Lighting

Rendering and Realism I

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

Realistic images are important for several reasons, including they just look better but also that people generally find them easier to understand.

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

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

Rendering and Realism II

Ray tracing and radiosity are *rendering* techniques which approach photorealism for certain scenes. However these techniques are too slow for real-time applications.

Research continues on improving advanced rendering techniques and their performance.

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.

A Simple Illumination Model

A simple lighting/illumination model can be based on three components:

- Ambient lighting
- Diffuse lighting
- Specular lighting

Ambient Lighting

Ambient light is a uniform intensity background light which illuminates all objects in the scene equally from all directions.

The ambient reflection can be modelled and calculated as

$$A = L_a M_a$$

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

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

Objects or scenes illuminated by only ambient light have equal intensity (or brightness) everywhere.

Color

Light sources and objects have color. Assume the RGB colour model is used.

If the ambient light color is (L_{aR}, L_{aG}, L_{aB}) and the object's ambient color (reflectivity) is (M_{aR}, M_{aG}, M_{aB}).

The ambient lighting equation

$$A = L_a M_a$$

is actually applied to each colour component

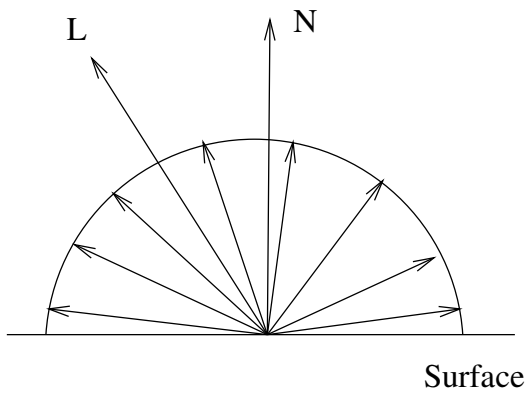
$$A_R = L_{aR} M_{aR} \tag{1}$$
$$A_G = L_{aG} M_{aG} \tag{2}$$
$$A_B = L_{aB} M_{aB} \tag{3}$$

Diffuse Lighting I

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

Diffuse lighting is one kind of lighting which has these properties.

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.

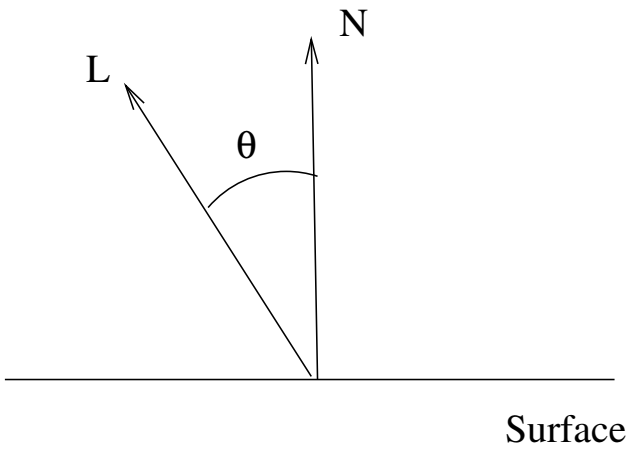


Diffuse Lighting II

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



Diffuse Lighting III

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

$$D = L_d M_d cos\theta$$

where L_d is the intensity of the (point) light source, M_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 **N** and the light source direction vector **L** are both normalised then the above equation can be simplified to

$$D = L_d M_d(\mathbf{N}.\mathbf{L})$$

Directional Light

If a light source is an infinite distance from the object then **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 as **L** is a constant.

Light Source Attenuation

From physics we know that the intensity of a light source reduces with distance, following an inverse square law. If we introduce a light source *attenuation* factor

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

then the diffuse component becomes

$$D = f_{att} L_d M_d (\mathbf{N}.\mathbf{L})$$

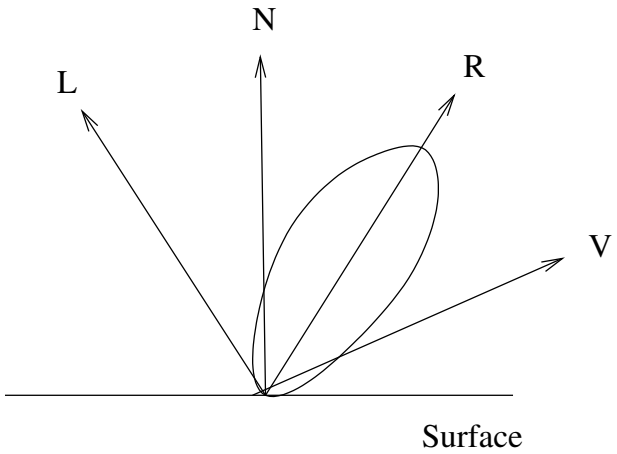
More generally, an attenuation function of the following form is allowed

