

CIS 736

Computer Graphics

Advanced CG Topics 3 of 8: Advanced Lighting Models

William H. Hsu

Department of Computing and Information Sciences, KSU

KSOL course pages: <http://snipurl.com/1y5gc>

Course web site: <http://www.kddresearch.org/Courses/CIS736>

Instructor home page: <http://www.cis.ksu.edu/~bhsu>

Readings:

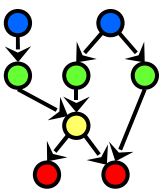
Handout 2: Lighting and Shading

Sections 2.6.2, 20.2, Eberly 2^e – see <http://snurl.com/1ye72>

NeHe tutorials 6 (basics), 11 (flag), 15 & 17 (fonts) – <http://nehe.gamedev.net>

More NeHe texturing tutorials: 35 (rendering), 42 (coloring)

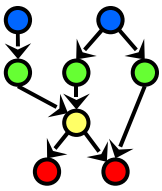




Online Recorded Lectures for CIS 736 *Computer Graphics*

- **Project Topics for CIS 736**
- **Advanced Topics in Computer Graphics (8)**
 - * 1. Filters for Texturing – first month
 - * 2. More Mappings – second month
 - * 3. Advanced Lighting Models – second month
 - * 4. Advanced Ray-Tracing – first month
 - * 5. Advanced Ray-Tracing, concluded – second month
 - * 6. Global Illumination: Photon Maps (Radiosity) – third month
 - * 7. More on Scientific, Data, Info Visualization – third month
 - * 8. Terrain – first month
- **Recommended Background Reading for CIS 736**
- **Shared Lectures with CIS 636 (*Computer Graphics*)**
 - * Regular in-class lectures (35) and labs (7)
 - * Guidelines for paper reviews – week of Mon 25 Feb 2008
 - * Preparing term project presentations and demos for graphics – April

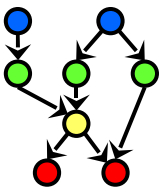




Overview

- Last time, we covered light-matter interaction.
- Now, apply it to rendering.
- Outline:
 - * Lighting and shading.
 - * Lighting models.
 - * Shading methods.



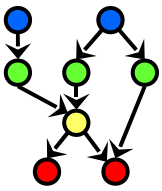


Those Were the Days

- **(Or: how not to motivate a 21st century computer graphics paper.)**
 - ⇒ **“In trying to improve the quality of the synthetic images, we do not expect to be able to display the object exactly as it would appear in reality, with texture, overcast shadows, etc. We hope only to display an image that approximates the real object closely enough to provide a certain degree of realism.”**

– Bui Tuong Phong, 1975

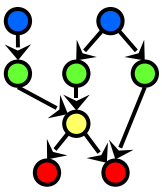




Lighting vs. Shading

- Commonly misused terms.
- What's the difference?

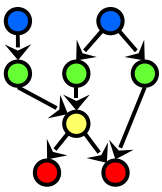




Lighting vs. Shading

- Commonly misused terms.
- What's the difference?
- Lighting designates the interaction between materials and light sources, as in last lecture.
- Shading is the process of determining the color of a pixel.
 - * Usually determined by lighting.
 - * Could use other methods: random color, NPR, etc.

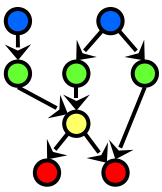




Lighting Models

- **Will discuss three**
 - * **Lambert**
 - ⇒ Purely diffuse surfaces
 - * **Phong**
 - ⇒ Adds perceptually-based specular term
 - * **Torrance-Sparrow**
 - ⇒ Provides physical approximation

