
CIS 454 Computer Graphics

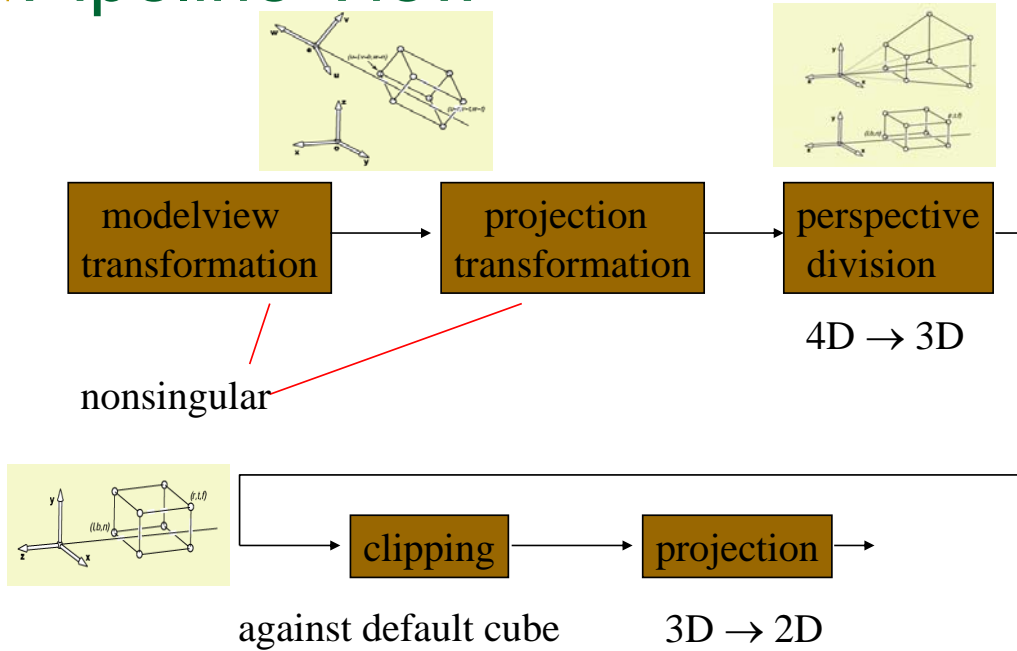
Lecture 19, 11/14/2006

Li Shen
Computer and Information Science
UMass Dartmouth

Computer Viewing in OpenGL

- Positioning the camera: [demo](#)
 - `glMatrixMode(GL_MODELVIEW);` [cube_view.c](#)
 - `glLoadIdentity();`
 - `glTranslatef(0.0, 0.0, -d);`
 - `glRotatef(90.0, 0.0, 1.0, 0.0);`
 - `glLoadIdentity();`
 - `gluLookAt(1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0., 1.0, 0.0);`
 - Selecting a lens and clipping:
 - `glMatrixMode(GL_PROJECTION);`
 - `glOrtho(left, right, bottom, top, near, far)`
 - `glFrustum(left, right, bottom, top, near, far)`
 - `gluPerspective(fovy, aspect, near, far)`
-

Pipeline View



Shading

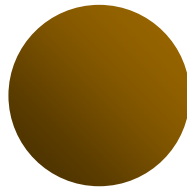
Objectives

demo

- Learn to shade objects so their images appear three-dimensional
- Introduce the types of light-material interactions
- Build a simple reflection model---the Phong model--- that can be used with real time graphics hardware
- Introduce modified Phong model
- Consider computation of required vectors

Shading

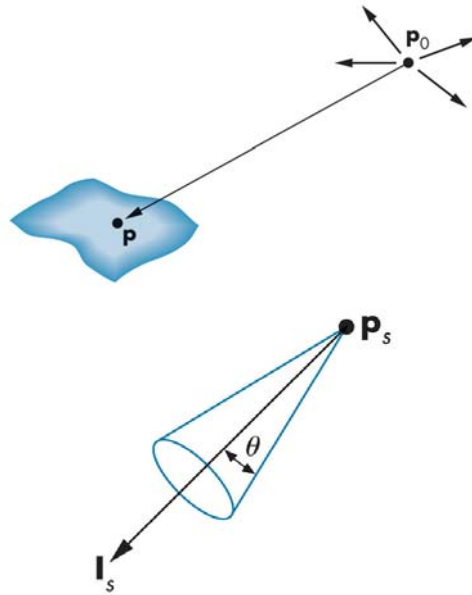
- Why does the image of a real sphere look like



