ARLAB

ME/CprE/ComS 557

Computer Graphics and Geometric Modeling

Light Sources

September 29, 2015

Rafael Radkowski

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Content

- Attenuation
- Light Sources
  - Point Light
  - Spot Light
  - Direct Light
- Gamma Correction
- Code example

Light Source

What we know so far:

Reflectance model for diffuse, specular, and ambient reflection

Color = Specular + Diffuse + Ambient + Emissive

But: a light source was considered as point (position in space)

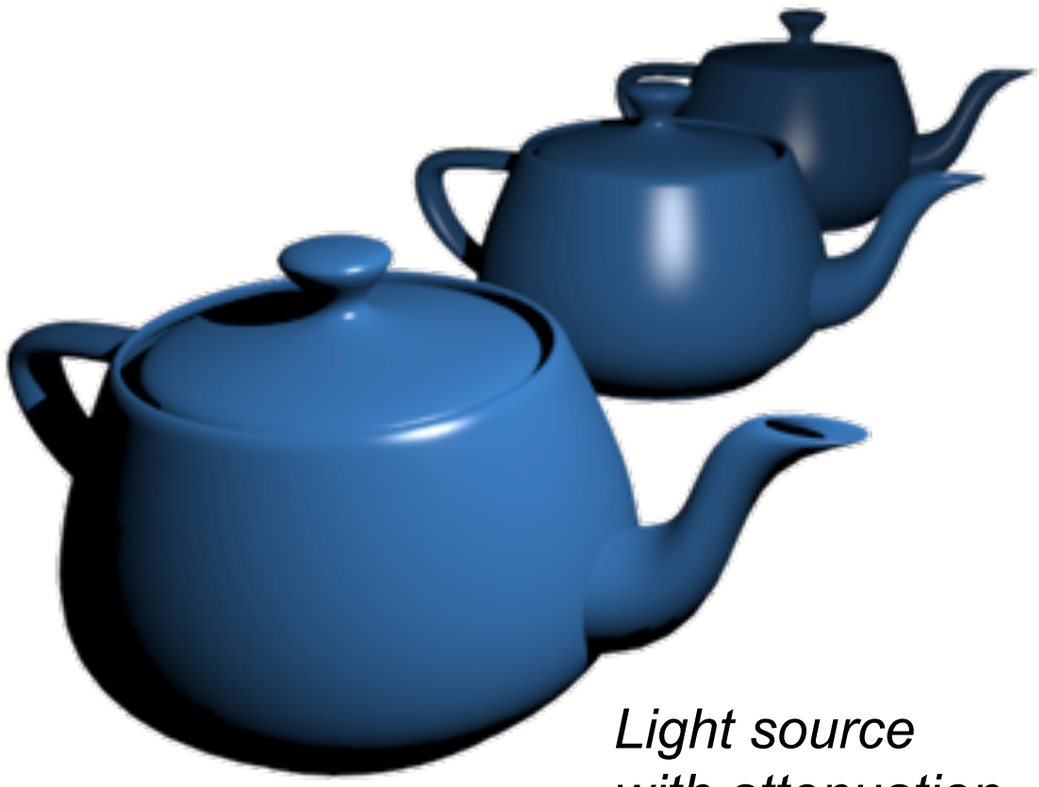
Diffuse light/ reflection

Diffuse + specular light / reflection

● Position of the light source

Attenuation

Attenuation is the loss of light intensity over distance.
The greater the distance, the lower the intensity.



Light source with attenuation

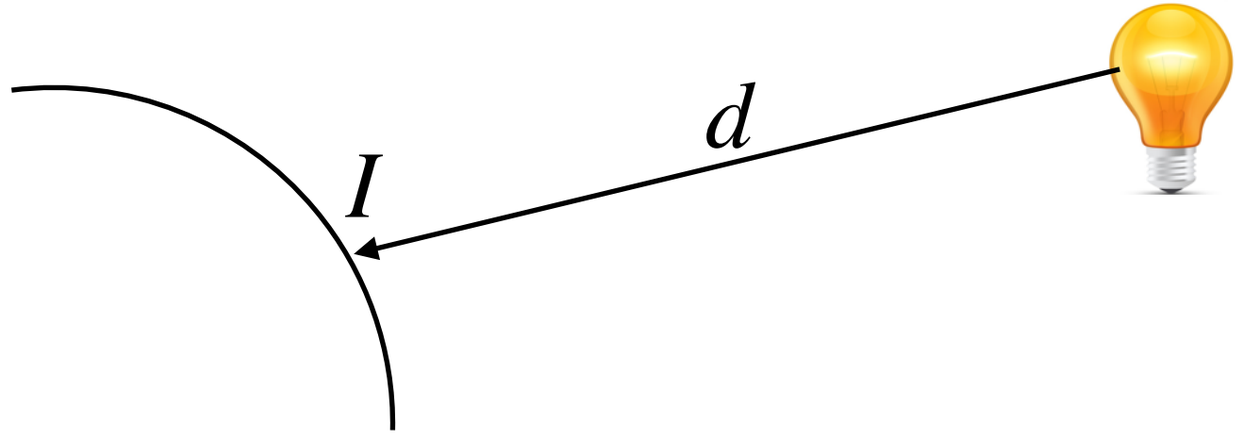
Light source without attenuation

● Position of the light source

Attenuation

Physical attenuation is proportional to the inverse of the distance squared:

$$I \propto \frac{1}{d^2}$$



I: intensity
d: distance light source to surface

In computer graphics, we use a modified equation to

- avoid division by 0
- have better control to balance the light

$$a = \frac{1}{1 + k \cdot d^2}$$

a: attenuation factor
d: distance light source to surface
k: attenuation factor

Code Example

This code is part of the shader program spotlight.vs

First we calculate the attenuation parameter.

```
//attenuation
float distanceToLight = length(light_position.xyz - surfacePostion.xyz);
float attenuation = 1.0 / (1.0 + attenuationCoefficient * pow(distanceToLight, 2));
```

That is what we have so far.

```
// Calculate the linear color
vec3 linearColor = out_ambient_color  + out_diffuse_color + out_specular_color);
```

Now we replace it with:

```
// Calculate the linear color
vec3 linearColor = out_ambient_color  + attenuation * ( out_diffuse_color +
out_specular_color);
```

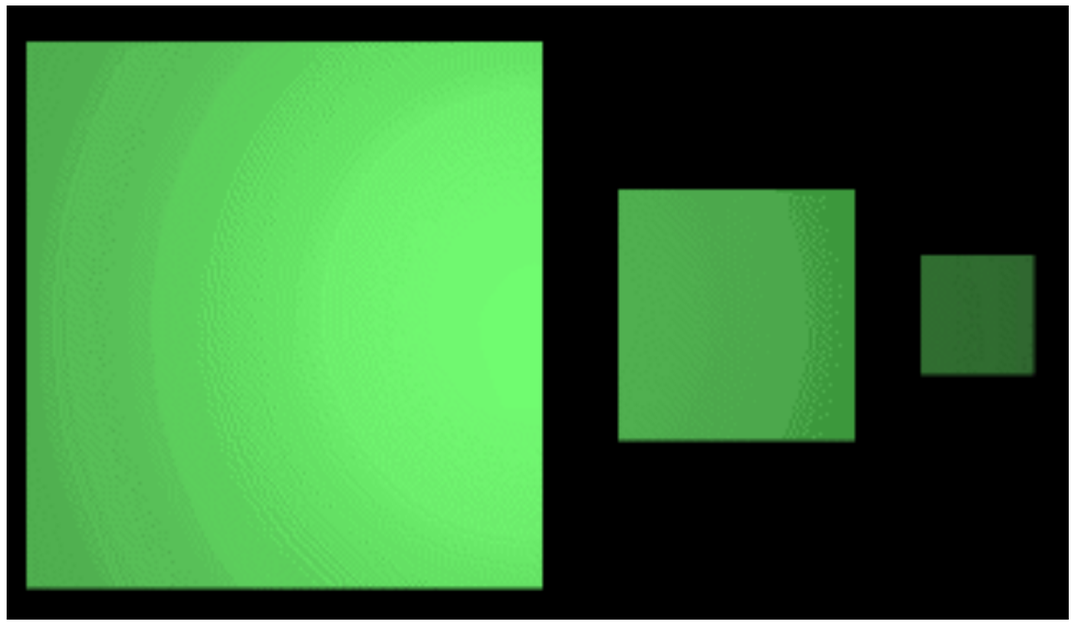
Sometimes....

ARLAB

It does not work very well because the quadratic parameter is difficult to balance in order to find the right effect.

But we can move to a linear parameter of even to a combination of linear and quadratic attenuation.

$$a = \frac{1}{1 + k_0 \cdot d + k_1 \cdot d^2}$$

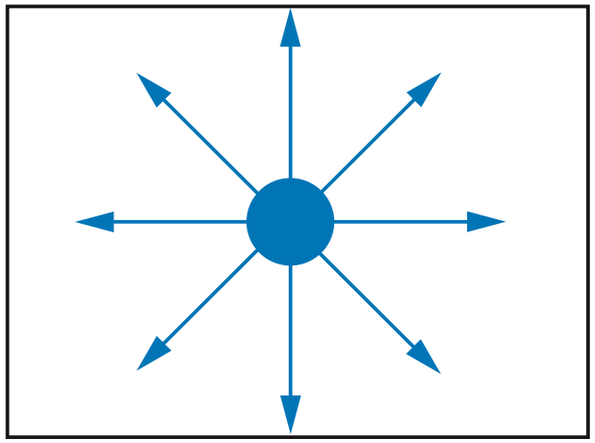


Linear attenuation is easier to balance

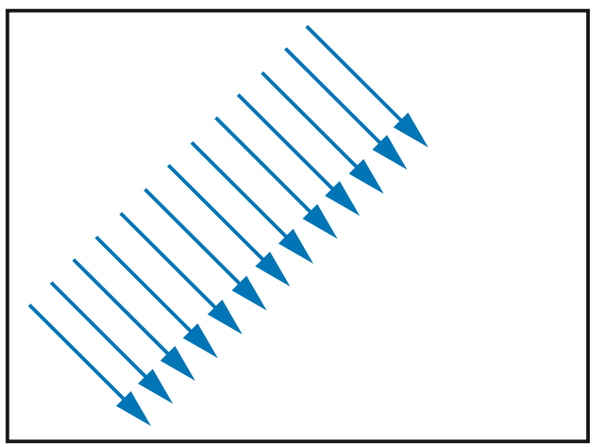
a: attenuation factor
d: distance light source to surface
k_0: linear attenuation factor
k_1: quadratic attenuation factor

Light Sources

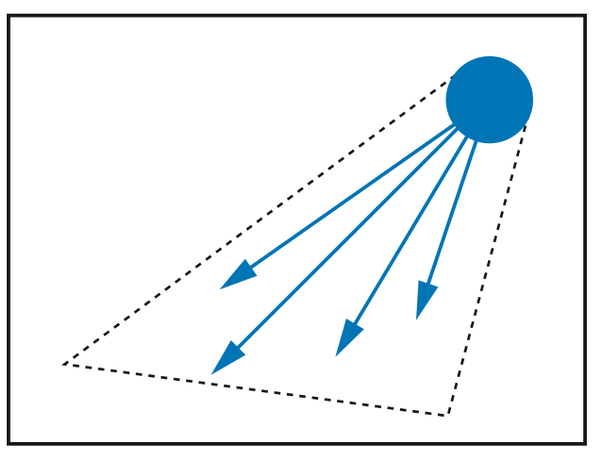
ARLAB



Point Light: a directed light source that emits light equally into all directions. The intensity of light decrease constantly with increasing distance between light source and surface. A candle light is an example.