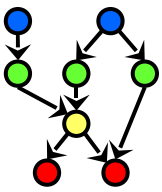




Lambert Lighting Model

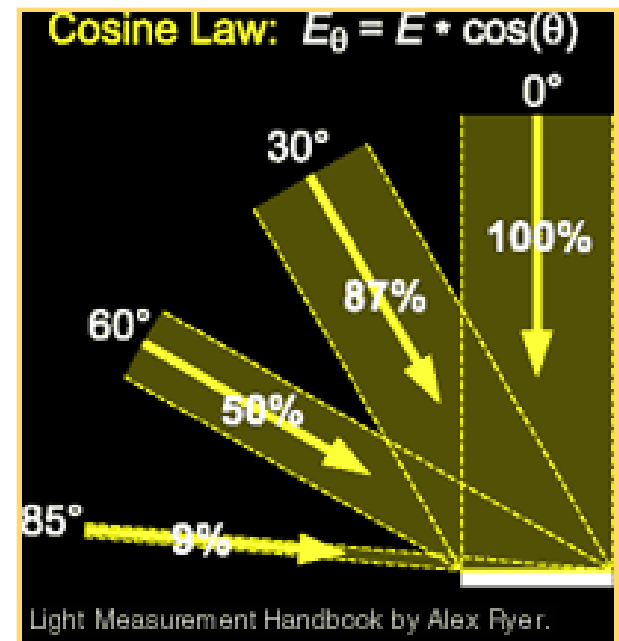
- **Sometimes mistakenly attributed to Gouraud.**
 - * Gouraud didn't introduce a new lighting model, just a shading method.
 - * Used approximations from Warnock and Romney.
 - ⇒ Both based on Lambert's cosine law.

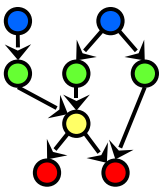




Lambert's Cosine Law

- The reflected luminous intensity in any direction from a perfectly diffusing surface varies as the cosine of the angle between the direction of incident light and the normal vector of the surface.
- Intuitively: cross-sectional area of the “beam” intersecting an element of surface area is smaller for greater angles with the normal.





Lambert's Cosine Law

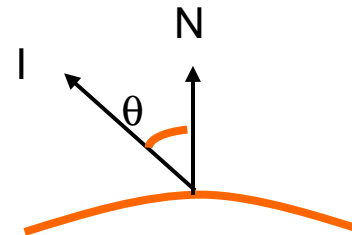
- Ideally diffuse surfaces obey cosine law.

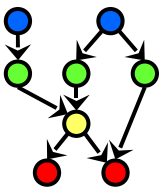
★ Often called *Lambertian* surfaces.

- $$I_d = k_d I_{incident} \cos \theta$$
$$= k_d I_{incident} (N \cdot L).$$

★ k_d is the diffuse reflectance of the material.

⇒ Wavelength dependent, so usually specified as a color.

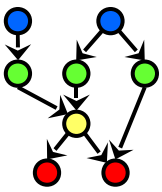




Phong Lighting Model

- Phong adds specular highlights.
- His original formula for the specular term:
 - * $W(i)[\cos s]^n$
 - ⇒ s is the angle between the view and specular reflection directions.
 - ⇒ “ $W(i)$ is a function which gives the ratio of the specular reflected light and the incident light as a function of the the incident angle i .”
 - ◆ Ranges from 10 to 80 percent.
 - ⇒ “ n is a power which models the specular reflected light for each material.”
 - ◆ Ranges from 1 to 10.





Phong Lighting Model

- **More recent formulations are slightly different.**

- ★ Replace $W(i)$ with a constant k_s , independent of the incident direction.

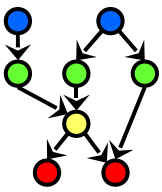
- ⇒ What do we lose when we do this?

- ★ $I_s = k_s I_{incident} \cos^n \phi$
 $= k_s I_{incident} (V \cdot R)^n.$

- ⇒ V is the view direction.

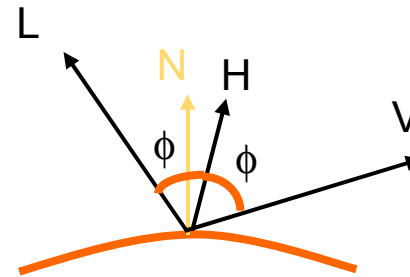
- ⇒ R is the specular reflection direction.

