Shading

Course web page:

http://goo.gl/EB3aA

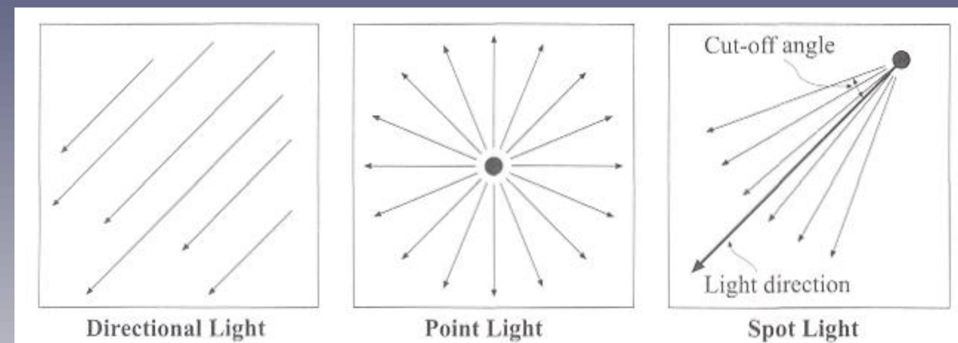
Outline

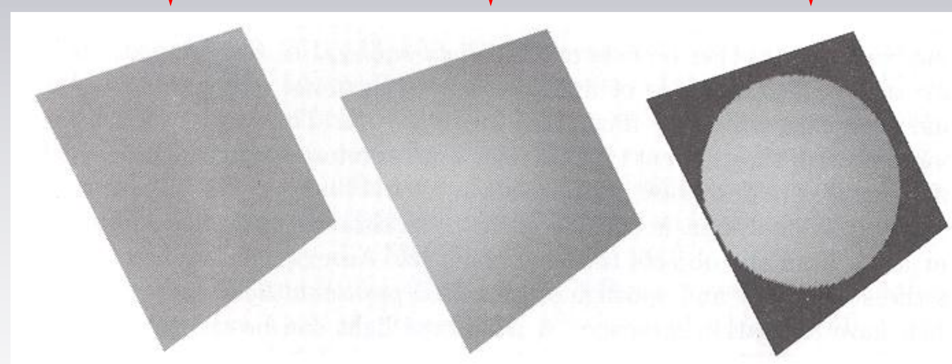
- Standard local model
  - Point light sources
  - BRDF = Ambient + diffuse + specular components
- Shading implementations
  - Gouraud shading
  - Phong shading
- OpenGL API

Light sources

- Properties
  - Intensity (total radiosity)
  - Color (intensity / wavelength)
- Geometry
  - **Point**: Shoots light in all directions
  - **Spotlight**: Angle-limited point source
  - **Directional**: Source distant enough that light rays are roughly parallel (e.g., like the sun relative to earth)
  - **Area**: Behaves like a continuous configuration of point sources inside, say, a polygon

Some light source types





from Akenine-Moller & Haines

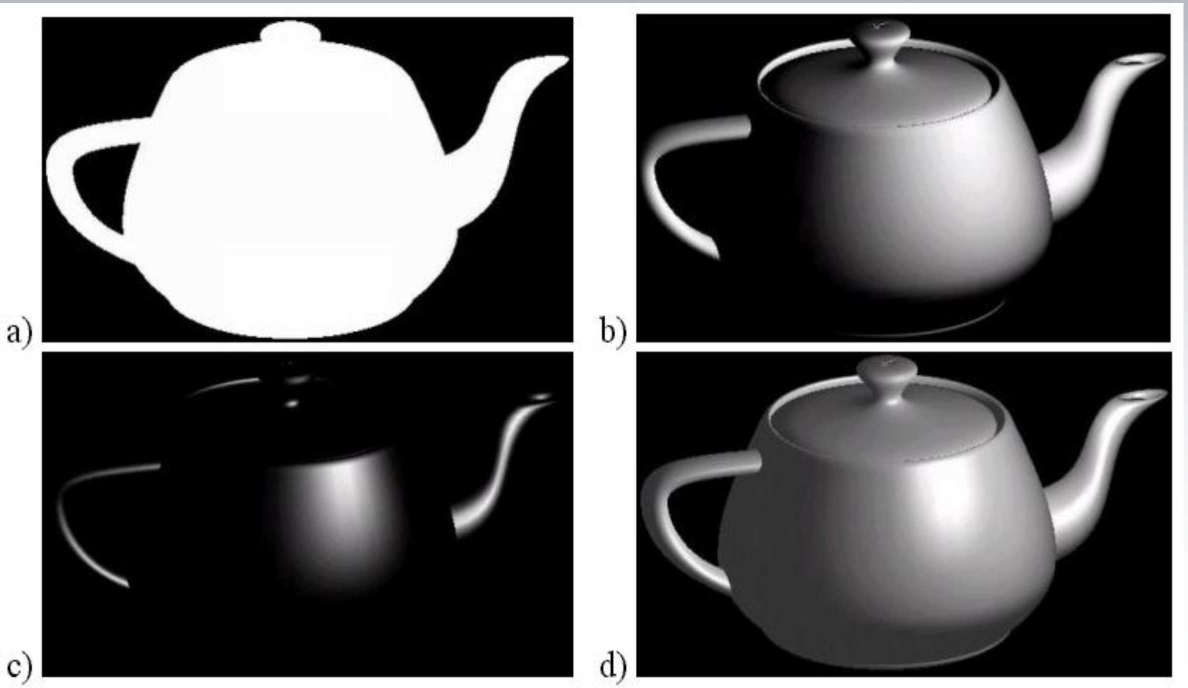
Light source types: Induced shading

Standard local model for graphics

- Final perceived brightness is a combination of **diffuse** and **specular** reflectance, plus an **ambient** term to approximate global lighting effects

