Shading Models

Dr. Mousumi Dutt

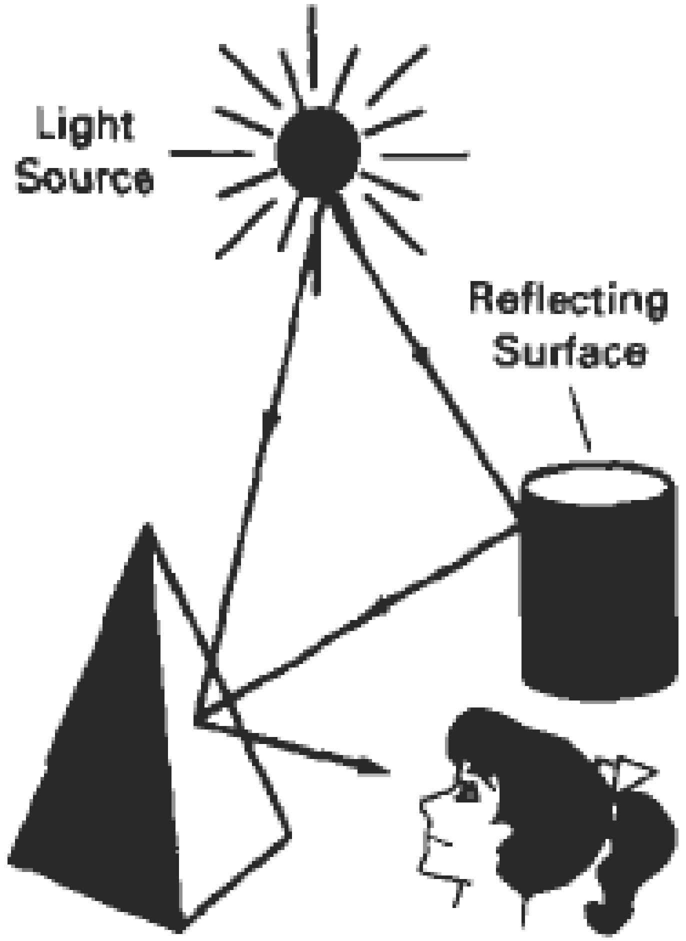
Introduction

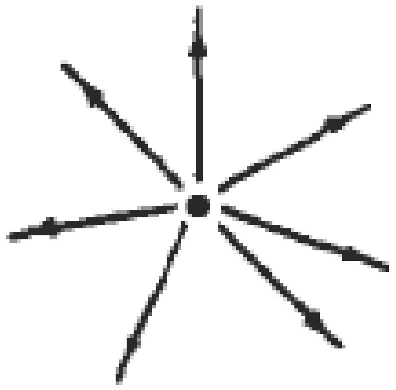
- Realistic displays of a scene are obtained by generating perspective projections of objects and by applying natural lighting effects to the visible surfaces
- An illumination model, also called a lighting model and sometimes referred to as a shading model, is used to calculate the intensity of light that we should see at a given point on the surface of an object
- A surface-rendering algorithm uses the intensity calculations from an illumination model to determine the light intensity for all projected pixel positions for the various surfaces in a scene
- Surface rendering can be performed by applying the illumination model to every visible surface point, or the rendering can be accomplished by interpolating intensities across the surfaces from a small set of illumination-model calculations.
- Scan-line, image-space algorithms typically use interpolation schemes, while ray-tracing algorithms invoke the illumination model at each pixel position

Introduction

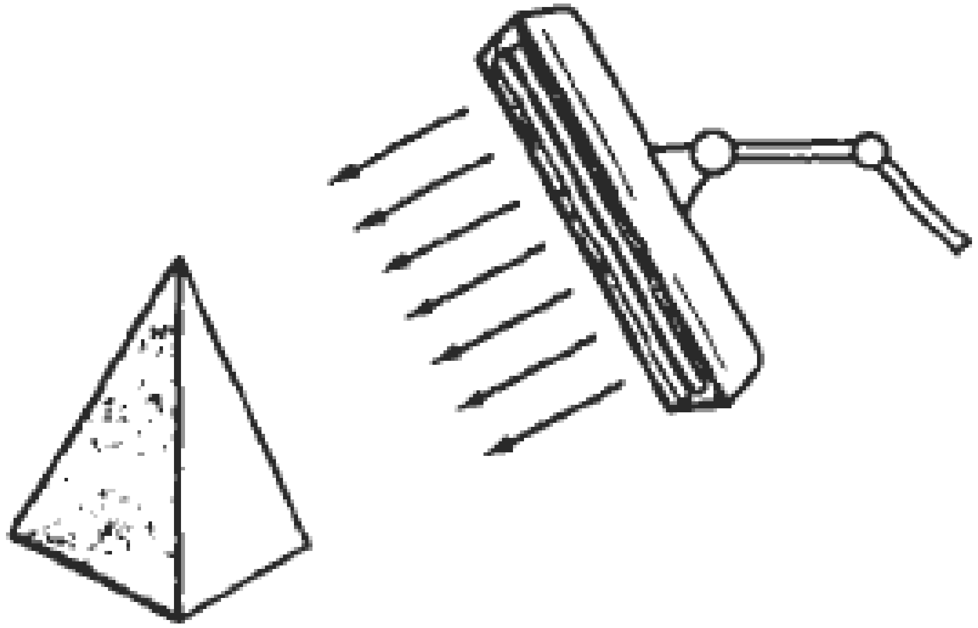
- When we view an opaque nonluminous object, we see reflected light from the surfaces of the object. The total reflected light is the sum of the contributions from light sources and other reflecting surfaces in the scene
- A surface that is not directly exposed to a light source may still be visible if nearby objects are illuminated
- light sources are referred to as light-emitting sources; and reflecting surfaces, such as the walls of a room, are termed light-reflecting sources
- A luminous object, in general, can be both a light source and a light reflector. For example, a plastic globe with a light bulb insidc both emits and reflects light from the surface of the globe. Emitted light from the globe may then illuminate other objects in the vicinity.
- The simplest model for a light emitter is a point source
- Rays from the source then follow radially diverging paths from the source position

Introduction





Diverging ray paths from a point light source.

