Hochiminh city University of Technology
Faculty of Computer Science and Engineering

COMPUTER GRAPHICS

CHAPTER 8:

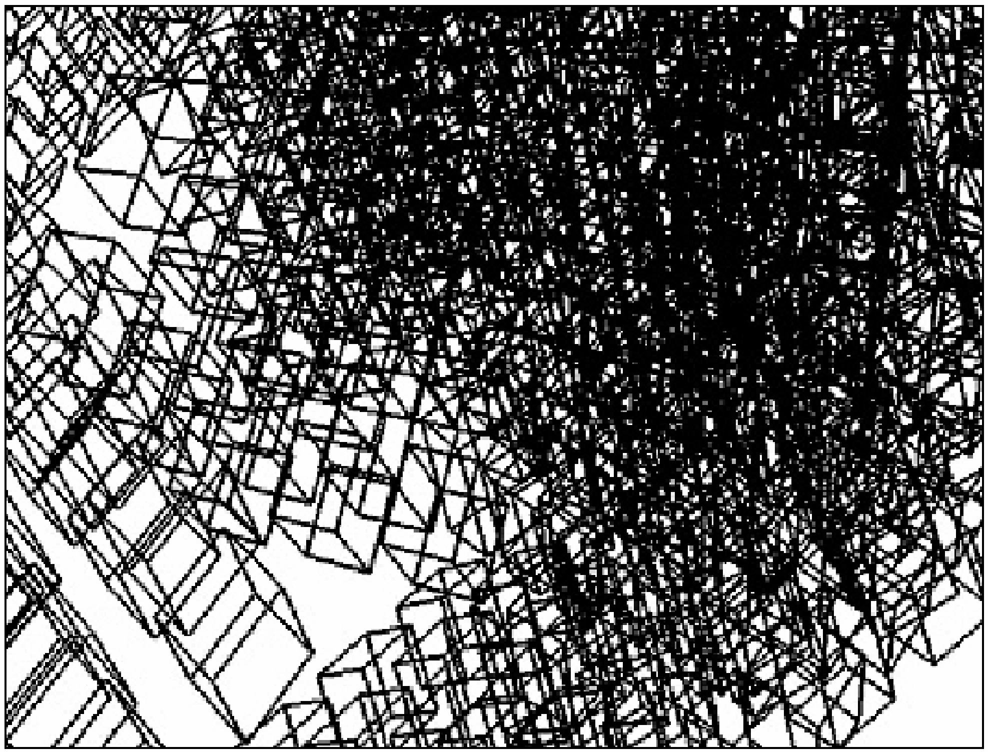
Lighting and Shading

OUTLINE

- Introduction
- Shading model
- Flat shading & smooth shading
- Using Light Sources in OpenGL
- Working with material in OpenGL
- Computation of Vectors

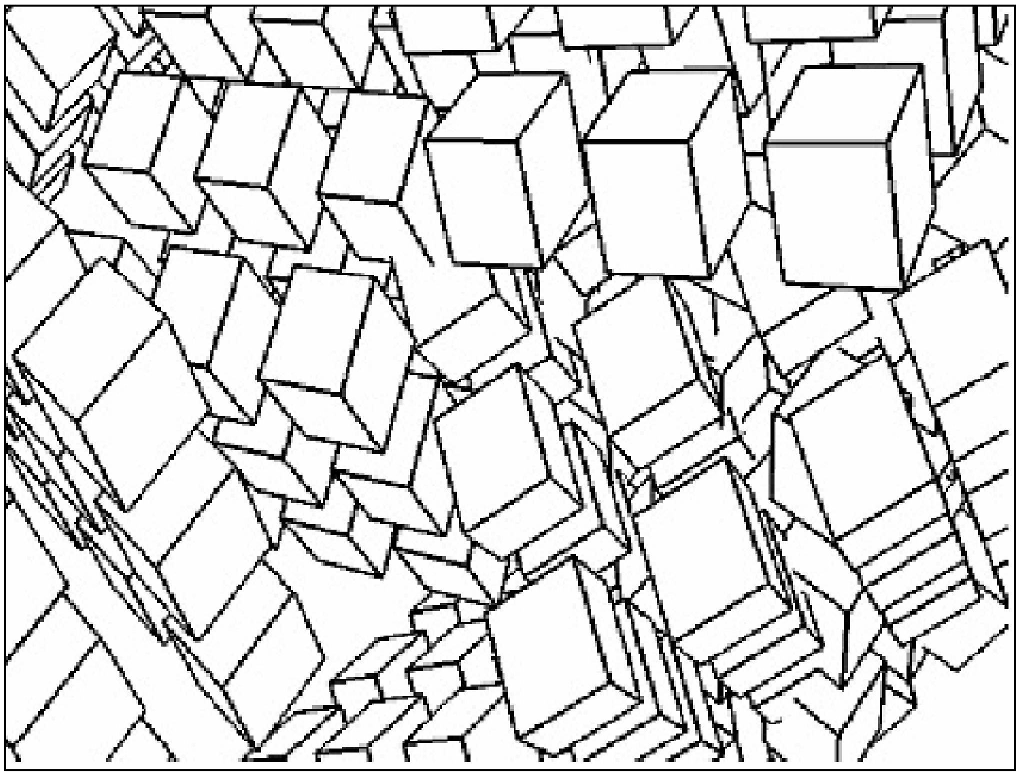
Introduction

- Wireframe
  - Simple, only edges of each object are drawn
  - Can see through object
  - It can be difficult to see what's what



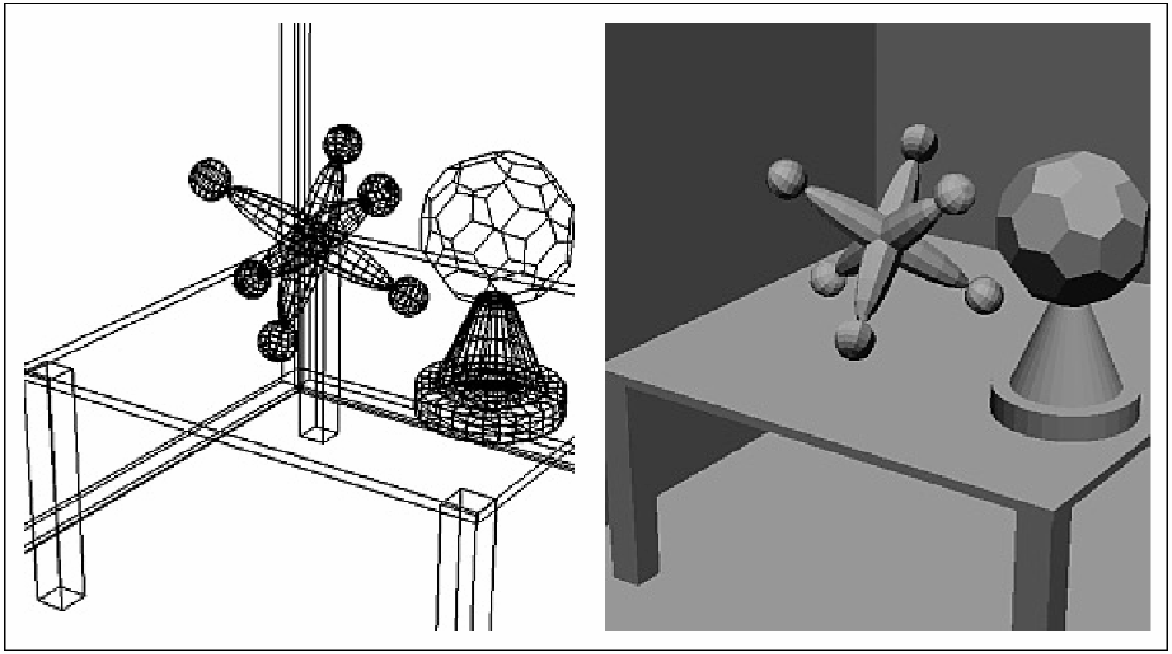
Introduction

- Line Drawing: wire-frame with hidden surface removal
  - Only edges are drawn
  - The objects now look solid, and it is easy to tell where one stops and the next begins



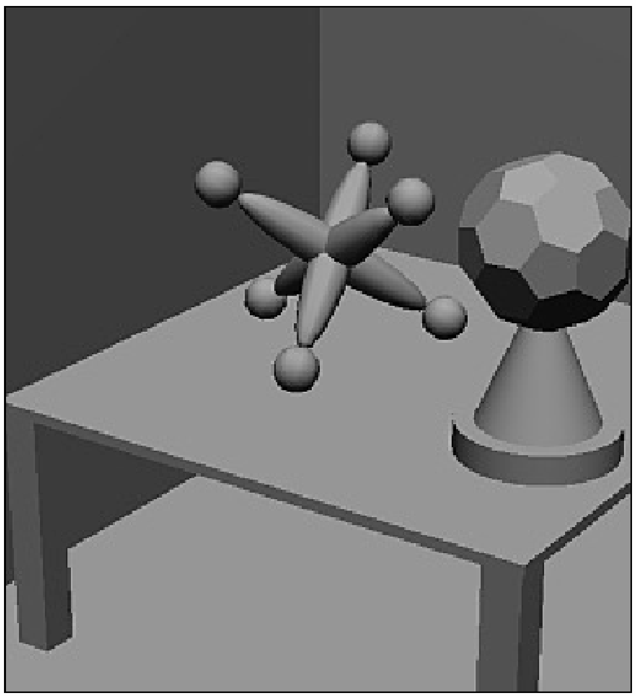
Introduction

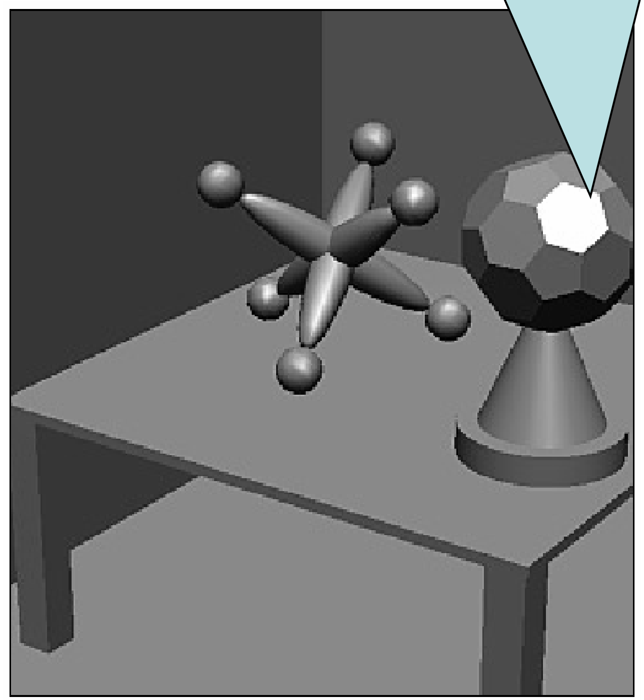
- Flat shading:
  - A calculation of how much light is scattered from each face is computed at a single point.
  - All points in a face are rendered with the same gray level
  - Can see the Boundary between polygons