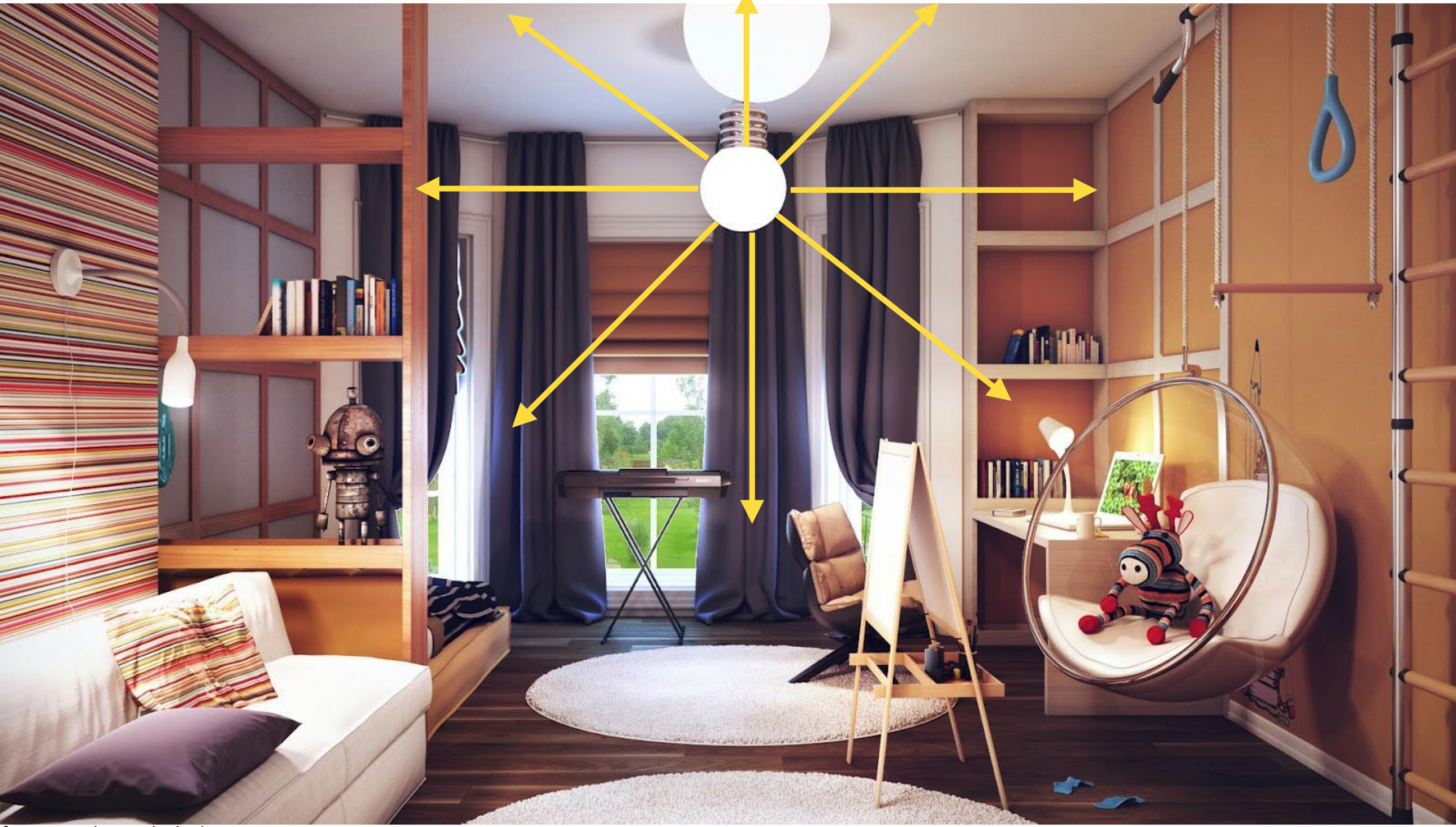
Directed Light: light that is emitted from a distant light source. Thus, all light rays run parallel. The distance between light source and surface has no effect on the intensity. The sun is an example.



Spotlight: is a point light which emits light only within a limited angle. The intensity of light decrease with increasing distance to the surface and to the edges of the spot. It can be used to simulate a flashlight.

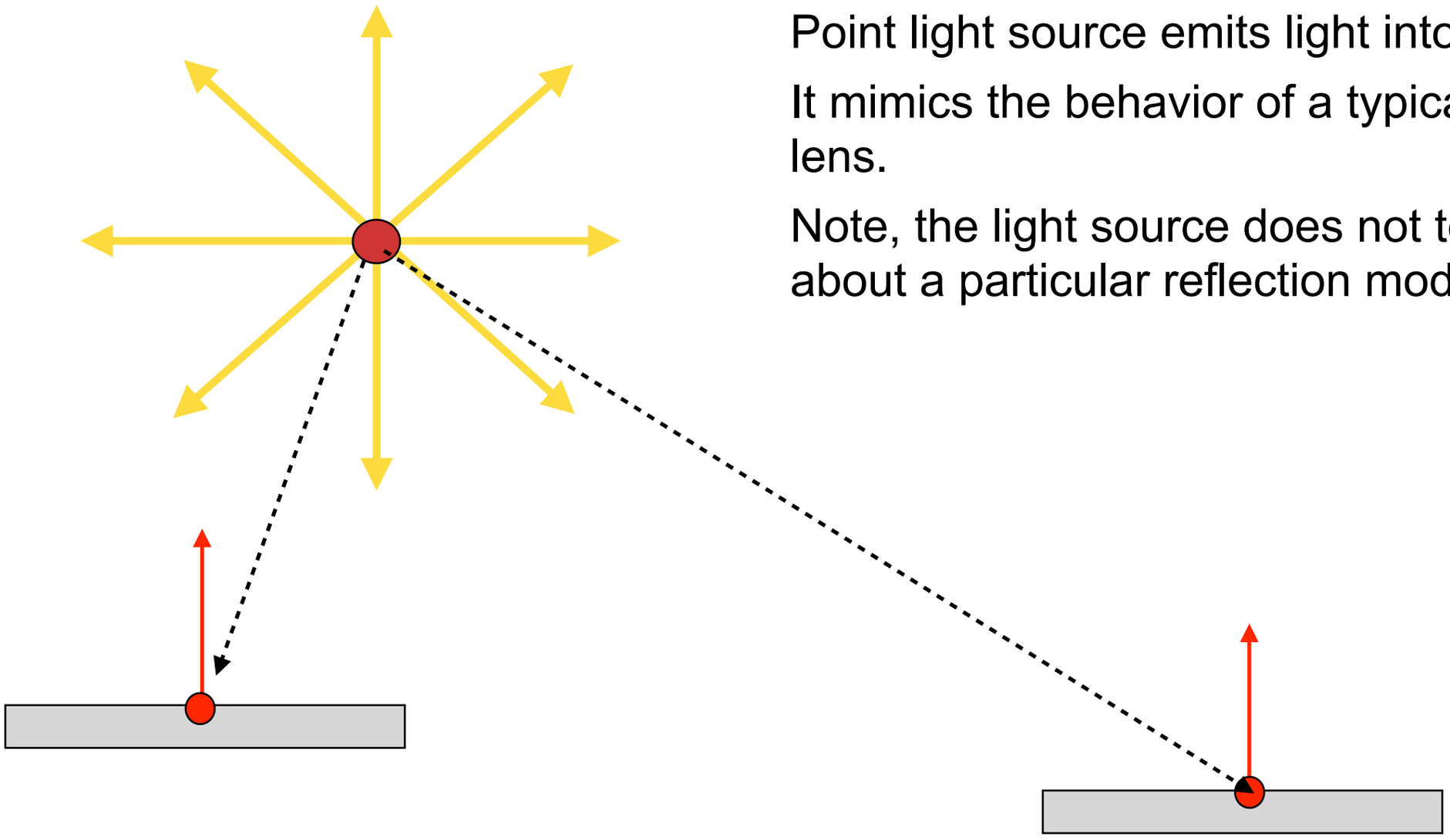
Point Light Source

ARLAB



from: www.home-designing.com

Point Light Source



Point light source emits light into all directions.

It mimics the behavior of a typical light without lens.

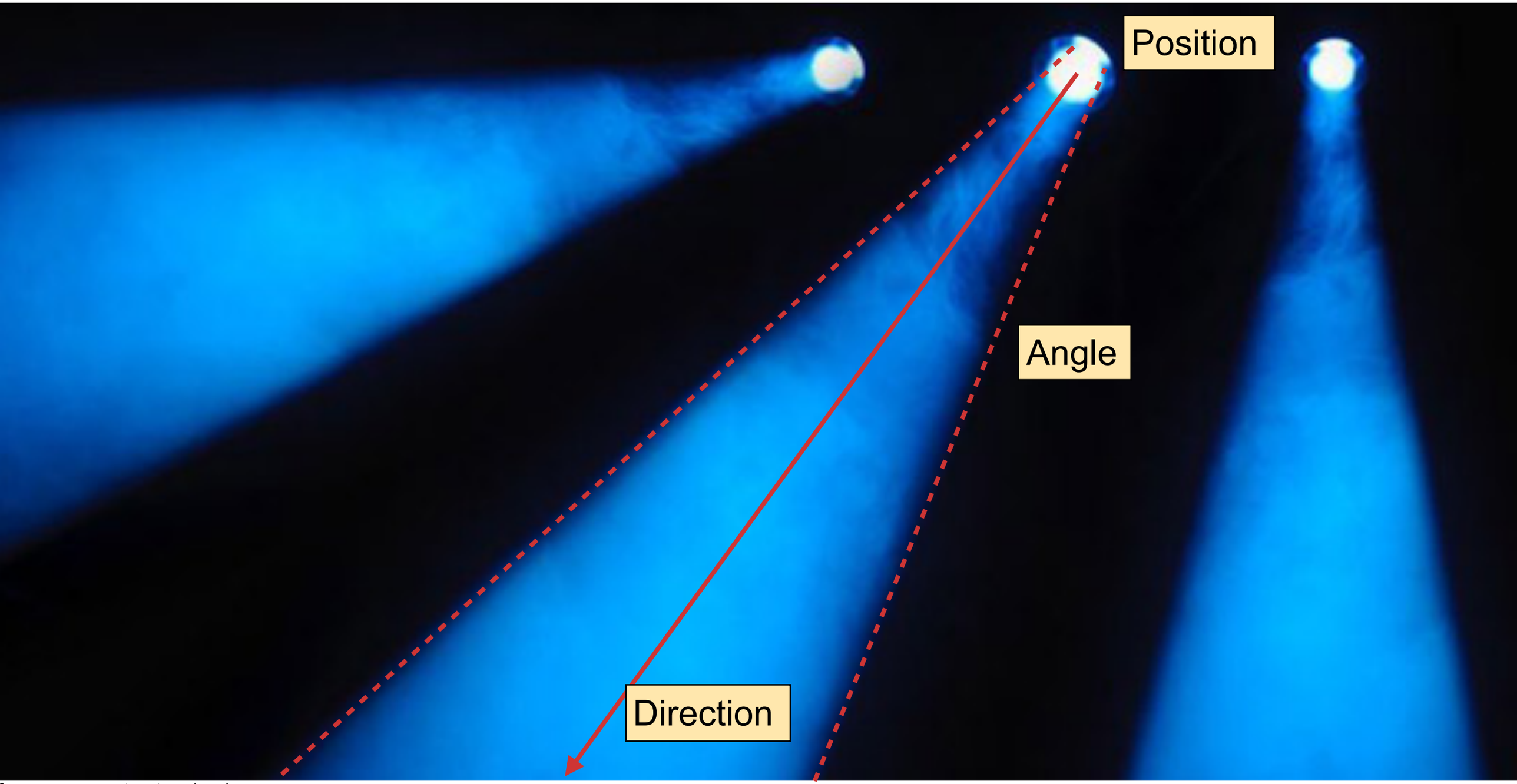
Note, the light source does not tell one anything about a particular reflection model.

