} + \text{Point Diffuse} + \text{Point Specular} \]
  \[ I(r,g,b) = kd(r,g,b) \times I_a \]
  \[ + \ Ip \times kd(r,g,b) \times (N \cdot L) \]
  \[ + \ Ip \times ks \times (R \cdot V)^n \]
- OpenGL generalizes this to include colored light sources
Illumination & Reflection in OpenGL

- Define Light Sources
- Define Material Properties
- Define polygons and their outward-directed normal vectors
- Specify Shading Model
- Enable Depth Testing (Z-Buffer)

Lighting

- OpenGL supports 4 types of light:
  - Ambient
  - Diffuse
  - Specular
  - Emitted
- Can be up to 8 different light sources
Defining a Light Source

- Set up Arrays of lighting values
  - Intensities:
    ```
    GLfloat ambLight0[4] = {0.3f, 0.3f, 0.3f, 1.0f}; // R,G,B,α
    GLfloat diffLight0[4] = {0.5f, 0.5f, 0.5f, 1.0f};
    GLfloat specLight0[4] = {0.0f, 0.0f, 0.0f, 1.0f};
    ```
  - Position:
    ```
    GLfloat posnLight0[4] = {1.0f, 1.0f, 1.0f, 0.0f}; // x,y,z,w
    ```
- Pass Arrays to OpenGL
  ```
  glLightfv(GL_LIGHT0, GL_AMBIENT, ambLight0);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, diffLight0);
  glLightfv(GL_LIGHT0, GL_SPECULAR, specLight0);
  glLightfv(GL_LIGHT0, GL_POSITION, posnLight0);
  ```

Enabling a Light Source

- Turn on Lighting
  ```
  glEnable(GL_LIGHTING);
  ```
- Turn on a Light Source
  ```
  glEnable(GL_LIGHT0);
  ```
Material Reflection Properties

- Ambient
- Diffuse
  - These are usually the same
- Specular

Set up Material Arrays
- ambient/diffuse reflection coefficients
  
  ```c
  GLfloat mat_ambdiff[ ] = {0.0f, 0.7f, 0.0f, 1.0f}; // diff. refl. coeffs.
  // 70% of green light reflected diffusely, no red or blue
  ```
- specular reflection coefficient
  
  ```c
  GLfloat mat_spec[ ] = {1.0f, 1.0f, 1.0f, 1.0f}; // spec. refl. coeffs.
  // bright white light reflected specularly (100% R, G, B)
  ```

Pass Material Arrays to OpenGL

```c
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat_ambdiff);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spec);
glMaterialf(GL_FRONT, GL_SHININESS, 20.0f);
// last parameter: specular exponent (0-128)
```
Defining Normals

- Must compute normals for all polygons
- OpenGL has no function to do that
  - So write your own
    - See notes from last class
- Assume the result is:
  
  \[
  \text{double } n[3];
  \]
- Use this when you define the polygon

  \[
  \text{glBegin(GL\_POLYGON)}
  \]
  \[
  \text{glNormal3f } ( (\text{GLfloat})n[0], (\text{GLfloat})n[1], (\text{GLfloat})n[2] );
  \]
  \[
  // glVertex3f() calls here for polygon vertices
  \]
  \[
  \text{glEnd();}
  \]

Specify a Shading Model and Enable Depth Testing

  \[
  \text{glShadeModel(GL\_FLAT); // use GL\_SMOOTH}
  \]
  \[
  \text{ // for Gouraud shading}
  \]
  \[
  \text{glEnable(GL\_DEPTH\_TEST);} \]
  \[
  \text{glClear (GL\_COLOR\_BUFFER\_BIT |}
  \]
  \[
  \text{GL\_DEPTH\_BUFFER\_BIT);} \]
  \[
  // clear frame buffer and z-buffer
  \]
Some sample code - view class::OnDraw()

```cpp
// Some initializations:
glShadeModel(GL_SMOOTH);
setEnabled(GL_DEPTH_TEST);
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
Clear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
MatrixMode(GL_MODELVIEW);
loadWindow();
Materialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat_ambdiff);
Materialfv(GL_FRONT, GL_SPECULAR, mat_spec); // The lighting,..
Materialf(GL_FRONT, GL_SHININESS, 20.0f); // and material,..
Lightfv(GL_LIGHT0, GL_AMBIENT, ambLight0); // arrays were,..
Lightfv(GL_LIGHT0, GL_DIFFUSE, diffLight0); // set up,..
Lightfv(GL_LIGHT0, GL_SPECULAR, specLight0); // before this,..
Lightfv(GL_LIGHT0, GL_POSITION, posnLight0); // code.
Enable(GL_LIGHTING); glEnable(GL_LIGHT0);
DrawCube(); // Helper function to define cube vertices/polygons/normals
flush();
```

