Binghamton University

EngiNet™

State University of New York

Thomas J. Watson

School of Engineering
and Applied Science

EngiNet™

WARNING

CS 460/560

Computer Graphics

Professor Richard Eckert

Lecture # 22

April 30, 2001

Lecture 22

- Illumination/Reflection
  - The Phong Model
  - Flat Shading
  - Interpolated Shading
- 3D Graphics using OpenGL
- See CS-460/560 Notes at:

http://www.cs.binghamton.edu/~reckert/460/ill_refl.htm

Illumination, Reflection, Shading

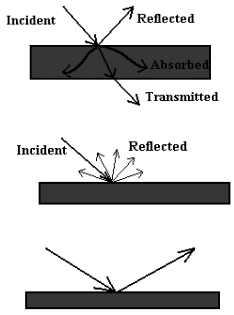
- Need to display surfaces in "natural" colors
  - Colors observed if we really saw the scene
- How do they get those colors?
- Observed Colors Depend on:
  - Light sources in scene
  - Material properties of object surfaces
  - How light interacts with those surfaces
    - Reflection, Transmission,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

Light Sources

- Approximate with two types:
  - 1. Ambient (nondirectional, diffuse, background light)
    - Take as a constant
    - 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 reflected to different degrees
  - Depends on physical (material) properties
  - Gives intrinsic color to materials
  - Approximate by giving 3 diffuse reflection coefficients
    - Fractions of red, green blue reflected
    - kr, kg, kb (0 <=k <=1)

      0 means no reflection in that color band

      1 means 100% reflection in that band

Phong 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 = ambient reflection coefficient

  Actually 3 values of k_d:

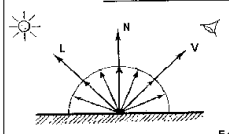
  kr, kg, kb

  (So this is really three equations)

  I_a = Intensity of ambient light in scene

  Ia, kr, kg, kb are adjustable parameters

Diffuse Reflection of a Point Source



L_i = Vector from reflecting point to light source

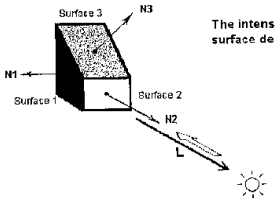
N = Normal vector to surface at reflecting point

I_i = Intrinsic intensity of the point source

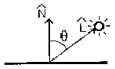
V = Vector from reflecting point to view point

k_d = Diffuse reflection coefficient (0-1)

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



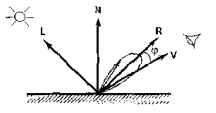
The intensity of light reflected from a diffuse surface depends on the angle between N and L.



$I = k_d I_i \cos(\theta)$

$I = k_d I_i \hat{N} \cdot \hat{L}_i$

Specular Reflection from a Point Source (Phong Model)



L_i = Vector to light source

N_i = Normal vector to reflecting surface

R = Vector in ideal reflection direction

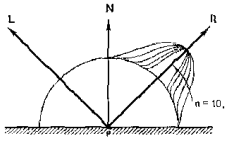
V = Vector to viewpoint

φ = Angle between R and V

k_S = Specular reflection coeffient

n = specular exponent (glossiness)

I_i = Intrinsic intensity of ith point source



$I = k_S I_i \cos^n \varphi$

$I = k_S I_i (R_i \cdot V)^n$

Combining Ambient, Diffuse, and Specular Terms:

$$I(r,g,b) = k_d(r,g,b)\, I_a + \sum_{i \text{ point sources}} I_i [k_d(r,g,b)(\hat{N} \cdot \hat{L}_i) + k_s(\hat{R}_i \cdot \hat{V})^n]$$

Final Phong Model Result (Single Light Source)

- Three color intensity equations:

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

  I(r,g,b) = kd(r,g,b)*Ia

  + Ip*kd(r,g,b)*(N.L)

