Illumination - Lighting

(original slides taken from David M. course)

Doron Nussbaum

- Colour
- Lighting models
- Phong Light model

Lighting / Illuminating / Shading

- Lighting - artificial computation for illuminating objects
- Illuminating – creating a sense of light that reaches objects
- Shading – producing different grades of colour (light).
- Used interchangibly
  - Assign a value of colour to a pixel

What is color, anyway?

- We think of it as a property of an object
  - "blue shirt", etc.
- Complex relationship between properties of material, lighting conditions, and properties of receptor
  - in graphics, need to account for properties of display device as well (monitor, printer)

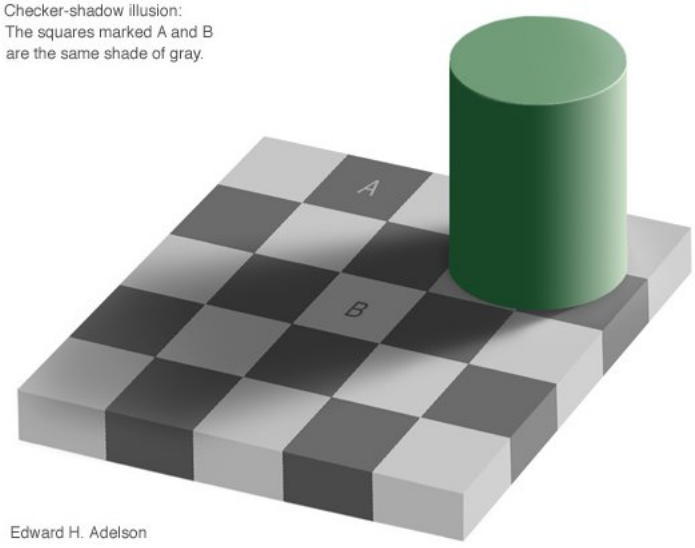
Human Color Vision

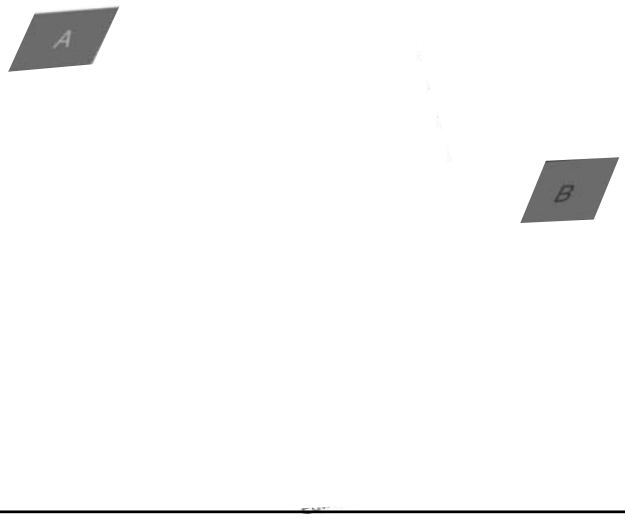
- Cones: three types – red (rho), green (gamma), blue (beta)
- Rho: 64% of cones
- Gamma: 32% of cones
- Beta: only 2% of cones, present outside foveal region
- Rho and Gamma spectral sensitivities overlap considerably

Metamerism

- Possible for distinct spectra to evoke the same sensor response
- Metamers: sets of spectra which are perceived as the same color
- Human vision system has 3 receptor types, so three-dimensional color space needed

Adelson's shadow

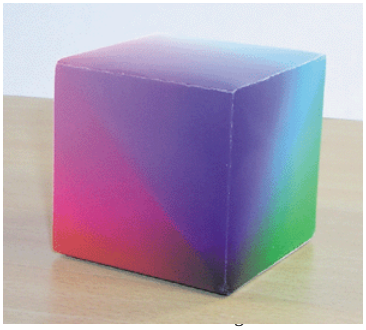




Adelson’s shadow

3D color space

- RGB: most common in graphics, tied to output capabilities of CRT monitors

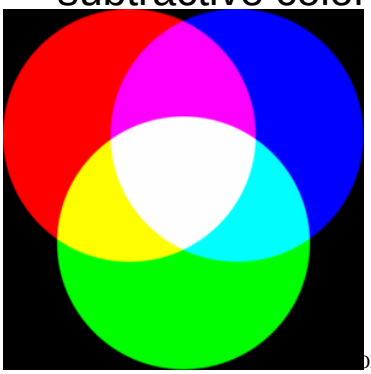


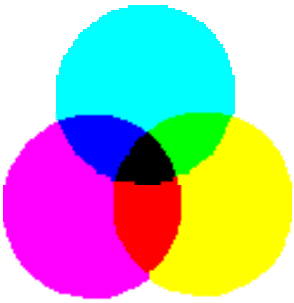
RGB

- Additive color model
  - Light
- Primary colors: R, G, B
  - (**R**,0,0), (0,**G**,0), (0,0,**B**)
- Secondary colors: yellow, magenta, cyan
  - (x,x,0), (x,0,x), (0,x,x)
- Host of other colors

