Illumination and Shading

Photorealism

Realistic images are important for several reasons but the most important is that people generally find them easier to understand.

Hidden surface removal, illumination (lighting) and shading, texture mapping, shadows, transparency, improved synthetic cameras and better models all contribute to increasing visual realism.

A long standing goal of computer graphics has been *photorealism* — the production of synthetic images indistinguishable from photographs of real scenes.

We now have *rendering* techniques — ray tracing and radiosity — in computer graphics which approach photorealism for certain scenes.

However these techniques are slow, far too slow for interactive work.

One trade-off which occurs in computer graphics applications is image quality versus image generation speed (or frame rate).

Research continues on both better techniques and speed improvements.

The hidden surface removal techniques, the z-buffer in particular, and the illumination (lighting) and shading techniques, such as Phong illumination and Gouraud shading, which we typically find in today's 3D graphics libraries, such as OpenGL, were first developed in the 1970s and improved thereafter.

Illumination and Shading

To produce images which look, at least to some extent, *realistic*, objects must be *lit* and *shaded*.

An *illumination* or *lighting* model is a model or technique for determining the colour of a surface of an object at a given *point*.

A *shading* model is a broader framework which determines how an illumination model is used and what parameters it receives. For instance, the illumination model may be used for every pixel covered by an object or just for the its vertices.

The basis of the calculations for shading objects is the interaction of light and the objects in an environment.

The physics of this interaction is complicated. Attempts to use calculations based on models from physics typically result in slow execution times.

What is typically used in today's interactive 3D graphics libraries — Phong illumination and Gouraud

shading — are techniques partly based on very simplified physics and partly based on hacks which work reasonably well in practice.

These simple techniques are able to produce attractive and useful results in interactive time frames (at least several frames/second) and with modern hardware acceleration real-time rates (say 30-60 frames/second).

A Simple Illumination Model

A simple illumination model (called a lighting model in OpenGL) can be based on three components: ambient reflection, diffuse reflection and specular reflection. The model we look at is known as the Phong model. The OpenGL lighting model has the components of the Phong model, but is more elaborate.

Ambient Reflection

Imagine a uniform intensity background light which illuminates all objects in the scene equally from all directions. This is known as *ambient* light.

Ignore colour for a while and assume monochrome (grey) ambient light. The ambient reflection can be expressed as

$$I = L_a k_a$$

where L_a is the intensity of the ambient light and k_a is the *coefficient of ambient reflection* of the object's material and ranges between $[0, 1]$.

The object's coefficient of ambient reflection is a *material property*.

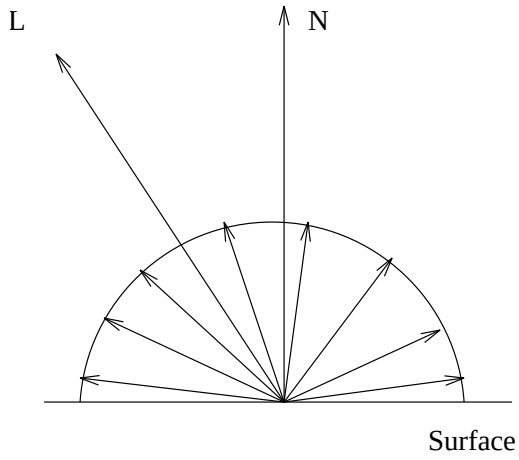
The ambient light model attempts to capture the effect of light which results from many object-object reflections without detailed calculations, that is, achieve computational efficiency. Those detailed calculations are performed in some rendering techniques such as radiosity which attempt to achieve higher quality images, but not real-time (yet!).

Diffuse Reflection

Objects illuminated by only ambient light have equal intensity everywhere.

If there are *light sources* in a scene then different objects should have different intensities based on distance and orientation with respect to the light source and the viewer.

A point on a diffuse surface appears equally bright from all viewing positions because it reflects light equally in all directions. That is, its intensity is independent of the position of the viewer.

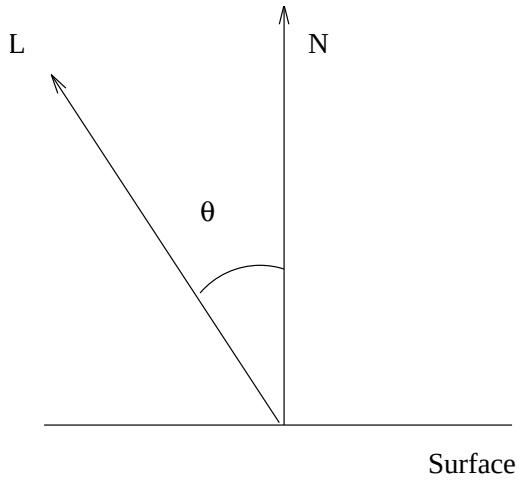


Whilst independent of the viewer, the intensity of a point on a diffuse surface does depend on the

orientation of the surface with respect to the light source and the distance to the light source.

A simple model for diffuse reflection is *Lambertian reflection*.

Assuming the following geometry



then the intensity of the diffuse reflection from a point light source is

$$I = L_d k_d cos\theta$$

where L_d is the intensity of the (point) light source, k_d is the diffuse reflection coefficient of the object's material, and θ is the angle between the normal to the surface and the light source direction vector.

If the normal vector to the surface $\mathbf{N}$ and the light source direction vector $\mathbf{L}$ are both normalised then the above equation can be simplified to

$$I = L_d k_d(\mathbf{N}.\mathbf{L})$$

If a light source is an infinite distance from the object then $\mathbf{L}$ will be the same for all points on the object — the light source becomes a *directional* light source. In this case less computation can be performed.

Adding the ambient reflection and diffuse reflection contributions together we get

$$I = L_a k_a + L_d k_d(\mathbf{N}.\mathbf{L})$$

Light Source Attenuation

The brightness of an object should depend not just on orientation but also on distance. This is called *light source attenuation*.

From physics we know that the intensity of a light source follows an inverse square law. If we introduce a light source *attenuation* factor

$$f_{att} = 1/d_L^2$$

then the illumination model becomes

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

However, this attenuation factor gives results which are too severe in practice. Instead an attenuation factor of the form is typically used

$$f_{att} = min(\frac{1}{c_1 + c_2 d_L + c_3 d_L^2}, 1)$$

where c_3 is often 0.0 (meaning it is *not* an inverse square relationship but a linear or constant one).

Colour

So far our illumination equation has not included any mention of colour. We introduce colour by giving each object (or its material) an ambient and diffuse colour and each light source a colour.

There are many different ways of specifying colour, or *colour models*. The most common is the RGB colour model where a colour is specified in terms of red, green and blue colour components.

If we use the RGB colour model then the ambient colour (reflectivity) of an object is (k_{aR}, k_{aG}, k_{aB}), the diffuse colour (reflectivity) of an object k_d is (k_{dR}, k_{dG}, k_{dB}) and the colour of the light emitted from the point light source as (L_{dR}, L_{dG}, L_{dB}).

Then the lighting equation becomes for say the red component

$$I_R = L_{aR}k_{aR} + f_{att}L_{dR}k_{dR}(\mathbf{N}.\mathbf{L})$$

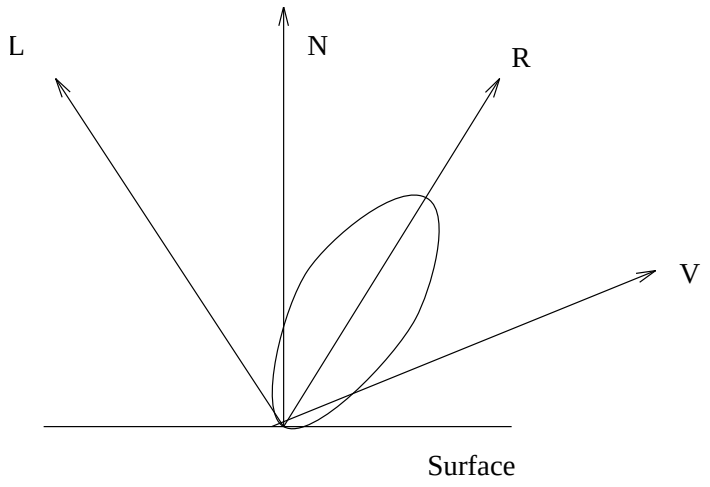
More generally, using wavelength λ for colour component gives

$$I_\lambda = L_{a\lambda}k_{a\lambda} + f_{att}L_{d\lambda}k_{d\lambda}(\mathbf{N}.\mathbf{L})$$

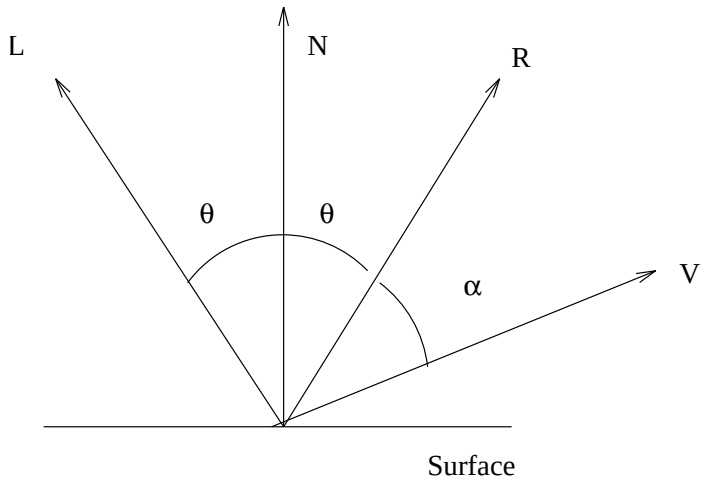
Specular Reflection

Specular reflection occurs on hard, shiny surfaces and is seen as highlights.

Specular highlights are strongly directional



The approach to specular reflection in the *Phong* model is that the specular reflection intensity drops off as the cosine of the angle between the normal and the specular reflection direction raised to some power n which indicates the *shininess* of the surface. The higher the power of n the smaller and brighter the highlight.



The specular component of the illumination model may thus be given as

$$I = f_{att} L_{s\lambda} k_{s\lambda} cos^n \alpha$$

If the direction of (specular) reflection $\mathbf{R}$ and the viewpoint direction $\mathbf{V}$ are normalised then the equation becomes

$$I = f_{att} L_{s\lambda} k_{s\lambda} (\mathbf{R}.\mathbf{V})^n$$

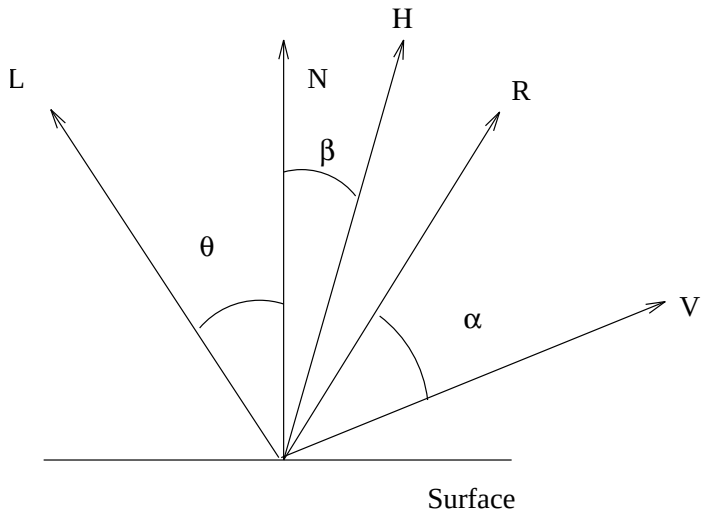
The full lighting equation becomes

$$I_\lambda = L_{a\lambda} k_{a\lambda} + f_{att}[L_{d\lambda} k_{d\lambda} (\mathbf{N}.\mathbf{L}) + L_{s\lambda} k_{s\lambda} (\mathbf{R}.\mathbf{V})^n]$$

Blinn-Phong Lighting: Halfway Vector

A modification to the Phong specular lighting model proposed by Blinn is to use the *halfway vector* instead of the *reflection vector*.

The halfway vector is the vector which lies half-way between the view vector and the light vector



The half-way vector is cheaper to compute than the reflection vector — if the light source is at infinity (directional light) and likewise the viewer is at infinity.

Multiple Light Sources

The simplest approach to handling multiple light sources is to just add the contribution from each light source. If there are l light sources the lighting equation becomes

$$I_\lambda = L_{a\lambda}k_{a\lambda} + \sum^{l} f_{att}[L_{d\lambda}k_{d\lambda}(\mathbf{N}.\mathbf{L}) + L_{s\lambda}k_{s\lambda}(\mathbf{R}.\mathbf{V})^n]]$$

A problem can now occur. The resulting intensity may be too high to display as is it simply accumulates. There are two common approaches:

1. clamp all RGB component values between [0,1]

2. compute the whole image and scale values according to the range of values in the image to fit the displayable range.

The first approach is more common (more efficient).

Shading

Illumination or lighting models determine the colour of a point on the surface of an object.

A *shading* model determines where the lighting model is applied.

Most common 3D graphics libraries are polygon based. The shading models are thus also polygon based.

The most common forms of shading are

Constant or Flat shading The lighting model is applied once for a whole polygon. The entire polygon is filled with the resulting colour. This is known as flat shading in OpenGL.

Gouraud shading The lighting model is applied at each vertex of the polygon. The polygon is filled by bi-linear interpolation of the resulting values. This is known as smooth shading in OpenGL.

Phong shading The lighting model is applied at every pixel covered by the polygon. The normal vector at each pixel is computed by bi-linear interpolation.

Gouraud and Phong shading are used to make polygonal objects — which are inherently faceted — appear smooth.

Flat Shading

Flat shading fills a polygon with a single colour computed from one lighting calculation.

Flat shading is appropriate under some conditions:

1. The light source is at infinity so $\mathbf{N}.\mathbf{L}$ is constant for all points on the polygon.

2. The viewer is at infinity so $\mathbf{N}.\mathbf{V}$

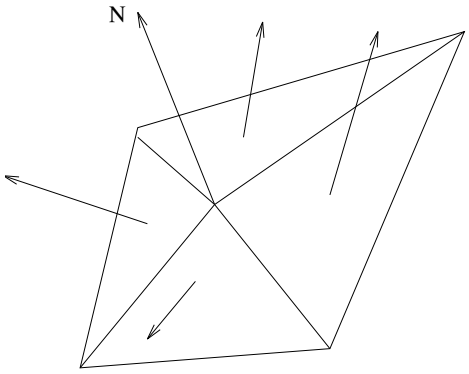
3. The object is indeed faceted and not an approximation to a curved object.

Gouraud Shading

Gouraud shading is a form of *interpolated shading*.

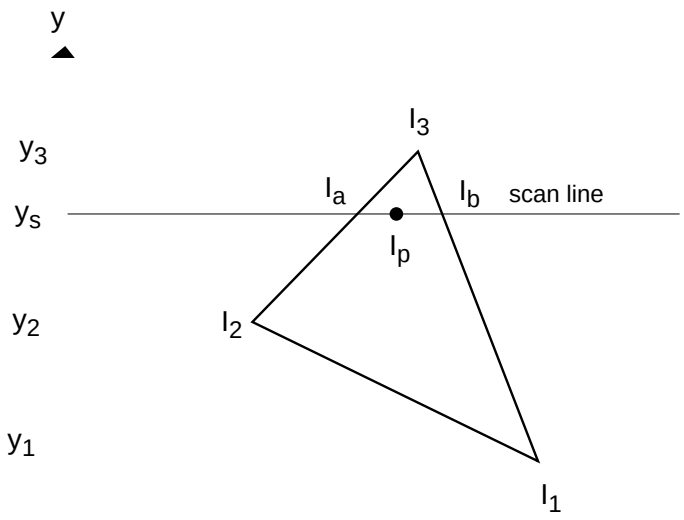
The illumination model is applied at vertices and the rest of the polygon's pixels are determined by bi-linear interpolation.

If a polygonal object is an approximation of a curved object, then the normal at a vertex is usually the average of the normals of the polygons which meet at that vertex.



Bi-linear Interpolation

Bi-linear interpolation is performed as follows

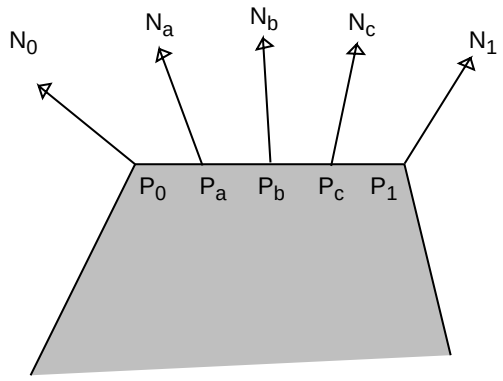


$$
\begin{aligned}
I_a &= I_2 + (\frac{y_s - y_2}{y_3 - y_2})(I_3 - I_2) \\
I_b &= I_1 + (\frac{y_s - y_1}{y_3 - y_1})(I_3 - I_1) \\
I_p &= I_a + (\frac{x_p - x_a}{x_b - x_a})(I_b - I_a)
\end{aligned}
$$

Gouraud shading permits polygonal models of curved objects to look smooth.

Phong Shading

Phong shading is another interpolated technique. The normal vector is interpolated across the surface and the illumination model is applied for each pixel.



It is typically much slower than Gouraud shading unless a machine has dedicated hardware.

Phong shading does a much better job of handling highlights than Gouraud shading. Gouraud shading for instance will miss highlights in the middle of polygons as it only applies the lighting model at the vertices. This can be overcome to some extent by using more polygons.

Gouraud shading in practice does a reasonable job for interactive computer graphics where realism is sacrificed for interactive update.

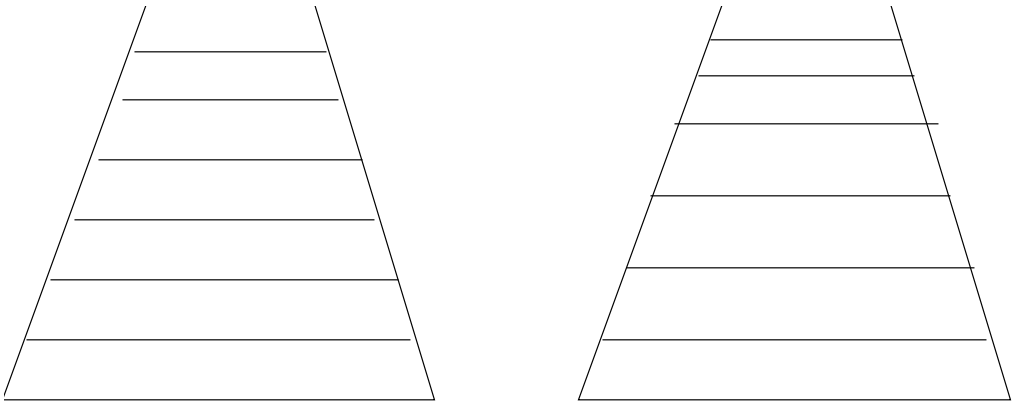
Interpolated Shading Problems

Gouraud and Phong shading are mainly intended to make polygonal objects look curved. They ultimately fail.

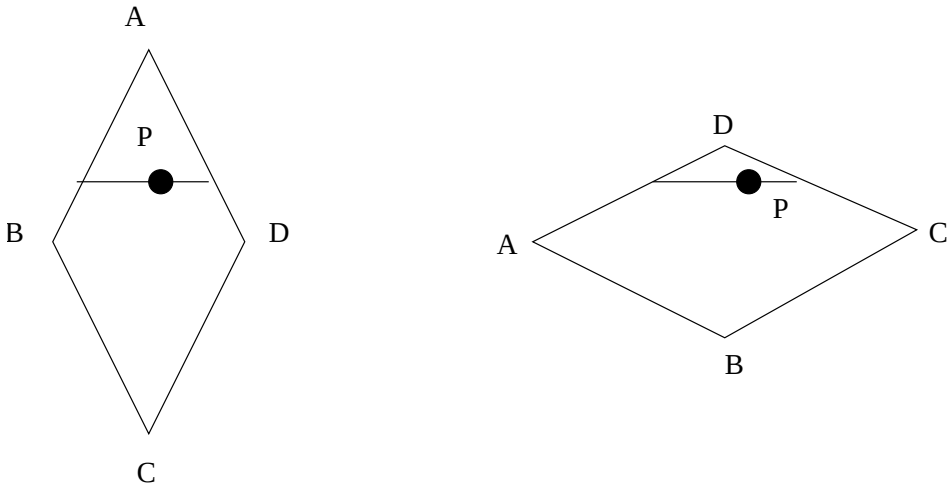
Here are some problems and failings:

Polygon silhoutte The silhoute edges of a polygonal model remain clearly polygonal.

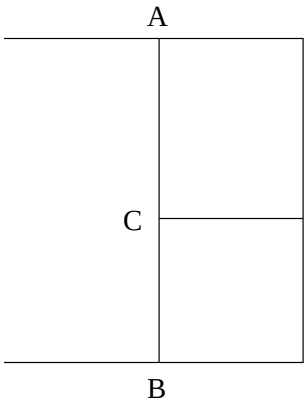
Perspective distortion Interpolation is performed after perspective transformation in normalised device coordinates rather than world coordinates.



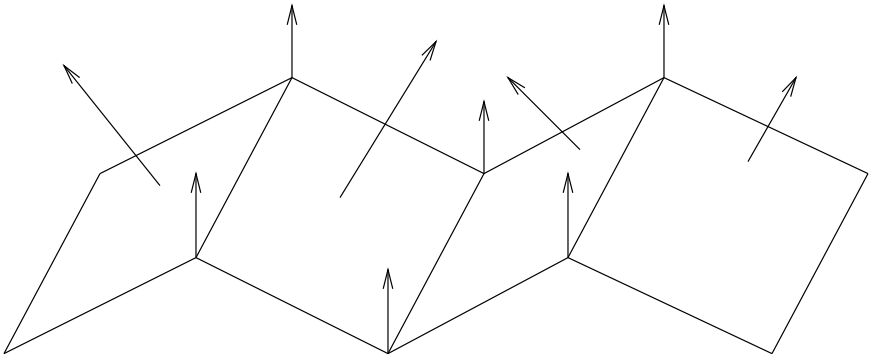
Orientation dependence The colour of a pixel may change when an object is rotated.



Shared vertices Discontinuities can occur at shared vertices. In the diagram below the large polygon does not include vertex C.



Unrepresentative normals The averaging of the polygon normals incident at a vertex may not represent the true surface orientation.

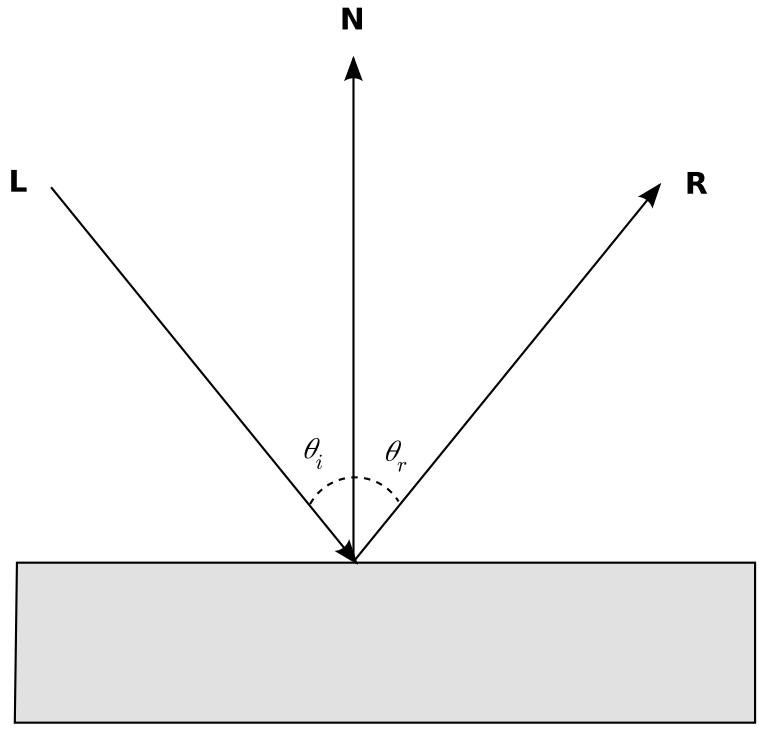


Reflection Vector

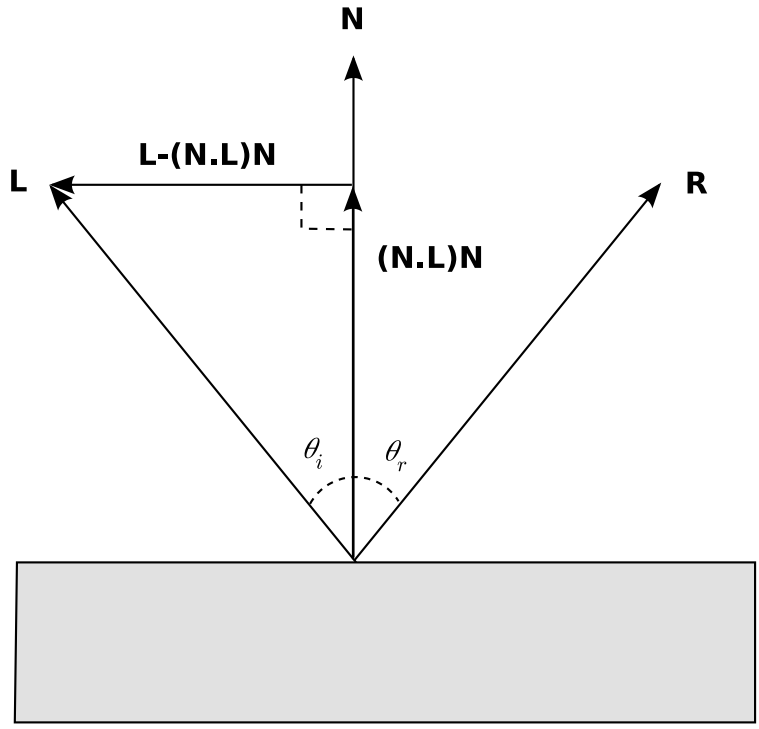
The reflection vector is needed to calculate the specular reflection contribution to lighting. The reflection vector is the direction of light reflected from the point on the surface it strikes.

The reflection vector $\mathbf{R}$ is calculated according to the *Law of Reflection*: the angle of incidence is equal to the angle of reflection and the reflection vector lies in the plane of incidence.

$$\theta_i = \theta_r$$



The reflection vector may be calculated as follows:



The light vector $\mathbf{L}$ is now shown as the vector pointing *to* the light source instead of showing the direction of incident light *from* the light source. Be carefull about this!

Any vector may be resolved into a *projected* component and a *perpendicular* component.

The projected component of $\mathbf{L}$ in the direction of the normal vector is

$$proj_{\mathbf{N}}\mathbf{L} = (\mathbf{N}.\mathbf{L})\mathbf{N}$$

The perpendicular component of $\mathbf{L}$ is then found by difference

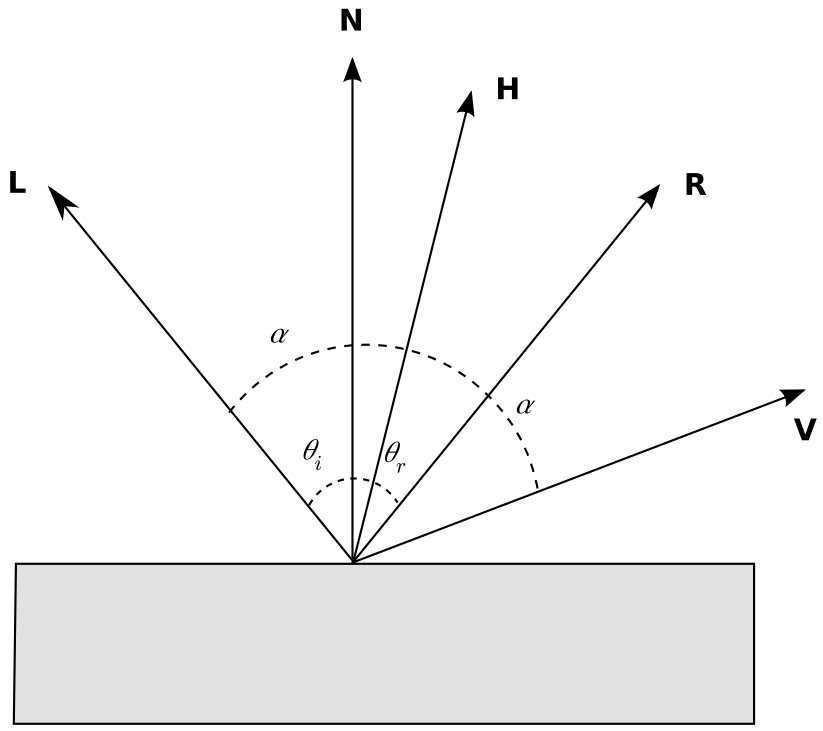
$$perp_{\mathbf{N}}\mathbf{L} = \mathbf{L} - proj_{\mathbf{N}}\mathbf{L} = \mathbf{L} - (\mathbf{N}.\mathbf{L})\mathbf{N}$$

The reflection vector is then

$$\begin{aligned} \mathbf{R} &= \mathbf{L} - 2perp_{\mathbf{N}}\mathbf{L}) \\ &= \mathbf{L} - 2[\mathbf{L} - (\mathbf{N}.\mathbf{L})\mathbf{N}] \\ &= 2(\mathbf{N}.\mathbf{L})\mathbf{N} - \mathbf{L} \end{aligned}$$

Halfway Vector

The halfway vector $\mathbf{H}$ is the vector halfway between the view vector $\mathbf{V}$ and the light vector $\mathbf{L}$



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

The halfway vector is used in the *Blinn-Phong*

lighting model for specular reflection. It is used primarily for computational efficiency.

Instead of using $\mathbf{R}.\mathbf{V}$, which involves calculating the reflection vector, $\mathbf{N}.\mathbf{H}$ is used instead.

This is quicker if the viewer and the light source are at infinity, in which case $\mathbf{L}$ and $\mathbf{V}$ do not change across a surface and $\mathbf{H}$ is a constant.

Because the $\mathbf{R}.\mathbf{V}$ term involves the reflection vector, it does change across a surface even if the light source and viewer are at infinity, meaning it must be recalculated at each point where the lighting calculation is applied.