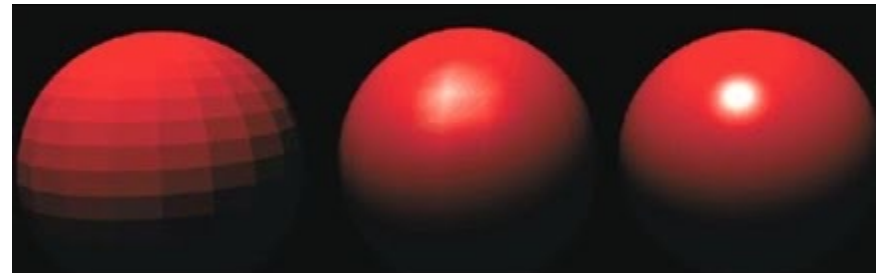
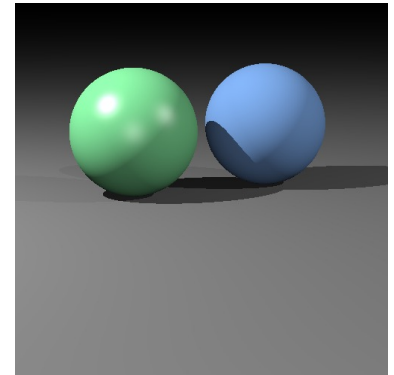
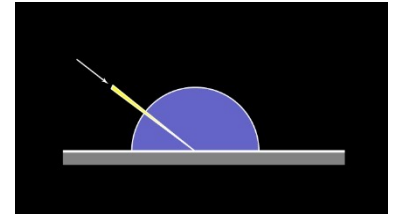
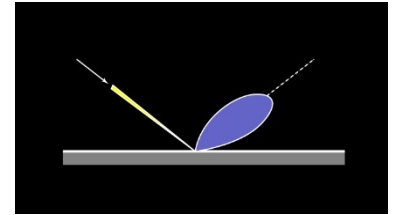


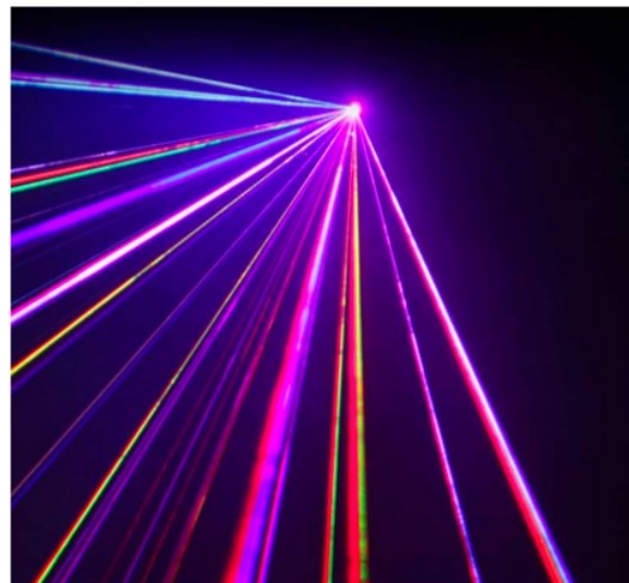
Lighting - CSEI 60

- Light and Surfaces
- Diffuse
- Specular
- Phong Model = Ambient + Diffuse + Specular
- BRDFs
- Which Normal?
- Spot Lights
- Bump/Normal Maps
- PTMs
- WebGL
- Administrative
- Q&A

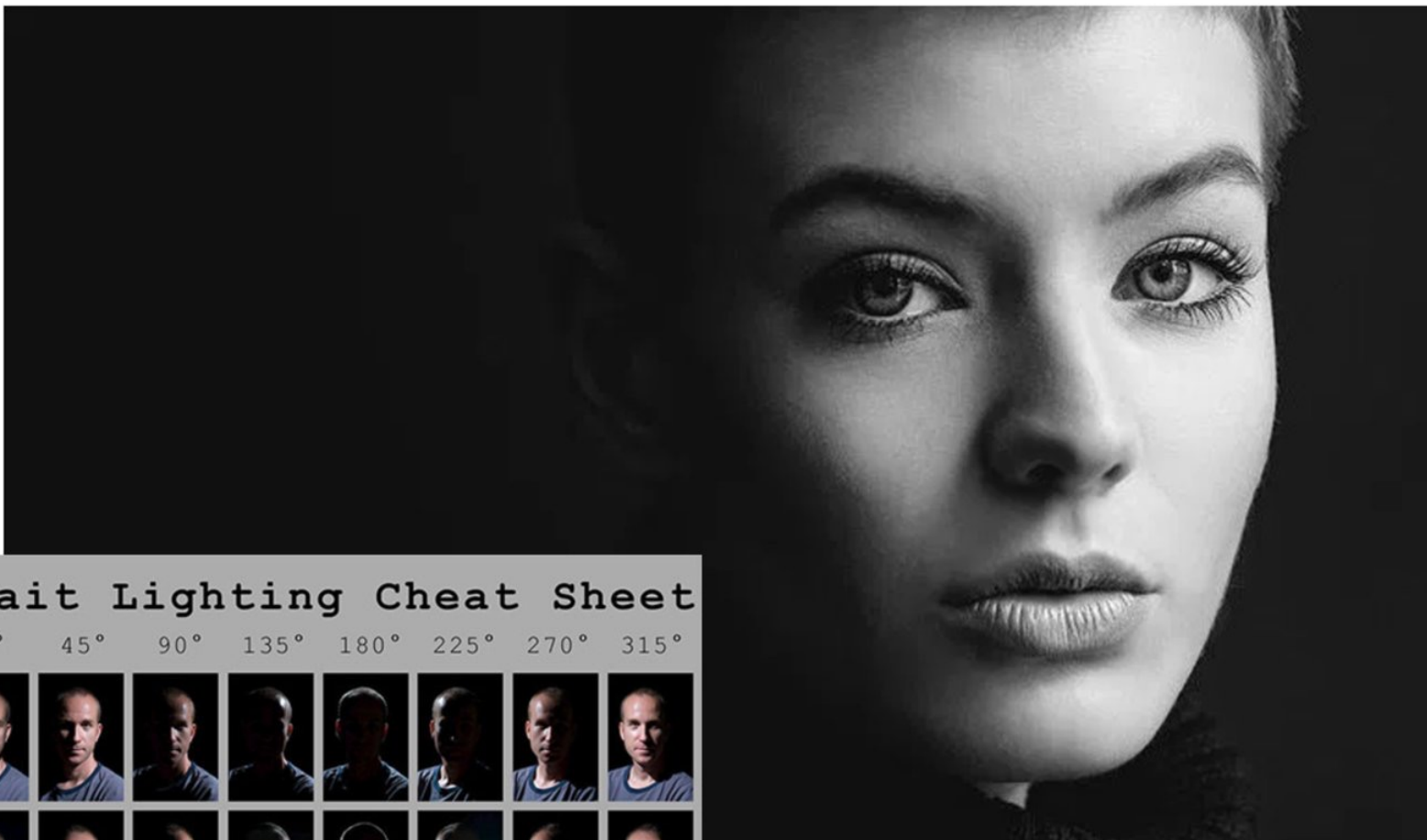


Light and Surfaces

Lighting



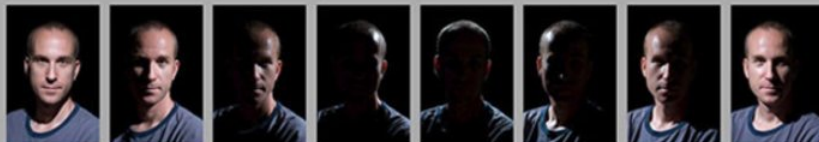
Lighting



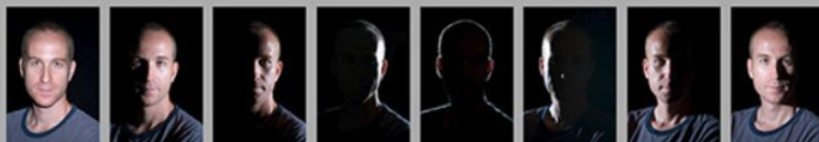
Portrait Lighting Cheat Sheet

0° 45° 90° 135° 180° 225° 270° 315°

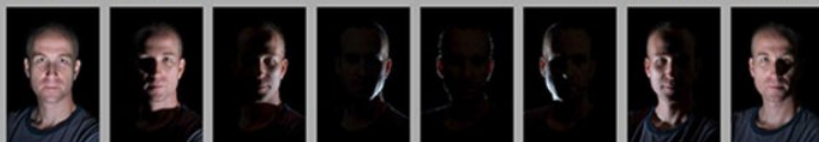
Flash
@45°
Down



Flash
@0°



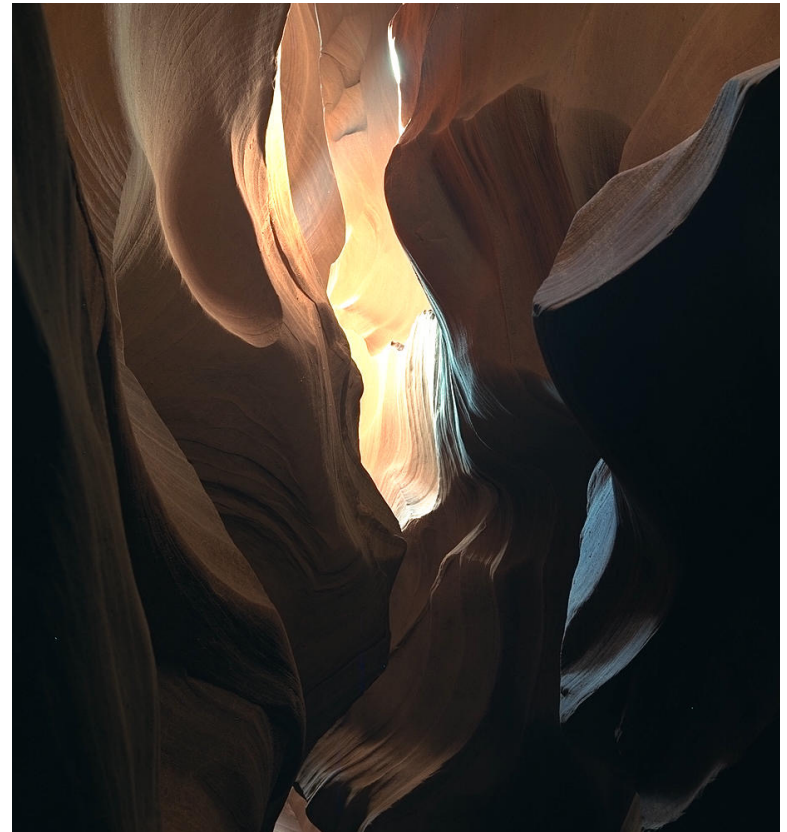
Flash
@45°
Up



(cc) DIYPhotography.net

Shading

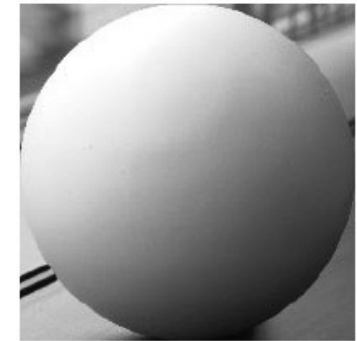
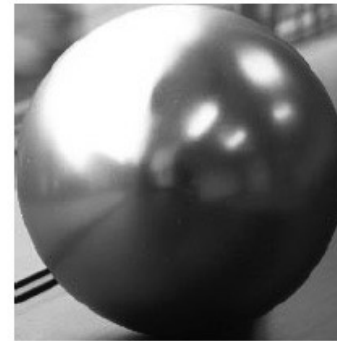
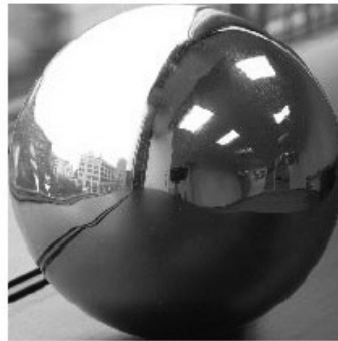
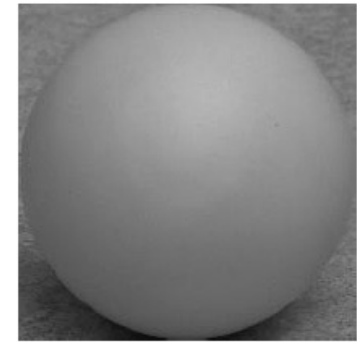
- Variation in observed color across an object
 - strongly affected by lighting
 - present even for homogeneous material
- caused by how a material reflects light
 - depends on
 - geometry
 - lighting
 - material
 - therefore gives cues to all 3



[Philip Greenspun]

Recognizing materials

- Human visual system is quite good at understanding shading



A

B

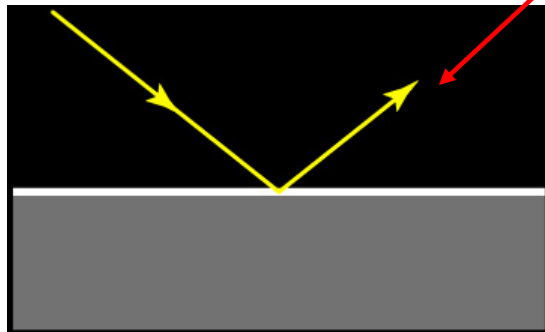
C

D

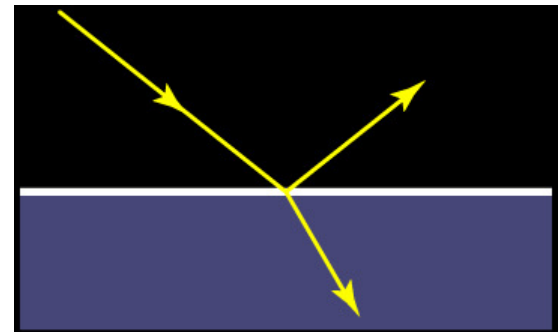
Simple materials



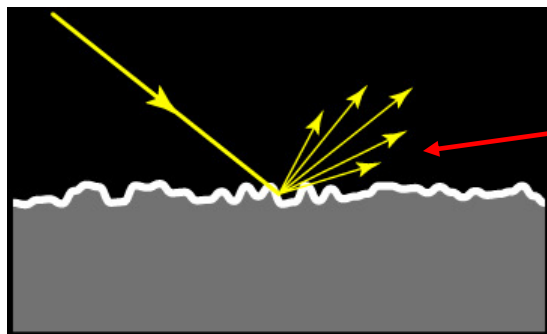
metal



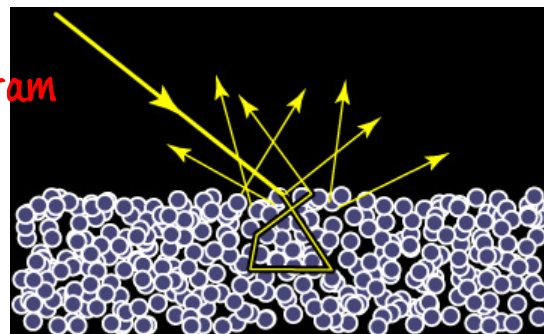
dielectric



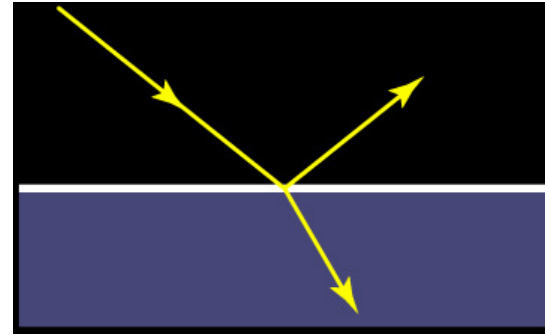
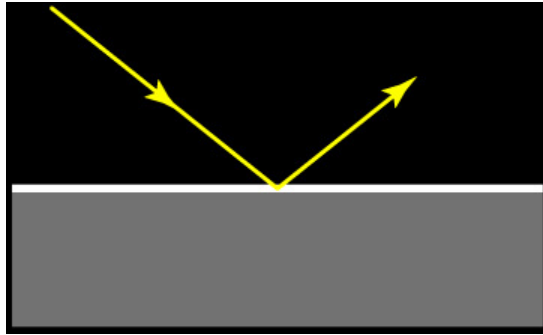
Adding microgeometry



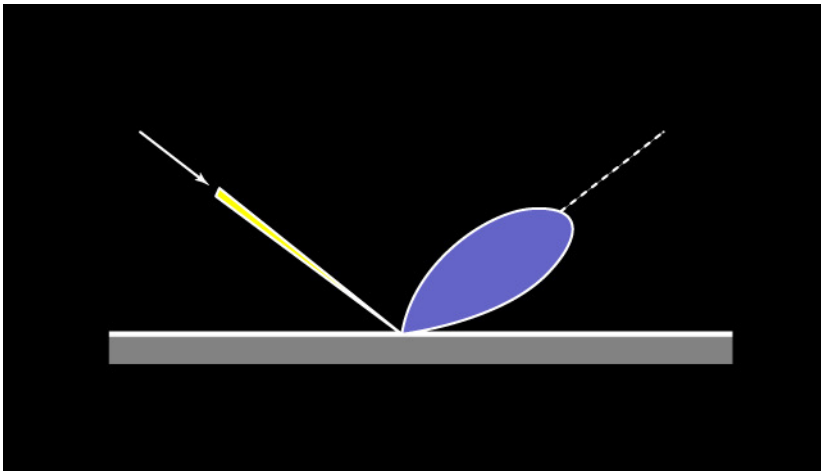
Goniometric diagram



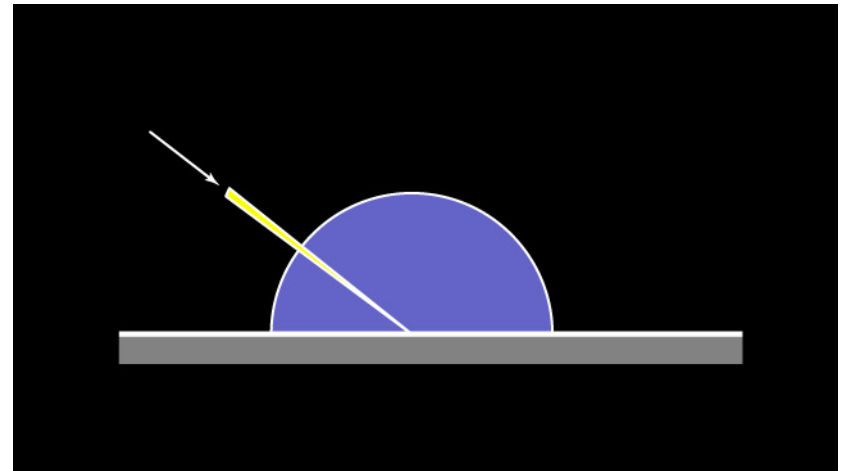
Classic reflection behavior



ideal specular (Fresnel)



rough specular



Lambertian

Reflection from diffuse surfaces



(Dorsey)

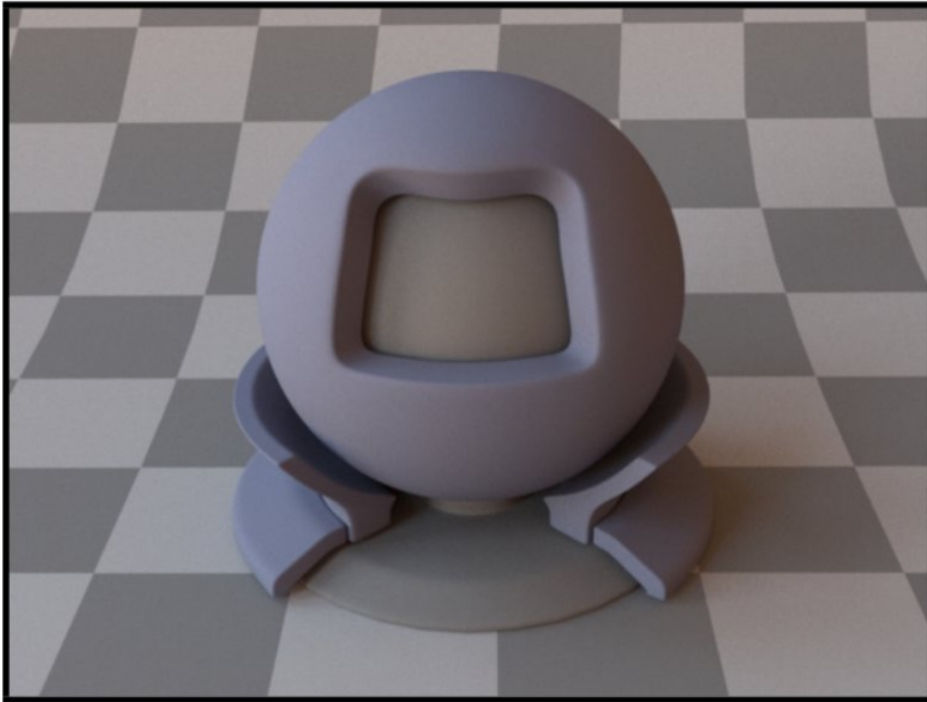
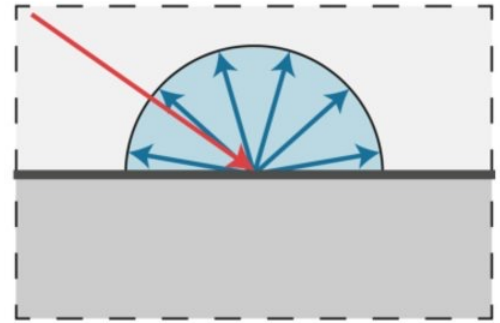


