Illumination Models

Rendering in Passes

- Rendering in Passes is not a new technique for 3D graphics
- The word *pass* originated in motion-control photography used for miniatures.
- Each time a motion is repeated it is called a "pass"
- Usually the first pass was fully lit and called the "beauty pass"
- Next the lighting and film would be changed for a High contrast "matte" pass.
- For a third pass a light may be turned on in the model.
- Finally all the passes used to be printed together optically for the final shot (before the day of digital compositing systems)

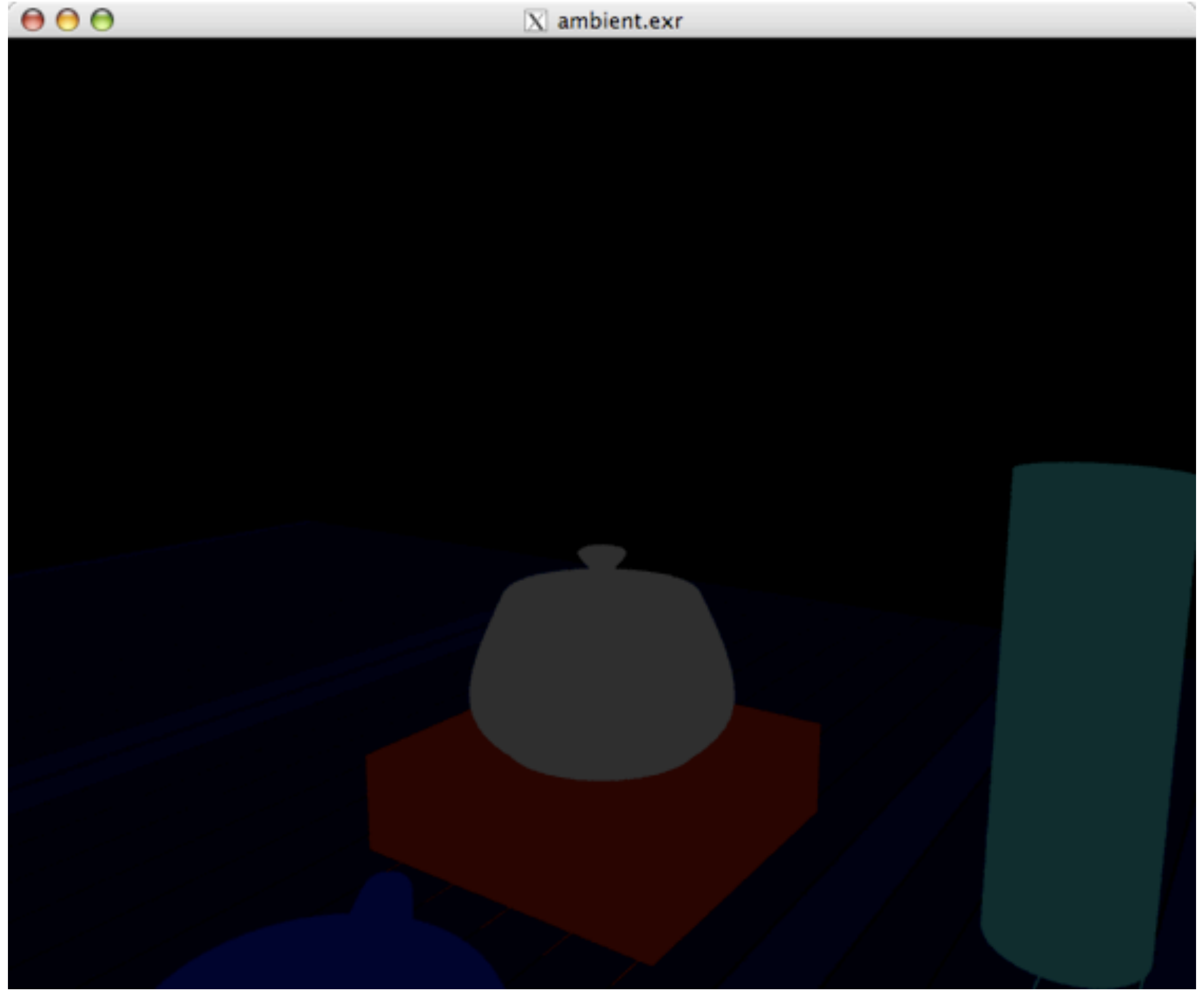
Render Passes

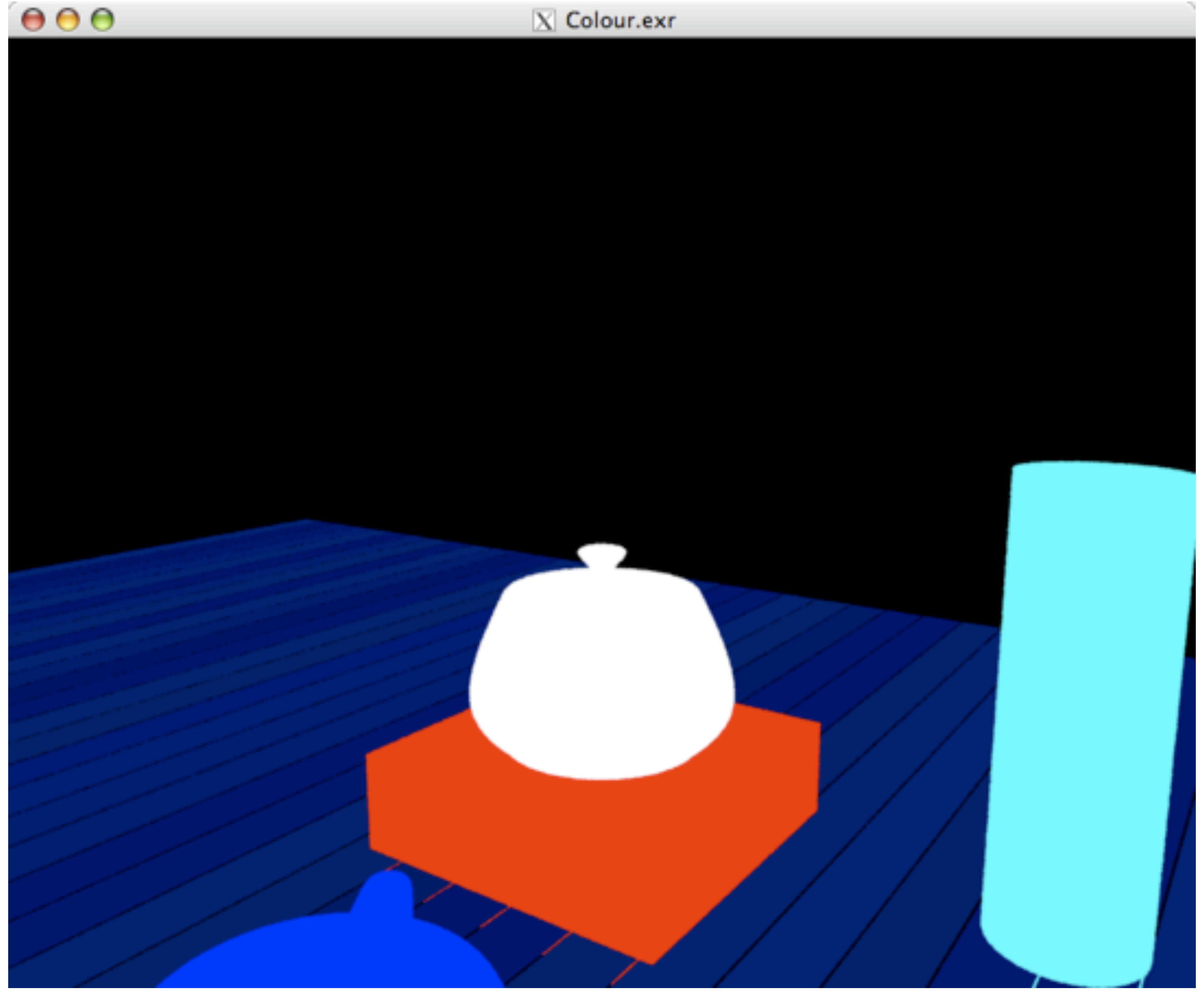
- It is usually preferable to split our renders up into multiple passes
- These can either be separate files for each pass or combined into one exr file.
- This will allow us to make adjustments to overall colour values in the image, and also add extra effects or even place elements in different positions
- It is also not uncommon to output other non colour values which give us information about elements of the scene such as depth from camera, surface normals, motion vectors etc.

Shader Development

- Typically when developing a shader we combine in the shader a number of different elements layered together.
- Usually for a simple pass we would have a basic colour element and then several elements which take into account the position of the viewer and the lights in the scene.
- Next week we will start to look at the mathematics behind this next week
- This week we will introduce the basic terms and techniques used in rendering.

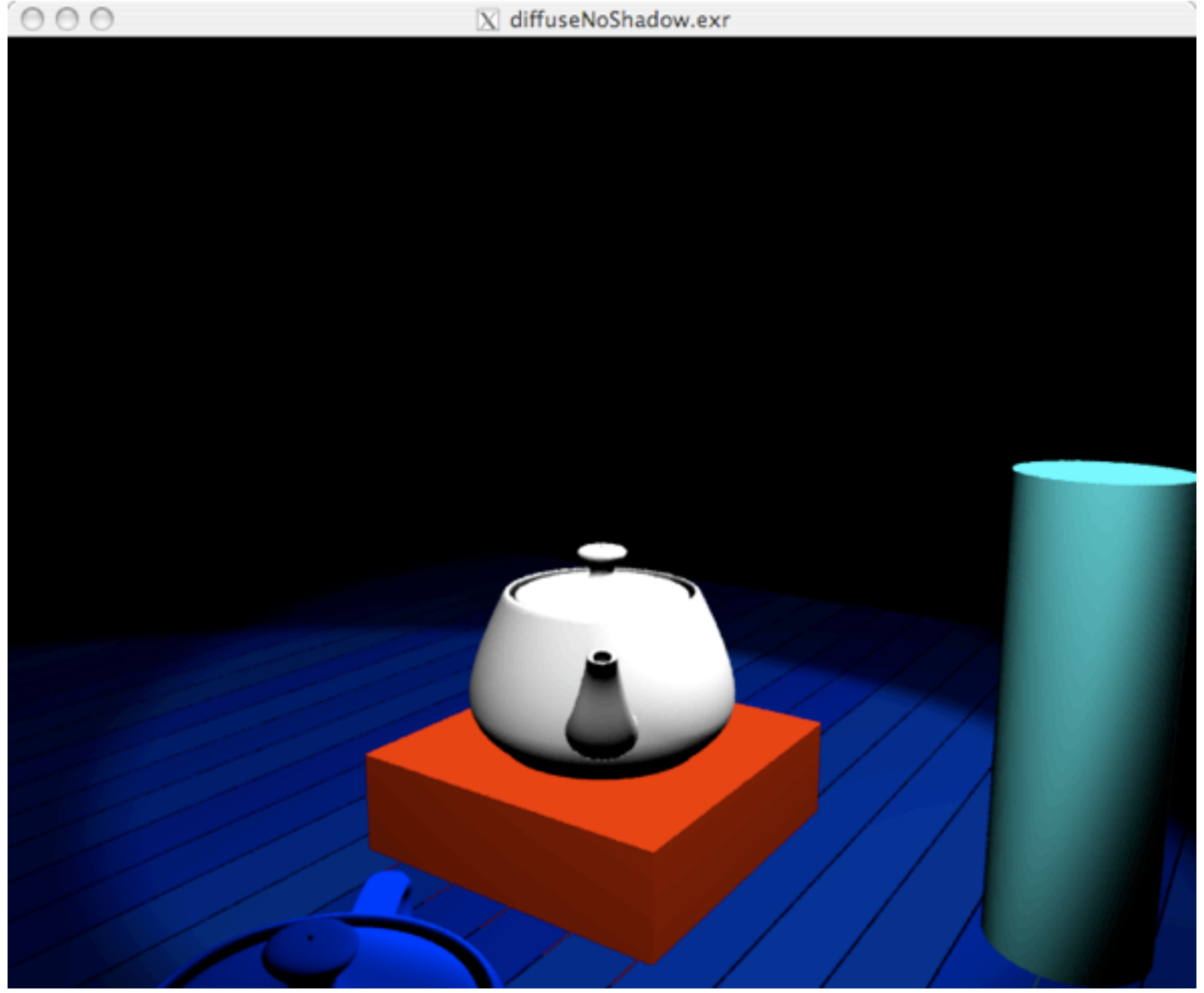
Ambient Pass

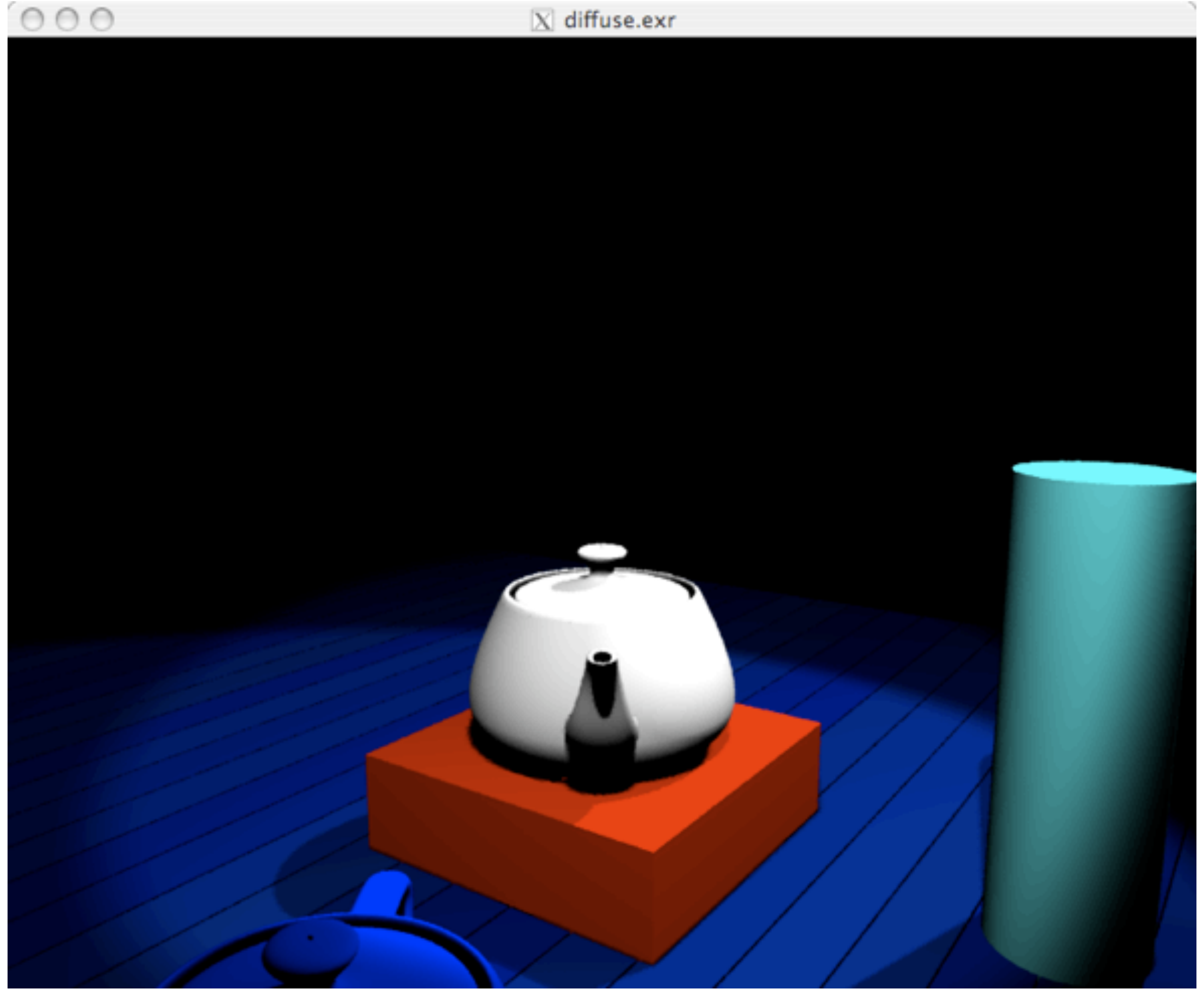




- The ambient pass on the right is the basic colour of the object multiplied by the ambient light values set for the scene.

- The colour on the left is just the colour assigned to the object

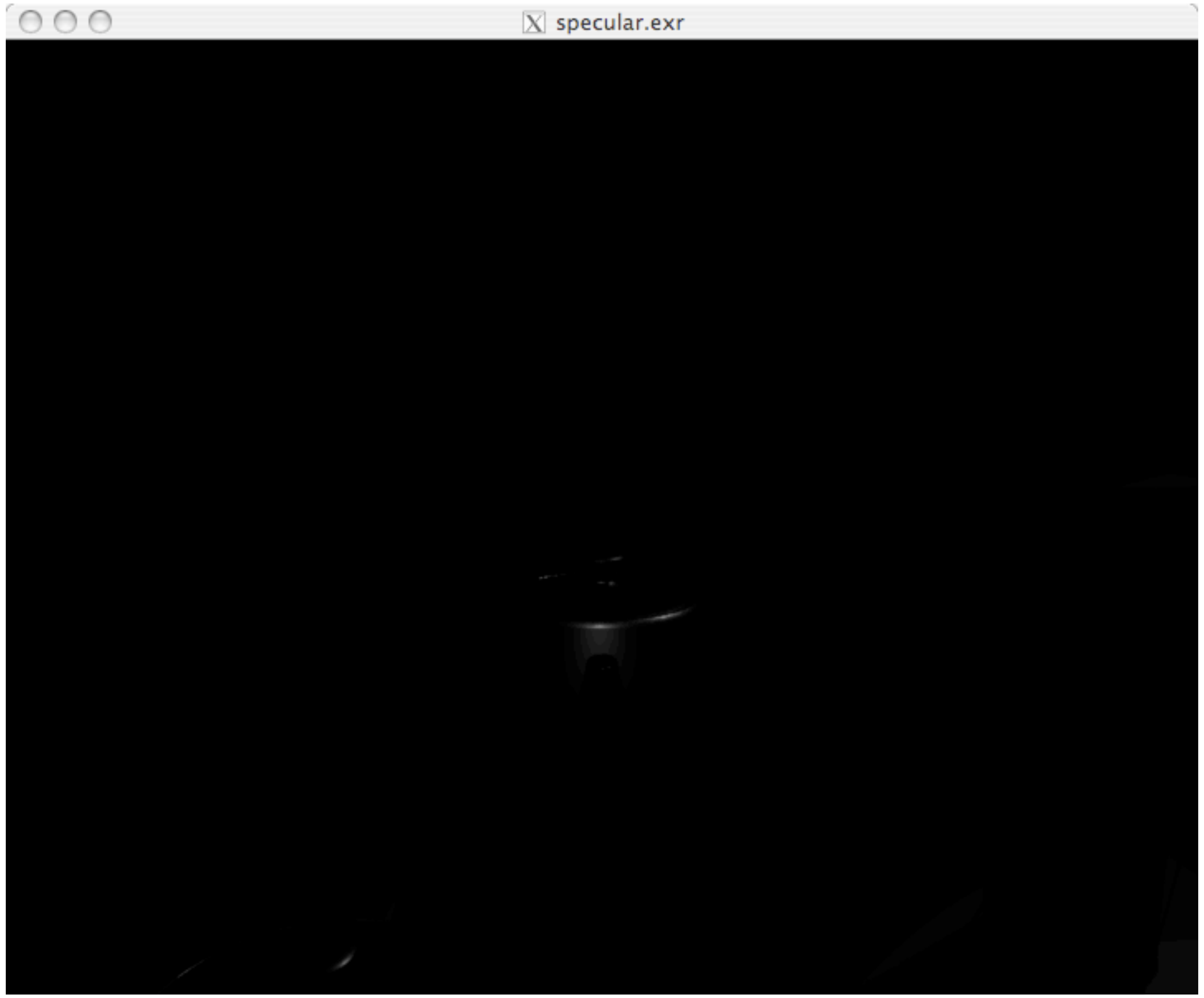




Diffuse Pass

- The diffuse pass usually has the basic shading model applied to the objects
- This pass takes into account Position of the lights in the scene as well as the orientation of the surface (based on the surface Normal)
- The top pass has no shadows
- The 2nd pass has the shadows included as by default the renderman ray trace shadows are in the diffuse pass

Specular Pass



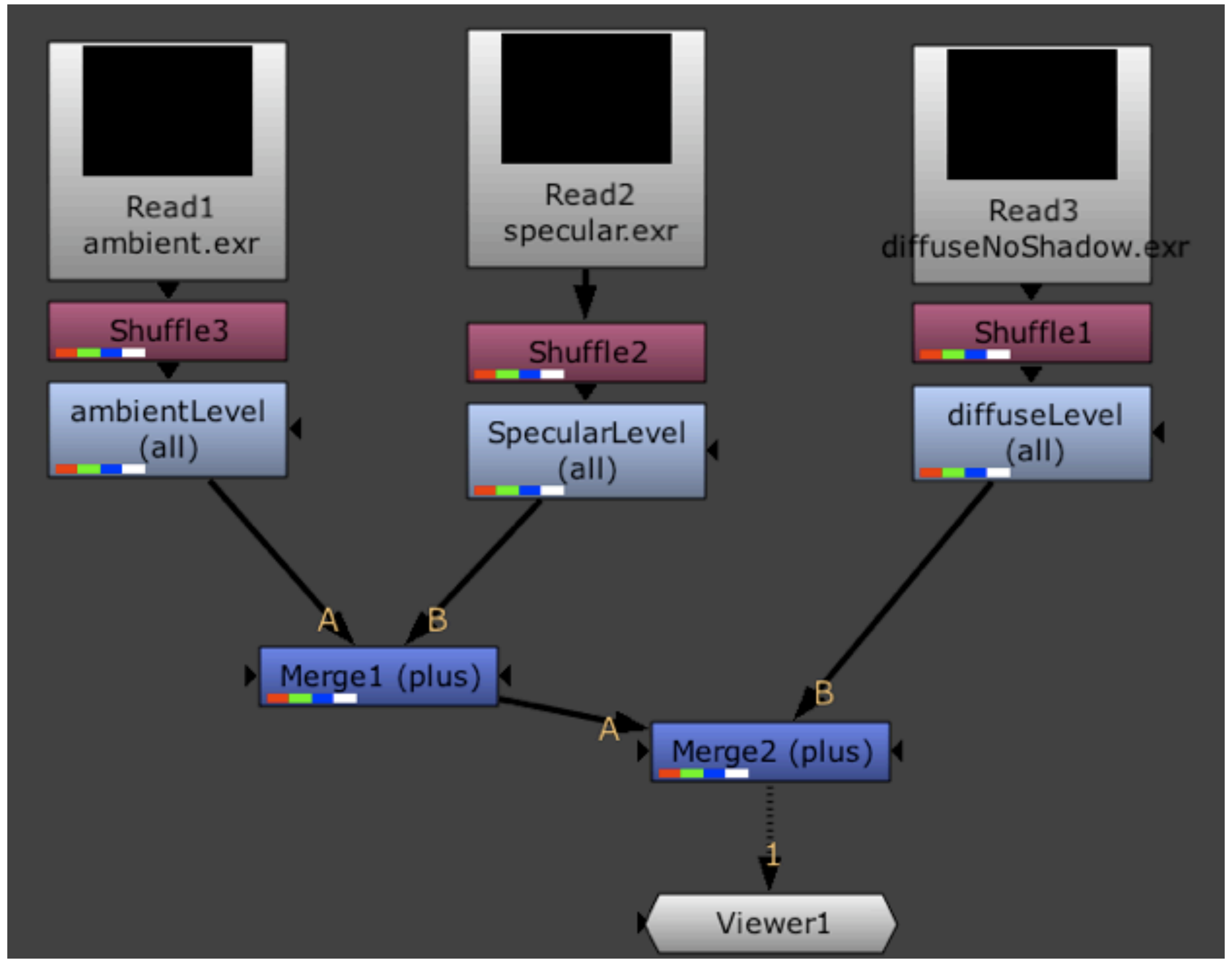
- The specular or Highlight pass takes into account the direction of the light

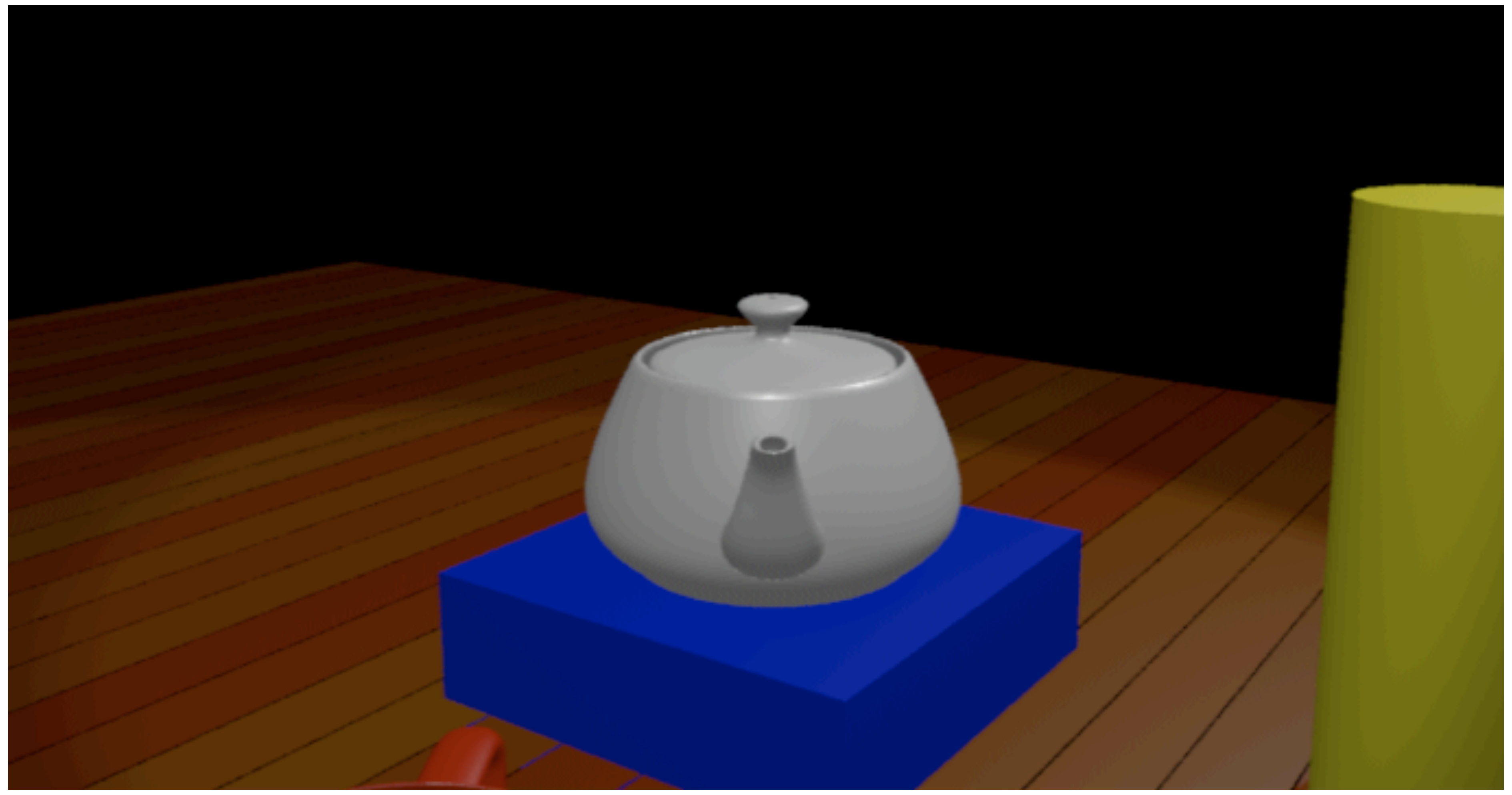
- Specular highlights are important in 3D computer graphics, as they provide a strong visual cue for the shape of an object and its location with respect to light sources in the scene.

Combining Basic Passes

- Usually when using a basic Blinn / Plastic / Phong Shader the output of the render will be a combination of the Ambient, Diffuse and Specular contributions

- If we have separate passes we can combine them to get the same effect using the following compositing structure

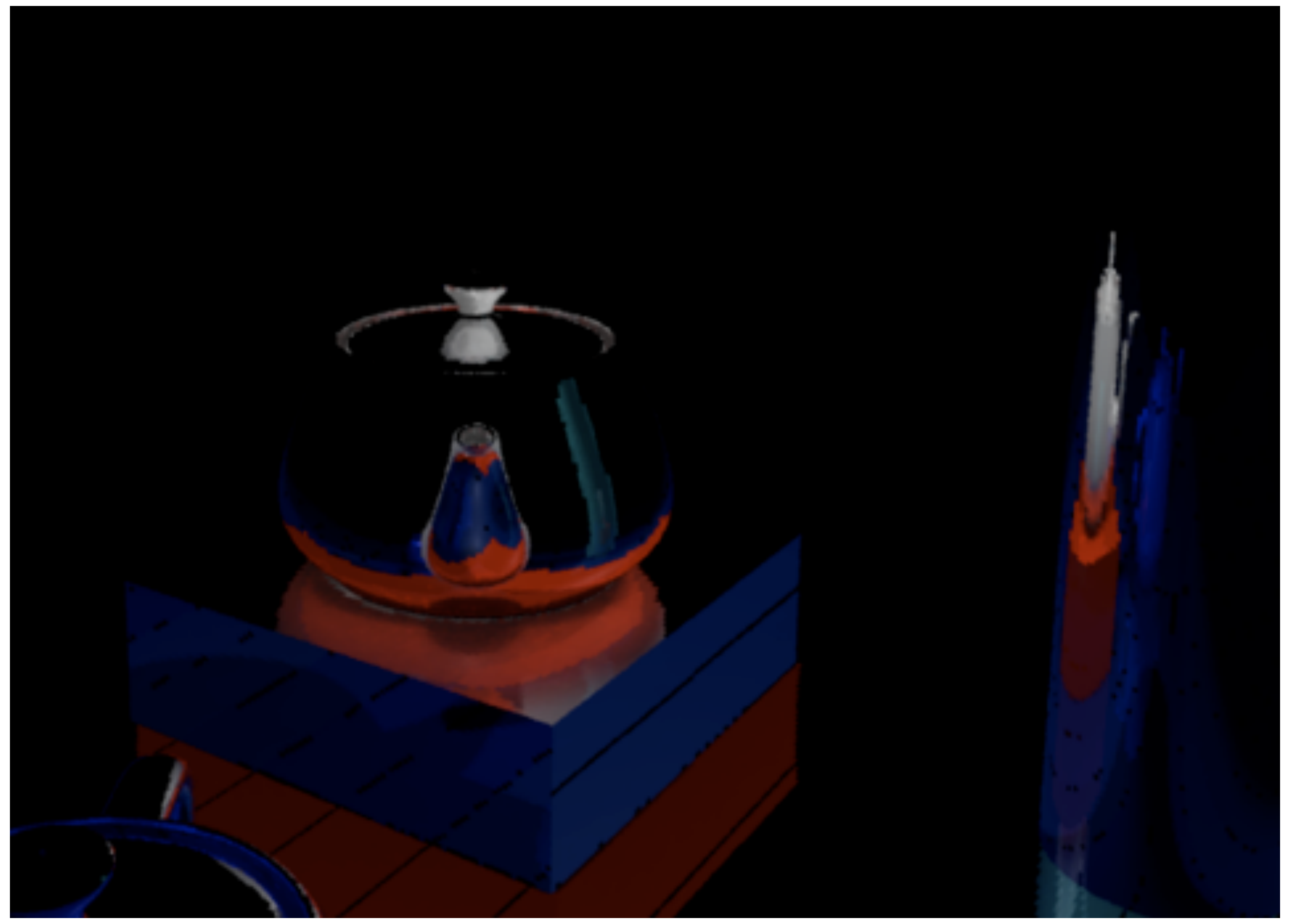


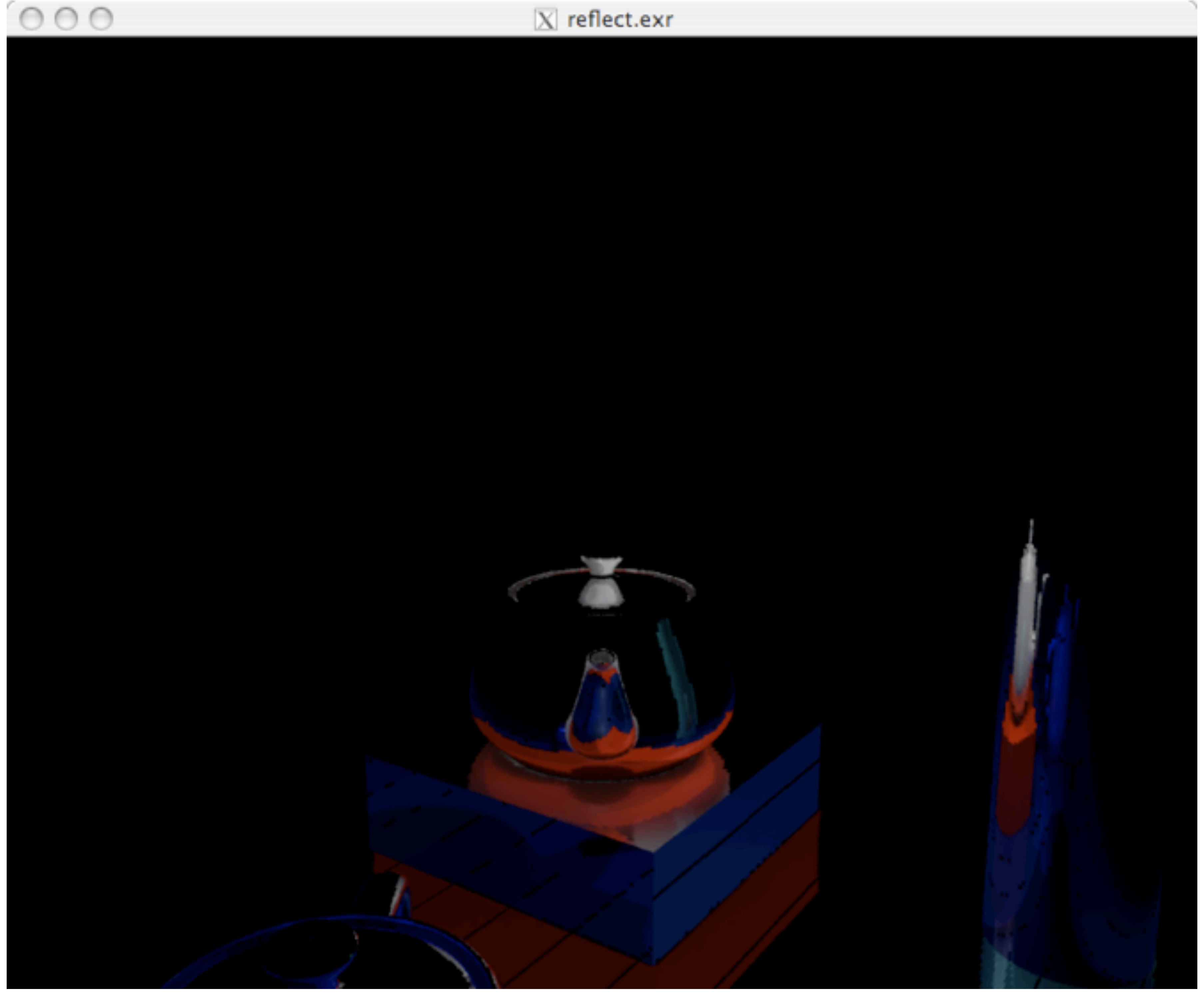


Ray tracing

- The previous images can be produced very easily using a process called scan line rendering
- This process works on a row-by-row basis rather than a polygon-by-polygon or pixel-by-pixel basis.
- However it doesn’t take into account rays bouncing off of object and reflections / refraction in the scene
- To do this we need to invoke ray-tracing which will bounce rays around the scene to gather more information about the lights in the scene.
- This increases the processing required and will slow down renders

