University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2007

Tamara Munzner

Lighting/Shading II

Week 6, Fri Feb 16

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2007

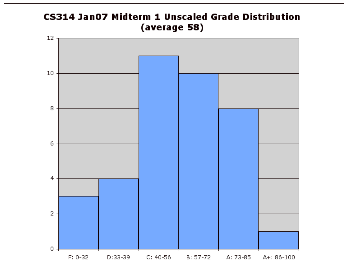
Correction/News

- Homework 2 was posted Wed
  - due Fri Mar 2
- Project 2 out today
  - due Mon Mar 5

News

- midterms returned
- project 2 out

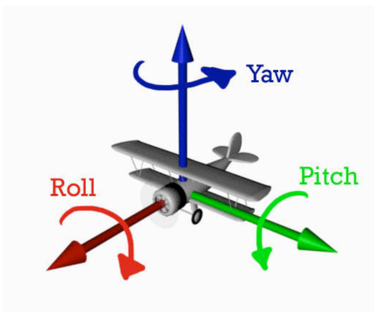
Midterm Grading



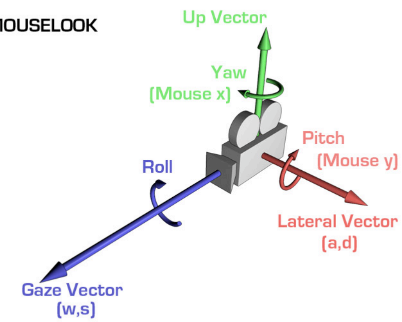
Project 2: Navigation

- five ways to navigate
  - Absolute Rotate/Translate Keyboard
  - Absolute Lookat Keyboard
    - move wrt global coordinate system
  - Relative Rolling Ball Mouse
    - spin around with mouse, as discussed in class
  - Relative Flying
  - Relative Mouselook
    - use both mouse and keyboard, move wrt camera
- template: colored ground plane

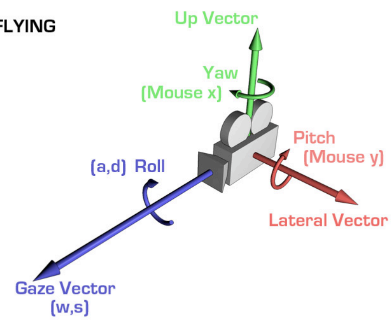
Roll/Pitch/Yaw



MOUSELOOK



FLYING



Demo

Hints: Viewing

- don't forget to flip y coordinate from mouse
  - window system origin upper left
  - OpenGL origin lower left
- all viewing transformations belong in modelview matrix, not projection matrix

Hint: Incremental Relative Motion

- motion is wrt current camera coords
  - maintaining cumulative angles wrt world coords would be difficult
  - computation in coord system used to draw previous frame (what you see!) is simple
    - at time k, want $p' = I_k I_{k-1} \ldots I_5 I_4 I_3 I_2 I_1 Cp$
    - thus you want to premultiply: p'=ICp
    - but postmultiplying by new matrix gives p'=CIp
  - OpenGL modelview matrix has the info! sneaky trick:
    - dump out modelview matrix with `glGetDoublev()`
    - wipe the stack with `glIdentity()`
    - apply incremental update matrix
    - apply current camera coord matrix
  - be careful to leave the modelview matrix unchanged after your display call (using push/pop)

Caution: OpenGL Matrix Storage

- OpenGL internal matrix storage is columnwise, not rowwise

```
a  e  i  m
b  f  j  n
c  g  k  o
d  h  l  p
```

  - opposite of standard C/C++/Java convention
  - possibly confusing if you look at the matrix from glGetDoublev()!

Reading for Wed/Today/Next Time

- FCG Chap 9 Surface Shading
- RB Chap Lighting

Review: Computing Barycentric Coordinates

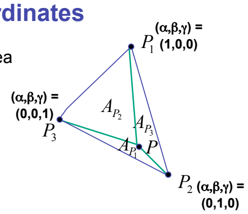
- 2D triangle area
  - half of parallelogram area
    - from cross product

$A = A_{P1} + A_{P2} + A_{P3}$

$\alpha = A_{P1} / A$

$\beta = A_{P2} / A$

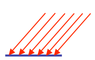
$\gamma = A_{P3} / A$



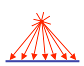
weighted combination of three points
[demo]

