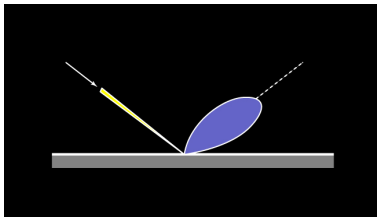
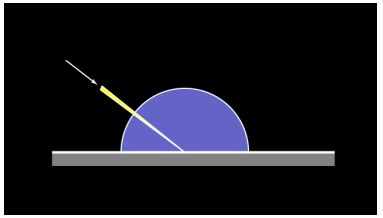
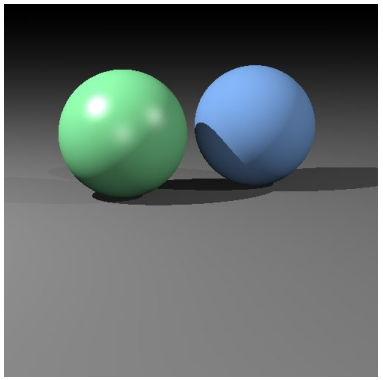
Lighting - CSE160 – Nov 12

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









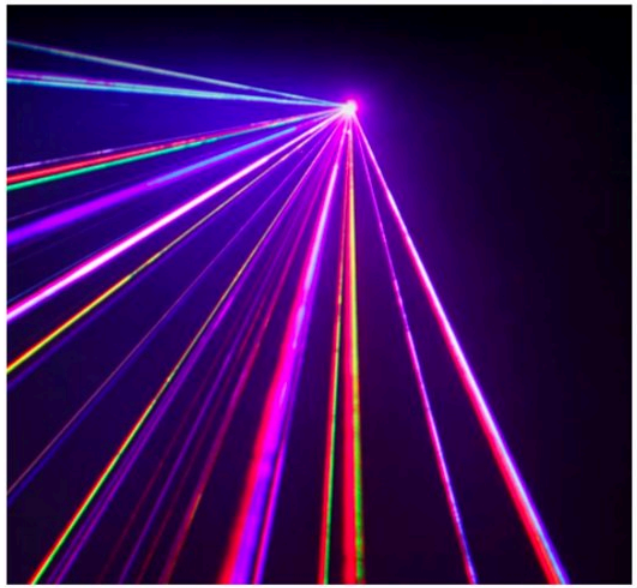
Light and Surfaces

Lighting



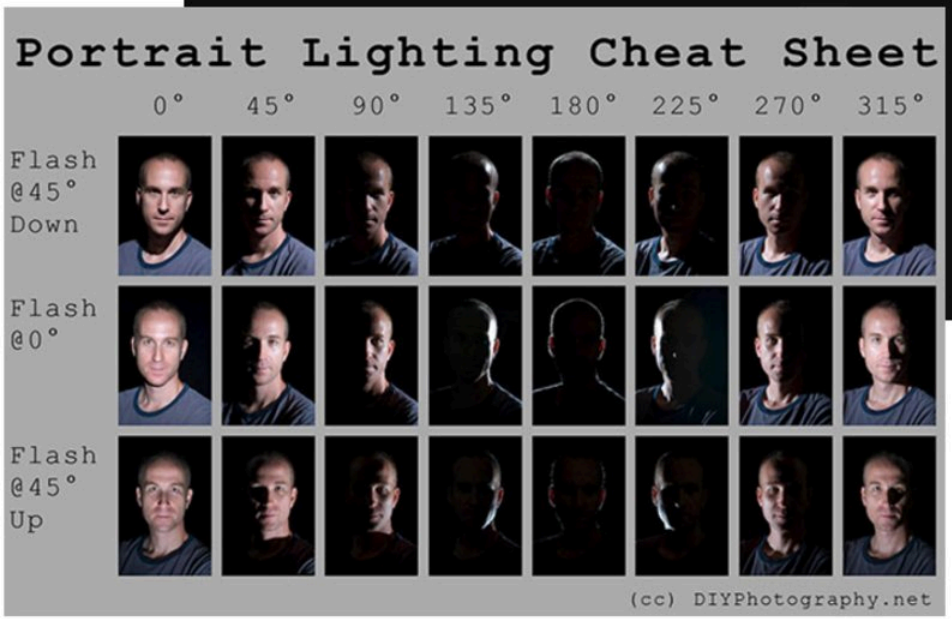






Lighting




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

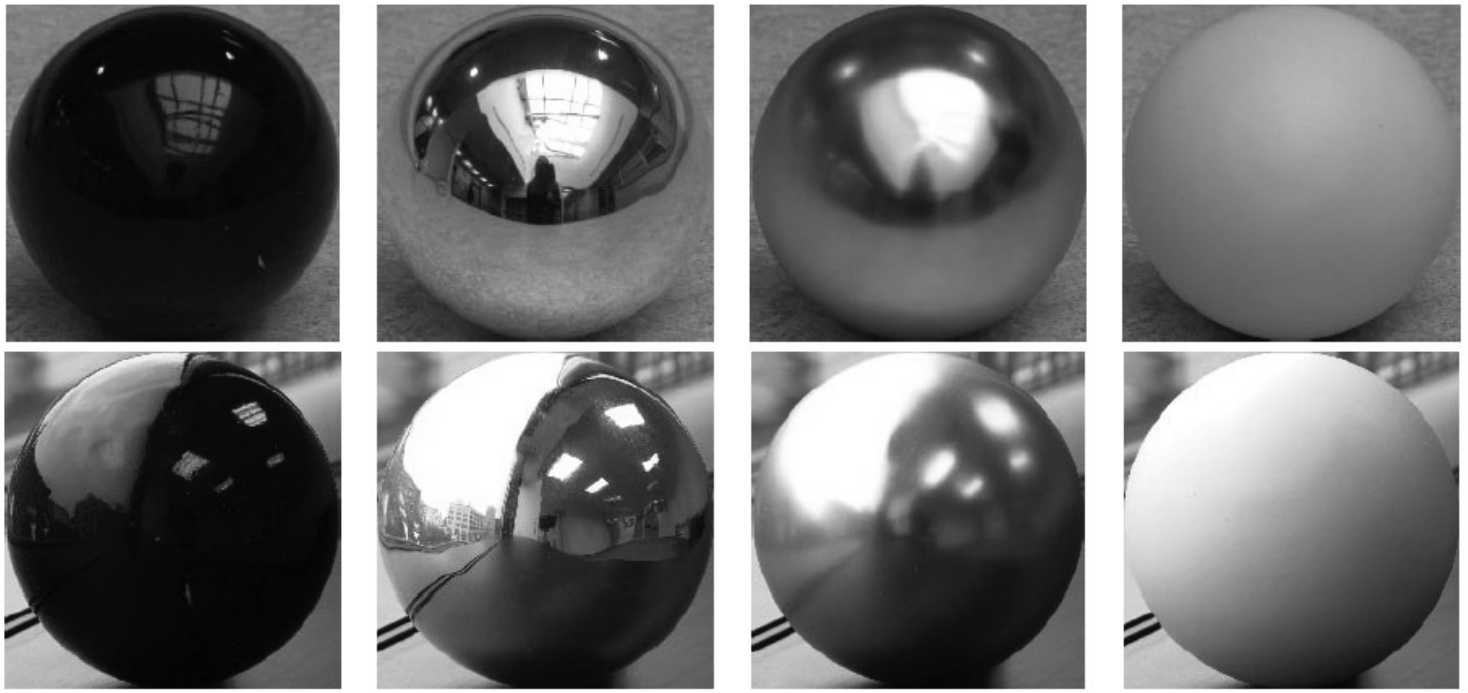


[Philip Greenspun]

© 2004 Steve Marschner

Recognizing materials

- Human visual system is quite good at understanding shading



[Dror, Adelson, & Willsky]

© 2004 Steve Marschner

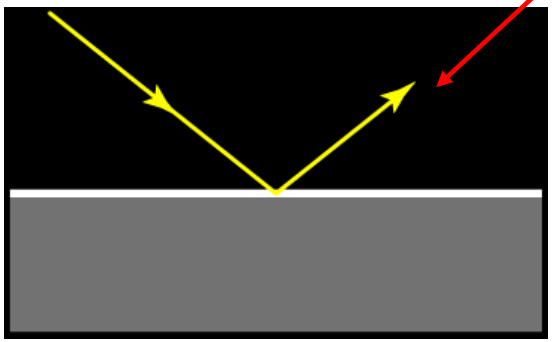
Simple materials



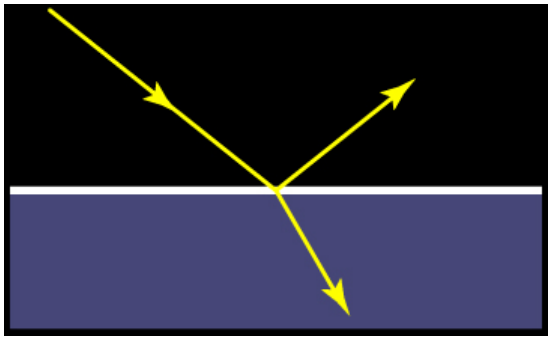


metal

Goniometric diagram



dielectric

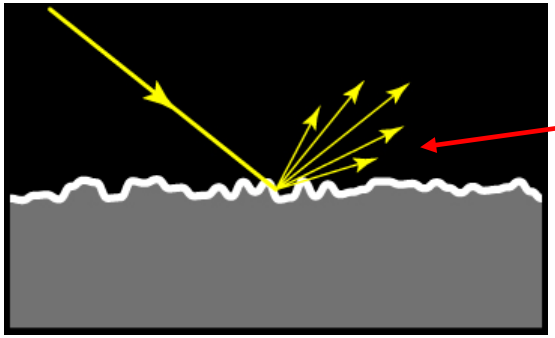


© 2004 Steve Marschner

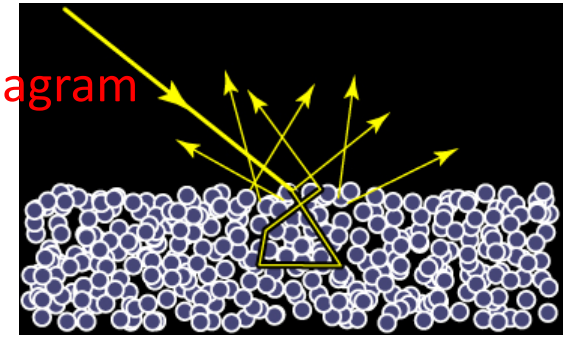
Adding microgeometry





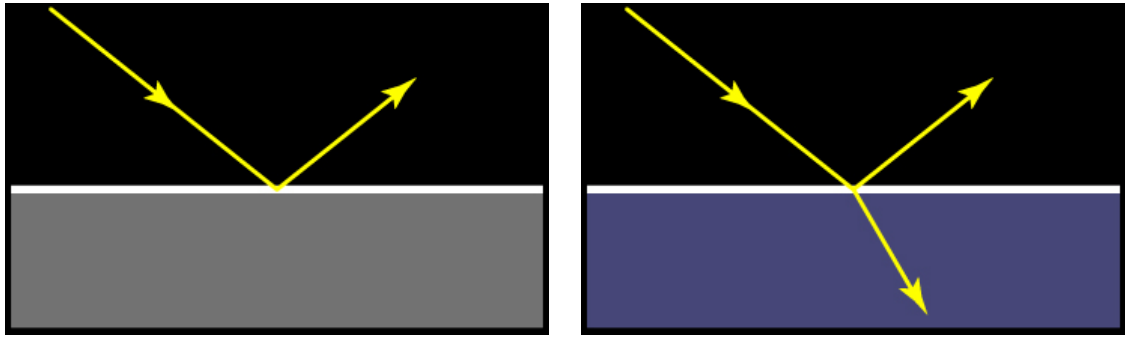


Goniometric diagram

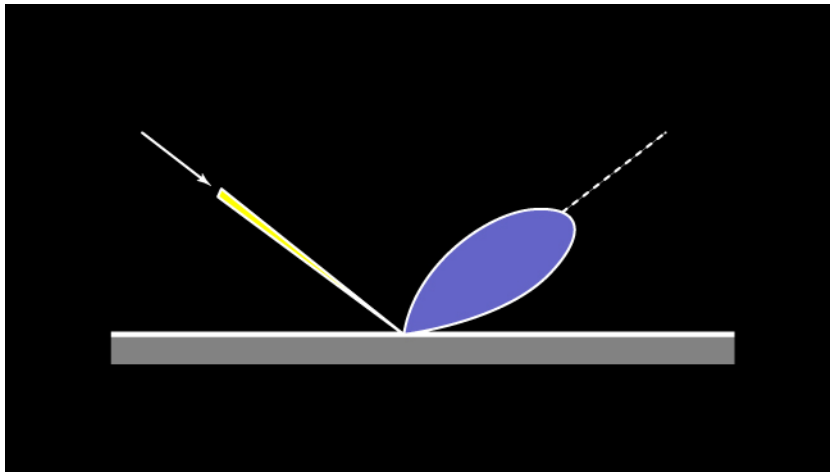


© 2004 Steve Marschner

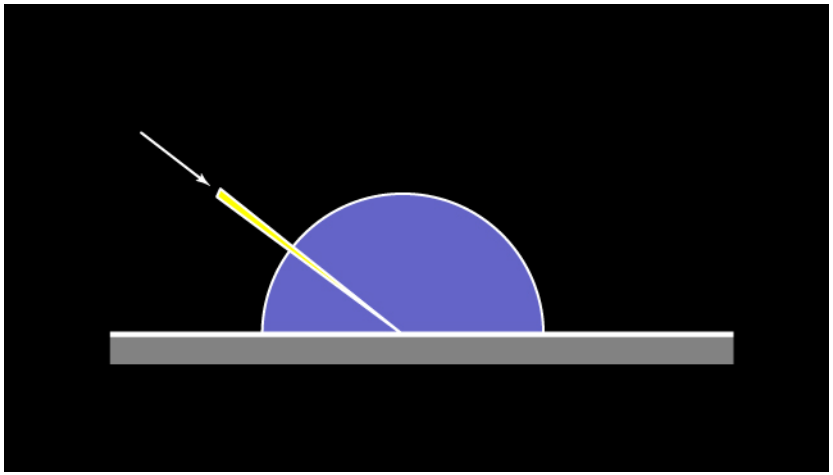
Classic reflection behavior



ideal specular (Fresnel)



rough specular



Lambertian

© 2004 Steve Marschner

Reflection from diffuse surfaces



(Dorsey)

two viewpoints, same illumination

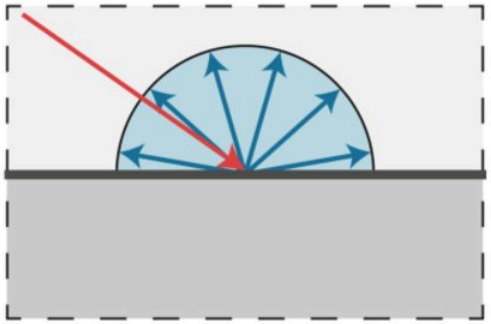


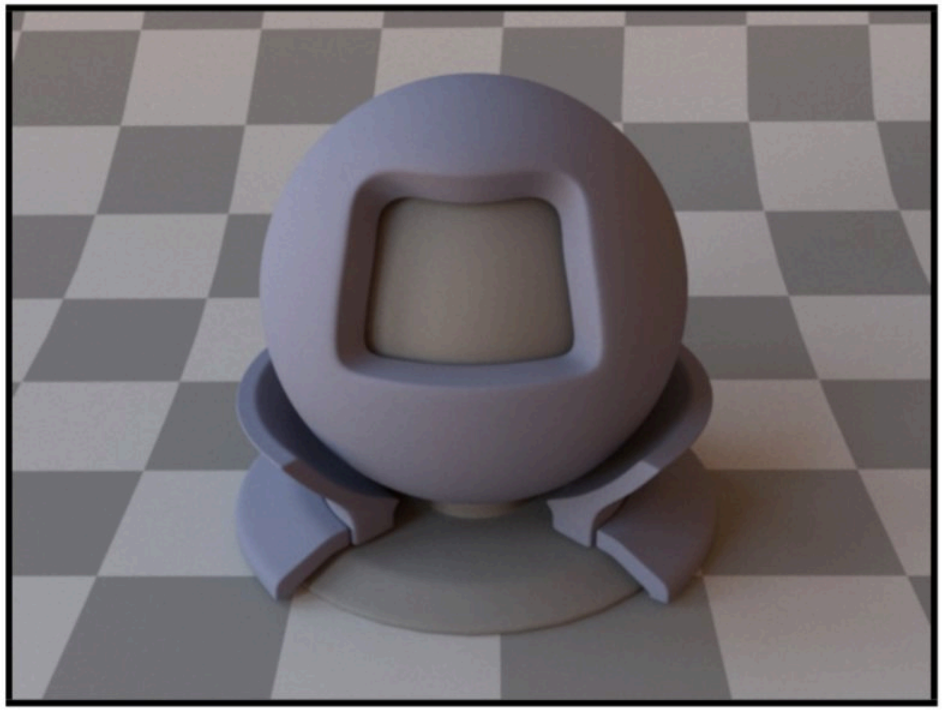
Johann Lambert
(1728-1777)

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

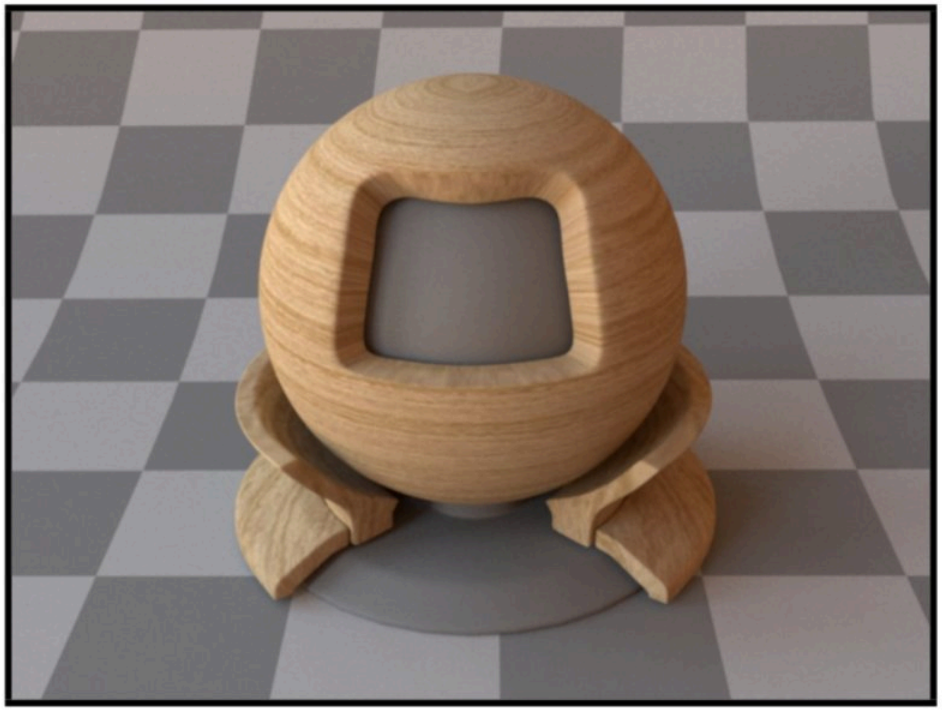
© 2009 Marc Levoy

Diffuse / Lambertian material





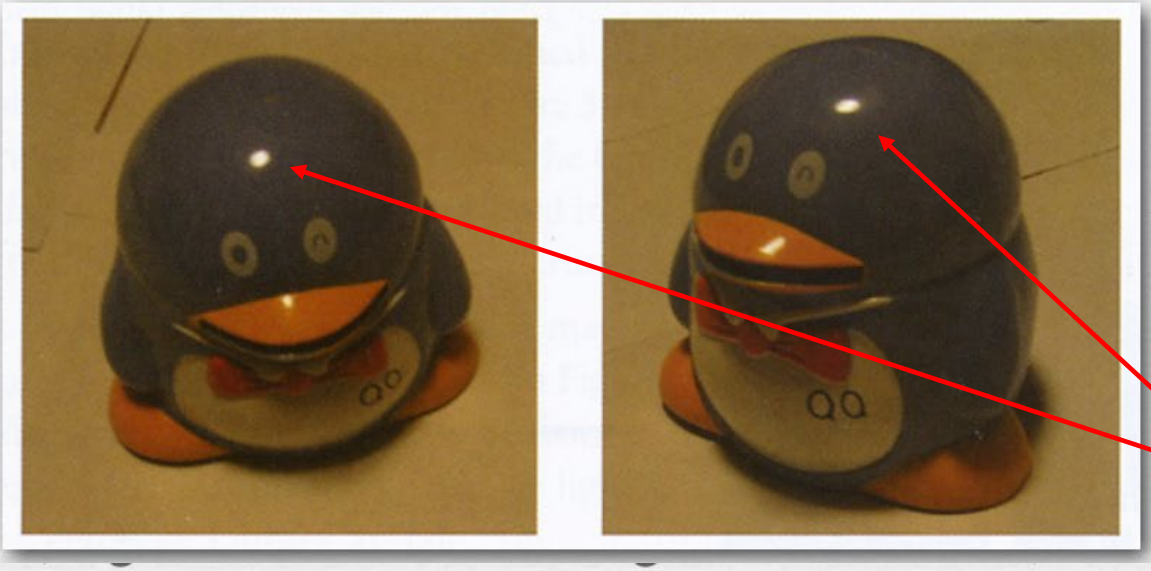
Uniform colored diffuse BRDF
Albedo (fraction of light reflected) is same for all surface points p



Textured diffuse BRDF
Albedo is spatially varying, and is encoded in texture map.