CMYK

- Cyan, Magenta, Yellow, blacK (key)
- subtractive color model for ink





Colour Representation

32 bits

alpha	R	G	B

- D3DCOLOR – RGB
  - 32 bits – each colour is an 8 bits number 0-255
  - Examples
    - D3DCOLOR red = D3DCOLOR_ARGB(255, 255, 0, 0);
    - D3DCOLOR red = D3DCOLOR_XRGB(255, 0, 0);
- D3DCOLORVALUE – RGB
  - RGB values are represented in the range of [0, 1]
    - 0 no intensity, 1 full intensity
  - 4 floats - each float is a number in [0, 1]
  - Examples
    - D3DCOLORVALUE red = {0.25, 0.0, 0.75, 1.0);
  - Use D3DXCOLOR to manipulate the colors

```
struct _D3DCOLORVALUE
    float r;
    float g;
    float b;
    float a;
}
```

Shading Overview

- Classical real-time shading:
  - vertices projected to screen
  - lighting calculation done at each vertex
  - results interpolated
    - along line (linear interpolation)
    - to interior of polygon (bilinear interpolation)
- per-pixel shading:
  - uses interpolated values and texture lookups as input into program that calculates pixel color

Lighting Model Components

- Light Source
  - Type,
  - Location
  - Colour
- Material Properties
  - Reflection
  - Refraction
  - Absorption
  - Radiation

Classic Lighting Model

- Actual behavior of light extremely complex
  - Assumption – illumination assumes that RGB can be computed independently of each other
- Widely used light source simplification (CG):
  - Ambient lighting
  - Diffuse lighting
  - Specular lighting

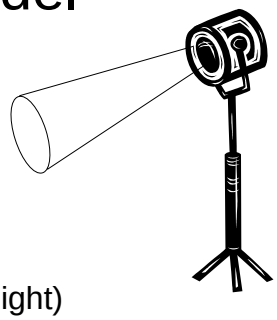
Classic Lighting Model

- Ambient lighting
  - Scattered
  - No particular source of light
  - Backlight (that exists)
    - results from bouncing of other lights
    - Provides general colour atmosphere of the environment
    - independent of view point
    - Undirected
  - Usually “flat” colour
- Diffuse lighting
- Specular lighting

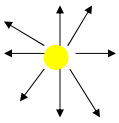
Classic Lighting Model

- Ambient lighting
- Diffuse lighting
  - Obeys a direction / parallel light
  - Independent of view point
  - Scatters equally in all directions when bouncing
  - Usually matte surfaces
- Specular lighting

Classic Lighting Model



- Ambient lighting
- Diffuse lighting
- Specular lighting
  - Obeys a direction
  - Point light or spotlight (e.g., flashlight)
  - Dependent of view point
  - Scatters in one directions when bouncing
  - Usually shiny surfaces
    - Effect of shininess (mirror)
  - Located in a particular location



Classic Lighting Model

- Actual behavior of light extremely complex
  - Assumption – illumination assumes that RGB can be computed independently of each other
- Widely used simplification in graphics:
  - Ambient lighting
    - Non-directional light
  - Diffuse lighting
    - matte surface
    - directional light / parallel light
  - Specular lighting
    - shiny surface
    - directional light / point light

Materials

- Materials properties affect the colour
- Radiation – material emits light
- Absorption/reflectance of light
  - Ambient
  - Diffuse
  - Specular
- Transparency level

- Note – diffuse and ambient light normally have the same setting

Three-term Lighting Model

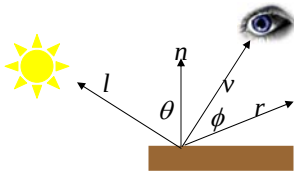
- Output colour consists of
- Material
  - Material radiaion +
  - Material ambient colour +
  - Material diffuse colour +
  - Material Specular colour
- Light
  - Ambient light /coefficient
  - Diffusion light / coefficient
  - Specular light / coefficient
  - Radiated light

Three-term Lighting Model

- Output colour intensity is computed for each light type components:
  - Ambient
    - $I_a = C$ (C_a is a constant ambient light intensity)
  - Diffuse (Lambertian)
    - direct lighting
    - $I_d = n \bullet l$
  - Specular
    - direct lighting
    - $I_s = (v \bullet r)^s$

