Shading

Shading

- Process of using an equation (a model) to compute the outgoing radiance along a view ray, based on material properties and light sources
- Radiance: density of light flow per area and per incoming direction

Blinn-Phong model

- **Ambient light model** - illuminates all surfaces equally.
- **Diffuse light model** - reflected intensity is independent of the viewing direction but does depend on the surface orientation with regards to the light source.
- **Specular light model** - a light shining on a specular surface causes a bright spot. Specular reflectance is view-dependent.

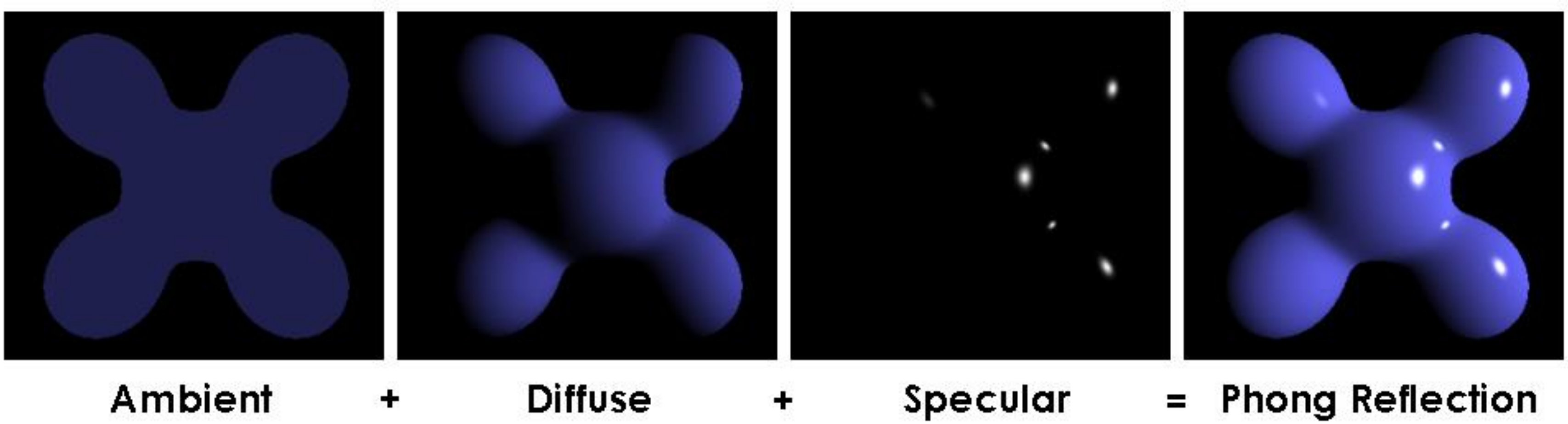


Image credit: Wikipedia

Ambient light

Simple illumination models means secondary light sources are ignored.

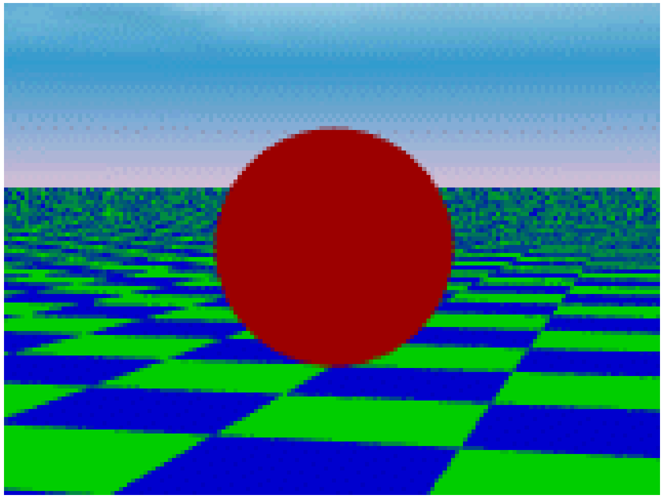
Objects not directly illuminated appear black.

Not very *realistic*.

Introduction of a non-directional light called **ambient light** $\mathbf{I_a}$

The resulting reflected light is constant for each surface and depends on the **ambient-reflection** coefficient $\mathbf{k_a}$

$$I_a = k_a I$$



Notation

In the previous equation ***k I***, both **k** and **I** are RGBA *vectors*.

k is a material property (ambient, diffuse, …).

I the light intensity property.

Each component corresponds to the intensity of red, green, blue and alpha. This number is in [0, 1].

The multiplication is *component-wise*.

