Illumination

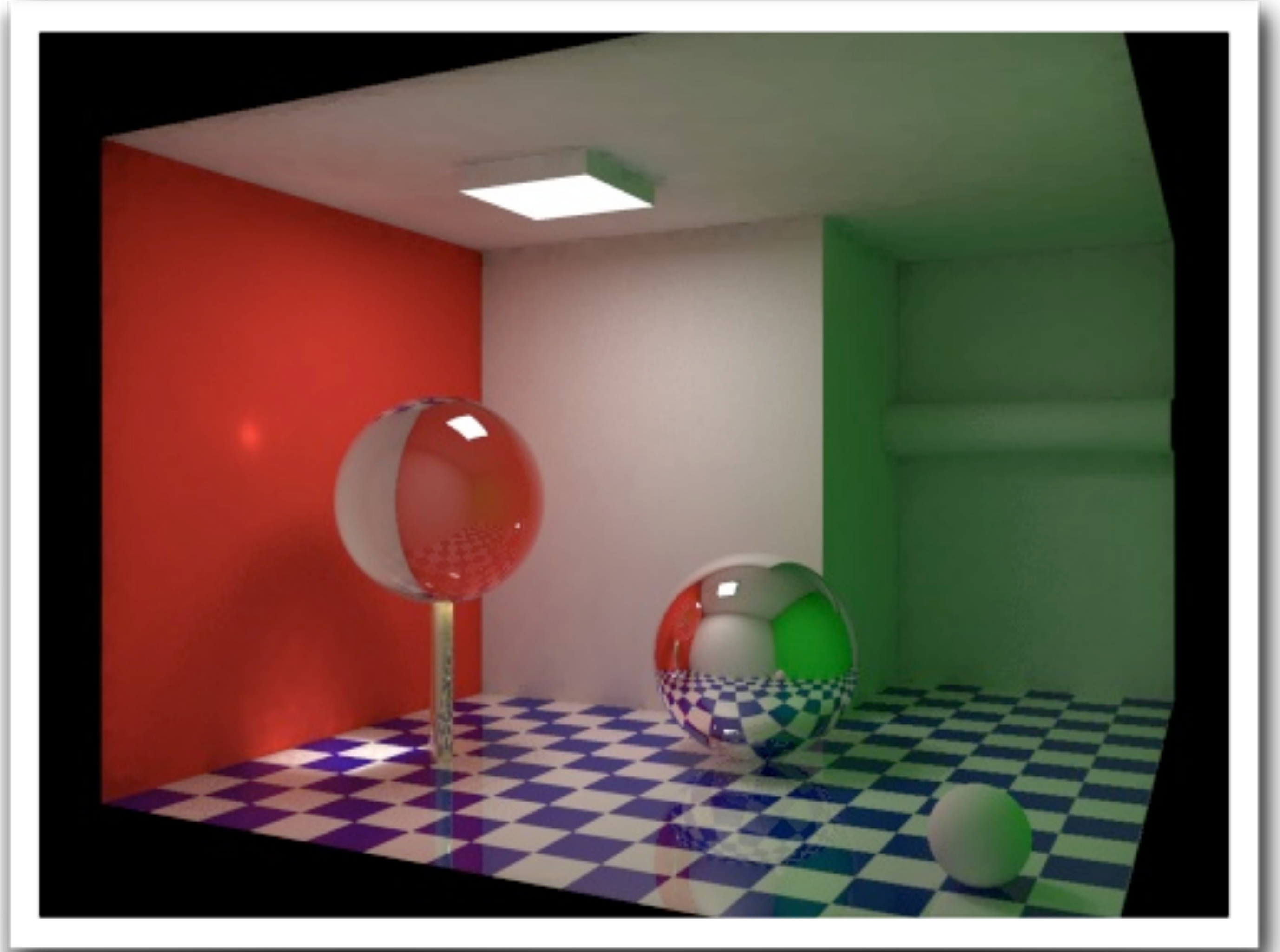


Image by Smijes08

Illumination - Object Appearance

Light transport in a scene

- Light emitted from light sources
- On impact with objects some light is reflected, refracted, and absorbed by materials
- The reflection distribution determines the “finish” of the material (matte, glossy, shiny, etc)
- The amount and properties of light that reaches the eye determines what we see

Illumination

Realistic light sources can be very complex

- Projector, TV, lamps of various kinds, the sun

Surfaces can have varying characteristics

- Translucent
- Rough (microstructures)
- Anisotropic
- Iridescent, fluorescent etc

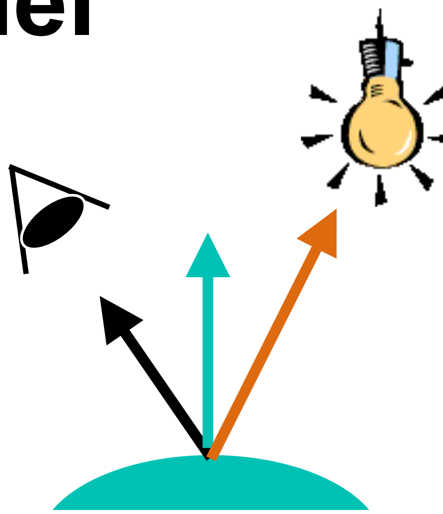
Local (Direct) Illumination

Local interaction between a light source and a surface

Simplified as follows:

- Point light source:
  - *Infinitesimal point in space*
  - *Isotropic light emission: emits light equally in all directions*
  - *Can be modelled with light rays starting at the light's location*

Basic Local Illumination Model



Light that reaches the eye

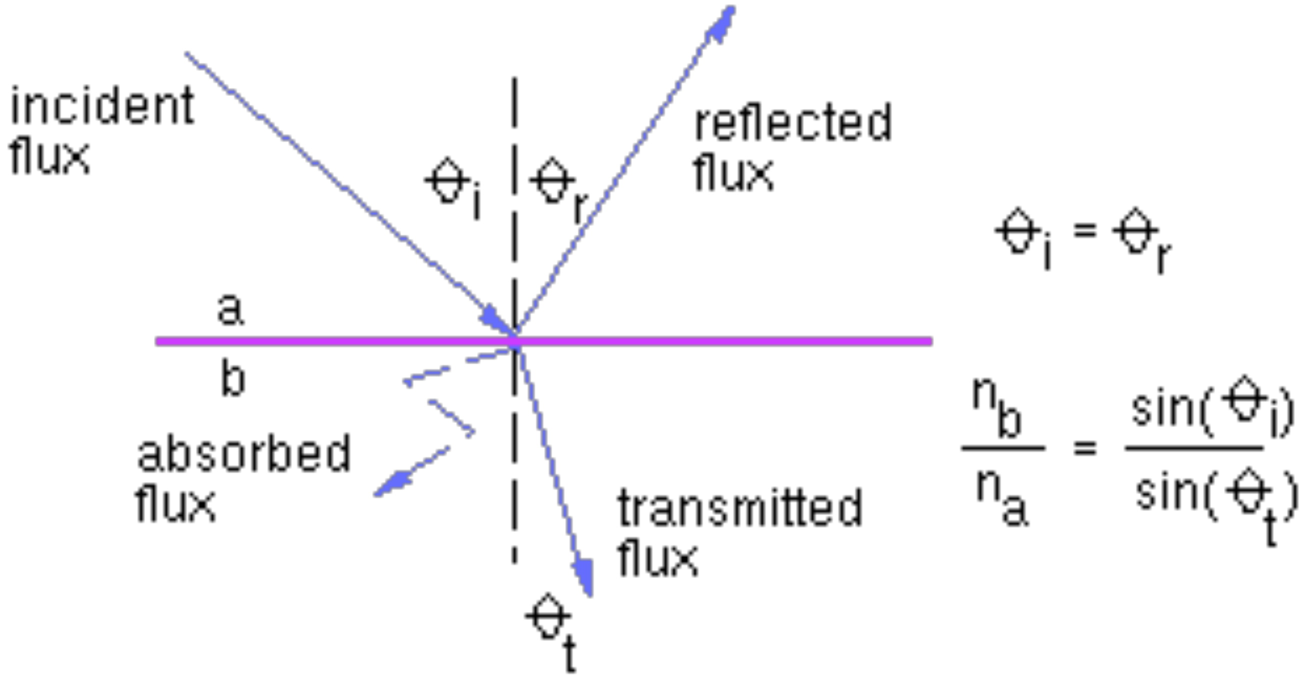
- Depends on light and eye positions, and surface's reflectance