Code from DrawCube() function

```cpp
// Position and rotate the cube:
translate(0.0f, 0.0f, -3.0f); // position cube inside viewing volume
rotate(20.0f, 1.0f, 0.0f, 0.0f); // rotate about x
rotate(20.0f, 0.0f, 1.0f, 0.0f); // rotate about y

// Draw the polygons of the cube, only front face is given here:
double p1[] = {-0.5, 0.5, 0.5}; double p2[] = {-0.5, -0.5, 0.5};
double p3[] = {0.5, -0.5, 0.5}; double n[3];
Calculate(p1, p2, p3, n);
begin(GL_POLYGON); // only 1 face here, other 5 must be defined
normal3f((GLfloat)n[0], (GLfloat)n[1], (GLfloat)n[2]);
vertex3f(-0.5f, 0.5f, 0.5f);
vertex3f(-0.5f, -0.5f, 0.5f);
vertex3f(0.5f, -0.5f, 0.5f);
vertex3f(0.5f, 0.5f, 0.5f);
end();
```
Code from CalcNormal (double *p1, double *p2, double *p3, double *n)

// Form two vectors from the points.
double a[3], b[3];
// Calculate the cross product of the two vectors.
// Normalize the new vector.
double length = sqrt(n[0]*n[0]+n[1]*n[1]+n[2]*n[2]);
n[0] = n[0] / length;

Shadows

- Very important to our perception of depth
- Shadow position/orientation give information as to how objects relate to each other in space
**Sharp Shadows from Point Sources**

- Point source of light
- Object

| BRIGHT | This region receives no light -- completely dark | BRIGHT |

**Soft Shadows from Extended Sources**

- Outside red lines: light received from B
- Outside black lines: light received from A

| Light received from both A and B |
| Light from A, but not B |
| Completely Dark |
| Light from B, but not A |
| Light received from both A and B |

**Umbra:** central area that receives no light (complete shadow)

**Penumbra:** areas in partial shadow (receive light from part of source)
Shadows from Point Sources

- Look at shadows from point sources
- If a point is in shadow, set Phong Ip to 0
  - Source gets no light from point source
  - So no reflection from point source
  - Still must include ambient term
- Lots of algorithms
- One of simplest: Shadow Z-Buffer
Shadow Z-Buffer Algorithm

- A two-stage process

1. Take Light Source as viewpoint & compute depths
   - Store results in shadow Z-buffer $Z'[x'][y']$
   - Each $Z'[x'][y']$ will contain distance of closest surface to light source

2. Normal Z-Buffer rendering
   - But if $(x,y,z)$ is closest to viewer (visible), transform to light space coordinates $(x',y',z')$
   - If $z' > Z'[x'][y']$ point is in shadow
     - Some object is closer to light & will block it
     - So only include ambient term in computation

![Shadow Z-Buffer Diagram]
Set up shadow Z-Buffer, $Z'[x'][y']$, using coordinate system whose origin is at light source
(same code as Z-Buffer, but using different origin)

$Z'[x][y]=\text{infinity for all } x,y$  // regular Z-buffer

for each polygon
  for each pixel $x, y$
    calculate $z$
    if $z < Z[x][y]$
      transform $x, y, z$ to light coord space $x', y', z'$
      if $z' > Z'[x'][y']$
        reduce intensity (include only ambient)
    $Z[x][y]=z; FB[x][y]=\text{intensity}$