Illumination, Reflection, Shading

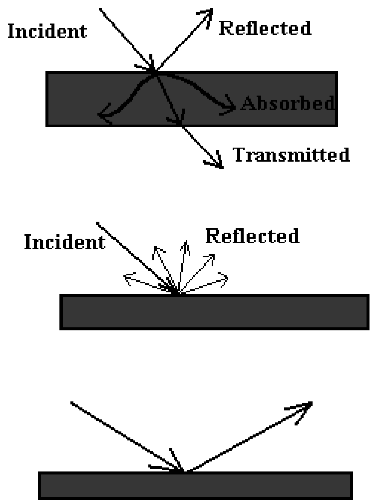
Illumination, Reflection, Shading

- 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
    - kr, kg, kb (0 <= k <= 1)
      - 0 means no reflection in that color band
      - 1 means 100% reflection in that band

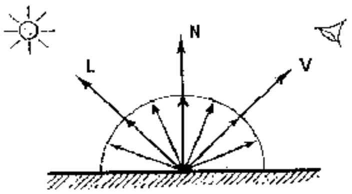
Phong Illumination/Reflection Model

- 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:
    - kr, kg, kb
    - 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
  - – Ia, kr, kg, kb are adjustable parameters

Diffuse Reflection of a Point Source



L_i = Vector from reflecting point to light source

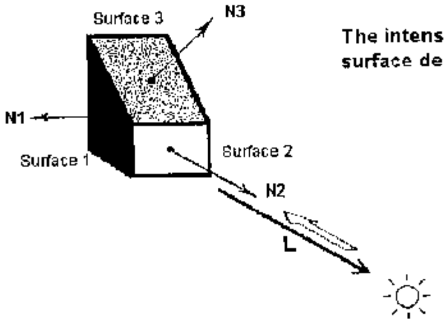
N = Normal vector to surface at reflecting point

I_i = Intrinsic intensity of the point source

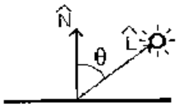
V = Vector from reflecting point to view point

k_d = Diffuse reflection coefficient (0-1)

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



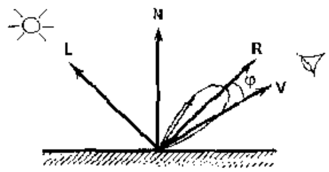
The intensity of light reflected from a diffuse surface depends on the angle between N and L.



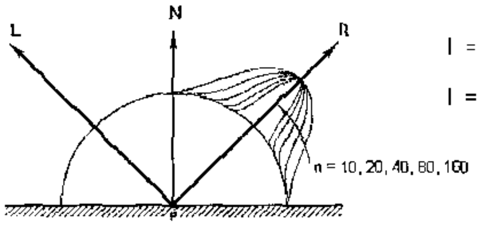
$$I = k_d I_i \cos(\theta)$$

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

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:

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

Final Phong Model Result (Single Light Source)

- Three color intensity equations:

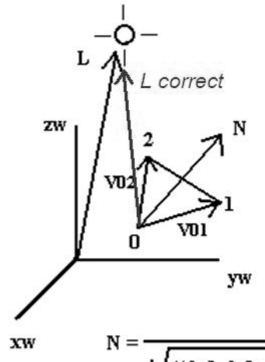
I(r,g,b) = Ambient + Point Diffuse + Point Specular
I(r,g,b) = kd(r,g,b)*Ia (ambient)
+ Ip*kd(r,g,b)*(N · L) (diffuse from point source)
+ Ip*ks*(R · V)^n (specular from pt. Source)

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



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

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

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

$N = V01 \times V02$

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

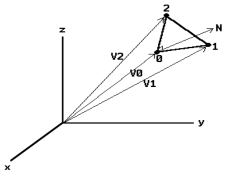
$$\sqrt{\quad} = \sqrt{(dy1*dz2-dy2*dz1)^2 + (dx1*dz2-dx2*dz1)^2 + (dx1*dy2-dx2*dy1)^2}$$

