

CS770/870 Spring 2017

Color and Shading

Related material
Cunningham: Ch 5
Hill and Kelley: Ch. 8
Angel 5e: 6.1-6.8
Angel 6e: 5.1-5.5

02/07/17

CS770/870 Spring 2017 Bergeron

Preview

- Making the scene more “realistic”
- Color models
 - representing the color of materials
- Lighting models
 - representing interaction of light with material color
- Shading models
 - determining the color of a pixel
 - based on lighting models (usually)
 - mesh shading
 - Gouraud shading
 - Phong shading

CS770/870 Spring 2017 Bergeron

2

Color Models

- We say that objects have color, but it's not quite right
 - **light** has color
 - objects have material properties that interact with the light that falls on them
 - some of the light that falls on an object is *reflected*,
 - some is *absorbed* and
 - some might be *transmitted* through it
 - we say that an object is green if it reflects only the green wavelength of light and absorbs all others
- To model color, we need to model light

CS770/870 Spring 2017 Bergeron

3

Achromatic Light

- Achromatic light is gray scale light
 - model only the *luminance* or *intensity* (physics terms) or *brightness* (a psychological term meaning *perceived intensity*)
 - most important (perceptual) component of light
- Modeling intensity/brightness
 - Assume human perceptual range can be represented as a value between 0.0 and 1.0
 - Display device has 256 values for intensity
 - What should the mapping be?

CS770/870 Spring 2017 Bergeron

4

Modeling Intensity

- Map brightness perception (0.0, 1.0) to hardware intensity (0,255)
 - linear mapping is easy and seems intuitive
 - but it's not right
 - human perception is very non-linear:
 - In 1834 Ernst Weber showed that $\Delta I/I \approx \text{constant}$, where ΔI is a noticeable intensity difference (*just noticeable difference*, or JND) and I is the intensity
 - This breaks down at very low and very high intensities, but
 - logarithmic mapping is much closer to our brightness perception scale

Logarithmic Intensity Mapping

- Assume the device's lowest intensity, I_0 , cannot be perceived; that is value 0.0
- Assume 1.0 is highest intensity that can be achieved by the device
- Assume I_1 is value of lowest intensity that can be perceived
- Want each selected intensity value to be r times the previous:
 - $I_2 = I_1 * r$, $I_3 = I_2 * r = I_1 * r^2$, ..., $I_{255} = I_1 * r^{254}$
 - but, $I_{255} = 1.0 = I_1 * r^{254}$ so $r = (1/I_1)^{1/254} = I_1^{-1/254}$

Intensity Mapping Problems

- Human perception does not match any particular device
- Different display devices have different *dynamic ranges* so the mappings will be different.
- Gamma correction - compare 2 devices' behavior in attempt to have both show same image with same end visual effect.

Chromatic Color

- In addition to intensity/brightness, we perceive 2 other aspects of light:
 - hue (dominant wavelength)
 - what we think of as “color”
 - red and pink and colors between are all “reds”
 - saturation
 - the “purity” of the color
 - bright red is called “fully saturated”
 - as we “add” white to red, it gets more and more pink

Basic Color Models

- Hue, saturation, and intensity can be used to model color
- Red, Green, and Blue can also be used
 - Given pure red, green and blue lights, can represent (most) visible colors by mixing appropriate combinations of them
- Since most display device hardware is based on mixing red, green, and blue light, we'll use this model first.

Modeling Light

- Light in a scene
 - ambient light
 - no explicit source; collection of all secondary light beams bouncing around
 - point light sources
 - light that appears to emanate from an infinitely small point at a particular location; casts “hard” shadows
 - area/volume light sources
 - light that emanates from a light source with a non-zero area or volume; casts “soft” shadows

Models for Direct Light

- Ambient light (constant color)
 - Each object (or face of an object) has some constant color throughout the face: reflection of ambient
- Reflection
 - diffuse reflection: matte appearance
 - specular reflection: shininess
- Transmission: light passes through a semi-transparent surface
 - diffuse transmission: not usually modeled
 - specular transmission: light bends as it passes through the surface

Ambient Light

- Light in the scene with no identified source
- Efficiently models the light that bounces around scene from all light sources
- User specifies ambient light in the scene: I_a
- For each surface, user specifies the color and percentage of ambient light reflected by that surface:

$$k_a * C_a * I_a$$

where C_a and I_a are r,g,b triples (vectors) and k_a is % as fraction

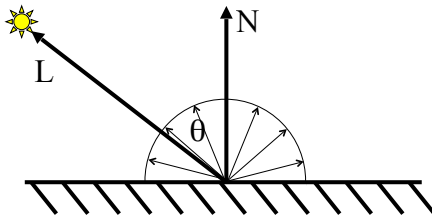
- Can also let each light contribute to the ambient light:

$$\sum k_a * C_a * I_L$$

Sometimes the constant k_a is folded into the C_a vector.

