JECRC Foundation

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE
(Department of Information Technology)
Session: 2020-21

JAIPUR ENGINEERING COLLEGE AND RESEARCH CENTRE

Year & Semester – III / V

Subject – Computer Graphics and Multimedia (5IT4-04)

VISSION AND MISSION OF INSTITUTE

Vision:

To become a renowned centre of outcome based learning, and work towards academic, professional, cultural and social enrichment of the lives of individuals and communities.

Mission:

- Focus on evaluation of learning outcomes and motivate students to inculcate research aptitude by project based learning.
- Identify areas of focus and provide platform to gain knowledge and solutions based on informed perception of Indian, regional and global needs.
- Offer opportunities for interaction between academia and industry.
- Develop human potential to its fullest extent so that intellectually capable and imaginatively gifted leaders can emerge in a range of professions.

VISSION AND MISSION OF DEPARTMENT

Vision:

To establish outcome based excellence in teaching, learning and commitment to support IT Industry.

Mission:

M1: To provide outcome based education.
M2: To provide fundamental & Intellectual knowledge with essential skills to meet current and future need of IT Industry across the globe.
M3: To inculcate the philosophy of continues learning, ethical values & Social Responsibility

Program Educational Objectives (PEOs)

PEO1: To strengthen students with fundamental knowledge, effective computing, problem solving and communication skills enable them to have successful career in Information Technology.

PEO2: To enable students in acquiring Information Technology's latest tools, technologies and management principles to give them an ability to solve multidisciplinary engineering problems.

PEO3: To impart students with ethical values and commitment towards sustainable development in collaborative mode.

PEO4: To reinforce students with research aptitude and innovative approaches which help them to identify, analyze, formulate and solve real life problems and motivates them for lifelong learning.

PEO5: To empower students with leadership quality and team building skills that prepare them for employment, entrepreneurship and to become competent professionals to serve societies and global needs

Program Outcomes(POs)

PO1:Problem Analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO2:Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO3:Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

Program Outcomes(POs)

PO4:Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5:Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6:The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

Program Outcomes(POs)

PO7:Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8:Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9:Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

Program Outcomes(POs)

PO10:Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11:Project Management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects in multidisciplinary environments.

PO12:Life –long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological changes needed.

Program Specific Outcomes(PSO)

PSO1: Graduates of the program would be able to develop mobile and web based IT solutions for real time problems.

PSO2: Graduates of the program would be able to apply the concepts of artificial intelligence, machine learning and deep learning.

Course Objectives:

- This course provides an introduction to the principles of computer graphics.
- This course will consider methods for modeling 2-D and 3-D objects and efficiently generating photorealistic renderings on color raster graphics devices.
- The emphasis of the course will be placed on understanding how the various elements that underlie computer graphics (algebra, geometry, algorithms and data structures, optics, and photometry) interact in the design of graphics software systems.
- It also introduces brief introduction about multimedia and animation.

Scope:

- **Computer Graphics Technology** is used across a range of industries. Computer graphics technology professionals use their knowledge and technical skills in graphic design and animation to design and create layouts, Web pages, and multimedia productions.
- **Multimedia Technology** is all about imparting information through use of computers by integrating various mediums like image, sound, text and videos. This field has gained popularity due to the varied palate of career opportunities it can offer.

Scope:

- **Work opportunities for quality animators** and related professionals like Graphic Designer, Multimedia Developer and Game Developer, Character Designers, Key Frame Animators, 3D Modelers, Layout Artists etc. exists in following sectors at large-
    - Advertising
    - Online and Print News Media
    - Film & Television
    - Cartoon production
    - Theater
    - Video Gaming
    - E-learning

Scope:

- Opportunities exists both with government as well as private sector enterprises.
- Animation itself is an industry, and as industry it’s on boom.
- There exist numerous animation houses both in India and abroad who work for clienteles.
- An animator and multimedia professional can also work as freelancer or start his / her own enterprise given he / she has entrepreneurial skills and funds for investments.
- Animators work in various capacities.

Course Outcomes (COs):

After completion of this course, students will be able to:

CO1: Understand the basic concepts, importance, applications of Computer Graphics also they will be able to understand working principle of display devices.

CO2: Interpret mathematical foundation to learn graphics algorithms to draw various shapes such as line, circle, ellipse, and curves on raster scan system.