- Light-material interactions cause each point to have a different color or shade
- Need to consider
 - Light sources
 - Material properties
 - Location of viewer
 - Surface orientation

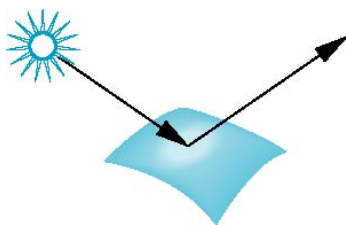
Simple Light Sources

- Point source
 - Model with position and color
 - Distant source = infinite distance away (parallel)
- Spotlight
 - Restrict light from ideal point source
- Ambient light
 - Same amount of light everywhere in scene
 - Can model contribution of many sources and reflecting surfaces

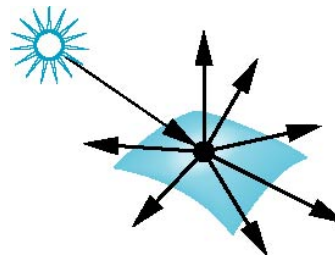


Surface Types

- The smoother a surface, the more reflected light is concentrated in the direction a perfect mirror would reflected the light
- A very rough surface scatters light in all directions



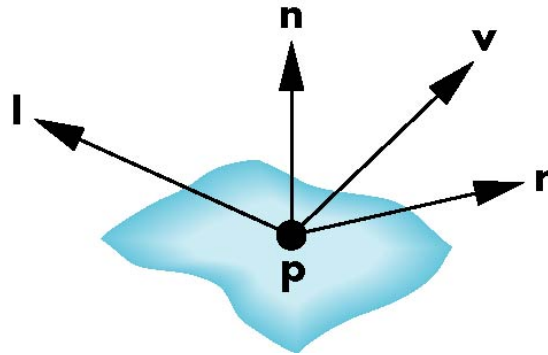
smooth surface



rough surface

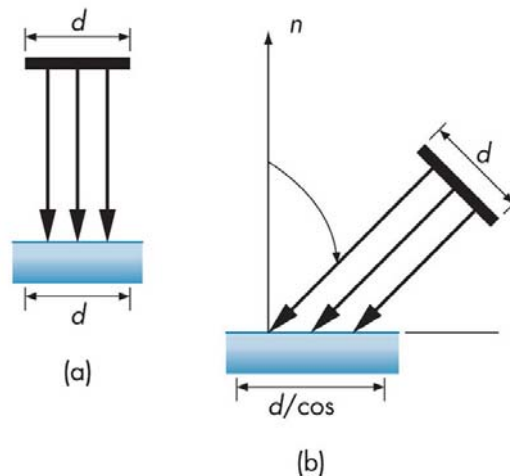
Phong Model

- A simple model that can be computed rapidly
- Has three components
 - Diffuse
 - Specular
 - Ambient
- Uses four vectors
 - To source
 - To viewer
 - Normal
 - Perfect reflector



Lambertian Surface

- Perfectly diffuse reflector
- Light scattered equally in all directions
- Amount of light reflected is proportional to the vertical component of incoming light
 - reflected light $\sim \cos \theta_i$
 - $\cos \theta_i = \mathbf{l} \cdot \mathbf{n}$ if vectors normalized
 - There are also three coefficients, k_r, k_b, k_g that show how much of each color component is reflected

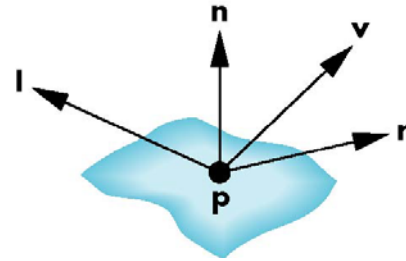


Phong Model: Diffuse Component

For each light source and each color component, the Phong model can be written (without the distance terms) as

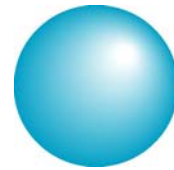
$$I = \underline{k_d I_d \mathbf{l} \cdot \mathbf{n}} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^\alpha + k_a I_a$$