Light represented with RGB triplets

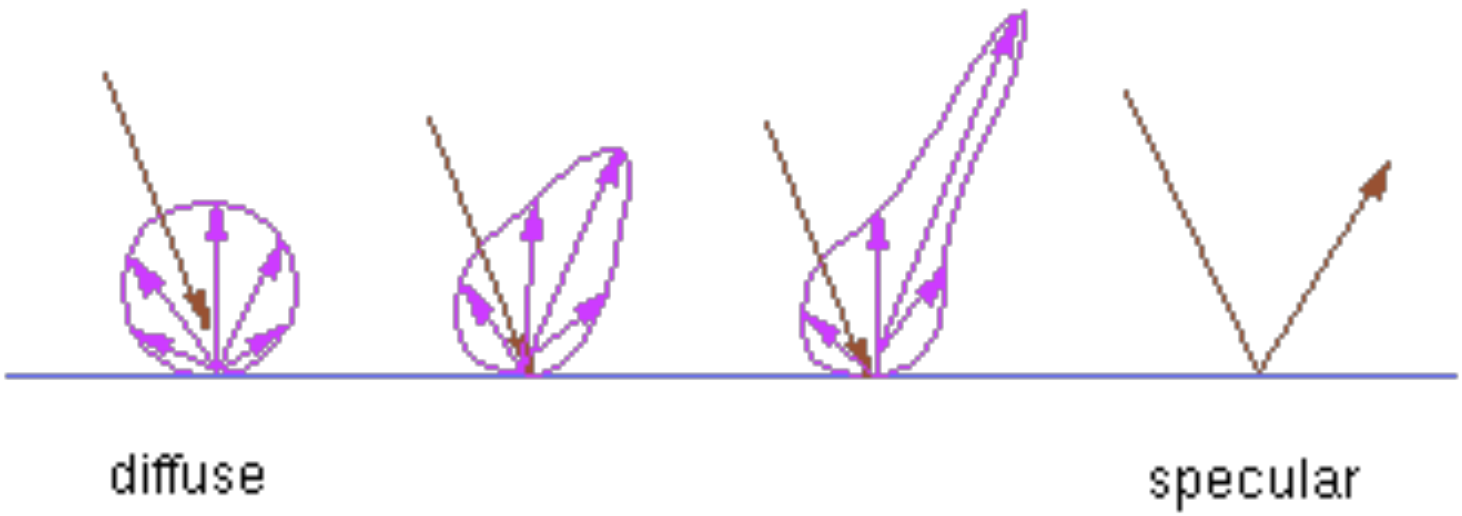
We want the amount (intensity) of light that the point reflects towards the viewer

Local Illumination physics

Law of reflection and Snell's law of refraction



What are we trying to model ?



- Keep things simple and computationally efficient
- Sufficient expressive power for a wide range of materials

Diffuse Reflection

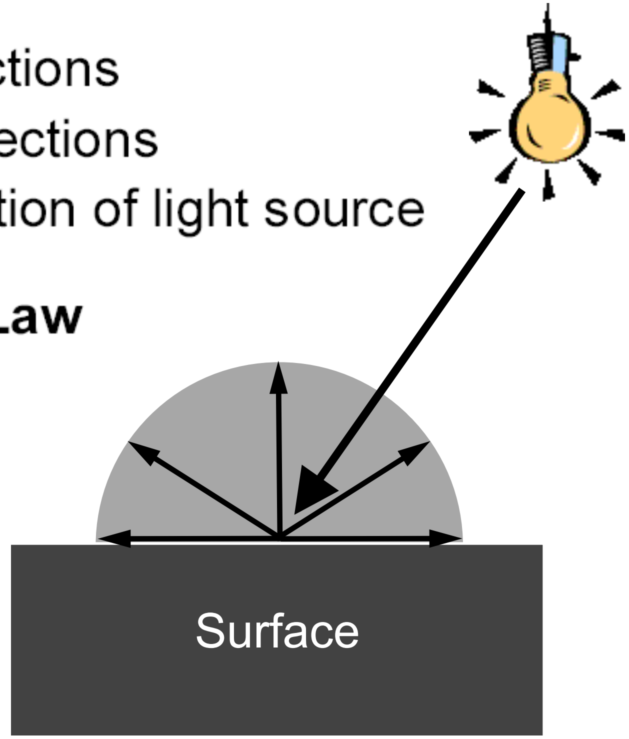
This is the simplest kind of reflection

- also called Lambertian reflection
- models dull, matte surfaces — materials like chalk

Ideal diffuse reflection

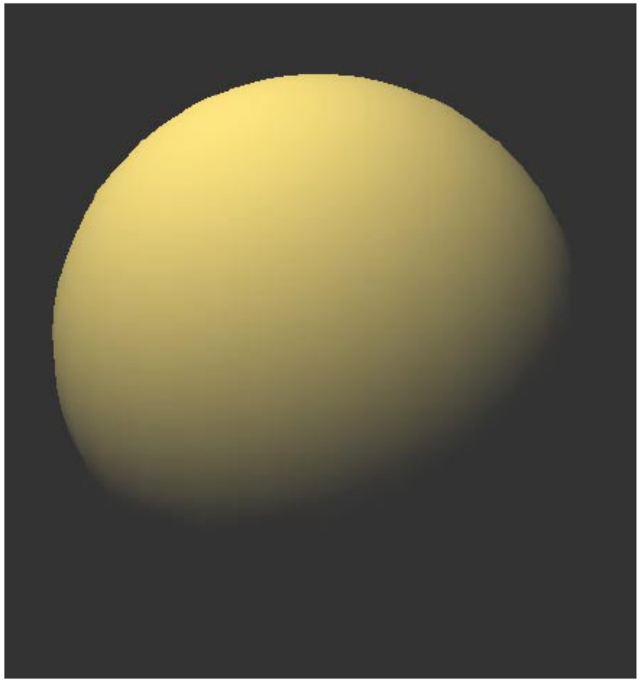
- scatters incoming light equally in all directions
- identical appearance from all viewing directions
- reflected intensity depends only on direction of light source

