CS 543: Computer Graphics
Lecture 8 (Part I): Shading

Emmanuel Agu

Recall: Setting Light Property

- Define colors and position a light

```
GLfloat light_ambient[] = {0.0, 0.0, 0.0, 1.0};
GLfloat  light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_position[] = {0.0, 0.0, 1.0, 1.0};

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

colors

Position

What if I set Position to (0,0,1,0)?

Recall: Setting Material Example

- Define ambient/diffuse/specular reflection and shininess

```
GLfloat mat_amb_diff[] = {1.0, 0.5, 0.8, 1.0};
GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat shininess[] = {5.0};

glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE,
                    mat_amb_diff);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_speacular);
glMaterialfv(GL_FRONT, GL_SHININESS, shininess);
```

refl. coeff.

(range: dull 0 – very shiny 128)

Recall: Calculating Color at Vertices

- Illumination from a light:

  Illum = ambient + diffuse + specular

  $$= K_a \times I + K_d \times I \times (\cos\theta) + K_s \times I \times \cos^n(\phi)$$

- If there are N lights

  Total illumination for a point P = $\sum$ **(Illum)**

- Sometimes light or surfaces are colored
- Treat R,G and B components separately
- i.e. can specify different RGB values for either light or material

To:

$$Illum\_r = K_{ar} \times I_r + K_{dr} \times I_r \times (\cos\theta) + K_{sr} \times I_r \times \cos^n(\phi)$$

$$Illum\_g = K_{ag} \times I_g + K_{dg} \times I_g \times (\cos\theta) + K_{sg} \times I_g \times \cos^n(\phi)$$

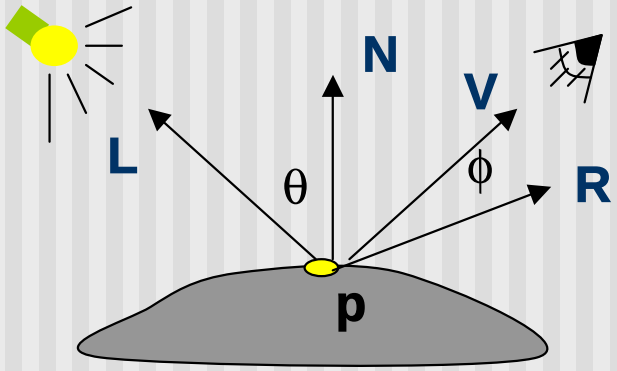
$$Illum\_b = K_{ab} \times I_b + K_{db} \times I_b \times (\cos\theta) + K_{sb} \times I_b \times \cos^n(\phi)$$

Recall: Calculating Color at Vertices

Illum = ambient + diffuse + specular

= Ka x I + Kd x I x (cos θ) + Ks x I x cos(ϕ)^n

- (cos θ) and cos(ϕ) are calculated as dot products of Light vector L, Normal N, and Mirror direction vector R



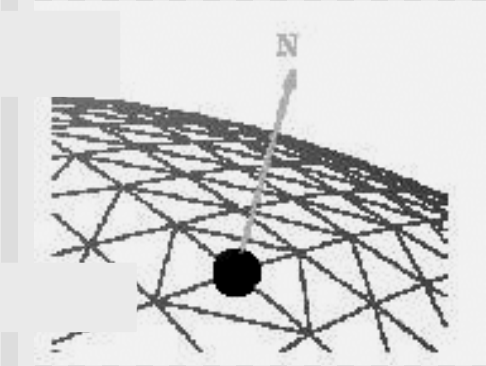
- To give

Illum = Ka x I + Kd x I x (N.L) + Ks x I x (R.V)^n

Surface Normals

- Correct normals are essential for correct lighting
- Associate a normal to each vertex

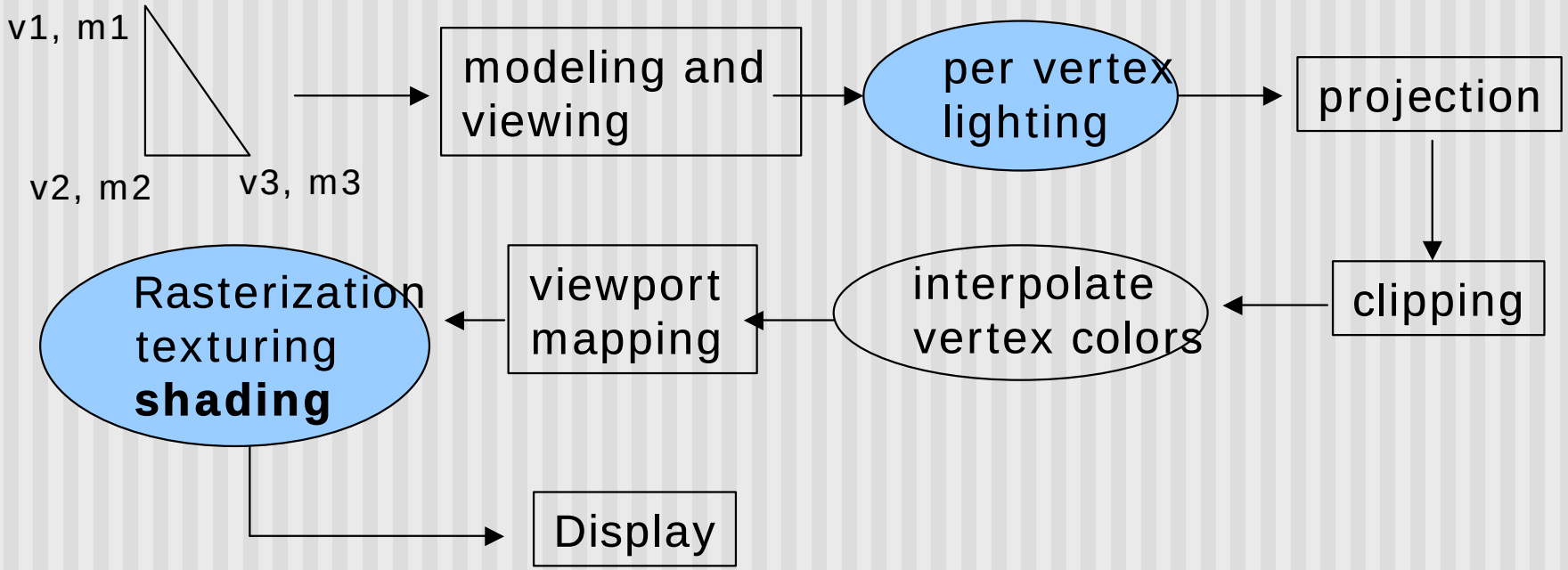
```
glBegin(...)
    glNormal3f(x,y,z)
    glVertex3f(x,y,z)
    ...
    glEnd()
```



- 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

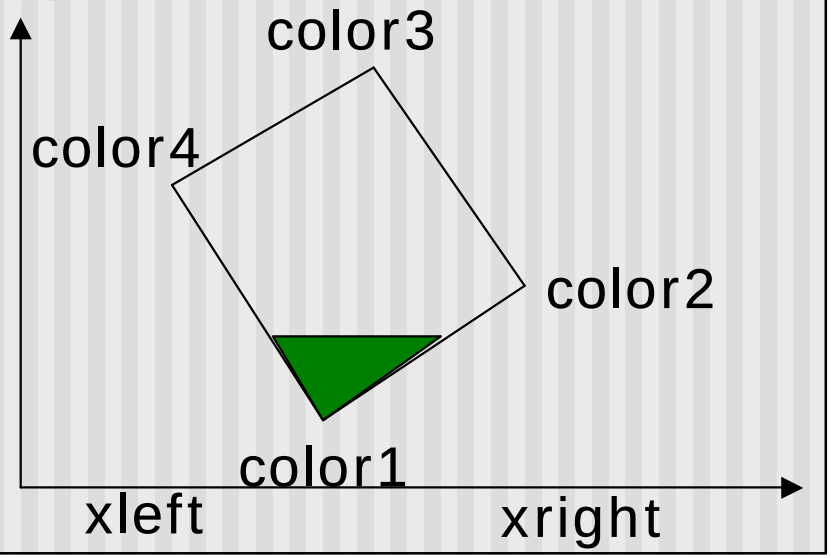
- Light calculation so far is at vertices
- Pixel may not fall right on vertex
- Shading: calculates color to set interior pixel to
- Where are lighting/shading performed in the pipeline?



Example Shading Function (Pg. 432 of Hill)

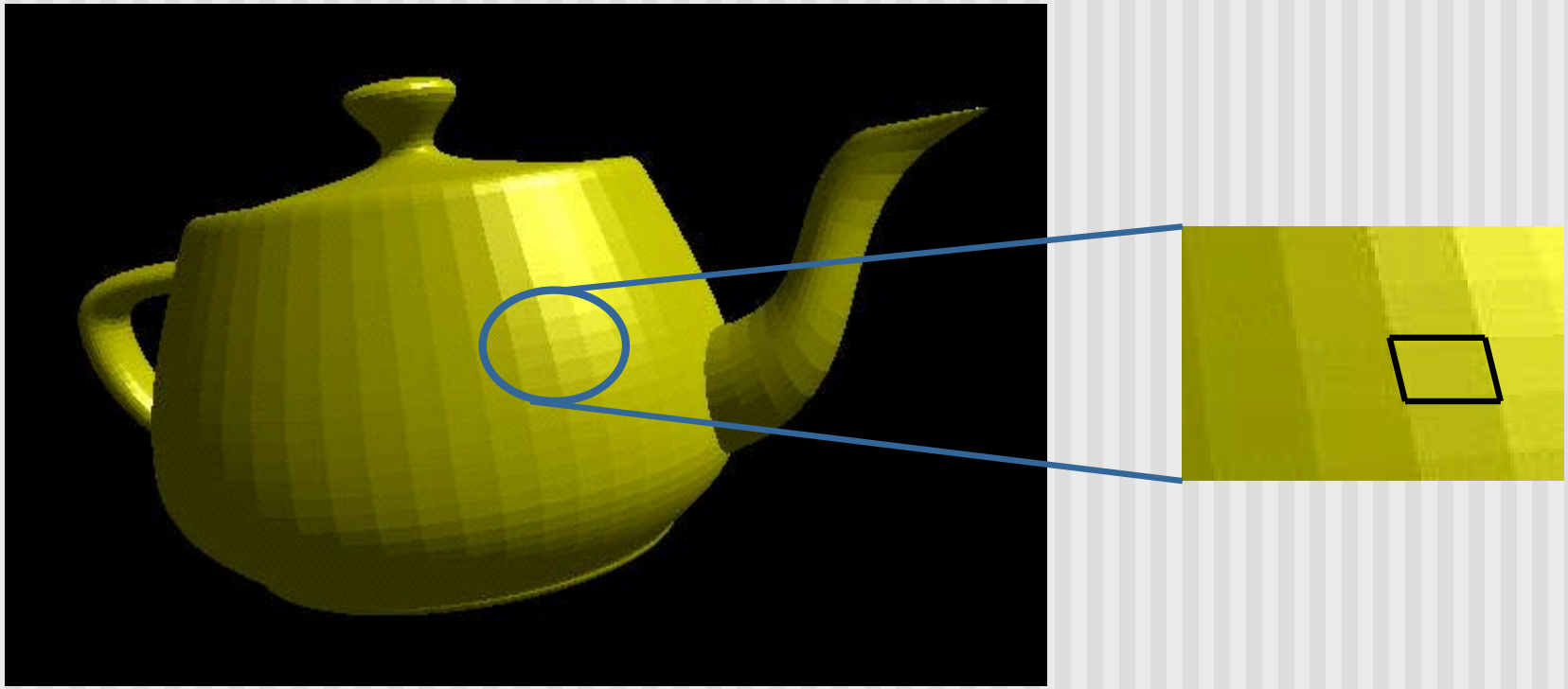
```
for(int y = ybott; y < ytop; y++)
{
   find xleft  and xright
   for(int x = xleft; x < xright; x++)
   {
       find the color c for this pixel
       put c into the pixel at (x, y)
   }
}
```

- Scans pixels, row by row, calculating color for each pixel



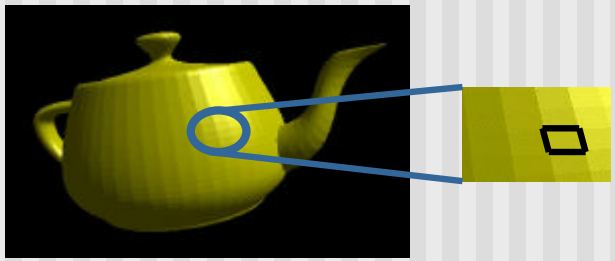
Polygon shading model

- Flat shading - compute lighting once and assign the color to the whole (mesh) polygon



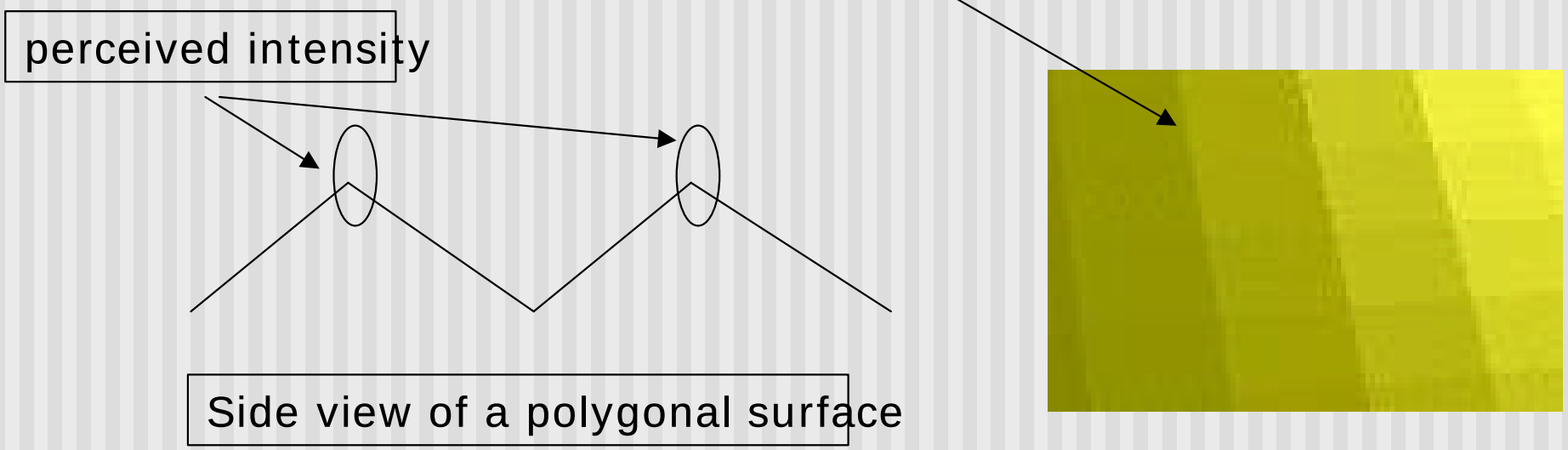
Flat shading

- Only use one vertex normaland 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)



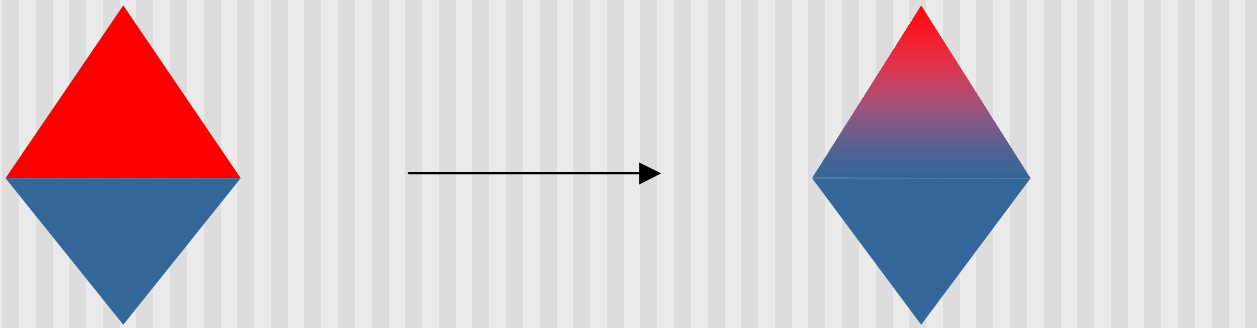
Mach Band Effect

- Flat shading suffers from "mach band effect"
- Mach band effect – human eyes accentuate the discontinuity at the boundary



Smooth shading

- Fix the mach band effect – remove edge discontinuity
- Compute lighting for more points on each face

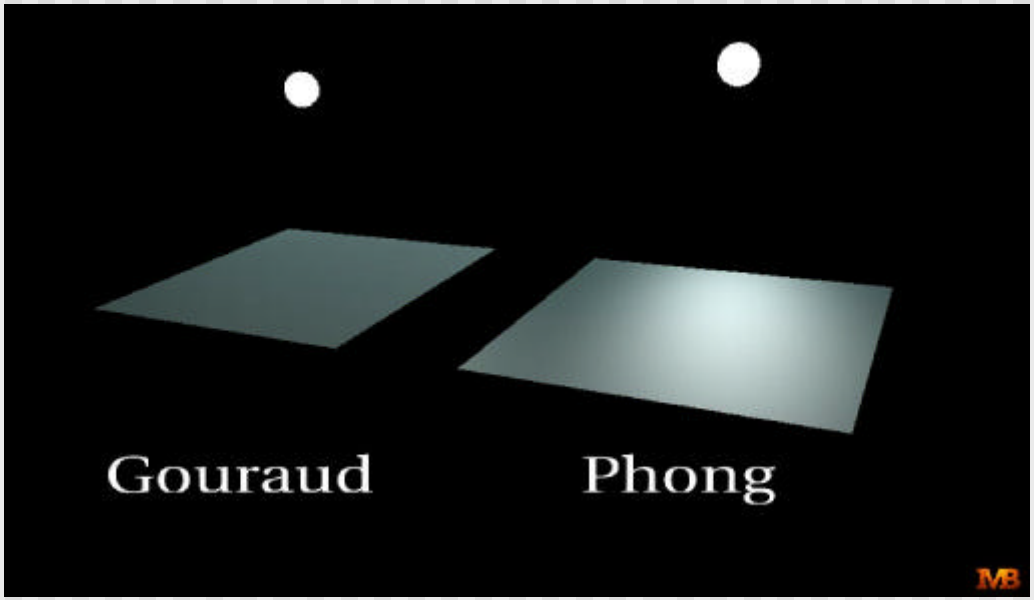


Flat shading

Smooth shading

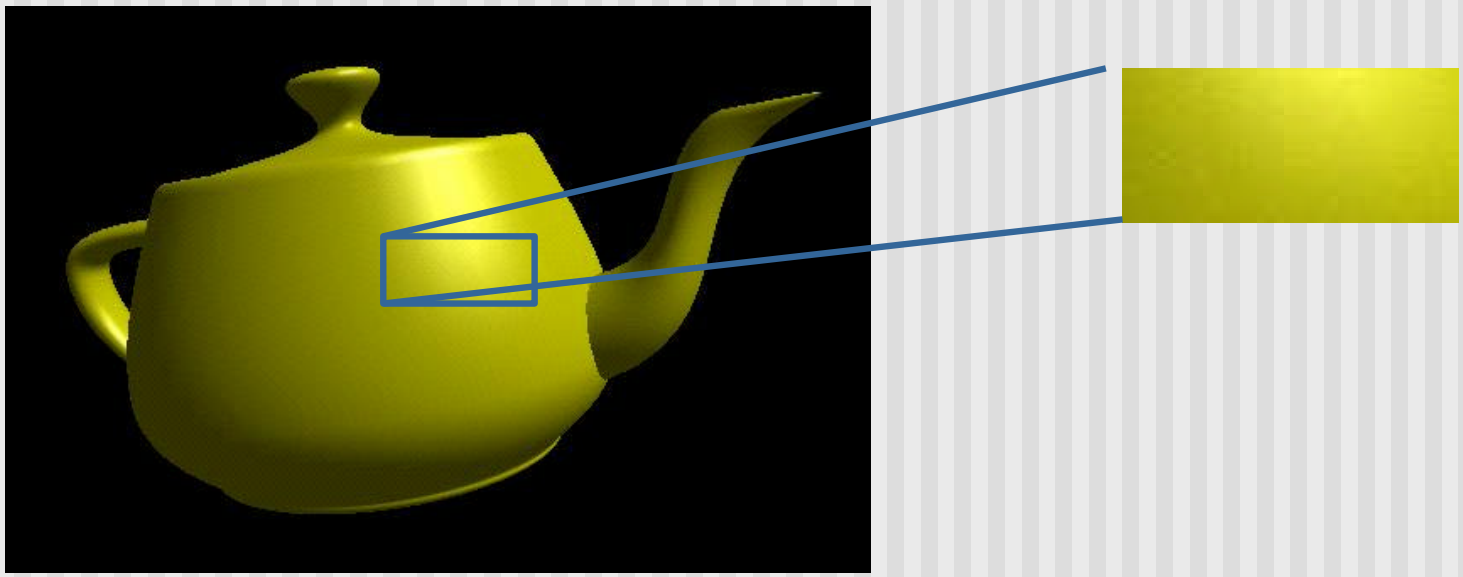
Smooth shading

- Two popular methods:
  - Gouraud shading (used by OpenGL)
  - Phong shading (better specular highlight, not in OpenGL)



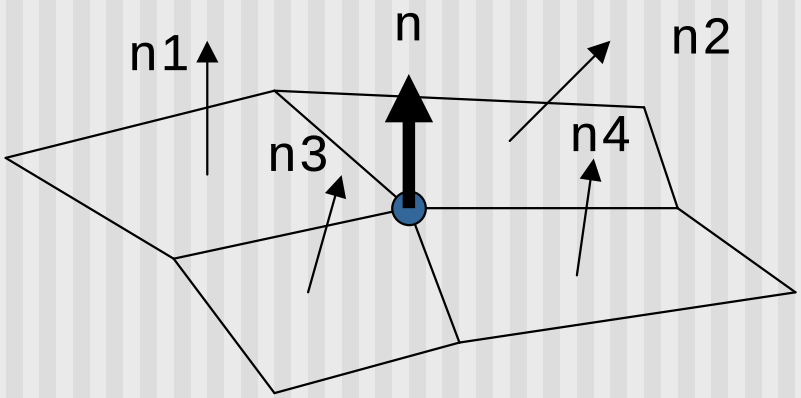
Gouraud Shading

- 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

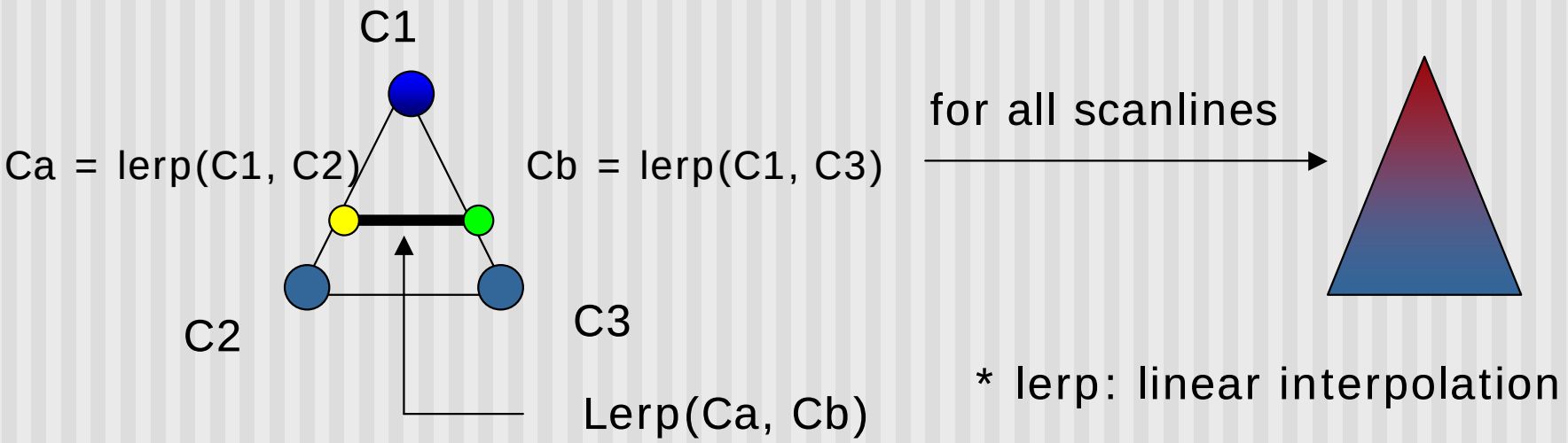
- Per-vertex lighting calculation
- Normal is needed for each vertex
- Per-vertex normal can be computed by averaging the adjust face normals



$$n = (n1 + n2 + n3 + n4) / 4.0$$

Gouraud Shading

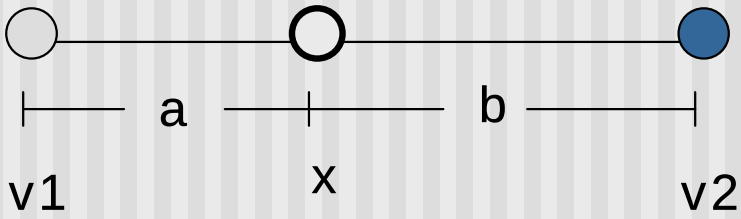
- Compute vertex illumination (color) before the projection transformation
- Shade interior pixels: color interpolation (normals are not needed)



* lerp: linear interpolation

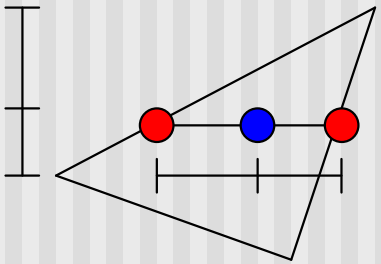
Gouraud Shading

- Linear interpolation



$$x = b / (a+b) * v1 + a/(a+b) * v2$$

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

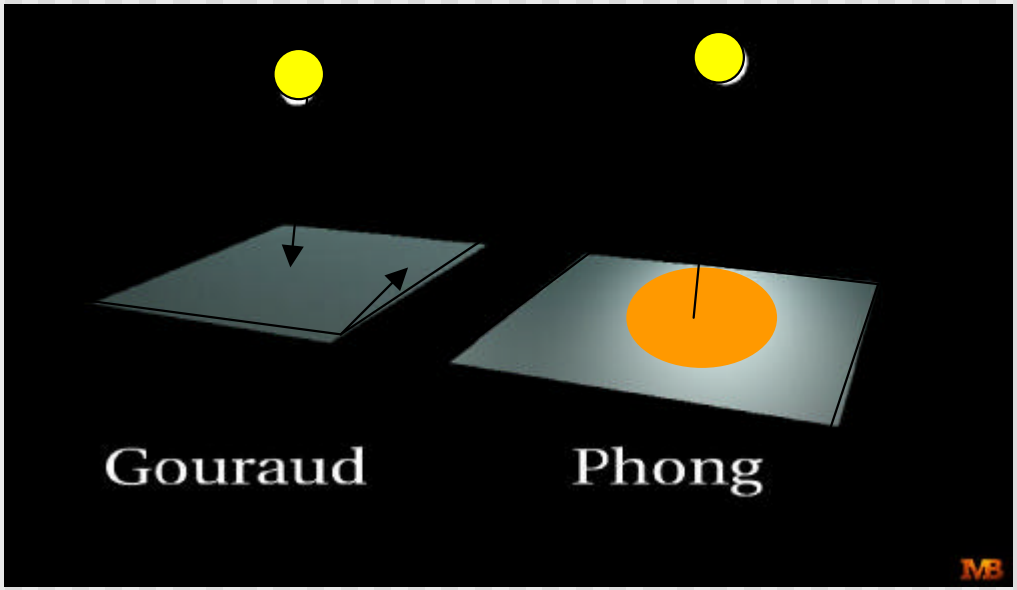


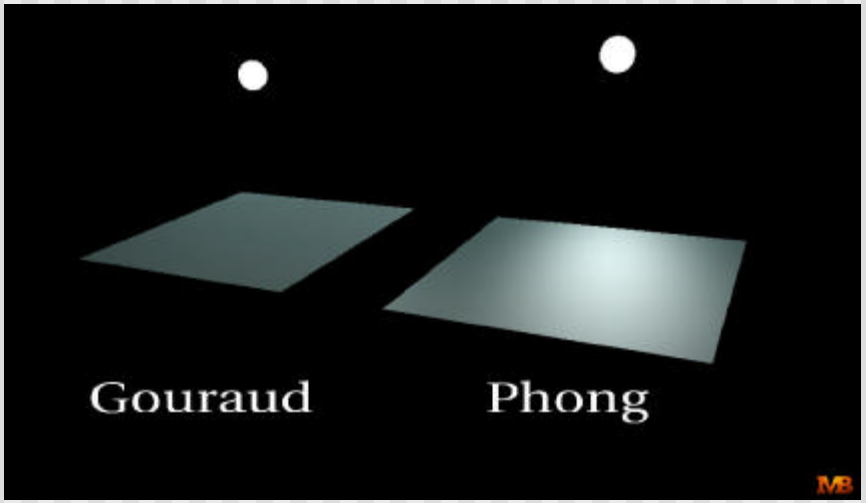
Gouraud Shading Function (Pg. 433 of Hill)

```
for(int y = y_bott; y < y_top; y++) // for each scan line
{
   find x_left  and x_right
   find color_left  and color_right
   color_inc = (color_right - color_left)/ (x_right - x_left)
   for(int x = x_left, c = color_left; x < x_right;
                                        x++, c+ = color_inc)
   {
       put c into the pixel at (x, y)
   }
}
```

Gouraud Shading Problem

- Lighting in the polygon interior can be inaccurate



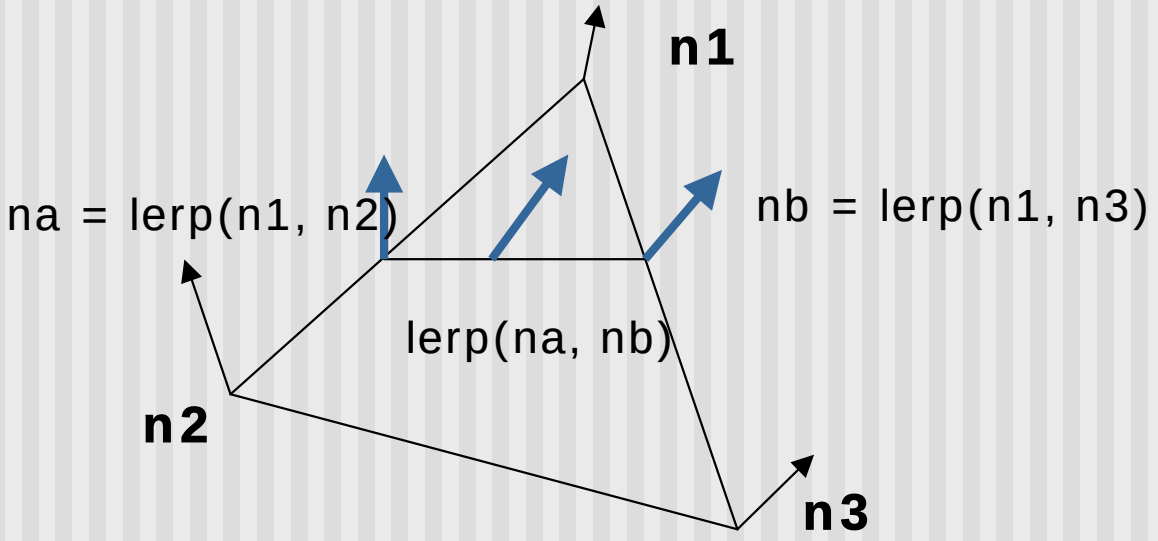


Phong Shading

- 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)

Phong Shading

- Normal interpolation



- Slow – not supported by OpenGL and most graphics hardware

References

- Hill, chapter 8