Example

Suppose:

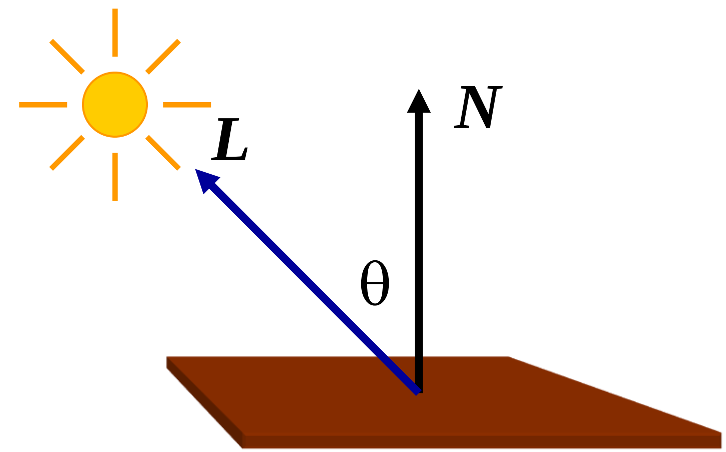
- A white light: **I** = [1, 1, 1, 1]
- A red object: **k** = [1, 0, 0, 1]

Then,

$$\boldsymbol{k}\,\boldsymbol{I} = [1 * 1, 0 * 1, 0 * 1, 1 * 1] = [1,0,0,1]$$

Diffuse reflection

Ideal diffuse reflection
(Lambert reflection)

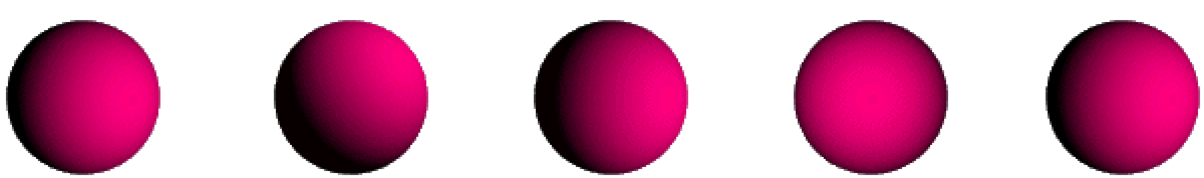


$$\boldsymbol{I}_d = \boldsymbol{k}_d \boldsymbol{I}\ \overline{\cos\theta} = \boldsymbol{k}_d \boldsymbol{I} \max\left(\vec{N}.\vec{L}, 0\right)$$

k_d - **diffuse-reflection** coefficient

$\vec{N}$: Unit normal vector to the surface
$\vec{L}$: Unit vector from current point to the light source

A sphere seen at different lighting angles



Surface normal

A normal is a vector orthogonal to all tangents at a given point

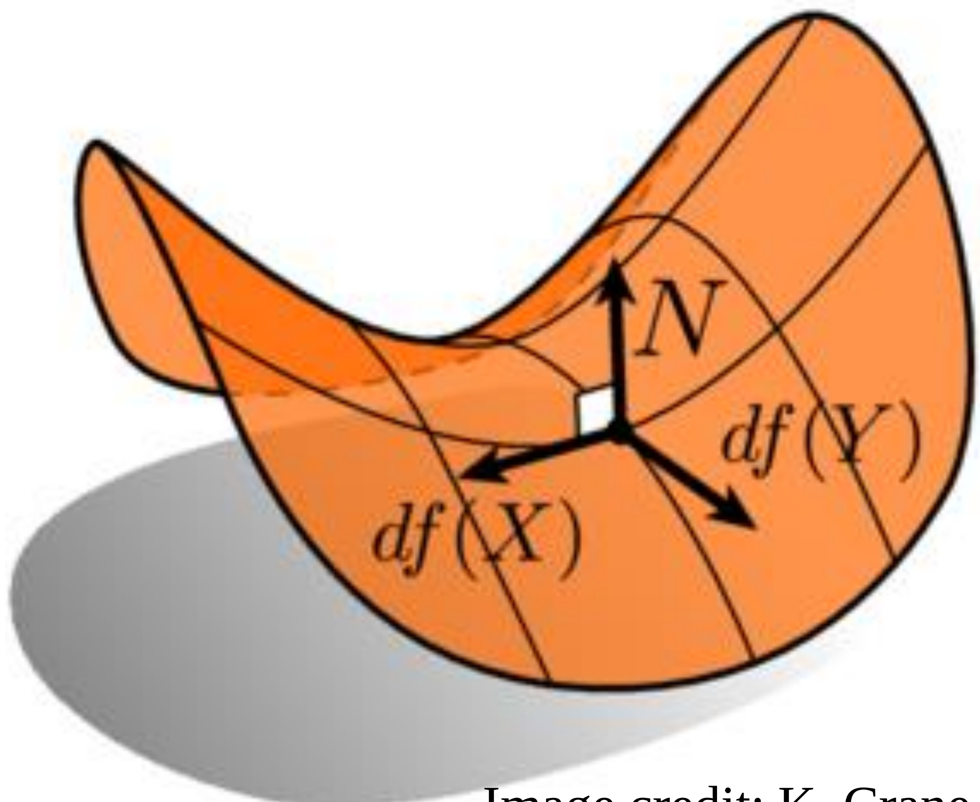
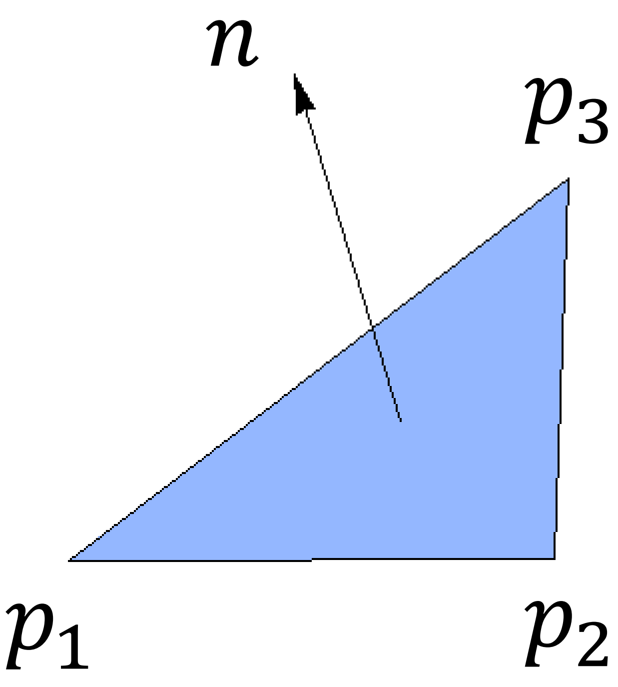


