Computer Graphics
Index

I. Content ........................................................................................................ II
II. List of Figures ............................................................................................ VII
III. List of Table ............................................................................................... IX
IV. Abbreviations ............................................................................................. X
V. Application .................................................................................................. 121
VI. Bibliography ............................................................................................... 123
VII. Self Assessment Answers ........................................................................ 126

Book at a Glance
6.7.3.2 Oblique Projection ................................................................. 87
6.7.4 Types of Perspective Projections .......................................... 88
Summary ....................................................................................... 89
References .................................................................................. 89
Recommended Reading ................................................................. 89
Self Assessment ............................................................................ 90

Chapter VII .................................................................................. 92
Hidden Surfaces and Lines ............................................................. 92
Aim .............................................................................................. 92
Objectives ................................................................................... 92
Learning outcome ......................................................................... 92
7.1 Introduction ............................................................................ 93
  7.1.1 Object-Space Method ......................................................... 93
  7.1.2 Image-Space Method ......................................................... 93
7.2 Z-Buffer Algorithm ................................................................. 93
  7.2.1 Advantages ........................................................................ 94
  7.2.2 Disadvantages ................................................................... 94
7.3 Scan Line Algorithm ............................................................... 94
7.4 Warnock’s Algorithm .............................................................. 94
  7.4.1 Advantages ......................................................................... 96
7.5 Hidden Line Methods ............................................................. 96
7.6 Binary Space Partition Trees (BSP) ........................................ 96
  7.6.1 The Pseudo Code for Building a BSP Tree ....................... 97
  7.6.2 The Pseudo Code for Displaying a BSP Tree .................... 98
Summary ....................................................................................... 99
References .................................................................................. 99
Recommended Reading ................................................................. 99
Self Assessment ............................................................................ 100

Chapter VIII .............................................................................. 102
Light, Colour and Shading ............................................................. 102
Aim .............................................................................................. 102
Objectives ................................................................................... 102
Learning outcome ......................................................................... 102
8.1 Introduction ............................................................................ 103
8.2 Diffuse Illumination .............................................................. 103
8.3 Specular Reflection ............................................................... 103
8.4 Shading Algorithms .............................................................. 103
  8.4.1 Constant-Intensity Shading .............................................. 103
8.5 Transparency .......................................................................... 104
8.6 Shadows ................................................................................ 104
8.7 Ray-Tracing ........................................................................... 105
  8.7.1 Gloss ................................................................................ 105
  8.7.2 Translucency .................................................................... 105
  8.7.3 Soft Shadows ..................................................................... 105
  8.7.4 Depth of Field ................................................................. 106
  8.7.5 Motion Blur ...................................................................... 106
8.8 Colour Tables .......................................................................... 106
Summary ....................................................................................... 108
References .................................................................................. 108
Recommended Reading ................................................................. 108
Self Assessment ............................................................................ 109
### List of Figures

<table>
<thead>
<tr>
<th>Fig.</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Applications of computer graphics</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Classification of computer graphics</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Line segment</td>
<td>4</td>
</tr>
<tr>
<td>1.4</td>
<td>Vectors</td>
<td>4</td>
</tr>
<tr>
<td>1.5</td>
<td>Vector generations</td>
<td>5</td>
</tr>
<tr>
<td>1.6</td>
<td>Vector generations</td>
<td>6</td>
</tr>
<tr>
<td>1.7</td>
<td>Bresenham’s line algorithm</td>
<td>8</td>
</tr>
<tr>
<td>1.8</td>
<td>Midpoint circle drawing algorithm</td>
<td>10</td>
</tr>
<tr>
<td>1.9</td>
<td>Ellipse drawing algorithms</td>
<td>12</td>
</tr>
<tr>
<td>1.10</td>
<td>Frame buffer using eight 5-bit shift register</td>
<td>14</td>
</tr>
<tr>
<td>1.11</td>
<td>Frame buffer graphics system</td>
<td>14</td>
</tr>
<tr>
<td>2.1</td>
<td>Cathode ray tube</td>
<td>19</td>
</tr>
<tr>
<td>2.2</td>
<td>Vector scan CRT</td>
<td>20</td>
</tr>
<tr>
<td>2.3</td>
<td>Architecture of a vector display</td>
<td>20</td>
</tr>
<tr>
<td>2.4</td>
<td>Architecture of a raster display</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>Raster scan CRT</td>
<td>21</td>
</tr>
<tr>
<td>2.6</td>
<td>Screen Cartesian reference system</td>
<td>24</td>
</tr>
<tr>
<td>2.7</td>
<td>Polyline drawing</td>
<td>25</td>
</tr>
<tr>
<td>2.8</td>
<td>Polyline with markers</td>
<td>25</td>
</tr>
<tr>
<td>2.9</td>
<td>Polygon</td>
<td>26</td>
</tr>
<tr>
<td>2.10</td>
<td>Rectangle</td>
<td>26</td>
</tr>
<tr>
<td>2.11</td>
<td>Convex polygons</td>
<td>27</td>
</tr>
<tr>
<td>2.12</td>
<td>Concave polygons</td>
<td>27</td>
</tr>
<tr>
<td>2.13</td>
<td>Vector refresh display system</td>
<td>29</td>
</tr>
<tr>
<td>2.14</td>
<td>Display file and interpreter</td>
<td>29</td>
</tr>
<tr>
<td>2.15</td>
<td>Raster scan system with a display processor</td>
<td>30</td>
</tr>
<tr>
<td>3.1</td>
<td>Terminal point</td>
<td>35</td>
</tr>
<tr>
<td>3.2</td>
<td>Convex polygon</td>
<td>35</td>
</tr>
<tr>
<td>3.3</td>
<td>Concave polygon</td>
<td>35</td>
</tr>
<tr>
<td>3.4</td>
<td>Representation of polygon</td>
<td>36</td>
</tr>
<tr>
<td>3.5</td>
<td>Boundary defined regions</td>
<td>37</td>
</tr>
<tr>
<td>3.6</td>
<td>Partial filling resulted using 4-connected algorithm</td>
<td>38</td>
</tr>
<tr>
<td>3.7</td>
<td>Scan line</td>
<td>40</td>
</tr>
<tr>
<td>3.8</td>
<td>Intersection points along the scan line that intersect polygon vertices</td>
<td>40</td>
</tr>
<tr>
<td>3.9</td>
<td>Filling pattern</td>
<td>46</td>
</tr>
<tr>
<td>4.1</td>
<td>Translation</td>
<td>52</td>
</tr>
<tr>
<td>4.2</td>
<td>Rotation of object about the origin</td>
<td>53</td>
</tr>
<tr>
<td>4.3</td>
<td>Scaling</td>
<td>54</td>
</tr>
<tr>
<td>4.4</td>
<td>Rotation about an arbitrary point</td>
<td>57</td>
</tr>
<tr>
<td>5.1</td>
<td>Display images</td>
<td>57</td>
</tr>
<tr>
<td>5.2</td>
<td>Two dimensional viewing transformation pipeline</td>
<td>62</td>
</tr>
<tr>
<td>5.3</td>
<td>Window and viewport</td>
<td>63</td>
</tr>
<tr>
<td>5.4</td>
<td>Picture defined in pixels</td>
<td>63</td>
</tr>
<tr>
<td>5.5</td>
<td>Four-bit codes for nine regions</td>
<td>66</td>
</tr>
<tr>
<td>5.6</td>
<td>Polygon clipping done by line clipping algorithm</td>
<td>70</td>
</tr>
<tr>
<td>6.1</td>
<td>Depth cueing</td>
<td>82</td>
</tr>
<tr>
<td>6.2</td>
<td>3-D translation</td>
<td>84</td>
</tr>
<tr>
<td>6.3</td>
<td>3-D scaling</td>
<td>84</td>
</tr>
<tr>
<td>6.4</td>
<td>Parallel projection of an object to the view plane</td>
<td>86</td>
</tr>
<tr>
<td>6.5</td>
<td>Perspective projection of an object to the view plane</td>
<td>87</td>
</tr>
<tr>
<td>7.1</td>
<td>Surrounding polygon</td>
<td>94</td>
</tr>
<tr>
<td>7.2</td>
<td>Overlapping or intersecting polygon</td>
<td>95</td>
</tr>
<tr>
<td>7.3</td>
<td>Inside or contained polygon</td>
<td>95</td>
</tr>
</tbody>
</table>
Fig. 7.4 Outside or disjoint polygon ............................................................................................................ 95
Fig. 8.1 Colour table .................................................................................................................................. 107
Fig. 9.1 The interpolation process ...........................................................................................................113
Fig. 9.2 Interpolation spline and approximation spline .............................................................................114
Fig. 9.3 Fractal surface ..............................................................................................................................117
List of Table

Table 2.1 Differentiation between vector and raster scan display .......................................................... 22
Abbreviations

BSP  -  Binary Space Partitioning
CAD  -  Computer-Aided Design
CPU  -  Central Processing Unit
CRT  -  Cathode-Ray-Tubes
DDA  -  Digital Differential Analyser
PDCS -  Physical Device Coordinate System
SRGP -  Simple Raster Graphics Package
WCS  -  World Coordinate System
Chapter I
Introduction to Computer Graphics

Aim
The aim of this chapter is to:

- introduce the basics of computer graphics
- explain various computer graphics algorithms
- explicate the advantages of computer graphics

Objectives
The objectives of this chapter are to:

- explain circle and ellipse algorithms
- analyse the basics of computer graphics
- define DDA, midpoint and Bresenham’s algorithms

Learning outcome
At the end of this chapter, you will be able to:

- understand the applications of computer graphics
- describe various algorithms
- identify frame buffer
1.1 Introduction

Computer graphics is generally regarded as a branch of computer science that deals with the theory and technology for computerised image synthesis. It is a tool used for storing, manipulating and correlating data. In ancient times, before man learnt to talk or write, he used drawings to communicate. Later, with the development of language, science and mathematics, drawing was reduced to secondary means of communication, to supplement and to illustrate and soon became a speciality of a draftsmen and a skill of an artist. The medium of advanced computer hardware and software graphics has made it possible to express data in pictorial form. The picture or graphics may be an engineering drawing, business graphs, architectural structures, a single frame from an animated movie or a machine part illustrated for a service manual. It is the fundamental cohesive concept in computer graphics.

In computer graphics, pictures or graphics are presented as a collection of discrete picture elements called pixels. The pixel is the smallest addressable screen element. It is the smallest piece of the display screen we can control. The control is achieved by setting the intensity and colour of the pixel which compose the screen. Each pixel on graphics display represents a region which theoretically contains infinite number of points and not mathematical points. The process of determining the appropriate pixels for representing picture or graphics object is known as rasterisation. The process of representing continuous picture or graphics object as a collection of discrete pixels is called scan conversion. The computer graphics allow rotation, translation, scaling, adding effects and performing various operations on the picture before displaying it. It allows the user to control and modify contents, structure and appearance of pictures.

1.1.1 Advantages of Computer Graphics

Following are the advantages of computer graphics:

- Today, with the advanced and high quality computer graphics, it is possible to interact effectively.
- It provides tools for producing pictures not only of real world objects but also of abstract, synthetic objects, such as 3D mathematical surfaces and also of the data that have no inherent geometry, etc.
- It has the ability to show moving pictures and thus, it is possible to produce animations with computer graphics.
- Graphics enables the user to control the animation by adjusting the speed, the portion of the total scene in the view, the geometry of the objects in the scene, the amount of detailing, the light and colour adjustments, etc.

1.1.2 Applications of Computer Graphics

The computer graphics are widely used in industries, government organisations, business, education, entertainment purpose.

- User interface: It provides an attractive and easy interaction between users and computers. The built-in graphics provided with user interfaces use visual control items such as buttons, menus, icons, scroll bar, etc, which allows the user to interact with computer only by mouse-click. To store, manipulate and input text, typing is necessary.
- Plotting of graphics and charts: Computer graphics is most commonly used to create 2D and 3D graphs of mathematical, physical and economic functions in the histograms, bars and pie-charts, in industries, business, government and educational organisations, etc.
- Computer aided drafting and designs: This application is majorly used by the architects, interior designers, engineers, structural designers etc., which makes possible for them to design various components and systems, such as automobile bodies, structures of building, airplanes, ships, optical systems and computer networks.
- Simulation and animation: Use of graphics in simulation makes mathematical models and mechanical systems more realistic and easy to study. With such advanced technology, production of animated movies and cartoon films has increased.
- Art and commerce: It allows the user to create artistic pictures which express messages and attract attention. Such pictures are useful in advertising.
• Process control: By the use of computer, it is possible to control various processes in the industry from remote control room. In such cases, process systems and processing parameters are shown on the computer with graphic symbols and identification which makes it easy for operator to monitor and control various processing parameters at a time.

• Cartography – computer graphics is used to represent geographic maps, weather maps, oceanographic charts, contour maps, population density maps, etc.

Fig. 1.1 Applications of computer graphics

1.1.3 Classification of Computer Graphics

Fig. 1.2 Classification of computer graphics
1.2 Lines
- Line is the fundamental element of the picture representation. It is nothing but the position in a plane defined as either pairs or triplets of numbers depending on whether the data are two or three dimensional.
- Two points would represent a line or edge, and a collection of three or more points a polygon.
- The representation of curved lines is usually accomplished by approximating them by short line segment.

1.3 Line Segment
When we consider the piece of line, i.e., only two points which lie between two endpoints, then it is called a line segment.

1.4 Vectors
- A vector has a single direction and length.
- Consider the figure below. A vector may be indicated by A and B, where A and B denotes the distance on x and y axes respectively.
- Unlike line segment, vector does not have a fixed position in the space.
- The vector does not tell us the starting point. It gives the user the direction and how far to move.
1.5 Pixels and Frame Buffers

- In raster scan display, a special area of memory is dedicated to graphics only. This memory area is called frame buffer.
- It holds the set of intensity values for all the screen points.
- The stored intensity values are retrieved from frame buffer and displayed on the screen one row at a time.
- Each screen point is referred to as a pixel or pel.
- The user can specify the pixel position on the screen by specifying row and column number.
- Intensity range for pixel positions depends on the capability of the raster system.
- It can be a black and white system or colour system.
- Only one bit per pixel is needed to control the intensity of the pixel position since, each pixel position in black and white system is either on or off.
- When colour and intensity variations are displayed, additional bits are required.
- Upto 24 bits per pixel are included in high quality display system, which can require several megabytes of storage space for frame buffer.
- The frame buffer is called a bitmap on a black and white system with one bit per pixel.
- The frame buffer is referred as a pixmap for systems with multiple bits per pixel.

1.6 Vector Generation

- The process of turning on the pixels for a line segment is called vector generation.
- Desired characters of line are:
  - The line should appear as a straight line and it should start and end accurately.
  - The line should be displayed with constant brightness along its length independence of its length and orientation.
  - The line should be drawn rapidly.
  - In fig. 1.5, we can see that the horizontal and vertical lines are straight and have the same width.

![Vertical line](image1)

![Horizontal line](image2)

Fig. 1.5 Vector generations
In the figure below, we can see that the 45° line is straight but its width is not constant.

**Fig. 1.6 Vector generations**

### 1.7 Vector Generation/Digital Differential Analyser (DDA) Algorithm

- Digital differential analyser (DDA) is the vector generation algorithm which steps along the line to determine the pixel which should be turned on.

Slope of a straight line is:

\[ m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \]

The above differential expression can be used to obtain a rasterised straight line. For any given x interval \( \Delta x \) along a line, we can compute the corresponding y interval \( \Delta y \) from above equation as:

\[ \Delta y = \frac{y_2 - y_1}{x_2 - x_1} \Delta x \]

Similarly, we can obtain the x interval \( \Delta x \) corresponding to a specified \( \Delta y \) as:

\[ \Delta x = \frac{x_2 - x_1}{y_2 - y_1} \Delta y \]

Once the intervals are known, the values for next x and next y on the straight line can be obtained as follows:

\[ x_{i+1} = x_i + \Delta x = x_i + \frac{x_2 - x_1}{y_2 - y_1} \Delta y \]

and

\[ y_{i+1} = y_i + \Delta y \]

\[ y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1} \Delta x \]
Above two equations represent a recursion relation for successive values of x and y along with the required line. Such a way of rasterising a line is called a digital differential analyser (DDA).

1.7.1 ‘C’ Code for DDA Line Drawing Algorithm

```c
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main ( )
{
float x, y, x1, y1, x2, y2, dx, dy, length/
int i, gd, gm/
clrscr ( )/
/* read two end points of line */
printf (“Enter the value of x1 : 	”);
scanf (“%f”, &x1);
printf (“Enter the value of y1 : 	”);
scanf (“%f”, &y1);
printf (“Enter the value of x2 : 	”);
scanf (“%f”, &x2);
printf (“Enter the value of y2 : 	”);
scanf (“%f”, &y2);
/* initialise graphics mode */
detectgraph (&gd, &gm);
initgraph (&gd, &gm, “ “);
dx=abs (x2-x1);
dy=abs (y2-y1);
if (dx ≥ dy)
{
    length = dx;
}
else
{
    length = dy;
}
dx = (x2-x1) / length;
dy = (y2-y1) / length;
x = x1 + 0.5;/* Factor 0.5 is added to round the value */
y = y1 + 0.5;/* Factor 0.5 is added to round the value */
i = ; /* Initialise loop counter */
while (i ≤ length)
{
    putpixel (x, y, 15);
x = x + dx;
y = y + dy;
i = i + 1;
    delay (100); /* Delay is purposely inserted to see */
    Observe the line process */
}
```
1.7.2 Advantages of DDA Algorithm

- It is the simplest algorithm and it does not require special skills for implementation.
- It is a faster method for calculating pixel positions than the direct use of equation $y = mx + b$.
- It eliminates the multiplication in the equation by making use of raster characteristics, so that appropriate increments are applied in the $x$ or $y$ direction to find the pixel positions along with the line path.

1.7.3 Disadvantages of DDA Algorithm

- Floating point arithmetic in DDA algorithm is time-consuming.
- The algorithm is orientation dependent. Hence, end point accuracy is poor.

1.8 Bresenham’s Line Algorithm

- Bresenham’s line algorithm uses only integer addition, subtraction and multiplication by 2.
- Computer performs integer addition and subtraction operations very rapidly.
- The computer is time efficient when performing integer multiplication by power of 2.
- Thus, it is an efficient method for scan-converting straight lines.
- The basic principle of Bresenham’s line algorithm is to select the optimum raster locations to represent a straight line.
- To accomplish this, the algorithm always increments either $x$ or $y$ by one unit depending on the slope of line.
- The increment in the other variable is determined by examining the distance between the actual location and the nearest pixel. This distance is called decision variable or the error.

Consider the figure given below:

![Fig. 1.7 Bresenham’s line algorithm](image-url)
1.8.1 ‘C’ Code for Bresenham’s Line Drawing Algorithm

```c
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main ( )
{
 float x, y, x1, y1, x2, y2, dx, dy, e;
 int i, gd, gm;
 clrscr ( );

 /* Read two end points of line */
 printf (“Enter the value of x1 : 	”);
 scanf (“%f”, &x1);
 printf (“Enter the value of y1 : 	”);
 scanf (“%f”, &y1);
 printf (“Enter the value of x2 : 	”);
 scanf (“%f”, &x2);
 printf (“Enter the value of y2 : 	”);
 scanf (“%f”, &y2);

 /* Initialise graphics mode */
 detectgraph (&gd, &gm);
 initgraph (&gd, &gm, “ “);
 dx=abs (x2-x1);
 dy=abs (y2-y1);

 /* Initialise decision variable */
 e = 2 * dy-dx;
 i = 1; /* Initialise loop counter */

 do
 { putpixel (x, y, 15);
   while (e ≥ 0)
   {
     y =y + 1;
     e = e – 2 * dx;
   }
   x = x + 1;
   e = e + 2 * dy;
   i = i + 1;
 } while (i ≤ dx);
 getch ( );
 closegraph ( );
}
```
1.9 Circle and Ellipse Drawing Algorithms

The different types of drawing algorithms.

1.9.1 Midpoint Circle Drawing Algorithm

- The midpoint circle drawing algorithm uses the eight way symmetry of the circle to generate it.
- It plots 1/8 part of the circle i.e., from 90° as shown in the fig. 1.8.
- As a circle is drawn from 90° to 45°, the x and y move in the positive and negative directions respectively.
- To draw 1/8 part of the circle, we take unit steps in the positive x direction and make use of decision parameters to determine which of the two possible y positions are closer to the circle path at each step.
- The figure 1.8 shows the two possible y positions \((y_i, y_i + 1)\) at sampling position \(x_i + 1\). Therefore, we have to determine whether the pixel at position \((x_i + 1, y_i)\) or at position \((x_i + 1, y_i + 1)\) is closer to the circle. Decision parameter is used for this purpose.
- It uses the circle function \(f_{circle}(x, y) = x^2 + y^2 - r^2\) evaluated at the midpoint between these two pixels.