Introduction

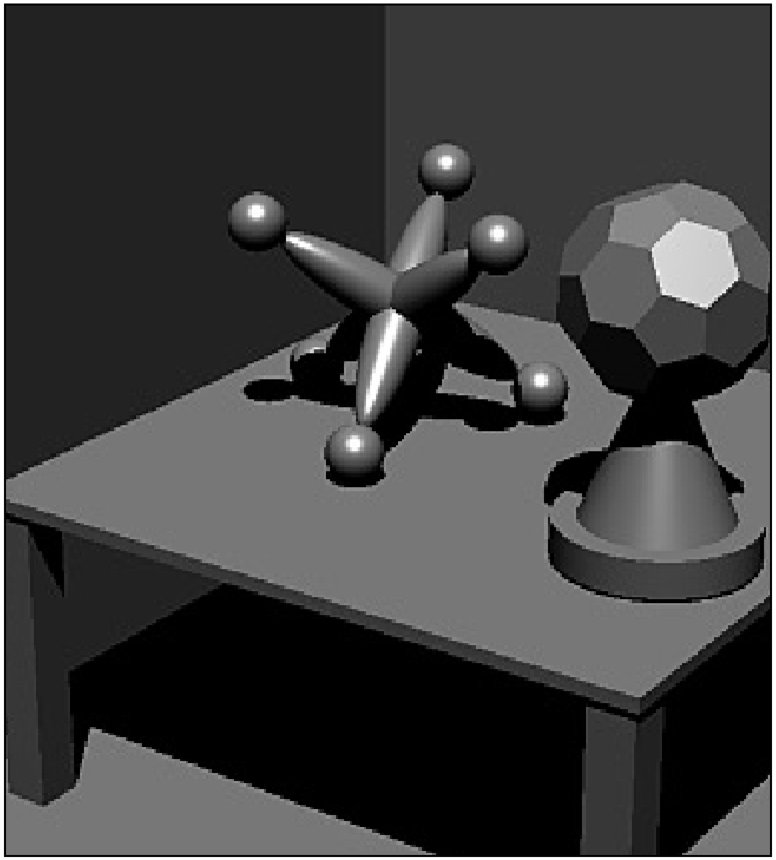
- Smooth shading (Gouraud shading):
  - Different points of a face are drawn with different gray levels found through an interpolation scheme
  - The edges of polygons disappear

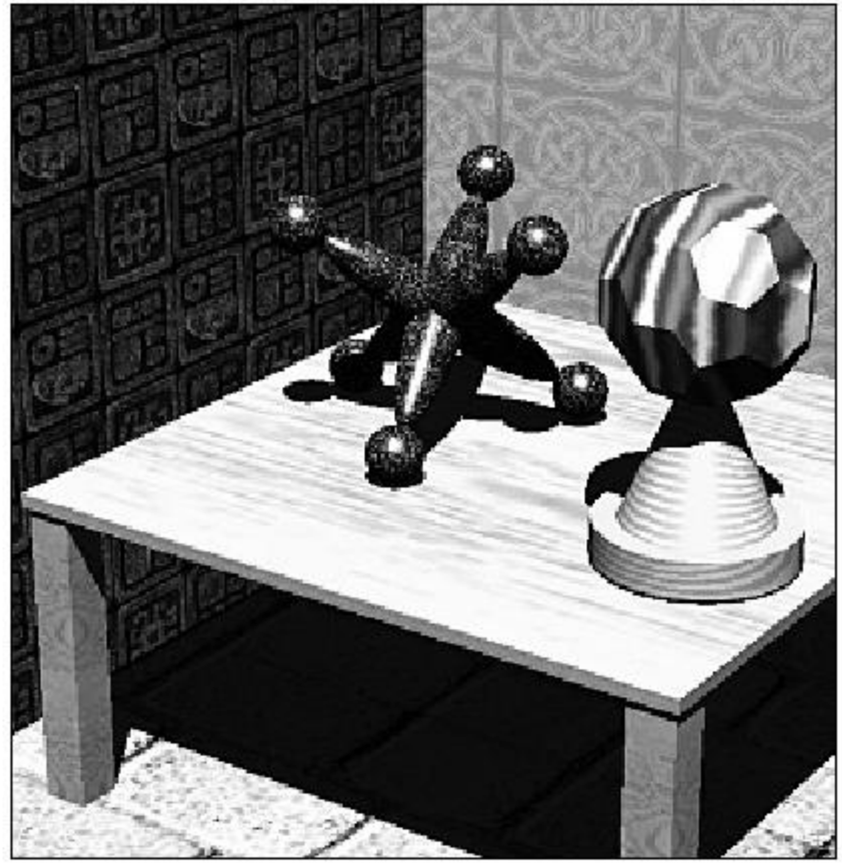




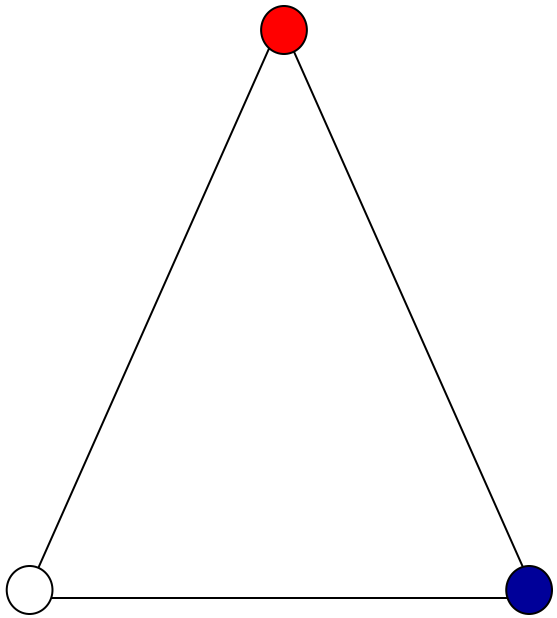
Introduction

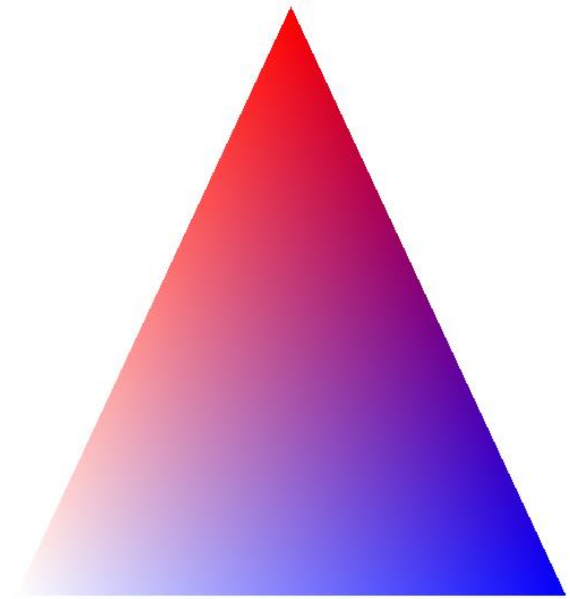
- Adding texture, shadow





Introduction





Before Projection

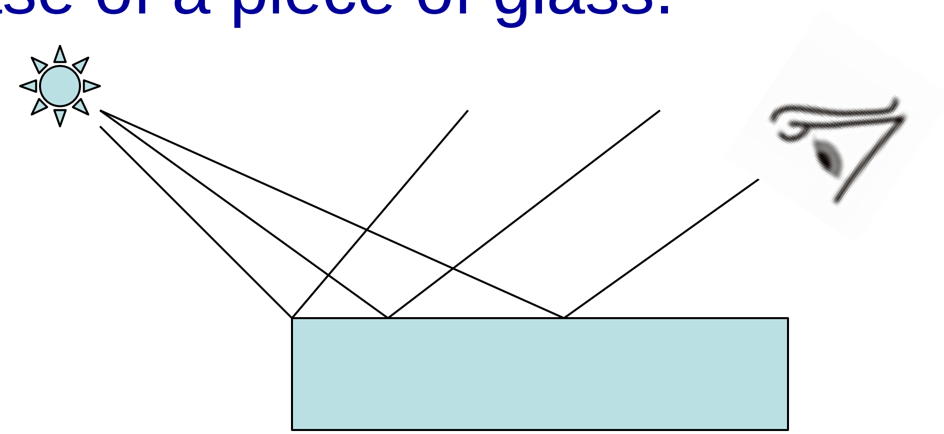
Physical Model

After Projection

Mathematical Model

Shading model

- Light sources "shine" on the various surfaces of the objects, and the incident light interacts with the surface in three different ways:
  - Some is absorbed by the surface and is convert to heat
  - Some is reflected from the surface
  - Some is transmitted into the interior of the objects, as in the case of a piece of glass.



Shading model

- Two types of reflection of incident light:
  - Diffuse: re-radiated uniformly in all directions. Interacts strongly with the surface, so it color is usually affected by the nature of material.
  - Specular: highly directional, incident light doesnot penetrate the object, but instead is reflected directly from its outer surface. The reflected light has the same color as incident light.

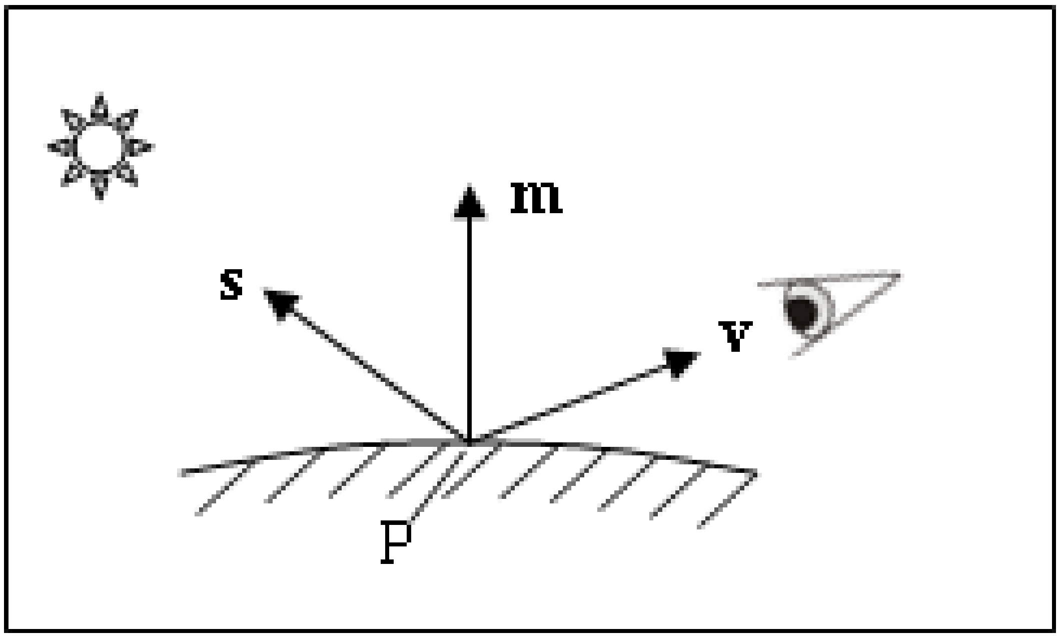
Shading model

- Why does the image of a real teapot look like

- Light-material interactions cause each point to have a different color or shade
- To calculate color of the object, need to consider:
  - **Light sources**
  - **Material properties**
  - **Location of viewer**
  - **Surface orientation**