Image credit: K. Crane

Triangle normal



$$n = \frac{v_1 \times v_2}{\|v_1 \times v_2\|}$$

$$v_1 = p_2 - p_1$$

$$v_2 = p_3 - p_1$$

Triangle normal

```
void cross(float* u, float* v, float* w) {
 w[0] = u[1] * v[2] - u[2] * v[1];
 w[1] = u[2] * v[0] - u[0] * v[2];
 w[2] = u[0] * v[1] - u[1] * v[0];
}

void normalize(float* u) {
 float umag = sqrt(u[0]*u[0]+u[1]*u[1]+u[2]*u[2]);
 if (fabs(umag) < 1e-5f) umag = 1.f;
 u[0] = u[0] / umag; u[1] = u[1] / umag; u[2] = u[2] / umag;
}

void sub(float* p1, float* p2, float* v21) {
 v21[0] = p1[0] - p2[0];
 v21[1] = p1[1] - p2[1];
 v21[2] = p1[2] - p2[2];
}
```

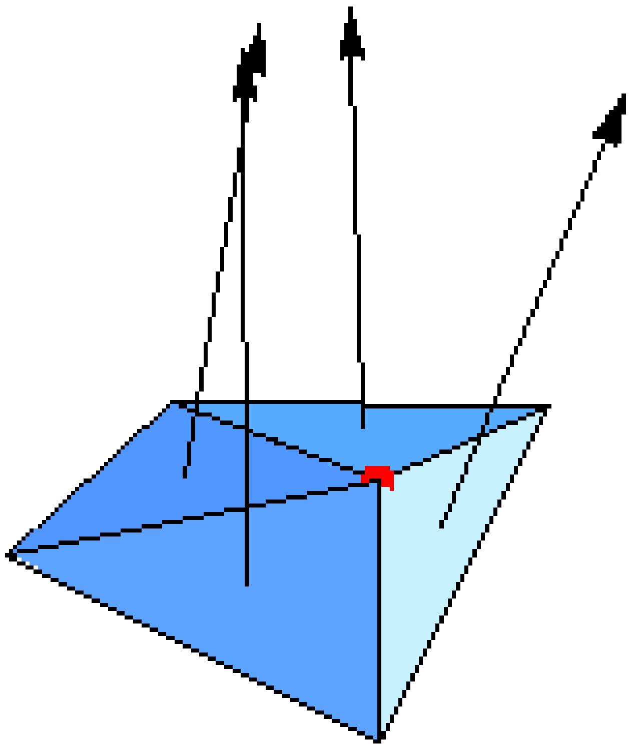
Triangle normal

```
void computeNormal(float* p1, float* p2,
                   float* p3, float* normal)
{
 float v1[3]; sub(p2, p1, v1);
 float v2[3]; sub(p3, p1, v2);
 cross(v1, v2, normal);
 normalize(normal);
}
```

Specify the normals to OpenGL

```
void
drawTriangle(float* p0, float* p1, float* p2,
             float* normal)
{
 glBegin(GL_TRIANGLES);
 glNormal3fv(normal);
 glVertex3fv(p0);
 glVertex3fv(p1);
 glVertex3fv(p2);
 glEnd();
}
```

Vertex normal



To compute the normal at the red vertex:

1. Compute the normal at each adjacent face
2. Compute the weighted sum of these normal

Possible weights:

- 1
- Triangle area
- …

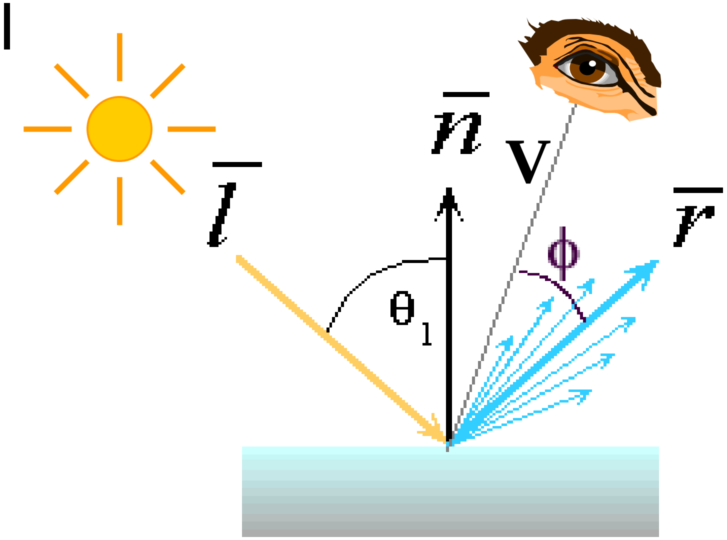
Specify (per vertex) normal to OpenGL

```
void
drawTriangle(float* p0, float* p1, float* p2,
             float* n0, float* n1, float* n2)
{
 glBegin(GL_TRIANGLES);
 glNormal3fv(n0);
 glVertex3fv(p0);
 glNormal3fv(n1);
 glVertex3fv(p1);
 glNormal3fv(n2);
 glVertex3fv(p2);
 glEnd();
}
```

Specular reflection

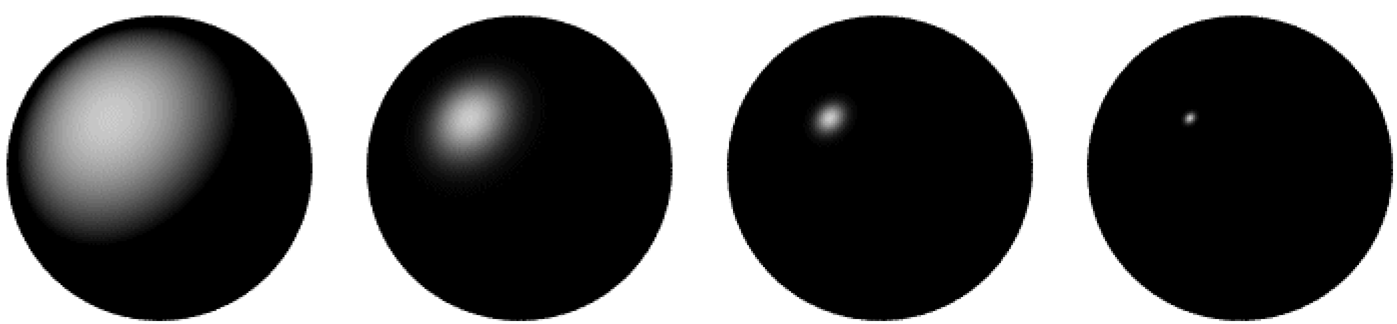
Phong specular-reflection model

$$I_s = k_s I(\overline{\cos\phi})^m$$
$$= k_s I \max(\vec{V}.\vec{R}, 0)^m$$



k_s - **specular** coefficient

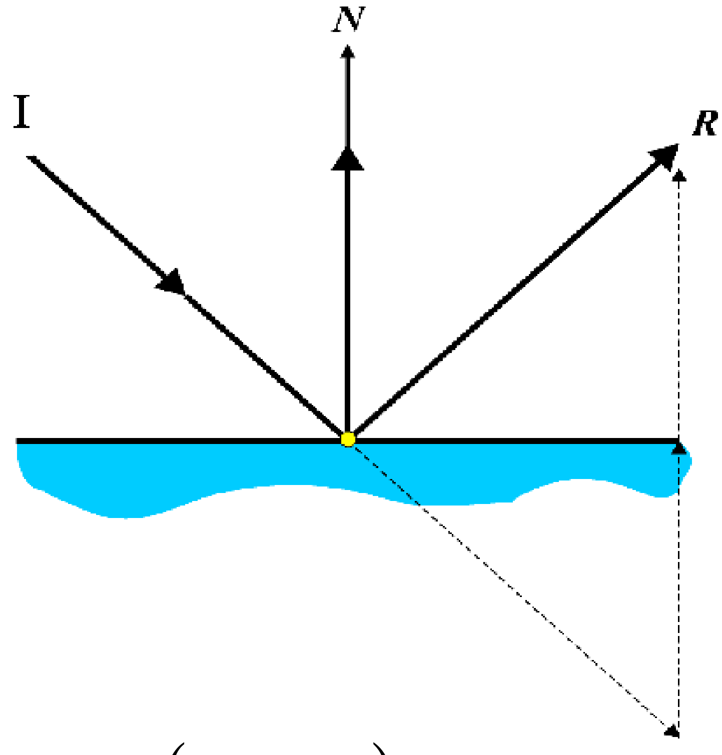
A sphere with different shininess coefficient (m)



Computation of the reflected vector

I: Incident vector
R: Reflected vector (pure reflection)

If both I and N are unit, then so is R



$$R = I + 2N(-I \cdot N)$$
$$= I - 2N(I \cdot N)$$

Half-vector variant (Blinn)