Described by:

- position
- intensity
- color
- attenuation

Spot Light Source

ARLAB



from: www.centrestageleeds.com

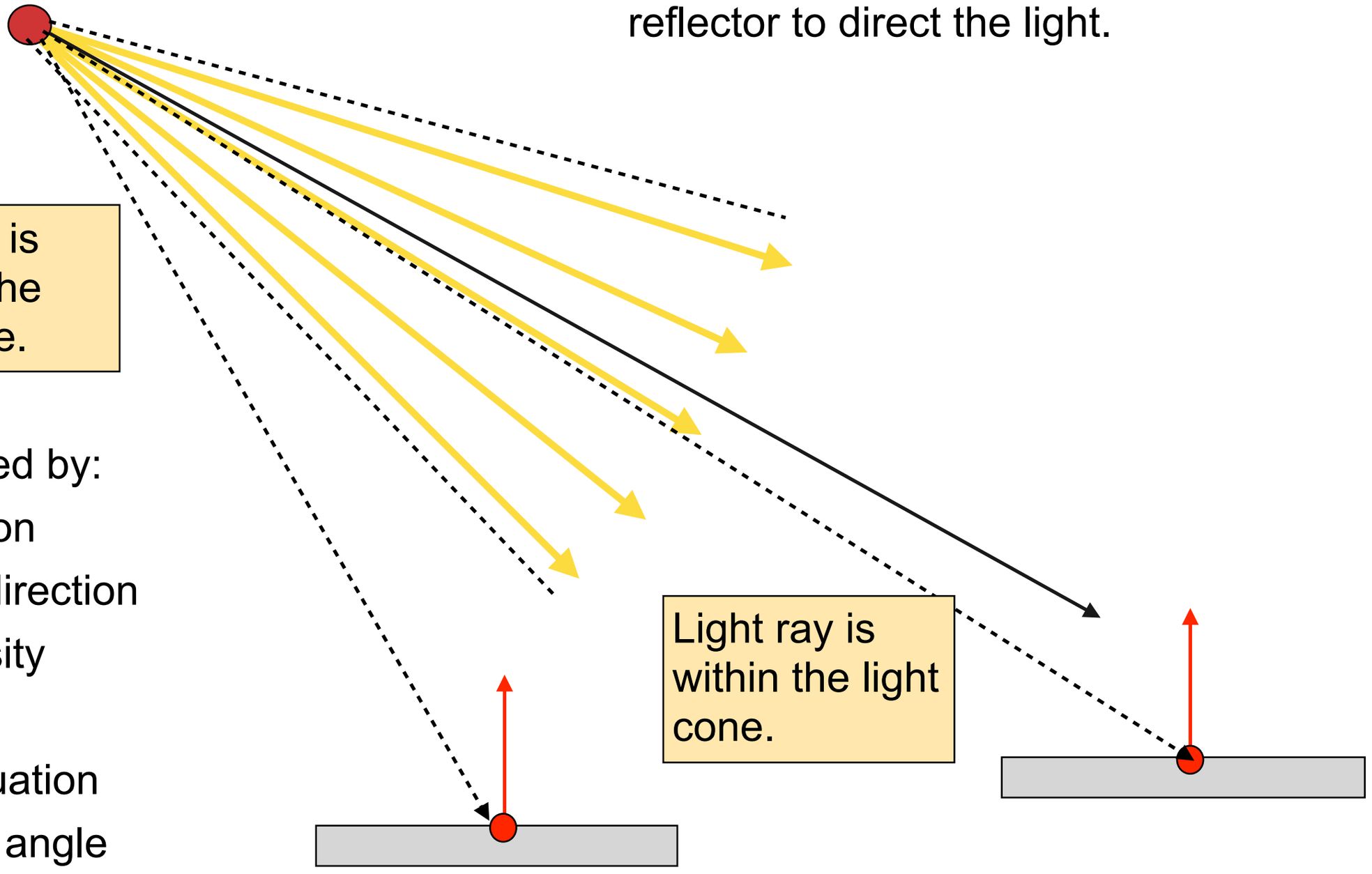
Spotlight

A spot light mimics light sources with a reflector to direct the light.

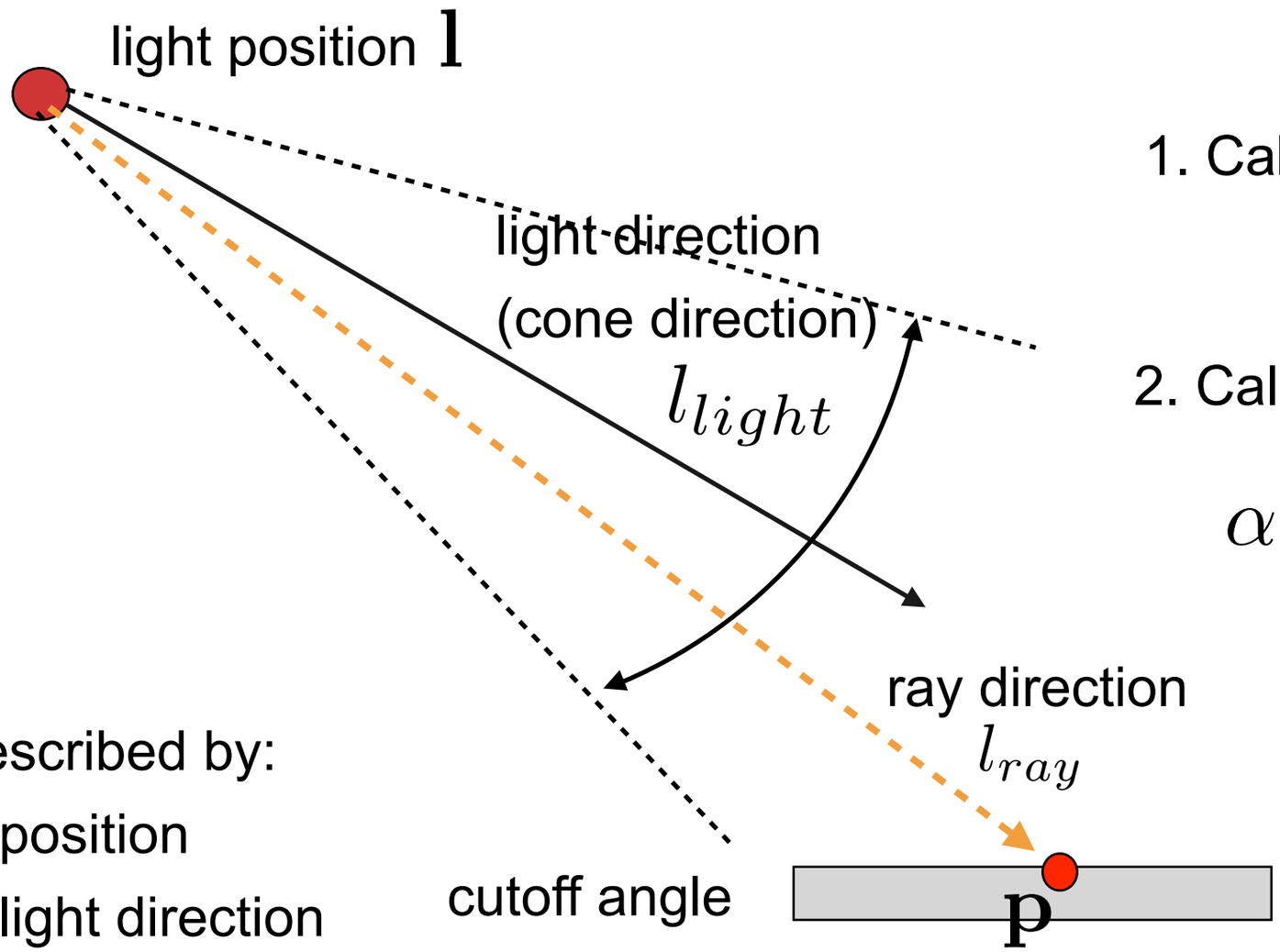
Light ray is outside the light cone.

Described by:

- position
- light direction
- intensity
- color
- attenuation
- cutoff angle



Spotlight Calculation



Described by:

- position
- light direction
- intensity
- color
- attenuation
- cutoff angle

1. Calculate the light ray direction:

$$\mathbf{l}_{ray} = \mathbf{p} - \mathbf{l}$$

2. Calculate the angle:

$$\alpha = \cos(\mathbf{l_{ray}} \cdot \mathbf{l_{light}})^{-1}$$

3. Check whether the angle is inside the cone:

$$if(\alpha > \text{cutoff angle})a = 0.0$$

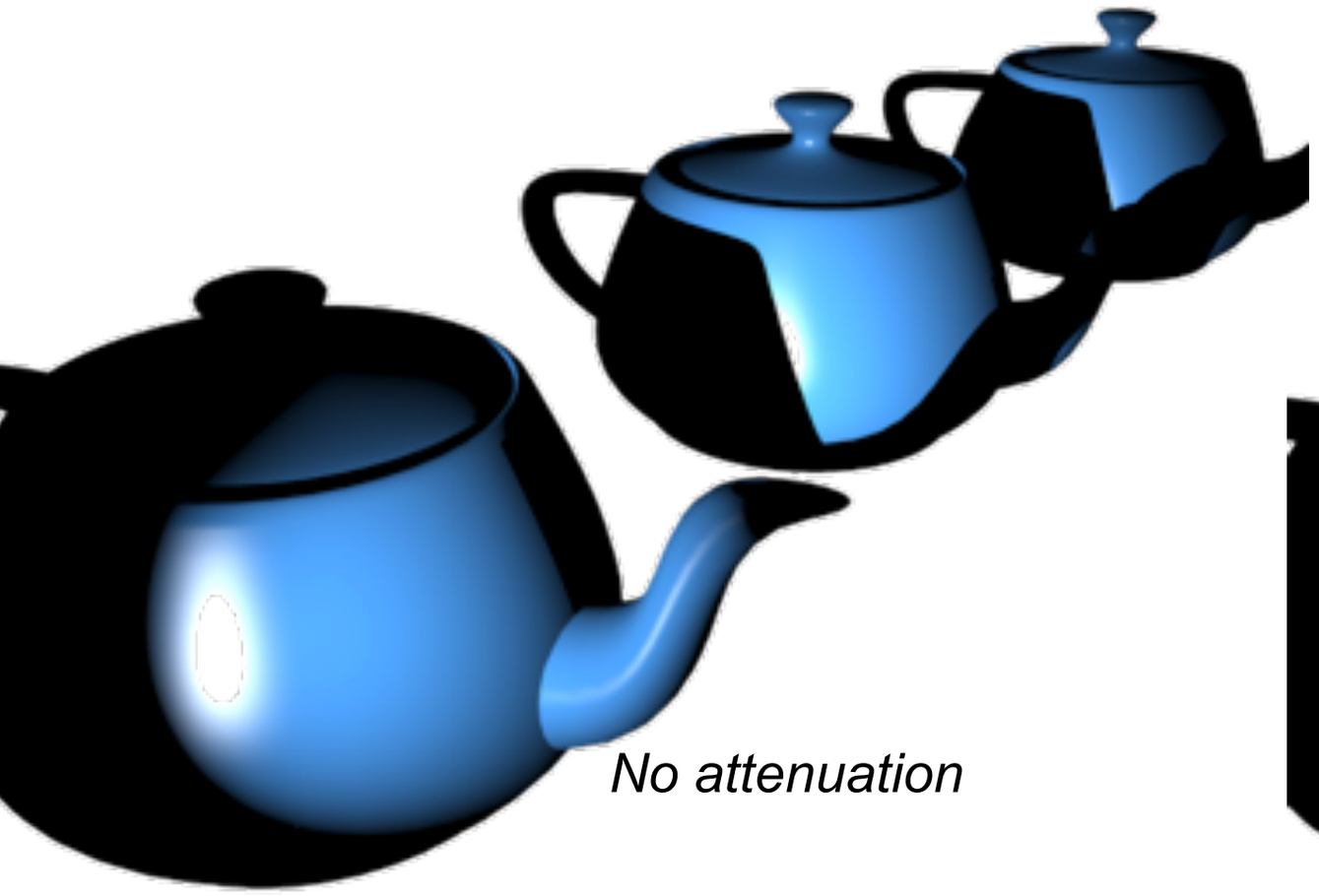
Code Example

ARLAB

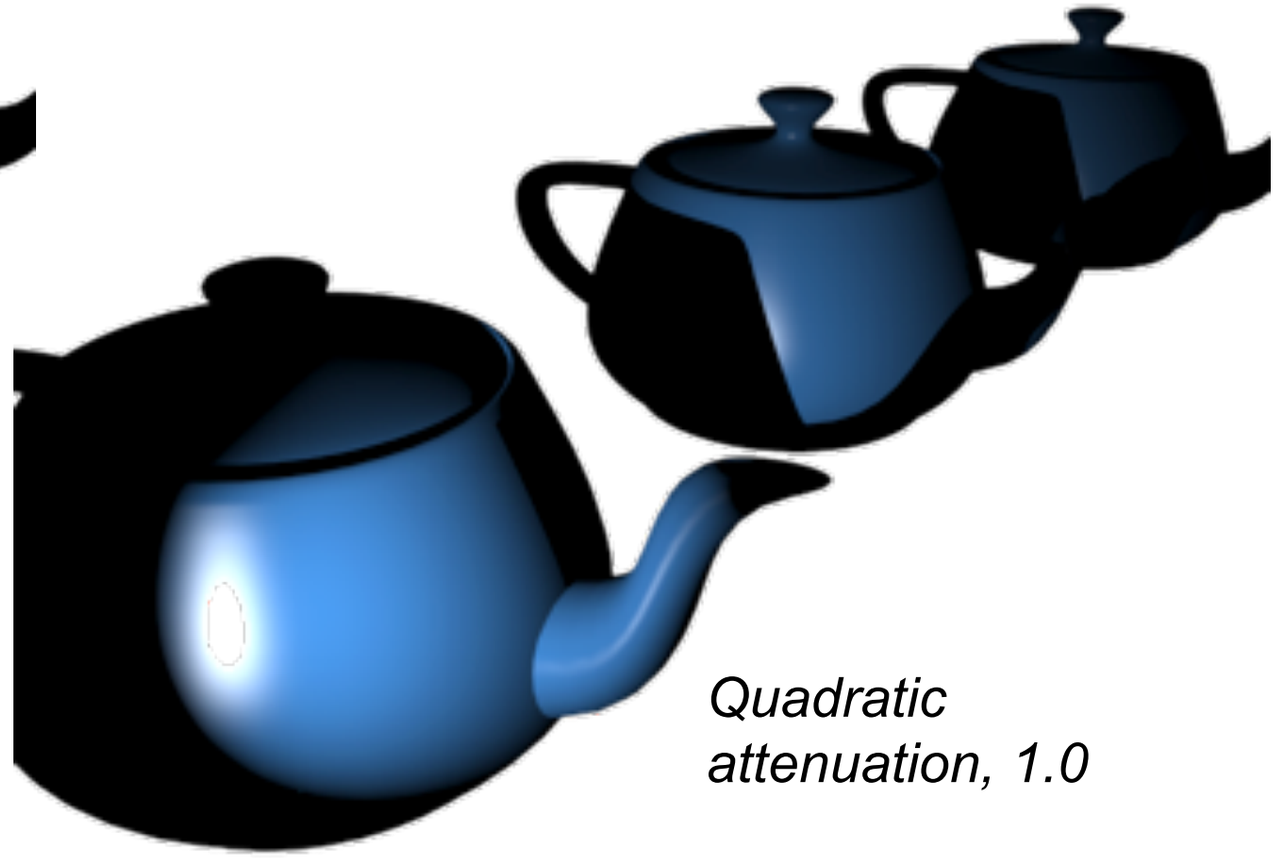
```
////////////////////////////////////////////////////////////////////////////////
//////////////
// Spotlight
// 1. Normalize the cone direction
vec3 cone_direction_norm = normalize(cone_direction);

// 2. Calculate the ray direction.
vec3 ray_direction = -surface_to_light.xyz;

// 3. Calculate the angle between light and surface using the dot product again.
//    To simplify our understanding, we use the degrees
float light_to_surface_angle =
          degrees(acos(dot(ray_direction, cone_direction_norm))) ;

// 4. Last, we compare the angle with the current direction and
//    reduce the attenuation to 0.0 if the light is outside the angle.
if(light_to_surface_angle > cone_angle){
    attenuation = 0.0;
}
```

Spot Light – Attenuation

ARLAB

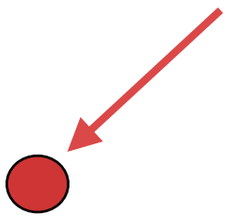


No attenuation



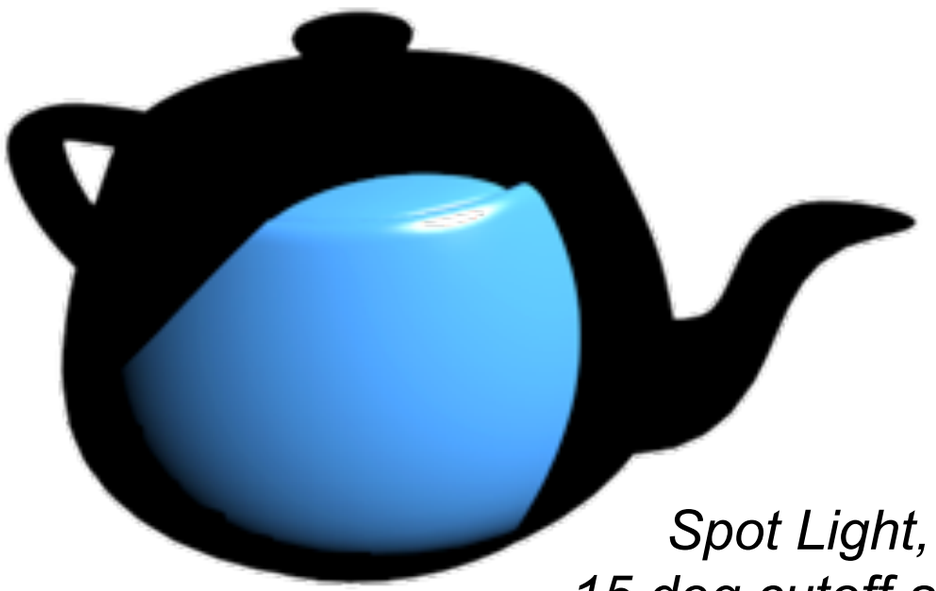
Quadratic attenuation, 1.0

Light comes from here



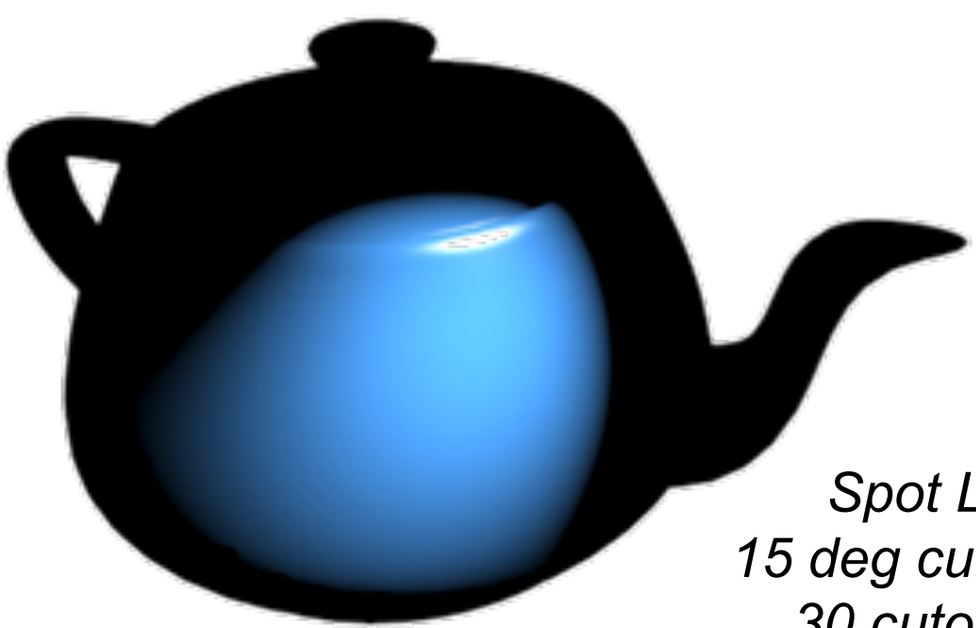
Smooth Spot Light

ARLAB



Spot Light,
15 deg cutoff angle

Sharp spotlight edge

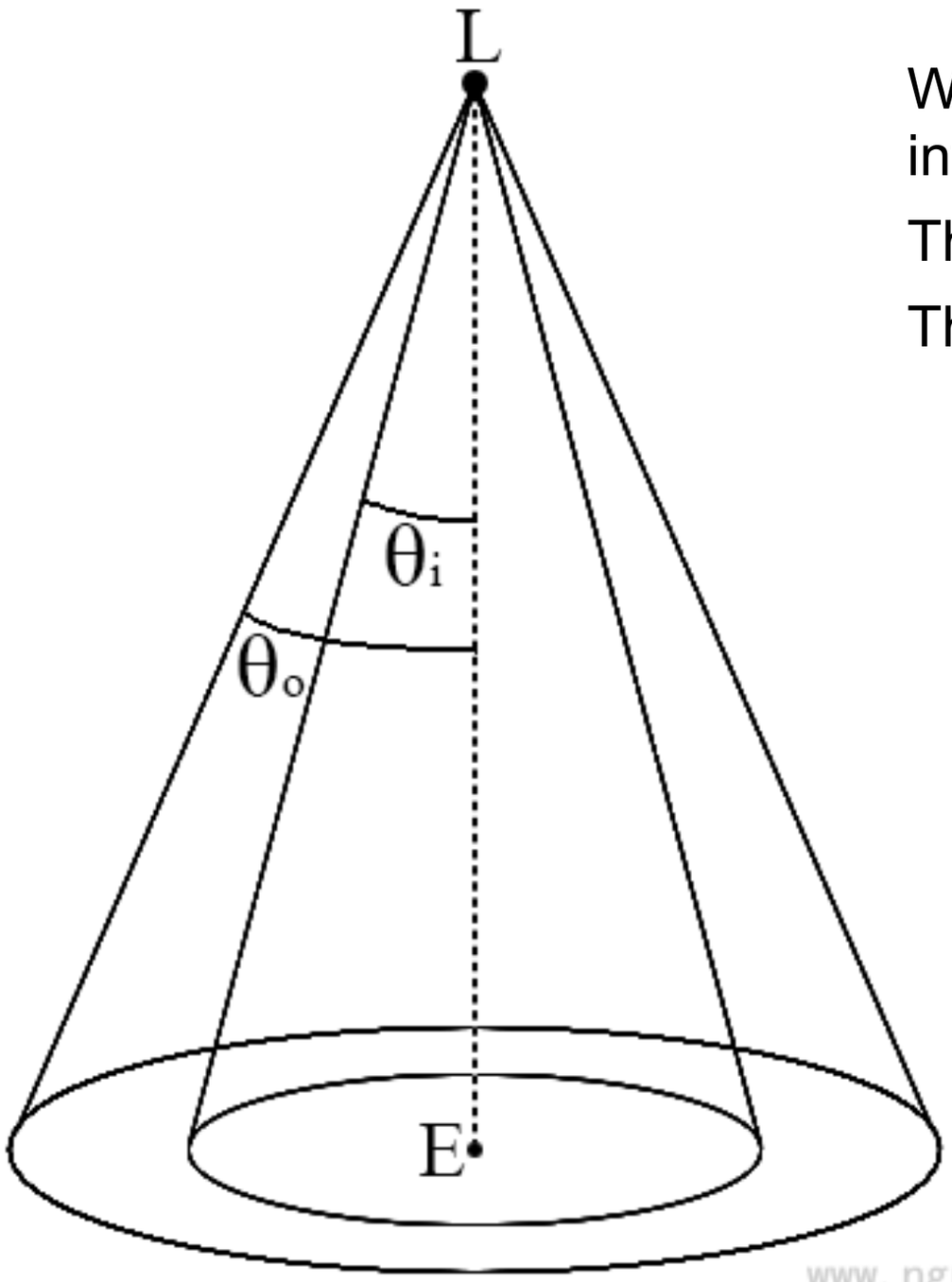


Spot Light,
15 deg cutoff angle
30 cutoff exp.

