

Shading (Part 2)

Brian Curless
CSEP 557
Winter 2013

1

Reading

Required:

- Angel chapter 5.

Optional:

- OpenGL red book, chapter 5.

2

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:

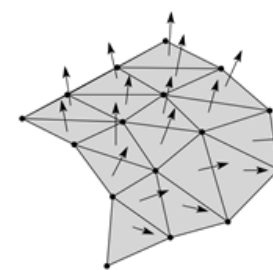
- Graphics hardware generally wants polygons (esp. triangles).
- Sometimes it's easier to write ray-surface intersection algorithms for polygonal models.

How do we compute the shading for such a surface?

3

Faceted shading

Assume each face has a constant normal:



$$= \dots \frac{(N \cdot L) \cdot (N \cdot H)}{\|L + V\|}$$

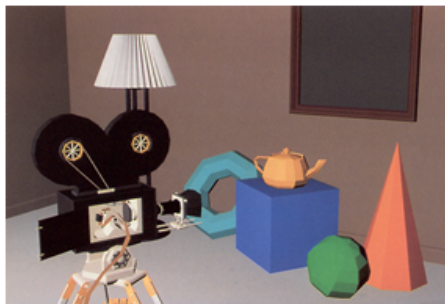
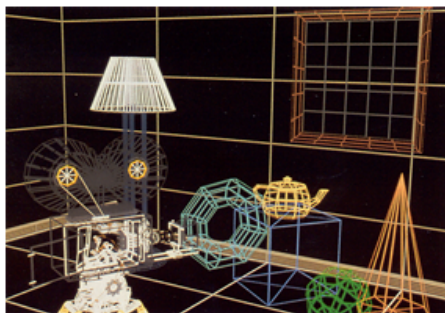
Handwritten notes: $L \rightarrow$, $V \rightarrow$, $H \rightarrow$ (normal vector), and $H = \frac{L+V}{\|L+V\|}$

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.

4

Faceted shading (cont'd)



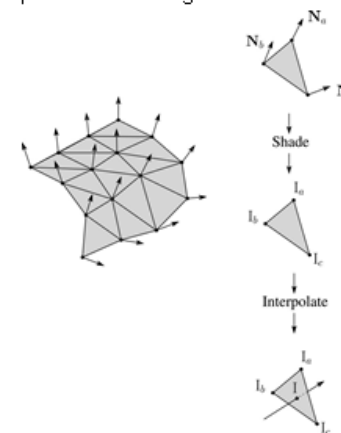
5 [Williams and Siegel 1990]

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.

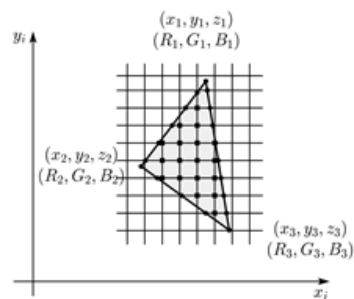


6

Rasterization with color

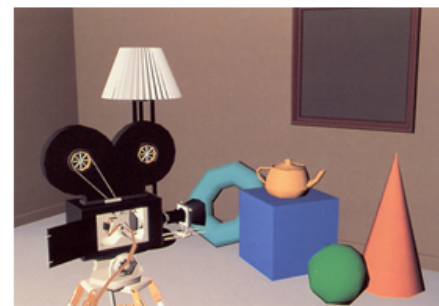
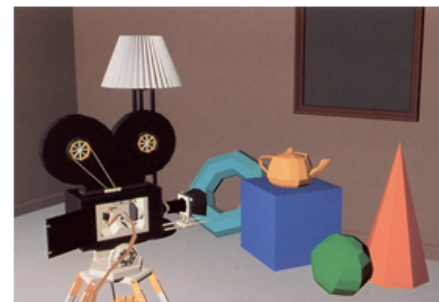
Recall that the z-buffer works by interpolating z-values across a triangle that has been projected into image space, a process called rasterization.

