unit4

8

Hidden Surface Elimination Methods

8.1 Introduction

For generation of realistic graphics displays, we have to identify those parts of a scene that are visible from a chosen viewing position. There are many algorithms called **visible surface algorithms** developed to solve this problem. In early days of computer graphics visible surface algorithms were called **hidden line** or **hidden surface algorithms**.

In a given set of 3D objects and viewing specification, we wish to determine which lines or surfaces of the objects are visible, so that we can display only the visible lines or surfaces. This process is known as hidden surfaces or hidden line elimination, or visible surface determination. The hidden line or hidden surface algorithm determines the lines, edges, surfaces or volumes that are visible or invisible to an observer located at a specific point in space. These algorithms are broadly classified according to whether they deal with object definitions directly or with their projected images. These two approaches are called **object-space methods** or **object precision methods** and **image-space methods**, respectively.

Object-space Method : Object-space method is implemented in the **physical coordinate system** in which objects are described. It compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible. Object-space methods are generally used in **line-display algorithms**.

Image-Space Method : Image space method is implemented in the **screen coordinate system** in which the objects are viewed. In an image-space algorithm, visibility is decided point by point at each pixel position on the view plane. Most **hidden line/surface algorithms** use image-space methods.

In this chapter we are going to study various visible surface detection or hidden line removal algorithms, algorithms for octrees, algorithms for curved surfaces, and visible-surface ray tracing.

8.2 Techniques for Efficient Visible-Surface Algorithms

We have seen that there are two basic approaches used for visible surface detection : Object precision algorithm and image precision algorithm. In both the algorithms we require to perform a number of potentially costly operations such as determination of projections of objects, whether or not they intersect and where they intersect, closest object in case of intersection and so on. To create and display picture in minimum time we have to perform visible surface algorithms more efficiently. The techniques to perform visible-surface algorithms efficiently are discussed in the following sections.

8.2.1 Coherence

The coherence is defined as the degree to which parts of an environment or its projection exhibit local similarities. Such as similarities in depth, colour, texture and so on. To make algorithms more efficient we can exploit these similarities (coherence) when we reuse calculations made for one part of the environment or a picture for other nearby parts, either without changes or with some incremental changes. Let us see different kinds of coherence we can use in visible surface algorithms.

- **Object coherence :** If one object is entirely separate from another, comparisons may need to be done only between the two objects, and not between their components faces or edges.
- **Face coherence :** Usually surface properties vary smoothly across a face. This allows the computations for one part of face to be used with incremental changes to the other parts of the face.
- **Edge coherence :** The visibility of edge may change only when it crosses a visible edge or penetrates a visible face.
- **Implied edge coherence** : If one planar face penetrates another their line of intersection can be determined from two points of intersection.
- **Area coherence :** A group of adjacent pixel is often belongs to the same visible face.
- **Span coherence :** It refers to a visibility of face over a span of adjacent pixels on a scan line. It is special case of area coherence.
- **Scan line coherence :** The set of visible object spans determined for one scan line of an image typically changes very little from the set on the previous line.
- **Depth coherence :** Adjacent parts of the same surface are typically same or very close depth. Therefore, once the depth at one point of the surface is determined the depth of the points on the rest of the surface can often be determined by at the most simple incremental calculation.
- **Frame Coherence :** Pictures of the same scene at two successive points in time are likely to be quite similar, except small changes in objects and view ports. Therefore, the calculations made for one picture can be reused for the next picture in a sequence.

8.2.2 Perspective Transformation

Visible-surface determination is done in a 3D space prior to the projection into 2D that destroys the depth information needed for depth comparisons, and depth comparisons are typically done after the normalizing transformation. Due to this projectors are parallel to the

know that scan line algorithm maintains the active edge list. This active edge list contains only edges that cross the current scan line, sorted in order of increasing x. The scan line method of hidden surface removal also stores a flag for each surface that is set on or off to indicate whether a position along a scan line is inside or outside of the surface. Scan lines are processed from left to right. At the leftmost boundary of a surface, the surface flag is turned ON; and at the rightmost boundary, it is turned OFF.

The Fig. 8.9 illustrates the scan line method for hidden surface removal. As shown in the Fig. 8.9, the active edge list for scan line 1 contains the information for edges AD, BC, EH and FG. For the positions along this scan line between edges AD and BC, only the flag for surface S_1 is ON. Therefore, no depth calculations are necessary, and intensity information for surface S_1 is entered into the frame buffer. Similarly, between edges EH and FG, only the flag for surface S_2 is ON and during that portion of scan line the intensity information for surface S_2 is entered into the frame buffer.

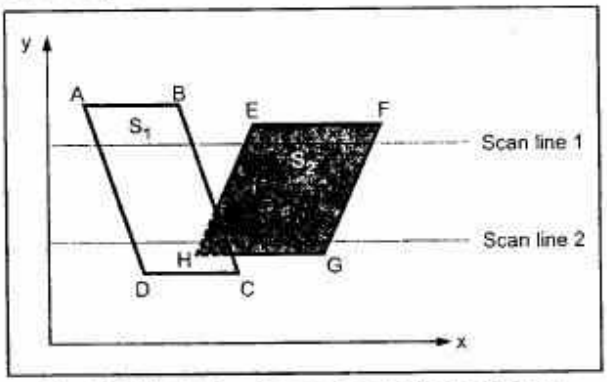


Fig. 8.9 Illustration of scan line method of hidden surface removal

For scan line 2 in the Fig. 8.9, the active edge list contains edges AD, EH, BC and FG. Along the scan line 2 from edge AD to edge EH, only the flag for surface S_1 is ON. However, between edges EH and BC, the flags for both surfaces are ON. In this portion of scan line 2, the depth calculations are necessary. Here we have assumed that the depth of S_1 is less than the depth of S_2 and hence the intensities of surface S_1 are loaded into the frame buffer. Then, for edge BC to edge FG portion of scan line 2 intensities of surface S_2 are entered into the frame buffer because during that portion only flag for S_2 is ON.

8.4.3 Z-Buffer Algorithm

One of the simplest and commonly used image space approach to eliminate hidden surfaces is the **Z-buffer** or **depth buffer** algorithm. It is developed by Catmull. This algorithm compares surface depths at each pixel position on the projection plane. The surface depth is measured from the view plane along the z axis of a viewing system. When object description is converted to projection coordinates (x, y, z), each pixel position on the view plane is specified by x and y coordinate, and z value gives the depth information. Thus object depths can be compared by comparing the z- values.

The Z-buffer algorithm is usually implemented in the normalized coordinates, so that z values range from 0 at the back clipping plane to 1 at the front clipping plane. The implementation requires another buffer memory called Z-buffer along with the frame buffer memory required for raster display devices. A Z-buffer is used to store depth values for each

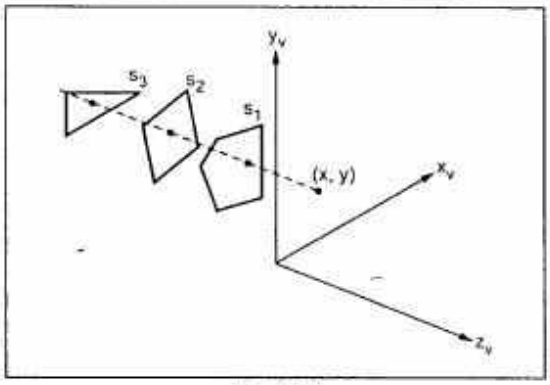


Fig. 8.10

(x, y) position as surfaces are processed, and the frame buffer stores the intensity values for each position. At the beginning Z-buffer is initialized to zero, representing the z-value at the back clipping plane, and the frame buffer is initialized to the background colour. Each surface listed in the display file is then processed, one scan line at a time, calculating the depth (z-value) at each (x, y) pixel position. The calculated depth value is compared to the value previously stored in the Z-buffer at that position. If the calculated depth values is greater than the value stored in the Z-buffer, the new depth value is stored, and the surface intensity at that position is determined and placed in the same xy location in the frame buffer.

For example, in Fig. 8.10 among three surfaces, surface S_1 has the smallest depth at view position (x, y) and hence highest z value. So it is visible at that position.

Z-buffer Algorithm

1. Initialize the Z-buffer and frame buffer so that for all buffer positions

   Z-buffer (x, y) = 0 and frame-buffer (x, y) = $I_{background}$

2. During scan conversion process, for each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visibility.

   Calculate z-value for each (x, y) position on the polygon

   If z > Z-buffer (x, y), then set

   Z-buffer (x, y) = z, frame-buffer (x, y) = $I_{surface}$ (x, y)

3. Stop

Note that, $I_{background}$ is the value for the background intensity, and $I_{surface}$ is the projected intensity value for the surface at pixel position (x, y). After processing of all surfaces, the Z-buffer contains depth values for the visible surfaces and the frame buffer contains the corresponding intensity values for those surfaces.

To calculate z-values, the plane equation

$$Ax + By + Cz + D = 0$$

is used where (x, y, z) is any point on the plane, and the coefficient A, B, C and D are constants describing the spatial properties of the plane. (Refer Appendix A for details)

Therefore, we can write

$$z = \frac{-Ax - By - D}{C}$$

Note, if at (x, y) the above equation evaluates to z_1, then at (x + Δx, y) the value of z, is

$$z_1 = \frac{A}{C}(\Delta x)$$

Only one subtraction is needed to calculate z(x + 1, y), given z(x, y), since the quotient A/C is constant and $\Delta x = 1$. A similar incremental calculation can be performed to determine the first value of z on the next scan line, decrementing by B/C for each Δy.

Advantages

1. It is easy to implement.
2. It can be implemented in hardware to overcome the speed problem.
3. Since the algorithm processes objects one at a time, the total number of polygons in a picture can be arbitrarily large.

Disadvantages

1. It requires an additional buffer and hence the large memory.
2. It is a time consuming process as it requires comparison for each pixel instead of for the entire polygon.

8.4.4 Warnock's Algorithm (Area Subdivision Algorithm)

An interesting approach to the hidden-surface problem was developed by Warnock. He developed area subdivision algorithm which subdivides each area into four equal squares. At each stage in the recursive-subdivision process, the relationship between projection of each polygon and the area of interest is checked for four possible relationships :

1. Surrounding Polygon - One that completely encloses the (shaded) area of interest (see Fig. 8.11 (a))
2. Overlapping or Intersecting Polygon - One that is partly inside and partly outside the area (see Fig. 8.11 (b))
3. Inside or Contained Polygon - One that is completely inside the area (see Fig. 8.11 (c)).
4. Outside or Disjoint Polygon - One that is completely outside the area (see Fig. 8.11 (d)).

(a) Surrounding (b) Overlapping (c) Inside or Contained (d) Outside or Disjoint

Fig. 8.11 Possible relationships with polygon surfaces and the area of interest

After checking four relationships we can handle each relationship as follows :

1. If all the polygons are disjoint from the area, then the background colour is displayed in the area.
2. If there is only one intersecting or only one contained polygon, then the area is first filled with the background colour, and then the part of the polygon contained in the area is filled with colour of polygon.
3. If there is a single surrounding polygon, but no intersecting or contained polygons, then the area is filled with the colour of the surrounding polygon.
4. If there are more than one polygon intersecting, contained in, or surrounding the area then we have to do some more processing.

See Fig. 8.12. In Fig. 8.12 (a), the four intersections of surrounding polygon are all closer to the viewpoint than any of the other intersections. Therefore, the entire area is filled with the colour of the surrounding polygon.

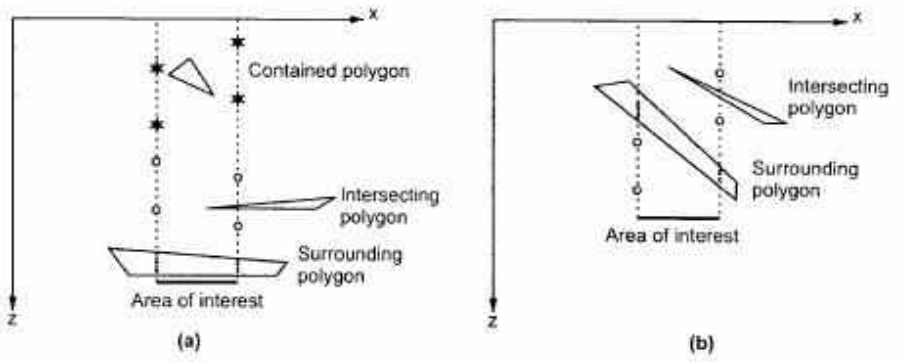


Fig. 8.12

However, Fig. 8.12 (b) shows that surrounding polygon is not completely in front of the intersecting polygon. In such case we cannot make any decision and hence Warnock's algorithm subdivides the area to simplify the problem. This is illustrated in Fig. 8.13. As shown in the Fig. 8.13 (a) we can not make any decision about which polygon is in front of the other. But after dividing area of interest polygon 1 is ahead of the polygon 2 in left area and polygon 2 is ahead of polygon 1 in the right area. Now we can fill these two areas with corresponding colours of the polygons.

The Warnock's algorithm stops subdivision of area only when the problem is simplified or when area is only a single pixel.

Advantages

1. It follows the divide-and-conquer strategy, therefore, parallel computers can be used to speed up the process.
2. Extra memory buffer is not required.

8.4.5 Back-Face Removal Algorithm

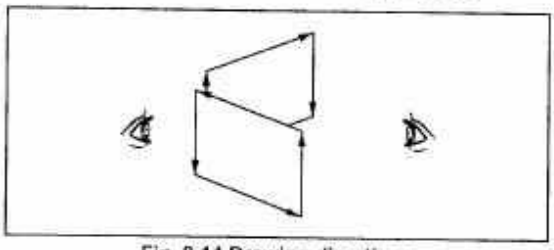


Fig. 8.14 Drawing directions

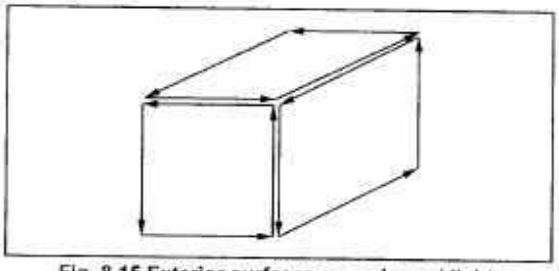


Fig. 8.15 Exterior surfaces are coloured light and drawn counter clockwise

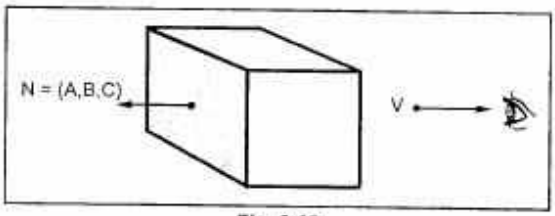


Fig. 8.16

We know that a polygon has two surfaces, a front and a back, just as a piece of paper does. We might picture our polygons with one side painted light and the other painted dark. But the question is "how to find which surface is light or dark". When we are looking at the light surface, the polygon will appear to be drawn with counter clockwise pen motions, and when we are looking at the dark surface the polygon will appear to be drawn with clockwise pen motions, as shown in the Fig. 8.14.

Let us assume that all solid objects are to be constructed out of polygons in such a way that only the light surfaces are open to the air; the dark faces meet the material inside the object. This means that when we look at an object face from the outside, it will appear to be drawn counterclockwise, as shown in the Fig. 8.15.

If a polygon is visible, the light surface should face towards us and the dark surface should face away from us. Therefore, if the direction of the light face is pointing towards the viewer, the face is visible (a front face), otherwise, the face is hidden (a back face) and should be removed.

The direction of the light face can be identified by examining the result

$$N \cdot V > 0$$

where

N : Normal vector to the polygon surface with cartesian components (A, B, C).

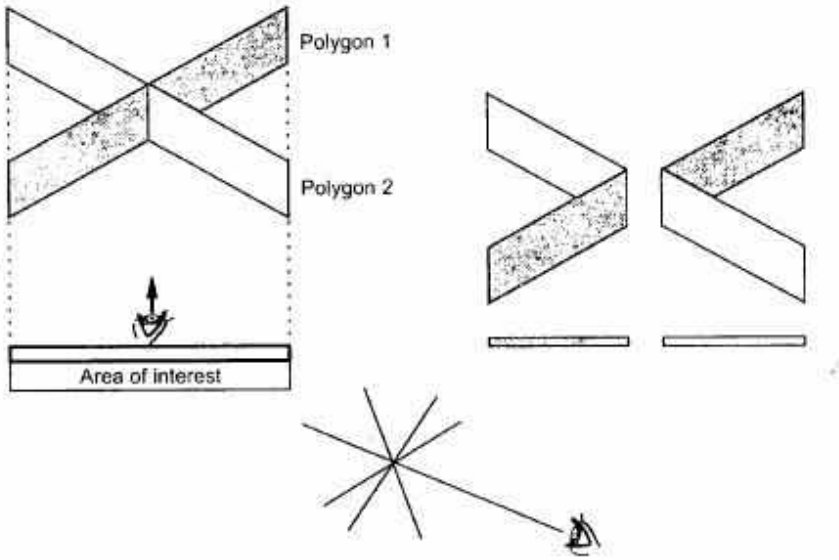


Fig. 8.13

Algorithm

1. Initialize the area to be the whole screen.
2. Create the list of polygons by sorting them with their z-values of vertices. Don't include disjoint polygons in the list because they are not visible.
3. Find the relationship of each polygon.
4. Perform the visibility decision test
   a) If all the polygons are disjoint from the area, then fill area with background colour.
   b) If there is only one intersecting or only one contained polygon then first fill entire area with background colour and then fill the part of the polygon contained in the area with the colour of polygon.
   c) If there is a single surrounding polygon, but no intersecting or contained polygons, then fill the area with the colour of the surrounding polygon.
   d) If surrounding polygon is closer to the viewpoint than all other polygons, so that all other polygons are hidden by it, fill the area with the colour of the surrounding polygon.
   e) If the **area is the pixel** (x, y), and neither a,b,c, nor d applies, compute the z coordinate at pixel (x, y) of all polygons in the list. The pixel is then set to colour of the polygon which is closer to the viewpoint.
5. If none of the above tests are true then subdivide the area and go to step 2.

V : A vector in the viewing direction from the eye (or "camera") position (Refer Fig. 8.16)

We know that, the dot product of two vector gives the product of the lengths of the two vectors times the cosine of the angle between them. This cosine factor is important to us because if the vectors are in the same direction ($0 \leq \theta < \pi/2$), then the cosine is positive and the overall dot product is positive. However, if the directions are opposite ($\pi/2 < \theta \leq \pi$), then the cosine and the overall dot product is negative (Refer Fig. 8.17).

Fig. 8.17 Cosine angles between two vectors

If the dot product is positive, we can say that the polygon faces towards the viewer; otherwise it faces away and should be removed.

In case, if object description has been converted to projection coordinates and our viewing direction is parallel to the viewing z_v axis, then $V = (0, 0, V_z)$ and

$$V \cdot N = V_z C$$

So that we only have to consider the sign of C, the Z component of the normal vector N. Now, if the z component is positive, then the polygon faces towards the viewer, if negative, it faces away.

Review Questions

1. Explain the two approaches used to determine hidden surfaces.
2. Discuss the techniques for efficient visible-surface algorithms.
3. What is coherence ? Discuss various types of coherence that can be used to make visible surface algorithms more efficient.
4. Write a short note on
   a) Perspective transformation
   b) Extents and bounding volumes
   c) Back-face culling
5. Explain the Robert's visible line algorithm.
6. Explain the Appel's visible line algorithm.
7. Explain the Haloed line algorithm.
8. Explain the painter's algorithm for hidden surface removal.
9. Explain the scanline algorithm for hidden surface removal.
10. Explain the Z-buffer algorithm for hidden surface removal.
11. List the advantages and disadvantages of Z-buffer algorithm.
12. Explain any one area subdivision algorithm for visible surface detection.
13. Describe the back face removal algorithm.

9.2.2 Interpolation

In the last section we have seen limitations of true curve generation approach. Furthermore in practice we have to deal with some complex curves for which no direct mathematical function is available. Such curves can be drawn using approximation methods. If we have set of sample points which lie on the required curve, then we can draw the required curve by filling portions of the curve with the pieces of known curves which pass through nearby sample points. The gap between the sample points can be filled by finding the co-ordinates of the points along the known approximating curve and connecting these points with line segments as shown in the Fig. 9.2.

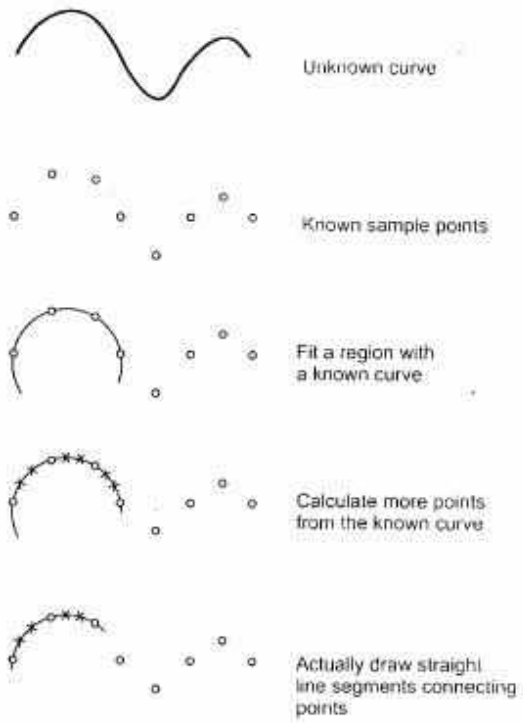


Fig. 9.2 The interpolation process

The main task in this process is to find the suitable mathematical expression for the known curve. There are polynomial, trigonometric, exponential and other classes of functions that can be used to approximate the curve. Usually polynomial functions in the

parametric form are preferred. The polynomial functions in the parametric form can be given as

$$x = f_x(u)$$
$$y = f_y(u)$$
$$z = f_z(u)$$

We can realise from above equations that the difference between 2 and 3 dimensions is just the addition of the third equation for z. Furthermore the parametric form treats all the three dimensions equally and allows multiple values (several values of y or z for a given x value). Due to these multiple values curves can double back or even cross themselves, as shown in Fig. 9.3.

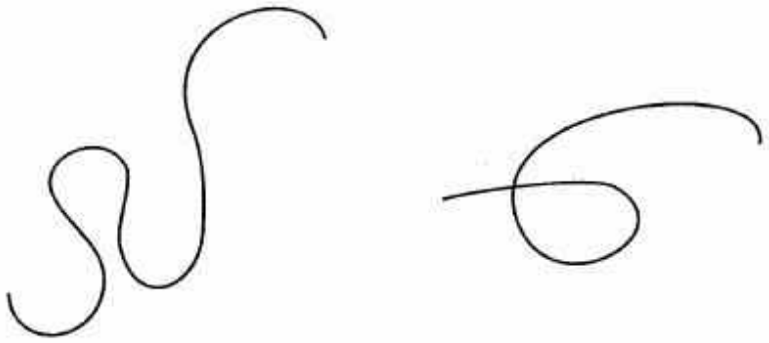


Fig. 9.3 Representation of curves with double back or crossing themselves

We have seen that, we have to draw the curve by determining the intermediate points between the known sample points. This can be achieved using interpolation techniques. Let's see the interpolation process.

Suppose we want a polynomial curve that will pass through n sample points.

$(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n)$

We will construct the function as the sum of terms, one term for each sample point. These functions can be given as

$$f_x(u) = \sum_{i=1}^{n} x_i B_i(u)$$

$$f_y(u) = \sum_{i=1}^{n} y_i B_i(u)$$

$$f_z(u) = \sum_{i=1}^{n} z_i B_i(u)$$

The function $B_i(u)$ is called '**blending function**'. For each value of parameter u, the blending function determines how much the i^{th} sample point affects the position of the

parametric form are preferred. The polynomial functions in the parametric form can be given as

$$x = f_x(u)$$
$$y = f_y(u)$$
$$z = f_z(u)$$

We can realise from above equations that the difference between 2 and 3 dimensions is just the addition of the third equation for z. Furthermore the parametric form treats all the three dimensions equally and allows multiple values (several values of y or z for a given x value). Due to these multiple values curves can double back or even cross themselves, as shown in Fig. 9.3.

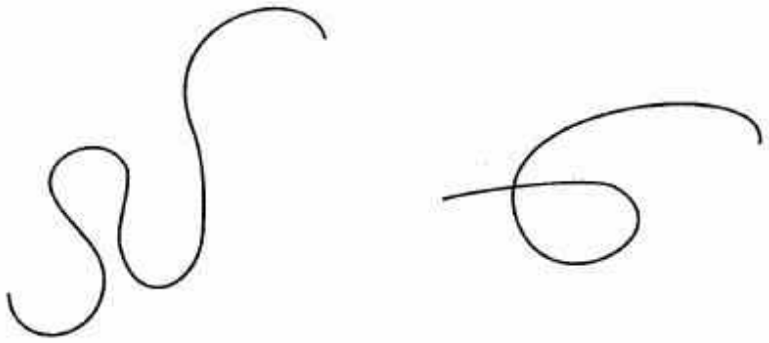


Fig. 9.3 Representation of curves with double back or crossing themselves

We have seen that, we have to draw the curve by determining the intermediate points between the known sample points. This can be achieved using interpolation techniques. Let's see the interpolation process.

Suppose we want a polynomial curve that will pass through n sample points.

$(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_n, y_n, z_n)$

We will construct the function as the sum of terms, one term for each sample point. These functions can be given as

$$f_x(u) = \sum_{i=1}^{n} x_i B_i(u)$$

$$f_y(u) = \sum_{i=1}^{n} y_i B_i(u)$$

$$f_z(u) = \sum_{i=1}^{n} z_i B_i(u)$$

The function $B_i(u)$ is called '**blending function**'. For each value of parameter u, the blending function determines how much the i^{th} sample point affects the position of the

curve. In other words we can say that each sample points tries to pull the curve in its direction and the function $B_i(u)$ gives the strength of the pull. If for some value of u, $B_i(u) = 1$ for unique value of i (i.e. $B_i(u) = 0$ for other values of i) then i^{th} sample point has complete control of the curve and the curve will pass through i^{th} sample point. For different value of u, some other sample point may have complete control of the curve. In such case the curve will pass through that point as well. In general, the blending functions give control of the curve to each of the sample points in turn for different values of u. Let's assume that the first sample point (x_1, y_1, z_1) has complete control when $u = -1$, the second when $u = 0$, the third when $u = 1$, and so on. i.e.

when $u = -1 \Rightarrow B_1(u) = 1$ and 0 for $u = 0, 1, 2, \ldots, n-2$

when $u = 0 \Rightarrow B_2(u) = 1$ and 0 for $u = -1, 1, \ldots, n-2$

$\vdots \quad \vdots \quad \vdots$

when $u = (n-2) \Rightarrow B_n(u) = 1$ and 0 for $u = -1, 0, \ldots, (n-1)$

To get $B_1(u) = 1$ at $u = -1$ and 0 for $u = 0, 1, 2, \ldots, n-2$, the expression for $B_1(u)$ can be given as

$$B_1(u) = \frac{u(u-1)(u-2)\ldots.[u-(n-2)]}{(-1)(-2)\ldots.(1-n)}$$

where denominator term is a constant used. In general form i^{th} blending function which is 1 at $u = i-2$ and 0 for other integers can be given as :

$$B_i(u) = \frac{(u+1)(u)(u-1)\ldots[u-(i-3)][u-(i-1)]\ldots.[u-(i-2)]}{(i-1)(i-2)(i-3)\ldots.(1)(-1)\ldots(i-n)}$$

The approximation of the curve using above expression is called **Lagrange interpolation.** From the above expression blending functions for four sample points can be given as

$$B_1(u) = \frac{u(u-1)(u-2)}{(-1)(-2)(-3)}$$

$$B_2(u) = \frac{(u+1)(u-1)(u-2)}{1(-1)(-2)}$$

$$B_3(u) = \frac{(u+1)u(u-2)}{(2)(1)(-1)}$$

$$B_4(u) = \frac{(u+1)u(u-1)}{(3)(2)(1)}$$

Using above blending functions, the expression for the curve passing through sampling points can be realised as follows :

$$x = x_1 B_1(u) + x_2 B_2(u) + x_3 B_3(u) + x_4 B_4(u)$$
$$y = y_1 B_1(u) + y_2 B_2(u) + y_3 B_3(u) + y_4 B_4(u)$$
$$z = z_1 B_1(u) + z_2 B_2(u) + z_3 B_3(u) + z_4 B_4(u)$$

It is possible to get intermediate points between two sampling points by taking values of u between the values of u related to the two sample points under consideration. For

example, we can find the intermediate points between second and third sample points for which values of u are 0 and 1, respectively; by taking values of u between 0 and 1. This is shown in Fig. 9.4.

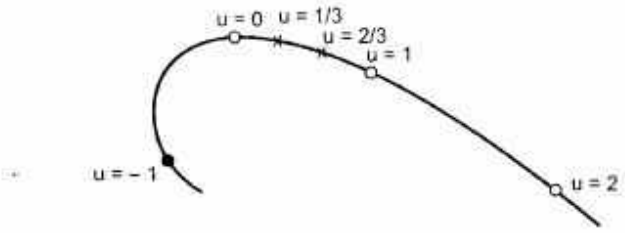


Fig. 9.4 Determining intermediate points for approximation of curve

The subsequent intermediate points can be obtained by repeating the same procedure. Finally the points obtained by this procedure are joined by small straight line segments to get the approximated curve.

Initially, sample points (1, 2, 3, 4) are considered and intermediate points between (2, 3) are obtained. Then sample point at one end is discarded and sample point at the other end is added to get new sample points (2, 3, 4, 5). Now the curve between sample points (3, 4) is approximated. The subsequent intermediate points can be obtained by repeating the same procedure. The initial and final portions of the curve require special treatment. For the first four points (1, 2, 3, 4) we have to draw region between points (1, 2) with u values between – 1 and 0.

Similarly the blending function for very last step of the curve should be evaluated with u values between 1 and 2.

Interpolating Algorithm

1. Get the sample points.
2. Get intermediate values of u to determine intermediate points.
3. Calculate blending function values for middle section of the curve.
4. Calculate blending function values for first section of the curve.
5. Calculate blending function values for the last section of the curve.
6. Multiply the sample points by blending functions to give points on approximation curve.
7. Connect the neighbouring points using straight line segments
8. Stop.

9.3 Spline Representation

To produce a smooth curve through a designated set of points, a flexible strip called **spline** is used. Such a spline curve can be mathematically described with a piecewise cubic polynomial function whose first and second derivatives are continuous across the various

curve sections. We can specify a spline curve by giving a set of coordinate positions, called **control points**, which indicates the general shape of the curve. When polynomial sections are fitted so that the curve passes through all control points, as shown in the Fig. 9.5 (a), the resulting curve is said to **interpolate** the set of control points. On the other hand, when the polynomials are fitted to the path which is not necessarily passing through all control points, the resulting curve is said to approximate the set of control points. This is illustrated in the Fig. 9.5 (b).

Fig. 9.5

9.3.1 Geometric and Parametric Continuity

To ensure a smooth transition from one section of a piecewise parametric curve to the next, we can impose various continuity conditions at the connection points. We see parametric continuity and geometric continuity conditions.

In geometric continuity we require parametric derivatives of two sections to be proportional to each other at their common boundary instead of equal to each other. Parametric continuity is set by matching the parametric derivatives of adjoining two curve sections at their common boundary. In zero order parametric continuity, given as C^0, it means simply the curve meet and same is for zero order geometric continuity. In first order parametric continuity called as C^1 means that first parametric derivatives of the coordinate functions for two successive curve sections are equal at the joining point and geometric first order continuity means the parametric first derivative are proportional at the intersection of two successive sections. Second order parametric continuity or C^2 continuity means that both the first and second parametric derivatives of the two curve sections are same at the intersection and for second order geometric continuity or C^2 continuity means that both the first and second parametric derivatives of the two curve sections are proportional at their boundary. Under C^2 continuity curvature of the two curve sections match at the joining positions.

Two curves

$$r(t) = (t^2 - 2t, t)$$
$$n(t) = (t^2 + 1, t + 1)$$

C^1 and G^1 are continuous at $r(1) = n(0)$

Derivative

$$r(t) = 2t - 2, 1$$
$$r(1) = 2 - 2, 1$$
$$= 0, 1$$

Derivative $n(t) = 2t, 1$

$n(0) = 0, 1$

$\therefore$ $r(1) = n(0)$, two curves are continuous.

Ex. 9.1 : *Show that two curves $n(t) = (t^2 + 2t - 2, t^2)$ and $r(t) = (t^2 + 2t + 1, t + 1)$ are both C^0 and G^0 continuous where they join at $n(1) = r(0)$. Do they meet C^1 and G^1 continuity.*

Sol. : $n(t) = (t^2 + 2t - 2, t^2)$

$r(t) = (t^2 + 2t + 1, t + 1)$

Zero order parametric continuity, described as C^0 continuity, means simply that the curves meet. That is, the values of x, y and z evaluated at u_2 for the first curve section are equal, respectively, to the values of x, y and z evaluated at u_1 for the next curve section. The **zero-order geometric continuity** described as G^0 continuity, is the same as zero-order parametric continuity.

We have,

$$n(1) = (1^2 + 2 - 2, 1^2)$$
$$= (1, 1)$$
$$r(0) = (0^2 + 0 + 1, 0 + 1)$$
$$= (1, 1)$$

Therefore, we can say that both curves are C^0 and G^0 continuous at n(1) and r(0). To check for C^1 and G^1 continuity we have to take first derivative of both the curves :

$$\text{Derivative } n(t) = (2t + 2, 2t)$$
$$\text{Derivative } r(t) = (2t + 2, 1)$$
$$\therefore \quad n(1) = (2 + 2, 2)$$
$$= (4, 2)$$
$$r(0) = (2, 1)$$

Since $n(1) \neq r(0)$, the two curves are not C^1 and G^1 continuous at n(1) and r(0).

9.3.2 Spline Specifications

There are three basic ways of specifying spline curve :

- We can state the set of boundary conditions that are imposed on the spline
- We can state the matrix that characteristics the spline or
- We can state the set of blending functions that calculate the positions along the curve path by specifying combination of geometric constraints on the curve.

Why to use cubic polynomials ?

Polylines and polygons are first-degree, piecewise linear approximation to curves and surfaces, respectively. But this lower degree polynomials give too little flexibility in controlling the shape of the curve. The higher-degree polynomials give reasonable design flexibility, but introduce unwanted wiggles and also require more computation. For this reason the third-degree polynomials are most often used for representation of curves. These polynomials are commonly known as cubic polynomials.

We can describe the parametric cubic polynomial that is to be fitted between each pair of control points with the following set of equations :

$$x(u) = ax\,u^3 + bx\,u^2 + cx\,u + dx$$
$$y(u) = ay\,u^3 + by\,u^2 + cy\,u + dy$$
$$z(u) = az\,u^3 + bz\,u^2 + cz\,u + dz \qquad (0 \le u \le 1) \qquad \ldots (9.6)$$

For each of these three equations, we need to determine the values of the four coefficients a, b, c and d in the polynomial representation for each of the n curve sections between the n + 1 control points. We do this by setting enough boundary conditions at the joints between curve sections so that we can obtain numerical values for all the coefficient. Let us see the common methods for setting the boundary conditions for cubic interpolation splines.

9.4 Bezier Curves

Bezier curve is an another approach for the construction of the curve. A Bezier curve is determined by a defining polygon. Bezier curves have a number of properties that make them highly useful and convenient for curve and surface design. They are also easy to implement. Therefore Bezier curves are widely available in various CAD systems and in general graphic packages. In this section we will discuss the cubic Bezier curve. The reason for choosing cubic Bezier curve is that they provide reasonable design flexibility and also avoid the large number of calculations.

Properties of Bezier curve

1. The basis functions are real.
2. Bezier curve always passes through the first and last control points i.e. curve has same end points as the guiding polygon.
3. The degree of the polynomial defining the curve segment is one less than the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is three, i.e. cubic polynomial.
4. The curve generally follows the shape of the defining polygon.
5. The direction of the tangent vector at the end points is the same as that of the vector determined by first and last segments.
6. The curve lies entirely within the convex hull formed by four control points.
7. The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
8. The curve exhibits the variation diminishing property. This means that the curve does not oscillate about any straight line more often than the defining polygon.
9. The curve is invariant under an affine transformation.

In cubic Bezier curve four control points are used to specify complete curve. Unlike the B-spline curve, we do not add intermediate points and smoothly extend Bezier curve, but we pick four more points and construct a second curve which can be attached to the first.

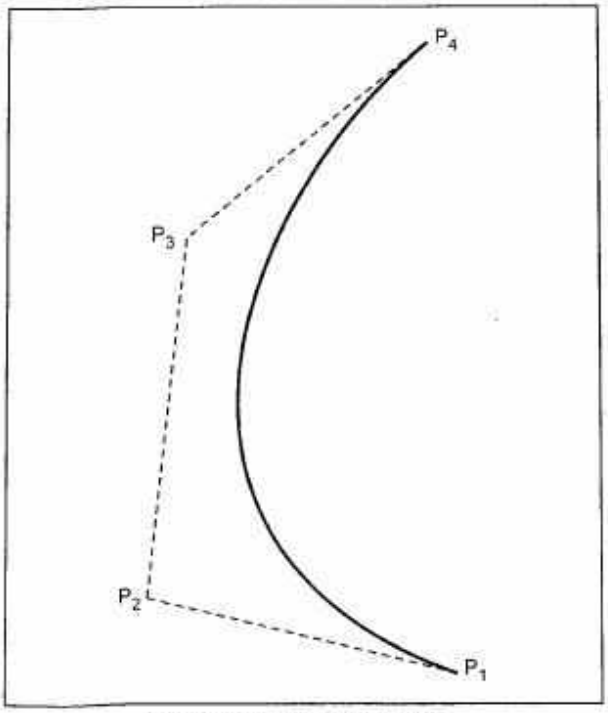


Fig. 9.6 A cubic Bezier spline

The second curve can be attached to the first curve smoothly by selecting appropriate control points.

Fig. 9.6 shows the Bezier curve and its four control points. As shown in the Fig. 9.6, Bezier curve begins at the first control point and ends at the fourth control point. This means that if we want to connect two Bezier curves, we have to make the first control point of the second Bezier curve match the last control point of the first curve. We can also observe that at the start of the curve, the curve is tangent to the line connecting first and second control points. Similarly at the end of curve, the curve is tangent to the line connecting the third and fourth control point. This means that, to join two Bezier curves smoothly we have to place the third and the fourth control points of the first curve on the same line specified by the first and the second control points of the second curve.

The Bezier matrix for periodic cubic polynomial is

$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\therefore \quad P(u) = U \cdot M_B \cdot G_B$$

where $$G_B = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

and the product $P(u) = U.M_B.G_B$ is

$$P(u) = (1-u)^3 P_1 + 3u(1-u)^2 P_2 + 3u^2(1-u) P_3 + u^3 P_4$$

Ex. 9.2 *Construct the Bezier curve of order 3 and with 4 polygon vertices A(1, 1), B(2, 3), C(4, 3) and D(6, 4).*

Sol. : The equation for the Bezier curve is given as

$$P(u) = (1-u)^3 P_1 + 3u(1-u)^2 P_2 + 3u^2(1-u) P_3 + u^3 P_4$$

for $0 \le u \le 1$

where P(u) is the point on the curve P_1, P_2, P_3, P_4

Let us take $u = 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}$

$$\therefore \quad P(0) = P_1 = (1, 1)$$

$$\therefore \quad P\left(\frac{1}{4}\right) = \left(1-\frac{1}{4}\right)^3 P_1 + 3\frac{1}{4}\left(1-\frac{1}{4}\right)^2 P_2 + 3\left(\frac{1}{4}\right)^2\left(1-\frac{1}{4}\right)P_3 + \left(\frac{1}{4}\right)^3 P_4$$

$$= \frac{27}{64}(1,1) + \frac{27}{64}(2,3) + \frac{9}{64}(4,3) + \frac{1}{64}(6,4)$$

$$= \left[\frac{27}{64}\times 1 + \frac{27}{64}\times 2 + \frac{9}{64}\times 4 + \frac{1}{64}\times 6, \quad \frac{27}{64}\times 1 + \frac{27}{64}\times 3 + \frac{9}{64}\times 3 + \frac{1}{64}\times 4\right]$$

$$= \left[\frac{123}{64}, \frac{139}{64}\right]$$

$$= (1.9218, 2.1718)$$

$$\therefore \quad P\left(\frac{1}{2}\right) = \left(1-\frac{1}{2}\right)^3 P_1 + 3\frac{1}{2}\left(1-\frac{1}{2}\right)^2 P_2 + 3\left(\frac{1}{2}\right)^2\left(1-\frac{1}{2}\right)P_3 + \left(\frac{1}{2}\right)^3 P_4$$

$$= \frac{1}{8}(1,1) + \frac{3}{8}(2,3) + \frac{3}{8}(4,3) + \frac{1}{8}(6,4)$$

$$= \left[\frac{1}{8}\times 1 + \frac{3}{8}\times 2 + \frac{3}{8}\times 4 + \frac{1}{8}\times 6, \quad \frac{1}{8}\times 1 + \frac{3}{8}\times 3 + \frac{3}{8}\times 3 + \frac{1}{8}\times 4\right]$$

$$= \left[\frac{25}{8}, \frac{23}{8}\right]$$

$$= (3.125, \quad 2.875)$$

$$\therefore \quad P\left(\frac{3}{4}\right) = \left(1-\frac{3}{4}\right)^3 P_1 + 3\frac{3}{4}\left(1-\frac{3}{4}\right)^2 P_2 + 3\left(\frac{3}{4}\right)^2\left(1-\frac{3}{4}\right)P_3 + \left(\frac{3}{4}\right)^3 P_4$$

$$= \frac{1}{64}P_1 + \frac{9}{64}P_2 + \frac{27}{64}P_3 + \frac{27}{64}P_4$$

$$= \frac{1}{64}(1,1) + \frac{9}{64}(2,3) + \frac{27}{64}(4,3) + \frac{27}{64}(6,4)$$

$$= \left[\frac{1}{64}\times 1 + \frac{9}{64}\times 2 + \frac{27}{64}\times 4 + \frac{27}{64}\times 6, \quad \frac{1}{64}\times 1 + \frac{9}{64}\times 3 + \frac{27}{64}\times 3 + \frac{27}{64}\times 4\right]$$

$$= \left[\frac{289}{64}, \frac{217}{64}\right]$$

$$= (4.5156, \quad 3.375)$$

$$P(1) = r_3 = (6, 4)$$

The Fig. 9.7 shows the calculated points of the Bezier curve and curve passing through it

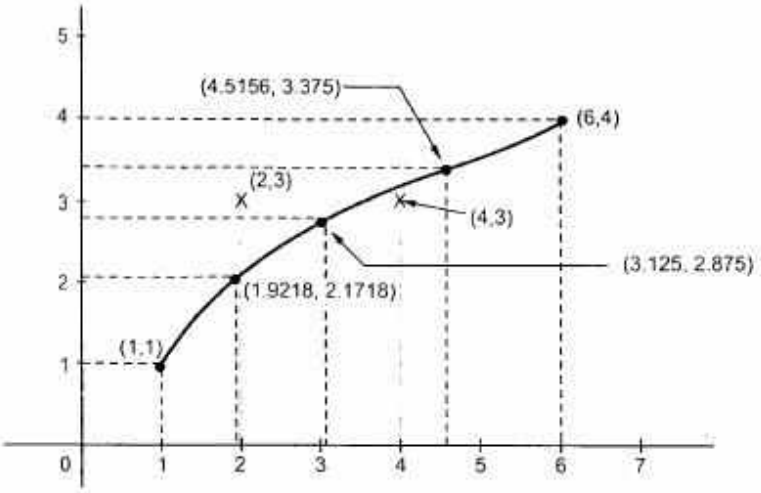


Fig. 9.7 Plotted Bezier curved

Another approach to construct the Bezier curve is called midpoint approach. In this approach the Bezier curve can be constructed simply by taking midpoints. In midpoint approach midpoints of the lines connecting four control points (A, B, C, D) are determined (AB, BC, CD). These midpoints are connected by line segments and their midpoints ABC and BCD are determined. Finally these two midpoints are connected by line segments and its midpoint ABCD is determined. This is illustrated in Fig. 9.8.

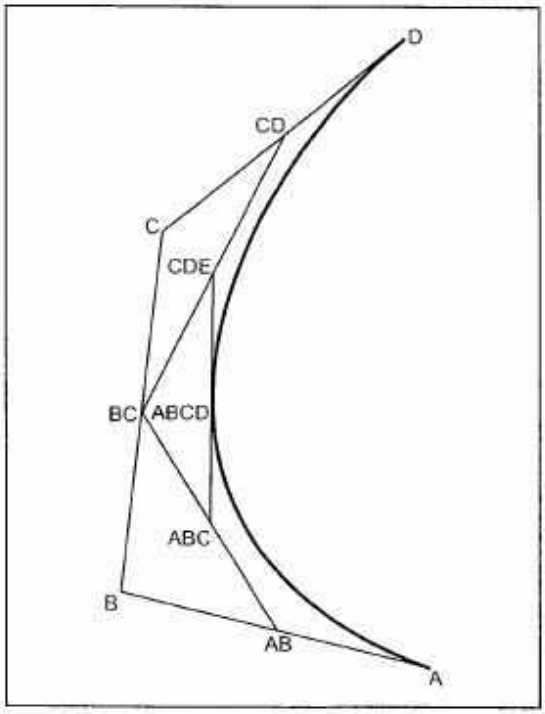


Fig. 9.8 Subdivision of a Bezier spline

The point ABCD on the Bezier curve divides the original curve into two sections. This makes the points A, AB, ABC and ABCD are the control points for the first section and the points ABCD, BCD, CD and D are the control points for the second section. By considering two sections separately we can get two more sections for each separate section i.e. the original Bezier curve gets divided into four different curves. This process can be repeated to split the curve into smaller sections until we have sections so short that they can be replaced by straight lines or even until the sections are not bigger than individual pixels.

```
        if(gd<0)
                {
                puts("CANNOT DETECT A GRAPHICS CARD");
                exit(1);
                }
        initgraph(&gd,&gm,"f:\\tc");
        }
main()
        {
        int i;
        float temp1,temp2;
        igraph();

        /* Read two end points and two control points of the curve
        ---------------------------------------------------------- */
        for(i=0;i<4;i++)
        {
        printf("Enter (x,y) coordinates of point%d : ",i+1);
        scanf("%f,%f",&temp1,&temp2);
        xxx[i][0] = temp1;
        xxx[i][1] = temp2;
        }
        bezier(xxx[1][0],xxx[1][1],xxx[2][0],xxx[2][1],xxx[3][0],xxx
        [3][1],8);
        getch();
        closegraph();
}
```

9.5 B-Spline Curves

We have seen that, a curve generated by using the vertices of a defining polygon is dependent on some interpolation or approximation scheme to establish the relationship between the curve and the polygon. This scheme is provided by the choice of basis function. The Bezier curve produced by the Bernstein basis function has a limited flexibility. First the number of specified polygon vertices fixes the order of the resulting polynomial which defines the curve. For example, polygon with four vertices results a cubic polynomial curve. The only way to reduce the degree of the curve is to reduce the number of vertices, and conversely the only way to increase the degree of the curve is to increase the number of vertices. The second limiting characteristics is that the value of the blending function is nonzero for all parameter values over the entire curve. Due to this change in one vertex, changes the entire curve and this eliminates the ability to produce a local change with in a curve.

There is another basis function, called the B-spline basis, which contains the Bernstein basis as a special case. The B-spline basis is nonglobal. It is nonglobal because each vertex B_i is associated with a unique basis function. Thus, each vertex affects the shape of the curve only over a range of parameter values where its associated basis function is nonzero. The B-spline basis also allows the order of the basis function and hence the degree of the resulting curve is independent on the number of vertices. It is possible to change the degree of the resulting curve without changing the number of vertices of the defining polygon.

If P(u) be the position vectors along the curve as a function of the parameter u, a B-spline curve is given by

$$P(u) = \sum_{i=1}^{n+1} B_i N_{i,k}(u) \quad u_{min} \le u < u_{max}, \quad 2 \le k \le n+1 \quad ...(9.7)$$

where the B_i are the position vectors of the n + 1 defining polygon vertices and the $N_{i,k}$ are the normalized B-spline basis functions. For the i^{th} normalized B-spline basis function of order k, the basis function $N_{i,k}(u)$ are defined as

$$N_{i,1}(u) = \begin{cases} 1 & \text{if} \quad x_i \le u < x_{i+1} \\ 0 & \text{Otherwise} \end{cases}$$

and

$$N_{i,k}(u) = \frac{(u - x_i) N_{i,k-1}(u)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - u) N_{i+1,k-1}(u)}{x_{i+k} - x_{i+1}} \quad ... (9.8)$$

The values of x_i are the elements of a knot vector satisfying the relation $x_i \le x_{i+1}$. The parameter u varies from u_{min} to u_{max} along the curve P(u). The choice of knot vector has a significant influence on the B-spline basis functions $N_{i,k}(u)$ and hence on the resulting B-spline curve. There are three types of knot vector : uniform, open uniform and nonuniform.

In a uniform knot vector, individual knot values are evenly spaced. For example,

[0 1 2 3 4]

For a given order k, uniform knot vectors give periodic uniform basis functions for which

$$N_{i,k}(u) = N_{i-1,k}(u-1) = N_{i+1,k}(u+1)$$

An open uniform knot vector has multiplicity of knot values at the ends equal to the order k of the B-spline basis function. Internal knot values are evenly spaced. Examples are,

k = 2 [0 0 1 2 3 3]
k = 3 [0 0 0 1 2 3 3 3]
k = 4 [0 0 0 0 1 2 2 2 2]

Generally, an open uniform knot vector is given by,

$$\begin{aligned} x_i &= 0 && 1 \le i \le k \\ x_i &= i - k && k+1 \le i \le n+1 \\ x_i &= n - k + 2 && n+2 \le i \le n+k+1 \end{aligned} \quad ... (9.9)$$

The curves resulted by the use of open uniform basis function are nearly like Bezier curves. In fact, when the number of defining polygon vertices is equal to the order of the B-spline basis and an open uniform knot vector is used, the B-spline basis reduces to the Bernstein basis. Hence, the resulting B-spline curve is a Bezier curve.

Ex. 9.3 *Calculate the four third-order basis function $N_{i,3}(u)$, $i = 1, 2, 3, 4$ with an open uniform knot vector.*

Sol. : We have to calculate four basis functions, therefore n = (4 – 1) = 3 and it is of order three, therefore k = 3. From equation 9.9 the open uniform knot vector is given as

$$[X] = [0\,0\,0\,1\,2\,2\,2]$$

Now, from equation 9.8, the basis functions for various parameters are as follows :

$0 \le u < 1$

$N_{3,1}(u) = 1;$ $\quad N_{i,1}(u) = 0, \quad i \neq 3$

$N_{2,2}(u) = 1 - u;$ $\quad N_{3,2}(u) = u, \quad N_{i,2}(u) = 0 \quad i \neq 2, 3$

$N_{1,3}(u) = (1-u)^2;$ $\quad N_{2,3}(u) = u(1-u) + \dfrac{(2-u)}{2}u$

$N_{3,3}(u) = \dfrac{u^2}{2};$ $\quad N_{i,3}(u) = 0; \quad i \neq 1, 2, 3$

$1 \le u < 2$

$N_{4,1}(u) = 1;$ $\quad N_{i,1}(u) = 0, \quad i \neq 4$

$N_{3,2}(u) = (2-4);$ $\quad N_{4,2}(u) = (u-1); \quad N_{i,2}(u) = 0, i \neq 3, 4$

$N_{2,3}(u) = \dfrac{(2-u)^2}{2};$ $\quad N_{3,3}(t) = \dfrac{u(2-u)}{2} + (2-u)(u-1);$

$N_{4,3}(u) = (u-1)^2;$ $\quad N_{i,3}(u) = 0; \quad i \neq 2, 3, 4$

Ex. 9.4 *Construct the B-spline curve of order 4 and with 4 polygon vertices A(1, 1), B(2, 3) C(4, 3) and D(6, 2).*

Sol. : Here n = 3 and k = 4 from equation 9.9 we have open uniform knot vector as

x = [0 0 0 0 1 1 1 1] and from equation 9.8 we have basis functions are

$0 \le u < 1$

$N_{4,1}(u) = 1;$ $\quad N_{i,1}(u) = 0, \quad i \neq 4$

$N_{3,2}(u) = (1-u);$ $\quad N_{4,2}(u) = u, \quad N_{i,2}(u) = 0, \quad i \neq 3, 4$

$N_{2,3}(u) = (1-u)^2;$ $\quad N_{3,3}(u) = 24(1-u);$

$N_{4,3}(u) = u^2;$ $\quad N_{i,3}(u) = 0; \quad i \neq 2, 3, 4$

$N_{1,4}(u) = (1-t)^3;$ $\quad N_{2,4}(u) = u(1-u)^2 + 2u(1-u)^2 = 3u(1-u)^2;$

$N_{3,4}(u) = 2u^2(1-u) + (1-u)u^2 = 3u^2(1-u);\ N_{4,4}(u) = u^3$

Using equation 1 the parametric B-spline is

$$P(u) = AN_{1,4}(u) + BN_{2,4}(u) + CN_{3,4}(u) + DN_{4,4}(u)$$

$$\therefore P(u) = (1-u)^3 A + 3u(1-u)^2 B + 3u^2(1-u)C + u^3 D$$

Let us take $u = 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1$

$$\therefore P(0) = A = [1, 1]$$

$$\therefore P\left(\frac{1}{4}\right) = \left(1-\frac{1}{4}\right)^3 A + 3\frac{1}{4}\left(1-\frac{1}{4}\right)^2 B + 3\left(\frac{1}{4}\right)^2\left(1-\frac{1}{4}\right)C + \left(\frac{1}{4}\right)^3 D$$

$$= \left(\frac{27}{64}\right)A + \left(\frac{27}{64}\right)B + \left(\frac{9}{64}\right)C + \left(\frac{1}{64}\right)D$$

$$= \left[\frac{27}{64}(1,1) + \frac{27}{64}(2,3) + \frac{9}{64}(4,3) + \frac{1}{64}(6,2)\right]$$

$$= \left[\frac{27}{64}\times 1 + \frac{27}{64}\times 2 + \frac{9}{64}\times 4 + \frac{1}{64}\times 6,\ \frac{27}{64}\times 1 + \frac{27}{64}\times 3 + \frac{9}{64}\times 3 + \frac{1}{64}\times 2\right]$$

$$= \left[\frac{123}{64}, \frac{137}{64}\right]$$

$$= [1.9218,\ 2.14]$$

$$\therefore P\left(\frac{1}{2}\right) = \left(1-\frac{1}{2}\right)^3 A + 3\frac{1}{2}\left(1-\frac{1}{2}\right)^2 B + 3\left(\frac{1}{2}\right)^2\left(1-\frac{1}{2}\right)C + \left(\frac{1}{2}\right)^3 D$$

$$= \frac{1}{8}A + \frac{3}{8}B + \frac{3}{8}C + \frac{1}{8}D$$

$$= \left[\frac{1}{8}(1,1) + \frac{3}{8}(2,3) + \frac{3}{8}(4,3) + \frac{1}{8}(6,2)\right]$$

$$= \left[\frac{1}{8}\times 1 + \frac{3}{8}\times 2 + \frac{3}{8}\times 4 + \frac{1}{8}\times 6,\ \frac{1}{8}\times 1 + \frac{3}{8}\times 3 + \frac{3}{8}\times 3 + \frac{1}{8}\times 2\right]$$

$$= \left[\frac{25}{8}, \frac{21}{8}\right]$$

$$= [3.125,\ 2.625]$$

$$\therefore P\left(\frac{3}{4}\right) = \left(1-\frac{3}{4}\right)^3 A + 3\frac{3}{4}\left(1-\frac{3}{4}\right)^2 B + 3\left(\frac{3}{4}\right)^2\left(1-\frac{3}{4}\right)C + \left(\frac{3}{4}\right)^3 D$$

$$= \frac{1}{64}A + \frac{9}{64}B + \frac{27}{64}C + \frac{27}{64}D$$

$$= \frac{1}{64}(1,1) + \frac{9}{64}(2,3) + \frac{27}{64}(4,3) + \frac{27}{64}(6,2)$$

$$= \left[\frac{1}{64}\times 1 + \frac{9}{64}\times 2 + \frac{27}{64}\times 4 + \frac{27}{64}\times 6,\ \frac{1}{64}\times 1 + \frac{9}{64}\times 3 + \frac{27}{64}\times 3 + \frac{27}{64}\times 2\right]$$

$$= \left[\frac{289}{64}, \frac{163}{64}\right]$$

$= [4.5156, 2.5468]$

$\therefore P(1) = D = [6, 2]$

The Fig. 9.9 shows the calculated points of the B-spline curve and curve passing through it.

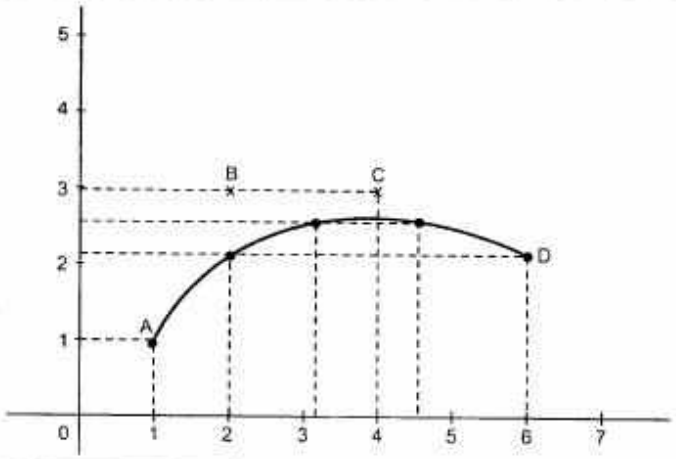


Fig. 9.9 Plotted B-spline curve

Properties of B-spline curve

- The sum of the B-spline basis functions for any parameter value u is 1.

  i.e. $\sum_{i=1}^{n+1} N_{i,k}(u) \equiv 1$

- Each basis function is positive or zero for all parameter values, i.e., $N_{i,k} \geq 0$.
- Except for k = 1 each basis function has precisely one maximum value.
- The maximum order of the curve is equal to the number of vertices of defining polygon.
- The degree of B-spline polynomial is independent on the number of vertices of defining polygon (with certain limitations).
- B-spline allows local control over the curve surface because each vertex affects the shape of a curve only over a range of parameter values where its associated basis function is nonzero.
- The curve exhibits the variation diminishing property. Thus the curve does not oscillate about any straight line move often than its defining polygon.
- The curve generally follows the shape of defining polygon.
- Any affine transformation can be applied to the curve by applying it to the vertices of defining polygon.
- The curve line within the convex hull of its defining polygon.

9.6 Parametric Bicubic Surfaces

Parametric bicubic surfaces are a generalization of parametric cubic curves. In section 9.3 we have seen the general form of parametric cubic curve

$$P(u) = U \cdot M \cdot G$$

If we now allow the points in G to vary in 3D along some path that is parameterized on t, we have

$$P(u, t) = U \cdot M \cdot G(t) = U \cdot M \cdot \begin{bmatrix} G_1(t) \\ G_2(t) \\ G_3(t) \\ G_4(t) \end{bmatrix} \quad \ldots(9.10)$$

Now, different values of t between 0 to 1 we get different curves. For slight different values of t we get slightly different curves. The set of all such curves arbitrarily close to each other for values of t between 0 and 1 defines a surface. If the $G_i(t)$ are themselves cubics, the surface is said to be a parametric bicubic surface, and $G_i(t)$ can be represented as

$$G_i(t) = U \cdot M \cdot G_i \quad \ldots (9.11)$$

where $G_i = [g_{i1}\ g_{i2}\ g_{i3}\ g_{i4}]^T$ and

g_{i1} is the first element of the geometry vector for curve $G_i(t)$.

The transpose of equation (9.11) can be given as

$$G_i(t)^T = G_i^T \cdot M^T \cdot U^T \quad \because (A \cdot B \cdot C)^T = C^T \cdot B^T \cdot A^T$$

Substituting the above result in equation (9.10)

We have

$$\begin{aligned} P(u, t) &= U \cdot M \cdot G_i^T \cdot M^T \cdot U^T \\ &= U \cdot M \cdot [g_{i1}\ g_{i2}\ g_{i3}\ g_{i4}] \cdot M^T \cdot U^T \\ &= U \cdot M \cdot \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix} \cdot M^T \cdot U^T \quad \ldots (9.12) \\ &= U \cdot M \cdot G \cdot M^T \cdot U^T \quad \text{where } 0 \le u, t \le 1 \quad \ldots (9.13) \end{aligned}$$

In terms of x, y, z separately the above equation can be written as

$$\begin{aligned} x(u, t) &= U \cdot M \cdot G_x \cdot M^T \cdot U^T \\ y(u, t) &= U \cdot M \cdot G_y \cdot M^T \cdot U^T \\ z(u, t) &= U \cdot M \cdot G_z \cdot M^T \cdot U^T \quad \ldots (9.14) \end{aligned}$$

9.6.1 Hermite Surfaces

The parametric bicubic equation for Hermite surface can be given as

$$P(u, t) = U \cdot M_H \cdot G_H(t) = U \cdot M_H \cdot \begin{bmatrix} P_1(t) \\ P_4(t) \\ DP_1(t) \\ DP_4(t) \end{bmatrix} \quad \ldots (9.15)$$

where

$$P_1(t) = U \cdot M_H \cdot \begin{bmatrix} g_{11} \\ g_{12} \\ g_{13} \\ g_{14} \end{bmatrix}$$

$$P_4(t) = U \cdot M_H \cdot \begin{bmatrix} g_{21} \\ g_{22} \\ g_{23} \\ g_{24} \end{bmatrix}$$

$$DP_1(t) = U \cdot M_H \cdot \begin{bmatrix} g_{31} \\ g_{32} \\ g_{33} \\ g_{34} \end{bmatrix}$$

$$DP_4(t) = U \cdot M_H \cdot \begin{bmatrix} g_{41} \\ g_{42} \\ g_{43} \\ g_{44} \end{bmatrix}$$

The above four equations can be rewritten together as

$$[P_1(t) \;\; P_4(t) \;\; DP_1(t) \;\; DP_4(t)] = U \cdot M_H \cdot G_H^T \quad \ldots (9.16)$$

where,

$$G_H = \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix}$$

Taking transpose of both sides we have

$$\begin{bmatrix} P_1(t) \\ P_4(t) \\ DP_1(t) \\ DP_4(t) \end{bmatrix} \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix} \cdot M_H^T \cdot U^T = G_H \cdot M_H^T \cdot U^T \quad \ldots (9.17)$$

Substituting the equation (9.17) in equation (9.15) we have,

$$P(u, t) = U \cdot M_H \cdot G_H \cdot M_H^T \cdot U^T \quad \ldots (9.18)$$

In terms of x, y, z separately the above equation can be written as

$$x(u, t) = U \cdot M_H \cdot G_{Hx} \cdot M_H^T \cdot U^T$$
$$y(u, t) = U \cdot M_H \cdot G_{Hy} \cdot M_H^T \cdot U^T$$
$$z(u, t) = U \cdot M_H \cdot G_{Hz} \cdot M_H^T \cdot U^T \quad \ldots (9.19)$$

9.6.2 B-Spline Surfaces

Applying similar procedure as that of Hermite surface we can represent B-spline surface as

$$x(u, t) = U \cdot M_{BS} \cdot G_{BSx} \cdot M_{BS}^T \cdot U^T$$
$$y(u, t) = U \cdot M_{BS} \cdot G_{BSy} \cdot M_{BS}^T \cdot U^T$$
$$z(u, t) = U \cdot M_{BS} \cdot G_{BSz} \cdot M_{BS}^T \cdot U^T \quad \ldots (9.20)$$

9.6.3 Bezier Surface

Applying similar procedure as that of Hermite surface we can represent Bezier surface as

$$x(u, t) = U \cdot M_B \cdot G_{Bx} \cdot M_B^T \cdot U^T$$
$$y(u, t) = U \cdot M_B \cdot G_{By} \cdot M_B^T \cdot U^T$$
$$z(u, t) = U \cdot M_B \cdot G_{Bz} \cdot M_B^T \cdot U^T \quad \ldots (9.21)$$

Review Questions

1. Explain the true curve generation algorithm.
2. List the problems in true curve generation algorithm.
3. What is interpolation ? Explain Lagrangian interpolation method.
4. What is spline ?
5. Differentiate between interpolation spline and approximation spline.
6. Give the various methods for specifying spline curve.
7. Why to use cubic polynominals ?
8. Write a short note on B-spline curve.
9. List the properties of B-spline curve.
10. Write a short note on Bezier curve.
11. Explain the properties of Bezier curve.

University Questions

1. Write detailed note on cubic B-splines **(Dec-96, May-97, May-2001)**
2. What do you understand by cubic B-splines? Discuss with suitable mathematical models. **(Dec-97)**

10 Light Shading

10.1 Introduction

So far we have seen how to construct three-dimensional objects, parallel and perspective projections of the objects, and removal of hidden surfaces and lines. In this chapter, we will see the shading of the three-dimensional objects and its model. The shading model is also called **illumination model** or **lighting model**. This model is used to calculate the intensity of light that we should see at a given point on the surface of an object.

Later part of this chapter gives the information about the colour models

10.2 Diffuse Illumination

An objects illumination is as important as its surface properties in computing its intensity. The object may be illuminated by light which does not come from any particular source but which comes from all directions. When such illumination is uniform from all directions, the illumination is called **diffuse illumination**. Usually, diffuse illumination is a background light which is reflected from walls, floor, and ceiling.

When we assume that going up, down, right and left is of same amount then we can say that the reflections are constant over each surface of the object and they are independent of the viewing direction. Such a reflection is called **diffuse reflection**. In practice, when object is illuminated, some part of light energy is absorbed by the surface of the object, while the rest is reflected. The ratio of the light reflected from the surface to the total incoming light to the surface is called **coefficient of reflection** or the **reflectivity**. It is denoted by R. The value of R varies from 0 to 1. It is closer to 1 for white surface and closer to 0 for black surface. This is because white surface reflects nearly all incident light whereas black surface absorbs most of the incident light. Reflection coefficient for gray shades is in between 0 to 1. In case of colour object reflection coefficient are various for different colour surfaces.

Lambert's Law

We have seen that, the diffuse reflections from the surface are scattered with equal intensity in all directions, independent of the viewing direction. Such surfaces are sometimes referred to as ideal diffuse reflectors. They are also called Lambertian reflector, since radiated light energy from any point on the surface is governed by **Lambert's cosine law**. This law states that the reflection of light from a perfectly diffusing surface varies as the

cosine of the angle between the normal to the surface and the direction of the reflected ray. This is illustrated in Fig. 10.1.

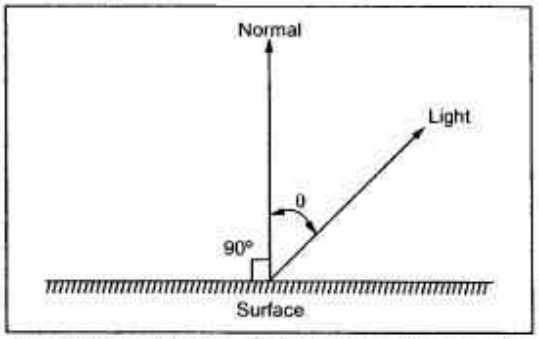


Fig. 10.1 The direction of light is measured from the surface normal

Thus if the incident light from the source is perpendicular to the surface at a perpendicular point, that point is fully illuminated. On the otherhand, as the angle of illumination moves away from the surface normal, the brightness of the point drops off; but the points appear to be squeezed closer together, and the net effect is that the brightness of the surface is unchanged. This is illustrated in Fig. 10.2. In other words we can say that the reduction in brightness due to cosine of angle gets cancelled by increase in the number of light-emitting points within the area of view.

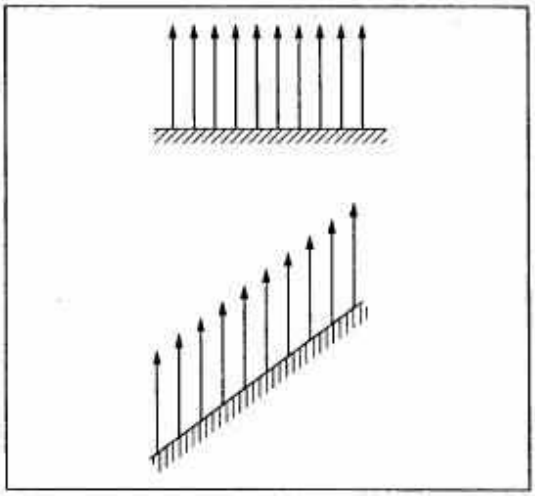


Fig. 10.2 Surface brightness

A similar cancellation effect can be observed as the surface is moved farther from the view point. As we move farther from the view port, the light coming from the surface spreads over a large area. This area increases by the square of distance, thus the amount of light reaching the eye decreases by the same factor. This factor is compensated by the size of the object. When object is moved farther from the viewport, it appears smaller. Therefore, eventhough there is less light, it is applied to a smaller area on the retina and hence the brightness of the surface remains unchanged.

The expression for the brightness of an object illuminated by diffuse ambient or background light can be given as

$$I_{ambdiff} = k_a I_a$$

where I_a is the intensity of the ambient light or background light, k_a is the ambient reflection coefficient and $I_{ambdiff}$ is the intensity of diffuse reflection at any point on the surface which is exposed only to ambient light. Using above equation it is possible, to create light or dark scenes or gray shaded objects. But in this simple model, every plane on a particular object will be shaded equally. The real shaded object does not look like this. For more realistic shading model we also have to consider the point sources of illumination.

10.3 Point-Source Illumination

Point sources emits rays from a single point and they can approximate real world sources such as a small incandescent bulbs or candles. A point source is a direction source, whose all the rays come from the same direction, therefore, it can be used to represent the distant sun by approximating it as an infinitely distant point source.

The modelling of point sources requires additional work because their effect depends on the surface's orientation. If the surface is normal (perpendicular) to the incident light rays, it is brightly illuminated. The surfaces turned away from the light source (oblique surfaces) are less brightly illuminated. This is illustrated in Fig. 10.3.

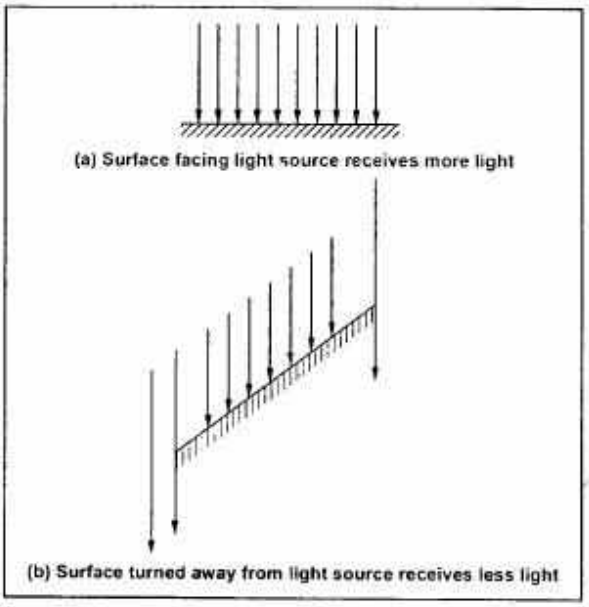


Fig. 10.3

For oblique surfaces, the illumination decreases by a factor of cos I, where I is the angle between the direction of the light and the direction normal to the surface plane. The angle I is know as angle of incidence. (See Fig. 10.4)

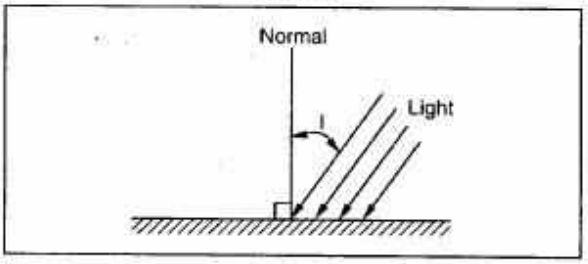


Fig. 10.4 The angle of incidence

The factor cos I is given as

$$\cos I = N \cdot L$$

where L is the vector of length 1 units pointing towards the light source and N is the vector of length 1 in the direction normal to the surface plane.

Considering both diffuse illumination and point source illumination, the shade of the visible surface of an object is given as

$$\begin{aligned} I_{ldiff} &= k_a I_a + k_d I_l (\cos I) \\ &= k_a I_a + k_d I_l (N \cdot L) \end{aligned}$$

where $k_a I_a$ is the intensity of light coming from visible surface due to diffuse illumination,

I_l is the intensity of light comes from the point source, k_d is the diffuse reflectivity coefficient and vector dot product (L.N) gives the cosine of the angle of incidence.

10.4 Specular Reflection

When we illuminate a shiny surface such as polished metal or an apple with a bright light, we observe highlight or bright spot on the shiny surface. This phenomenon of reflection of incident light in a concentrated region around the specular reflection angle is called **specular reflection**. Due to specular reflection, at the highlight, the surface appears to be not in its original colour, but white, the colour of incident light.

The Fig. 10.5 shows the specular reflection direction at a point on the illuminated surface. The specular reflection angle equals the angle of the incident light, with the two angles measured on opposite sides of the unit normal surface vector N. As shown in the Fig. 10.5, R is the unit vector in the direction of ideal specular reflection, L is the unit vector directed toward the point light source and V is the unit vector pointing to the viewer from the surface position.

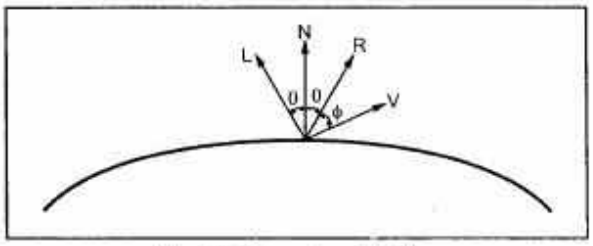


Fig. 10.5 Specular reflection

The angle ϕ between vector R and vector V is called **viewing angle**. For an ideal reflector (perfect mirror), incident light is reflected only in the specular reflection direction. In such case, we can see reflected light only when vector V and R coincide, i.e., $\phi = 0$.

10.4.1 The Phong Illumination Model

Phong Bui-Tuong developed a popular illumination model for nonperfect reflectors. It assumes that maximum specular reflection occurs when ϕ is zero and falls off sharply as ϕ increases. This rapid fall-off is approximated by $\cos^n \phi$, where n is the specular reflection parameter determined by the type of surface. The values of n typically vary from 1 to several hundred, depending on the surface material. The larger values (say, 100 or more) of n are used for very shiny surface and smaller values are used for dull surfaces. For a perfect reflector, n is infinite. For rough surface, such as chalk, n would be near to 1. Fig. 10.6 and Fig 10.7 show the effect of n on the angular range of specular reflection.

Fig. 10.6 Effect of n on the angular range of specular reflection

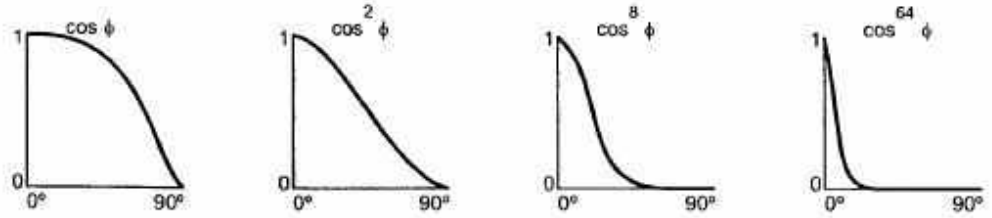


Fig. 10.7 Different values of $\cos^n \phi$ used in the Phong illumination model

The amount of incident light specularly reflected depends on the angle of incidence θ, material properties of surface, polarization and colour of the incident light. The model is approximated for monochromatic specular intensity variations using a specular-reflection coefficient, W(θ), for each surface. We can write the equation for Phong specular reflection model as

$$I_{spec} = W(\theta) I_l \cos^n \phi$$

where I_l is the intensity of the light source and ϕ is the angle between viewing vector and specular reflection vector R.

W(θ) is typically set to a constant k_s, the material's specular-reflection coefficient, which ranges from between 0 to 1. The value of k_s is selected experimentally to produce aesthetically pleasing results. Note that V and R are the unit vectors in the viewing and specular-reflection directions, respectively. Therefore, we can calculate the value of cos ϕ with the dot product V · R. Considering above changes we can rewrite the equation for intensity of the specular reflection as

$$I_{spec} = k_s I_l (V \cdot R)^n$$

The vector R in the above equation can be calculated in terms of vector L and N. This calculation requires mirroring L about N. As shown in Fig. 10.8, this can be accomplished with some simple geometry. Since N and L are normalized, the projection of L onto N is N cos θ. Note that R = N cos θ + S, where |S| is sin θ. But, by vector subtraction and congruent triangles, S is just N cos θ – L. Therefore,

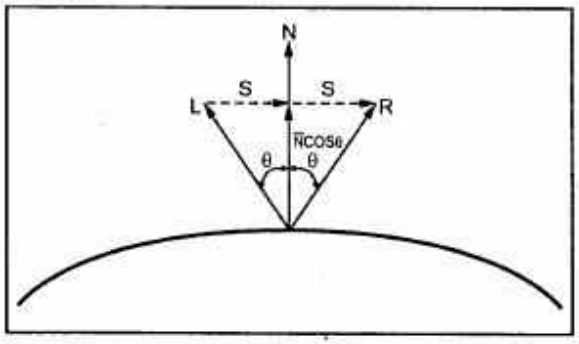


Fig. 10.8 Calculating the reflection vector

$$R = N \cos\theta + N \cos\theta - L$$
$$= 2 N \cos\theta - L$$

Substituting N · L for cos θ we have,

$$R = 2N (N \cdot L) - L$$

10.4.2 The Halfway Vector

More simplified way of formulation of Phong's illumination model is the use of halfway vector H. It is called halfway vector because its direction is halfway between the directions of the light source and the viewer as shown in the Fig. 10.9.

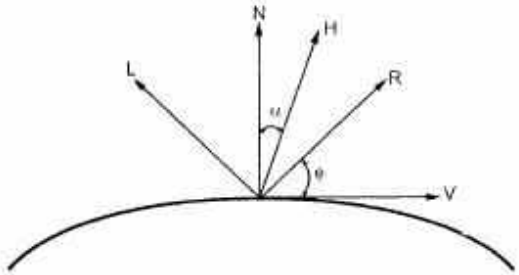


Fig. 10.9 Halfway vector H

If we replace V · R in the Phong model with the dot product N · H, this simply replaces the empirical $\cos\phi$ calculation with the empirical $\cos\alpha$ calculation (Refer Fig. 10.9). The halfway vector is given as

$$H = \frac{L + V}{|L + V|}$$

When the light source and the viewer are both at infinity, then the use of N · H offers a computational advantage, since H is constant for all surface points. Substituting N · H in place of V · R the intensity for specular reflection is given as

$$I_{spec} = k_s I_l (N \cdot H)^n$$

For given light-source and viewer positions, vector H gives the orientation direction for the surface that would produce maximum specular reflection in the viewing direction. Thus, H is also referred to as the surface orientation direction for maximum highlights.

10.5 Combined Diffuse and Specular Reflections

For a single point light source, the combined diffuse and specular reflections from any point on the illuminated surface is given as

$$\begin{aligned} I &= I_{diff} + I_{spec} \\ &= k_a I_a + k_d I_l (N \cdot L) + k_s I_l (N \cdot H)^n \end{aligned}$$

For a multiple point light source the above equation can be modified as

$$I = k_a I_a + \sum_{i=1}^{M} I_{l_i} [k_d (N \cdot L_i) + k_s (N \cdot H_i)^n]$$

Therefore, in case of multiple point light sources the light reflected at any surface point is given by summing the contributions from the individual sources.

hading Algorithms

From the previous discussion it is clear that we can shade any surface by calculating the surface normal at each visible point and applying the desired illumination model at that point. Unfortunately, this shading method is expensive. In this section, we discuss more efficient shading methods for surfaces defined by polygons. Each polygon can be drawn with a single intensity, or with different intensity obtained at each point on the surface. Let us see various shading methods.

10.6.1 Constant-Intensity Shading

The fast and simplest method for shading polygon is constant shading, also known as **faceted shading** or **flat shading**. In this method, illumination model is applied only once for each polygon to determine single intensity value. The entire polygon is then displayed with the single intensity value.

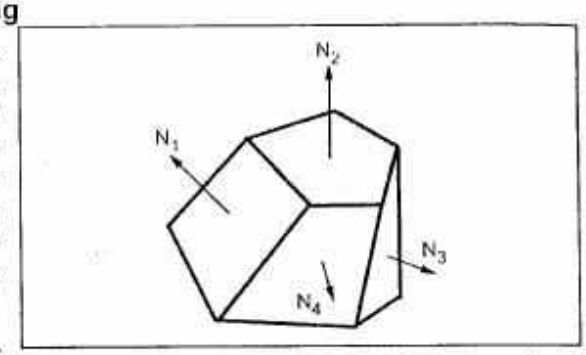


Fig. 10.10 Polygons and their surface normals

This method is valid for the following assumptions :

1. The light source is at infinity, so N · L is constant across the polygon face.
2. The viewer is at infinity, so V · R is constant over the surface.
3. The polygon represents the actual surface being modeled, and is not an approximation to a curved surface.

If either of the first two assumptions are not true still we can use constant intensity shading approach; however, we require some method to determine a single value for each of L and V vectors.

10.6.2 Gouraud Shading

In this method, the intensity interpolation technique developed by Gouraud is used, hence the name. The polygon surface is displayed by linearly interpolating intensity values across the surface. Here, intensity values for each polygon are matched with the values of adjacent polygons along the common edges. This eliminates the intensity discontinuities that can occur in flat shading.

By performing following calculations we can display polygon surface with Gouraud shading.

1. Determine the average unit normal vector at each polygon vertex.
2. Apply an illumination model to each polygon vertex to determine the vertex intensity.
3. Linearly interpolate the vertex intensities over the surface of the polygon.

We can obtain a normal vector at each polygon vertex by averaging the surface normals of all polygons sharing that vertex. This is illustrated in Fig. 10.11.

As shown in the Fig. 10.11, there are three surface normals N_1, N_2 and N_3 of polygon sharing vertex V. Therefore, normal vector at vertex V is given as

$$N_v = \frac{N_1 + N_2 + N_3}{|N_1 + N_2 + N_3|}$$

In general, for any vertex position V, we can obtain the unit vertex normal by equation

$$N_V = \frac{\sum_{i=1}^{n} N_i}{\left|\sum_{i=1}^{n} N_i\right|}$$

where n is the number of surface normals of polygons sharing that vertex.

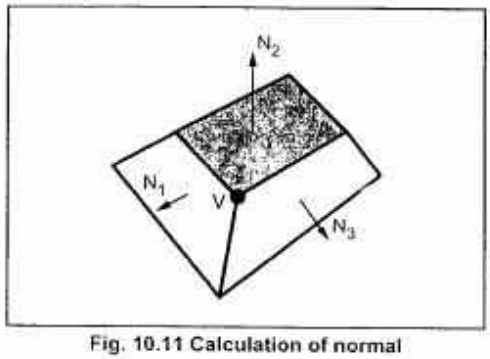


Fig. 10.11 Calculation of normal vector at polygon vertex V

The next step in Gouraud shading is to find vertex intensities. Once we have the vertex normals, their vertex intensities can be determined by applying illumination model to each polygon vertex. Finally, each polygon is shaded by linear interpolating of vertex intensities along each edge and then between edges along each scan line. This is illustrated in Fig. 10.12.

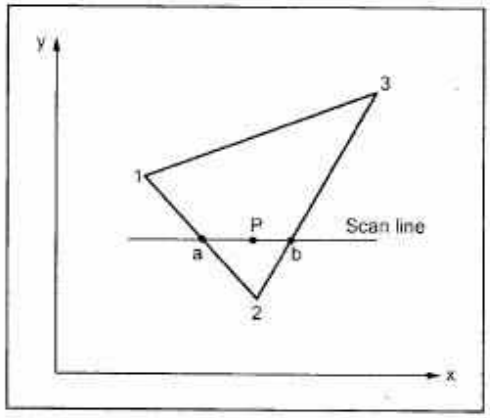


Fig. 10.12

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 example, in Fig. 10.12, the polygon edge with endpoint vertices 1 and 2 is intersected by the scan line at point 'a'. The intensity at point 'a' can be interpolated from intensities I_1 and I_2 as

$$I_a = \frac{y_a - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_a}{y_1 - y_2} I_2$$

Similarly, we can interpolate the intensity value for right intersection (point b) from intensity values I_2 and I_3 as

$$I_b = \frac{y_a - y_2}{y_3 - y_2} I_3 + \frac{y_3 - y_b}{y_3 - y_2} I_2$$

Once the intensities of intersection points a and b are calculated for a scan line, the intensity of an interior point (such as P in Fig. 10.12) can be determined as

$$I_p = \frac{x_b - x_p}{x_b - x_a} I_a + \frac{x_p - x_a}{x_b - x_a} I_b$$

During the scan conversion process, usually incremental calculations are used to obtain the successive edge intensity values between the scan lines and to obtain successive intensity along a scan line. this eliminates the repeatative calculations.

If the intensity at edge position (x, y) is interpolated as

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

then we can obtain the intensity along this edge for the next scan line, y – 1 as (see Fig. 10.13)

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

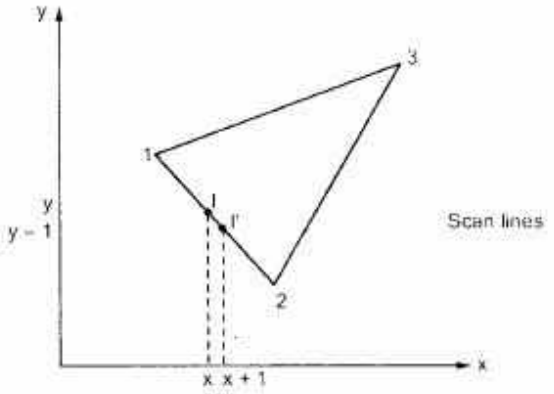


Fig. 10.13 Calculation of incremental interpolation of intensity values along a polygon edge for successive scan lines

Similarly, we can obtain intensities at successive horizontal pixel positions along each scan line (see Fig. 10.14) as

$$I' = I + \frac{I_b - I_a}{x_b - x_a}$$

Advantages

1. It removes the intensity discontinuities exists in constant shading model.
2. It can be combined with a hidden surface algorithm to fill in the visible polygons along each scan line.

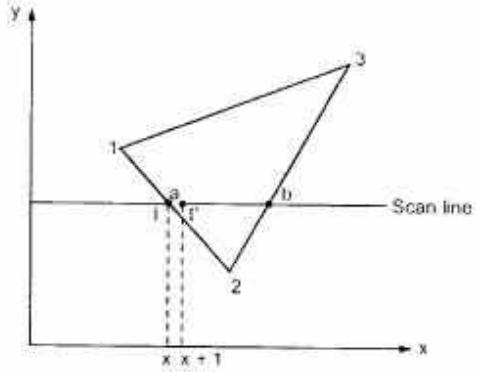


Fig. 10.14 Calculation of incremental interpolation of intensity values along a scan line

Disadvantages

1. Highlights on the surface are sometimes displayed with anomalous shapes.
2. The linear intensity interpolation can result bright or dark intensity streaks to appear on the surface. These bright or dark intensity streaks, are called **Mach bands**. The mach band effect can be reduced by breaking the surface into a greater number of smaller polygons.
3. Sharp drop of intensity values on the polygon surface can not be displayed.

10.6.3 Phong Shading

Phong shading, also known as **normal-vector interpolation shading**, interpolates the surface normal vector N, instead of the intensity. By performing following steps we can display polygon surface using Phong shading.

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

The first steps in the Phong shading is same as first step in the Gouraud shading. In the second step the vertex normals are linearly interpolated over the surface of the polygon. This is illustrated in Fig. 10.15. As shown in the Fig. 10.15, 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 :

$$N = \frac{y - y_2}{y_1 - y_2} N_1 + \frac{y_1 - y}{y_1 - y_2} N_2$$

Like, Gouraud shading, here also we can use incremental methods to evaluate normals between scan lines and along each individual scan line. Once the surface normals are evaluated the surface intensity at that point is determined by applying the illumination model.

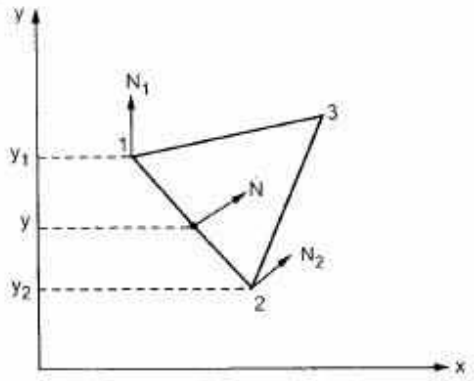


Fig. 10.15 Calculation of interpolation of surface normals along a polygon edge

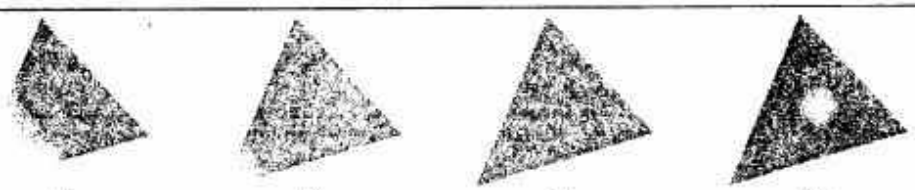
Advantages

1. It displays more realistic highlights on a surface. (See Fig. 10.16 d)
2. It greatly reduces the Mach-band effect.
3. It gives more accurate results.

Disadvantage

1. It requires more calculations and greatly increases the cost of shading steeply.

Fig. 10.16 shows the improvement in display of polygon surface using Phong shading over Gouraud shading.



(a) Gouraud shading (b) Phong shading (c) Gouraud shading (d) Phong shading

Fig. 10.16

Method of Speeding Up Phong Shading Technique

Phong shading is applied by determining the average unit normal vector at each polygon vertex and then linearly interpolating the vertex normals over the surface of the polygon. Then apply an illumination model along each scan line to calculate projected pixel intensities for the surface points. Phong shading can be speeded up by 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

$$N = A_x + B_y + C$$

where A, B, C are determined from three vertex equations

$N_k = Ax_k + By_k + C \quad k = 1, 2, 3$ with (x_k, y_k) denoting a vertex positions. Discarding the reflectivity and attenuation parameters, the calculations for light source diffuse reflection from a surface point (x, y) as

$$I_{diff}(x, y) = \frac{L \cdot N}{|L|\ |N|} = \frac{L \cdot (A_s + B_y + C)}{|L|\ |A_s + B_y + C|}$$
$$= \frac{(L \cdot A)\, x + (L \cdot B)\, y + L \cdot C}{|L|\ |A_s + B_y + C|}$$

Now the expression can be rewritten in the form as

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

Where parameters a, b, c and d are used to represent the various dot products. We can express the denominator as a Taylor - series expansion and retain terms up to second degree in x and y.

$$I_{diff}(x, y) = T_5 x^2 + T_4 x y + T_3 y^2 + T_2 x + T_1 y + T_0$$

where each T_k is a functions of parameter a, b, c and so forth.

Using forward differences, we can evaluate above equation with only two additions for each pixel position (x, y) once the initial forward difference parameters have been evaluated. Thus the fast phong shading technique reduces the calculations and speed up the process.

10.6.4 Halftone Shading

Many displays and hardcopy devices are bilevel. They can only produce two intensity levels. In such displays or hardcopy devices we can create an apparent increase in the number of available intensities. This is achieved by incorporating multiple pixels positions into the display of each intensity value. When we view a very small area from a sufficiently large viewing distance, our eyes average fine details within the small area and record only the overall intensity of the area. This phenomenon of apparent increase in the number of available intensities by considering combine intensity of multiple pixels is known as **halftoning**. The halftoning is commonly used in printing black and white photographs in newspapers, magazines and books. The pictures produced by halftoning process are called **halftones**.

In computer graphics, halftone reproductions are approximated using rectangular pixel regions, say 2×2 pixels or 3×3 pixels. These regions are called halftone patterns or pixel patterns. Fig. 10.17 shown the halftone patterns to create number of intensity levels.

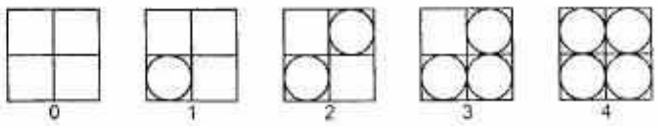


Fig. 10.17 (a) 2 × 2 Pixel patterns for creating five intensity levels

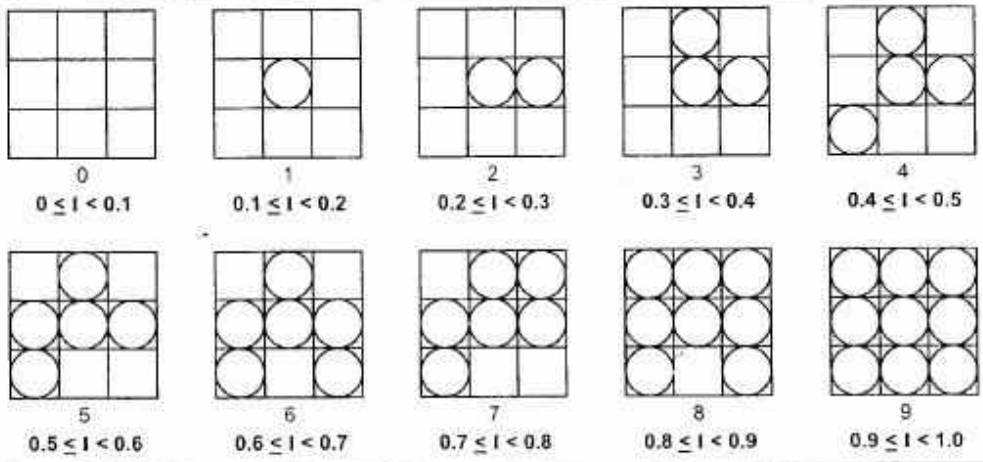


Fig. 10.17 (b) 3 × 3 Pixel patterns for creating ten intensity levels

10.6.5 Dithering Techniques

Dithering refers to techniques for approximating halftones without reducing resolution, as pixel grid patterns do. The term dithering is also applied to halftone approximation methods using pixel grids, and sometimes it is used to refer to colour halftone approximations only.

Random values added to pixel intensities to break up contours are often referred as **dither noise**. Number of methods are used to generate intensity variations. Ordered dither methods generate intensity variations with a one-to-one mapping of points in a scene to the display pixels. To obtain n^2 intensity levels, it is necessary to set up an $n \times n$ dither matrix D_n whose elements are district positive integers in the range of 0 to $n^2 - 1$. For e.g. it is possible to generate four intensity levels with

$$D_2 = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$$ and it is possible to generate nine intensity levels with

$$D_3 = \begin{bmatrix} 7 & 2 & 6 \\ 4 & 0 & 1 \\ 3 & 8 & 5 \end{bmatrix}$$

The matrix elements for D_2 and D_3 are in the same order as the pixel mask for setting up 2×2 and 3×3 pixel grids respectively. For bilevel system, we have to determine display intensity values by comparing input intensities to the matrix elements. Each input intensity is first scaled to the range $0 \le I \le n^2$. If the intensity I is to be applied to screen position (x, y), we have to calculate row and column numbers for the either matrix as

$$i = (x \bmod n) + 1, \qquad j = (y \bmod n) + 1$$

If $I > D_n(i, j)$ the pixel at position (x, y) is turned on; otherwise the pixel is not turned on. Typically, the number of intensity levels is taken to be a multiple of 2. High order dither matrices can be obtained from lower order matrices with the recurrence relation.

$$D_n = \begin{bmatrix} 4\,D_{n/2} + D_2(1,1)\,u_{n/2} & 4\,D_{n/2} + D_2(1,2)\,u_{n/2} \\ 4\,D_{n/2} + D_2(2,1)\,u_{n/2} & 4\,D_{n/2} + D_2(2,2)\,u_{n/2} \end{bmatrix}$$

assuming $n \geq 4$. Parameter $u_{n/2}$ is the unity matrix.

Another method for mapping a picture with $m \times n$ points to a display area with $m \times n$ pixels is **error diffusion**. Here, the error between an input intensity value and the displayed pixel intensity level at a given position is dispersed, or diffused to pixel positions to the right and below the current pixel position.

10.7 Transparency

In the shading models we have not considered the transparent objects. A transparent surface, in general, produces both reflected and transmitted light. It has a transparency coefficient T as well as values for reflectivity and specular reflection. The coefficient of transparency depends on the thickness of the object because the transmission of light depends exponentially on the distance which the light ray must travel within the object. The expression for coefficient of transparency is given as

$$T = te^{-ad}$$

Where t is the coefficient of property of material which determines how much of the light is transmitted at the surface instead of reflected, a is the coefficient of property of material which tells how quickly the material absorbs or attenuates the light, d is the distance the light must travel in the object.

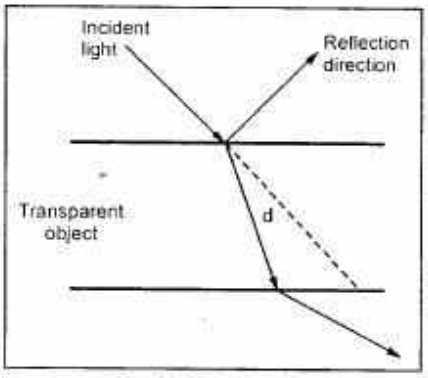


Fig. 10.18 Refraction

When light crosses the boundary between two media it changes the direction as shown in the Fig. 10.18. This effect is called **refraction**. The effect of refraction is observed because the speed of light is different in different materials resulting different path for refracted light from that of incident light. The direction of the refracted light is specified by the **angle of refraction** (θ_r). It is the function of the property materials called the **index of refraction** (n). The angle of refraction θ_r is calculated from the angle of incidence θ_i, the index of refraction n_i of the incident material (usually air), and the index of refraction n_r of the refracting material according to Snell's law :

$$\sin\theta_r = \frac{n_i}{n_r}\sin\theta_i$$

In practice, the index of refraction of a material is a function of the wave length of the incident light, so that the different colour components of a light ray refracts at different angles. The transparency and absorption coefficients are also depend on colour. Therefore, when we are dealing with colour objects we require three pairs of transparency and absorption coefficients.

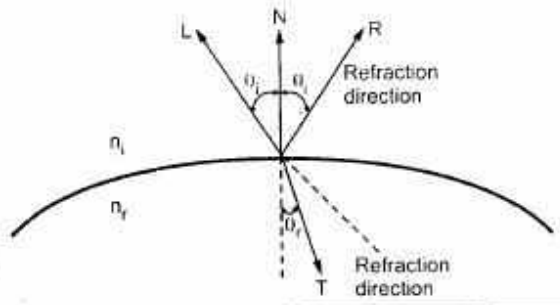


Fig. 10.19 Refraction direction and angle of refraction θ_i

For modeling of transparent surface we have to consider contributions from the light reflected from the surface and the light coming from behind the object. If we assume for a given surface that

- The transparency coefficient for the object is a constant
- Refraction effects are negligible and
- No light source can be seen directly through the object,

then the light coming through the object is given as,

$$v = v_r + t\, v_t$$

where v is the total amount of light,

v_r is the amount of light reflected from the surface,

t is the transparency coefficient and,

v_t is the light coming from behind the object.

To get more realistic images we have to consider the angular behavior of the reflection v_r, transmission at the surface and also the attenuation due to thickness. The simple approximation for this behavior can be given as

$$t = (t_{max} - t_{min})(N \cdot E)^{\alpha} + t_{min}$$

where $(N \cdot E)^{\alpha}$ is the cosine of the angle between the eye and the surface normal raised to the power. This angle decides the distance the light must travel through the object. When viewed straight on, angle is 0 i.e. cosine of angle is 1 (highest) and the distance travelled by light is minimum. When viewed at a glancing angle, cosine is less than 1 and the distance travelled by light is more. Therefore, we can say that cosine of angle is maximum when surface is viewed straight on and it drops off for glancing views. The power of angle represented by α enhances the effect. The values of α of 2 or 3 give reasonable effects.

10.8 Shadows

A shadowed object is one which is hidden from the light source. It is possible to use hidden surface algorithms to locate the areas where light sources produce shadows. In order to achieve this we have to repeat the hidden-surface calculation using light source as the viewpoint. This calculation divides the surfaces into shadowed and unshadowed groups. The surfaces that are visible from the light source are not in shadow; those that are not visible from the light source are in shadow. Surfaces which are visible and which are also

visible from the light source are shown with both the background illumination and the light-source illumination. Surfaces which are visible but which are hidden from the light source are displayed with only the background illumination, as shown in the Fig. 10.20.

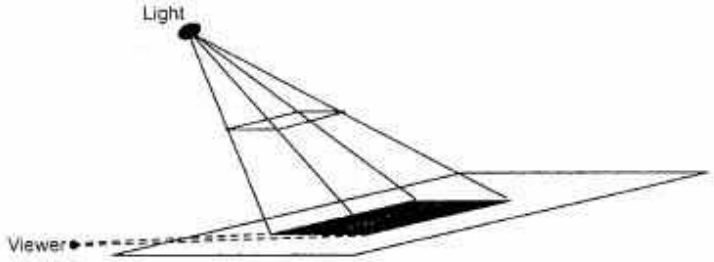


Fig. 10.20 Shadow

Another way to locate shadow areas is the use of **shadow volumes**. A shadow volume is defined by the light source and an object and is bounded by a set of invisible shadow polygons, as shown in the Fig. 10.21. This volume is also known as polygon's shadow volume. By comparing visible polygon with this volume we can identify the portions which lie inside of the volume and which are outside of the volume. The portions which lie inside of the volume are shadowed, and their intensity calculations do not include a term from the light source. the polygons or portions of polygons which lie outside the shadow volume are not shaded by this polygon, but might be shaded by some other polygon so they still must be checked against the other shadow volume.

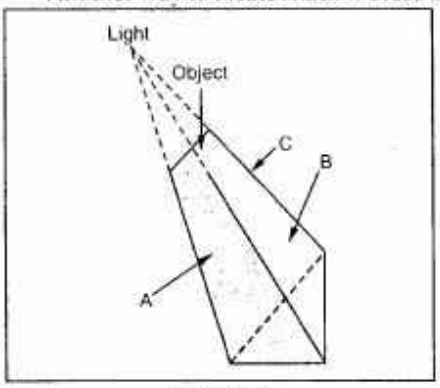


Fig. 10.21

10.9 Ray-Tracing

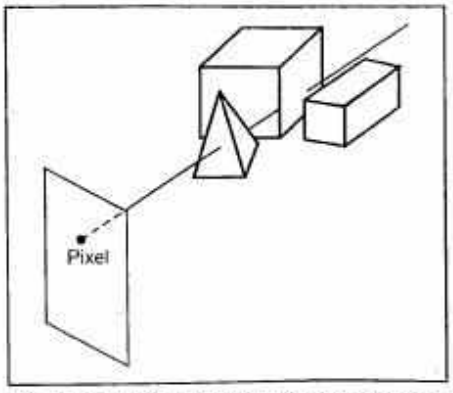


Fig. 10.22 A ray along the line of sight from a pixel position through a scene

If we consider the line of sight from a pixel position on the view plane through a scene, as in Fig. 10.22, we can determine which objects in the scene (if any) intersect this line. From the intersection points with different object, we can identify the visible surface as the one whose intersection point is closest to the pixel. **Ray tracing** is an extension of this basic idea. Here, instead of identifying for the visible surface for each pixel, we continue to bounce the ray around the picture. This is illustrated in Fig. 10.23. When the ray is bouncing from one surface to another surface, it contributes the

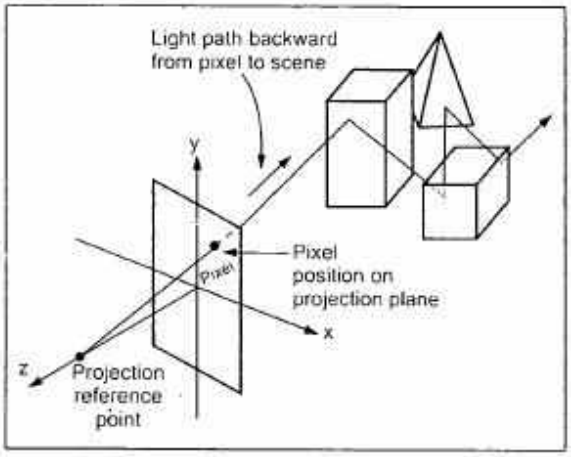


Fig. 10.23 Bouncing of ray around the scene

intensity for that surfaces. This is a simple and powerful rendering technique for obtaining global reflection and transmission effects.

As shown in the Fig. 10.23, usually pixel positions are designated in the xy plane and projection reference point lie on the z axis, i.e. the pixel screen area is centered on viewing coordinate origin. With this coordinate system the contributions to a pixel is determined by tracing a light path backward from the pixel to the picture.

For each pixel ray, each surface is tested in the picture to determine if it is intersected by the ray. If surface is intersected, the distance from the pixel to the surface intersection point is calculated. The smallest calculated intersection distance identifies the visible surface for that pixel. Once the visible surface is identified the ray is reflected off the visible surface along a specular path where the angle reflection equals angle of incidence. If the surface is transparent, the ray is passed through the surface in the refraction direction. The ray reflected from the visible surface or passed through the transparent surface in the refraction direction is called **secondary ray**. The ray after reflection or refraction strikes another visible surface. This process is repeated recursively to produce the next generations of reflection and refraction paths. These paths are represented by **ray tracing tree** as shown in the Fig. 10.24.

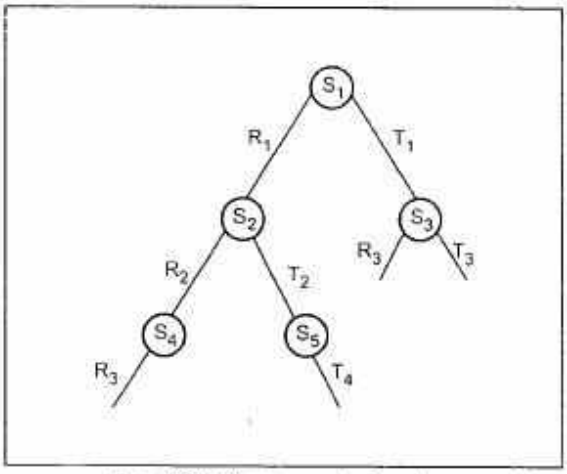


Fig. 10.24 Binary ray-tracing tree

As shown in the Fig. 10.24, the left branches in the binary ray tracing tree are used to represent reflection paths, and right branches are used to represent transmission paths. The recursion depth for ray tracing tree is determined by the amount of storage available, or by the user. The ray path is terminated when predetermined depth is reached or if ray strikes a light source. As we go from top to bottom of the tree, surface intensities are attenuated by distance from the parent surface. The surface intensities of all the nodes are added traversing the ray tree from bottom to top to determine the intensity of the pixel.

If pixel ray does not intersect to any surface then the intensity value of the background is assigned to the pixel. If a pixel ray intersects a nonreflecting light source, the pixel can be assigned the intensity of the source, although light sources are usually placed beyond the path of the initial rays.

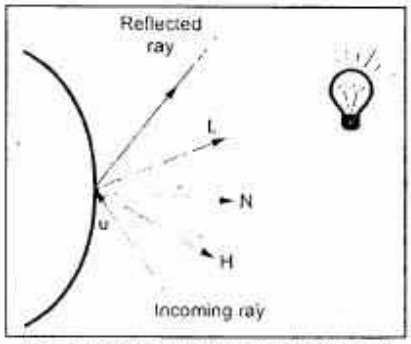


Fig. 10.25 Surface intersected by a ray and the unit vectors

The Fig. 10.25 shows a surface intersected by a ray and unit vectors needed for the reflected light-intensity calculations. Here, u is the unit vector in the direction of the ray path, N is the unit surface normal, R is the unit reflection vector, L is the unit vector pointing to the light source, and H is the unit vector halfway between V (viewer) and L (light source).

If any object intersects the path along L between the surface and the point light source, the surface is in shadow with respect to that source. Hence a path along L is referred to as **shadow ray**. Ambient light at the surface is given as $K_a I_a$, diffuse reflection due to the surface is proportional to K_d (N.L), and the specular-reflection component is proportional to K_s $(H.N)^{n_s}$. We know that, the specular reflection direction for R depends on the surface normal and the incoming ray direction. It is given as

$$R = u - (2u \,.\, N)\, N$$

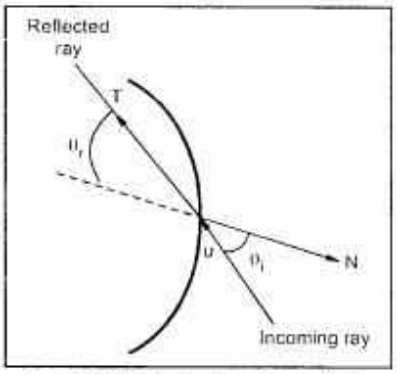


Fig. 10.26 Refracted ray through the transparent material

In a transparent material light passes through the material and we have to calculate intensity contributions from light transmitted through the material. Referring the Fig. 10.26, we can obtain the unit transmission vector from vectors u and N as

$$T = \frac{\eta_i}{\eta_r} u - (\cos\theta_r - \frac{\eta_i}{\eta_r}\cos\theta_i) N$$

where η_i and η_r are the indices of relfection in the incident material and the refracting meterial, respectively. The angle of refraction θ_r is given by Snell's law

$$\cos\theta_r = \sqrt{1 - \left(\frac{\eta_i}{\eta_r}\right)^2 (1 - \cos^2\theta_i)}$$

10.9.1 Ray Surface Intersection Calculations

The ray equation is given as

$$P = P_0 + su$$

Where P_0 is the initial position of ray, P is any point along the ray path at distance s from P_0 and u is the unit direction vector. The ray equation gives the coordinates of any point P along the ray path at a distance s from P_0. Initially, P_0 is set to the position of the pixel on the projection plane, or it is chosen as a projection reference point. Unit vector u is initially obtained from the position of the pixel through which the ray passes and the projection reference point

$$u = \frac{P_{pixel} - P_{prp}}{|P_{pixel} - P_{prp}|}$$

Vectors P_0 and u are updated for the secondary rays at the ray-surface intersection point at each intersected surface. For the secondary rays, reflection direction for u is R and the transmission direction is T. We can locate the surface intersections by simultaneously solving the ray equation and the surface equation for the individual objects in the scene.

The simplest object to ray trace is sphere, i.e. we can easily identify that whether the ray does intersect the sphere or not; and if it intersects we can easily obtain the surface intersection coordinates from the ray equation. Consider the sphere of radio r and center position P_c, as shown in Fig. 10.27. P is any point on the sphere which satisfies the sphere equation :

$$|P - P_c|^2 - r^2 = 0$$

Substituting the value of P from ray equation we can write above equation as

$$|P_0 + s_u - P_c|^2 - r^2 = 0$$

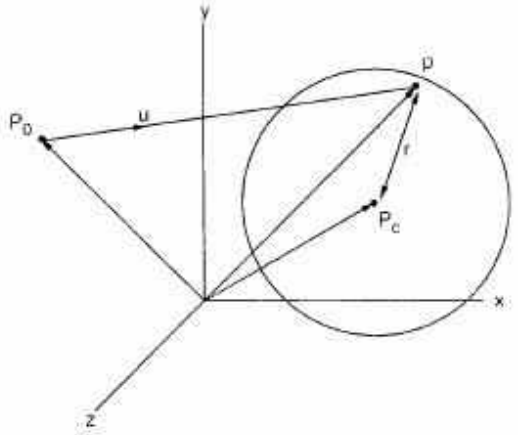


Fig. 10.27 A ray intersecting a sphere having radius r centered on position P_c

If we assume $\Delta P = P_c - P_0$ and expand the dot product, we get the quadratic equation

$$s^2 - 2(u \cdot \Delta P)\, s + (|\Delta P|^2 - r^2) = 0$$

By solving quadratic equation we get,

$$s = u \cdot \Delta P \pm \sqrt{(u \cdot \Delta P)^2 - |\Delta P|^2 + r^2}$$

In the above equation if the discriminant is negative we can say that the ray does not intersect the sphere; otherwise the surface intersection coordinates can be obtained from the ray equation.

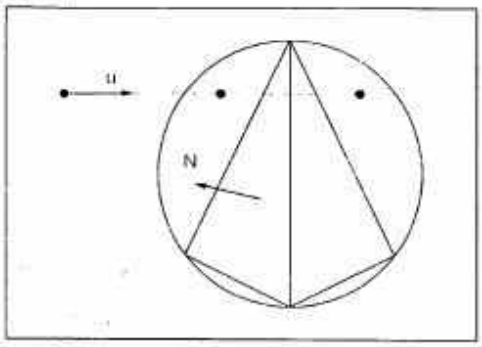


Fig. 10.28 Polyhedron bounded by a sphere

In case of polyhedra more processing is required to locate the surface intersections. For this reason instead of doing intersection calculations directly, first the intersection test is carried out on bounding volume, as shown in Fig. 10.28; and if the ray intersects the bounding volume further tests and intersection calculations are carried out. As shown in Fig. 10.28, the polyhedron is bounded by a sphere. If a ray does not intersect the sphere, we do not need to do any further testing on the polyhedron. But if ray does intersect the sphere, we have to locate front faces with the test

$$u \cdot N < 0$$

Where N is a surface normal. For each face of the polyhedron that satisfies inequality in above equation, we have to solve the plane equation as

$$N \cdot P = -D$$

for surface position P that also satisfies the ray equation. With these initial calculations we can say that the position P is both on the plane and on the ray path if

$$N \cdot (P_0 + Su) = -D$$

and the distance from the initial ray position to the plane is

$$s = -\frac{D + N \cdot P_0}{N \cdot u}$$

The above calculations gives us a position on the infinite plane that contain the polygon face, however they do not satisfy that the position is inside or outside the polygon boundaries. Therefore, to determine whether the ray intersected this face of the polyhedron, we have to perform an inside-outside test discussed in section 3.5.

In case of other objects, such as quadric or spline surfaces we have to follow the same procedure to calculate ray-surface intersection positions.

10.9.2 Reducing Object-Intersection Calculations

When scene contains more than one objects, most of the processing time for each ray is spent in checking objects that are not visible along the ray path. Therefore, as discussed earlier, adjacent objects are enclosed in groups within a bounding volume, such as a sphere or a box. We can then proceed for intersection calculations only when the ray intersects the bounding volume. This approach can be extended to include a hierarchy of bounding volumes. That is, we enclose several bounding volumes within a larger volume and carry out the intersection test hierarchically. In this, we first test the outer bounding volume and then if necessary test the smaller inner bounding volumes and so on.

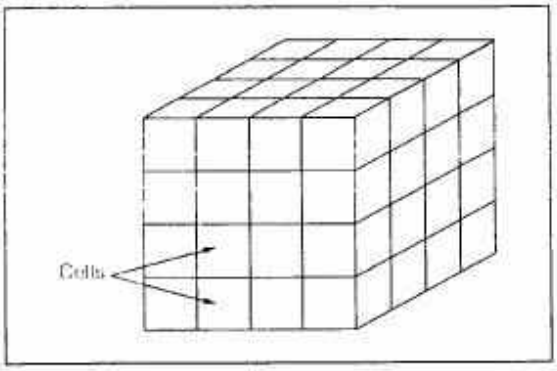


Fig. 10.29 Subdivision of cube into cells

Another method known as **space-subdivision method** is also used to reduce intersection calculations. In this method, the scene is enclosed within a cube and a cube then successively subdivided until each subregion (cell) contains no more than a preset maximum number of surfaces. For example, one surface per cell. We then trace rays through the individual cells of the cube, performing intersection tests only within those cells containing surfaces. There is a trade-off between the cell size and the number of surfaces per cell. If we set the maximum number of surfaces per cell too low, cell size can become too small and cell number can become too large increasing cell-traversal processing.

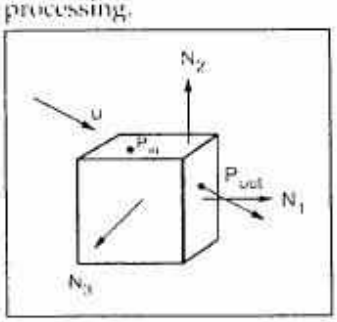


Fig. 10.30 Traversal of ray through a cell

Initially, we have to determine the intersection point on the front face of the cube. It can be determined by checking the intersection coordinates against the cell boundary positions. We then need to process the ray through the cells by determining the entry and exit points as shown in the Fig. 10.30, for each cell traversed by the ray until ray intersect and object surface or exit the cube.

If a ray direction is u and a ray entry point is P_{in} for a cell, the potential exit faces are those for which

$$u \cdot N_k > 0$$

where N_k are the normal vectors. If these vectors are aligned with coordinate axes, then

$$N_k = \begin{cases} (\pm 1, 0, 0) \\ (0, \pm 1, 0) \\ (0, 0, \pm 1) \end{cases}$$

and to determine the three candidate exits plane we have to check only the sign of each component of u. The exit position on each candidate plane can be obtained from the ray equation as

$$P_{out,k} = P_{in} + S_k u$$

where S_k is the distance along the ray from P_{in} to $P_{out,k}$. Substituting the ray equation into the plane equation for each face of the cell we have,

$$N_k \cdot P_{out,k} = -D$$

Now, the ray distance to each candidate exit face can be given as

$$S_k = \frac{-D - N_k \cdot P_{in}}{N_k \cdot u}$$

We have to select smallest S_k. The above calculations are simple when the normal vectors are aligned with the candidate axes. For example, if a candidate normal vector is (0, 1, 0), then for that plane we have

$$S_k = \frac{x_k - x_{in}}{u_y}$$

where $u = (u_x, u_y, u_z)$, and x_k is the value of the right boundary face for the cell.

10.9.3 Antialiased Ray Tracing

Traditional ray tracing systems suffer from aliasing artifacts. The term aliasing in computer graphics is loosely defined. It can mean almost anything unwanted in the rendered image. Typically aliasing is used to describe jagged edges. In the real world things are not quite as perfect as in a computer generated world-edges and boundaries are not as sharp, reflections are not as perfect, and things can be in and out of focus. If a renderer is being used to approximate reality then these things must be taken into account.

Three basic techniques are used to perform antialiased ray tracing. These are super sampling, adaptive sampling and stochastic sampling. In these sampling methods, the pixel is treated as a finite square area instead of a single point.

10.9.3.1 Super Sampling

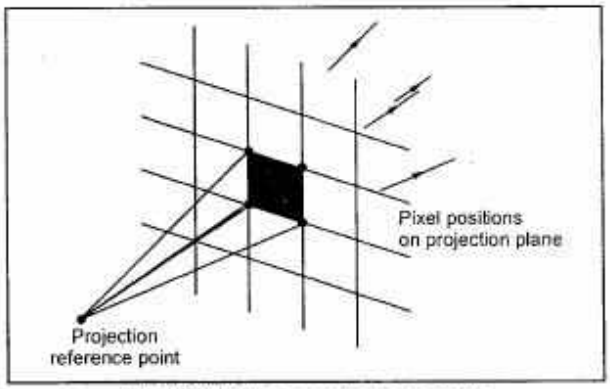


Fig. 10.31 Super sampling procedure

In supersampling, multiple, evenly spaced rays (samples) are taken over each pixel area. The Fig. 10.31 shows a simple supersampling procedure with four rays per pixel, one at each pixel corner. To determine the overall pixel intensity, the intensity of these pixel rays are averaged. However, if the intensities for the four rays are not appropriately equal, or if some small object lies between the four rays, we have to divide the pixel area into subpixels and repeat the process. This is illustrated in Fig. 10.32. Here, pixel area divided into nine subpixels using 16 rays, one at each subpixel corner.

Fig. 10.32 Subdivision of pixel into nine subpixels

10.9.3.2 Adaptive Sampling

In adaptive sampling, multiple, unevenly spaced rays (samples) are taken in some regions of the pixel area. For example, more rays can be taken near object edges to obtain a [illegible] of the pixel intensities. Again, to determine the overall pixel intensity where multiple rays are used, the intensity of rays from subpixels are averaged. However, the subpixels that do not have nearly equal intensity rays are further subdivided until each subpixel has approximately equal intensity rays or an upper bound, say, 256, has been reached for the number of rays per pixel.

10.9.3.3 Stochastic Sampling / Distributed Ray Tracing

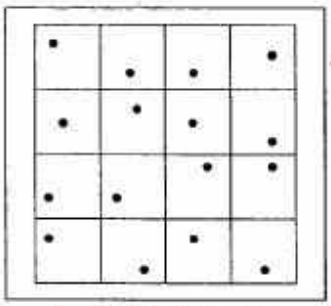


Fig. 10.33 The random distribution of rays

Distributed ray tracing is a stochastic sampling method. It is not ray tracing on a distributed system. It is a ray tracing method based on randomly distributed rays over the pixel area (Refer Fig. 10.33) used to reduce aliasing effect. In this method, the multiple samples are taken and averaged together. The location of where the sample is random so that the resulting average in an approximation of a finite area covered by the samples.

The random distribution of a number of rays over the pixel surface is achieved by the technque called **jittering**. In this technique, initially, pixel area is divided into the 16 subareas as shown in the Fig. 10.33. Then random ray positions are obtained by jittering the center coordinates of each subpixel area by small amounts say δx and δy, where both δx and δy are assigned values in the inverter (– 0.5, 0.5). Therefore, if center position of a cell is specified as (x, y) then the jitter position is $(x + \delta x, y + \delta y)$.

10.9.3.4 Advantages of Distributed Ray Tracing

The intensity of a point in a scene can be represented analytically by an integral over the illumination function and the reflectance function. The evaluation of this integral, while extremely accurate, is too expansive for most graphics applications. Traditional ray tracing makes assumptions about the components of this integral to simplify evaluation. For example, the Phong model assumes that diffuse light is reflected equally in all directions, and specular light is at full intensity in the reflected direction and falls off exponentially with the cosine of the angle away from this direction. In addition, light sources are modeled as single points, so the light that emanates from a source and hits a surface can be represented by a single ray.

Distributed ray tracing uses a slightly better approximation for the illumination and reflectance integrals. The idea is based in the theory of **oversampling**. Instead of approximating an integral by a single scalar value, the function is point sampled and these samples are used to define a more accurate scalar value. The practical benefits of this are :

- Gloss (fuzzy reflections)
- Translucency
- Soft shadows
- Depth of field
- Motion blur

Gloss

Traditional ray tracing is good at representing perfect reflecting surfaces, but poor at representing glossy or partially reflecting surfaces. Only when surfaces are perfect mirrors do the relfections look identical to the scene they are reflecting. More often surfaces are glossy and reflect a blurred image of the scene. This is due to the light scattering properties of the surface. Reflections in traditional ray tracing are always sharp, even partial reflections. Glossy surfaces are generated in distributed ray tracing by randomly distributing rays reflected by a surface. Instead of casting a single ray out in the reflecting direction, a packet of rays are sent out around the reflecting direction. The actual value of reflectance can be found by taking the statistical mean of the values returned by each of these rays.

Translucency

Traditional ray tracing is good at representing perfectly transparent surfaces, but poor at representing translucent surfaces. Real surfaces that are translucent generally transmit a blurred image of the scene behind them. Distirbuted ray tracing achieves this type of translucent surface by casting randomly distributed rays in the general direction of the transmitted ray from traditional ray tracing. The value computed from each of these rays is then averaged to form the true translucent component.

Soft Shadows

Shadows in traditional ray tracing are discrete. When shading a point, each light source is checked to see if it is visible. If the source is visible it has a contribution to the shading of the point, otherwise it does not. The light source itself is modeled by a single point, which is fairly accurate for sources that are a great distance away, but a poor representation for large sources or sources that are close. The result of this discrete decision making is that the edges of shadows are very sharp. There is a distinct transition from when points are visible to the light source to when they are not. Shadows in the real world are much softer. The transition from fully shadowed to partially shadowed is gradual. This is due to the finite area of real light sources, and scattering of light of other surfaces. Distributed ray tracing attempts to approximate soft shadows by modeling light sources as spheres. When a point is tested to see if it is in shadow, a set of rays are cast about the projected area of the light source. The amount of light transmitted from the source to the point can be approximated by the ratio of the number of rays that hit the source to the number of rays cast. This ratio can be used in the standard Phong lighting calculations to scale the amount of light that hits a surface.

Depth of Field

Both the human eye and cameras have a finite lens aperture, and therefore have a finite depth of field. Objects that are two far away or two close will appear unfocused and blurry. Almost all computer graphics rendering techniques use a **pinhole camera** model. In this model all objects are in perfect focus regardless of distance. In many ways this is advantageous, blurring due to lack of focus is often unwanted in images. However,

simulating depth of field can lead to more realistic looking images because it more accurately models true optical systems. Distributed ray tracing creates depth of field by placing an artificial lens in front of the view plane. Randomly distributed rays are used once again to simulate the blurring of depth of field. The first ray cast is not modified by the lens. It is assumed that the focal point of the lens is at a fixed distance along this ray. The rest of the rays sent out for the same pixel will be scattered about the surface of the lens. At the point of the lens they will be bent to pass through the focal point. Points in the scene that are close to the focal point of the lens will be in sharp focus. Points closer or further away will be blurred.

Motion Blur

Animation in computer graphics is produced by generating a sequence of still images and then playing them back in order. This is yet another sampling process, but it is temporal rather than spatial. In movie cameras, each frame represents an average of the scene during the time that the camera shutter is open. If objects in the scene are in motion relative to the camera, then they will appear blurred on the film. Distributed ray tracing can simulate this blurring by distributing rays temporally as well as spatially. Before each ray is cast, objects are translated or rotated to their correct position for that frame. The rays are then averaged afterwards to give the actual value. Objects with the most motion will have the most blurring in the rendered image.

10.10 Colour Models

A colour model is a specification of a 3D colour coordinate system and a visible subset in the coordinate system within which all colours in a particular colour range lie. For example, RGB colour model is the unit cube subset of the 3D Cartesian coordinate system. The colour model allows to give convenient specification of colours in the specific colour range or gamut. There are three hardware oriented colour models : RGB, used for colour CRT monitors, YIQ used for the broadcast TV colour system, and CMY (Cyan, Magenta, Yellow) used for some colour printing devices. However, these models are not easy to use because they does not relate directly to intuitive colour notions of hue, saturation, and brightness. Therefore, another class of colour model has been developed. These include HSV, HLS and HVC models. In this chapter we are going to study RGB, CMY, HSV and HLS models.

10.10.1 Properties of Light

A light source produced by a sun or electric bulb emits all frequencies within the visible range to give white light. When this light is incident upon an object, some frequencies are absorbed and some are reflected by the object. The combination of reflected frequencies decides the colour of the object. If the lower frequencies are predominant in the reflected frequencies, the object colour is red. In this case, we can say that the perceived light has a dominant frequency at the red end of the spectrum. Therefore, the dominant frequency decides the colour of the object. Due to this reason dominant frequency is also called the **hue** or simply the **colour**.

Apart from the frequency there are two more properties which describe various characteristics of light. These are : brightness and saturation (purity). The brightness refers to the intensity of the perceived light. The saturation describes the purity of the colour. Pastels and pale colours are described as less pure or less saturated. When the two

properties purity and dominant frequency are used collectively to describe the colour characteristics, are referred to as **chromaticity**.

We know that two different colour light sources with suitably chosen intensities can be used to produce a range of other colour. But when two colour sources are combined to produce white colour, they are referred to as **complementary colours**. Red and cyan, green and magenta, and blue and yellow are complementary colour pairs. Usually, the colour model use combination of three colours to produce wide range of colours, called the **colour gamut** for that model. The basic colours used to produce colour gamut in particular model are called **primary colours**.

10.10.2 CIE Chromaticity Diagram

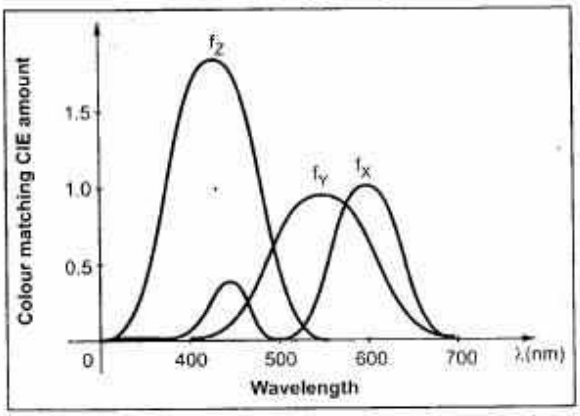


Fig. 10.34 Amounts of CIE primaries needed to display spectral colours

Matching and therefore defining a coloured light with a combination of three fixed primary colours is desirable approach to specify colour. In 1931, the Commission Internationale de l' Eclairage (CIE) defined three standard primaries, called X, Y and Z to replace red, green and blue. Here, X, Y and Z represent vectors in a three-dimensional, additive colour space. The three standard primaries are imaginary colours. They are defined mathematically with positive colour-matching functions, as shown in Fig. 10.34. They specify the amount of each primary needed to describe any spectral colour.

The advantage of using CIE primaries is that they eliminate matching of negative colour values and other problems associated with selecting a set of real primaries.

Any colour (C_λ) using CIE primaries can be expressed as

$$C_\lambda = X\,\mathbf{X} + Y\,\mathbf{Y} + Z\,\mathbf{Z}$$

where X, Y and Z are the amounts of the standard primaries needed to match C_λ and **X**, **Y** and **Z** represent vectors in a three-dimensional, additive colour space.

With above expression we can define chromaticity values by normalizing against luminance (X + Y + Z). The normalizing amounts can be given as

$$x = \frac{X}{X+Y+Z}, \quad y = \frac{X}{X+Y+Z}, \quad Z = \frac{Z}{X+Y+Z}$$

Notice that x + y + z = 1. That is, x, y and z are on the (X + Y + Z = 1) plane. The complete description of colour is typically given with the three values x, y and Y. The remaining values can be calculated as follows :

$$z = 1 - x - y, \qquad X = \frac{x}{y}Y, \qquad Z = \frac{z}{y}Y$$

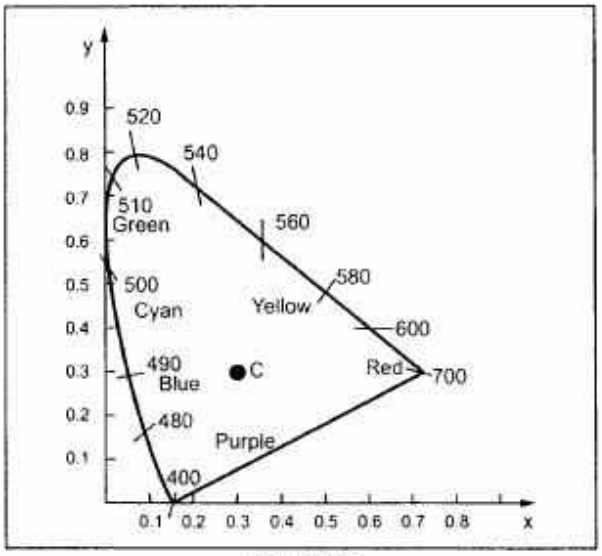


Fig. 10.35

Chromaticity values depend only on dominant wavelength and saturation and are independent of the amount of luminous energy. By plotting x and y for all visible colours, we obtain the CIE chromaticity diagram shown in Fig. 10.35, which is the projection onto the (X, Y) plane of the (X + Y + Z = 1) plane.

The interior and boundary of the tongue-shaped region represent all visible chromaticity values. The points on the boundary are the **pure** colours in the electromagnetic spectrum, labeled according to wavelength in nanometers from the red end to the violet end of the spectrum. A standard white light, is formally defind by a light source illuminant C, marked by the center dot. The line joining the red and violet spectral points is called the purple line, which is not the part of the spectrum.

The CIE chromaticity diagram is useful in many ways :

- It allows us to measure the dominant wavelength and the purity of any colour by matching the colour with a mixture of the three CIE primaries.
- It identifies the complementary colours.
- It allows to define colour gamuts or colour ranges, that show the effect of adding colours together.

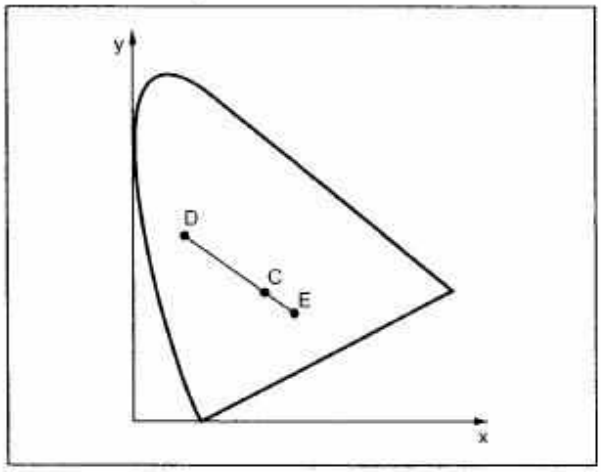


Fig. 10.36 Complementary colours on chromaticity diagram

The Fig. 10.36 represents the complementary colours on the chromaticity diagram. The straight line joining colours represented by points D and E passes through point C (represents white light). This means that when we mix proper amounts of the two colours D and E in Fig. 10.36, we can obtain white light. Therefore, colours D and E are complementary colours, and with point C on the chromaticity diagram we can identify the complement colour of the known colour.