Review: Light Sources

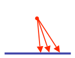
- directional/parallel lights
  - point at infinity: $(x,y,z,0)^T$



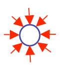
- point lights
  - finite position: $(x,y,z,1)^T$



- spotlights
  - position, direction, angle



- ambient lights



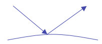
Lighting I

Light Source Placement

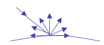
- geometry: positions and directions
  - standard: world coordinate system
    - effect: lights fixed wrt world geometry
    - demo: http://www.xmission.com/~nate/tutors.html
  - alternative: camera coordinate system
    - effect: lights attached to camera (car headlights)
  - points and directions undergo normal model/view transformation
- illumination calculations: camera coords

Types of Reflection

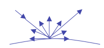
- *specular* (a.k.a. *mirror* or *regular*) reflection causes light to propagate without scattering.



- *diffuse* reflection sends light in all directions with equal energy.

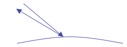


- *mixed* reflection is a weighted combination of specular and diffuse.

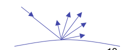


Types of Reflection

- *retro-reflection* occurs when incident energy reflects in directions close to the incident direction, for a wide range of incident directions.

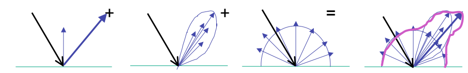


- *gloss* is the property of a material surface that involves mixed reflection and is responsible for the mirror like appearance of rough surfaces.



Reflectance Distribution Model

- most surfaces exhibit complex reflectances
  - vary with incident and reflected directions.
  - model with combination

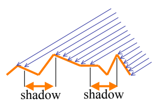


specular + glossy + diffuse = reflectance distribution

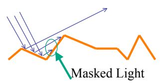
Surface Roughness

- at a microscopic scale, all real surfaces are rough

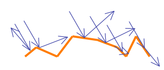
- cast shadows on themselves



- "mask" reflected light:



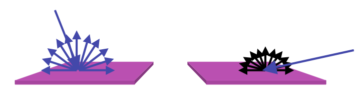
Surface Roughness



- notice another effect of roughness:
  - each "microfacet" is treated as a perfect mirror.
  - incident light reflected in different directions by different facets.
  - end result is mixed reflectance.
    - smoother surfaces are more specular or glossy.
    - random distribution of facet normals results in diffuse reflectance.

Physics of Diffuse Reflection

- ideal diffuse reflection
  - very rough surface at the microscopic level
    - real-world example: chalk
  - microscopic variations mean incoming ray of light equally likely to be reflected in any direction over the hemisphere
  - what does the reflected intensity depend on?



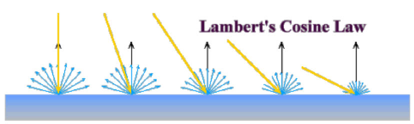
Lambert's Cosine Law

- ideal diffuse surface reflection

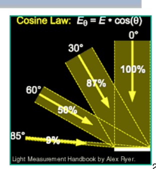
  the energy reflected by a small portion of a surface from a light source in a given direction is proportional to the cosine of the angle between that direction and the surface normal

- reflected intensity
  - independent of viewing direction
  - depends on surface orientation wrt light
- often called Lambertian surfaces

Lambert's Law

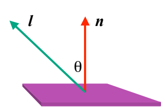


intuitively: cross-sectional area of the "beam" intersecting an element of surface area is smaller for greater angles with the normal.



Computing Diffuse Reflection

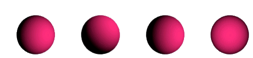
- depends on angle of incidence: angle between surface normal and incoming light
  - $I_{diffuse} = k_d \, I_{light} \cos\theta$
- in practice use vector arithmetic
  - $I_{diffuse} = k_d \, I_{light} \, (\mathbf{n} \bullet \mathbf{l})$



- **always normalize vectors used in lighting!!!**
  - $\mathbf{n}$, $\mathbf{l}$ should be unit vectors
- scalar (B/W intensity) or 3-tuple or 4-tuple (color)
  - k_d: diffuse coefficient, surface color
  - I_{light}: incoming light intensity
  - $I_{diffuse}$: outgoing light intensity (for diffuse reflection)