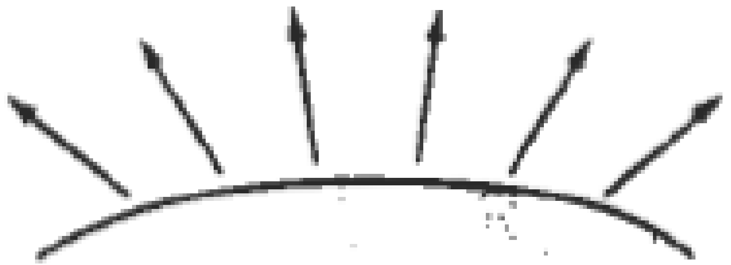


An object illuminated with a distributed light source.

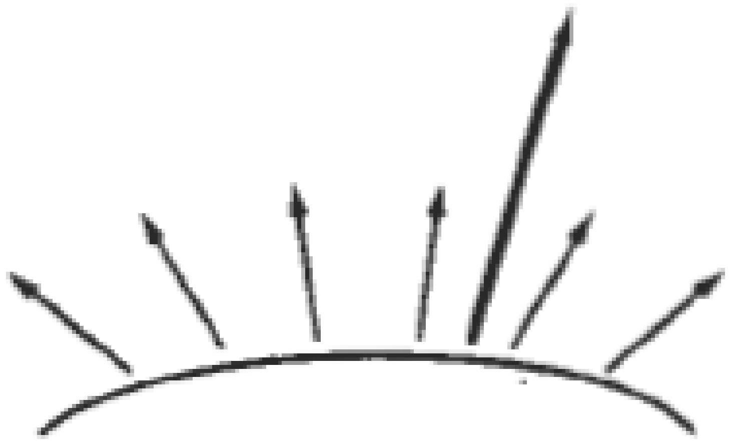
Introduction

Surfaces that are rough, or grainy, tend to scatter the reflected light in all directions. This scattered light is called diffuse reflection

In addition to diffuse reflection, light sources create highlights, or bright spots, called specular reflection



Diffuse reflections from a surface.



Specular reflection superimposed on diffuse reflection vectors.

Introduction

A simple way to model the combination of light reflections from various surfaces to produce a uniform illumination called the ambient light, or background light

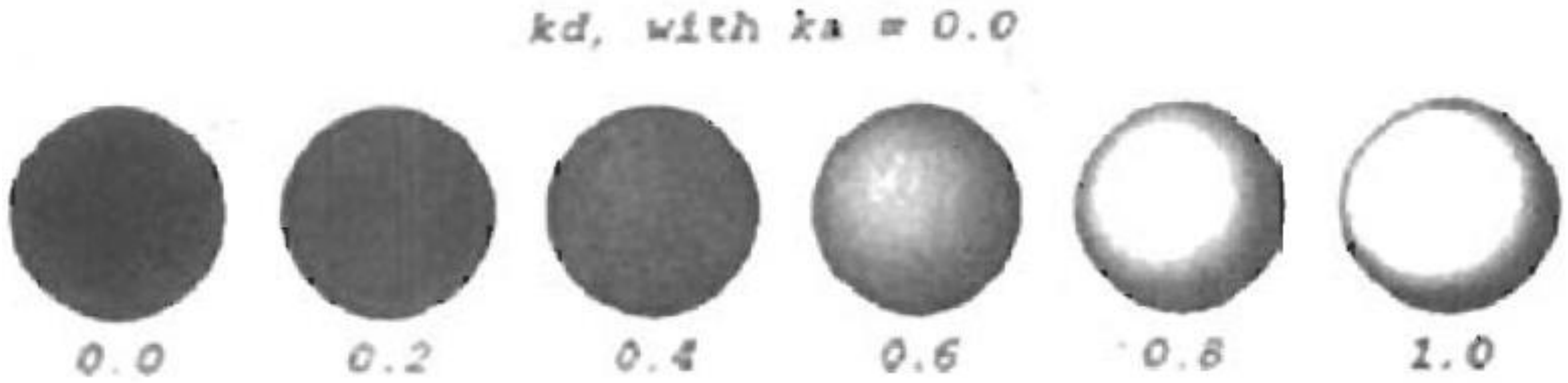
Ambient light has no spatial or directional characteristics.

The amount of ambient light incident on each object is a constant for all surfaces and over all directions

Diffuse reflections are constant over each surface in a scene, independent of the viewing direction

The fractional amount of the incident light that is diffusely reflected can be set for each surface with parameter k_d, the diffuse-reflection coefficient, or diffuse reflectivity

