4. Illumination, Viewing, Projection

R. Mukundan,
Department of Computer Science and Software Engineering
University of Canterbury.

4.1 Lighting Calculations

The hardware accelerated lighting model that is traditionally used in computer graphics applications is based on Phong-Blinn approximation for an omni-directional point-light source. A local illumination model that does not account for complex effects such as reflections, refractions, shadows and indirect illumination is found to be generally adequate for a majority of graphics applications. In this model, light-material interaction is simply modelled using a component-wise multiplication of material colour and light colour. We can represent colour by a vector comprising of red, green and blue components as $\boldsymbol{c} = (r, g, b)$. In the discussion that follows, $\boldsymbol{M_a}$, $\boldsymbol{M_d}$, $\boldsymbol{M_s}$ denote respectively the ambient, diffuse and specular components of material colour, and $\boldsymbol{L_a}$, $\boldsymbol{L_d}$, $\boldsymbol{L_s}$ the corresponding components of the light source. Each of these colour components is typically a 3-tuple consisting of red, green and blue values. For notational convenience, we represent $\boldsymbol{M_a}$ by the vector (r_{ma}, g_{ma}, b_{ma}), $\boldsymbol{L_a}$ by the vector (r_{ia}, g_{ia}, b_{ia}), and so on. The ambient light-material interaction is then modelled by the component-wise vector product

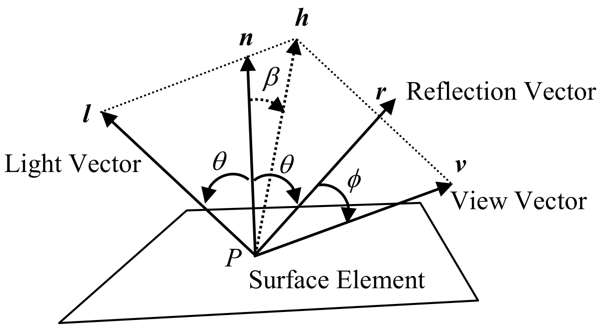
$$\boldsymbol{L_a M_a} = (r_{ma}r_{ia}\,,\ g_{ma}g_{ia}\,,\ b_{ma}b_{ia})$$

The figure below shows the geometry of unit vectors used for computing diffuse and specular reflections from a surface. From a point P on a surface, $\boldsymbol{l}$ denotes the unit vector towards the light source, $\boldsymbol{n}$ the unit surface normal vector, and $\boldsymbol{v}$ the unit vector towards the viewer. The perceived intensity of reflection at the viewer's position varies with changes in the angles between these vectors. The variations in diffuse and specular reflections are represented by multiplicative factors k_d and k_s respectively. According to the Lambertian reflectance model, the intensity of diffuse reflection from a surface is uniform in all directions, and varies as the cosine of the angle θ between the light source vector $\boldsymbol{l}$ and the surface normal vector $\boldsymbol{n}$, and is therefore proportional to $\boldsymbol{l \bullet n}$. If the angle between the two vectors is greater than 90 degrees, the normal vector faces away from the light source vector, and the surface is in shadow. In such a situation, the value of k_d must be set to 0. We therefore have the following view-independent factor for the diffuse term:

$$k_d = \max(\boldsymbol{l \bullet n}, 0)$$

The specular reflection factor k_s is computed as a function of the cosine of the angle ϕ between the unit specular reflection vector $\boldsymbol{r}$ and the unit view vector $\boldsymbol{v}$. The exponent f is known as the shininess term or the Phong's constant. The exponent is useful in controlling the concentration (or spread) of the specular highlight.

$$k_s = \max(\cos^f\phi, 0) = \max((\boldsymbol{r \bullet v})^f,\ 0)$$



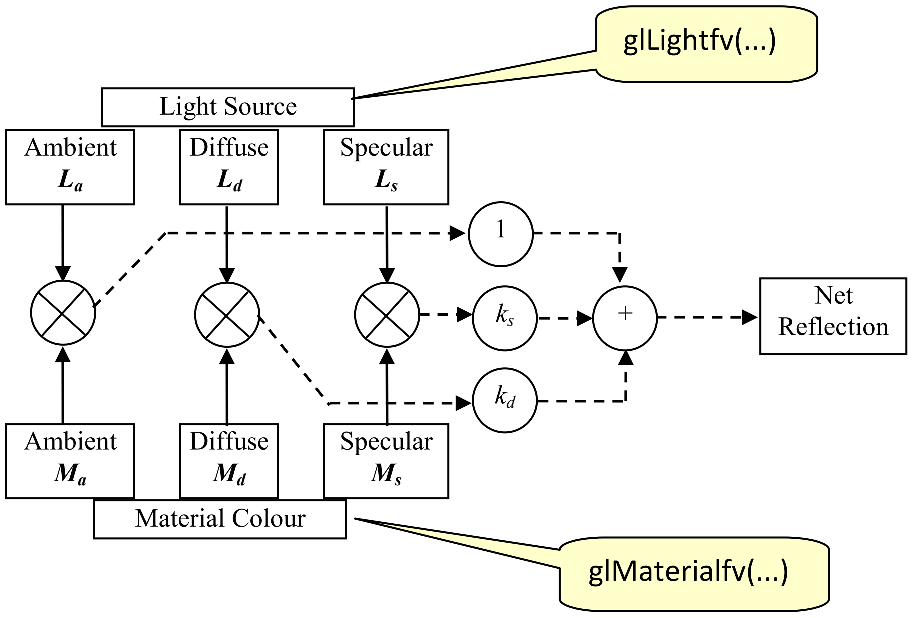
The reflection vector $\boldsymbol{r}$ is computed using the light source vector $\boldsymbol{l}$ and the normal vector $\boldsymbol{n}$ as $\boldsymbol{r} = 2(\boldsymbol{l}\bullet\boldsymbol{n})\,\boldsymbol{n} - \boldsymbol{l}$

The Blinn's approximation eliminates the need for computing the specular reflection vector $\boldsymbol{r}$ as shown above, by defining a unit vector $\boldsymbol{h}$ along the direction $\boldsymbol{l} + \boldsymbol{v}$. This vector is called the half-way vector. If the angle between $\boldsymbol{n}$ and $\boldsymbol{h}$ is denoted by β, then $\boldsymbol{n}\bullet\boldsymbol{h} = \cos\beta$. The Blinn model uses β as an approximation of ϕ, and the corresponding approximation for k_s given by

$$k_s = \max((\boldsymbol{n}\bullet\boldsymbol{h})^f,\ 0).$$

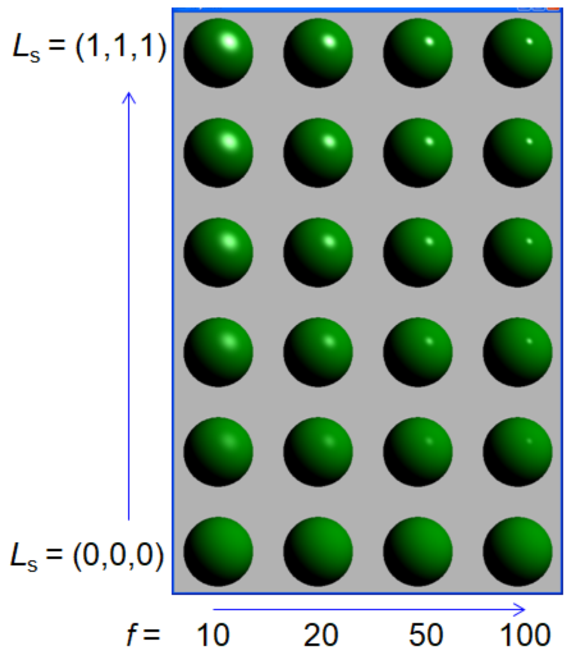
Specular reflections are seen only in the vicinity of the reflection vector, where the angle ϕ has a low value. When the angle ϕ is small, angle β is also small. It can be analytically proven than $\phi = 2\beta$. In particular, when $\phi = 0$, the value of β also becomes 0.

A schematic of the lighting computation using the Phong-Blinn illumination model outlined above is given in the following diagram.



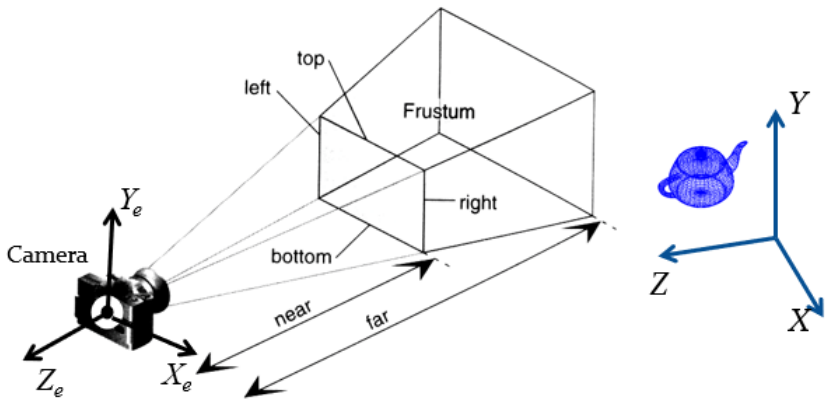
The specular material property $\boldsymbol{M}_s$ is generally set as white to get a bright specular highlight. The ambient and diffuse material properties $\boldsymbol{M}_a$, $\boldsymbol{M}_d$ usually have the same colour value as the material colour. A light's ambient colour $\boldsymbol{L}_a$ is usually set as a low intensity gray value (eg. (0.2, 0.2, 0..2)), and $\boldsymbol{L}_d$, $\boldsymbol{L}_s$ are commonly set to white.

The following figure shows the variations in the specular highlight with respect to changes in the light's specular colour $\boldsymbol{L}_s$ and the Phong's constant *f*.



4.2 View Transformation

The view transformation matrix $\boldsymbol{V}$ is created by the `gluLookAt()` function that specifies the position and the orientation of the camera. This matrix converts coordinates of vertices and components of vectors from the world coordinate frame (X, Y, Z) to the eye coordinate frame (X_e, Y_e, Z_e). The default setting of the view matrix is the identity matrix (default camera).



The camera's position ***e*** = (e_x, e_y, e_z) and the directions (unit vectors) along the camera axes can be easily extracted from the view matrix. If ***u*** = (u_x, u_y, u_z), ***v*** = (v_x, v_y, v_z), ***w*** = (w_x, w_y, w_z) are the unit vectors along X_e, Y_e, Z_e axes respectively, the view matrix is given by

$$\begin{bmatrix} u_x & u_y & u_z & -e_x u_x - e_y u_y - e_z u_z \\ v_x & v_y & v_z & -e_x v_x - e_y v_y - e_z v_z \\ w_x & w_y & w_z & -e_x w_x - e_y w_y - e_z w_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It can be easily verified that the above matrix transforms the point (e_x, e_y, e_z, 1) to the point (0, 0, 0, 1), the vector (u_x, u_y, u_z, 0) to (1, 0, 0, 0), the vector (v_x, v_y, v_z, 0) to (0, 1, 0, 0), and the vector (w_x, w_y, w_z, 0) to (0, 0, 1, 0). The above 4×4 view matrix is sometimes written in a compact form as

$$\begin{bmatrix} \boldsymbol{u} & -\boldsymbol{e}.\boldsymbol{u} \\ \boldsymbol{v} & -\boldsymbol{e}.\boldsymbol{v} \\ w & -\boldsymbol{e}.\boldsymbol{w} \\ 0 & 1 \end{bmatrix}$$

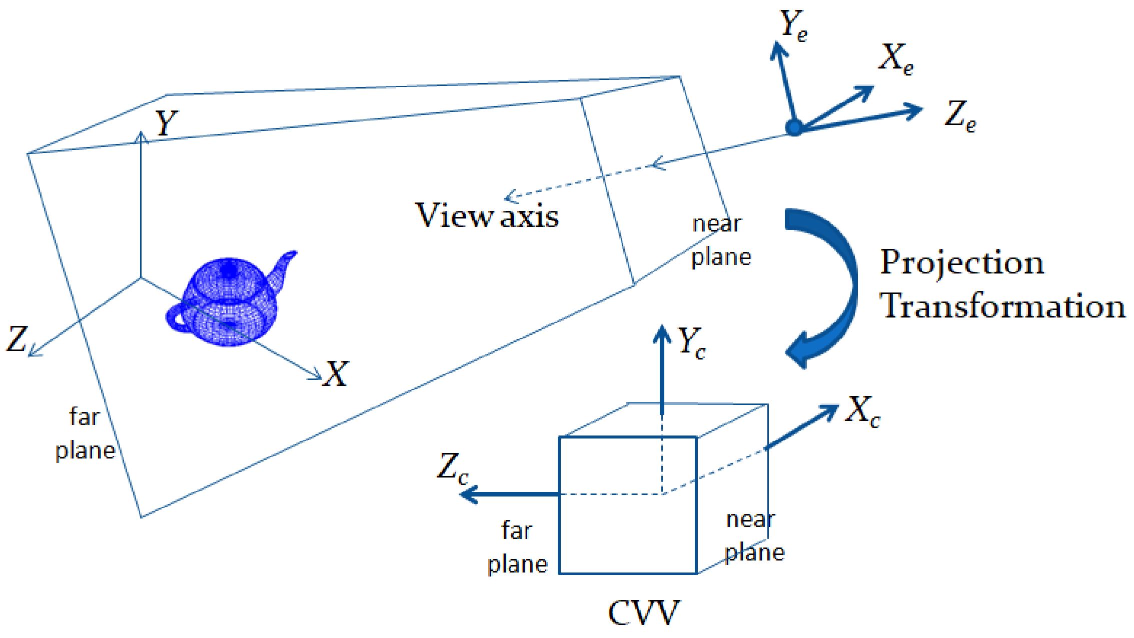
Lighting calculations are usually performed in the eye-coordinate space. If a vertex *P* has coordinates (*x*, *y*, *z*), and the camera is placed at (e_x, e_y, e_z), the view vector ***v*** at *P* is given by (e_x, e_y, e_z) − (*x*, *y*, *z*). In eye coordinates, this expression for the view vector becomes (0, 0, 0) – (x_e, y_e, z_e) = – (x_e, y_e, z_e).

4.3 Projection Transformation

OpenGL provides four functions for implementing projection models in a graphics application. These are:

- `gluOrtho2D(xmin, xmax, ymin, ymax)`: 2D Orthographic projection
- `glOrtho(xmin, xmax, ymin, ymax, zmin, zmax)`: 3D Orthographic projection
- `glFrustum(left, right, bottom, top, near, far)`: 3D Perspective projection
- `gluPerspective(for, aspect, near, far)`: 3D Perspective projection

These functions generate a 4×4 projection matrix that transform points and vectors from the eye coordinate space to the clip coordinate space. The projection matrix maps the view frustum to a cube with its sides located at unit distance from the centre. This cube is known as the canonical view volume (CVV). The regular structure of CVV is convenient for clipping primitives. The left-handed reference axes attached to the CVV are known as the clip coordinate axes. The projection matrix maps points within the view frustum to points within the CVV with clip coordinates in the range [−1, 1].



The z-coordinate of a point in clip space (z_c) denotes the pseudo-depth of the point. The pseudo depth of a point varies from -1 on the near plane to +1 on the far plane. The true depth d of a point in the range [0, 1] can be obtained from the pseudo-depth using a simple transformation, $d = (z_c + 1) / 2$.

The transformation matrices created in the OpenGL pipeline can be categorized into four classes:

- The model transformations are generated using functions such as `glRotatef()`, `glScalef()`, and `glTranslatef()`. The product of all transformations applied to a vertex is the **model matrix**.
- The **view matrix** is generated using the `gluLookAt()` function.
- The **projection matrix** is generated using one of the four functions defining the view volume, listed at the beginning of this section. All the above transformations (model, view, projection) are applied to vertices.
- The **viewport transformation matrix** is applied to fragments generated after the rasterization stage, and transforms clip coordinates to viewport coordinates.

$$\begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} View \\ Matrix \end{bmatrix} \begin{bmatrix} Model \\ Matrix \end{bmatrix}}_{\textbf{Model-View Matrix}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} Projection \\ Matrix \end{bmatrix} \begin{bmatrix} View \\ Matrix \end{bmatrix} \begin{bmatrix} Model \\ Matrix \end{bmatrix}}_{\textbf{Model-View-Projection Matrix}} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$