  + Ip*ks*$(R.V)^n$

It can be shown that R = 2*(N.L)N - L

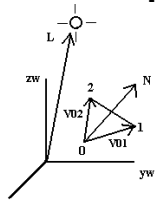
where N and L are unit vectors

Note that specular term has no color dependency

(First approximation)

If viewer moves, specular term must be recomputed

Computing N and L



$$L = \frac{(Lx,Ly,Lz)}{\sqrt{(Lx^2+Ly^2+Lz^2)}}$$

dx1=x1-x0, dy1=y1-y0, dz1=z1-z0

dx2=x2-x0, dy2=y2-y0, dz2=z2-z0

$N = V01 \times V02$

$$N = \frac{(dx1,dy1,dz1) \times (dx2,dy2,dz2)}{\sqrt{((dx1,dy1,dz1)\times(dx2,dy2,dz2)) \bullet ((dx1,dy1,dz1)\times(dx2,dy2,dz2))}}$$

$$\sqrt{\ } = \sqrt{(dy1*dz2-dy2*dz1)^2 + (dx1*dz2-dx2*dz1)^2 + (dx1*dy2-dx2*dy1)^2}$$

Assumes light source is at infinity

If N.L < 0, no light received from point source, so only use ambient light

CalcNormal(double *p1, double *p2, double *p3, double *n)

```
// Form two vectors from the points.
double a[3], b[3];
a[0] = p2[0] - p1[0];   a[1] = p2[1] - p1[1];    a[2] = p2[2] - p1[2];
b[0] = p3[0] - p1[0];   b[1] = p3[1] - p1[1];    b[2] = p3[2] - p1[2];
// Calculate the cross product of the two vectors.
n[0] = a[1] * b[2] - a[2] * b[1];
n[1] = a[2] * b[0] - a[0] * b[2];
n[2] = a[0] * b[1] - a[1] * b[0];
// Normalize the new vector.
double length = sqrt(n[0]*n[0]+n[1]*n[1]+n[2]*n[2]);
n[0] = n[0] / length;
n[1] = n[1] / length;
n[2] = n[2] / length;  }
```

Intensity Computations

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

$$\frac{I(color)}{Imax(color)} = \frac{FB(color)}{FBmax(color)}$$

    - For True color, FBmax(color)=255
    - Imax(color) from formula with all dot products = 1
  - Paint Polygon with resulting FB values
- This is Lambertian Flat Shading

Rendering Process (Flat Shading)

- 1. Set up polygon model data structures
  - Include info needed for subsequent shading
    - Object list, polygon list, vertex list, light sources (L,Ip), 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
- 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].poly[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
```

- 2. Array of polygons (e.g., for polygon j):

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

- 3. Lighting Parameters:

```
Ia                  // Ambient Light Intensity (Ia)
num_lights          // Number of light sources
Lx[k], Ly[k], Lz[k] // World coords of each light source
Ip[k]               // intensity of each light source
```

Scene Description Files

- Viewing parameters (ρ, θ, ϕ, 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 (αz,αy,αz)
  - 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, ρ, θ, ϕ
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 scaling factors
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)

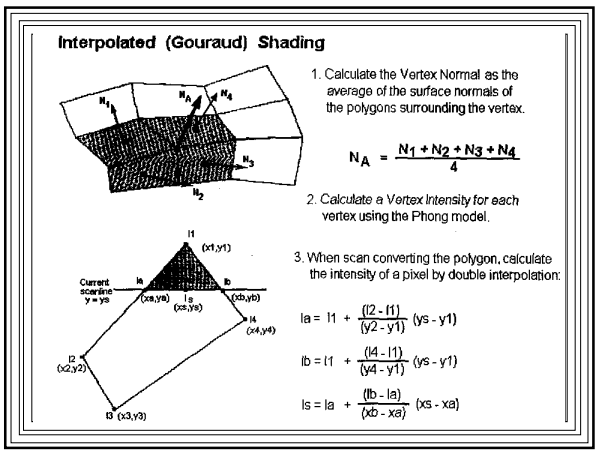
Example Generic Object Description File

// 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
20 // n specular exponent

Interpolated Shading

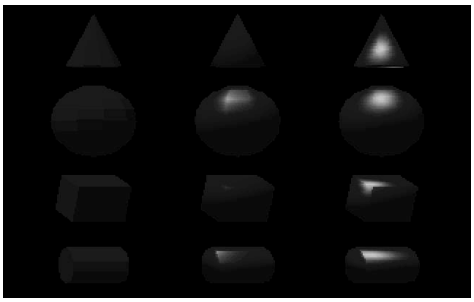
- To "fake" curved surfaces
- Easiest way--Gouraud shading:
  - Compute vertex intensities
  - Double Interpolate values across polygon
    - Should be done at same time as Z-Buffer interpolations
  - Gives a curved appearance to surfaces



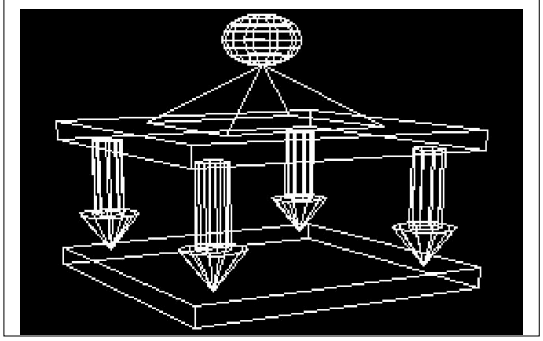
Phong Smooth Shading

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

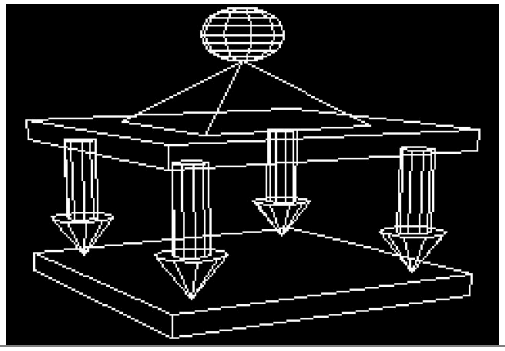
Flat Gouraud Phong



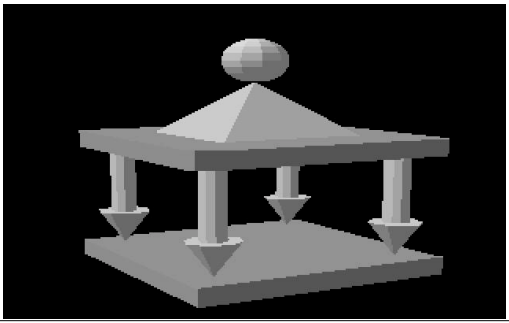
Polygon Mesh (no hidden surface removal)



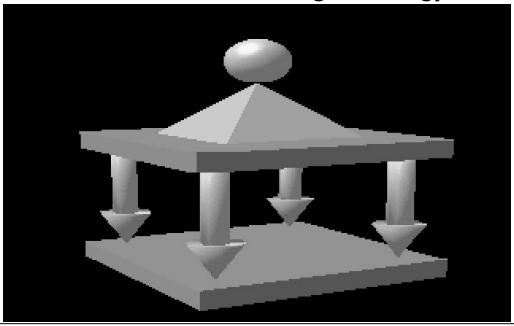
Polygon Mesh (Back-Face Culling)



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



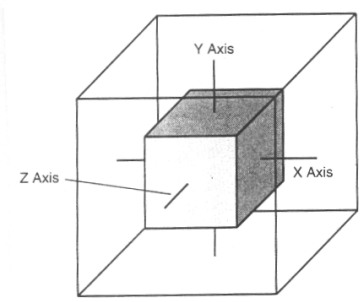
Polygon Mesh (Z-Buffer and Flat/Gouraud/Phong shading)



3D Graphics Using OpenGL

- Building Polygon Models
- Matrix Transformations
- Lighting and Reflection

OpenGL 3D Coordinate System



Defining 3D Polygons in OpenGL

- e.g, front face of a cube