[Mitsuba renderer, Wenzel Jakob, 2010]

Reflection from shiny surfaces



(Dorsey)

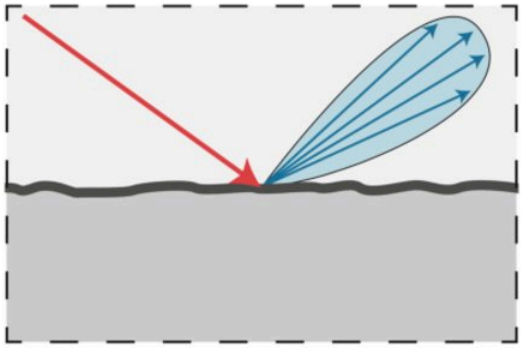
two viewpoints,
same illumination
(i.e. fixed to object)

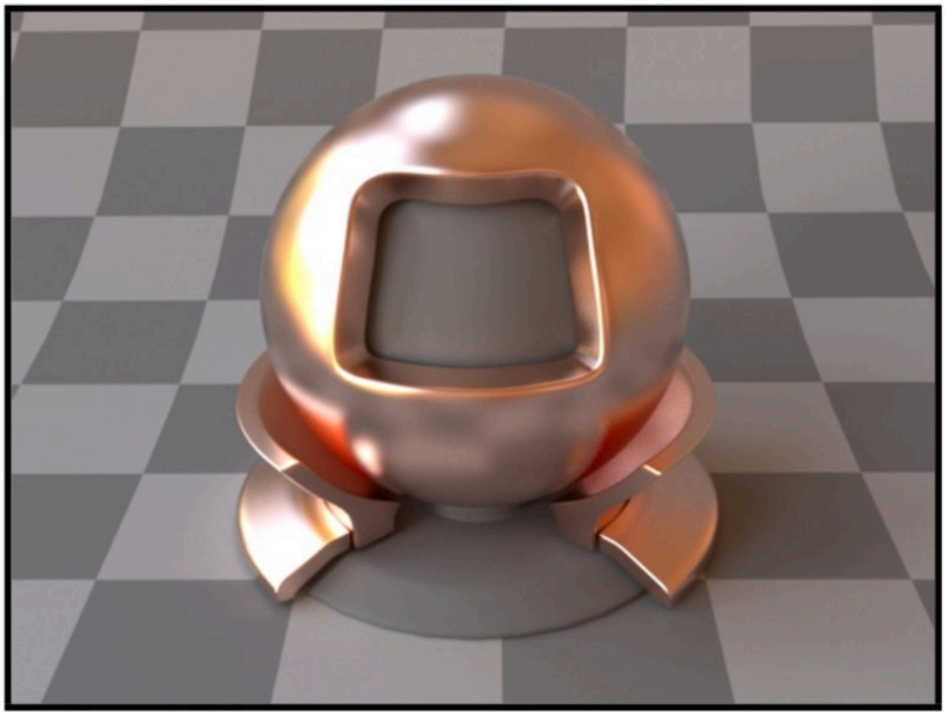
Dot moved!

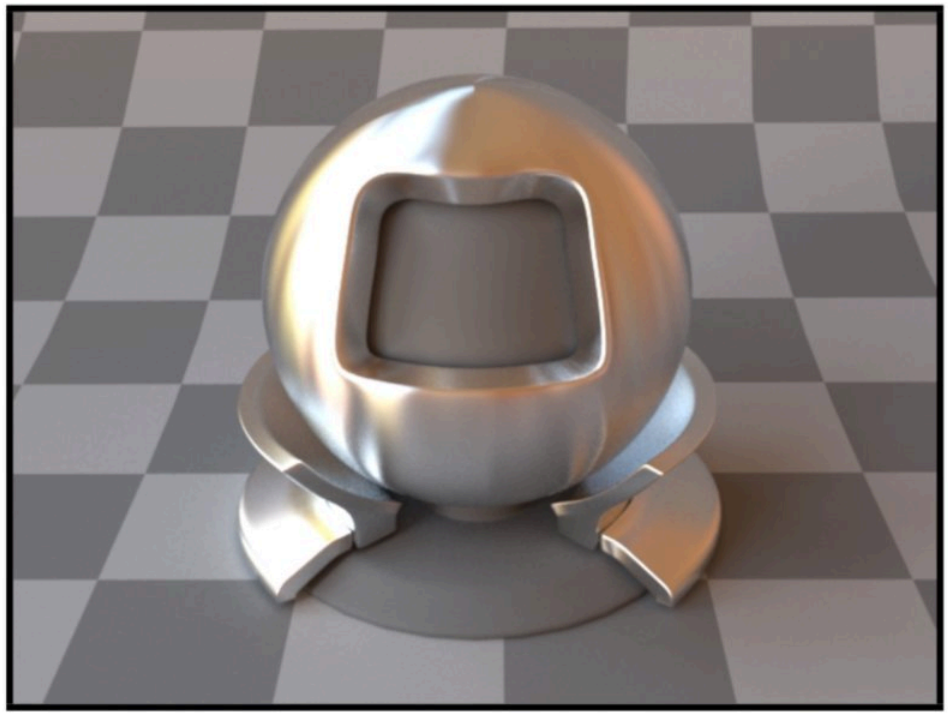
- location of highlight changes with movement of light or viewer

© 2009 Marc Levoy

Glossy material (BRDF)







Copper

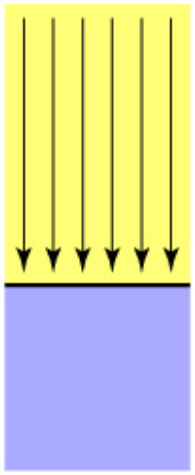
Aluminum

[Mitsuba renderer, Wenzel Jakob, 2010]

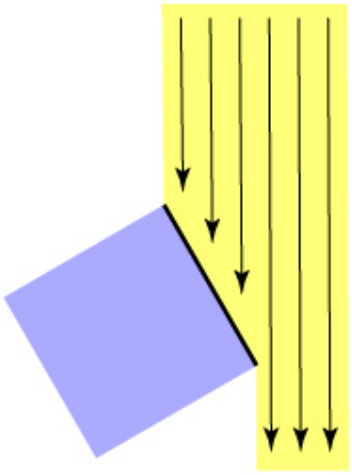
Diffuse

Diffuse reflection

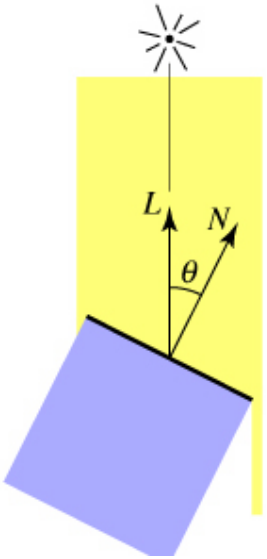
- Light is scattered uniformly in all directions
  - the surface color is the same for all viewing directions
- Lambert's cosine law



Top face of cube receives a certain amount of light



Top face of 60º rotated cube intercepts half the light



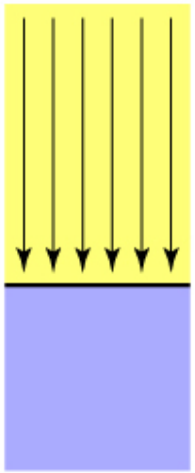
In general, light per unit area is proportional to $\cos \theta = L \cdot N$

© 2004 Steve Marschner

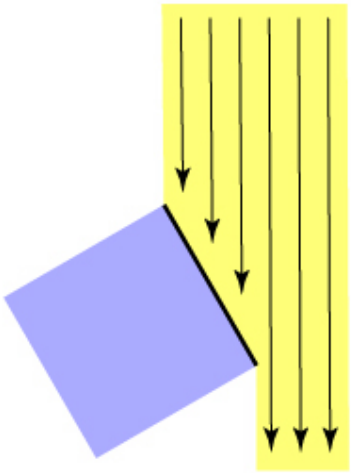
Diffuse reflection

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

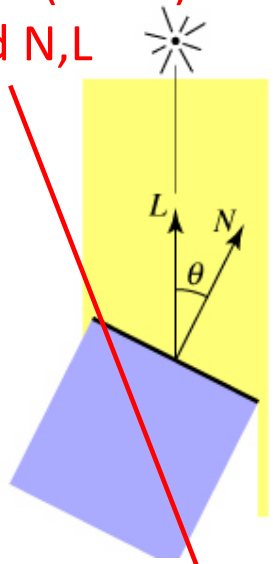
N dot L = |N| |L| cos(theta)
Assume normalized N,L



Top face of cube receives a certain amount of light



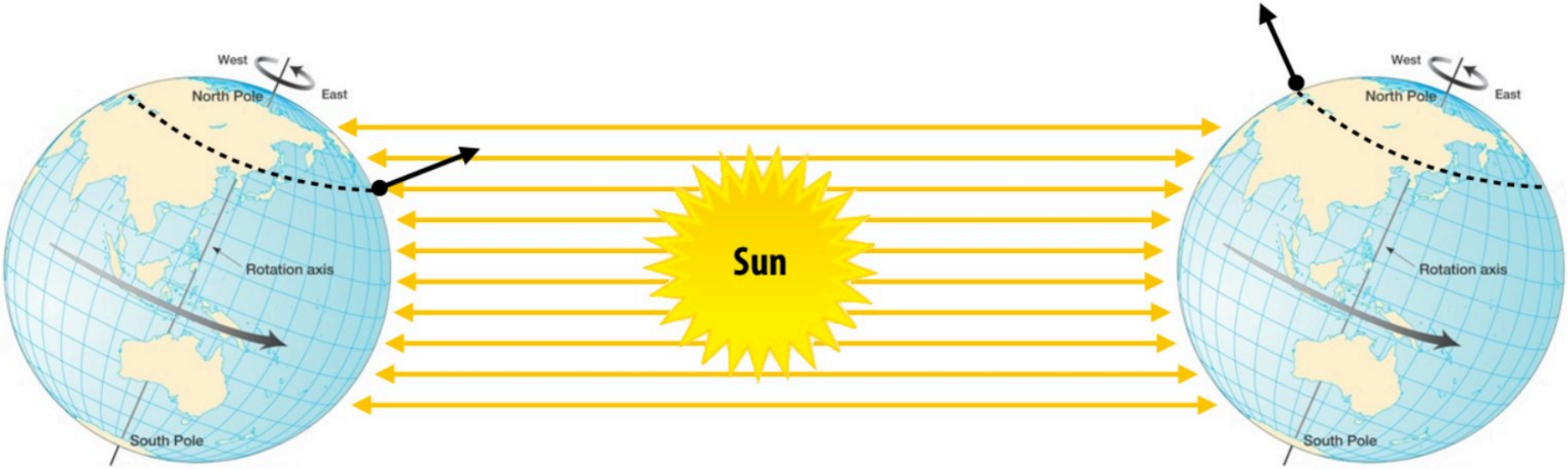
Top face of 60º rotated cube intercepts half the light



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

© 2004 Steve Marschner

Why do we have seasons?



Summer
(Northern hemisphere)

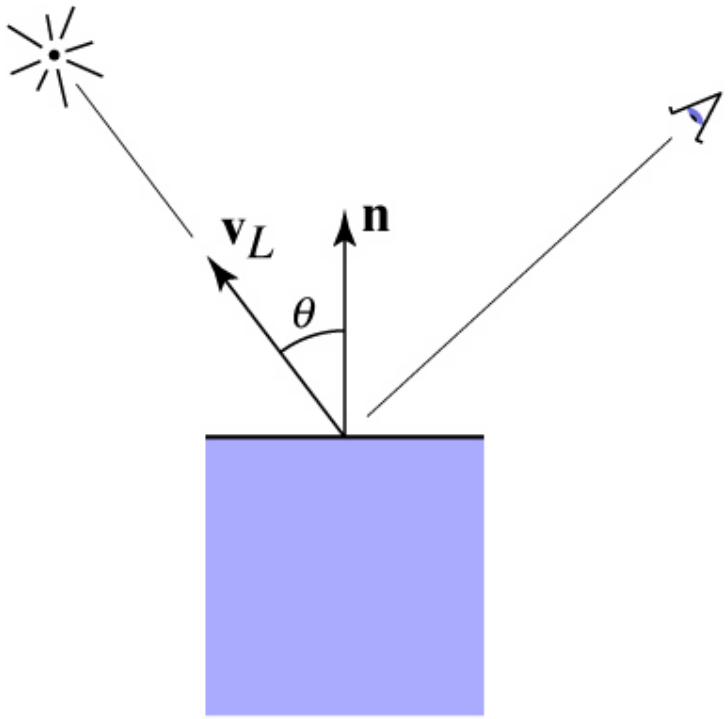
Winter
(Northern hemisphere)

Earth's axis of rotation: ~23.5° off axis

[Image credit: Pearson Prentice Hall]

Lambertian shading

- Shading independent of view direction



illumination from source

$$L_d = k_d \, I \, \max(0, \mathbf{n} \cdot \mathbf{v}_L)$$

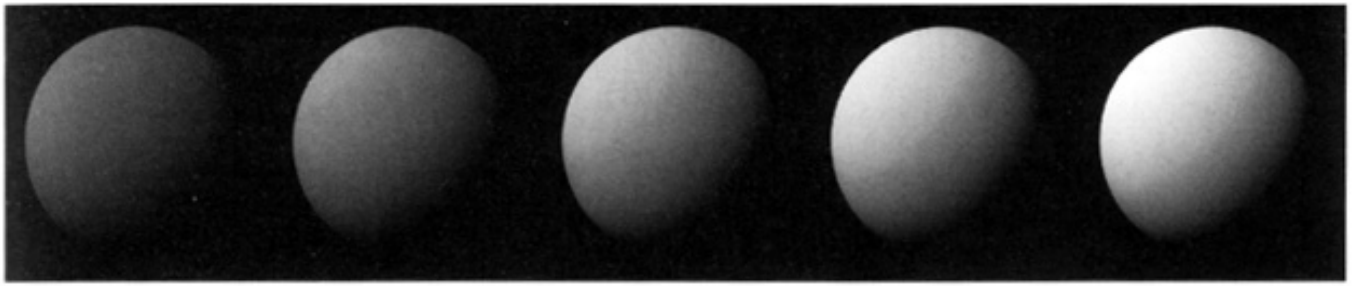
diffuse coefficient

diffusely reflected light

© 2004 Steve Marschner

Lambertian shading

- Produces matte appearance

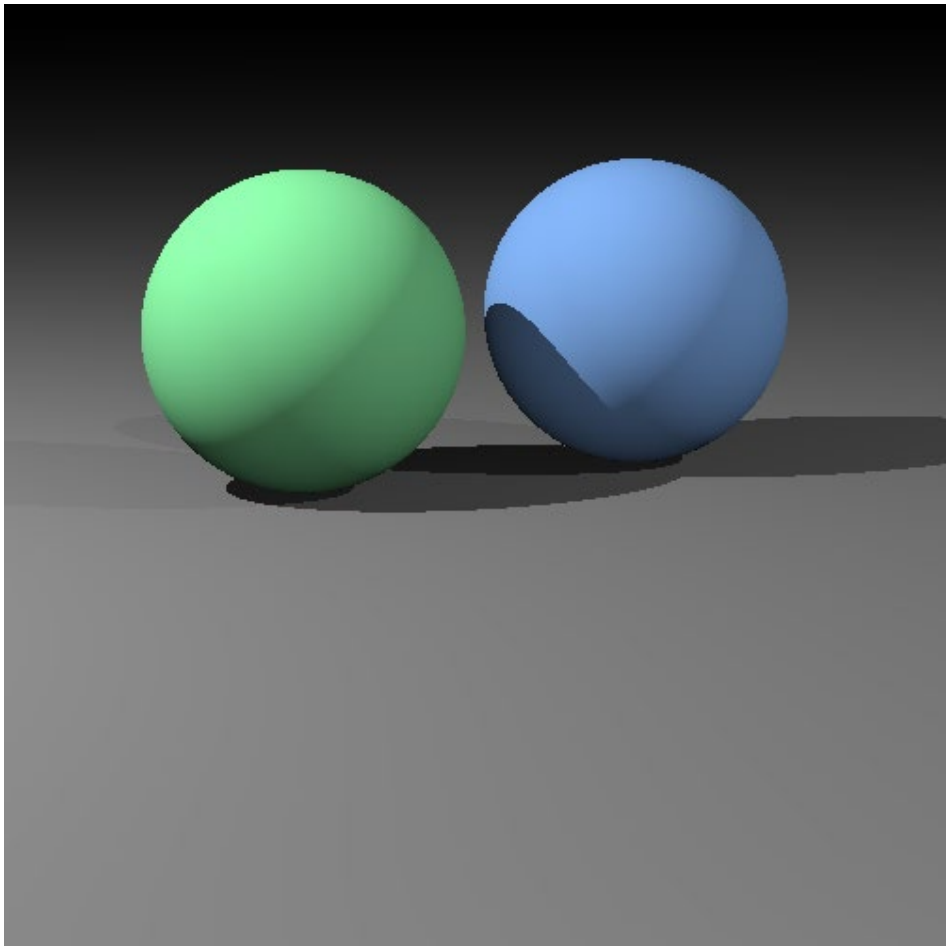


$k_D \longrightarrow$

[Foley et al.]