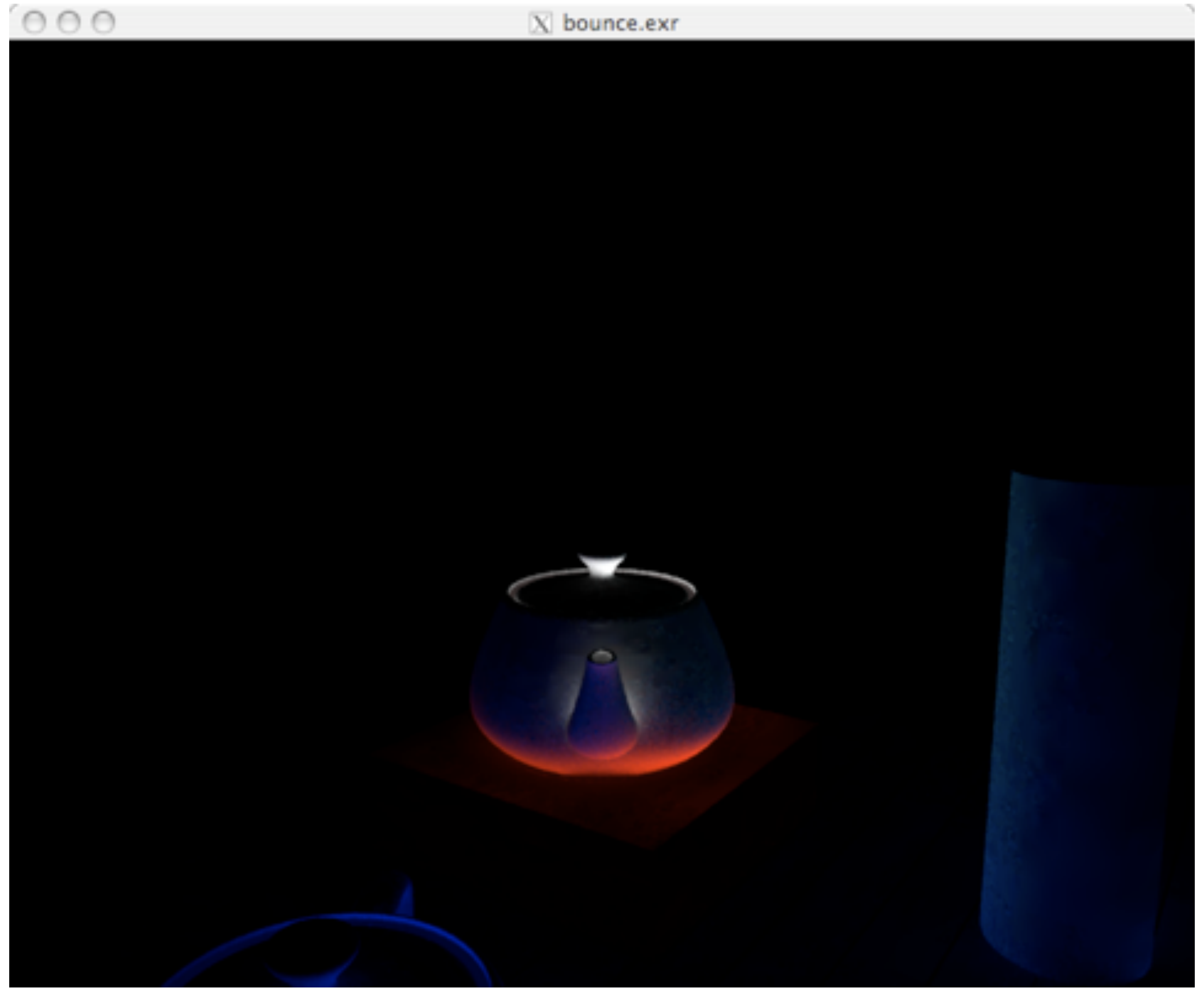
Reflection pass

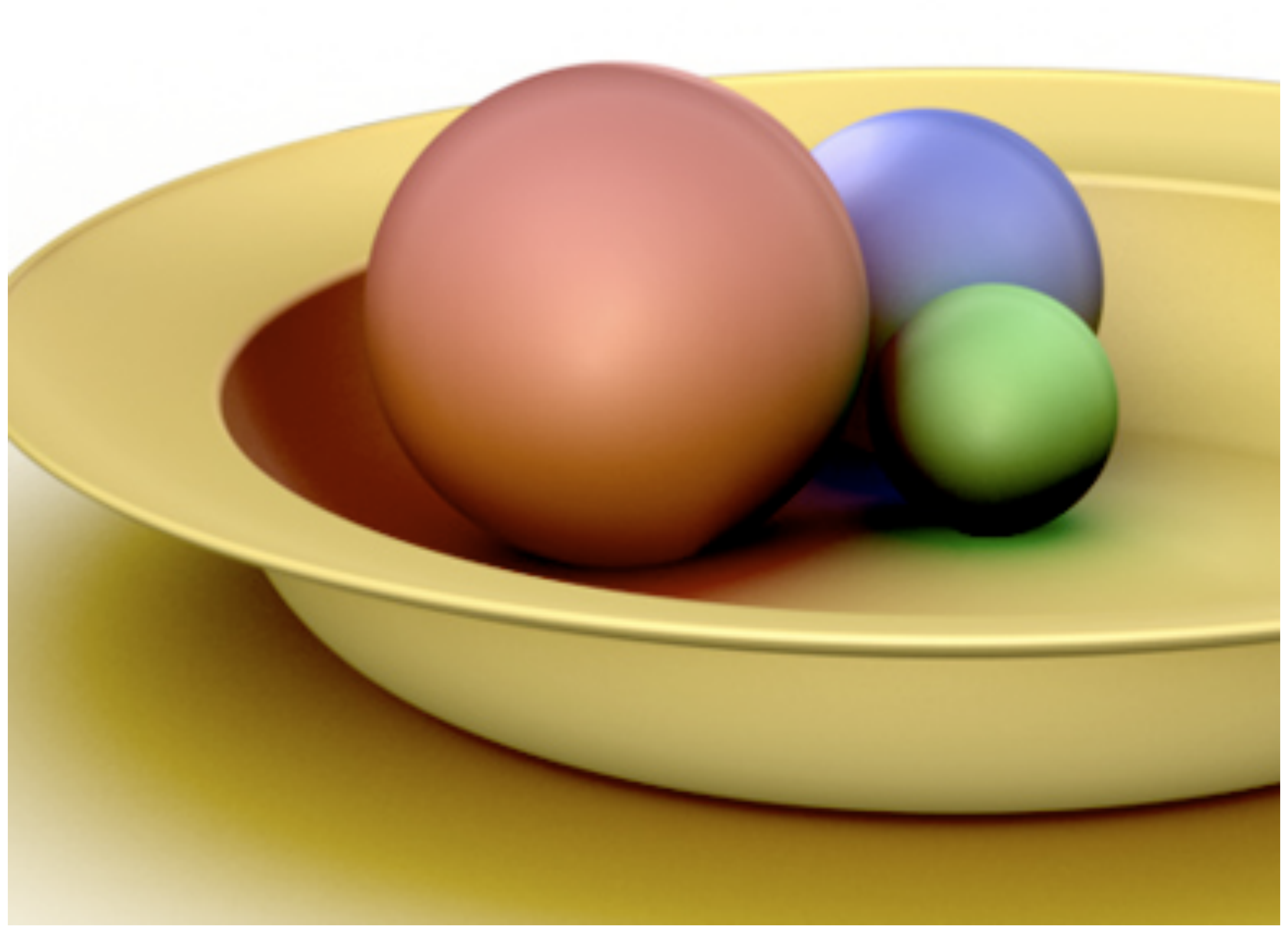




- Usually we can determine the number of times the rays bounce for the reflection pass
- In this case 2 but increasing to 3 or 4 will increase the render times logarithmically
- It is best to keep the bounce levels as low as possible

Indirect diffuse



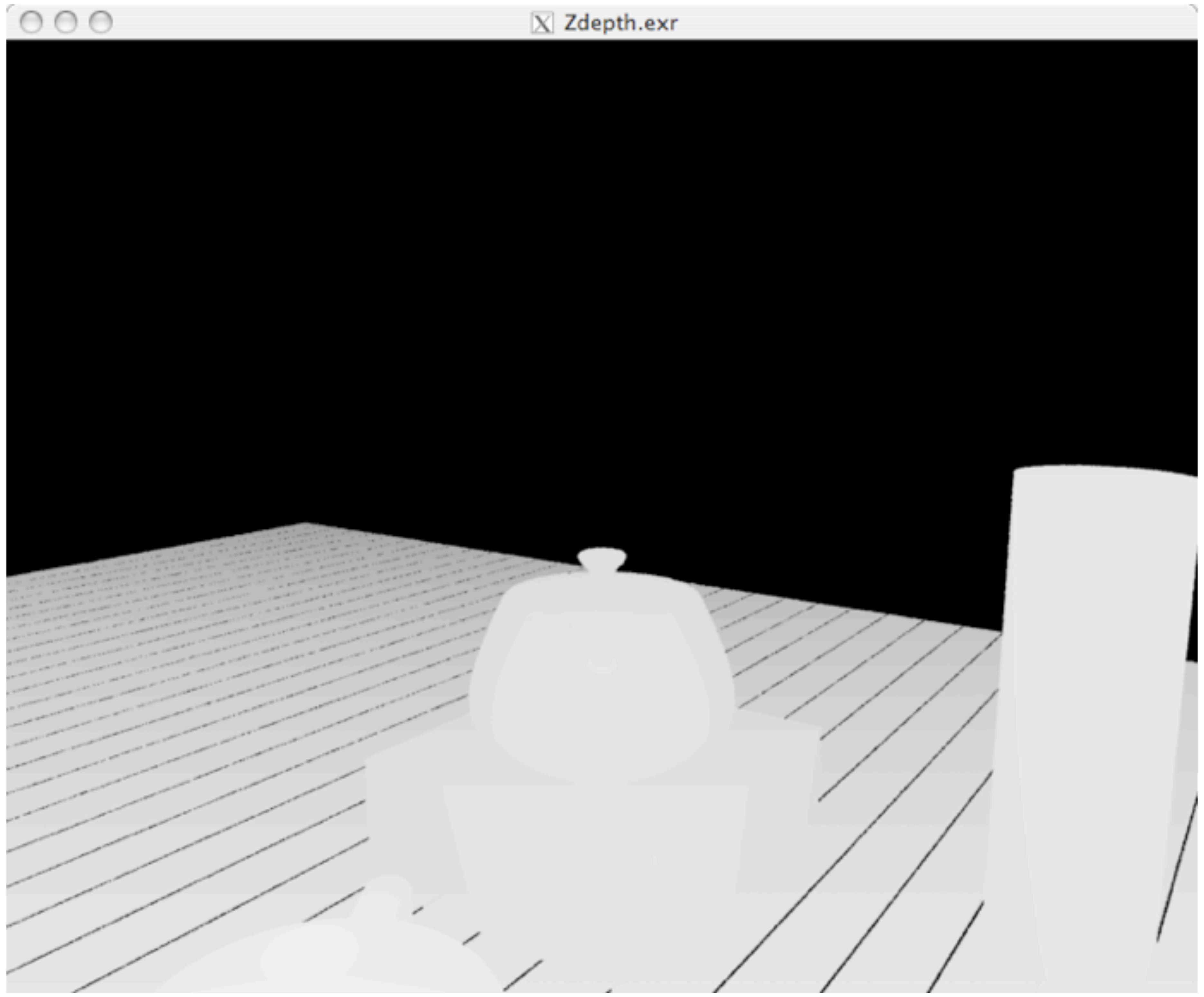


- Indirect diffuse illumination can add different elements such as colour bleeding
- There are a number of methods of calculating this including, Photon mapping
- Renderman has a build in function called indirect diffuse
- again this process can be slow

Advanced Passes

- So far we have dealt with passes that contain colour information
- However we can save out other information instead of rgb values.
- Typically we still save to the rgb channels but we store other data in the same format.
- For example we may want to store the surface normal values which are usually in the form of a Normalized vector **N** [x,y,z] so the r channel becomes x, g=y and b=z.
- We can view these images but they are typically used in the compositor for advanced effects such as re-lighting or Zdepth DOF effects

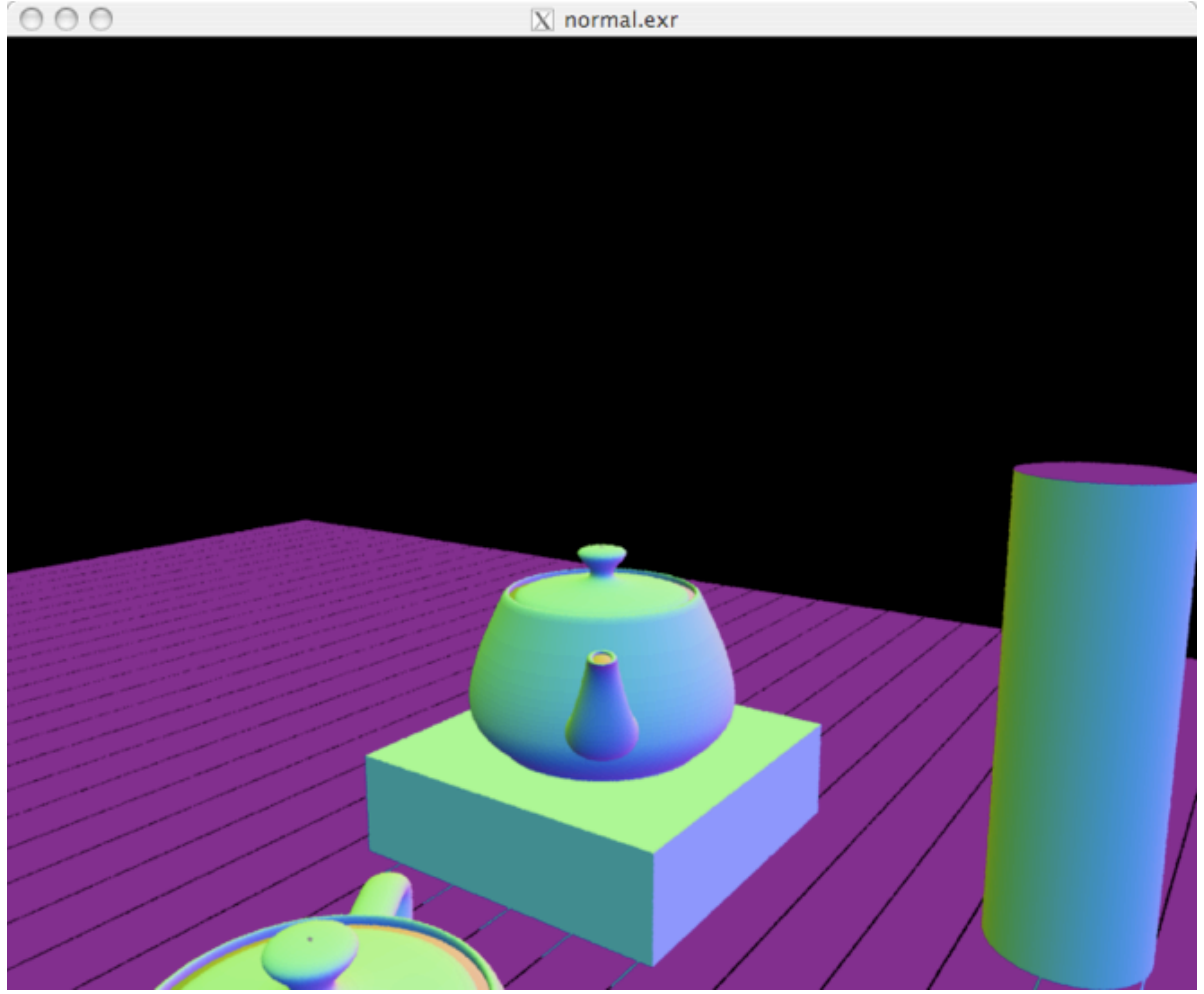
zDepth



```
Zdepth = 1-depth(P);
```

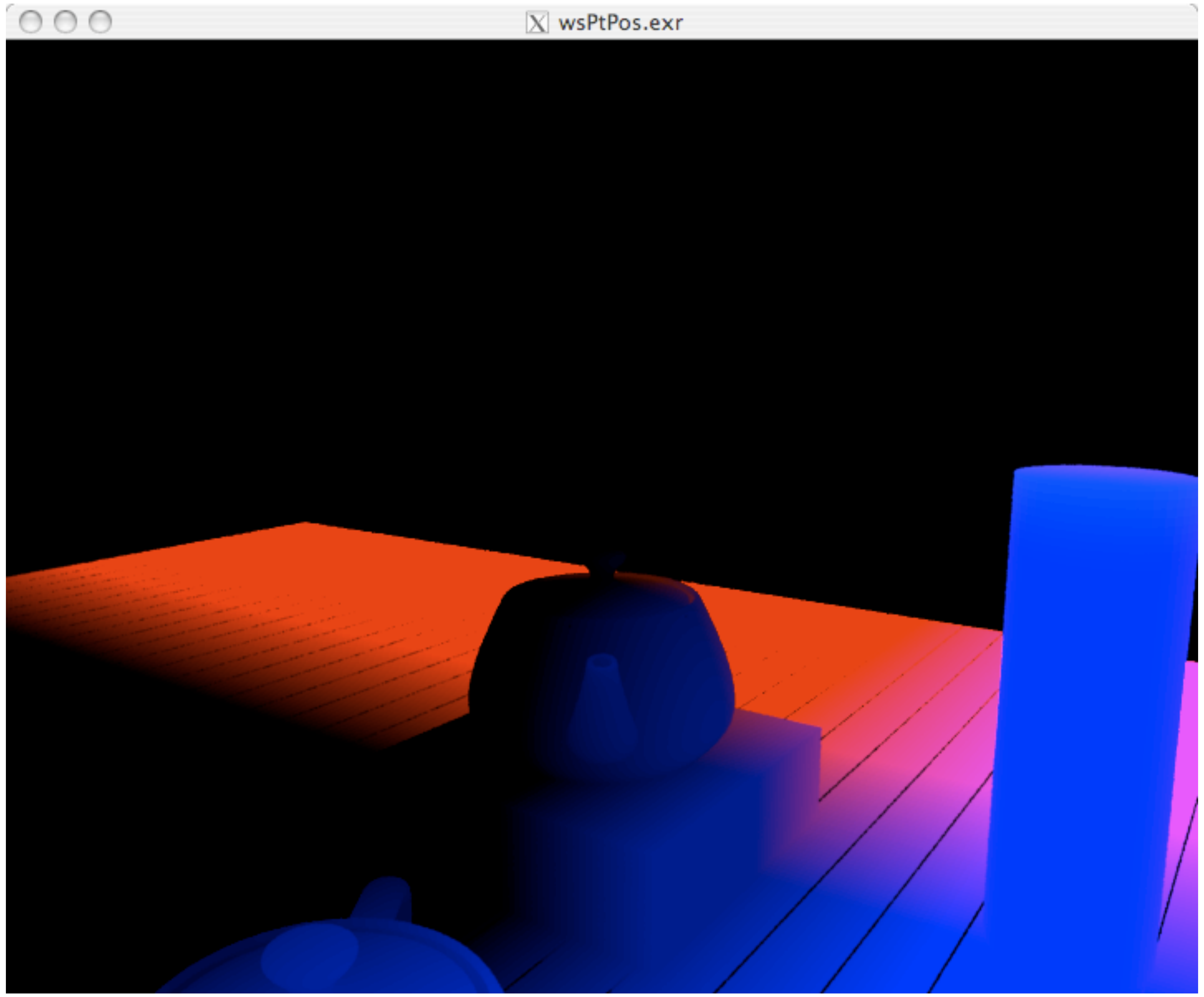
- This zDepth pass was created in renderman using the built in depth function
- This allows us to pass the current point being shaded and get the depth relative to the camera

Normals



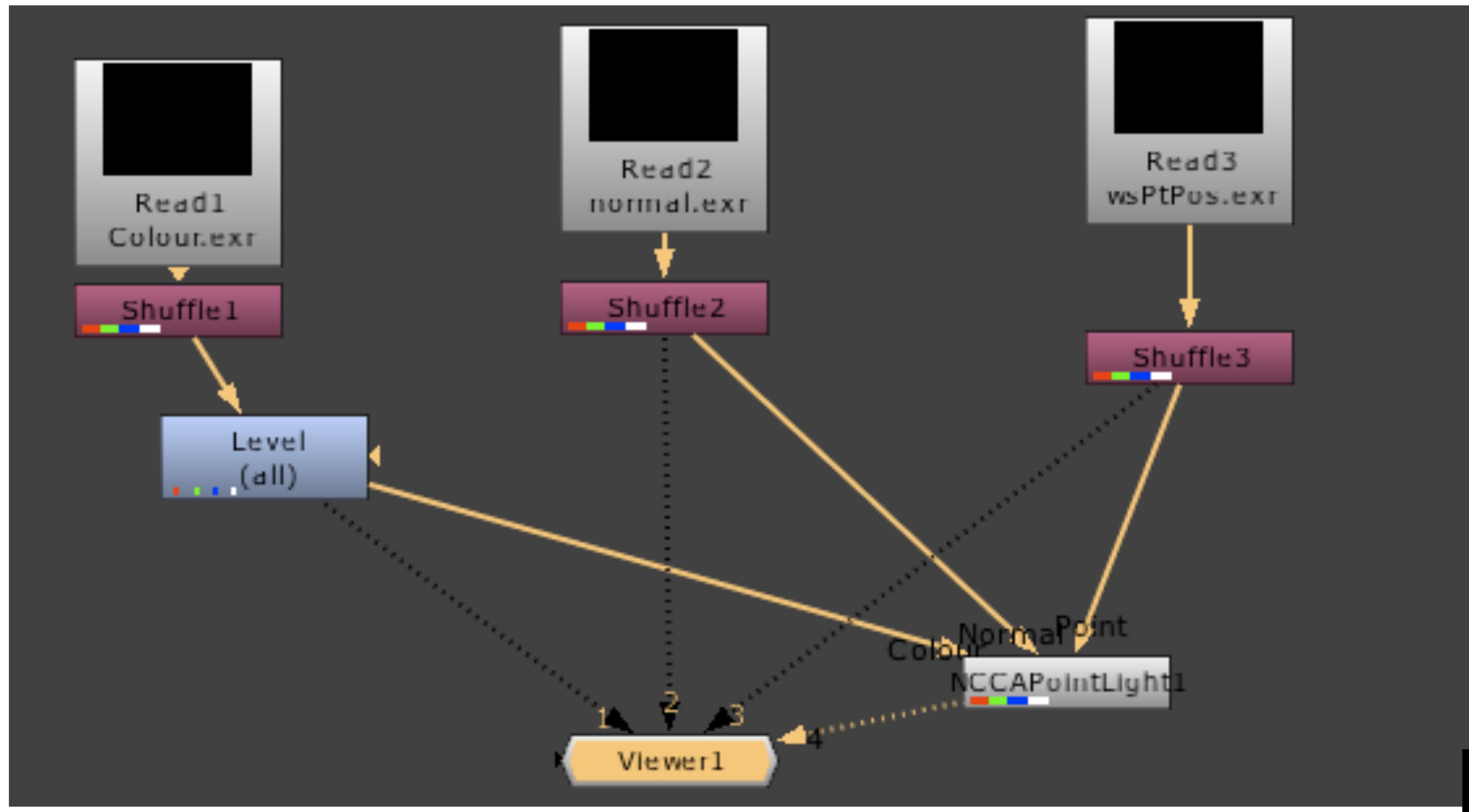
- The surface normal of the object in view is stored in the image
- This is usually normalized and forced to face towards the camera / viewer to ensure it is in the correct direction

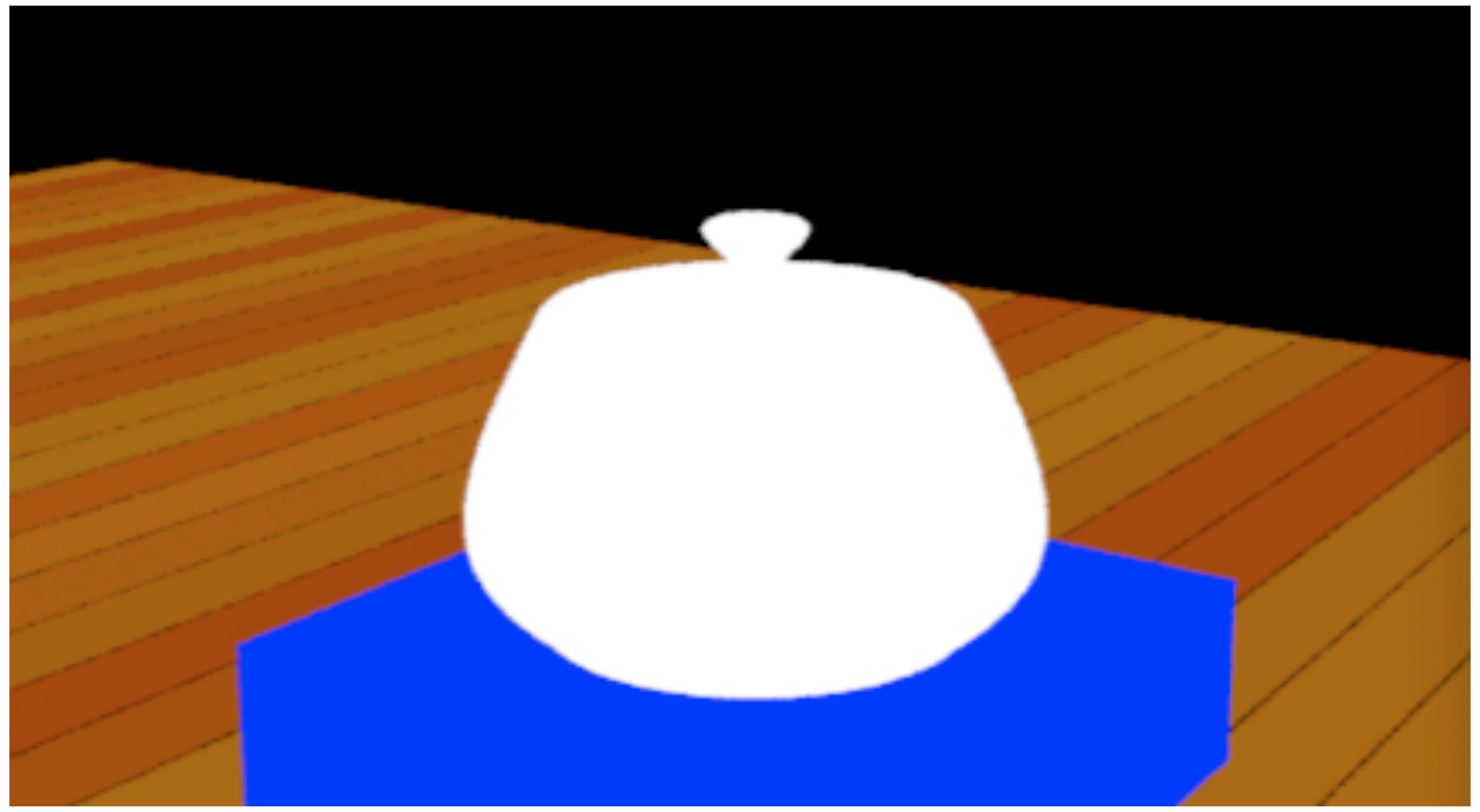
Points

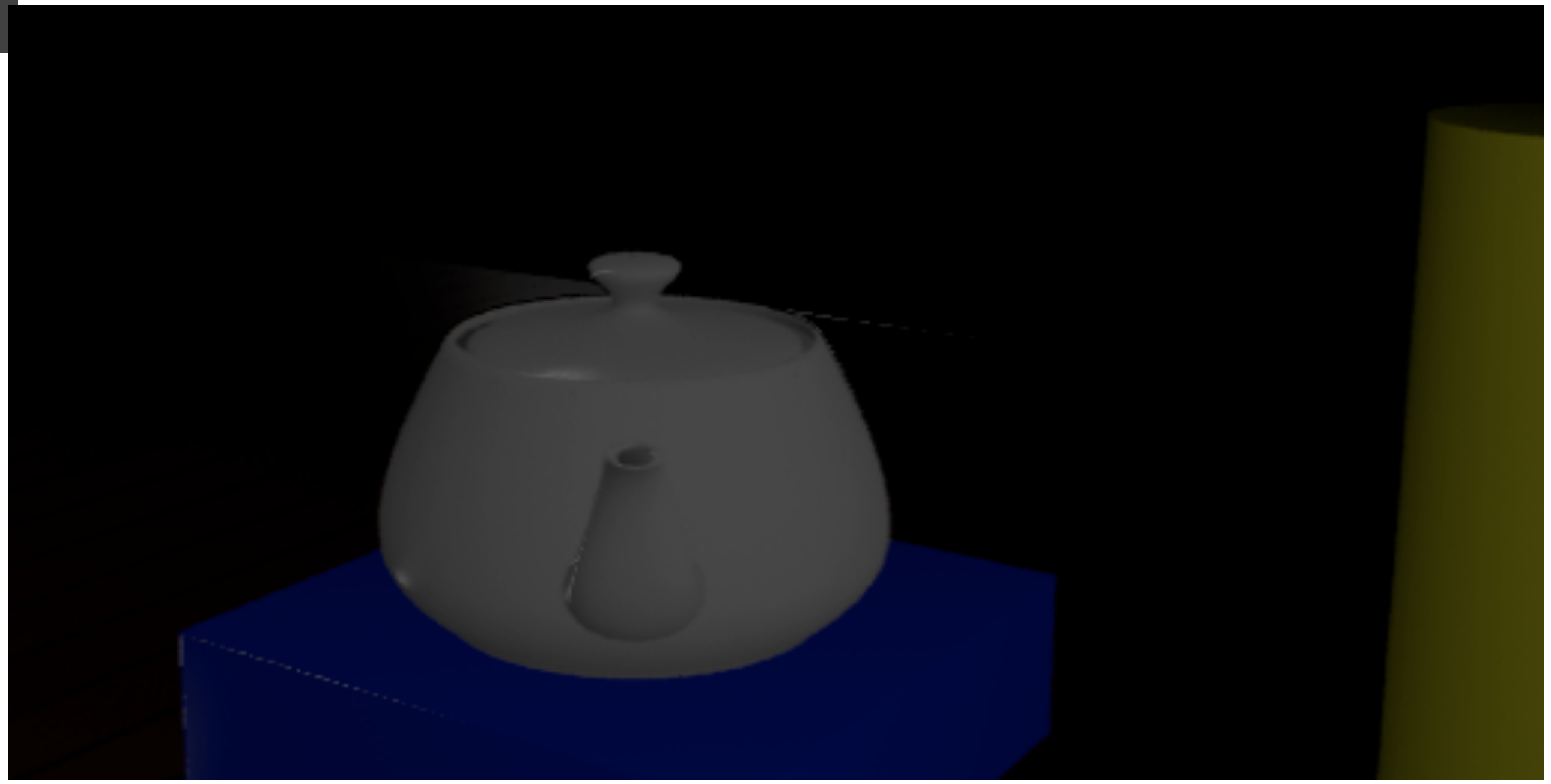


- This pass contains the visible point transformed into world co-ordinates.
- These values can be used for re-lighting / additional lighting in the scene

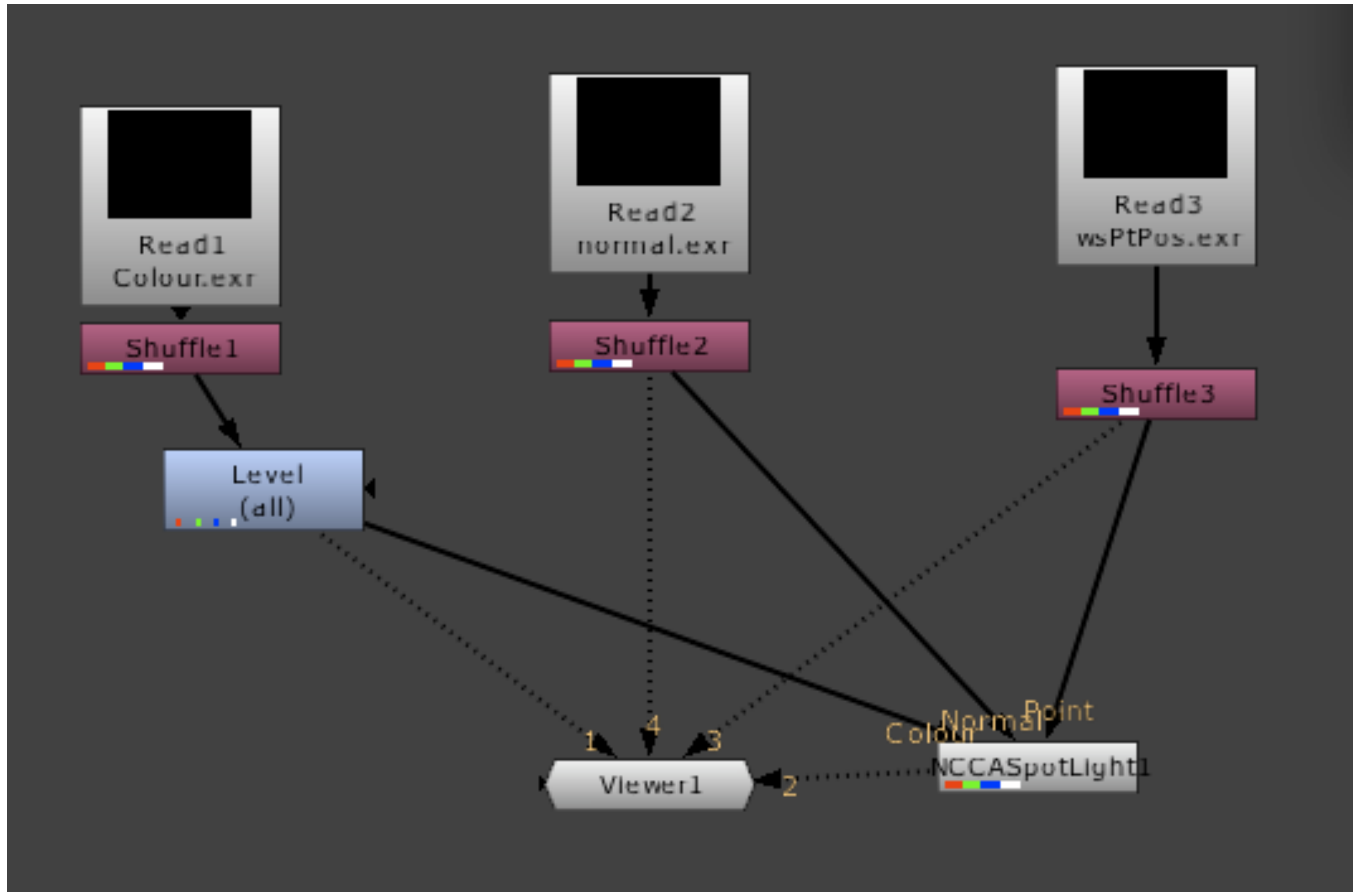
Point re-lighting

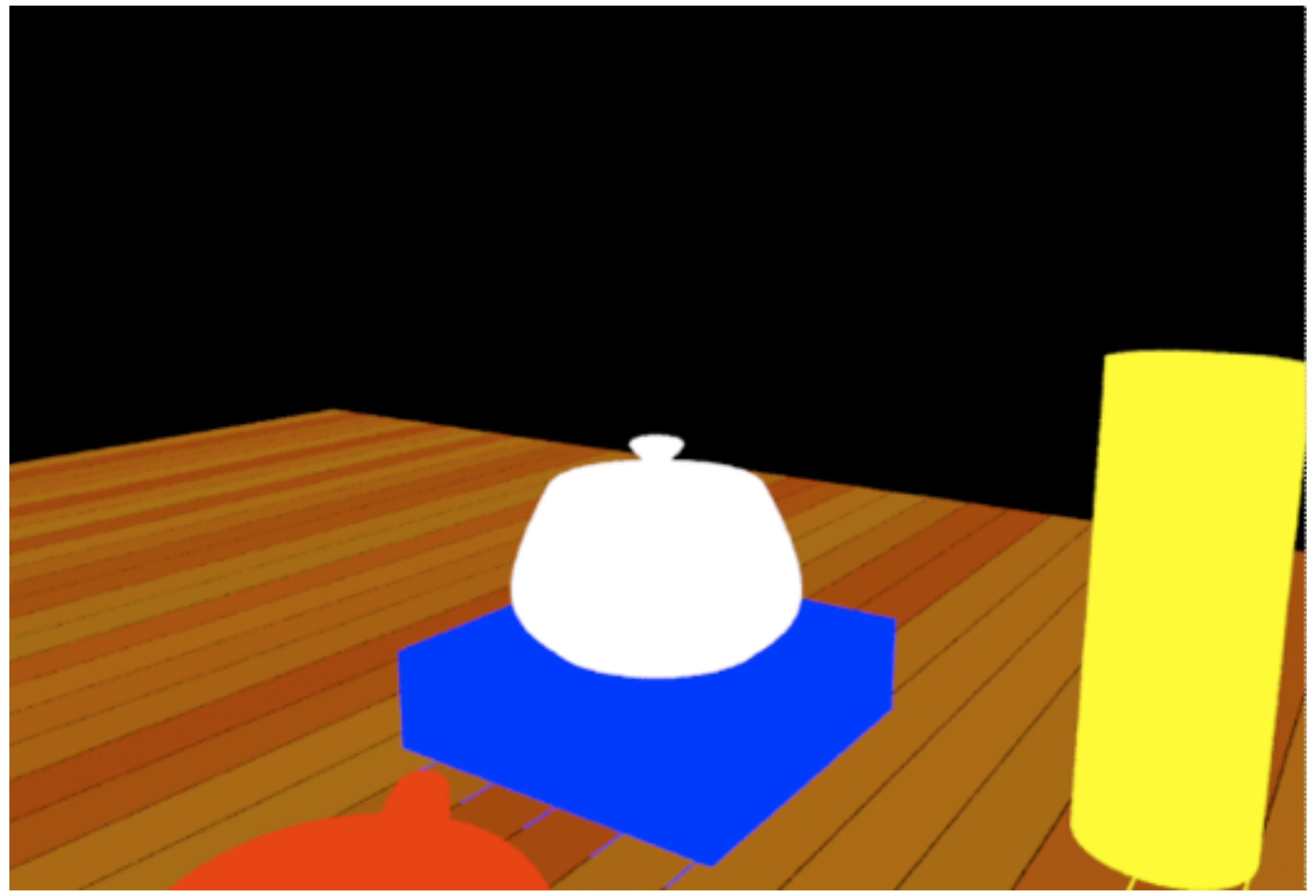


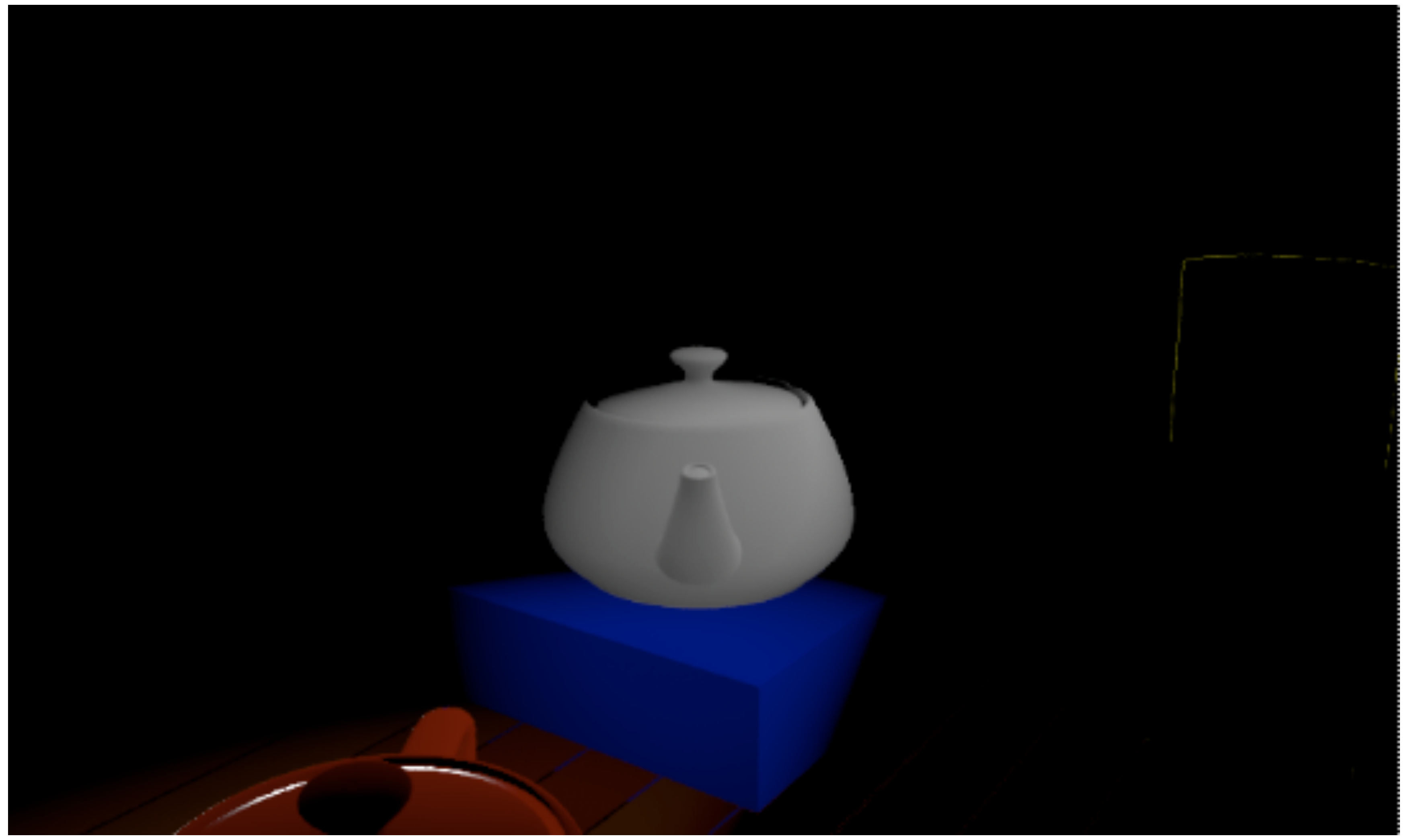




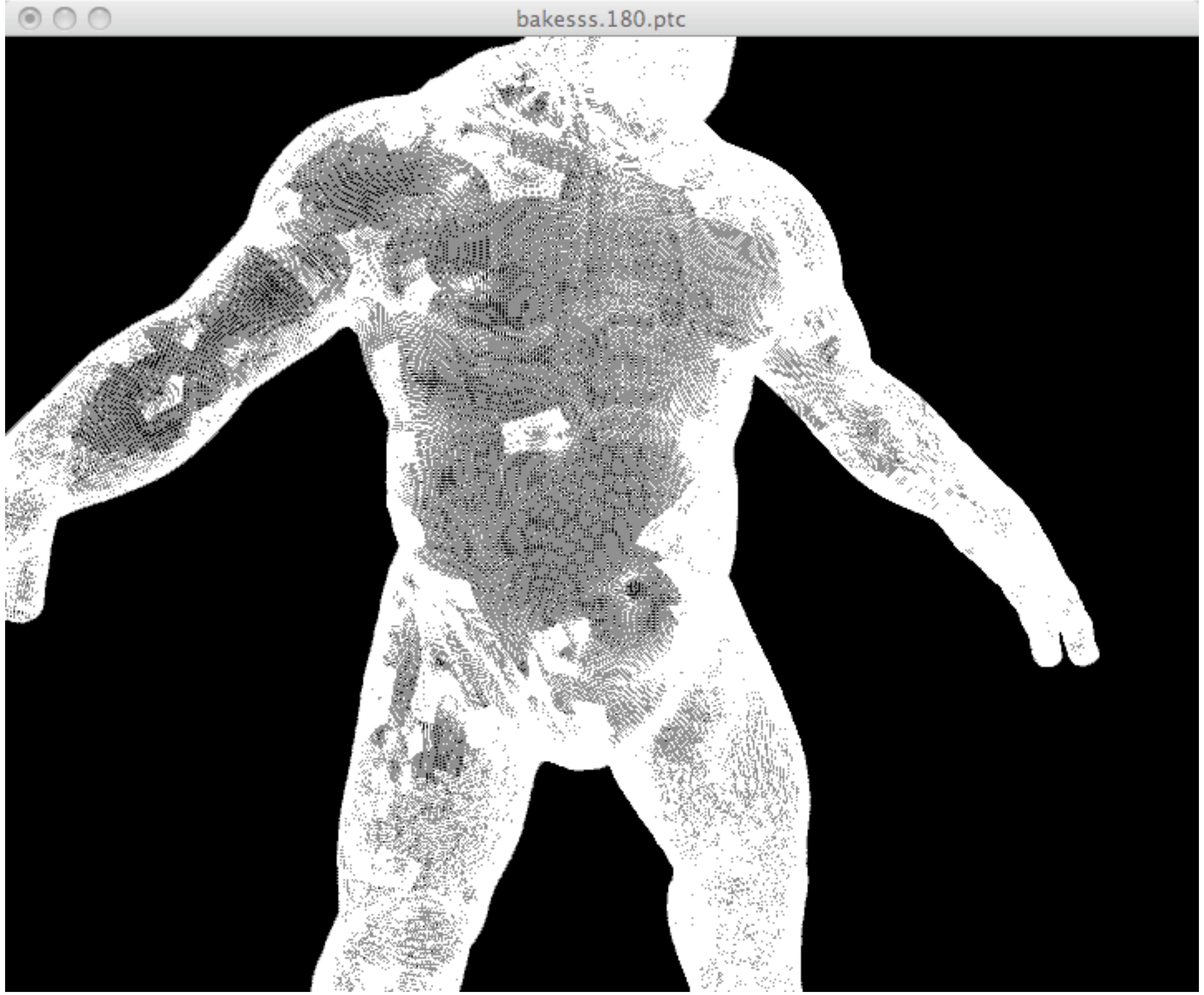
Spot Light


Read1
Colour.exr
Read2
normal.exr
Read3
wsPtPos.exr
Shuffle1
Shuffle2
Shuffle3
Level
(all)
1
4
3
2
Viewer1
Colour
Normal
Point
NCCASpotLight1





Point Clouds



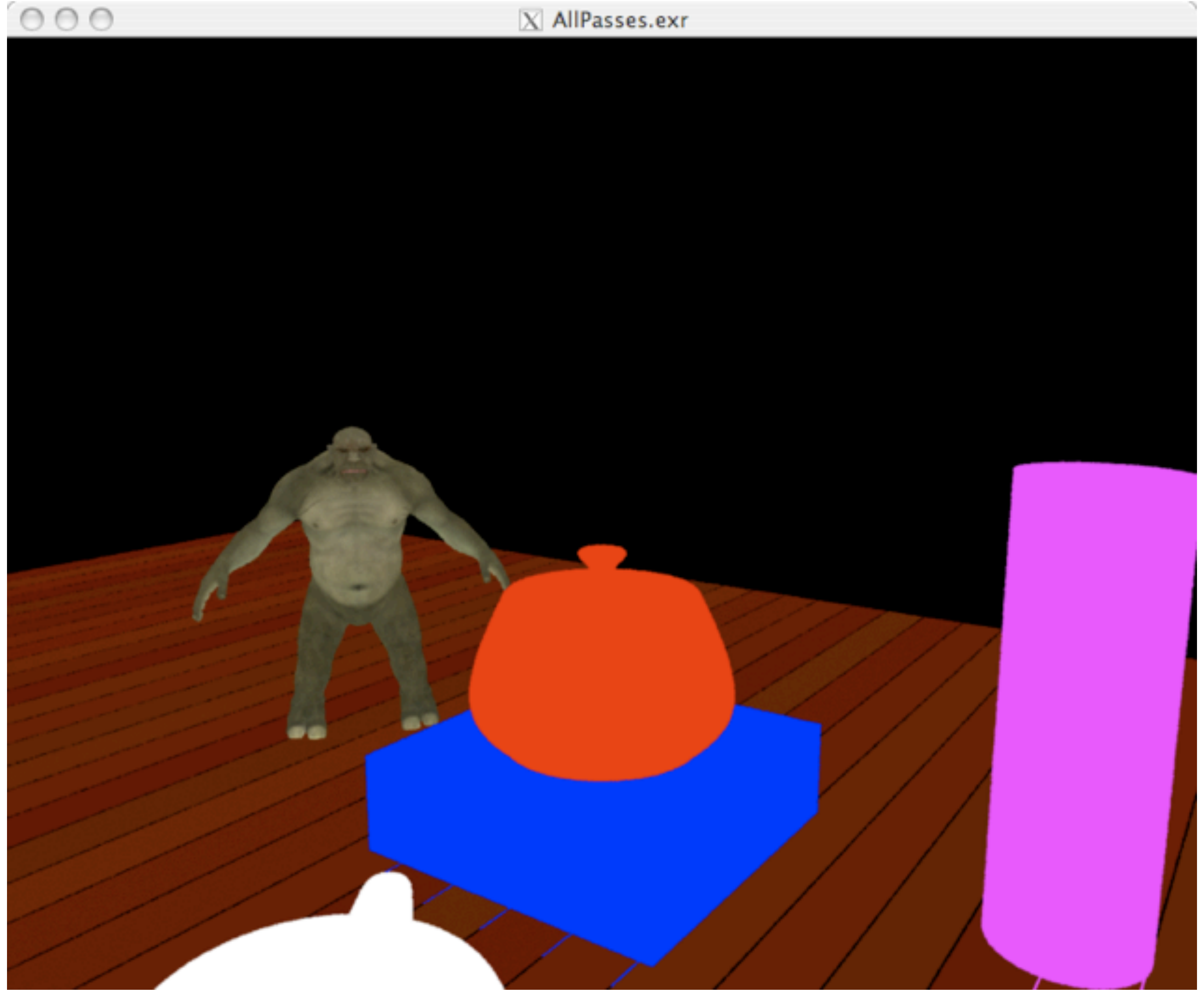
Illumination Models

- Illumination models are used to generate the colour of an object's surface at a given point on that surface.
- The factors that govern the illumination model determine the visual representation of that surface.
- Due to the relationship defined in the model between the surface of the objects and the lights affecting it, illumination models are also called shading models or lighting models.

Light

- The physics of light is a complex subject.
- Shading models are approximations of these laws, in varying levels of realism / complexity.
- This is based on the fact that surfaces, for the most part, are approximations.
- Micro facet details defines the lighting characteristics of surface colour.
- CG Object representation (usually) does not transcend to that level.
- Radiosity /global illumination algorithms mimic photonic reactions with surfaces.

The Simplest Shading Model



- The simplest shading model is a simple constant illumination.
- The model represents an un-realistic assumption in that the surface is self illuminating (the colour of this constant shading is defined by the user).

Illumination Equation

- We can now introduce the *illumination equation*, using the notion that

$$I = value$$

- Where I represents the illumination colour intensity and value represents the expression resulting in that colour value.

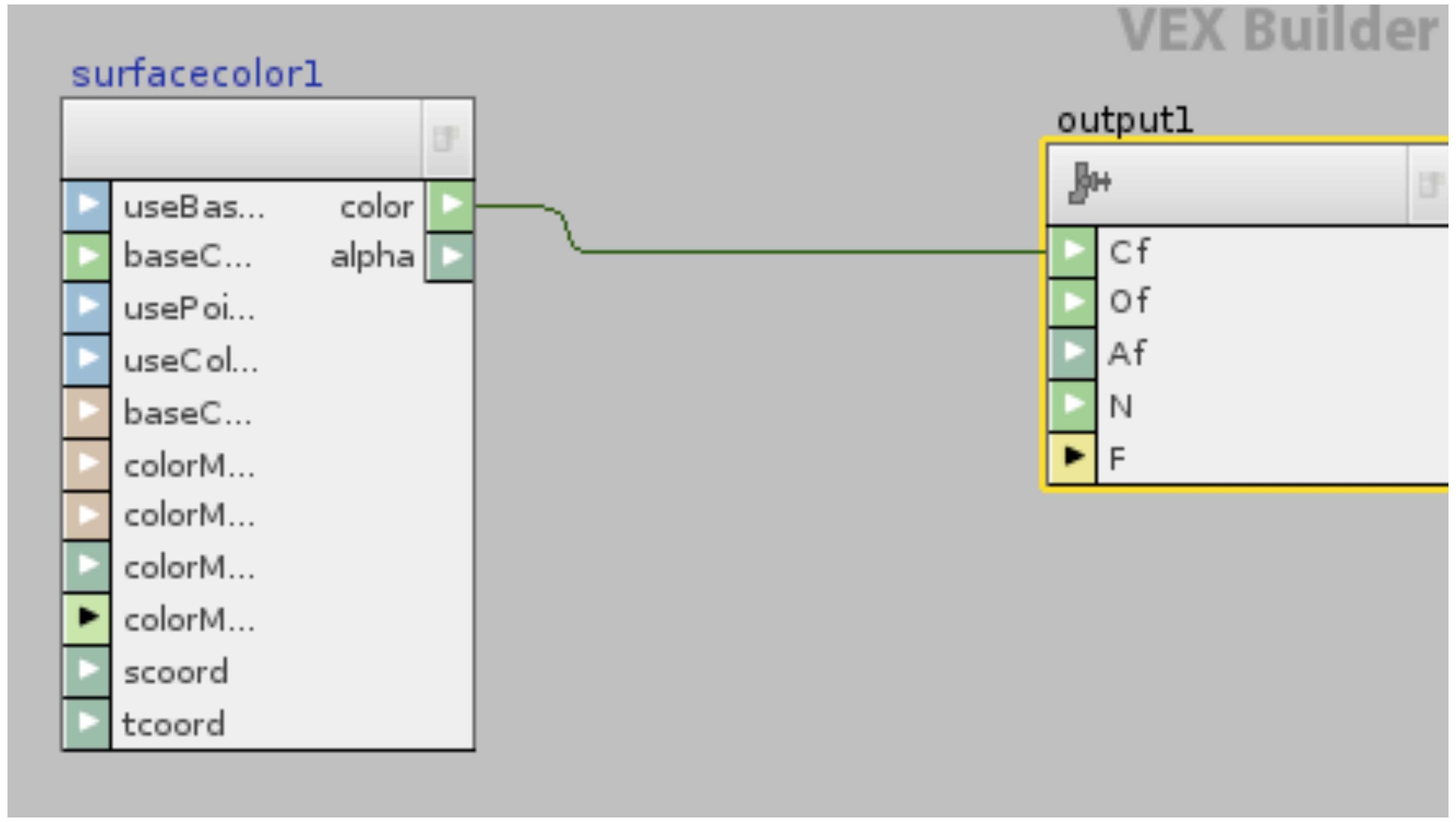
- Constant shading illumination equation can be defined as:

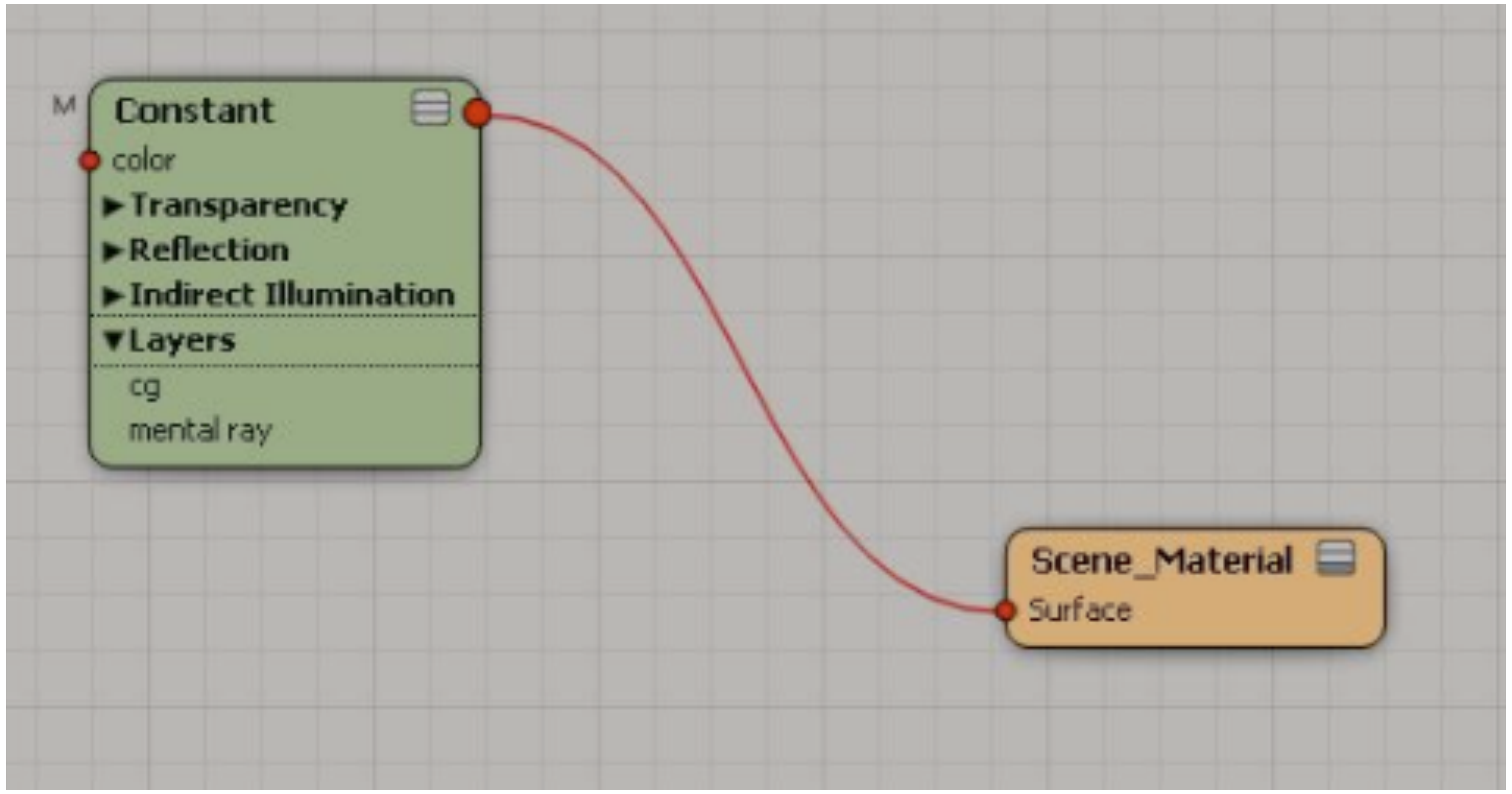
$$I = K_i$$

Constant Shader

- Most shading languages have the concept of an input colour which is passed to the shader to do the illumination
- In renderman this is know as Cs and the final output value for the surface Colour is known as Ci
- In Mental ray and Matra this can be done visually using the shader editors as shown on the next slide

Shader Builders





Renderman

```
surface Constant( string tx="")
{
Oi=Os;

if(tx!="")
   Ci=Oi*texture(tx);
else
   Ci= Oi*Cs;

}
```

Constant Illumination

- This simple equation has no reference to light sources, and as such every point on the object has the same intensity.
- This equation need only be calculated once per object.
- The process of evaluating the illumination equation at one or more points on an object's surface is referred to as lighting the object.
- In the previous example we also use a texture lookup to change the colour of the surface but there is still no lighting

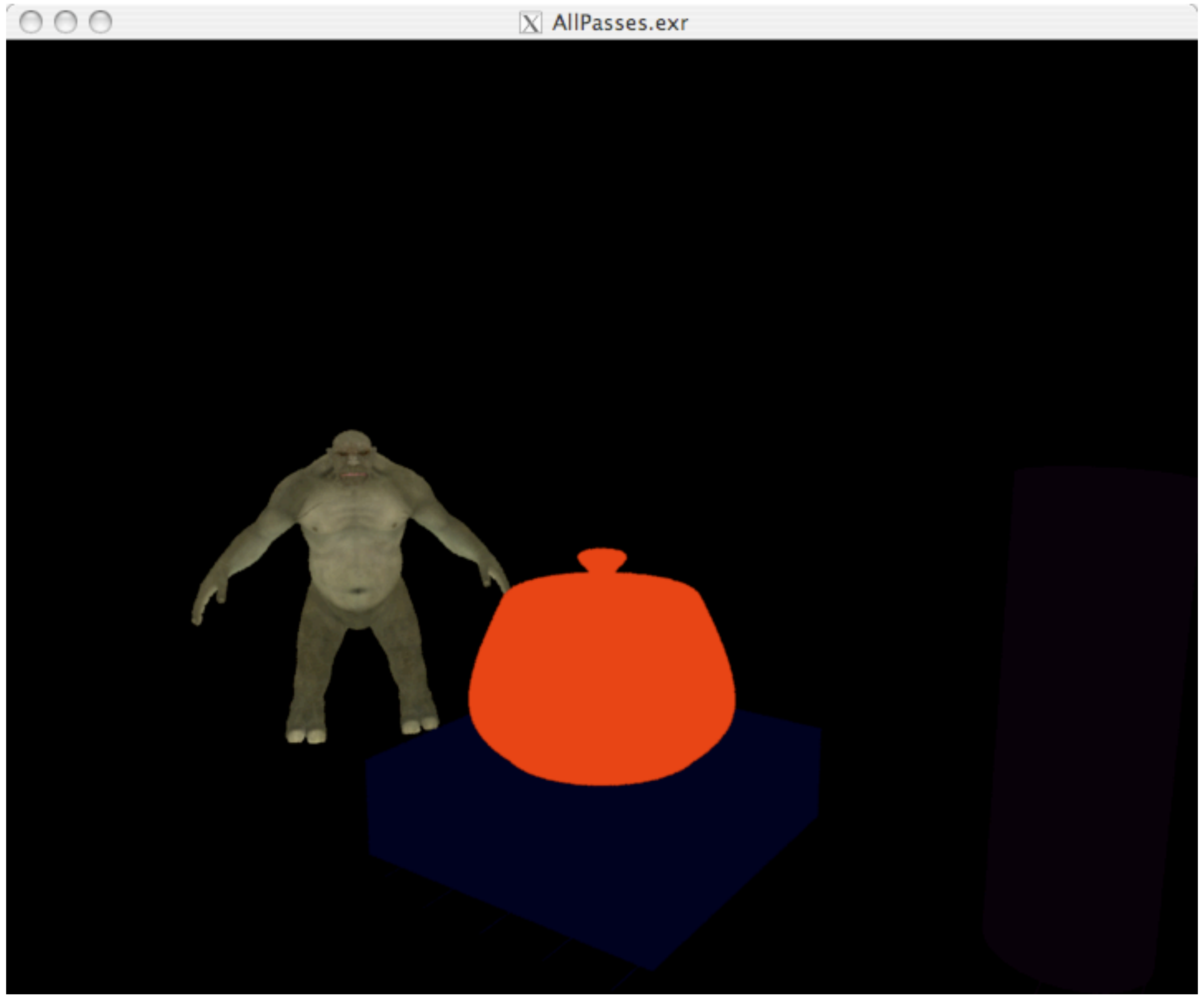
Ambient Light

- Ambient light is usually a scene-based intensity of non directional light.
- It affects every object in that scene.
- We can then incorporate this into our illumination equation

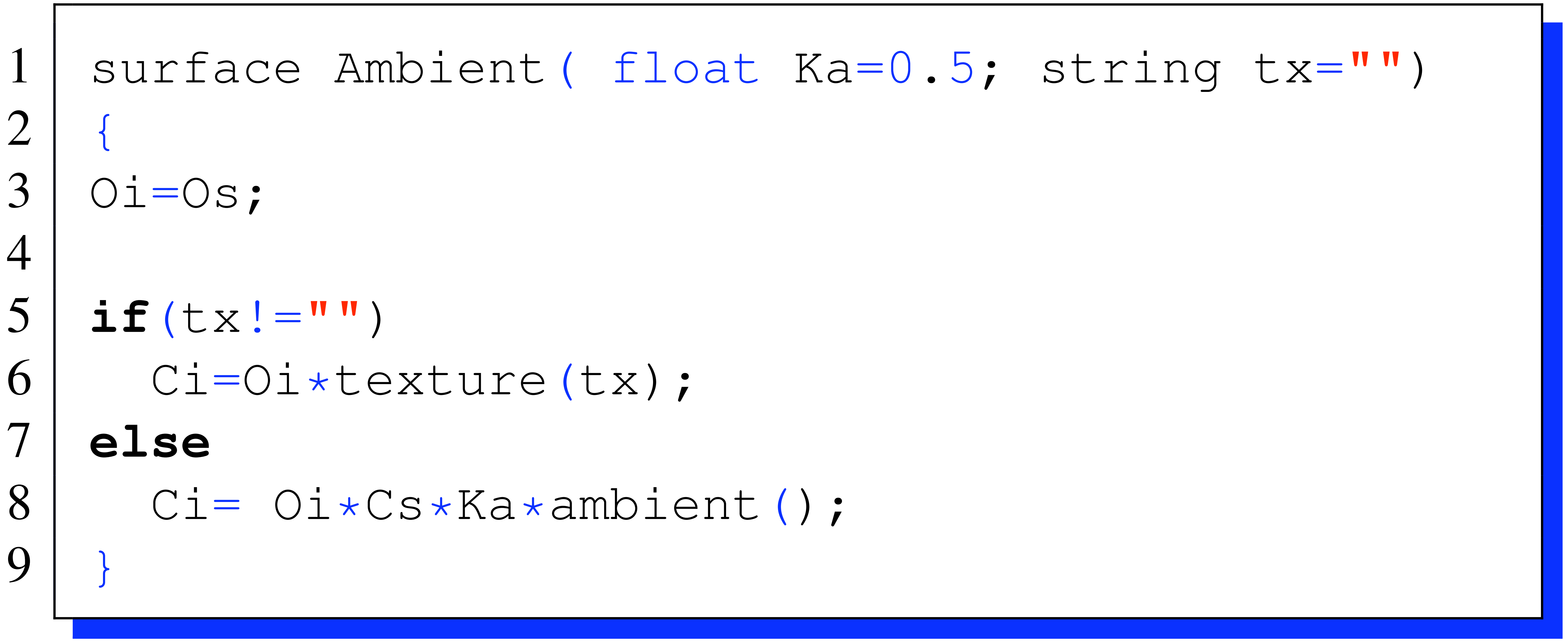
$$I = I_a K_a$$

- I_a is the ambient light intensity the scene-based value that remains the same for all surfaces in the scene.
- K_a is the ambient reflection coefficient a material based property that determines how much ambient light is actually reflected.

Ambient Light



- Material based properties are the properties that characterise one surface from another.
- This equation allows individual surface properties to “reflect” a level of ambient light.
- The scene above has various levels of Ka with a constant value of 0.5 for the ambient light intensity
- The renderman shader is shown on the next page



```
surface Ambient( float Ka=0.5; string tx="")
{
Oi=Os;

if(tx!="")
   Ci=Oi*texture(tx);
else
   Ci= Oi*Cs*Ka*ambient();
}
```

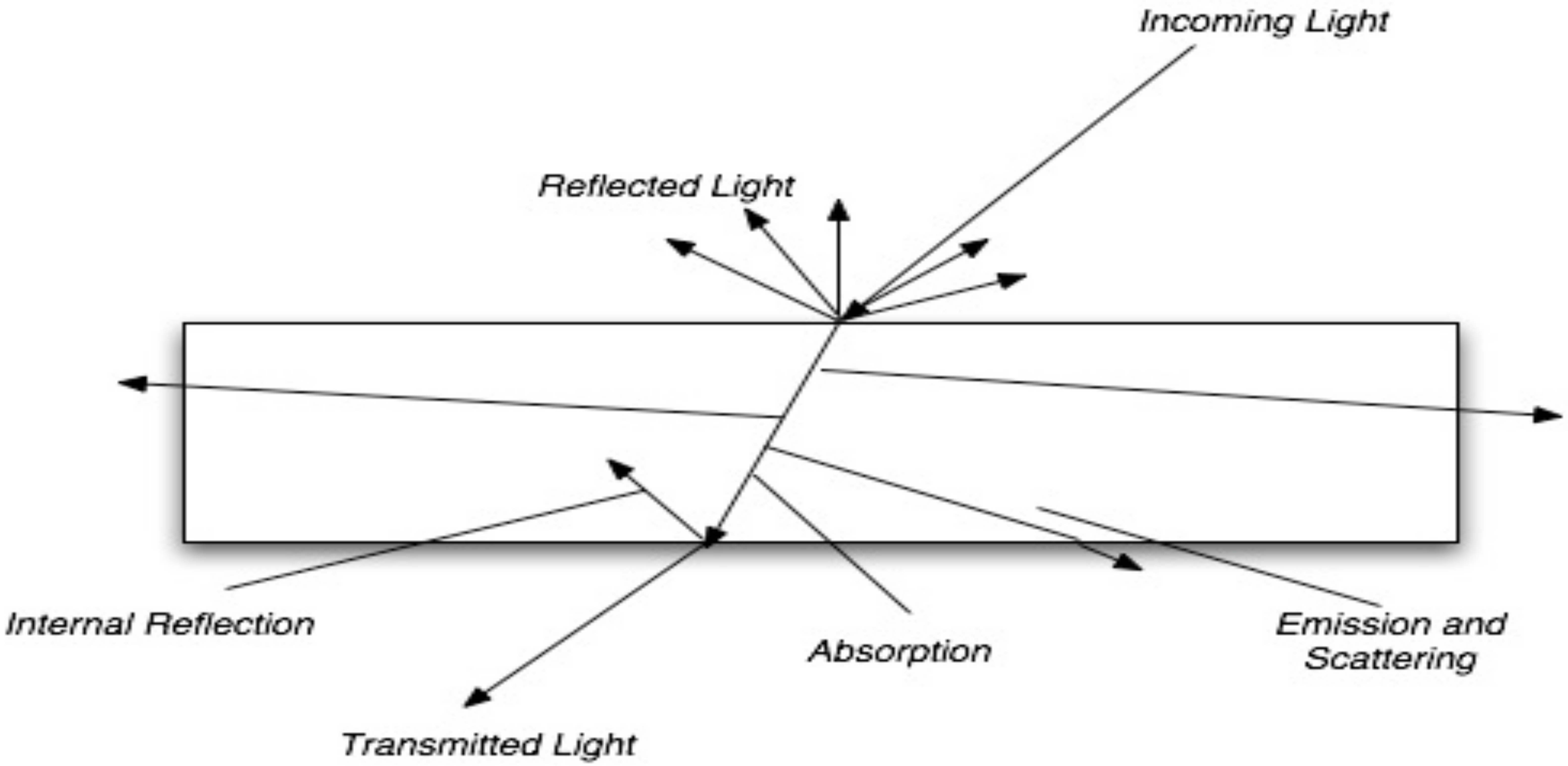
- We make use of the renderman ambient function which returns the level of the ambient light in the scene

- We then multiply this by Ka and the colour value to get us the final colour output

BRDF

- Bidirectional Reflectance Distribution Function
- It describes how much light is reflected when light makes contact with a material (and hence can be used to specify different material types)
- The degree to which light is reflected depends on the viewer and light position relative to the surface normal and tangent
- BRDF is a function of incoming (light) direction and outgoing (view) direction relative to a local orientation at the light interaction point

BRDF


Incoming Light
Reflected Light
Internal Reflection
Absorption
Emission and Scattering
Transmitted Light

BRDF

- Depending upon the Wavelength of the Light there are different levels of absorption, reflection and transmission.
- Therefore BRDF is also dependent upon Wavelength.
- We need to specify the conservation of energy thus
- light incident at surface = Reflected Light + Absorbed Light + Transmitted Light

BRDF

- Position of the light is also important as different areas of the surface will behave differently.
- This can be used to create surface detail, such as the rings and knots in wood.

BRDF

- BRDF is usually specified as a function thus

$$BRDF_\lambda(\theta_i, \phi_i, \theta_o, \phi_o, u, v)$$

- λ is the wavelength of the light
- $\theta_i \phi_i$ is the incoming light in Spherical Co-ordinates
- θ_o, ϕ_o is the outgoing reflected light in Spherical Co-ordinates
- u,v is the position of the surface in parametrised in texture co-ordinate space

BRDF

- If there is no positional invariance then the BRDF may be specified thus

$$BRDF_\lambda(\theta_i, \phi_i, \theta_o, \phi_o)$$

- This only works for materials with no surface variation (Homogeneous) and can be speeded up using lookup tables and texture modulation

BRDF Definition

$$BRDF = \frac{L_o}{E_i}$$

- W_i incoming light direction
- W_o reflected light direction (outgoing)
- $L_o$quantity of reflected light in direction W_o
- E_i quantity of light arriving from direction W_i

Types of BRDF

- There are two main types of BRDF
- isotropic BRDFs and anisotropic BRDFs
- The important properties of BRDFs are reciprocity and conservation of energy
- BRDFs that have these properties are considered to be physically plausible

Isotropic BRDF

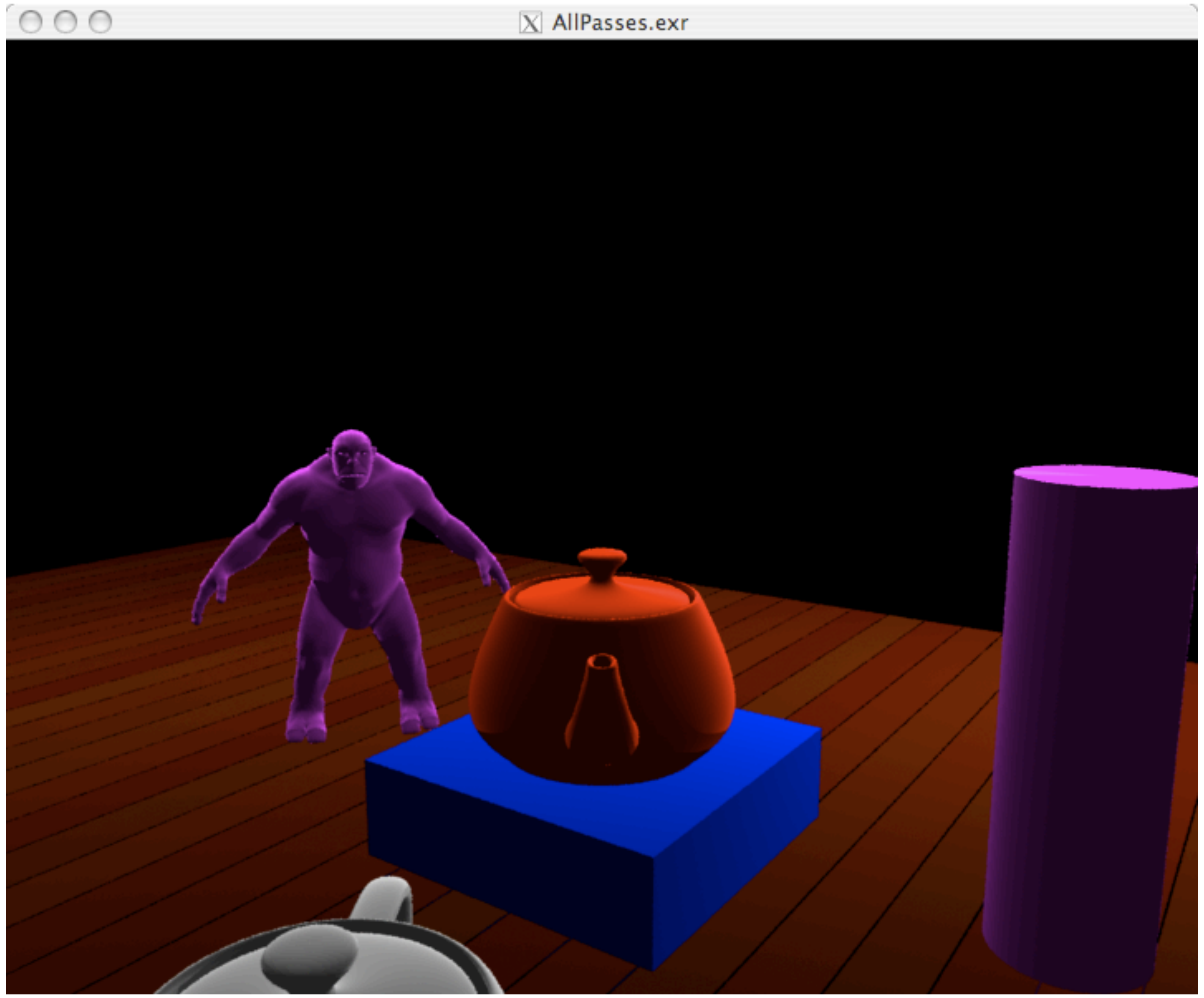
- These type of surfaces are invariant to with respect to rotation around the surface normal
- effectively this means that it should respond the same way from wherever we view it.
- and are quicker to calculate from the point of view of the render

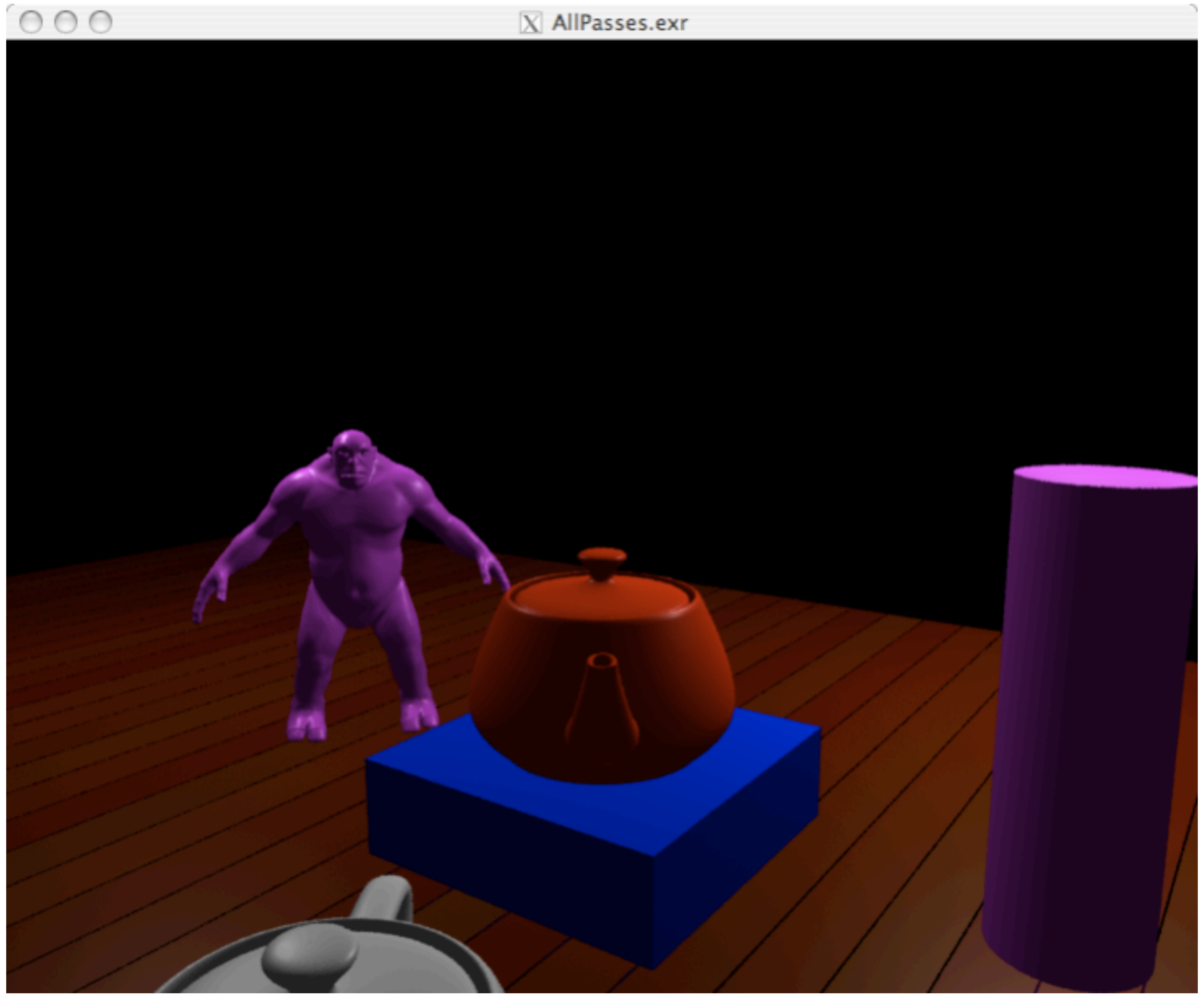
Anisotropic BRDF

- BRDFs that describes reflectance properties that do exhibit change with respect to rotation of the surface around the surface normal vector
- Anisotropy (the opposite of isotropy) is the property of being directionally dependent.
- Something which is anisotropic may appear different or have different characteristics in different directions.
- Seen in a materials such as Velvet

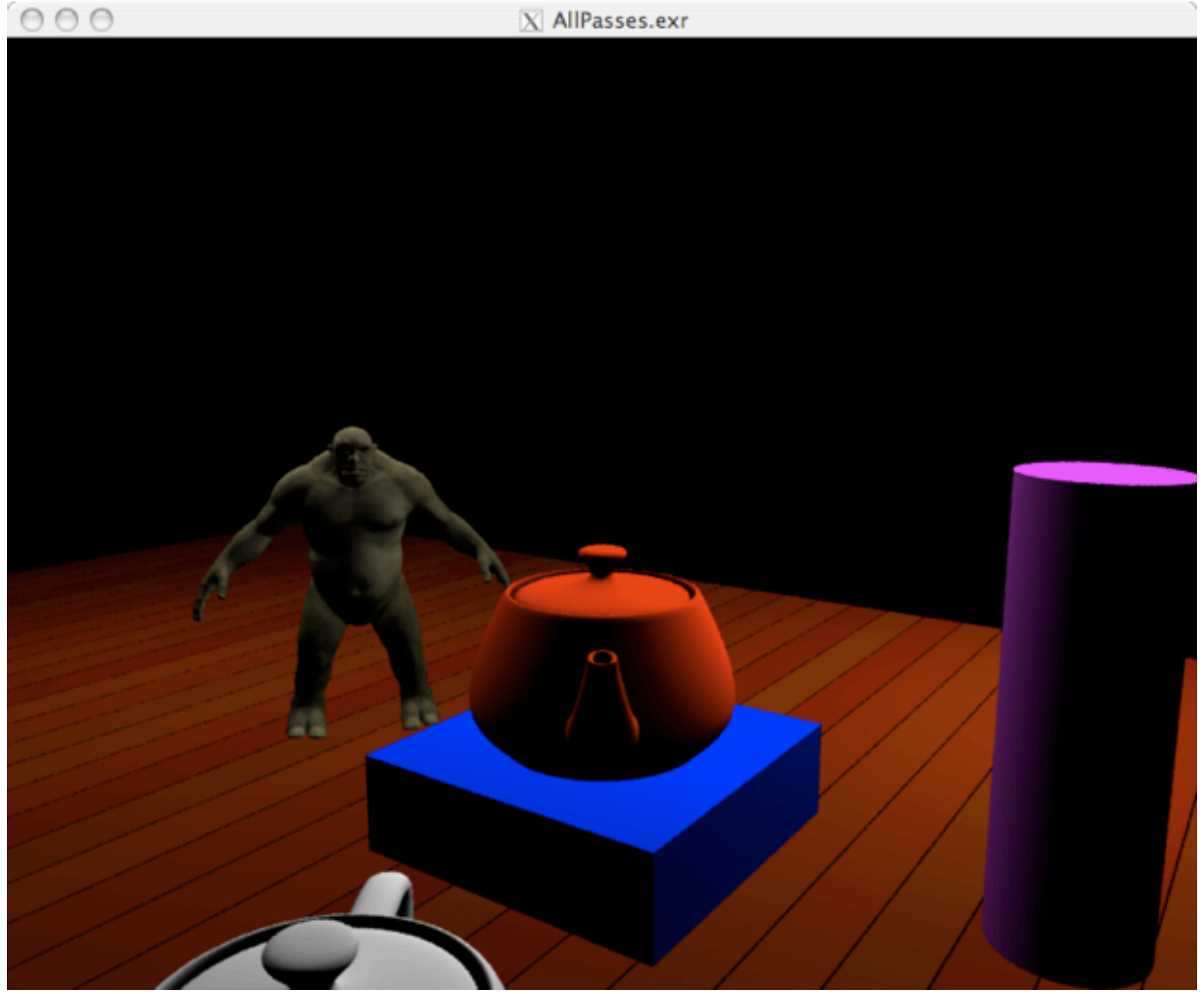
Anistropic

isotropic





Diffuse Reflection



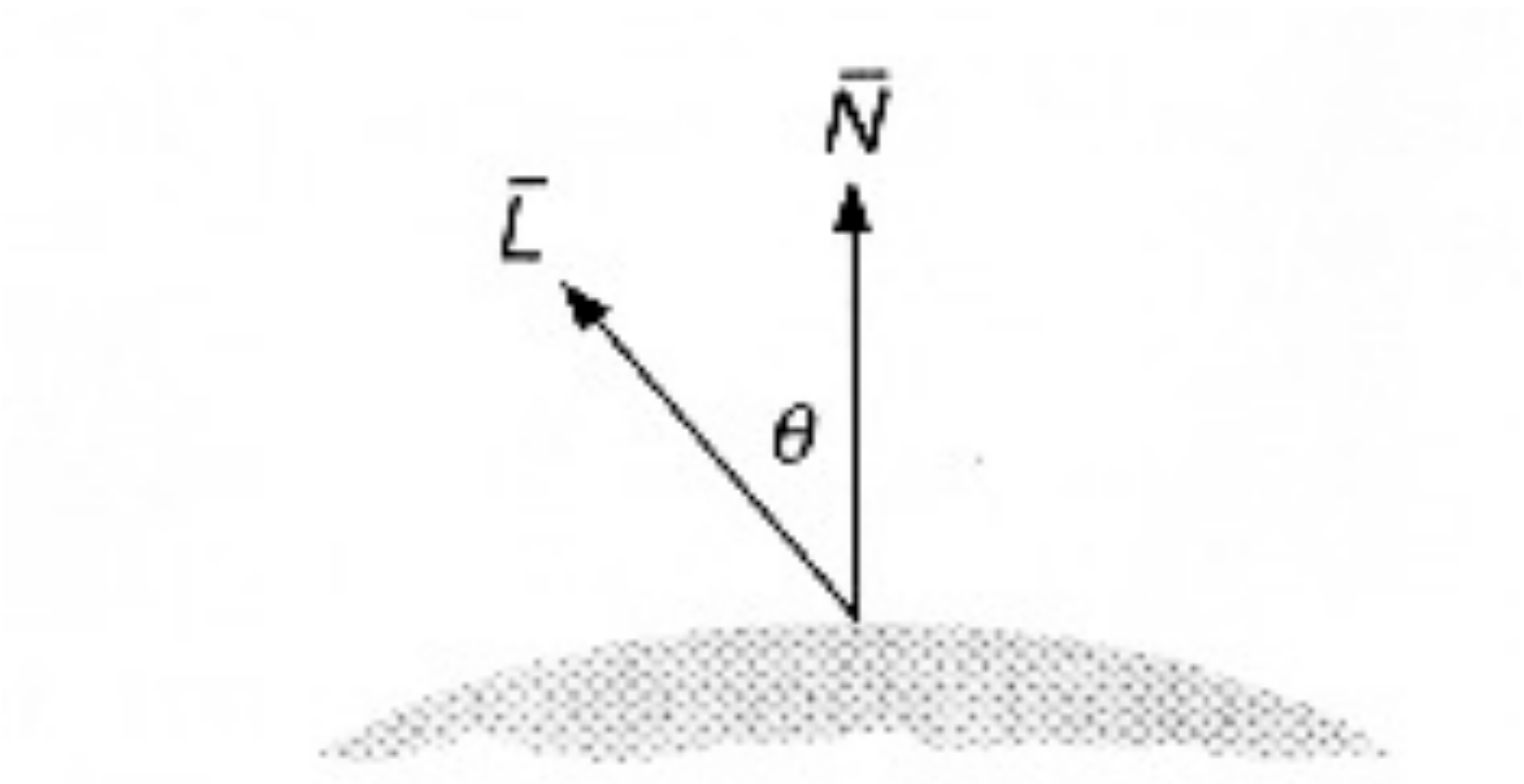
- Diffuse reflections consider point lights to generate shading properties
- a change in colour intensity across the surface of an object in relation to light sources.

Diffuse Reflection

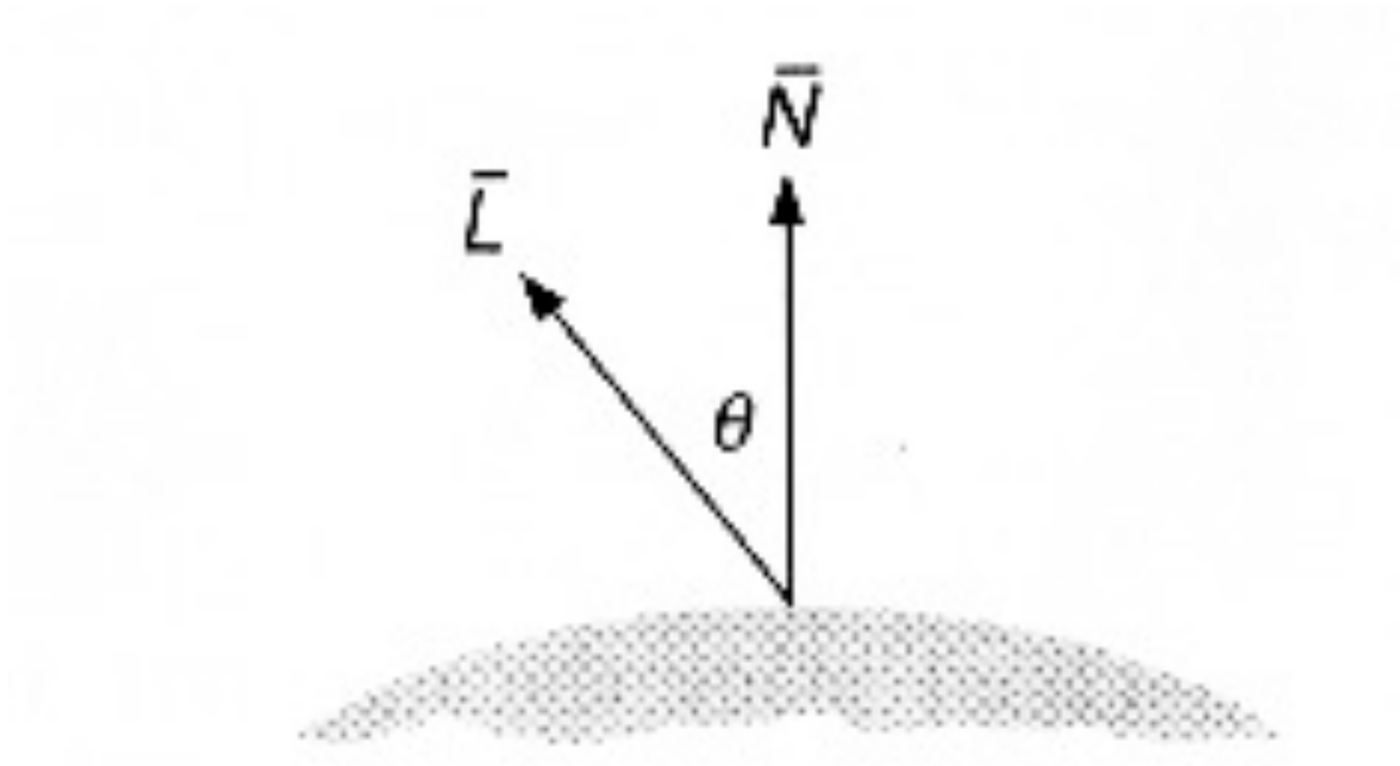
- The simplest of the diffuse models is the Lambert Illumination Model.
- In Lambertian Reflection light is reflected with equal intensity in all directions (isotropic).
- The distribution is a basic consideration towards surface detail:
- Light scattering on the surface and in the medium.

Lambert’s Law

- Lambert’s law states :
- that the intensity of illumination on a diffuse surface is proportional to the cosine of the angle generated between the surface normal vector and the surface to the light source vector.



Lambert's Law



- The only data used in this equation is the surface normal and a light vector that uses the light source position(taken as a point light for simplicity).
- The intensity is irrespective of the actual viewpoint, hence the illumination is the same when viewed from any direction.

Lambert's Model

The equation for Lambert's illumination model is:

$$I = I_p k_d \cos(\theta)$$

Where:
I_p is the ***intensity*** of the point light source
k_d is the material ***diffuse reflection coefficient*** the amount of diffuse light reflected.

By using the dot product between 2 vectors ***v_1*** and ***v_2***

$$v_1 \bullet v_2 = |v_1||v_2|\cos(\theta)$$

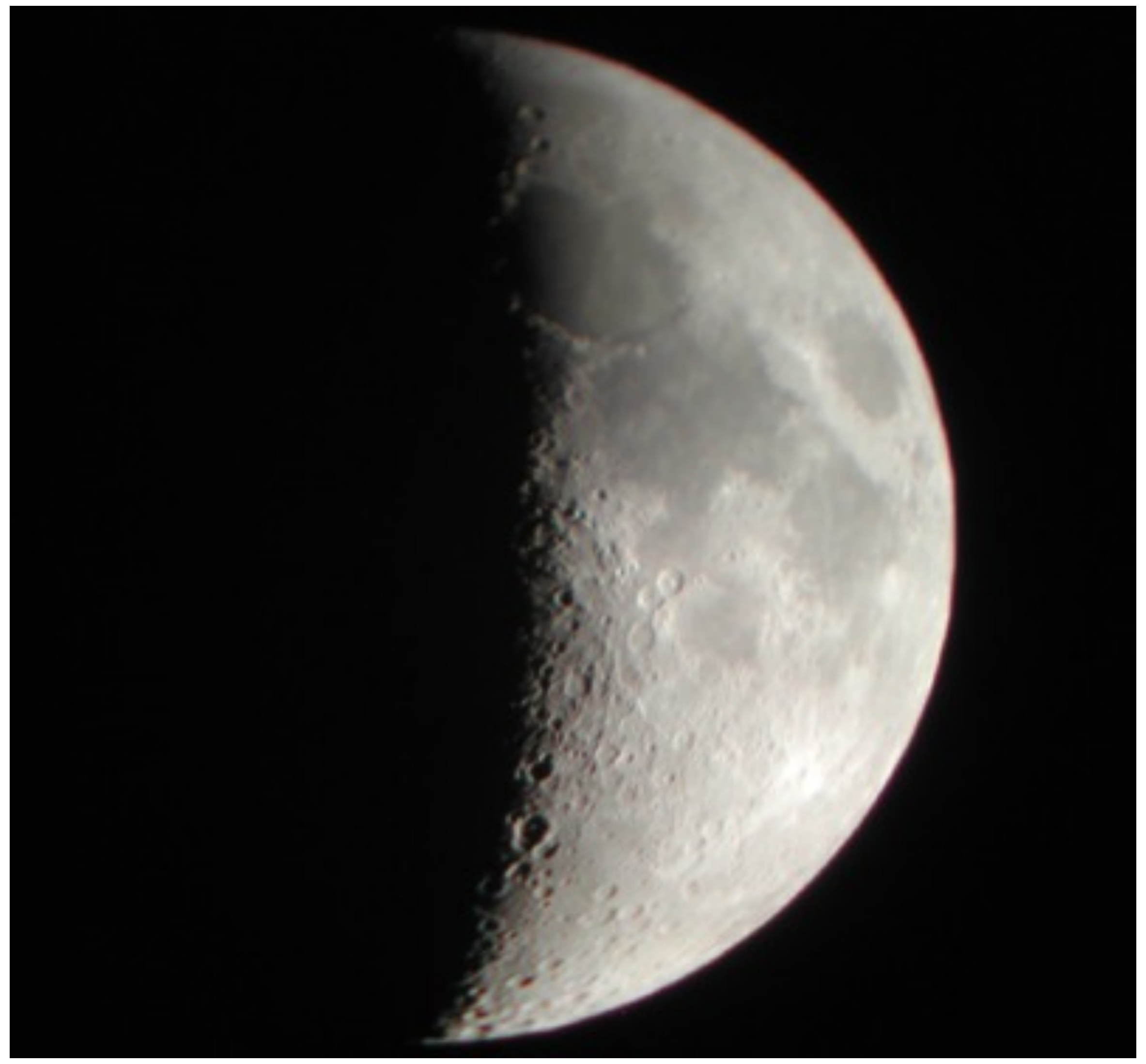
if ***N*** and ***L*** are normalised, we can re-write the illumination equation:

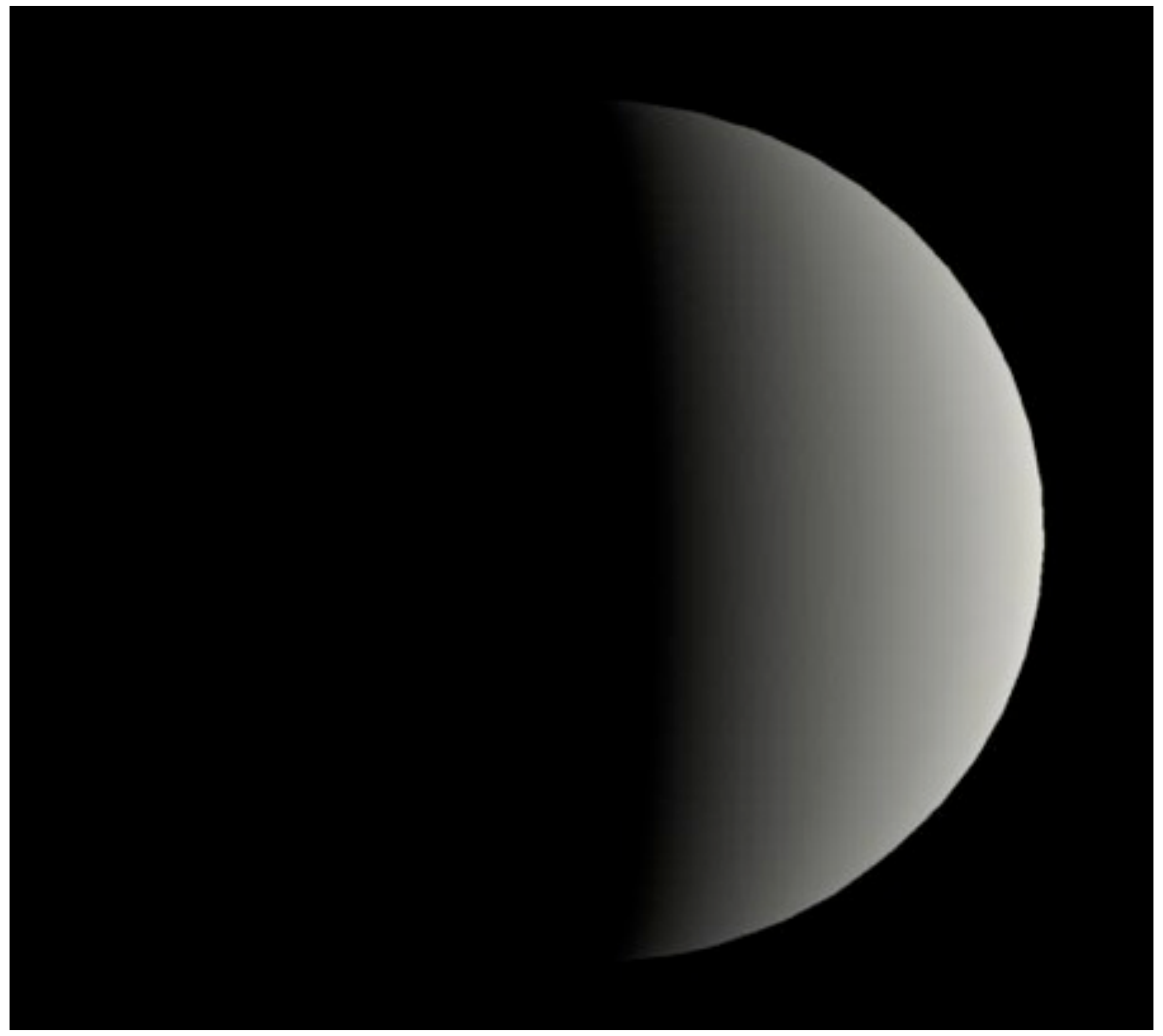
$$I = I_p k_d (N \bullet L)$$

Renderman Version

```
surface Diffuse
(
 float Kd=1;
 string tx="";
)
{
/* init the shader values */
normal Nf = faceforward(normalize(N),I);
vector V = -normalize(I);

/* now calculate the shading values */
color Diffuse=0;
illuminance( P , Nf, PI/2.0 )
{
  Diffuse += Cl * normalize(L).Nf;
}

color Ct=Cs;
// use image based textures
if(tx!="")
{
  Ct=Oi*texture(tx);
}
// calculate the colour
Ci= Oi*Ct*(Kd*Diffuse);

}
```

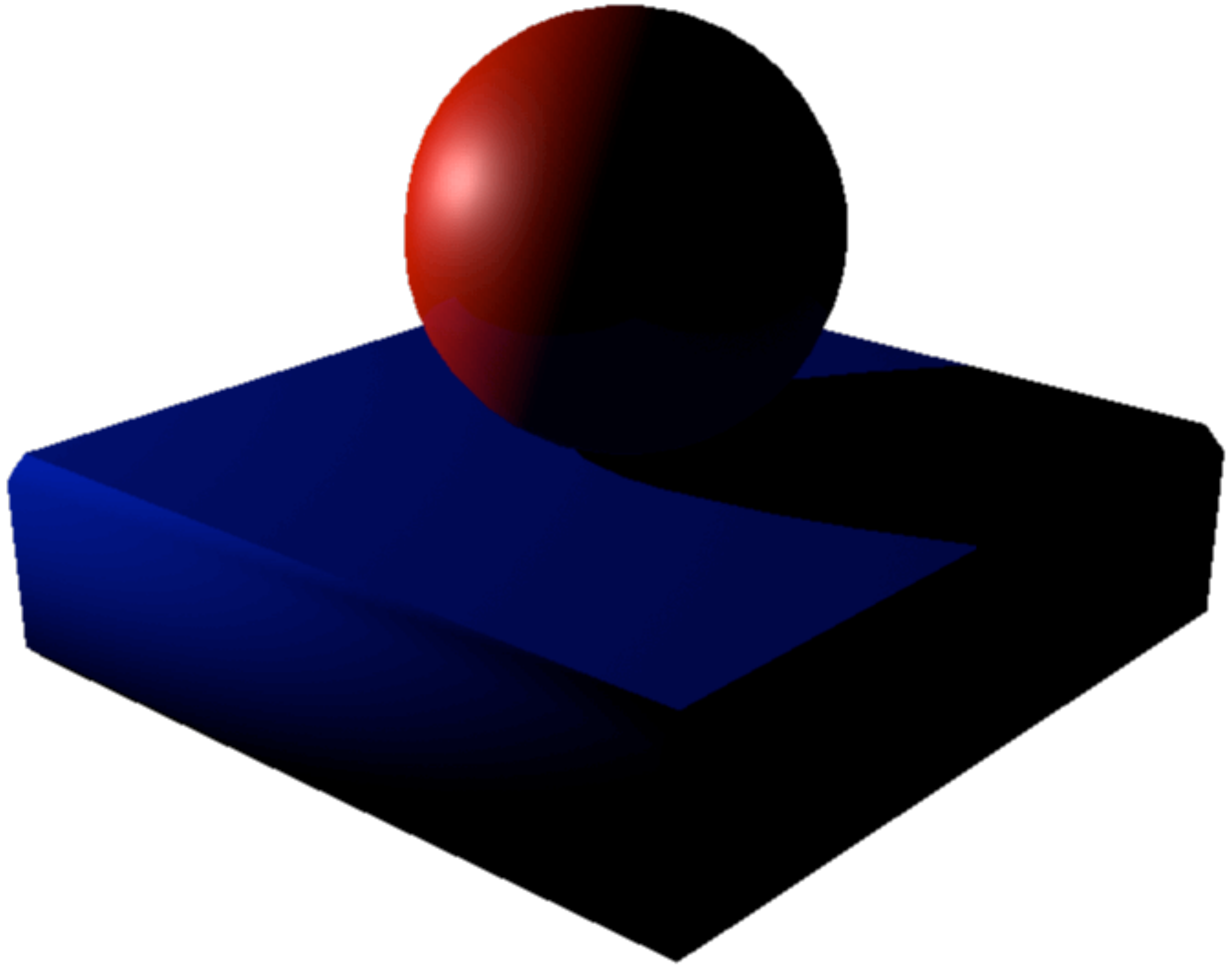
Problem with Lambert





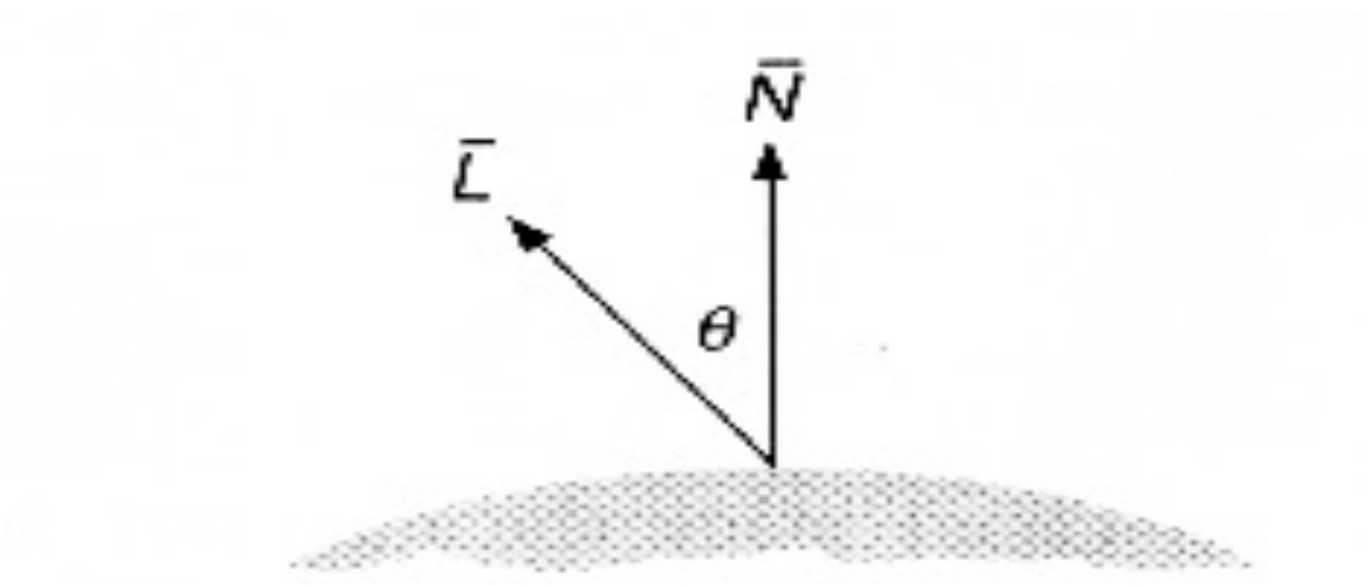
- You can see how Lambert's law is too general

Problems with Lambert

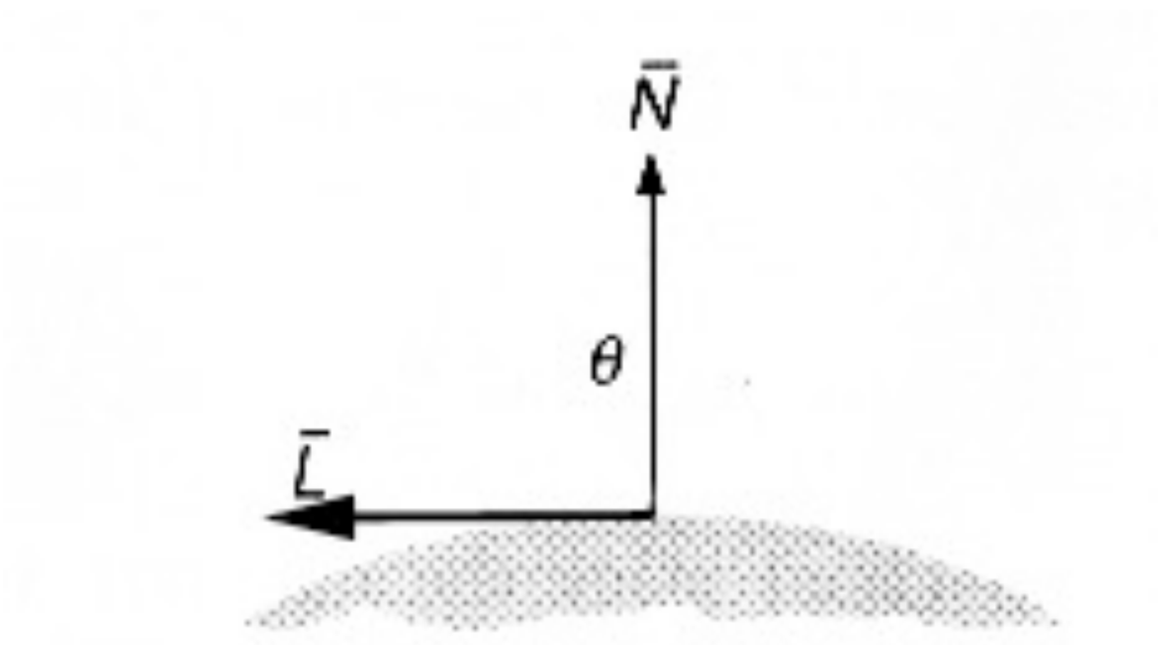


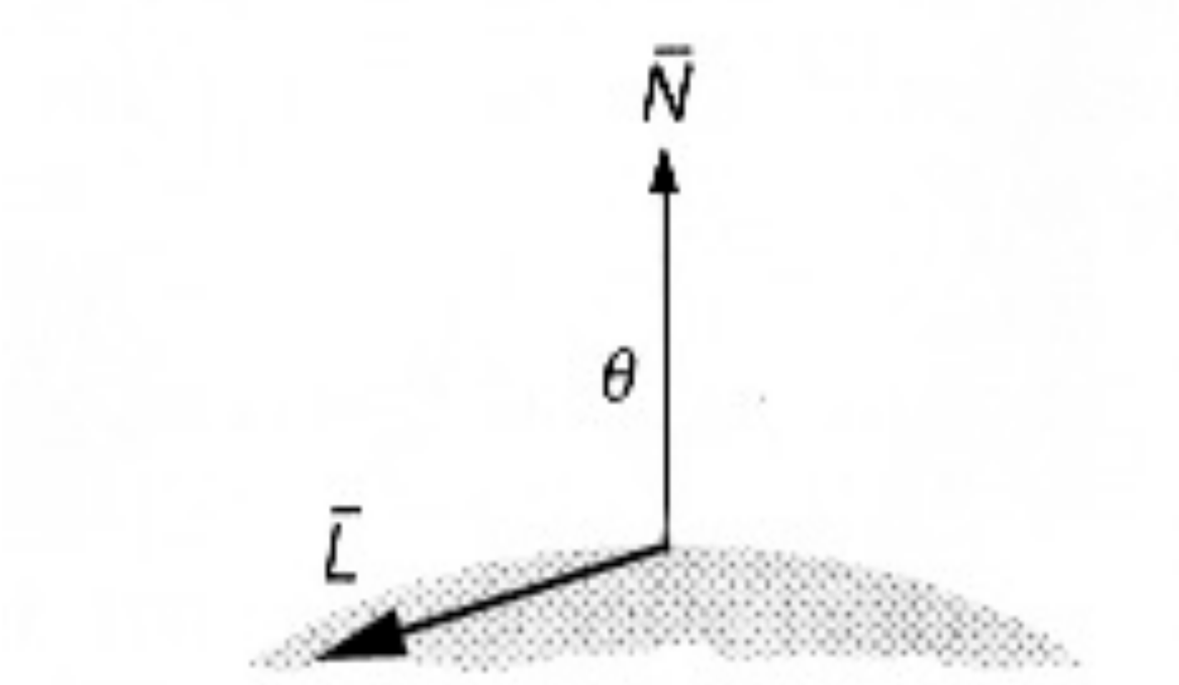
Occlusion

$$I = I_p k_d (N \bullet L)$$



$$I = I_p k_d \max(N \bullet L, 0)$$





both result in a value of 0

Lights

- In general lights that we add to a scene in an animation package are not physically based.
- They are usually general approximations of a lighting function provided as part of the shading functions
- Usually Lights are in fact just other shaders which are queried as part of the rendering pipeline to see what light colour energy is to be provided to the surface when calculating the illumination equations

Ambient Light

- The ambient parameter in most programs is an unrealistic effect (as explained by the example earlier)
- In real life ambient light is a widely distributed "indirect" light that has bounced off (or been transmitted through) objects in your scene.
- In CG programs ambient means a flat, uniform brightness added to all elements of the scene with no attempt at diffusion or scattering.

Point (omnidirectional) Lights

- A Point light simulates rays shining out from one infinitely small point in space
- Usually the have a notional value of being from some position in the scene and will illuminate in all directions
- This is quite an unrealistic lighting model as in real life most lights will cast light more in one direction than others.

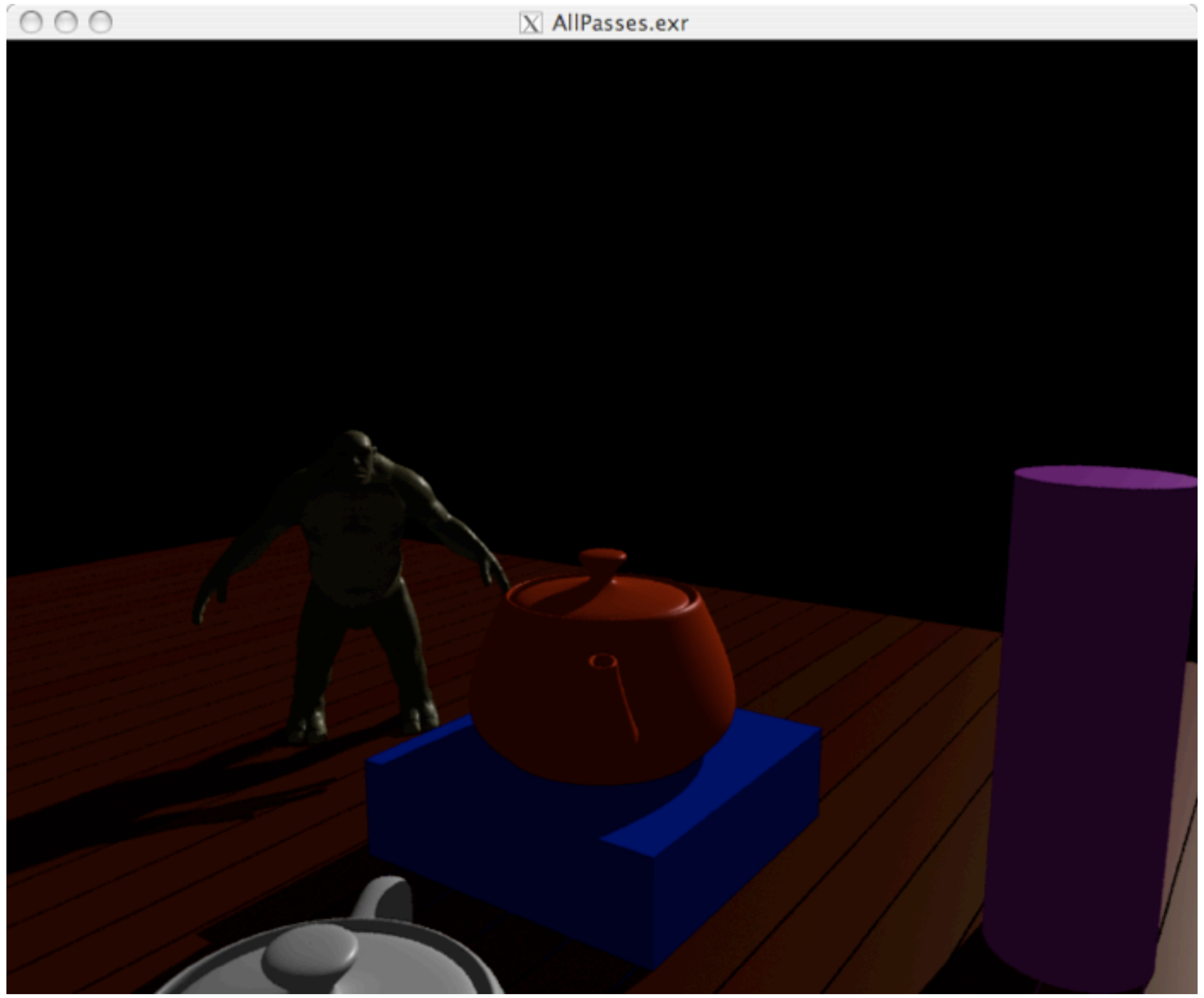
Point Light Shader

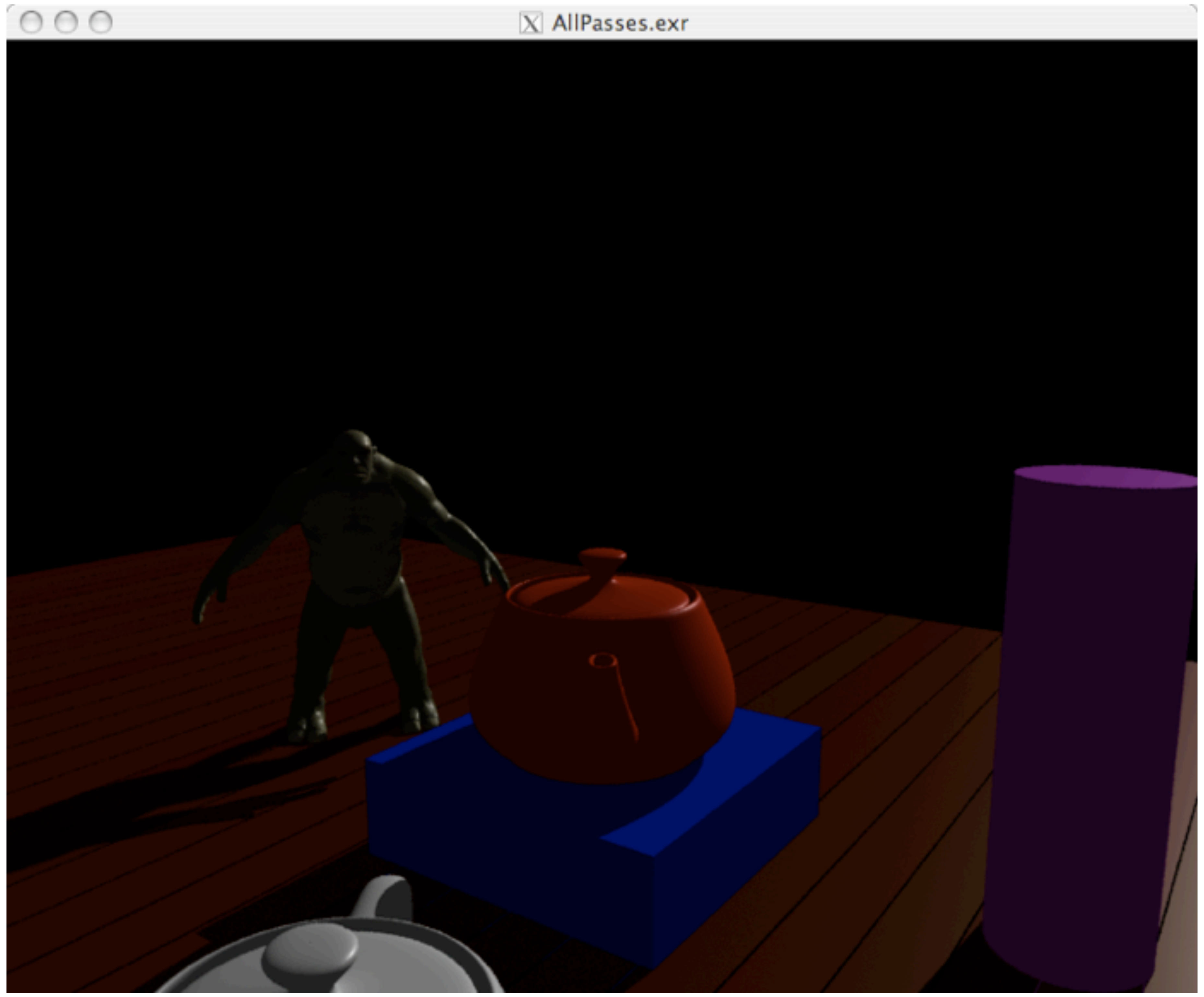
- The following is a renderman pointlight shader it gathers all the illumination from the current position

```
light
pointlight(
    float intensity = 1;
    color lightcolor = 1;
    point from = point "shader" (0,0,0);
)
{
    illuminate( from )
  Cl = intensity * lightcolor / L.L;
}
```

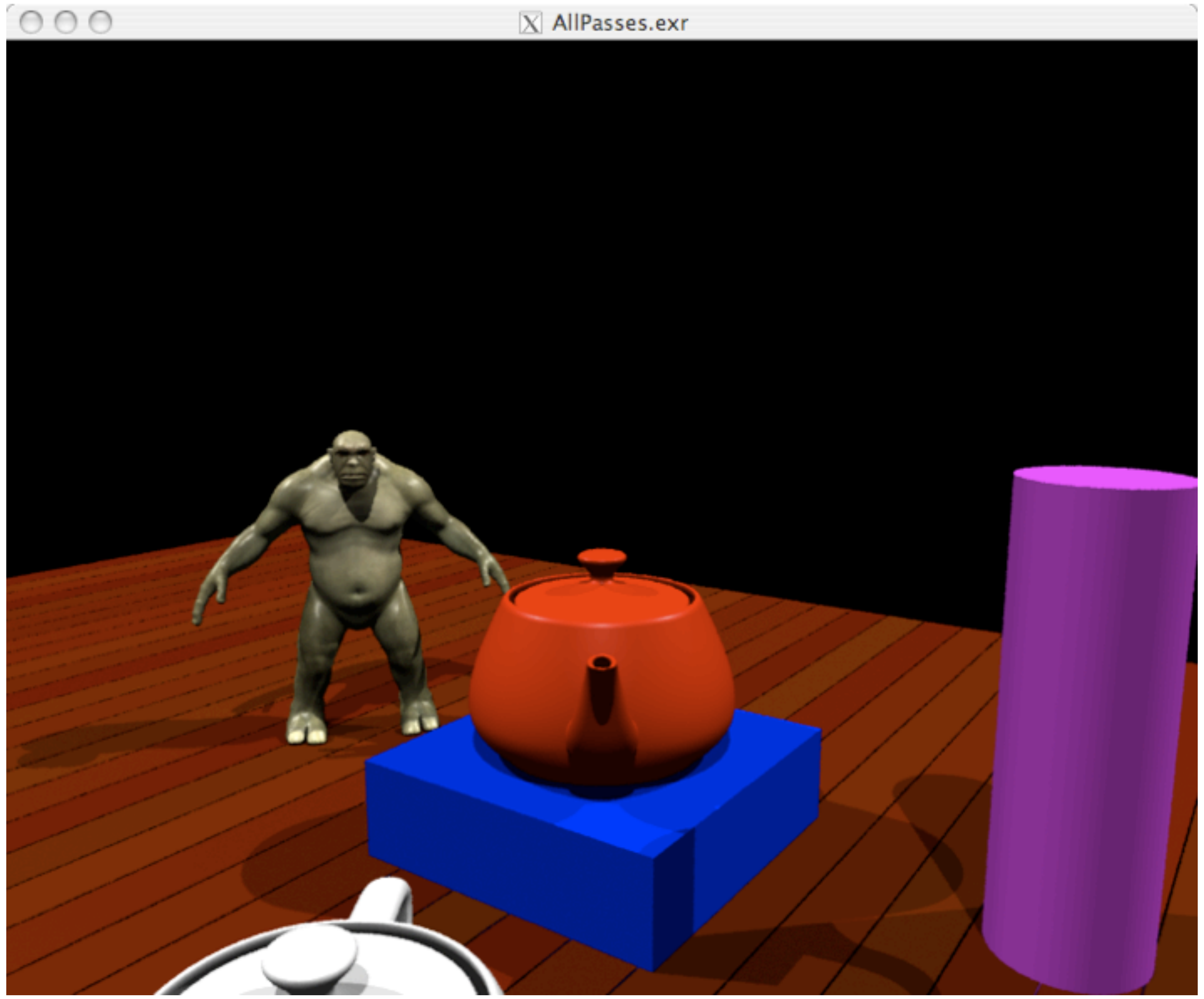
distant lights

- A directional or distant light sets a single vector for all it's illumination
- It hits every object in the scene from the same angle no matter where the object is located
- The position of the light from the object has no bearing on the illumination as only the direction vector is used
- Point lights can be specified using a to and from parameter but this is just used to calculate the direction vector by subtracting them



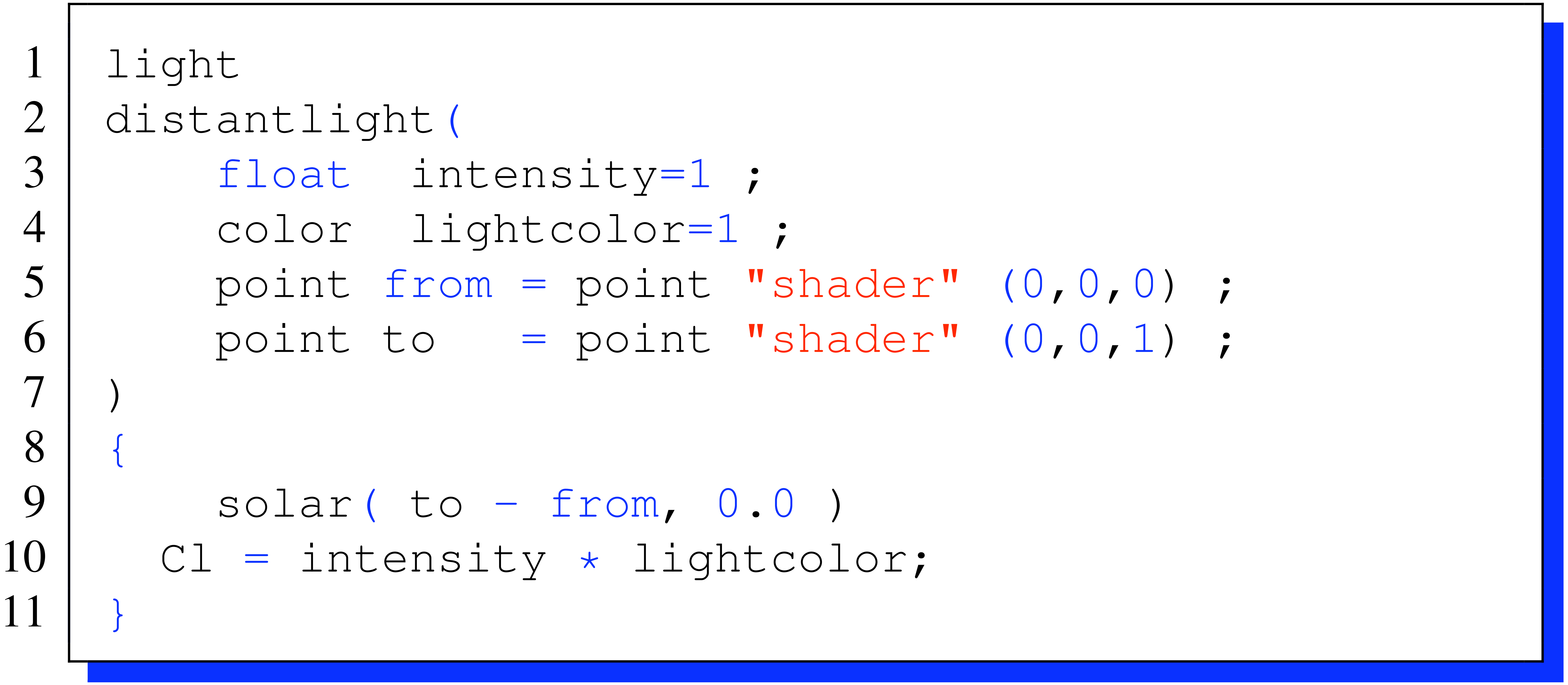


- Distant light with from position [1,1,1] and [10,10,10] give the same illumination



- Directional lights can be hard to aim but do provide a good method of fill lighting

renderman distantlight



```
light
distantlight(
    float  intensity=1 ;
    color  lightcolor=1 ;
    point from = point "shader" (0,0,0) ;
    point to   = point "shader" (0,0,1) ;
)
{
    solar( to - from, 0.0 )
  Cl = intensity * lightcolor;
}
```

spotlights

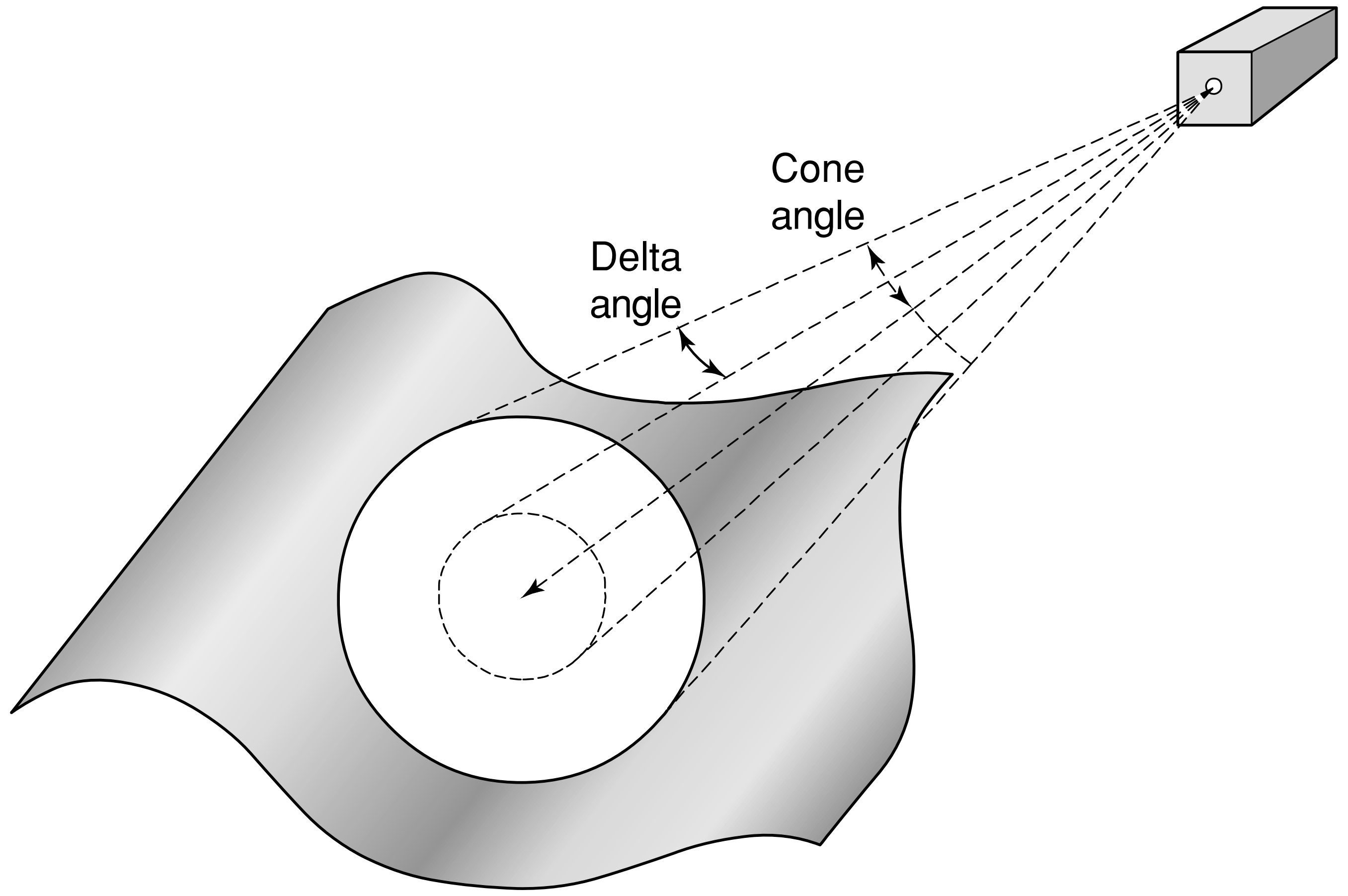
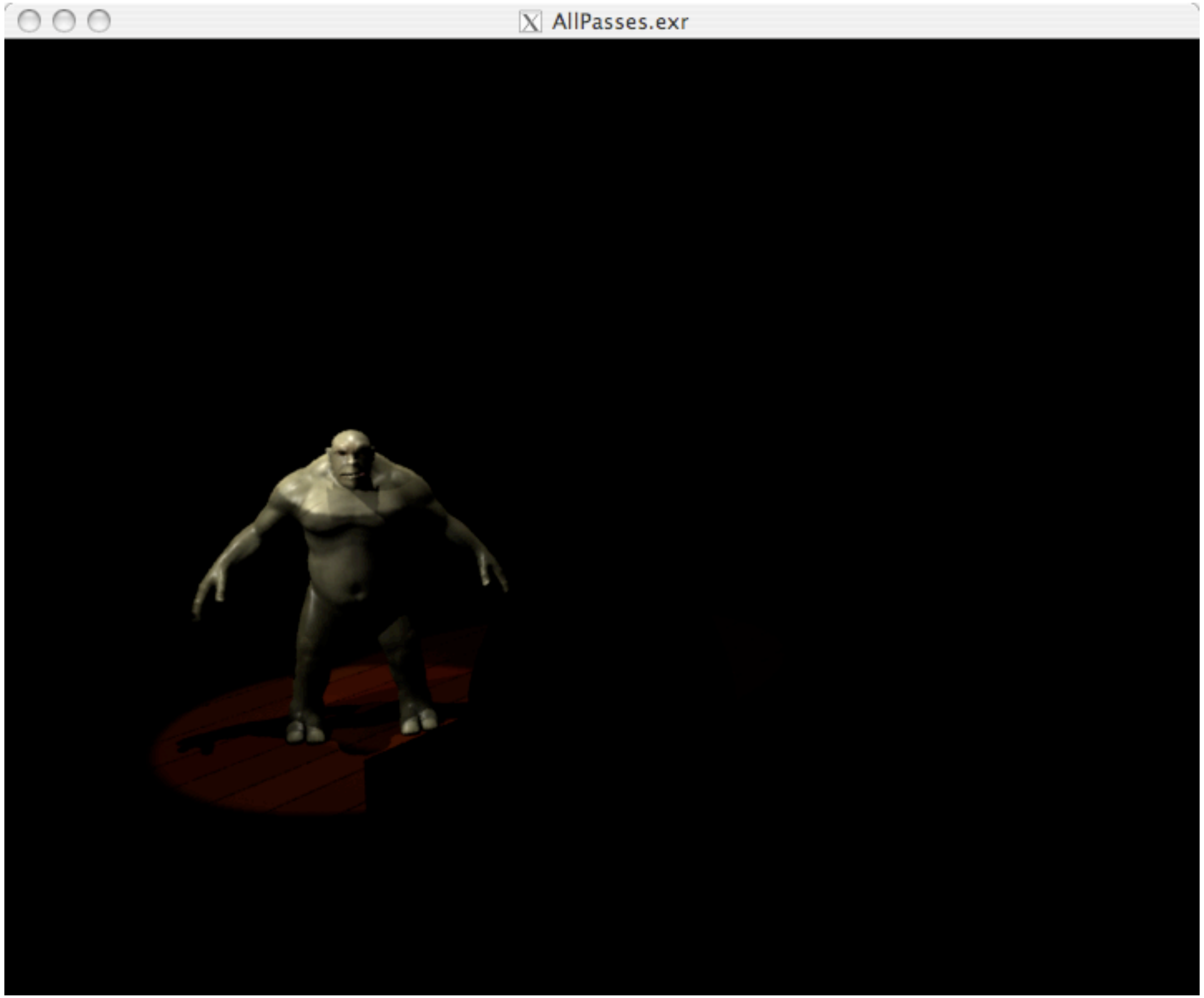
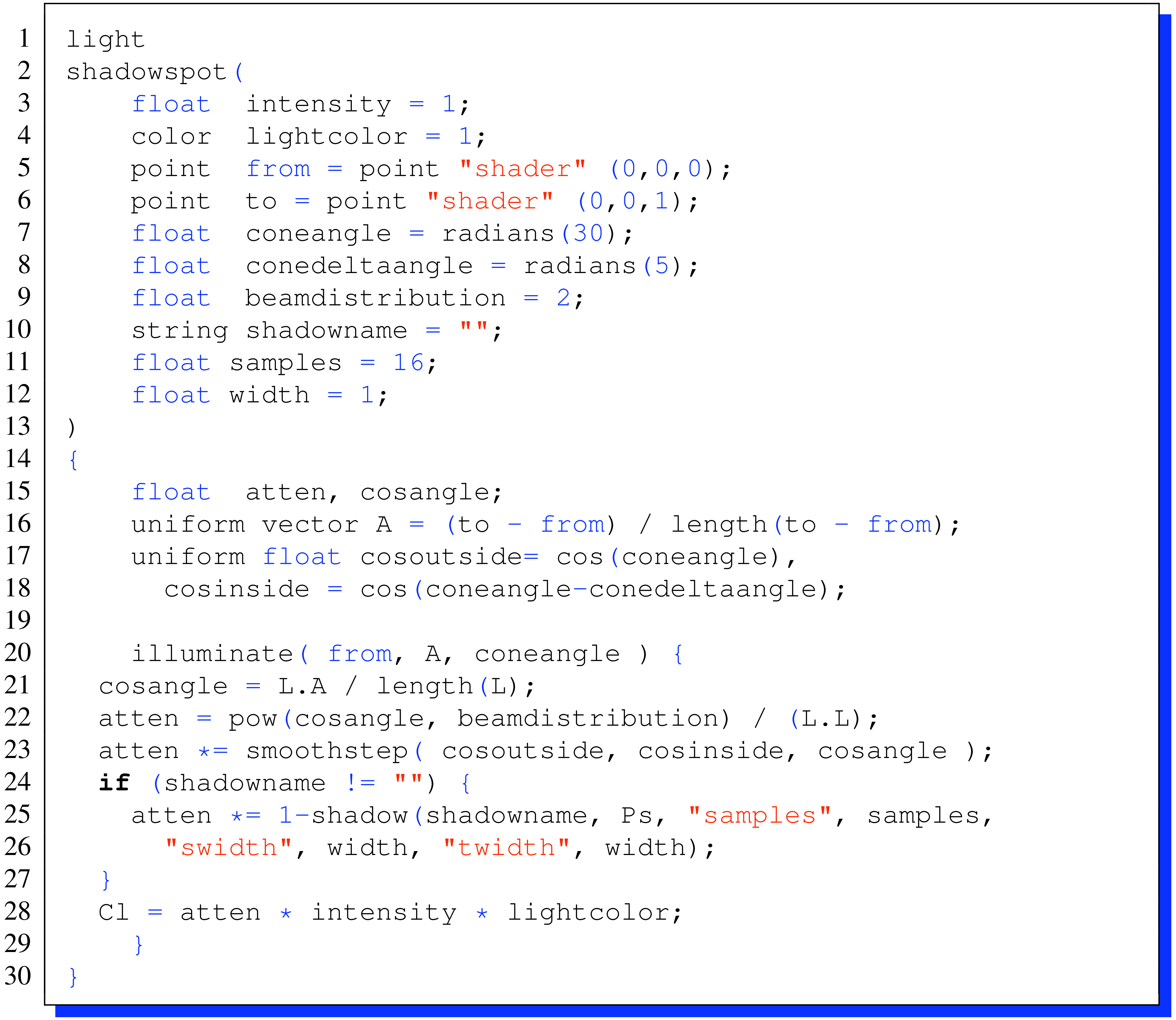


Figure 11.5. Cone angle and delta angle

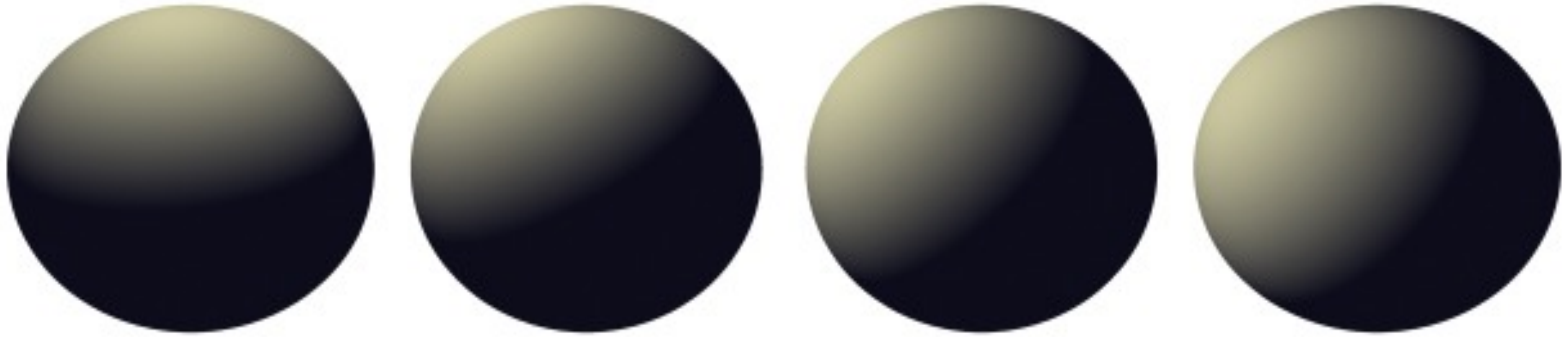
- A spotlight simulates light radiating from a point in a similar fashion to a pointlight
- However a spot light’s illumination is limited to a user defined cone
- This is usually specified by two angles as shown above


AllPasses.exr

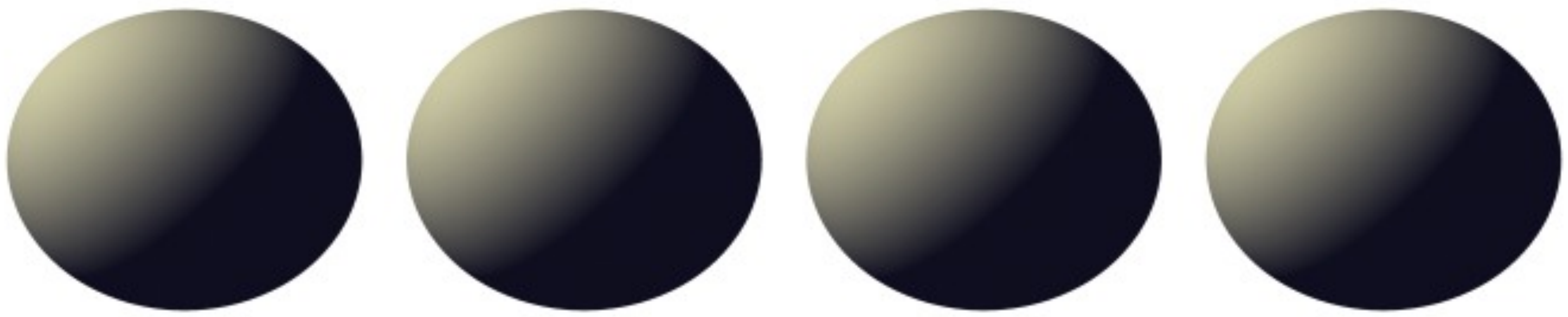
renderman spotlight



```
light
shadowspot(
    float  intensity = 1;
    color  lightcolor = 1;
    point  from = point "shader" (0,0,0);
    point  to = point "shader" (0,0,1);
    float  coneangle = radians(30);
    float  conedeltaangle = radians(5);
    float  beamdistribution = 2;
    string shadowname = "";
    float samples = 16;
    float width = 1;
)
{
    float  atten, cosangle;
    uniform vector A = (to - from) / length(to - from);
    uniform float cosoutside= cos(coneangle),
      cosinside = cos(coneangle-conedeltaangle);

    illuminate( from, A, coneangle ) {
  cosangle = L.A / length(L);
  atten = pow(cosangle, beamdistribution) / (L.L);
  atten *= smoothstep( cosoutside, cosinside, cosangle );
  if (shadowname != "") {
    atten *= 1-shadow(shadowname, Ps, "samples", samples,
      "swidth", width, "twidth", width);
  }
  Cl = atten * intensity * lightcolor;
    }
}
```



- So far, we have considered only point lights
- A light source at a near-infinite distance away from a surface has near-parallel rays.
- The vector made from the surface to the light source is always the same for every surface.
- This is known as a directional light – light from a known direction not position.
- We do not specify a position for directional lights, just a vector to indicate ray direction.



- To implement a directional light into the illumination equation,
- we simply use the same light vector in every different surface illumination equation
- i.e. **L** does not change, as it is constant for the light source.

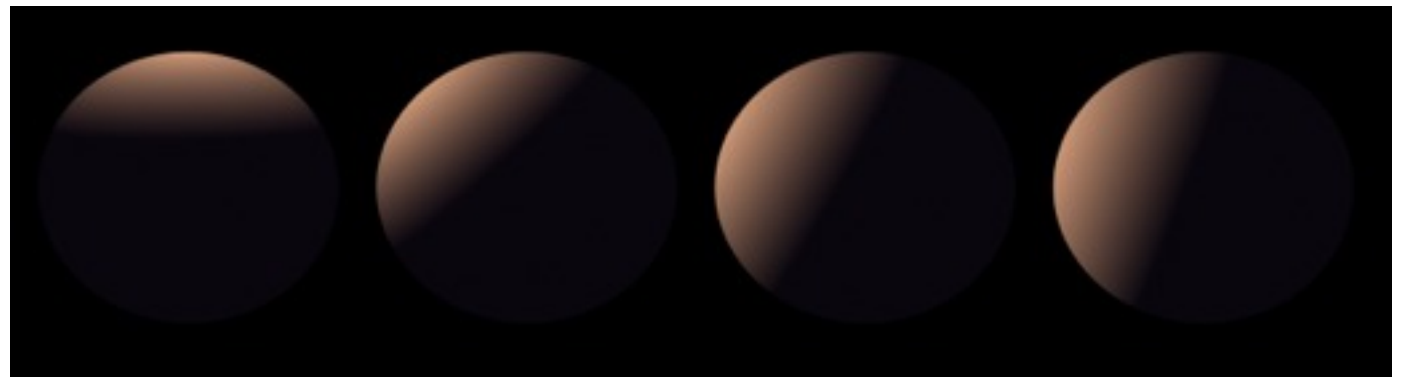
$$I = I_p k_d (N \bullet L)$$

Lambert's lighting model can be combined with the previous *ambient light equation*.

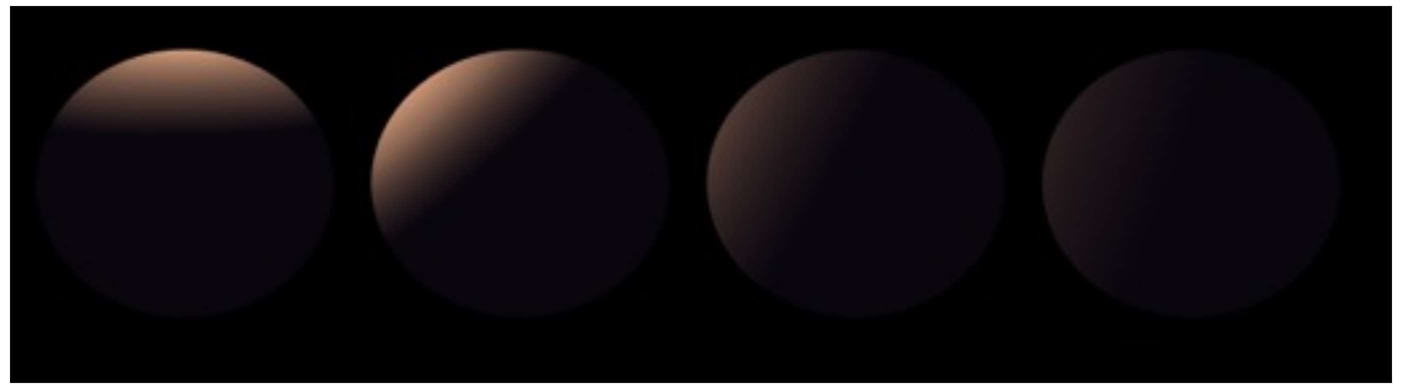
Ambient Light $I = I_a k_a$

$I = I_p k_d (N \bullet L)$ Lambert's

$$I = I_a k_a + I_p k_d (N \bullet L)$$



The above is an example of lighting with no attenuation and below is an example with a more realistic attenuation of light.



Real light has an ***attenuation factor*** – light intensity becomes weaker over distance.
In a similar manner, the perception of sound volume decreases the further away you are from its source.

To include attenuation into our illumination equation, we need to insert an ***attenuation factor*** into the lighting section – in this case labelled $\mathbf{f_{att.}}$

With this multiplying factor in place, we can control the intensity of light based on distance.

$$I = I_a k_a + f_{att} I_p k_d (N \bullet L)$$

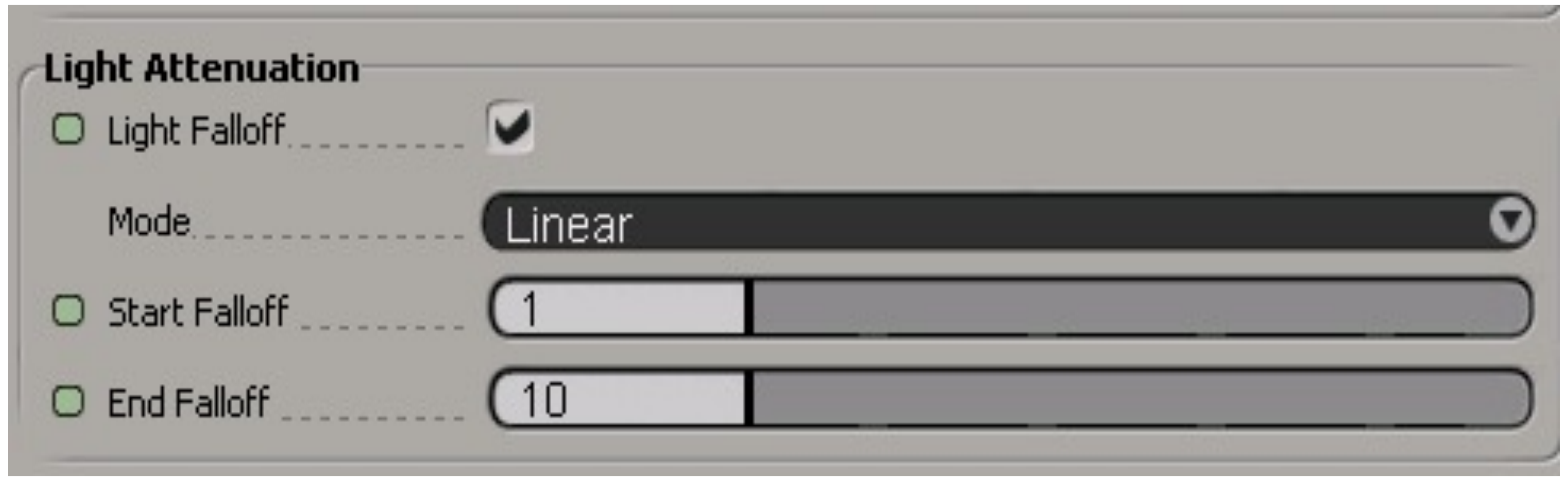
An $\boldsymbol{f_{att}}$ of 0 would effectively turn the light off.
An $\boldsymbol{f_{att}}$ of 1 would result in a maximum intensity of the light.

Light falloff obeys what is commonly known as the inverse square law its intensity decreases exponentially in relation to distance.

$$f_{att} = \frac{1}{{d_L}^2}$$

If we consider the distance from the surface point to the light ***L*** as $\boldsymbol{d_{L\ldots}}$

- This gives a small range of useable light and most CG applications use less realistic methods to light scenes.
- e.g. you can set the falloff as a linear amount, with the distances from which to range the falloff being defined by the user or even no fall off at all.



Light Attenuation

○ Light Falloff ☑

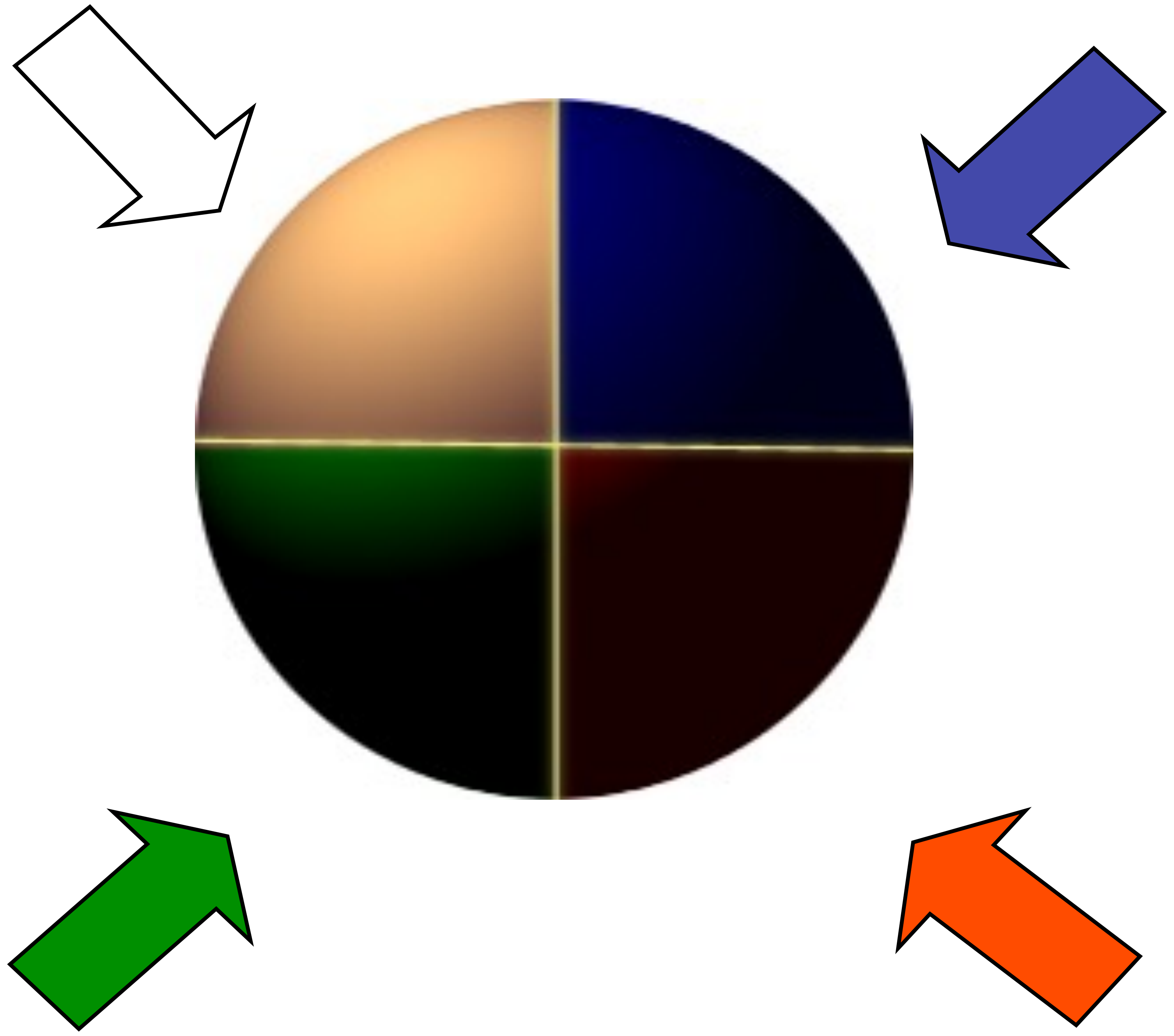
Mode Linear

○ Start Falloff 1

○ End Falloff 10

So far the notion of using the illumination equation has had no reference to actual colour – only monochromatic intensity.

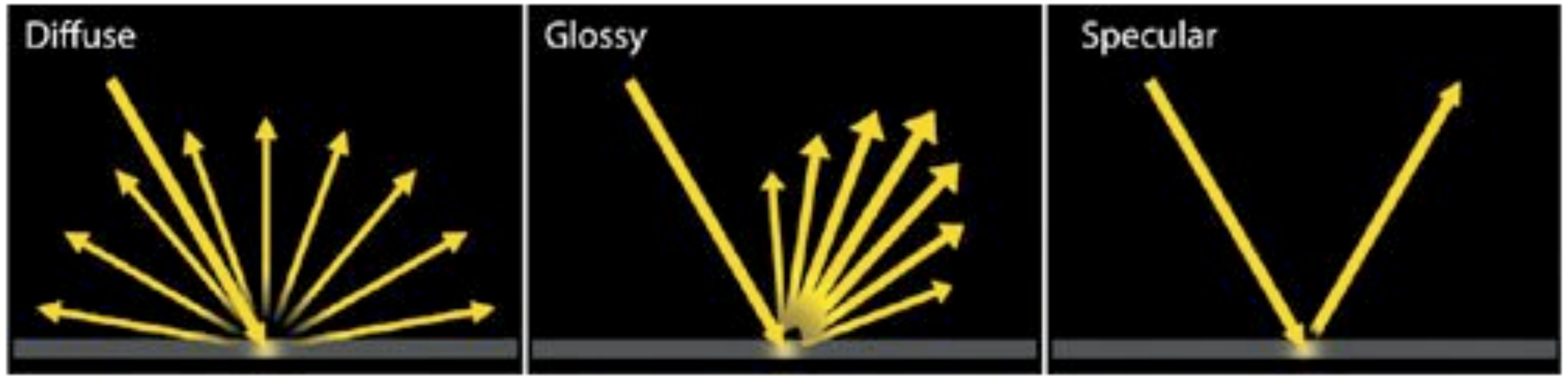
What we require to do so, are 3 equations, to represent the Red, Green and Blue data.



$$I_R = I_{aR} k_{aR} + f_{att} I_{pR} k_{dR} (N \bullet L)$$

$$I_G = I_{aG} k_{aG} + f_{att} I_{pG} k_{dG} (N \bullet L)$$

$$I_B = I_{aB} k_{aB} + f_{att} I_{pB} k_{dB} (N \bullet L)$$



- The figure shows how diffuse, glossy, and specular reflection appear on surfaces.
- Diffuse reflection (left) gives materials a matte appearance, so that they don't show any reflections or highlights.
- Glossy reflections (centre) are soft, and diverging rays naturally make reflections appear softer with distance
- Specular reflections (right) are crisp and mirror-like.

Shiny surfaces exhibit ***specular reflection*** – the reflection of the light source towards the viewer.

Specular reflection has 2 main colour biases:

1. The ***colour*** of the specular reflection is determined by the ***light source colour.***

or

2. The ***colour*** of the specular reflection is determined by the ***colour of the surface***.

Objects that have waxed, or transparent surfaces (apples, plastic, etc.) tend to reflect the colour of the light source.

Plastic, for example, is composed of colour pigments suspended in a transparent material.

…and Gold has a *gold* coloured highlight.

- A common misconception is that specular highlights are centred in the brightest point of the diffuse shading.

- In reality, the positioning of the specular highlights is derived separately from the diffuse shading.

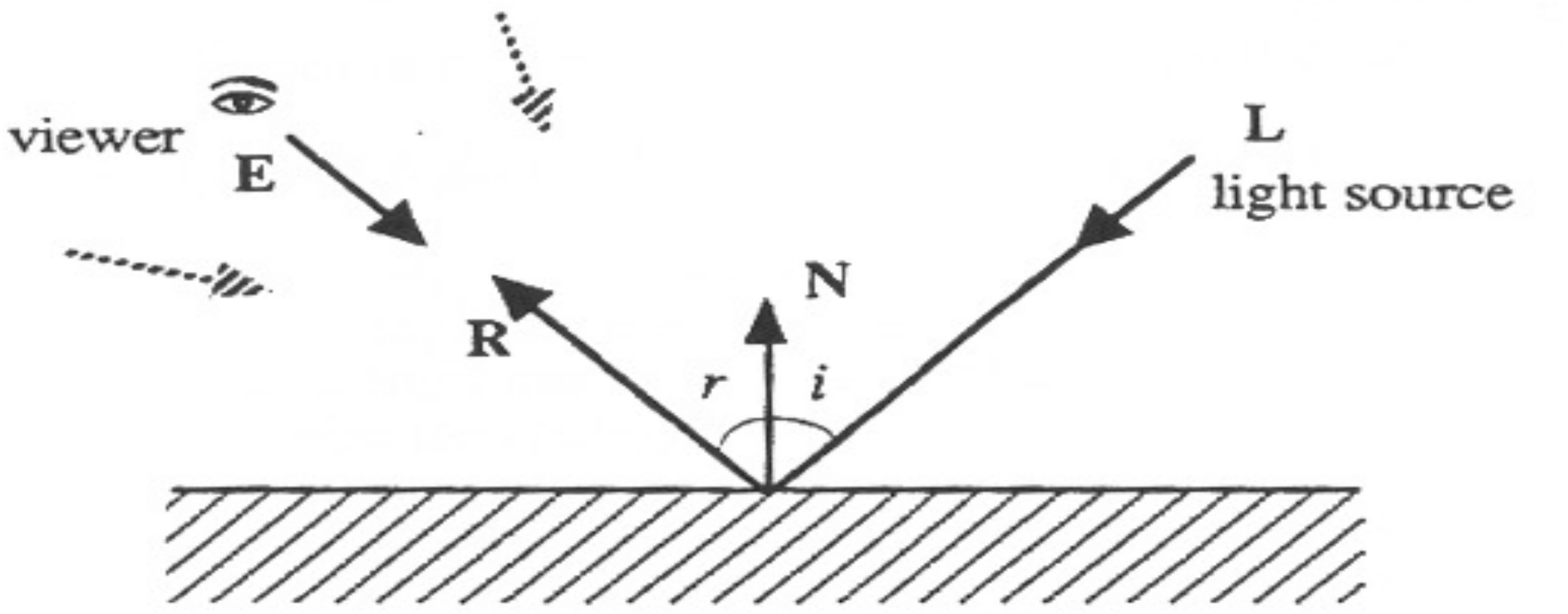
- Specular shading is calculated only from a specific camera angle, and is based on the angle between the light, the surface, and the camera.

- Because of this, specular highlights are an example of view-dependent shading.

- View-dependent shading is any effect that varies depending on the camera angle from which it was rendered.

- Specularity, reflections, and refraction are all examples of view-dependent shading; they seem to shift across a surface when you view it from different angles.

- Contrast this with non-view-dependent shading, such as diffuse shading and cast shadows, which can be computed without regard for the camera angle.



The above diagram describes a perfectly mirrored surface.

The viewer is seen looking at a point on the surface: viewing vector ***E***.

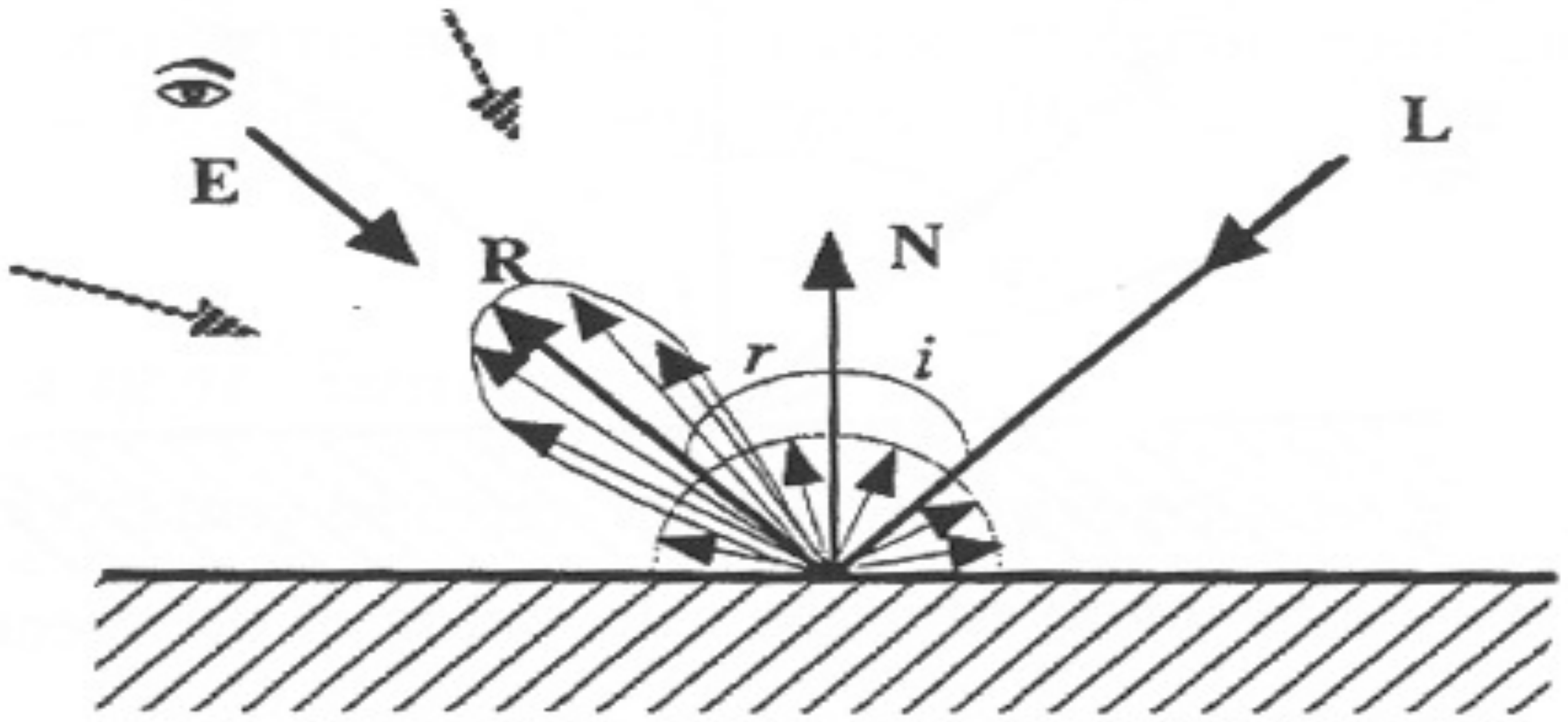
The light source ***L*** emanates a ray of light that that hits the surface with an angle of ***i*** relative to the normal.

The angle of reflection of the light ray then leaves with an angle ***r*** relative to the normal.
The angle of incidence ***i*** is the same as angle ***r***.

In a perfect mirror (above), the viewer may ***ONLY*** see the light ray if the viewing angle ***E*** is directly opposite of the reflection vector ***R***.

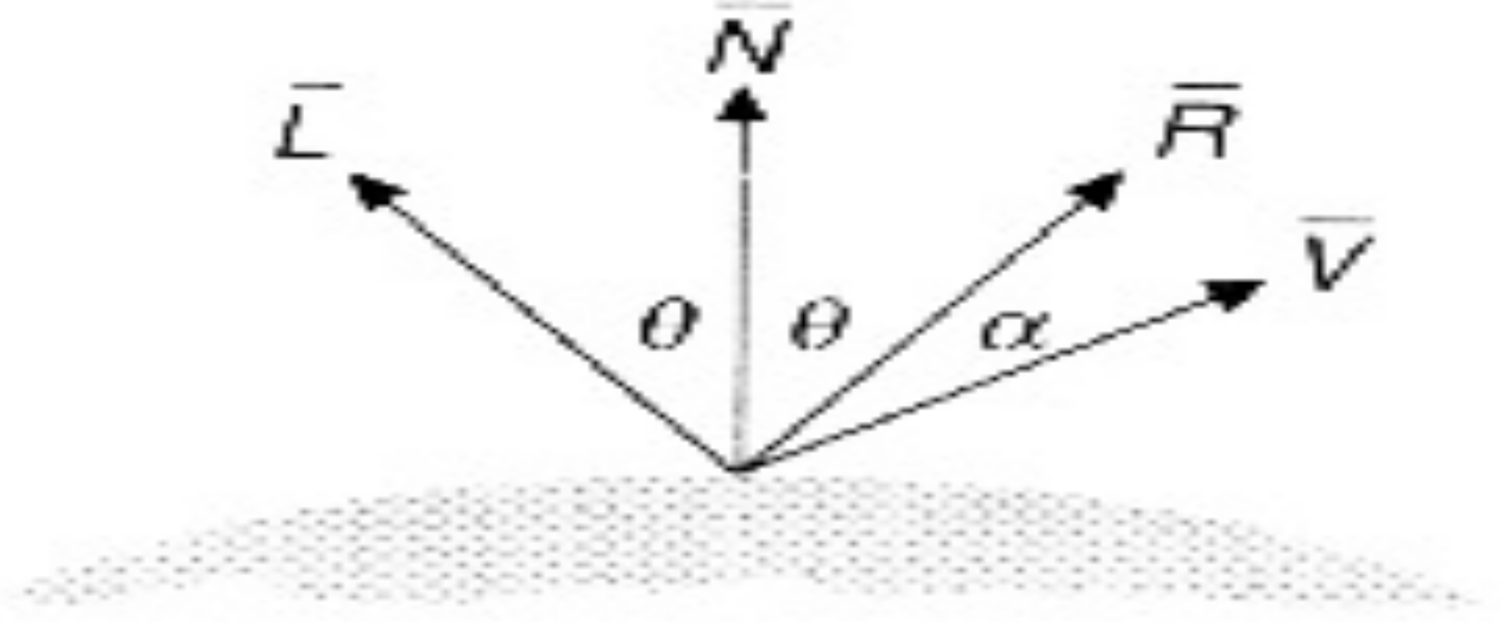
Most shiny surfaces are not perfect mirrors.
Shiny surfaces will reflect the largest intensity where the viewing angle is directly opposite the reflection angle.

They usually also reflect a diminishing gradient of highlight as well, enabling light to be seen from angles not directly opposed to the angle of reflection.



Phong Bui-Tuong developed an illumination model for non-perfect reflectors that has become widely used to portray realistic shiny surfaces.

It is commonly known as the Phong illumination model.



In the above diagram, the angle of incidence in relation to the surface normal is theta.

V represents the viewing vector (reversed so we are looking from the surface)

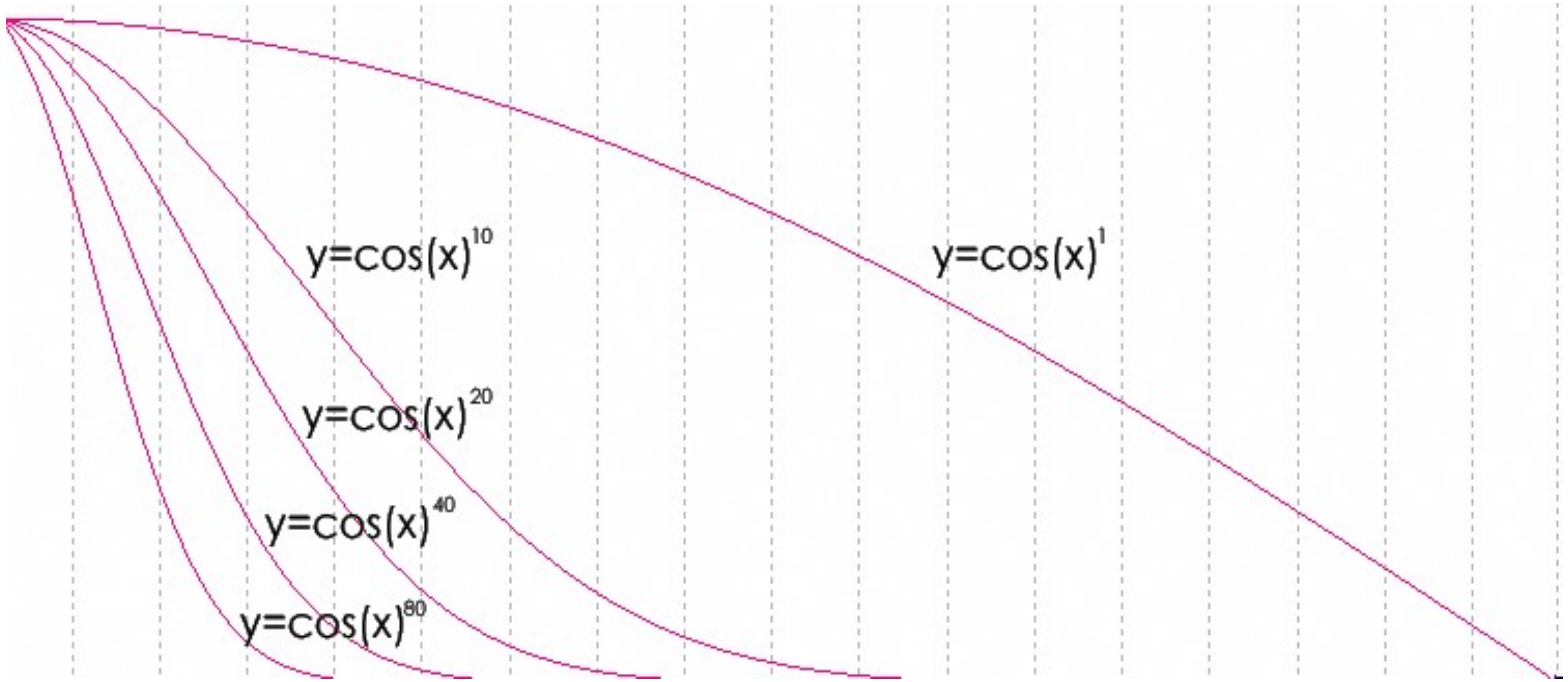
alpha the angle between the viewing vector and the reflection vector.

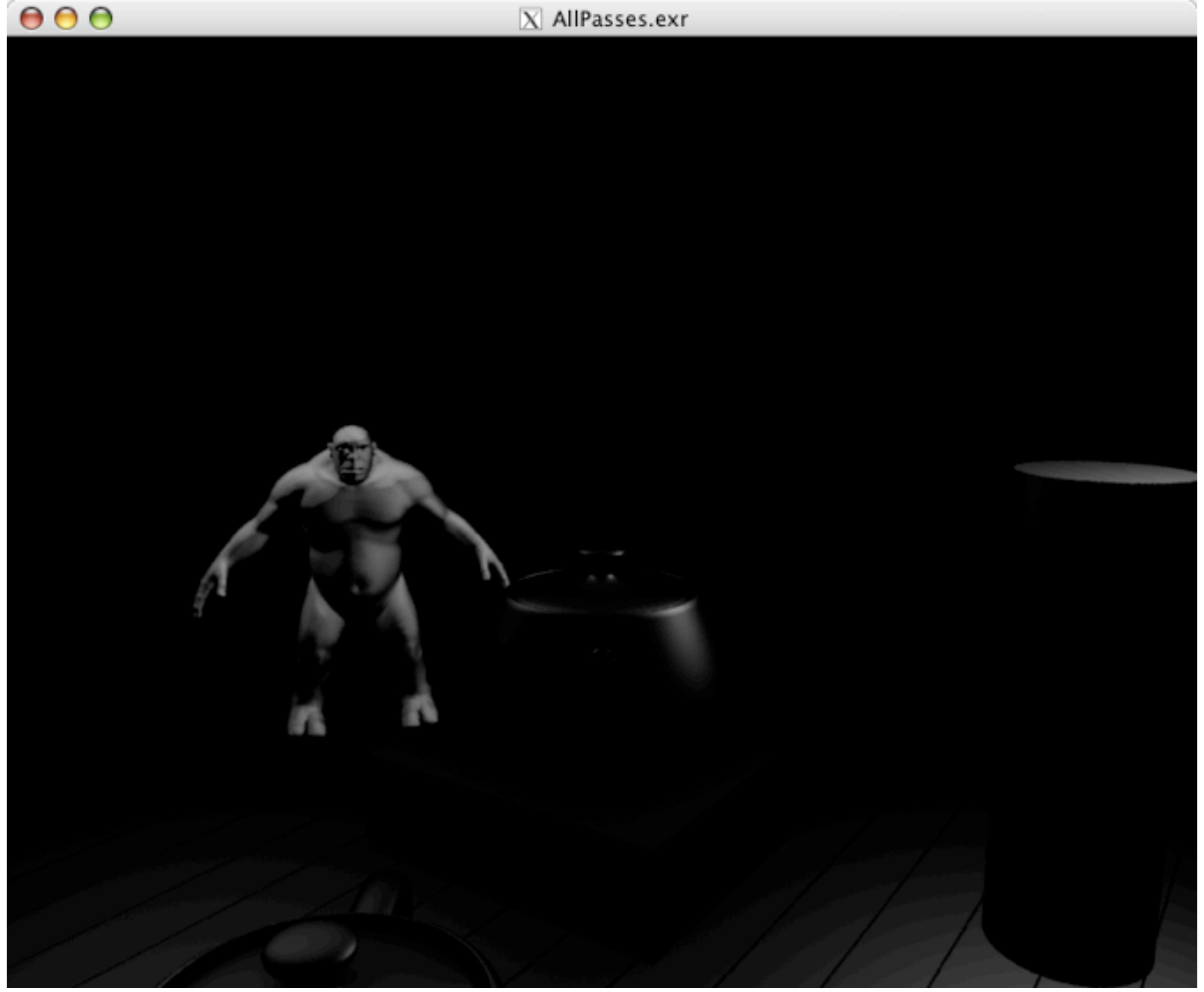
Phong postulated that maximum specular reflection was achieved when the angle between the viewing angle and the the reflection angle was smallest – i.e. ***alpha*** is zero.

He then stipulated that the specular reflection falls off sharply as ***alpha*** increases which he stated could be represented by ***cos***^{n} ***alpha***.

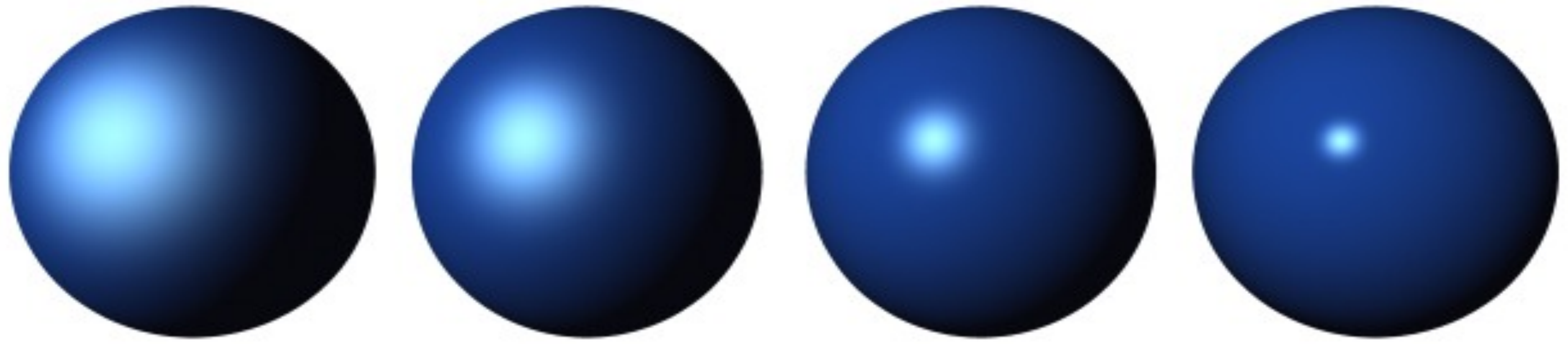
What does the cos^{n} alpha mean?

Lets examine some graphical representations of various exponents of cos alpha.





```
surface Phong
(
 float Ks=1;
 float size=100;
 color SpecColour=1;
)
{
// init the shader values
normal Nf = faceforward(normalize(N),I);
vector V = -normalize(I);

// now calculate the shading values


color Phong=0;
  vector R = reflect( -normalize(V), normalize(N) );

  illuminance( P , Nf, PI/2.0 )
  {
    vector Ln = normalize(L);
    Phong += Cl * pow(max(0.0,R.Ln), size);
  }

Ci= Oi*(SpecColour*Ks*Phong);
}
```



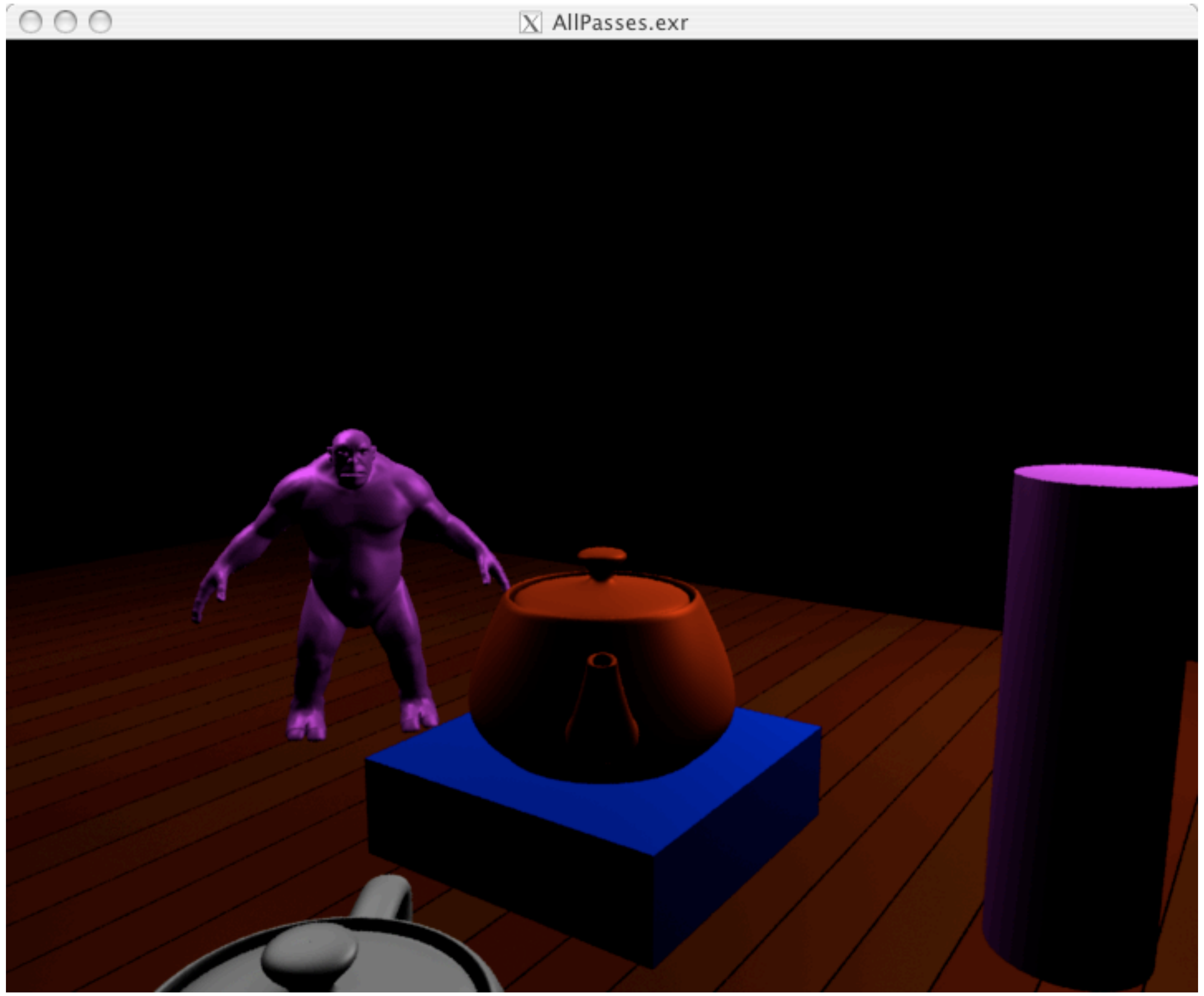
The above examples have Phong falloff vales of 8, 16, 64 and 256 (from left to right). The Phong Illumination Equation reads as follows:

Ambient Lambert Phong

$$I = \boxed{I_a k_a} + \boxed{f_{att} I_p k_d (N \bullet L)} + \boxed{f_{att} I_p k_s (R \bullet V)^n}$$

ks represent the specular reflection coefficient,
n the exponent of the cosine function,

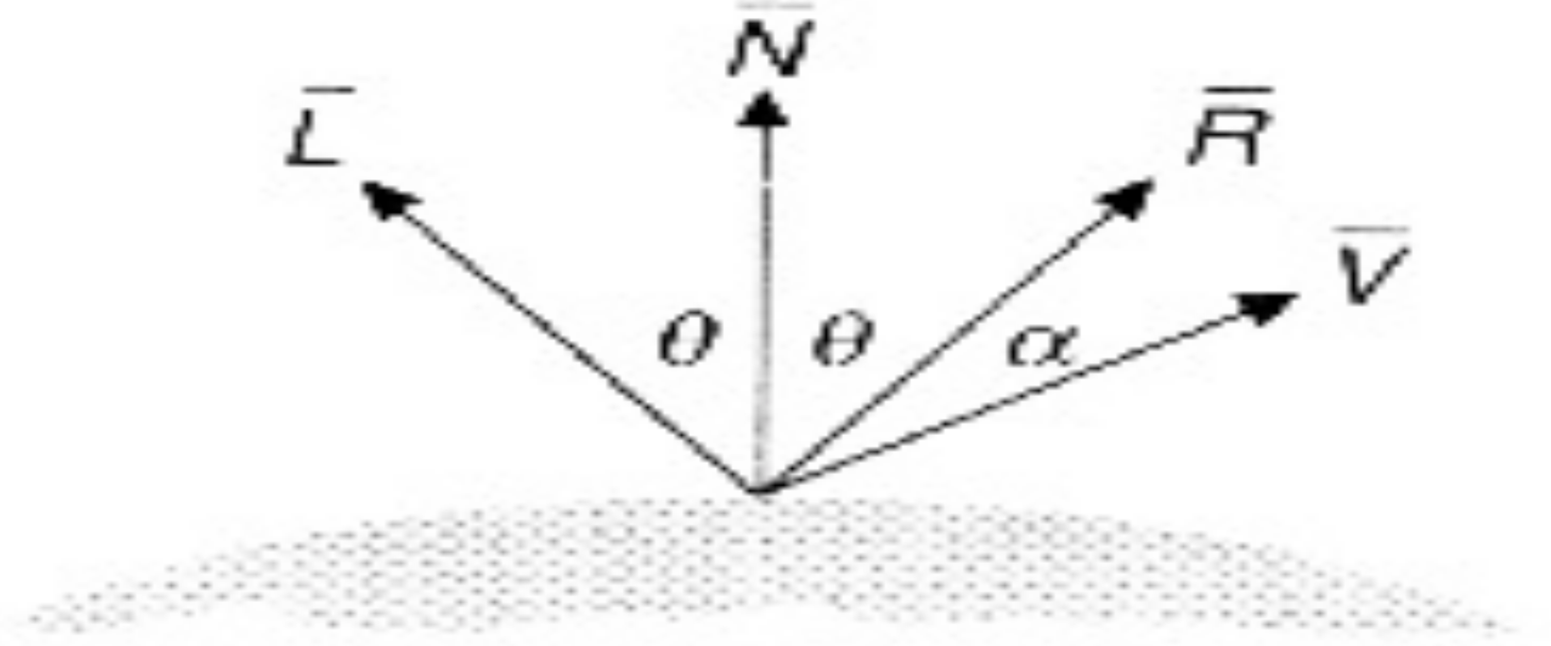
and the cosine of the angle between the *viewing vector* and the *reflection vector* can be calculated via the dot product of the 2 ***normalised*** respective vectors.



```
surface basicPlastic
(
 float Ka=1;
 float Kd=0.5;
 float Ks=0.5;
 float size = 10;
 color specularcolor = 1;
)
{
// init the shader values
normal Nf = faceforward(normalize(N),I);
vector V = -normalize(I);

// now calculate the shading values


color Diffuse=0;
color Phong=0;
normal Nn=ntransform("object",N);
vector R = reflect( -normalize(V), normalize(N) );
  illuminance( P, Nf, PI/2 )
  {
  Diffuse += Cl * normalize(L).Nf;
  vector Ln = normalize(L);
  Phong += Cl * pow(max(0.0,R.Ln), size);
  }

Ci= Oi*Cs * (Ka*ambient() + Kd*Diffuse + (specularcolor *
     Ks*Phong));
}
```

Ambient Lambert Phong

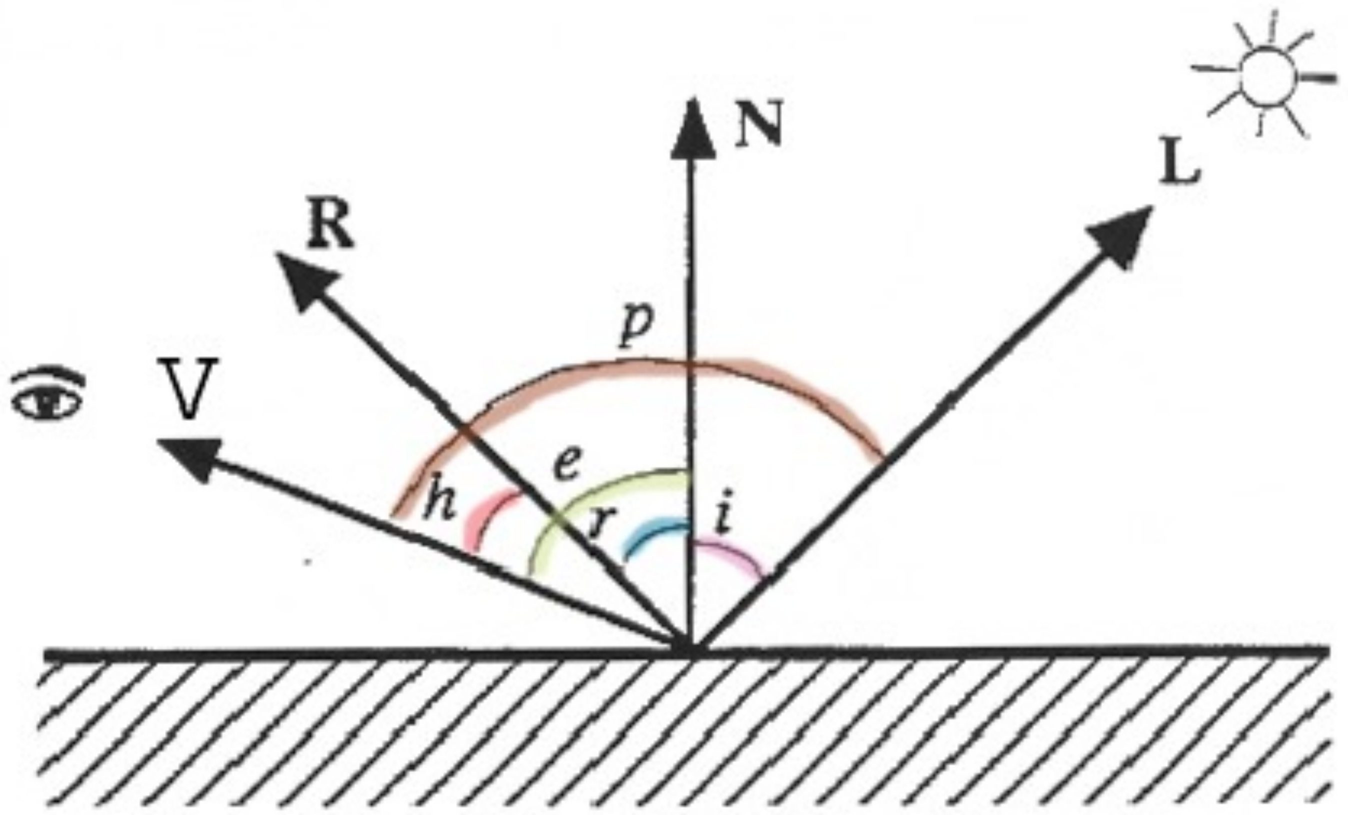
$$I = \boxed{I_a k_a} + \boxed{f_{att} I_p k_d (N \bullet L)} + \boxed{f_{att} I_p k_s (R \bullet V)^n}$$



The vectors used in Lambert's models are easily generated from information available – the surface normal, the surface point and the light position.

The vectors used in Phong's addition are less easily generated.
V is fairly straightforward (the viewer position and surface point position are available).

The vector ***R*** however has to be generated from existing data.



The above diagram incorporates extra detail: **h** represents the angle between ***V*** and ***R***.

$$h = e - r = e - i$$

$$p = e + i$$

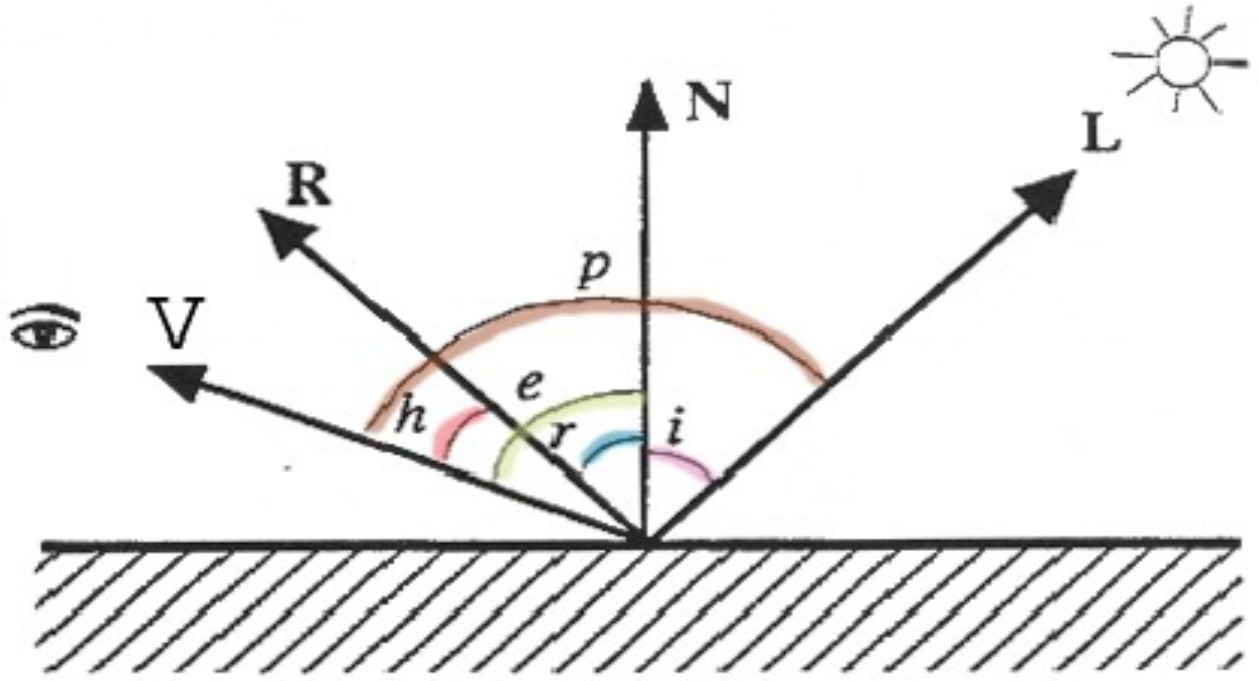
Thus using trigonometric functions:

$$\cos(h) = \cos(e - i) = \cos(e) \times \cos(i) + \sin(e) \times \sin(i)$$

$$\cos(p) = \cos(e + i) = \cos(e) \times \cos(i) - \sin(e) \times \sin(i)$$

If we add the two together:

$$\cos(h)+\cos(p)=\cos(e)\times\cos(i)+\sin(e)\times\sin(i)+\cos(e)\times\cos(i)-\sin(e)\times\sin(i)$$



Therefore:

$$\cos(h)+\cos(p)=2\cos(e)\times\cos(i)$$

And so…

$$\cos(h)=2\cos(e)\times\cos(i)-\cos(p)$$

If we substitute known vectors into the equation:

$$\cos(h)=2\times(N\bullet V)\times(N\bullet L)-(V\bullet L)$$

Thus from our Phong equation:

$$R\bullet V=2\times(N\bullet V)\times(N\bullet L)-(V\bullet L)$$

We are dealing with a surface with multiple facets.

This micro facet distribution held the key to a lot of further developments.

Blinn stipulated that if a surface was a collection of randomly distributed facets, the orientation of the best facet for specular reflection would be one where:

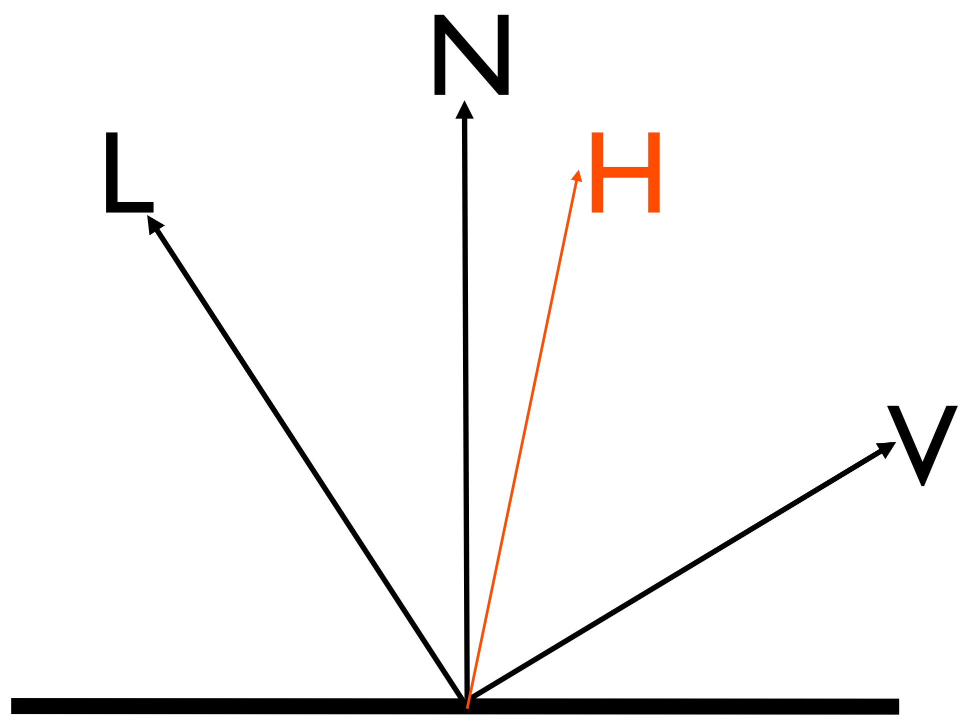
$$L \bullet N_B = V \bullet N_B$$

Where

L=the light direction

N_B=Blinn normal

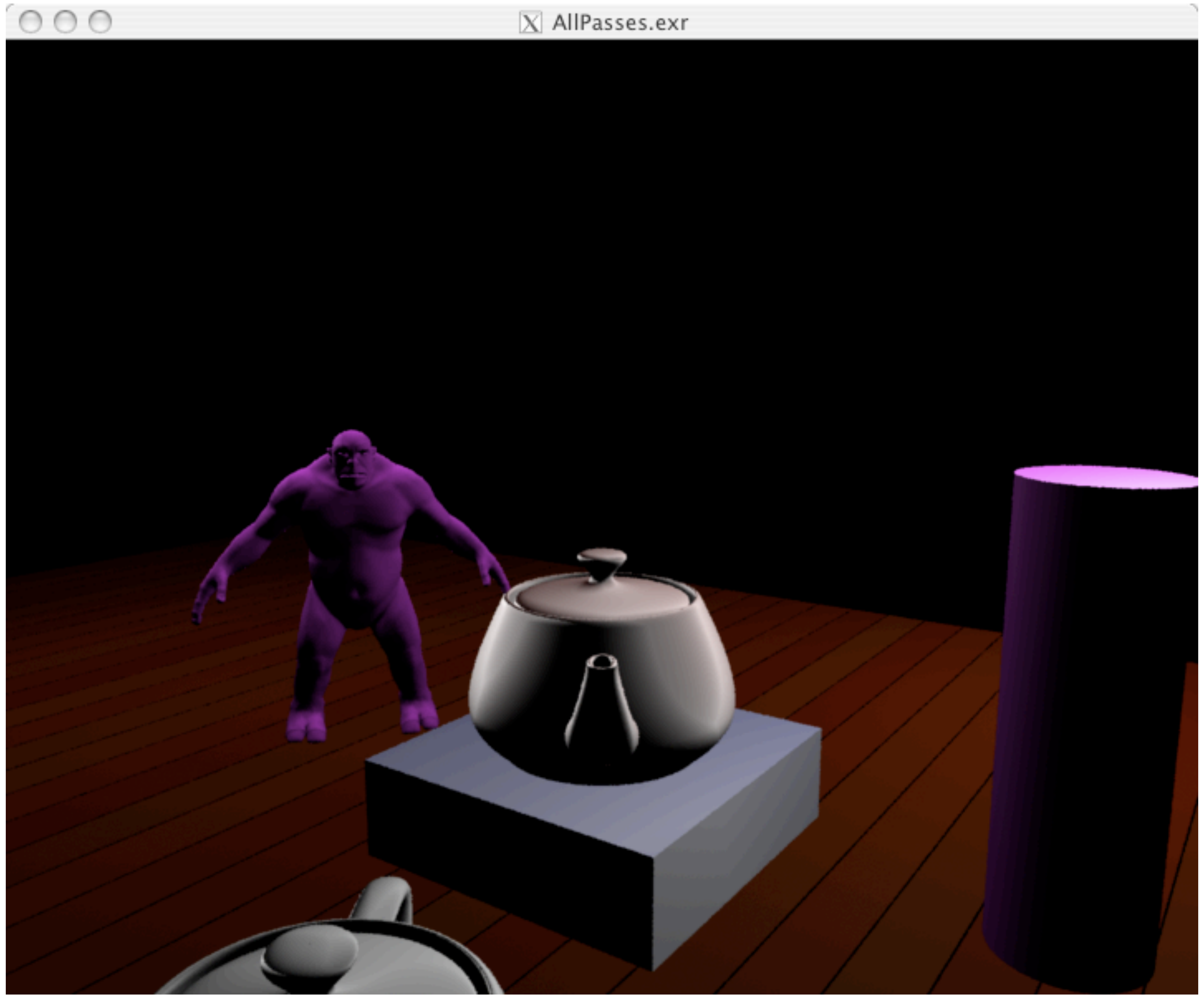
V=Eye/Viewer direction



Thus Blinn proposed a new vector replace the normal to represent basis for specular reflection.

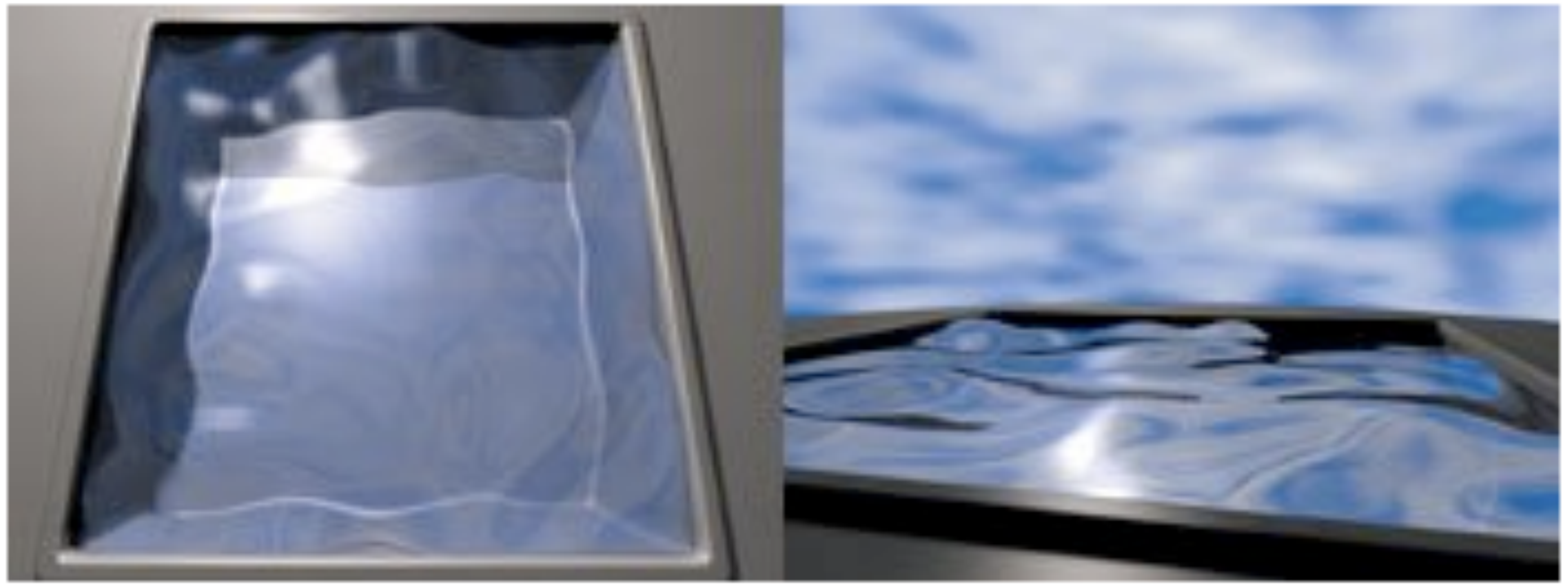
$$H = \frac{(L + V)}{2}$$

As the viewer approaches a view perpendicular to the surface normal of illumination, the specular highlights a proportionately larger than Phongs.



```
surface Blinn
(
 float Ks=0.5;
 float Kd=0.5;
 color SpecColour=1;
 float specularRollOff=0.002;
 float eccentricity=1;
)
{
// init the shader values
normal Nf = faceforward(normalize(N),I);
vector V = -normalize(I);

// now calculate the shading values
color Blinn=0;
float E;
vector H, Ln,  Nn;
float NH, NH2, NHSQ, Dd, Gg, VN, VH, LN, Ff, tmp;
float nondiff, nonspec;
illuminance( P , Nf, PI/2.0 )
{
if(eccentricity != 1)
  E = 1 / (eccentricity * eccentricity - 1);
else
  E = -1e5;
VN = V.Nf;
Ln = normalize(L);
H = normalize(Ln+V);
NH = Nf.H;
NHSQ = NH*NH;
NH2 = NH * 2;
VH = V.H;
LN = Ln.Nf;
if( VN < LN )
  {
  if( VN * NH2 < VH )
    Gg = NH2 / VH;
  else
    Gg = 1 / VN;
  }
else
  {
  if( LN * NH2 < VH )
    Gg = (LN * NH2) / (VH * VN);
  else
    Gg = 1 / VN;
  }
// poor man's Fresnel
tmp = pow((1 - VH), 3);
Ff = tmp + (1 - tmp) * specularRollOff;
Blinn = Gg * Ff * SpecColour;
}

Ci= Oi*Cs * (Kd*diffuse(Nf))+(Ks*Blinn);
}
```

- The French physicist Augustin-Jean Fresnel (1788-1827) advanced the wave theory of light through a study of how light was transmitted and propagated by different objects.
- One of his observations is now known in computer graphics as the Fresnel effect the observation that the amount of light you see reflected from a surface depends on the viewing angle.

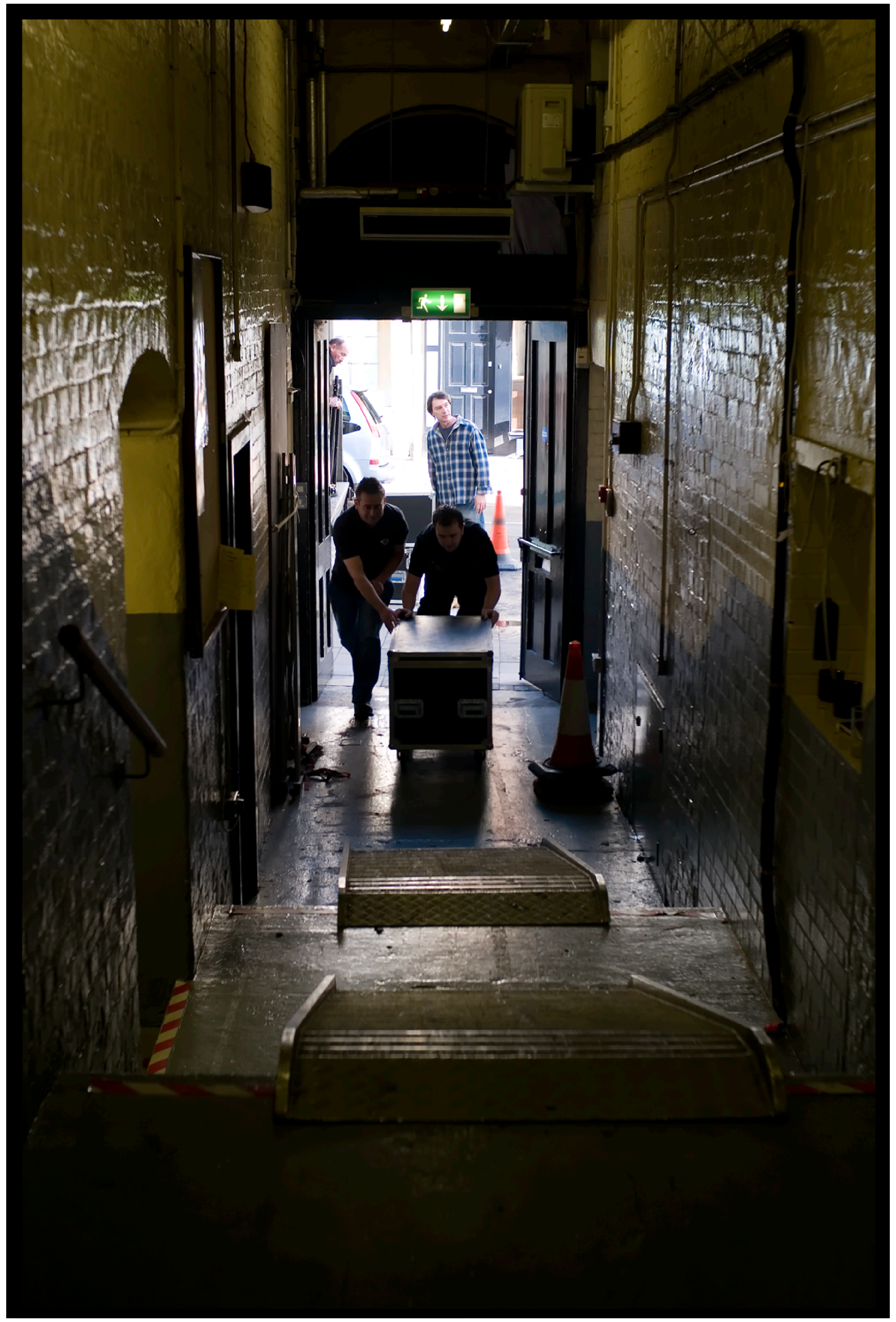


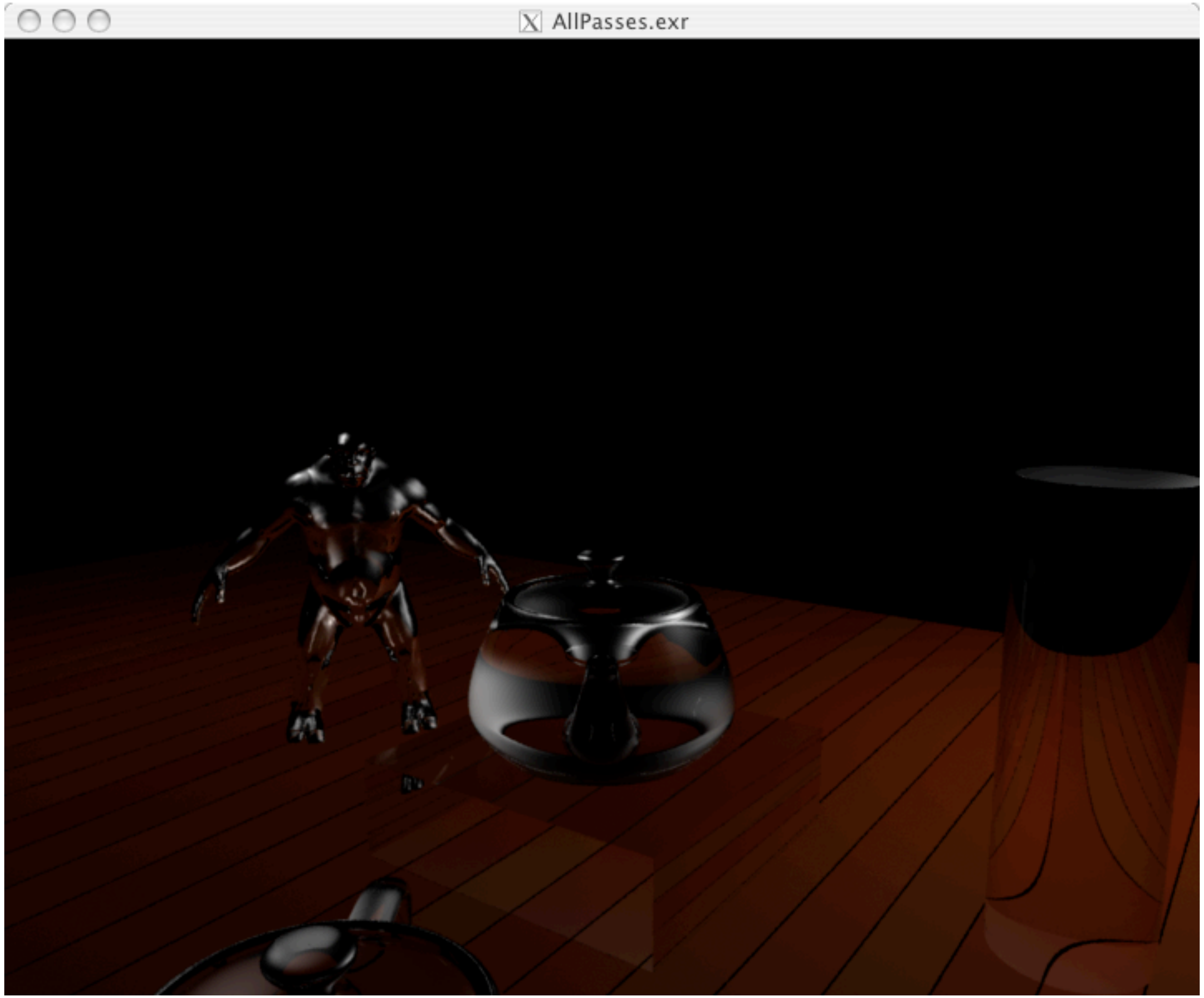
- If you look straight down at a pool of water, you won't see very much reflected light on the surface of the pool.

- From the high angle, without seeing reflections, you can see down through the surface to the bottom of the pool.

- At a glancing angle (looking with your eye level with the water, from the edge of the water surface), you will see much more specularity and reflections on the water surface, and might not be able to see what's under the water at all.

- A Fresnel shader will let you specify a specular colour to be seen on parts of a surface directly facing the camera, and another specular colour to be seen on parts of a surface that are perpendicular to the camera.
- Besides specular colour increasing at the edge of an object, the specular highlight size and reflectivity also increase.





AllPasses.exr

Light is an electromagnetic field.

The angle at which incoming light strikes a surface causes light to reflect in different manners.

This is due to the orientation of the electromagnetic field when it hits a surface.

A non-homogeneous material may have different reflectance for r,g and b

n_1=index of refraction of environment

n_2=index of refraction of medium

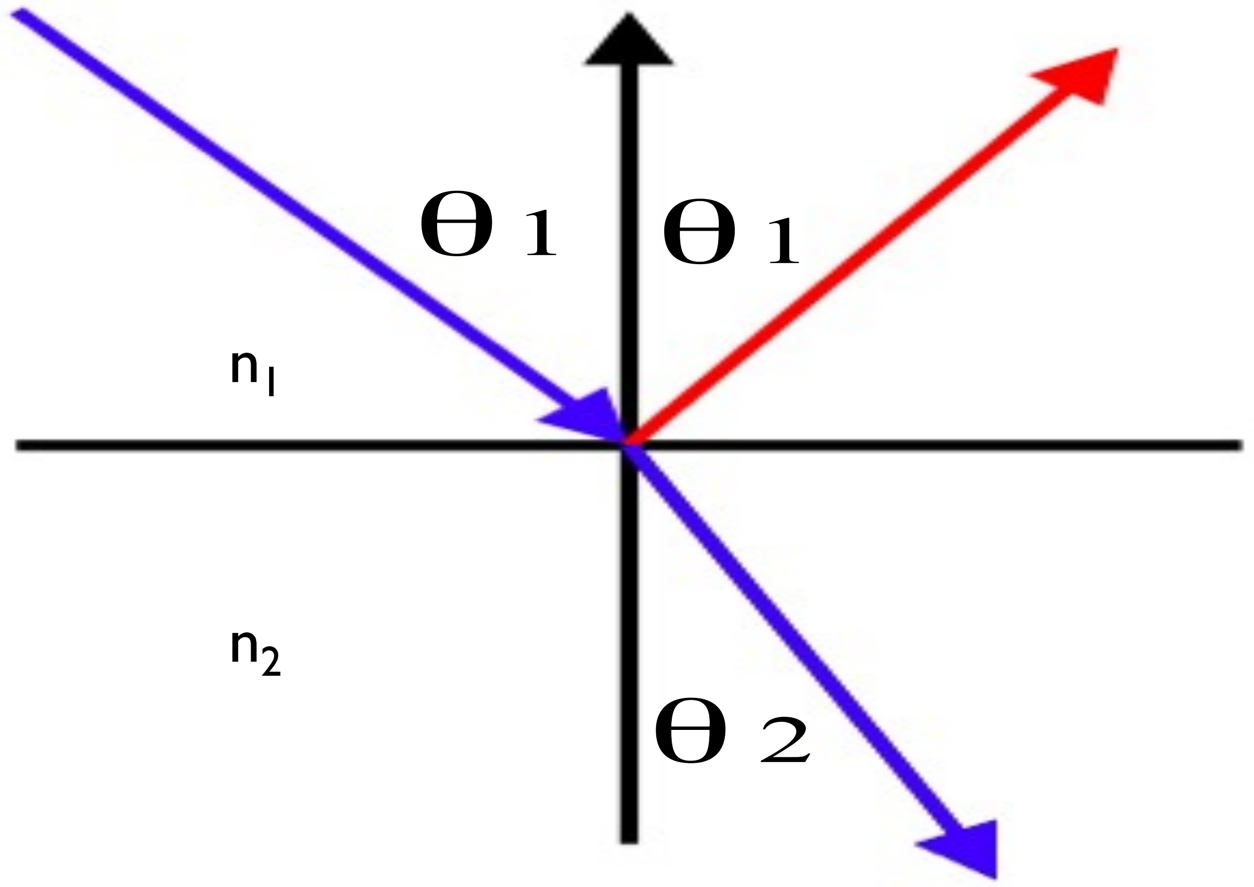
$$F = \frac{1}{2}\left(\rho_1{}^2 + \rho_2{}^2\right)$$

$$\rho_1 = \frac{n_2\cos\theta_1 - n_1\cos\theta_2}{n_2\cos\theta_1 + n_1\cos\theta_2}$$

$$\rho_2 = \frac{n_1\cos\theta_1 - n_2\cos\theta_2}{n_2\cos\theta_1 + n_1\cos\theta_2}$$

ρ_1 = reflect coeff. light parallel to plane of incidence

ρ_2 = reflect coeff. light orthogonal to plane of incidence



Anisotropic functions take ***direction*** of micro facets into account.

These equations accommodate surfaces such as brushed metal where groves aligned in a given direction.

The equation usually takes advantage of UV coordinates (as surface derivatives, the rate of change in the surface as the current position).

These models take into account self shadowing in the groves to limit illumination, thus allowing a given direction and the viewer to dictate illumination.

Cooke and Torrance proposed a method whereby scattering of light is wavelength independent i.e. different coloured light behaves in different manners.

There are 3 functions that contribute to this model:

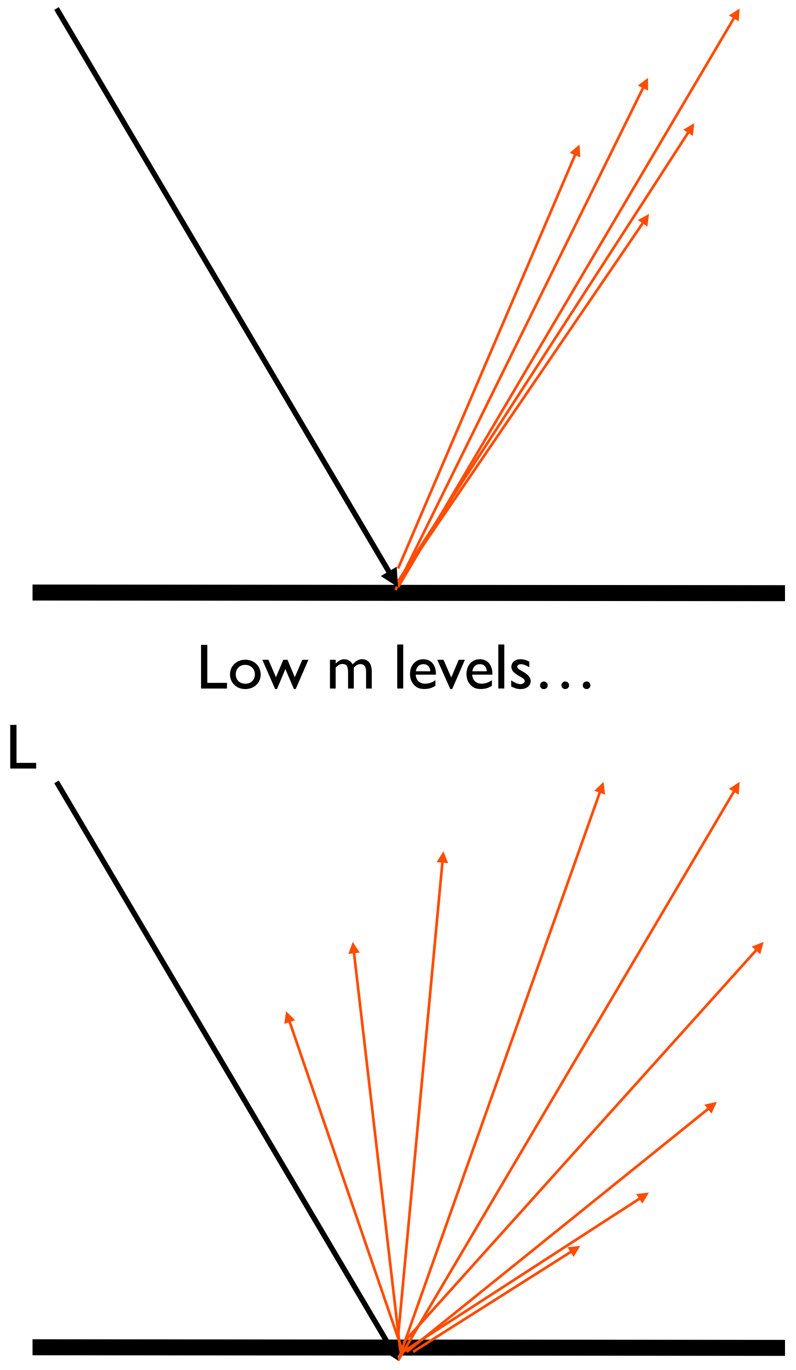
Micro Facet Distribution

Geometric Attenuation

Fresnel

E.g. based on a Beckman distribution function:

$$D = \frac{e^{-\left(\frac{\tan \beta}{m}\right)^2}}{4m^2 \cos^4 \beta}$$



High m levels…

Where

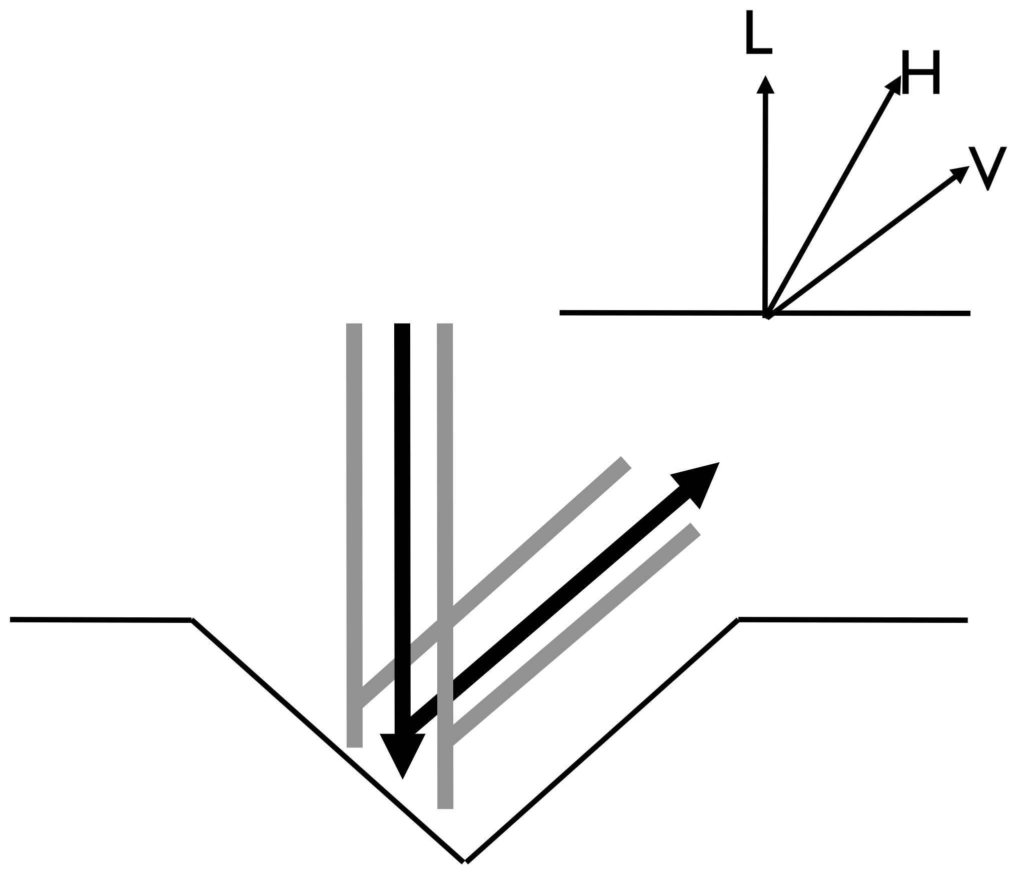
β = the angle between N and H

m = the root-mean-square slope of the micro-facets.

Large m indicates steep slopes between facets(light spread out)

Small m indicates smaller falloffs

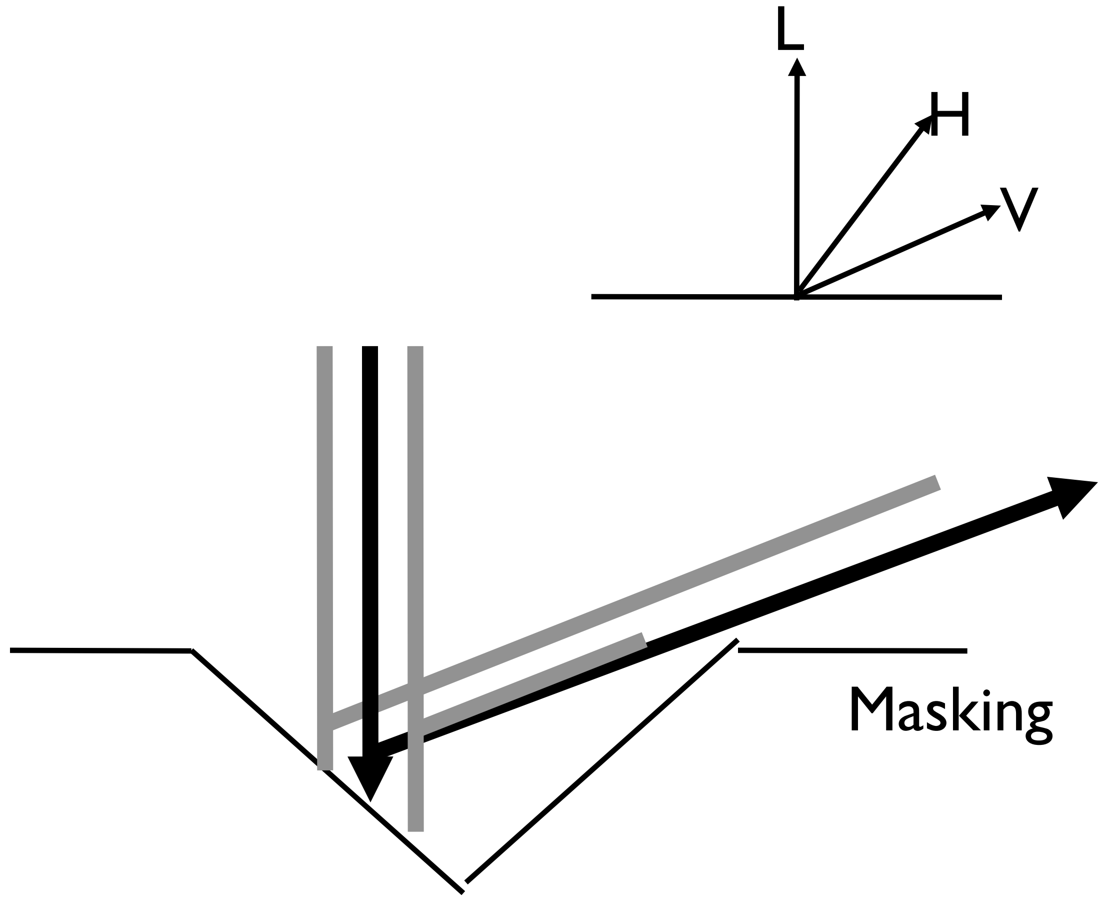
Accounts for shadowing and masking of micro facets by each other…

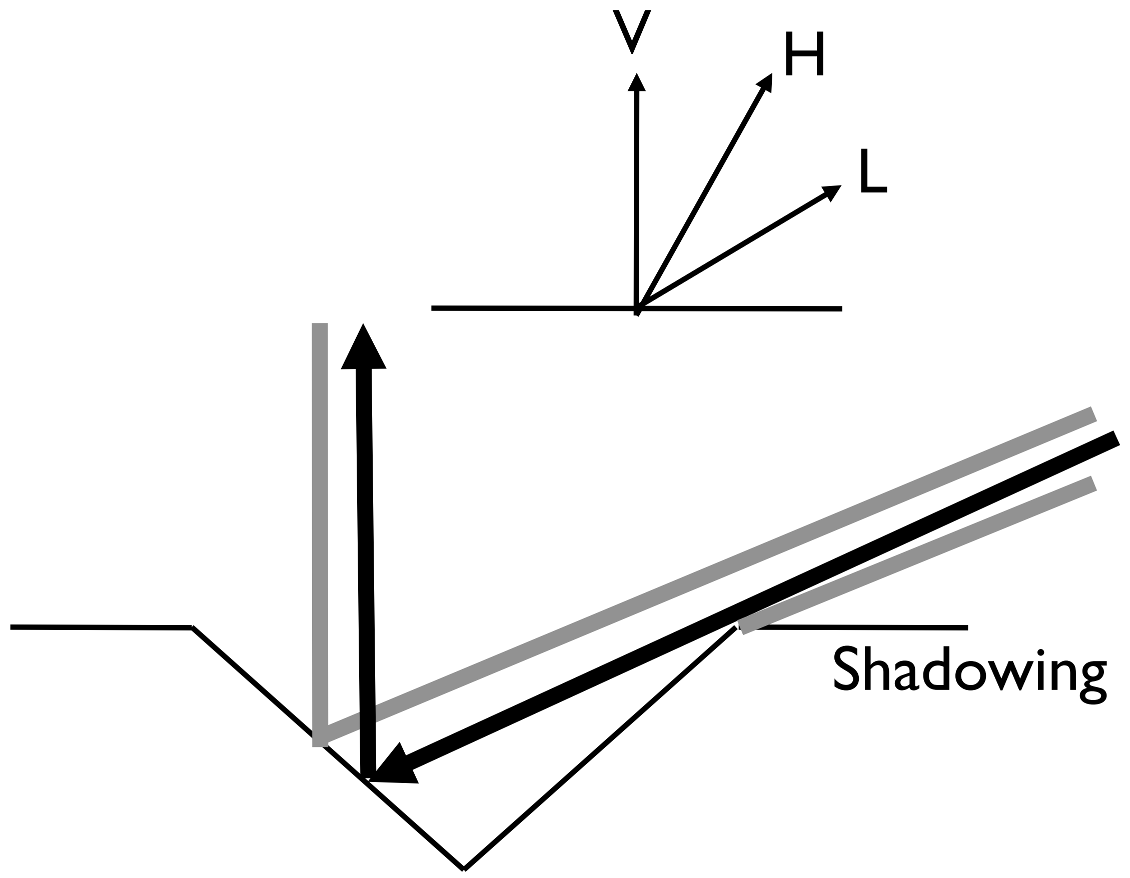


$$G_{masking} = \frac{2(N \bullet H)(N \bullet V)}{V \bullet H}$$

$$G_{shadowing} = \frac{2(N \bullet H)(N \bullet L)}{V \bullet H}$$

$$G = \min(1, G_{masking}, G_{shadowing})$$

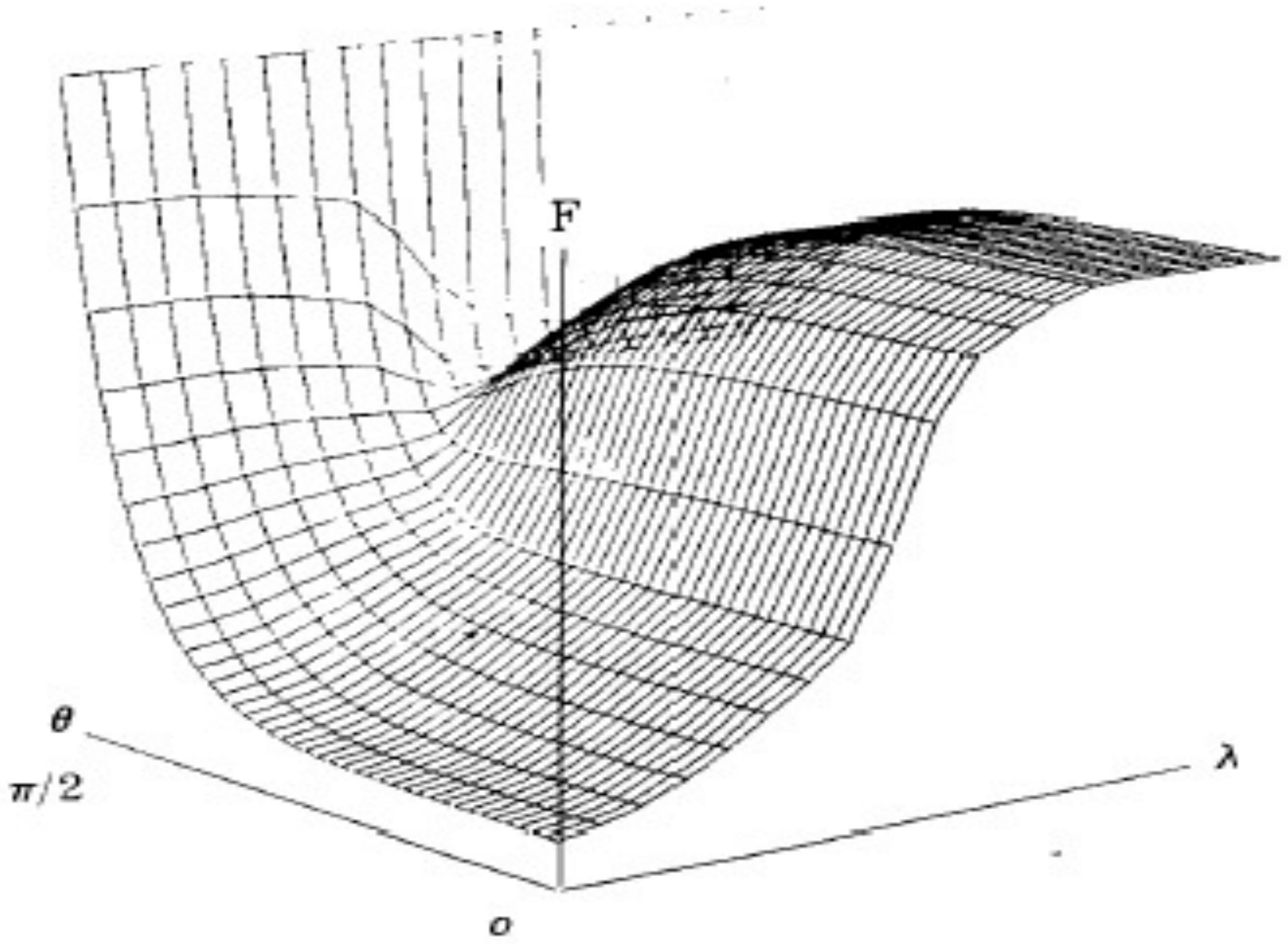




$$R = \frac{F}{\pi} \frac{D}{N \bullet L} \frac{G}{N \bullet V}$$

$$G = \min\left(1, \frac{2(N \bullet H)(N \bullet V)}{V \bullet H}, \frac{2(N \bullet H)(N \bullet L)}{V \bullet H} \right)$$

$$D = \frac{e^{-\left(\frac{\tan \beta}{m} \right)^2}}{4m^2 \cos^4 \beta}$$



The reflectance of copper as a function of wavelength and incidence angle.

Oren Nayar

- This model assumes rough surfaces to have microscopic grooves and hills.
- These are modelled mathematically as a collection of micro-facets having a statistical distribution of relative directions.
- This allows us to model materials such as Clay, stone etc.

Oren Nayar

$$L_r(\theta_r, \theta_i, \phi_r - \phi_i, \sigma) = \frac{\rho}{\pi} E_0 \cos\theta_i (A + B \max(0, \cos(\phi_i - \phi_r)) \sin\alpha \tan\beta)$$

- where

$$A = 1 - 0.5 \frac{\sigma^2}{\sigma^2 + 0.33}$$

$$B = 0.45 \frac{\sigma^2}{\sigma^2 + 0.09}$$

$$\alpha = \max(\theta_i, \theta_r)$$

$$\beta = \min(\theta_i, \theta_r)$$

ρ is the reflectivity of the surface (Kd*Cs)

E_o is the energy arriving from the light Cl

θ_i is the angle between the surface normal and the direction of the light source

θ_r is the angle between the surface normal and the vector in the direction of the viewer

$\phi_r - \phi_i$ is the angle (about the normal) between incoming and reflected light

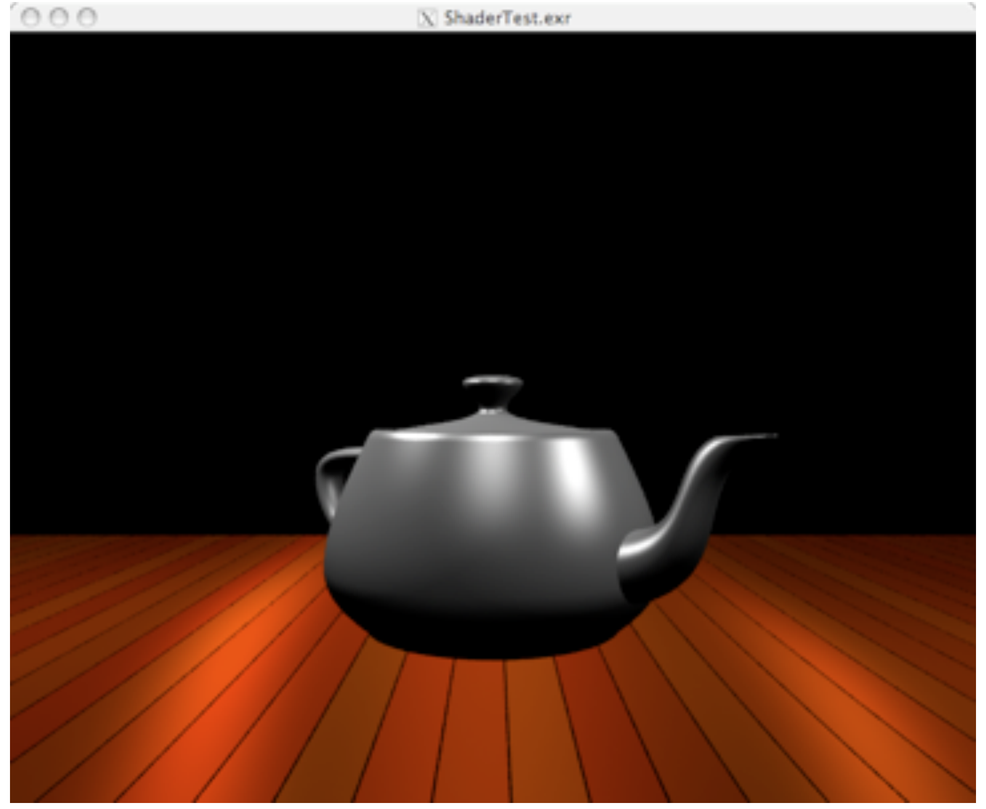
Oren Nayar

σ is the standard deviation of the angle distribution of the microfacets (in radians). Larger values represent more rough surfaces; smaller values repres smoother surfaces.

If $\sigma = 0$,the surface is perfectly smooth, and this function reduces to a simple Lambertian reflectance model.

This parameter is called "roughness"

```
color
LocIllumOrenNayar (normal N;  vector V;  float roughness;)
{
    // Surface roughness coefficients for Oren/Nayar's formula
    float sigma2 = roughness * roughness;
    float A = 1 - 0.5 * sigma2 / (sigma2 + 0.33);
    float B = 0.45 * sigma2 / (sigma2 + 0.09);
    // Useful precomputed quantities
    float  theta_r = acos (V . N);        // Angle between V and N
    vector V_perp_N = normalize(V-N*(V.N)); // Part of V perpendicular to N

    // Accumulate incoming radiance from lights in C
    color  C = 0;
    extern point P;
    illuminance (P, N, PI/2)
    {
    vector LN = normalize(L);
    float cos_theta_i = LN . N;
    float cos_phi_diff = V_perp_N . normalize(LN - N*cos_theta_i);
    float theta_i = acos (cos_theta_i);
    float alpha = max (theta_i, theta_r);
    float beta = min (theta_i, theta_r);
    C += 1 * Cl * cos_theta_i * (A + B * max(0,cos_phi_diff) * sin(alpha) *
        tan(beta));
    }
    return C;
}
```



Ward Anistropic

- The Ward anistropic model describes a surface similar to brushed metal where the machined grooves face in a particular direction.

$$\frac{1}{\sqrt{\cos\theta_i \cos\theta_r}} \frac{1}{4\pi\alpha_x\alpha_y} \exp\left[-2\frac{\left(\frac{\hat{h}\cdot\hat{x}}{\alpha_x}\right)^2 + \left(\frac{\hat{h}\cdot\hat{x}}{\alpha_y}\right)^2}{1+\hat{h}\cdot\hat{n}}\right]$$

θ_i is the angle between the surface normal and the direction of the light source

θ_r is the angle between the surface normal and the direction of the viewer

Ward Anistropic

$\hat{x}$ and $\hat{y}$ are the two perpendicular tangent directions of the surface

α_x and α_y are the standard deviations of the slope in the x and y directions called x and y roughness

$\hat{n}$ is the unit surface normal normalize(N)

$\hat{h}$ is the half angle between incident and reflection rays

H=normalize(normalize(-l)+normalize(L))

```
float sqr (float x) { return x*x; }

color LocIllumWardAnisotropic (
        normal N;  vector V;
                vector xdir;
                float xroughness;
                float yroughness;
        )
{

float cos_theta_r = clamp (N.V, 0.0001, 1);
vector X = xdir / xroughness;
vector Y = (N ^ xdir) / yroughness;

color C = 0;
illuminance (P, N, PI/2)
  {
  vector LN = normalize (L);
  float cos_theta_i = LN . N;
  if (cos_theta_i > 0.0)
    {
    vector H = normalize (V + LN);
    float rho = exp (-2 * (sqr(X.H) + sqr(Y.H)) / (1 + H.N))
    / sqrt (cos_theta_i * cos_theta_r);
    C += Cl * (  cos_theta_i * rho);
    }
  }
return C / (4 * xroughness * yroughness);
}
```

references

- Computer Graphics Principles and Practice. 2nd Ed. Foley, van Damm etc al Addison Wesley 1995
- Essential Renderman Fast. Ian Stephenson Springer-Verlag, 2003.
- [Digital] Lighting and Rendering Jeremy Birn New Riders 2000
- Larry Gritz Anthony A Apodaca. Advanced Renderman (Creating CGI for Motion Pictures). Morgan Kaufmann, 2000.

- http://en.wikipedia.org/wiki/Anistropic
- http://en.wikipedia.org/wiki/Brdf
- http://graphics.cs.ucdavis.edu/~bcbudge/ecs298_2004/ (Randall Rauwendaal BRDF.ppt)
- Ian Stephenson. Essential Renderman Fast. Springer-Verlag, 2003.
- Computer Graphics With OpenGL, F.S. Hill jr, Prentice Hall