Johann Lambert
(1728-1777)

two viewpoints, same illumination

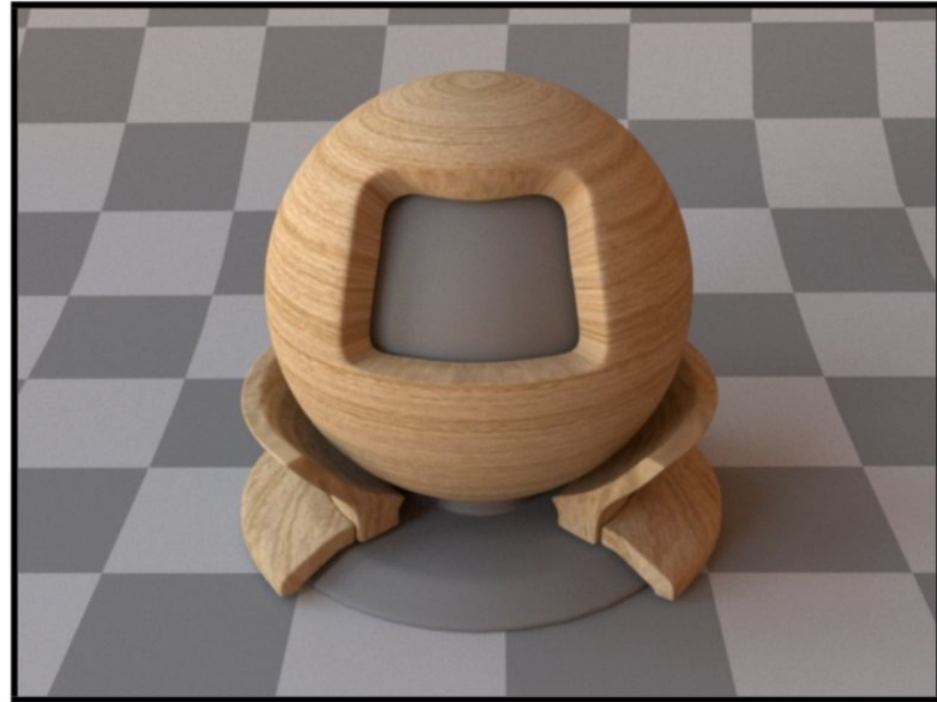
- ♦ rough surfaces reflect light uniformly in all directions
 - appearance is independent of viewing direction
 - if perfectly so, surface is called ideal diffuse (“Lambertian”)

Diffuse / Lambertian material



Uniform colored diffuse BRDF

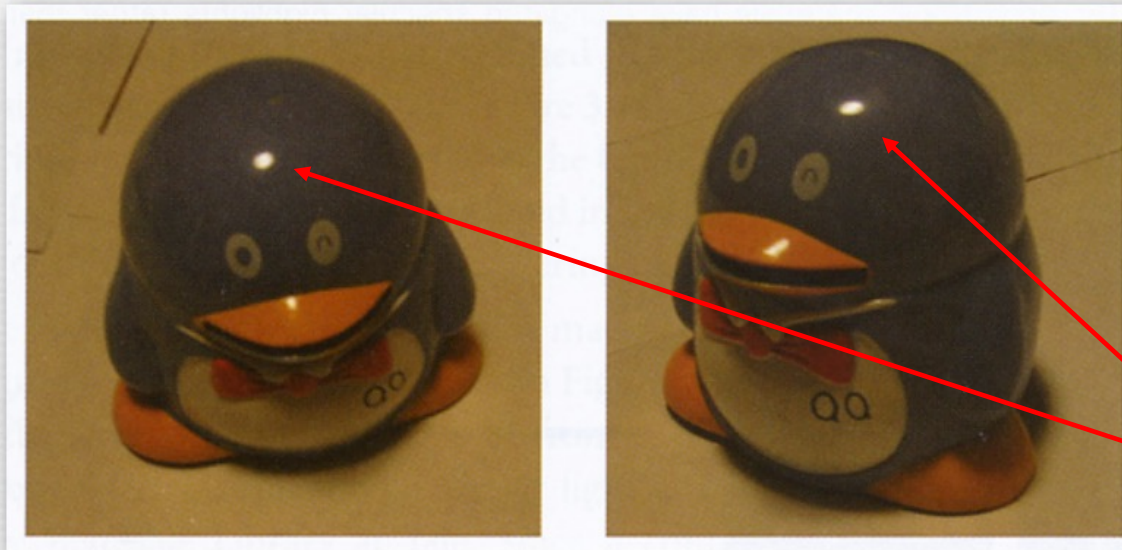
Albedo (fraction of light reflected) is same for all surface points p



Textured diffuse BRDF

Albedo is spatially varying, and is encoded in texture map.

Reflection from shiny surfaces



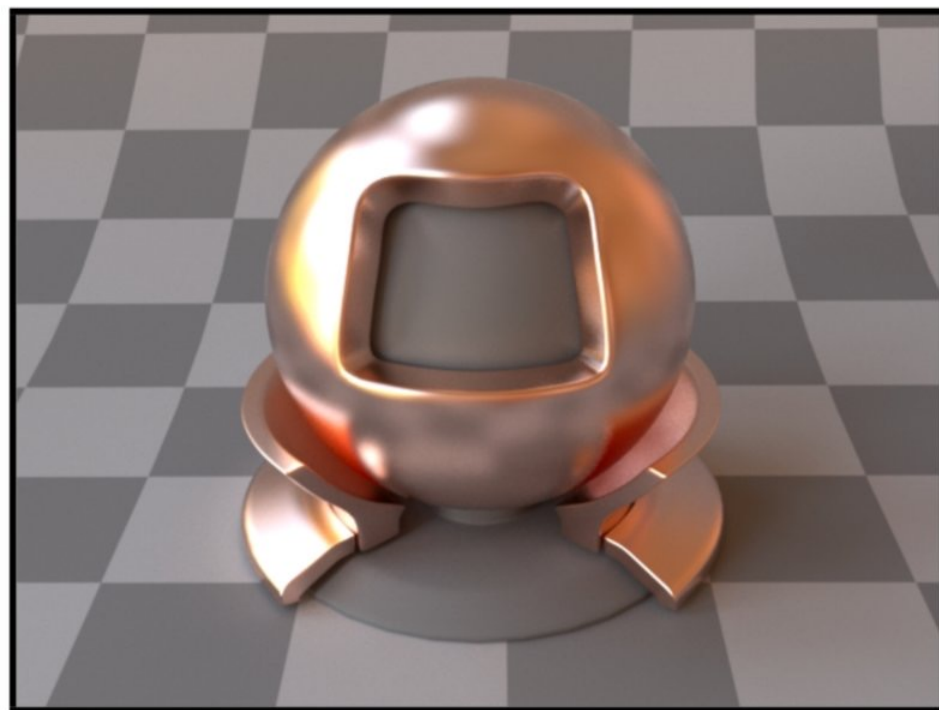
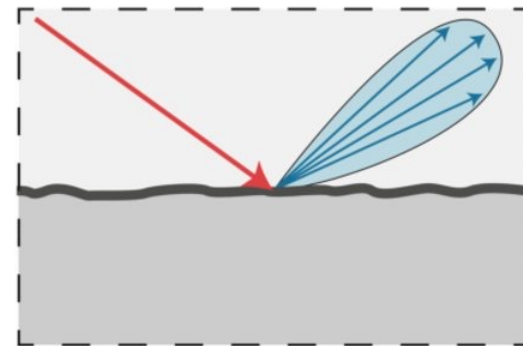
(Dorsey)

two viewpoints,
same illumination
(i.e. fixed to object)

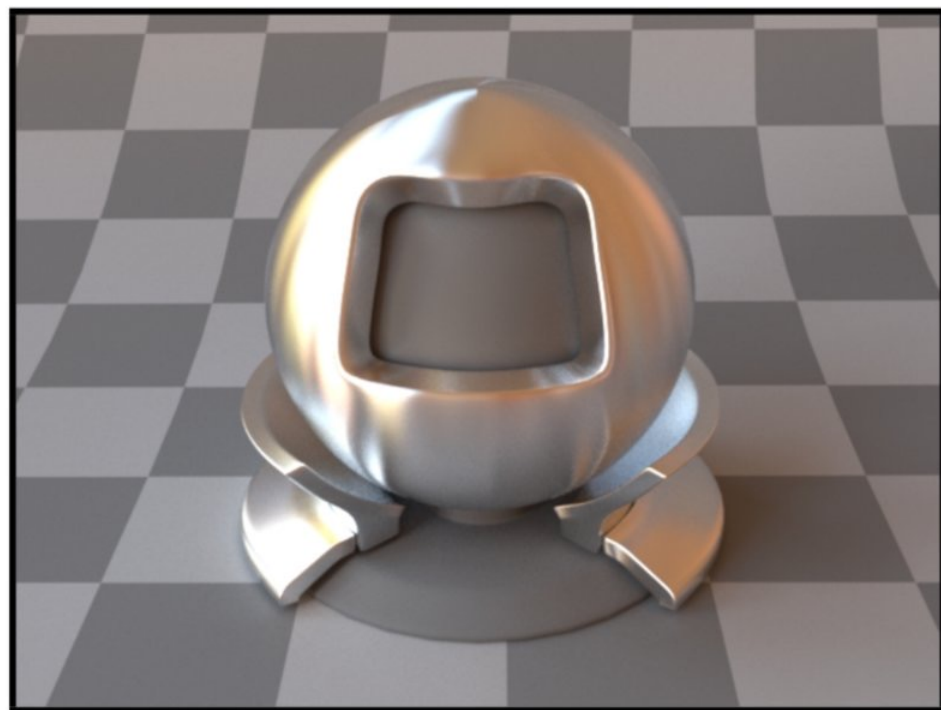
Dot moved!

- location of highlight changes with movement of light or viewer

Glossy material (BRDF)



Copper



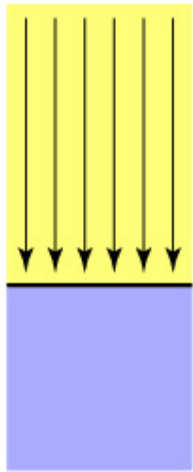
Aluminum

[Mitsuba renderer, Wenzel Jakob, 2010]

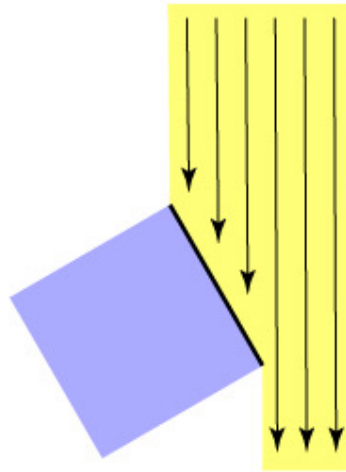
Diffuse

Diffuse reflection

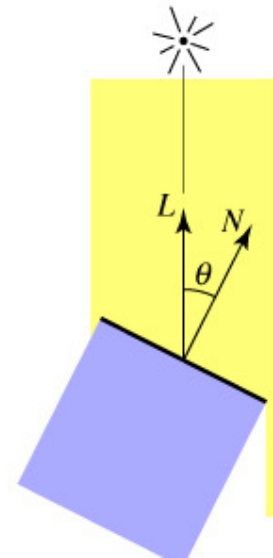
- Light is scattered uniformly in all directions
 - the surface color is the same for all viewing directions
- Lambert's cosine law



Top face of cube
receives a certain
amount of light



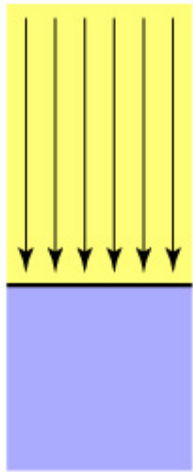
Top face of
60° rotated cube
intercepts half the light



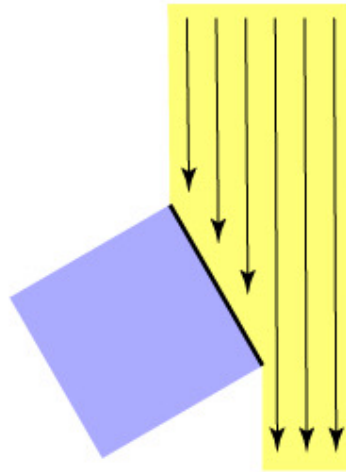
In general, light per unit
area is proportional to
 $\cos \theta = L \cdot N$

Diffuse reflection

- Light is scattered uniformly in all directions
 - the surface color is the same for all viewing directions
- Lambert's cosine law



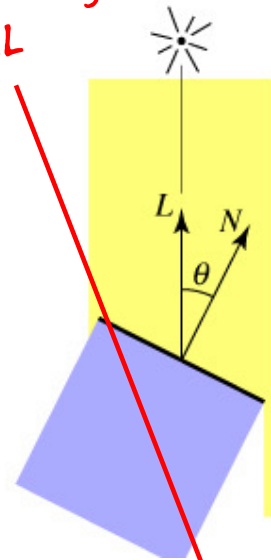
Top face of cube
receives a certain
amount of light



Top face of
60° rotated cube
intercepts half the light

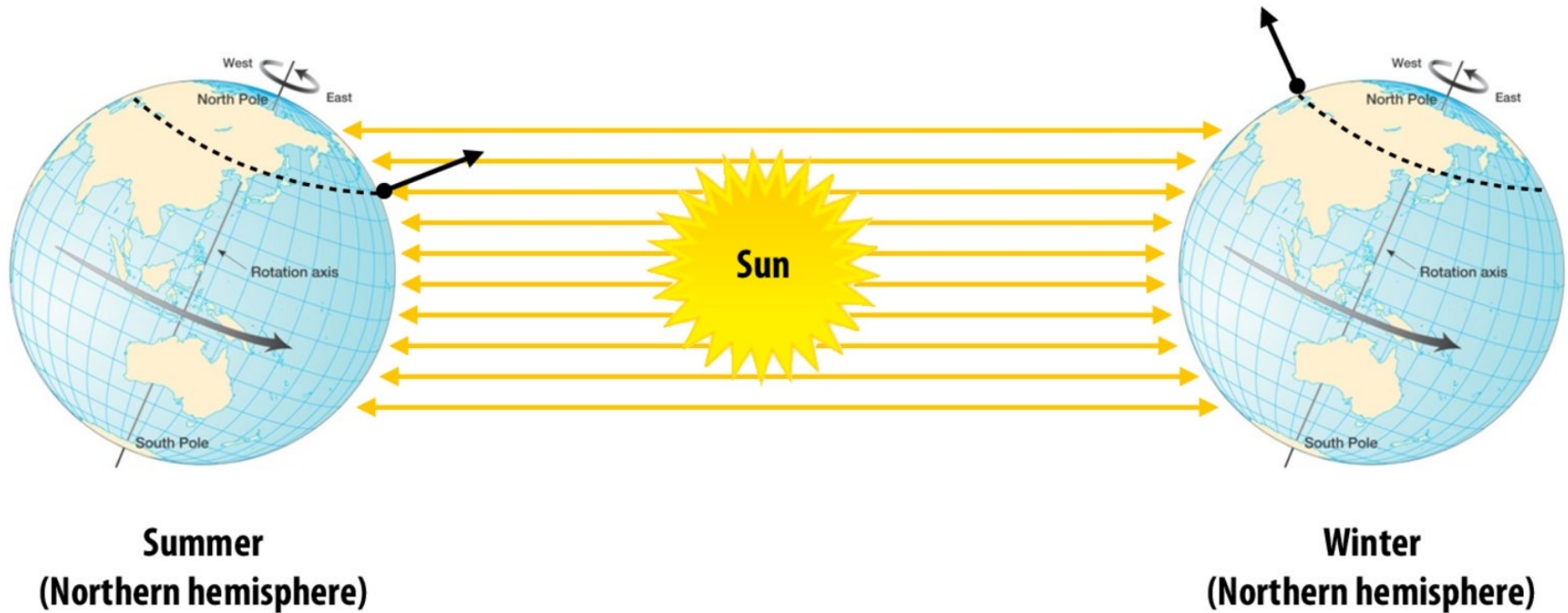
$$N \cdot L = |N| |L| \cos(\theta)$$

Assume normalized N, L



In general, light per unit
area is proportional to
 $\cos \theta \equiv L \cdot N$

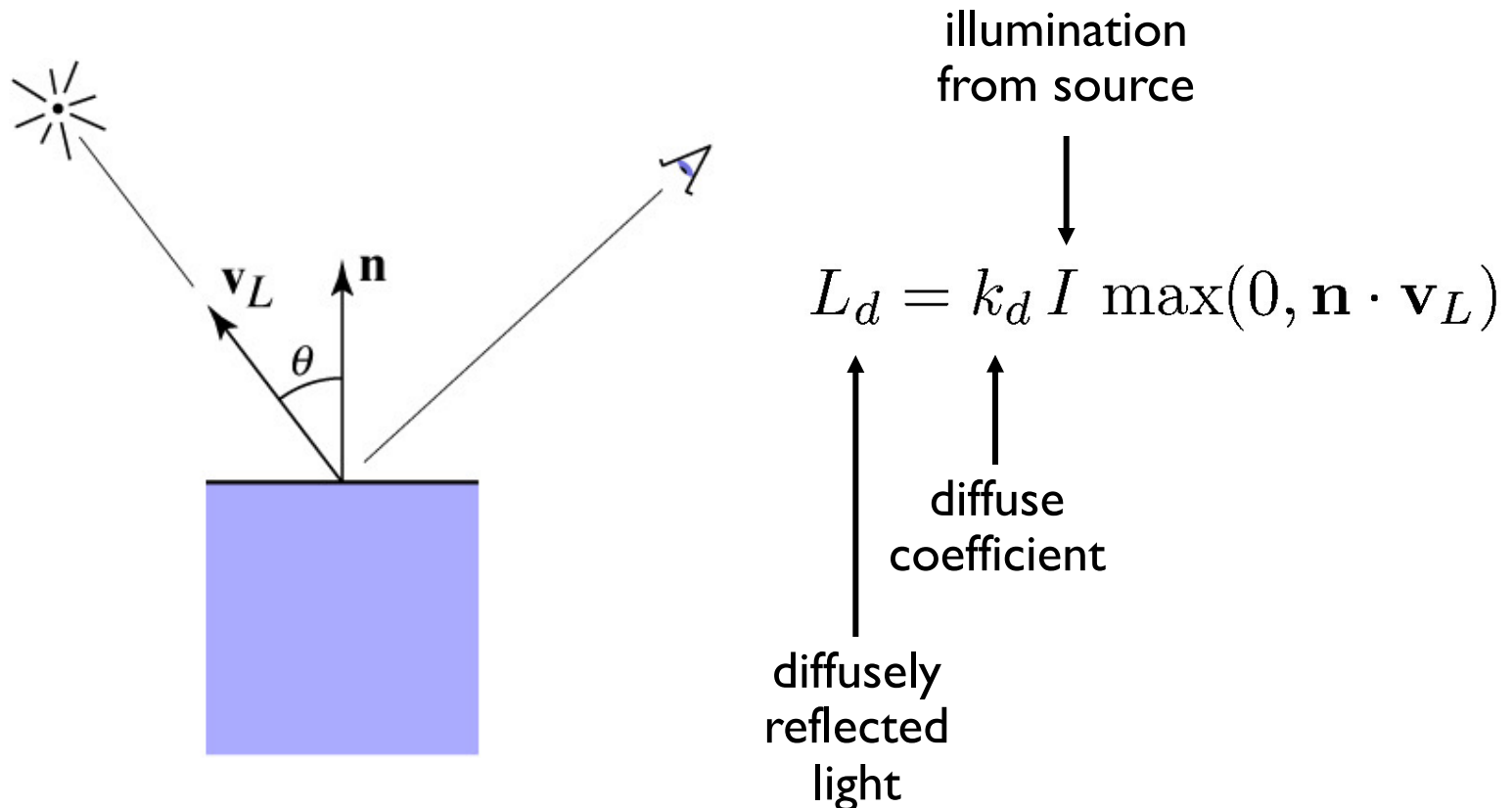
Why do we have seasons?