```
glBegin(GL_POLYGON)
    glVertex3f(-0.5f, 0.5f, 0.5f);
    glVertex3f(-0.5f, -0.5f, 0.5f);
    glVertex3f(0.5f, -0.5f, 0.5f);
    glVertex3f(0.5f, 0.5f, 0.5f);
glEnd();
```

Projection Transformation

First tell OpenGL you're using the projection matrix

```
glMatrixMode(GL_PROJECTION);
```

Then Initialize it to the Identity matrix

```
glLoadIdentity();
```

Then define the viewing volume

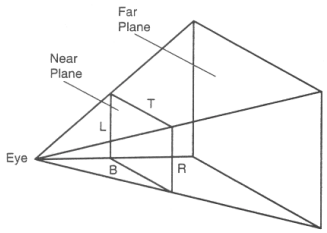
```
glFrustum(-1.0, 1.0, -1.0, 1.0, 2.0, 7.0);
    (left, right, bottom, top, near, far)
```

Eye is at (0,0,0)

The Viewing Volume

Everything outside viewing volume is clipped

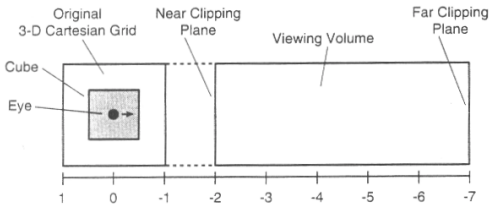
Think of near plane as being window's client area



Modelview Transformation

Our cube is not visible

It lies in front of near clipping plane



Modelview Transformation

To perform geometric translations, rotations, scalings

Need to move our cube into the viewing volume

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslate(0.0f, 0.0f, -3.5f);
Translates cube down z-axis by 3.5 units

OpenGL performs transformations on all vertices

First modelview transformation

Then perspective transformation

Scaling and Rotating a Model

glScalef(2.0f, 2.0f, 2.0f); //twice as big
parameters: sx, sy, sz

glRotatef(30.0f, 0.0f, 0.0f, 1.0f) //30degrees about z-axis
parameters: angle, (x,y,z) coordinates of vector about which to rotate

Typical code for a polygon mesh model

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-1.0, 1.0, -1.0, 1.0, 2.0, 7.0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -3.5f);
glRotatef(30.0f, 0.0f, 0.0f, 1.0f);
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);      // set background color
glClear(GL_COLOR_BUFFER_BIT);  // clear window
glColor3f(0.0f, 0.0f, 0.0f);                // drawing color
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
glBegin(GL_POLYGON); //define polygon vertices here
glEnd();
```

Illumination & Reflection in OpenGL

- Uses the Phong Illumination/Reflection Model

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

- Three color intensity equations:

$I(r,g,b) = \text{Ambient} + \text{Point Diffuse} + \text{Point Specular}$

$I(r,g,b) = kd(r,g,b)*Ia + Ip*kd(r,g,b)*(N.L) + Ip*ks*(R.V)^n$

Illumination & Reflection in OpenGL

- Define Light Sources
- Define Material Properties
- Define Normal Vectors
- Specify Shading Model
- Enable Depth Testing (Z-Buffer)

Defining a Light Source

- Set up Arrays of lighting values

```
glfloat ambLight0[ ] = {0.3f, 0.3f, 0.3f, 1.0f}; // R,G,B,α
glfloat diffLight0[ ] = {0.5f, 0.5f, 0.5f, 1.0f};
glfloat specLight0[ ] = {0.0f, 0.0f, 0.0f, 1.0f};
glfloat posnLight0[ ] = {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);
```

  – (Set Modelview matrix to Identity first)

Enabling a Light Source

- Turning on the light source

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

Material Reflection Properties

- Set up Material Arrays
  - ambient/diffuse reflection coefficients

```
glfloat mat_ambdiff[ ] = {0.0f, 0.7f, 0.0f, 1.0f};
```

  - specular reflection coefficient

```
glfloat mat_spec[ ] = {1.0f, 1.0f, 1.0f, 1.0f};
```

- Pass Material Arrays to OpenGL

```
glMaterialfv(GL_FRONT,
   GL_AMBIENT_AND_DIFFUSE,
   mat_ambdiff);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spec);
glMaterialf(GL_FRONT, GL_SHININESS, 20.0f);
```

Defining Normals

- Must compute normals for all polygons
- OpenGL has no function to do that
  - So write your own
- Assume the result is:

```
double n[3];
```

- Use this when you define the polygon

```
glBegin(GL_POLYGON)
    glNormal3f((Glfloat)n[0], (Glfloat)n[1], (Glfloat)n[2]);
    // glVertex3f() calls here for polygon vertices
glEnd();
```

Specify a Shading Model & Enable Depth Testing

```
glShadeModel(GL_FLAT);  // use GL_SMOOTH
   for Gouraud shading
glEnable(GL_DEPTH_TEST);
glClear (GL_COLOR_BUFFER_BIT |
   GL_DEPTH_BUFFER_BIT);
      // clear frame buffer and z-buffer
```

Some sample code (view class: OnDraw)

```
glShadeModel(GL_SMOOTH);
glEnable(GL_DEPTH_TEST);
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT |GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_MODELVIEW);  glLoadIdentity();
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,mat_ambdiff);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_spec);
glMaterialf(GL_FRONT, GL_SHININESS, 20.0f);
glLightfv(GL_LIGHT0, GL_AMBIENT, ambLight0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffLight0);
glLightfv(GL_LIGHT0, GL_SPECULAR, specLight0);
glLightfv(GL_LIGHT0, GL_POSITION, posnLight0);
glEnable(GL_LIGHTING); glEnable(GL_LIGHT0);
DrawCube();
glFlush();
```

Code from DrawCube() function

```
glTranslatef(0.0f, 0.0f, -3.0f);    // position cube
glRotatef(20.0f, 1.0f, 0.0f, 0.0f);
glRotatef(20.0f, 0.0f, 1.0f, 0.0f);
// Draw the polygons of the cube, one face 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];
CalcNormal(p1, p2, p3, n);
glBegin(GL_POLYGON);
      glNormal3f((GLfloat)n[0], (GLfloat)n[1], (GLfloat)n[2]);
      glVertex3f(-0.5f, 0.5f, 0.5f);
      glVertex3f(-0.5f, -0.5f, 0.5f);
      glVertex3f(0.5f, -0.5f, 0.5f);
      glVertex3f(0.5f, 0.5f, 0.5f);
glEnd();
```

Code from CalcNormal(double *p1, double *p2, double *p3, double *n)

```
// Form two vectors from the points.
double a[3], b[3];
a[0] = p2[0] - p1[0];    a[1] = p2[1] - p1[1];    a[2] = p2[2] - p1[2];
b[0] = p3[0] - p1[0];    b[1] = p3[1] - p1[1];    b[2] = p3[2] - p1[2];
// Calculate the cross product of the two vectors.
n[0] = a[1] * b[2] - a[2] * b[1];
n[1] = a[2] * b[0] - a[0] * b[2];
n[2] = a[0] * b[1] - a[1] * b[0];
// Normalize the new vector.
double length = sqrt(n[0]*n[0]+n[1]*n[1]+n[2]*n[2]);
n[0] = n[0] / length;
n[1] = n[1] / length;
n[2] = n[2] / length;
```