Ambient

Diffuse



Specular

Total

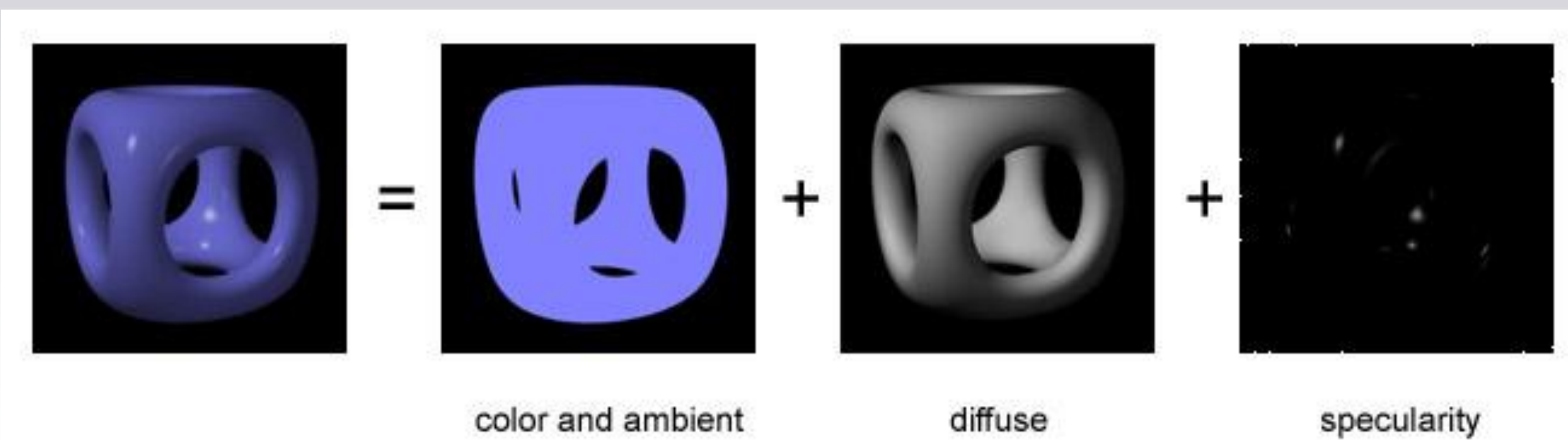
Reflectance equation: Total illumination

- For greater control of appearance, a different light radiance is typically specified in OpenGL for each type of reflectance

$$s_{diff}, s_{spec}, s_{amb}$$

- Actual light at a pixel is combination of three effects:

$$i_{total} = i_{amb} + i_{diff} + i_{spec}$$



from Wikipedia

Reflectance equation: Ambient component (Shirley 10.1.2)

- Light reflections, refractions off of other objects typically mean that light is coming from more directions than just sources
- Model this with **ambient** light, which guarantees that all scene objects get some minimum illumination

$$i_{amb} = m_{amb} s_n$$

Reflectance equation

- Radiance for a viewing direction given all incoming light (also called *rendering* equation in Shirley 20.2):

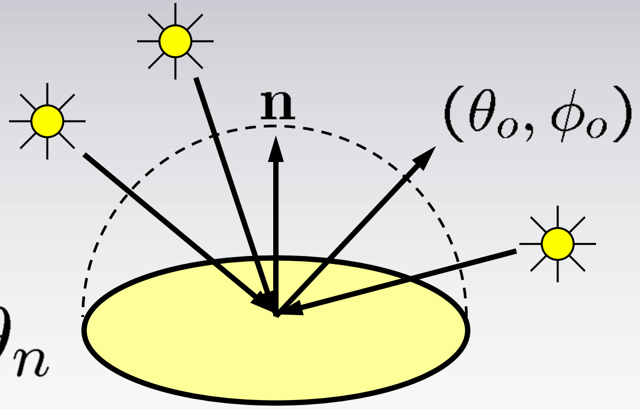
$$L_o(\mathbf{x}, \theta_o, \phi_o) = \int_\Omega f(\theta_o, \phi_o, \theta_i, \phi_i) L_i(\mathbf{x}, \theta_i, \phi_i) \cos\theta_i d\omega$$

- This is expensive to compute in general, so the standard local approach is approximation:
  - Approximate incoming light as **ambient** (whole hemisphere) + set of point light sources
  - Approximate BRDF of surface as combination of **diffuse** (matte) and **specular** (shiny) factors

Reflectance equation for *N* point sources: Lambertian surface material (Shirley 10.1.1)

- If the surface is Lambertian (diffuse), the BRDF is constant regardless of the viewing direction
- Call this the **diffuse material reflectance** m_{diff} and let radiance due to each light n be s_n
- Book uses c_r for m_{diff}, c_l for s_n
- Then we have:

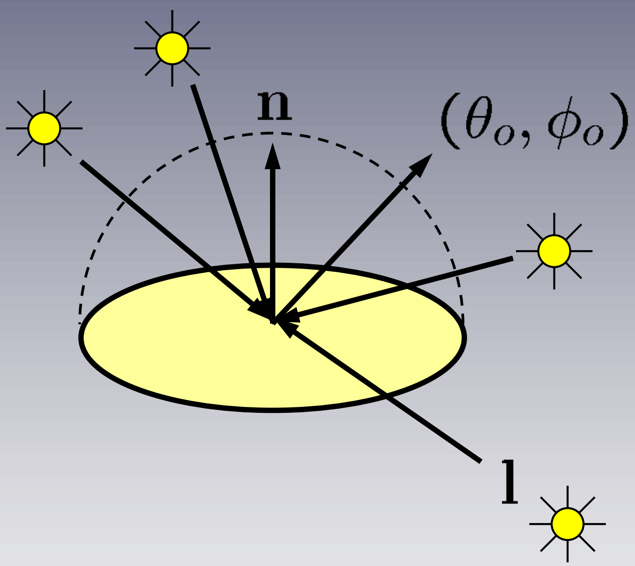
$$L_o(\mathbf{x}, \theta_o, \phi_o) = \sum_{n=1}^{N} m_{diff} \, s_n \cos\theta_n$$



With appropriate units, we can use this as the pixel brightness

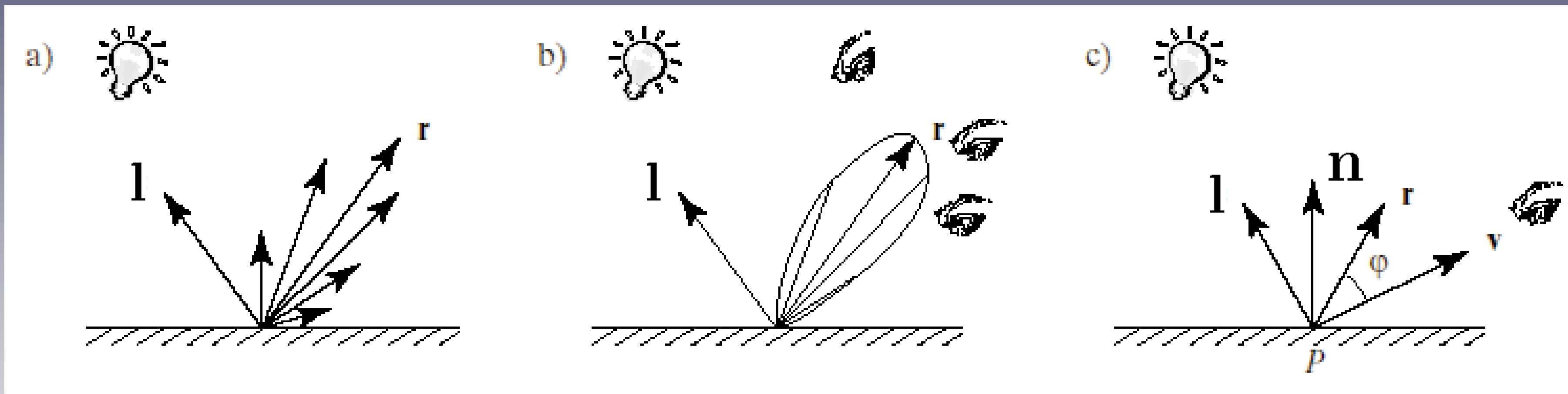
Reflectance equation for a single point source: Lambertian surface material