![Fig. 1.8 Midpoint circle drawing algorithm](image)

‘C’ Code for Midpoint Circle Drawing Algorithm

```c
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main ( )
{
 float p;
 int i, gd, gm, x, y;
 int r;

 /* initialise graphics
 - - - - - - - - - - - - - - - - - - - - - - - - */
 detectgraph )&gd, &gm);
 initgraph (&gd, &gm, " ");
 /* Read the radius
 - - - - - - - - - - - - - - - - - - - - - - - - */
 printf (“Enter the radius of the circle :”);
```
scanf("%d", &r);

x=0;
y=r;
p = 1.25 – r;
do
{
    putpixel (200+x, 200+y, 15);
    putpixel (200+y, 200+x, 15);
    putpixel (200+x, 200 - y, 15);
    putpixel (200+y, 200 - x, 15);
    putpixel (200 – x, 200 - y, 15);
    putpixel (200 – x, 200+y, 15);
    putpixel (200 – y, 200+x, 15);
    putpixel (200 – y, 200 – x, 15);
    if (p < 0)
    {
        x = x+1;
y = y;
p = p+2 * x+1;
    }
    else
    {
        x = x+1;
y = y+1
        p = p+2 (x – y) +1;
    }
delay (10000);
}
while (x < y);
getch ( );
closegraph ( );

1.9.2 Ellipse Drawing Algorithm

- The midpoint ellipse drawing algorithm uses the four way symmetry of the ellipse to generate it.
- Fig. 1.9 shows the division of the first quadrant according to the slope of an ellipse with \( r_x < r_y \).
- As ellipse is drawn from 90° to 0°, the x moves in the positive direction and y moves in the negative direction, and ellipse passes through two regions.
- Like circle function, the ellipse function \( f_{\text{ellipse}(xy)}(r_x^2 x^2 + r_y^2 y^2 - r_x^2 r_y^2) \) serves as the decision parameter in the midpoint algorithm.
- At each sampling position, the next pixel along the ellipse path is selected according to the sign of the ellipse function evaluated at midpoint between the two candidate pixels.
Fig. 1.9 Ellipse drawing algorithms

‘C’ Code for Midpoint Ellipse Drawing Algorithm

```c
#include<stdio.h>
#include<graphics.h>
#include<math.h>

main ( )
{
    long d1, d2;
    int i, gd, gm, x, y;
    long rx, ry, rxsq, tworxsq, tworysq, dx, dy;

    /* Read the radius x and y - - - - - - - - - - - - - - - - - - - - - - - - */
    printf ("Enter the x radius of the ellipse :");
    scanf (%1d", &rx);

    printf ("Enter the y radius of the ellipse :");
    scanf (%1d", &ry);

    /* initialise graphics - - - - - - - - - - - - - - - - - - - - - - - - */
    detectgraph (&gd, gm);
    initgraph (&gd, &gm, " ");

    rxsq = rx * rx;
    rysq = ry * ry;
    tworxsq = 2 * rxsq;
    tworysq = 2 * rysq;
    x = 0;
    y = ry;

    /********* Midpoint Ellipse ********* /
    do {
        int iy = rysq - rxsq;
        int i = tworysq - tworxsq;
        int b = rxsq * rysq;
        int c = rysq * rxsq;

        d1 = iy - c;
        d2 = i + b;

        /* Now we have d1 and d2 for deciding the next point */
        if (d1 > 0) {
            y = y - 1;
            d1 = d1 + rysq;
        } else {
            x = x + 1;
            d2 = d2 + rxsq;
        }

        /* Now we have decided the next point */
        plot (&gd, &gm, x, y);

        /* Now we have decided the next point */
        x = x + 1;
        y = y - 1;
        d1 = d1 + rysq;
        d2 = d2 + rxsq;
    } while (x < y);

    printf ("Close the ellipse.");
    getch ( );
}
```

d1 = rysq – rxsq * ry + (0.25 * rxsq);

dx = tworysq * x;

dy + two * y;
do
{
    putpixel (200 + x, 200 + y, 15);
    putpixel (200 – x, 200 – y, 15);
    putpixel (200 + x, 200 -, 15);
    putpixel (200 – x, 200 + y, 15);

    if )d1 < 0)
    {
        x = x + 1;
        y = y;
        dx = dx + tworysq;
        d1 = d1 + dx + rysq;
    }
    else
    {
        x = x + 1;
        y = y – 1;
        dx = dx + tworysq;
        dy = dy – twoexsq;
        d1 = d1 + dx – dy + rysq;
    }
    delay (10);
}
while (dx < dy);

d2 = rysq* (x + 0.5) * (x + 0.5) + rxsq* (y – 1) * (y-1) – rxsq * rysq;
do
{
    putpixel (200 + x, 200 + y, 15);
    putpixel (200 – x, 200 – y, 15);
    putpixel (200 – x, 200 + y, 15);
    putpixel (200 – x, 200 + y, 15);

    if (d2 > 0)
    {
        x = x;
        y = y – 1;
        dy = dy – tworxsq;
        d2 = d2 – dy + rxsq;
    }
    else
    {
        x = x + 1;
        y = y – 1;
        dy = dy – tworxsq;
        dx = dx + tworusq;
        d2 = d2 + dx – dy + rxsq;
    }
}
while (y > 0);
getch ( );
closegraph ( );
1.10 Display of Frame Buffer

- Frame buffer is implemented using rotating random access semiconductor memory.
- However, frame buffer can also be implemented using shift registers.
- Conceptually, shift register is operated as first-in-first-out fashion i.e., similar to queue.
- We know that, when queue is full and if we want to add new data bit then first data bit is pushed out from the bottom and then the new data bit is added at the top.
- Here, the data ejected out of the queue can be interpreted as the intensity of a pixel on a scan line.

![Frame buffer using eight 5-bit shift register](image)

Fig. 1.10 Frame buffer using eight 5-bit shift register

- The above figure shows the implementation of frame buffer using shift register.
- One shift register is required per pixel on a scan line and the length of shift register in bits is equal to number of scan lines.
- Here, there are 8 pixels per scan line and there are in all 5 scan lines. Therefore, 8 shift registers, each of 5 bit length are used to implement frame buffer.
- The synchronisation between the output of the shift register and the video scan rate is the maintained data corresponding to particular scan line is displayed correctly.
- Both, rotating memory and shift register frame buffer implementations have low levels of interactivity.
- The interactivity in rotating memory is limited due to disk access time and it is reduced in shift register implementations because changes can only be made as buts are being added to the register.

![Frame buffer graphics system](image)

Fig. 1.11 Frame buffer graphics system

- The above figure shows the frame buffer graphics system. It contains CPU, frame buffer, display controller and video monitor.
- An application program running in the computer updates the frame buffer as per the picture information.
- The display controller cycles through the frame buffer in scan line order (top to bottom) and pass the corresponding information to the video monitor to refresh the display.
**Summary**

- Computer graphics is generally regarded as a branch of computer science that deals with the theory and technology for computerised image synthesis.
- The process of determining the appropriate pixels for representing picture or graphics object is known as rasterisation.
- Line is the fundamental element of the picture representation. It is nothing but the position in a plane defined as either pairs or triplets of numbers depending on whether the data are two or three dimensional.
- When we consider the piece of line, i.e., only two points which lie between two endpoints, then it is called a line segment.
- A vector has a single direction and length.
- In raster scan display, a special area of memory is dedicated to graphics only. This memory area is called as frame buffer.
- Each screen point is referred to as a pixel or pel.
- The process of turning on the pixels for a line segment is called vector generation.
- Digital differential analyser (DDA) is the vector generation algorithm which steps along the line to determine the pixel which should be turned on.
- Bresenham’s line algorithm uses only integer addition, subtraction and multiplication by 2.
- The basic principle of Bresenham’s line algorithm is to select the optimum raster locations to represent a straight line.
- The midpoint circle drawing algorithm uses the eight way symmetry of the circle to generate it.
- The midpoint ellipse drawing algorithm uses the four way symmetry of the ellipse to generate it.
- Frame buffer is implemented using rotating random access semiconductor memory.

**References**


**Recommended Reading**

Self Assessment

1. Which of the following statements is true?
   a. Computer graphics is generally regarded as a branch of computer science that deals with the theory and technology for image synthesis.
   b. Computer graphics is generally regarded as a branch of computer science that deals with the theory and technology for computerised image synthesis.
   c. Computer graphics is generally regarded as a branch of computer technology that deals with the theory for computerised image synthesis.
   d. Computer graphics is generally regarded as a branch of science that deals with the theory and technology for computerised image synthesis.

2. The process of representing continuous picture or graphics object as a collection of discrete pixels is called __________.
   a. scan conversion
   b. rasterisation
   c. frame buffer
   d. vector generation

3. Which of the following statements is false?
   a. Frame buffer is implemented using rotating random access semiconductor memory.
   b. The midpoint circle drawing algorithm uses the four way symmetry of the ellipse to generate it.
   c. The frame buffer is called a bitmap on a black and white system with one bit per pixel.
   d. The frame buffer is referred as a pixmap for systems with multiple bits per pixel.

4. __________ is the fundamental element of the picture representation.
   a. Line segment
   b. Line
   c. Vectors
   d. Polygon

5. What is an ellipse function?
   a. \( f_{circle}(x, y) = x + y^2 - r^2 \)
   b. \( f_{circle}(x, y) = x^2 + y^2 - r^2 \)
   c. \( f_{ellipse}(x, y) = \left( r_1 y \right)^2 \left( x^2 + r_2 y^2 - r_3 r_4 \right) \)
   d. \( f_{ellipse}(x, y) = \left( r_1 x \right)^2 \left( x^2 + r_2 y^2 - r_3 r_4 \right) \)

6. Which of the following statements is true?
   a. The midpoint circle drawing algorithm uses the eight way symmetry of the circle to generate it.
   b. The Bresenham’s circle drawing algorithm uses the eight way symmetry of the circle to generate it.
   c. The DDA ellipse drawing algorithm uses the eight way symmetry of the circle to generate it.
   d. The midpoint circle drawing algorithm uses the four way symmetry of the circle to generate it.
7. What is the slope of a straight line?
   
   a. \( \Delta y = \frac{y_2 - y_1}{x_2 - x_1} \Delta x \)
   
   b. \( m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \)
   
   c. \( m = \frac{\Delta y}{\Delta x} = \frac{x_2 - x_1}{y_2 - y_1} \)
   
   d. \( \Delta y = \frac{y_2}{x_2} \Delta x \)

8. The frame buffer is referred to as a _________ for systems with multiple bits per pixel.
   
   a. pixmap
   
   b. pel
   
   c. pixel
   
   d. resolution

9. What is each screen point referred to as?
   
   a. pixel
   
   b. pel
   
   c. pixmap
   
   d. pix

10. One shift register is required per pixel on a scan line and the length of _________ in bits is equal to number of scan lines.
   
   a. line segment
   
   b. shift register
   
   c. vectors
   
   d. frame buffers
Chapter II
Graphics Primitives

Aim

The aim of this chapter is to:

• explain display devices
• define raster scan
• introduce random scan display

Objectives

The objectives of this chapter are to:

• define different types of display devices
• explain the input and output devices
• enlist the characteristics of video display devices

Learning outcome

At the end of this chapter, you will be able to:

• define display processor
• identify vector scan display and raster scan display
• understand primitive operations
2.1 Display Devices

The display devices are also known as output devices. An output device is a piece of computer hardware equipment used to communicate the results of data processing carried out by a computer system to the outside world.

Output devices include:
- Cathode ray tube
- Random scan display
- Raster scan display
- Monitors
- Speakers
- Headphones
- Printer

The most commonly used output devices in graphics system is a video monitor. The operation of most video monitors is based on the standard cathode ray tube (CRT) design.

2.1.1 Cathode Ray Tubes (CRT)

- A CRT is an evaluated glass tube.
- Beam of electrons are produced by an electron gun at the rear of the tube which is directed towards the front of the screen.
- The inner side of the screen is coated with phosphor substance which gives off light when it is stroked by electrons.
- It is possible to control the point at which the electron beam strikes the screen and therefore, the position of the dot upon the screen, by deflecting the electron beam.
- The figure below shows the electrostatic deflection of the electron beam in a CRT.

![Fig. 2.1 Cathode ray tube](image.png)

- The deflection system of the cathode ray tube consists of two pairs of parallel plates, referred as the vertical and horizontal deflection plates.
- The voltage applied to vertical plates controls the vertical deflection of the electron beam and voltage applied to the horizontal deflection plates controls the horizontal deflection of the electron beam.
There are two ways which are used to produce images on the CRT screen.
- Vector scan/random scan
- Raster scan

2.1.2 Vector Scan/Random Scan Display
- From the beam across the CRT, the information about both, magnitude and direction is required. This information is generated with the help of vector graphics generator.

![Fig. 2.2 Vector scan CRT](image)

![Fig. 2.3 Architecture of a vector display](image)

- From the above figure, we can see the typical vector display architecture.
- It consists of display controller, Central Processing Unit (CPU), display buffer memory and a CRT.
- A display controller is connected as an input or output peripheral to the central processing unit (CPU).
- The display buffer memory stores the computer produced display list or display program.
- The display controller interprets commands for plotting points, lines and characters and sends digital and point coordinates to a vector generator.
- The vector generator then converts and sends digital coordinate values to analog voltages for beam-deflection circuits that display an electron beam writing on the CRT’s phosphor coating.
- In vector display, beam is deflected from end point to end point, hence this technique is also called random scan.
- We know as beam strikes phosphor it emits light.
• But a phosphor light decays after few milliseconds and therefore, it is necessary to repeat through the display list to refresh the phosphor at least 30 times per second to avoid flicker.
• As display buffer is used to store display list and it is used for refreshing the display buffer memory, it is also called refresh buffer.

2.1.3 Raster Scan Display

![Architecture of a raster display](image)

**Fig. 2.4 Architecture of a raster display**

• As shown in the above figure, the display is stored in the form of 1s and 0s in the refresh buffer.
• The video controller reads this refresh buffer and produces the actual image on the screen.
• It does this by scanning one scan line at a time, from top to bottom and then back to the top.
• Raster scan is the most common method of displaying images on the CRT screen.
• In this method, the horizontal and vertical deflection signals are generated to move the beam all over the screen in a pattern as shown in the figure below.

![Raster scan CRT](image)

**Fig. 2.5 Raster scan CRT**
Here, the beam is swept back and forth from the left to the right across the screen.
• When the beam is moved from the left to the right, it is ON and when it is moved from the right to the left, the beam is OFF.
• When the beam reaches the bottom of the screen, it is turned OFF and is rapidly retraced back to the top left to start again.
• A display produced in this way is called raster scan display.
• In the raster scan display, the screen image is maintained by repeatedly scanning the same image. This process is known as refreshing of screen.
• In raster scan display, a special area of memory is dedicated to graphics only. This memory area is called frame buffer.
• It holds the set of intensity values for all the screen points. The stored intensity values are retrieved from frame buffer and displayed on the screen one row at a time.

<table>
<thead>
<tr>
<th>Vector Scan Display</th>
<th>Raster Scan Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>• The beam is moved between the end points of the graphics primitives.</td>
<td>• The beam is moved all over the screen, one scan line at a time.</td>
</tr>
<tr>
<td>• Vector display flickers when the number of primitives in the buffer becomes too large.</td>
<td>• The refresh process is independent of the complexity of the image.</td>
</tr>
<tr>
<td>• Scan conversion is not required.</td>
<td>• Graphics primitives are specified in terms of their endpoints and must be scan converted into their corresponding pixels in the frame buffer.</td>
</tr>
<tr>
<td>• Scan conversion hardware is not required.</td>
<td>• Because each primitive must be scan-converted, real time dynamics is far more computational and requires separate scan conversion hardware.</td>
</tr>
<tr>
<td>• It draws continuous and smooth lines.</td>
<td>• It can display mathematically smooth lines, polygons and boundaries of curved primitives only one by approximating them with pixels on raster grid.</td>
</tr>
<tr>
<td>• Cost is more.</td>
<td>• Cost is low.</td>
</tr>
<tr>
<td>• It only draws lines and characters.</td>
<td>• It has ability to display areas filled with solid colours or patterns.</td>
</tr>
</tbody>
</table>

Table 2.1 Differentiation between vector and raster scan display

2.1.4 Important Characteristics of Video Display Devices

• Persistence: The major difference between phosphors is their persistence. It decides how long they continue to emit light after the electron beam is removed. Persistence is defined as the time it taken for the emitted light from the screen to decay to one-tenth of its original intensity. Lower persistence phosphors require higher refreshing rates to maintain a picture on the screen without flicker. It is useful for displaying animations and also displaying static and highly complex pictures.
• Resolution: Resolution indicates the maximum number of points that can be displayed without overlap on the CRT. It is defined as the number of points per centimetre that can be plotted horizontally and vertically. Resolution depends on the type of phosphor, the intensity to be displayed and the focusing and deflection systems used in CRT.
• Aspect ratio: it is the ratio of vertical points to horizontal points to produce equal length lines in both directions on the screen. An aspect ratio of 4:5 means that a vertical line plotted with four points has the same length as a horizontal line plotted with five points.


2.2 Interactive Devices

An interactive device in other words is called the input device which gives input to the graphics system and provides necessary feedback to the user about the accepted input. Interactive devices include:

- keyboard
- mouse
- trackball
- joystick
- data glove
- touch panels
- light pen

2.2.1 The Keyboard

The keyboard is a primary input device. It includes different keys and each key sends a different signal to the CPU. Most of the keys are useful for entering text and numbers. Typing is the ability to enter text and numbers. A standard keyboard includes about 100 keys.

2.2.2 Mouse

Mouse is an additional input, pointed device. This device enables the user to control the mouse pointer by moving the device around on a flat surface.

2.2.3 Trackball

A trackball works like an upside-down mouse as a pointing device. Trackball contains an exposed ball that can be rolled with an index finger or thumb to move the pointer around the screen. A trackball needs less space than a mouse as it is not necessary to move whole device.

2.2.4 Joystick

A joystick is a personal computer input device consisting of a handheld stick that pivots about one end and transmits its angle in two or three dimensional to a computer.

2.2.5 Data Glove

The data glove is used to grasp a virtual object. It is constructed with a series of sensors that detect hand and figure motions. The input from the glove can be used to position or manipulate objects in a virtual scene.

2.2.6 Light Pen

A Light Pen is a pointing device shaped like a pen acts as computer input device. The tip of the pen contains a light-sensitive element which, when placed against the screen, detects the light from the screen enabling the computer to identify the location of the pen on the screen.

2.3 Data Generating Devices

Scanners

- The scanner is a device, which can be used to store drawings, graphs, photos or text available in printed form for computer processing.
- The scanners use the optical scanning mechanism to scan the information.
- The scanner records the gradation of gray scales or colour and stores them in the array.
- It stores the image information in specific file format such as JPEG, GIF, TIFF, BMP, etc.
- Once the image is scanned, it allows the user to apply certain transformations like to rotate, scale or crop the image using image processing software such as photoshop or photopaint.
2.4 Primitive Operations

Simple Raster Graphics Package (SRGP) is a device independent of graphics package which supports graphics primitives such as lines, rectangles, circles, ellipses and text strings.

2.4.1 Syntax of Graphics Primitives

- Most of the graphics packages use Cartesian coordinate systems.
- There are two possible orientations for a Cartesian screen reference system.
- Consider the figure below. The standard coordinate orientation with the coordinate origin at the lower left corner of the screen is shown.
- In some systems, particularly in personal computers, the coordinate origin is at the upper left corner.

SRGP supports various graphics primitives such as lines, polynomials, circles, ellipse and text.

2.4.1.1 Lines and Polylines

Procedure 1: Valid SRGP_lineCoord (int x1, int y1, int x2, int y2);
[Draws a line from (x1, y1) to (x2, y2)]
Example: SRGP_lineCoord (0,0, 150, 250);

Procedure 2: Void SRGP_line (point P1, point P2);
[Here, point is defined structure having information of x and y values
Information of x and y values]
Typedef struct
{
    int x, y;
}
point ;

When successive vertices are connected by sequence of lines, the line is called polyline. The SRGP supports two separate procedures for polyline. Procedures store the coordinate information in arrays.