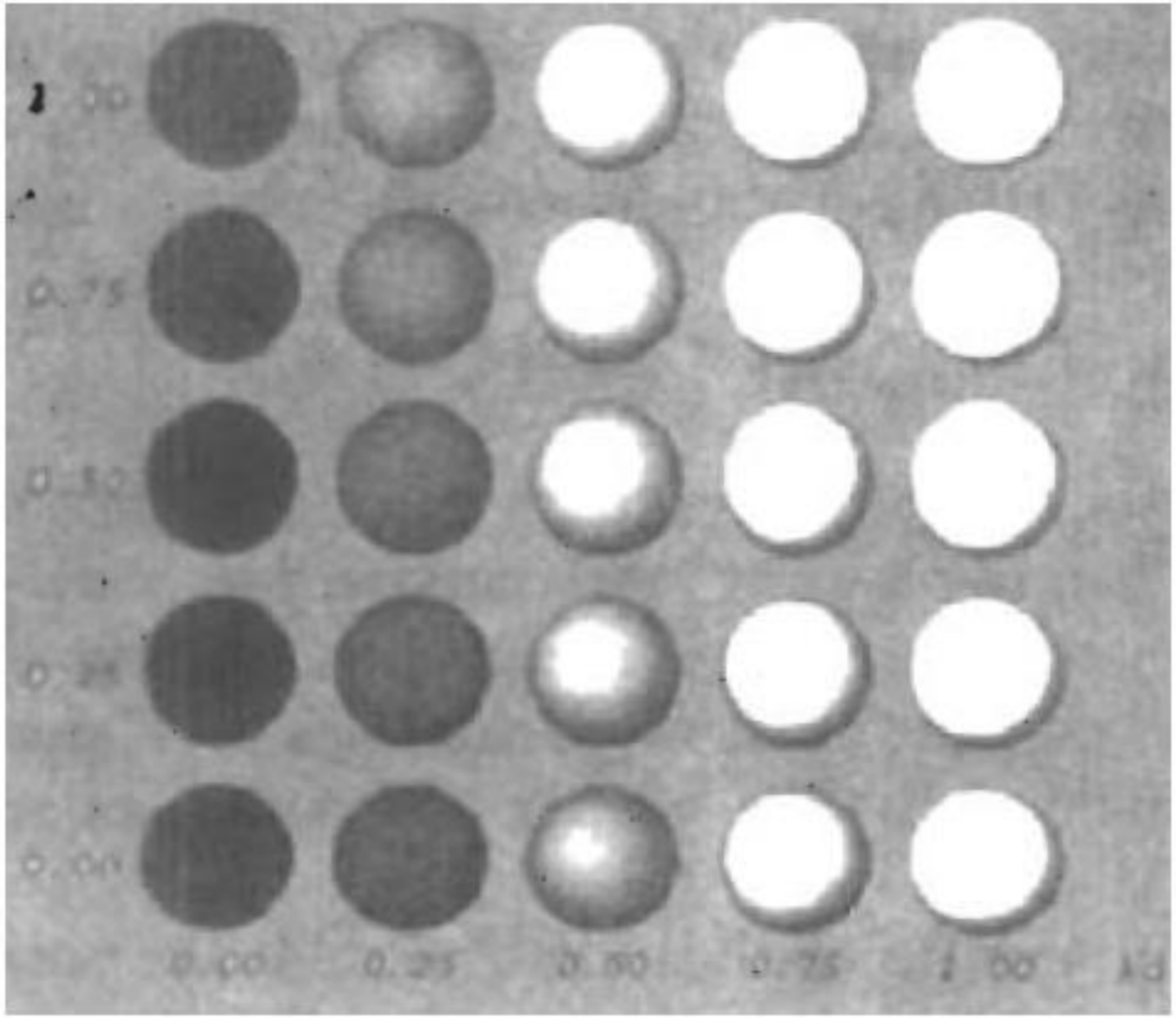
Introduction



Diffuse reflections from a spherical surface illuminated by a point light source for values of the diffuse reflectivity coefficient in the interval $0 \leq k_d \leq 1$.

Introduction

Diffuse reflections from a spherical surface illuminated with ambient light and a single point source for values of k_a and k_d in the interval (0, 1).



Polygon Rendering Methods

- The objects are usually polygon-mesh approximations of curved-surface objects, but they may also be polyhedra that are not curved-surface approximations.
- Scanline algorithms typically apply a lighting model to obtain polygon surface rendering in one of two ways
- Each polygon can be rendered with a single intensity, or the intensity can be obtained at each point of the surface using an interpolation scheme

Constant-Intensity Shading

A fast and simple method for rendering an object with polygon surfaces is **constant-intensity shading**, also called **flat shading**. In this method, a single intensity is calculated for each polygon. All points over the surface of the polygon are then displayed with the same intensity value. Constant shading can be useful for quickly displaying the general appearance of a curved surface,

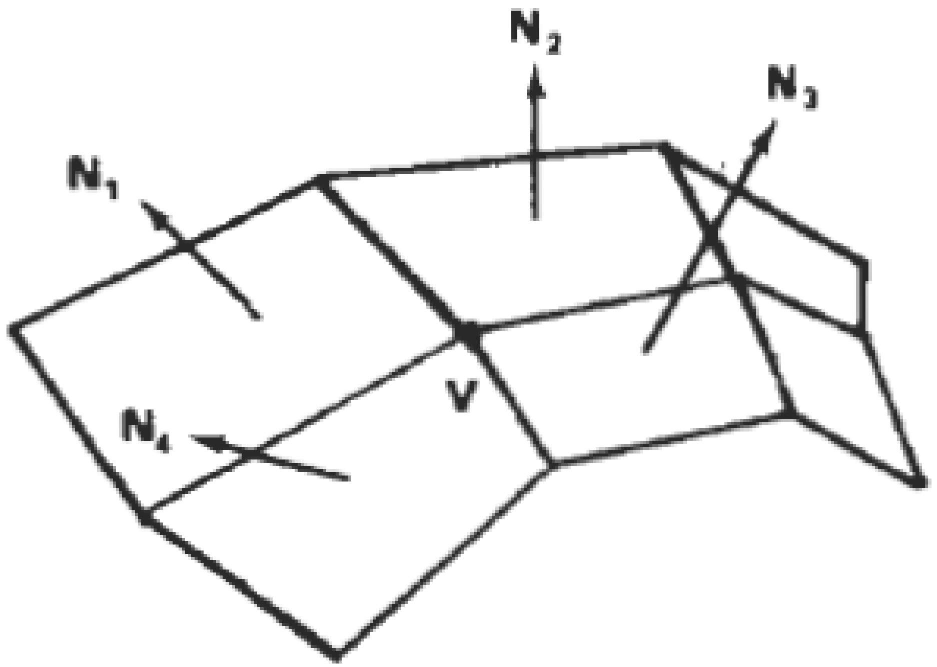
In general, flat shading of polygon facets provides an accurate rendering for an object if all of the following assumptions are valid:

- The object is a polyhedron and is not an approximation of an object with a curved surface.

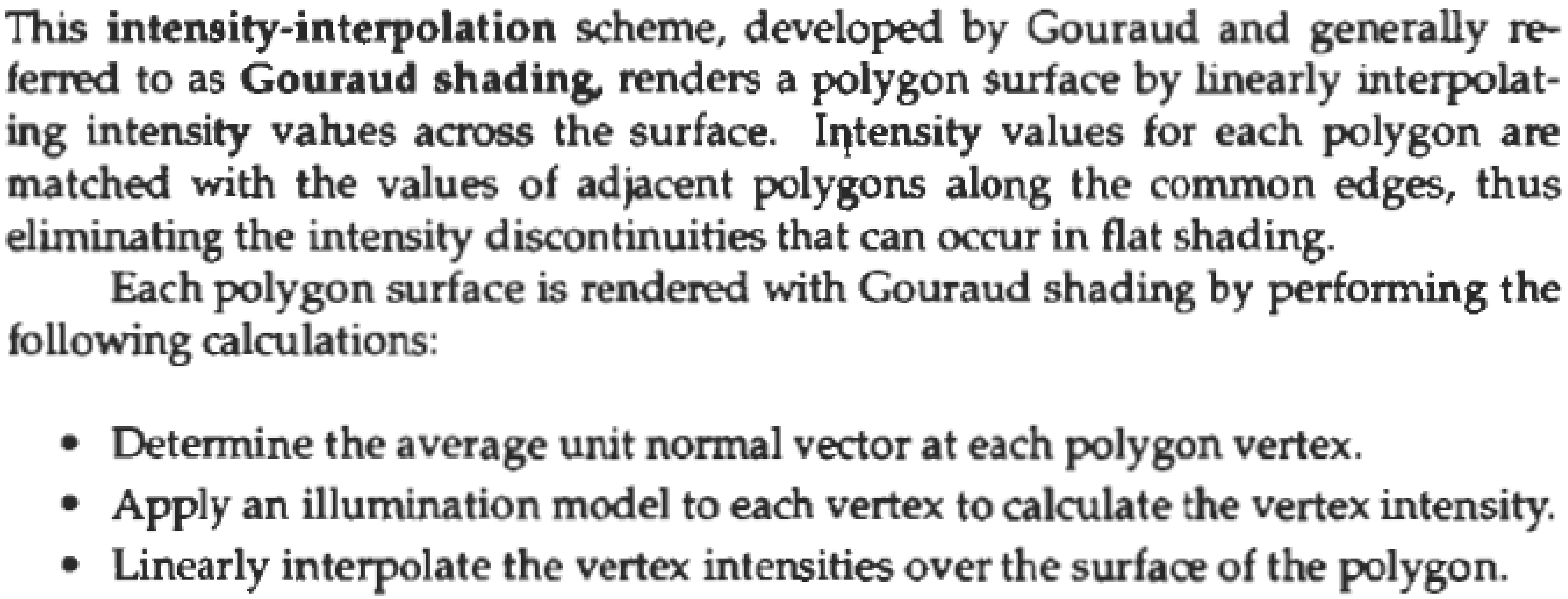
Constant-Intensity Shading

- All light sources illuminating the object are sufficiently far from the surface so that **N** · **L** and the attenuation function are constant over the surface.
- The viewing position is sufficiently far from the surface so that **V** · **R** is constant over the surface.

Even if all of these conditions are not true, we can still reasonably approximate surface-lighting effects using small polygon facets with flat shading and calculate the intensity for each facet, say, at the center of the polygon.



Gouraud Shading



This **intensity-interpolation** scheme, developed by Gouraud and generally referred to as **Gouraud shading**, renders a polygon surface by linearly interpolating intensity values across the surface. Intensity values for each polygon are matched with the values of adjacent polygons along the common edges, thus eliminating the intensity discontinuities that can occur in flat shading.

Each polygon surface is rendered with Gouraud shading by performing the following calculations:

- Determine the average unit normal vector at each polygon vertex.
- Apply an illumination model to each vertex to calculate the vertex intensity.
- Linearly interpolate the vertex intensities over the surface of the polygon.

Gouraud Shading

At each polygon vertex, we obtain a normal vector by averaging the surface normals of all polygons sharing that vertex, as illustrated in Fig. 14-44. Thus, for any vertex position V, we obtain the unit vertex normal with the calculation

$$\mathbf{N}_V = \frac{\sum_{k=1}^{n} \mathbf{N}_k}{\left|\sum_{k=1}^{n} \mathbf{N}_k\right|} \qquad (14\text{-}37)$$

Once we have the vertex normals, we can determine the intensity at the vertices from a lighting model.

Figure 14-45 demonstrates the next step: interpolating intensities along the polygon edges. For each scan line, the intensity at the intersection of the scan line with a polygon edge is linearly interpolated from the intensities at the edge endpoints. For the example in Fig. 14-45, the polygon edge with endpoint vertices at positions 1 and 2 is intersected by the scan line at point 4. A fast method for obtaining the intensity at point 4 is to interpolate between intensities I_1 and I_2 using only the vertical displacement of the scan line:

Gouraud Shading

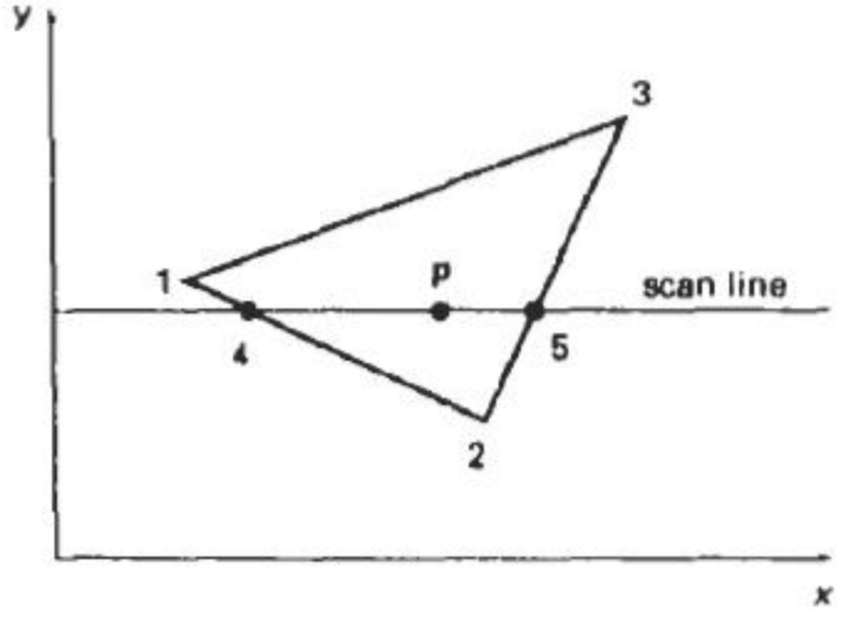


Figure 14-45
For Gouraud shading, the intensity at point 4 is linearly interpolated from the intensities at vertices 1 and 2. The intensity at point 5 is linearly interpolated from intensities at vertices 2 and 3. An interior point **p** is then assigned an intensity value that is linearly interpolated from intensities at positions 4 and 5.

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2 \qquad (14\text{-}38)$$

Similarly, intensity at the right intersection of this scan line (point 5) is interpolated from intensity values at vertices 2 and 3. Once these bounding intensities are established for a scan line, an interior point (such as point **p** in Fig. 14-45) is interpolated from the bounding intensities at points 4 and 5 as

$$I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5 \qquad (14\text{-}39)$$

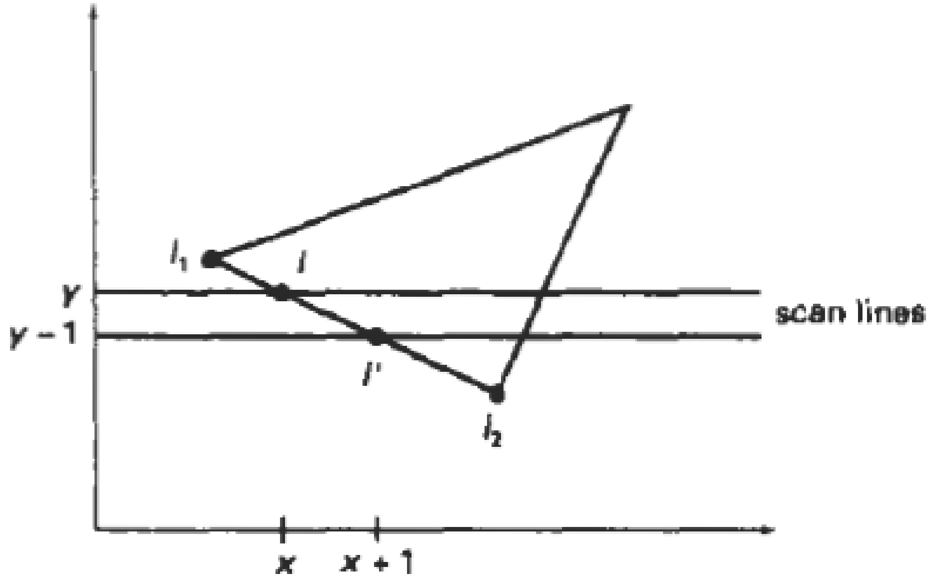
Gouraud Shading

Incremental calculations are used to obtain successive edge intensity values between scan lines and to obtain successive intensities along a scan line. As shown in Fig. 14-46, if the intensity at edge position (x, y) is interpolated as

$$I = \frac{y - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y}{y_1 - y_2} I_2 \tag{14-40}$$

then we can obtain the intensity along this edge for the next scan line, $y - 1$, as

$$I' = I + \frac{I_2 - I_1}{y_1 - y_2} \tag{14-41}$$



Gouraud Shading

Similar calculations are used to obtain intensities at successive horizontal pixel positions along each scan line.

When surfaces are to be rendered in color, the intensity of each color component is calculated at the vertices. Gouraud shading can be combined with a hidden-surface algorithm to fill in the visible polygons along each scan line. An example of an object shaded with the Gouraud method appears in Fig. 14-47.

Gouraud shading removes the intensity discontinuities associated with the constant-shading model, but it has some other deficiencies. Highlights on the surface are sometimes displayed with anomalous shapes, and the linear intensity interpolation can cause bright or dark intensity streaks, called Mach bands, to appear on the surface. These effects can be reduced by dividing the surface into a greater number of polygon faces or by using other methods, such as Phong shading, that require more calculations.