- If angle with light source is greater than 90 degrees, light source is **behind** surface and therefore doesn't illuminate it (directly)
- This corresponds to a negative cosine: $\cos\theta = \mathbf{n} \cdot \mathbf{l} < 0$
- So adjust the formula for one light:



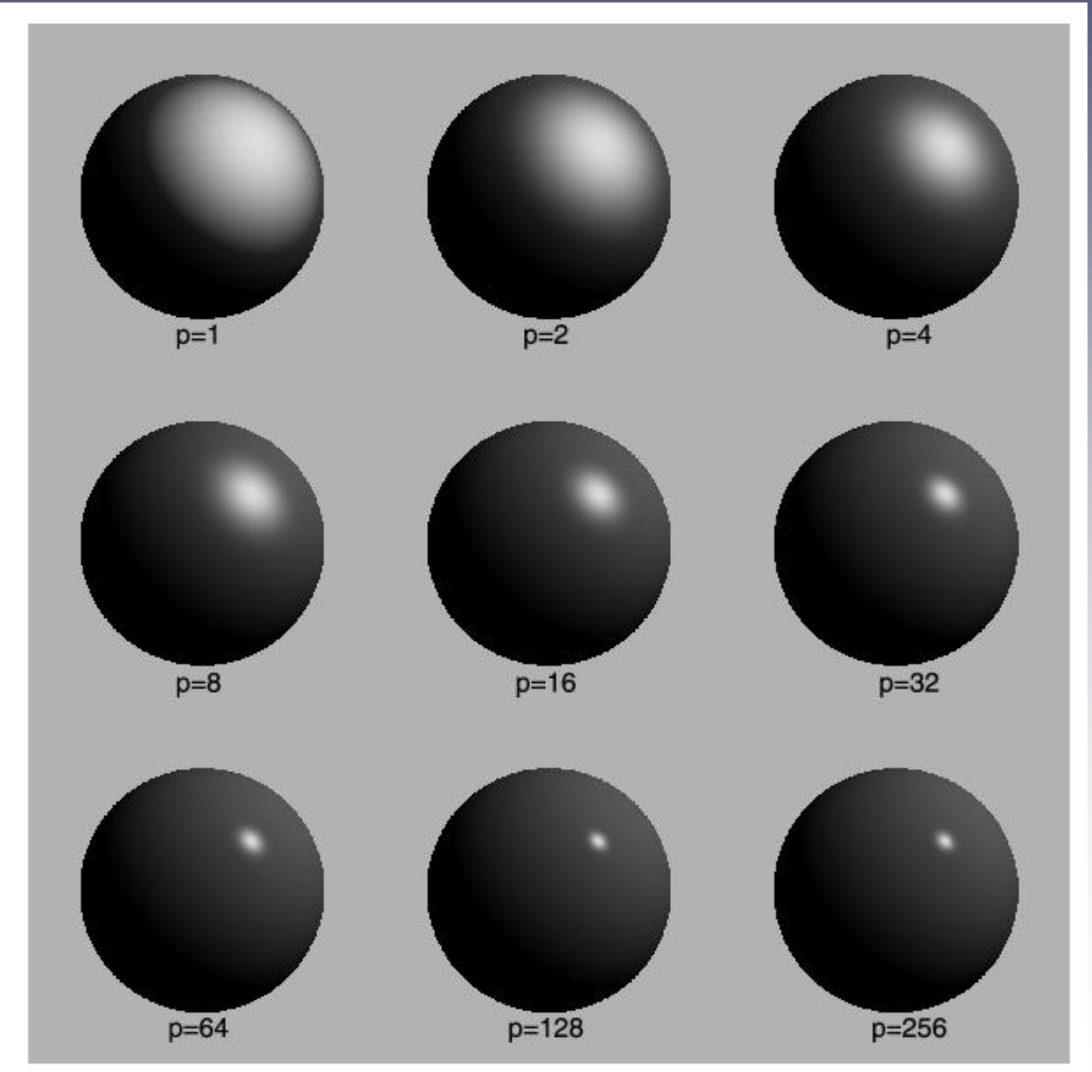
$$i_{diff} = \max(0, (\mathbf{n} \cdot \mathbf{l}) m_{diff} s_n)$$

Reflectance equation for a point source: Specular surface material



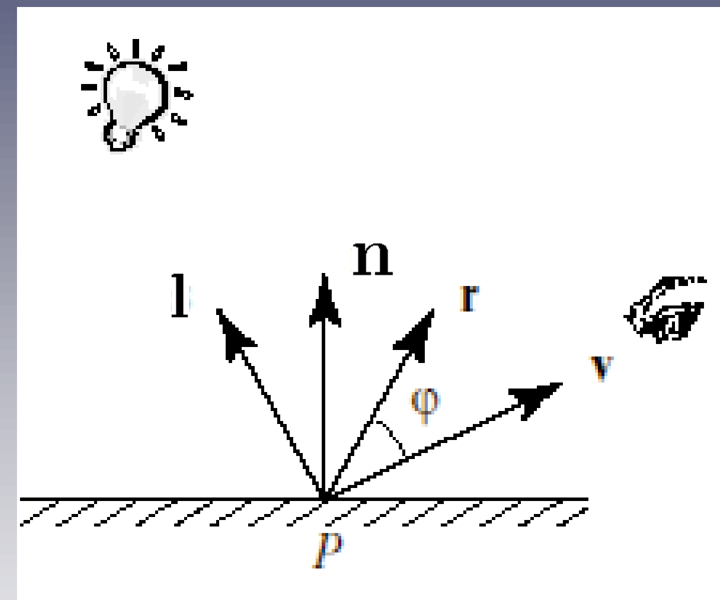
- Specular lobe: The further away the viewing direction **v** is from the reflection direction **r**, the less light is visible
- The shinier (more specular) the material, the more quickly the **highlight** diminishes

Effects of specular exponent value



Reflectance equation for a point source: Phong lighting equation (Shirley 10.2.1)

- (This is **different** from Phong shading later in lecture)
- Approximate these intuitions with the quantity $(\mathbf{r} \cdot \mathbf{v})^{m_{shine}}$
  - The larger the angle, the smaller $\mathbf{r} \cdot \mathbf{v}$ (both unit vectors)
  - The larger the exponent, the faster the quantity gets small (because $\mathbf{r} \cdot \mathbf{v}$ is ≤ 1)
  - Book uses **e** for **v,** *p* for m_{shine}