Diffuse Lighting Examples

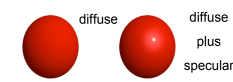
- Lambertian sphere from several lighting angles:



- need only consider angles from 0° to 90°
  - [demo] Brown exploratory on reflection
    - http://www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/reflection2D/reflection_2d_java_browser.html

Specular Reflection

- shiny surfaces exhibit specular reflection
  - polished metal
  - glossy car finish



- specular highlight
  - bright spot from light shining on a specular surface
- view dependent
  - highlight position is function of the viewer's position

Specular Highlights

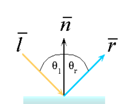
Michiel van de Panne

Physics of Specular Reflection

- at the microscopic level a specular reflecting surface is very smooth
- thus rays of light are likely to bounce off the microgeometry in a mirror-like fashion
- the smoother the surface, the closer it becomes to a perfect mirror

Optics of Reflection

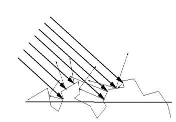
- reflection follows *Snell's Law:*
  - incoming ray and reflected ray lie in a plane with the surface normal
  - angle the reflected ray forms with surface normal equals angle formed by incoming ray and surface normal



$$\theta_{(l)ight} = \theta_{(r)eflection}$$

Non-Ideal Specular Reflectance

- Snell's law applies to perfect mirror-like surfaces, but aside from mirrors (and chrome) few surfaces exhibit perfect specularity
- how can we capture the "softer" reflections of surface that are glossy, not mirror-like?
- one option: model the microgeometry of the surface and explicitly bounce rays off of it
- or…

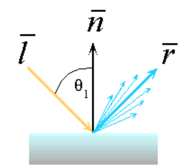


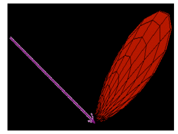
Empirical Approximation

- we expect most reflected light to travel in direction predicted by Snell's Law
- but because of microscopic surface variations, some light may be reflected in a direction slightly off the ideal reflected ray
- as angle from ideal reflected ray increases, we expect less light to be reflected

Empirical Approximation

- angular falloff





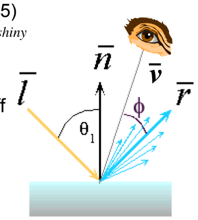
- how might we model this falloff?

Phong Lighting

- most common lighting model in computer graphics
  - (Phong Bui-Tuong, 1975)

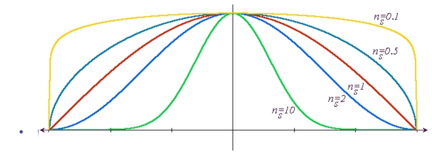
$$\mathbf{I}_{\mathbf{specular}} = \mathbf{k}_s \mathbf{I}_{\mathbf{light}} (\cos\phi)^{n_{shiny}}$$

- n_{shiny} : purely empirical constant, varies rate of falloff
- k_s: specular coefficient, highlight color
- no physical basis, works ok in practice



Phong Lighting: The n_{shiny} Term

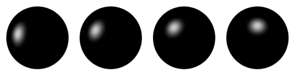
- Phong reflectance term drops off with divergence of viewing angle from ideal reflected ray



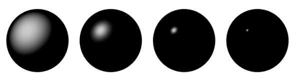
Viewing angle – reflected angle

Phong Examples

varying l



varying n_{shiny}

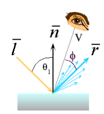


Calculating Phong Lighting

- compute **cosine** term of Phong lighting with vectors

$$\mathbf{I}_{\mathbf{specular}} = \mathbf{k}_s \mathbf{I}_{\mathbf{light}} (\mathbf{v} \bullet \mathbf{r})^{n_{shiny}}$$

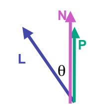
- v: unit vector towards viewer/eye
- r: ideal reflectance direction (unit vector)
- k_s: specular component
  - highlight color
- I_{light}: incoming light intensity



- how to efficiently calculate **r** ?