Procedure 3: Void SRGP_polylineCoord (int vertexCount, int* xArray, int* yArray);
[Here, xArray and yArray are pointers to user declared arrays for x and y coordinates respectively. vertexCount specified the number of vertices in the Polyline.]

Procedure 4: Void SRGP_polyline (int vertexCount, point * vertices);
[Here, vertices is a pointer to array of structure point]
Example: SRGP_polyline (6, VArray). (Refer figure 2.7)
2.4.1.2 Markers and Polymarkers

Sometimes it is convenient to place markers such as dots, asterisks or circles at the data points on graphs. SRGP supports the procedure to place markers and polymarkers in case of polylines.

Procedure 1: Void SRGP_MarkerCoord (int x, int y);
Procedure 2: Void SRGP_Marker (Point P);
Procedure 3: Void SRGP_polyMarkerCoord (int vertexCount, int*xArray, int* yArray);
Procedure 4: Void SRGP_polyMaker (int vertexCount, point * vertices);

Example: SRGP_PolyMaker (6, Array) (Refer to the figure below.)

![Fig. 2.8 Polyline with markers](image)

2.4.1.3 Polygons and Rectangles

Polyline is a chain of connected line segments. When starting point and terminal point of any polyline is same i.e., when polyline is closed then it is called polygon.

![Fig. 2.8 Polyline](image)

Procedure 1: Void SRGP_polygon (int vertexCount, point * vertices);
Example: SRGP_polygon (6, V Array); (Refer to the figure below.)
Ant rectangle can be specified as a polygon having four vertices, but an upright rectangle (one whose edges are parallel to the edges of the screen) can also be specified with the SRGP “rectangle primitive using only two vertices (bottom left and top right corners)

```c
2.4.2 Attributes

There are two types of Attributes. They are as follows:

**Line Attributes**

- The SRGP provides attributes for line drawing to control their appearance.
- These attributes are line style, line width, colour and pen style.
- There are separate procedures to set the attributes.
- Once the attributes are set, graphics primitive procedures follow set attributes to draw the graphics.

```
Procedure 4: void SRGP_loadColourTable (int start, int count, unsigned *r, unsigned *g, unsigned *b)
Procedure 5: void SRGP_loadCommonColour (int entry, char *ColourName);

**Marker Attributes**
Marker has two attributes: size and style.

Procedure 1: void SRGP_setMarkerSize (int width_in_pixels);
[The parameters width_in_pixel describes the dimensions of the imaginary square that circumscribes a marker’s image]
Procedure 2: void SRGP_setMarkerStyle (markerStyleType);
[SRGP supports three different marker styles. There are defines as follows,
  Typedef enum
  { MARKER_CIRCLE,
    MARKER_SQUARE,
    MARKER_X
  } MarkerStyleType;
The default marker style is circle]

2.4.3 Filled Primitives and Their Attributes
Filled Primitives and Fill Attributes are as follows:

**Filled Primitives**
- SRGP supports filling procedures.
- Its filled versions of area defining primitives, draw the interior pixels with no outline.
- To draw a filled polygon, SRGP_fillPolygon or SRGP_fillPolygonCoord procedures can be used with the same parameter lists used in the unfilled version of these calls.
- The other area_filling primitives are also defined in the same way, by prefixing “fill” to their names.
- There are two types of polygons: Convex and Concave.
  - Convex polygon: A convex polygon is a polygon in which the line segment joining any two points within the polygon lies completely inside the polygon.

  ![Fig. 2.11 Convex polygons](image)

  - Concave polygon: A concave polygon is a polygon in which the line segment joining any two points within the polygon may not lie completely inside the polygon.

  ![Fig. 2.12 Concave polygons](image)
**Fill Attributes**

There are two attributes supported by SRGP for filling the regions: fill style and fill pattern.

Procedure 1: Void SRGP_setFillStyle(fillstyle);

   [The fill style attributes can be used to control the appearance of a filled primitive’s interior in four
different ways. Its definition is as follows,

typedef enum
   {
      SOLID,
      BITMAP_PATTERN_OPAQUE,
      BITMAP_PATTERN_TRANSPARENT,
      PIXMAP_PATTERN
   } fillstyle;]

Procedure 2: Void SRGP_setFillBitmapPattern(int pattern_Index);

   [Bitmap fill patterns are bitmap arrays of 1s and 0s chosen from a table of available patterns.
   Pattern_Index specifies a particular entry in the pattern table. Each entry in the pattern table
   stores a unique pattern]

Procedure 3: Void SRGP_setBackgroundColour(int colour_Index);

Procedure 4: Void SRGP_setFillPixmapPattern(int pattern_Index);

**Pen Pattern for Outlines**

- The main advantage of patterning technique is area-defining primitives.
- The patterning can also be used to affect the appearance of lines and outline primitives and that is possible
  through the pen_style attribute.
- The attributes such as line_width, line_style, pen_style can be used for patterning.

Procedure 1: Void SRGP_setPenStyle(Penstyle Type);

   [The pen styles are already defined as follows.

typedef enum
   {
      Continuous solid,
      Dashed solid,
      Dashed bitmap pattern opaque,
      Dashed bitmap pattern transferred,
   } Penstyle Type;

   The default pen style is continuous solid]

Procedure 2: Void SRGP_setPenBitmapPattern(int patternIndex);

   [Bitmap fill patterns are bitmap arrays of 1s and 0s chosen from a table of available patterns.
   Pattern_Index specifies a particular entry in the pattern table. Each entry in the pattern table
   stores a unique pattern]

Procedure 3: Void SRGP_setPenPixmapPattern(int patternIndex);

**2.5 Display File Structure**

- The file used to store the commands necessary for drawing the line segments is called display file.
- In vector refresh display system, display processor uses the information in the display file to draw lines with
  the help of vector generating algorithms.
From the fig. 2.13, we can say that display files provide an interface between the image specification process and the image display process.

It also describes image in a compact format.

The concept of display file may be applied to devices other than refresh displays. Such files are called pseudo display files, or metafiles.

Fig. 2.14 shows the structure of display file. It contains series of commands.

Each display file command contains two fields, an operation code (opcode) and operands.

Opcode identifies the command such as draw line, move cursor, etc., and operand provides the coordinates of a point to process the command.

2.6 Display File Interpreter

Display file contains information necessary to construct the picture. This information is in the form of commands.

The program which converts these commands into actual picture is called display file interpreter.

It is an interface between graphics representation in the display file and the display device.

From the above figure, we can see that the display process is divided into two steps.

First the image is stored in the display file structure and then it is interpreted by an appropriate interpreter to get the actual image.

Instead of this, if we store actual image for particular display it may not run on other display.

To achieve the device independence, the image is stored in the raw format, i.e., in the display file format and then it is interpreted by an appropriate interpreter to run on required display.

Another advantage of using interpreter is that saving raw image takes much less storage space than saving the picture itself.
2.7 Normalised Device Coordinates

- Different display devices may have different screen sizes as measures in pixels.
- Sizes of the screen in pixels increases as resolution of the screen increases.
- When picture is defined in the pixel values then it is displayed large in size on the low resolution screen while the small in size on the high resolution screen.
- To avoid this and to make our program devices independent, we have to define the picture coordinates in some units other than pixels and use the interpreter to convert these coordinates to appropriate pixel values for the particular display device.
- The device independent units are called the normalised device coordinates.

2.8 Display Processor

- In the graphics systems, a separate computer is used to interpret the commands in the display file. Such computer is known as display processor.
- Display processor access display file and it cycles through each command in the display file once during every refresh cycle.
- The main task of display processor is to digitise a picture definition given in an application program into a set of pixel intensity values for storage in the frame buffer.
- This digitisation process is known as scan conversion.
- Display processors are also designed to perform a number of additional operations. These operations include,
  - generating various line styles (dashed, dotted, or solid)
  - display colour areas
  - performing certain transformations
  - manipulating on displayed objects

![Fig. 2.15 Raster scan system with a display processor](image-url)
Summary

• The display devices are also known as output devices. An output device is a piece of computer hardware equipment used to communicate the results of data processing carried out by a computer system to the outside world.
• The most commonly used output devices in graphics system is a video monitor.
• A CRT is an evaluated glass tube.
• Raster scan is the most common method of displaying images on the CRT screen.
• In raster scan display a special area of memory is dedicated to graphics only. This memory area is called frame buffer.
• The scanner is a device, which can be used to store drawings, graphs, photos or text available in printed form for computer processing.
• Simple Raster Graphics Package (SRGP) is a device independent of graphics package which supports graphics primitives such as lines, rectangles, circles, ellipses and text strings.
• Polyline is a chain of connected line segments.
• The SRGP provides attributes for line drawing to control their appearance.
• A convex polygon is a polygon in which the line segment joining any two points within the polygon lies completely inside the polygon.
• A concave polygon is a polygon in which the line segment joining any two points within the polygon may not lie completely inside the polygon.
• In some graphics systems a separate computer is used to interpret the commands in the display file. Such computer is known as display processor.

References


Recommended Reading

Self Assessment

1. The display devices are also known as ________ devices.
   a. output
   b. input
   c. raster scan
   d. random scan

2. The most commonly used output devices in graphics system is a ________.
   a. CPU
   b. video monitor
   c. keyboard
   d. scanner

3. A display controller is connected as an input or output peripheral to the ________.
   a. (CPU)
   b. monitor
   c. scanner
   d. mouse

4. Which of the following statements is false?
   a. The display controller interprets commands for plotting points, lines and characters and sends digital and
      point coordinates to a vector generator.
   b. In vector display, beam is deflected from end point to end point, hence this technique is also called random
      scan.
   c. As display buffer is used to store display list and it is used for refreshing the display buffer memory, it is
      also called refresh buffer.
   d. In vector display, beam is deflected from end point to end point, hence this technique is also called raster
      scan.

5. Which of the following statements is false?
   a. The scanner records the gradation of gray scales or colour and stores them in the array.
   b. A concave polygon is a polygon in which the line segment joining any two points within the polygon lies
      completely inside the polygon.
   c. Once the image is scanned, it allows the user to apply certain transformations like to rotate, scale or crop
      the image using image processing software such as photoshop or photopaint.
   d. A convex polygon is a polygon in which the line segment joining any two points within the polygon lies
      completely inside the polygon.

6. ________ is a device independent of graphics package which supports graphics primitives such as lines,
   rectangles, circles, ellipses and text strings.
   a. CPU
   b. CRT
   c. Simple Raster Graphics Package (SRGP)
   d. Scanner
7. Which of the following statements is false?
   a. Most of the graphics packages do not use Cartesian coordinate systems.
   b. The video controller reads this refresh buffer and produces the actual image on the screen.
   c. The main task of display processor is to digitise a picture definition given in an application program into a set of pixel intensity values for storage in the frame buffer.
   d. Most of the graphics packages use Cartesian coordinate systems.

8. In raster scan display, the special area of memory that is dedicated only to graphics, is called as ______.
   a. Frame
   b. frame buffer
   c. buffer
   d. operands

9. What are the fields of display file command?
   a. pseudo display files
   b. operation code (opcode) and operands
   c. metafiles
   d. frame buffer

10. In some graphics systems, a separate computer is used to interpret the commands in the display file. Such computer is known as ________.
    a. raster scan display
    b. random scan display
    c. pseudo files
    d. display processor
Chapter III

Polygons

Aim

The aim of this chapter is to:

- define polygons
- explain boundary fill and edge fill algorithms
- explicate the features of scan line algorithm

Objectives

The objectives of this chapter are to:

- enlist different types of polygons
- explain the fill with pattern techniques
- define scan conversion

Learning outcome

At the end of this chapter, you will be able to:

- identify polygon
- understand algorithms
- differentiate between different types of algorithms
3.1 Introduction
A polyline is a chain of connected line segments. It is specified by giving the vertices \( x, y, z, \ldots \) and so on. The vertex is called the initial or starting point and the last vertex is called the final or terminal point. (Refer to the figure below). When starting point and terminal point of any polyline is same, i.e., when polyline is closed, then it is called polygon. (Refer fig. 3.2)

![Polyline](image1)

Fig. 3.1 Terminal point

3.2 Types of Polygons
The classification of polygons is based on where the line segment joining any two points within the polygon is going to lie. There are two types of polygons: Convex and Concave.

- Convex polygon: A convex polygon is a polygon in which the line segment joining any two points within the polygon lies completely inside the polygon.

![Convex polygon](image2)

Fig. 3.2 Convex polygon

- Concave polygon: A concave polygon is a polygon in which the line segment joining any two points within the polygon may not lie completely inside the polygon.

![Concave polygon](image3)

Fig. 3.3 Concave polygon
3.3 Representation of Polygons

- Closed polyline is a polygon.
- Each polygon has sides and edges.
- The end points of the sides are called the polygon vertices.
- It is necessary to first decide how to represent a polygon before adding it to the computer systems.
- There are three approaches to represent polygons according to the graphics systems:
  - Polygon drawing primitive approach
  - Trapezoid primitive approach
  - Line and point approach
- Some graphics devices support polygon drawing primitive approach. They can directly draw the polygon shapes.
- On such devices polygons are saved as a unit.
- Some graphics devices support trapezoid primitive. In such devices, trapezoids are formed from two scan lines and two line segments. (Refer to the figure below.)

![Fig. 3.4 Representation of polygon](image)

3.4 Entering Polygons

The information of entering number of sides and the coordinates of the vertex points can be obtained by using following algorithm.

Algorithm: Entering the polygon into the display file

1. Read AX and AY of length N
   [AX and AY are arrays containing the vertices of the polygon and N is the number of polygon sides]
2. \[i = 0\] [initialise counter to count number of sides]
   \[\text{DF}_\text{OP}[i] \leftarrow N\]
   \[\text{DF}_x[i] \leftarrow AX[i]\]
   \[\text{DF}_y[i] \leftarrow AY[i]\]
   \[i \leftarrow i + 1\]
   [Load polygon command]
3. do
   \{\]
   \[DF_\text{OP}[i] \leftarrow 2\]
   \[DF_x[i] \leftarrow AX[i]\]
   \[DF_y[i] \leftarrow AY[i]\]
   \[I \leftarrow i + 1\]
   \}\[While (i < N)\] [Enter line commands]
4. \[\text{DF}_\text{OP}[i] \leftarrow 2\]
   \[\text{DF}_x[i] \leftarrow AX[0]\]
DF_\text{y}[i] \leftarrow AY[0]
[Enter last line command]

5. Stop

3.5 Polygon Filling

- Filling the polygon means highlighting all the pixels which lie inside the polygon with any colour than background colour.
- Polygons are easier to fill since they have liner boundaries.
- There are two basic approaches to fill the polygon.
- One way to fill a polygon is to start a given “seed” point known to be inside the polygon and highlight outward from this point until we encounter the boundary pixels. This approach is called seed fill.
- Another way to fill a polygon is to apply the inside test i.e. to check whether the pixel is inside the polygon or outside the polygon and then highlight pixels which lie inside the polygon. This approach is known as scan-line algorithm.

3.5.1 Seed Fill

- The seed fill algorithm is further classified as flood fill algorithm and boundary fill algorithm.
- Algorithms that fill interior-designed regions are called flood-fill algorithms.
- Algorithms that fill boundary-defined regions are called boundary-fill algorithms or edge-fill algorithms.

3.5.1.1 Boundary Fill Algorithm / Edge Fill Algorithm

- Edges of the polygons are drawn in this method.
- Then starting with some seed, any point inside the polygon we examine the neighbouring pixels to check whether the boundary pixel is reached.
- If boundary pixels are not reached, pixels are highlighted and the process is continued until boundary pixels are reached.
- Boundary defined regions may be either 4-connected or 8-connected. (Refer to the figure below.)
- If a region is 4-connected, then every pixel in the region may be reached by a combination of moves in only four directions.
- For an 8-connected region every pixel in the region may be reached by a combination of moves in the two horizontal, two vertical and four diagonal directions.

![Fig. 3.5 Boundary defined regions](image-url)
In some cases, an 8-connected algorithm is more accurate than the 4-connected algorithm.

![Fig. 3.6 Partial filling resulted using 4-connected algorithm](image)

---

**3.5.1.2 ‘C’ Code for Boundary Algorithm (8-Connected Region)**

```c
#include<stdio.h>
#include<graphics.h>

main ( )
{
    int gd, gm;

    /* initialise graphics mode
       -----------------------------------*/
    detectgraph (&gd, &gm);
    initgraph (&gd, *gm, " ");

    rectangle (50, 50, 100, 100);
    flood (55, 55, 4, 15);
    getch ( );
    closegraph ( );
}

flood (seed_x, seed_y, foreground_col, background_col)
{
    if (getpixel (seed_x, seed_y) != background_col &&
        getpixel (seed_x, seed_y) != foreground_col)
    {
        putpixel (seed_x, seed_y, foreground_col);
        flood (seed_x+1, seed_y, foreground_col, background_col);
        flood (seed_x-1, seed_y, foreground_col, background_col);
        flood (seed_x, seed_y+1, foreground_col, background_col);
        flood (seed_x, seed_y-1, foreground_col, background_col);
        flood (seed_x+1, seed_y+1, foreground_col, background_col);
        flood (seed_x-1, seed_y-1, foreground_col, background_col);
        flood (seed_x-1, seed_y+1, foreground_col, background_col);
        flood (seed_x-1, seed_y+1, foreground_col, background_col);
    }
}
```
3.5.1.3 Flood Fill Algorithm

- Sometimes, it is required to fill in an area that is not defined within a single colour boundary.
- In such cases, we can fill areas by replacing a specified interior colour instead of searching for a boundary colour. This approach is called a flood fill algorithm.
- Like boundary fill algorithm, here also we start with some seed and examine the neighbouring pixels.
- However, here pixels are checked for a specified interior colour instead of boundary colour and they are replaced by new colour.
- Using either a 4-connected or 8-connected approach, we can step through pixel positions until all interior point have been filled.

Procedure: flood_fill (x, y, old_color, new_colour),
{
    if (getpixel (x, y) = old_color)
    {
        putpixel (x, y, new_color);
        flood_fill (x+1, y, old_color, new_colour);
        flood_fill (x-1, y, old_color, new_color);
        flood_fill (x, y+1, old_color, new_colour);
        flood_fill (x, y-1, old_color, new_colour);
        flood_fill (x+1, y+1, old_color, new_colour);
        flood_fill (x-1, y-1, old_color, new_colour);
        flood_fill (x+1, y-1, old_color, new_colour);
        flood_fill (x-1, y+1, old_color, new_colour);
    }
}

3.5.2 Scan Line Algorithm

- Recursive algorithm for seed fill method has got two difficulties.
- The first difficulty is that if some inside pixels are already displayed in fill colour then recursive branch terminates, leaving further internal pixels unfilled.
- To avoid this difficulty, we have to first change the colour of any internal pixels that are initially set to the fill colour before applying the seed fill procedures.
- Another difficulty with this method is that, it cannot be used for large polygons.
- This is because, recursive seed fill procedures require stacking of neighbouring points and in case of large polygons stack space may be insufficient for stacking.
- To avoid this problem, more efficient method can be used. Such method fills horizontal pixel spans across scan lines, instead of proceeding to 4-connected or 8-connected neighbouring points.
- This is achieved by identifying the rightmost and leftmost pixels of the seed pixel and then drawing a horizontal line between these two boundary pixels.
This procedure is repeated with changing the seed pixel above and below the line just drawn until complete polygon is filled.

With this efficient method, we have to stack only a beginning position for each horizontal pixel span, instead of stacking all unprocessed neighbouring positions around the current positions.

![Scan line diagram](image)

**Fig. 3.7 Scan line**

The important task in the scan line algorithm is to find the intersection points of the scan line with the polygon boundary.

When intersection points are even, they are sorted from left to right, paired and pixels between paired points are set to the fill colour.

But in some cases intersection point is a vertex.

When scan line intersects polygon vertex a special handling is required to find the exact intersection points.

To handle such cases, we must look at the other endpoints of the two line segments of the polygon which meet at this vertex.

If these points lie on the same (up and down) side of the scan line, then the point in question counts as an even number of intersections.

If they lie on opposite sides of the scan line, then the point is counted as single intersection.