Smooth spotlight edge

● Position of the light source

Smooth Spotlight

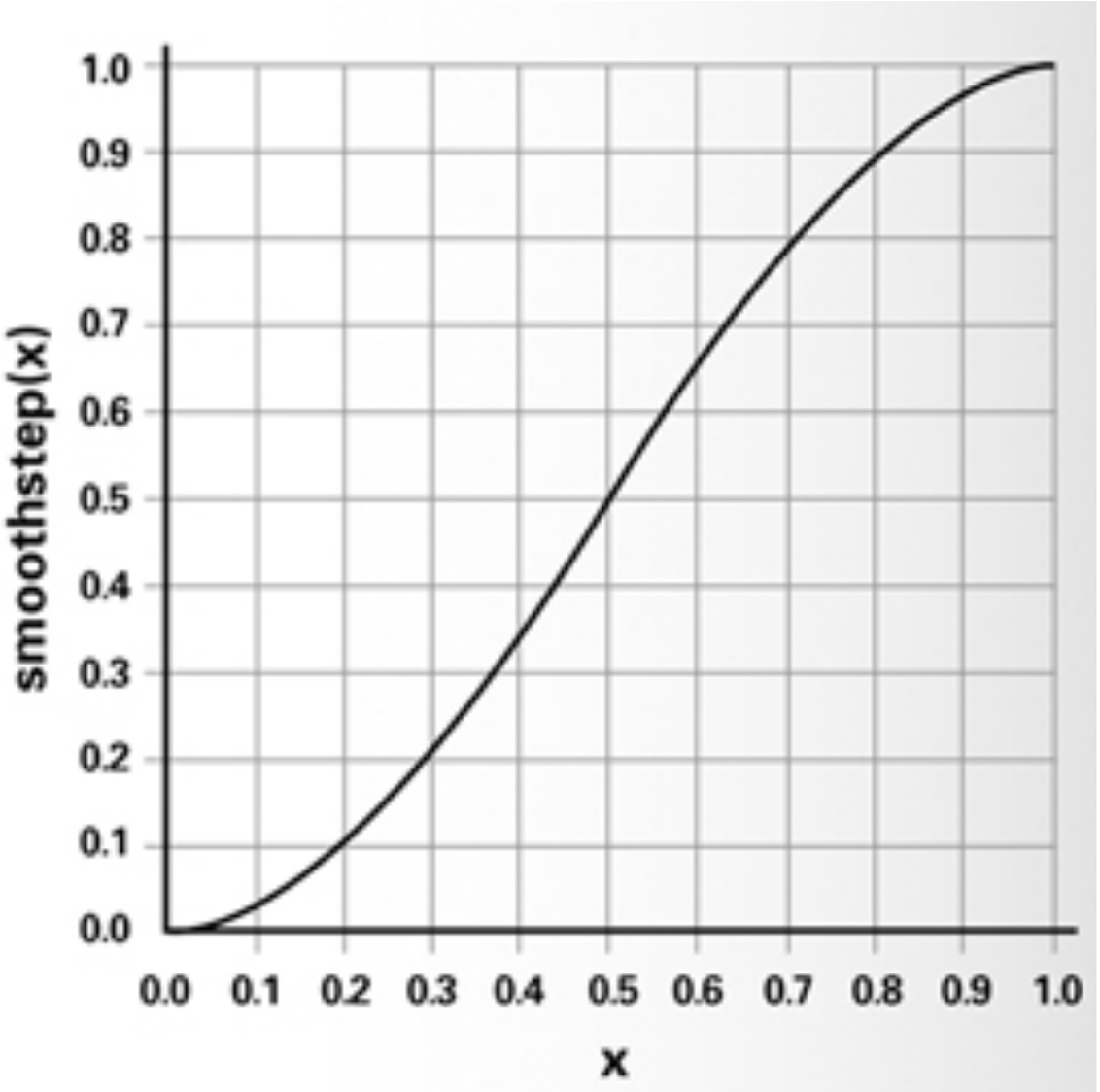


We work with two spotlight cones and interpolate between both of them.

The inner circle is the hard spotlight.

The outer circle is the soft spotlight.

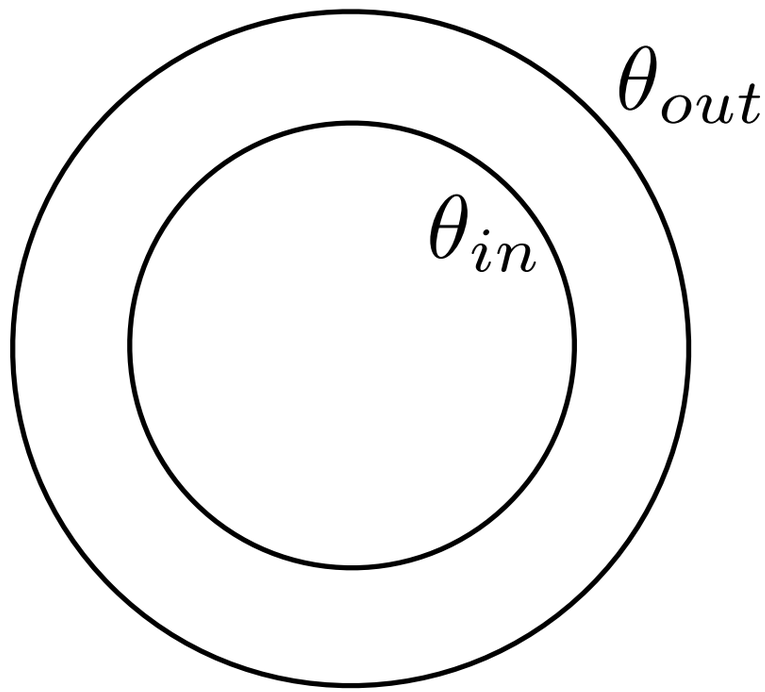
Smooth Spotlight



We interpolate between inner and outer circle using a Hermine interpolation

$$smoothstep(x) = 3x^2 - 2x^3$$

$$x = \theta_{out} - \theta_{in}$$



GLSL Function - smoothstep

ARLAB

Perform Hermite interpolation between two values

```
genType smoothstep( genType edge0, genType edge1, genType x);
```

Parameters:

- edge0 - Specifies the value of the lower edge of the Hermite function.
- edge1 - Specifies the value of the upper edge of the Hermite function.
- x - Specifies the source value for interpolation.

Code:

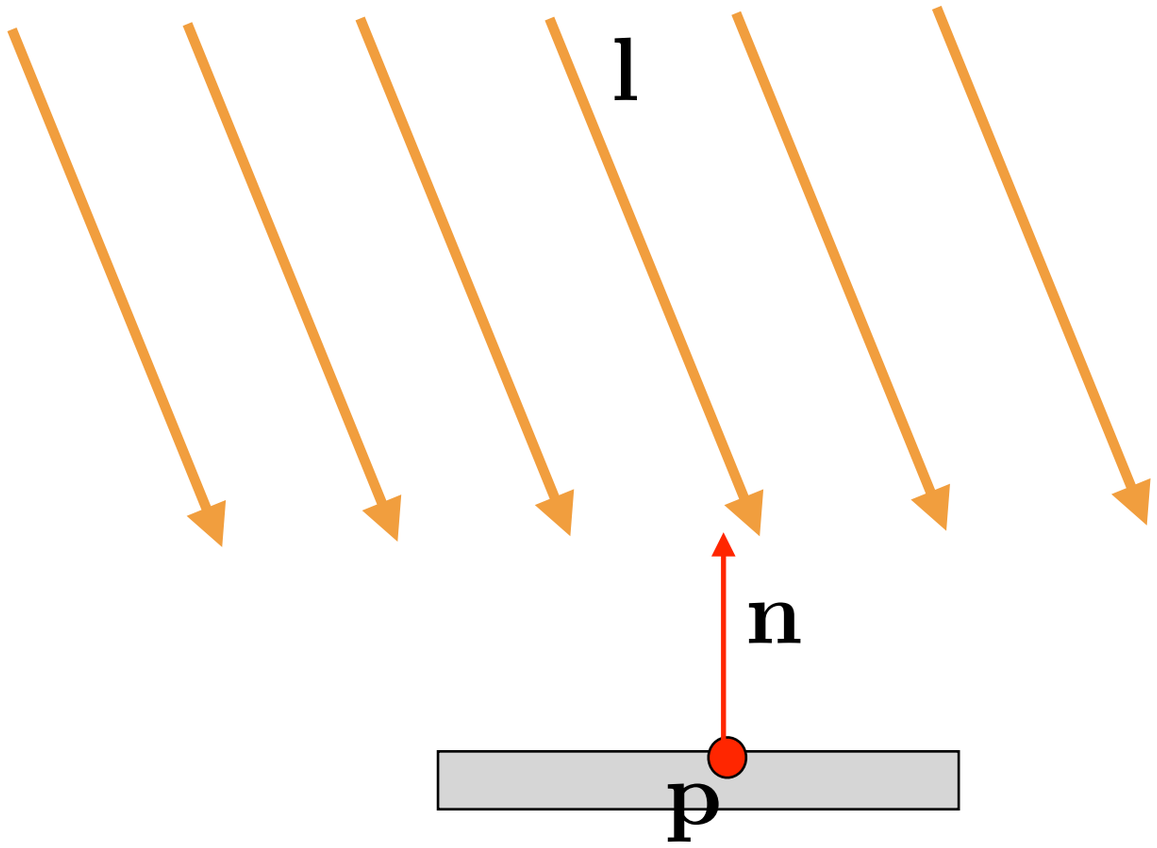
```
float smoothstep(float edge0, float edge1, float x)
{
    // Scale, bias and saturate x to 0..1 range
    x = clamp((x - edge0)/(edge1 - edge0), 0.0, 1.0);
    // Evaluate polynomial
    return x*x*(3 - 2*x);
}
```

Directional Light

ARLAB

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Directional Light



Described by:

- light direction
- intensity
- color

1. Calculate the angle between light and surface normal n.

$$\alpha = \mathbf{l} \cdot \mathbf{n}$$

2. Check whether the surface is directed towards the light.

$$if(\alpha < 0.0)\text{draw}$$

Code Example

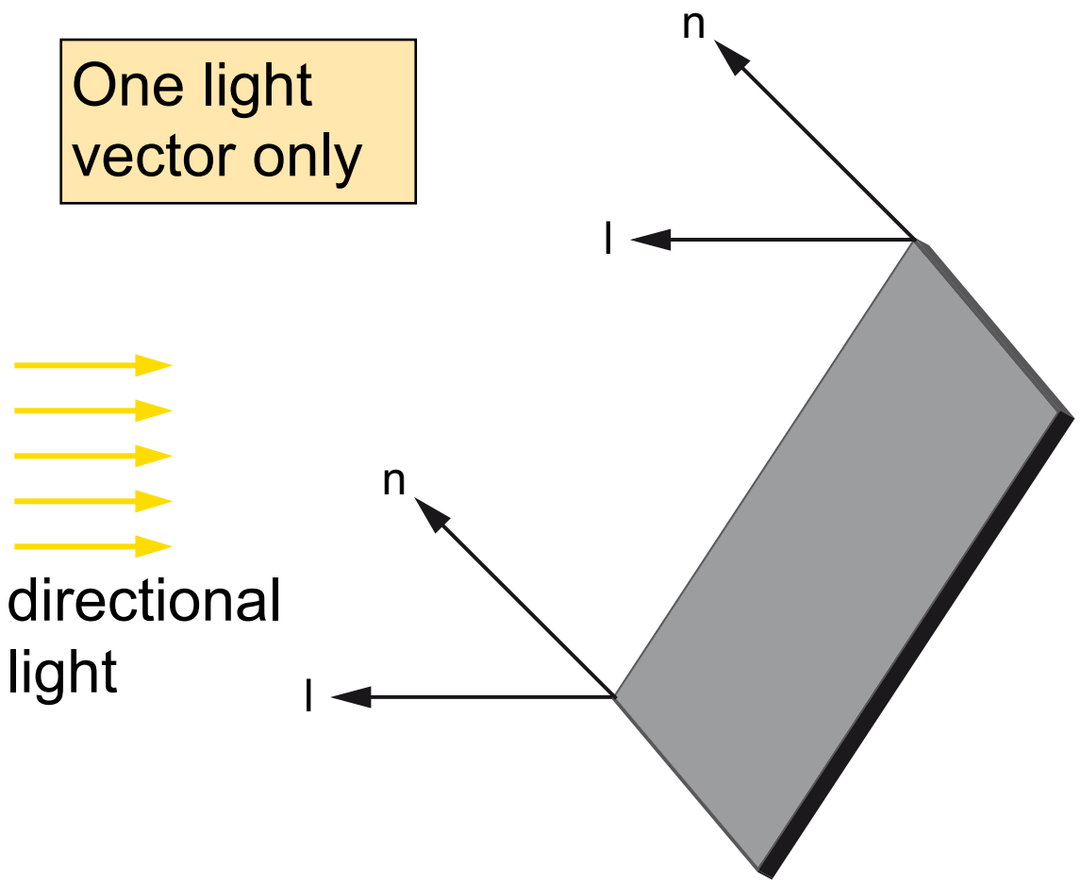
ARLAB

```
////////////////////////////////////////////////////////////////////////////////
// Directional light
//
if(light_position.w == 0.0) {
    // this is a directional light.

    // 1. the values that we store as light position is our light direction.
    vec3 light_direction = normalize(light_position.xyz);

    // 2. We check the angle of our light
    float light_to_surface_angle = dot(light_direction, transformedNormal.xyz);

    // 3. Check the angle, if the angle is smaller than 0.0,
    if(light_to_surface_angle > 0.0)attenuation = 1.0;
    else attenuation = 0.0;
}
```