Light is reflected according to Lambert's Law

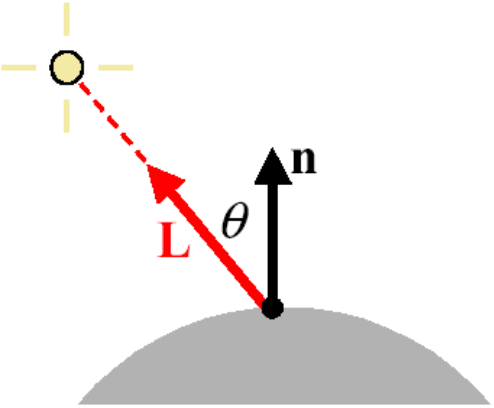


Lambert's Law for Diffuse Reflection

Purely diffuse object



$$I = I_L k_d \cos\theta$$
$$= I_L k_d (\mathbf{n} \cdot \mathbf{L})$$



I : resulting intensity

I_L : light source intensity

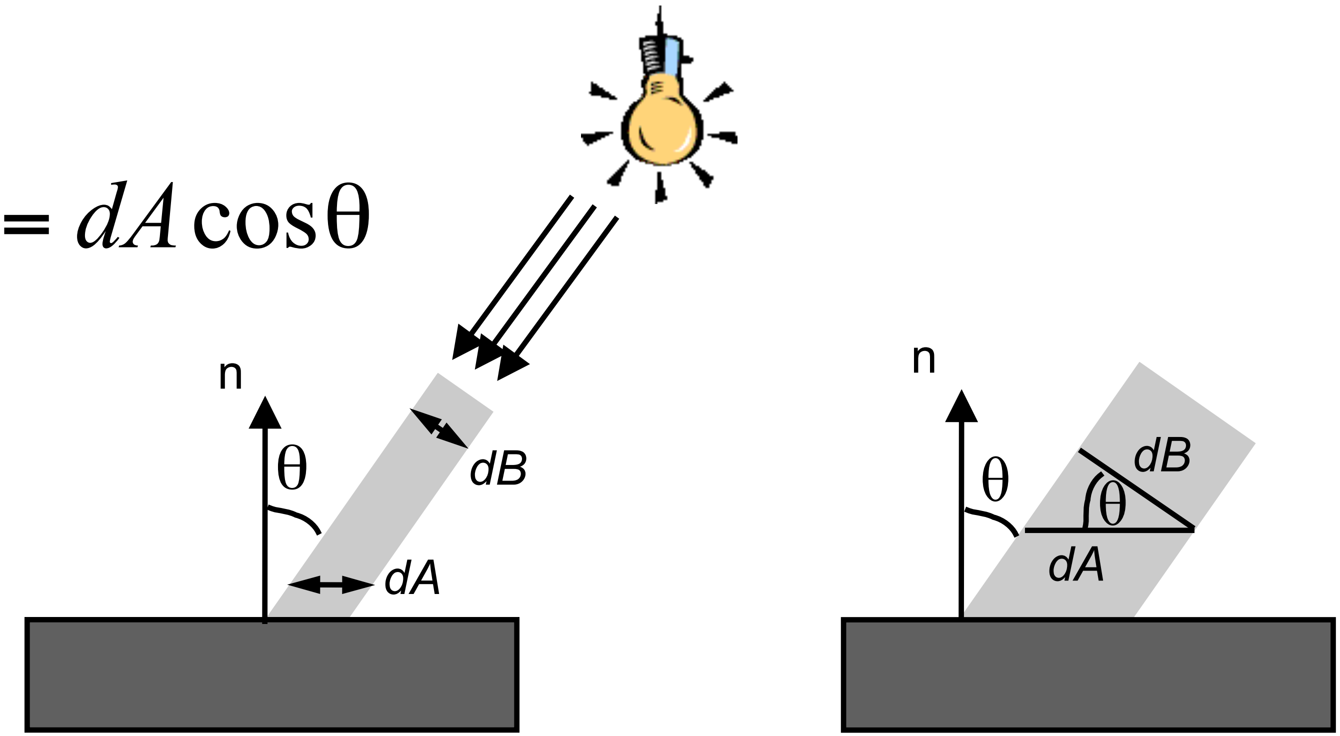
k_d : (diffuse) surface reflectance coefficient

$$k_d \in [0,1]$$

θ : angle between normal & light direction

Lambert's cosine law

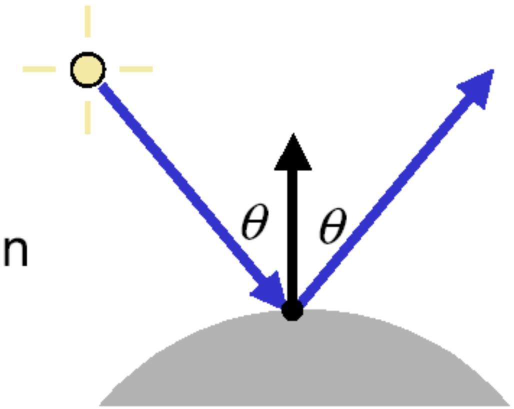
$$dB = dA\cos\theta$$



Specular Reflection

Diffuse reflection is nice, but many surfaces are shiny

- their appearance changes as the viewpoint moves
- they have glossy specular highlights (or specularities)
- because they reflect light coherently, in a preferred direction

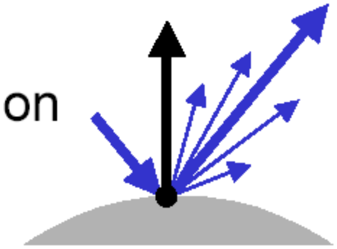


A mirror is a perfect specular reflector

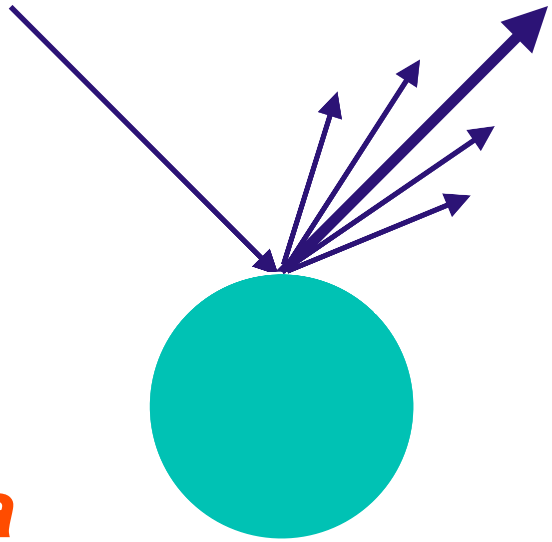
- incoming ray reflected about normal direction
- nothing reflected in any other direction

Most surfaces are imperfect specular reflectors

- reflect rays in cone about perfect reflection direction



How do we model specular reflection?



We want a

- simple,
- efficient,
- and intuitive

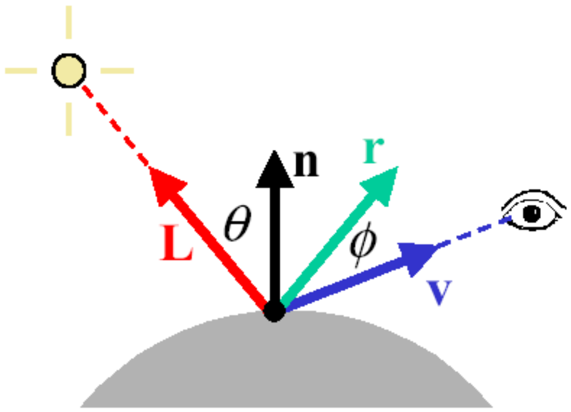
model

Phong Illumination Model

$$I = I_L k_d \cos\theta + \boxed{I_L k_s \cos^n \phi}$$
$$= I_L k_d (\mathbf{n}\cdot\mathbf{L}) + \boxed{I_L k_s (\mathbf{r}\cdot\mathbf{v})^n}$$

One particular specular reflection model

- quite common in practice
- it is purely empirical
- there's *no physical basis* for it



