Gouraud vs. Phong interpolation

Now we know how to compute the color at a point on a surface using the Blinn-Phong lighting model.

Does graphics hardware do this calculation at every point? Not by default...

Smooth surfaces are often approximated by polygonal facets, because:

• Graphics hardware generally wants polygons (esp. triangles).
• Sometimes it easier to write ray-surface intersection algorithms for polygonal models.

How do we compute the shading for such a surface?

Faceted shading

Assume each face has a constant normal:

For a distant viewer and a distant light source and constant material properties over the surface, how will the color of each triangle vary?

Result: faceted, not smooth, appearance.
Faceted shading (cont’d)

Gouraud interpolation

To get a smoother result that is easily performed in hardware, we can do Gouraud interpolation.

Here’s how it works:

1. Compute normals at the vertices.
2. Shade only the vertices.
3. Interpolate the resulting vertex colors.

Rasterization with color

Recall that the z-buffer works by interpolating z-values across a triangle that has been projected into image space, a process called rasterization.

During rasterization, colors can be smeared across a triangle as well:

Faced shading vs. Gouraud interpolation
Gouraud interpolation artifacts

Gouraud interpolation has significant limitations.

1. If the polygonal approximation is too coarse, we can miss specular highlights.

2. We will encounter **Mach banding** (derivative discontinuity enhanced by human eye).

This is what graphics hardware does by default.

A substantial improvement is to do...

Phong interpolation

To get an even smoother result with fewer artifacts, we can perform **Phong interpolation**.

Here’s how it works:

1. Compute normals at the vertices.
2. Interpolate normals and normalize.
3. Shade using the interpolated normals.

**Default pipeline: Gouraud interpolation**

**Default vertex processing:**
- \( L \leftarrow \text{determine lighting direction} \)
- \( V \leftarrow \text{determine viewing direction} \)
- \( N \leftarrow \text{normalize}(n_i) \)
- \( c_{\text{phong}} \leftarrow \text{shade with } L, V, N, k_d, k_s, n_i \)
- attach \( c_{\text{phong}} \) to vertex as “varying” \( v_j \leftarrow \text{project } v \text{ to image} \)

\( v_j, v_j', v_j'' \rightarrow \text{triangle} \)

**Default fragment processing:**
- color \( \leftarrow c_{\text{phong}} \)
**Programmable pipeline:**

**Phong-interpolated normals!**

**Vertex shader:**
- attach \( n_e \) to vertex as "varying"
- attach \( v_e \) to vertex as "varying"
- \( v_i \) ← project \( v \) to image

\[ v_i^1, v_i^2, v_i^3 \rightarrow \text{triangle} \]

**Fragment shader:**
- \( L \leftarrow \) determine lighting direction
- \( V \leftarrow \) determine viewing direction
- \( N \leftarrow \) normalize(\( n_e \))
- color ← shade with \( L, V, N, k_s^p, k_t^p, n_e^p \)

**Surface normals**

How can we compute the normal to a surface at a given point?

**Tangent vectors and tangent planes**

**Normals on a surface of revolution**
Summary

You should understand the equation for the Blinn-Phong lighting model described in the “Iteration Four” slide:

• What is the physical meaning of each variable?
• How are the terms computed?
• What effect does each term contribute to the image?
• What does varying the parameters do?

You should also understand the differences between faceted, Gouraud, and Phong interpolated shading.

And you should understand how to compute the normal to a surface of revolution.