© 2004 Steve Marschner

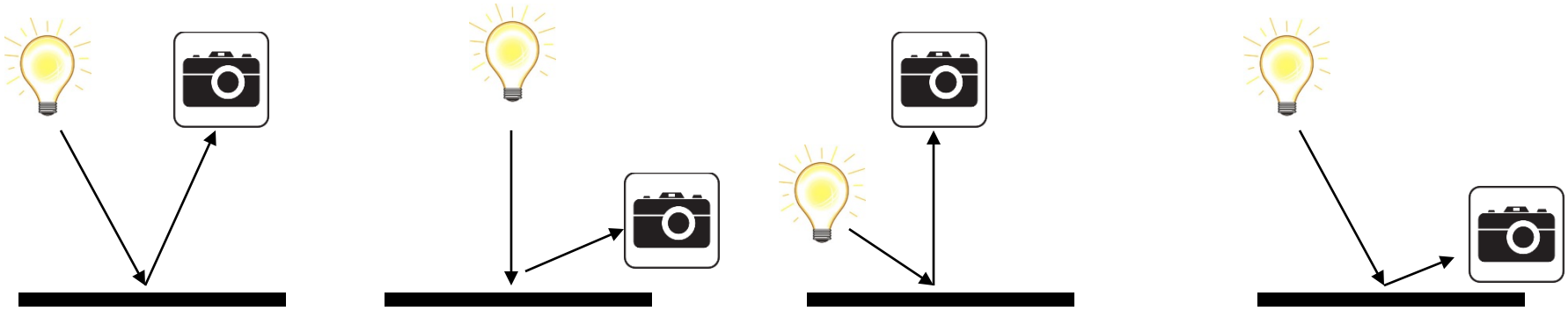
Diffuse shading



© 2004 Steve Marschner

Q Lambertian Shading

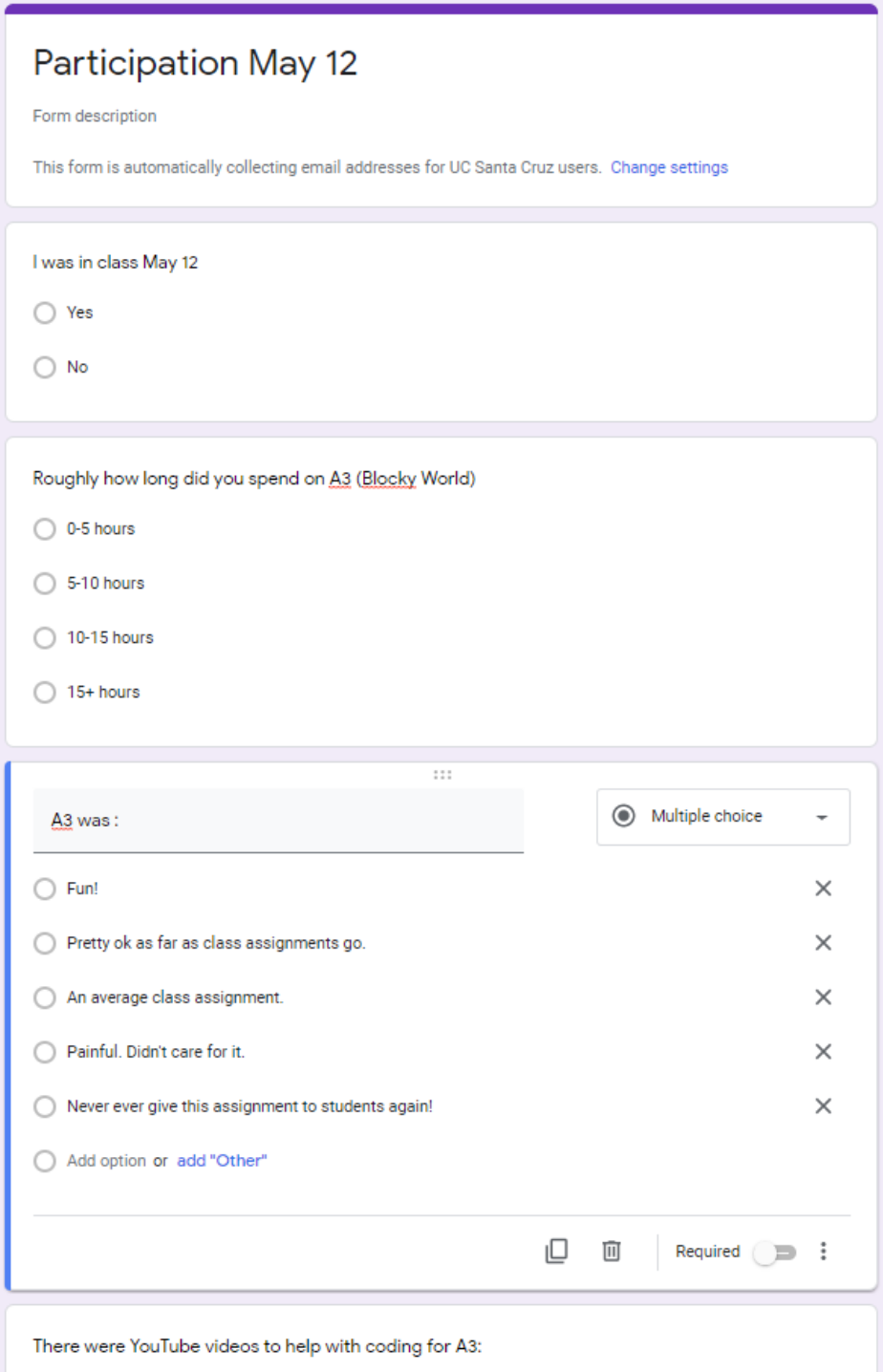
In which configuration does the Lambertian surface appear brightest?



(A) (B) (C) (D)

Participation Survey

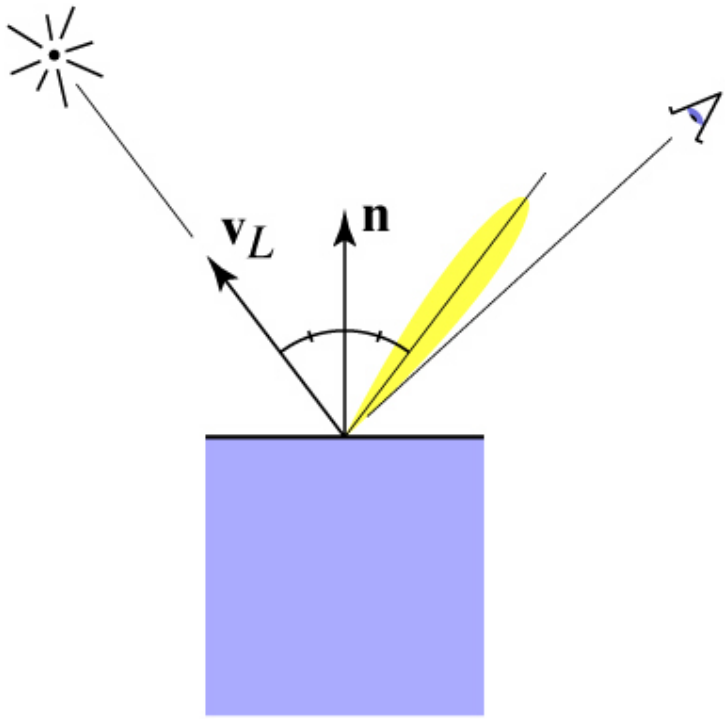
- http://tiny.cc/160-1112



Specular

Specular shading (Phong model)

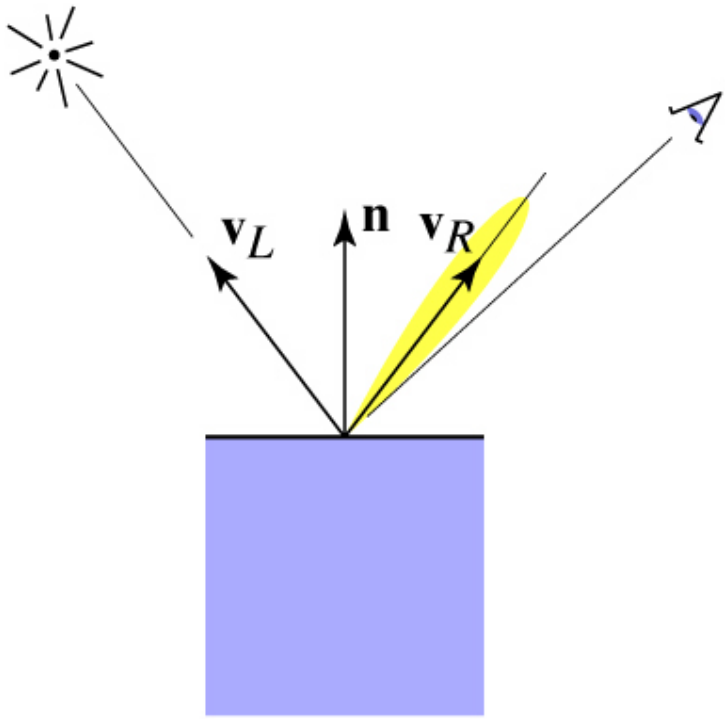
- Intensity depends on view direction
  - bright near mirror configuration



© 2004 Steve Marschner

Specular shading (Phong model)

- Intensity depends on view direction
  - bright near mirror configuration

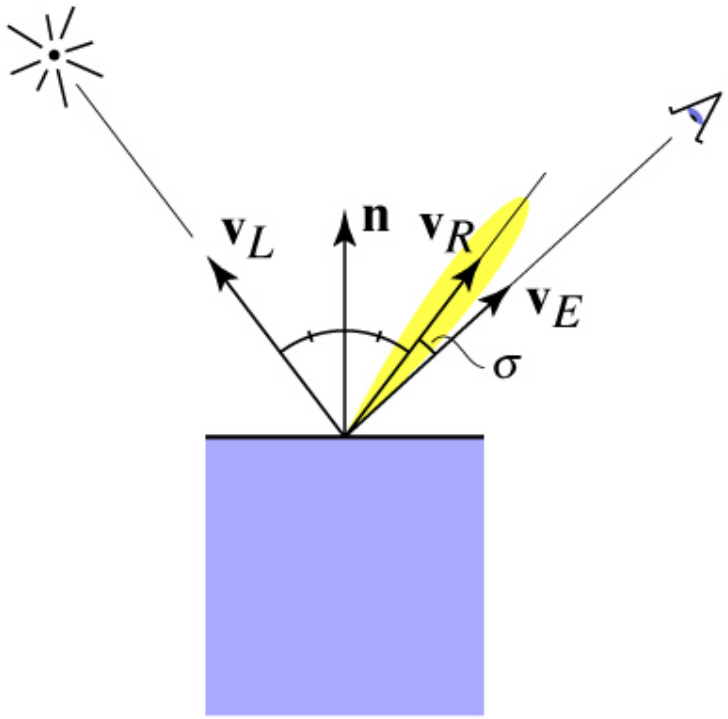


$$\begin{aligned}\mathbf{v}_R &= \mathbf{v}_L + 2((\mathbf{n}\cdot\mathbf{v}_L)\mathbf{n} - \mathbf{v}_L) \\ &= 2(\mathbf{n}\cdot\mathbf{v}_L)\mathbf{n} - \mathbf{v}_L\end{aligned}$$

© 2004 Steve Marschner

Specular shading (Phong model)

- Intensity depends on view direction
  - bright near mirror configuration



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

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

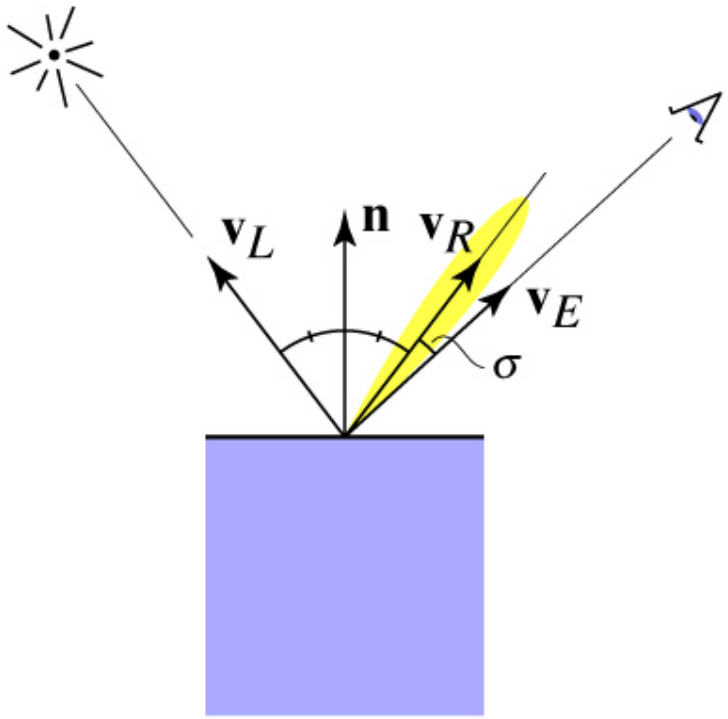
specularly reflected light

specular coefficient

© 2004 Steve Marschner

Specular shading (Phong model)

- Intensity depends on view direction
  - bright near mirror configuration



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

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

specularly reflected light

specular coefficient

© 2004 Steve Marschner

Phong model—plots

- Increasing *n* narrows the lobe

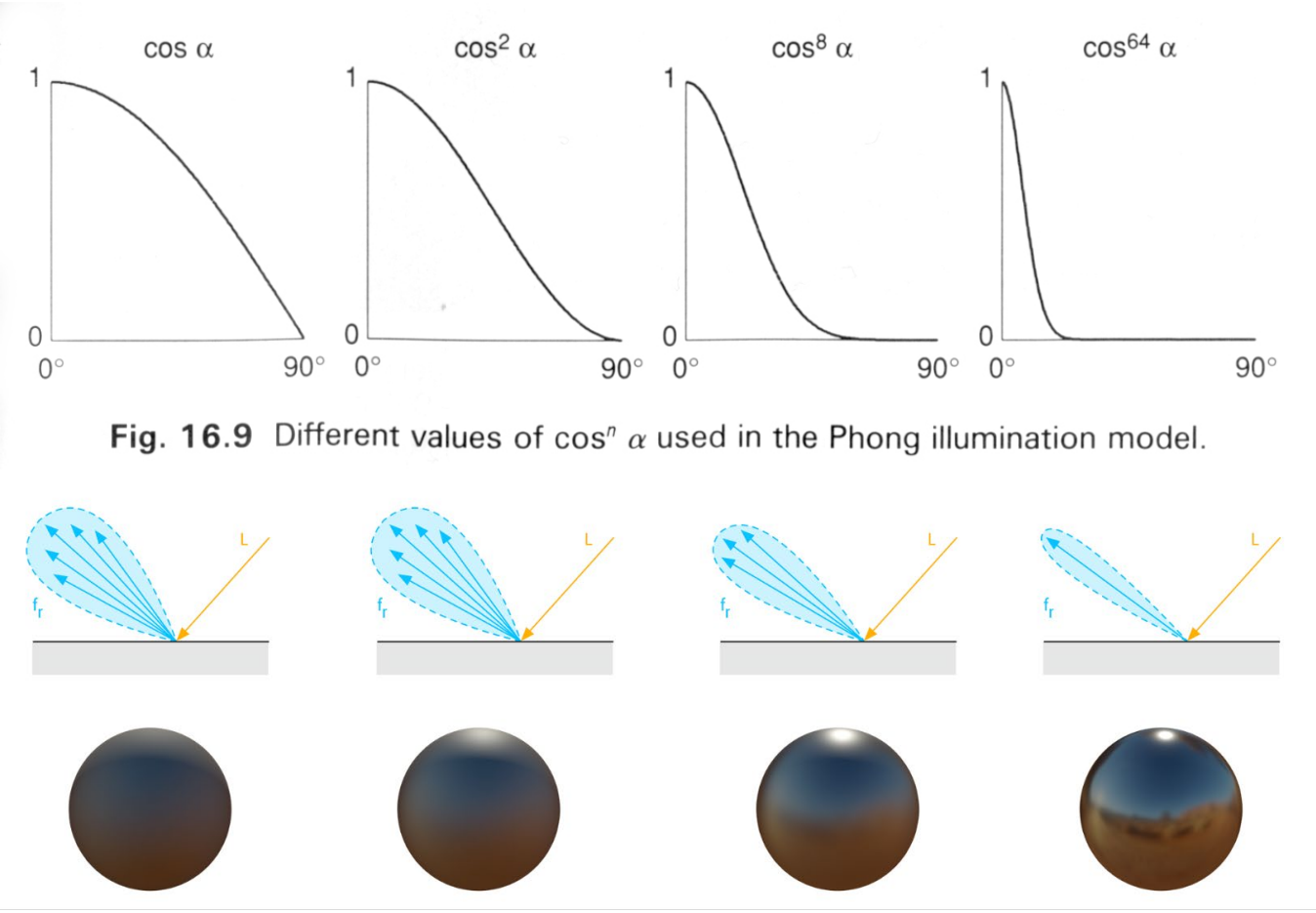


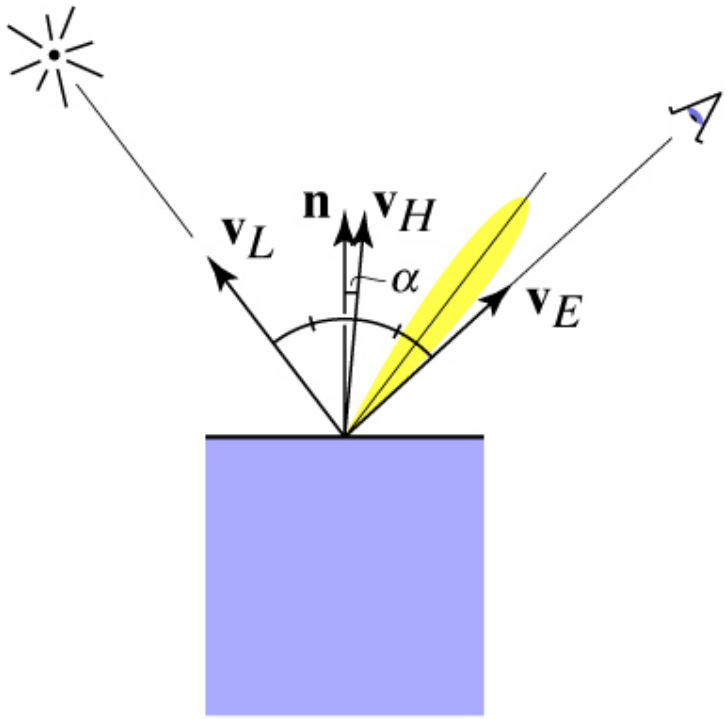
Fig. 16.9 Different values of $\cos^n \alpha$ used in the Phong illumination model.

[Foley et al.]

