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

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

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

Review: Computing Barycentric Coordinates
- 2D triangle area
- half of parallelogram area
- from cross product
  \[ A = \mathbf{A}_{p1} + \mathbf{A}_{p2} + \mathbf{A}_{p3} \]
  \[ \alpha = \frac{\mathbf{A}_{p1}}{A} \]
  \[ \beta = \frac{\mathbf{A}_{p2}}{A} \]
  \[ \gamma = \frac{\mathbf{A}_{p3}}{A} \]
  weighted combination of three points
  \[ \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \]

Hints: 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 \( \mathbf{p}' = \mathbf{I} \mathbf{p}_{k-1} + \mathbf{g}(\mathbf{p}_{k-1}, \mathbf{C}_p) \)
    - thus you want to premultiply, \( \mathbf{I} > \mathbf{C}_p \)
    - but postmultiplying by new matrix gives \( \mathbf{p}' = \mathbf{C}_p \mathbf{p} \)
- OpenGL modelview matrix has the \( \mathbf{I} \) sneaky trick:
  - dump out modelview matrix with \texttt{glGetDoublev(GL_MODELVIEW_MATRIX)}
  - wipe the stack with \texttt{glLoadIdentity}()
  - 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
  \[ \begin{bmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & o \\ d & h & l & p \end{bmatrix} \]
  opposite of standard C/C++/Java convention
  possibly confusing if you look at the matrix from \texttt{glGetDoublev()}!

Reading for Wed/Today/Next Time
- FCG Chap 9 Surface Shading
- RB Chap Lighting

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
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.

Surface Roughness
- at a microscopic scale, all real surfaces are rough
- cast shadows on themselves
- “mask” reflected light:

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

Diffuse Lighting Examples
- Lambertian sphere from several lighting angles:
- need only consider angles from 0° to 90°
- [demo] Brown exploratory on reflection

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

Non-Ideal Specular Reflectance
- Snell's law applies to perfect mirror-like surfaces, but aside from mirrors (and chrome) few surfaces exhibit perfect specularly
- 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

Blinn-Phong Model
- variation with better physical interpretation
- Jim Blinn, 1977
- \( h \): halfway vector
- \( h \) must also be explicitly normalized: \( h / |h| \)
- highlight occurs when \( h \) near \( n \)

Jim Blinn, 1977

Phong Lighting
- most common lighting model in computer graphics
- (Phong Bui-Tuong, 1975)
- \( I_{\text{specular}} = k_s I_{\text{light}} (\cos \phi)^{n_{\text{shiny}}} \)
- \( n_{\text{shiny}} \): purely empirical constant, varies rate of falloff
- \( k_s \): specular coefficient, highlight color
- no physical basis, works ok in practice

Phong Examples
- varying \( l \)
- varying \( n_{\text{shiny}} \)

Calculating Phong Lighting
- compute cosine term of Phong lighting with vectors
- \( v \): unit vector towards viewer/eye
- \( r \): divergence of viewing angle from ideal reflected ray

Calculating R Vector
- \( P = N \cos \theta \): projection of \( L \) onto \( N \)
- \( P = N (N \cdot L) \)

Light Source Falloff
- non-quadratic falloff
- many systems allow for other falloffs
- allows for faking effect of area light sources
- OpenGL / graphics hardware
- \( I_L \): intensity of light source
- \( x \): object point
- \( r \): distance of light from \( x \)
- \( k_0 \): normalize your vectors when calculating!

Lighting Review
- lighting models
- ambient
- \( k_0 \) must also be explicitly normalized: \( h / |h| \)
- highlight occurs when \( h \) near \( n \)
- \( h \) varies, \( n \) constant, varies rate of falloff
- \( k_0 \): specular coefficient, highlight color
- no physical basis, works ok in practice
- reminder: normalize your vectors when calculating!
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)

```c
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);

// 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/
```