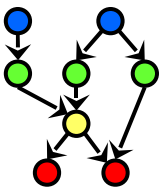




Blinn-Phong Model

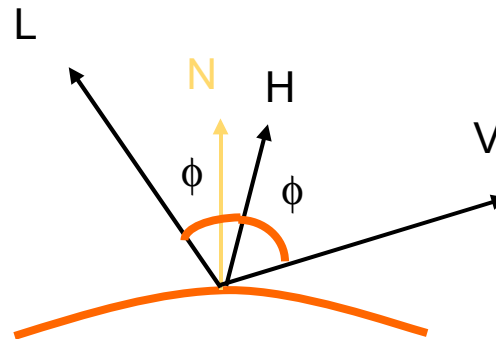
- Popular variation of Phong model.
- Uses the *halfway vector*, H .
- $I_s = k_s I_{incident} (N \cdot H)^n$.
 - * $H = (L + V) / |L + V|$
- What are the advantages?

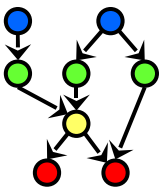




Blinn-Phong Model

- Popular variation of Phong model.
- Uses the *halfway vector*, H .
- $I_s = k_s I_{incident} (N \cdot H)^n$
 - * $H = (L + V) / |L + V|$
- Faster to compute than reflection vector.
- Still view-dependent since H depends on V .

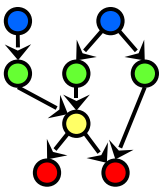




Blinn-Phong Highlights

- Does using $N \cdot H$ vs. $R \cdot V$ affect highlights?
 - * Yes, the highlights “spread”.
 - * Why?
- Is this bad?

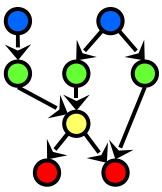




Blinn-Phong Highlights

- Does using $N \cdot H$ vs. $R \cdot V$ affect highlights?
 - * Yes, the highlights “spread”.
 - * Why?
- Is this bad?
 - * Not really, for two reasons.
 - ⇒ Can always just adjust the exponent.
 - ⇒ Phong and Blinn-Phong are not physically based, so it doesn't really matter!

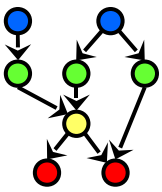




Torrance-Sparrow Model

- Introduced by Torrance and Sparrow in 1967 as a theoretical model.
- Introduced to CG community by Blinn in 1977.
 - ★ same paper as “Halfway Vector” (Blinn-Phong).
- Attempts to provide a more physical model for specular reflections from real surfaces.
 - ★ Points out that intensity of specular highlights is dependent on the incident direction relative to normal.
 - ★ Phong attempted to model this with $w(i)$ factor?

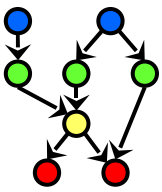




Torrance-Sparrow Model

- Back to micro facets.
- Assumptions:
 - * Diffuse component comes from multiple reflections between facets and from internal scattering.
 - * Specular component of surface comes from facets oriented in direction of H .



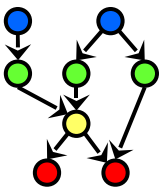


Torrance-Sparrow Model

- $I_s = DGF / (N \cdot V)$

- * D is the distribution function of the micro facet directions on the surface.
- * G is the amount that facets shadow and mask each other.
- * F is the Fresnel reflection law.