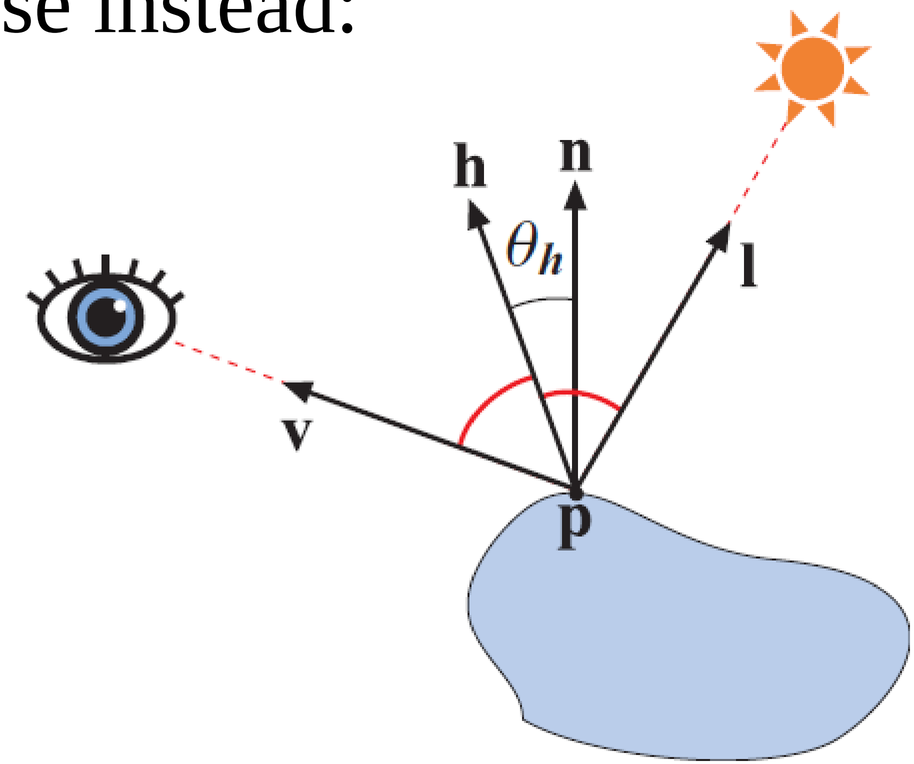
For the specular term use instead:

$$\boldsymbol{I_s} = \boldsymbol{k_s I}(\overline{\cos\theta_h})^{\boldsymbol{m}}$$

$$= \boldsymbol{k_s I}\max\left(\vec{\boldsymbol{h}}.\vec{\boldsymbol{n}}, \boldsymbol{0}\right)^{m}$$

where

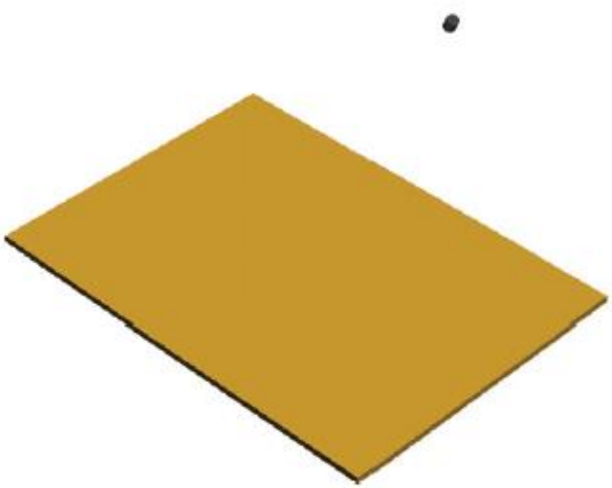
$$\vec{\boldsymbol{h}} = \frac{\vec{\boldsymbol{v}} + \vec{\boldsymbol{l}}}{\|\vec{\boldsymbol{v}} + \vec{\boldsymbol{l}}\|}$$



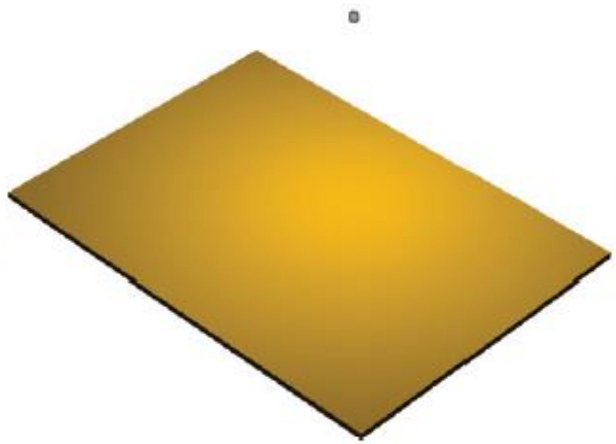
Real-time rendering, Haines et al.

Light sources

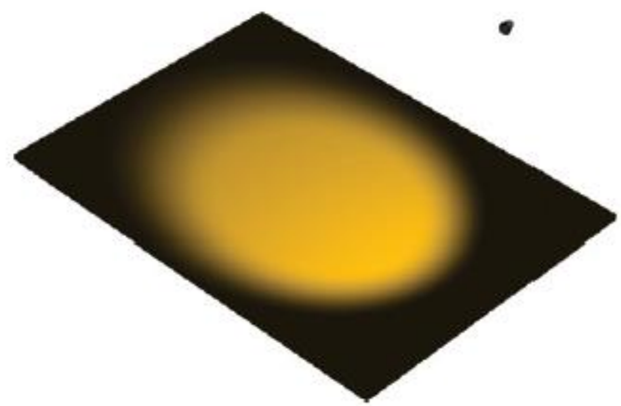
- Directional light
- Point-light
- Spot-light