For each color component we add contributions from all sources



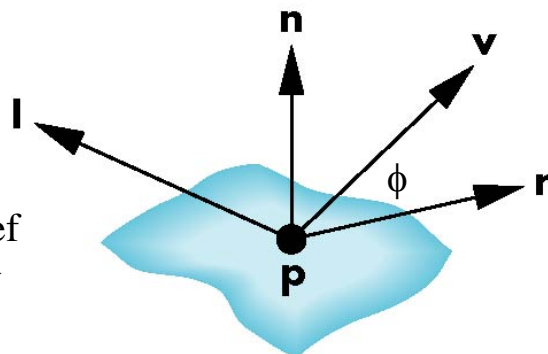
Modeling Specular Reflections

- Phong proposed using a term that dropped off as the angle between the viewer and the ideal reflection increased



$$I_r \sim k_s I \cos^\alpha \phi$$

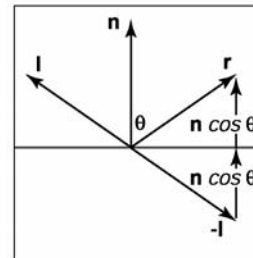
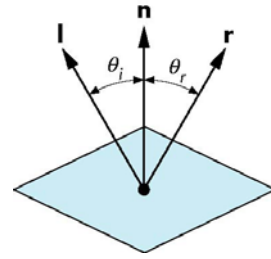
reflected intensity I_r
 absorption coef k_s
 incoming intensity I
 shininess coef $\cos^\alpha \phi$



Ideal Reflector

- Normal is determined by local orientation
- Angle of incidence = angle of reflection
- The three vectors must be coplanar
- Compute \mathbf{r}

$$\mathbf{r} = -\mathbf{l} + 2(\mathbf{l} \cdot \mathbf{n}) \mathbf{n}$$

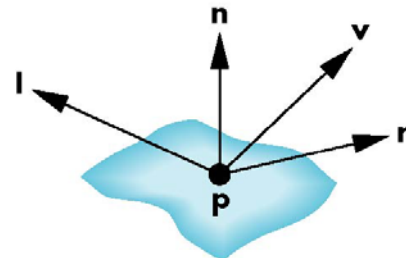
Figure 7: The geometry for calculating the vector \mathbf{r} .

Phong Model: Specular Component

For each light source and each color component, the Phong model can be written (without the distance terms) as

$$I = k_d I_d \mathbf{l} \cdot \mathbf{n} + \frac{k_s I_s (\mathbf{v} \cdot \mathbf{r})^\alpha}{\mathbf{v} \cdot \mathbf{r}} + k_a I_a$$

For each color component we add contributions from all sources



Ambient Light

- Ambient light is the result of multiple interactions between (large) light sources and the objects in the environment
- Amount and color depend on both the color of the light(s) and the material properties of the object
- Add $k_a I_a$ to diffuse and specular terms

reflection coef

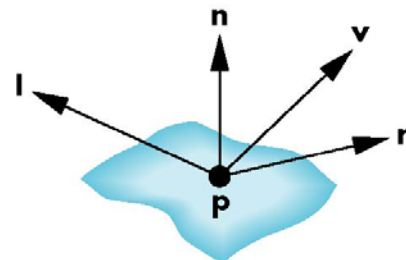
intensity of ambient light

Phong Model: Ambient Component

For each light source and each color component, the Phong model can be written (without the distance terms) as

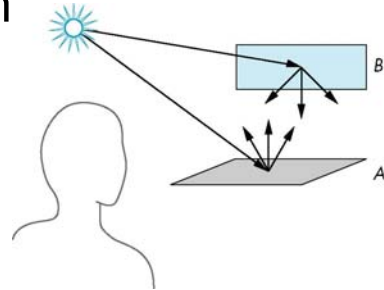
$$I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^\alpha + \underline{k_a I_a}$$

For each color component we add contributions from all sources



Distance Terms

- The light from a point source that reaches a surface is inversely proportional to the square of the distance between them
- We can add a factor of the form $\frac{1}{(ad + bd + cd^2)}$ to the diffuse and specular terms
- The constant and linear terms soften the effect of the point source