CO3: Apply methods for modeling 2-D and 3-D objects such as transformation, clipping , viewing , rendering and dithering techniques.

CO4: Design animation sequences, as well as recursively defined curves such as Koch curves, C curves and many more.

Mapping of Course Outcomes with Program Outcomes

CO\PO	Blooms Taxonomy Level	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	L1	H	M	L	L	L	L	L	M	M	M	L	H
CO2	L2	M	H	M	H	H	L	L	M	M	M	L	H
CO3	L3	M	M	H	H	H	L	L	M	M	M	L	H
CO4	L2	M	H	M	H	H	L	L	M	M	M	L	H

Mapping of Course Outcomes with Program Specific Outcomes

CO\PSO	Blooms Taxonomy Level	PSO1	PSO2
CO1	L1	Yes	NA
CO2	L2	Yes	NA
CO3	L3	Yes	NA
CO4	L2	Yes	NA

Unit-III: Graphics Primitives
CONTENTS

- Points, Line, Circle, Ellipse as primitives
- Scan Conversion Algorithms
- Filled Area Primitives
- Character Generation
- Aliasing and Anti-Aliasing Algorithms

Unit-III: Graphics Primitives/Output Primitives

Introduction:

- **picture : a picture** is completely specified by the set of intensities for the pixel positions in the display.
- a picture as a set of complex objects, such as trees and terrain or furniture and walls, positioned at specified coordinate locations within the scene.
- Shapes and colors of the objects. can be described internally with pixel arrays or with sets of basic geometric structures, such as straight line segments and polygon color areas.

Unit-III: Graphics Primitives/Output Primitives

Introduction:

- The scene is then displayed either by loading the pixel arrays into the frame buffer or by scan converting the basic geometric-structure specifications into pixel patterns.
- Typically, graphics programming packages provide functions to describe a scene in terms of these basic geometric structures, referred to as output primitives, and to group sets of output primitives into more complex structures.

Unit-III: Graphics Primitives/Output Primitives

Introduction:

- Each output primitive is specified with input coordinate data and other information about the way that object is to be displayed.
- Points and straight line segments are the simplest geometric components of pictures.
- Additional output primitives that can be used to construct a picture include circles and other conic sections, quadric surfaces, spline curves and surfaces, polygon color areas, and character strings.

Unit-III: Graphics Primitives/Output Primitives

Introduction:

- We begin our discussion of picture-generation procedures by examining device-level algorithms for displaying two-dimensional output primitives, with particular emphasis on scan-conversion methods for raster graphics systems.

Unit-III: Graphics Primitives/Output Primitives

Points and Lines:

- Point plotting is accomplished by converting a single coordinate position furnished by an application program into appropriate operations for the output device in use.
- Line drawing is accomplished by calculating intermediate positions along the line path between two specified endpoint positions. An output device is then directed to fill in these positions between the endpoints.

Unit-III: Graphics Primitives/Output Primitives

Points and Lines:

- Digital devices display a straight line segment by plotting discrete points between the two endpoints. Discrete coordinate positions along the line path are calculated from the equation of the line. For a raster video display, the line color (intensity) is then loaded into the frame buffer at the corresponding pixel coordinates.
- Reading from the frame buffer, the video controller then "plots" the screen pixels. Screen locations are referenced with integer values, so plotted positions may only approximate actual Line positions between two specified endpoints. A computed line position of (10.48,20.51), for example, would be converted to pixel position (10,21).

Unit-III: Graphics Primitives/Output Primitives

Points and Lines:

- Thus rounding of coordinate values to integers causes lines to be displayed with a stairstep appearance ("the jaggies").
- The characteristic stairstep shape of raster lines is particularly noticeable on systems with low resolution, and we can improve their appearance somewhat by displaying them on high-resolution systems. More effective techniques for smoothing raster lines are based on adjusting pixel intensities along the line paths.
- For the raster-graphics device-level algorithms discussed in this lecture object positions are specified directly in integer device coordinates.