![Intersection points along the scan line that intersect polygon vertices](image)

**Fig. 3.8 Intersection points along the scan line that intersect polygon vertices**

3.5.2.1 ‘C’ Code for Scan Line Algorithm for Filling Polygon

```c
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
```

40
/* defining the structure to store edges  
----------------------------------------------*/

struct edge  
{  
int x1;  
int y1;  
int x2;  
int y2;  
int flag;  
};

void main ( )  
{  
int gd = DETECT, gm, n, i, j, k;  
struct edge ed [10], temped;  
float dx, dy, m[10], x_int[10], inter_x[10];  
inithgraph (&gd, &gm, " ");

/* Read the number of vertices of the polygon  
----------------------------------------------------*/
printf ("Enter the number vertices of the graph: ");
scanf ("%d", &n);

/* Read the vertices of the polygon and also find Ymax and Ymin  
------------------------------------------------------*/
printf ("Enter the vertices: 
");
for (i=0; i< n; i++)  
{  
printf ("x[%d] : ", i);  
scanf ("%d", &x[i]);  
printf ("y[%d] : ", i);  
scanf ("%d", &y[i]);  
if (y[i] > ymax)  
ymax = y[i];  
if (y[i] < ymin)  
ymin = y[i];  
ed[i].x1 = x[i];  
ed[i].y1 = y[i];  
}

/*Store the edge information  
-----------------------------------*/
for  (i = 0; i< n-1; i++)  
{  
ed[i].x2 = ed[i+1].x1;  
ed[i].y2 = ed[i+1].y1;  
ed[i].flag = 0;  
}  
ed[i].x2 = ed[0].x1;  
ed[i].y2 = ed[0].y1;  
ed[i].flag = 0;

/* Check for y1>y2, if not interchange y1 and y2
with corresponding x1 and x2-------------*/
for (i=0; i<n; i++)
{
    if(ed[i].y1<ed[i].y2)
    {
        temp = ed[i].x1;
        ed[i].x1 = ed[i].x2;
        ed[i].x2 = temp;
        temp = ed[i].y1;
        ed[i].y1 = ed[i].y2;
        ed[i].y2 = temp;
    }
}

/* Draw the polygon
---------------------------------------*/
for (i=0; i<n; i++)
{
    line(ed[i].x1.ed[i].y1.ed[i].ed[i].y2);
}

/* Sorting of edges in the order of y1, y2, x1
------------------------------------------------*/
for (i=0; i<n-1; i++)
{
    for (j=0; j<n-1; j++)
    {
        if(ed[j].y1<ed[j+1].y1)
        {
            temped = ed[j];
            ed[j] = ed[j+1];
            ed[j+1] = temped;
        }
        if(ed[j].y1 = =ed[j+1].y1
        {
            if (ed[j].y2<ed[j+1].y2)
            {
                temped = ed[j];
                ed[j] = ed [j+1];
                ed[j+1] = temped;
            }
        }
if(ed[j+1] = temped;
}
}
if(ed[j].x1<ed[j+1].x1)
{
    temped = ed[j];
ed[j] = ed[j+1];
ed[j+1] = temped;
}
/* Calculating 1/slope of each edge and storing top x coordinate of the edge-------------------------*/
for (i=0; i<n; i++)
{
    dx = ed[i].x2 – ed[i].x1;
dy = ed[i].y2 – ed[i].y1;
    if(dy ==0)
    {
        m[i] = 0;
    }
    else
    {
        m[i] = dx/dy;
    }
    inter_x[i] = ed[i].x1;
}yy = ymax;
while(yy>ymin)
{
    /* Marking active edges-------------------------------*/
    for(i=0; i<n; i++)
    {
        if(yy > ed[i].y2 && yy <= ed[i].y1)
        {
            ed[i].flag = 1;
        }
else
{
ed[i].flag = 0;
}

/* Finding the x intersections
----------------------------------*/
j=0;
for(i=0; i<n; i++)
{
if(ed[i].flag == 1)
{
if(yy == ed[i].y1)
{
x_int[j] = ed[i].x1;
    j++;
    if(ed[i-1].y1 == yy && ed[i-1].y1<yy)
    {
x_int[j] = ed[i].x1;
        j++;
    }
    if ed[i-1].y1 = = yy && ed[i+1].y1<yy)
    {
x_int[j] = ed[i].x1;
        j++;
    }
    if ed[i-1].y1 = = yy && ed[i-1].y1<yy)
    {
x_int[j] = ed[i].x1;
        j++;
    }
}
if ed[i-1].y1 = = yy && ed[i+1].y1<yy)
{
x_int[j] = ed[i].x1;
    j++;
}
}
else
{
{  
x_int[j] = inter_x[i]+(-m[i]);

inter_x[i] = x_int[j];

j++;
}
}
}

/* Sorting the x intersections  
--------------------------------*/
for(i=0; i<j; i++)
{
for(k=0; k<j-1; k++)
{
if(x_int[k]>x_int[k+1])
{
    temp = x_int[k];
    x_int[k] = x_int[k+1];
    x_int[k+1] = temp;
}
}

/* Extracting pairs of x values to draw lines  
-----------------------------------------------*/
for(i=0; i<j; i+=2)
{
    line(x_int[i],yy,x_int[i+1],yy);
}

yy--;
 delay(50);
}
getch ( );
}
3.5.2.2 Features of Scan Line Algorithm

- Scan line algorithm is used for filling polygons.
- This algorithm solves the problem of hidden surfaces while generating display scan line.
- It is used in orthogonal projection.
- It is non recursive algorithm.
- In scan line algorithm, we have stack only at the beginning position for each horizontal pixel scan, instead of stacking all unprocessed neighbouring positions around the current positions. Therefore, it is an efficient algorithm.

3.6 Filling with Patterns

- Pattern is a small group of pixels with a fixed colour combination used to fill the particular area in the picture.
- These patterns can be of any shape.
- However, rectangular and square patterns are generally used to fill area.
- Filling an area with a rectangular pattern is called tiling and rectangular patterns are sometimes referred to as tiling patterns.
- Pattern fill can be implemented by modifying the scan line algorithm, so that a selected pattern is superimposed onto the scan lines.
- In the pattern filling, it is necessary to decide which pixel in the pattern corresponds to the current pixel of the primitive.
- Relating the area of the pattern with the area of the primitive is known as anchoring.
- There are two methods to implement pattern filling:
  - Stamping
  - True filling
- In stamping, we know the size of pattern.
- The pattern will be rectangular in nature, say having ‘x’ row and ‘y’ columns, we can find the centre of that pattern and then we can proceed as per scan line.
- Our aim will be to find the position of the centre of each row and then check whether it is inside the polygon.
- When the calculated centre position is inside the polygon, mark the full pattern with that point as the centre. In this case, we are not bothered whether the complete rectangular region which is covered by the pattern inside the polygon. This is known as stamping.
- To implement true filling, we should be first finding the centre of the pattern. We should confirm that the pixel is inside the polygon.

![Filling the figure with stamping](image1)

![True filling by the pattern gets cut](image2)

Fig. 3.9 Filling pattern

3.7 Scan Conversion

Display file stores the information about lines and character whereas scan conversion gives the information about every pixel on the screen. This information must be organised and presented at video rates in a scan line order, which is from the top to the bottom and from left to right.
Summary

- A polyline is a chain of connected line segments. It is specified by giving the vertices $x, y, z, \ldots$ and so on.
- The vertex is called the initial or stating point and the last vertex is called the final or terminal point.
- The classification of polygons is based on where the line segment joining any two points within the polygon is going to lie.
- A convex polygon is a polygon in which the line segment joining any two points within the polygon lies completely inside the polygon.
- A concave polygon is a polygon in which the line segment joining any two points within the polygon may not lie completely inside the polygon.
- Filling the polygon means highlighting all the pixels which lie inside the polygon with any colour than background colour.
- Algorithms that fill interior-designed regions are called flood-fill algorithms.
- Algorithms that fill boundary-defined regions are called boundary-fill algorithms or edge-fill algorithms.
- Boundary defined regions may be either 4-connected or 8-connected.
- The important task in the scan line algorithm is to find the intersection points of the scan line with the polygon boundary.
- Filling an area with a rectangular pattern is called tiling and rectangular patterns are sometimes referred to as tiling patterns.
- Display file store the information about lines and character whereas scan conversion gives the information about every pixel on the screen.

References


Recommended Reading

Self Assessment

1. A polyline is a chain of connected ________.
   a. line segments
   b. lines
   c. polygons
   d. points

2. Which of the following statements is false?
   a. The vertex is called the initial or starting point and the last vertex is called the final or terminal point.
   b. A convex polygon is a polygon in which the line segment joining any two points within the polygon lies completely inside the polygon.
   c. A concave polygon is a polygon in which the line segment joining any two points within the polygon may not lie completely inside the polygon.
   d. A concave polygon is a polygon in which the line segment joining any two points within the polygon lies completely inside the polygon.

3. When starting point and terminal point of any polyline is same i.e., when polyline is closed then it is called ________.
   a. line
   b. quadrilateral
   c. polygon
   d. segment

4. Which of the following statements is false?
   a. Filling the polygon means highlighting all the pixels which lie outside the polygon with any colour than background colour.
   b. Polygons are easier to fill since they have linear boundaries.
   c. Filling an area with a rectangular pattern is called tiling and rectangular patterns are sometimes referred to as tiling patterns.
   d. Filling the polygon means highlighting all the pixels which lie inside the polygon with any colour than background colour.

5. What is known to be inside the polygon and highlight outward from the point until we encounter the boundary pixels?
   a. Boundary fill
   b. Seed fill
   c. Pattern fill
   d. Flood fill

6. What are the algorithms that fill interior-designed regions called?
   a. flood-fill algorithms.
   b. pattern fill algorithms
   c. boundary fill algorithms
   d. scan line algorithm
7. Which of the following statements is true?
   a. Algorithms that fill boundary-defined regions are called fill algorithms or edge-fill algorithms.
   b. Algorithms that fill boundary-defined regions are called flood fill algorithms or edge-fill algorithms.
   c. Algorithms that fill boundary-defined regions are called boundary-fill algorithms or edge-fill algorithms.
   d. Algorithms that fill boundary-defined regions are called pattern fill algorithms or edge-fill algorithms.

8. Which of the following statements is false?
   a. Recursive algorithm for seed fill method has got two difficulties.
   b. The important task in the scan line algorithm is to find the intersection points of the scan line with the polygon boundary.
   c. Scan line algorithm is used for filling polygons.
   d. Scan line algorithm is not used in orthogonal projection.

9. _______ is a small group of pixels with a fixed colour combination used to fill the particular area in the picture.
   a. Pattern
   b. Shape
   c. Polygon
   d. Polyline

10. Which of the following statements is true?
    a. Boundary defined regions may be either 6-connected or 8-connected.
    b. Boundary defined regions may be either 4-connected or 8-connected.
    c. Boundary defined regions may be either 4-connected or 6-connected.
    d. Boundary defined regions may be either 2-connected or 4-connected.
Chapter IV
2D Transformations

Aim
The aim of this chapter is to:

• define 2d transformation
• explain matrices and its uses
• explicate homogeneous coordinates

Objectives
The objectives of this chapter are to:

• explain matrix multiplication and identity matrix
• define translation, rotation and scaling in 2D transformation
• elucidate rotation about an arbitrary point

Learning outcome
At the end of this chapter, you will be able to:

• solve matrices
• understand composition of 2D transformations
• identify homogeneous coordinates in translation
4.1 Matrices

- In computer graphics, images are generated from a series of line segments which are represented by the coordinates of their end points.
- Due to this we can make changes in an image by performing mathematical operations on these coordinates. When any changes are made we say that the image is transformed.
- To study various image transformations, mathematical tools like matrix multiplication is needed.
- Matrix is a two dimensional array of number.

Example:

\[
\begin{bmatrix}
0 & 1 \\
2 & 3
\end{bmatrix}
\begin{bmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8
\end{bmatrix}
\begin{bmatrix}
1 \\
2 \\
3
\end{bmatrix}
\begin{bmatrix}
0 & -1 & 3 \\
0 & 1 & 2 \\
5 & 4 & 2
\end{bmatrix}
\]

The elements in the matrix are identified by specifying the row and column number.

Example:

\[
A = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]

In the above example, element 8 is identified as \( A (3, 2) \), i.e., element 8 is in the 3\(^{rd}\) row and 2\(^{nd}\) column of the matrix A.

### 4.1.1 Matrix Multiplication

- Matrix multiplication is more complex than the simple product of two numbers.
- It involves simple products and sums of the matrix elements.
- We can multiply two matrices \( A \) and \( B \) together only if the number of column of the first matrix \( A \) is the same as the number of rows of the second matrix \( B \).

Consider:

\[
B = \begin{bmatrix}
1 & 2 \\
-2 & 0 \\
3 & 1
\end{bmatrix}
\]

Here we can multiply matrix \( A \) and \( B \) because matrix \( A \) has 3 columns and matrix \( B \) has 3 rows. Matrix multiplication is not commutative like number multiplication that is, while we can multiply \( A \) times \( B \), we cannot multiply \( B \) times \( A \), because \( B \) has 2 columns and \( A \) has 3 rows. When we multiply two matrices we get resultant matrices such that it has same number of rows as the first matrix of the two being multiplied and the same number of columns as the second matrix. Therefore, multiplication of \( A \) and \( B \) gives 3 x 4 resultant \( C \) matrix.

The generalised formula for matrix multiplication is given by,

\[
C (i, k) = \sum A (i, j) B (j, k)
\]

In the example, the element \( C (1, 1) \) is found by multiplying each element of the first row of \( A \) by the corresponding element of the first column of \( B \) and adding these products together.

\[
C (1, 1) = A (1, 1) B (1, 1) + A (1, 2) B (2, 1) + A (1, 3) B (3, 1)
= (1) (1) + (2) (-2) + (3) (3)
= 6
\]
Similarly,
\[
C (1, 2) = A (1, 1) B (1, 2) + A (1, 2) B (2, 2) + A (1, 3) B (3, 2) \\
= (1) (2) + (2) (0) + (3) (1) \\
= 5
\]

Performing similar arithmetic we get:

\[
C = \begin{bmatrix}
6 & 5 \\
12 & 14 \\
18 & 23
\end{bmatrix}
\]

### 4.1.2 Identity Matrix

The square matrix which has all the elements equal to 0 except the elements of the main diagonal, which are all 1 is called identity matrix.

Example:

\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

The identity matrix is denoted as matrix I

Therefore,

\[ A = A \times I \]

### 4.2 Two Dimensional Transformations

Let's focus on few of the aspects of two dimensional transformation.

#### 4.2.1 Translation

- Translation is a process of changing the position of an object in a straight-line path from one co-ordinate location to another.
- We can translate a two dimensional point by adding translation distances, \( t_x \) and \( t_y \), to the original coordinate position \((x, y)\) to move the point to a new position \((x', y')\). (Refer to the figure below.)

\[
x' = x + t_x \quad \text{........................ (i)}
\]

\[
y' = y + t_y \quad \text{........................ (ii)}
\]

![Fig. 4.1 Translation](image)

- The translation distance pair \((t_x, t_y)\) is called a translation vector or shift vector.
- Translation equations (i) and (ii) can be expressed as a single matrix equation by using column vectors to represent coordinate positions and the translation vector:

\[
P = \begin{bmatrix}
x \\
y
\end{bmatrix}
\]

\[
P' = \begin{bmatrix}
x' \\
y'
\end{bmatrix}
\]

\[
T = \begin{bmatrix}
t_x \\
t_y
\end{bmatrix}
\]
• This allows us to write the two dimensional translation equations in the matrix form:
  \[ P' = P + T \] ................................ (iii)

4.2.2 Rotation

• A two dimensional rotation is applied to an object repositioning it along a circular path in the xy plane.
• To generate a rotation, we specify a rotation angle \( \theta \) and the position of the rotation about which the object is to be rotated.
• Let us consider the rotation of the object about the origin as shown in the figure below.

![Fig. 4.2 Rotation of object about the origin](image)

Here, \( r \) is the constant distance of the point from the origin, angle \( \phi \) is the original angular position of the point from the horizontal, and \( \theta \) is the rotation angle. Using standard trigonometric equations, we can express the transformed coordinates in terms of angles \( \theta \) and \( \phi \) as:

\[
\begin{align*}
  x' &= r \cos (\phi + \theta) = r \cos \phi \cos \theta - r \sin \phi \sin \theta \\
  y' &= r \sin (\phi + \theta) = r \cos \phi \sin \theta + r \sin \phi \cos \theta
\end{align*}
\]................................. (iv)

The original coordinates of the point polar coordinates are given as:

\[
\begin{align*}
  x &= r \cos \phi \\
  y &= r \sin \phi
\end{align*}
\]................................. (v)

Substituting equations (v) into (iv), we get the transformation equations for rotating a point \((x, y)\) through an angle \( \theta \) about the origin as:

\[
\begin{align*}
  x' &= x \cos \theta - y \sin \theta \\
  y' &= x \sin \theta + y \cos \theta
\end{align*}
\]................................. (vi)

The above equations can represent in the matrix form as:

\[
[x' \ y'] = [x, \ y] \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}
\]

\[ P' = P \cdot R \]................................. (vii)

where \( R \) is rotation matrix and it is given as:

\[
R = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}
\]................................. (viii)

It is important to note that positive values for the rotation angle define counter-clockwise rotations about the rotation point and negative values rotate objects in the clockwise sense.

For negative values of \( \theta \), i.e., for clockwise rotation, the rotation matrix becomes:

\[
R = \begin{bmatrix} \cos \theta & \sin (-\theta) \\ -\sin \theta & \cos (-\theta) \end{bmatrix}
\]
\[
R = \begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix}
\]

if, \( \cos (-\theta) = \cos \theta \) and \( \sin (-\theta) = -\sin \theta \) .......................... (ix)

4.2.3 Scaling

• A scaling transformation changes the size of an object.

• This operation can be carried out for polygons by multiplying the coordinate values \((x, y)\) of each vertex by scaling factors \(S_x\) and \(S_y\) to produce the transformed coordinates \((x', y')\).

\[
x' = x \cdot S_x
\]
and
\[
y' = y \cdot S_y
\]

Scaling factor \(S_x\) scales object in the x direction and scaling factor \(S_y\) scales object in the y direction. The equation (x) can be written in the matrix form as:

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
x \\
y
\end{bmatrix} \begin{bmatrix}
S_x & 0 \\
0 & S_y
\end{bmatrix}
\]

\[
= \begin{bmatrix}
x \cdot S_x \\
y \cdot S_y
\end{bmatrix}
\]

\[
= P \cdot S
\]

Fig. 4.3 Scaling

• Any positive numeric values are valid for scaling factors \(S_x\) and \(S_y\).

• Values less than 1 reduce the size of the objects and values greater than 1 produce an enlarged object.

• For both, \(S_x\) and \(S_y\) values equal to 1, the size of object does not change.

• To get uniform scaling it is necessary to assign same value for \(S_x\) and \(S_y\).

• Unequal values for \(S_x\) and \(S_y\) result in a differential scaling.
4.3 Homogeneous Coordinates

- In design and picture formation process, many times we may require to perform translation, rotations and scaling to fit the picture components into their proper positions.

- The basic transformations can be expressed in the general matrix form as:

\[ P' = P \cdot M_1 + M_2 \] .......................... (xii)

For translation: \[ P' = P \cdot \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \]

\[ \text{i.e.,} \quad M_1 = \text{Identity matrix} \]
\[ M_2 = \text{Translation factor} \]

For rotation: \[ P' = P \cdot \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} + \begin{bmatrix} \theta & \theta \end{bmatrix} \]

\[ \text{i.e.,} \quad M_1 = \text{Rotational matrix} \]
\[ M_2 = 0 \]

For scaling: \[ P' = P \cdot \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \]

\[ \text{i.e.,} \quad M_1 = \text{Scaling matrix} \]
\[ M_2 = 0 \]

- Now to produce a sequence of transformations with above equations, such as translation followed by rotation and then scaling, we must calculate the transformed coordinate one step at a time.

- First, coordinates are translated, and then these translated coordinates are scaled, and finally the scaled coordinates are rotated.

- But this sequential transformation process is not efficient.

- More efficient approach is to combine sequential transformation into one transformation, so that the final coordinate positions are obtained directly from initial coordinates.

- In order to combine sequence of transformations, we have to eliminate the matrix addition associated with translation terms in \[ M_2 \].

- To achieve this, we have to represent matrix \[ M_1 \] as 3 x 3 matrix, instead of 2 x 2 introducing an additional dummy coordinate \( W \).

- Here, points are specified by these numbers instead of two. This coordinate system is called homogeneous coordinate system.