Light Sources

- In the Phong Model, we add the results from each light source

$$I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^\alpha + k_a I_a$$

- Each light source has separate **diffuse**, **specular**, and **ambient** terms to allow for maximum flexibility even though this form does not have a physical justification
- Separate **red**, **green** and **blue** components
- Hence, 9 coefficients for each point source
 - $I_{dr}, I_{dg}, I_{db}, I_{sr}, I_{sg}, I_{sb}, I_{ar}, I_{ag}, I_{ab}$

Material Properties

- Material properties match light source properties
 - Nine absorption coefficients
 - $k_{dr}, k_{dg}, k_{db}, k_{sr}, k_{sg}, k_{sb}, k_{ar}, k_{ag}, k_{ab}$
 - Shininess coefficient α

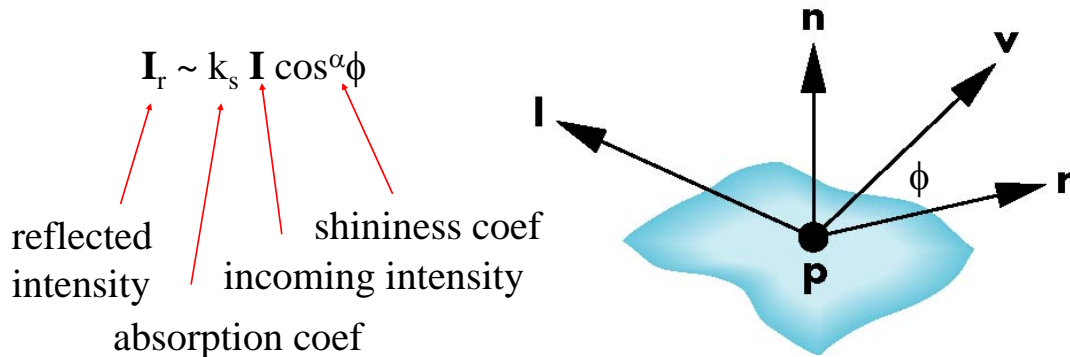
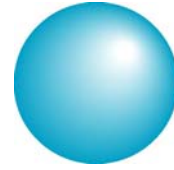
- $I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^\alpha + k_a I_a$

Modified Phong Model

- The specular term in the Phong model is problematic because it requires the calculation of a new reflection vector and view vector for each vertex
- Blinn suggested an approximation using the halfway vector that is more efficient

Modeling Specular Reflections

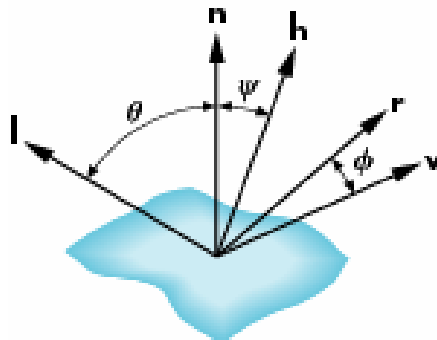
- Phong proposed using a term that dropped off as the angle between the viewer and the ideal reflection increased



