Computer Graphics (CS 4731)
Lecture 18: Lighting, Shading and Materials (Part 3)

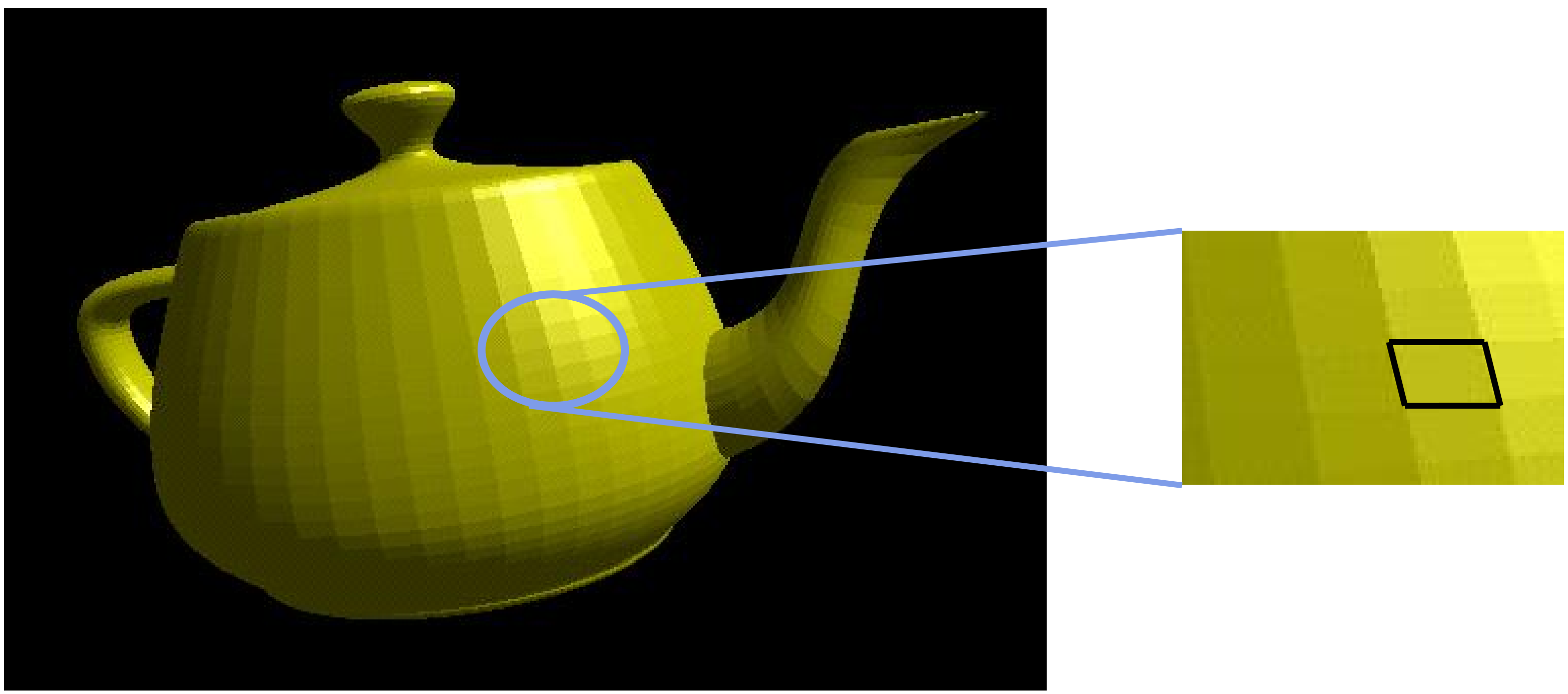
Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

Recall: Flat Shading

- compute lighting once for each face, assign color to whole face



Recall: Flat Shading Implementation

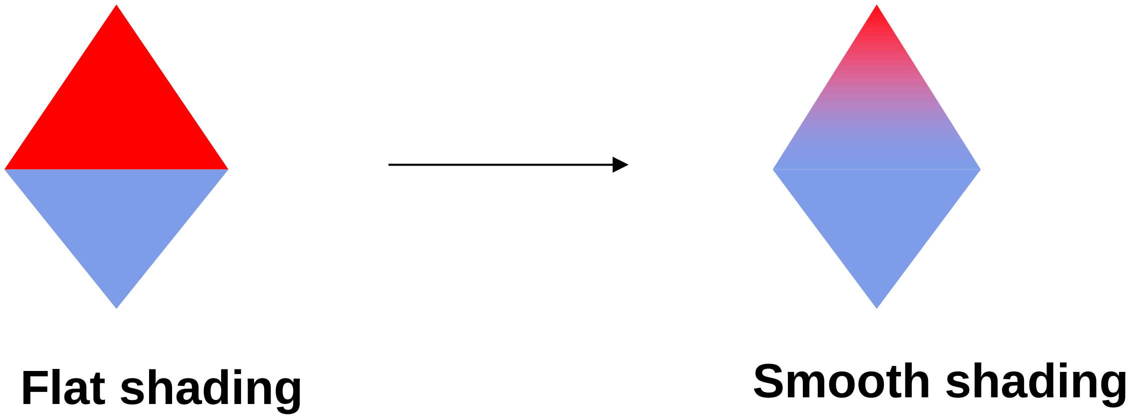
```
flat out vec4 color;      //vertex shader
 ……
  color = ambient + diffuse + specular;
   color.a = 1.0;

flat in vec4 color;    //fragment shader

void main() {
   gl_FragColor = color;
}
```

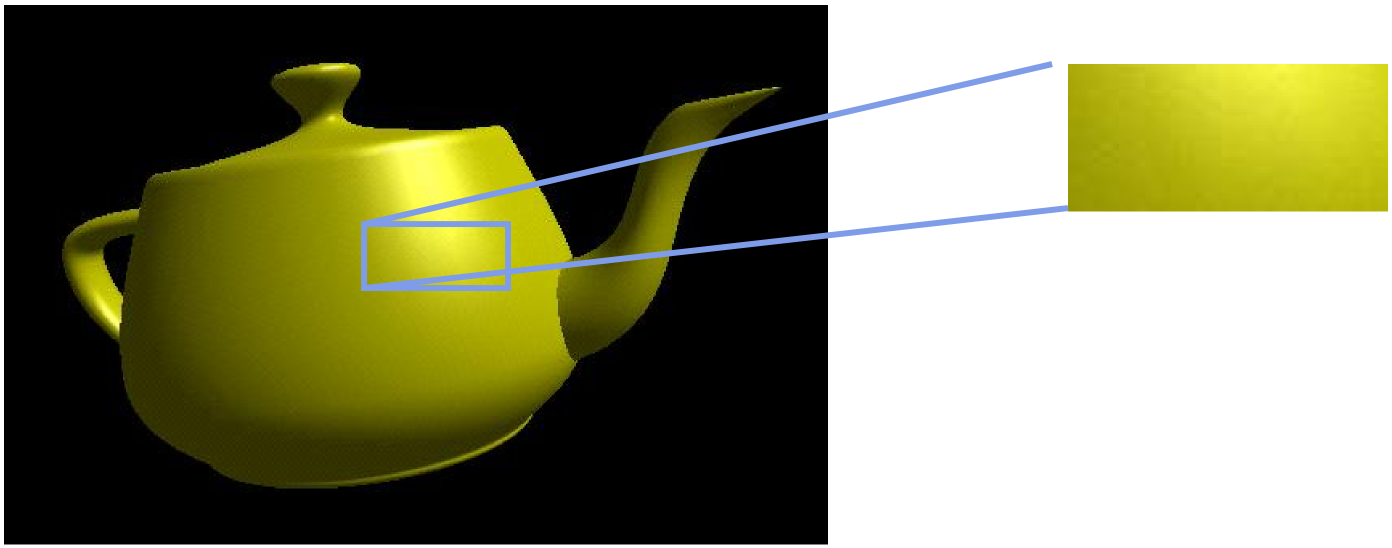
Recall: Smooth shading

- 2 popular methods:
  - Gouraud shading
  - Phong shading



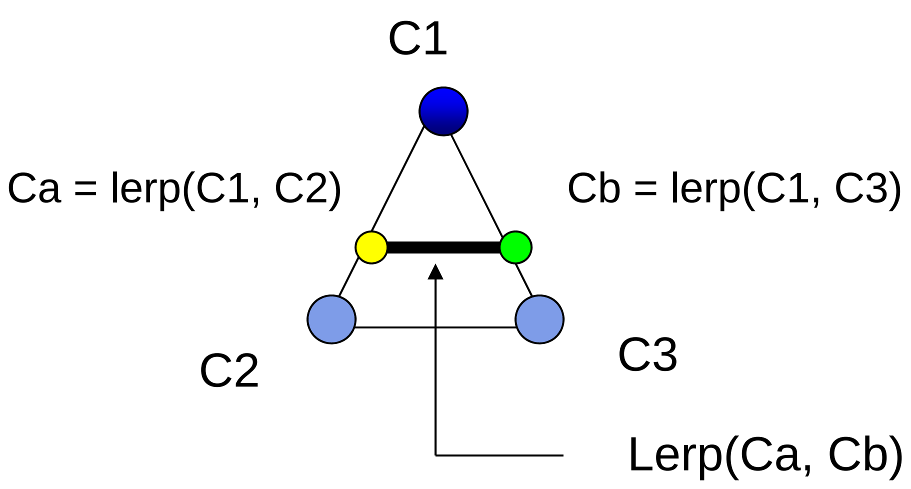
Recall: Gouraud Shading

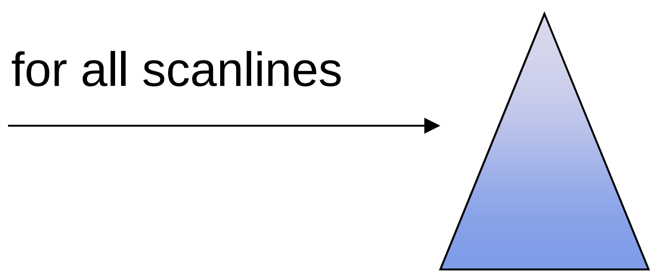
- **Vertex shader:** lighting calculated for each vertex
- Default shading. Just suppress keyword **flat**
- Colors interpolated for interior pixels
- **Interpolation?** Assume linear change from one vertex color to another



Gouraud Shading

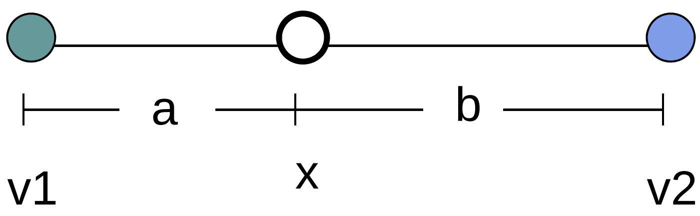
- Compute vertex color in vertex shader
- Shade interior pixels: vertex color **interpolation**





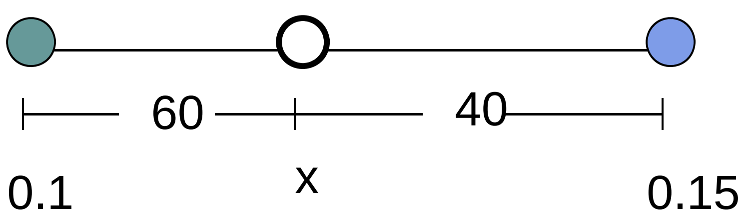
* lerp: linear interpolation

Linear interpolation Example



$$x = \frac{b}{(a+b)} * v1 + \frac{a}{(a+b)} * v2$$

- If a = 60, b = 40
- RGB color at v1 = (0.1, 0.4, 0.2)
- RGB color at v2 = (0.15, 0.3, 0.5)
- Red value of v1 = 0.1, red value of v2 = 0.15

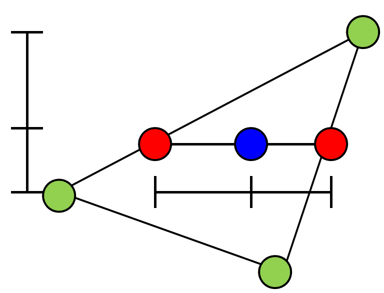


Red value of x = 40 /100 * 0.1 + 60/100 * 0.15
= 0.04 + 0.09 = 0.13

Similar calculations for Green and Blue values

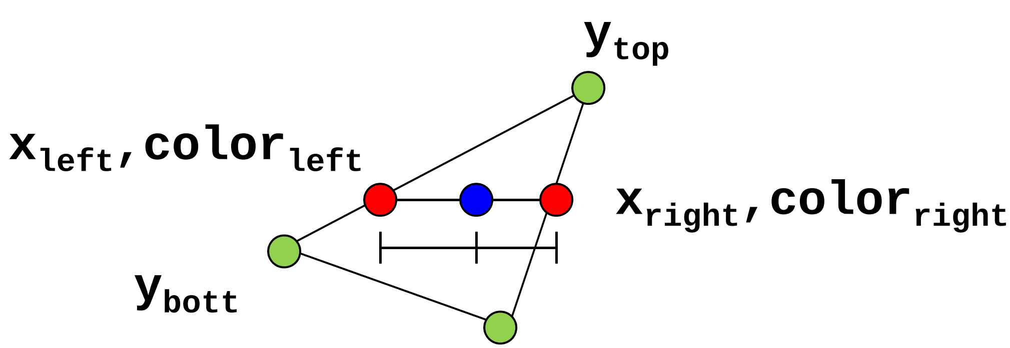
Gouraud Shading

- Interpolate triangle color
  1. Interpolate **y distance** of end points (green dots) to get color of two end points in scanline (red dots)
  2. Interpolate **x distance** of two ends of scanline (red dots) to get color of pixel (blue dot)



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)
   }
}
```

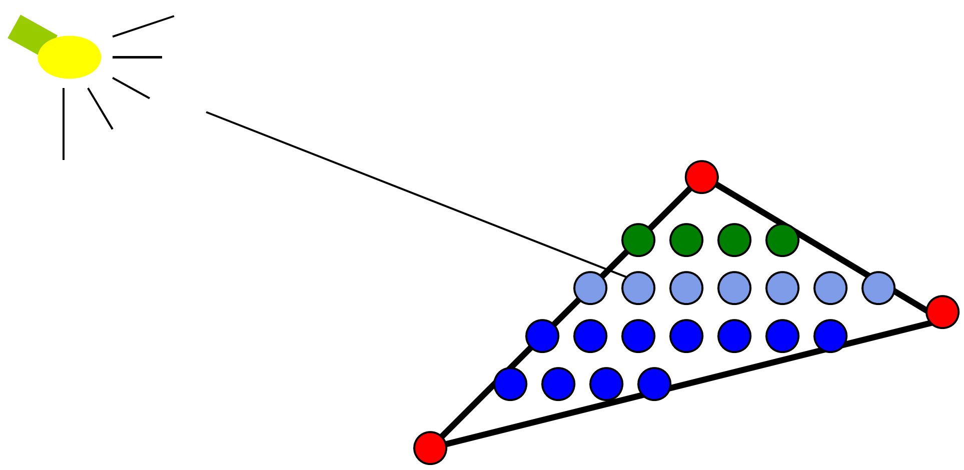


Gouraud Shading Implemenation

- Vertex lighting interpolated across entire face pixels if passed to fragment shader in following way
  1. **Vertex shader:** Calculate output color in vertex shader, Declare output vertex color as **out**

$$I = k_d I_d \; \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^{\beta} + k_a I_a$$

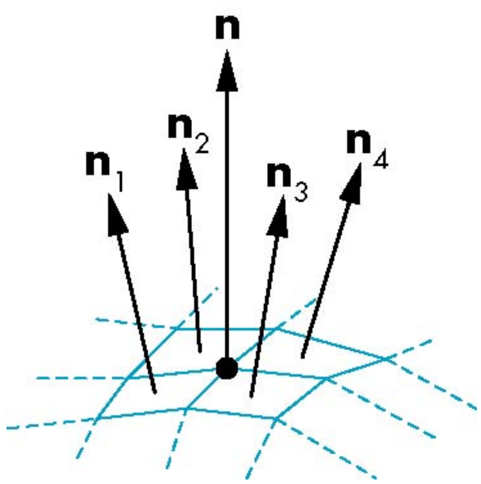
  2. **Fragment shader:** Declare color as **in**, use it, already interpolated!!



Calculating Normals for Meshes

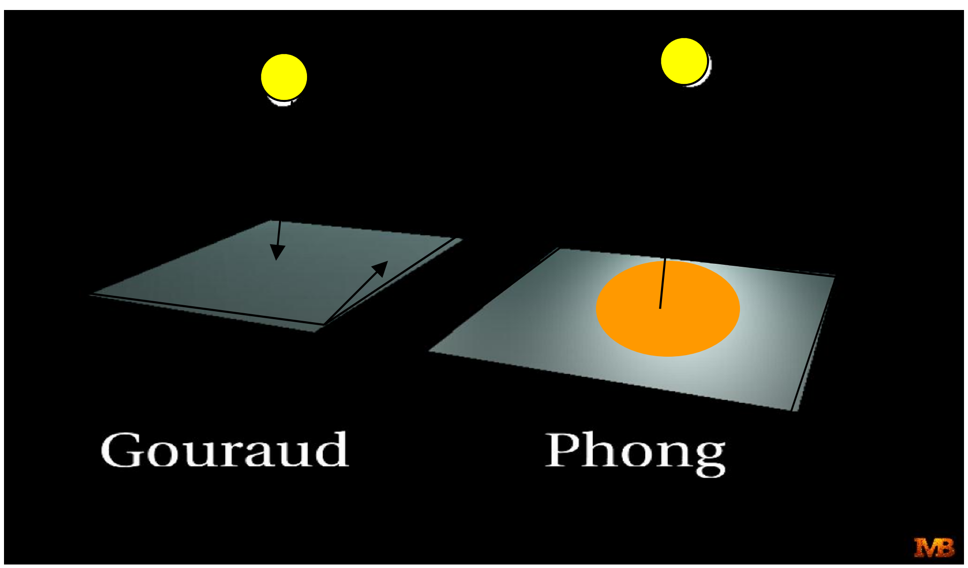
- For meshes, already know how to calculate face normals (e.g. Using Newell method)
- For polygonal models, Gouraud proposed using average of normals around a mesh vertex

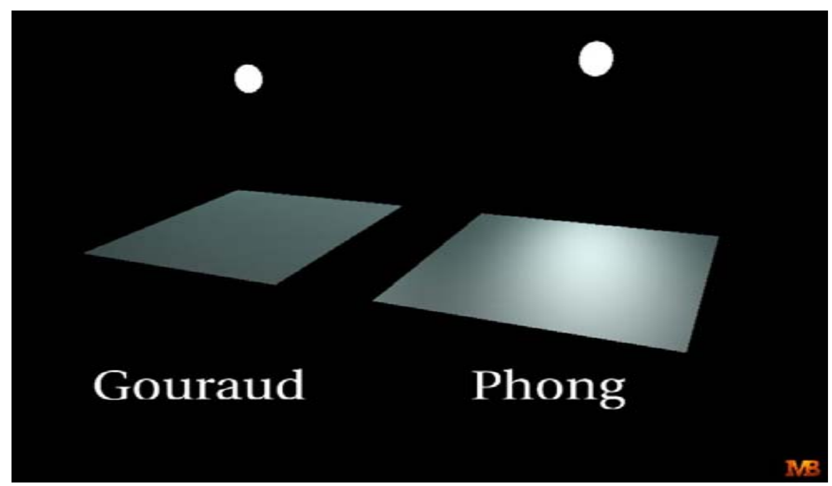
$$\mathbf{n} = (\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4)/\ |\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4|$$



Gouraud Shading Problem

- If polygon mesh surfaces have high curvatures, Gouraud shading may show edges
- Lighting in the polygon interior can be inaccurate
- Phong shading may look smooth



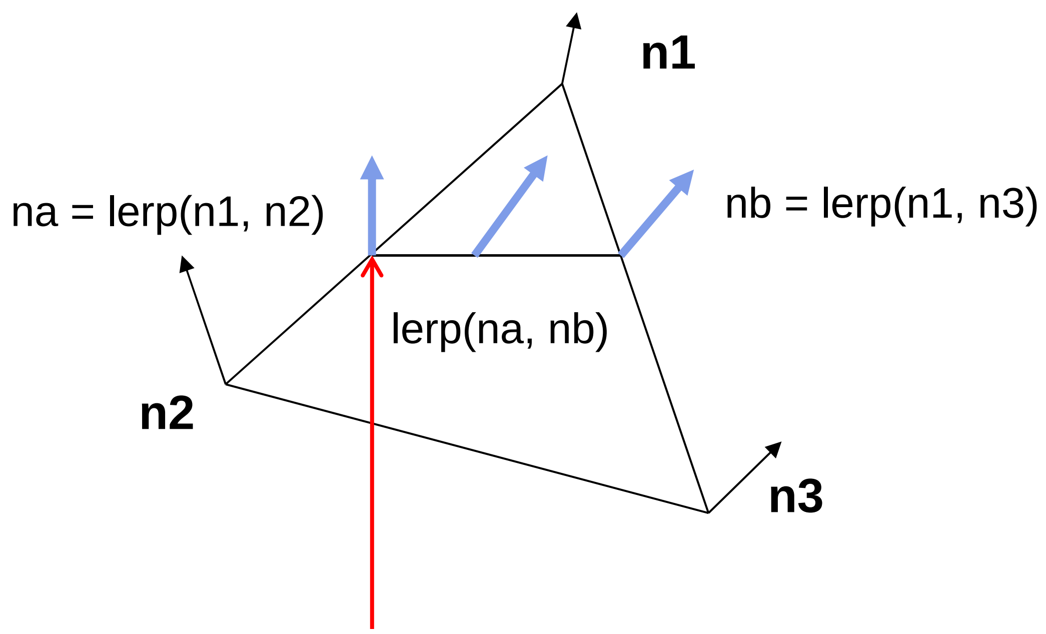


Phong Shading

- Need normals for all pixels – not provided by user
- Instead of interpolating vertex color
  - Interpolate **vertex normal and vectors** to calculate normal (and vectors) at each ***each pixel*** inside polygon
  - Use pixel normal to calculate Phong at pixel (**per pixel lighting**)
- Phong shading algorithm interpolates normals and compute lighting in fragment shader

Phong Shading (Per Fragment)

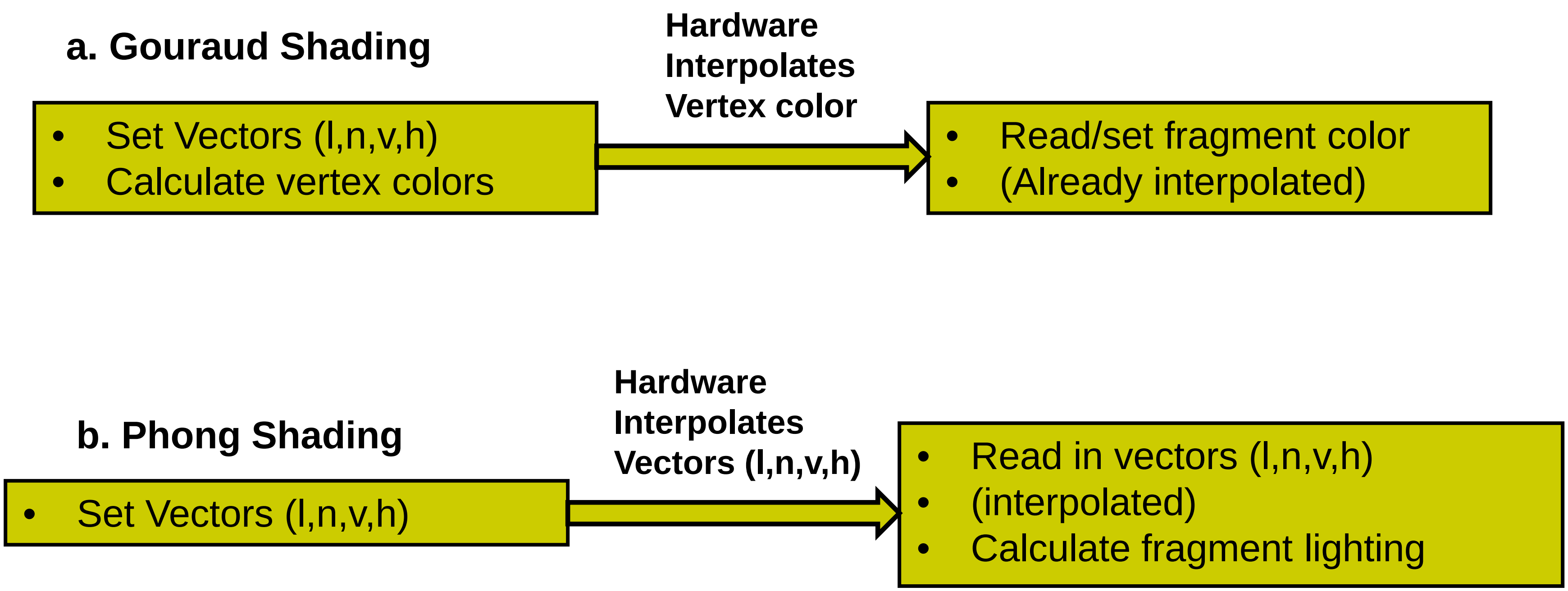
- Normal interpolation



At each pixel, need to interpolate
Normals (n) and vectors v and l

Gouraud Vs Phong Shading Comparison

- Phong shading more work than Gouraud shading
  - Move lighting calculation to fragment shaders
  - Just set up vectors (l,n,v,h) in vertex shader



Per-Fragment Lighting Shaders I

```
// vertex shader
in vec4 vPosition;
in vec3 vNormal;

// output values that will be interpolatated per-fragment
out vec3 fN;
out vec3 fE;
out vec3 fL;

uniform mat4 ModelView;
uniform vec4 LightPosition;
uniform mat4 Projection;
```

Declare variables **n, v, l** as **out** in vertex shader

Per-Fragment Lighting Shaders II

```
void main()
{
    fN = vNormal;
    fE = -vPosition.xyz;
    fL = LightPosition.xyz;

    if( LightPosition.w != 0.0 ) {
        fL = LightPosition.xyz - vPosition.xyz;
    }

    gl_Position = Projection*ModelView*vPosition;
}
```

Set variables **n, v, l** in vertex shader

Per-Fragment Lighting Shaders III

```
// fragment shader

// per-fragment interpolated values from the vertex shader
in vec3 fN;
in vec3 fL;
in vec3 fE;

uniform vec4 AmbientProduct, DiffuseProduct, SpecularProduct;
uniform mat4 ModelView;
uniform vec4 LightPosition;
uniform float Shininess;
```

Declare vectors n, v, l as **in** in fragment shader (**Hardware interpolates these vectors**)

Per=Fragment Lighting Shaders IV

```
void main()
{
   // Normalize the input lighting vectors

   vec3 N = normalize(fN);
   vec3 E = normalize(fE);
   vec3 L = normalize(fL);

   vec3 H = normalize( L + E );
   vec4 ambient = AmbientProduct;
```

Use interpolated variables n, v, l in fragment shader

$$I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^{\beta} + k_a I_a$$

Per-Fragment Lighting Shaders V

```
float Kd = max(dot(L, N), 0.0);
   vec4 diffuse = Kd*DiffuseProduct;

   float Ks = pow(max(dot(N, H), 0.0), Shininess);
   vec4 specular = Ks*SpecularProduct;

   // discard the specular highlight if the light's behind the vertex
   if( dot(L, N) < 0.0 )
        specular = vec4(0.0, 0.0, 0.0, 1.0);

   gl_FragColor = ambient + diffuse + specular;
   gl_FragColor.a = 1.0;
}
```

Use interpolated variables n, v, l in fragment shader

$$I = k_d I_d \; \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^{\beta} + k_a I_a$$

Toon (or Cel) Shading

- Non-Photorealistic (NPR) effect
- Shade in bands of color

Toon (or Cel) Shading

- How?
- Consider **(l · n)** diffuse term (or cos Θ) term

$$I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^{\beta} + k_a I_a$$

- Clamp values to ranges to get toon shading effect

$\mathbf{l} \cdot \mathbf{n}$	Value used
Between 0.75 and 1	0.75
Between 0.5 and 0.75	0.5
Between 0.25 and 0.5	0.25
Between 0.0 and 0.25	0.0

References

- Interactive Computer Graphics (6th edition), Angel and Shreiner
- Computer Graphics using OpenGL (3rd edition), Hill and Kelley