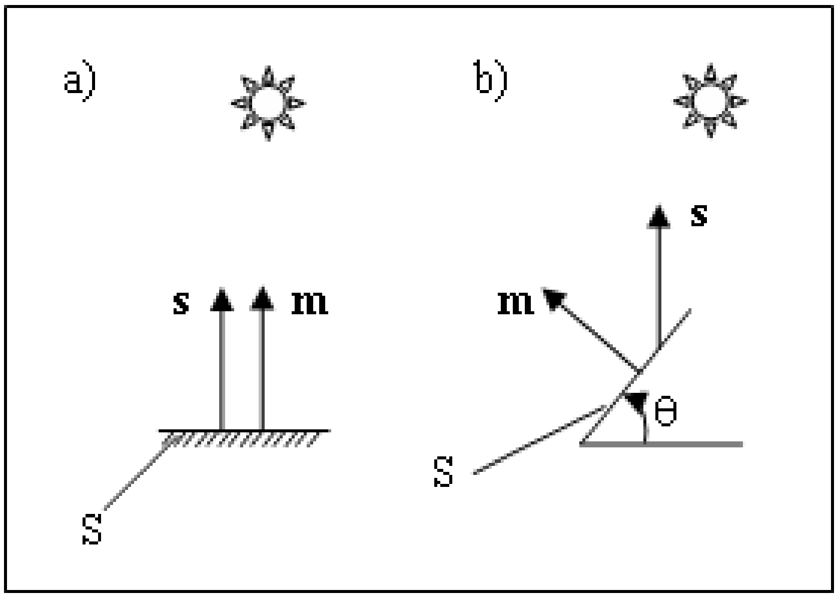
Shading model

- Geometric ingredients for finding reflected light
  - The normal vector m to the surface at P
  - The vector v from P to the viewer's eye
  - The vector s from P to the light source



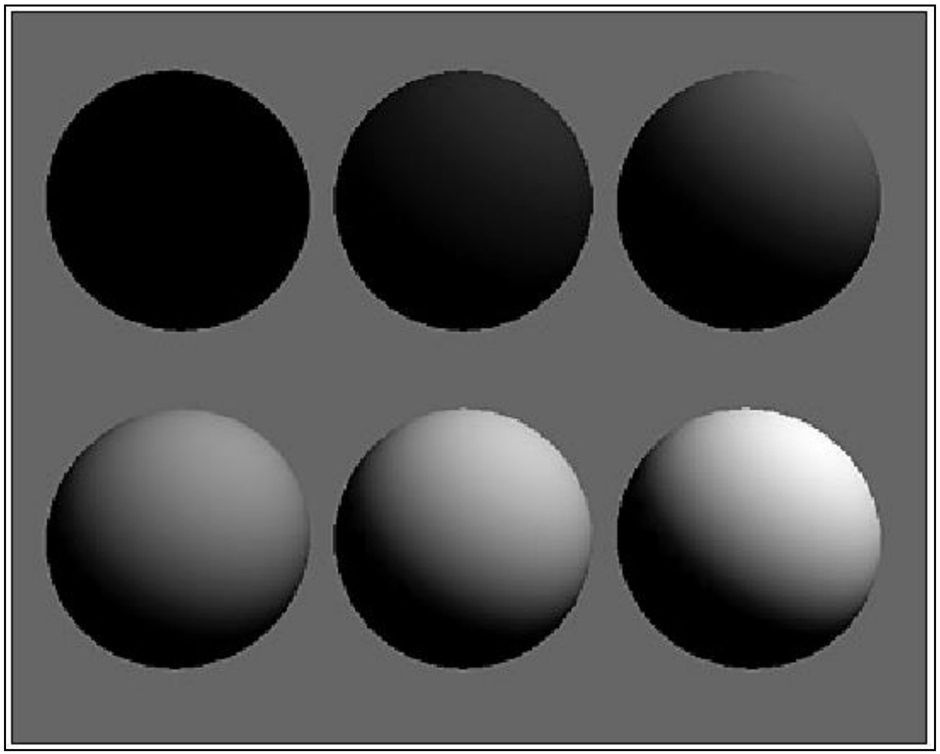
Shading model

- Computing the Diffuse Component
  - Diffuse component with an intensity denoted by I_d
  - Scattering uniform in all directions → depend only on m, s
  - Lambert's law: $$I_d = I_s \rho_d \frac{s \bullet m}{|s||m|} \qquad I_d = I_s \rho_d \max\left(\frac{s \bullet m}{|s||m|}, 0\right)$$
  - I_s: intensity of light source, ρ_d : diffuse reflection coefficient



Shading model

- Computing the Diffuse Component
  - diffuse reflection coefficient : 0, 0.2, 0.4, 0.6, 0.8, 1.0

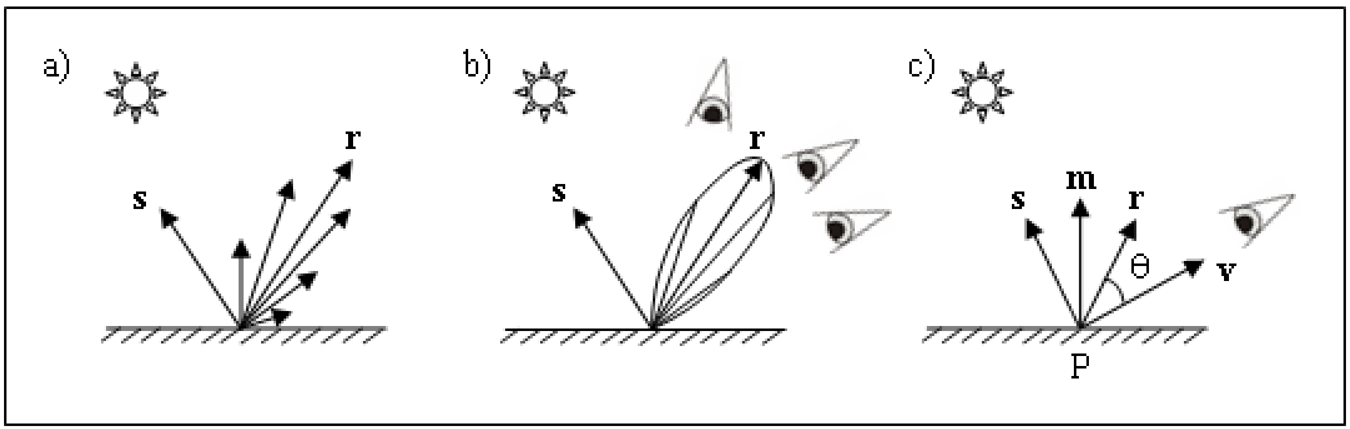


Shading model

- Specular Reflection
  - Specular reflection causes highlights, which can add significantly to the realism of a picture when objects are shiny
  - The amount of light reflected is greated in the direction r.

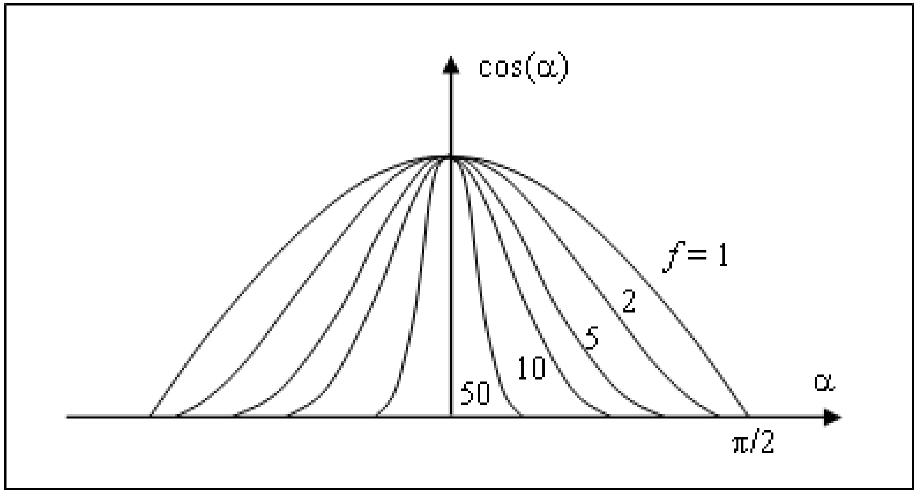
$$r = -s + 2\frac{s \bullet m}{|m|^2}m \qquad I_{sp} = I_s \rho_s \left(\frac{r}{|r|} \bullet \frac{v}{|v|} \right)^f$$

f : [1, 200]



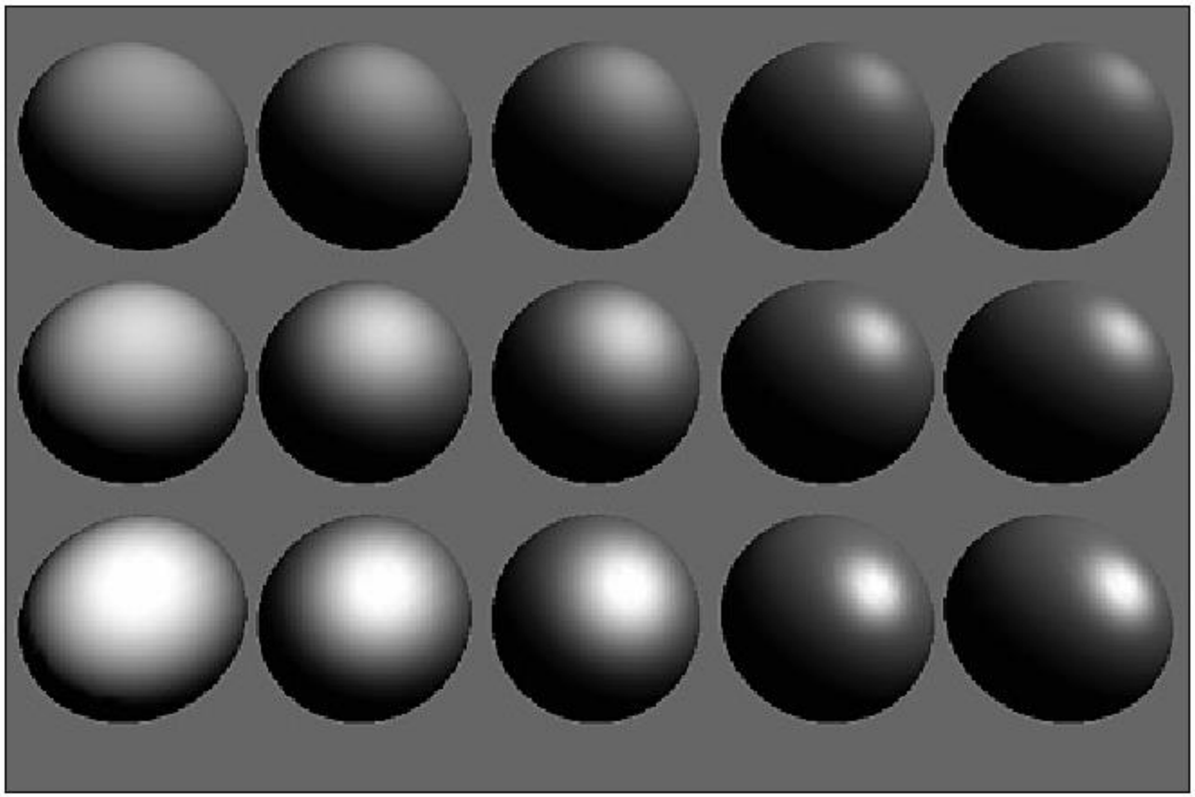
Shading model

- Specular Reflection
  - As f increase, the reflection becomes more mirror like and is more highly concentrated along the direction r.