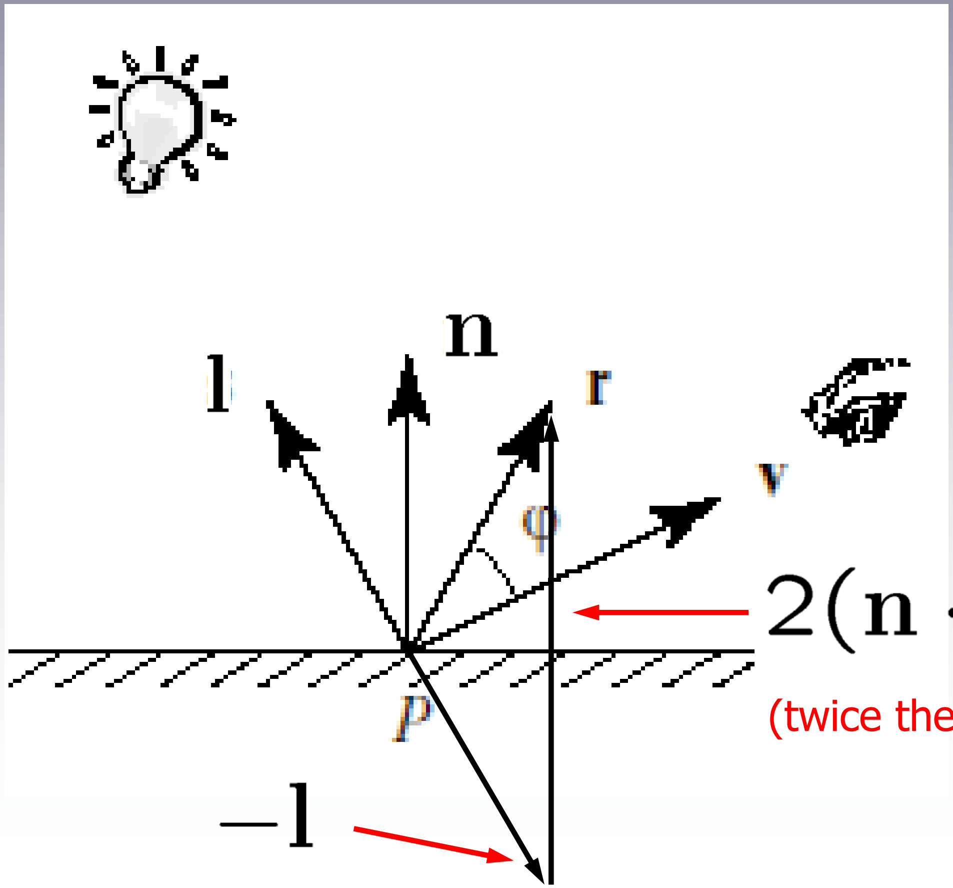
- So we can make the following formula for the specular intensity due to a single light source:

$$i_{spec} = \max(0, \mathbf{r} \cdot \mathbf{v})^{m_{shine}} m_{spec} s_n$$

Reflectance equation for a single source: Calculating the reflection direction

- Can calculate **r** from **n**, **l** via:

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

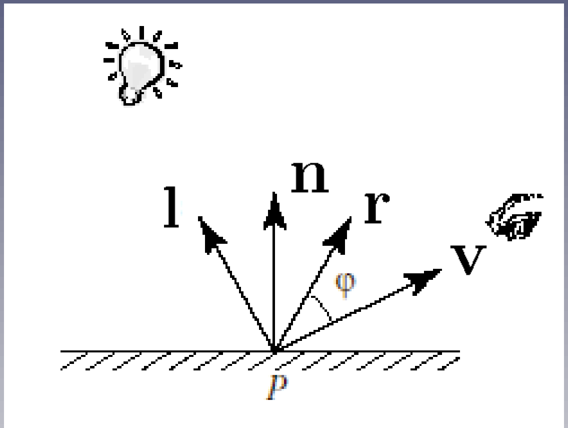


$2(\mathbf{n} \cdot \mathbf{l})\mathbf{n}$

(twice the projection of **l** onto **n**)

Lighting a point

- Let **c** = (r, g, b) be **perceived** material color (called **i** on previous slides), **s**(l) be color of light *l*
- Sum over all lights *l* for each color channel (clamp overflow to [0, 1]):



from Hill

$$\mathbf{c}_{total} = \sum_l \mathbf{c}_{amb}(l) + \mathbf{c}_{diff}(l) + \mathbf{c}_{spec}(l)$$

$$\mathbf{c}_{amb}(l) = \mathbf{m}_{amb} \otimes \mathbf{s}_{amb}(l)$$

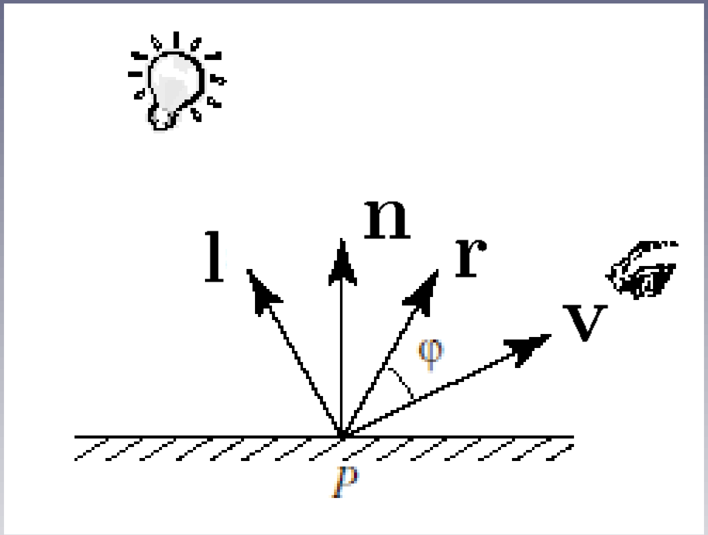
componentwise vector product

$$\mathbf{c}_{diff}(l) = \max(0, \mathbf{n} \cdot \mathbf{l}(l))\mathbf{m}_{diff} \otimes \mathbf{s}_{diff}(l)$$

$$\mathbf{c}_{spec}(l) = \max(0, \mathbf{v} \cdot \mathbf{r}(l))^{shine}\mathbf{m}_{spec} \otimes \mathbf{s}_{spec}(l)$$

Lighting details

- What do we need to "light" a piece of surface?
  - Normal **n** = (nx, ny, nz)
  - Light direction **l** = (lx, ly, lz) (computed from surface & light positions)
  - View direction **v** = (vx, vy, vz) (computed from surface and eye positions)
  - Surface properties & light colors
    - Specular, diffuse, ambient



from Hill

- Lighting's place in the pipeline
  - Must do before perspective transformation (in world or camera coordinates) because of nonlinear distortion of z

Compute lighting at each pixel?