- The homogeneous coordinate system allows us to express all transformation equations as matrix multiplication.

- The homogeneous coordinate is represented by a triplet \((X_w, Y_w, W)\),

\[ x = \frac{X_w}{W} \text{ and } y = \frac{Y_w}{W} \]
4.3.1 Homogeneous Coordinates for Translation
The homogeneous coordinates for translation are given as:
\[
T = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
t_x & t_y & 1
\end{bmatrix}
\]
Therefore, we get:
\[
\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
t_x & t_y & 1
\end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}
\]

4.3.2 Homogeneous Coordinates for Rotation
The homogeneous coordinates for rotation are given as:
\[
R = \begin{bmatrix}
\cos \theta & \sin \theta & 0 \\
-\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\]
Therefore, we get:
\[
\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \begin{bmatrix}
\cos \theta & \sin \theta & 0 \\
-\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix} = \begin{bmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + \cos \theta \\ 1 \end{bmatrix}
\]

4.3.3 Homogeneous Coordinates for Scaling
The homogeneous coordinate for scaling are given as:
\[
S = \begin{bmatrix}
S_x & 0 & 0 \\
0 & S_y & 0 \\
0 & 0 & 1
\end{bmatrix}
\]
Therefore, we get:
\[
\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \begin{bmatrix}
S_x & 0 & 0 \\
0 & S_y & 0 \\
0 & 0 & 1
\end{bmatrix} = \begin{bmatrix} x \cdot S_x \\ y \cdot S_y \\ 1 \end{bmatrix}
\]

4.4 Composition of 2D Transformations
The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformations, one after the other.

4.4.1 Rotation About an Arbitrary Point
To rotate object about an arbitrary point, \((X_p, Y_p)\) we have to carry out three steps:
- Translate point \((X_p, Y_p)\) to the origin.
- Rotate it about the origin.
- Finally, translate the centre of rotation back where it belongs (Refer to the figure below.)
Fig. 4.4 Rotation about an arbitrary point

The translation matrix to move point \((x_p, y_p)\) to the origin is given as:

\[
T_1 = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
x_p & -y_p & 1
\end{bmatrix}
\]

The rotation matrix for counter clockwise rotation of point about the origin is given as:

\[
R = \begin{bmatrix}
\cos\theta & \sin\theta & 0 \\
-\sin\theta & \cos\theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

The translation matrix to move the centre point back to its original position is given by:

\[
T_2 = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
x_p & y_p & 1
\end{bmatrix}
\]

Therefore, the overall transformation matrix for a counter clockwise rotation by an angle \(\theta\) about the point \((x_p, y_p)\) is given as:

\[
T_1 \cdot R \cdot T_2 = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
x_p & -y_p & 1
\end{bmatrix}
\begin{bmatrix}
\cos\theta & \sin\theta & 0 \\
-\sin\theta & \cos\theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
x_p & y_p & 1
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\cos\theta & \sin\theta & 0 \\
-\sin\theta & \cos\theta & 0 \\
x_p\cos\theta + y_p\sin\theta + x_p & x_p\sin\theta - y_p\cos\theta + y_p & 1
\end{bmatrix}
\]

\[
\text{(xviii)}
\]
Summary

- Matrix multiplication is more complex than the simple product of two numbers.
- In computer graphics, images are generated from a series of line segments which are represented by the coordinates of their end points.
- The generalised formula for matrix multiplication is given by, $C(i, k) = \sum A(i, j) B(j, k)$.
- The square matrix which has all the elements equal to 0 except the elements of the main diagonal, which are all 1 is called identity matrix.
- Translation is a process of changing the position of an object in a straight-line path from one co-ordinate location to another.
- A scaling transformation changes the size of an object.
- In design and picture formation process, many times we may require to perform translation, rotations, and scaling to fit the picture components into their proper positions.

- The homogeneous coordinates for translation are given as, $T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & 1 \end{bmatrix}$

- The homogeneous coordinates for rotation are given as, $R = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- The homogeneous coordinate for scaling are given as, $S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

References


Recommended Reading


Self Assessment

1. Which of the following statements is true?
   a. Matrix is a single dimensional array of number.
   b. Matrix is a two dimensional array of number.
   c. Matrix is a multidimensional array of number.
   d. Matrix is not a two dimensional array of number.

2. ________ involves simple products and sums of the matrix elements.
   a. Matrix multiplication
   b. Matrix addition
   c. Matrix subtraction
   d. Matrix division

3. Give a generalised formula for matrix multiplication.
   a. \[ C(i, k) = \sum_{j} A(i, j) B(j, k) \]
   b. \[ C(i, k) = \sum_{j} B(j, k) \]
   c. \[ C(i, k) = \sum_{j} A(i, j) B(j, k) \]
   d. \[ C(i, k) = \sum_{j} A(i, j) \]

4. What is the square matrix which has all the elements equal to 0, except the elements of the main diagonal, which are all 1 called as?
   a. Matrix
   b. Matrix multiplication
   c. Translation vector
   d. Identity matrix

5. By what is the identity matrix denoted?
   a. I
   b. A
   c. M
   d. O

6. Which of the following statements is true?
   a. The translation distance pair \((t_x, t_y)\) is called identity matrix.
   b. The translation distance pair \((S_x, S_y)\) is called a translation vector or shift vector.
   c. The translation distance pair \((t_x, t_y)\) is called a translation vector or shift vector.
   d. The translation distance pair \((t_x, t_y)\) is not called a translation vector or shift vector.

7. Which of the following statements is true?
   a. A scaling transformation changes the shape of an object.
   b. A scaling transformation changes the size of an object.
   c. A scaling transformation does not change the size of an object.
   d. A scaling transformation changes the angle of an object.
8. Which of the following statements is true?
   a. The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformations, one after the other.
   b. The basic purpose of composing transformations is to lose efficiency by applying a single composed transformation to a point, rather than applying a series of transformations, one after the other.
   c. The basic purpose of composing transformations is to gain efficiency by applying a double composed transformation to a point, rather than applying a series of transformations, one after the other.
   d. The basic purpose of composing transformations is to lose efficiency by applying a single composed transformation to a point, rather than applying a double of transformations, one after the other.

9. What is the process of changing the position of an object in a straight-line path from one co-ordinate location to another called as?
   a. Scaling
   b. Transformation
   c. Translation
   d. Rotation

10. The homogeneous coordinates for translation is given as, ________.
    a. \[
    T = \begin{bmatrix}
    0 & 0 & 0 \\
    0 & 1 & 0 \\
    t_x & t_y & 1
    \end{bmatrix}
    \]
    b. \[
    T = \begin{bmatrix}
    1 & 0 & 0 \\
    0 & 1 & 0 \\
    t_x & t_y & 1
    \end{bmatrix}
    \]
    c. \[
    T = \begin{bmatrix}
    1 & 0 & 0 \\
    0 & 0 & 0 \\
    t_x & t_y & 1
    \end{bmatrix}
    \]
    d. \[
    T = \begin{bmatrix}
    1 & 1 & 0 \\
    0 & 1 & 0 \\
    t_x & t_y & 1
    \end{bmatrix}
    \]
Chapter V
Windowing and Clipping

Aim
The aim of this chapter is to:

• explain windowing and clipping
• elucidate coordinate reference frame
• explicate polygon clipping

Objectives
The objectives of this chapter are to:

• explain normalised coordinates
• explicate Sutherland-Hodgeman polygon clipping algorithm
• analyse window to viewport coordinate transformation

Learning outcome
At the end of this chapter, the student will be able to:

• understand two dimensional viewing functions
• define viewing transformations
• recognise the various types of clipping algorithms
5.1 Introduction
A graphics package allows us to specify which part of a defined picture is to be displayed and where that part is to be displayed on the display device. The package also provides the use of the scaling, translation and rotation techniques to generate a variety of different views of a single picture. Different views of a picture can be generated by applying the appropriate scaling and translation. We have to identify the visible part of the picture for inclusion in the display image while doing this. This selection process is not straightforward. Certain lines may lie partly inside or outside the visible portion of the picture. These lines cannot be omitted entirely from the display image because the image would become inaccurate. (Refer to the figure below). The process of selecting and viewing the picture with different views is called windowing. The process which divides each element of the picture into its visible and invisible portions, allowing the invisible portion to be discarded is called clipping.

![Fig. 5.1 Display images](image)

5.2 Viewing Transformation
- The picture is stored in the computer memory using any convenient Cartesian coordinate system, referred to as World Coordinate System (WCS).
- However, when the picture is displayed on the display device, it is measured in physical device coordinate system (PDCS) corresponding to the display device.
- Displaying an image of a picture involves mapping the coordinates of the points and lines.
- The mapping of coordinates is achieved with the use of coordinate transformation known as viewing transformation.
- Sometimes, the two dimensional viewing transformation is simply referred to as the window to view port transformation or the windowing transformation.
- The viewing transformation which maps picture coordinates in the WCS to display coordinates in PDCS is performed by:
  - converting world coordinates to viewing coordinates
  - normalising viewing coordinates
  - converting normalised viewing coordinates to device coordinates

![Fig. 5.2 Two dimensional viewing transformation pipeline](image)
• World coordinate system (WCS) is infinite in extent and the device display area is finite.
• Therefore, to perform a viewing transformation we select a finite world coordinate area for display called a window.
• An area on a device to which a window is mapped is called as a viewport.
• The window defines what is to be viewed, whereas the viewport defines where it is to be displayed. (Refer to the figure below.)

![Fig. 5.3 Window and viewport](image)

- The above figure shows the two dimensional viewing pipeline.
- The window defined in world coordinate is first transformed into the viewport coordinate.
- For this viewing coordinate reference frame is used.
- It provides a method for setting up arbitrary orientations for rectangular windows.
- Once the viewing reference frame is established, we can transform descriptions in world coordinates for viewing coordinates.
- Further, we define a viewport in normalised coordinates.

### 5.2.1 Normalised Coordinates

- Different display devices may have different screen sizes as measured in pixels.
- As the resolution of the screen increases, the size of the screen in pixels increases.
- The picture is displayed large in size on the low resolution screen when it is defined in the pixel values, and it is displayed small in size on the high resolution screen as shown in the figure below.

![Fig. 5.4 Picture defined in pixels](image)
To avoid this and to make our programs to be device independent, picture coordinates should be defined in some units other than pixels and use the interpreter to convert these coordinates to appropriate pixel values for the particular display device. The device independent units are called the normalised device coordinates.

In these units, the screen measures 1 unit wide and 1 unit length as shown in the fig. 5.5. The interpreter uses simple linear formula to convert normalise device coordinates to the actual device coordinates.

\[
x = x_n \times X_w \quad \text{(i)}
\]
\[
y = y_n \times Y_v \quad \text{(ii)}
\]

where,

\[
x : \text{Actual device x coordinate}
\]
\[
y : \text{Actual device y coordinate}
\]
\[
x_n : \text{Normalised x coordinate}
\]
\[
y_n : \text{Normalised y coordinate}
\]
\[
X_w : \text{Width of actual screen in pixels}
\]
\[
Y_v : \text{Height of actual screen in pixels}
\]

The transformations which map the viewing coordinate to normalised device coordinate are called normalisation transformation.

It involves scaling of x and y. Thus, it is also referred to as, scaling transformation.

### 5.3 Window to Viewport Coordinate Transformation

- Once the window coordinates are transferred to viewing coordinates we choose the window extent in viewing coordinates and select the viewport limits in normalised coordinates.
- Object descriptions are then transferred to normalised device coordinates. In this transformation, the relative placement of the object in the normalised coordinates is same as in the viewing coordinates.

### 5.4 Workstation Transformation

- The transformation of object description from normalised coordinates to device coordinates is called workstation transformation.
- The workstation transformation is accomplished by selecting a window area in normalised space and a viewport area in the coordinates of the display devices.
- By using workstation transformation, we can partition a view, so that different parts of normalised space can be displayed on different output devices.

### 5.5 Two Dimensional Viewing Functions

- A Programmer’s Hierarchical Interactive Graphics Standard (PHIGS) is a package which provides functions for controlling graphical output and input.
- It is used by application programs as a standard interface to graphics devices.

\[
\text{evaluateViewOrientationMatrix} (X_{0v}, Y_{0v}, X_{uv}, Y_{uv}, \text{error}, \text{ViewMatrix})
\]

- $X_{0v}, Y_{0v}$: Coordinates of viewing origin.
- $X_{uv}, Y_{uv}$: World coordinates positions for the view up vectors.
- error: error code in integer generated if there is an error in input parameters.
- ViewMatrix: Matrix for the world to viewing transformation.
This function generates ViewMatrix for the world to viewing transformation if there is no error in input parameters.

\[
\text{evaluateViewMappingMatrix} (x_{\text{wmin}}, x_{\text{wmax}}, y_{\text{wmin}}, y_{\text{wmax}}, x_{\text{vmin}}, x_{\text{vmax}}, y_{\text{vmin}}, y_{\text{vmax}}, \text{error}, \text{view MappingMatrix})
\]

\[
x_{\text{wmin}}, x_{\text{wmax}}, y_{\text{wmin}}, y_{\text{wmax}} : \text{window limits in viewing coordinates}
\]
\[
x_{\text{vmin}}, x_{\text{vmax}}, y_{\text{vmin}}, y_{\text{vmax}} : \text{viewport limits in normalised coordinates}
\]

Thus, functions set up the elements of a window to viewport mapping matrix.

SetViewRepresentations (ws, ViewIndex, ViewMappingMatrix, XClipmin, XClipmax, Yclipmin, YClipmax, Clipxy)

\(ws\) : Output device (workstation)

\(\text{ViewIndex}\) : Integer identifier for window-viewport pair. Therefore, it refers viewMatrix and corresponding viewMappingMatrix.

Clipping limits: XClipmin, XClipmax, YClipmin, YClipmax.

\(\text{Clipxy}\) : It is assigned with either the value noclip or the value chip. This allows us to turn off clipping if we want to view the parts of the scene outside the viewport. We can also select noclip to speed up processing when we know that all of the scene is included in the viewport limits.

SetViewIndex (viewIndex)

This function selects a particular set of options from the viewing table. This view-index selection is applicable to subsequently specified output primitives.

SetWorkstationWindow (ws, xwsWindmin, xwsWindmax, ywsWindmin, ywsWindmax)

SetWorkstationViewport (ws, xwsVPortmin, xwsVPortmax, ywsVPortmin, ywsVPortmax)

\(Ws\) : Workstation number

\(xws\text{Windmin}, xws\text{VPortmax}, yws\text{Windmin}, yws\text{VPortmax}\) : Viewport limits in integer specified in normalised coordinates.

5.6 2D Clipping

- The procedure that identifies the portions of a picture that are either inside or outside of a specified region of space is referred to as clipping.
- The region against which an object is to be clipped is called a clip window or clipping window.
- It usually is in a rectangular shape.
- The clipping algorithm determines which points, lines or portions of lines lie within the clipping window. These points, lines or portions of lines are retained for display. All others are discarded.

5.6.1 Point Clipping

The points are said to be interior to the clipping window if:

\[
x_{\text{wmin}} \leq x \leq x_{\text{wmax}} \quad \text{and} \quad y_{\text{wmin}} \leq y \leq y_{\text{wmax}}
\]

The equal sign indicates that points on the window boundary which are included within the window.
5.6.2 Line Clipping

- The lines are said to be interior to the clipping window and hence, visible if both end points are interior to the window.
- However, if both end points of a line are exterior to the window, the line is not necessarily completely exterior to the window.
- If both end points of a line are completely to the right of, completely above, or completely below the window, then the line is completely exterior to the window and hence invisible.
- The lines which cross one or more clipping boundaries require calculation of multiple intersection points to decide the visible portion of them.
- To minimize the intersection calculations and to increase the efficiency of the clipping algorithm, initially, completely visible and invisible lines are identified and then the intersection points are calculated for remaining lines.
- There are many line clipping algorithms.

5.7 Cohen-Sutherland Subdivision Line Clipping Algorithm

- This is one of the oldest and most popular line clipping algorithm developed by Dan Cohen and Ivan Sutherland.
- To speed up the processing this algorithm performs initial tests that reduce the number of intersections that must be calculated.
- This algorithm uses a four digit (bit) code to indicate which of nine regions contain the end point of line.
- The four bit codes are called region codes or outcodes.
- These codes identify the location of the point relative to the boundaries of the clipping rectangle as shown in the fig. 5.5

![Fig. 5.5 Four-bit codes for nine regions](image)

- Each bit position in the region code is used to indicate one of the four relative coordinate positions of the point with respect to the clipping window: to the left, right, top or bottom. The rightmost bit is the first bit and the bits are set to 1 based on the following scheme:
  - Set Bit 1 – if the end point is to the left of the window
  - Set Bit 2 – if the end point is to the right of the window
  - Set Bit 3 – if the end point is below the window
  - Set Bit 4 – if the end point is above the window
- Otherwise, the bit is set to zero.
5.7.1 ‘C’ Code for Sutherland and Cohen Subdivision Line Clipping Algorithm

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<math.h>
#include<graphics.h>

/* Defining structure for nd point of line */
typedef struct coordinate
{
    int x, y;
    char code [4];
} PT;

void drawwindow ( );
void drawline (PT P1, PT p2, int c1);
PT setcode (PT p);
int visibility (PT p1, PT p2);
PT reserendpt (PT p1, PT p2);

main ( )
{
    int gd=DETECT, gm, v;
    PT p1, p2, ptemp;
    initgraph (&gd, &gm, " ");
    cleardevice ( );
    printf ("\n\nENTER END-POINT 1 (x, y): ");
    scanf ("%d, %d", &p2.x, &p2.y);
    cleardevice ( );
    drawwindow ( );
    getch ( );
    drawline (p1, p2, 15);
    getc ( );
p1=set code (p1);
p2=set code (p2);
v=visibility (p1, p2);
switch (v)
{
    case 0: cleardevice ( ); /* Line completely visible */
        drawwindow ( );
        drawline (p1, p2, 15);
        break;
    case 1: cleardevice ( ); /* Line completely invisible */
        drawwindow ( );
        break;
    case 2: cleardevice ( ); /* Line partly visible */
        p1=resetendpt (p1, p2);
```
p2 = resetendpt(p2, p1);

drawwindow();

drawline(p1, p2, 15);

break;
}
getch();
closegraph();
return(0);

/* Function to draw window */
void drawwindow()
{
    setcolor(RED);
    line(150, 100, 450, 100);
    line(450, 100, 450, 350);
    line(450, 350, 150, 350);
    line(150, 350, 150, 100);
}

/* Function to draw line between two points */
void drawline(PT p1, PT p2, int c1)
{
    setcolor(c1);
    line(p1.x, p1.y, p2.x, p2.y);
}

/* Function to set code of the coordinates */
PT setcode(PT p)
{
    PT ptemp;
    if (p.y<100)
        ptemp.code[0] = '1'; /* TOP */
    else
        ptemp.code[0] = '0';
    if (p.y>350)
        ptemp.code[1] = '1'; /* BOTTOM */
    else
        ptemp.code[1] = '0';
    if (p.x>450)
        ptemp.code[2] = '1'; /* RIGHT */
    else
        ptemp.code[2] = '0';
    if (p.x<150) /* LEFT */
        ptemp.x = p.x;
    ptemp.y = p.y;
    return(ptemp);
}

/* Function to determine visibility of line */
int visibility (PT p1, PT p2)
{
    int i, flag=0;
    for (i = 0; i<4; i++)
    {
        if ((p1.code[i] != '0') || (p2.code[i] != '0'))
            flag=1;
    }
    if (flag == 0)
        return (0);
    for (i=0; i<4; i++)
    {
        if ((p1.code[1] == p1.code[i]) && (p1.code[i] == '1'))
            flag=0;
    }
    if (flag == 0)
        return (1);
    return (2);
}

/* Function to find new end points
------------------------------------------*/

PT resetendpt (PT p1, PT p2)
{
    PT temp;
    int x, y, i;
    float m, k;
    if (p1.code[3] == '1') /* Cutting LEFT Edge*/
        x=150;
    if (p1.code[2] == '1') /* Cutting RIGHT Edge */
        x=450;
    {
        m=(p2.y-p1.y)/(p2.x-p1.x);
        k=(p1.y+(m* (x-p1.x)));
        temp.y=k;
        temp.x=x;
        for (i=0; i<4; i++)
            if (temp.y<=350&&temp.y>=100)
                return (temp);
    }
    if (p1.code[0] == '1') /* Cutting TOP Edge */
        y=100;
    if (p1.code[1] == '1') /* Cutting BOTTOM Edge */
        y=350;
    if ((p1.code[0] == '1') || (p1.code[1] == '1'))
    {
        m=(p2.y-p1.y) / (p2.x-p1.x);
        k=(float) p1.x+(float) (y-p1.y) / m;
        temp.x=k;
        temp.y=y;
        for (i=0; i<4; i++)
            temp.code[i]=p1.code[i];
        return (temp);
5.8 Polygon Clipping

- A polygon is nothing but the collection of lines.
- Therefore, we might think that line clipping algorithm can be used directly for polygon clipping.
- However, when a closed polygon is clipped as a collection of lines with line clipping algorithm, the original closed polygon becomes one or more open polygon or discrete lines as shown in the figure below.
- Thus, we need to modify the line clipping algorithm to clip polygons.

![Polygon clipping](image)

Fig. 5.6 Polygon clipping done by line clipping algorithm

5.9 Sutherland-Hodgeman Polygon Clipping Algorithm

- A polygon can be clipped by processing its boundary as a whole against each window edge.
- This is achieved by processing all polygon vertices against each clip rectangle boundary in turn.
- Beginning with the original set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices.
- The new set of vertices could then be successively passed to a right boundary clipper, a top boundary clipper and a bottom boundary clipper.
- At each step, a new set of polygon vertices is generated and passed to the next window boundary clipper.
- This is the fundamental idea used in the Sutherland-Hodgeman algorithm.

5.9.1 ‘C’ Code for Sutherland-Hodgeman Polygon Clipping Algorithm

