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**

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

For all scanlines

*(lerp: linear interpolation)*
Linear interpolation Example

If \( a = 60 \), \( b = 40 \)

- RGB color at \( v_1 = (0.1, 0.4, 0.2) \)
- RGB color at \( v_2 = (0.15, 0.3, 0.5) \)
- Red value of \( v_1 = 0.1 \), red value of \( v_2 = 0.15 \)

\[
x = \frac{b}{a+b} \cdot v_1 + \frac{a}{a+b} \cdot v_2
\]

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)

```java
for(int y = ybott; y < ytop; 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 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

\[ \mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4}{|\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4|} \]
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

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

**b. Phong Shading**
- Set Vectors (l,n,v,h)
- Read in vectors (l,n,v,h)
  - (interpolated)
- Calculate fragment lighting
Per-Fragment Lighting Shaders I

// 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;
}
```glsl
// 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 (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

    // Use interpolated variables n, v, l in fragment shader
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;

\[ I = k_d I_d \cdot n + k_s I_s (n \cdot h)^\beta + k_a I_a \]
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>
BRDF Evolution

- BRDFs have evolved historically
- 1970’s: Empirical models
  - Phong’s illumination model
- 1980s:
  - Physically based models
  - Microfacet models (e.g. Cook Torrance model)
- 1990’s
  - Physically-based appearance models of specific effects (materials, weathering, dust, etc)
- Early 2000’s
  - Measurement & acquisition of static materials/lights (wood, translucence, etc)
- Late 2000’s
  - Measurement & acquisition of time-varying BRDFs (ripening, etc)
Physically-Based Shading Models

- Phong model produces pretty pictures
- Cons: empirical (fudged?) \( \cos(\alpha \phi) \), plastic look
- Shaders can implement better lighting/shading models
- Big trend towards Physically-based lighting models
- Physically-based?
  - Based on physics of how light interacts with actual surface
  - Apply Optics/Physics theories
- Classic: Cook-Torrance shading model (TOGS 1982)
Cook-Torrance Shading Model

- Same ambient and diffuse terms as Phong
- New, better specular component than \( \cos^{\alpha} \phi \),

\[
\cos^{\alpha} \phi \rightarrow \frac{F(\phi, \eta)DG}{(n \cdot v)}
\]

- Where
  - D - Distribution term
  - G – Geometric term
  - F – Fresnel term
Distribution Term, D

- **Idea**: surfaces consist of small V-shaped **microfacets (grooves)**

- Many grooves at each surface point
- Grooves facing a direction contribute
- $D(\delta)$ term: what fraction of grooves facing each angle $\delta$
- E.g. half of grooves at hit point face 30 degrees, etc
Cook-Torrance Shading Model

- Define angle $\delta$ as deviation of $h$ from surface normal
- Only microfacets with pointing along halfway vector, $h = s + v$, contributes

\[ D(\delta) = \frac{1}{4m^2 \cos^4(\delta)} e^{-\left(\frac{\tan(\delta)}{m}\right)^2} \]

- $m$ expresses roughness of surface. How?
Cook-Torrance Shading Model

- $m$ is Root-mean-square (RMS) of slope of V-groove
- $m = 0.2$ for nearly smooth
- $m = 0.6$ for very rough
Self-Shadowing (G Term)

- Some grooves on extremely rough surface may block other grooves
Geometric Term, G

- Surface may be so rough that interior of grooves is blocked from light by edges
- Self blocking known as **shadowing** or **masking**
- Geometric term G accounts for this
- Break G into 3 cases:
  - G, case a: No self-shadowing (light in-out unobstructed)

  ![Diagram of self-shadowing](image)

- Mathematically, G = 1
Geometric Term, $G$

- $G_m$, case b: No blocking on entry, blocking of exitting light (masking)

- Mathematically,

$$G_m = \frac{2(n \cdot h)(n \cdot s)}{h \cdot s}$$
Geometric Term, $G$

- $G_s$, case c: blocking of incident light, no blocking of exiting light (shadowing)
- Mathematically,

$$G_s = \frac{2(n \cdot h)(n \cdot v)}{h \cdot s}$$

- $G$ term is minimum of 3 cases, hence

$$G = (1, G_m, G_s)$$
Fresnel Term, $F$

- So, again recall that specular term
  
  $$spec = \frac{F(\phi, \eta)DG}{(n \cdot v)}$$

- Microfacets not perfect mirrors
- $F$ term, $F(\phi, \eta)$ gives fraction of incident light reflected
  
  $$F = \frac{1}{2} \frac{(g-c)^2}{(g+c)^2} \left[ 1 + \left( \frac{c(g+c)-1}{c(g-c)-1} \right)^2 \right]$$

  $F$ is function of material and incident angle

- where $c = \cos(\phi) = n \cdot s$ and $g^2 = \eta^2 + c^2 + 1$
- $\phi$ is incident angle, $\eta$ is refractive index of material
Other Physically-Based BRDF Models

- Oren-Nayar – Diffuse term changed not specular
- Aishikhminn-Shirley – Grooves not v-shaped. Other Shapes
- Microfacet generator (Design your own microfacet)
BV BRDF Viewer

BRDF viewer (View distribution of light bounce)
BRDF Evolution

- BRDFs have evolved historically
  - 1970’s: Empirical models
    - Phong’s illumination model
  - 1980s:
    - Physically based models
    - Microfacet models (e.g. Cook Torrance model)
  - 1990’s
    - Physically-based appearance models of specific effects (materials, weathering, dust, etc)
  - Early 2000’s
    - Measurement & acquisition of static materials/lights (wood, translucence, etc)
  - Late 2000’s
    - Measurement & acquisition of time-varying BRDFs (ripening, etc)
Measuring BRDFs

Murray-Coleman and Smith Gonioreflectometer. (Copied and Modified from [Ward92]).
Measured BRDF Samples

- Mitsubishi Electric Research Lab (MERL)
  http://www.merl.com/brdf/
- Wojciech Matusik
- MIT PhD Thesis
- 100 Samples
BRDF Evolution

- BRDFs have evolved historically
- 1970’s: Empirical models
  - Phong’s illumination model
- 1980s:
  - Physically based models
  - Microfacet models (e.g. Cook Torrance model)
- 1990’s
  - Physically-based appearance models of specific effects (materials, weathering, dust, etc)
- Early 2000’s
  - Measurement & acquisition of static materials/lights (wood, translucence, etc)
- Late 2000’s
  - Measurement & acquisition of time-varying BRDFs (ripening, etc)
**Time-varying BRDF**

- **BRDF**: How different materials reflect light
- **Time varying?**: how reflectance changes over time
References