- Most accurate approach: Compute component illumination at each pixel (aka surface patch) with individual light directions, viewing directions, etc.
- But this is expensive...
- Approximation: Compute quantities at vertices of primitive and linearly interpolate to interior pixels
  - Like DDA or midpoint line-drawing, idea is to just increment some value(s) for each new pixel to save per-pixel calculations

Linear Interpolation (aka `lerp`)

- Parametric definition of a line segment:

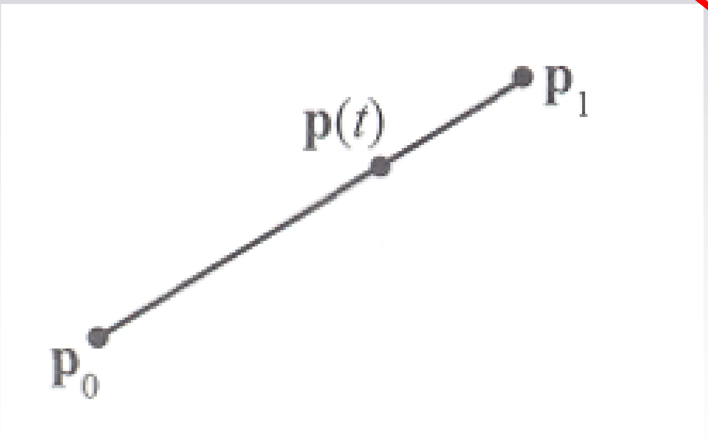
$$\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0),\ \text{where } t \text{ in } [0, 1]$$

$$= \mathbf{p}_0 - t\mathbf{p}_0 + t\mathbf{p}_1$$

$$= (1 - t)\mathbf{p}_0 + t\mathbf{p}_1$$

`= lerp(p0, p1, t)`

like a "blend" of the two endpoints

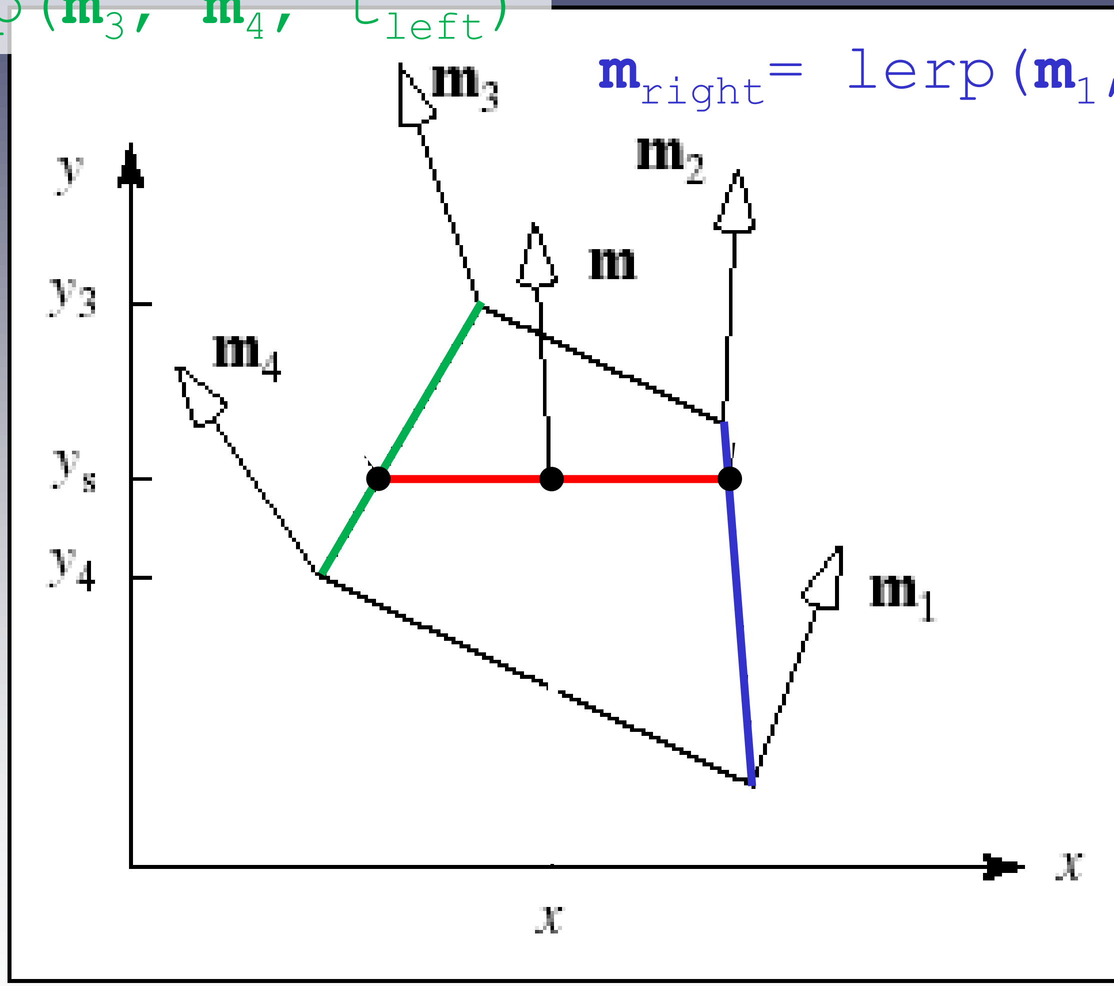


from Akenine-Möller & Haines

Bilinear interpolation for LIGHTING

$\mathbf{m}_{left}$ = lerp($\mathbf{m}_3$, $\mathbf{m}_4$, t_{left})

$\mathbf{m}_{right}$ = lerp($\mathbf{m}_1$, $\mathbf{m}_2$, t_{right})



$\mathbf{m}$ = lerp($\mathbf{m}_{left}$, $\mathbf{m}_{right}$, t)