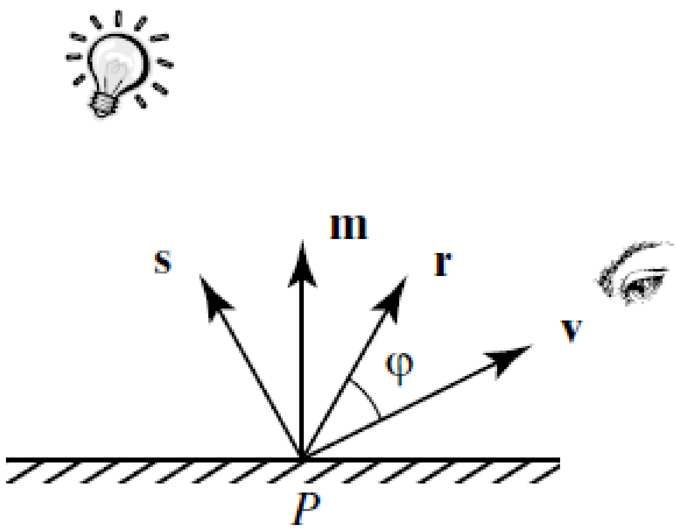
Shading model

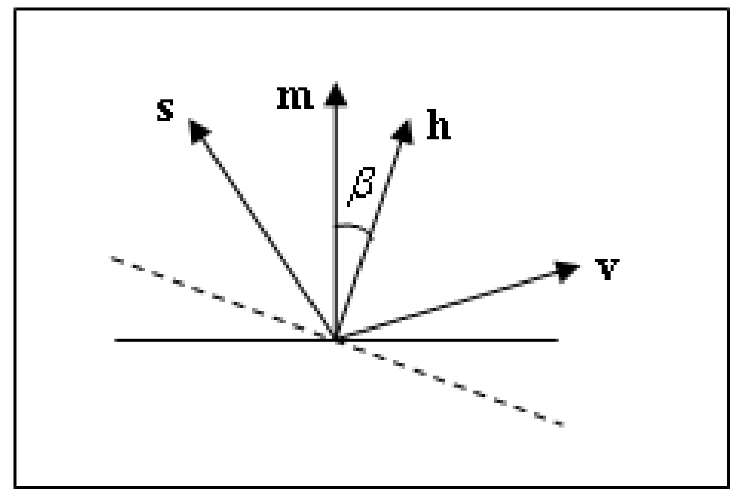
- Specular Reflection
  - ρ_s from top to bottom: 0.25, 0.5, 0.75. *f* from left to right: 3, 6, 9, 25, 200



Shading model

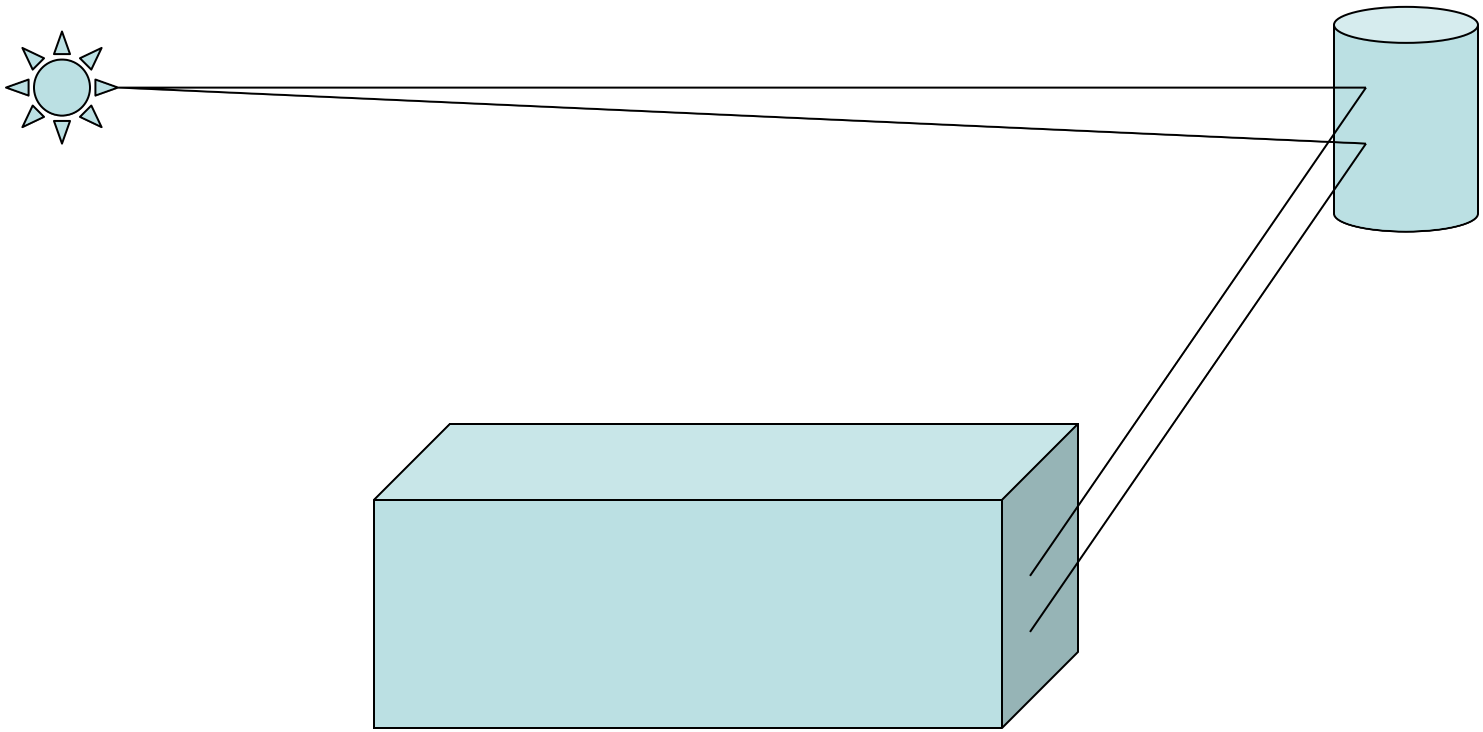
- Specular Reflection
  - Reduce computation time, use halfway vector h = s + v.
  - Use β as the approximation of angle between r and v





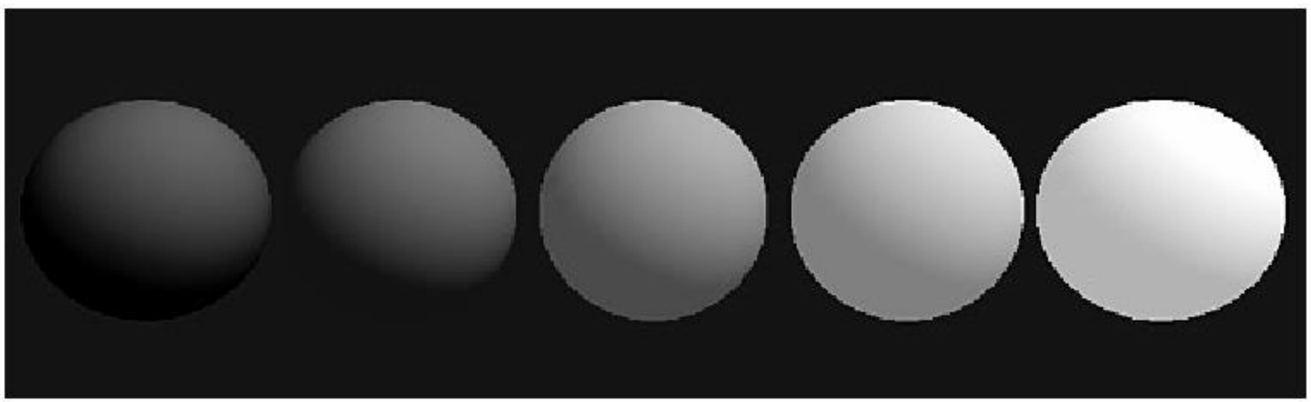
Shading model

- Ambient Light



Shading model

- Ambient Light
  - To overcome the problem of totally dark shadows, we imagine that a uniform “background glow” called ambient light exist in the environment
  - Not situated at any particular place, and spreads in all direction uniformly
  - The source is assigned an intensity I_a. ρ_a: ambient reflection coefficient.



Shading Model

0.9, 0.1, 0.4

0.3, 0.2, 0.5

1.0, 0.3, 0.9

Shading model

- Combining Light Contribution

$$I = I_a \rho_a + I_d \rho_d \times lambert + I_{sp} \rho_s \times phong^f$$

$$lambert = \max(0, \frac{s \cdot m}{|s||m|}) \quad and \quad phong = \max(0, \frac{h \cdot m}{|h||m|})$$

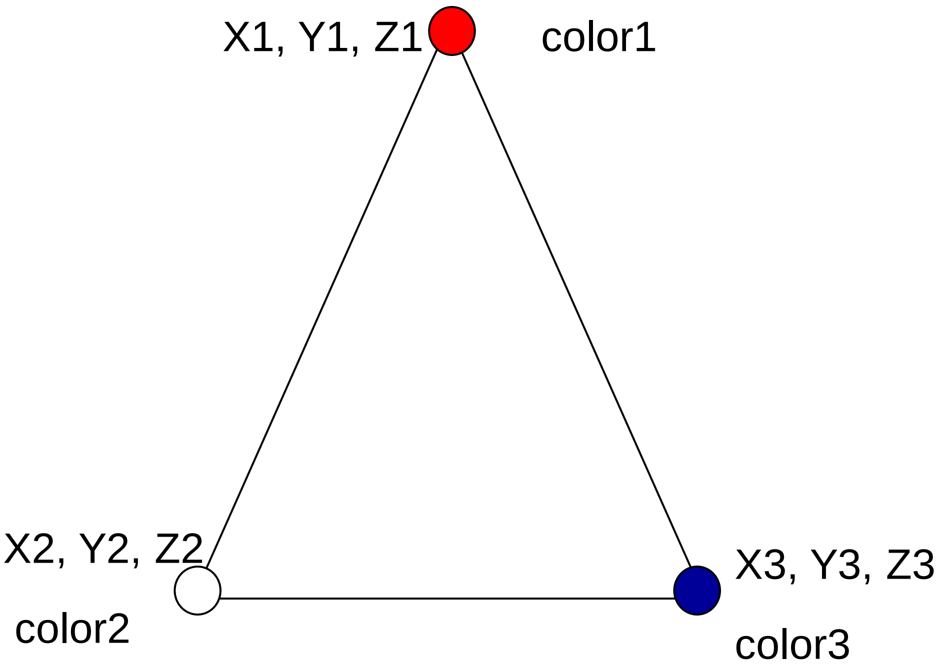
- Adding color

$$I_r = I_{ar} \rho_{ar} + I_{dr} \rho_{dr} \times lambert + I_{spr} \rho_{sr} \times phong^f$$

$$I_g = I_{ag} \rho_{ag} + I_{dg} \rho_{dg} \times lambert + I_{spg} \rho_{sg} \times phong^f$$

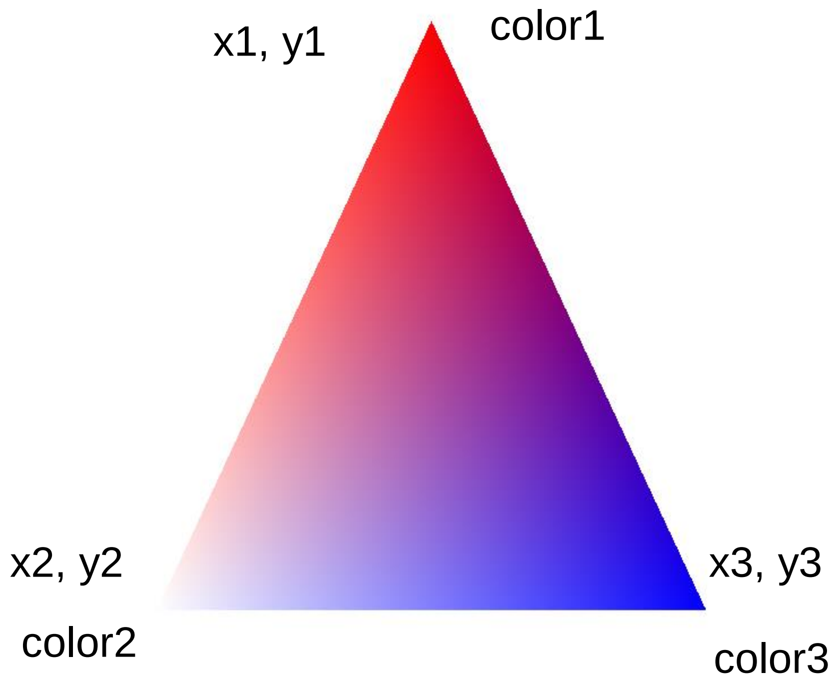
$$I_b = I_{ab} \rho_{ab} + I_{db} \rho_{db} \times lambert + I_{spb} \rho_{sb} \times phong^f$$