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

Specular Reflection I

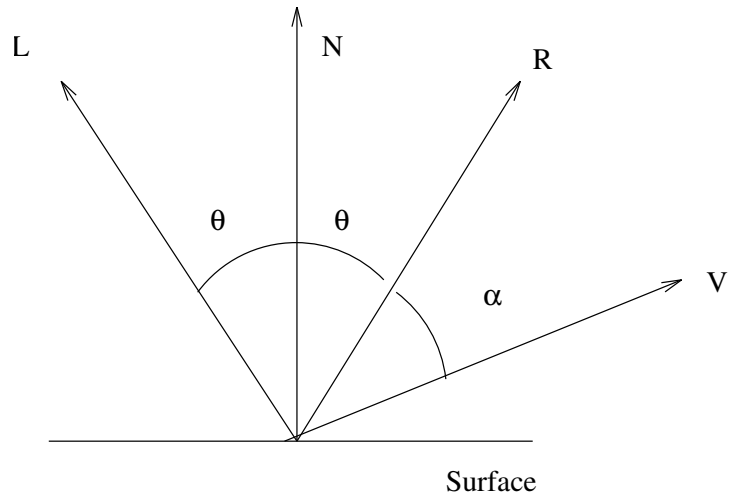
Specular reflection occurs on hard, shiny surfaces and is seen as highlights.

Specular highlights are strongly directional



Specular Reflection II

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.



Specular Reflection III

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

$$S = f_{att} L_s M_s cos^n \alpha$$

If the direction of (specular) reflection **R** and the viewpoint direction **V** are normalised then the equation becomes

$$S = f_{att} L_s M_s (\mathbf{R}.\mathbf{V})^n$$

Full Lighting Model

The full lighting model (equation) becomes

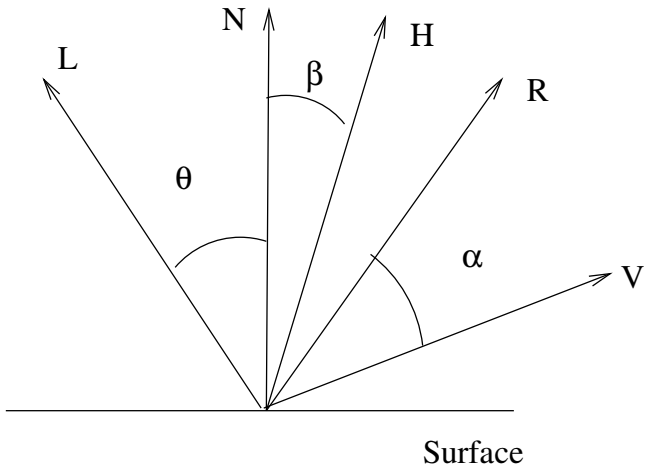
$$C = A + D + S = L_a M_a + f_{att}[L_d M_d(\mathbf{N}.\mathbf{L}) + L_s M_s(\mathbf{R}.\mathbf{V})^n]$$

where C is the final color.

Blinn-Phong Lighting: Halfway Vector

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

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



Then **N**.**H** is used instead of **R**.**V**.

The halfway vector is used primarily for computational efficiency.

Multiple Light Sources I

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

$$C = L_a M_a + \sum^{l} f_{att}[L_d M_d(\mathbf{N}.\mathbf{L}) + L_s M_s(\mathbf{R}.\mathbf{V})^n]]$$

A slight elaboration is to allow each light source to contribute ambient light.

This is basically the lighting model in OpenGL, except that specular lighting uses Blinn-Phong rather than Phong.

Clamping

The lighting model adds ambient, diffuse and specular lighting contributions, and for multiple light sources.

A problem which can occur is that the resulting intensity (colour) 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, and more efficient. As more and more lights are added it means objects tend to become white, i.e. (1, 1, 1) which is the 'maximum' color which can be displayed.

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. Lighting can be applied per-polygon, per-vertex or per-pixel.

Per-Polygon, Per-Vertex and Per-Pixel Shading

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

Per-Vertex 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. Also known as Gouraud shading.