Unit-III: Graphics Primitives

Line Drawing Algorithms:

1. **Digital Differential Analyser (DDA)**
2. **Bresenham's Line Drawing Algorithm**

Unit-III: Graphics Primitives

DDA Line Drawing Algorithm:

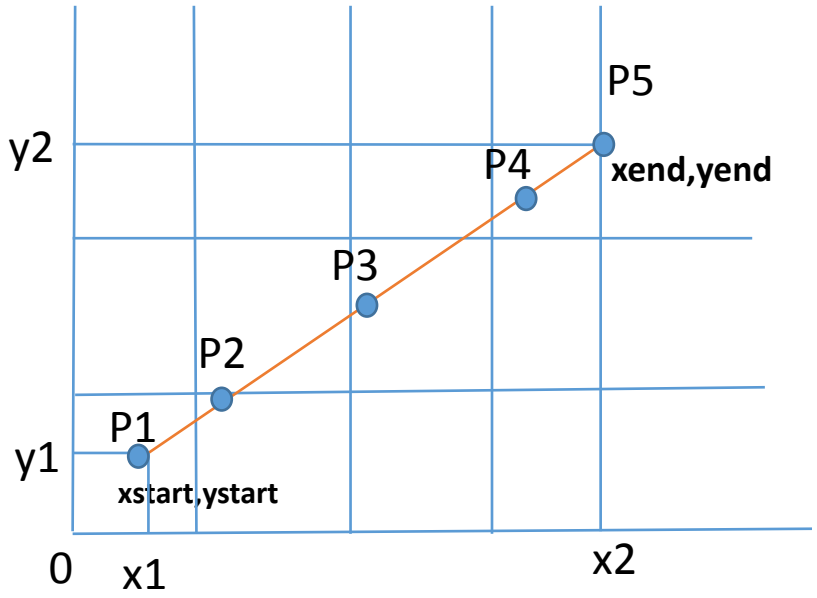
- The *Digital* ***Differential*** Analyzer **(DDA)** is a scan-conversion line algorithm based on calculating either Δy or $\Delta \boldsymbol{x}$.

- On raster systems, lines are plotted with pixels, and step sizes in the horizontal and vertical directions are constrained by pixel separations. That is, we must "sample" a line at discrete positions and determine the nearest pixel to the line at each sampled position.

Unit-III: Graphics Primitives

DDA Line Drawing Algorithm:

- Digital Differential Analyzer (DDA) line drawing algorithm is the simplest line drawing algorithm in computer graphics.
- It works on incremental method. It plots the point from source to destination by incrementing the source coordinates.



$\Delta y = yend - ystart$ (dy)
$\Delta x = xend - xstart$ (dx)

$m = \Delta y/\Delta x$ or dy/dx

Unit-III: Graphics Primitives

DDA Line Drawing Algorithm:

A linear DDA starts by calculating the smaller value of **dy** or **dx** for a unit increment of the other. A line is then sampled at unit intervals in one coordinate and corresponding integer values nearest the line path are determined for the other coordinate.

$dy = y_{end} - y_{start}$

$dx = x_{end} - x_{start}$

Slope $m=dy/dx$ if $dx > dy$, $m < 1$

Considering a line with positive slope, if the slope is less than or equal to 1, we sample at unit x intervals (dx=1) and compute successive y values as

$$\mathbf{y_{k+1}=y_k + m}$$

$$\mathbf{x_{k+1}= x_k + 1}$$

Unit-III: Graphics Primitives

DDA Line Drawing Algorithm:

Subscript k takes integer values starting from 0, for the 1st point and increases by 1 until endpoint is reached. y value is rounded off to nearest integer to correspond to a screen pixel.

For lines with slope greater than 1, we reverse the role of x and y i.e. we sample at dy=1 and calculate consecutive x values as

$$x_{k+1} = x_k + 1/m$$

$$y_{k+1} = y_k + 1$$

Unit-III: Graphics Primitives

DDA Line Drawing Algorithm:

Step1: Start Algorithm

Step2: Declare x_1,y_1,x_2,y_2,dx,dy,x,y as integer variables.

Step3: Enter value of x_1,y_1,x_2,y_2.