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



7

Faceted shading vs. Gouraud interpolation



8

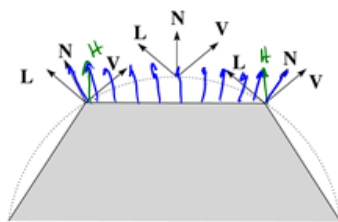
[Williams and Siegel 1990]

Gouraud interpolation artifacts

0/1/1

Gouraud interpolation has significant limitations.

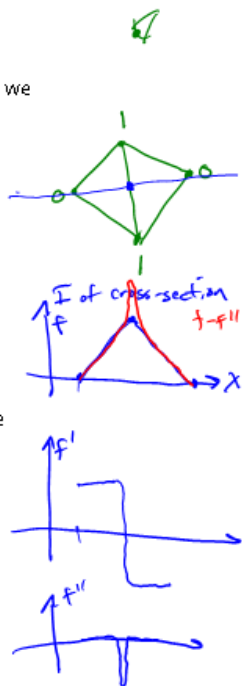
1. If the polygonal approximation is too coarse, we can miss specular highlights.



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...



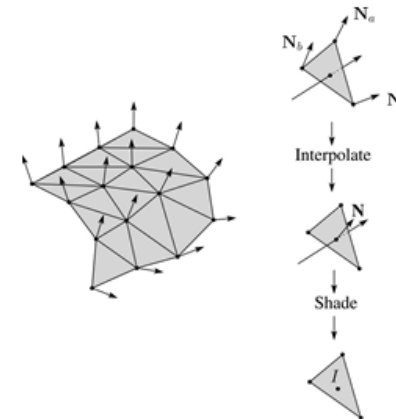
9

Phong interpolation

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

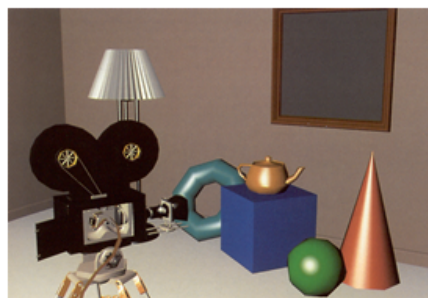
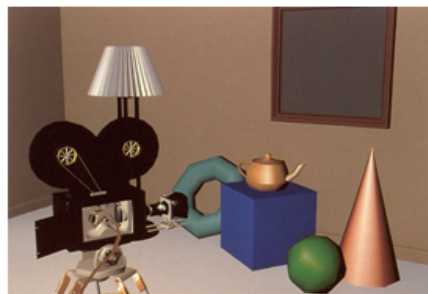
Here's how it works:

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



10

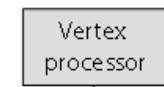
Gouraud vs. Phong interpolation



11

[Williams and Siegel 1990]

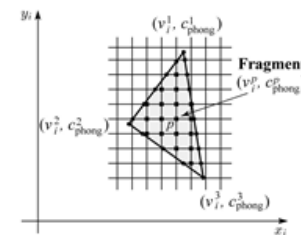
Default pipeline: Gouraud interpolation



Default vertex processing:

$L \leftarrow$ determine lighting direction
 $V \leftarrow$ determine viewing direction
 $N \leftarrow$ normalize(n_s)
 $c_{\text{phong}} \leftarrow$ shade with L, V, N, k_d, k_s, n_s
 attach c_{phong} to vertex as "varying"
 $v_i \leftarrow$ project v to image

$v_1^i, v_2^i, v_3^i \rightarrow$ triangle



Default fragment processing:

color $\leftarrow c_{\text{phong}}^i$

12

Programmable pipeline: Phong-interpolated normals!