I : resulting intensity

I_L : light source intensity

k_s : (specular) surface reflectance coefficient

$$k_s \in [0,1]$$

ϕ : angle between viewing & reflection direction

n : "shininess" factor

Computing R

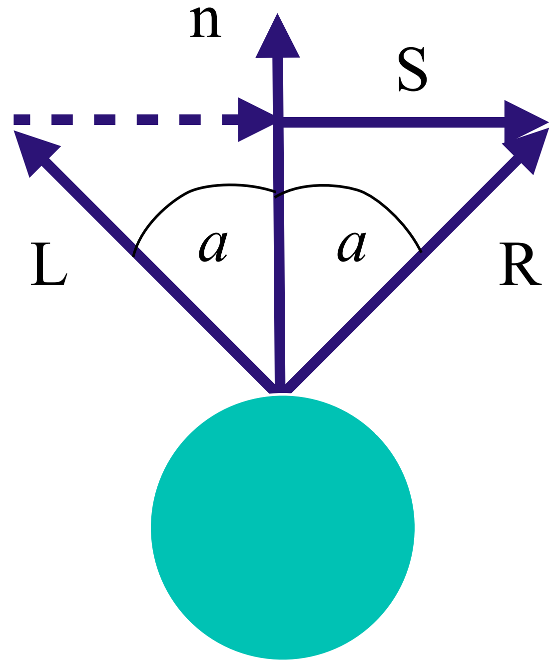
- Convention L towards light
- n, R, L unit vectors

$$R = (n \cdot L)n + S$$

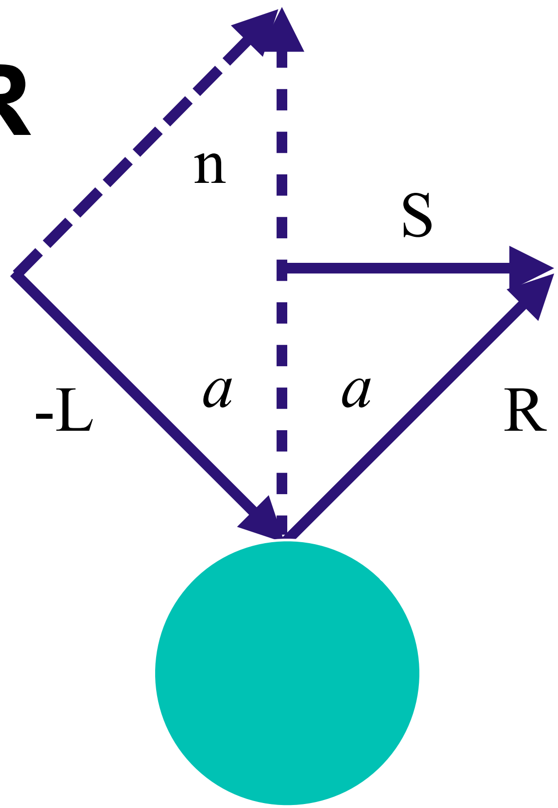
$$S = (n \cdot L)n - L$$

substituting we get

$$R = 2(n \cdot L)n - L$$



Computing R



- Convention L towards light
- n,R,L unit vectors

$$R = (n \cdot L)n + S$$

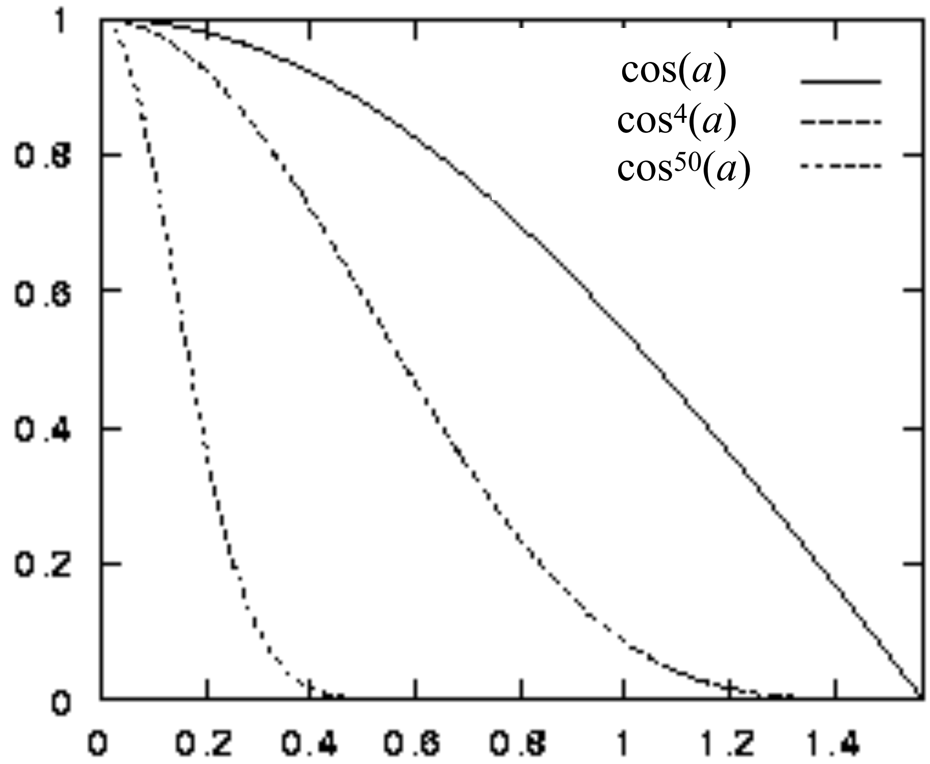
$$S = (n \cdot L)n - L$$

substituting we get

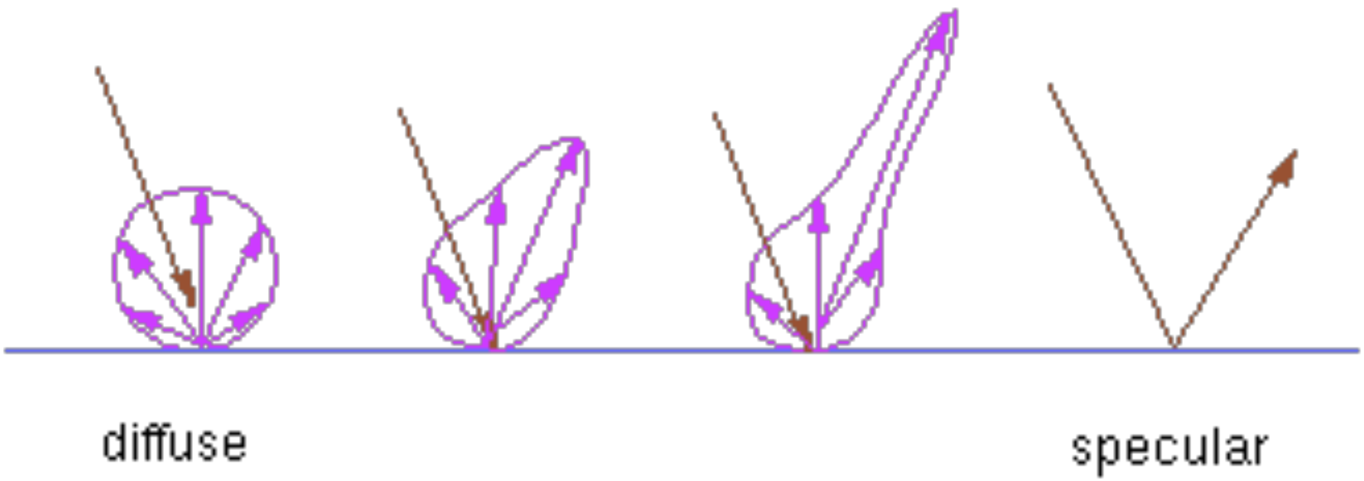
$$R = 2(n \cdot L)n - L$$

Sanity check: we can visualize what we computed for R *as the doted vector*

The effect of the exponent *n*



Comparison

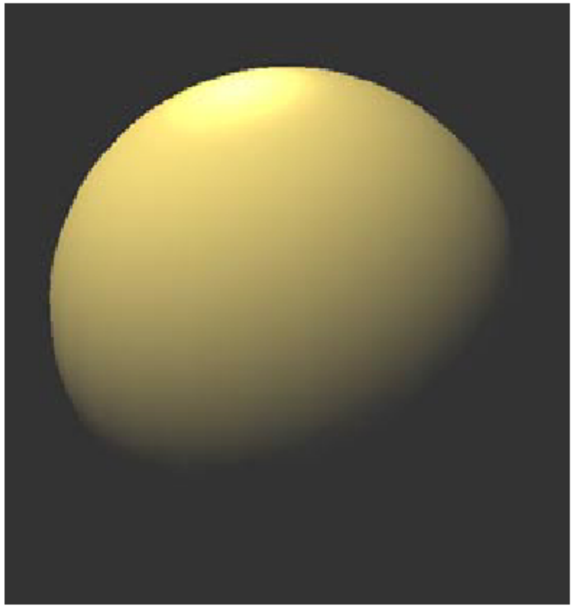


Examples of Phong Specular Model

Diffuse only

Diffuse + Specular (shininess 5)