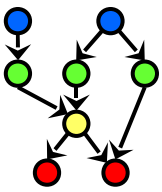




D: Micro Facet Distribution

- T-S used simple Gaussian distribution:
 - * $D = e^{-(\alpha \sigma)^2}$
 - * α = deviation angle from halfway vector, H .
 - * σ = standard deviation.
 - ⇒ Large values = dull, small values = shiny

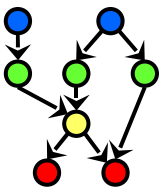




Denominator

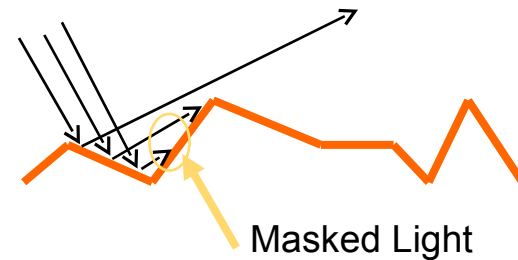
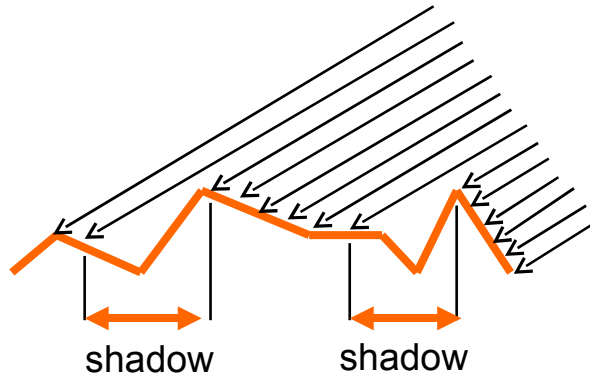
- Intensity proportional to number of facets in H direction.
 - * So, must account for fact that observer sees more surface area when surface is tilted.
 - ⇒ Change in area proportional to cosine of tilt angle.
 - * Hence, $N \cdot V$ in denominator.





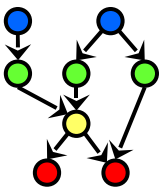
G: Geometrical Attenuation Factor

- Remember micro facet shadowing and masking?



- Blinn derives this factor for symmetrical v-shaped groove facets (see paper)

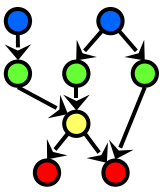




F: Fresnel Reflection

- Fraction of light incident on a facet that is actually reflected rather than absorbed.
- Function of angle of incidence and index of refraction.
 - * $F(\phi, \eta)$.
 - * For metals (large η), $F(\phi, \eta)$ nearly constant at 1.
 - * For non-metals (small η), $F(\phi, \eta)$ has exponential appearance. Near zero for $\phi = 0$, to 1 at $\phi = \pi / 2$.

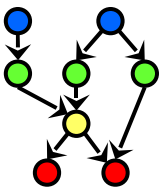




Shading

- **Have seen some methods for computing lighting.**
 - * **Given normal, light direction, material properties.**
 - * **Non-diffuse models need view direction.**
- **Now explore methods of applying that lighting (or other color) to pixels of rasterized surface.**

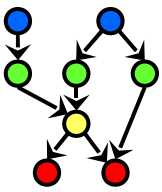




Types of Shading

- In polygonal rendering, three main types
 - * Flat shading
 - * Gouraud shading
 - * Phong shading
- Roughly correspond to
 - * Per-polygon shading
 - * Per-vertex shading
 - * Per-pixel shading

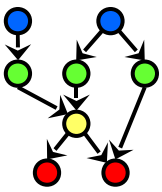




Flat Shading

- **Fast and simple.**
- **Compute the color of a polygon.**
- **Use that color on every pixel of the polygon.**