The Halfway Vector

- \mathbf{h} is normalized vector halfway between \mathbf{l} and \mathbf{v}

$$\mathbf{h} = (\mathbf{l} + \mathbf{v}) / |\mathbf{l} + \mathbf{v}|$$



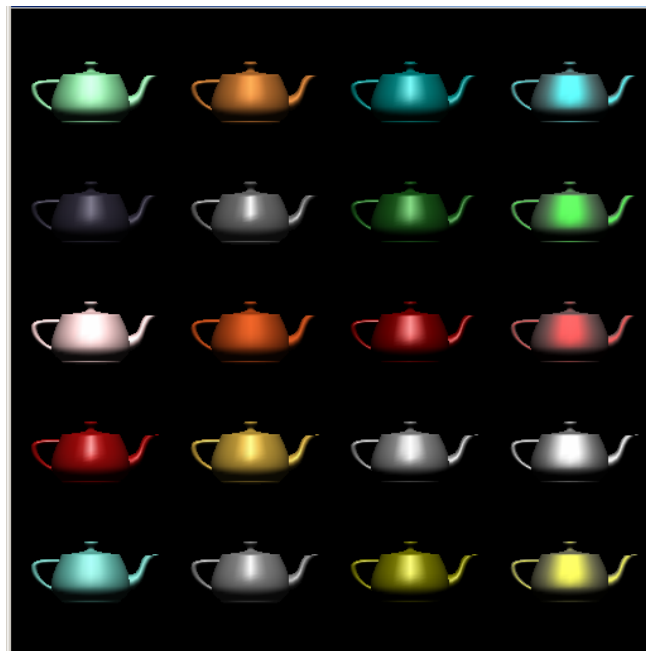
Using the Halfway Angle

- Replace $(\mathbf{v} \cdot \mathbf{r})^\alpha$ by $(\mathbf{n} \cdot \mathbf{h})^\beta$
- β is chosen to match shininess
- Note that halfway angle is half of angle between \mathbf{r} and \mathbf{v} if vectors are coplanar
- Resulting model is known as the modified Phong or Blinn lighting model
 - Specified in OpenGL standard

Example

demo

Only differences in these teapots are the parameters in the modified Phong model



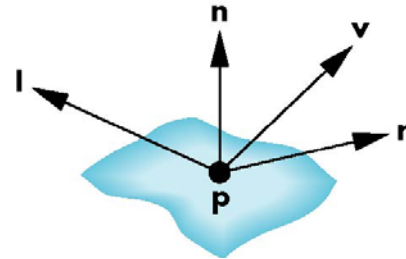
$$I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^\beta + k_a I_a$$

Phong Model

For each light source and each color component, the Phong model can be written (without the distance terms) as

$$I = k_d I_d \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^\alpha + k_a I_a$$

For each color component we add contributions from all sources



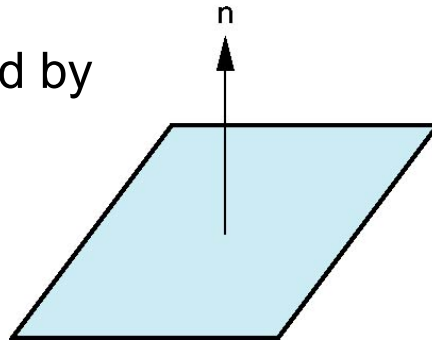
Computation of Vectors

- \mathbf{l} and \mathbf{v} are specified by the application
- Can compute \mathbf{r} from \mathbf{l} and \mathbf{n}
- Problem is determining \mathbf{n}
- For simple surfaces \mathbf{n} can be determined but how we determine \mathbf{n} differs depending on underlying representation of surface
- OpenGL leaves determination of normal to application
 - Exception for GLU quadrics and Bezier surfaces (Chapter 11)

Plane Normals

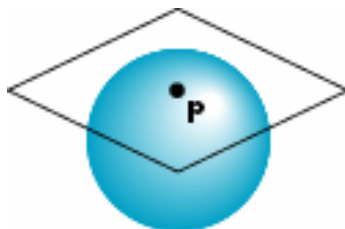
- Equation of plane: $ax+by+cz+d = 0$
- From Chapter 4 we know that plane is determined by three points p_0, p_1, p_2 or normal \mathbf{n} and p_0
- Normal can be obtained by

$$\mathbf{n} = (p_2 - p_0) \times (p_1 - p_0)$$



Normal to Sphere

- Implicit function $f(x,y,z)=0$
- Normal given by gradient
- Sphere $f(\mathbf{p}) = \mathbf{p} \cdot \mathbf{p} - 1$
- $\mathbf{n} = [\partial f / \partial x, \partial f / \partial y, \partial f / \partial z]^T = \mathbf{p}$



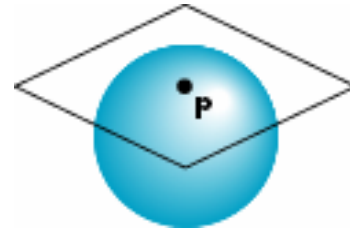
Parametric Form

- For sphere

$$x=x(u,v)=\cos u \sin v$$

$$y=y(u,v)=\cos u \cos v$$

$$z=z(u,v)=\sin u$$



- Tangent plane determined by vectors

$$\partial \mathbf{p} / \partial u = [\partial x / \partial u, \partial y / \partial u, \partial z / \partial u]^T$$

$$\partial \mathbf{p} / \partial v = [\partial x / \partial v, \partial y / \partial v, \partial z / \partial v]^T$$