Per-Pixel The lighting model is applied at every pixel covered by the polygon. In *Phong shading* the normal vector at each pixel is computed using bi-linear interpolation and then Phong specular lighting applied.

Per-vertex (and per-pixel) shading are used to make polygonal objects — which are inherently faceted — appear smooth.

Per Polygon or Flat Shading

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

Flat shading is appropriate under some conditions:

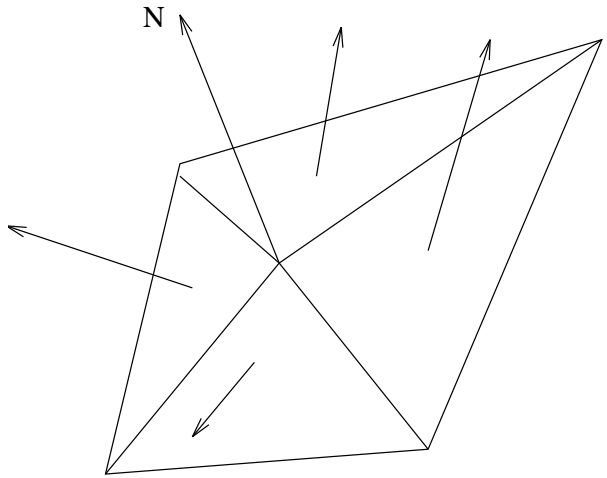
1. The light source is at infinity so **N**.**L** is constant for all points on the polygon.
2. The viewer is at infinity so **N**.**V** is the same for all vertices.
3. The object is indeed faceted and not an approximation to a curved object.

Per Vertex or Gouraud Shading

Gouraud shading is a form of *interpolated shading*.

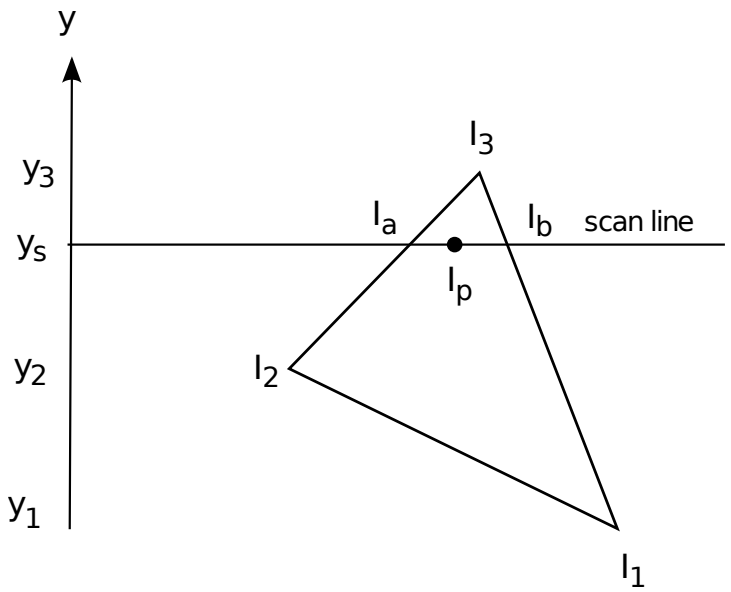
The illumination model is applied at vertices and the rest of the polygon's pixels are determined by bi-linear interpolation.

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

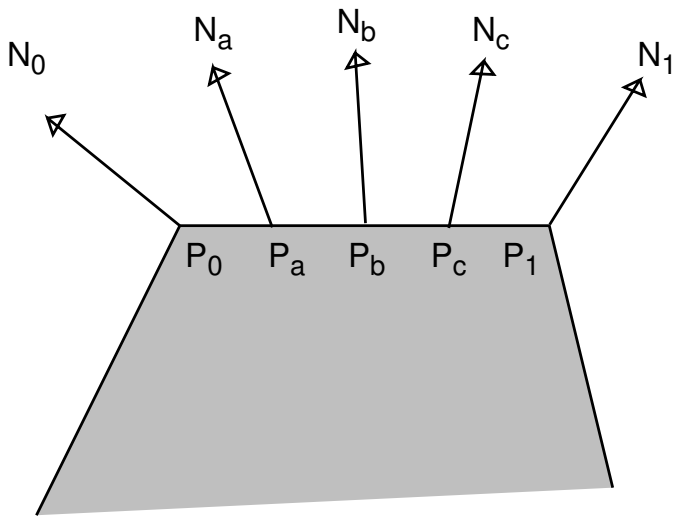
Bi-linear interpolation is performed as follows



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

Phong Shading I

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



Phong Shading II

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. However, this can be overcome by using more polygons.