```c
#include<stdio>
#include<graphics.h>
#include<math.h>
typedef struct
{
  float x;
  float y;
  }PT;
int n;
maint()
{
  int i, j, gd, gm;
```
pt d, p1, p2, p[20], p1l, pp[20];
detectgraph (&gd, &gm);
initgraph (&gd, &gm, “”);

/* Read coordinates of clipping window */
printf (“Enter coordinates (left, top) of point1 : “);
scanf (“%f, %f”, &p1.x, &p1.y);
printf (“Enter coordinates (right, bottom) of point2 : “)
scanf (“%f, %f”, &p2.x, &p2.y);

/* Enter the number of vertex */
printf (“Enter the number of vertex : “);
scanf (“%d”, n);

/* Read vertex coordinates of clipping window */
for (i=o; i<n; i++)
{
    printf (“Enter coordinates of vertex%d : “, i+1):
    scanf (“%f, %f”, &p[i].x, &p[i].y);
    p[i].x = p[0].x;
    p[i].y = p[0];
    cleardevice ();
    drawpolygon ();
    rectangle (p1.x, p1.y, p2.x, p2.y);
    getch ();
    left (p1, p, pp);
    right (p2, p, pp);
    top (p1, p, pp);
    bottom (p2, p, pp)
cleardevice ( );
    rectangle (p1.x, p1.y, p2.x, p2.y);
    rectangle (p1.x, p1.y, p2.x, p2.y);
    drawpolygon (p, n);
    getch ();
closegraph ( );
}

left (PT p1,PT p[20], PT pp[20])
{
    int i, j=0;
    for (i=0; i<n; i++)
    {
        if (p[i].x < p1.x && p[i+1].x >= p1.x)
        {
            if (p[i+1].x=p[i].x!=0)
            {
                pp[j].y=(p[i+1].y-p[i].y) / (p[i+1].x-p[i].x) * (p1.x-p[i].x)+p[i].y;
            }
        else
            
        
    }
```c
{ 
    pp[j].y = p[i].y;
}
pp[j].x = p1.x;
    j++;
pp[j].x = p[i+1].x;
pp[j].y = p[i+1].y;
    j++;
}
if (p[i].x > p1.x && p[i+1].x >= p1.x)
{ 
    pp[j].y = p[i+1].y;
    pp[j].x = p[i+1].x;
    j++;
}
if (p[i].x > p1.x && p[i+1].x <= p1.x)
{ 
    if (p[i+1].x-p[i].x!=0)
    { 
        pp[j].y = (p[i+1].y-p[i].y / (p[i+1].x-p[i].x))*
        = (p1.x-p[i].x)+p[i].y;
        }
    else
    { 
        pp[j].y = p[i].y;
    }
    pp[j].x = p1.x;
    j++;
}
for (i=0; i<j; i++)
{ 
    p[i].x = pp[i].x;
    p[i].y = pp[i].y;
    n=j;
}
right (PT p2, PT p[20], PT pp[20])
{ 
    int i, j=0;
    for (i=0; i<n; i++)
    { 
        if (p[i].x > p2.x && p[i+1].x <= p2.x)
        { 
            if (p[i+1].x-p[i].x!=0)
```
{ }  
pp[j].y = 9p[i+1].y-p[i].y) / (p[i+1].x-p[i].x)  
= (p2.x-p[i].x) + p[i].y;  
}  
else  
{  
pp[j].y = p[i].y;  
}  
pp[j].x = p2.x;  
j++;  
pp[j].x=p[i+1].x;  
pp[j].y+p[i+1].y;  
j++;  
}  
if (p[i].x < p2.x && p[i+1].x <= p2.x)  
{  
if (p[i+1].x-p[i].x!=0)  
{  
pp[j].y = (p[i+1].y-p[i].y) / (p[i+1].x-p[i].x)*  
(p2.x-p[i].x)+p[i].y;  
}  
else  
{  
pp[j].y = p[i].y;  
}  
pp[j].x = p2.x;  
j++;  
}  
for (i=0; i<j; i++)  
{  
p[i].x = pp[i].x;  
p[i].y = pp[i].y;  
}  
p[i].x = pp[0].x;  
p[i].y = pp[0].y;  
n=j;  
}  


top (PT p1, PT p[20], PT pp[20])  
{  
int i, j=0;  
for (i=0; i<n; i++)  
{  
if (p[i].y < p1.y && p[i+1].y >= p1.y)  
{  
if (p[i+1].y-p[i].y!=0)  
{  
pp[j].x = (p[i+1].x-p[i].x) / (p[i+1].y-p[i].y)*  
(p1.y-p[i].y)+p[i].x;  
}  
}  
}  
}
else
    }
pp[j].x = p[i].x;
}
pp[j].y = p1.y;
j++;
pp[j].x=p[i+1].x;
pp[j].y=p[i+1].y;
j++;
}
if (p[i].y > p1.y && p[i+1].y >= p1.y)
{
    pp[j].y = p[i+1].y;
    pp[j].x = p[i+1].x;
j++;
}
if (p[i].y > p1.y && p[i+1].y <= p1.y)
{
    if (p[i+1].y-p[i].y!=0)
    {
        pp[j].x = (p[i+1].x-p[i].x) / (p[i+1].y-p[i].y)*
(p1.y-p[i].y)+p[i].x;
    }
    else
    {
        pp[j].x = p[i].x;
    }
    pp[j].y = p1.y;
j++;
}
for (i=0; i<j; i++)
{
    p[i].x = pp[i].x;
p[i].y = pp[i].y;
}
bottom (PT p2, PT p[20], PT pp[20])
{
    int i, j=0;
    for (i=0; i<n; i++)
    {
        if (p[i].y > p2.y && p[i+1].y <= p2.y)
        {
            if (p[i+1].y-p[i].y!=0)
            {
                pp[j].x = (p[i+1].x-p[i].x) / (p[i+1].y-p[i].y)*
(p2.y-p[i].y)+p[i].x;
            }
            else
            {
            }
        }
    }
}
pp[j].x = p[i].x;
}
pp[j].y = p2.y;
j++;
pp[j].x=p[i+1].x;
pp[j].y=p[i+1].y;
j++;
}
if (p[i].y < p2.y && p[i+1].y <= p2.y)
{
pp[j].y = p[i+1].y;
pp[j].x = p[i+1].x;
j++;
}
if (p[i].y < p2.y && p[i+1].y >=p2.y)
{
    if (p[i+1].y –p[i].y!=0)
    {
        pp[j].x = (p[i+1].x-p[i].x) / (p[i+1].y-p[i].y)*
        (p2.y-p[i].y)+p[i].x;
    }
    else
    {
        pp[j].x =p[i].x;
    }
pp[j].y =p2.y;
j++;
}
for )i=o; i<j; i++)
{
p[i].x = pp[i].x;
p[i].y = pp[i].y;
}
p[i].x = pp[0].x;
p[i].y = pp[0].y;
n=j;
}
drawpolygon (PT x[20], int n)
{
    int i;
for (i=0; i<n;i++)
    {
        line (x[i].x,x[i].y,x[i+1].x,x[i+1].y);
    }
    line (x[i].x,x[i].y,x[0].x,x[0].y);
}
Summary

- A graphics package allows us to specify which part of a defined picture is to be displayed and where that part is to be displayed on the display device.
- The picture is stored in the computer memory using any convenient Cartesian coordinate system, referred to as world coordinate system (WCS).
- When the picture is displayed on the display device it is measured in physical device coordinate system (PDCS) corresponding to the display device.
- World coordinate system (WCS) is infinite in extent and the device display area is finite.
- An area on a device to which a window is mapped is called as a viewport.
- As the resolution of the screen increases, the size of the screen in pixels increases.
- The matrix used for this composite transformation is given by: $M_{WCS} = T \cdot R$
- The transformation of object description from normalized coordinates to device coordinates is called workstation transformation.
- A Programmer’s Hierarchical Interactive Graphics Standard (PHIGS) is a package which provides functions for controlling graphical output and input.
- The procedure that identifies the portions of a picture that are either inside or outside of a specified region of space is referred to as clipping.
- The lines are said to be interior to the clipping window and hence visible, if both end points are interior to the window.

References


Recommended Reading

Self Assessment

1. The process of selecting and viewing the picture with different views is called ________.
   a. clipping
   b. windowing
   c. viewing
   d. mapping

2. The picture is stored in the computer memory using any convenient Cartesian coordinate system, referred to as ________.
   a. world coordinate system (WCS)
   b. physical device coordinate system (PDCS)
   c. windowing
   d. clipping

3. Displaying an image of a picture involves _________ the coordinates of the points and lines.
   a. clipping
   b. windowing
   c. mapping
   d. viewing

4. Which of the following statements is true?
   a. World coordinate system (WCS) is finite in extent and the device display area is finite.
   b. World coordinate system (WCS) is infinite in extent and the device display area is infinite.
   c. World coordinate system (WCS) is not infinite in extent and the device display area is not finite.
   d. World coordinate system (WCS) is infinite in extent and the device display area is finite.

5. What is the area on a device to which a window is mapped called?
   a. Viewport
   b. Window
   c. Monitor
   d. Display

6. Which of the following statements is true?
   a. For the viewing coordinate reference frame is not used.
   b. For the viewing coordinate reference frame is used.
   c. For the viewing coordinate reference frame buffer is used.
   d. For the viewing coordinate reference viewport is used.

7. What is the transformation which maps the viewing coordinate to normalised device coordinate called?
   a. Scaling
   b. Translation
   c. Normalisation transformation
   d. Workstation transformation
8. Give the matrix used for the composite transformation.
   a. $M_{WC,VC} = T / R$
   b. $M_{WC,VC} = T + R$
   c. $M_{WC,VC} = T * R$
   d. $M_{WC,VC} = T \cdot R$

9. Which of the following statements is true?
   a. The transformation of object description from normalised coordinates to device coordinates is called workstation transformation.
   b. The transformation of object description from normalised coordinates to device coordinates is called Normalisation transformation.
   c. The transformation of object description from normalised coordinates to device coordinates is called Translation transformation.
   d. The transformation of object description from normalised coordinates to device coordinates is called Scaling transformation.

10. Which of the following statements is true?
    a. A world coordinate system (WCS) is a package which provides functions for controlling graphical output and input.
    b. A Programmer’s Hierarchical Interactive Graphics Standard (PHIGS) is a package which does not provide functions for controlling graphical output and input.
    c. A Programmer’s Hierarchical Interactive Graphics Standard (PHIGS) is a package which provides functions for controlling graphical output and input.
    d. A physical device coordinate system (PDCS) is a package which provides functions for controlling graphical output and input.
Chapter VI
3-D Transformations

Aim
The aim of this chapter is to:

• explain 3d geometry
• elucidate the techniques to achieve realism
• explicate 3d transformations

Objectives
The objectives of this chapter are to:

• explain 3D primitive
• analyse various algorithms
• define 3D viewing

Learning outcome
At the end of this chapter, you will be able to:

• recognise the different techniques of achieving realism
• understand translation, scaling, rotation and reflection in 3D transformation
• describe the different types of parallel projections
6.1 Introduction

Some graphics applications are two-dimensional, such as charts and graphics, certain maps and so on. However, to create a realistic picture, scene or model we need to represent it in 3-D graphics. The creation of realistic pictures is an important task in fields such as simulation, design, entertainment, advertising, research, education, command and control.

To create a realistic picture, we must process the scene or picture through viewing coordinate transformations and projection routine that transform three-dimensional viewing coordinates onto two-dimensional device coordinates.

We must identify the visible parts of the scene for a selected view and we must apply the surface rendering algorithm to create a realistic scene or picture.

6.2 3-D Geometry

- We need additional third axis in 3-D geometry to specify the coordinates of any point.
- Here, the three axes are arranged at right angles to each other. Label the width and height axes x and y respectively, to correspond to the two dimensional case.
- The third axis, for depth, is named the z-axis.
- There are two types of three dimensional reference systems according to the orientation for the coordinate axes: Right handed system and left handed system.
- The right handed system uses the right hand thumb to point the positive z direction when we imagine the fingers curling from positive x axis to the positive y axis (through 90°) grasping the z axis.
- In left handed Cartesian coordinate system, the left hand thumb is used to point the positive z direction when we imagine the figures of the left hand curl from the positive x axis to the positive y axis (through 90°) to grasp the z axis.
- However, we adopt right handed system for computer graphics.
- Now, with this coordinate system, we can specify any point in space by the order triple (x, y, z).
- In 3-D geometry, the line is specified by a pair of equations.

\[
\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1} \quad \text{and} \quad \frac{x-x_1}{x_2-x_1} = \frac{z-z_1}{z_2-z_1}
\]

Where \((x_1, y_1, z_1)\) and \((x_2, y_2, z_2)\) are the two points which specify the line. A plane is specified by a single equation of the form.

\[
Ax + By + Cz + D = 0
\]

6.3 3-D Primitive

Like 2-D primitive, 3-D primitive has to draw points, lines and plane in three dimensions. Here, we have to provide the three coordinate specification instead of two.

6.3.1 Algorithm 1: 3-D Absolute Move

MOVE_ABS_3-D (X, Y, Z)

Arguments X, Y, Z are the coordinates of points to move the pen to
Global DF_CUR_X, DF_CUR_Y, DF_CUR_Z; current pen position coordinates
BEGIN
DF_CUR_X ← X;
DF_CUR_Y ← Y;
DF_CUR_Z ← Z;
DF_ENTER (1);
RETURN;
END;
6.3.2 Algorithm 2: 3-D Relative Move

MOVE_REL_3-D (DX, DY, DZ)
Arguments  DX, DY, DZ specify the changes to be made in current pen position
Global  DF_CUR_X, DF_CUR_Y, DF_CUR_Z; current pen position coordinates
BEGIN
    DF_CUR_X ← DF_CUR_X + DX;
    DF_CUR_Y ← DF_CUR_Y + DY;
    DF_CUR_Z ← DF_CUR_Z + DZ;
    DF_ENTER (1);
RETURN;
END;

6.3.3 Algorithm 3: 3-D Absolute Line Drawing Routine

LINE_ABS_3-D (X, Y, Z)
Arguments  X, Y, Z are the coordinates of point to draw the line to
Global  DF_CUR_X, DF_CUR_Y, DF_CUR_Z; current pen position coordinates
BEGIN
    DF_CUR_X ← X;
    DF_CUR_Y ← Y;
    DF_CUR_Z ← Z;
    DF_ENTER (2);
RETURN;
END;

6.3.4 Algorithm 4: 3-D Relative Line Drawing Routine

LINE_REL_3-D (DX, DY, DZ)
Arguments  DX, DY, DZ specify the displacement over which a line is to be drawn
Global  DF_CUR_X, DF_CUR_Y, DF_CUR_Z; current pen position coordinates
BEGIN
    DF_CUR_X ← DF_CUR_X + DX;
    DF_CUR_Y ← DF_CUR_Y + DY;
    DF_CUR_Z ← DF_CUR_Z + DZ;
    DF_ENTER (2);
RETURN;
END;

6.3.5 Algorithm 5: Absolute Polygon Drawing Routine

POLYGON_ABS_3-D (AX, AY, AZ, N)
Arguments  N specifies number of polygon sides AX, AY, AZ arrays of the coordinates of the vertices
Global  DF_CUR_X, DF_CUR_Y, DF_CUR_Z; current pen position coordinates
BEGIN
    IF N < 3 THEN RETURN ERROR; less number of sides
    DF_CUR_X ← AX [N];
    DF_CUR_Y ← AY [N];
    DF_CUR_Z ← AZ [N];
    DF_ENTER (N);
    FOR J = 1 TO N DO LINE_ABS_3D (AX [I], AY [I], AZ [I]);
RETURN;
END;
6.3.6 Algorithm 6: 3-D Relative Polygon Drawing Routine

Arguments  N specifies the number of polygon sides AX, AY, AZ arrays of displacement for the polygon sides

Global   DF_CUR_X, DF_CUR_Y, DF_CUR_Z; current pen position coordinates

BEGIN
If N < # THEN RETURN ERROR; less number of sides
DF_CUR_X ← DF_CUR_X + AX [1]; [Move to
DF_CUR_Y ← DF_CUR_Y + AY [1]; starting
DF_CUR_Z ← DF_CUR_Z + AZ [1]; vertex ]
TEMPX ← DF_CUR_X; [Save starting
TEMPY ← DF_CUR_Y; vertex for
TEMPZ ← DF_CUR_Z; closing the polygon ]
DF.Enter (N);
FOR I=2 TO N DO LINE_REL_3D (AX [I], AY [I], AZ [I]);
LINE_ABS_3D (TEMPX, TEMPY, TEMPZ); close the polygon
RETURN;
END;

6.4 Techniques To Achieve Realism

• To obtain the realistic picture, we have to first setup a coordinate reference for camera.
• This coordinate reference defines the position and orientation for the plane of the camera.
• This plane must be used to display a view of the objects in the scene.
• To display a view of the object, its description has to be transferred to the camera reference coordinates and
projected onto the selected display plane.
• Then we can display object in wireframe form or we can apply lighting and surface rendering techniques to
shade the visible surfaces.

6.4.1 Depth Cueing

• To create realistic image, the depth information is important so that we can easily identify, for a particular
viewing direction, which is the front and which is the back of displayed objects.
• The depth of an object can be represented by the intensity of the image.
• The parts of the objects closest to the viewing position are displayed with highest intensities, and objects farther
away are displayed with decreasing intensities. This effect is known as depth cueing.
• The figure below shows a wireframe object displayed with depth cueing, so that the intensity of lines decreases
from the front to the back of the object.

![Fig. 6.1 Depth cueing](image-url)
6.4.2 Surface Rendering
- Surface rendering involves setting the surface intensity of objects according to the lighting conditions in the scene and according to assigned surface characteristics.
- The lighting conditions specify the intensity and positions of light sources and the general background illumination required for a scene.
- On the other hand, the surface characteristics of objects specify the degree of transparency and smoothness or roughness of the surface.
- Usually, the surface rendering methods are combined with perspective and visible-surface identification to generate a high degree of realism in a displayed scene.

6.4.3 Stereoscopic Views
- Another method for adding a sense of realism is to display objects using stereoscopic views.
- Stereoscopic devices present two views of a scene: one for the left eye and the other for the right eye.
- The two views are generated by selecting viewing positions that correspond to the two eye positions of a viewer.
- These two views are alternately flashed on a single screen about 20 times per second, in synchronisation with the monitor refresh cycle.

6.4.4 Material Properties
- Realism is further enhanced if the material properties of each object are taken into account when its shading is determined.
- Some materials are dull and disperse reflected light about equally in all directions, like a piece of chalk.
- On the other hand, some materials are shiny and reflect light only in certain directions relative to the viewer and light source, like a mirror.

6.4.5 Shadows
- We can further explain realism by reproducing shadows casted by objects on one another.
- Shadows enhance realism and provide additional depth cues.

6.5 Three Dimensional Transformations
- Like two dimensional transformations, three dimensional transformations are formed by composing the basic transformations of translation, scaling and rotation.
- Each of these transformations can be represented as a matrix transformation with homogeneous coordinates.
- Therefore, any sequence of transformations can be represented as a single matrix, formed by combining the matrix for the individual transformations in the sequence.

6.5.1 Translation
- Three dimensional transformation matrix for translation with homogeneous coordinates is as given below. It specifies three coordinates with their own translation factor.

\[
T = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
t_x & t_y & t_z & 1
\end{bmatrix}
\]

\[
P' = P \cdot T
\]
Like two dimensional transformations, an object is translated in three dimensions by transforming each vertex of the object.

\[
[x \ y \ z \ 1] = [x \ y \ z \ 1] \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
t_x & t_y & t_z & 1
\end{bmatrix}
\]

\[
[x' \ y' \ z' \ 1] = [x + t_x \ y + t_y \ z + t_z \ 1]
\]

6.5.2 Scaling

• Three dimensional transformation matrix for scaling with homogeneous coordinates is as given below:

\[
S = \begin{bmatrix}
S_x & 0 & 0 & 0 \\
0 & S_y & 0 & 0 \\
0 & 0 & S_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

• It specifies three coordinates with their own scaling factor.
