9 Shading

Goal: To use the lighting and reflectance model to shade facets of a polygonal mesh — that is, to assign intensities to pixels to give the impression of opaque surfaces rather than wireframes.

Assume we're given the following:

- $\bar{e}^w$ - center of projection in world coordinates
- $\bar{l}^w$ - point light source location
- I_a, I_d - intensities of ambient and directional light sources
- r_a, r_d, r_s - coefficients for ambient, diffuse, and specular reflections
- α - exponent to control width of highlights

9.1 Flat Shading

With **flat shading**, each triangle of a mesh is filled with a single color.

For a triangle with counterclockwise vertices $\bar{p}_1$, $\bar{p}_2$, and $\bar{p}_3$, as seen from the outside, let the midpoint be $\bar{p} = \frac{1}{3}(\bar{p}_1 + \bar{p}_2 + \bar{p}_3)$ with normal $\vec{n} = \frac{(\bar{p}_2 - \bar{p}_1) \times (\bar{p}_3 - \bar{p}_1)}{\|(\bar{p}_2 - \bar{p}_1) \times (\bar{p}_3 - \bar{p}_1)\|}$. Then we may find the intensity at $\bar{p}$ using the Phong model and fill the polygon with that:

$$E = \tilde{I}_a r_a + r_d \tilde{I}_d \max(0, \vec{n} \cdot \vec{s}) + r_s \tilde{I}_d \max(0, \vec{r} \cdot \vec{c})^\alpha, \quad (1)$$

where $\vec{s} = \frac{\bar{l}^w - \bar{p}}{\|\bar{l}^w - \bar{p}\|}$, $\vec{c} = \frac{\bar{e}^w - \bar{p}}{\|\bar{e}^w - \bar{p}\|}$, and $\vec{r} = -\vec{s} + 2(\vec{s} \cdot \vec{n})\vec{n}$.

Flat shading is a simple approach to filling polygons with color, but can be inaccurate for smooth surfaces, and shiny surfaces. For smooth surfaces—which are often tesselated and represented as polyhedra, using flat shading can lead to a very strong faceting effect. In other words, the surface looks very much like a polyhedron, rather than the smooth surface it's supposed to be. This is because our visual system is very sensitive to variations in shading, and so using flat shading makes faces really look flat.

9.2 Interpolative Shading

The idea of **interpolative shading** is to avoid computing the full lighting equation at each pixel by interpolating quantites at the vertices of the faces.

Given vertices $\bar{p}_1$, $\bar{p}_2$, and $\bar{p}_3$, we need to compute the normals for each vertex, compute the radiances for each vertex, project onto the window in device coordinates, and fill the polygon using scan conversion.

There are two methods used for interpolative shading:

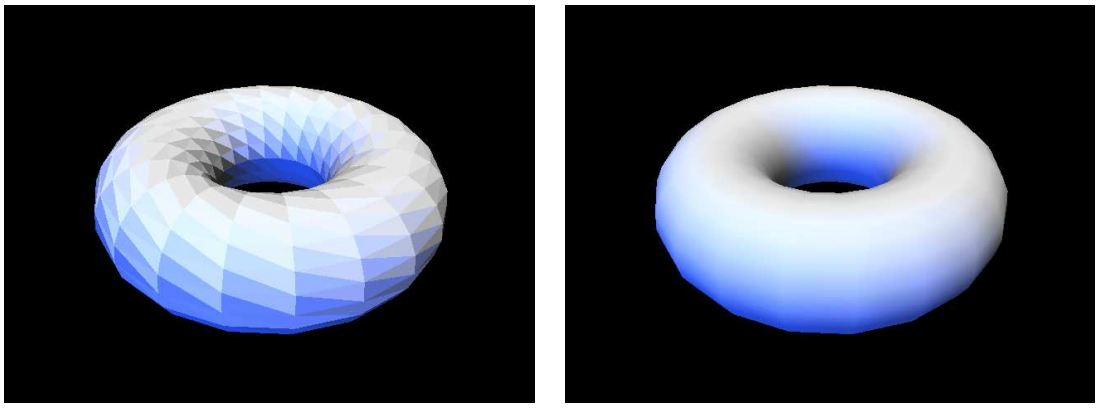
Gouraud Shading The radiance values are computed at the vertices and then linearly interpolated within each triangle.

Phong shading The normal values at each vertex are linearly interpolated within each triangle, and the radiance is computed at each pixel.

Gouraud shading is more efficient, but Phong shading is more accurate. When will Gouraud shading give worse results?

9.3 Shading in OpenGL

OpenGL only directly supports Gouraud shading or flat shading. Gouraud is enabled by default, computing vertex colors, and interpolating colors across triangle faces. Flat shading can be enabled with `glShadeModel(GL_FLAT)`. This renders an entire face with the color of a single vertex, giving a faceted appearance.



Left: Flat shading of a triangle mesh in OpenGL. *Right:* Gouraud shading. Note that the mesh appears smooth, although the coarseness of the geometry is visible at the silhouettes of the mesh.

With *pixel shaders* on programmable graphics hardware, it is possible to achieve Phong shading by using a small program to compute the illumination at each pixel with interpolated normals. It is even possible to use a *normal map* to assign arbitrary normals within faces, with a pixel shader using these normals to compute the illumination.