Earth's axis of rotation: $\sim 23.5^\circ$ off axis

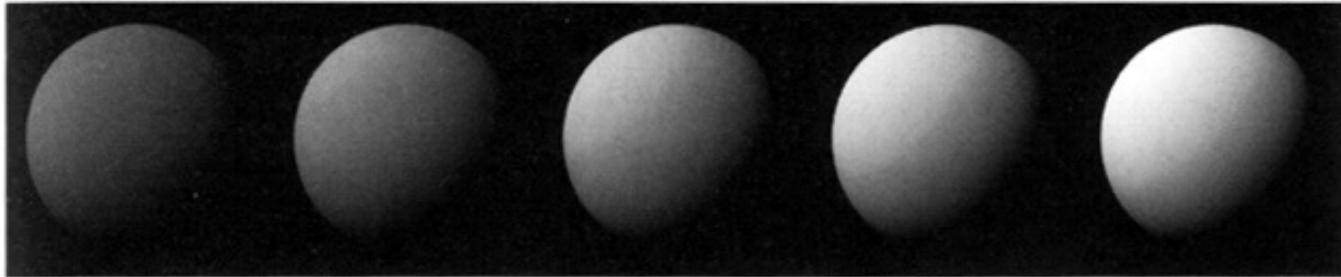
Lambertian shading

- Shading independent of view direction



Lambertian shading

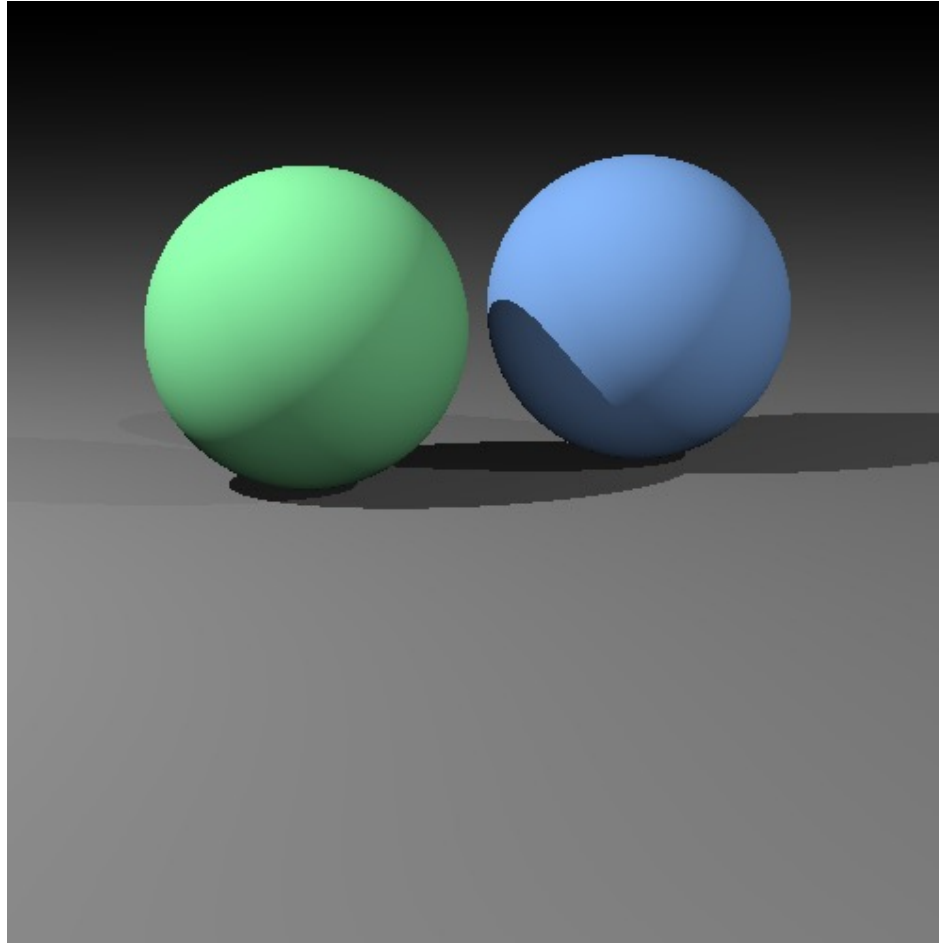
- Produces matte appearance



$k_D \longrightarrow$

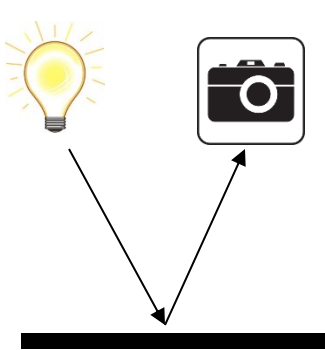
[Foley et al.]

Diffuse shading

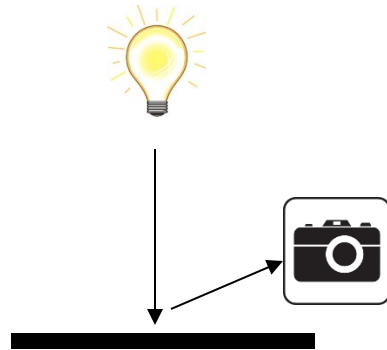


Q Lambertian Shading

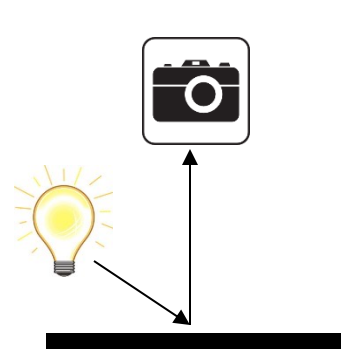
In which configuration does the Lambertian surface appear brightest?



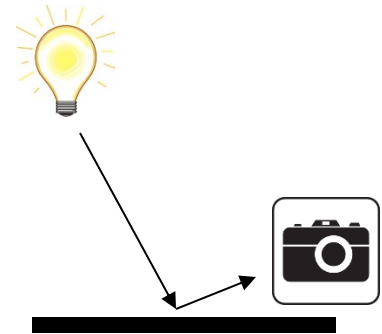
(A)



(B)



(C)



(D)

Participation Survey

Participation May 12

Form description

This form is automatically collecting email addresses for UC Santa Cruz users. [Change settings](#)

I was in class May 12

- ☐ Yes
- ☐ No

Roughly how long did you spend on [A3 \(Blocky World\)](#)

- ☐ 0-5 hours
- ☐ 5-10 hours
- ☐ 10-15 hours
- ☐ 15+ hours

[A3](#) was :

☒ Multiple choice

- ☐ Fun!
- ☐ Pretty ok as far as class assignments go.
- ☐ An average class assignment.
- ☐ Painful. Didn't care for it.
- ☐ Never ever give this assignment to students again!
- ☐ Add option or [add "Other"](#)



Required

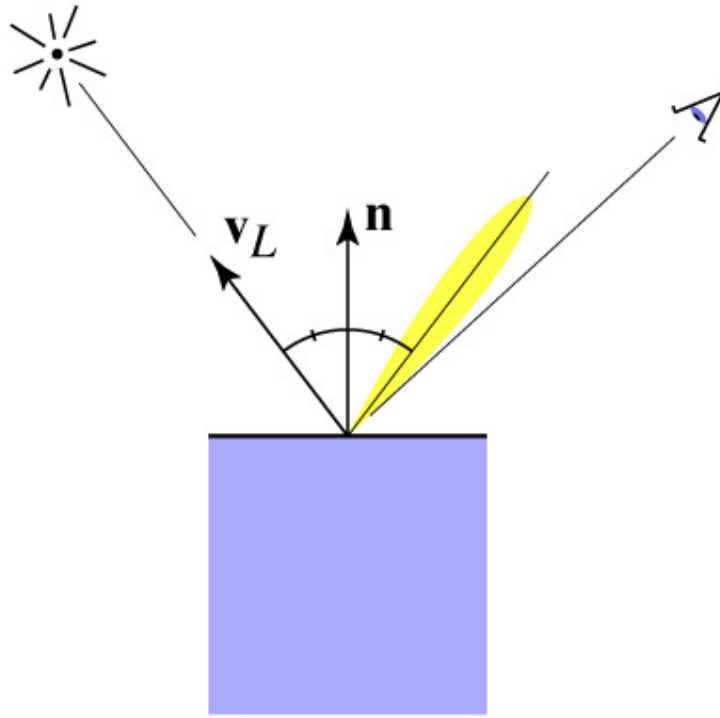


There were YouTube videos to help with coding for A3:

Specular

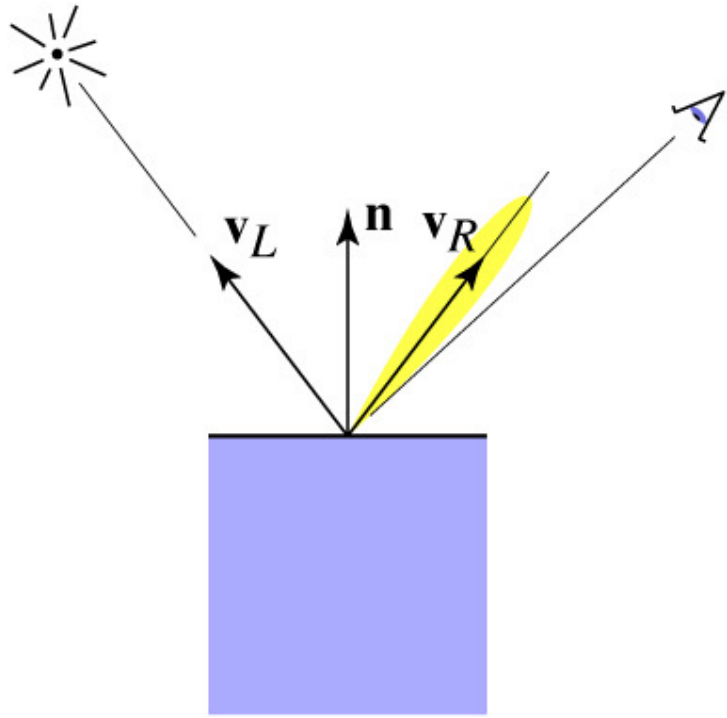
Specular shading (Phong model)

- Intensity depends on view direction
 - bright near mirror configuration



Specular shading (Phong model)

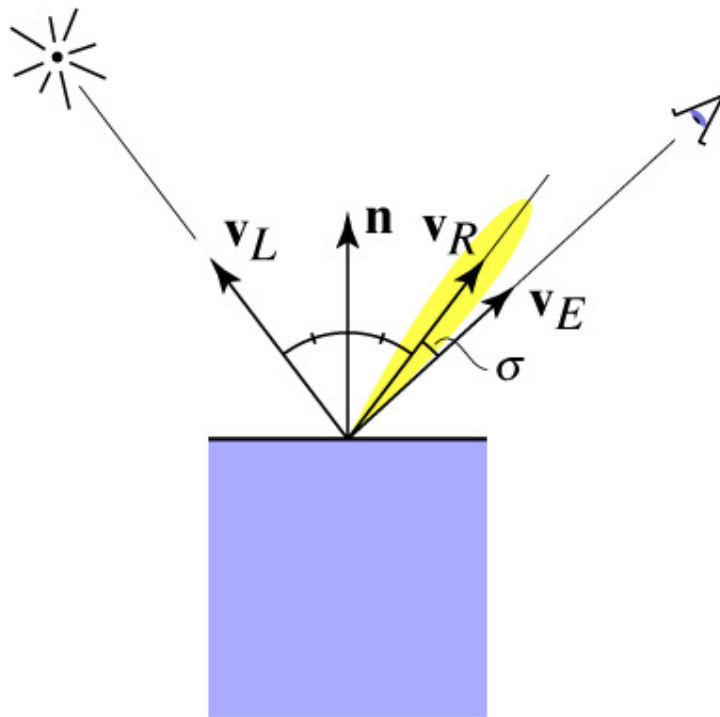
- Intensity depends on view direction
 - bright near mirror configuration



$$\begin{aligned}\mathbf{v}_R &= \mathbf{v}_L + 2((\mathbf{n} \cdot \mathbf{v}_L)\mathbf{n} - \mathbf{v}_L) \\ &= 2(\mathbf{n} \cdot \mathbf{v}_L)\mathbf{n} - \mathbf{v}_L\end{aligned}$$

Specular shading (Phong model)

- Intensity depends on view direction
 - bright near mirror configuration



$$\begin{aligned}\mathbf{v}_R &= \mathbf{v}_L + 2((\mathbf{n} \cdot \mathbf{v}_L)\mathbf{n} - \mathbf{v}_L) \\ &= 2(\mathbf{n} \cdot \mathbf{v}_L)\mathbf{n} - \mathbf{v}_L\end{aligned}$$

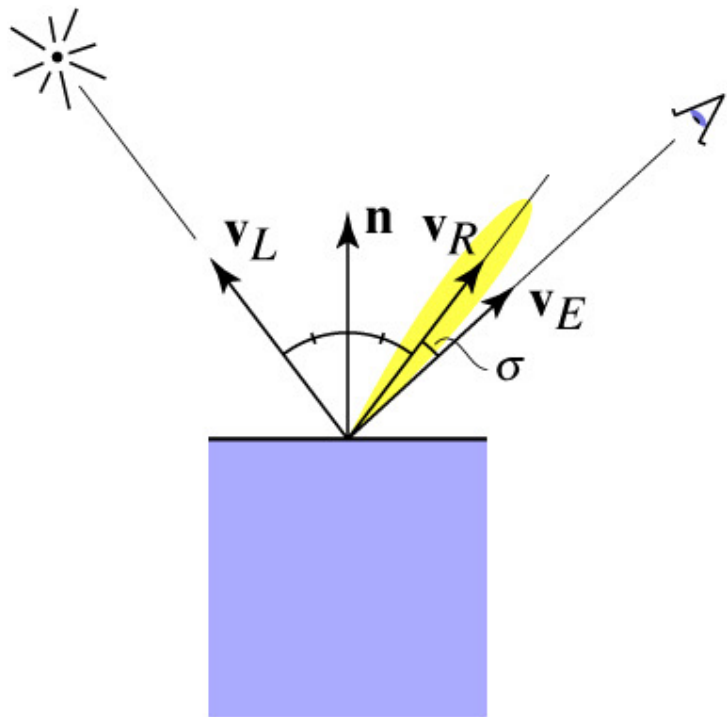
$$\begin{aligned}L_s &= k_s I \max(0, \cos \sigma)^n \\ &= k_s I \max(0, \mathbf{v}_E \cdot \mathbf{v}_R)^n\end{aligned}$$

↑
specularly
reflected
light

↑
specular
coefficient

Specular shading (Phong model)

- Intensity depends on view direction
 - bright near mirror configuration



$$\begin{aligned}\mathbf{v}_R &= \mathbf{v}_L + 2((\mathbf{n} \cdot \mathbf{v}_L)\mathbf{n} - \mathbf{v}_L) \\ &= 2(\mathbf{n} \cdot \mathbf{v}_L)\mathbf{n} - \mathbf{v}_L\end{aligned}$$

$$\begin{aligned}L_s &= k_s I \max(0, \cos \sigma)^n \\ &= k_s I \max(0, \mathbf{v}_E \cdot \mathbf{v}_R)^n\end{aligned}$$

↑
specularly
reflected
light

↑
specular
coefficient

Phong model—plots

- Increasing n narrows the lobe

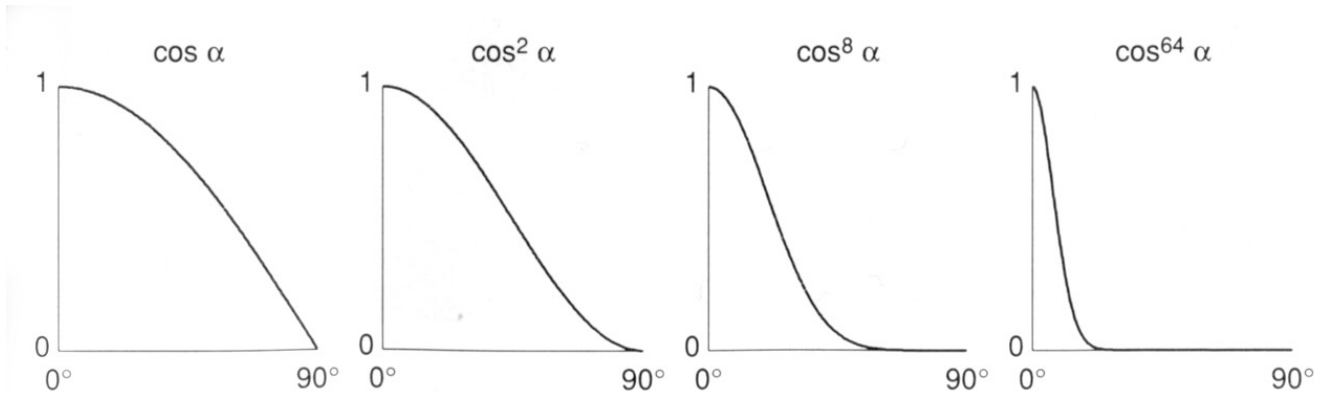
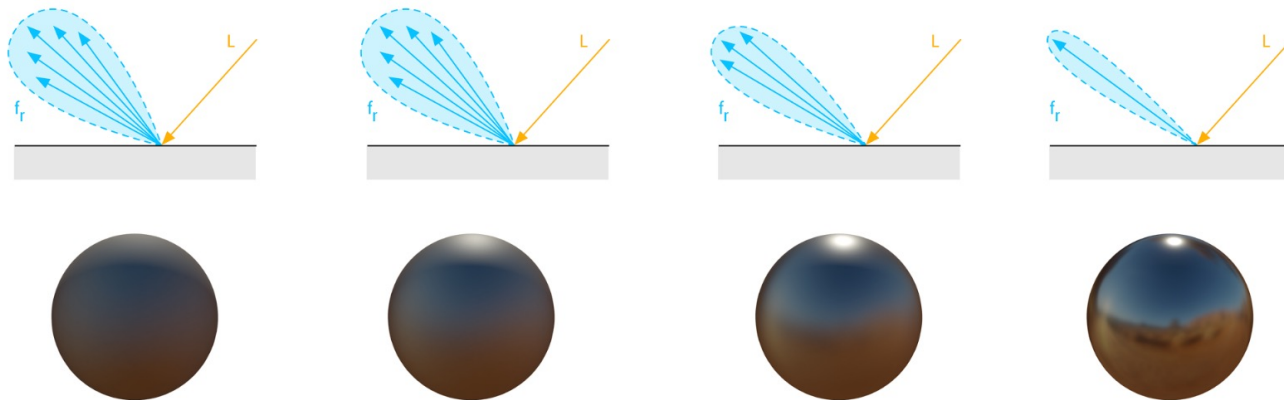


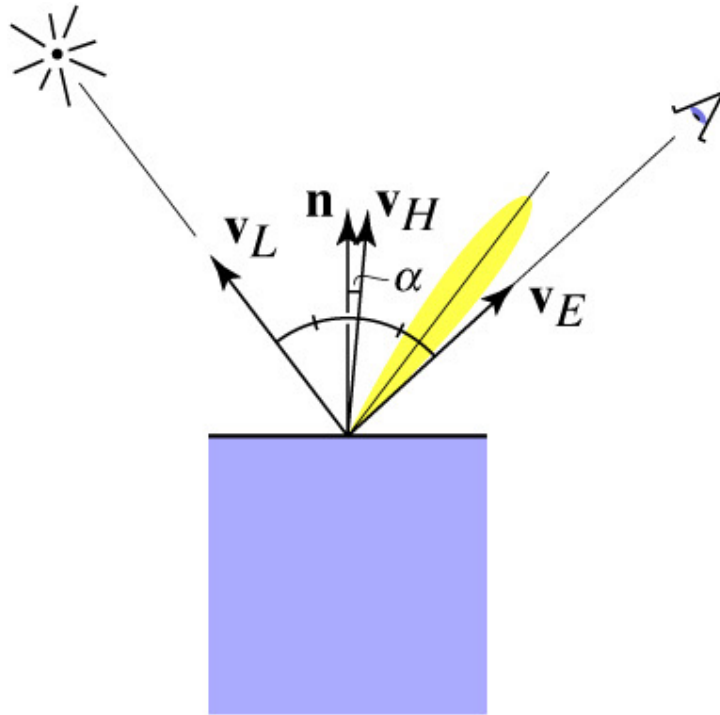
Fig. 16.9 Different values of $\cos^n \alpha$ used in the Phong illumination model.



[Foley et al.]