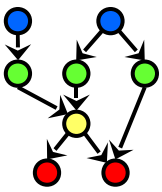




Gouraud Shading

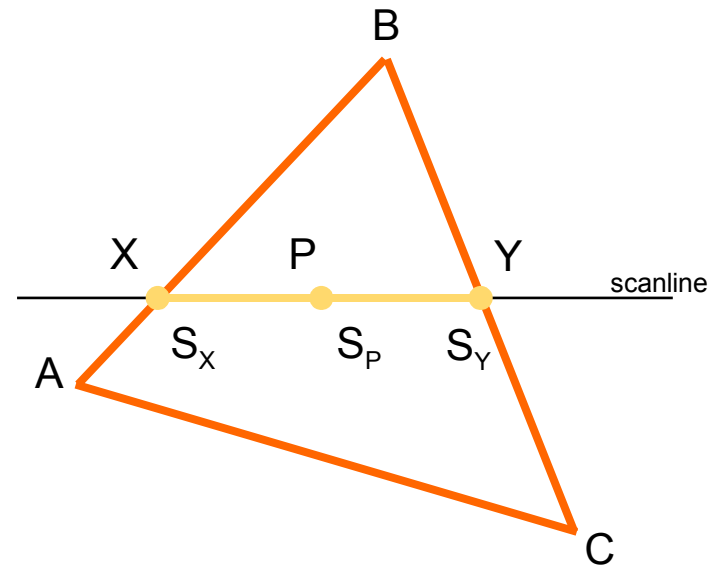
- **Still pretty fast and simple.**
- **Gives better sense of form than flat shading for many applications.**
- **Basic Idea:**
 - * **Compute color at each vertex.**
 - * **Bi-linearly interpolate color for each interior pixel.**

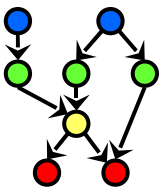




Gouraud Shading

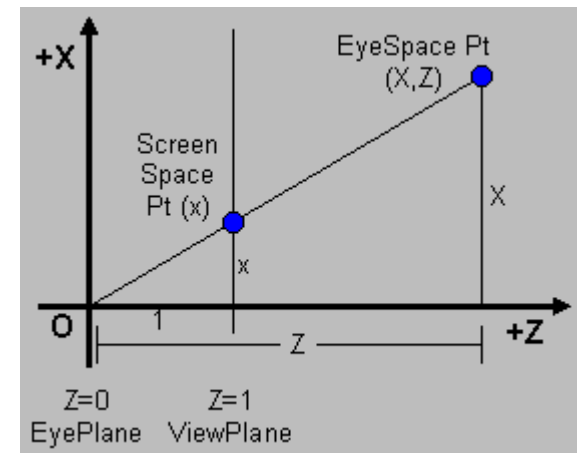
- Compute S_A , S_B , S_C for triangle ABC.
 - * S_i = shade of point i.
- For a scanline XY, compute S_X , S_Y by linearly interpolating (lerping).
 - * e.g. $t_{AB} = |AX| / |AB|$.
 - * $S_X = t_{AB} * S_A + (1-t_{AB}) * S_B$
- Compute S_P
 - * By lerping between S_X and S_Y .

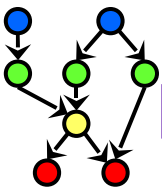




Linear Interpolation Concerns

- **Perspective projection complicates linear interpolation.**
 - ★ Relationship between screen space distance and eye space distance is nonlinear.
 - ★ Therefore, relationship between interpolation in the two spaces is also nonlinear.
 - ★ Thus, screen space linear interpolation of colors (and texture coordinates) results in incorrect values.
- **Note: potential homework / test problem!**

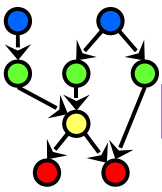




Perspectively-correct Interpolation [1]

- **Could interpolate in eye space, then project every interpolated point.**
 - ★ **Way too much work!**
- **Can we interpolate in screen space and correct for perspective nonlinearity?**
 - ★ **Yes!**

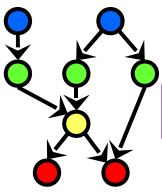




Perspectively-correct Interpolation [2]

- For a detailed derivation, see:
 - ⇒ <http://www.cs.unc.edu/~hoff/techrep/persp/persp.html>
- Here, we skip to the punch line:
 - ★ Given two eye space points, E_1 and E_2 .
 - ⇒ Can lerp in eye space: $E(T) = E_1(1-T) + E_2(T)$.
 - ⇒ T is eye space parameter, t is screen space parameter.
 - ★ To see relationship, express in terms of screen space t :
 - ⇒ $E(t) = [(E_1/Z_1)*(1-t) + (E_2/Z_2)*t] / [(1/Z_1)*(1-t) + (1/Z_2)*t]$