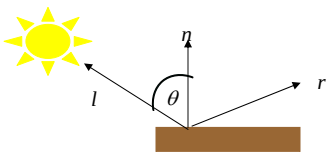
Model Geometry

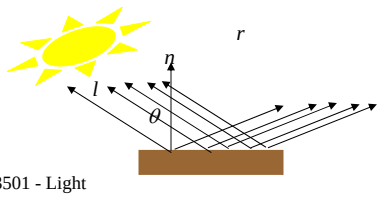
- l_p – light position
- l – light vector (l_p - p)
- n – normal vector to surface
- p – point on surface
- v_p – viewpoint
- v – viewpoint vector (v_p - p)
- r – reflection vector
- θ - angle between l and n
- ϕ - *angle between* v *and* r



Diffuse component

- Matte objects: “not at all shiny”
  - chalk, paper, rough wood, concrete
- Does not depend on viewing direction
  - Impendent of viewpoint
- Intensity of reflected light is proportional to the cos(θ)
  (angle between light source and surface normal)





Diffuse component - Lambert

- Reflected light is proportional to cos(θ)
- n is surface normal
- l is direction towards light
- All lights are “parallel”

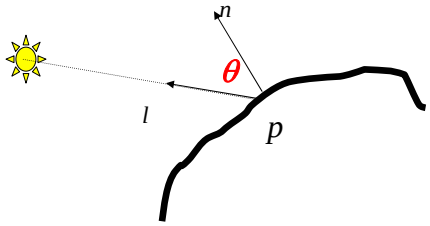
$I_d = \cos(\theta)$

$\cos(\theta) = l.n$

$I_d = l.n$

Note θ may be negative

$I_d = \max(l.n, 0)$



Diffuse component - Lambert

- Reflected light is proportional to cos(θ)
- n is surface normal
- l is direction towards light
- All lights are "parallel"

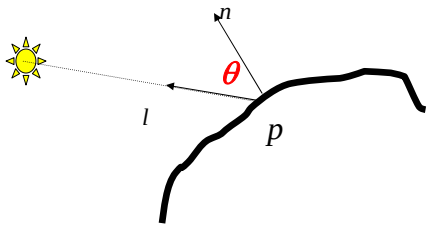
$$I_d = m_d * L_d * \cos(\theta)$$

$$\cos(\theta) = l.n$$

$$I_d = m_d * L_d * l.n$$

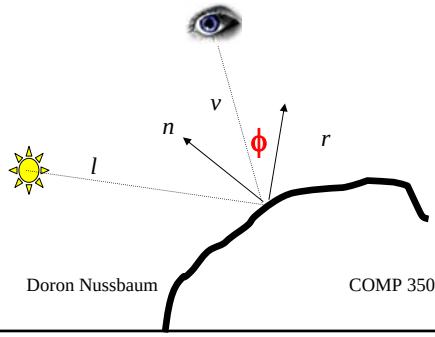
Note θ may be negative

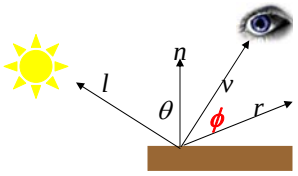
$$I_d = m_d * L_d * \max(l.n, 0)$$



Specular component (last)

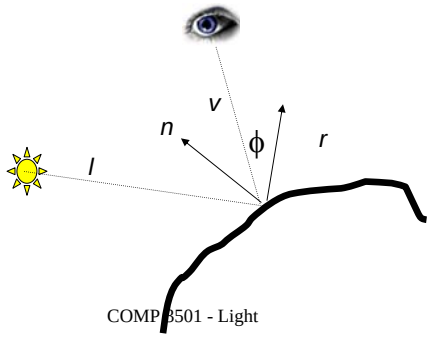
- Highlights on "shiny" (highly reflective) surfaces
- Intensity is proportional to cos(ϕ)
  - ϕ is the angle between viewing direction and direction of ideal reflection





