Shading Polygonal Geometry
Smooth surfaces are often approximated by polygons

Shading approaches:

1. Flat
2. Smooth (Gouraud)
3. Phong
Flat Shading

In general, l, n, and v vary from point to point on a surface. If we assume a distant viewer, v can be thought of as constant. If we assume a distant light source, l can be thought of as constant. For a flat polygon, n is constant.

If the light source or viewer is not at inf, we need heuristic for picking color – e.g., first vertex, or polygon center.
Flat shading doesn’t usually look too good. The **lateral inhibition** effect makes flat shading seem even worse.
Smooth Shading

We assign the vertex normals based on the surrounding polygon normals.

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

do the shading calculation once per vertex

We assign the vertex normals based on the surrounding polygon normals.
Interpolating Normals

- Must renormalize
Interpolating Normals

- Must renormalize
Interpolating Normals

- Must renormalize
We can interpolate attributes using barycentric coordinates

\[ \mathbf{c} = \alpha \mathbf{c}_0 + \beta \mathbf{c}_1 + \gamma \mathbf{c}_2 \]

Gouraud shading
(Gouraud, 1971)

http://jtibble.dyndns.org/graphics/eecs487/eecs487.html

Using barycentric coordinates also has the advantage that we can easily interpolate colors or other attributes from triangle vertices
Phong Shading

Phong shading requires normals to be interpolated across each polygon -- this wasn’t part of the fixed function pipeline. This can now be done in the pipeline in the fragment shader.
Comparison

- Phong interpolation looks smoother -- can see edges on the Gouraud model
- but Phong is a lot more work
- both Phong and Gouraud require vertex normals
- both Phong and Gouraud leave silhouettes
Problems with Interpolated Shading

- Polygonal silhouette
- Perspective distortion
- Orientation dependence
- Unrepresentative surface normals

[Foley, van Dam, Feiner, Hughes]
Programmable Shading
Fixed-Function Pipeline

- The application supplies geometric primitives through a graphics API such as **OpenGL** or **DirectX**
- control of pipeline operation through state variables only
Programmable Pipeline

Supply shader programs to be executed on GPU as part of pipeline

- can supply shader programs to carry out vertex processing, geometry processing, and pixel processing
Phong reflectance in vertex and pixel shaders using GLSL

```glsl
void main(void)
{
    vec4 v = gl_modelView_Matrix * gl_Vertex;
    vec3 n = normalize(gl_NormalMatrix * gl_Normal);
    vec3 l = normalize(gl_lightSource[0].position - v);
    vec3 h = normalize(l - normalize(v));

    float p = 16;
    vec4 cr = gl_FrontMaterial.diffuse;
    vec4 cl = gl_LightSource[0].diffuse;
    vec4 ca = vec4(0.2, 0.2, 0.2, 1.0);

    vec4 color;
    if (dot(h,n) > 0)
        color = cr * (ca + cl * max(0.0, dot(h,n)))
             + cl* pow(dot(h,n), p);
    else
        color = cr * (ca + cl * max(0.0, dot(h,n)));

    gl_FrontColor = color;
    gl_Position = ftransform();
}
```

Phong reflectance as a vertex shader
- vertex shaders can be used to move/animate verts
- linear interpolation of vertex lighting

```glsl
varying vec4 v;
varying vec3 n;

void main(void)
{
    vec3 l = normalize(gl_lightSource[0].position - v);
    vec3 h = normalize(l - normalize(v));

    float p = 16;
    vec4 cr = gl_FrontMaterial.diffuse;
    vec4 cl = gl_LightSource[0].diffuse;
    vec4 ca = vec4(0.2, 0.2, 0.2, 1.0);

    vec4 color;
    if (dot(h,n) > 0)
        color = cr * (ca + cl * max(0.0, dot(h,n)))
             + cl* pow(dot(h,n), p);
    else
        color = cr * (ca + cl * max(0.0, dot(h,n)));

    gl_FragColor = color;
}
```

[Shirley and Marschner]
Programmable shader examples from NVIDIA and ATI
Computing Normal Vectors
Plane Normals

\[ \mathbf{v} = (\mathbf{p}_2 - \mathbf{p}_0) \times (\mathbf{p}_1 - \mathbf{p}_0) \]

\[ \mathbf{n} = \frac{\mathbf{v}}{||\mathbf{v}||} \]
Implicit function normals

\[ f(p) = 0 \]

\[ \nabla f(p) \]

\[ \nabla f = \left( \begin{array}{c} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{array} \right) \]

sphere

\[ p \cdot p - r^2 = 0 \]

plane

\[ n \cdot (p - p_0) = 0 \]
Parametric form

\[ p(u, v) = \begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix} \]

tangent vectors

\[
\begin{align*}
\frac{\partial p}{\partial u} & \quad \frac{\partial p}{\partial v}
\end{align*}
\]

normal

\[
\frac{\frac{\partial p}{\partial u} \times \frac{\partial p}{\partial v}}{\left| \frac{\partial p}{\partial u} \times \frac{\partial p}{\partial v} \right|}
\]
<table>
<thead>
<tr>
<th>Statistics</th>
<th>Status Distribution</th>
<th>Grade Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Null</td>
<td>greater than 100</td>
</tr>
<tr>
<td>31</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>In Progress</td>
<td>90 - 100</td>
</tr>
<tr>
<td>14.75</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>Needs Grading</td>
<td>80 - 89</td>
</tr>
<tr>
<td>37.00</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Range</td>
<td>Exempt</td>
<td>70 - 79</td>
</tr>
<tr>
<td>22.25</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>60 - 69</td>
</tr>
<tr>
<td>23.73</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>Median</td>
<td></td>
<td>50 - 59</td>
</tr>
<tr>
<td>25.00</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td></td>
<td>40 - 49</td>
</tr>
<tr>
<td>5.70</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Variance</td>
<td></td>
<td>30 - 39</td>
</tr>
<tr>
<td>32.49</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20 - 29</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 - 19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 - 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>less than 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>