Phong variant: Blinn-Phong

- Rather than computing reflection directly, just compare to normal bisection property

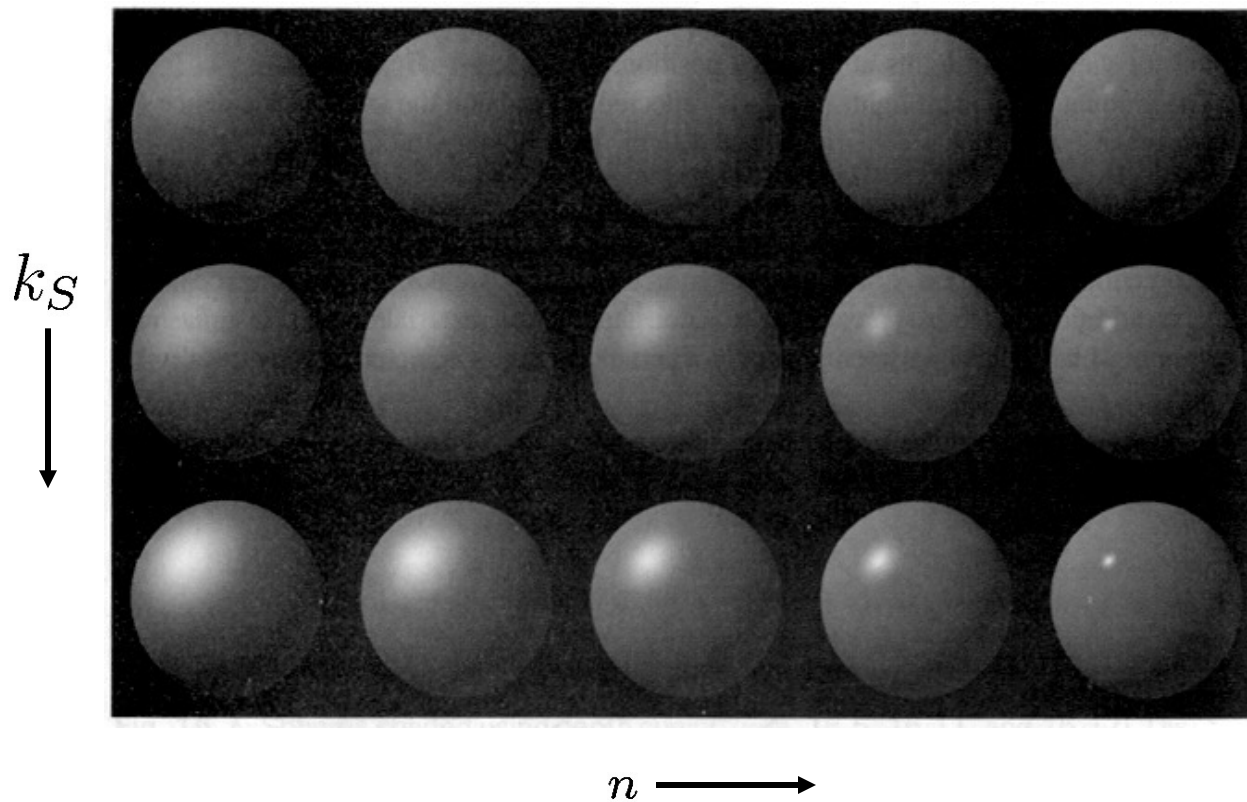


$$\begin{aligned}\mathbf{v}_H &= \text{bisector}(\mathbf{v}_L, \mathbf{v}_E) \\ &= \frac{(\mathbf{v}_L + \mathbf{v}_E)}{\|\mathbf{v}_L + \mathbf{v}_E\|}\end{aligned}$$

$$\begin{aligned}L_s &= k_s I \max(0, \cos \alpha)^n \\ &= k_s I \max(0, \mathbf{n} \cdot \mathbf{v}_H)^n\end{aligned}$$

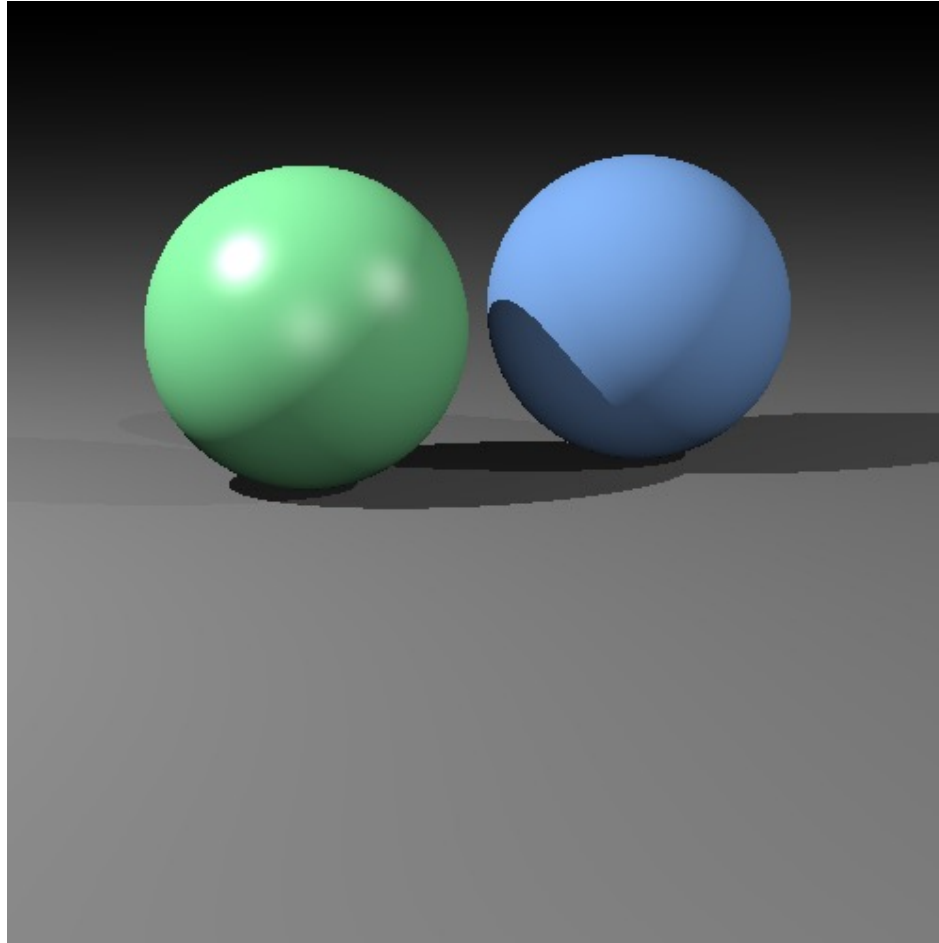
Specular shading

- Phong and Blinn-Phong



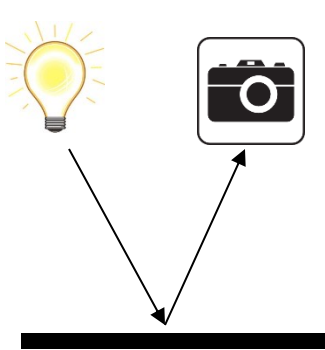
[Foley et al.]

Diffuse + Phong shading

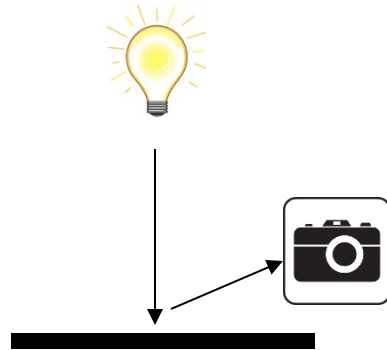


Q Specular Shading

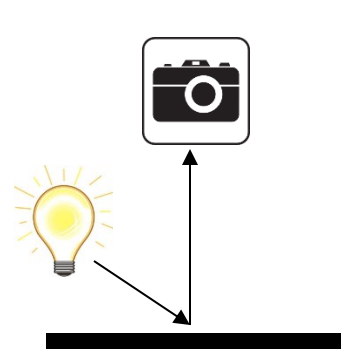
In which configuration does the specular surface appear brightest?



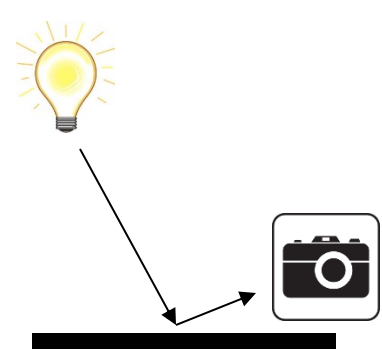
(A)



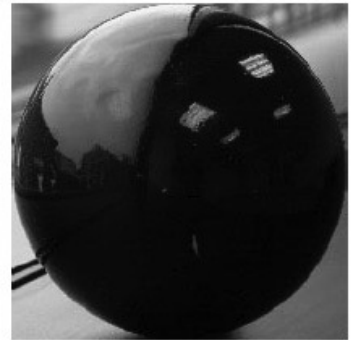
(B)



(C)



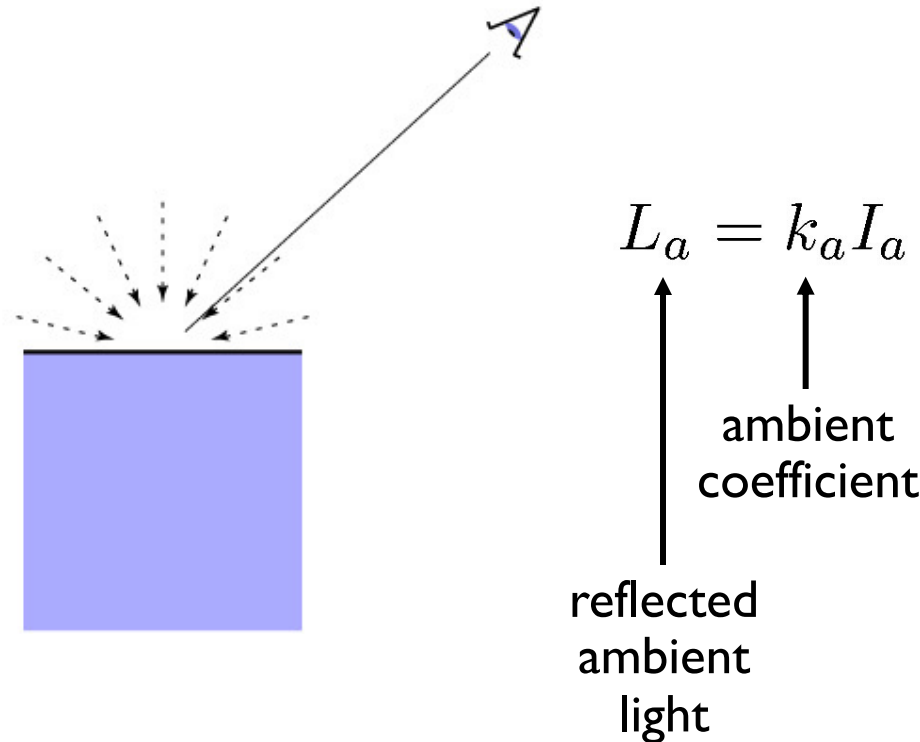
(D)



Phong Model = Ambient + Diffuse + Specular

Ambient shading

- Shading does not depend on anything
 - add constant color to account for disregarded illumination and fill in black shadows



Putting it together

- Usually include ambient, diffuse, Phong in one model

$$\begin{aligned} L &= L_a + L_d + L_s \\ &= \underset{\text{ambient}}{k_a I_a} + \underset{\text{diffuse}}{I (k_d \max(0, \mathbf{n} \cdot \mathbf{v}_L))} + \underset{\text{specular}}{k_s \max(0, \mathbf{n} \cdot \mathbf{v}_H)^n} \end{aligned}$$

- The final result is the sum over many lights


$$\begin{aligned} L &= L_a + \sum_i (L_d)_i + (L_s)_i \\ &= k_a I_a + \sum_i I_i (k_d \max(0, \mathbf{n} \cdot (\mathbf{v}_L)_i) + k_s \max(0, \mathbf{n} \cdot (\mathbf{v}_H)_i)^n) \end{aligned}$$

Putting it together

- Usually include ambient, diffuse, Phong in one model

$$\begin{aligned} L &= L_a + L_d + L_s \\ &= k_a I_a + I (k_d \max(0, \mathbf{n} \cdot \mathbf{v}_L) + k_s \max(0, \mathbf{n} \cdot \mathbf{v}_H)^n) \end{aligned}$$

Definitely Vectors



- The final result is the sum over many lights


$$\begin{aligned} L &= L_a + \sum_i (L_d)_i + (L_s)_i \\ &= k_a I_a + \sum_i I_i (k_d \max(0, \mathbf{n} \cdot (\mathbf{v}_L)_i) + k_s \max(0, \mathbf{n} \cdot (\mathbf{v}_H)_i)^n) \end{aligned}$$

Putting it together

- Usually include ambient, diffuse, Phong in one model

$$\begin{aligned} L &= L_a + L_d + L_s \\ &= k_a I_a + I (k_d \max(0, \mathbf{n} \cdot \mathbf{v}_L) + k_s \max(0, \mathbf{n} \cdot \mathbf{v}_H)^n) \end{aligned}$$

Dot product (not cross product)



- The final result is the sum over many lights

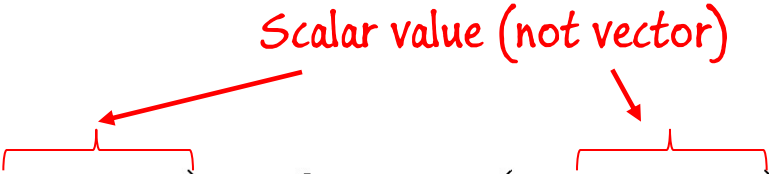
$$\begin{aligned} L &= L_a + \sum_i (L_d)_i + (L_s)_i \\ &= k_a I_a + \sum_i I_i (k_d \max(0, \mathbf{n} \cdot (\mathbf{v}_L)_i) + k_s \max(0, \mathbf{n} \cdot (\mathbf{v}_H)_i)^n) \end{aligned}$$

Putting it together

- Usually include ambient, diffuse, Phong in one model

$$\begin{aligned} L &= L_a + L_d + L_s \\ &= k_a I_a + I \left(k_d \max(0, \mathbf{n} \cdot \mathbf{v}_L) + k_s \max(0, \mathbf{n} \cdot \mathbf{v}_H)^n \right) \end{aligned}$$

Scalar value (not vector)



- The final result is the sum over many lights

$$\begin{aligned} L &= L_a + \sum_i (L_d)_i + (L_s)_i \\ &= k_a I_a + \sum_i I_i \left(k_d \max(0, \mathbf{n} \cdot (\mathbf{v}_L)_i) + k_s \max(0, \mathbf{n} \cdot (\mathbf{v}_H)_i)^n \right) \end{aligned}$$

Putting it together

- Usually include ambient, diffuse, Phong in one model

$$\begin{aligned} L &= L_a + L_d + L_s \\ &= k_a I_a + I (k_d \max(0, \mathbf{n} \cdot \mathbf{v}_L) + k_s \max(0, \mathbf{n} \cdot \mathbf{v}_H)^n) \end{aligned}$$

- The final result is the sum over many lights

Colors! Could be either :
- scalar – this is an intensity
- vector – this is RGB

$$\begin{aligned} L &= L_a + \sum_i (L_d)_i + (L_s)_i \\ &= k_a I_a + \sum_i I_i (k_d \max(0, \mathbf{n} \cdot (\mathbf{v}_L)_i) + k_s \max(0, \mathbf{n} \cdot (\mathbf{v}_H)_i)^n) \end{aligned}$$

Putting it together

- Usually include ambient, diffuse, Phong in one model

$$\begin{aligned} L &= L_a + L_d + L_s \\ &= \underbrace{k_a I_a}_{\text{ambient}} + \underbrace{I (k_d \max(0, \mathbf{n} \cdot \mathbf{v}_L))}_{\text{diffuse}} + k_s \max(0, \mathbf{n} \cdot \mathbf{v}_H)^n \end{aligned}$$

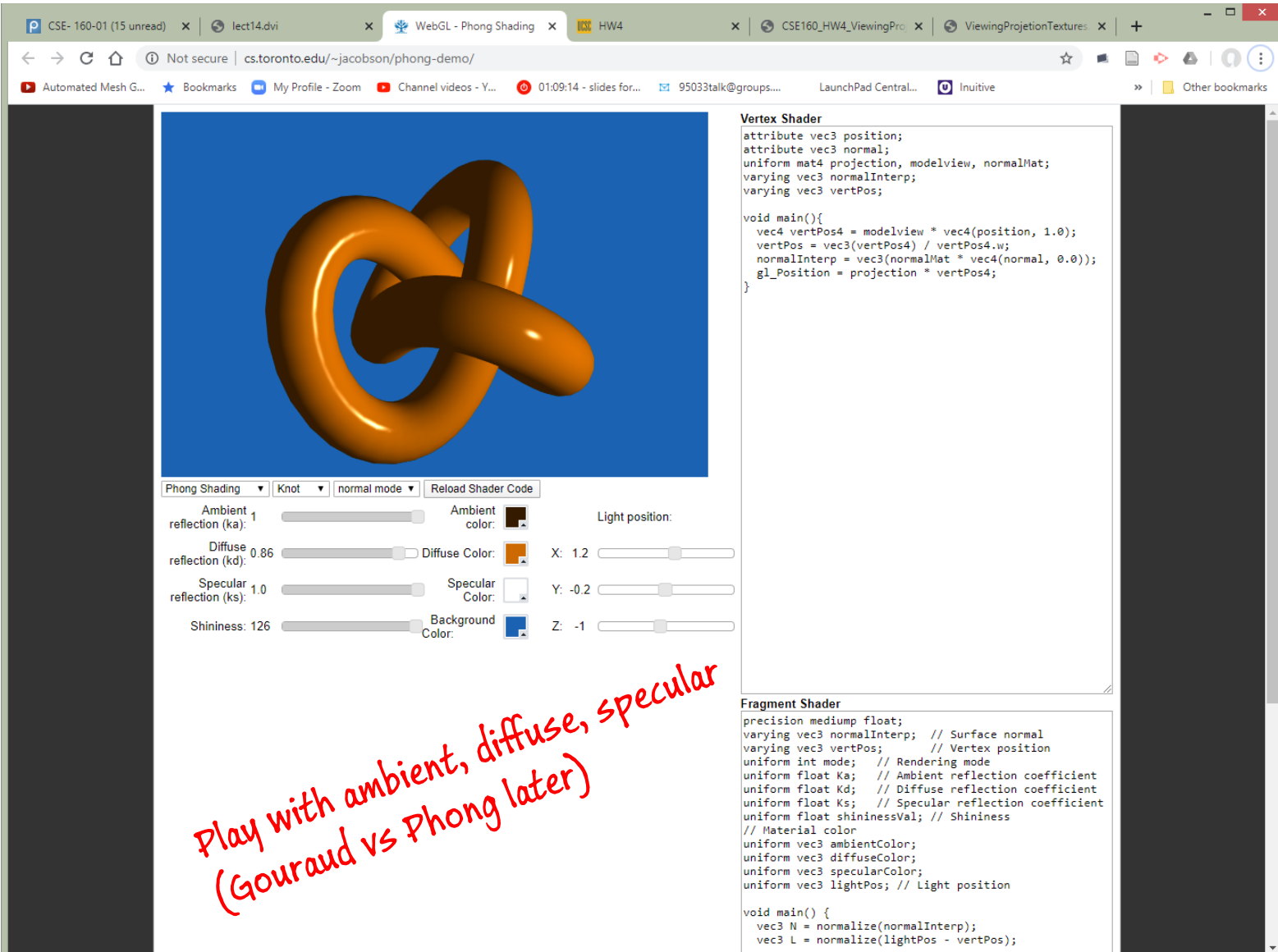
- The final result is the sum over many lights

$$\begin{aligned} L &= L_a + \sum_i (L_d)_i + (L_s)_i \\ &= k_a I_a + \sum_i I_i (k_d \max(0, \mathbf{n} \cdot (\mathbf{v}_L)_i) + k_s \max(0, \mathbf{n} \cdot (\mathbf{v}_H)_i)^n) \end{aligned}$$

Could be (vector)(vector):
Is that a 'dot' or a 'cross'?
Neither! Its elementwise.
Treat each of R,G,B independently.

Really great example to play with the parameters:

<http://www.cs.toronto.edu/~jacobson/phong-demo/>



The screenshot shows a web browser window with the URL <http://www.cs.toronto.edu/~jacobson/phong-demo/>. The browser tabs include "CSE- 160-01 (15 unread)", "lect14.dvi", "WebGL - Phong Shading", "HW4", "CSE160_HW4_ViewingProc", and "ViewingProjetionTextures". The browser address bar shows "Not secure | cs.toronto.edu/~jacobson/phong-demo/".

The main content area displays a 3D rendering of a knot (a trefoil knot) in a bright orange color, set against a solid blue background. The knot is rendered with smooth shading, showing highlights and shadows that give it a three-dimensional appearance.

Below the rendering, there are several interactive controls:

- Phong Shading** (selected)
- Knot** (selected)
- normal mode** (selected)
- Reload Shader Code** (button)
- Ambient reflection (ka):** 1.0 (slider)
- Diffuse reflection (kd):** 0.86 (slider)
- Specular reflection (ks):** 1.0 (slider)
- Shininess:** 126 (slider)
- Ambient color:** (color picker)
- Diffuse Color:** (color picker)
- Specular Color:** (color picker)
- Background Color:** (color picker)
- Light position:** X: 1.2, Y: -0.2, Z: -1 (sliders)

On the right side of the browser window, there are two code editors:

Vertex Shader

```
attribute vec3 position;
attribute vec3 normal;
uniform mat4 projection, modelview, normalMat;
varying vec3 normalInterp;
varying vec3 vertPos;

void main(){
    vec4 vertPos4 = modelview * vec4(position, 1.0);
    vertPos = vec3(vertPos4) / vertPos4.w;
    normalInterp = vec3(normalMat * vec4(normal, 0.0));
    gl_Position = projection * vertPos4;
}
```

Fragment Shader

```
precision mediump float;
varying vec3 normalInterp; // Surface normal
varying vec3 vertPos; // Vertex position
uniform int mode; // Rendering mode
uniform float Ka; // Ambient reflection coefficient
uniform float Kd; // Diffuse reflection coefficient
uniform float Ks; // Specular reflection coefficient
uniform float shininessVal; // Shininess
// Material color
uniform vec3 ambientColor;
uniform vec3 diffuseColor;
uniform vec3 specularColor;
uniform vec3 lightPos; // Light position

void main() {
    vec3 N = normalize(normalInterp);
    vec3 L = normalize(lightPos - vertPos);
```

Ambient + Diffuse + Specular



Ambient+Diffuse+Specular



Ambient+Diffuse



Diffuse+Specular



Ambient Only



Diffuse Only



Specular Only

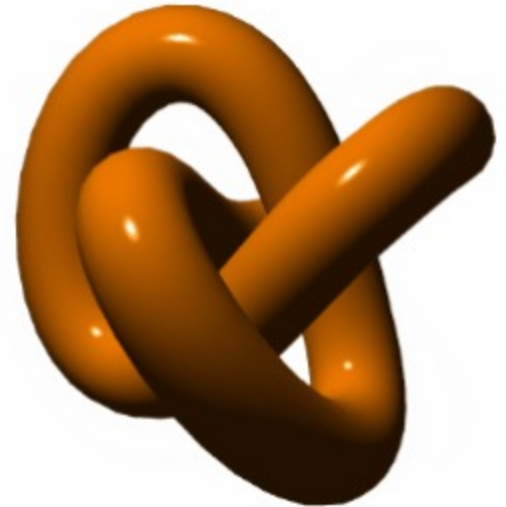
Specular Exponent



Specular exponent $n=4$

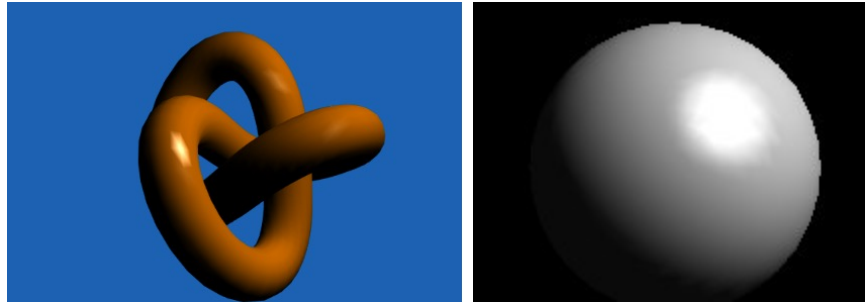


Specular exponent $n=32$



Specular exponent $n=128$

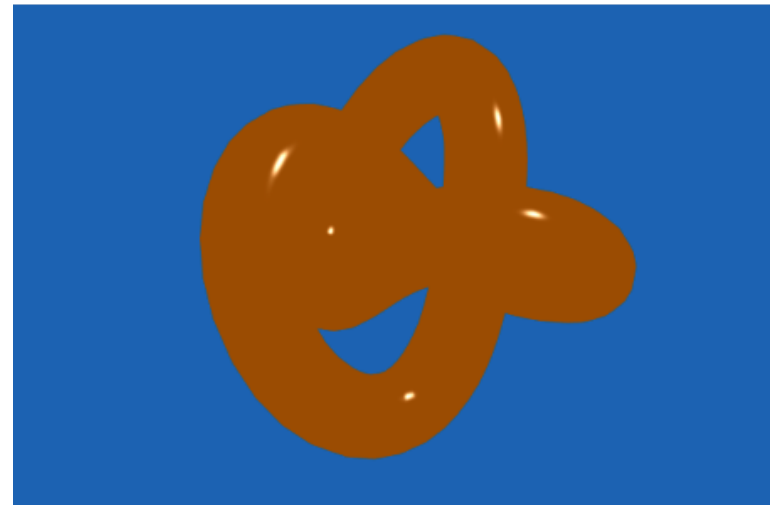
Lighting Coefficients Q



What are the coefficients for this rendering?

- (A) $k_{\text{Ambient}} = 0$, $k_{\text{Diffuse}} = 0$, $k_{\text{Specular}} = 0$
- (B) $k_{\text{Ambient}} = 0$, $k_{\text{Diffuse}} = 0.5$, $k_{\text{Specular}} = 0.5$
- (C) $k_{\text{Ambient}} = 0.5$, $k_{\text{Diffuse}} = 0.5$, $k_{\text{Specular}} = 0$
- (D) $k_{\text{Ambient}} = 0.5$, $k_{\text{Diffuse}} = 0$, $k_{\text{Specular}} = 0.5$
- (E) $k_{\text{Ambient}} = 0$, $k_{\text{Diffuse}} = 0$, $k_{\text{Specular}} = 0.5$

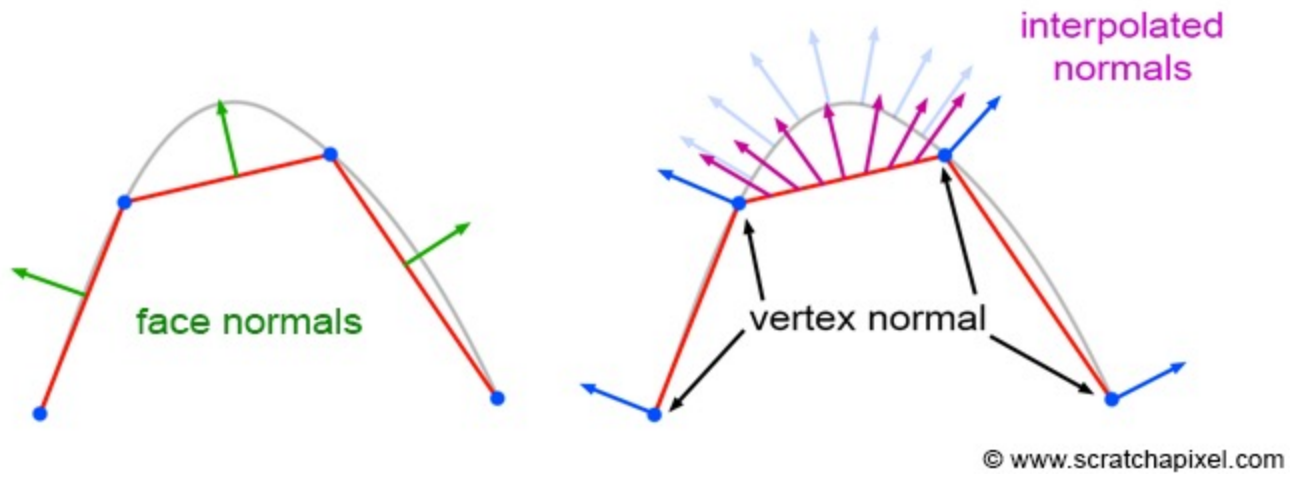
Lighting Coefficients Q



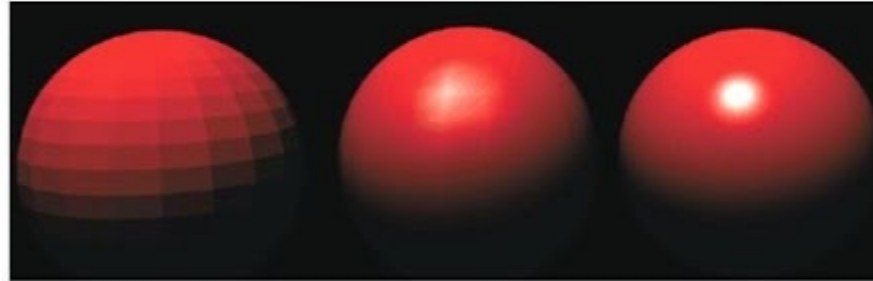
What are the coefficients for this rendering?

- (A) $k_{\text{Ambient}} = 0$, $k_{\text{Diffuse}} = 0$, $k_{\text{Specular}} = 0$
- (B) $k_{\text{Ambient}} = 0$, $k_{\text{Diffuse}} = 0.5$, $k_{\text{Specular}} = 0.5$
- (C) $k_{\text{Ambient}} = 0.5$, $k_{\text{Diffuse}} = 0.5$, $k_{\text{Specular}} = 0$
- (D) $k_{\text{Ambient}} = 0.5$, $k_{\text{Diffuse}} = 0$, $k_{\text{Specular}} = 0.5$
- (E) $k_{\text{Ambient}} = 0$, $k_{\text{Diffuse}} = 0$, $k_{\text{Specular}} = 0.5$

Which Normal?



FLAT SHADING, GOURAUD SHADING & PHONG SHADING

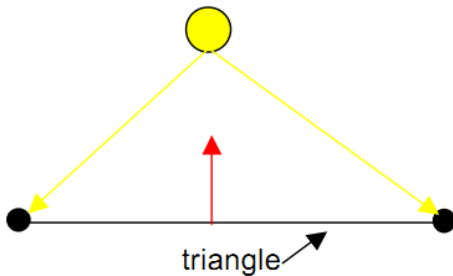


Flat

Gouraud

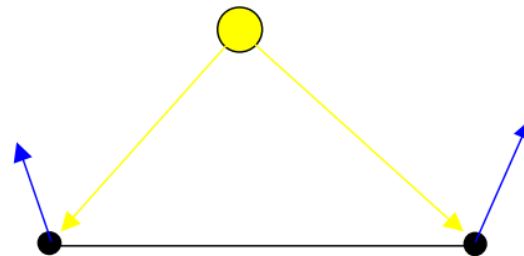
Phong

Flat shading



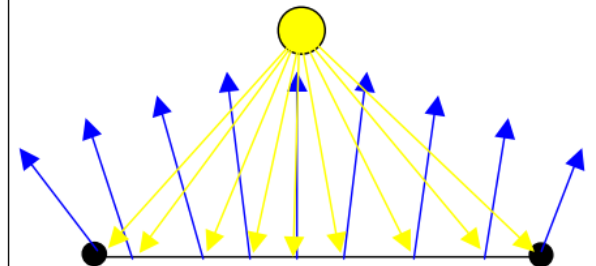
Only the first normal of the triangle is used to compute lighting in the entire triangle.

Gouraud shading



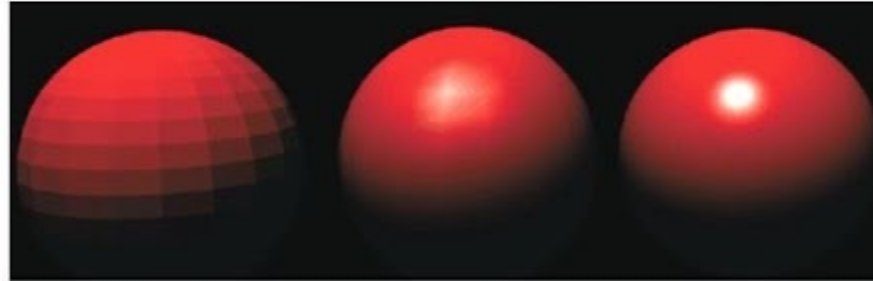
The light intensity is computed at each vertex and interpolated across the surface.

Phong shading



Normals are interpolated across the surface, and the light is computed at each fragment.

FLAT SHADING, GOURAUD SHADING & PHONG SHADING

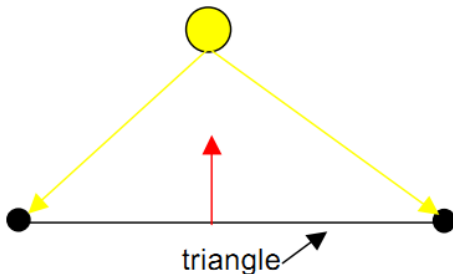


Flat

Gouraud

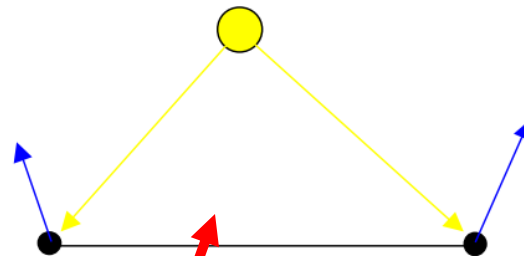
Phong

Flat shading



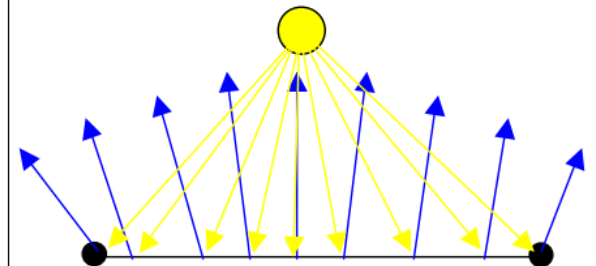
Only the first normal of the triangle is used to compute lighting in the entire triangle.

Gouraud shading



The light intensity is computed at each vertex and interpolated across the surface.

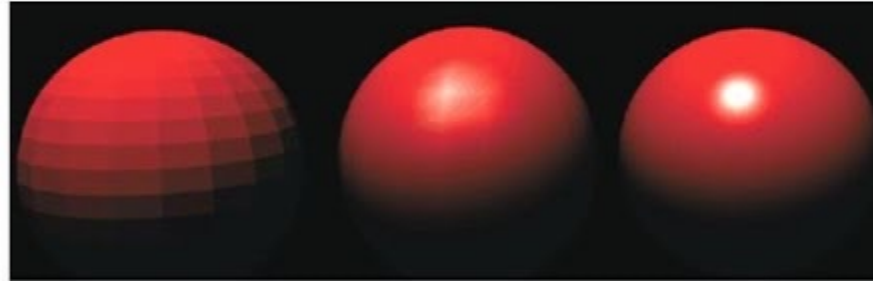
Phong shading



Normals are interpolated across the surface, and the light is computed at each fragment.

Calculate lighting at vertex, interpolate light intensity

FLAT SHADING, GOURAUD SHADING & PHONG SHADING

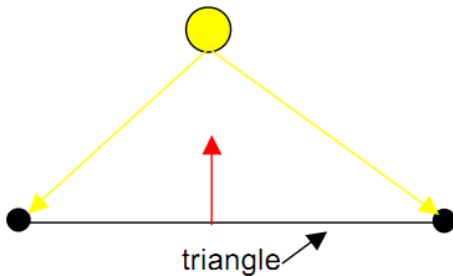


Flat

Gouraud

Phong

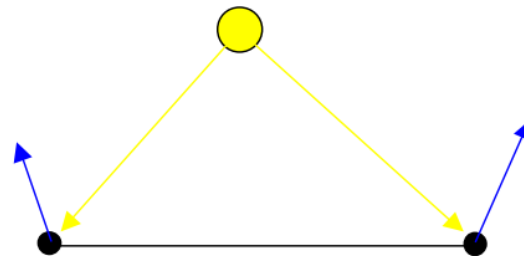
Flat shading



triangle

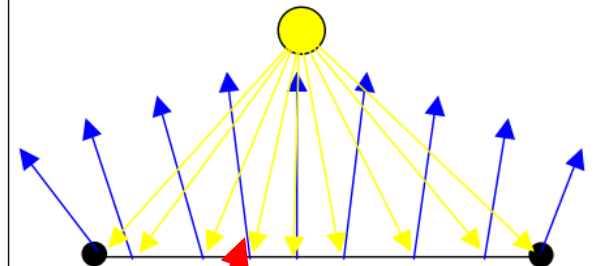
Only the first normal of the triangle is used to compute lighting in the entire triangle.

Gouraud shading



The light intensity is computed at each vertex and interpolated across the surface.

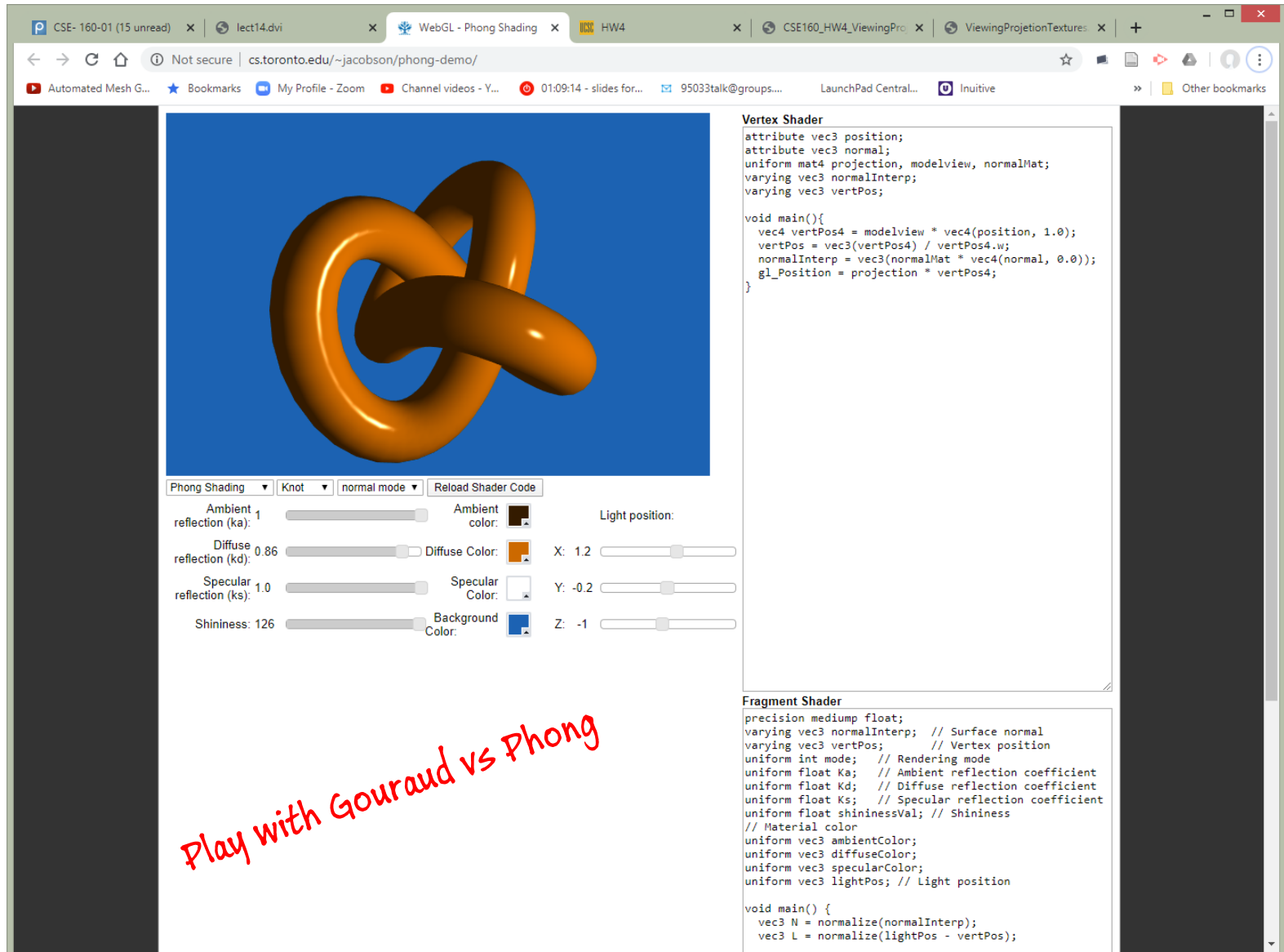
Phong shading



Normals are interpolated across the surface, and the light is computed at each fragment.

Interpolate normal, calculate lighting at fragment

Really great example to play with the parameters:
<http://www.cs.toronto.edu/~jacobson/phong-demo/>



The screenshot displays a web browser window with the URL <http://www.cs.toronto.edu/~jacobson/phong-demo/>. The browser's address bar shows "Not secure | cs.toronto.edu/~jacobson/phong-demo/". The page features a 3D rendering of a knot (a trefoil knot) in a blue environment. Below the rendering, there is a control panel with the following settings:

- Phong Shading (selected)
- Knot (selected)
- normal mode (selected)
- Reload Shader Code (button)
- Ambient reflection (ka): 1.0
- Diffuse reflection (kd): 0.86
- Specular reflection (ks): 1.0
- Shininess: 126
- Ambient color: (dark brown)
- Diffuse Color: (orange)
- Specular Color: (white)
- Background Color: (blue)
- Light position: X: 1.2, Y: -0.2, Z: -1.0

On the right side, there are two code editors for shaders:

Vertex Shader

```
attribute vec3 position;
attribute vec3 normal;
uniform mat4 projection, modelview, normalMat;
varying vec3 normalInterp;
varying vec3 vertPos;

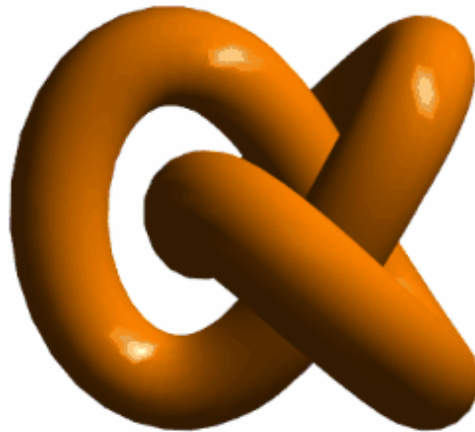
void main(){
    vec4 vertPos4 = modelview * vec4(position, 1.0);
    vertPos = vec3(vertPos4) / vertPos4.w;
    normalInterp = vec3(normalMat * vec4(normal, 0.0));
    gl_Position = projection * vertPos4;
}
```

Fragment Shader

```
precision mediump float;
varying vec3 normalInterp; // Surface normal
varying vec3 vertPos; // Vertex position
uniform int mode; // Rendering mode
uniform float Ka; // Ambient reflection coefficient
uniform float Kd; // Diffuse reflection coefficient
uniform float Ks; // Specular reflection coefficient
uniform float shininessVal; // Shininess
// Material color
uniform vec3 ambientColor;
uniform vec3 diffuseColor;
uniform vec3 specularColor;
uniform vec3 lightPos; // Light position

void main() {
    vec3 N = normalize(normalInterp);
    vec3 L = normalize(lightPos - vertPos);
```

Play with Gouraud vs Phong



Gouraud Shading

Gouraud Shading ▼ Knot ▼ normal mode ▼ Reload Shader Code

Ambient reflection (ka): 1.0 Ambient color:

Diffuse reflection (kd): 1.0 Diffuse Color:

Specular reflection (ks): 1.0 Specular Color:

Shininess: 64 Background Color:

Light position: X: 1 Y: 1 Z: -1

Vertex Shader

```
attribute vec3 position;
attribute vec3 normal;
uniform mat4 projection, modelview, normalMat;
varying vec3 normalInterp;
varying vec3 vertPos;

uniform int mode; // Rendering mode
uniform float Ka; // Ambient reflection coefficient
uniform float Kd; // Diffuse reflection coefficient
uniform float Ks; // Specular reflection coefficient
uniform float shininessVal; // Shininess
// Material color
uniform vec3 ambientColor;
uniform vec3 diffuseColor;
uniform vec3 specularColor;
uniform vec3 lightPos; // Light position
varying vec4 color; //color

void main(){
    vec4 vertPos4 = modelview * vec4(position, 1.0);
    vertPos = vec3(vertPos4) / vertPos4.w;
    normalInterp = vec3(normalMat * vec4(normal, 0.0));
    gl_Position = projection * vertPos4;

    vec3 N = normalize(normalInterp);
    vec3 L = normalize(lightPos - vertPos);
    // Lambert's cosine law
    float lambertian = max(dot(N, L), 0.0);
    float specular = 0.0;
    if(lambertian > 0.0) {
        vec3 R = reflect(-L, N); // Reflected light
        vec3 V = normalize(-vertPos); // Vector to viewer
        // Compute the specular term
        float specAngle = max(dot(R, V), 0.0);
        specular = pow(specAngle, shininessVal);
    }
    color = vec4(Ka * ambientColor +
                Kd * lambertian * diffuseColor +
                Ks * specular * specularColor, 1.0);

    // only ambient
```

Calculate lighting at vertex

Fragment Shader

```
precision mediump float;

varying vec4 color;
void main() {
    gl_FragColor = color;
}
```



Phong Shading

Vertex Shader

```
attribute vec3 position;
attribute vec3 normal;
uniform mat4 projection, modelview, normalMat;
varying vec3 normalInterp;
varying vec3 vertPos;

void main(){
    vec4 vertPos4 = modelview * vec4(position, 1.0);
    vertPos = vec3(vertPos4) / vertPos4.w;
    normalInterp = vec3(normalMat * vec4(normal, 0.0));
    gl_Position = projection * vertPos4;
}
```

Calculate lighting at fragment

Fragment Shader

```
precision mediump float;
varying vec3 normalInterp; // Surface normal
varying vec3 vertPos; // Vertex position
uniform int mode; // Rendering mode
uniform float Ka; // Ambient reflection coefficient
uniform float Kd; // Diffuse reflection coefficient
uniform float Ks; // Specular reflection coefficient
uniform float shininessVal; // Shininess
// Material color
uniform vec3 ambientColor;
uniform vec3 diffuseColor;
uniform vec3 specularColor;
uniform vec3 lightPos; // Light position

void main() {
    vec3 N = normalize(normalInterp);
    vec3 L = normalize(lightPos - vertPos);
```

Phong Shading ▼ Knot ▼ normal mode ▼ Reload Shader Code

Ambient reflection (ka): 1.0 Ambient color:

Diffuse reflection (kd): 1.0 Diffuse Color:

Specular reflection (ks): 1.0 Specular Color:

Shininess: 64 Background Color:

Light position:

X: 1

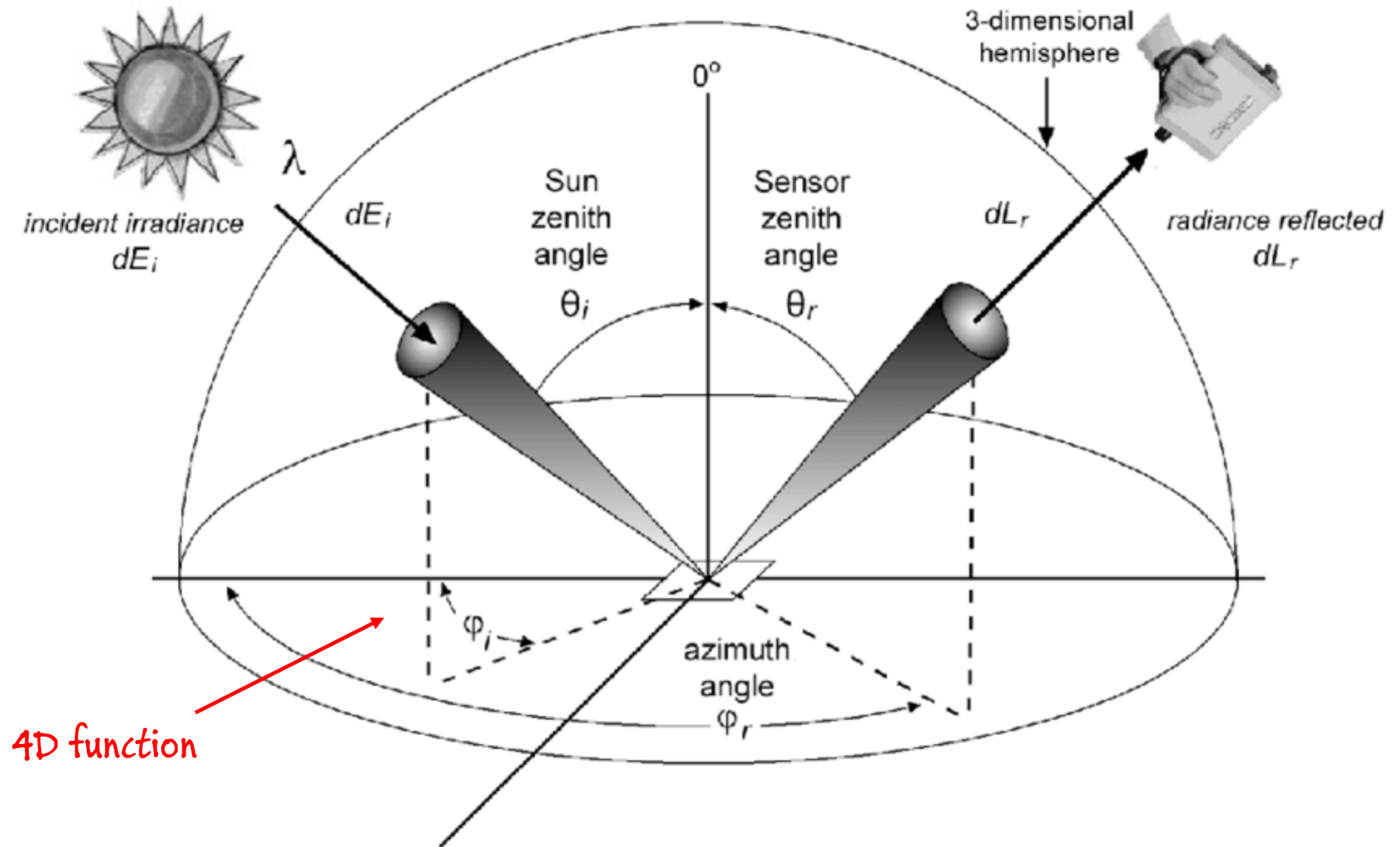
Y: 1

Z: -1

BRDFs



Bidirectional Reflectance Distribution Function (BRDF)



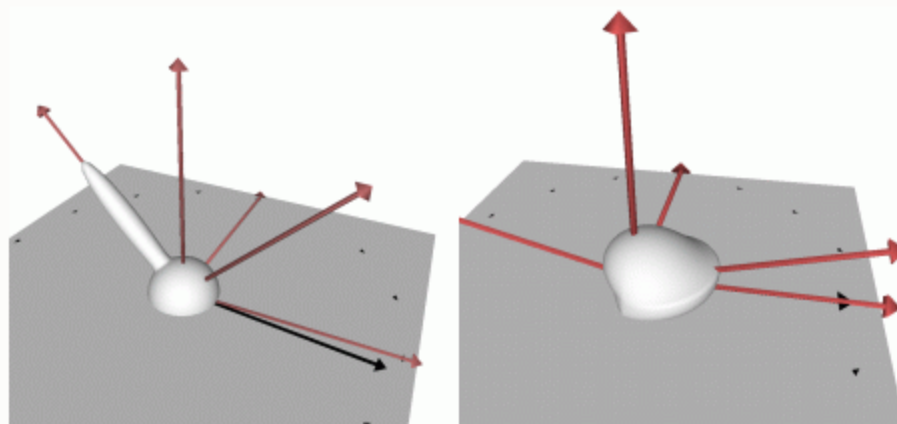


Figure 3: Phong reflection model (left), Oren-Nayar Diffuse Microfacet BRDF (right).

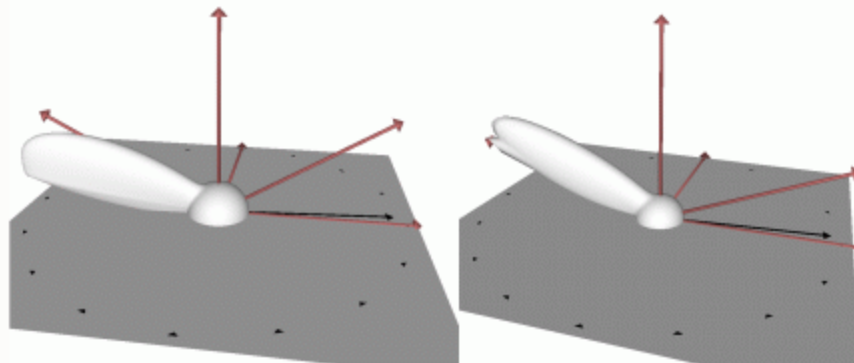
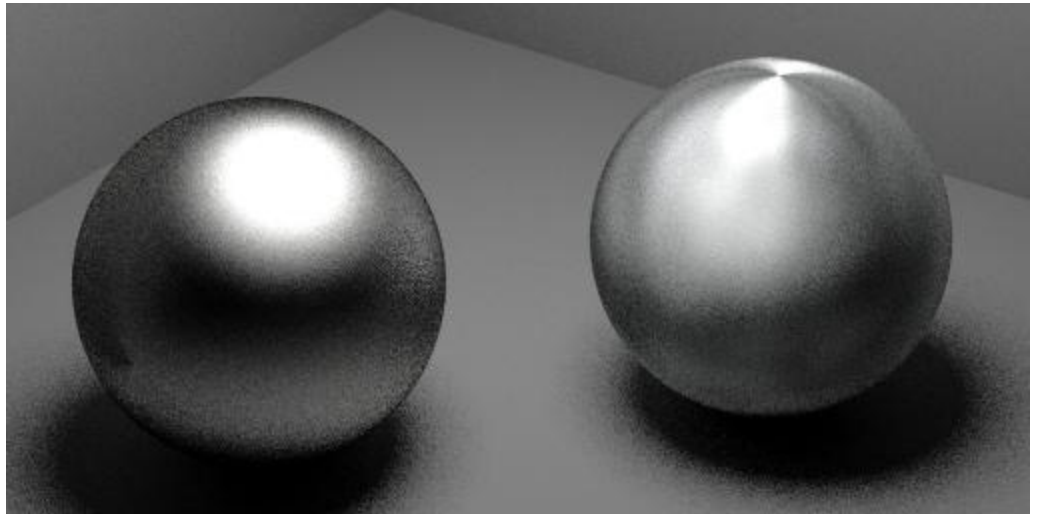
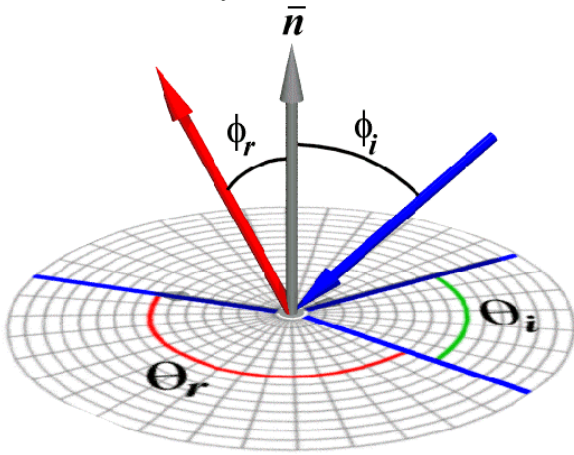
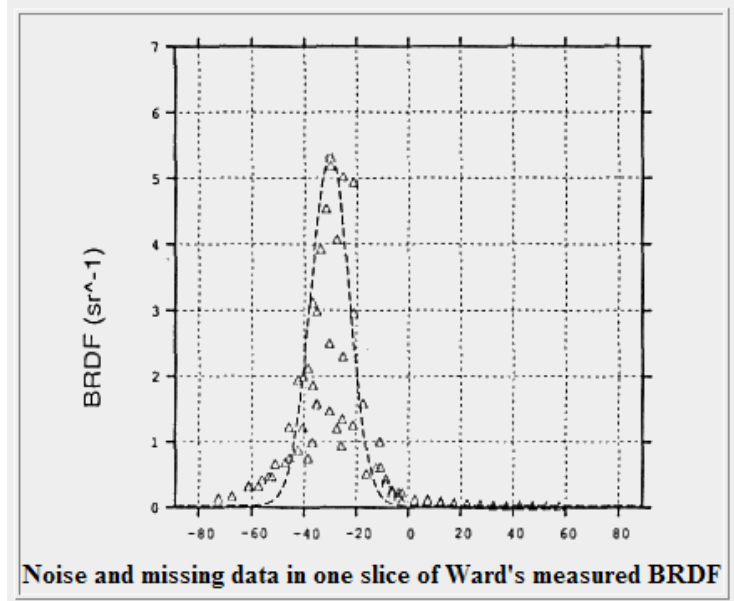
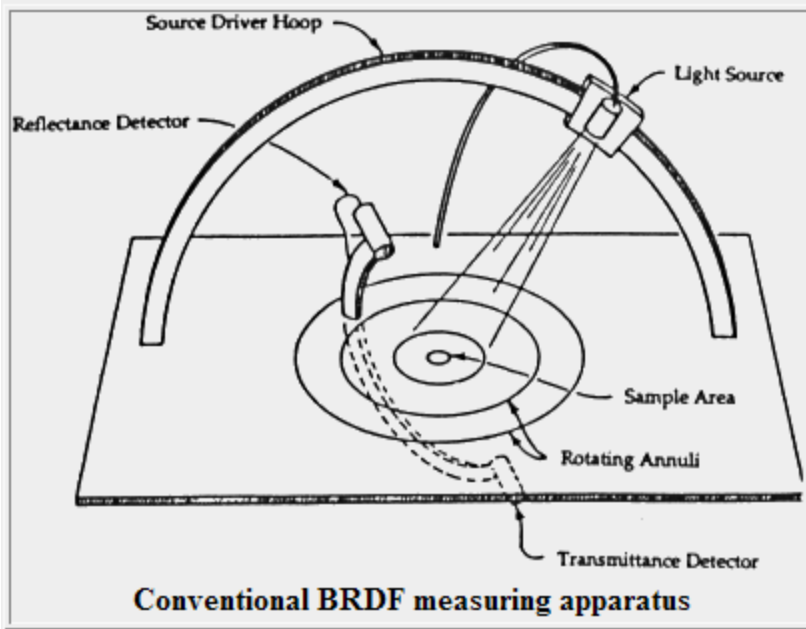


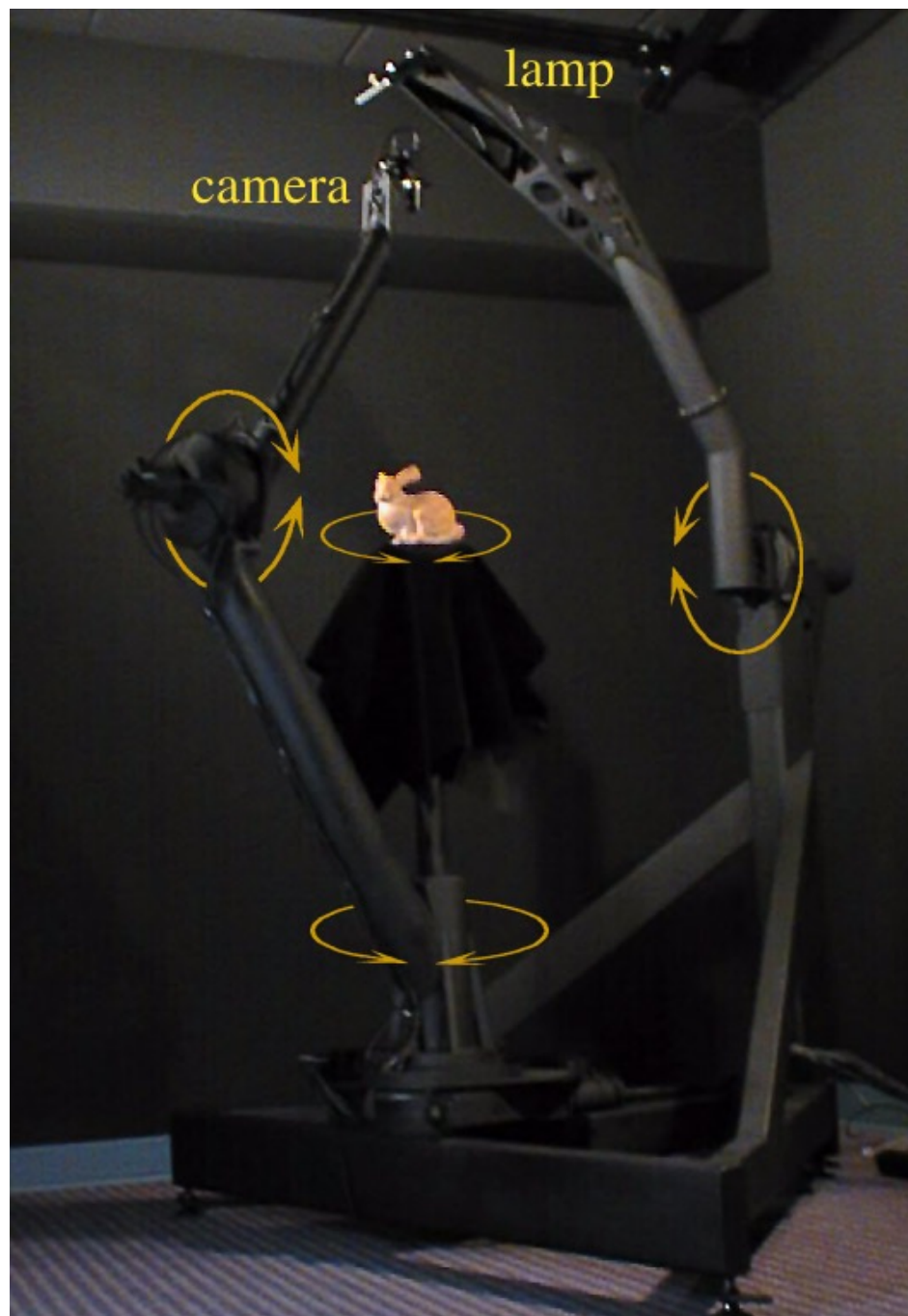
Figure 4: Cook-Torrance Specular Microfacet BRDF (left), He-Torrance Comprehensive Analytic Model (right).

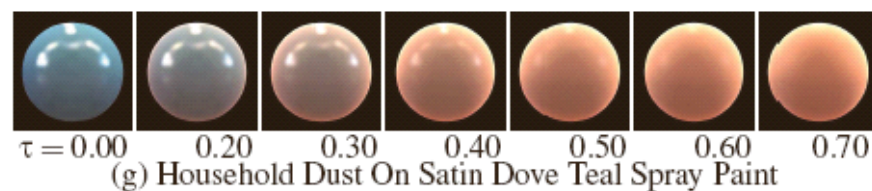
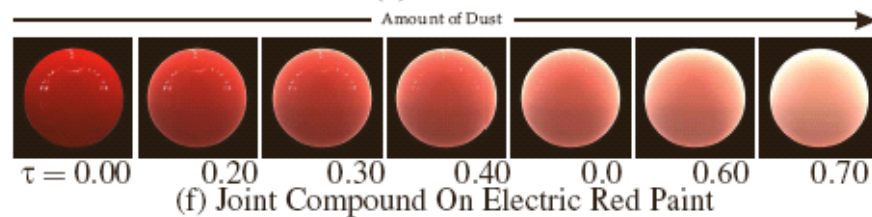
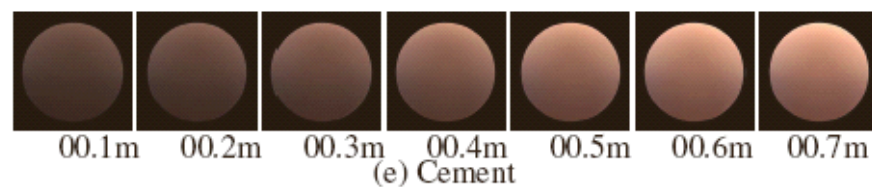
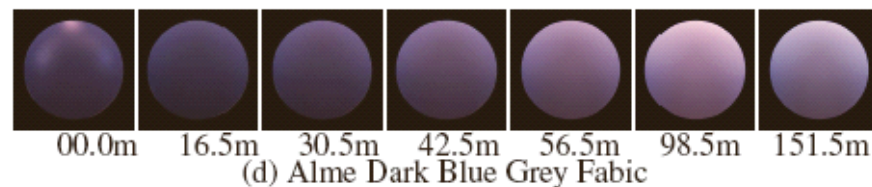
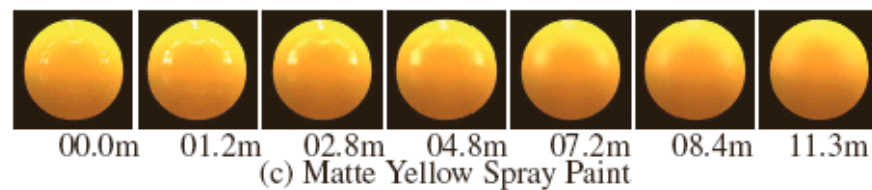
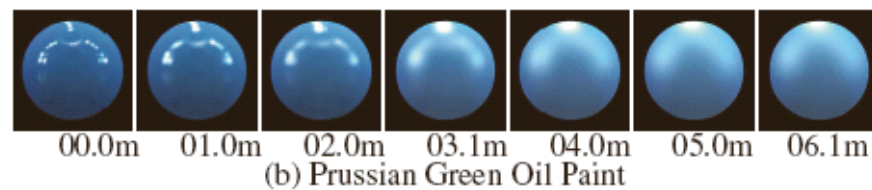
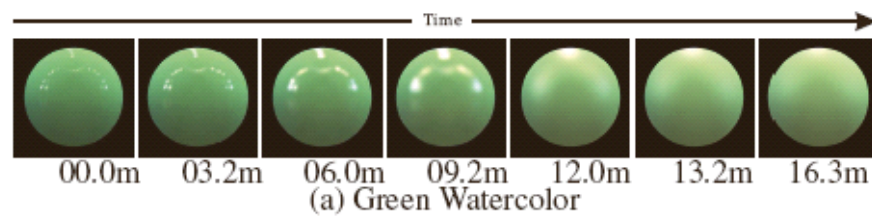
Isotropic vs anisotropic

$$\rho(\theta_r, \phi_r, \theta_i, \phi_i)$$







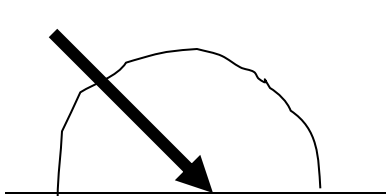


BRDF Q

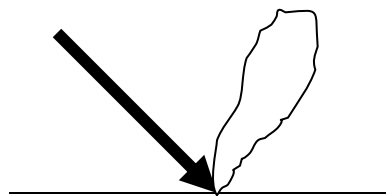
3. You are rendering a scene for the highway department (since they are the government, you have scored a \$1 million dollar contract to make one picture). They insist that your renderings include the shiny little reflective markers along the side of the road. It turns out these are made from *retroreflective* material, meaning that light is reflected mostly back in the direction of the light source.



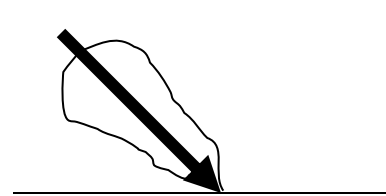
Sketch a goniometric diagram for a retroreflective surface.



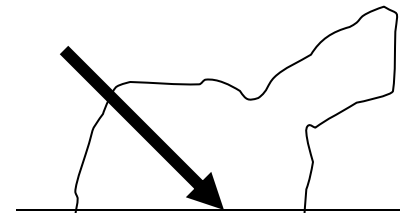
(A)



(B)



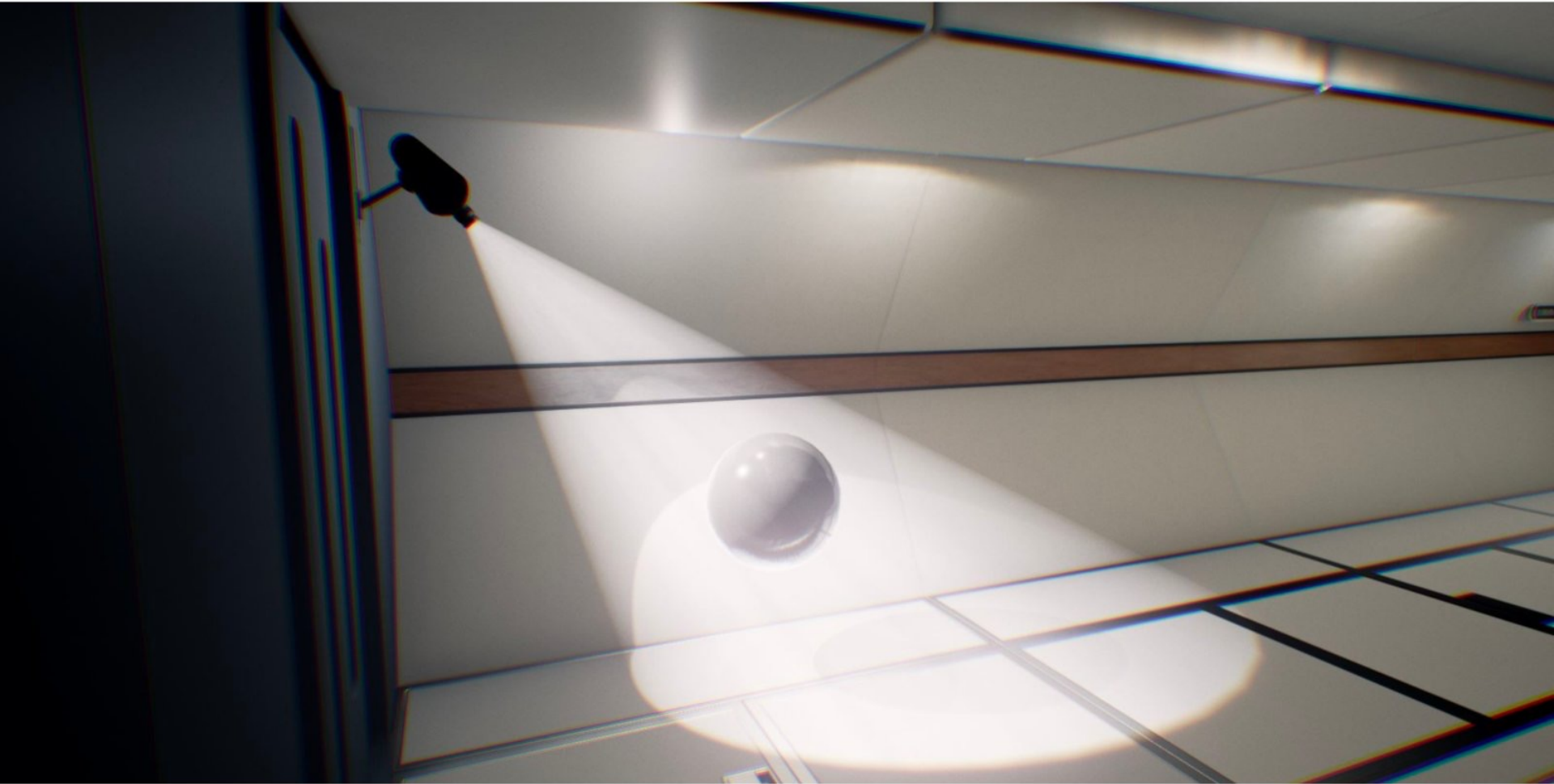
(C)



(D)

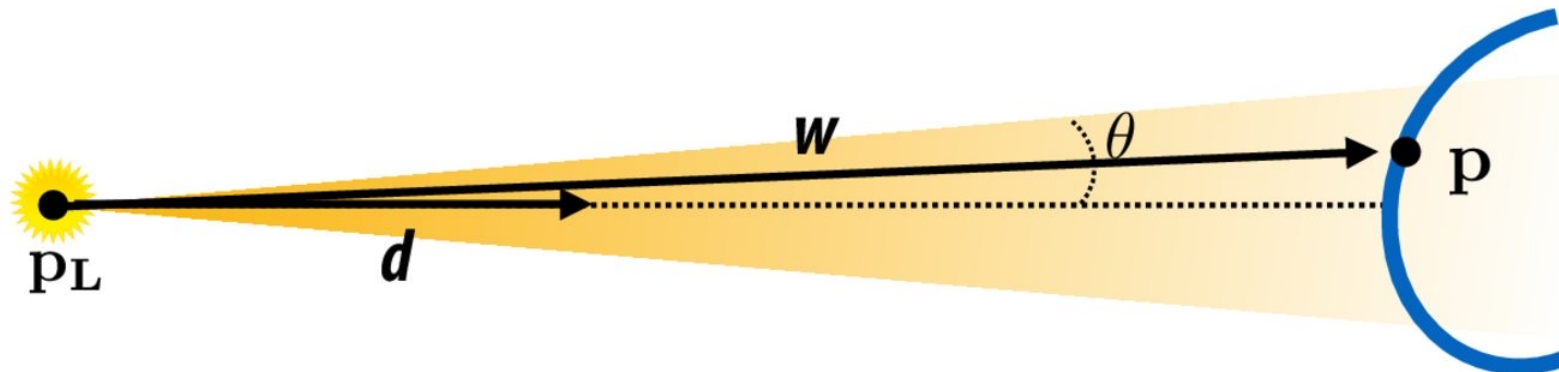
Spot Lights

Spot light



Spot light

(does not emit equally in all directions)



$$\mathbf{w} = \text{normalize}(\mathbf{p} - \mathbf{p}_L)$$

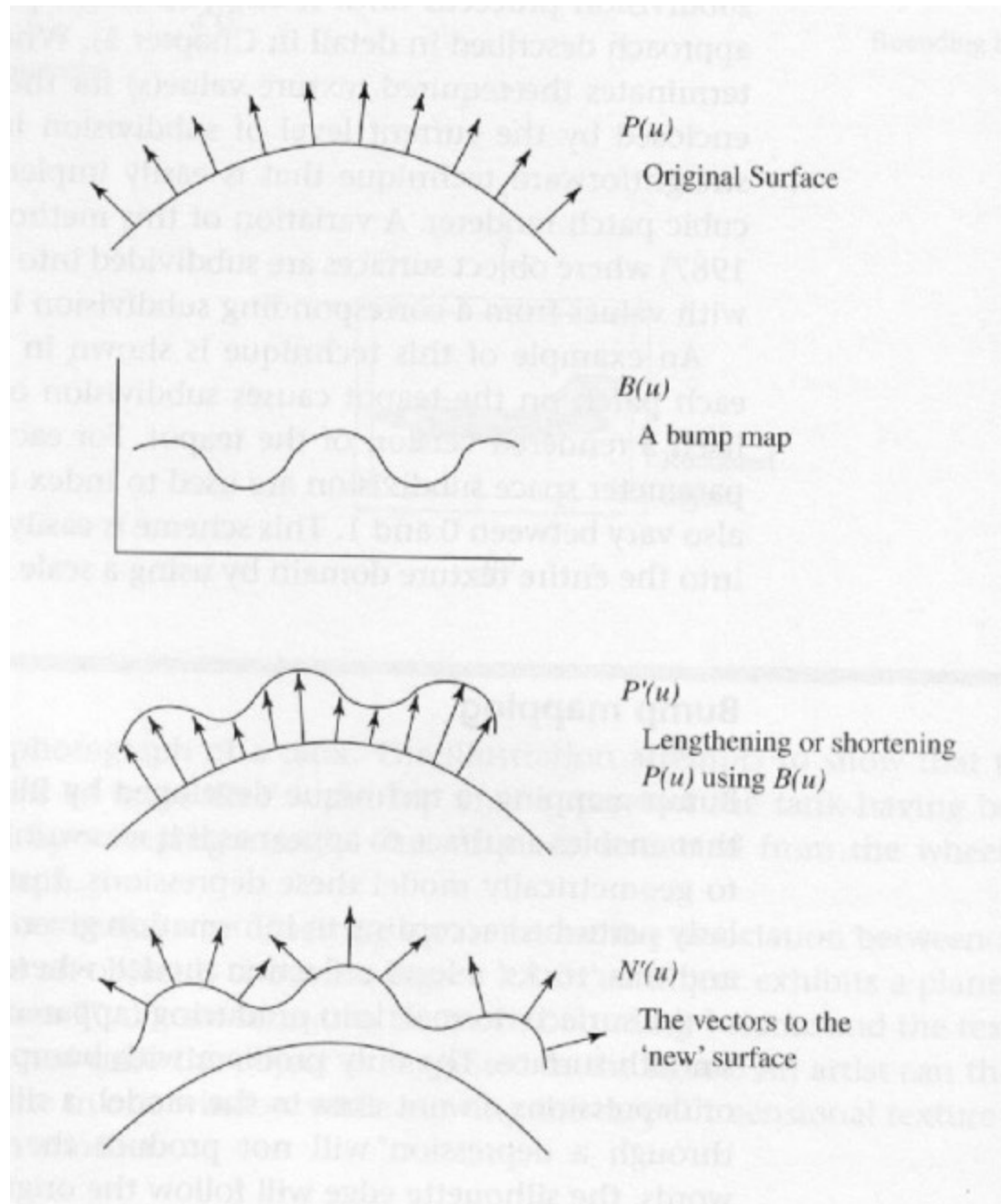
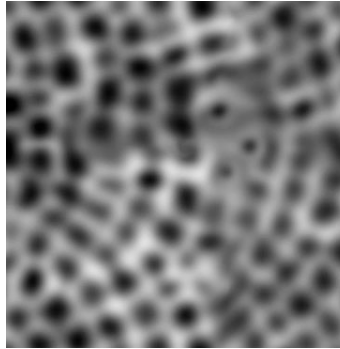
$$L(\mathbf{w}) = 0 \quad \text{if } \mathbf{w} \cdot \mathbf{d} > \cos \theta$$
$$= L_0 \quad \text{otherwise}$$

Or, if spotlight intensity falls off from direction d

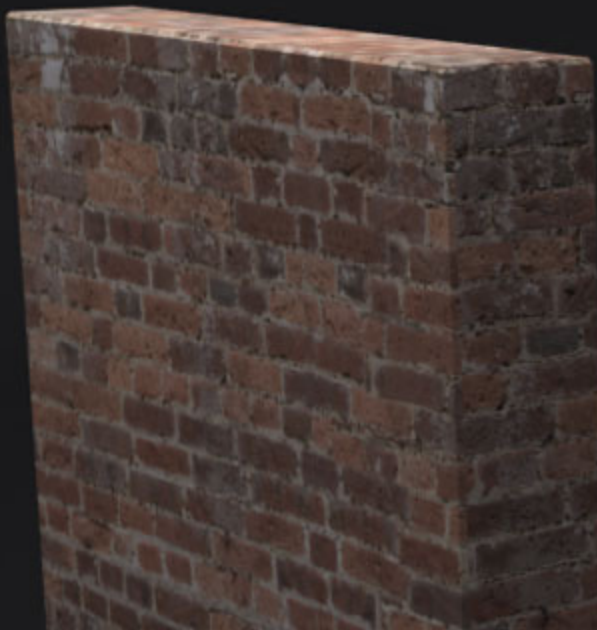
$$L(\mathbf{w}) \approx \mathbf{w} \cdot \mathbf{d}$$

Bump/Normal Maps

Bump Map

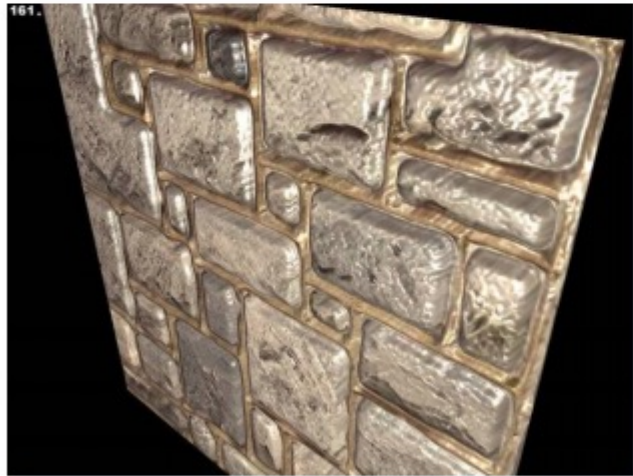
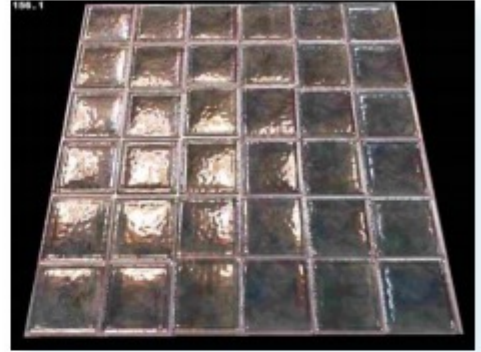
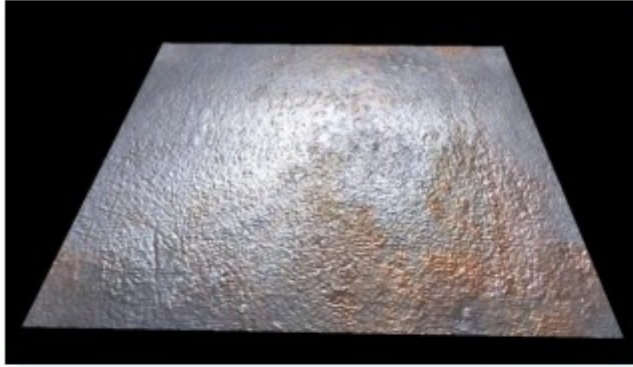


Without bump map

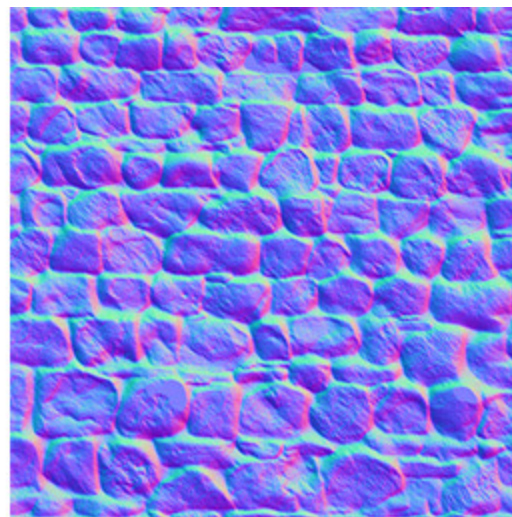
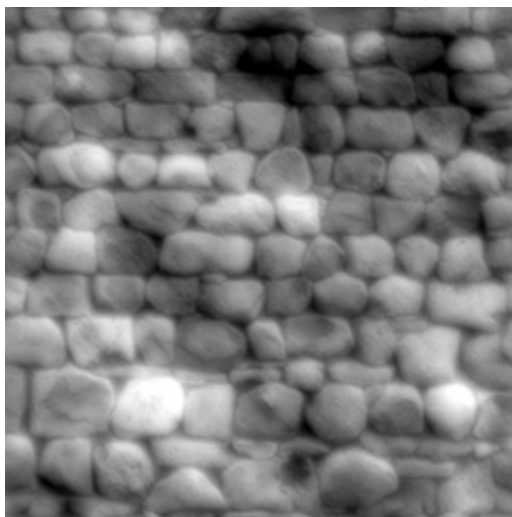


With bump map





Bump Map vs Normal Map



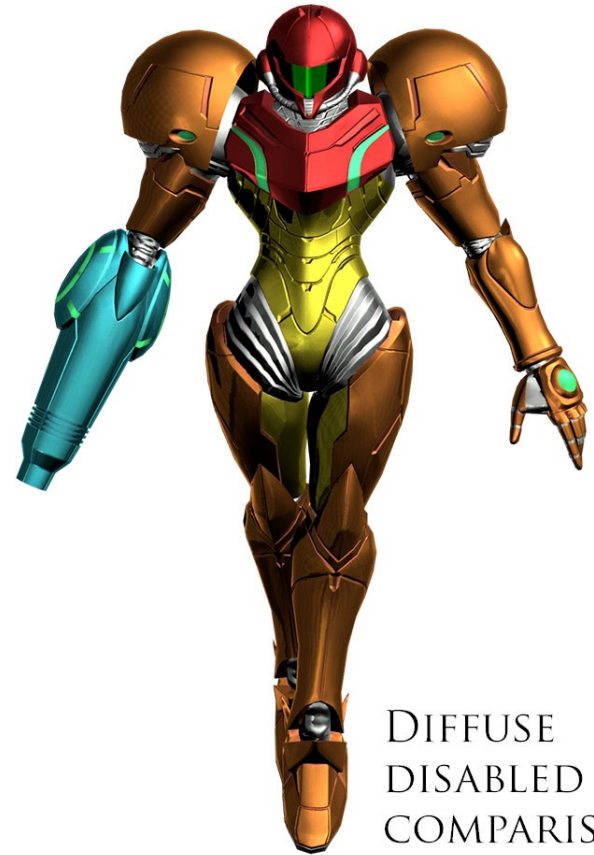
NO NORMAL MAP



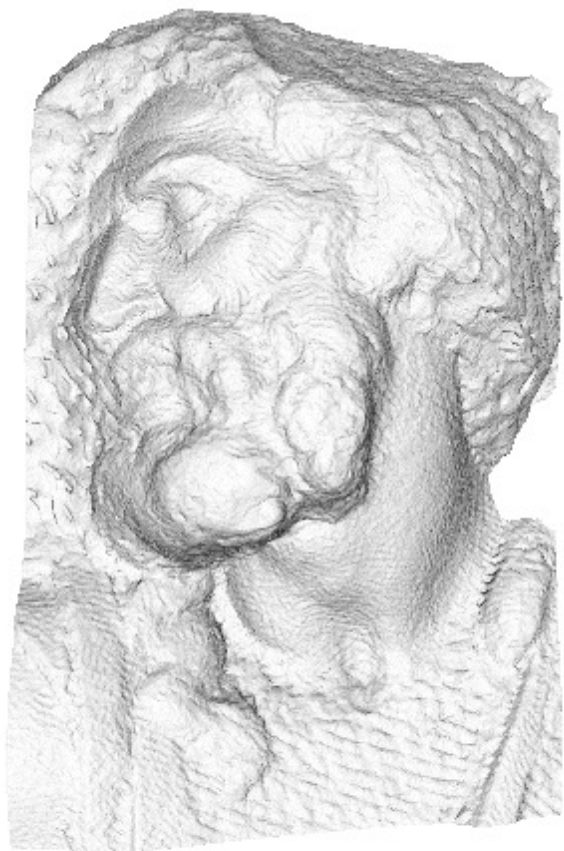
NORMAL
MAP



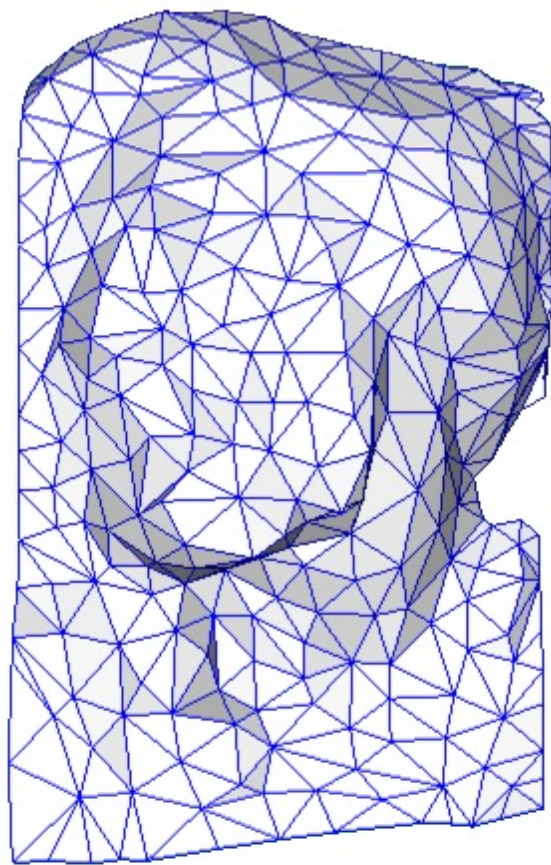
WITH NORMAL MAP



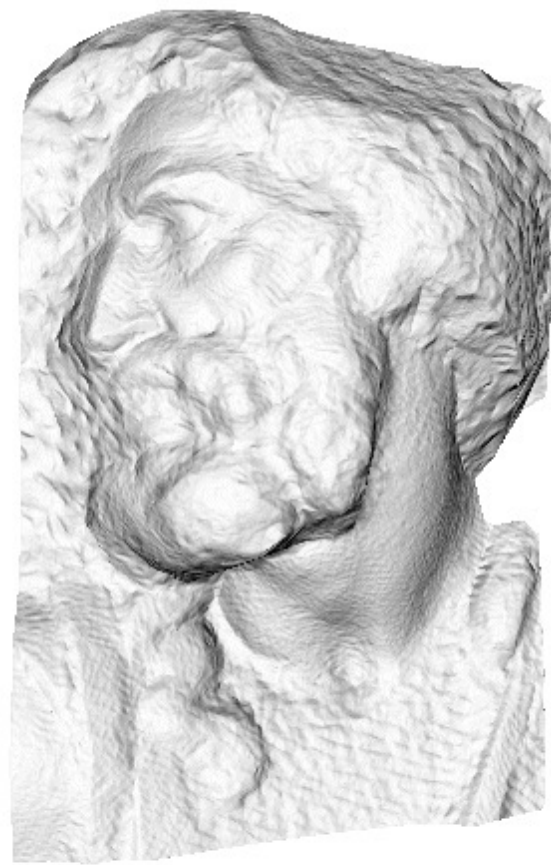
DIFFUSE
DISABLED FOR
COMPARISON



original mesh
4M triangles



simplified mesh
500 triangles



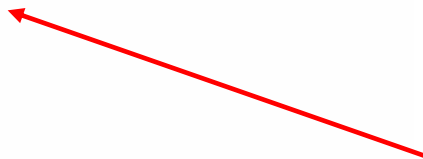
simplified mesh
and normal mapping
500 triangles





WebGL

Listing 8.5 PointLightedCube_perFragment.js



```
1 // PointLightedCube_perFragment.js
2 // Vertex shader program
3 var VSHADER_SOURCE =
4     'attribute vec4 a_Position;\n' +
5     ...
6     'uniform mat4 u_ModelMatrix;\n' + // Model matrix
7     'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
8     'varying vec4 v_Color;\n' +
9     'varying vec3 v_Normal;\n' +
10    'varying vec3 v_Position;\n' +
11    'void main() {\n' +
12    '    gl_Position = u_ModelMatrix * a_Position;\n' +
13    '    // Calculate the vertex position in the world coordinate
14    '    v_Position = vec3(u_ModelMatrix * a_Position);\n' +
15    '    v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
16    '    v_Color = a_Color;\n' +
17    '}\n';
18
19
20
21 // Fragment shader program
22 var FSHADER_SOURCE =
23     ...
24     'uniform vec3 u_LightColor;\n' + // Light color
25     'uniform vec3 u_LightPosition;\n' + // Position of the light source
26     'uniform vec3 u_AmbientLight;\n' + // Ambient light color
27     'varying vec3 v_Normal;\n' +
28     'varying vec3 v_Position;\n' +
29     'varying vec4 v_Color;\n' +
30     'void main() {\n' +
31     '    // Normalize normal because it's interpolated and not 1.0 (length)
32     '    vec3 normal = normalize(v_Normal);\n' +
33     '    // Calculate the light direction and make it 1.0 in length
34     '    vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
35     '    // The dot product of the light direction and the normal
36     '    float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
37     '    // Calculate the final color from diffuse and ambient reflection
38     '    vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
39     '    vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
40     '    gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
41     '}\n';
```

Listing 8.5 PointLightedCube_perFragment.js

```
1 // PointLightedCube_perFragment.js
2 // Vertex shader program
3 var VSHADER_SOURCE =
4     'attribute vec4 a_Position;\n' +
5     ...
6     'uniform mat4 u_ModelMatrix;\n' + // Model matrix
7     'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
8     'varying vec4 v_Color;\n' +
9     'varying vec3 v_Normal;\n' +
10    'varying vec3 v_Position;\n' +
11    'void main() {\n' +
12    '    gl_Position = u_MvpMatrix * a_Position;\n' +
13    '    // Calculate the vertex position in the world coordinate
14    '    v_Position = vec3(u_ModelMatrix * a_Position);\n' +
15    '    v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
16    '    v_Color = a_Color;\n' +
17    '}\n';
18
19 // Fragment shader program
20 var FSHADER_SOURCE =
21     ...
22     'uniform vec3 u_LightColor;\n' + // Light color
23     'uniform vec3 u_LightPosition;\n' + // Position of the light source
24     'uniform vec3 u_AmbientLight;\n' + // Ambient light color
25     'varying vec3 v_Normal;\n' +
26     'varying vec3 v_Position;\n' +
27     'varying vec4 v_Color;\n' +
28     'void main() {\n' +
29     '    // Normalize normal because it's interpolated and not 1.0 (length)
30     '    vec3 normal = normalize(v_Normal);\n' +
31     '    // Calculate the light direction and make it 1.0 in length
32     '    vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
33     '    // The dot product of the light direction and the normal
34     '    float nDotL = max(dot(lightDirection, normal), 0.0);\n' +
35     '    // Calculate the final color from diffuse and ambient reflection
36     '    vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
37     '    vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
38     '    gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
39     '}\n';
```

$\text{ProjectionMatrix} * \text{ViewMatrix} * \text{ModelMatrix}$



Listing 8.5 PointLightedCube_perFragment.js

```
1 // PointLightedCube_perFragment.js
2 // Vertex shader program
3 var VSHADER_SOURCE =
4     'attribute vec4 a_Position;\n' +
5     ...
6     'uniform mat4 u_ModelMatrix;\n' + // Model matrix
7     'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
8     'varying vec4 v_Color;\n' +
9     'varying vec3 v_Normal;\n' +
10    'varying vec3 v_Position;\n' +
11    'void main() {\n' +
12    '    gl_Position = u_MvpMatrix * a_Position;\n' +
13    '    // Calculate the vertex position in the world coordinate
14    '    v_Position = vec3(u_ModelMatrix * a_Position);\n' +
15    '    v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
16    '    v_Color = a_Color;\n' +
17    '}\n';
18
19 // Fragment shader program
20 var FSHADER_SOURCE =
21     ...
22     'uniform vec3 u_LightColor;\n' + // Light color
23     'uniform vec3 u_LightPosition;\n' + // Position of the light source
24     'uniform vec3 u_AmbientLight;\n' + // Ambient light color
25     'varying vec3 v_Normal;\n' +
26     'varying vec3 v_Position;\n' +
27     'varying vec4 v_Color;\n' +
28     'void main() {\n' +
29     '    // Normalize normal because it's interpolated and not 1.0 (length)
30     '    vec3 normal = normalize(v_Normal);\n' +
31     '    // Calculate the light direction and make it 1.0 in length
32     '    vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
33     '    // The dot product of the light direction and the normal
34     '    float nDotL = max(dot(lightDirection, normal), 0.0);\n' +
35     '    // Calculate the final color from diffuse and ambient reflection
36     '    vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
37     '    vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
38     '    gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
39     '}\n';
```

Only use the ModelMatrix to get World Coordinates

Listing 8.5 PointLightedCube_perFragment.js

```
1 // PointLightedCube_perFragment.js
2 // Vertex shader program
3 var VSHADER_SOURCE =
4     'attribute vec4 a_Position;\n' +
5     ...
6     'uniform mat4 u_ModelMatrix;\n' + // Model matrix
7     'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
8     'varying vec4 v_Color;\n' +
9     'varying vec3 v_Normal;\n' +
10    'varying vec3 v_Position;\n' +
11    'void main() {\n' +
12    '    gl_Position = u_ModelMatrix * a_Position;\n' +
13    '    // Calculate the vertex position in the world coordinate
14    '    v_Position = vec3(u_ModelMatrix * a_Position);\n' +
15    '    v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
16    '    v_Color = a_Color;\n' +
17    '}\n';
18
19 // Fragment shader program
20 var FSHADER_SOURCE =
21     ...
22     'uniform vec3 u_LightColor;\n' + // Light color
23     'uniform vec3 u_LightPosition;\n' + // Position of the light source
24     'uniform vec3 u_AmbientLight;\n' + // Ambient light color
25     'varying vec3 v_Normal;\n' +
26     'varying vec3 v_Position;\n' +
27     'varying vec4 v_Color;\n' +
28     'void main() {\n' +
29     '    // Normalize normal because it's interpolated and not 1.0 (length)
30     '    vec3 normal = normalize(v_Normal);\n' +
31     '    // Calculate the light direction and make it 1.0 in length
32     '    vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
33     '    // The dot product of the light direction and the normal
34     '    float nDotL = max(dot(lightDirection, normal), 0.0);\n' +
35     '    // Calculate the final color from diffuse and ambient reflection
36     '    vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
37     '    vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
38     '    gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
39     '}\n';
```

NormalMatrix not ModelMatrix - see book

Listing 8.5 PointLightedCube_perFragment.js

```
1 // PointLightedCube_perFragment.js
2 // Vertex shader program
3 var VSHADER_SOURCE =
4   'attribute vec4 a_Position;\n' +
5   ...
6   'uniform mat4 u_ModelMatrix;\n' + // Model matrix
7   'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
8   'varying vec4 v_Color;\n' +
9   'varying vec3 v_Normal;\n' +
10  'varying vec3 v_Position;\n' +
11  'void main() {\n' +
12  '  gl_Position = u_ModelMatrix * a_Position;\n' +
13  '  // Calculate the vertex position in the world coordinate
14  '  v_Position = vec3(u_ModelMatrix * a_Position);\n' +
15  '  v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
16  '  v_Color = a_Color;\n' +
17  '}\n';
18
19 // Fragment shader program
20 var FSHADER_SOURCE =
21   ...
22   'uniform vec3 u_LightColor;\n' + // Light color
23   'uniform vec3 u_LightPosition;\n' + // Position of the light source
24   'uniform vec3 u_AmbientLight;\n' + // Ambient light color
25   'varying vec3 v_Normal;\n' +
26   'varying vec3 v_Position;\n' +
27   'varying vec4 v_Color;\n' +
28   'void main() {\n' +
29   '  // Normalize normal because it's interpolated and not 1.0 (length)
30   '  vec3 normal = normalize(v_Normal);\n' +
31   '  // Calculate the light direction and make it 1.0 in length
32   '  vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
33   '  // The dot product of the light direction and the normal
34   '  float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
35   '  // Calculate the final color from diffuse and ambient reflection
36   '  vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
37   '  vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
38   '  gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
39   '}\n';
```

Interpolate these things in the fragment shader

Listing 8.5 PointLightedCube_perFragment.js

```
1 // PointLightedCube_perFragment.js
2 // Vertex shader program
3 var VSHADER_SOURCE =
4   'attribute vec4 a_Position;\n' +
5   ...
6   'uniform mat4 u_ModelMatrix;\n' + // Model matrix
7   'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
8   'varying vec4 v_Color;\n' +
9   'varying vec3 v_Normal;\n' +
10  'varying vec3 v_Position;\n' +
11  'void main() {\n' +
12  '  gl_Position = u_ModelMatrix * a_Position;\n' +
13    // Calculate the vertex position in the world coordinate
14  '  v_Position = vec3(u_ModelMatrix * a_Position);\n' +
15  '  v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
16  '  v_Color = a_Color;\n' +
17  '}\n';
18
19 // Fragment shader program
20 var FSHADER_SOURCE =
21   ...
22   'uniform vec3 u_LightColor;\n' + // Light color
23   'uniform vec3 u_LightPosition;\n' + // Position of the light source
24   'uniform vec3 u_AmbientLight;\n' + // Ambient light color
25   'varying vec3 v_Normal;\n' +
26   'varying vec3 v_Position;\n' +
27   'varying vec4 v_Color;\n' +
28   'void main() {\n' +
29     // Normalize normal because it's interpolated and not 1.0 (length)
30     '  vec3 normal = normalize(v_Normal);\n' +
31     // Calculate the light direction and make it 1.0 in length
32     '  vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
33     // The dot product of the light direction and the normal
34     '  float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
35     // Calculate the final color from diffuse and ambient reflection
36     '  vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
37     '  vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
38     '  gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
39     '}\n';
```

Normalize again since interpolation
might mess up the length

Listing 8.5 PointLightedCube_perFragment.js

```
1 // PointLightedCube_perFragment.js
2 // Vertex shader program
3 var VSHADER_SOURCE =
4     'attribute vec4 a_Position;\n' +
5     ...
6     'uniform mat4 u_ModelMatrix;\n' + // Model matrix
7     'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
8     'varying vec4 v_Color;\n' +
9     'varying vec3 v_Normal;\n' +
10    'varying vec3 v_Position;\n' +
11    'void main() {\n' +
12    '    gl_Position = u_ModelMatrix * a_Position;\n' +
13    '    // Calculate the vertex position in the world coordinate
14    '    v_Position = vec3(u_ModelMatrix * a_Position);\n' +
15    '    v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
16    '    v_Color = a_Color;\n' +
17    '}\n';
18
19 // Fragment shader program
20 var FSHADER_SOURCE =
21     ...
22     'uniform vec3 u_LightColor;\n' + // Light color
23     'uniform vec3 u_LightPosition;\n' + // Position of the light source
24     'uniform vec3 u_AmbientLight;\n' + // Ambient light color
25     'varying vec3 v_Normal;\n' +
26     'varying vec3 v_Position;\n' +
27     'varying vec4 v_Color;\n' +
28     'void main() {\n' +
29     '    // Normalize normal because it's interpolated and not 1.0 (length)
30     '    vec3 normal = normalize(v_Normal);\n' +
31     '    // Calculate the light direction and make it 1.0 in length
32     '    vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
33     '    // The dot product of the light direction and the normal
34     '    float nDotL = max(dot(lightDirection, normal), 0.0);\n' +
35     '    // Calculate the final color from diffuse and ambient reflection
36     '    vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
37     '    vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
38     '    gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
39     '}\n';
```

LightDirection



Listing 8.5 PointLightedCube_perFragment.js

```
1 // PointLightedCube_perFragment.js
2 // Vertex shader program
3 var VSHADER_SOURCE =
4     'attribute vec4 a_Position;\n' +
5     ...
6     'uniform mat4 u_ModelMatrix;\n' + // Model matrix
7     'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
8     'varying vec4 v_Color;\n' +
9     'varying vec3 v_Normal;\n' +
10    'varying vec3 v_Position;\n' +
11    'void main() {\n' +
12    '    gl_Position = u_ModelMatrix * a_Position;\n' +
13    '    // Calculate the vertex position in the world coordinate
14    '    v_Position = vec3(u_ModelMatrix * a_Position);\n' +
15    '    v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
16    '    v_Color = a_Color;\n' +
17    '}\n';
18
19 // Fragment shader program
20 var FSHADER_SOURCE =
21     ...
22     'uniform vec3 u_LightColor;\n' + // Light color
23     'uniform vec3 u_LightPosition;\n' + // Position of the light source
24     'uniform vec3 u_AmbientLight;\n' + // Ambient light color
25     'varying vec3 v_Normal;\n' +
26     'varying vec3 v_Position;\n' +
27     'varying vec4 v_Color;\n' +
28     'void main() {\n' +
29     '    // Normalize normal because it's interpolated and not 1.0 (length)
30     '    vec3 normal = normalize(v_Normal);\n' +
31     '    // Calculate the light direction and make it 1.0 in length
32     '    vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
33     '    // The dot product of the light direction and the normal
34     '    float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
35     '    // Calculate the final color from diffuse and ambient reflection
36     '    vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
37     '    vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
38     '    gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
39     '}\n';
```

N dot L !!!

Listing 8.5 PointLightedCube_perFragment.js

```
1 // PointLightedCube_perFragment.js
2 // Vertex shader program
3 var VSHADER_SOURCE =
4     'attribute vec4 a_Position;\n' +
5     ...
6     'uniform mat4 u_ModelMatrix;\n' + // Model matrix
7     'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
8     'varying vec4 v_Color;\n' +
9     'varying vec3 v_Normal;\n' +
10    'varying vec3 v_Position;\n' +
11    'void main() {\n' +
12    '    gl_Position = u_ModelMatrix * a_Position;\n' +
13    '    // Calculate the vertex position in the world coordinate
14    '    v_Position = vec3(u_ModelMatrix * a_Position);\n' +
15    '    v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
16    '    v_Color = a_Color;\n' +
17    '}\n';
18
19 // Fragment shader program
20 var FSHADER_SOURCE =
21     ...
22     'uniform vec3 u_LightColor;\n' + // Light color
23     'uniform vec3 u_LightPosition;\n' + // Position of the light source
24     'uniform vec3 u_AmbientLight;\n' + // Ambient light color
25     'varying vec3 v_Normal;\n' +
26     'varying vec3 v_Position;\n' +
27     'varying vec4 v_Color;\n' +
28     'void main() {\n' +
29     '    // Normalize normal because it's interpolated and not 1.0 (length)
30     '    vec3 normal = normalize(v_Normal);\n' +
31     '    // Calculate the light direction and make it 1.0 in length
32     '    vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
33     '    // The dot product of the light direction and the normal
34     '    float nDotL = max(dot(lightDirection, normal), 0.0);\n' +
35     '    // Calculate the final color from diffuse and ambient reflection
36     '    vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
37     '    vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
38     '    gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
39     '}\n';
```

Light color

Triangle vertex color

Listing 8.5 PointLightedCube_perFragment.js

```
1 // PointLightedCube_perFragment.js
2 // Vertex shader program
3 var VSHADER_SOURCE =
4     'attribute vec4 a_Position;\n' +
5     ...
6     'uniform mat4 u_ModelMatrix;\n' + // Model matrix
7     'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
8     'varying vec4 v_Color;\n' +
9     'varying vec3 v_Normal;\n' +
10    'varying vec3 v_Position;\n' +
11    'void main() {\n' +
12    '    gl_Position = u_ModelMatrix * a_Position;\n' +
13    '    // Calculate the vertex position in the world coordinate
14    '    v_Position = vec3(u_ModelMatrix * a_Position);\n' +
15    '    v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
16    '    v_Color = a_Color;\n' +
17    '}\n';
18
19 // Fragment shader program
20 var FSHADER_SOURCE =
21     ...
22     'uniform vec3 u_LightColor;\n' + // Light color
23     'uniform vec3 u_LightPosition;\n' + // Position of the light source
24     'uniform vec3 u_AmbientLight;\n' + // Ambient light color
25     'varying vec3 v_Normal;\n' +
26     'varying vec3 v_Position;\n' +
27     'varying vec4 v_Color;\n' +
28     'void main() {\n' +
29     '    // Normalize normal because it's interpolated and not 1.0 (length)
30     '    vec3 normal = normalize(v_Normal);\n' +
31     '    // Calculate the light direction and make it 1.0 in length
32     '    vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
33     '    // The dot product of the light direction and the normal
34     '    float nDotL = max(dot(lightDirection, normal), 0.0);\n' +
35     '    // Calculate the final color from diffuse and ambient reflection
36     '    vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
37     '    vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
38     '    gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
39     '}\n';
```

A4: Need to add Specular

Administrative

Q&A

End