Recall: Flat Shading

- compute lighting once for each face, assign color to whole face
Recall: Flat Shading Implementation

```cpp
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

\[ C_a = \text{lerp}(C_1, C_2) \]
\[ C_b = \text{lerp}(C_1, C_3) \]

Lerp(Ca, Cb) for all scanlines

* lerp: linear interpolation
Linear interpolation Example

- 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 $v2 = 0.15$

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

Red value of $x = \frac{40}{100} \cdot 0.1 + \frac{60}{100} \cdot 0.15 = 0.04 + 0.09 = 0.13$

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 \( x_{\text{left}} \) and \( x_{\text{right}} \)
    find \( \text{color}_{\text{left}} \) and \( \text{color}_{\text{right}} \)
    \( \text{color}_{\text{inc}} = (\text{color}_{\text{right}} - \text{color}_{\text{left}}) / (x_{\text{right}} - x_{\text{left}}) \)
    for(int x = x_{\text{left}}, c = color_{\text{left}}; x < x_{\text{right}}; x++, c+ = color_{\text{inc}})
    {
        put c into the pixel at \((x, y)\)
    }
}
Gouraud Shading Implementation

- 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

\[ n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|} \]
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

a. Gouraud Shading
- Set Vectors (l,n,v,h)
- Calculate vertex colors
- Hardware Interpolates Vertex color
- Read/set fragment color
  - (Already interpolated)

b. Phong Shading
- Set Vectors (l,n,v,h)
- Hardware Interpolates Vectors (l,n,v,h)
- Read in vectors (l,n,v,h)
  - (interpolated)
  - Calculate fragment lighting
// vertex shader
in vec4 vPosition;
in vec3 vNormal;

// output values that will be interpolated 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
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;
}
Per-Fragment Lighting Shaders III

// fragment shader

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

Declare vectors n, v, l as in in fragment shader

(Computer hardware interpolates these vectors)

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

    I = k_d I_d l · n + k_s I_s (n · h) ^ β + k_a I_a
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;

I = k_d I_d l \cdot n + k_s I_s (n \cdot 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 $(l \cdot n)$ diffuse term (or $\cos \theta$) term

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

- Clamp values to ranges to get toon shading effect

<table>
<thead>
<tr>
<th>$l \cdot n$</th>
<th>Value used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between 0.75 and 1</td>
<td>0.75</td>
</tr>
<tr>
<td>Between 0.5 and 0.75</td>
<td>0.5</td>
</tr>
<tr>
<td>Between 0.25 and 0.5</td>
<td>0.25</td>
</tr>
<tr>
<td>Between 0.0 and 0.25</td>
<td>0.0</td>
</tr>
</tbody>
</table>
References