Gouraud Shading

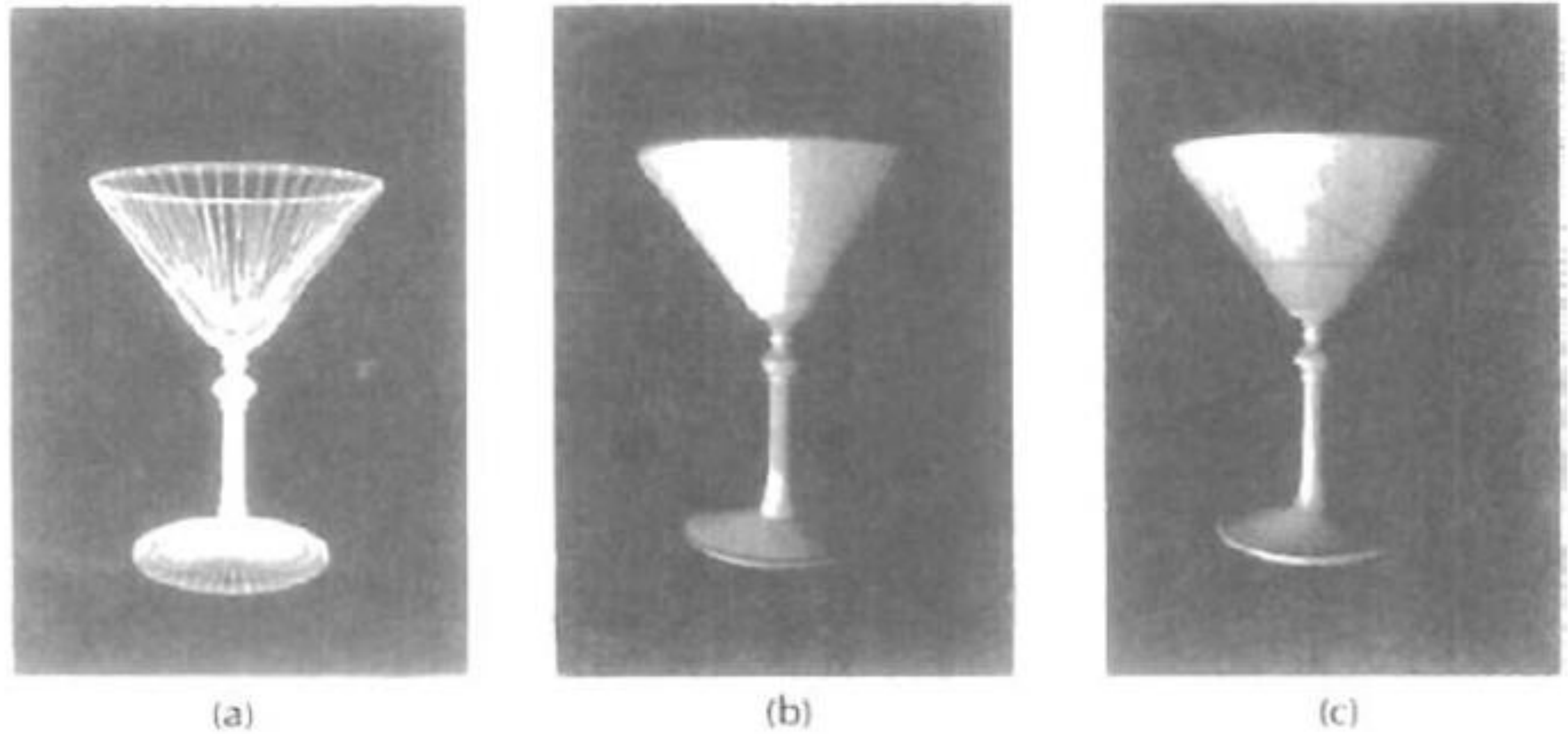
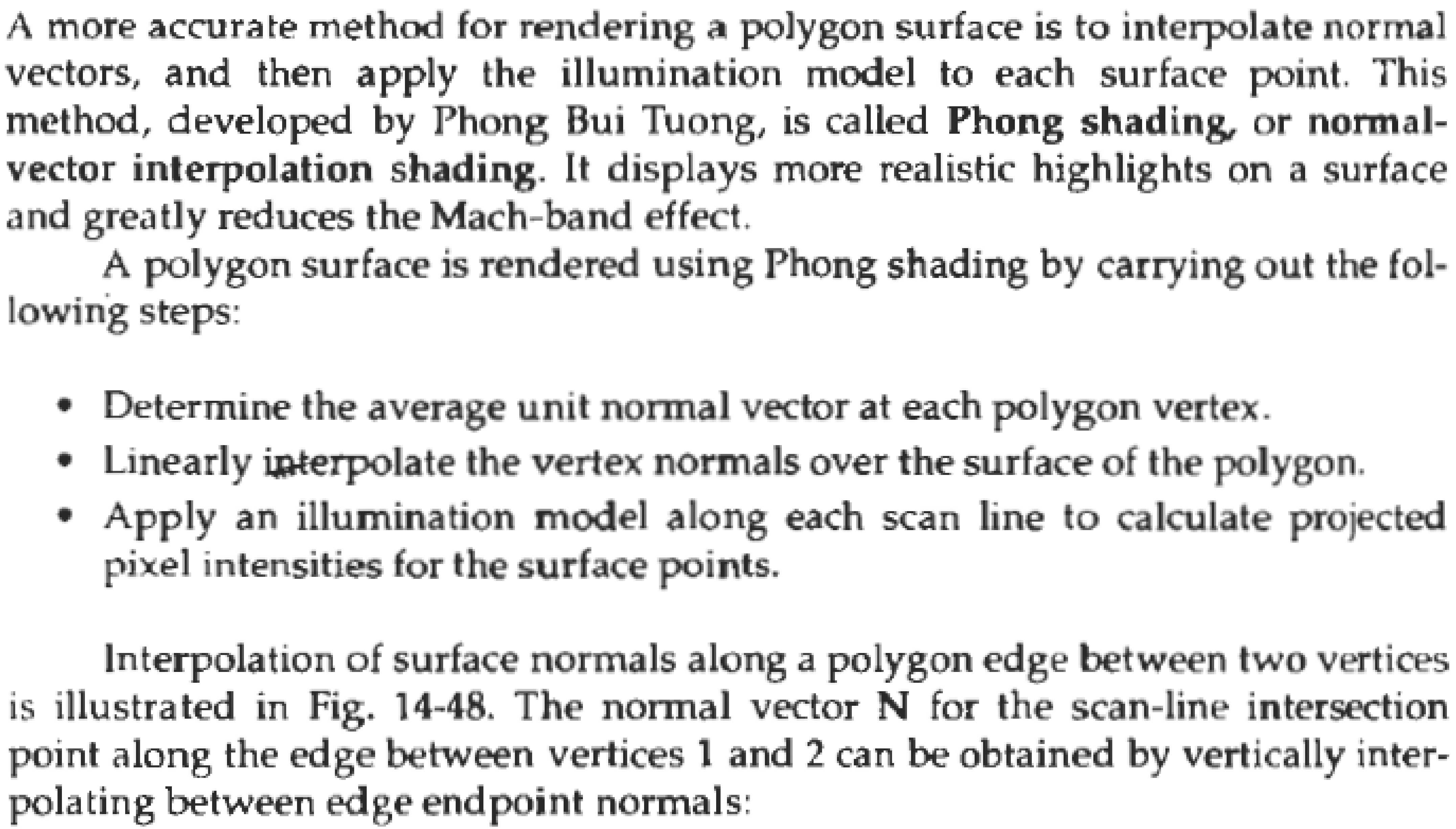


Figure 14-47
A polygon mesh approximation of an object (a) is rendered with flat shading (b) and with Gouraud shading (c).

Phong Shading



A more accurate method for rendering a polygon surface is to interpolate normal vectors, and then apply the illumination model to each surface point. This method, developed by Phong Bui Tuong, is called **Phong shading**, or **normal-vector interpolation shading**. It displays more realistic highlights on a surface and greatly reduces the Mach-band effect.

A polygon surface is rendered using Phong shading by carrying out the following steps:

- Determine the average unit normal vector at each polygon vertex.
- Linearly interpolate the vertex normals over the surface of the polygon.
- Apply an illumination model along each scan line to calculate projected pixel intensities for the surface points.

Interpolation of surface normals along a polygon edge between two vertices is illustrated in Fig. 14-48. The normal vector **N** for the scan-line intersection point along the edge between vertices 1 and 2 can be obtained by vertically interpolating between edge endpoint normals:

Phong Shading

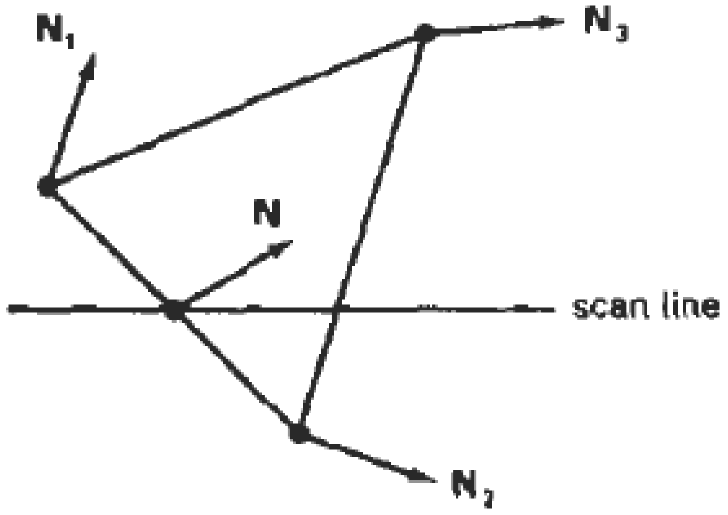


Figure 14-48
Interpolation of surface normals along a polygon edge

$$\mathbf{N} = \frac{y - y_2}{y_1 - y_2}\mathbf{N}_1 + \frac{y_1 - y}{y_1 - y_2}\mathbf{N}_2 \qquad (14\text{-}42)$$

Incremental methods are used to evaluate normals between scan lines and along each individual scan line. At each pixel position along a scan line, the illumination model is applied to determine the surface intensity at that point.

Intensity calculations using an approximated normal vector at each point along the scan line produce more accurate results than the direct interpolation of intensities, as in Gouraud shading. The trade-off, however, is that Phong shading requires considerably more calculations.

First Phong Shading

Surface rendering with Phong shading can be speeded up by using approximations in the illumination-model calculations of normal vectors. **Fast Phong shading** approximates the intensity calculations using a Taylor-series expansion and triangular surface patches.

Since Phong shading interpolates normal vectors from vertex normals, we can express the surface normal **N** at any point (x, y) over a triangle as

$$\mathbf{N} = \mathbf{A}x + \mathbf{B}y + \mathbf{C} \tag{14-43}$$

where vectors **A**, **B**, and **C** are determined from the three vertex equations:

$$\mathbf{N}_k = \mathbf{A}x_k + \mathbf{B}y_k + \mathbf{C}, \qquad k = 1, 2, 3 \tag{14-44}$$

with (x_k, y_k) denoting a vertex position.

First Phong Shading

Omitting the reflectivity and attenuation parameters, we can write the calculation for light-source diffuse reflection from a surface point (x, y) as

$$\begin{aligned} I_{\text{diff}}(x, y) &= \frac{\mathbf{L} \cdot \mathbf{N}}{|\mathbf{L}|\,|\mathbf{N}|} \\ &= \frac{\mathbf{L} \cdot (\mathbf{A}x + \mathbf{B}y + \mathbf{C})}{|\mathbf{L}|\,|\mathbf{A}x + \mathbf{B}y + \mathbf{C}|} \\ &= \frac{(\mathbf{L} \cdot \mathbf{A})x + (\mathbf{L} \cdot \mathbf{B})y + \mathbf{L} \cdot \mathbf{C}}{|\mathbf{L}|\,|\mathbf{A}x + \mathbf{B}y + \mathbf{C}|} \end{aligned} \tag{14-45}$$

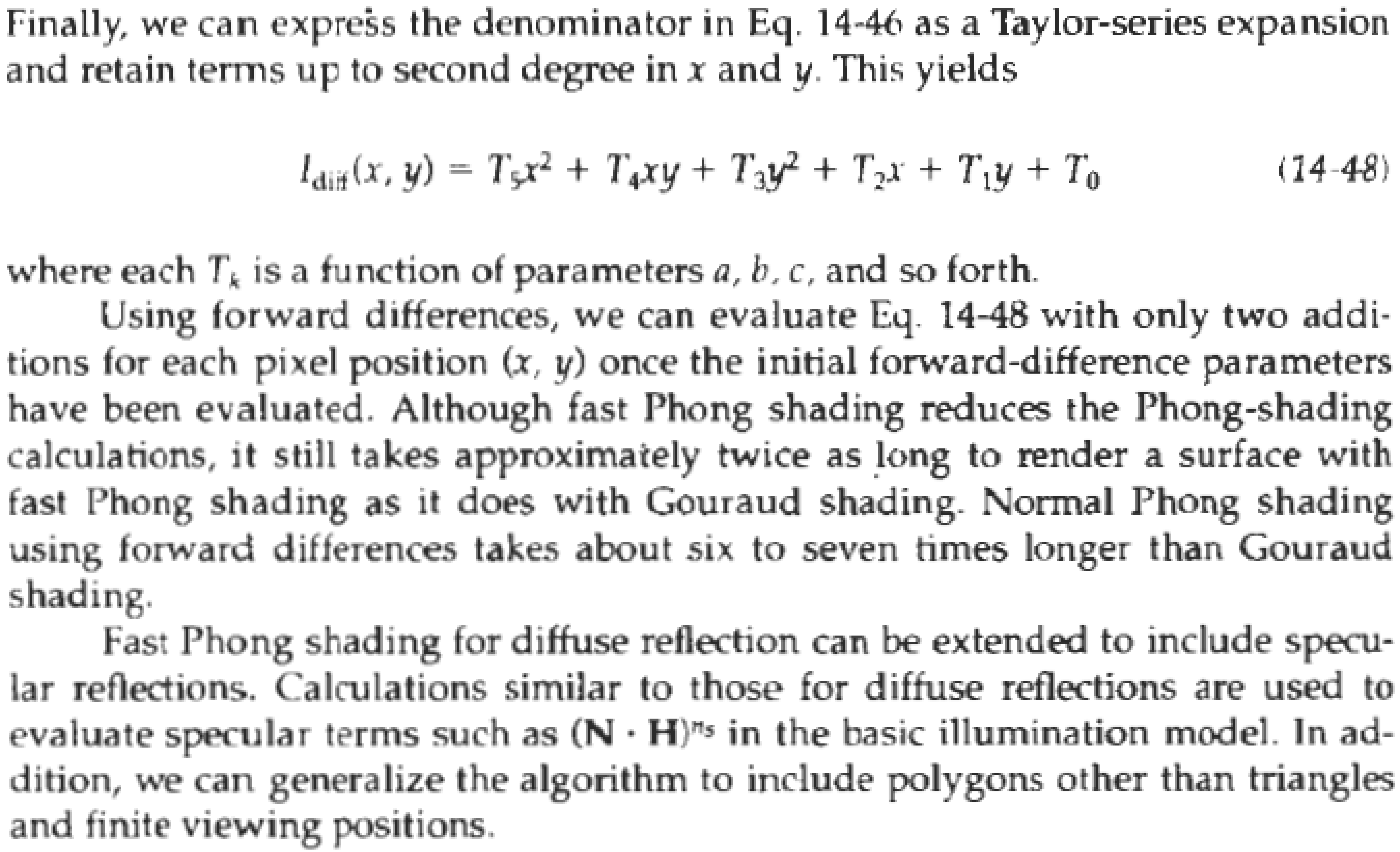
We can rewrite this expression in the form

$$I_{\text{diff}}(x, y) = \frac{ax + by + c}{(dx^2 + exy + fy^2 + gx + hy + i)^{1/2}} \tag{14-46}$$

where parameters such as a, b, c, and d are used to represent the various dot products. For example,

$$a = \frac{\mathbf{L} \cdot \mathbf{A}}{|\mathbf{L}|} \tag{14-47}$$

First Phong Shading



Finally, we can express the denominator in Eq. 14-46 as a Taylor-series expansion and retain terms up to second degree in x and y. This yields

$$I_{\text{diff}}(x, y) = T_5x^2 + T_4xy + T_3y^2 + T_2x + T_1y + T_0 \tag{14-48}$$

where each T_k is a function of parameters a, b, c, and so forth.

Using forward differences, we can evaluate Eq. 14-48 with only two additions for each pixel position (x, y) once the initial forward-difference parameters have been evaluated. Although fast Phong shading reduces the Phong-shading calculations, it still takes approximately twice as long to render a surface with fast Phong shading as it does with Gouraud shading. Normal Phong shading using forward differences takes about six to seven times longer than Gouraud shading.

Fast Phong shading for diffuse reflection can be extended to include specular reflections. Calculations similar to those for diffuse reflections are used to evaluate specular terms such as $(\mathbf{N} \cdot \mathbf{H})^{n_s}$ in the basic illumination model. In addition, we can generalize the algorithm to include polygons other than triangles and finite viewing positions.

Ray Tracing

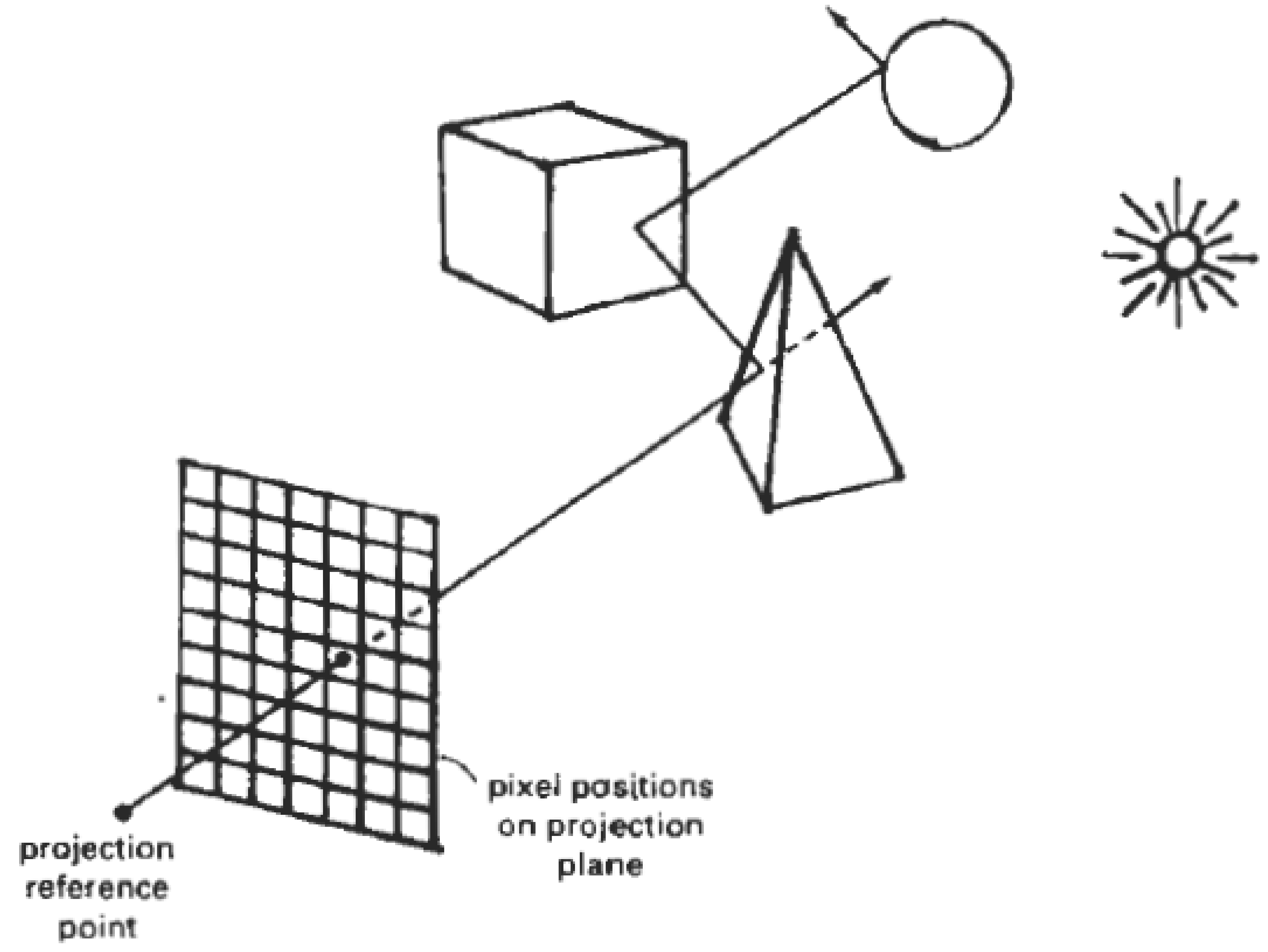
Ray tracing is an extension of this basic idea. Instead of merely looking for the visible surface for each pixel, we continue to bounce the ray around the scene

The basic ray-tracing algorithm also provides for visible-surface detection, shadow effects, transparency, and multiple light-source illumination

Many extensions to the basic algorithm have been developed to produce photorealistic displays

Ray-traced displays can be highly realistic, particularly for shiny objects, but they require considerable computation time to generate

Ray Tracing


pixel positions on projection plane
projection reference point

Ray Tracing

Ray tracing functions nearly the same way, except that everything generally moves in the opposite direction. Inside the software, ray-traced light begins at the viewer (from the camera lens, essentially) and moves outward, plotting a path that bounces across multiple objects, sometimes even taking on their color and reflective properties, until the software determines the appropriate light source(s) that would affect that particular ray. This technique of simulating vision *backward* is far more efficient for a computer to handle than trying to trace the rays *from* the light source. After all, the only light paths that need to be rendered are the ones that fit into the user's field of view. It takes far less computing power to display what's in front of you than it would to render the rays emitted from all sources of light in a scene.