Diffuse + Specular (shininess 50)



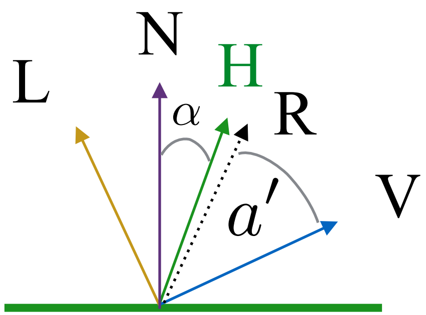
The Blinn-Torrance Specular Model

Agrees better with experimental results

Halfway vector H between L,V

$$H = \frac{L+V}{|L+V|}$$

$$I_s = I_s K_s (H \cdot N)^n$$



Usually $H \cdot N > R \cdot V$ so to match

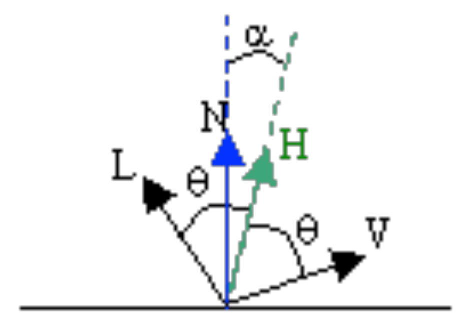
$(R \cdot V)^n$ use $(H \cdot N)^{n'}, n' > n$

It measures how far from the normal N, H is, which varies similar to how far from V, R is.

Advantages of the Blinn Specular Model

- Theoretical basis
- No need to compute reflective direction R
- N·H cannot be negative if N·L>0 and N·V>0
- If the light is directional and we have orthographic projection then N·H constant

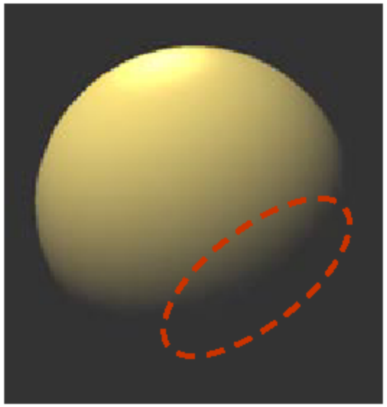
$$H = \frac{L + V}{\|L + V\|}$$



The Ambient Glow

So far, areas not directly illuminated by any light appear black

- this tends to look rather unnatural
- in the real world, there's lots of ambient light



To compensate, we invent new light source

- assume there is a constant ambient "glow"
- this ambient glow is *purely fictitious*

Just add in another term to our illumination equation

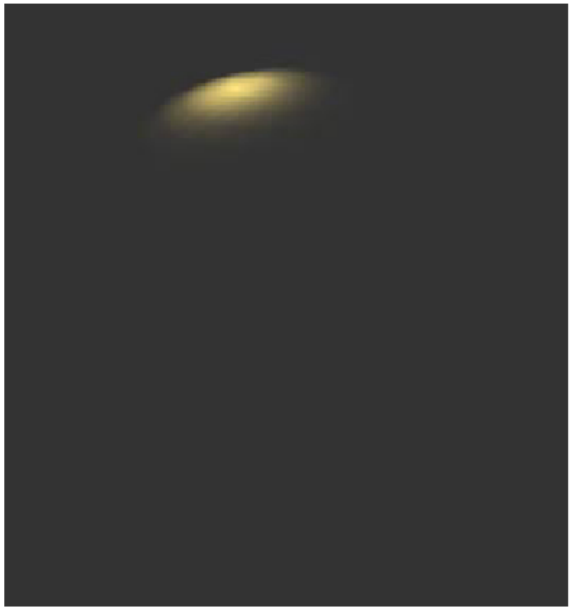
$$I = I_L k_d \cos\theta + I_L k_s \cos^n \phi + I_a k_a$$

I_a : ambient light intensity

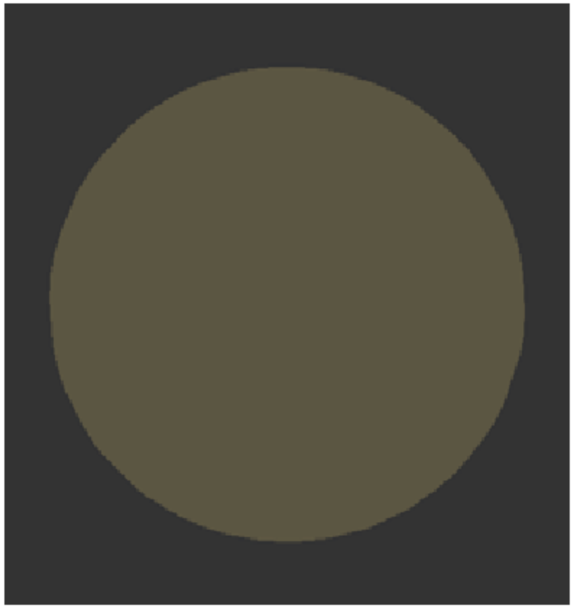
k_a : (ambient) surface reflectance coefficient