Directional



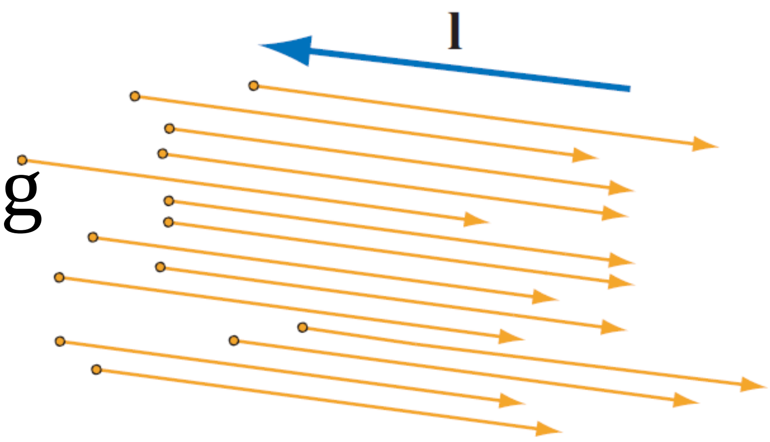
Point-light



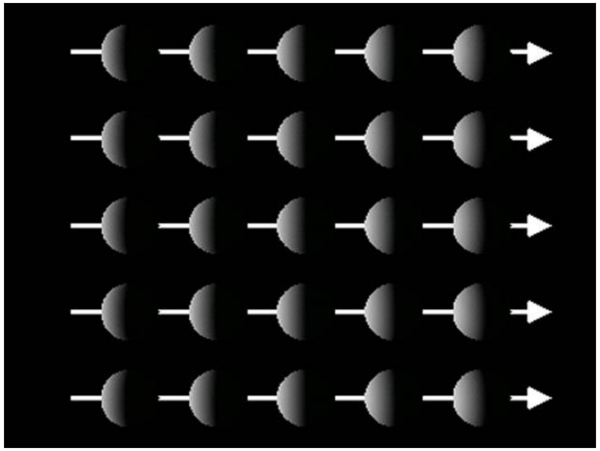
Spot-light

Real-time rendering, Haines et al.

Directional light

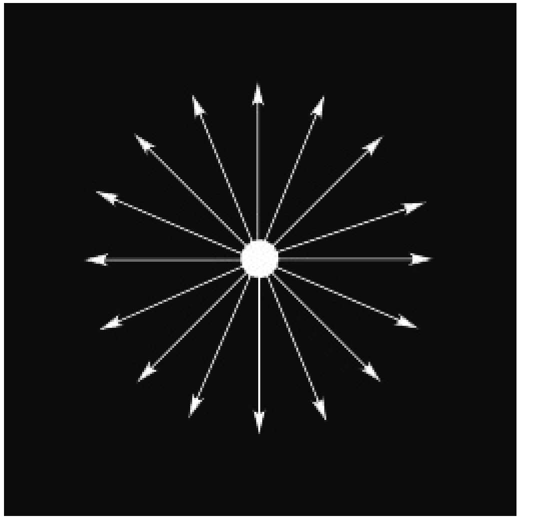


- Rays of light, parallel, coming from a source infinitely far away
- Used to model light source like the sun

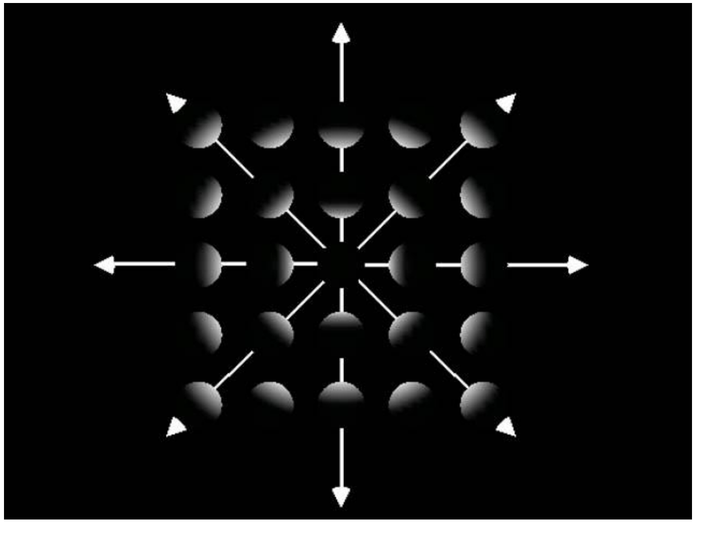


Point-light

- Light emit from a point in all directions
- Intensity decreases with the distance d from the source