Calculating R Vector

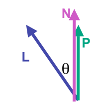
P = **N** cos θ = projection of **L** onto **N**



Calculating R Vector

P = **N** cos θ = projection of **L** onto **N**

P = N (N · L)

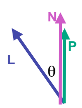


Calculating R Vector

P = **N** cos θ |**L**| |**N**| projection of **L** onto **N**

P = **N** cos θ   **L, N** are unit length

P = N (N · L)



Calculating R Vector

P = **N** cos θ |**L**| |**N**| projection of **L** onto **N**

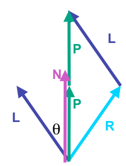
P = **N** cos θ   **L, N** are unit length

P = N (N · L)

2 P = R + L

2 P – L = R

2 (N (N · L)) - L = R



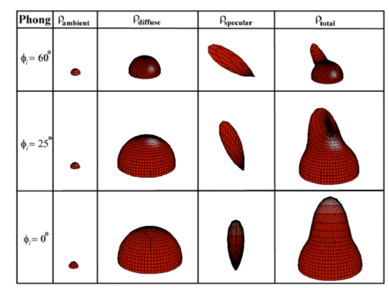
Phong Lighting Model

- combine ambient, diffuse, specular components

$$\mathbf{I}_{\mathbf{total}} = \mathbf{k}_s \mathbf{I}_{\mathbf{ambient}} + \sum_{i=1}^{\#lights} \mathbf{I}_i (\mathbf{k}_d (\mathbf{n} \bullet \mathbf{l}_i) + \mathbf{k}_s (\mathbf{v} \bullet \mathbf{r}_i)^{n_{shiny}})$$

- commonly called *Phong lighting*
  - once per light
  - once per color component
- reminder: normalize your vectors when calculating!

Phong Lighting: Intensity Plots

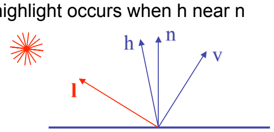


Blinn-Phong Model

- variation with better physical interpretation
  - Jim Blinn, 1977

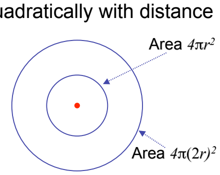
$$I_{out}(\mathbf{x}) = \mathbf{k}_s (\mathbf{h} \bullet \mathbf{n})^{n_{shiny}} \bullet I_{in}(\mathbf{x}); \text{with } \mathbf{h} = (\mathbf{l} + \mathbf{v})/2$$

- ***h***: halfway vector
  - h must also be explicitly normalized: h / |h|
  - highlight occurs when h near n



Light Source Falloff

- quadratic falloff
- brightness of objects depends on power per unit area that hits the object
- the power per unit area for a point or spot light decreases quadratically with distance



Light Source Falloff

- non-quadratic falloff
- many systems allow for other falloffs
- allows for faking effect of area light sources
- OpenGL / graphics hardware
  - I_o: intensity of light source
  - $\boldsymbol{x}$: object point
  - r: distance of light from x

$$I_{in}(\mathbf{x}) = \frac{1}{ar^2 + br + c} \cdot I_0$$

Lighting Review

- lighting models
- ambient
  - normals don't matter
- Lambert/diffuse
  - angle between surface normal and light
- Phong/specular
  - surface normal, light, and viewpoint

Lighting in OpenGL

- light source: amount of RGB light emitted
  - value represents percentage of full intensity e.g., (1.0,0.5,0.5)
  - every light source emits ambient, diffuse, and specular light
- materials: amount of RGB light reflected
  - value represents percentage reflected e.g., (0.0,1.0,0.5)
- interaction: multiply components
  - red light (1,0,0) x green surface (0,1,0) = black (0,0,0)

Lighting in OpenGL

```
glLightfv(GL_LIGHT0, GL_AMBIENT, amb_light_rgba );
glLightfv(GL_LIGHT0, GL_DIFFUSE, dif_light_rgba );
glLightfv(GL_LIGHT0, GL_SPECULAR, spec_light_rgba );
glLightfv(GL_LIGHT0, GL_POSITION, position);
glEnable(GL_LIGHT0);

glMaterialfv( GL_FRONT, GL_AMBIENT, ambient_rgba );
glMaterialfv( GL_FRONT, GL_DIFFUSE, diffuse_rgba );
glMaterialfv( GL_FRONT, GL_SPECULAR, specular_rgba );
glMaterialfv( GL_FRONT, GL_SHININESS, n );
```

- warning: glMaterial is expensive and tricky
  - use cheap and simple glColor when possible
  - see OpenGL Pitfall #14 from Kilgard's list

http://www.opengl.org/resources/features/KilgardTechniques/oglpitfall/