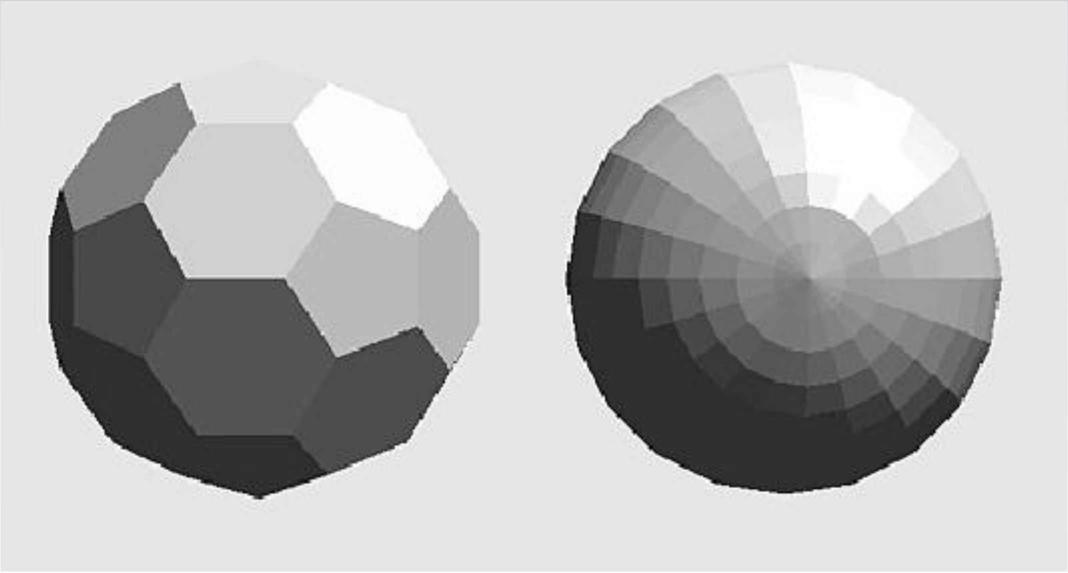
Shading methods: Notes

- **Flat**: Compute $\mathbf{c}_{total}$ at one vertex per polygon, use same value for every pixel in polygon
  - **Infinite** viewpoint **v**/light **l**: Same value for all vertices in scene
- **Gouraud**: Compute **different** $\mathbf{c}_{total}$ at each vertex of a polygon, **interpolate** to interior pixels
  - Different vertex colors because **l**, **v**, **r** , and possibly **n** are different at each vertex
- **Phong**: Interpolate **normals** from polygon vertices to interior, recompute $\mathbf{c}_{total}$ at each pixel
  - Interpolation changes length of normals, so be sure to normalize them to unit length before computing $\mathbf{c}_{total}$

Depth information needed for Z-buffering is just **one more** parameter in Gouraud-style vertex interpolation

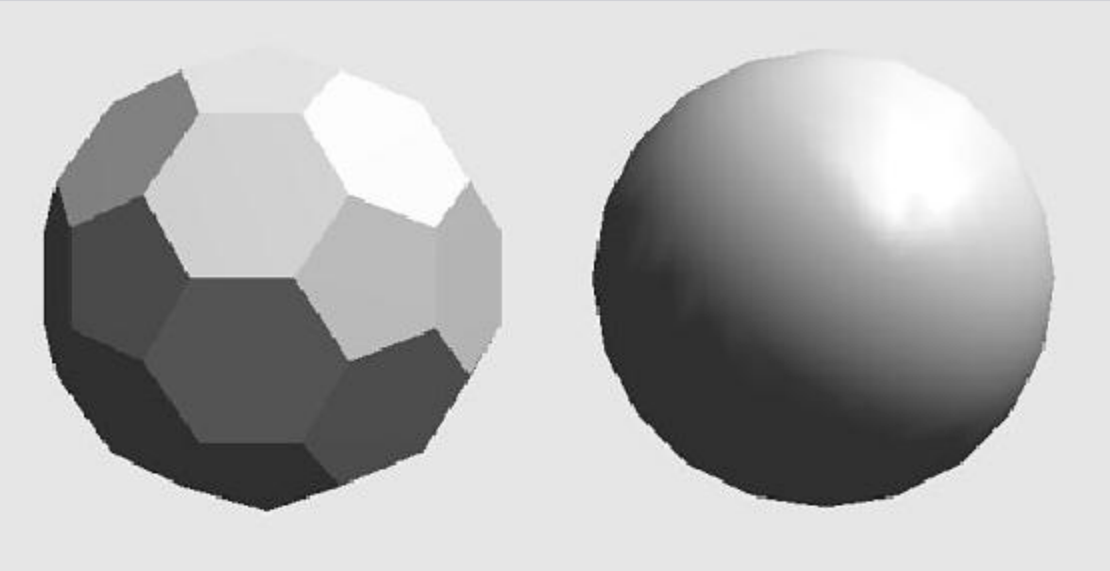
Flat Shading

- Normal same for all polygon vertices so same color used for every pixel in polygon
- Good approximation for directional lights
  - Light source direction is same for every point on facet
- OpenGL: **`glShadeModel(GL_FLAT)`**



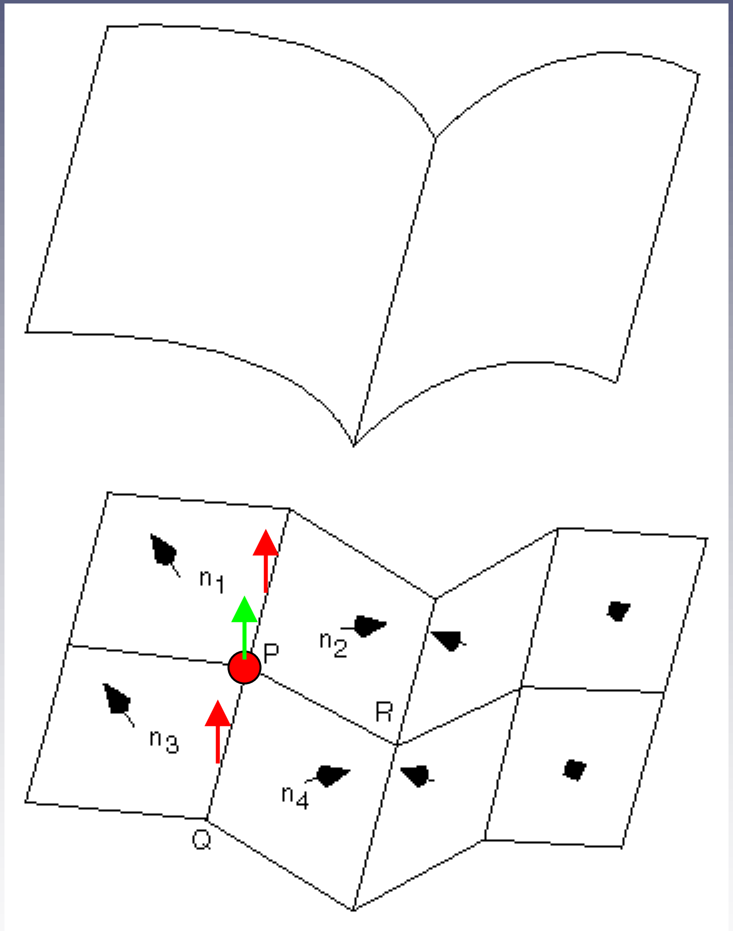
Gouraud Shading

- Colors computed at polygon vertices and linearly interpolated (like Z-buffer algorithm)
  - Poor handling of specularities because of interpolation
  - Slower than flat shading
- OpenGL: **`glShadeModel(GL_SMOOTH)`**