Introduction



Before Projection

Physical Model



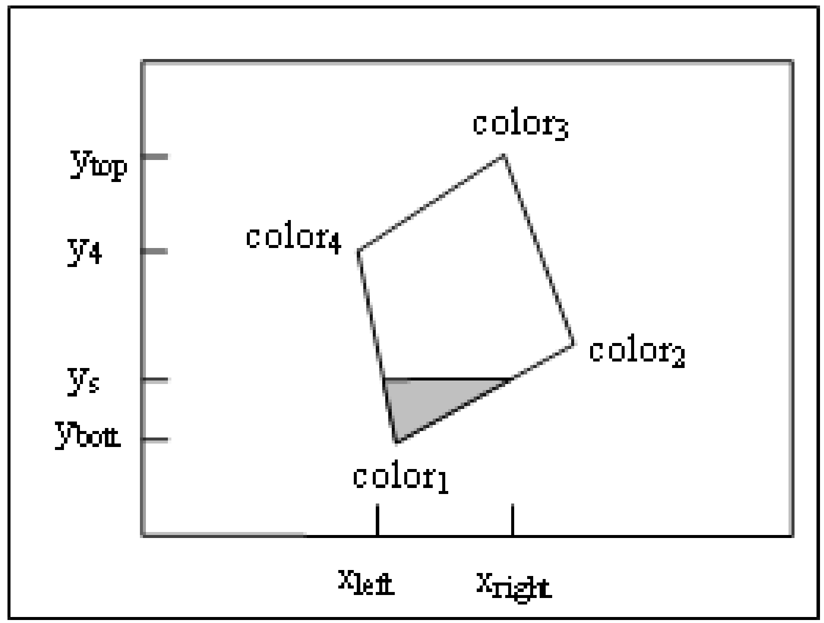
After Projection

Mathematical Model

Flat shading & smooth shading

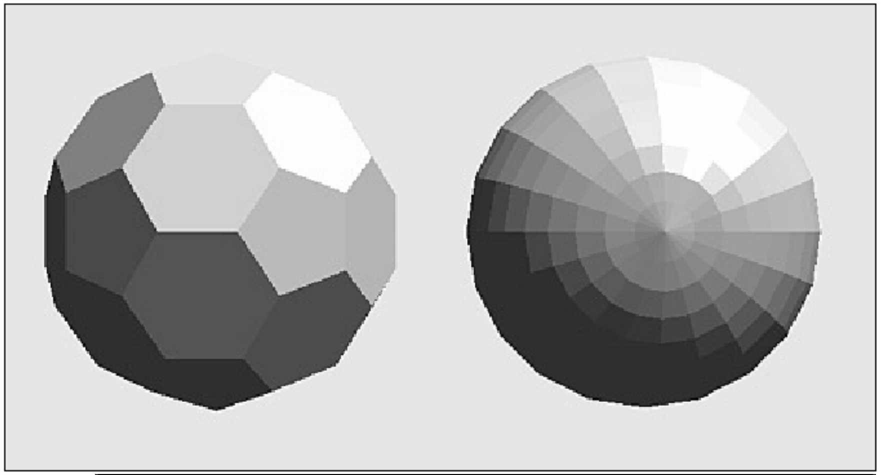
- Painting a face
  - The pixels in a polygon are visited in a regular order, usually scan line by scan line from bottom to top, and across each scan line from left to right
  - Convex polygons can be made highly efficient, since, at each scan line, there is a single unbroken "run" of pixels.

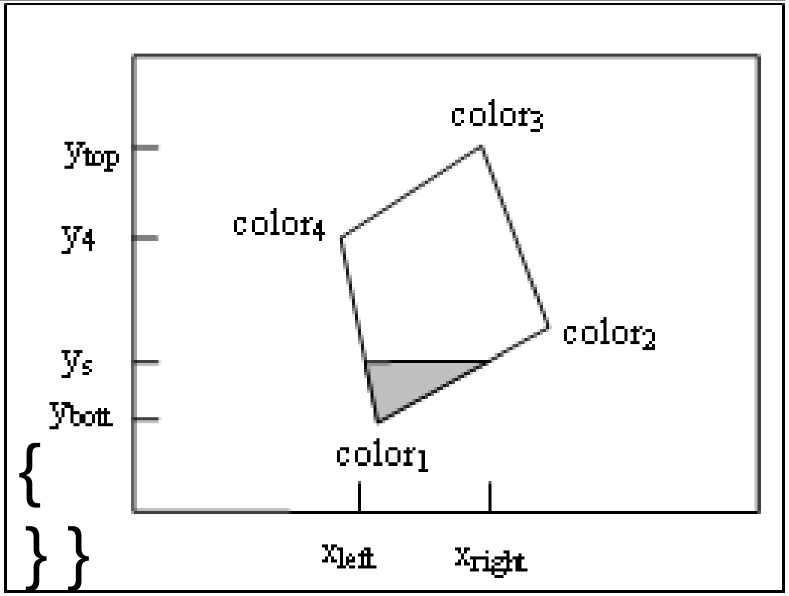
for (int y = ybott ; y <= ytop ; y++) {
    find xleft and xright;
    for (int x = xleft; x <= xright; x++) {
        find the color c for this pixel;
        put c into the pixel at (x, y); } }



Flat shading & smooth shading

- Flat shading
  - Face is flat, light source are quite distant → diffuse light component varies little over different points
  - glShadeModel(GL_FLAT);





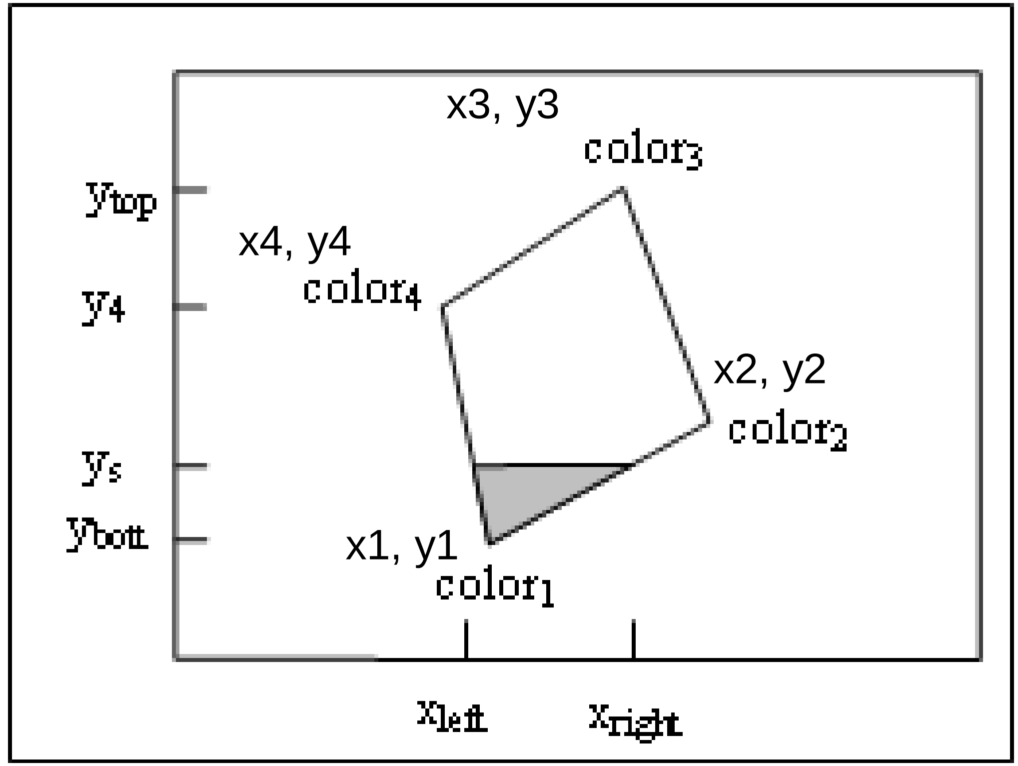

```
for (int y = ybott ; y <= ytop ; y++) {
        find xleft and xright;
        find the color c for this scan line;
        for (int x = xleft;  x <= xright; x++) {
                put c into the pixel at (x, y); } }
```

Flat shading & smooth shading

- Flat Shading

(xleft – x4)/(xleft – x1) =
(cl – color4)/(cl-color1)

(xright – x2)/(xright – x1)
= (cr – color2)/(cr-color1)



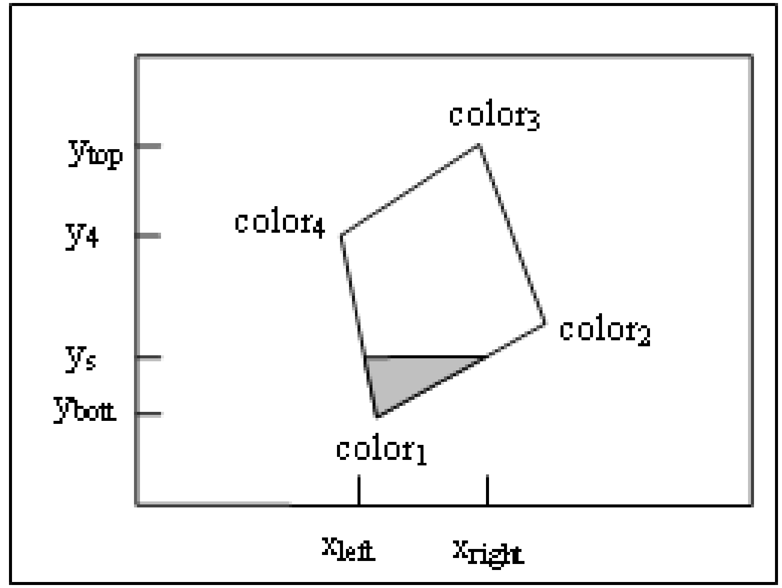
Flat shading & smooth shading

- Smooth shading – Gouraud shading

$color_{\text{left}}$ = lerp($color_1$, $color_4$, f), $f = \dfrac{y_s - y_{bott}}{y_4 - y_{bott}}$

$$c(x) = lerp\left(color_{left}, color_{right}, \frac{x - x_{left}}{x_{right} - x_{left}} \right)$$

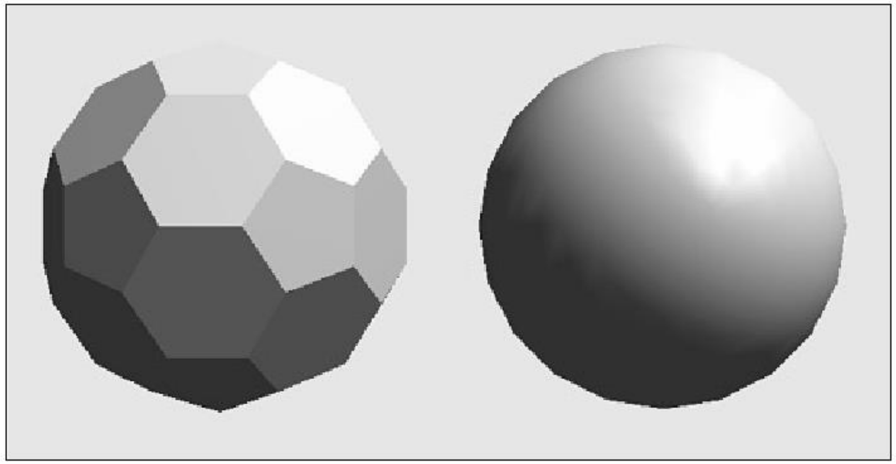
$$c(x+1) = c(x) + \frac{color_{right} - color_{left}}{x_{right} - x_{left}}$$

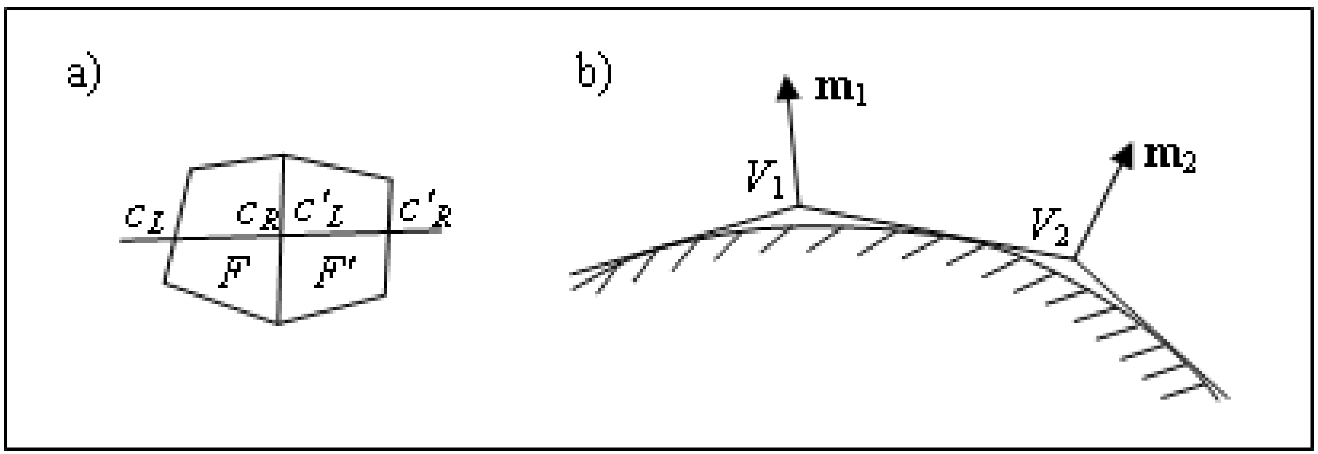


for (int y = ybott ; y <= ytop ; y++){
find xleft and xright;
find colorleft and colorright;
colorinc = (colorright – colorleft)/ (xright – xleft)
for (int x = xleft, c = colorleft; x <= xright; x++, c += colorinc) {
put c into the pixel at (x, y);}}

Flat shading & smooth shading

- Smooth shading – Gouraud shading
  - glShadeModel(GL_SMOOTH);

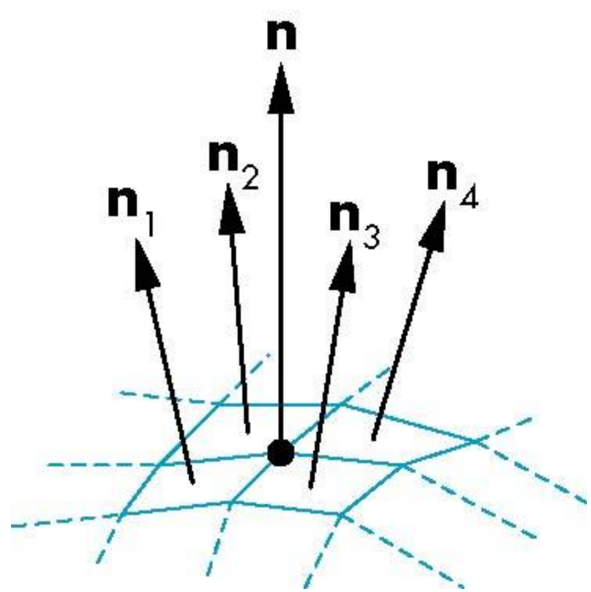




Flat shading & smooth shading

- Smooth shading – Gouraud shading
  - For polygonal models, Gouraud proposed we use the average of the normals around a mesh vertex

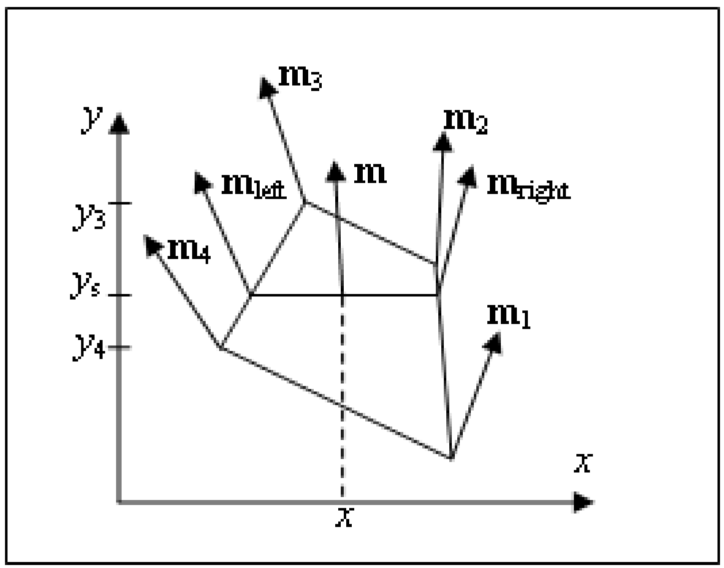
$$\mathbf{n} = (\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4)/\ |\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4|$$



Flat shading & smooth shading

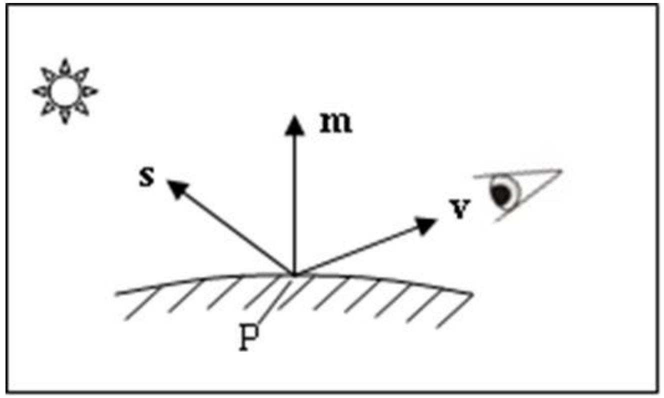
- Smooth shading – Phong shading
  - Slow calculation, better realism
  - OpenGL doesn't support

$$m_{left} = lerp\left(m_4, m_3, \frac{y_s - y_4}{y_3 - y_4} \right)$$



Steps in OpenGL shading

- -Enable shading and select model
- -Specify normal
- -Specify lights
- -Specify material properties

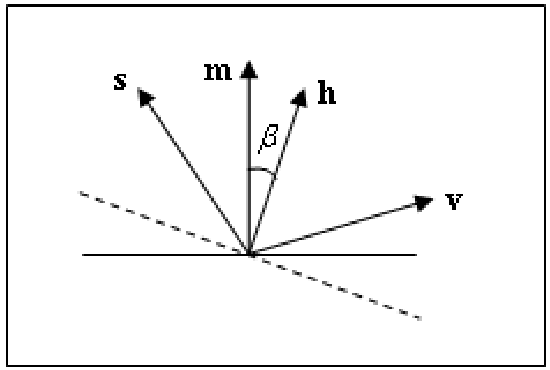


$$I_r = I_{ar}\rho_{ar} + I_{dr}\rho_{dr} \times lambert + I_{spr}\rho_{sr} \times phong^f$$

$$I_g = I_{ag}\rho_{ag} + I_{dg}\rho_{dg} \times lambert + I_{spg}\rho_{sg} \times phong^f$$

$$I_b = I_{ab}\rho_{ab} + I_{db}\rho_{db} \times lambert + I_{spb}\rho_{sb} \times phong^f$$

$$lambert = \max(0, \frac{s \cdot m}{|s||m|}) \quad and \quad phong = \max(0, \frac{h \cdot m}{|h||m|})$$



Enabling Shading

- Shading calculations are enabled by
  - glEnable(GL_LIGHTING)
  - **Once lighting is enabled, glColor() ignored**
- Must enable each light source individually
  - glEnable(GL_LIGHTi) **i=0,1…..**

Specify normals

- In OpenGL the normal vector is part of the state
- Set by **glNormal*()**
  - glNormal3f(x, y, z);
  - glNormal3fv(p);
- Usually we want to set the normal to have unit length so cosine calculations are correct
  - glEnable(GL_NORMALIZE) **allows for autonormalization at a performance penalty**

Using Light Sources in OpenGL

- Creating a light Source

  - Position

  GLfloat myLightPosition[] = {3.0, 6.0, 5.0, 1.0};

  glLightfv(GL_LIGHT0, GL_POSITION, myLightPosition);

  (x, y, z, 1) → point light source, (x, y, z, 0) → directional light source

  - Color

  GLfloat amb0[] = {0.2, 0.4, 0.6, 1.0};

  GLfloat diff0[] = {0.8, 0.9, 0.5, 1.0};

  GLfloat spec0[] = {1.0, 0.8, 1.0, 1.0};

  glLightfv(GL_LIGHT0, GL_AMBIENT, amb0);

  glLightfv(GL_LIGHT0, GL_DIFFUSE, diff0);

  glLightfv(GL_LIGHT0, GL_SPECULAR, spec0);

Using Light Sources in OpenGL

- Creating a light source

- Color

```
GLfloat     amb0[] = {0.2, 0.4, 0.6, 1.0};
GLfloat     diff0[] = {0.8, 0.9, 0.5, 1.0};
GLfloat     spec0[] = {1.0, 0.8, 1.0, 1.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, amb0);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diff0);
glLightfv(GL_LIGHT0, GL_SPECULAR, spec0);
```

$$I_r = I_{ar}\rho_{ar} + I_{dr}\rho_{dr} \times lambert + I_{spr}\rho_{sr} \times phong^f$$

$$I_g = I_{ag}\rho_{ag} + I_{dg}\rho_{dg} \times lambert + I_{spg}\rho_{sg} \times phong^f$$

$$I_b = I_{ab}\rho_{ab} + I_{db}\rho_{db} \times lambert + I_{spb}\rho_{sb} \times phong^f$$