© 2004 Steve Marschner

Phong variant: Blinn-Phong

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



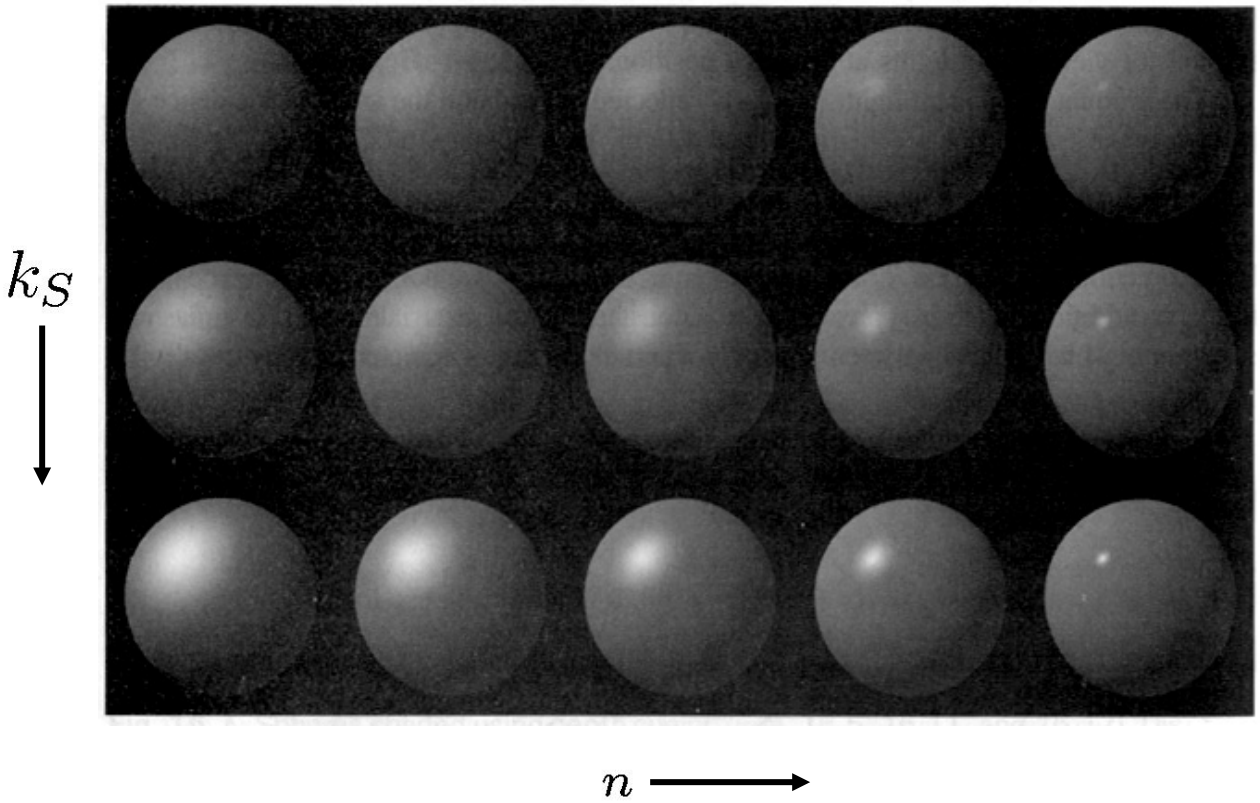
$$\mathbf{v}_H = \text{bisector}(\mathbf{v}_L, \mathbf{v}_E)$$
$$= \frac{(\mathbf{v}_L + \mathbf{v}_E)}{\|\mathbf{v}_L + \mathbf{v}_E\|}$$

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

© 2004 Steve Marschner

Specular shading

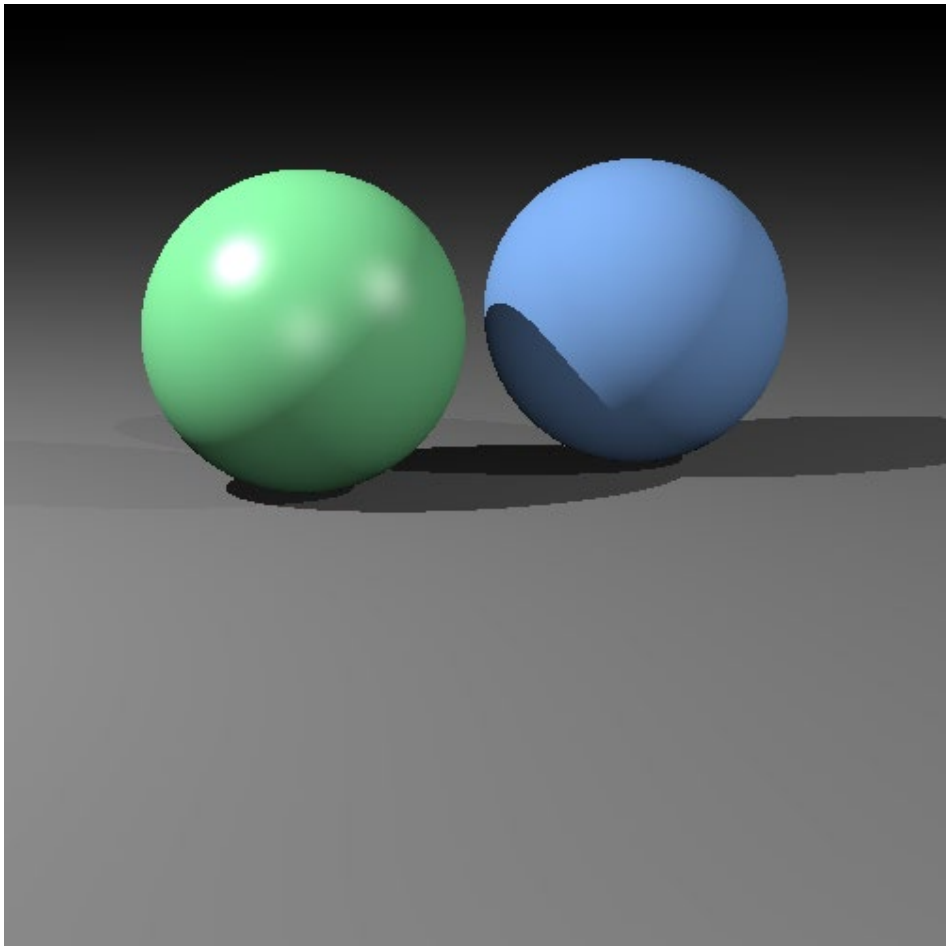
- Phong and Blinn-Phong



[Foley et al.]

© 2004 Steve Marschner

Diffuse + Phong shading

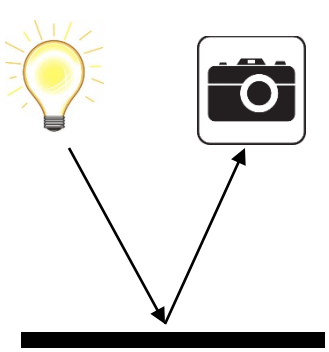


© 2004 Steve Marschner

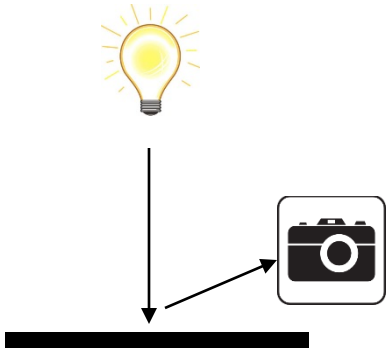
Q Specular Shading

In which configuration does the specular surface appear brightest?

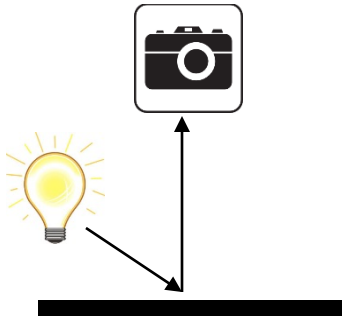




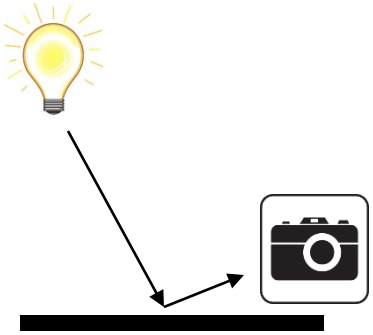
(A)



(B)



(C)

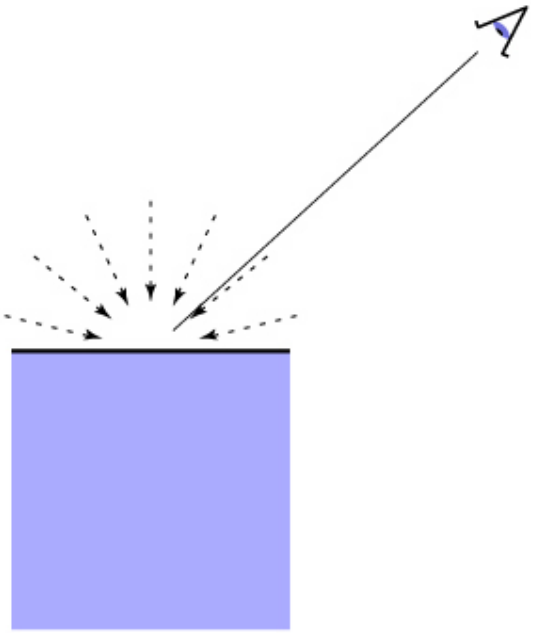


(D)

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



$$L_a = k_a I_a$$

ambient coefficient

reflected ambient light

© 2004 Steve Marschner

Putting it together

- Usually include ambient, diffuse, Phong in one model

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

ambient diffuse specular

- The final result is the sum over many lights

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

© 2004 Steve Marschner

Putting it together

- Usually include ambient, diffuse, Phong in one model

Definitely Vectors

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

- The final result is the sum over many lights

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

© 2004 Steve Marschner

Putting it together

- Usually include ambient, diffuse, Phong in one model

Dot product (not Cross product)

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

- The final result is the sum over many lights

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

© 2004 Steve Marschner

Putting it together

- Usually include ambient, diffuse, Phong in one model

Scalar value (not vector)

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

- The final result is the sum over many lights

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

© 2004 Steve Marschner

Putting it together

- Usually include ambient, diffuse, Phong in one model

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

- The final result is the sum over many lights

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

$$L = L_a + \sum_i (L_d)_i + (L_s)_i$$

$$= k_a I_a + \sum_i I_i \left(k_d \max(0, \mathbf{n} \cdot (\mathbf{v}_L)_i) + k_s \max(0, \mathbf{n} \cdot (\mathbf{v}_H)_i)^n\right)$$

© 2004 Steve Marschner

Putting it together

- Usually include ambient, diffuse, Phong in one model

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

- The final result is the sum over many lights

$$L = L_a + \sum_i (L_d)_i + (L_s)_i$$

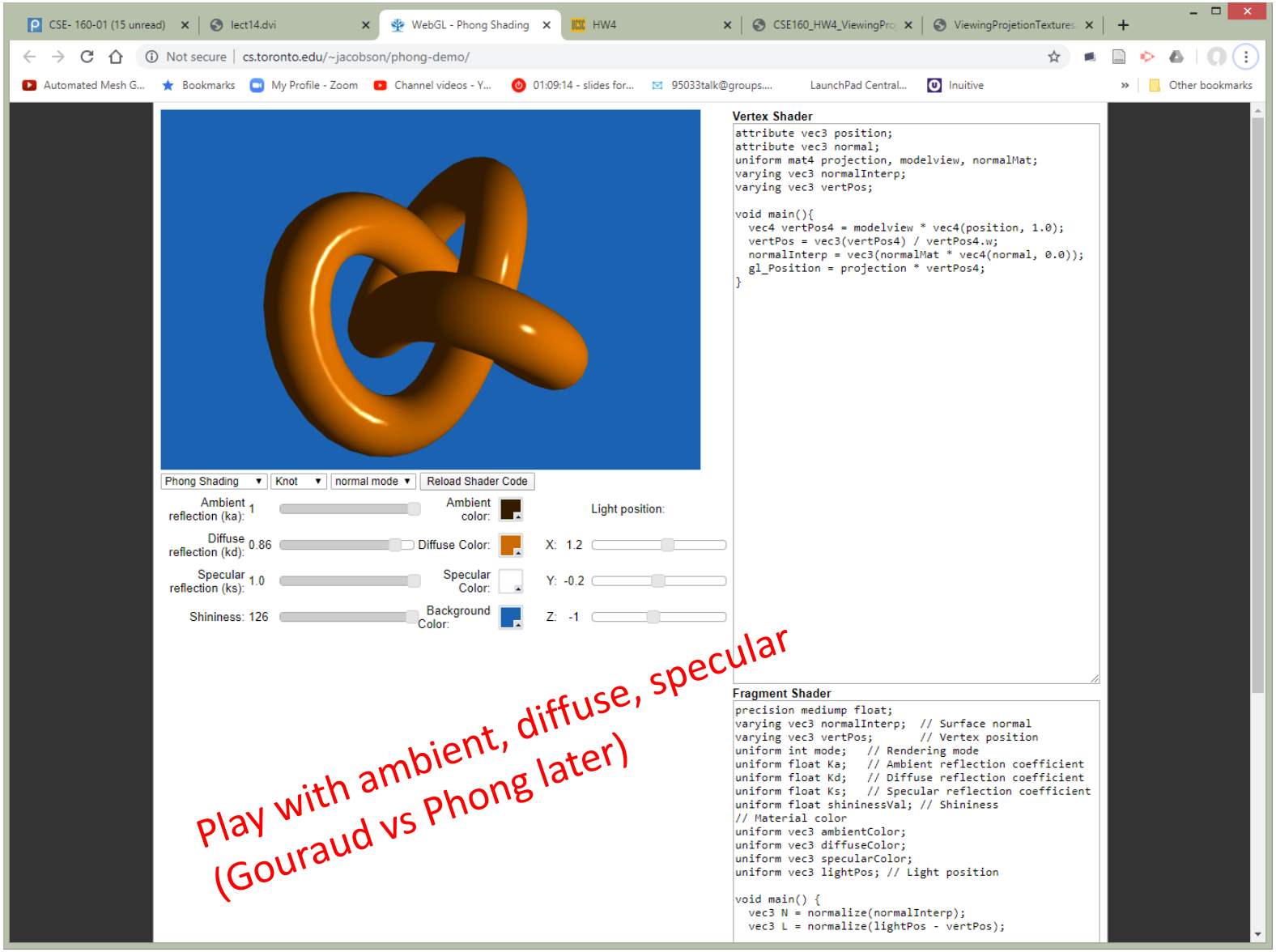
$$= k_a I_a + \sum_i I_i\,(k_d \max(0, \mathbf{n} \cdot (\mathbf{v}_L)_i) + k_s \max(0, \mathbf{n} \cdot (\mathbf{v}_H)_i)^n)$$

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

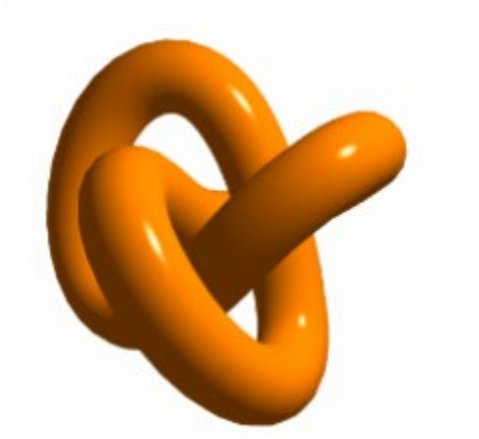
© 2004 Steve Marschner

Really great example to play with the parameters:

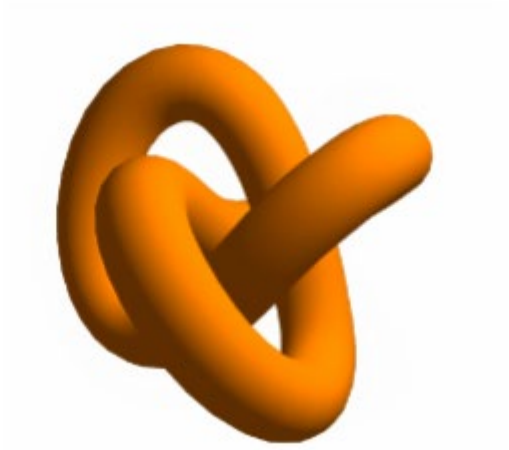
http://www.cs.toronto.edu/~jacobson/phong-demo/



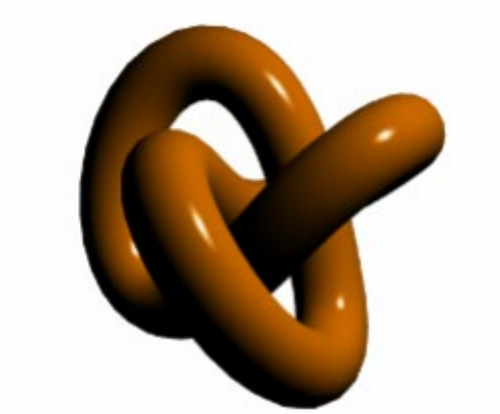
Ambient + Diffuse + Specular



Ambient+Diffuse+Specular



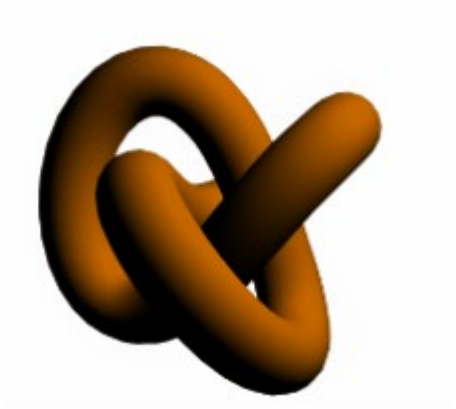
Ambient+Diffuse



Diffuse+Specular



Ambient Only



Diffuse Only



Specular Only

Specular Exponent

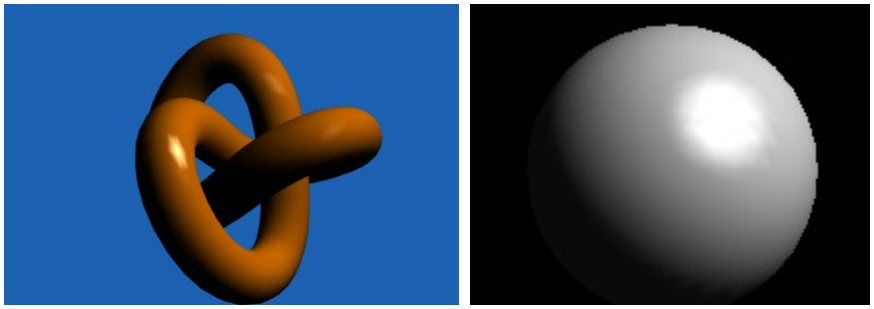


Specular exponent n=4

Specular exponent n=32

Specular exponent n=128

Lighting Coefficients Q



What are the coefficients for this rendering?