- Normal given by cross product

$$\mathbf{n} = \partial \mathbf{p} / \partial u \times \partial \mathbf{p} / \partial v$$

General Case

- We can compute parametric normals for other simple cases
 - Quadrics
 - Parametric polynomial surfaces
 - Bezier surface patches (Chapter 11)

Summary

[demo](#)

- Learn to shade objects so their images appear three-dimensional
 - Introduce the types of light-material interactions
 - Build a simple reflection model---the Phong model--- that can be used with real time graphics hardware
 - Introduce modified Phong model
 - Consider computation of required vectors
-

Shading in OpenGL

Objectives

- Introduce the OpenGL shading functions
- Discuss polygonal shading
 - Flat
 - Smooth
 - Gouraud

[sphere.c](#)

Steps in OpenGL Shading

1. Enable shading and select model
 2. Specify normals
 3. Specify material properties
 4. Specify lights
-

Normals

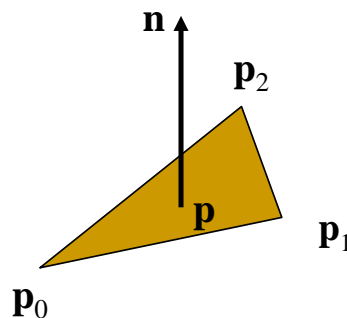
- In OpenGL the normal vector is part of the state
- Set by `glNormal*()`
 - `glNormal3f(x, y, z);`
 - `glNormal3fv(p);`
- Usually we want to set the normal to have unit length so cosine calculations are correct
 - Length can be affected by transformations
 - Note that scaling does not preserved length
 - `glEnable(GL_NORMALIZE)` allows for autonormalization at a performance penalty

Normal for Triangle

$$\text{plane } \mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$$

$$\mathbf{n} = (\mathbf{p}_2 - \mathbf{p}_0) \times (\mathbf{p}_1 - \mathbf{p}_0)$$

$$\text{normalize } \mathbf{n} \leftarrow \mathbf{n} / |\mathbf{n}|$$



Note that right-hand rule determines outward face

Enabling Shading

- Shading calculations are enabled by
 - `glEnable(GL_LIGHTING)`
 - Once lighting is enabled, `glColor()` ignored
 - Must enable each light source individually
 - `glEnable(GL_LIGHTi)` $i=0,1,\dots$
 - Can choose light model parameters
 - `glLightModeli(parameter, GL_TRUE)`
 - `GL_LIGHT_MODEL_LOCAL_VIEWER` do not use simplifying distant viewer assumption in calculation
 - `GL_LIGHT_MODEL_TWO_SIDED` shades both sides of polygons independently
-

Steps in OpenGL Shading

1. Enable shading and select model
 2. Specify normals
 3. Specify material properties
 4. Specify lights
-

Defining a Point Light Source

- For each light source, we can set an RGBA for the diffuse, specular, and ambient components, and for the position

```
GL float diffuse0[]={1.0, 0.0, 0.0, 1.0};
GL float ambient0[]={1.0, 0.0, 0.0, 1.0};
GL float specular0[]={1.0, 0.0, 0.0, 1.0};
GLfloat light0_pos[]={1.0, 2.0, 3.0, 1.0};

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightv(GL_LIGHT0, GL_POSITION, light0_pos);
glLightv(GL_LIGHT0, GL_AMBIENT, ambient0);
glLightv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
glLightv(GL_LIGHT0, GL_SPECULAR, specular0);
```

$$I = k_d I_d | \mathbf{n} \cdot \mathbf{h} | + k_s I_s (\mathbf{n} \cdot \mathbf{h})^B + k_a I_a$$

Distance and Direction

- The source colors are specified in RGBA
- The position is given in homogeneous coordinates
 - If $w = 1.0$, we are specifying a finite location
 - If $w = 0.0$, we are specifying a parallel source with the given direction vector
- The coefficients in the distance terms are by default $a=1.0$ (constant terms), $b=c=0.0$ (linear and quadratic terms). Change by

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 1.0);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.0);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.0);
```

Distance term: add a factor of the form $\frac{1}{ad + bd + cd^2}$ to the diffuse and specular terms

Notes

- Presentation + project
 - Sample project
-

After Class

- Read Chapter 5
 - Work on HW3 and HW4
-