Point / Spot Light vs. Directional Light

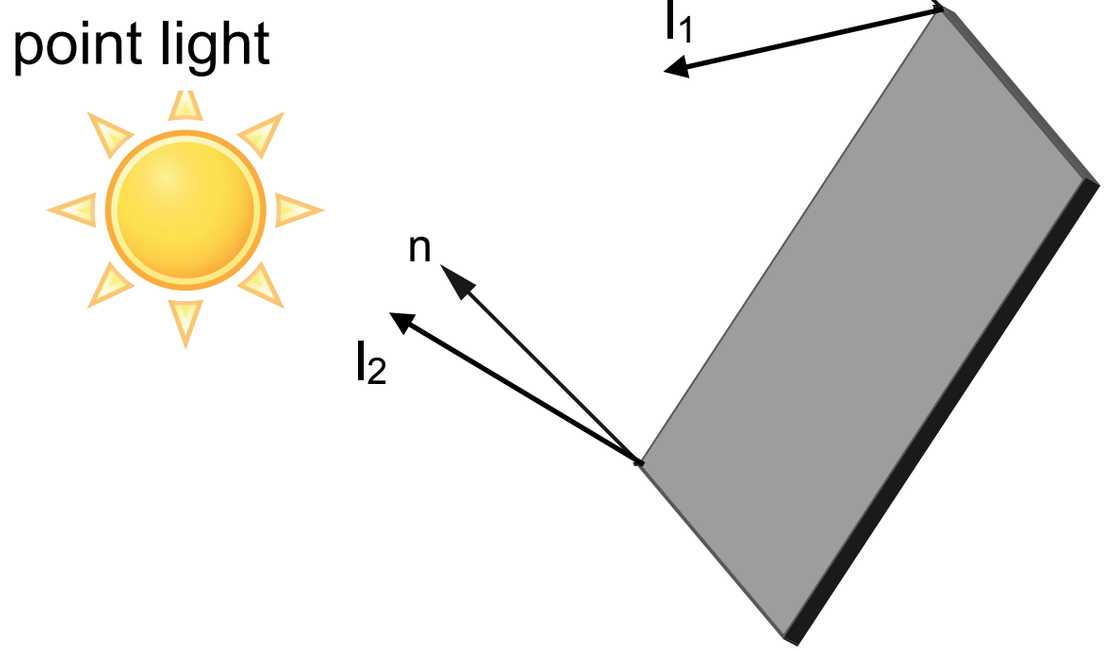
ARLAB

One light vector only



Directional Light; w = 0.0

One light vector for each vertex



Point Light; w = 1.0

```
GLfloat lightPos0[] = {4.0f, 0.0f, 8.0f, 1.0f };
```

w

```
GLfloat lightPos1[] = {-1.0f, 0.5f, 0.5f, 0.0f };
```

w

Directional light is faster to calculate because the graphics card can pre-calculate the light vector once for all vertices.

Light and Color Perception

ARLAB

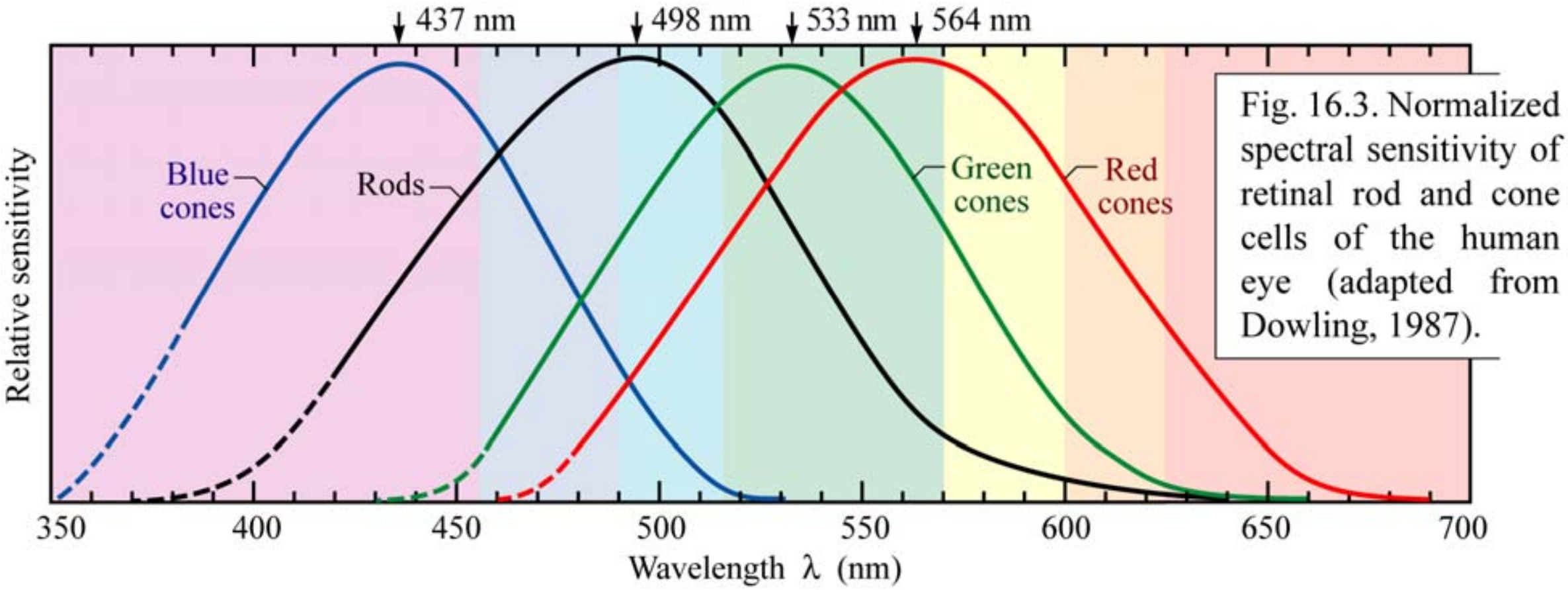


Fig. 16.3. Normalized spectral sensitivity of retinal rod and cone cells of the human eye (adapted from Dowling, 1987).

The human eye perceives light / color in a non-linear way.

Source: Human eye sensitivity and photometric quantities, "Light-Emitting Diodes" Second Edition, by E. Fred Schubert, 4

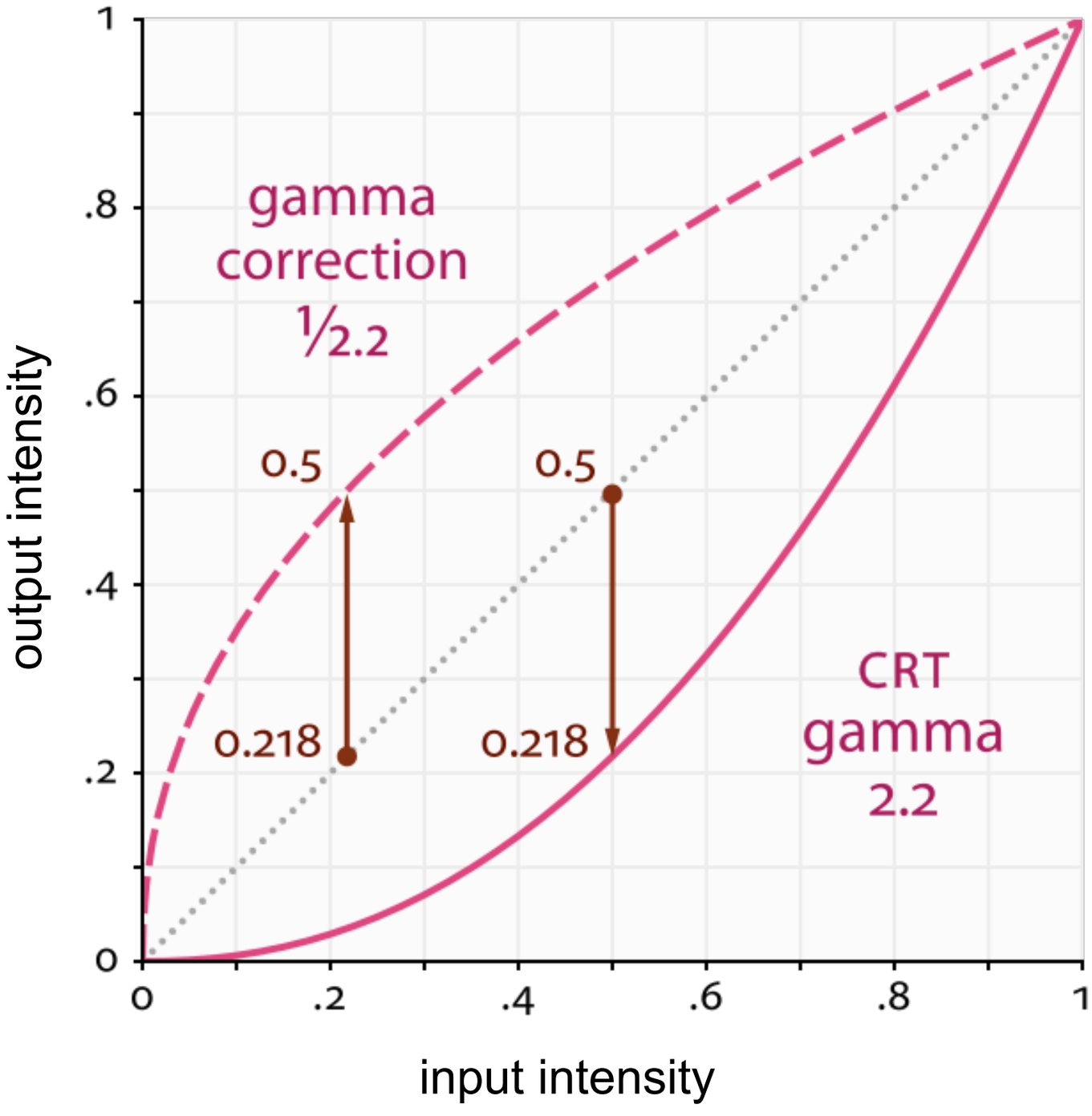
Color Models

Color models scale linear, we expect to see twice the intensity of a color when we double the value.

This is a generic example that mimics the effect (since the display uses gamma correction, it is difficult to demonstrate this effect.)

Colors look natural with gamma correction than without.

Gamma Correction



Gamma correction of images adapts the image color to the non-linear manner in which humans perceive light and color.