Assumes light source is at infinity

If N·L < 0, no light received, 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
   a[0] = v1[0] - v0[0];   a[1] = v1[1] - v0[1];    a[2] = v1[2] - v0[2];
   b[0] = v2[0] - v0[0];   b[1] = v2[1] - v0[1];    v[2] = v2[2] - v0[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];
   double length = sqrt(n[0]*n[0]+n[1]*n[1]+n[2]*n[2]);
   n[0] = n[0] / length;   // Normalize
   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:

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

    - 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,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
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 (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)

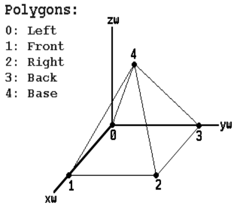
Example Scene Description File

```
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 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

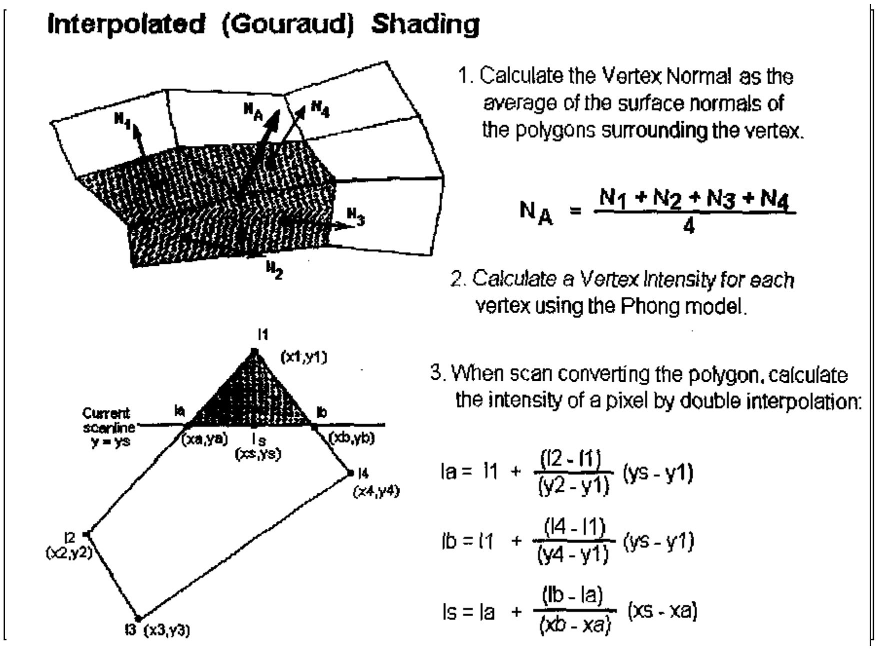


```
// pyramid.des file:


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.

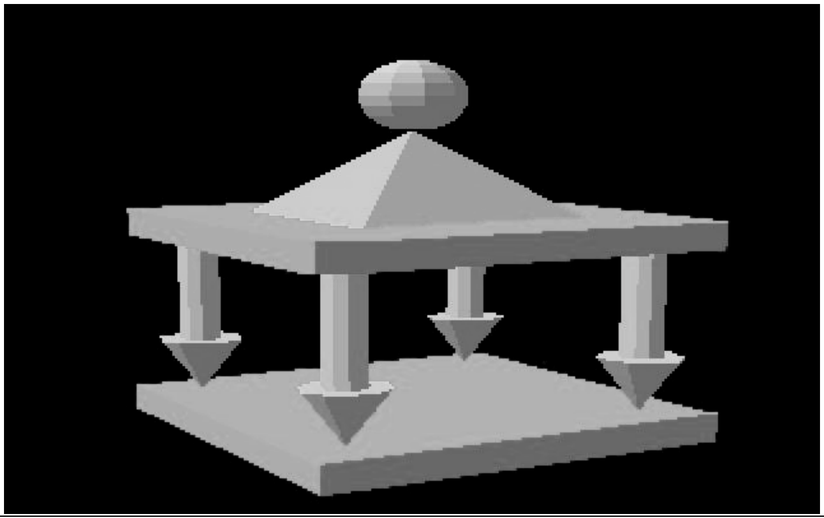
3. When scan converting the polygon, calculate the intensity of a pixel by double interpolation:

$$Ia = I1 + \frac{(I2 - I1)}{(y2 - y1)} (ys - y1)$$

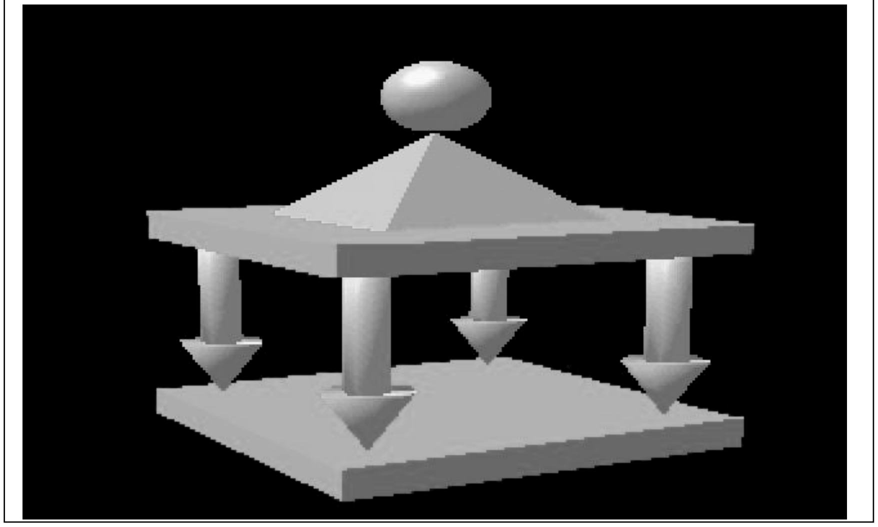
$$Ib = I1 + \frac{(I4 - I1)}{(y4 - y1)} (ys - y1)$$

$$Is = Ia + \frac{(Ib - Ia)}{(xb - xa)} (xs - xa)$$

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



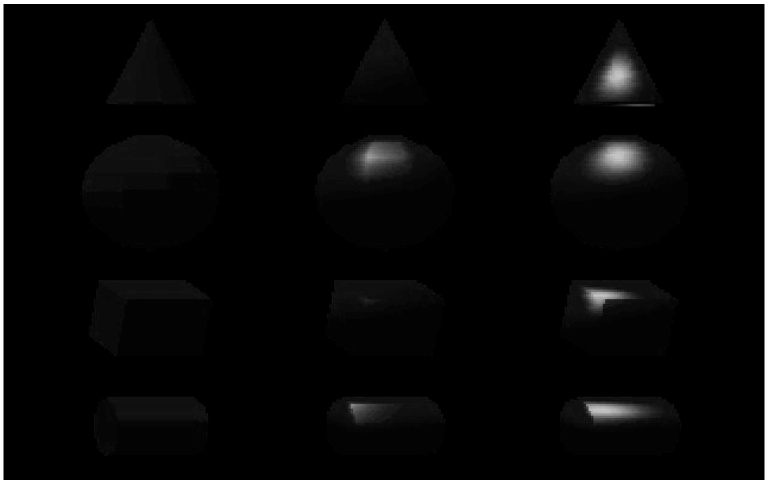
Polygon Mesh (Z-Buffer and Flat/Gouraud/Phong shading)