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

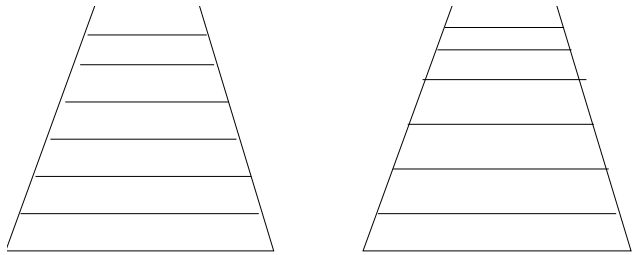
Interpolated Shading Problems I

Gouraud and Phong shading are mainly intended to make polygonal objects look curved. However there are problems and failings.

Here are some problems and failings:

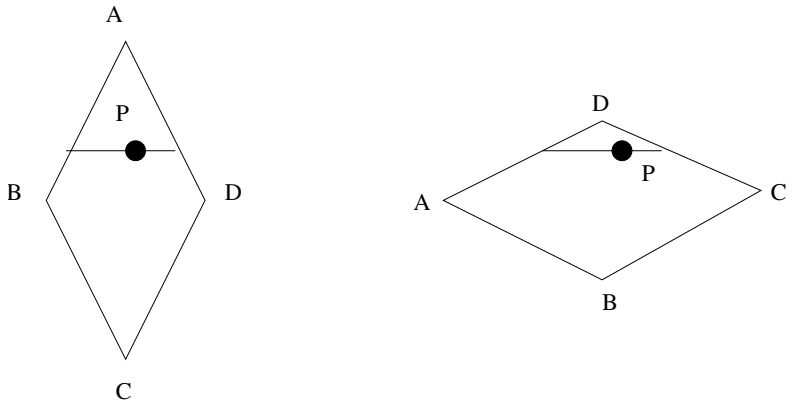
Polygon silhoutte The silhoute edges of a polygonal model remain clearly polygonal.

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

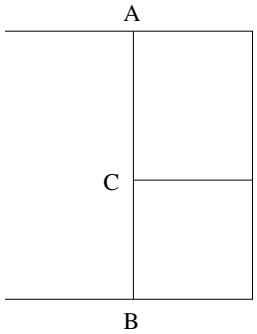


Interpolated Shading Problems II

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

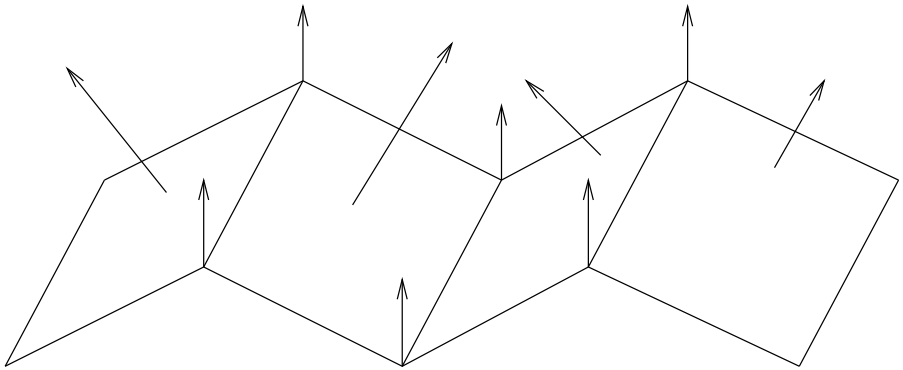


Shared vertices Discontinuities can occur at shared vertices. In the diagram below the large polygon does not include vertex C.