Our Three Basic Components of Illumination

Diffuse

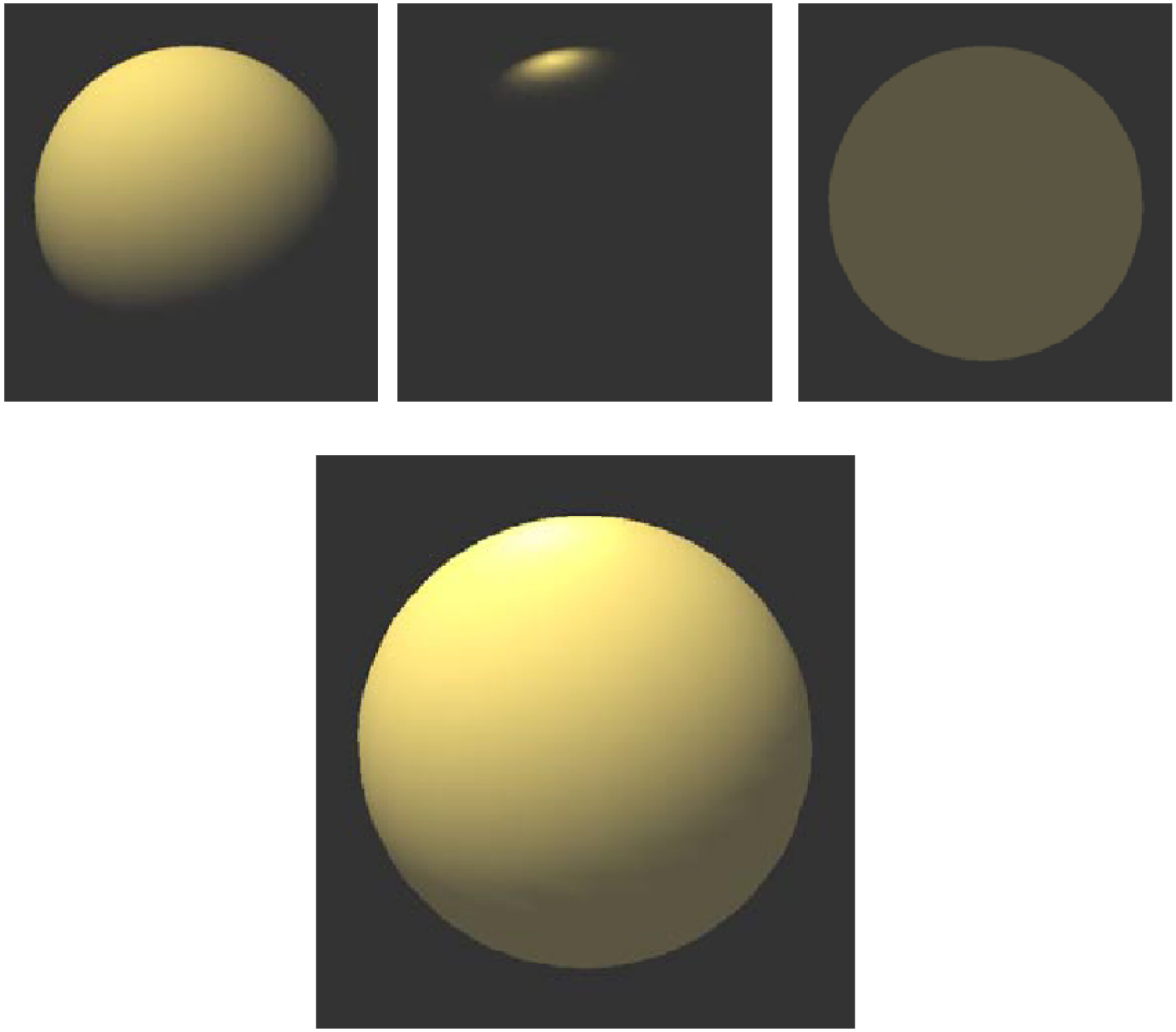


Specular



Ambient

Combined for the Final Result : ADS - Lighting



Lights and materials

$$ObjectColor_r = I_r = I_{a,r} K_{a,r} + I_{i,r} K_{diff,r}(N \cdot L) + I_{i,r} K_{spec,r} (R \cdot V)^n$$

$$ObjectColor_g = I_g = I_{a,g} K_{a,g} + I_{i,g} K_{diff,g}(N \cdot L) + I_{i,g} K_{spec,g}(R \cdot V)^n$$

$$ObjectColor_b = I_b = I_{a,b} K_{a,b} + I_{i,b} K_{diff,b}(N \cdot L) + I_{i,b} K_{spec,b}(R \cdot V)^n$$

Material properties:

$K_a, K_{diff}, K_{spec}, n$

Light properties

I_a, I_{diff}, I_{spec}

Questions

If you shine red light (1,0,0) to a white object what color does the object appear to have?

Questions

If you shine red light (1,0,0) to a white object what color does the object appear to have?

- Red: ~(1,0,0)*(1,1,1) = ~(1,0,0)
- May not be exactly (1,0,0) but it would be a shade of red

Questions

What if you shine red light (1,0,0) to a green object (0,1,0) ?

Questions

What if you shine red light (1,0,0) to a green object (0,1,0) ?

- Object will look black

Questions

What is the color of the highlight?

Questions

What is the color of the highlight?

- For non-metallic materials it is the color of the light
- For certain metallic materials it is the color of the material

Special cases

$$ObjectColor_r = I_r = I_{a,r} K_{a,r} + I_{i,r} K_{diff,r} (N \cdot L) + I_{i,r} K_{spec,r} (R \cdot V)^n$$

$$ObjectColor_g = I_g = I_{a,g} K_{a,g} + I_{i,g} K_{diff,g} (N \cdot L) + I_{i,g} K_{spec,g} (R \cdot V)^n$$

$$ObjectColor_b = I_b = I_{a,b} K_{a,b} + I_{i,b} K_{diff,b} (N \cdot L) + I_{i,b} K_{spec,b} (R \cdot V)^n$$

- What should be done if $I_{r,b,g} > 1$?

Special cases

$$ObjectColor_r = I_r = I_{a\_r}K_{a\_r} + I_{i\_r}K_{diff\_r}(N \cdot L) + I_{i\_r}K_{spec\_r}(R \cdot V)^n$$

$$ObjectColor_g = I_g = I_{a\_g}K_{a\_g} + I_{i\_g}K_{diff\_g}(N \cdot L) + I_{i\_g}K_{spec\_g}(R \cdot V)^n$$

$$ObjectColor_b = I_b = I_{a\_b}K_{a\_b} + I_{i\_b}K_{diff\_b}(N \cdot L) + I_{i\_b}K_{spec\_b}(R \cdot V)^n$$

- What should be done if $I_{r,b,g} > 1$?
- This is an important issue that falls under the general notion of **tone mapping**.
-

  - Clamp the value of I to one. Problem? (10,1,1) -> (1,1,1)
  - Scale so that maximum becomes 1? (10,1,1) --> (1,0.1,0.1)
  - Scale non-linearly?

Special cases

$ObjectColor_r = I_r = I_{a\_r}K_{a\_r} + I_{i\_r}K_{diff\_r}(N \cdot L) + I_{i\_r}K_{spec\_r}(R \cdot V)^n$

$ObjectColor_g = I_g = I_{a\_g}K_{a\_g} + I_{i\_g}K_{diff\_g}(N \cdot L) + I_{i\_g}K_{spec\_g}(R \cdot V)^n$

$ObjectColor_b = I_b = I_{a\_b}K_{a\_b} + I_{i\_b}K_{diff\_b}(N \cdot L) + I_{i\_b}K_{spec\_b}(R \cdot V)^n$

- What should be done if N*L < 0?
  Clamp the value of I to zero or flip the normal.

Special cases

$$ObjectColor_r = I_r = I_{a\_r}K_{a\_r} + I_{i\_r}K_{diff\_r}(N \cdot L) + I_{i\_r}K_{spec\_r}(R \cdot V)^n$$

$$ObjectColor_g = I_g = I_{a\_g}K_{a\_g} + I_{i\_g}K_{diff\_g}(N \cdot L) + I_{i\_g}K_{spec\_g}(R \cdot V)^n$$

$$ObjectColor_b = I_b = I_{a\_b}K_{a\_b} + I_{i\_b}K_{diff\_b}(N \cdot L) + I_{i\_b}K_{spec\_b}(R \cdot V)^n$$

- How can we handle multiple light sources?
  Sum the intensity of the individual contributions.

Shading Polygons: Flat Shading

Illumination equations are evaluated at surface locations

- so where do we apply them?

We could just do it once per polygon

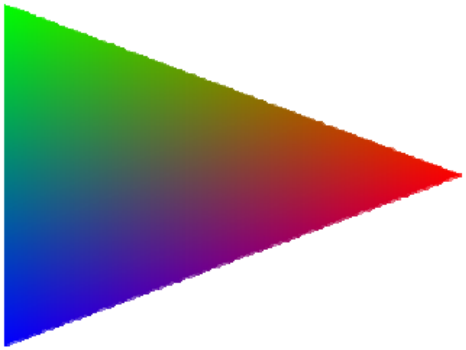
- fill every pixel covered by polygon with the resulting color

Apply the ADS model in the vertex shader
Tell the rasterizer not to interpolate per pixel (-- keyword “flat” find out the details on your own)

Shading Polygons: Gouraud Shading

Alternatively, we could evaluate at every vertex

- compute color for each covered pixel
- linearly interpolate colors over polygon