Vertex processor

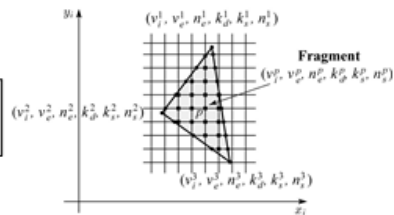
Primitive assembler

Rasterizer

Fragment processor

Vertex shader:
attach n_v to vertex as "varying"
attach v_o to vertex as "varying"
 $v_f \leftarrow$ project v to image

$v_1^1, v_1^2, v_1^3 \rightarrow$ triangle

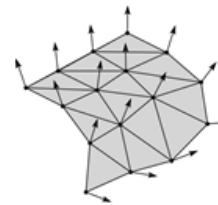


Fragment shader:
 $L \leftarrow$ determine lighting direction
 $V \leftarrow$ determine viewing direction
 $N \leftarrow$ normalize(n_f^i)
color \leftarrow shade with $L, V, N, k_s^i, k_d^i, n_s^i$

13

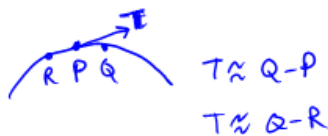
Surface normals

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



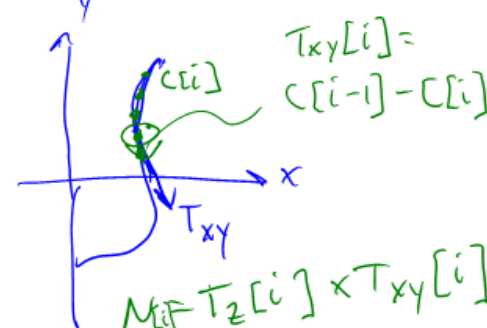
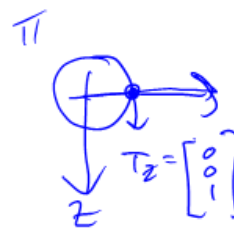
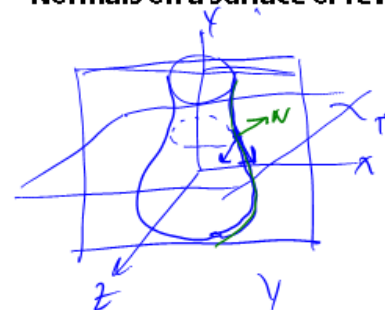
14

Tangent vectors and tangent planes



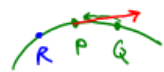
15

Normals on a surface of revolution



This is normal to the surface for points in xy -plane. Can rotate to get others:
 $N_{[i,j]} = R_y(\theta_j) N_{[i]}$

16



Finite diff.

$$T \approx Q - P$$

$$T = \lim_{Q \rightarrow P} \frac{Q - P}{\|Q - P\|}$$

Central difference

$$T \approx Q - R$$

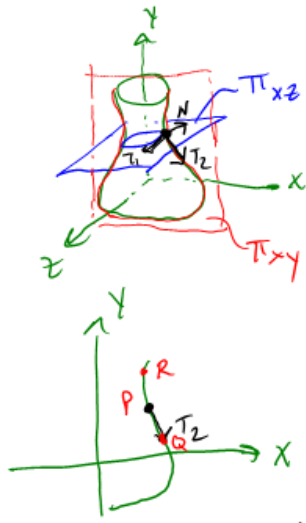
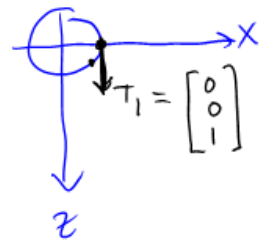
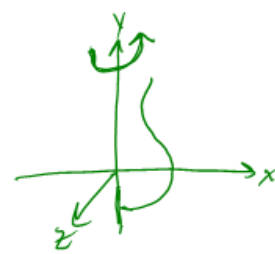
$$T = \lim_{Q, R \rightarrow P} \frac{Q - R}{\|Q - R\|}$$



$$T_1 \approx Q_1 - P$$

$$T_2 \approx Q_2 - P$$

$$N = \frac{T_2 \times T_1}{\|T_2 \times T_1\|}$$



Can rotate when creating points on surface of rev.

$$N = \frac{T_1 \times T_2}{\|T_1 \times T_2\|}$$

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.