\[ P' = P \cdot S \]

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  1
\end{bmatrix} = \begin{bmatrix}
  x & y & z & 1
\end{bmatrix} \begin{bmatrix}
  S_x & 0 & 0 & 0 \\
  0 & S_y & 0 & 0 \\
  0 & 0 & S_z & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix} = \begin{bmatrix}
  x \cdot S_x & y \cdot S_y & z \cdot S_z & 1
\end{bmatrix}^{-1}
\]

- A scaling of an object with respect to a selected fixed position can be represented with the following transformation sequence:
  - translate the fixed point to the origin
  - scale the object
  - translate the fixed point back to its original position

### 6.5.3 Rotation

- Unlike the two dimensional rotation, where all transformations are carried out in xy plane, a three dimensional rotation can be specified around any line in space.
- Therefore, for three dimensional rotations we have to specify an axis of rotation about which the object is to be rotated along with the angle of rotation.
- The easier rotation axes to handle are those that are parallel to the coordinate axes. It is possible to combine the coordinate axis rotations to specify any general rotation

### 6.5.4 Reflection

- A three dimensional reflection can be performed relative to a deflected reflection axis or with respect to a selected reflection plane.
- The three dimensional reflection matrices are set up similarly to those for two dimensions.
- Reflections relative to a given axis are equivalent to 180° rotations about that axis.
- Reflections with respect to a plane are equivalent to 180° rotations in four dimensional spaces.

### 6.6 Three Dimensional Viewing

- The 3-D viewing process is inherently more complex than the 2-D viewing process.
- In two dimensional viewing, we have 2-D window and 2-D viewport and objects in the world coordinates are clipped against the window and then are transformed into the viewport for display.
- The complexity added in the three dimensional viewing is because of the added dimension and the fact that even though objects are three dimensional the display devices are only 2-D.
- The mismatch between 3-D objects and 2-D displays is compensated by introducing projections. The projections transform 3-D objects into 2-D projection plane.
- In 3-D viewing, we specify a view volume in the world coordinates using modelling transformation.
- The world coordinate positions of the objects are then converted into viewing coordinates by viewing transformations.
- The projection transformation is then used to convert 3-D description of objects in viewing coordinates to the 2-D projection coordinates.
- Finally, the workstation transformation transforms the projection coordinates into the device coordinates.
6.7 Projections

After converting the description of objects from world coordinates to viewing coordinates, we can project the three dimensional objects onto two dimensional view plane. There are two basic ways of projecting objects onto the view plane:

- Parallel projection
- Perspective projection

6.7.1 Parallel Projection

- In parallel projection, z coordinate is discarded and parallel lines from each vertex on the object are extended until they intersect the view plane.
- The point of intersection is the projection of the vertex.
- We connect the projected vertices by line segments which correspond to connections on the original object.

![Fig. 6.4 Parallel projection of an object to the view plane](image)

Fig. 6.4 shows that a parallel projection preserves proportions of objects but does not produce the realistic views.

6.7.2 Perspective Projection

- The perspective projection, on the other hand, produces realistic views but does not preserve relative proportions.
- In perspective projection, the lines of projection are not parallel.
- Instead, they all converge, at a single point called the centre of the projection or projection reference point.
- The object positions are transformed to the view plane along these converged projection lines and the projected view of an object is determined by calculating the intersection of the converged projection lines with the view plane.
6.7.3 Types of Parallel Projections

- Parallel projections are basically categorised into two types, depending on the relation between the direction of projection and the normal to the view plane.
- When the direction of the projection is normal (perpendicular) to the view plane, we have an orthographic parallel projection. Otherwise, we have an oblique parallel projection.

6.7.3.1 Orthographic Projection

- The most common types of orthographic projections are the front projection, top projection and side projection.
- In all these projections, the projection plane (view plane) is perpendicular to the principle axis.
- These projections are often used in engineering drawing to depict machine parts, buildings, etc.
- The orthographic projection can display more than one face of an object. Such an orthographic projection is called axonometric orthographic projection.
- It uses projection planes that are not normal to a principle axis.
- Parallelism of lines is preserved but angles are not.
- The most commonly used axonometric orthographic projection is the isometric projection.
- The isometric projection can be generated by aligning the view plane so that it intersects each coordinate axis in which the object is defined at the same distance from the origin.

6.7.3.2 Oblique Projection

- An oblique projection is obtained by projecting points along parallel lines that are not perpendicular to the projection plane.
- The view plane normal and the direction of projection are not the same.
- The oblique projections are further classified as the cavalier and cabinet projection.
- For the cavalier projection, the direction of projection makes a 45° angle with the view plane.
- As a result, the projection of a line perpendicular to the view plane has the same length as the line itself, in other words there is no foreshortening.
6.7.4 Types of Perspective Projections

- The perspective projection of any set of parallel lines that are not parallel to the projection plane converge to a vanishing point.
- The vanishing point for any set of lines that are parallel to one of the three principle axes of an object is referred to as a principle vanishing point or axis vanishing point.
- There are at most three such points, corresponding to the number of principle axes cut by the projection plane.
- The perspective projection is classified according to number of principle vanishing points in a projection: one point, two point or three point projections.
Summary

- Some graphics applications such as charts, graphics, certain maps are two-dimensional. However, to create a realistic picture, scene or model we need to represent it in 3-D graphics.
- We need additional third axis in 3-D geometry to specify the coordinates of any point.
- The third axis, for depth, is named the z-axis.
- There are two types of three dimensional reference systems according to the orientation for the coordinate axes: Right handed system and left handed system.
- In 3-D geometry, the line is specified by a pair of equations.
  \[
  \frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1} = \frac{z-z_1}{z_2-z_1}
  \]
- To display a view of the object, its description has to be transferred to the camera reference coordinates and projected onto the selected display plane.
- In parallel projection, z coordinate is discarded and parallel lines from each vertex on the object are extended until they intersect the view plane.
- The perspective projection on the other hand produces realistic views but does not preserve relative projections.
- To create realistic image, the depth information is important, so that we can easily identify, for a particular viewing direction, which is the front and which is the back of displayed objects.
- Surface rendering involves setting the surface intensity of objects according to the lighting conditions in the scene and according to assigned surface characteristics.
- Stereoscopic devices present two views of a scene: one for the left eye and the other for the right eye.
- A three dimensional reflection can be performed relative to a deflected reflection axis or with respect to a selected reflection plane.

References

- *Lecture - 8 3D Viewing* [Video online] Available at: <http://www.youtube.com/watch?v=k2ajK9uM0C0> [Accessed 21 June 2013].

Recommended Reading

Self Assessment

1. In ________, z coordinate is discarded and parallel lines from each vertex on the object are extended until they intersect the view plane.
   a. perspective projection
   b. parallel projection
   c. orthographic projection
   d. oblique projection

2. Which of the following statements is false?
   a. The easier rotation axes to handle are those that are parallel to the coordinate axes.
   b. The orthographic projection can display more than one face of an object.
   c. An oblique projection is obtained by projecting points along parallel lines that are not perpendicular to the projection plane.
   d. The orthographic projection cannot display more than one face of an object.

3. ________ enhance realism and provide additional depth cues.
   a. Material properties
   b. Surface rendering
   c. Shadows
   d. Reflections

4. In ________ the lines of projection are not parallel.
   a. perspective projection
   b. parallel projection
   c. orthographic projection
   d. oblique projection

5. Which of the following statements is true?
   a. The vanishing point for any set of lines that are perpendicular to one of the three principle axes of an object is referred to as a principle vanishing point or axis vanishing point.
   b. The vanishing point for any set of lines that are parallel to one of the three principle axes of an object is referred to as a principle vanishing point or axis vanishing point.
   c. The vanishing point for any set of lines that are parallel to two of the three principle axes of an object is referred to as a principle vanishing point or axis vanishing point.
   d. The vanishing point for any set of lines that are parallel to one of the three principle axes of an object is referred to as a vanishing point.

6. What produces realistic views but does not preserve relative projections?
   a. Orthographic projection
   b. Oblique projections
   c. Perspective projection
   d. Parallel projection
7. Which of the following statements is false?
   a. To create realistic image, the depth information is important so that we can easily identify, for a particular viewing direction, which is the front and which is the back of displayed objects.
   b. The depth of an object can be represented by the intensity of the image.
   c. For the cavalier projection, the direction of projection makes a 45° angle with the view plane.
   d. For the cavalier projection, the direction of projection makes a 90° angle with the view plane.

8. The __________ characteristics of objects specify the degree of transparency and smoothness or roughness of the surface.
   a. surface
   b. reflection
   c. shadow
   d. depth cue

9. The parts of the objects closest to the viewing position are displayed with highest intensities, and objects farther away are displayed with decreasing intensities. What is this effect known as?
   a. depth cueing
   b. perspective projection
   c. parallel projection
   d. oblique projection

10. Give the pair of equations to specify line in 3D geometry.

    \[ \frac{y-y_1}{y_2-y_1} = \frac{z-z_1}{z_2-z_1} \quad \text{and} \quad \frac{x-x_1}{x_2-x_1} = \frac{z-z_1}{z_2-z_1} \]
    a. \( x-x_1 \quad x_1-x_2 \) and \( y-y_1 \quad y_1-y_2 \)
    b. \( x-x_1 \quad x_1-x_2 \) and \( y-y_1 \quad y_1-y_2 \)
    c. \( x-x_1 \quad x_1-x_2 \) and \( y-y_1 \quad y_1-y_2 \)
    d. \( x-x_1 \quad x_1-x_2 \) and \( y-y_1 \quad y_1-y_2 \)
Chapter VII
Hidden Surfaces and Lines

Aim
The aim of this chapter is to:

- introduce the concept of hidden surfaces
- explain out-space and image-space methods
- explicate scan line algorithm

Objectives
The objectives of this chapter are to:

- describe Z-buffer algorithm
- elucidate Warnock’s algorithm
- explain binary space partition trees

Learning outcome
At the end of this chapter, you will be able to:

- understand hidden line methods
- identify the advantages and disadvantages of Z-buffer algorithm
- describe the Pseudo code for building a binary space partition tree
7.1 Introduction

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 surfaces or algorithm determines the lines, edges, surfaces or volumes, which are visible or invisible to an observer 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 and image-space method, respectively.

7.1.1 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.

7.1.2 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 or surface algorithms use image-space methods.

7.2 Z-Buffer Algorithm

- Z-buffer algorithm is one of the simplest and commonly used image space approaches to eliminate hidden surfaces.
- It is also known as depth buffer algorithm.
- It was 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 viewing plane is specified by x and y co-ordinated, 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 normalised coordinated, 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 (x, y) position as surfaces are processed, and the frame buffer stores the intensity values for each position.
- In the beginning, Z-buffer is initialised to zero, representing the z-value at the back clipping plane, and the frame buffer is initialised 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 value 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.
7.2.1 Advantages
• It is easy to implement.
• It can be implemented in hardware to overcome the speed problem.
• Since the algorithm processes objects one at a time, the total number of polygons in a picture can be arbitrary large.

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

7.3 Scan Line Algorithm
• A scan line method of hidden surface removal is another approach of image space method.
• It is an extension of the scan line algorithm for filling polygon interiors.
• Here, the algorithm deals with more than one surfaces, as each scan line is processed, it examines all polygon surfaces intersecting that line to determine which are visible.
• It then does the depth calculation and finds which polygon is nearest to the view plane.
• Finally, it enters the intensity value of the nearest polygon at that position into the frame buffer.
• 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.

7.4 Warnock’s 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:
  • Surrounding polygon – one that completely encloses the (shade) area of interest.

![Fig. 7.1 Surrounding polygon](image-url)
- Overlapping or intersecting polygon – one that is partly inside and partly outside the area.

Fig. 7.2 Overlapping or intersecting polygon

- Inside or contained polygon – one that is completely inside the area.

Fig. 7.3 Inside or contained polygon

- Outside or disjoint polygon – one that is completely outside the area.

Fig. 7.4 Outside or disjoint polygon

- If all the polygons are disjoint from the area, then the background colour is displayed in that area.
- 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.
- 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.
- If there are more than one polygons intersecting, contained in, or surrounding the area then we have to do some more processing.
7.4.1 Advantages
- It follows the divide-and-conquer strategy; therefore, parallel computers can be used to speed up the process.
- Extra memory buffer is not required.

7.5 Hidden Line Methods
- The calligraphic displays were available long before raster displays became economical.
- We know that in raster displays, we can use painters’ algorithm in which we actually draw hidden line, but finally hidden lines are overdrawn by visible lines.
- This is not the case with calligraphic displays.
- In calligraphic displays, we can not draw the hidden portions of lines and thus for each line we have to decide not only what objects lie in front of it but also just how those objects hide it.
- The objects within the image may not hide the line or may hide it partly or completely.
- To find how objects hide the lines, it is necessary to compare lines with relevant edges of the objects, i.e. it is not necessary to compare each line against all of the polygon edges in an object in order to find how objects hide the lines.
- In practice, we have to compare each line with the specific edges which can check whether the line is visible or not. These edges are known as contour edges, which are on the boundary where a front face meets a back face.
- We can find the contour edges by examining the object.
- For a solid object, each edge has two polygons adjacent to it.
- The edge for which both the polygons represent front faces or back faces is called interior edge.
- The edge for which both the polygons represent front face and other represents back face is called contour edge.
- For each intersection of line with contour edge, the line either passes behind an object or emerges from it. So with each such intersection, we can find the visible and invisible portions of the line, and we can draw only visible portions of lines.

7.6 Binary Space Partition Trees (BSP)
- The binary space partitioning (BSP) tree visible surface algorithm assumes that for a given viewpoint a polygon is correctly rendered, if all the polygons on its side away from the viewpoint are rendered first; then the polygon itself is rendered and finally, all the polygons on the side near the viewport are rendered.
- It is a two-part algorithm, in which, a scene is subdivided into two sections at each step with a plane that can be at any position and orientation.
- The BSP tree algorithm uses of the polygons in the scene as the separation dividing plane.
- Other polygons in the scene that are entered on one side of the separating plane are placed in the appropriate half space.
- Polygons that intersect the separating plane are split along the separating plane, and each portion is placed in the appropriate half space.
- Each half space is then recursively subdivided using one of the polygons in the half space as the separating plane. This subdivision continues until there is only a single polygon in each half space.
- The subdivided space is conveniently represented by a binary tree.
7.6.1 The Pseudo Code for Building a BSP Tree