Interpolated Shading Problems III

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

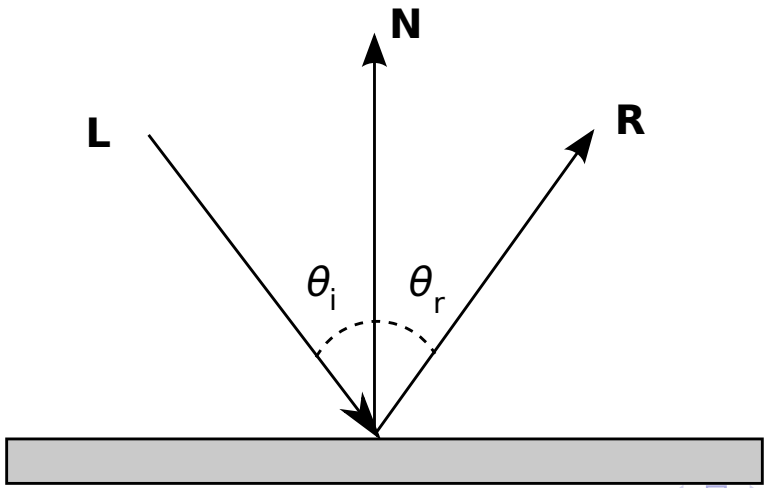


Reflection Vector I

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.

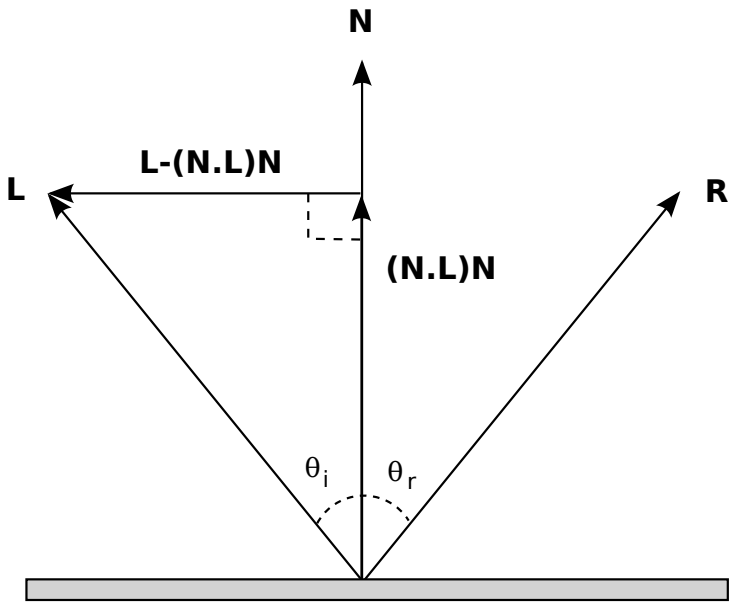
The reflection vector **R** is calculated according to the *Law of Reflection*: the angle of incidence is equal to the angle of reflection and the reflection vector lies in the plane of incidence.

$$\theta_i = \theta_r$$



Reflection Vector II

The reflection vector may be calculated as follows:



Reflection Vector III

The light vector **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 **L** in the direction of the normal vector is

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

The perpendicular component of **L** is then found by difference

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

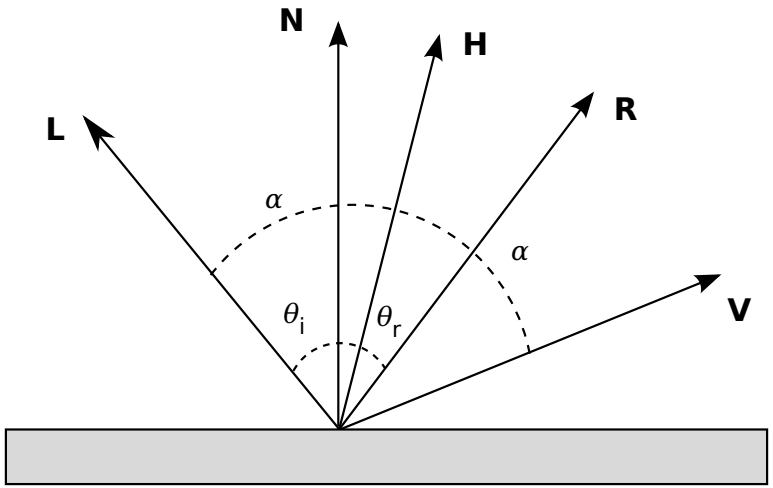
Reflection Vector IV

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 I

The halfway vector **H** is the vector halfway between the view vector **V** and the light vector **L**



Halfway Vector II

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

The halfway vector is used in the *Blinn-Phong* lighting model for specular reflection. It is used primarily for computational efficiency.

Instead of using **R**.**V**, which involves calculating the reflection vector, **N**.**H** is used instead.

This is quicker if the viewer and the light source are at infinity, in which case **L** and **V** do not change across a surface, i.e. the same for all vertices, and **H** is a constant.

Because the **R**.**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.