Misses details that don't fall on vertex

- specular highlights, for instance

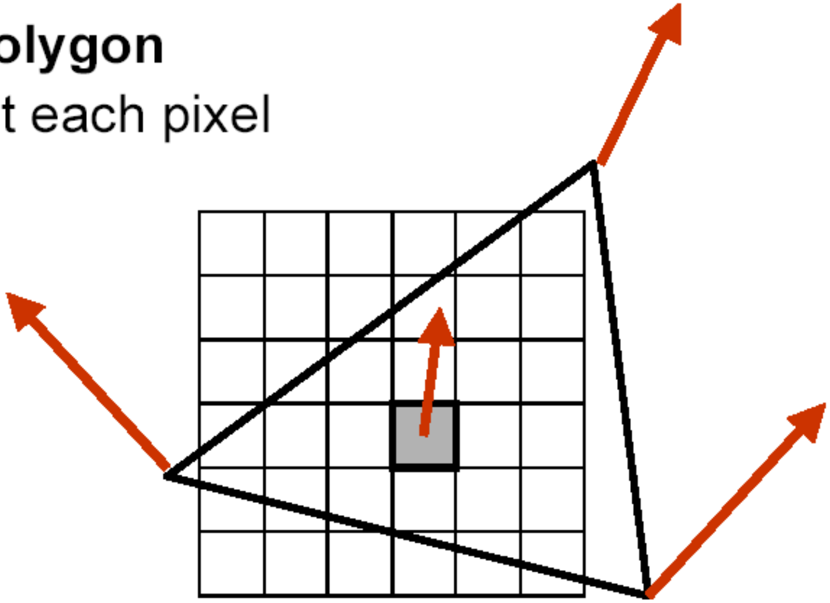
Apply the ADS lighting model in the vertex shader
Default interpolation

Shading Polygons: Phong Shading

Don't just interpolate colors over polygons

Interpolate surface normal over polygon

- evaluate illumination equation at each pixel



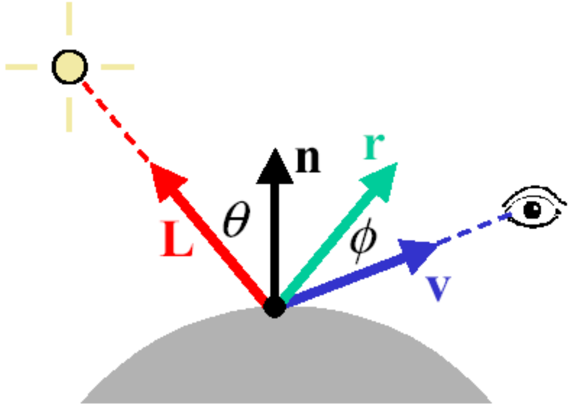
Apply ADS, in the fragment shader with the interpolated normal per pixel

Summarizing the Shading Model

We describe local appearance with illumination equations

- consists of a sum of set of components — light is additive
- treat each wavelength independently
- currently: diffuse, specular, and ambient terms

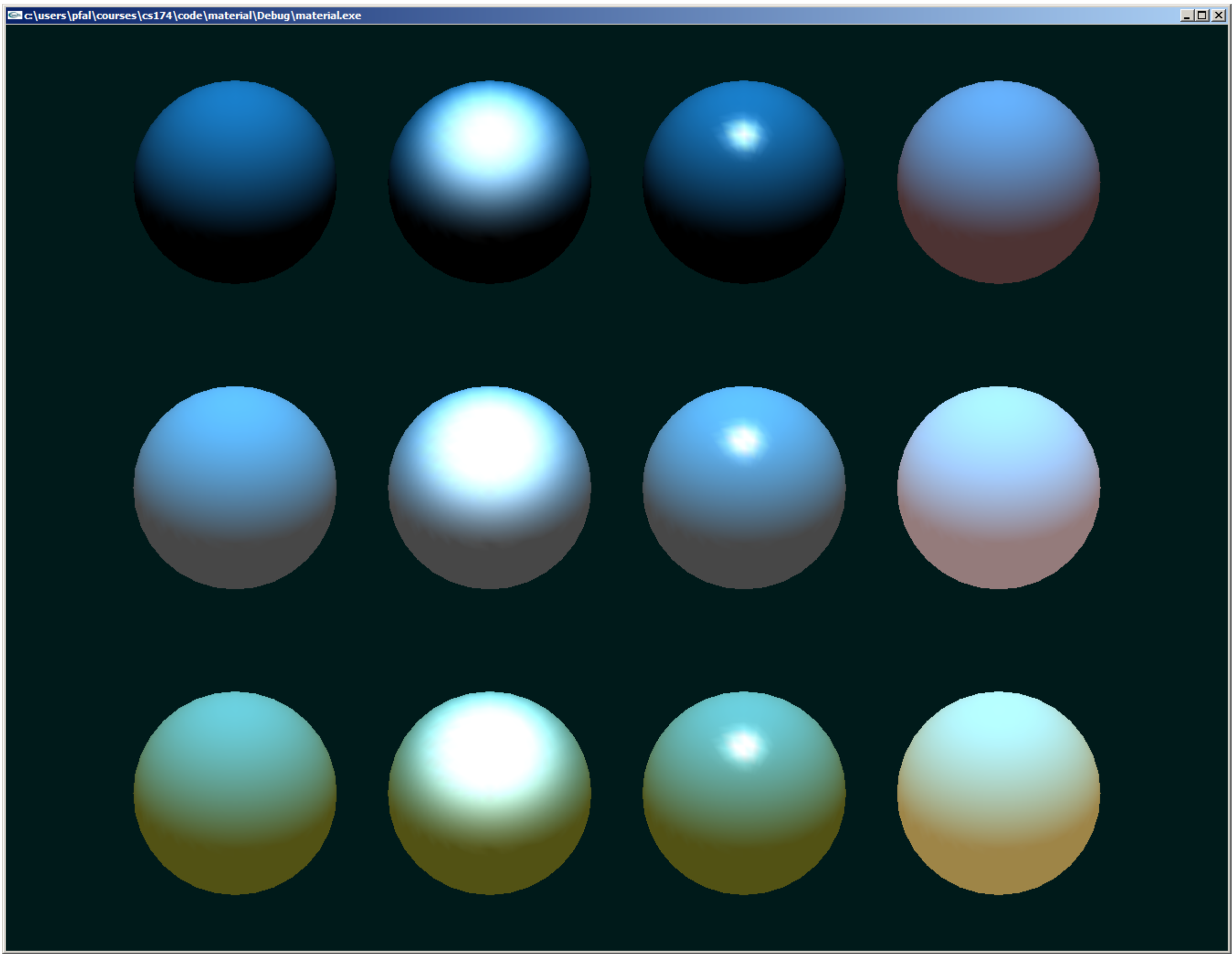
$$I = I_L k_d \cos\theta + I_L k_s \cos^n \phi + I_a k_a$$



Must shade every pixel covered by polygon

- flat shading: constant color
- Gouraud shading: interpolate corner colors
- Phong shading: interpolate corner normals

Examples of Phong Illuminated materials



IMPORTANT: Which coordinate system?

In which system do we normally do the lighting calculations

- Viewing coordinate system
- Why?

Shader based ADS Lighting

Per vertex

Per pixel

Per vertex ADS lighting

Vertex Shader applies the Phong illumination model per vertex

Fragment shader receives the interpolated colour from the rasterizer

Per vertex ADS lighting

Vertex Shader

```
// Parameters
in vec4 vPosition;
in vec3 vNormal;
in vec4 vColor ;
in vec2 vTexCoord ;

uniform vec4 ambientProduct, diffuseProduct, specularProduct;
uniform vec4 lightPosition;
uniform float shininess;

out vec4 fColor;
```

Per vertex ADS lighting

Vertex Shader

```
uniform vec4 ambientProduct, diffuseProduct, specularProduct;
uniform vec4 lightPosition;
uniform float shininess;

void
main()
{
    // Transform vertex position into eye coordinates
    vec3 pos = (modelViewMatrix * vPosition).xyz;

    // Transform vertex normal into eye coordinates
    vec3 N = normalize( (normalMatrix*vec4(vNormal,0.0)).xyz);

    // Outputs
    fColor = ads(pos, lightPosition.xyz, N);  // Anything interesting
                                              // about light's position?
    gl_Position = projectionMatrix * modelViewMatrix*vPosition ;
}
```

Per vertex ADS lighting

Vertex Shader

```
vec4 ads(vec3 pos, vec3 lpos, vec3 N) {
    vec3 L = normalize(lpos - pos) ;
    vec3 V = normalize(-pos) ;  // why?
    vec3 R = reflect(-L, N) ;

    // Compute terms in the illumination equation
    vec4 ambient = ambientProduct;
    float costheta = max( dot(L, N), 0.0 );

    vec4 diffuse = vec4(0.0, 0.0, 0.0, 1.0);
    vec4 specular = vec4(0.0, 0.0, 0.0, 1.0);
    diffuse = costheta*diffuseProduct;
    float cosphi = pow( max(dot(R, V), 0.0), shininess );
    specular = cosphi * specularProduct;

    if( dot(L, N) < 0.0 ) {
        specular = vec4(0.0, 0.0, 0.0, 1.0);
    }
    vec4 color = ambient + diffuse + specular;
    color.a = 1.0 ;   // WHY??
    return color ;
}
```

Per vertex ADS lighting

Fragment Shader

```
in vec4 fColor;
layout (location=0) out vec4 fragColor ;

void
main()
{
    fragColor = fColor;
}
```

Homework

Modify what is needed so that the light is stationary in the word coordinate system

Per fragment ADS lighting

Vertex shader outputs the necessary information to the fragment shader

Fragment Shader receives the interpolated information and applies the Phong illumination model per fragment

What is this "necessary" information?