Diffuse Reflection

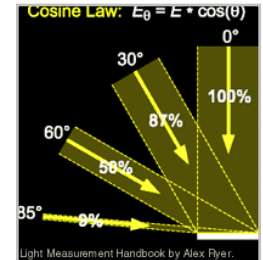
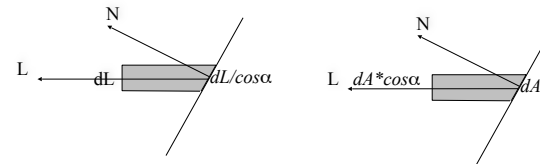
- Lambert's law for a perfect diffuser material
 - light from a light source is reflected from any point on the surface of a perfect diffuser equally in all directions; surface modeled as infinitely many *microfacets* aimed randomly in all directions
 - amount of reflection is based on the cosine of the angle between the surface normal at the point and the vector from the point to the light.



if L and N are unit vectors
 $I_d = k_d * C_d * I_L * L \cdot N$
 where $L \cdot N = \|L\| \|N\| \cos \theta = \cos \theta$
 and k_d is *diffuse* coefficient and C_d is the diffuse reflection color of the surface. Nearly always, $C_d = C_a$

Diffuse Reflection Revealed

- Consider a beam of light as a cylinder with base area dL
 - It illuminates a surface area, $dL/\cos\alpha$
 - Can also say an area, dA , of the surface is lit by a cross section of the light beam of area $dA * \cos\alpha$
- For a perfect diffuser the same number of photons will be reflected from dA towards the viewer regardless of the location of the viewer

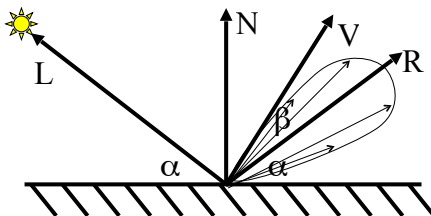


Specular Reflection

- Near the *direction of principal reflection*, shiny surfaces reflect the color of the light more than the color of the surface
 - Phong modeled this with a shininess coefficient, n , in the formula below:

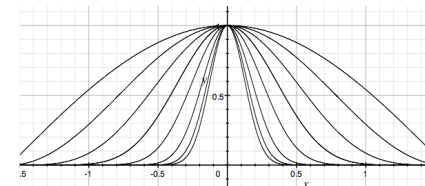
R is dir. of principal reflection
 $\|R\| = 1$
 V points at viewer, $\|V\| = 1$
 L points at Light source

$I_s = k_s * C_s * I_L * (V \cdot R)^n$ where
 C is the specular color of the surface. Usually $C_s = I_L$ (or 1,1,1) so that only reflected light seen but there are exceptions.



Shininess Coefficient

- The n coefficient raises the cosine ($V \cdot R$) to a power
- What effect does it have? As n gets larger, $\cos^n$ gets narrower.
 - shiny highlight gets smaller
 - Note that the brightness does not change



Specular Transmission

- Specular transmission can only be modeled in the context of the visible surface algorithm used to determine what is visible.
 - We'll come back to it when we talk about the z-buffer algorithm and ray tracing

Phong Lighting Model

- The basic lighting model is due to Bui Tuong Phong:
- Given
 - a point on a surface, P
 - a unit normal, N
 - a light source in the direction of the unit vector, L , with a light intensity of I_L
 - ambient light with an intensity, I_a
 - coefficients (rgb) of reflection of the surface: k_a, k_d, k_s
 - the shininess coefficient, n

$$I_p = k_a * C_a * I_a + k_d * C_d * I_L * N \cdot L + k_s * I_L * (V \cdot R)^n$$

$$I_p = k_a * C_a * I_a + I_L * (k_d * C_d * N \cdot L + k_s * (V \cdot R)^n)$$

$$I_p = k_a * C * I_a + I_L * (k_d * C * N \cdot L + k_s * (V \cdot R)^n) \text{ if } C_a = C_d$$

Finding the R Vector

- N is derived from the object specification, L is easily computed from the light's location and P
- But, how do we get R , the principal reflection direction?

Project L onto N to get $N \cos \theta = N(N \cdot L)$

From left triangle and vector arithmetic

$$N \cos \theta = L + S$$

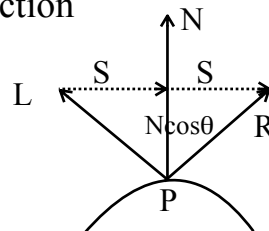
$$\text{So } S = N \cos \theta - L = N(N \cdot L) - L$$

Add "another" S from end of first S ; it meets end of R

From right triangle

$$R = N \cos \theta + S = N(N \cdot L) + N(N \cdot L) - L = 2N(N \cdot L) - L$$

$$\text{Can also derive from } R = L + S + S = L + 2N(N \cdot L) - 2L$$



Multiple Light Sources

- So far, we have only modeled 1 (point) light source
- Easy to add more

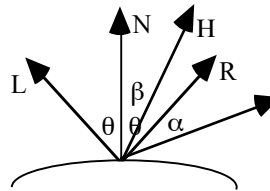
$$I_p = k_a * I_a + \sum_L I_L * (k_d * N \cdot L + k_s * (V \cdot R)^n)$$