Using Light Sources in OpenGL

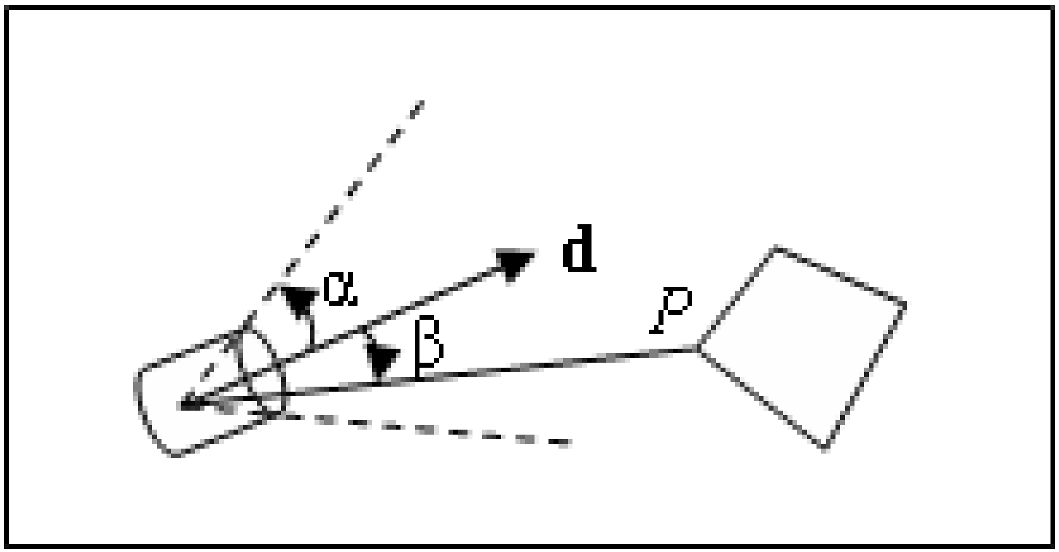
- Splotlights

  glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0); // angle

  glLightf(GL_LIGHT0, GL_SPOT_EXPONENT, 4.0); // ε = 4.0

  GLfloat   dir[] = {2.0, 1.0, -4.0}; // direction

  glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);



Working with material in OpenGL

```
GLfloat      myColor[] = {0.8, 0.2, 0.0, 1.0};
glMaterialfv(GL_FRONT, GL_DIFFUSE, myColor);
```

GL_BACK,
GL_FRONT_AND_BACK

GL_AMBIENT,
GL_SPECULAR,GL_EMISSION

$$I_r = I_{ar}\rho_{ar} + I_{dr}\rho_{dr} \times lambert + I_{spr}\rho_{sr} \times phong^f$$

$$I_g = I_{ag}\rho_{ag} + I_{dg}\rho_{dg} \times lambert + I_{spg}\rho_{sg} \times phong^f$$

$$I_b = I_{ab}\rho_{ab} + I_{db}\rho_{db} \times lambert + I_{spb}\rho_{sb} \times phong^f$$

Working with material in OpenGL

```
GLfloat ambient[] = {0.2, 0.2, 0.2, 1.0};
GLfloat diffuse[] = {1.0, 0.8, 0.0, 1.0};
GLfloat specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat shine = 100.0
glMaterialfv(GL_FRONT, GL_AMBIENT, ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, specular);
glMaterialf(GL_FRONT, GL_SHININESS, shine);
```

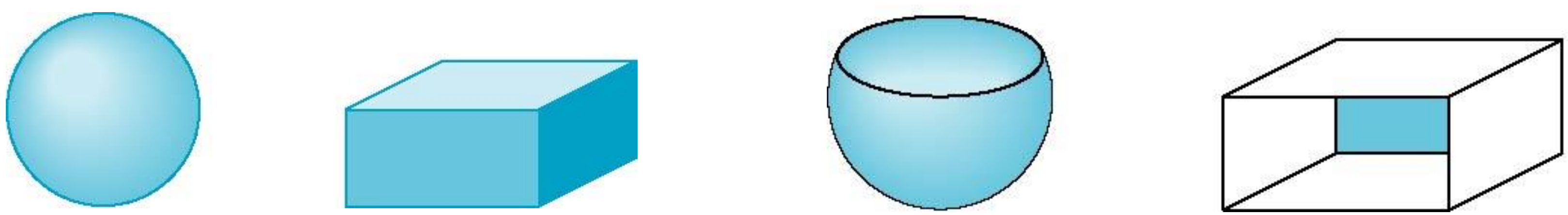
$$I_r = I_{ar}\rho_{ar} + I_{dr}\rho_{dr} \times lambert + I_{spr}\rho_{sr} \times phong^f$$

$$I_g = I_{ag}\rho_{ag} + I_{dg}\rho_{dg} \times lambert + I_{spg}\rho_{sg} \times phong^f$$

$$I_b = I_{ab}\rho_{ab} + I_{db}\rho_{db} \times lambert + I_{spb}\rho_{sb} \times phong^f$$

Front and Back Faces

- The default is shade only front faces which works correctly for convex objects
- If we set two sided lighting, OpenGL will shade both sides of a surface
- Each side can have its own properties which are set by using **GL_FRONT**, **GL_BACK**, or **GL_FRONT_AND_BACK**



back faces not visible　　　　back faces visible

Xem Video Clip: Chương 3 – Phần 7

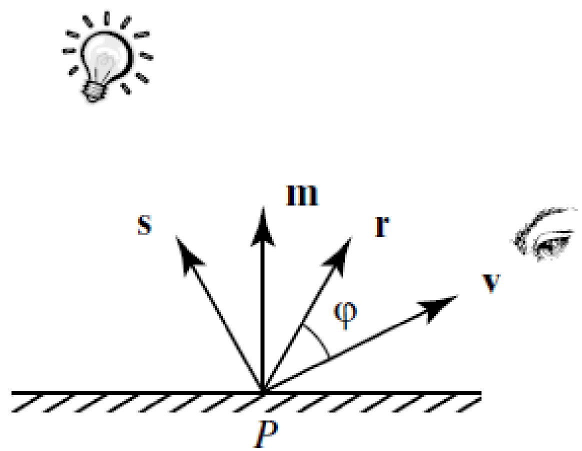
Front and Back Faces

- Specify Front Faces
  - glFrontFace(GL_CCW), glFrontFace(GL_CW)
- Cull Face
  - glEnable(GL_CULL_FACE)
  - glCullFace(GLenum mode);
    - GL_FRONT,
    - GL_BACK,
    - GL_FRONT_AND_BACK

Xem Video Clip: Chương 3 – Phần 7

Computation of Vectors

- **s** and **v** are specified by the application
- Can compute **r** from **s** and **m**
- Problem is determining **m**
- For simple surfaces is can be determined but how we determine **m** differs depending on underlying representation of surface
- OpenGL leaves determination of normal to application



Computation of Vectors

- If the face is flat → face's normal vector is vertices normal vector
- m = (V1 – V2) x (V3 – V4)
- Two problem: 1) two vector nearly parallel, 2) not all the vertices lie in the same plane

$$m_x = \sum_{i=0}^{N-1} (y_i - y_{next(i)})(z_i + z_{next(i)})$$

$$m_y = \sum_{i=0}^{N-1} (z_i - z_{next(i)})(x_i + x_{next(i)})$$

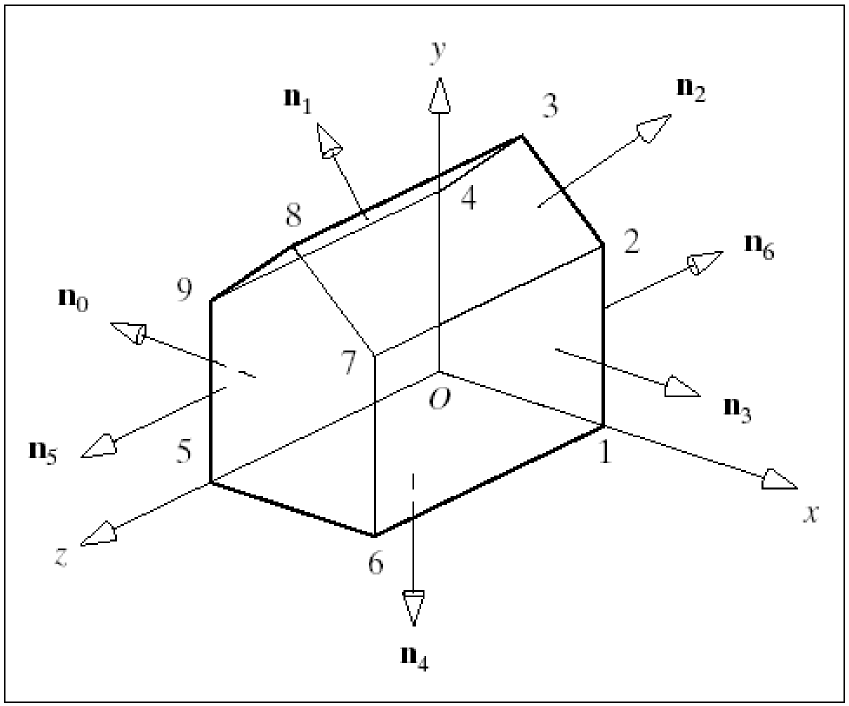
$$m_z = \sum_{i=0}^{N-1} (x_i - x_{next(i)})(y_i + y_{next(i)})$$