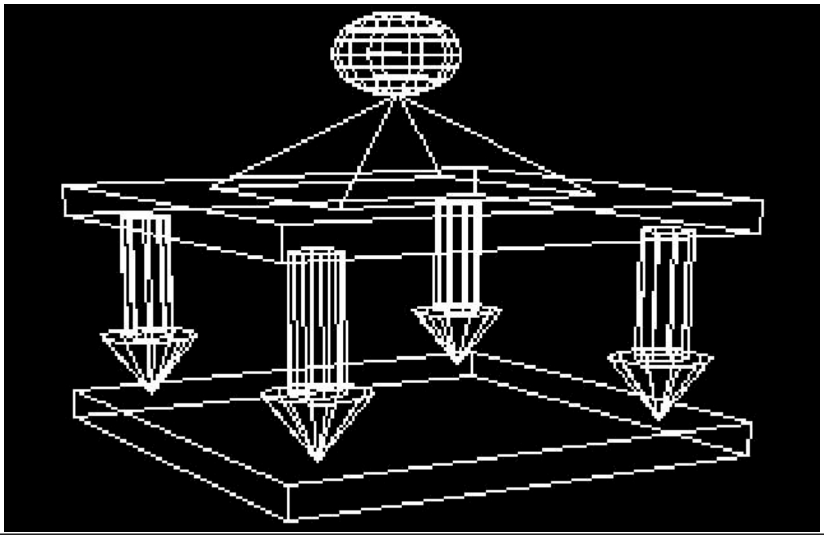
Phong Smooth Shading

- 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

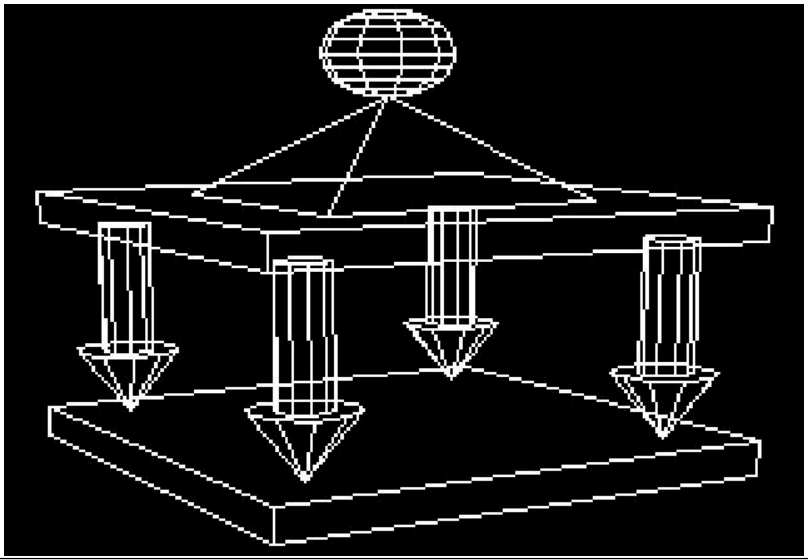
Flat Gouraud Phong



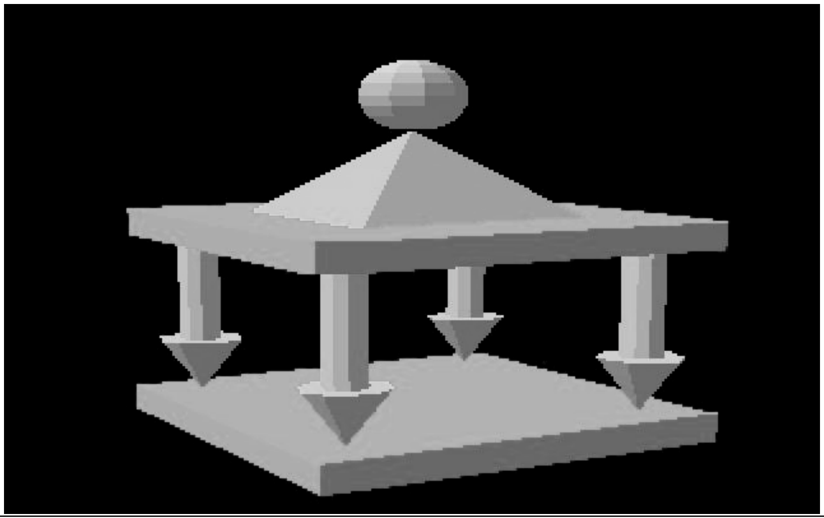
Polygon Mesh (no hidden surface removal)



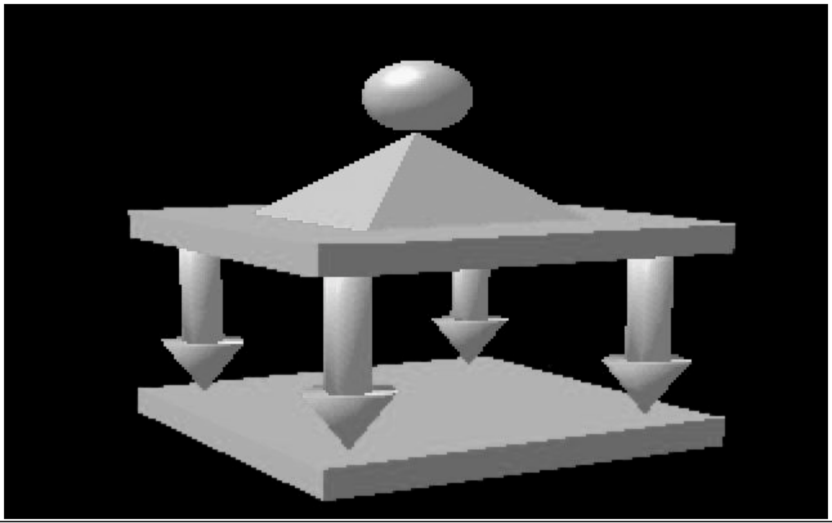
Polygon Mesh (Back-Face Culling)



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



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



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) = Ambient + Point Diffuse + Point Specular

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

$+ Ip*kd(r,g,b)*(N \cdot L)$

$+ Ip*ks*(R \cdot V)^n$

- 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[ ] = {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};
```

  - Position:

```
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);
```

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

Material Reflection Properties

- Set up Material Arrays
  - ambient/diffuse reflection coefficients

```
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

```
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

```
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:

```
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 and 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);   // The lighting..
glMaterialf(GL_FRONT, GL_SHININESS, 20.0f);         // and material..
glLightfv(GL_LIGHT0, GL_AMBIENT, ambLight0);        // arrays were..
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffLight0);        // set up..
glLightfv(GL_LIGHT0, GL_SPECULAR, specLight0);     // before this..
glLightfv(GL_LIGHT0, GL_POSITION, posnLight0);     // code.
glEnable(GL_LIGHTING); glEnable(GL_LIGHT0);
DrawCube();  // Helper function to define cube vertices/polygons/normals
glFlush();
```