(A) kAmbient = 0, kDiffuse = 0, kSpecular = 0
(B) kAmbient = 0, kDiffuse = 0.5, kSpecular = 0.5
(C) kAmbient = 0.5, kDiffuse = 0.5, kSpecular = 0
(D) kAmbient = 0.5, kDiffuse = 0, kSpecular = 0.5
(E) kAmbient = 0, kDiffuse = 0, kSpecular = 0.5

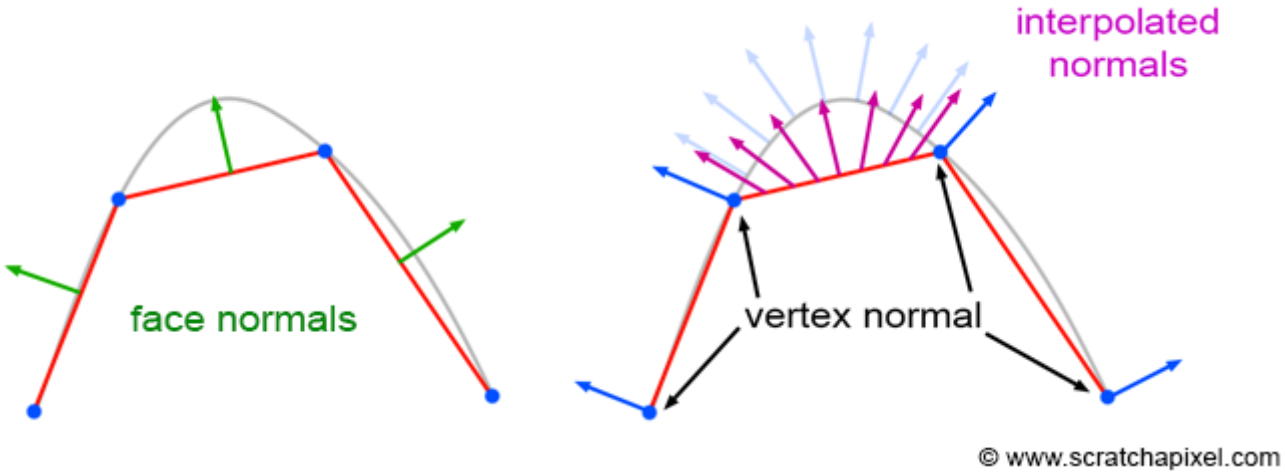
Lighting Coefficients Q



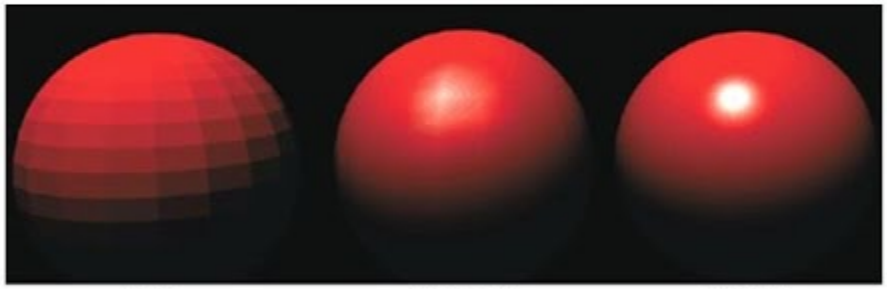
What are the coefficients for this rendering?

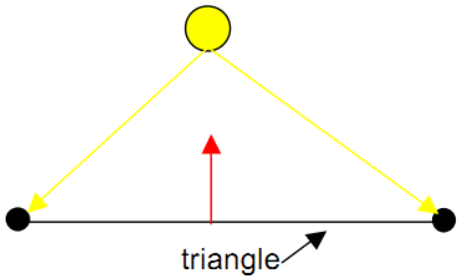
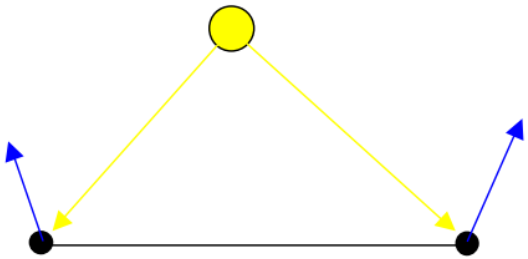
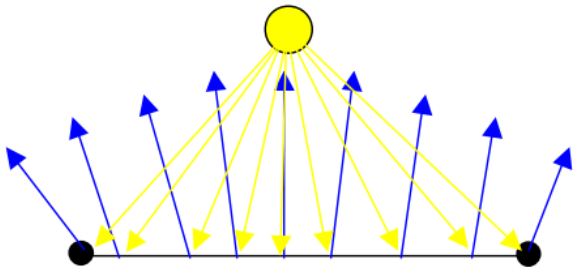
(A) kAmbient = 0, kDiffuse = 0, kSpecular = 0
(B) kAmbient = 0, kDiffuse = 0.5, kSpecular = 0.5
(C) kAmbient = 0.5, kDiffuse = 0.5, kSpecular = 0
(D) kAmbient = 0.5, kDiffuse = 0, kSpecular = 0.5
(E) kAmbient = 0, kDiffuse = 0, kSpecular = 0.5

Which Normal?

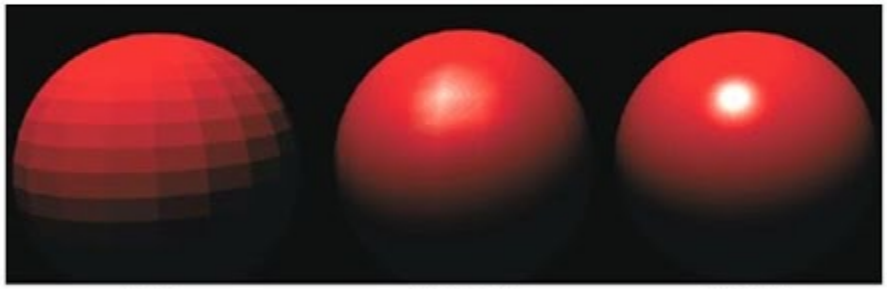

face normals
interpolated
normals
vertex normal
© www.scratchapixel.com

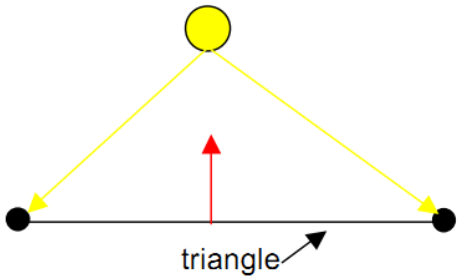
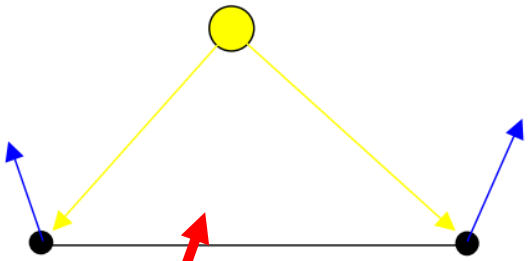
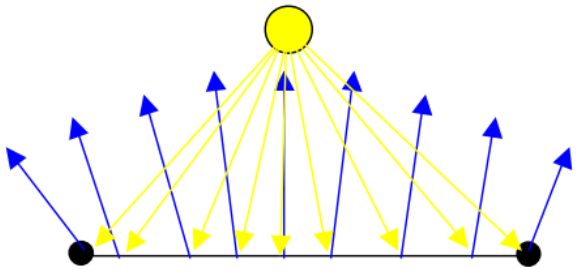
FLAT SHADING, GOURAUD SHADING & PHONG SHADING



Flat shading	Goraud shading	Phong shading
		
Only the first normal of the triangle is used to compute lighting in the entire triangle.	The light intensity is computed at each vertex and interpolated across the surface.	Normals are interpolated across the surface, and the light is computed at each fragment.

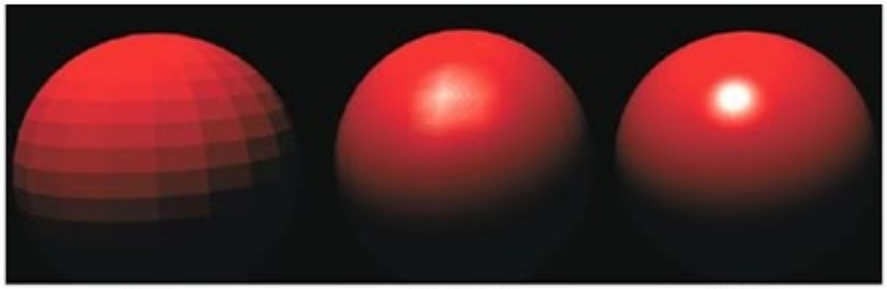
FLAT SHADING, GOURAUD SHADING & PHONG SHADING

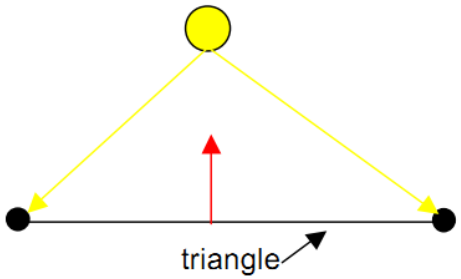
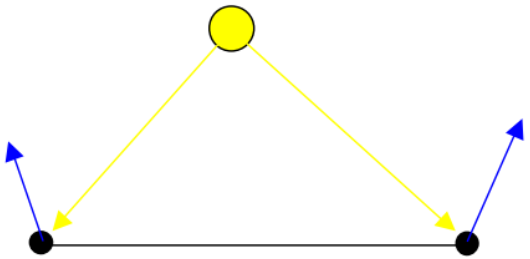
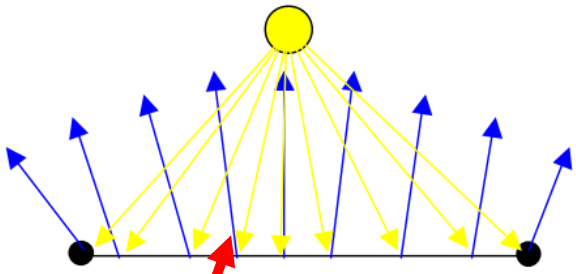


Flat shading	Goraud shading	Phong shading
		
Only the first normal of the triangle is used to compute lighting in the entire triangle.	The light intensity is computed at each vertex and interpolated across the surface.	Normals are interpolated across the surface, and the light is computed at each fragment.

Calculate lighting at vertex, interpolate light intensity

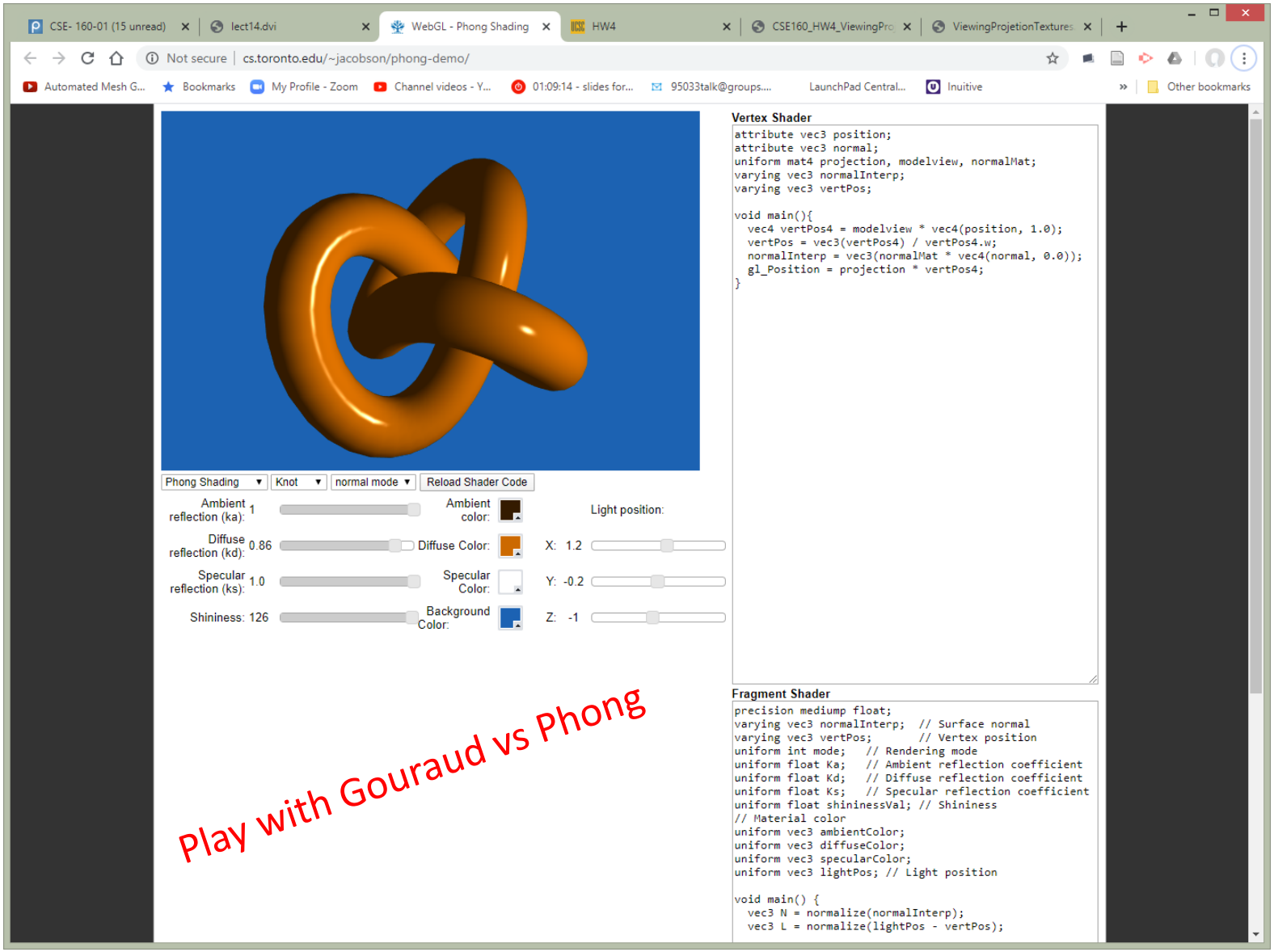
FLAT SHADING, GOURAUD SHADING & PHONG SHADING



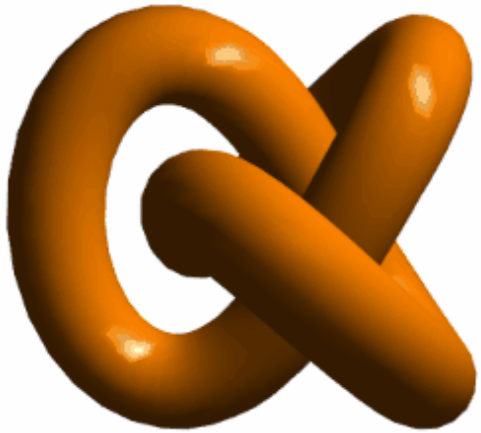
Flat shading	Goraud shading	Phong shading
		
Only the first normal of the triangle is used to compute lighting in the entire triangle.	The light intensity is computed at each vertex and interpolated across the surface.	Normals are interpolated across the surface, and the light is computed at each fragment.

Interpolate normal, calculate lighting at fragment

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



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 coefficie
uniform float Kd;   // Diffuse reflection coefficie
uniform float Ks;   // Specular reflection coeffic
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 ligh
    vec3 V = normalize(-vertPos); // Vector to viewe
    // 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
```

Fragment Shader

```
precision mediump float;

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

Calculate lighting at vertex

Phong Shading



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

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 fragme

Fragment Shader

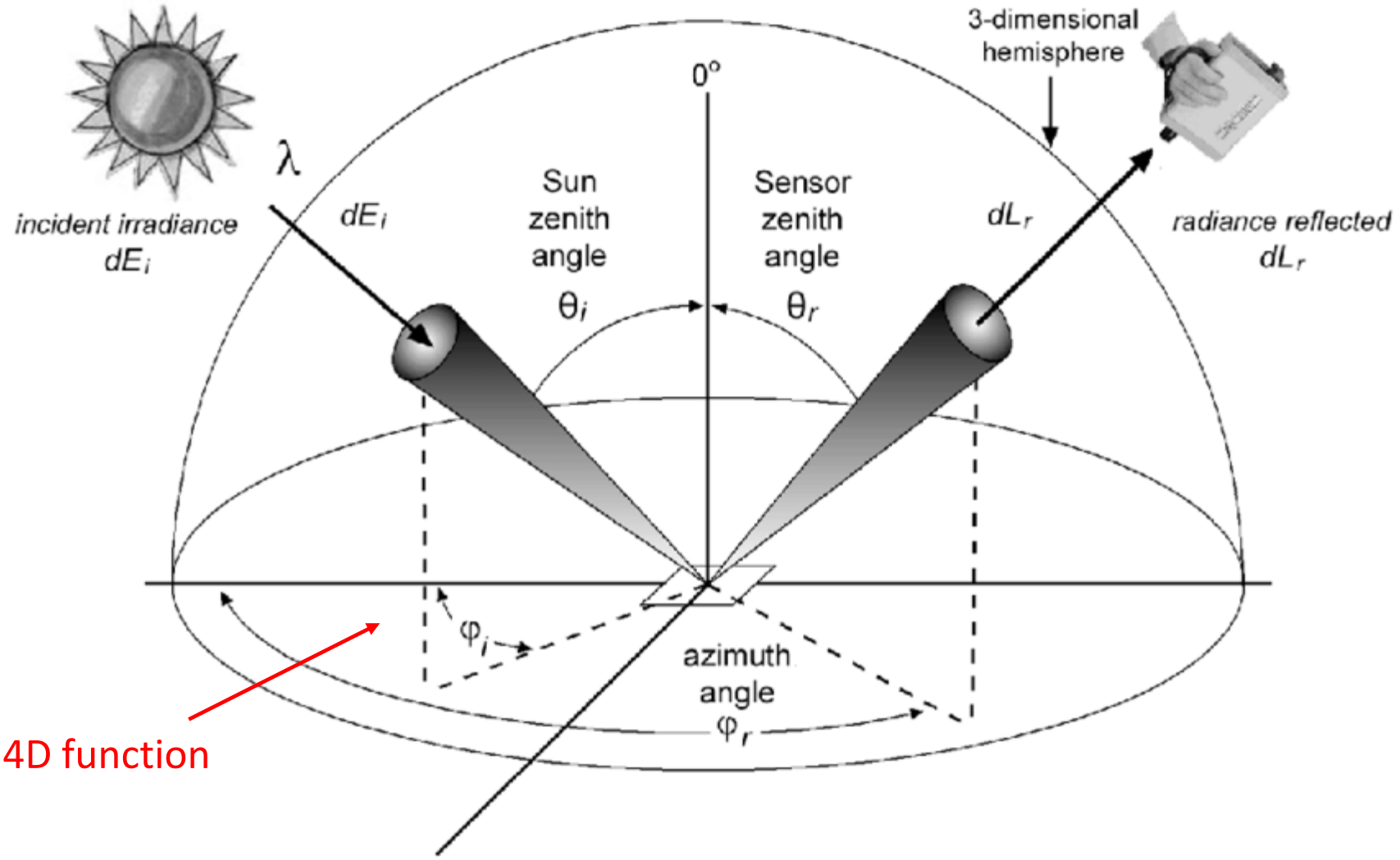
```
precision mediump float;
varying vec3 normalInterp;  // Surface normal
varying vec3 vertPos;       // Vertex position
uniform int mode;   // Rendering mode
uniform float Ka;   // Ambient reflection coefficient
uniform float Kd;   // Diffuse reflection coefficient
uniform float Ks;   // Specular reflection 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);
```

BRDFs



Bidirectional Reflectance Distribution Function (BRDF)



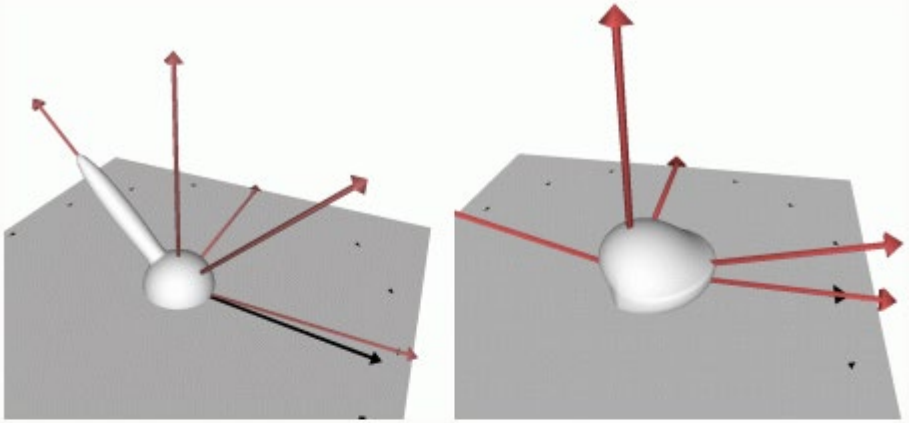


Figure 3: Phong reflection model (left), Oren-Nayar Diffuse Microfacet BRDF (right).

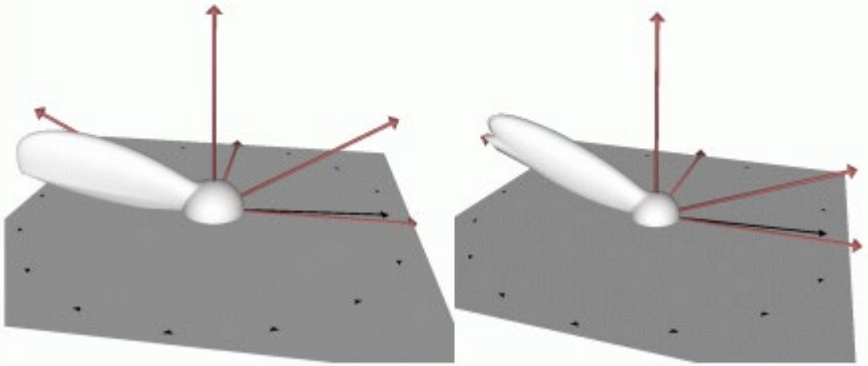
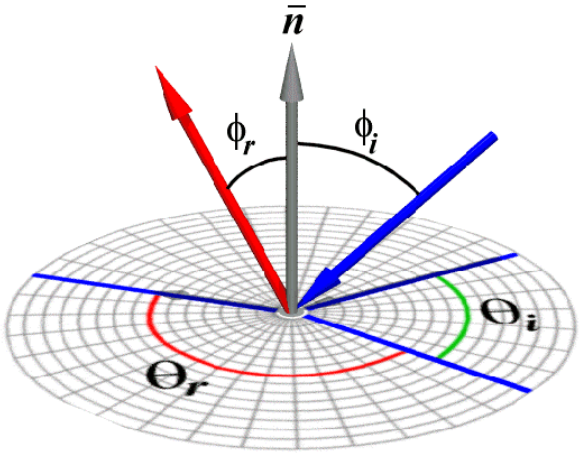
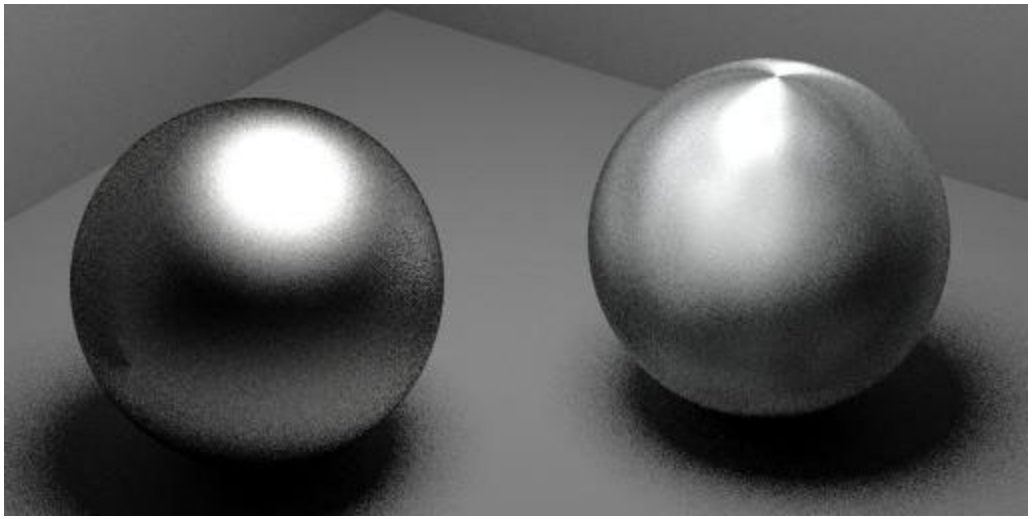


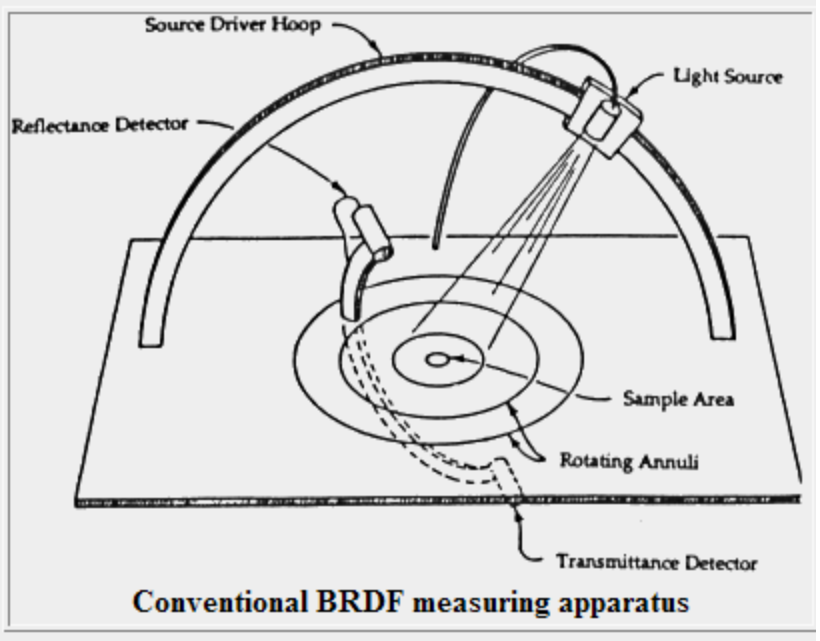
Figure 4: Cook-Torrance Specular Microfacet BRDF (left), He-Torrance Comprehensive Analytic Model (right).

Isotropic vs anisotropic

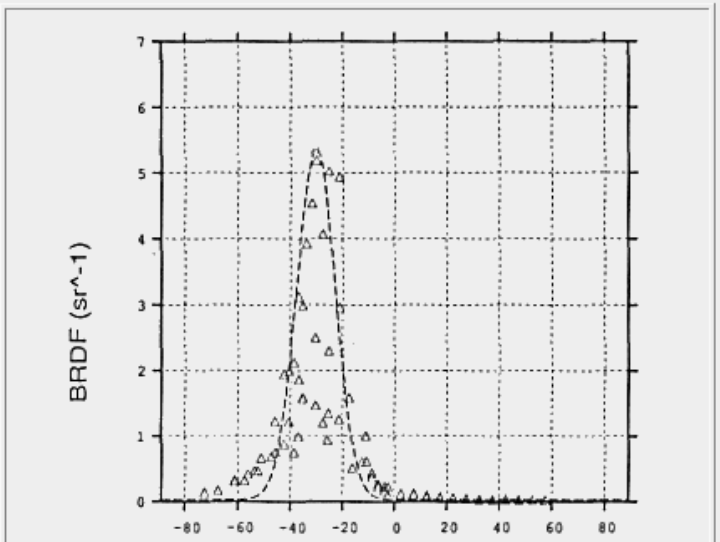
$\rho(\theta_r, \phi_r, \theta_i, \phi_i)$





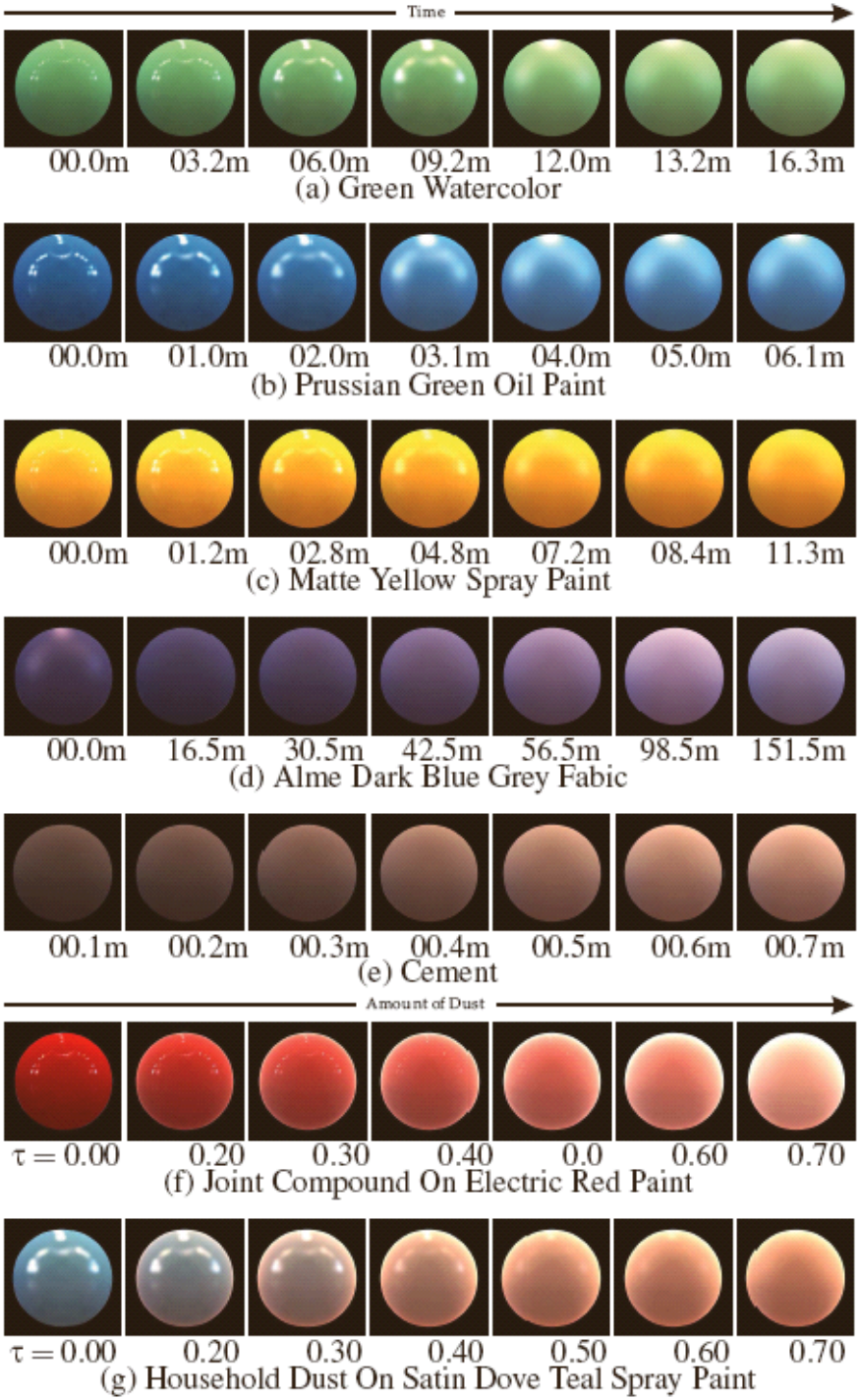


Conventional BRDF measuring apparatus



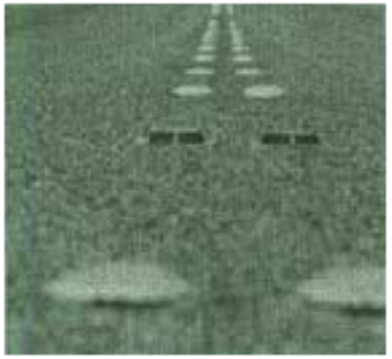
Noise and missing data in one slice of Ward's measured BRDF


lamp
camera

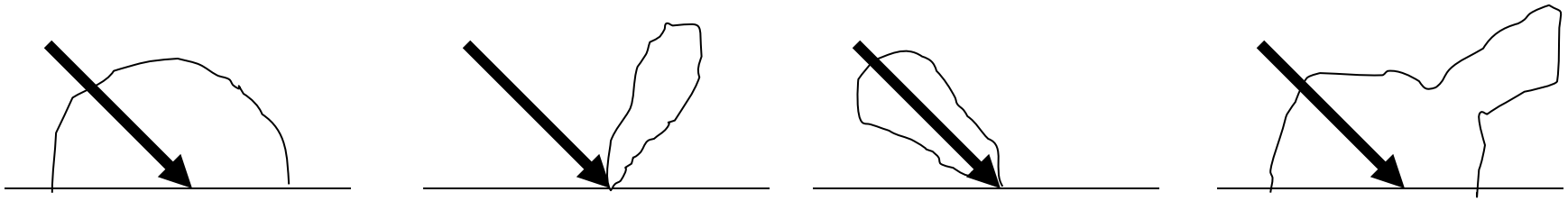

Time
00.0m 03.2m 06.0m 09.2m 12.0m 13.2m 16.3m
(a) Green Watercolor
00.0m 01.0m 02.0m 03.1m 04.0m 05.0m 06.1m
(b) Prussian Green Oil Paint
00.0m 01.2m 02.8m 04.8m 07.2m 08.4m 11.3m
(c) Matte Yellow Spray Paint
00.0m 16.5m 30.5m 42.5m 56.5m 98.5m 151.5m
(d) Alme Dark Blue Grey Fabic
00.1m 00.2m 00.3m 00.4m 00.5m 00.6m 00.7m
(e) Cement
Amount of Dust
τ = 0.00 0.20 0.30 0.40 0.0 0.60 0.70
(f) Joint Compound On Electric Red Paint
τ = 0.00 0.20 0.30 0.40 0.50 0.60 0.70
(g) Household Dust On Satin Dove Teal Spray Paint

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.



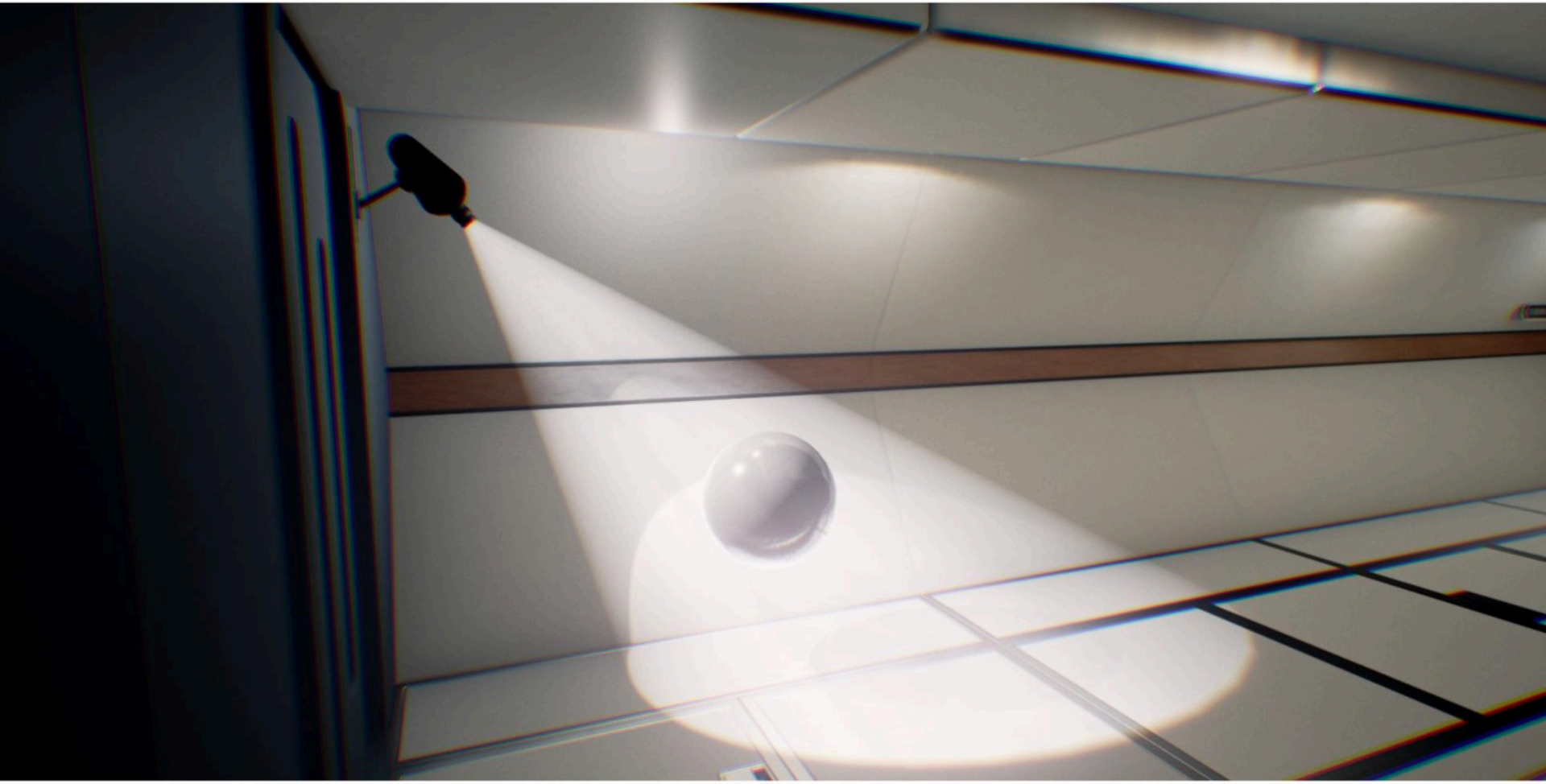
Sketch a goniometric diagram for a retroreflective surface.