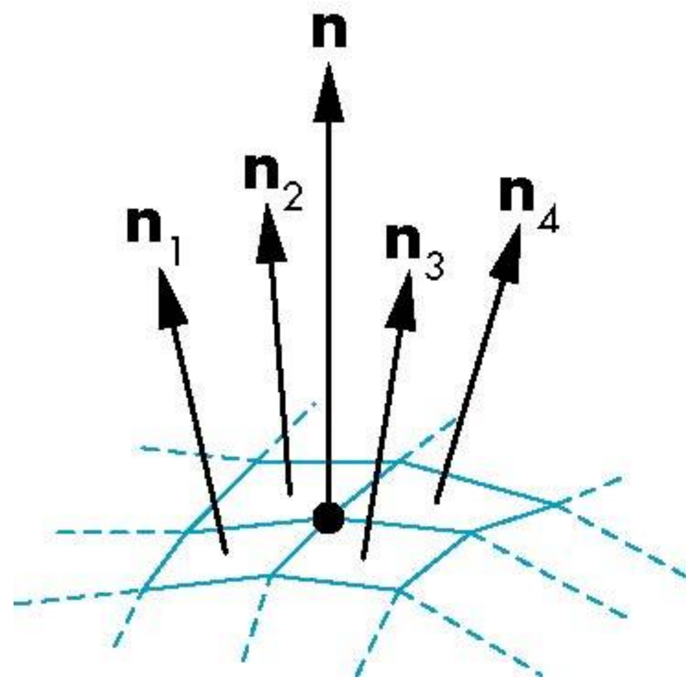
- next(*j*) = (*j* + 1) mod *N*

- Traversed in CCW

- m outward

Computation of Vectors

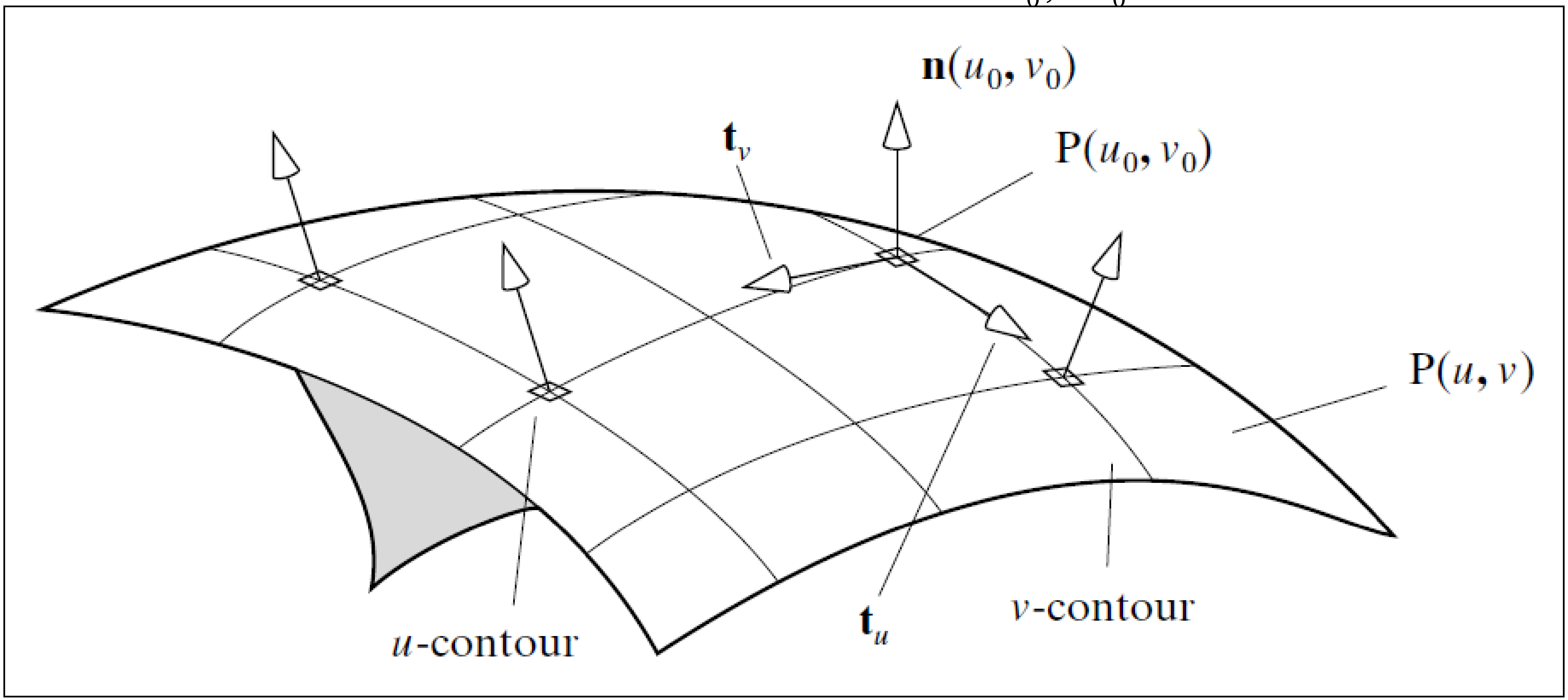




Computation of Vectors

- Normal vector for a surface given parametrically

$$n(u_0, v_0) = \left(\frac{\partial p}{\partial u} \times \frac{\partial p}{\partial v} \right) \Bigg|_{u=u_0, v=v_0}$$



Computation of Vectors

- Normal vector for a surface given implicitly

$$\mathrm{n}(x_0, y_0, z_0) = \nabla F\Big|_{x=x_0, y=y_0, z=z_0} = \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right)\Bigg|_{x=x_0, y=y_0, z=z_0}$$

EX: the implicit form of plane:

$$F(x, y, z) = \mathbf{n} \bullet ((x, y, z) - A) = 0$$

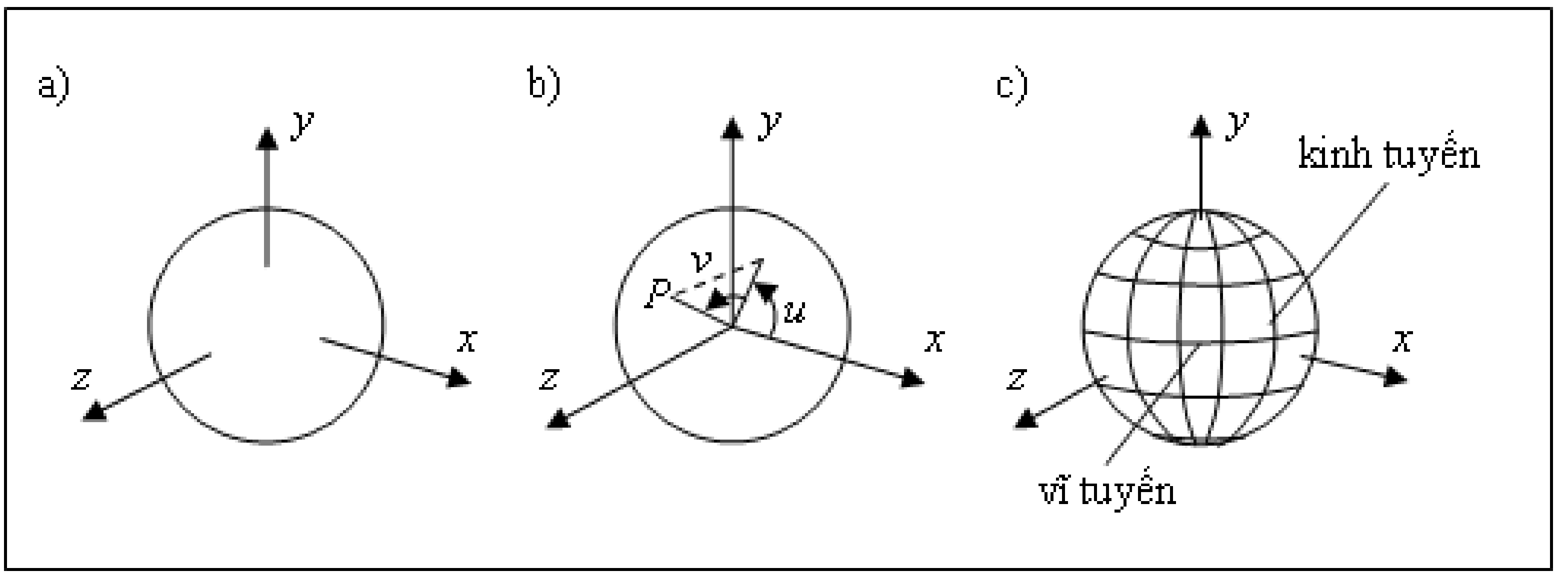
or: $n_x x + n_y y + n_z z - \mathbf{n} \bullet A = 0$

→ normal is (n_x, n_y, n_z)

Generic Shapes

- The sphere:
  - ✓ Implicit form $F(x, y, z) = x^2 + y^2 + z^2 - 1$

    → normal $(2x, 2y, 2z)$
  - ✓ Parametric form $p(u, v) = (\cos(v)\cos(u), \cos(v)\sin(u), \sin(v))$



Generic Shapes

- The sphere
  - ✓ Parametric form p(u, v) = (cos(v)cos(u), cos(v)sin(u), sin(v))
  - ✓ dp/du = (-cos(v)sin(u), cos(v)cos(u), 0)
  - ✓ dp/dv = (-sin(v)cos(u), -sin(v)sin(u), cos(v))
  - ✓ n(u, v) = (dp/du) x (dp/dv) = cos(v)p(u,v)

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

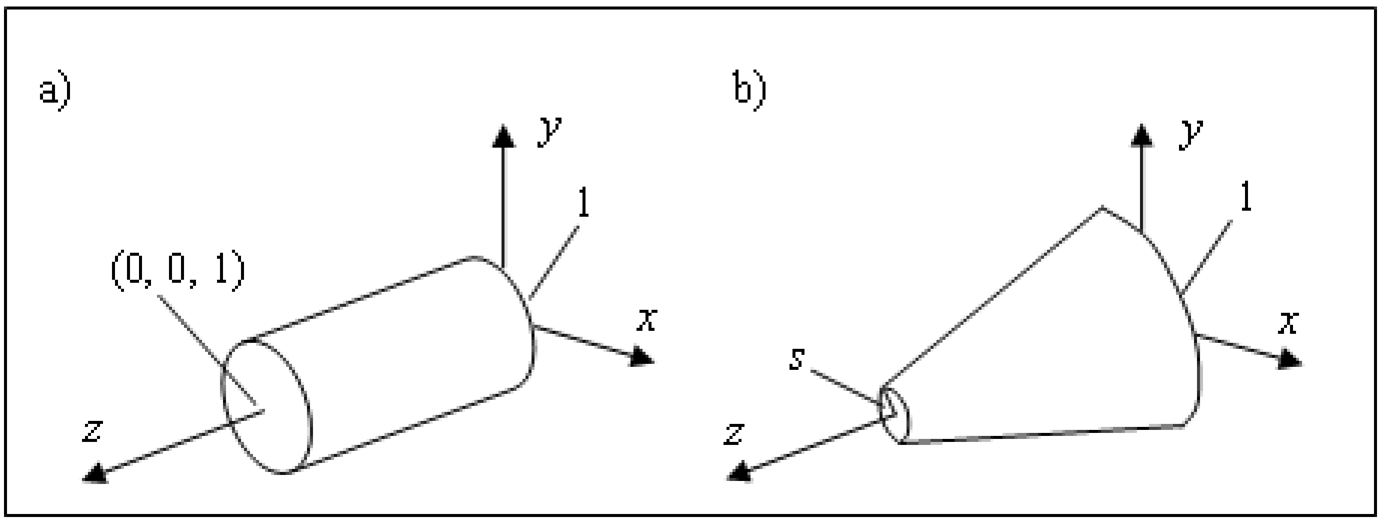
Generic Shapes

- Cylinder:
  - Implicit form $F(x, y, z) = x^2 + y^2 - (1 + (s - 1)z)^2 \qquad 0 < z < 1$

    $\rightarrow \mathbf{n}(x, y, z) = (x, y, -(s - 1)(1 + (s - 1)z))$
  - Parametric form

    $P(u, v) = ((1 + (s - 1)v)\cos(u), (1 + (s - 1)v)\sin(u), v)$

    $\rightarrow \mathbf{n}(u, v) = (\cos(u), \sin(u), 1-s)$



Generic Shape

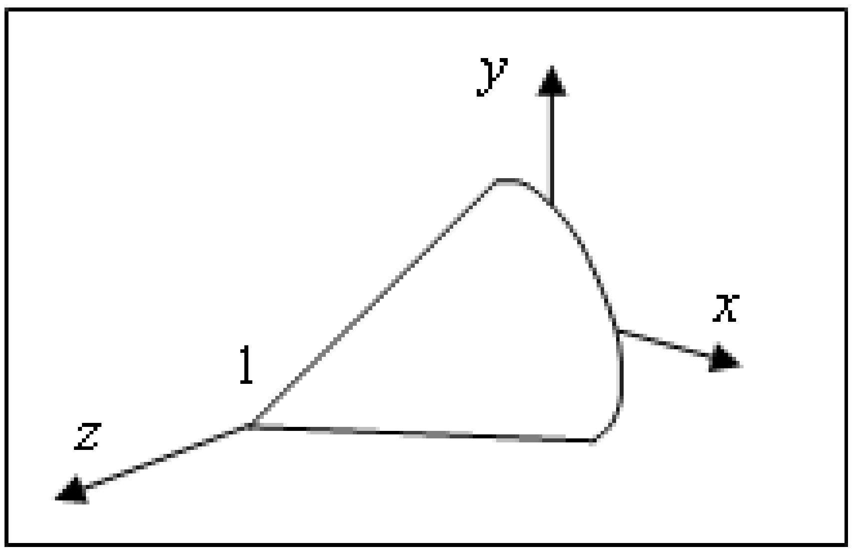
- Cone

  - Implicit: $F(x, y, z) = x^2 + y^2 - (1 - z)^2 = 0 \quad 0 < z < 1$

    $\rightarrow n = (x, y, 1 - z)$

  - parametric $P(u, v) = ((1 - v)\cos(u), (1 - v)\sin(u), v)$

    $\rightarrow n(u, v) = (\cos(u), \sin(u), 1)$



Further Reading

- “**Interactive Computer Graphics: A Topdown Approach Using OpenGL**”, *Edward Angel*
  - Chapter 6: Lighting and Shading
- “Đồ họa máy tính trong không gian ba chiều”, Trần Giang Sơn
  - Chương 3: Tô màu vật thể ba chiều (3.1 → 3.4)