It was originally invented to take advantages of the bandwidth necessary to transport a signal to the display.

$$I_{out} = A \cdot I_{in}^{\gamma}$$

- A is a constant, which is A = 1, in a common case. But it can range between 0-1. It is used to boost the intensity.
- γ is the correction coefficient. which depends on the display. A gamma value for a CRT monitor is 1/2.2

Code Example

```
// Calculate the linear color
vec3 linearColor = out_ambient_color  + attenuation *
                   (out_diffuse_color + out_specular_color);
```

```
// Gamma correction
vec3 gamma = vec3(1.0/2.2);
vec3 finalColor = pow(linearColor, gamma);
```

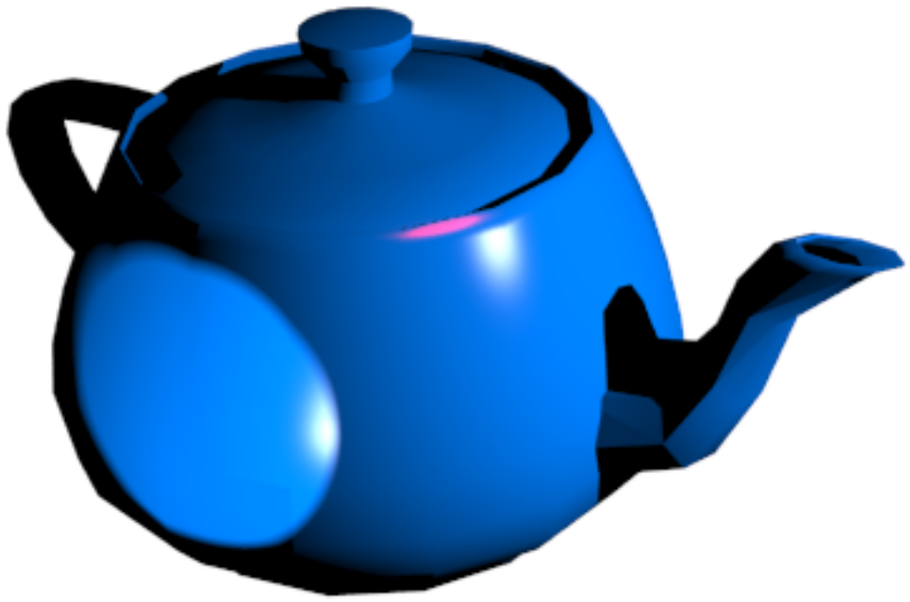
```
// Pass the color
pass_Color =  finalColor;
```

ARLAB

How do we transform a light source?

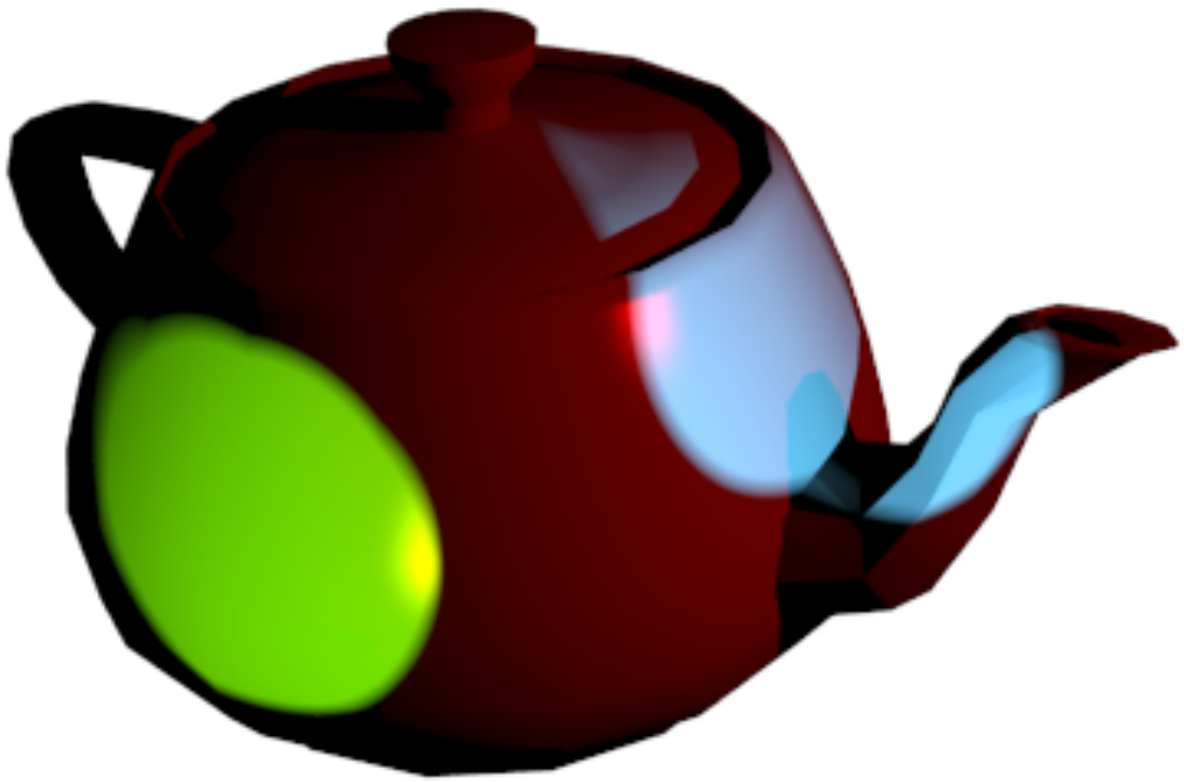
IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

What can we do to maintain multiple different light sources at the same time?



ARLAB

How can we add color to light?

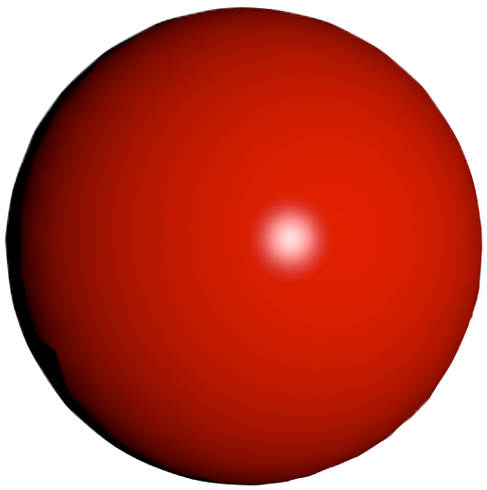


ARLAB

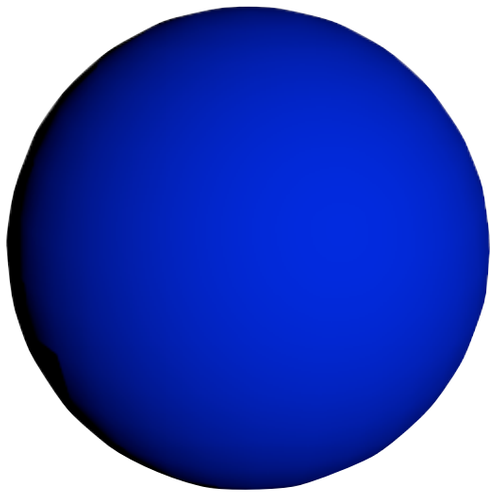
How can we improve the visual quality of the scene - or - what parameters are responsible for the visual quality?

Light and Material

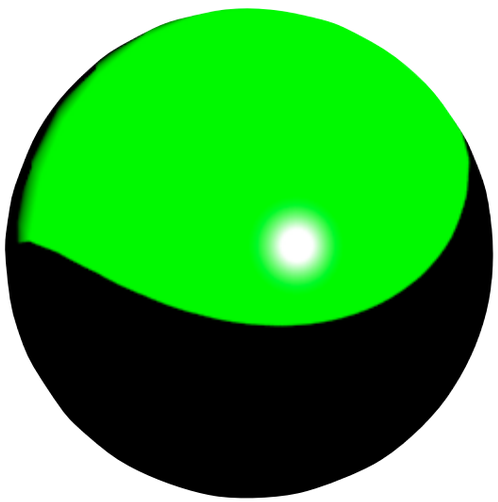
ARLAB



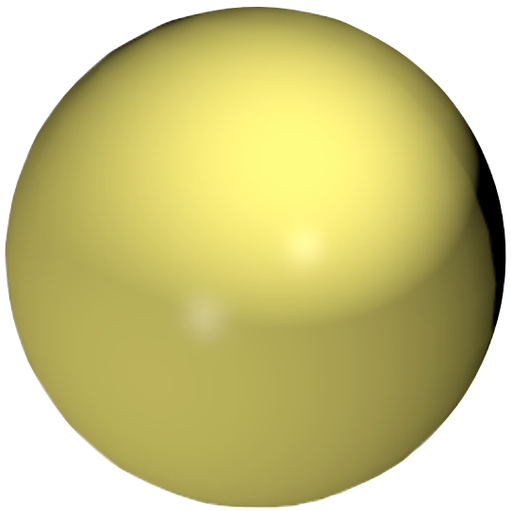
Highlight on the surface of a diffuse object



Diffuse surface, no highlights are visible



Spotlight with small highlight and sharp cutoff angle



Little highlight and smooth cutoff angle

ARLAB

*How do we realize transparency?

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

ARLAB

Code Example

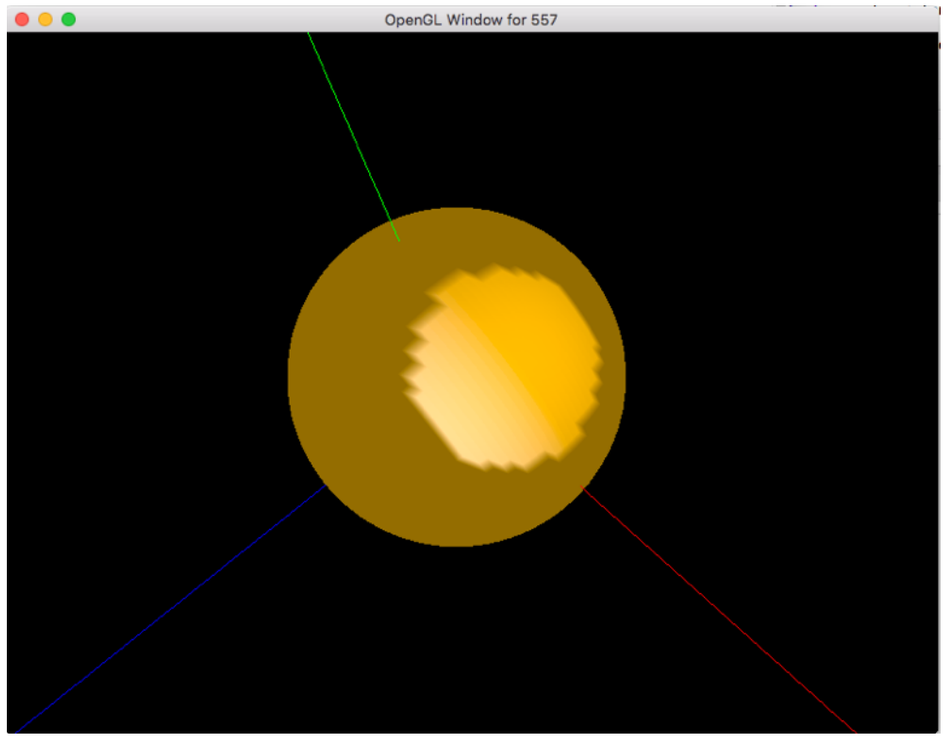
IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Two New Examples

ARLAB

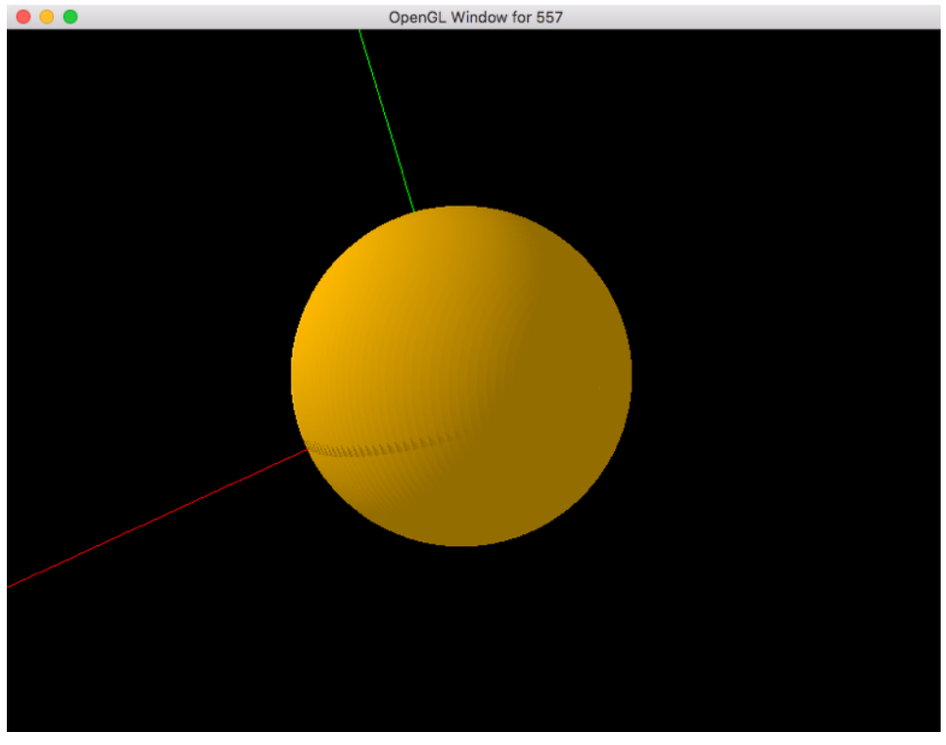
07_SpotLight

Shows how to realize a spot light.