- If we want each light to contribute to the ambient component, just add in

$$\sum_L k_a * I_a$$

Blinn-Phong Specular Reflection

- Blinn developed an alternative specular reflection model
 - Instead of $(R \cdot V)^n$, use $(N \cdot H)^n$
 - where H is the vector half way between L and V
 - if H is also a unit vector, $N \cdot H = \cos \beta$
 - α and β are different angles, but are very similar as V approaches R
- How do we get H ? Easier than R !
 $H = (L + V) / \|L + V\|$



Light Attenuation

- In the real world, light diminishes with distance
 - related to the distance squared from light to object
 - need large distances to matter
- Can simulate a more flexible attenuation by defining an attenuation factor for each light:

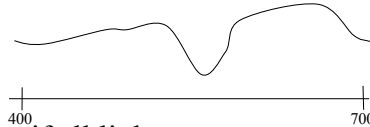
$$f = \min\left(\frac{1}{c_1 + c_2 d_L + c_3 d_L^2}, 1\right)$$

where c_1 , c_2 , and c_3 are defined by user

c_1 keeps the denominator from getting too small for small distances, and the other two constants allow some tuning, usually based on the relative distances of the objects with each other and the light source.

What about Color?

- Visible light is an energy distribution from 400nm to 700nm
- Our visual system integrates the spectral energy distribution into a perception of color
- Lighting model would be most correct if all light terms were expressed as functions of wavelength, λ



$$I_p(\lambda) = k_a(\lambda) * I_a(\lambda) + \sum_L f_L * I_L(\lambda) * (k_d(\lambda) * N \cdot L + k_s(\lambda) * (V \cdot R)^n)$$

and evaluated for many different λ values (8-10 or more).

- For most purposes, 3 λ values are sufficient and computationally tractable. Usually these are values representing red, green, blue, so we have to compute 3 equations of the form above.

Polygon Mesh Shading

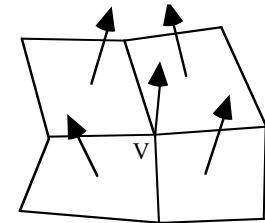
- Lots of objects can be efficiently and effectively modeled as a mesh of polygons (triangles, usually)
- Shading options
 - flat shading
 - Gouraud color interpolation
 - Phong normal interpolation

Flat Shading

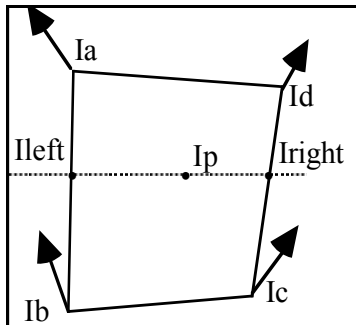
- Get normal to plane of the polygon
 - comes from polygon vertices
 - if it is a triangle, the polygon is guaranteed to be planar
 - if it has more than 3 vertices, there is a simple algorithm that will compute the plane normal if the vertices are coplanar, or the least error approximating plane
- Apply lighting model to get a color
 - L is usually computed as a vector from a point near the center of the polygon
- Color the entire polygon with that color

Polygon Vertex Calculation

- Both Gouraud and Phong shading require normals to polygon vertices
- If mesh is generated from an underlying surface specification, get normals from the surface spec
- If all you have is the mesh, compute vertex normals by averaging polygon normals that share the vertex



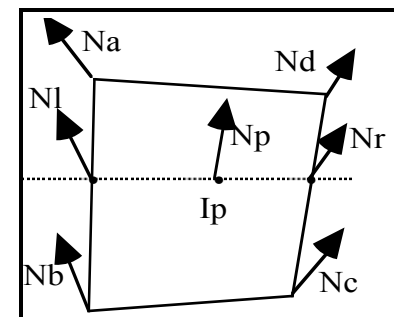
Gouraud Mesh Shading



- Intensity interpolation
 - Use the vertex normals to compute intensities at each vertex — I_a , I_b , I_c , and I_d
 - Interpolate intensity along edges — get I_{left} from I_a and I_b , and I_{right} from I_c and I_d
 - Interpolate intensities between edges across scanline — get I_p from I_{left} and I_{right} .

Phong Mesh Shading

- Normal interpolation
 - Interpolate normals along edges
 - Interpolate normals across scanline
 - Apply shading model at each pixel.



Comparing Mesh Shading

1. Gouraud interpolation: shading model applied once/vertex
2. Phong interpolation: shading model applied once/pixel
3. Interpolation costs are the same: 3 colors or 3 component vector
4. Phong handles specular highlights more accurately
5. Gouraud is done in all graphics cards

Generic Problems

- **General problems with interpolatory shading**

polygon silhouette: curved surface silhouettes still show polygonal approximation

perspective distortion: linear interpolation done in perspective world; not accurate in real world

orientation dependence: interpolation is dependent on orientation

shared vertices: get discontinuities if 2 adjacent polygons don't share vertex lying on shared edge.

unrepresentative vertex normals: zigzag polygons can produce inaccurate vertices; appears flat