Per fragment ADS lighting

Vertex shader outputs the necessary information to the fragment shader

Fragment Shader receives the interpolated information and applies the Phong illumination model per fragment

What is this "necessary" information?

- Remember the fragment shader does not have direct access to the original vertex attributes

Per fragment ADS lighting

Vertex Shader

```
in vec4 vPosition;
in vec3 vNormal;
in vec4 vColor ;


uniform mat4 modelViewMatrix;
uniform mat4 normalMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;

out vec3 fPos ; // vertex position in eye coords
out vec3 fLpos ; // light position in eye coords
out vec3 fN ;    // vertex normal in eye coords

void  main() {
```

Per fragment ADS lighting

Vertex Shader

```
out vec3 fPos ;  // vertex position in eye coords
out vec3 fLpos ; // light position in eye coords
out vec3 fN ;    // vertex normal in eye coords
```

```
void  main() {
   // Transform vertex position into eye coordinates
    fPos = (modelViewMatrix * vPosition).xyz;

    //transform normal in eye coordinates
    fN = normalize( (normalMatrix*vec4(vNormal,0.0)).xyz);
    // pass through light position
    fLpos = lightPosition.xyz ;

    // Transform vertex position in clip coordinates
    gl_Position = projectionMatrix * modelViewMatrix * vPosition;
}
```

Per fragment ADS lighting

Fragment Shader

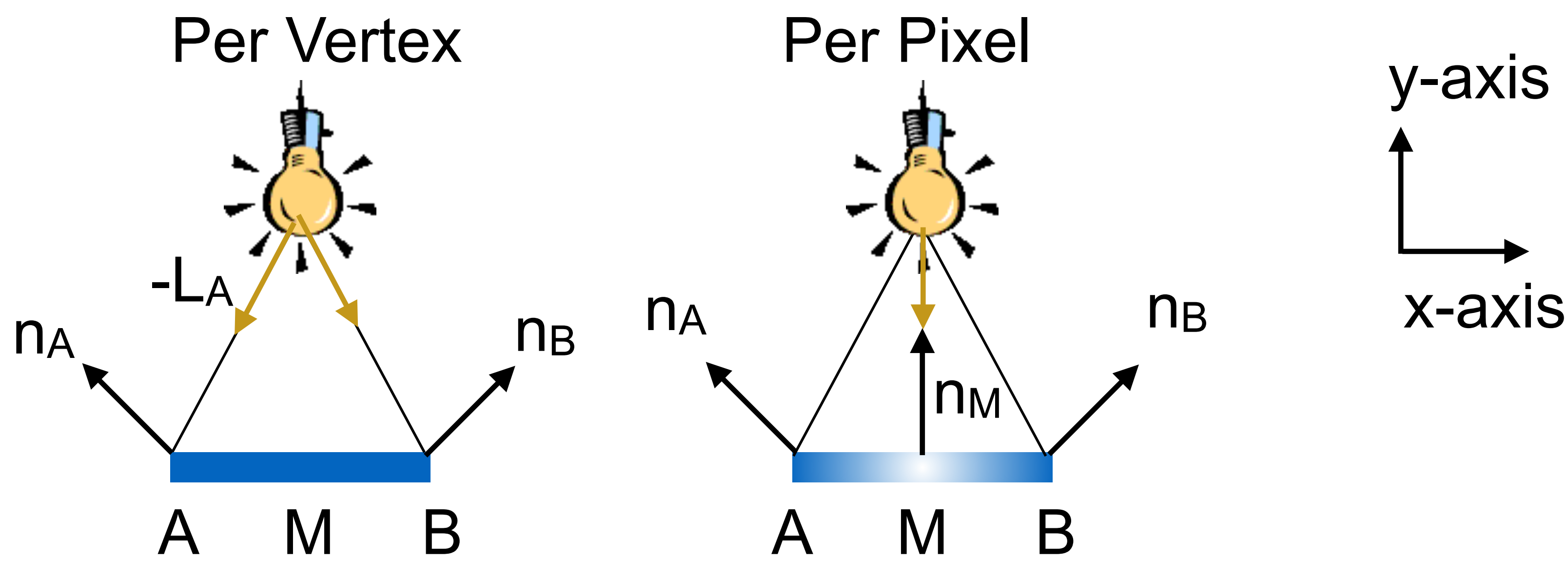
```
precision mediump float;

uniform vec4 ambientProduct, diffuseProduct, specularProduct;
uniform float shininess;

in vec3 fPos ;
in vec3 fLpos ;
in vec3 fN ;
in vec2 fTexCoord ;
layout (location = 0) out vec4 fragColor ;

 void  main() {
    fragColor = ads(fPos,fLpos,fN) ;
 }
```

Per fragment ADS lighting

Fragment Shader

```
// EXACTLY the same as in the case of per vertex ADS

vec4 ads(vec3 pos, vec3 lpos, vec3 N) {
    vec3 L = normalize(lpos - pos) ;
    vec3 V = normalize(-pos) ;  // why?
    vec3 R = reflect(-L, N) ;

    // Compute terms in the illumination equation
    vec4 ambient = ambientProduct;
    float costheta = max( dot(L, N), 0.0 );

    vec4 diffuse = vec4(0.0, 0.0, 0.0, 1.0);
    vec4 specular = vec4(0.0, 0.0, 0.0, 1.0);
    diffuse = costheta*diffuseProduct;
    float cosPhi = pow( max(dot(R, V), 0.0), shininess );
    specular = cosPhi * specularProduct;

    if( dot(L, N) < 0.0 ) {
        specular = vec4(0.0, 0.0, 0.0, 1.0);
    }
    vec4 color = ambient + diffuse + specular;
    color.a = 1.0 ;
    return color ;
}
```

PerVertex vs PerFragment



$n_A = (-0.707, 0.707, 0)$
$n_B = (0.707, 0.707, 0)$

$n_M = 0.5\, n_A + 0.5 n_B = (0,1,0)$

$c(A) = IL(\cos(n_A, L_A))$
$c(M) = 0.5c(A) + 0.5c(B)$

$c(M) = Illum(\cos(n_M, \text{y-axis}))$

Homework

What do we need to do to support multiple lights?

Two-sided lighting?

- How do you illuminate a back facing triangle?
  - *gl_FrontFacing in Fragement shader for per fragment lighting*
  - *How do we do it for per vertex lighting?*

What Have We Ignored?

Some local phenomena

- shadows — every point is illuminated by every light source
- attenuation — intensity falls off with square of distance to light
- transparent objects — light can be transmitted through surface

Global illumination

- reflections of objects in other objects
- indirect diffuse light — ambient term is just a hack

Realistic surface detail

- can make an orange sphere
- but it doesn't have the texture of the real fruit

Realistic light sources

Standard Example: The Cornell Box