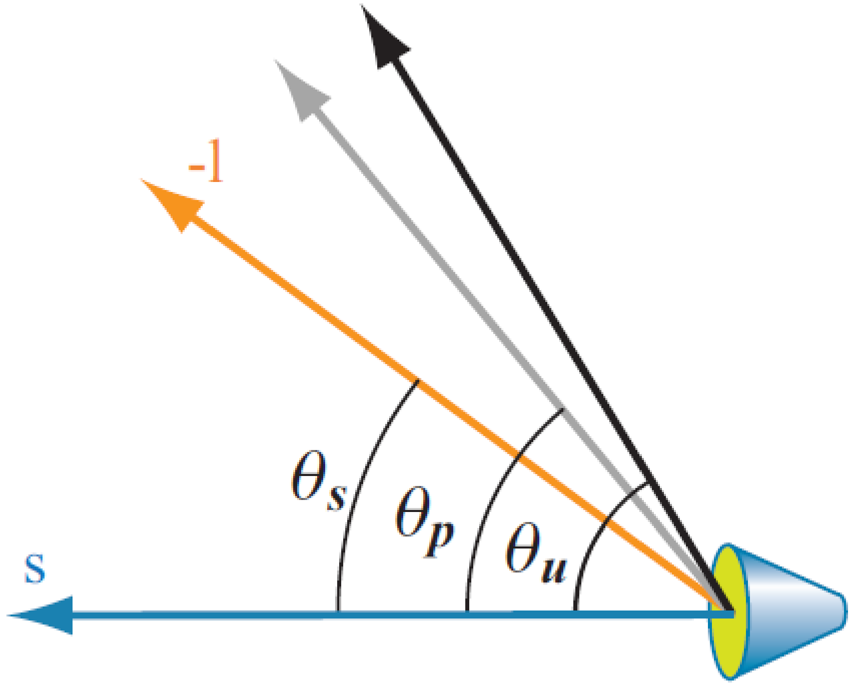
$$f_{att}(d) = \frac{1}{a_0 + a_1 d + a_2 d^2}$$



Spot-light

- Directional variance in the intensity

$$I_L(l) = \begin{cases} I_{Lmax}(\cos\theta_s)^m, & \theta_s \le \theta_u \\ 0, & \theta_s > \theta_u \end{cases}$$



Total lighting

Lighting is additive: We need to sum the contributions of the individual light sources

$$\sum_{lights} f_{att}(k_a I_{la} + k_d I_{ld} \max(N.L, 0) + k_s I_{ls} \max(V.R, 0)^m)$$

Polygon rendering methods

Flat shading, single intensity is calculated for each triangle. Flat shading produces discontinuities between of intensity along common edges of adjacent triangles.
Gouraud shading, intensity calculated at vertices is interpolated across the triangle.
Phong shading, vertex normal is interpolated across the triangle, the intensity is calculated per fragment using the interpolated normal.

Flat Gouraud Phong

Real-time rendering, Haines et al.

OpenGL lighting model

- Fixed rendering pipeline of OpenGL uses a model very similar to Blinn-Phong with the addition of a global ambient term and an emissive term
- Emissive term: simulate glowing object (for example a light-bulb)

OpenGL: Light properties

glLight{if} (GLenum *light*, Glenum *pname*, TYPE *param*);
glLight{if}v(GLenum *light*, Glenum *pname*, TYPE **param*);

Specify parameters for the light with ID ***light***, which can be **GL_LIGHT0**,...,**GL_LIGHT7** (at least 8 lights are available)

pname	Default	Meaning
GL_AMBIENT	(0.0,0.0,0.0,1.0)	Ambient light intensity
GL_DIFFUSE	(1.0,1.0,1.0,1.0) for light 0 (0.0,0.0,0.0,1.0) for other	Diffuse light intensity
GL_SPECULAR	(1.0,1.0,1.0,1.0) for light 0 (0.0,0.0,0.0,1.0) for other	Specular light intensity
GL_POSITION	(0.0,0.0,0.0,1.0)	(x,y,z,w) position of light
GL_SPOT_DIRECTION	(0.0,0.0,0.0,1.0)	(x,y,z) direction of spotlight
GL_SPOT_EXPONENT	0.0	Spotlight exponent
GL_SPOT_CUTOFF	180.0	Spotlight cutoff angle
GL_CONSTANT_ATTENUATION	1.0	Constant attenuation factor
GL_LINEAR_ATTENUATION	0.0	Linear attenuation factor
GL_QUADRATIC_ATTENUATION	0.0	Quadratic attenuation factor

OpenGL: Directional light

Example: Create a directional light with direction (3, 3, 3)

```
GLfloat l_dir[] = {3.0f, 3.0f, 3.0f, 0.0f};
glLightfv(GL_LIGHT0, GL_POSITION, l_dir);
```

In homogeneous coordinates, $(x, y, z, 0)$ corresponds to the 3D vector (x, y, z).