(A) (B) (C) (D)

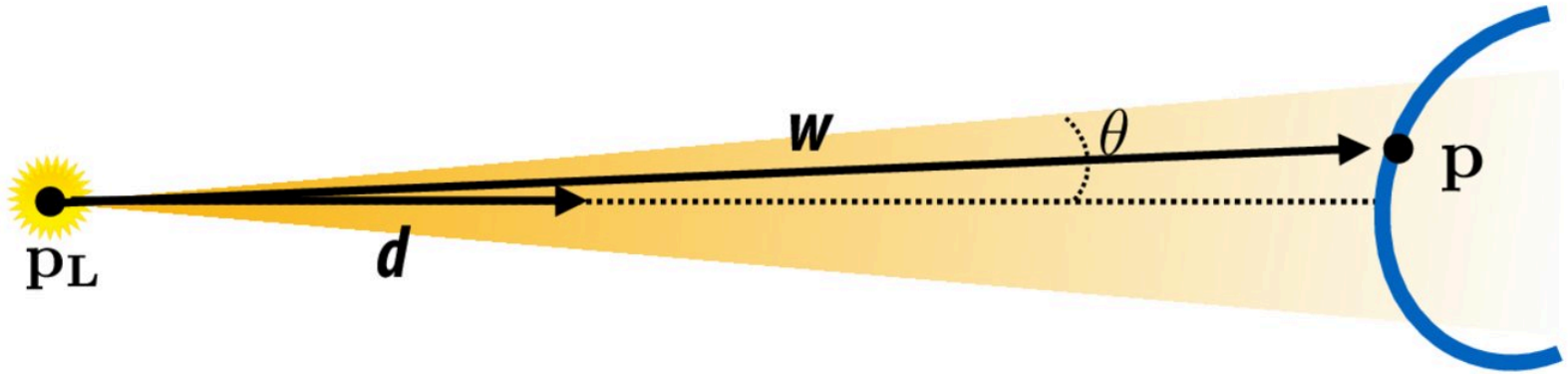
Spot Lights

Spot light



Spot light

(does not emit equally in all directions)



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

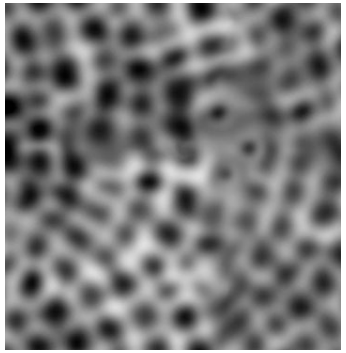
$$L(\mathbf{w}) = 0 \quad \textbf{if} \quad \mathbf{w} \cdot \mathbf{d} > \cos\theta$$
$$= L_0 \quad \textbf{otherwise}$$

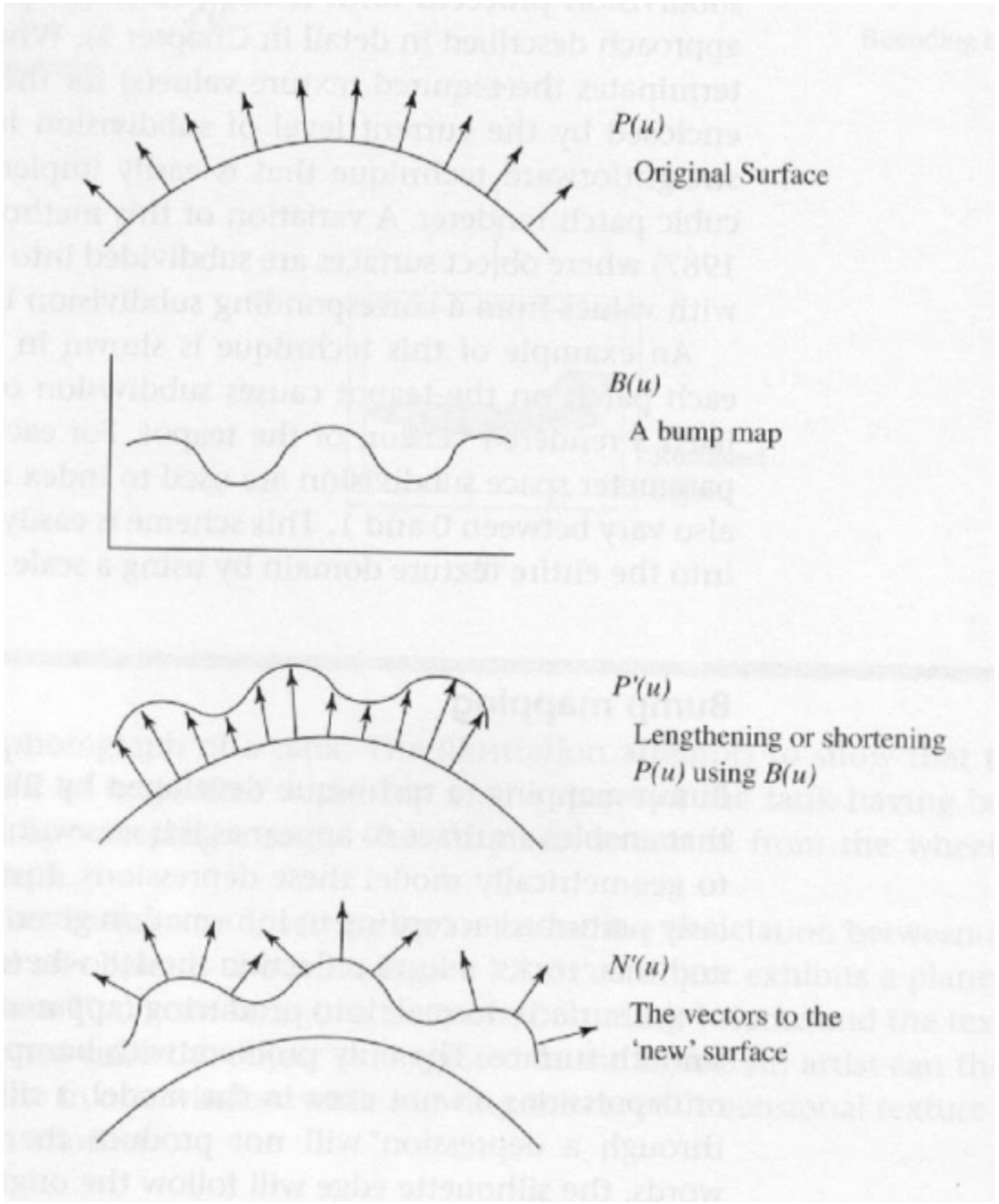
Or, if spotlight intensity falls off from direction *d*

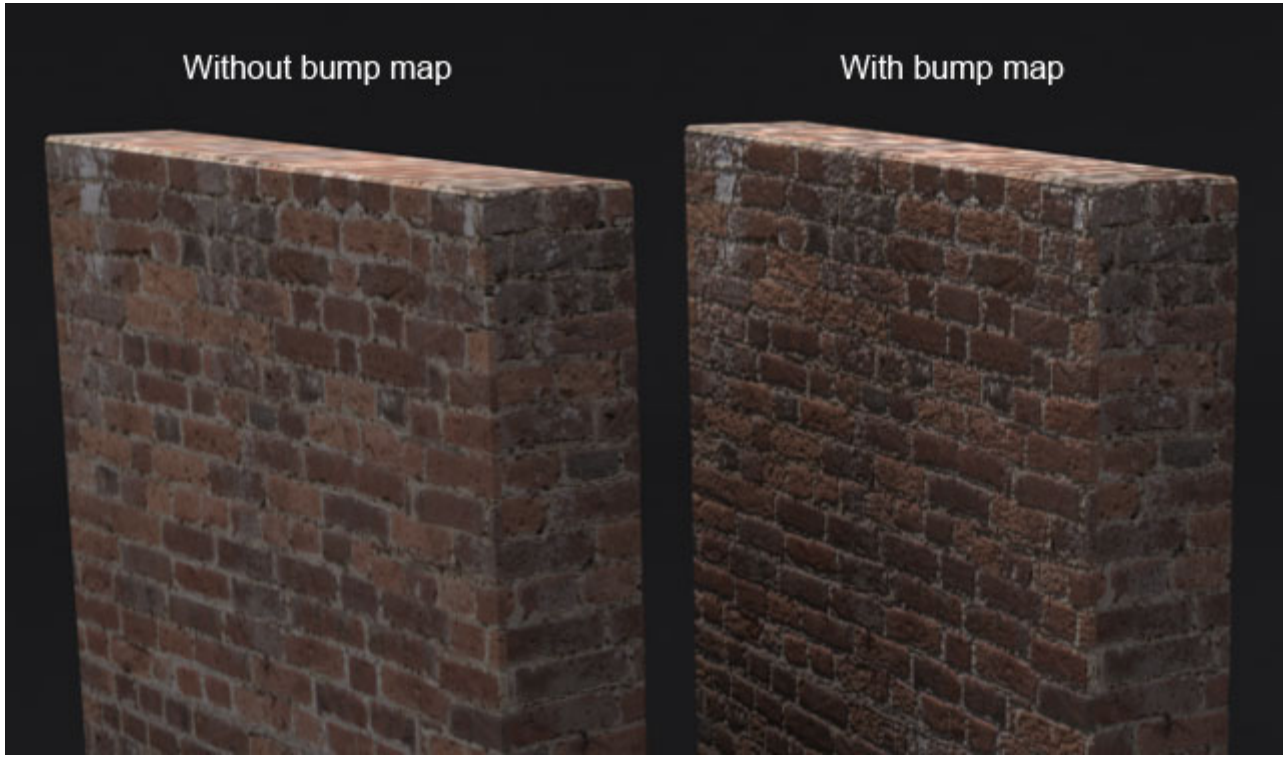
$$L(\mathbf{w}) \approx \mathbf{w} \cdot \mathbf{d}$$

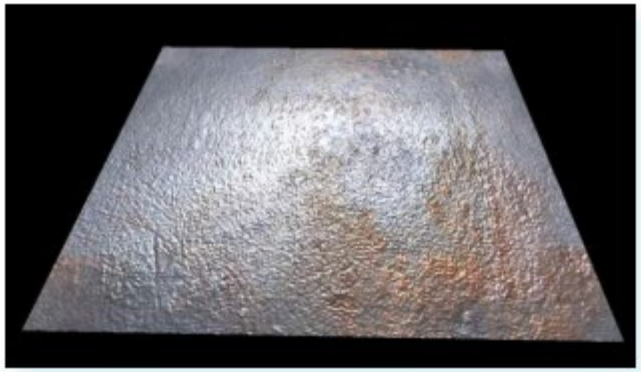
Bump/Normal Maps

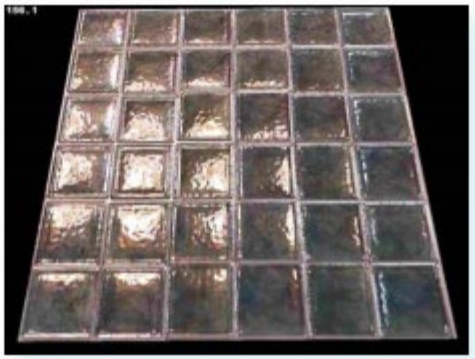
Bump Map

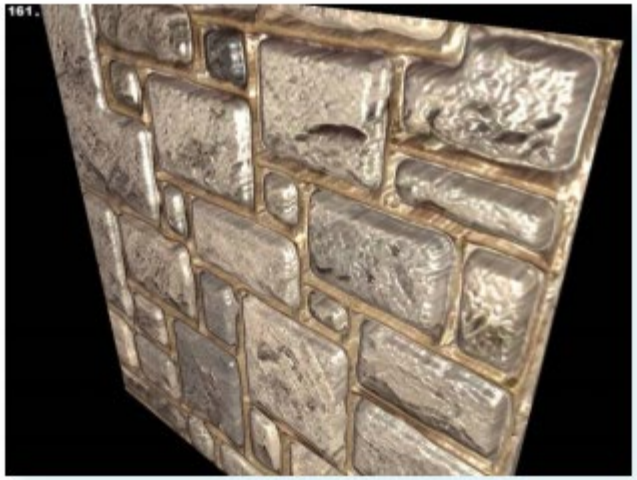



P(u)
Original Surface
B(u)
A bump map
P'(u)
Lengthening or shortening
P(u) using B(u)
N'(u)
The vectors to the
'new' surface


Without bump map
With bump map

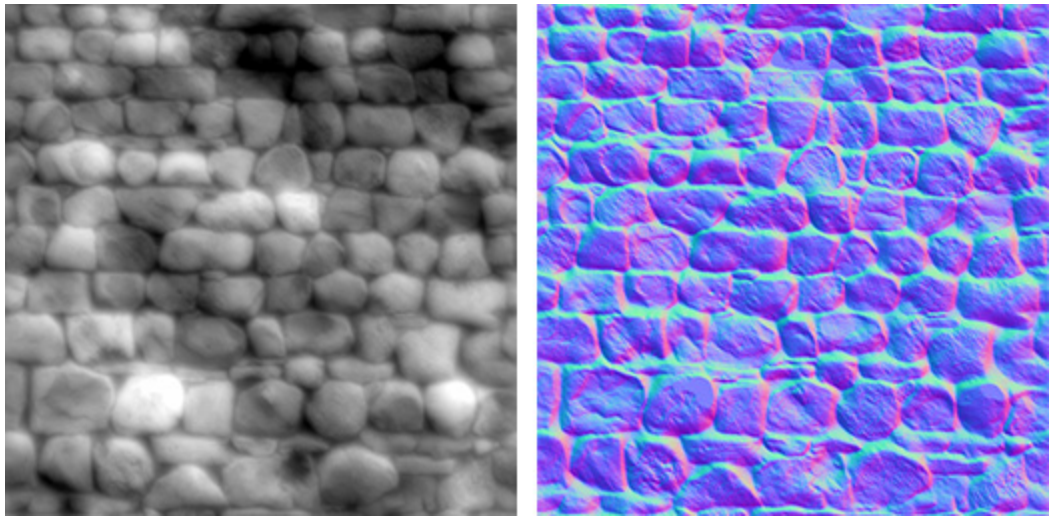




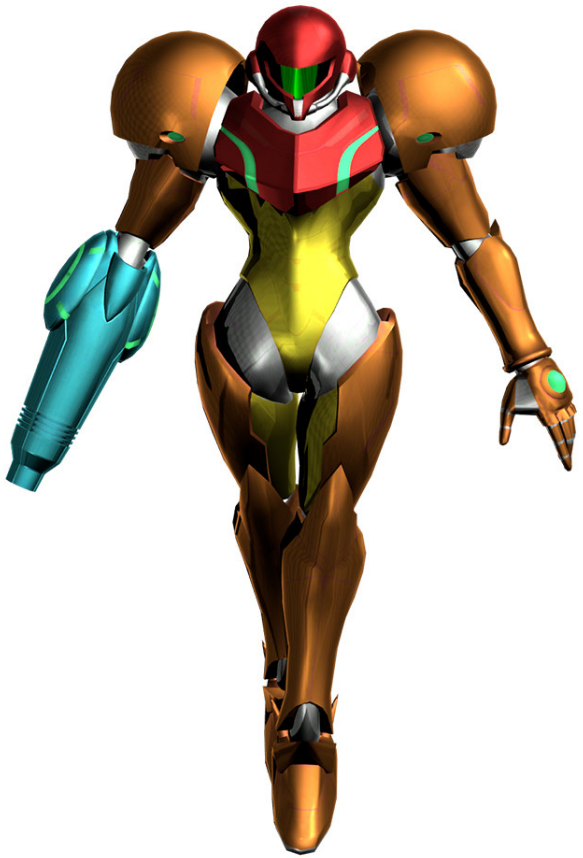




Bump Map vs Normal Map



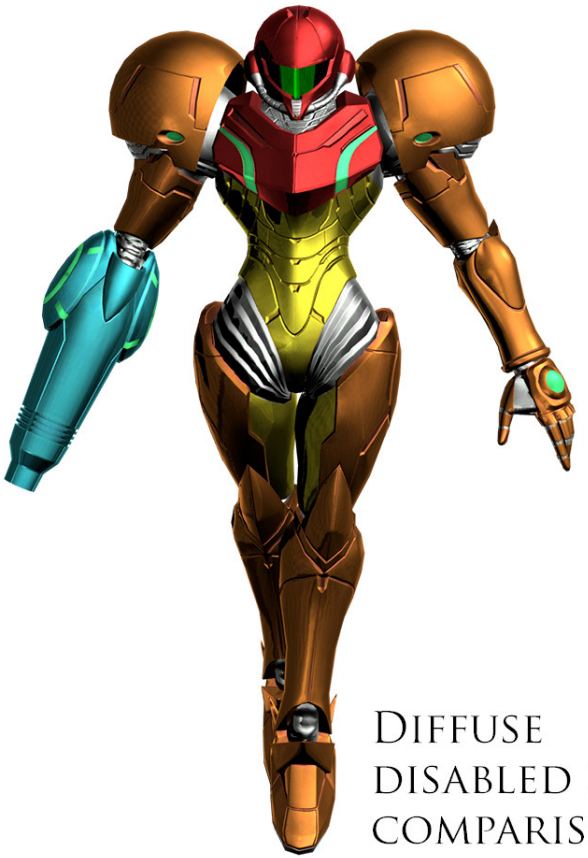
No Normal Map



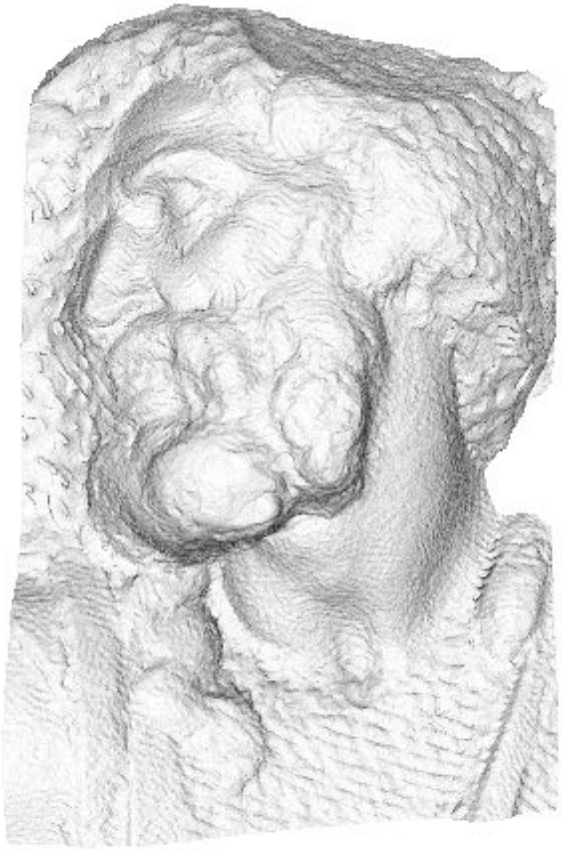
Normal Map



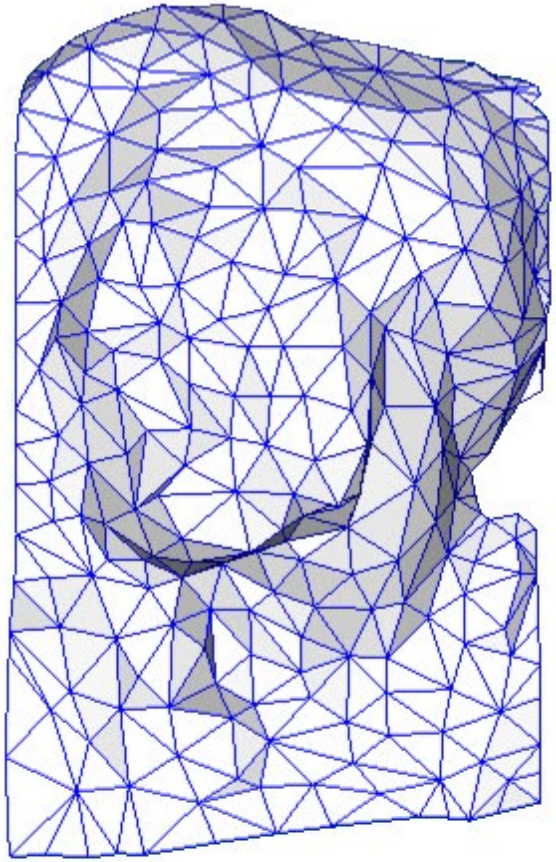
With Normal Map



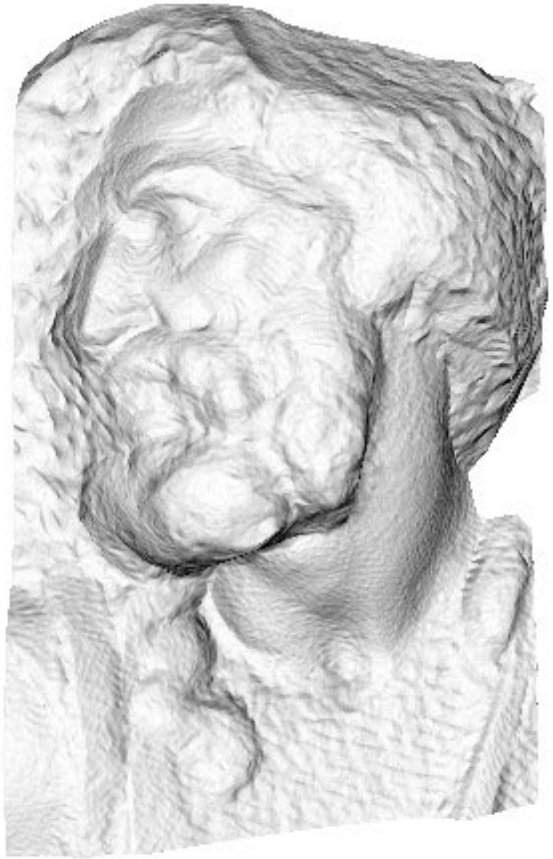
Diffuse disabled for comparison



original mesh
4M triangles



simplified mesh
500 triangles



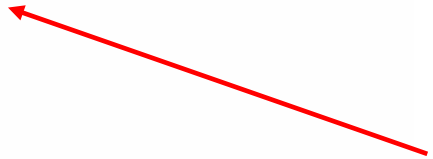
simplified mesh
and normal mapping
500 triangles