Specular Component

- $I_s = (v \bullet r)^s$
- V is view direction
- R is reflection direction



Specular component

- Shininess
  - Use power to emphasize the spotlight and shininess
- Effect of power s –
  - Low values of s spreads out highlights
  - High values of s focus the highlight

$$I_s = (\cos(\phi))^s$$

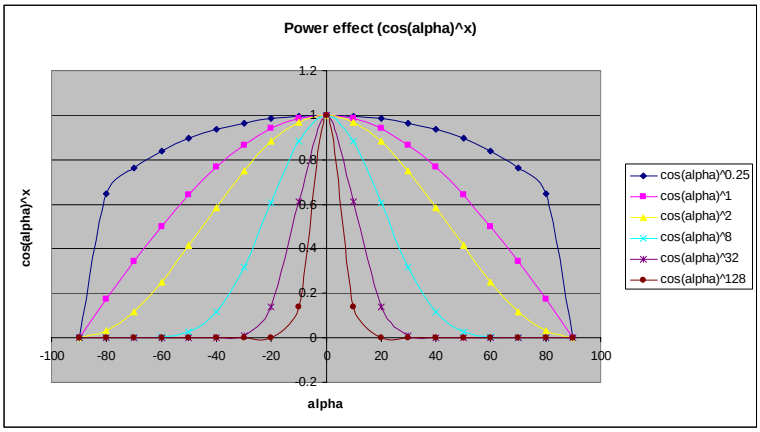
$$\cos(\phi) = r.v$$

$$Is = (r.v)^s$$

Note ϕ may be negative

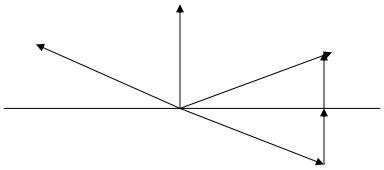
$$I_s = (\max(r.v,0))^s$$

Effect of Power



Specular component

- How to compute reflection direction?



- R = -L + 2*(L • N)N

Highlight spread

- Dot product of normalized vectors is cosine of angle between vectors
- (V • R)^*s* = (cos ϕ)^*p*
- Parameter *s* controls width of highlight:
  - low *s* (say 1-5): broad highlight
  - large *s*: (say 30-200): sharp highlight

Ambient component

- Some amount of light always present
- Add small constant light intensity
- Just a hack to account for scattered light reaching everywhere
- Ambient occlusion: small-scale self-shadowing, can be precomputed (static scene) or computed in real time (SSAO, Crysis)

Three-term Lighting Model

- Final colour is affected by:
  - Intensity of each of the light types – I_a, I_d, I_s
  - Light composition of the light types – l_a, l_d, l_s
  - Material composition for each of the light types– m_a, m_d, m_s
- Each component is consists of the RGB values
  - Vector (r, g, b)
- Examples
  - l_a = (0.2, 0.2, 0.6)
  - m_s = (1.0, 0.0, 0.0) material absorbs reflects only red

Three-term Lighting Model

- Final colour is computed by:

$$Colour = I_a * l_a * m_a + I_d * l_d * m_d + I_s * l_s * m_s$$
$$= C_a * l_a * m_a + \max(l.n,0) * l_d * m_d + (\max(v.r,0))^s * l_s * m_s$$

- In general if (light and/or material are not available then one can use constants K_a, K_d, K_s)

$$Colour = K_a + I_d * K_d + I_s * K_s$$
$$= K_a + \max(l.n,0) * K_d + (\max(v.r,0))^s * K_s$$

Setting of parameters

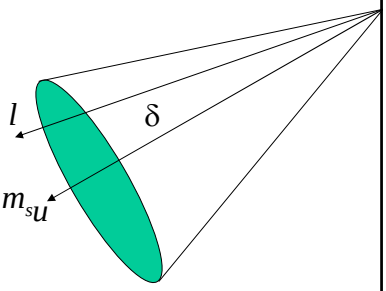
- Material
  - Usually ambient material and diffuse material are the same
- Light
  - Usually diffuse light and specular light are the same

Three-term Lighting Model

- Shading calculation done with three components:
  - $I = k_d \max(n \cdot l),0) + k_s \max(v \cdot r),0)^s + k_a$
  - k is surface albedo
  - Actually have three such equations, one each for R, G, and B
  - Not shown: lighting modulated by color of surface (material properties), incoming light

Spotlight as Light Source

- Light intensity is proportional to cos(δ)
  - δ is the angle between light ray, l, and cone centre vector, u

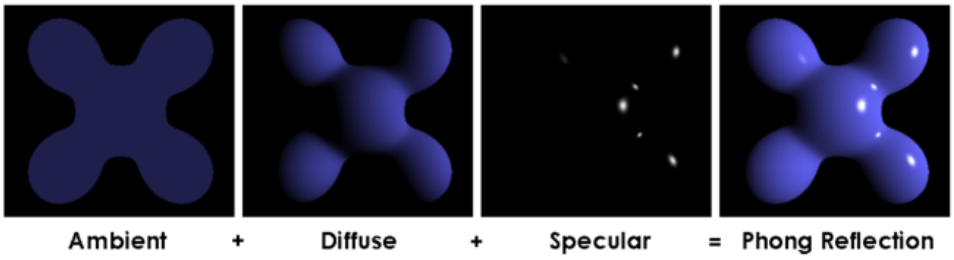


$$ls = (\max(u.-l,0))^{s_1}$$

$$Colour = I_a * l_a * m_a + I_d * l_d * m_d + I_s * l_s * m_s$$

$$Colour = C_a * l_a * m_a + \max(l.n,0) * l_d * m_d + (\max(v.r,0))^{s} * (\max(u.-l,0))^{s_1} * m_s$$