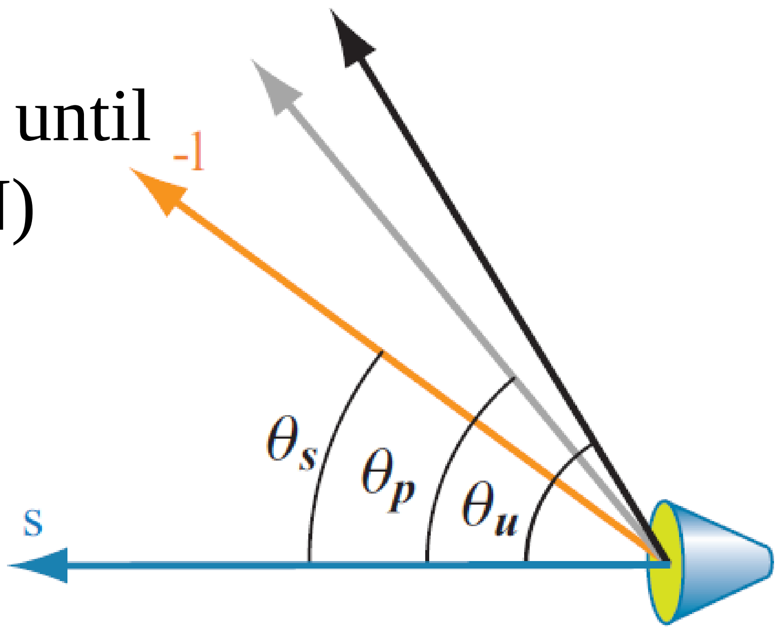
Spot-light

$$I_L(l) = \begin{cases} I_{Lmax}(\cos\theta_s)^m, & \theta_s \le \theta_u \\ 0, & \theta_s > \theta_u \end{cases}$$

$$\cos\theta_s = \boldsymbol{s}.(-\boldsymbol{l})$$

s: GL_SPOT_DIRECTION

l: light direction from surface point until spot-light position (GL_POSITION)



θ_u: GL_SPOT_CUTOFF

m: GL_SPOT_EXPONENT

Material properties

glMaterial{if} (GLenum *face*, Glenum *pname*, TYPE *param*);
glMaterial{if}v(GLenum *face*, Glenum *pname*, TYPE **parm*);

Specifies current material property for use in lighting calculations.

`Face = GL_FRONT, GL_BACK` or `GL_FRONT_AND_BACK`

pname	Default	Meaning
GL_AMBIENT	(0.2,0.2,0.2,1.0)	Ambient color of material
GL_DIFFUSE	(0.8,0.8,0.8,1.0)	Diffuse color of material
GL_AMBIENT_AND_DIFFUSE		Ambient and diffuse color
GL_SPECULAR	(0.0,0.0,0.0,1.0)	Specular color of material
GL_SHININESS	0.0	Specular exponent
GL_EMISSION	(0.0,0.0,0.0,1.0)	Emissive color of material
GL_COLOR_INDEXES	(0,1,1)	Ambient, diffuse and specular color indices

Shading model and enabling lighting

`glShadeModel(GL_FLAT);`
Flat shading model for polygons

`glShadeModel(GL_SMOOTH);`
Smooth (Gouraud) shading model for polygons

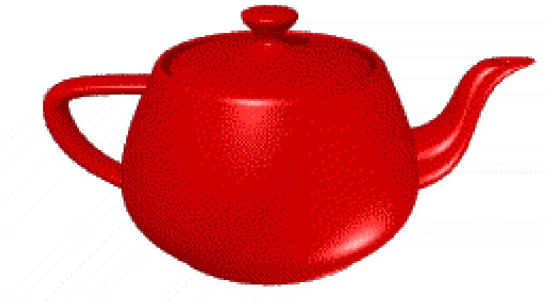
`glEnable(GL_LIGHTING);`
Enables lighting

`glDisable(GL_LIGHTING);`
Disables lighting

`glEnable(GL_LIGHT0);`
Enables light source 0

`glEnable(GL_LIGHT1);`
Enables light source 1 …

GL_FLAT



GL_SMOOTH

Specifying lighting/materials in OpenGL

1) Specify material properties:

```
void glMaterialfv(GLenum face,GLenum param,GLfloat *value);
    face = {GL_FRONT|GL_BACK|GL_FRONT_AND_BACK}
    param = {GL_AMBIENT|GL_DIFFUSE|GL_EMISSIVE|GL_SPECULAR}
    value = float[4] // RGBA

void glMaterialf(GLenum face,GL_SHININESS,GLfloat value);
```

2) Specify ambient light:

```
void glLightModelfv(GLenum param,GLfloat *value);
    param = GL_LIGHT_MODEL_AMBIENT
    value = float[4] // RGBA
```

3) Specify lights colors:

```
void glLightfv(GLenum light,GLenum param,GLfloat *value);
    light = {GL_LIGHT0|GL_LIGHT1|…}
    param = {GL_AMBIENT|GL_DIFFUSE|GL_SPECULAR}
    value = float[4] // RGBA
```

Programming lighting (cont)

4) Specify location of the light locations/directions:

```
void glLightfv(GLenum light, GL_POSITION,GLfloat *value);
    light = {GL_LIGHT0|GL_LIGHT1|…}
    value = float[4] // x,y,z,w
```

Coordinates of the light source are transformed by the current transformation matrix

5) Enable lighting

```
void glEnable(GLenum type); type = GL_LIGHTING;
```

6) Enable individual lights

```
void glEnable(GLenum type); type = GL_LIGHT0, GL_LIGHT1, …
```

7) Specify shading model

void glShadeModel(GLenum type);

type = GL_FLAT – flat shading

type = GL_SMOOTH – smooth shading (Gouraud Shading)