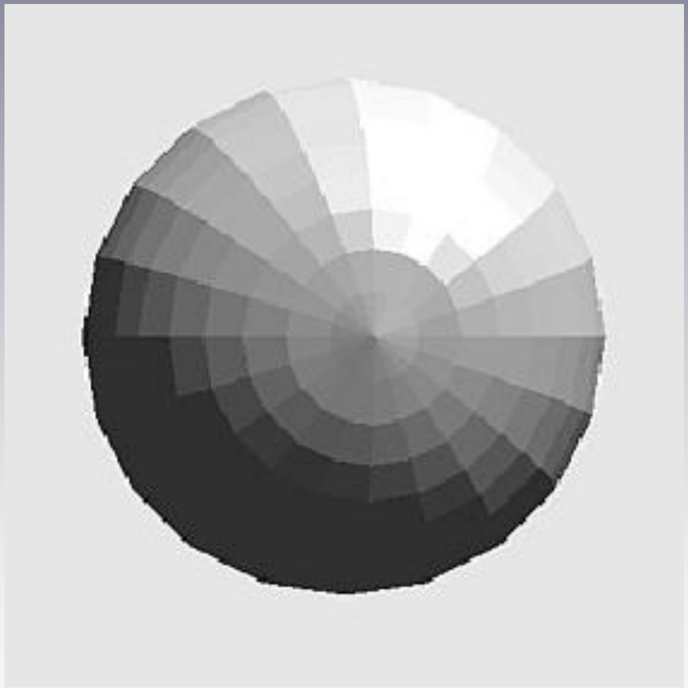
OpenGL: Normals

- Can compute normals of polygons using cross product formula
- But how to handle shared edges and vertices?
  - Average all normals at shared vertices for smooth shading
  - Don't average where you want to preserve sharp creases/folds (flat shading)

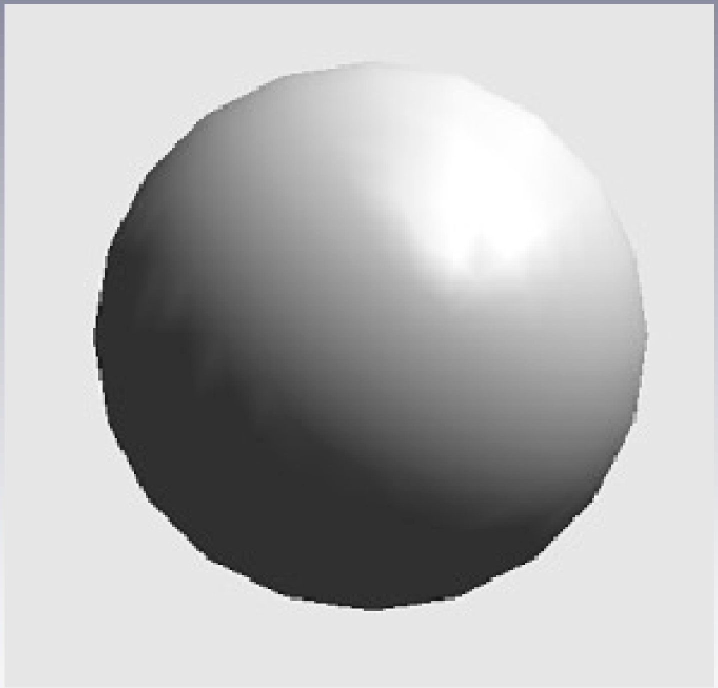


From Red book

Example: Vertex normal handling

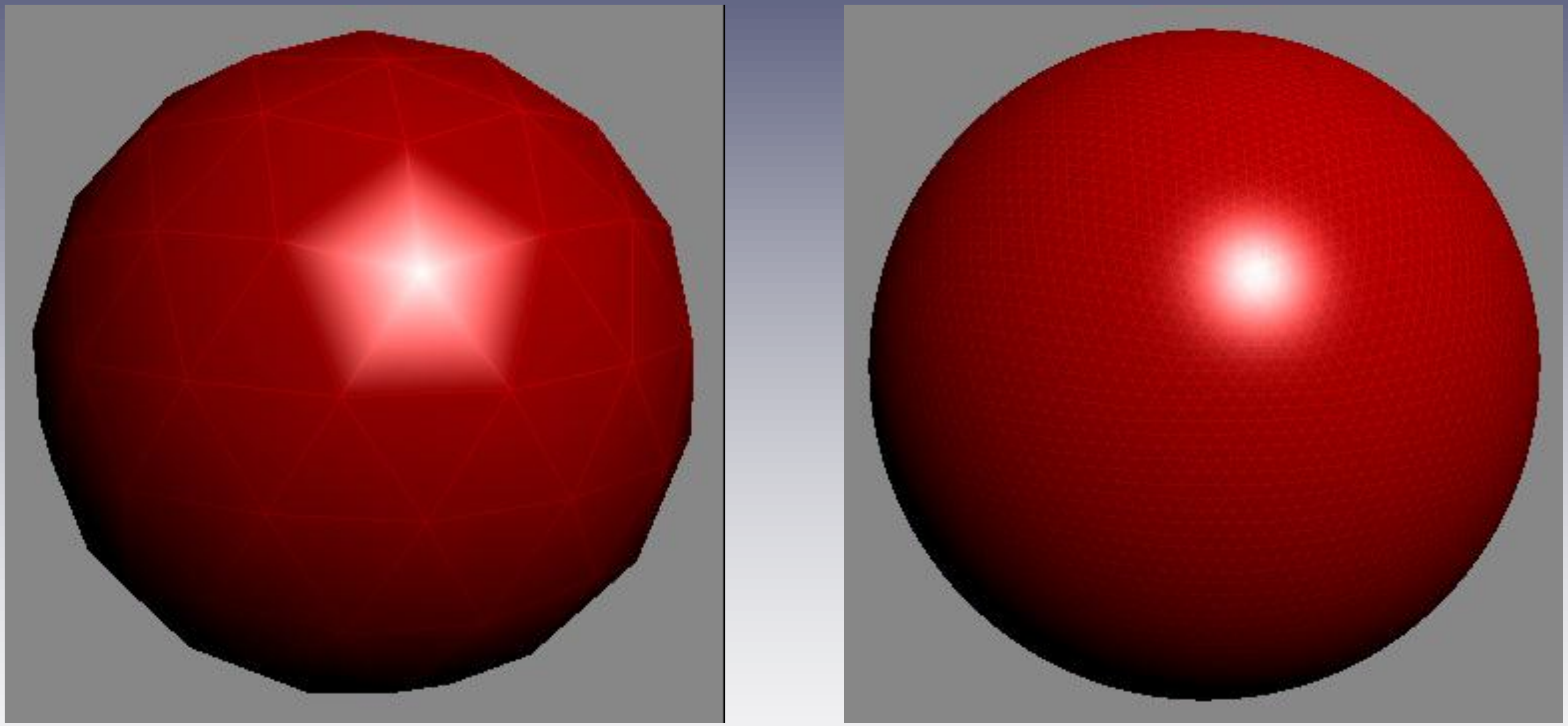


Sharp edges maintained
(no averaging)