3-term lighting



"Phong" because per-pixel lighting

Light Attenuation

- Light intensity falls off with distance, d, from source

$$I(d) = \frac{I(0)}{d^2}$$

- Results are usually poor so (cheating or hacking)
  - Can "play" with a0, a1, a2 (e.g., a_0=a_1=0 and a_2=1 correct comp.)

$$I(d) = \frac{I(0)}{a_0 + a_1 d + a_2 d^2}$$

- Atmospheric haze: cue for large distances

Shading

- Computing the actual colour value of the pixels.
- What information is available?

Interpolation

- Remember – information available only at vertices
- Classically, lighting calculation done only at vertices also
- Need to interpolate to remainder of primitive
- flat shading also possible

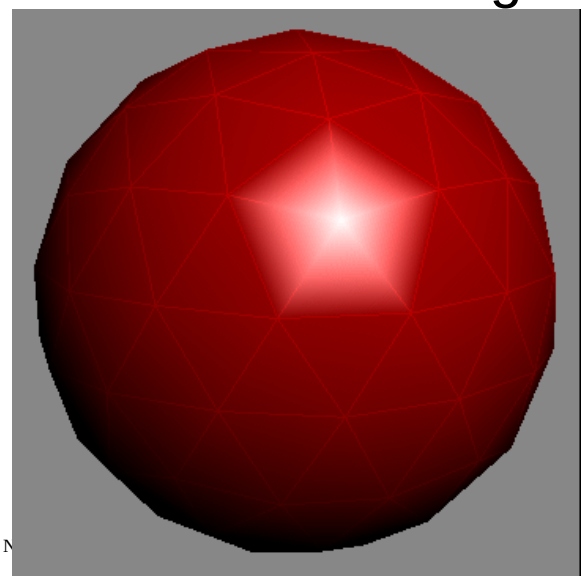
Linear Interpolation

- Linear interpolation: simplest method to interpolate
- $i(t) = i(0)*(1-t) + i(1)*t$, $t \in [0,1]$
- values known at 0 and 1, interpolated otherwise
- Widely used – here just for shading lines

Gouraud Shading

- Compute illumination at vertices
- Bilinearly interpolate to interior
- Gouraud shading proper:
  - compute surface normal vector from polygon
    - (cross product)
  - average all polygons normal vectors touching vertex to obtain vertex normal

Gouraud Shading



Phong Shading

- Lighting can change faster than geometry
- With insufficient vertex density, features can be missed

- Phong shading:
  - interpolate normal vectors to triangle interior
  - perform per-pixel lighting
- Much more compute intensive (costly) but better results

Phong vs Gouraud shading

- Historically, Gouraud usually used in real-time graphics
  - "why do all video games look the same?"
- Now, pixel shaders are the norm – per-pixel lighting possible (expected)

Custom Shading

- With the advent of programmable shaders, we are no longer restricted to the 3-term lighting model
- Pixel shaders now standard

- Phong shading
- other, specialized lighting models, effects

Three-term Lighting Model

- $I = k_d(N \bullet L) + k_s(V \bullet R)^n + k_a$
- Important quantities:
  - material of surface
  - normal vector
  - shininess
  - light direction, eye direction
- Where they come from
  - property of model/configuration
  - interpolated from vertex
  - stored in texture (or computed procedurally)

Toon Shading (last)

- Shading style characterized by
  - large flat-colored regions
    - "shading" quantized into few colors
  - black outlines denoting
    - silhouettes
    - internal object boundaries
      - (eg, eyes)
    - creases
  - Recent toon shaders often omit outlines

Quantized Shading

- Create 1D texture map showing progression of colors
- Calculate lighting as normal (diffuse+specular)
- Use lighting result to index into texture

- If only few colors, can use if statements
  - bad for reusability, good for rapid prototyping

Outlines

- Silhouettes: where depth differences exceed a threshold
  - can render to texture and find depth differences in second pass of pixel shader
- Boundaries: property of model, annotated
- Creases: property of model
  - could be annotated (artist, precomputed)
  - could be obtained by differencing normal map in pixel shader