Code from DrawCube() function

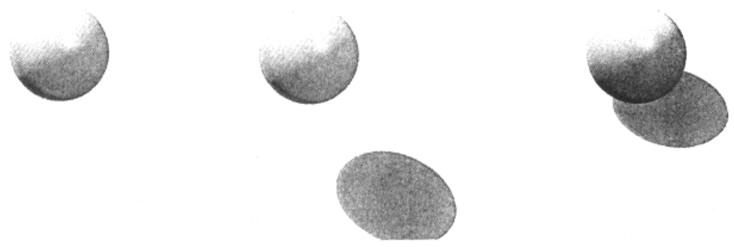
```
glTranslatef(0.0f, 0.0f, -3.0f);    // position cube inside viewing volume
glRotatef(20.0f, 1.0f, 0.0f, 0.0f);  // rotate about x
glRotatef(20.0f, 0.0f, 1.0f, 0.0f);  // rotate about y
// Draw the polygons of the cube, only front face is given here:
double p1[] = {-0.5, 0.5, 0.5}; double p2[] = {-0.5, -0.5, 0.5};
double p3[] = {0.5, -0.5, 0.5}; double n[3];
CalcNormal(p1, p2, p3, n);
glBegin(GL_POLYGON); // only 1 face here, other 5 must be defined
        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)

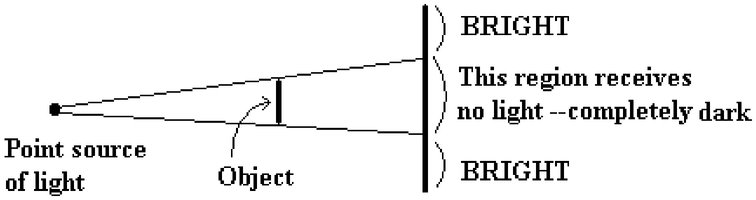
```
// Form two vectors from the points.
double a[3], b[3];
a[0] = p2[0] - p1[0];    a[1] = p2[1] - p1[1];    a[2] = p2[2] - p1[2];
b[0] = p3[0] - p1[0];    b[1] = p3[1] - p1[1];    b[2] = p3[2] - p1[2];
// Calculate the cross product of the two vectors.
n[0] = a[1] * b[2] - a[2] * b[1];
n[1] = a[2] * 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;
```

Shadows

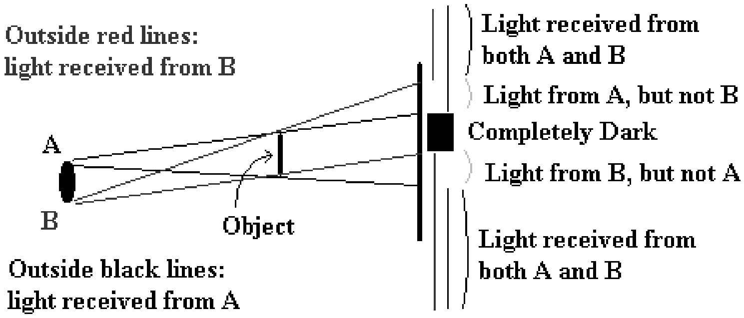
- Very important to our perception of depth
- Shadow position/orientation give information as to how objects relate to each other in space



Sharp Shadows from Point Sources

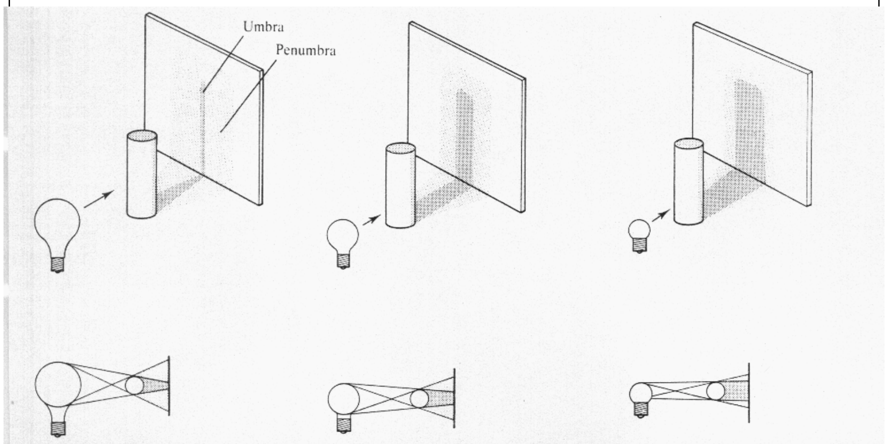


Soft Shadows from Extended Sources



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

Penumbra: areas in partial shadow (receive light from part of source)



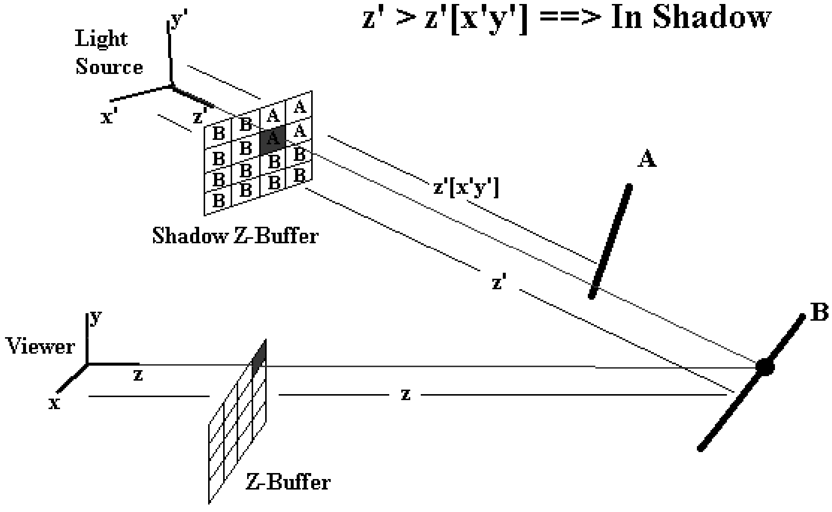
Shadows from Point Sources

- 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



```
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-buf[x][y]=infinity for all x,y     // regular Z-buffer
for each polygon
    for each pixel x,y
        calculate z
        if z < Z-buf[x][y]
            transform x,y,z to light coord space x’,y’,z’
            if z’ > Z’[x’][y’]
                reduce intensity (include only ambient)
            Z-buf[x][y]=z; FB[x][y]=intensity
```