Adjacent vertices averaged

Gouraud shading artefacts



Issues can typically be resolved with more detailed geometry

Phong Shading

- Normal vectors interpolated between vertices, but color computed at each pixel inside polygon
  - Better handling of specularities
  - Slower than Gouraud shading
- Not built into OpenGL
- On modern graphics cards, this would be called a fragment or pixel shader (vs. a vertex shader)

OpenGL lighting steps

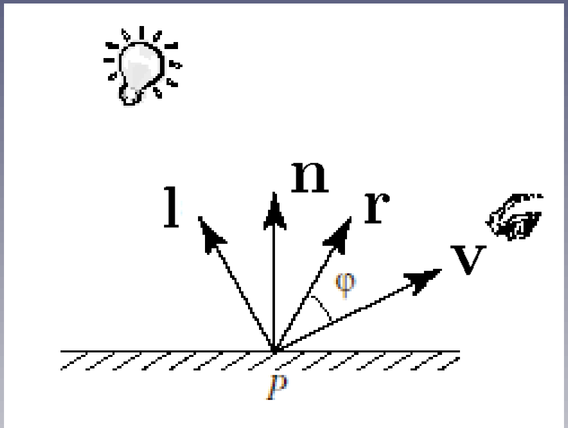
1. Attach normals to vertices with **`glNormal()`**
2. Place lights in scene, set properties
3. Choose lighting model with **`glLightModel()`**
   - GL_LIGHT_MODEL_AMBIENT
   - GL_LIGHT_MODEL_LOCAL_VIEWER
   - GL_LIGHT_MODEL_TWO_SIDE
4. Define material properties
5. Enable lighting and individual lights

OpenGL: Defining Lights

- **`glLight(light, pname, param)`**
  - **`light`**: Which light (**`GL_LIGHT0`**, **`GL_LIGHT1`**, etc.)
  - **`pname`**: Which characteristic
    - Position
    - Specular, diffuse, ambient color (these are the **s**'s from earlier formulas)
    - Spotlight direction, cutoff angle, etc.
    - Distance attenuation
  - **`param`**: Value(s) of **`pname`**
- Transformed by modelview matrix like geometric primitives

Lighting a point

- Let **c** = (r, g, b) be **perceived** material color (called **i** on previous slides), **s**(l) be color of light *l*
- Sum over all lights *l* for each color channel (clamp overflow to [0, 1]):



from Hill

$$\mathbf{c}_{total} = \sum_{l} \mathbf{c}_{amb}(l) + \mathbf{c}_{diff}(l) + \mathbf{c}_{spec}(l)$$

$$\mathbf{c}_{amb}(l) = \mathbf{m}_{amb} \otimes \mathbf{s}_{amb}(l)$$

$$\mathbf{c}_{diff}(l) = \max(0, \mathbf{n} \cdot \mathbf{l}(l))\mathbf{m}_{diff} \otimes \mathbf{s}_{diff}(l)$$

$$\mathbf{c}_{spec}(l) = \max(0, \mathbf{v} \cdot \mathbf{r}(l))^{shine}\mathbf{m}_{spec} \otimes \mathbf{s}_{spec}(l)$$

OpenGL lights: Example (from Red book)

```
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
```

Means infinite distance away
= Directional light

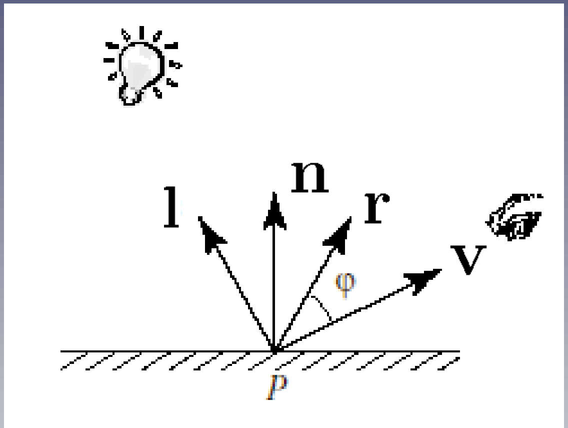
```
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

OpenGL: Material properties

- Applies to subsequent vertices:
  `glMaterial(face, pname, param)`
  - **`face`**: Which face(s) to apply properties to (**`GL_FRONT`**, **`GL_BACK`**, or **`GL_FRONT_AND_BACK`**)
  - **`pname`**: Which characteristic (these are the **m**'s on the "Lighting a point" slide)
    - Ambient color
    - Diffuse color
    - Specular color
    - Shininess
  - **`param`**: Value(s) of **`pname`**

Lighting a point

- Let **c** = (r, g, b) be **perceived** material color (called **i** on previous slides), **s**(l) be color of light *l*
- Sum over all lights *l* for each color channel (clamp overflow to [0, 1]):



from Hill

$$\mathbf{c}_{total} = \sum_l \mathbf{c}_{amb}(l) + \mathbf{c}_{diff}(l) + \mathbf{c}_{spec}(l)$$

$$\mathbf{c}_{amb}(l) = \mathbf{m}_{amb} \otimes \mathbf{s}_{amb}(l)$$

$$\mathbf{c}_{diff}(l) = \mathsf{max}(0, \mathbf{n} \cdot \mathbf{l}(l))\mathbf{m}_{diff} \otimes \mathbf{s}_{diff}(l)$$

$$\mathbf{c}_{spec}(l) = \mathsf{max}(0, \mathbf{v} \cdot \mathbf{r}(l))^{shine}\mathbf{m}_{spec} \otimes \mathbf{s}_{spec}(l)$$

OpenGL materials: Example

```
GLfloat no_mat[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat mat_ambient[] = { 0.7, 0.7, 0.7, 1.0 };
GLfloat mat_diffuse[] = { 0.1, 0.0, 1.0, 1.0 };
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat no_shininess[] = { 0.0 };
GLfloat low_shininess[] = { 5.0 };
GLfloat high_shininess[] = { 100.0 };

glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);

glutSolidSphere(radius, slices, stacks);
```

See Robins' tutor program **lightmaterial**