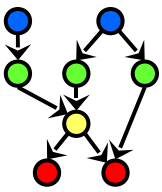




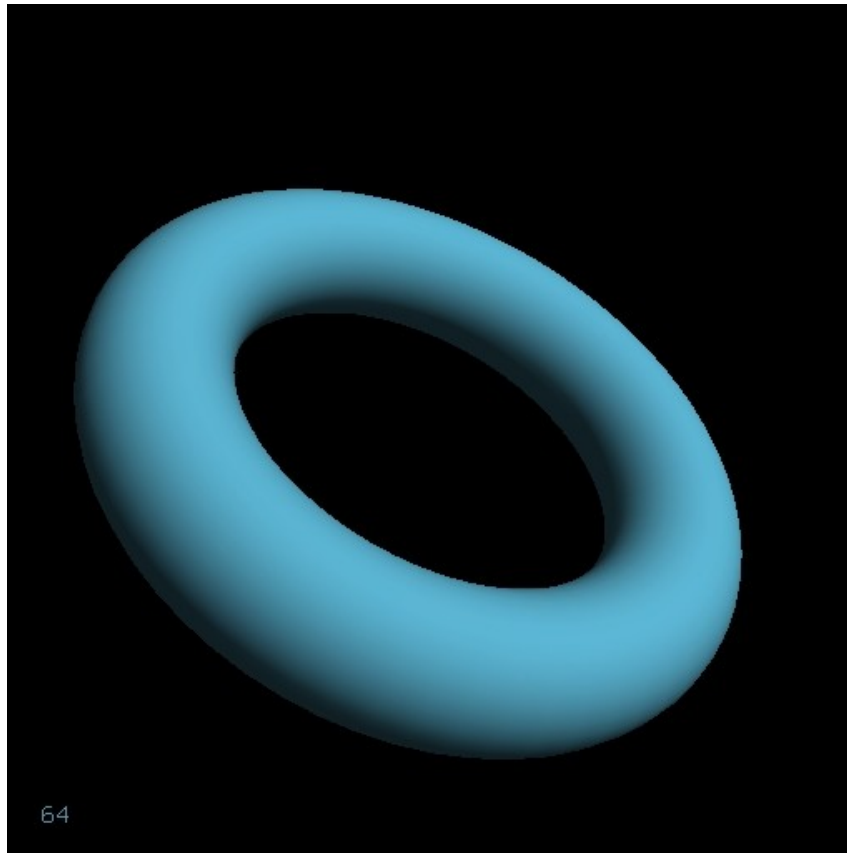
Perspectively-correct Interpolation [3]

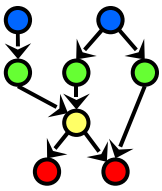
- $E(t) = [(E_1/Z_1)*(1-t) + (E_2/Z_2)*t] / [(1/Z_1)*(1-t) + (1/Z_2)*t]$
 - ★ $E_1/Z_1, E_2/Z_2$ are projected points.
 - ★ Because Z_1, Z_2 are depths corresponding to E_1, E_2 .
- Looking closely, can see that interpolation along an eye space edge = interpolation along projected edge in screen space divided by the interpolation of $1/Z$.