WebGL

Listing 8.5 PointLightedCube_perFragment.js



```
// PointLightedCube_perFragment.js
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  ...
  'uniform mat4 u_ModelMatrix;\n' +  // Model matrix
  'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
  'varying vec4 v_Color;\n' +
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'void main() {\n' +
  '  gl_Position = u_MvpMatrix * a_Position;\n' +
     // Calculate the vertex position in the world coordinate
  '  v_Position = vec3(u_ModelMatrix * a_Position);\n' +
  '  v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
  '  v_Color = a_Color;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  ...
  'uniform vec3 u_LightColor;\n' +     // Light color
  'uniform vec3 u_LightPosition;\n' +  // Position of the light source
  'uniform vec3 u_AmbientLight;\n' +   // Ambient light color
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
     // Normalize normal because it's interpolated and not 1.0 (length)
  '  vec3 normal = normalize(v_Normal);\n' +
     // Calculate the light direction and make it 1.0 in length
  '  vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
     // The dot product of the light direction and the normal
  '  float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
     // Calculate the final color from diffuse and ambient reflection
  '  vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
  '  vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
  '  gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
  '}\n';
```

Listing 8.5 PointLightedCube_perFragment.js

```
// PointLightedCube_perFragment.js
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  ...
  'uniform mat4 u_ModelMatrix;\n' +  // Model matrix
  'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
  'varying vec4 v_Color;\n' +
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'void main() {\n' +
  '  gl_Position = u_MvpMatrix * a_Position;\n' +
     // Calculate the vertex position in the world coordinate
  '  v_Position = vec3(u_ModelMatrix * a_Position);\n' +
  '  v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
  '  v_Color = a_Color;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  ...
  'uniform vec3 u_LightColor;\n' +     // Light color
  'uniform vec3 u_LightPosition;\n' +  // Position of the light source
  'uniform vec3 u_AmbientLight;\n' +   // Ambient light color
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
     // Normalize normal because it's interpolated and not 1.0 (length)
  '  vec3 normal = normalize(v_Normal);\n' +
     // Calculate the light direction and make it 1.0 in length
  '  vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
     // The dot product of the light direction and the normal
  '  float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
     // Calculate the final color from diffuse and ambient reflection
  '  vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
  '  vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
  '  gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
  '}\n';
```

ProjectionMatrix * ViewMatrix * ModelMatri

Listing 8.5 PointLightedCube_perFragment.js

```
// PointLightedCube_perFragment.js
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  ...
  'uniform mat4 u_ModelMatrix;\n' +  // Model matrix
  'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
  'varying vec4 v_Color;\n' +
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'void main() {\n' +
  '  gl_Position = u_MvpMatrix * a_Position;\n' +
    // Calculate the vertex position in the world coordinate
  '  v_Position = vec3(u_ModelMatrix * a_Position);\n' +
  '  v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
  '  v_Color = a_Color;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  ...
  'uniform vec3 u_LightColor;\n' +     // Light color
  'uniform vec3 u_LightPosition;\n' +  // Position of the light source
  'uniform vec3 u_AmbientLight;\n' +   // Ambient light color
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
    // Normalize normal because it's interpolated and not 1.0 (length)
  '  vec3 normal = normalize(v_Normal);\n' +
    // Calculate the light direction and make it 1.0 in length
  '  vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
    // The dot product of the light direction and the normal
  '  float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
    // Calculate the final color from diffuse and ambient reflection
  '  vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
  '  vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
  '  gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
  '}\n';
```

Only use the ModelMatrix to get World Coord

Listing 8.5 PointLightedCube_perFragment.js

```
// PointLightedCube_perFragment.js
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  ...
  'uniform mat4 u_ModelMatrix;\n' +  // Model matrix
  'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
  'varying vec4 v_Color;\n' +
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'void main() {\n' +
  '  gl_Position = u_MvpMatrix * a_Position;\n' +
    // Calculate the vertex position in the world coordinate
  '  v_Position = vec3(u_ModelMatrix * a_Position);\n' +
  '  v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
  '  v_Color = a_Color;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  ...
  'uniform vec3 u_LightColor;\n' +     // Light color
  'uniform vec3 u_LightPosition;\n' +  // Position of the light source
  'uniform vec3 u_AmbientLight;\n' +   // Ambient light color
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
    // Normalize normal because it's interpolated and not 1.0 (length)
  '  vec3 normal = normalize(v_Normal);\n' +
    // Calculate the light direction and make it 1.0 in length
  '  vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
    // The dot product of the light direction and the normal
  '  float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
    // Calculate the final color from diffuse and ambient reflection
  '  vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
  '  vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
  '  gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
  '}\n';
```

NormalMatrix not ModelMatrix - see book

Listing 8.5 PointLightedCube_perFragment.js

```
// PointLightedCube_perFragment.js
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  ...
  'uniform mat4 u_ModelMatrix;\n' +  // Model matrix
  'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
  'varying vec4 v_Color;\n' +
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'void main() {\n' +
  '  gl_Position = u_MvpMatrix * a_Position;\n' +
     // Calculate the vertex position in the world coordinate
  '  v_Position = vec3(u_ModelMatrix * a_Position);\n' +
  '  v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
  '  v_Color = a_Color;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  ...
  'uniform vec3 u_LightColor;\n' +     // Light color
  'uniform vec3 u_LightPosition;\n' +  // Position of the light source
  'uniform vec3 u_AmbientLight;\n' +   // Ambient light color
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
     // Normalize normal because it's interpolated and not 1.0 (length)
  '  vec3 normal = normalize(v_Normal);\n' +
     // Calculate the light direction and make it 1.0 in length
  '  vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
     // The dot product of the light direction and the normal
  '  float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
     // Calculate the final color from diffuse and ambient reflection
  '  vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
  '  vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
  '  gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
  '}\n';
```

Interpolate these things in the fragment shader

Listing 8.5 PointLightedCube_perFragment.js

```
// PointLightedCube_perFragment.js
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  ...
  'uniform mat4 u_ModelMatrix;\n' +  // Model matrix
  'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
  'varying vec4 v_Color;\n' +
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'void main() {\n' +
  '  gl_Position = u_MvpMatrix * a_Position;\n' +
     // Calculate the vertex position in the world coordinate
  '  v_Position = vec3(u_ModelMatrix * a_Position);\n' +
  '  v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
  '  v_Color = a_Color;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  ...
  'uniform vec3 u_LightColor;\n' +     // Light color
  'uniform vec3 u_LightPosition;\n' +  // Position of the light source
  'uniform vec3 u_AmbientLight;\n' +   // Ambient light color
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
     // Normalize normal because it's interpolated and not 1.0 (length)
  '  vec3 normal = normalize(v_Normal);\n' +
     // Calculate the light direction and make it 1.0 in length
  '  vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
     // The dot product of the light direction and the normal
  '  float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
     // Calculate the final color from diffuse and ambient reflection
  '  vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
  '  vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
  '  gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
  '}\n';
```

Normalize again since interpolation might mess up the length

Listing 8.5 PointLightedCube_perFragment.js

```
// PointLightedCube_perFragment.js
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  ...
  'uniform mat4 u_ModelMatrix;\n' +  // Model matrix
  'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
  'varying vec4 v_Color;\n' +
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'void main() {\n' +
  '  gl_Position = u_MvpMatrix * a_Position;\n' +
     // Calculate the vertex position in the world coordinate
  '  v_Position = vec3(u_ModelMatrix * a_Position);\n' +
  '  v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
  '  v_Color = a_Color;\n' +
  '}\n';

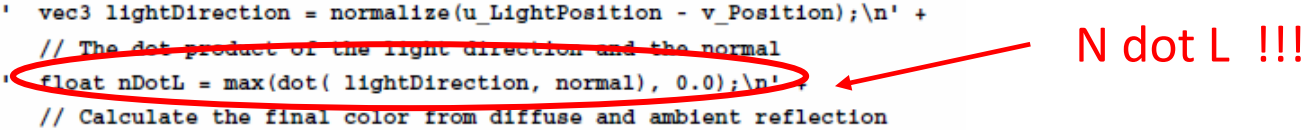
// Fragment shader program
var FSHADER_SOURCE =
  ...
  'uniform vec3 u_LightColor;\n' +     // Light color
  'uniform vec3 u_LightPosition;\n' +  // Position of the light source
  'uniform vec3 u_AmbientLight;\n' +   // Ambient light color
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
     // Normalize normal because it's interpolated and not 1.0 (length)
  '  vec3 normal = normalize(v_Normal);\n' +
     // Calculate the light direction and make it 1.0 in length
  '  vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
     // The dot product of the light direction and the normal
  '  float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
     // Calculate the final color from diffuse and ambient reflection
  '  vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
  '  vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
  '  gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
  '}\n';
```

LightDirection

Listing 8.5 PointLightedCube_perFragment.js

```
// PointLightedCube_perFragment.js
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  ...
  'uniform mat4 u_ModelMatrix;\n' +  // Model matrix
  'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
  'varying vec4 v_Color;\n' +
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'void main() {\n' +
  '  gl_Position = u_MvpMatrix * a_Position;\n' +
     // Calculate the vertex position in the world coordinate
  '  v_Position = vec3(u_ModelMatrix * a_Position);\n' +
  '  v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
  '  v_Color = a_Color;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  ...
  'uniform vec3 u_LightColor;\n' +     // Light color
  'uniform vec3 u_LightPosition;\n' +  // Position of the light source
  'uniform vec3 u_AmbientLight;\n' +   // Ambient light color
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
     // Normalize normal because it's interpolated and not 1.0 (length)
  '  vec3 normal = normalize(v_Normal);\n' +
     // Calculate the light direction and make it 1.0 in length
  '  vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
     // The dot product of the light direction and the normal
  '  float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
     // Calculate the final color from diffuse and ambient reflection
  '  vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
  '  vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
  '  gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
  '}\n';
```



Listing 8.5 PointLightedCube_perFragment.js

```
// PointLightedCube_perFragment.js
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  ...
  'uniform mat4 u_ModelMatrix;\n' +  // Model matrix
  'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
  'varying vec4 v_Color;\n' +
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'void main() {\n' +
  '  gl_Position = u_MvpMatrix * a_Position;\n' +
     // Calculate the vertex position in the world coordinate
  '  v_Position = vec3(u_ModelMatrix * a_Position);\n' +
  '  v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
  '  v_Color = a_Color;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  ...
  'uniform vec3 u_LightColor;\n' +     // Light color
  'uniform vec3 u_LightPosition;\n' +  // Position of the light source
  'uniform vec3 u_AmbientLight;\n' +   // Ambient light color
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
     // Normalize normal because it's interpolated and not 1.0 (length)
  '  vec3 normal = normalize(v_Normal);\n' +
     // Calculate the light direction and make it 1.0 in length
  '  vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
     // The dot product of the light direction and the normal
  '  float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
     // Calculate the final color from diffuse and ambient reflection
  '  vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
  '  vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
  '  gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
  '}\n';
```

Light color

Triangle vertex color

Listing 8.5 PointLightedCube_perFragment.js

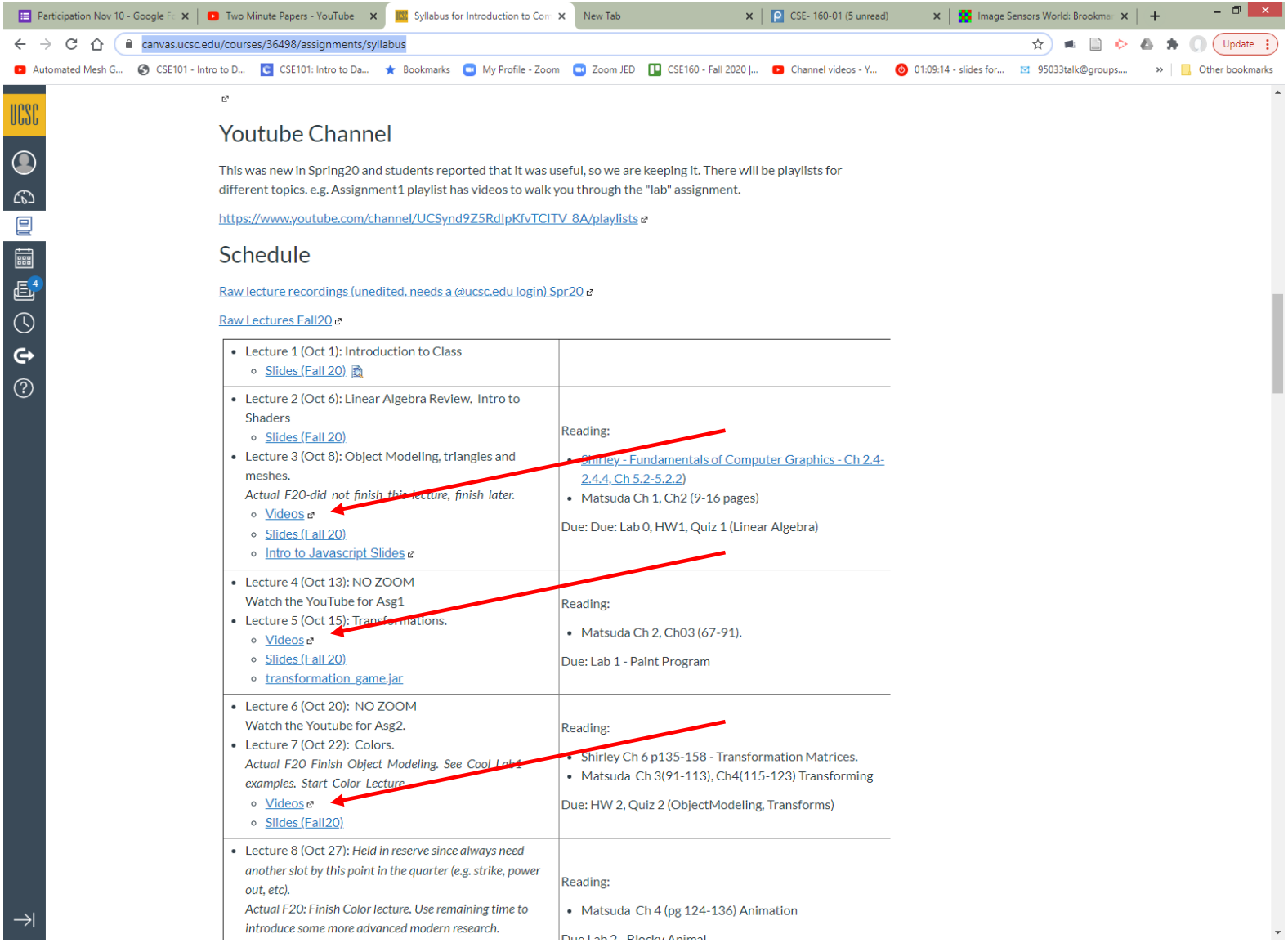
```
// PointLightedCube_perFragment.js
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  ...
  'uniform mat4 u_ModelMatrix;\n' +  // Model matrix
  'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
  'varying vec4 v_Color;\n' +
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'void main() {\n' +
  '  gl_Position = u_MvpMatrix * a_Position;\n' +
     // Calculate the vertex position in the world coordinate
  '  v_Position = vec3(u_ModelMatrix * a_Position);\n' +
  '  v_Normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
  '  v_Color = a_Color;\n' +
  '}\n';

// Fragment shader program
var FSHADER_SOURCE =
  ...
  'uniform vec3 u_LightColor;\n' +     // Light color
  'uniform vec3 u_LightPosition;\n' +  // Position of the light source
  'uniform vec3 u_AmbientLight;\n' +   // Ambient light color
  'varying vec3 v_Normal;\n' +
  'varying vec3 v_Position;\n' +
  'varying vec4 v_Color;\n' +
  'void main() {\n' +
     // Normalize normal because it's interpolated and not 1.0 (length)
  '  vec3 normal = normalize(v_Normal);\n' +
     // Calculate the light direction and make it 1.0 in length
  '  vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
     // The dot product of the light direction and the normal
  '  float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
     // Calculate the final color from diffuse and ambient reflection
  '  vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
  '  vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
  '  gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
  '}\n';
```

A4: Need to add Specular

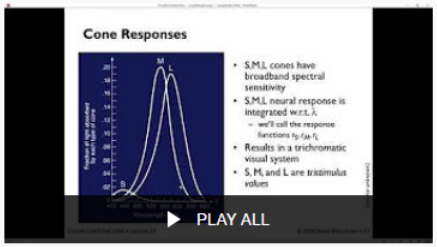
Administrative

Chopped up videos



Participation Nov 10 - Go | Two Minute Papers - You | Syllabus for Introduction | L7: Color - YouTube | New Tab | CSE- 160-01 (5 unread) | Image Sensors World: Bro
youtube.com/playlist?list=PLbyTU_tFIkcPvnBXuw0JkLhQmiZwNDzEc
Update
Automated Mesh G... | CSE101 - Intro to D... | CSE101: Intro to Da... | Bookmarks | My Profile - Zoom | Zoom JED | CSE160 - Fall 2020 |... | Channel videos - Y... | 01:09:14 - slides for... | 95033talk@groups.... | Other bookmarks

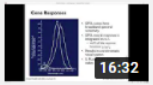
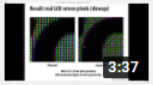
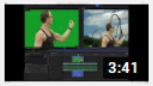
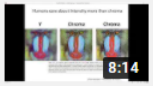
Home
Trending
Subscriptions
Library
History
Your videos
Your movies
Watch later
Liked videos
SUBSCRIPTIONS
aLeX
UCSC CSE160
Browse channels
MORE FROM YOUTUBE
YouTube Premium
Movies & Shows
Gaming
Live
Fashion & Beauty
Learning
Settings
Report history
Help



L7: Color

10 videos • 59 views • Last updated on May 5, 2020

UCSC CSE160 SUBSCRIBED

1  **Perception** — UCSC CSE160
2  **Additive Color** — UCSC CSE160
3  **RGB Color Space Sample Problem with Explanation** — UCSC CSE160
4  **Alpha** — UCSC CSE160
5  **Subtractive Color** — UCSC CSE160
6  **CMY Color Space Example Problem with Explanation** — UCSC CSE160
7  **HSV Color** — UCSC CSE160
8  **HSV COLOR SPACE SAMPLE PROBLEM WITH EXPLANATION** — UCSC CSE160
9  **YIQ YUV Color Space** — UCSC CSE160
10  **Color Gamuts, Color Matching, and XYZ** — UCSC CSE160

Due Dates

- Due Yesterday
  - Quiz 3 (Color/Texture)
- Due next Monday
  - Lab Assignment 3 (Blocky World)

Q&A

End