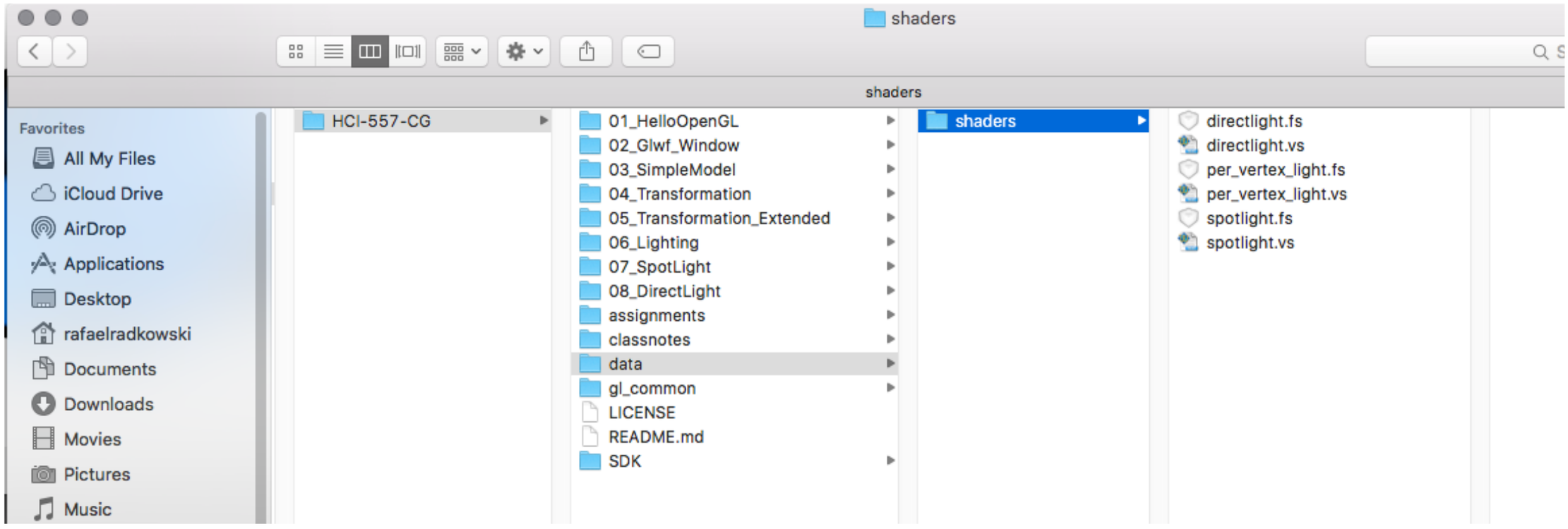
08_DirectLight

Shows how to realize a direct light.



GLSL Shader Code Location

Shader code is not saved in files in **data/shaders**



.vs - vertex shader code

.fs - fragment shader code

Advantage: no need to compile the C++ application when you make changes

Disadvantage: an additional file to handle and to load.

Code Example

ARLAB

```
void main(void)
{
    vec3 normal = normalize(in_Normal);
    vec4 transformedNormal =  normalize(transpose(inverse(modelMatrixBox)) * vec4( normal, 1.0 ));
    vec4 surfacePostion = modelMatrixBox * vec4(in_Position, 1.0);

    vec4 surface_to_light =   normalize( light_position -  surfacePostion );

    // Diffuse color
    float diffuse_coefficient = max( dot(transformedNormal, surface_to_light), 0.0);
    vec3 out_diffuse_color = diffuse_color  * diffuse_coefficient * diffuse_intensity;

    // Ambient color
    vec3 out_ambient_color = vec3(ambient_color) * ambient_intensity;

    // Specular color
    vec3 incidenceVector = -surface_to_light.xyz;
    vec3 reflectionVector = reflect(incidenceVector, transformedNormal.xyz);
    vec3 cameraPosition = vec3( -viewMatrixBox[3][0], -viewMatrixBox[3][1], -viewMatrixBox[3][2]);
    vec3 surfaceToCamera = normalize(cameraPosition - surfacePostion.xyz);
    float cosAngle = max( dot(surfaceToCamera, reflectionVector), 0.0);
    float specular_coefficient = pow(cosAngle, shininess);
    vec3 out_specular_color = specular_color * specular_coefficient * specular_intensity;


    //attenuation
    float distanceToLight = length(light_position.xyz - surfacePostion.xyz);
    float attenuation = 1.0 / (1.0 + attenuationCoefficient * pow(distanceToLight, 2));


    ////////////////////////////////////////////////////////////////////////////////
    // Spotlight
    // 1. Normalize the cone direction
    vec3 cone_direction_norm = normalize(cone_direction);

    // 2. Calculate the ray direction. We already calculated the surface to light direction.
    //    All what we need to do is to inverse this value
    vec3 ray_direction = -surface_to_light.xyz;

    // 3. Calculate the angle between light and surface using the dot product again.
    //    To simplify our understanding, we use the degrees
    float light_to_surface_angle = degrees(acos(dot(ray_direction, cone_direction_norm))) ;
```

Shaders.h/.cpp

ARLAB

Shaders.h/.cpp is a file with function to support shader code loading, program set up, and error handling.

Creates a shader program

```
GLuint CreateShaderProgram(string vertex_source, string fragment_source);
```

- vertex source - the vertex shader source code
- fragment_source - the fragment shader source code
- return - Gluint of the shader program

Loads a shader program from a file and creates the related program

```
GLuint LoadAndCreateShaderProgram(string vertex_file, string fragment_file);
```

- vertex_file - the file which stores the vertex shader code
- fragment_file - the file which stores the fragment shader code
- return - Gluint of the shader program

GLSL Reference page

The OpenGL Shading Language reference pages have been merged with the OpenGL API reference pages.

https://www.opengl.org/sdk/docs/man4/

OpenGL Software Development Kit

DOCUMENTATION, SAMPLE CODE, LIBRARIES, AND TOOLS FOR CREATING OPENGL-BASED APPLICATIONS

SDK Home | Documentation | Libraries | Tutorials | Tools | Forums

Name

acos — return the arccosine of the parameter

Declaration

genType **acos**(genType *x*);

Parameters

x

Specify the value whose arccosine to return.

Description

atan returns the angle whose trigonometric cosine is x. The range of values returned b result is undefined if $|x| > 1$.

GLSL only

Version Support

Function Name	OpenGL Shading Language Version											
	1.10	1.20	1.30	1.40	1.50	3.30	4.00	4.10	4.20	4.30	4.40	4.50
acos	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

See Also

sin, cos, asin, tan

GLSL Reference page

ARLAB

SDK Home | Documentation | Libraries | Tutorials | Tools | Forums

Once created, a named texture may be re-bound to its same original target as often as needed. It is usually much faster to use **glBindTexture** to bind an existing named texture to one of the texture targets than it is to reload the texture image using glTexImage1D, glTexImage2D, glTexImage3D or another similar function.

Notes

The `GL_TEXTURE_2D_MULTISAMPLE` and `GL_TEXTURE_2D_MULTISAMPLE_ARRAY` targets are available only if the GL version is 3.2 or higher.

Errors

`GL_INVALID_ENUM` is generated if *`target`* is not one of the allowable values.

`GL_INVALID_VALUE` is generated if *`target`* is not a name returned from a previous call to glGenTextures.

`GL_INVALID_OPERATION` is generated if *`texture`* was previously created with a target that doesn't match that of *`target`*.

Associated Gets

glGet with argument `GL_TEXTURE_BINDING_1D`, `GL_TEXTURE_BINDING_2D`, `GL_TEXTURE_BINDING_3D`, `GL_TEXTURE_BINDING_1D_ARRAY`, `GL_TEXTURE_BINDING_2D_ARRAY`, `GL_TEXTURE_BINDING_RECTANGLE`, `GL_TEXTURE_BINDING_BUFFER`, `GL_TEXTURE_BINDING_CUBE_MAP`, `GL_TEXTURE_BINDING_CUBE_MAP`, `GL_TEXTURE_BINDING_CUBE_MAP_ARRAY`, `GL_TEXTURE_BINDING_2D_MULTISAMPLE`, or `GL_TEXTURE_BINDING_2D_MULTISAMPLE_ARRAY`.

OpenGL only

Version Support

	OpenGL Version											
Function / Feature Name	**2.0**	**2.1**	**3.0**	**3.1**	**3.2**	**3.3**	**4.0**	**4.1**	**4.2**	**4.3**	**4.4**	**4.5**
glBindTexture	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

See Also

glDeleteTextures, glGenTextures, glGet, glGetTexParameter, glIsTexture, glTexImage1D, glTexImage2D, glTexImage2DMultisample, glTexImage3D, glTexImage3DMultisample, glTexBuffer, glTexParameter

Copyright

GLSL Reference page

ARLAB

sqrt — return the square root of the parameter

```
genType sqrt( genType x);

genDType sqrt( genDType x);
```

`genType` : a generic datatype which is most of the time **float, vec2, vec3, vec4**

`genDType` : a generic datatype which is most of the time **double, vec2, vec3, vec4**

- Input datatype does not need to be the same as the output datatype, so you must know what the function does for you.
- BUT: if the input is a float datatype, the output is also a float datatype; same for double

GLSL Shader Functions

ARLAB

length — calculate the length of a vector

```
float length(genType x);
```

Parameters:

x - Specifies a vector of which to calculate the length.

pow — return the value of the first parameter raised to the power of the second

```
genType pow( genType x, genType y);
```

Parameters:

x - Specify the value to raise to the power y.

y - Specify the power to which to raise x.

https://www.opengl.org/sdk/docs/man4/

GLSL Shader Functions

ARLAB

normalize — calculate the normalize product of two vectors

```
genType normalize(genType v);
```

Parameters:

v - Specifies the vector to normalize.

degrees — convert a quantity in radians to degrees

```
genType degrees(genType radians);
```

Parameters:

- radians - Specify the quantity, in radians, to be converted to degrees.

https://www.opengl.org/sdk/docs/man4/

GLSL Shader Functions

ARLAB

acos — return the arccosine of the parameter

```
genType acos(genType x);
```

Parameters:

x - Specify the value whose arccosine to return.

dot — calculate the dot product of two vectors

```
float dot(genType x, genType y);
```

Parameters:

- x - Specifies the first of two vectors
- y - Specifies the second of two vectors

https://www.opengl.org/sdk/docs/man4/

Thank you!

Questions

Rafael Radkowski, Ph.D.
Iowa State University
Virtual Reality Applications Center
1620 Howe Hall
Ames, Iowa 5001, USA

+1 515.294.5580
+1 515.294.5530(fax)

rafael@iastate.edu

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY