Chapter 7

INCREMENTAL SHADING TECHNIQUES

Incremental shading models take a very drastic approach to simplifying the rendering equation, namely eliminating all the factors which can cause multiple interdependence of the radiant intensities of different surfaces. To achieve this, they allow only coherent transmission (where the refraction index is 1), and incoherent reflection of the light from abstract lightsources, while ignoring the coherent and incoherent reflection of the light coming from other surfaces.

The reflection of the light from abstract lightsources can be evaluated without the intensity of other surfaces being known, so the dependence between them has been eliminated. In fact, coherent transmission is the only feature left which can introduce dependence, but only in one way, since only those objects can alter the image of a given object which are behind it, looking at the scene from the camera.

Suppose there are n_l abstract lightsources (either directional, positional or flood type) and that ambient light is also present in the virtual world. Since the flux of the abstract lightsources incident to a surface point can be easily calculated, simplifying the integrals to sums, the shading equation has the following form:

$$I^{\text{out}} = I_e + k_a \cdot I_a + k_t \cdot I_t + \sum_{l}^{n_l} r_l \cdot I_l \cdot k_d \cdot \cos\phi_{\text{in}} + \sum_{l}^{n_l} r_l \cdot I_l \cdot k_s \cdot \cos^n\psi \quad (7.1)$$

where k_a is the reflection coefficient of the ambient light, k_t, k_d and k_s are the

transmission, diffuse and specular coefficients respectively, ϕ_{in} is the angle between the direction of the lightsource and the surface normal, ψ is the angle between the viewing vector and the mirror direction of the incident light beam, n is the specular exponent, I_l and I_a are the incident intensities of the normal and ambient lightsources at the given point, I_t is the intensity of the surface behind a transmissive object, I_e is the own emission, and r_l is the shadow factor representing whether a lightsource can radiate light onto the given point, or whether the energy of the beam is attenuated by transparent objects, or whether the point is in shadow, because another opaque object is hiding it from the lightsource:

$$r_l = \begin{cases} 1 \text{ if the lightsource } l \text{ is visible from this point} \\ \prod_i k_t^{(i)} \text{ if the lightsource is masked by transparent objects} \\ 0 \text{ if the lightsource is hidden by an opaque object} \end{cases} \tag{7.2}$$

where $k_t^{(1)}, k_t^{(2)}, \ldots, k_t^{(n)}$ are the transmission coefficients of the transparent objects between the surface point and lightsource l.

The factor r_l is primarily responsible for the generation of shadows on the image.

7.1 Shadow calculation

The determination of r_l is basically a visibility problem considering whether a lightsource is visible from the given surface point, or equally, whether the surface point is visible from the lightsource. Additionally, if there are transparent objects, the solution also has to determine the objects lying in the path of the beam from the lightsource to the surface point.

The second, more general case can be solved by ray-tracing, generating a ray from the surface point to the lightsource and calculating the intersections, if any, with other objects. In a simplified solution, however, where transparency is ignored in shadow calculations, that is where r_l can be either 0 or 1, theoretically any other visible surface algorithm can be applied setting the eye position to the lightsource, then determining the surface parts visible from there, and declaring the rest to be in shadow. The main difficulty of shadow algorithms is that they have to store the information

regarding which surface parts are in shadow until the shading calculation, or else that question has to be answered during shading of each surface point visible in a given pixel, preventing the use of coherence techniques and therefore limiting the possible visibility calculation alternatives to expensive ray-tracing.

An attractive alternative algorithm is based on the application of the z-buffer method, requiring additional z-buffers, so-called **shadow maps**, one for each lightsource (figure 7.1).

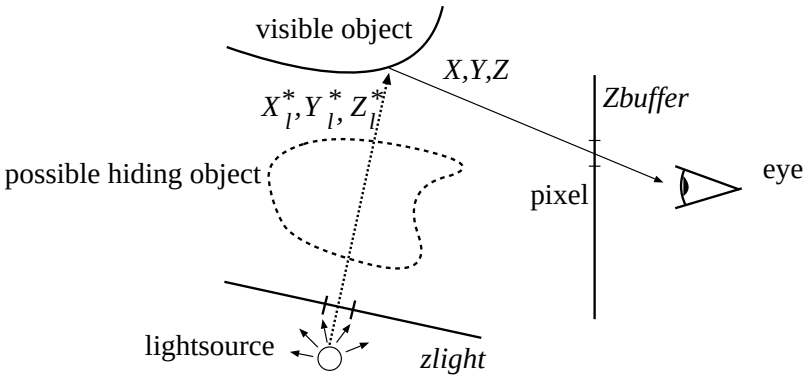

Figure 7.1: Shadow map method

The algorithm consists of a z-buffer step from each lightsource l setting the eye position to it and filling its shadow map $zlight_l[X,Y]$, then a single modified z-buffer step for the observer's eye position filling $Zbuffer[X,Y]$.

From the observer's eye position, having checked the visibility of the surface in the given pixel by the $Z_{\text{surface}}[X,Y] < Zbuffer[X,Y]$ inequality, the algorithm transforms the 3D point $(X, Y, Z_{\text{surface}}[X,Y])$ from the observer's eye coordinate system (screen coordinate system) to each lightsource coordinate system, resulting in:

$$(X, Y, Z_{\text{surface}}[X,Y]) \stackrel{T}{\Longrightarrow} (X^*_l, Y^*_l, Z^*_l). \tag{7.3}$$

If $Z^*_l > zlight[X^*_l, Y^*_l]$, then the surface point was not visible from the lightsource l, hence, with respect to this lightsource, it is in shadow ($r_l = 0$).

The calculation of shadows seems time consuming, as indeed it is. In many applications, especially in CAD, shadows are not vital, and they can

even confuse the observer, making it possible to speed up image generation by ignoring the shadows and assuming that $r_l = 1$.

7.2 Transparency

If there are no transparent objects image generation is quite straightforward for incremental shading models. By applying a hidden-surface algorithm, the surface visible in a pixel is determined, then the simplified shading equation is used to calculate the intensity of that surface, defining the color or (R, G, B) values of the pixel.

Should transparent objects exist, the surfaces have to be ordered in decreasing distance from the eye, and the shading equations have to be evaluated according to that order. Suppose the color of a "front" surface is being calculated, when the intensity of the "back" surface next to it is already available (I_{back}), as is the intensity of the front surface, taking only reflections into account ($I^{\text{ref}}_{\text{front}}$). The overall intensity of the front surface, containing both the reflective and transmissive components, is:

$$I_{\text{front}}[X, Y] = I^{\text{ref}}_{\text{front}} + k_t \cdot I_{\text{back}}[X, Y]. \tag{7.4}$$

The transmission coefficient, k_t, and the reflection coefficients are obviously not independent. If, for example, k_t were 1, all the reflection parameters should be 0. One way of eliminating that dependence is to introduce corrected reflection coefficients by dividing them by $(1-k_t)$, and calculating the reflection I^*_{front} with these corrected parameters. The overall intensity is then:

$$I_{\text{front}}[X, Y] = (1 - k_t) \cdot I^*_{\text{front}} + k_t \cdot I_{\text{back}}[X, Y]. \tag{7.5}$$

This formula can be supported by a pixel level trick. The surfaces can be rendered independently in order of their distance from the eye, and their images written into the frame buffer, making a weighted sum of the reflective surface color, and the color value already stored in the frame buffer (see also subsection 8.5.3 on support for translucency and dithering).

7.3 Application of the incremental concept in shading

So far, the simplified shading equation has been assumed to have been evaluated for each pixel and for the surface visible in this pixel, necessitating the determination of the surface normals to calculate the angles in the shading equation.

The speed of the shading could be significantly increased if it were possible to carry out the expensive computation just for a few points or pixels, and the rest could be approximated from these representative points by much simpler expressions. These techniques are based on linear (or in extreme case constant) approximation requiring a value and the derivatives of the function to be approximated, which leads to the incremental concept. These methods are efficient if the geometric properties can also be determined in a similar way, connecting incremental shading to the incremental visibility calculations of polygon mesh models. Only polygon mesh models are considered in this chapter, and should the geometry be given in a different form, it has to be approximated by a polygon mesh before the algorithms can be used. It is assumed that the geometry will be transformed to the screen coordinate system suitable for visibility calculations and projection.

There are three accepted degrees of approximation used in this problem:

1. **Constant shading** where the color of a polygon is approximated by a constant value, requiring the evaluation of the shading equation once for each polygon.

2. **Gouraud shading** where the color of a polygon is approximated by a linear function, requiring the evaluation of the shading equation at the vertices of the polygon. The color of the inner points is determined by incremental techniques suitable for linear approximation.

3. **Phong shading** where the normal vector of the surface is approximated by a linear function, requiring the calculation of the surface normal at the vertices of the polygon, and the evaluation of the shading equation for each pixel. Since the color of the pixels is a non-linear function of the surface normal, Phong shading is, in fact, a non-linear approximation of color.

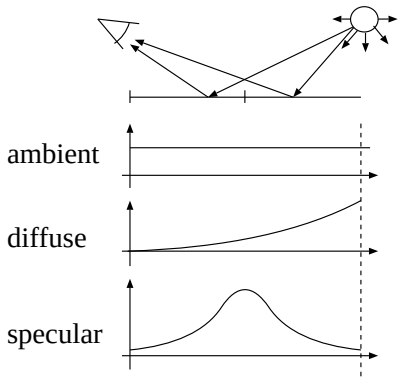

Figure 7.2: Typical functions of ambient, diffuse and specular components

In figure 7.2 the intensity distribution of a surface lit by positional and ambient lightsources is described in terms of ambient, diffuse and specular reflection components. It can be seen that ambient and diffuse components can be fairly well approximated by linear functions, but the specular term tends to show strong non-linearity if a highlight is detected on the surface. That means that constant shading is acceptable if the ambient lightsource is dominant, and Gouraud shading is satisfactory if k_s is negligible compared with k_d and k_a, or if there are no highlights on the surface due to the relative arrangement of the lightsources, the eye and the surface. If these conditions do not apply, then only Phong shading will be able to provide acceptable image free from artifacts.

Other features, such as shadow calculation, texture or bump mapping (see chapter 12), also introduce strong non-linearity of the intensity distribution over the surface, requiring the use of Phong shading to render the image.

7.4 Constant shading

When applying constant shading, the simplified rendering equation missing out the factors causing strong non-linearity is evaluated once for each polygon:

$$I^{\text{out}} = I_e + k_a \cdot I_a + \sum_{l}^{n_l} I_l \cdot k_d \cdot \max\{(\vec{N} \cdot \vec{L}), 0\}. \tag{7.6}$$

In order to generate the unit surface normal $\vec{N}$ for the formula, two alternatives are available. It can either be the "average" normal of the real surface over this polygon estimated from the normals of the real surface in the vertices of the polygon, or else the normal of the approximating polygon.

7.5 Gouraud shading

Having approximated the surface by a polygon mesh, Gouraud shading requires the evaluation of the rendering equation at the vertices for polygons, using the normals of the real surface in the formula. For the sake of simplicity, let us assume that the polygon mesh consists of triangles only (this assumption has an important advantage in that three points are always on a plane). Suppose we have already evaluated the shading equation for the vertices having resultant intensities I_1, I_2 and I_3, usually on representative wavelengths of red, green and blue light. The color or (R, G, B) values of the inner pixels are determined by linear approximation from the vertex colors. This approximation should be carried out separately for each wavelength.

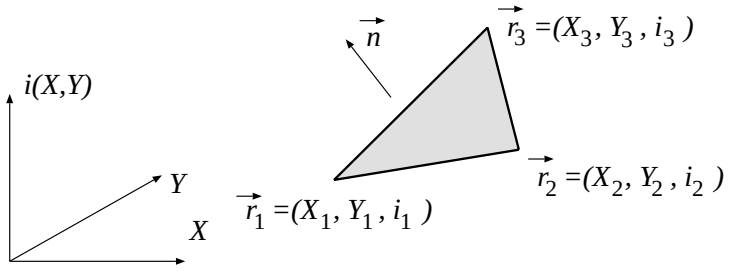

Figure 7.3: Linear interpolation in color space

Let i be the alias of any of I^{red}, I^{green} or I^{blue}. The function $i(X, Y)$ of the pixel coordinates described in figure 7.3 forms a plane through the vertex points $\vec{r}_1 = (X_1, Y_1, i_1)$, $\vec{r}_2 = (X_2, Y_2, i_2)$ and $\vec{r}_3 = (X_3, Y_3, i_3)$ in (X, Y, i) space. For notational convenience, we shall assume that $Y_1 \leq Y_2 \leq Y_3$ and (X_2, Y_2) is on the left side of the $[(X_1, Y_1); (X_3, Y_3)]$ line, looking at the triangle from the camera position. The equation of this plane is:

$$\vec{n} \cdot \vec{r} = \vec{n} \cdot \vec{r}_1 \quad \text{where} \quad \vec{n} = (\vec{r}_2 - \vec{r}_1) \times (\vec{r}_3 - \vec{r}_1). \tag{7.7}$$

Denoting the constant $\vec{n} \cdot \vec{r}_1$ by C, and expressing the equation in scalar form substituting the coordinates of the normal of the plane, $\vec{n} = (n_X, n_Y, n_i)$, the function $i(X, Y)$ has the following form:

$$i(X, Y) = \frac{C - n_X \cdot X - n_Y \cdot Y}{n_i}. \tag{7.8}$$

The computational requirement of two multiplications, two additions and a division can further be decreased by the incremental concept (recall section 2.3 on hardware realization of graphics algorithms).

Expressing $i(X + 1, Y)$ as a function of $i(X, Y)$ we get:

$$i(X+1,Y) = i(X,Y) + \frac{\partial i(X,Y)}{\partial X} \cdot 1 = i(X,Y) - \frac{n_X}{n_i} = i(X,Y) + \delta i_X. \tag{7.9}$$

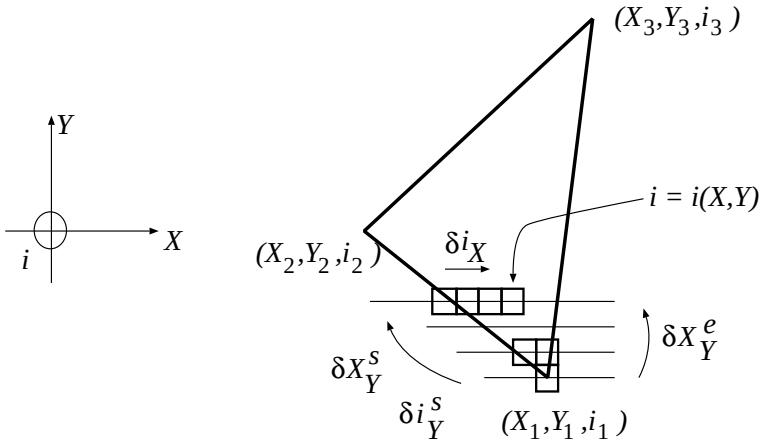

Figure 7.4: Incremental concept in Gouraud shading

Since δi_X does not depend on the actual X, Y coordinates, it has to be evaluated once for the polygon. Inside a scan-line, the calculation of a pixel color requires a single addition for each color coordinate according to equation 7.9. Concerning the X and i coordinates of the boundaries of the scan-lines, the incremental concept can also be applied to express the starting and ending pixels.

Since i and X vary linearly along the edge of the polygon, equations 2.33, 2.34 and 2.35 result in the following simple expressions in the range of $Y_1 \leq Y \leq Y_2$, denoting K_s by X_{start} and K_e by X_{end}, and assuming that the triangle is left oriented as shown in figure 7.4:

$$\begin{aligned} X_{\text{start}}(Y+1) &= X_{\text{start}}(Y) + \frac{X_2 - X_1}{Y_2 - Y_1} = X_{\text{start}}(Y) + \delta X^s_Y \\ X_{\text{end}}(Y+1) &= X_{\text{end}}(Y) + \frac{X_3 - X_1}{Y_3 - Y_1} = X_{\text{end}}(Y) + \delta X^e_Y \\ i_{\text{start}}(Y+1) &= i_{\text{start}}(Y) + \frac{i_2 - i_1}{Y_2 - Y_1} = i_{\text{start}}(Y) + \delta i^s_Y \end{aligned} \tag{7.10}$$

The last equation represents in fact three equations, one for each color coordinate, (R, G, B). For the lower part of the triangle in figure 7.4, the incremental algorithm is then:

$X_{\text{start}} = X_1 + 0.5$; $X_{\text{end}} = X_1 + 0.5$;
$R_{\text{start}} = R_1 + 0.5$; $G_{\text{start}} = G_1 + 0.5$; $B_{\text{start}} = B_1 + 0.5$;
for $Y = Y_1$ **to** Y_2 **do**
    $R = R_{\text{start}}$; $G = G_{\text{start}}$; $B = B_{\text{start}}$;
    for $X = \text{Trunc}(X_{\text{start}})$ **to** $\text{Trunc}(X_{\text{end}})$ **do**
        write($X, Y, \text{Trunc}(R), \text{Trunc}(G), \text{Trunc}(B)$);
        $R \mathrel{+}= \delta R_X$; $G \mathrel{+}= \delta G_X$; $B \mathrel{+}= \delta B_X$;
    endfor
    $X_{\text{start}} \mathrel{+}= \delta X^s_Y$; $X_{\text{end}} \mathrel{+}= \delta X^e_Y$;
    $R_{\text{start}} \mathrel{+}= \delta R^s_Y$; $G_{\text{start}} \mathrel{+}= \delta G^s_Y$; $B_{\text{start}} \mathrel{+}= \delta B^s_Y$;
endfor

Having represented the numbers in a fixed point format, the derivation of the executing hardware of this algorithm is straightforward by the methods outlined in section 2.3 (on hardware realization of graphics algorithms). Note that this algorithm generates a part of the triangle below Y_2 coordinates. The same method has to be applied again for the upper part.

Recall that the very same approach was applied to calculate the Z coordinate in the z-buffer method. Because of their algorithmic similarity, the same hardware implementation can be used to compute the Z coordinate, and the R, G, B color coordinates.

The possibility of hardware implementation makes Gouraud shading very attractive and popular in advanced graphics workstations, although it has

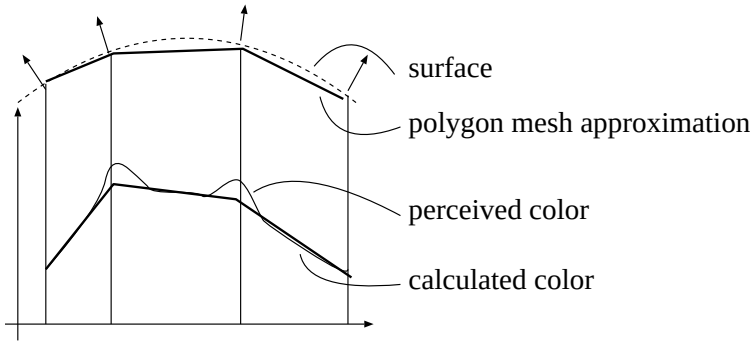

Figure 7.5: Mach banding

several severe drawbacks. It does not allow shadows, texture and bump mapping in its original form, and introduces an annoying artifact called **Mach banding** (figure 7.5). Due to linear approximation in color space, the color is a continuous, but not differentiable function. The human eye, however, is sensitive to the drastic changes of the derivative of the color, overemphasizing the edges of the polygon mesh, where the derivative is not continuous.

7.6 Phong shading

In Phong shading only the surface normal is approximated from the real surface normals in the vertices of the approximating polygon; the shading equation is evaluated for each pixel. The interpolating function of the normal vectors is linear:

$$\begin{aligned} n_X &= a_X \cdot X + b_X \cdot Y + c_X, \\ n_Y &= a_Y \cdot X + b_Y \cdot Y + c_Y, \\ n_Z &= a_Z \cdot X + b_Z \cdot Y + c_Z. \end{aligned} \tag{7.11}$$

Constants $a_X, \ldots, c_Z$ can be determined by similar considerations as in Gouraud shading from the normal vectors at the vertices of the polygon (triangle). Although the incremental concept could be used again to reduce the number of multiplications in this equation, it is not always worth doing, since the shading equation requires many expensive computational steps

which mean that this computation is negligible in terms of the total time required.

Having generated the approximation of the normal to a surface visible in a given pixel, the complete rendering equation is applied:

$$I^{\text{out}} = I_e + k_a \cdot I_a + \sum_l^{n_l} r_l \cdot I_l \cdot k_d \cdot \max\{(\vec{N} \cdot \vec{L}), 0\} +$$

$$\sum_l^{n_l} r_l \cdot I_l \cdot k_s \cdot \max\{[2(\vec{N} \cdot \vec{H})^2 - 1]^n, 0\} \tag{7.12}$$

Recall that dot products, such as $\vec{N} \cdot \vec{L}$, must be evaluated for vectors in the world coordinate system, since the viewing transformation may alter the angle between vectors. For directional lightsources this poses no problem, but for positional and flood types the point corresponding to the pixel in the world coordinate system must be derived for each pixel. To avoid screen and world coordinate system mappings on the pixel level, the corresponding (x, y, z) world coordinates of the pixels inside the polygon are determined by a parallel and independent linear interpolation in world space. Note that this is not accurate for perspective transformation, since the homogeneous division of perspective transformation destroys equal spacing, but this error is usually not noticeable on the images.

Assuming only ambient and directional lightsources to be present, the incremental algorithm for half of a triangle is:

$X_{\text{start}} = X_1 + 0.5$; $X_{\text{end}} = X_1 + 0.5$;
$\vec{N}_{\text{start}} = \vec{N}_1$;
for $Y = Y_1$ **to** Y_2 **do**
    $\vec{N} = \vec{N}_{\text{start}}$;
    for $X = \text{Trunc}(X_{\text{start}})$ **to** $\text{Trunc}(X_{\text{end}})$ **do**
        $(R, G, B) = \text{ShadingModel}(\ \vec{N}\ )$;
        write$(\ X, Y, \text{Trunc}(R), \text{Trunc}(G), \text{Trunc}(B)\ )$;
        $\vec{N}$ += $\delta\vec{N}_X$;
    endfor
    X_{start} += δX^s_Y; X_{end} += δX^e_Y;
    $\vec{N}_{\text{start}}$ += $\delta\vec{N}^s_Y$;
endfor

The rendering equation used for Phong shading is not appropriate for incremental evaluation in its original form. For directional and ambient lightsources, however, it can be approximated by a two-dimensional Taylor series, as proposed by Bishop [BW86], which in turn can be calculated incrementally with five additions and a non-linear function evaluation typically implemented by a pre-computed table in the computer memory.

The coefficients of the shading equation, k_a, k_d, k_s and n can also be a function of the point on the surface, allowing **textured surfaces** to be rendered by Phong shading. In addition it is possible for the approximated surface normal to be perturbed by a normal vector variation function causing the effect of bump mapping (see chapter 12).