```
struct
{
    polygon root;
    BSP_tree * backChild, * frontChild;
}
BSP_tree;
BSP_tree * BSP_makeTree_makeTree (polygon * polyList)
{
    polygon root;
    polygon * backlist, * frontList;
    polygon P, backPart, FrontPart;
    polygon P, backPart, FrontPart;
    if (PolyList == NULL) Then
        BSP_Tree = NULL;
    else
        { root = selectpolygon (&polyList);
            backlist = NULL;
            frontList = NULL;
            for (each remaining polygon P in polyList)
                { if (Polygon P in front of root) then
                    AddtoBSPList (P, &backlist);
                    elseif (polygon P in back of root) then
                        AddtoBSPList (P, &backlist);
                    else
                        { Spliptpolygon (P, root, &frontPart, &backPart);
                            AddtoBSPList (frontPart, &frontList);
                            AddtoBSPList (backPart, &backlist);
                        }
                    endif
                }
        return combineBSP_tree (BSP_makeTree {frntList), root, BSP_makeTree (backlist);
    }
} /* BSP_makeTree */
```
7.6.2 The Pseudo Code for Displaying a BSP Tree

Void DisplayBSP_Tree (BSP_tree * tree) {
    if (tree != NULL) then
        { /* Display back child, root, and front child. */
            DisplayBSP_Tree (tree → backChild);
            DisplayBSP_Tree (tree → frontChild);
        }
    else
        { /* Display front child, root, and back child. */
            DisplayBSP_Tree (tree → frontChild);
            DisplayPolygon (Tree → root);
            DisplayBSP_Tree (tree → bakChild);
        }
    endif
} /* DisplayBSP_Tree */
Summary

- A scan line method of hidden surface removal is another approach of image space method.
- The hidden line or hidden surfaces or algorithm determines the lines, edges, surfaces or volumes, which are visible or invisible to an observer at a specific point in space.
- Object-Space method is implemented in the physical coordinate system in which objects are described.
- Image space method is implemented in the screen coordinate system in which the objects are viewed.
- Z-buffer algorithm is one of the simplest and commonly used image space approaches to eliminate hidden surfaces.
- Surrounding polygon is the one that completely encloses the (shade) area of interest.
- Overlapping or intersecting polygon is the one that is partly inside and partly outside the area.
- Inside or Contained polygon is the one that is completely inside the area.
- Outside or Disjoint polygon is the one that is completely outside the area.
- To find how objects hide the lines, it is necessary to compare lines with relevant edges of the objects i.e., it is not necessary to compare each line against all of the polygon edges in an object in order to find how objects hide the lines.
- The edge for which, both the polygons represent front face and other represents back face is called contour edge.
- The edge for which, both the polygons represent front faces or back faces is called interior edge.
- The binary space partitioning (BSP) tree visible surface algorithm assumes that for a given viewpoint a polygon is correctly rendered, if all the polygons on its side away from the viewpoint are rendered first; then the polygon itself is rendered; and finally, all the polygons on the side near the viewport are rendered.

References

- Dr. Chawla, A., Lecture - 12 Hidden Surface Removal [Video online] Available at: <http://www.youtube.com/watch?v=GxSCRLF37z4> [Accessed 21 June 2013].

Recommended Reading

Self Assessment

1. ________ is implemented in the physical coordinate system in which objects are described.
   a. Image space method
   b. Object space method
   c. Z-buffer algorithm
   d. Scan line algorithm

2. ________ is implemented in the screen coordinate system in which the objects are viewed.
   a. Scan line algorithm
   b. Object space method
   c. Image space method
   d. Z-buffer

3. ________ is one of the simplest and commonly used image space approaches to eliminate hidden surfaces.
   a. Z-buffer algorithm
   b. Scan line algorithm
   c. DDA algorithm
   d. Bresenham’s algorithm

4. Which of the following statements is true?
   a. The image space 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.
   b. Scan lines are processed from left to right.
   c. The scan line method of hidden surface removal does not store 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.
   d. The object space 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.

5. Which algorithm did Warnock develop?
   a. Z-buffer algorithm
   b. DDA algorithm
   c. Scan line algorithm
   d. Subdivision algorithm

6. Which of the following statements is true?
   a. Surrounding polygon is the one that completely encloses the (shade) area of interest.
   b. Overlapping or intersecting polygon is the one that completely encloses the (shade) area of interest.
   c. Inside or Contained polygon is the one that completely encloses the (shade) area of interest.
   d. Outside or Disjoint polygon is the one that completely encloses the (shade) area of interest.

7. What represents subdivided space?
   a. Scan line algorithm
   b. Image space method
   c. binary tree
   d. subdivision algorithm
8. Which of the following statements is false?
   a. The edge for which both the polygons represent front face and other represents back face is called contour edge.
   b. The edge for which both the polygons represent front faces or back faces is called binary tree.
   c. The objects within the image may not hide the line or may hide it partly or completely.
   d. The edge for which both the polygons represent front faces or back faces is called interior edge.

9. _________ is the one that is completely inside the area.
   a. Surrounding polygon
   b. Overlapping or intersecting polygon
   c. Outside or Disjoint polygon
   d. Inside or Contained polygon

10. Which algorithm uses the polygons in the scene as the separation dividing plane.
    Polygons that intersect the separating plane are split along the separating plane, and each portion is placed in the appropriate half space.
    a. BSP algorithm
    b. Scan line algorithm
    c. Z-buffer algorithm
    d. Subdivision algorithm
Chapter VIII

Light, Colour and Shading

Aim

The aim of this chapter is to:

- explain the light concepts
- elucidate the colour details
- explicate the shading details

Objectives

The objectives of this chapter are to:

- elucidate the importance of lighting, shading and colour in computer graphics
- explain colour tables and transparency
- define ray-tracing

Learning outcome

At the end of this chapter, you will be able to:

- understand diffuse illumination
- define gloss
- describe shading algorithm
8.1 Introduction
The shading model is used to calculate the intensity of light that we should see at a given point on the surface of an object. The shading model is also called illumination model or lighting model.

8.2 Diffuse Illumination

- An object's 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 the 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.
- 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 black surface absorbs most of the incident light. Reflection coefficient for gray shades is in between 0 to 1.

8.3 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 in its original colour, but the colour of incident light is white.

8.4 Shading Algorithms

- We can shade any surface by calculating the surface normal at each visible point and applying the desired illumination model at that point.
- Each polygon can be drawn with a single intensity, or with different intensity obtained at each point on the surface.

8.4.1 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 single intensity value.
- This method is valid for the following assumptions:
  - The light source is at infinity, so N.L is constant across the polygon face.
  - The viewer is at infinity, so V.R is constant over the surface.
  - The polygon represents the actual surface being modelled, 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.
8.5 Transparency

- 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 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,
- \( T \) = the coefficient of property of material which determines how much of the light is transmitted at the surface instead of reflected.
- \( a \) = the coefficient property of a material which tells how quickly the material absorbs or attenuates the light.
- \( d \) = the distance light must travel in the object.

- When the light crosses the boundary between two media it changes the direction. This effect is called refraction.
- The effect of refraction is observed because the speed of light in different materials resulting different path for refracted light for that of incident light.
- The direction of refracted light is specified by the angle of refraction \( \theta_r \).
- It is the function of the property materials called the index of refraction \( n \).
- The angle of refraction \( \theta_r \), is calculated from the angle of incident \( \theta_i \), the index of refraction \( n_i \) of the incident material and the index of refraction \( n_r \) of the refracting material according to Snell’s law,

\[
\sin \theta_r = \frac{n_i}{n_r} \sin \theta_i
\]

8.6 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 viewport.
- 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.
- Another way to locate shadow areas is the use of shadow volumes.
- The shadow volume is defined by the light source and an object and is bounded by a set of visible shadow polygons. This volume is also known as polygon’s shadow volume.
8.7 Ray-Tracing

- If we consider the line of sight from a pixel position on the view plane through a scene, we can determine which objects in the scene intersect this line.
- From the intersection points with different objects, 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.

8.7.1 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 reflections which 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 set 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.

8.7.2 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.
- Distributed ray tracing achieves these types 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.

8.7.3 Soft Shadows

- Shadows in traditional ray tracing are discrete.
- When shading a point, each light source is checked to see if it is visible.
- The light sources itself is modelled 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 sources 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 modelling 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.
8.7.4 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 too far away or too close will appear unfocused and blurry.
- Almost all computer graphics rendering techniques use a pinhole camera mode.
- 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, simulation 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.

8.7.5 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 cameras, then they will appear blurred on the film.
- Distributed ray tracing can simulate this blurring by distributing rays temporarily 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 fast motion will have the large amount of blurriness in the rendering image.

8.8 Colour Tables

- Most of the display devices are capable of generating a large variety of colours.
- They have 8-bit of control over each of the red, green and blue channels. Therefore, they can have 256 intensity levels.
- With 253 intensity levels of each colour they can produce 16,777,216 different colours.
- To specify the colour of any pixel 24-bit (8-bit per colour) are required.
- To store 24-bit information of each colour (total of 16,777,216 colours) in the frame buffer is not cost-effective.
- Therefore, in normal practice, only few colours are specified in the table called colour table.
- A colour table allows us to map between a colour index in the frame buffer and a colour specification.
- The colours in the colour table are referred by the colour index.
Fig. 8.1 Colour table
Summary

- The shading model is used to calculate the intensity of light that we should see at a given point on the surface of an object. The shading model is also called illumination model or lighting model.
- An object’s illumination is as important as its surface properties, in computing its intensity.
- 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 on a concentrated region around the specular reflection angle is called specular reflection.
- We can shade any surface by calculating the surface normal at each visible point and applying the desired illumination model at that point.
- The fast and simplest method for shading polygons is constant shading, also known as faceted shading or flat shading.
- A transparent surface, in general, produces both, reflected and transmitted light.
- The expression for coefficient of transparency is given as, \( T = e^{-ad} \).
- A shadowed object is one which is hidden from the light source.
- Glossy surfaces are generated in distributed ray tracing by randomly distributing rays reflected by a surface.
- Shadows in the real world are much softer.
- Both, the human eye and cameras have a finite lens aperture, and therefore have a finite depth of field.
- If objects in the scene are in motion relative to the cameras, then they will appear blurred on the film.
- Colour tables have 8-bit of control over each of the red, green and blue channels. Therefore, they can have 256 intensity levels.
- A colour table allows us to map between a colour index in the frame buffer and a colour specification.
- 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.

References


Recommended Reading

1. An object’s ________ is as important as its surface properties in computing its intensity.
   a. shadow
   b. lighting
   c. illumination
   d. reflection

2. Which of the following statements is true?
   a. Colour tables have 8-bit of control over each of the yellow, green and blue channels. Therefore, they can have 256 intensity levels.
   b. Colour tables have 8-bit of control over each of the red, green and blue channels. Therefore, they can have 256 intensity levels.
   c. Colour tables have 8-bit of control over each of the red, black and blue channels. Therefore, they can have 256 intensity levels.
   d. Colour tables have 8-bit of control over each of the red, green and white channels. Therefore, they can have 256 intensity levels.

3. Due to ________ at the highlight, the surface appears to be in its original colour, but the colour of incident light is white.
   a. specular reflection
   b. coefficient of reflection
   c. reflectivity
   d. shadow

4. Which of the following statements is true?
   a. The percentage of the light reflected from the surface to the total incoming light to the surface is called coefficient of reflection or the reflectivity.
   b. The product of the light reflected from the surface to the total incoming light to the surface is called coefficient of reflection or the reflectivity.
   c. The ratio of the light reflected from the surface to the total outgoing light to the surface is called coefficient of reflection or the reflectivity.
   d. 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.

5. What depends on the thickness of the object?
   a. Coefficient of transparency
   b. Coefficient of reflection
   c. Reflectivity
   d. Illumination of the object

6. Real surfaces that are ________ generally transmit a blurred image of the scene behind them.
   a. transparent
   b. translucent
   c. opaque
   d. white in colour
7. Which of the following statements is false?
   a. More often surfaces are glossy and reflect a blurred image of the scene. This is due to the light scattering properties of the surface.
   b. Reflections in traditional ray tracing are always sharp, even partial reflections.
   c. Glossy surfaces are generated in distributed ray tracing by randomly distributing rays reflected by a surface.
   d. More often surfaces are glossy and reflect a blurred image of the scene. This is due to the shadow scattering properties of the surface.

8. What is the fastest and simplest method for shading polygon known as?
   a. shading
   b. lighting
   c. faceted shading
   d. reflectivity

9. Which of the following statements is false?
   a. Shadows in the real world are much harder.
   b. 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.
   c. 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.
   d. Shadows in the real world are much softer.

    a. \( \sin \theta_r = \frac{n_r}{n_i} \sin \theta_i \)
    b. \( \sin \theta_r = \frac{n_i}{n_r} \sin \theta_i \)
    c. \( \sin \theta_r = \frac{n_i}{n_r} \sin \theta_i \)
    d. \( \sin \theta_r = \frac{n_i}{n_r} \sin \theta_i \)
Chapter IX
Curves, Fractals and Animation

Aim
The aim of this chapter is to:

- explain curve
- discuss fractals
- explain interpolation

Objectives
The objectives of this chapter are to:

- classify the use of curves in 3D animation
- explain the use of fractals in 3D animation
- elucidate the use of interpolation in 3D animation

Learning outcome
At the end of this chapter, you will be able to:

- understand different types of curves
- identify spline representation
- describe fractal surfaces
9.1 Introduction

The only primitive in geometric objects that computer graphics hardware typically can draw are points, lines and polygons. Usually, there are special mechanisms to draw images, text characters but these are not commonly considered ‘geometric’ since they are not usually transformed by geometric operations like rotation and perspective projection. Obviously it is important that graphics programs be able to draw more complex geometric forms from simple circles to complex surfaces in three dimensions. We can use two approaches to draw curved lines. One approach is to use a curve generation algorithm such as DDA. In this approach, a true curve is created. A major problem in the design of graphics libraries is to provide a high level interface to the hardware that allows users to draw curves and surfaces to line segments and polygons themselves.

9.1.1 Curve Generation

- We can use two approaches to draw curved lines. One approach is to use a curve generation algorithm such as DDA. In this approach, a true curve is created.
- In the second approach, the curve is approximated by a number of small straight line segments.
- This can be achieved with the help of interpolation techniques.

9.1.2 Problems in True-Curve Generation Approach

- To specify a curve, we need more information than just its endpoints.
- It is difficult to apply transformations.
- Example: a circle when scaled in only one direction becomes an ellipse. If our algorithm supports only circular arc generation then ability to scale pictures is required.
- New clipping algorithm is required to clip arcs.
- The curve generation algorithms for curves other than circular or elliptical such as airplane wings or cars or human faces are complex.

9.2 Interpolation

- 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 curves, then we can draw the required curve by filling portions of curves with the pieces of known curves which pass through nearby sample points.
- The gap between the sample points can be filled by finding the coordinates of the points along the known approximating curve and connecting these points with line segments.
The main task in this process is to find the suitable mathematical expression for the known curve.

- There are polynomials, trigonometric, exponential and other classes of functions that can be used to approximate the curve.
- Usually polynomial functions in the parametric forms are preferred.
- The polynomial functions in the parametric form can be given as:

\[
\begin{align*}
x &= f_x(u) \\
y &= f_y(u) \\
z &= f_z(u)
\end{align*}
\]
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 specify a spine curve by giving a set of coordinate positions, called control points, which indicates the general shape of the curve.
- When polynomial sections are fitted for the curve to pass through all control points, 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.

Fig. 9.2 Interpolation spline and approximation spline

9.3.1 Spline Specifications

There are three basic ways of specifying spline curves:
- We can state the set of boundary conditions that are imposed on the spline.
- We can state the matrix that has characteristics of a spline.
- 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.

9.4 Bezier Curves

- Bezier curve is 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 conveniently for curve and surface design.
- They are easy to implement. Therefore, Bezier curves are widely available in various CAD systems and in general graphic packages.
9.4.1 Properties of Bezier Curve

- The basic functions are real.
- Bezier curve always passes through the first and last control points, i.e., curve has same end points as the guiding polygon.
- The degree of the polynomials 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.
- The curve generally follows the shape of the defining polygon.
- The direction of the tangent vector at the end points is the same as that of the vector determined by first and last segments.
- The curve lies entirely within the convex hull formed that the polynomial smoothly follows the control points.
- The curve exhibits the variation diminishing property. This means that the curve does not oscillate about any straight line defining polygon.
- The curve is invariant under an affine transformation.

9.5 B-Spline Curves

- 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.
- 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.
- The only way to reduce the degree of the curve is to reduce the number of vertices.
- The second limiting characteristic is that the value of the blending function is non-zero 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.

9.5.1 Properties of B-Spline Curve

- The sum of the B-spline basis functions for any parameter value (u) is 1.
  \[ \sum_{i=1}^{n+1} N_{i,k}(u) = 1 \]
- 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 equals 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).
- 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 lies within the convex hull of its defining polygon.
9.6 Fractals

In nature we come across rough, jagged, random surfaces and edges when we try to draw mountains, trees, rivers, etc. Such rough, jagged and random surfaces are called fractals.

9.6.1 Classification of Fractals

The fractals can be classified as:

- self similar
- self affine
- invariant

9.6.1.1 Self Similar Fractals

These fractals have parts which are scaled-down versions of the entire object.

- In these fractals, object subparts are constructed by applying scaling parameters to the overall initial shape.
- It is a choice user to user, the same scaling factors for all subparts, or use different scaling factors for different scaled down parts of the object.
- Another sub-class of self similar fractals is statistically self similar fractals, in which user can also apply random variations to the scaled down subparts.
- These fractals are commonly used to model trees, shrubs, plants, etc.

9.6.1.2 Self Affine Fractals

These fractals have parts those are formed with different scaling parameters, \( s_x, s_y, s_z \) in different coordinate directions. In these fractals, we can also apply random variations to obtain statistically self-affined fractals. These fractals are commonly used to model water clouds and terrain.

9.6.1.3 Invariant Fractals

In these fractals, non-linear transformation is used. It includes self squaring fractals, which are formed which squaring fractions in complex space and self inverse form with inversion procedures.

9.7 Fractal Lines

Fractal lines can be generated by performing following steps:

- If the straight line segment is specified by points \((x, y, z)\) and \((a, b, c)\), find the midpoint of line by using following expression,

\[
\frac{x + a}{2}, \frac{y + b}{2}, \frac{z + c}{2}
\]

- Add an offset to the midpoint in such a way that the resultant midpoint should not lie on the line. This can be achieved by adding offset term to each coordinate as follows,

\[
\frac{x + a}{2} + d_x, \frac{y + b}{2} + d_y, \frac{z + c}{2} + d_z
\]

In order to get random effect, calculate the offset as shown below,

\[
d_x = L \times W \times \text{GAUSS}
\]
\[ d_y = L \times W \times \text{GAUSS} \]
\[ d_z = L \times W \times \text{GAUSS} \]

Where,
- \( L \): Length of the segment
- \( W \): waiting function governing the curve roughness (i.e. fractal dimension)
- \( \text{GAUSS} \): Gaussian variable which returns random values between -1 and 1 with 0 mean i.e., returned values consist of equal amount of positive and negative values.

The shifted midpoint divides the original line into two parts. Repeat the same process for each part separately.

Repeat the processes until the segments become small enough. By following the above procedures we can get fractal line as below:

- The above implementation can be easily achieved using a recursive procedure.

### 9.8 Fractal Surfaces

The concept of fractal lines can be extended to generate fractal surfaces. There are many ways to accomplish this. Here, we will use method base on triangles. We know that triangle has three vertex points. We can generate a fractal surface for the area between them.

![Fig. 9.3 Fractal surface](image)
Summary

- To specify a curve, we need more information than just its endpoints.
- The curve generation algorithms for curves other than circular or elliptical such as airplane wings or cars or human faces, are complex.
- If we have a set of sample points which lie on the required curves, then we can draw the required curve by filling portions of curves with the pieces of known curves which pass through nearby sample points.
- To produce a smooth curve through a designated set of points, a flexible strip called spline is used.
- Bezier curve is another approach for the construction of the curve.
- The degree of the polynomials 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.
- A curve generated by using the vertices of a defining polygon dependeds on some interpolation or approximation scheme to establish the relationship between the curve and the polygon.
- Self similar fractals have parts those are scaled-down versions of the entire object.
- Self affine fractals have parts those are formed with different scaling parameters, $S_x$, $S_y$, $S_z$ in different coordinate directions.
- In invariant fractals, non-linear transformation is used.
- The concept of fractal lines can be extended to generate fractal surfaces.

References


Recommended Reading

Self Assessment

1. When polynomial sections are fitted for a curve to pass through all control points, the resulting _______ is said to interpolate the set of control points.
   a. curve
   b. point
   c. polygon
   d. vector

2. _______ always passes through the first and last control points, i.e., curve has same end points as the guiding polygon.
   a. B-spline
   b. Cuve
   c. Spline
   d. Bezier curve

3. What is used to produce a smooth curve through a designated set of points?
   a. Spline
   b. Line
   c. Points
   d. Curves

4. How is a Bezier curve determined?
   a. By defining polygon.
   b. By defining line segments.
   c. By defining spline.
   d. By defining points.

5. Which of the following statements is false?
   a. Bezier curve is another approach for the construction of the curve.
   b. A Bezier curve is determined by a defining polygon.
   c. Bezier curves have a number of properties that make them highly useful and conveniently for curve and surface design.
   d. A Bezier curve is determined by a defining spline.

6. Which of the following statements is true?
   a. The gap between the sample points can be filled by finding the coordinates of the points along the known approximating curve and connecting these points with lines.
   b. The gap between the sample points can be filled by finding the coordinates of the points along the known approximating curve and connecting these points with polygons.
   c. The gap between the sample points can be filled by finding the coordinates of the points along the known approximating curve and connecting these points with line segments.
   d. The gap between the sample points can be filled by finding the coordinates of the points along the known approximating curve and connecting these points with polylines.
7. Which of the following statements is true?
   a. To specify a curve, we need more information than just its endpoints.
   b. To specify a fractal, we need more information than just its endpoints.
   c. To specify a polygon, we need more information than just its endpoints.
   d. To specify a polyline, we need more information than just its endpoints.

8. The sum of the B-spline basis functions for any parameter value (u) is ________.
   a. 2
   b. 3
   c. 1
   d. 5

9. ________ have parts that are scaled-down versions of the entire object.
   a. Curves
   b. Fractals
   c. Polygons
   d. Vectors

10. Which of the following concepts can be extended to generate fractal surfaces?
    a. Fractal lines
    b. Lines
    c. Curves
    d. Polygons
Application I

Obtain transformation matrix for rotation about the line joining the points (0, 0, 0) and (1, 1, 1) with the angle of rotation 45° in counter-clockwise sense.

Solution:
In this case, the line passes through the origin, so the translation is not required. Therefore, \( R_T \) can be given as,

\[
R_T = R_{xy} R_x R_{xy}^{-1}
\]

By usual notations,

\[
\lambda = \sqrt{1 + 1} = \sqrt{2}
\]

\[
|V| = \sqrt{1 + 1 + 1} = \sqrt{3}
\]

Here, a = 1, b = 1, c = 1.

\[
R_{xy} = \begin{bmatrix}
\frac{\sqrt{2}}{2} & 0 & \frac{1}{2} \\
-\frac{1}{2} & 0 & \frac{\sqrt{2}}{2} \\
0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2}
\end{bmatrix}, \quad R = \begin{bmatrix}
\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\
-\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
R_{xy}^{-1} R_{xy}^{-1} = \begin{bmatrix}
\frac{\sqrt{2}}{2} & 0 & -\frac{1}{2} \\
\frac{1}{2} & 0 & \frac{\sqrt{2}}{2} \\
0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2}
\end{bmatrix}
\]

\[
R_T = R_{xy} R_x R_{xy}^{-1}
\]

\[
R_T = \begin{bmatrix}
0.80473 & 0.5058 & -0.3106 & 0 \\
-0.3106 & 0.80473 & 0.5058 & 0 \\
0.5058 & -0.3106 & 0.80473 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
**Application II**

Construct a B-spline curve of order 4 and with 4 polygon vertices $A(1, 1)$, $B(2, 3)$, $C(4, 3)$ and $D(6, 2)$.

**Application III**

Find the transformation for:

- a. Cavalier projection with $\theta = 45^\circ$
- b. Cabinet projection with $\theta = 30^\circ$
References


Recommended Reading

Self Assessment Answers

Chapter I
1. b
2. a
3. b
4. c
5. d
6. a
7. c
8. a
9. c
10. b

Chapter II
1. a
2. b
3. a
4. d
5. b
6. c
7. a
8. b
9. b
10. d

Chapter III
1. a
2. d
3. c
4. a
5. b
6. a
7. c
8. d
9. a
10. b
Chapter IV
1. b
2. a
3. c
4. d
5. a
6. c
7. b
8. a
9. c
10. b

Chapter V
1. b
2. a
3. c
4. d
5. a
6. b
7. c
8. d
9. a
10. c

Chapter VI
1. b
2. d
3. c
4. a
5. b
6. d
7. d
8. a
9. a
10. c
Chapter VII
1. b
2. c
3. a
4. b
5. d
6. a
7. d
8. b
9. d
10. a

Chapter VIII
1. c
2. b
3. a
4. d
5. a
6. b
7. d
8. c
9. a
10. c

Chapter IX
1. a
2. d
3. a
4. a
5. d
6. c
7. a
8. c
9. b
10. a