Gouraud Example

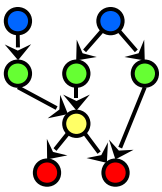




Mach Bands

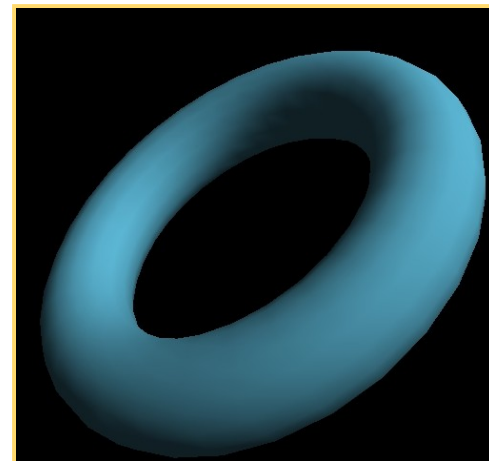
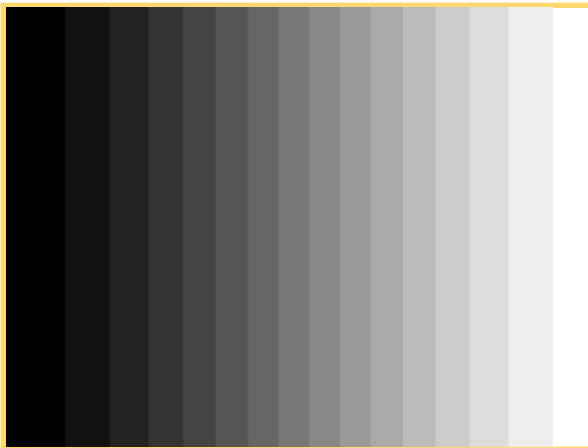
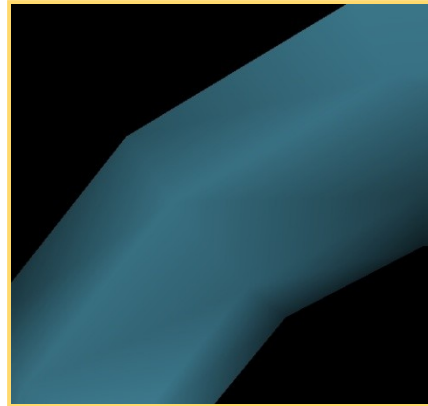
- Gouraud discusses “artifact” of lerping.
- Mach bands:
 - ✦ Caused by interaction of neighboring retinal neurons.
 - ✦ Acts as a sort of high-pass filter, accentuating discontinuities in first derivative.
 - ✦ Linear interpolation causes first deriv. Discontinuities at polygon edges.

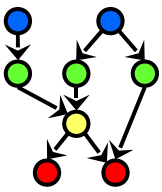




Mach Bands

- Simple examples

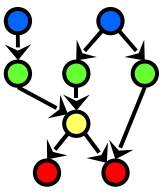




Improvements

- **Gouraud suggests higher-order interpolation would alleviate mach banding.**
 - * **But stresses the performance cost.**
 - * **Probably not worth it.**
- **Phong shading helps the problem.**

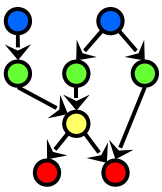




Phong Shading

- Phong shading is *not* what current graphics hardware implements.
 - * APIs (D3D, OGL) employ Blinn-Phong *lighting* and Gouraud *shading*.
- Phong shading applies lighting computation *per-pixel*.
 - * Uses linear interpolation of normal vectors, rather than colors.

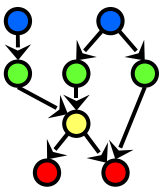




Phong Shading

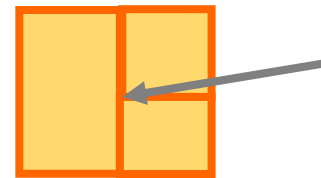
- **Interpolation just as with colors in Gouraud shading.**
 - * Interpolate scan line endpoint normals N_a, N_b from endpoints of intercepted edges.
 - * Interpolate normal N_p at each pixel from N_a, N_b .
 - * Normalize N_p .
 - ⇒ (Interpolation of unit vectors does not preserve length).
 - * Back-transform N_p to eye space, compute lighting.

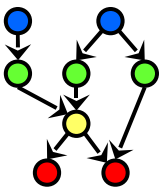




Phong Shading

- **Results are much improved over Gouraud.**
 - ★ **Harder to tell low- from high-polygon models.**
 - ★ **Still some indicators and problems:**
 - ⇒ **Silhouette still has a low tessellation.**
 - ⇒ **Shared vs. Unshared vertices.**
 - ⇒ **Mach banding.**
 - ◆ **Yep, can still get first derivative discontinuities.**



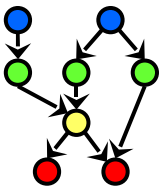


Other Types of Per-pixel Shading [1]

- **Ray tracing.**

- * **Doesn't use Gouraud or Phong shading.**
- * **Each pixel uses own ray to determine color.**
 - ⇒ **Can apply arbitrary lighting model.**
 - ⇒ **Classical (Whitted) ray tracing uses Phong model.**
- * **Since ray tracing determines colors based on intersections, don't have to use polygonal geometry.**
 - ⇒ **Thus, can potentially use exact normals, rather than interpolation.**

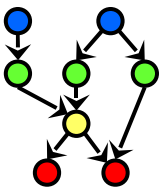




References

- **Gouraud, Phong, Blinn papers**
 - * Available in *Seminal Graphics*, ACM press.
- **Glassner, *Principles of Digital Image Synthesis*, volume two.**
 - * Highly detailed and low level.
- **Möller and Haines, *Real-Time Rendering*.**
 - * A great book, with the best bibliography you can find.

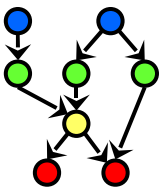




References

- Rogers, *Procedural Elements for Computer Graphics*.
 - * One of author's favorites.
- Foley, van dam, *et al. Computer Graphics, Principles and Practice*.
 - * Not the best treatment, but it covers everything.



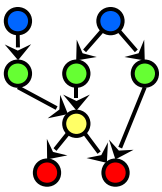


Summary

- **Textures**
 - * **Definitions**
 - * **Design principles**
- **Texture Pipeline**
 - * **Part of polygons-to-pixels**
 - * **Uses same spaces (coordinate systems) and more**
- **Using Simple Intermediate Surfaces (Cylinder, Sphere, Plane, Box)**
- **Procedural Textures**
 - * **Simple regular patterns (e.g., checkerboard)**
 - * **Other textures: terrain, clouds, etc.**
- **Perlin Noise Models**
 - * **Interpolation approach**
 - * **Turbulence for material properties**
- **Anisotropic Filtering**
- **References:**

Gröller & Jeschke (2002), Isenberg (2005), Jacobs (2007)

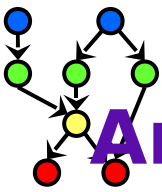




Terminology

- **Texture**
- **Coordinate Systems (Spaces)**
 - * **Model / Object**: 3-D (x, y, z)
 - * **World / Scene**: 3-D (x, y, z)
 - * **Camera / Eye**: 3-D (u, v, n)
 - * **Window / Screen**: 2-D (u, v)
 - * **Texture**: 1-D, 2-D, or 3-D; (s, t) for 2-D
- **Texture Pipeline**
- **Perlin Noise**
 - * **Interpolation**
 - * **Turbulence for material properties**
- **Anisotropic Filtering**
- **References: Gröller & Jeschke (2002), Isenberg (2005), Jacobs (2007)**





Next:

Animation (Class), Ray Tracing (736)

- **Vicki Shreiner: Animation and Depth Buffering**
 - * Double buffering
 - * Lights: positioning
 - * Illumination: light models, attenuation
 - * Material properties
 - * Animation basics in OpenGL
- **Vicki Schreiner: Imaging and Raster Primitives**
- **Ed Angel: Texture Mapping**
- **Dave Shreiner: Advanced Topics**
 - * Display lists and vertex arrays
 - * Accumulation buffer
 - * Fog
 - * Stencil buffering
 - * Fragment programs (to be concluded in Tutorial 3)