Step4: Calculate dx = x_2-x_1

Step5: Calculate dy = y_2-y_1

Step6: If ABS (dx) > ABS (dy)

Then step = abs (dx)

Else

step = abs(dy)

Unit-III: Graphics Primitives

DDA Line Drawing Algorithm:

Step7: x_{inc}=dx/step

y_{inc}=dy/step

assign x = x_1

assign y = y_1

Step8: Set pixel (x, y)

Step9: x = x + x_{inc}

y = y + y_{inc}

Set pixels (Round (x), Round (y))

DDA Line Drawing Algorithm:

Step10: Repeat step 9 until $x = x_2$

Step11: End Algorithm

Unit-III: Graphics Primitives

DDA Line Drawing Algorithm:

Example: If a line is drawn from (2, 3) to (6, 15) with use of DDA. How many points will needed to generate such line?

Solution: P_1 (2,3) P_{13} (6,15)

$x_1=2$ $y_1=3$

$x_2= 6$ $y_2=15$

$dx = 6 - 2 = 4$ $dy = 15 - 3 = 12$

Unit-III: Graphics Primitives

m=dy/dx = 12/4 = 3, 1/m = 1/3

Here dy > dx, step = dy = 12

xinc = 4/12 = 1/3

yinc = 12/12 = 1

For calculating next value of x takes x = x + 1/m

$P_1(2, 3)$	point plotted
$P_2(2\frac{1}{3}, 4)$	point plotted
$P_3(2\frac{2}{3}, 5)$	point not plotted
$P_4(3, 6)$	point plotted
$P_5(3\frac{1}{3}, 7)$	point not plotted
$P_6(3\frac{2}{3}, 8)$	point not plotted
$P_7(4, 9)$	point plotted
$P_8(4\frac{1}{3}, 10)$	point not plotted
$P_9(4\frac{2}{3}, 11)$	point not plotted
$P_{10}(5, 12)$	point plotted
$P_{11}(5\frac{1}{3}, 13)$	point not plotted
$P_{12}(5\frac{2}{3}, 14)$	point not plotted
$P_{13}(6, 15)$	point plotted

Unit-III: Graphics Primitives

DDA:

Advantages:

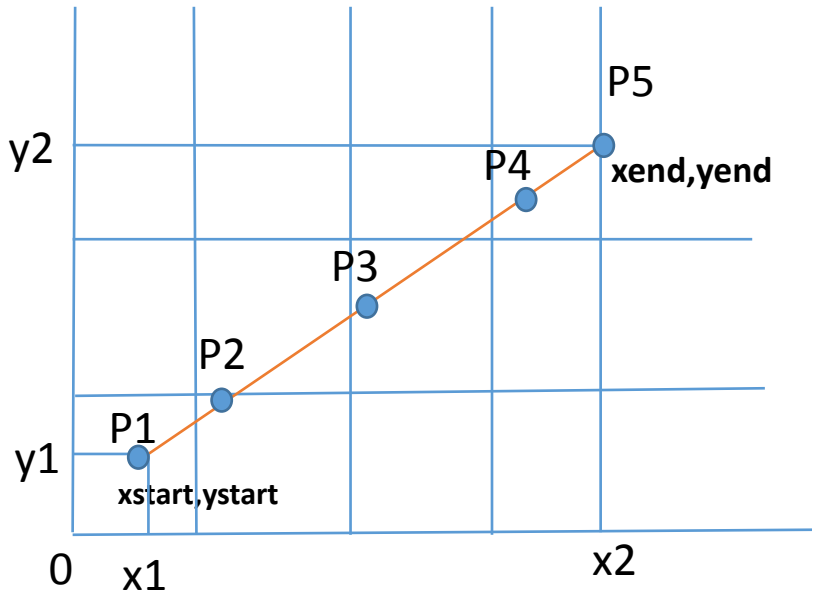
1. The DDA algorithm is a faster method for calculating pixel positions

Unit-III: Graphics Primitives

DDA:

Disadvantages:

1. The accumulation of roundoff error in successive additions of the floating-point increment, however, can cause the calculated pixel positions to drift away from the true line path for long line segments.
2. Furthermore, the rounding operations and floating-point arithmetic in procedure **lineDDA** are still time-consuming.



1.Y=mx+c

2.P1=(xstart,ystart)=1,1
3.P5=(xend,yend)=5,5
4.dx=4
5.dy=4
6. if (abs(dx)>abs(dy))
7. then step=dx else step =dy
8.xinc=dx/step
9.yinc=dy/step

10. m = dy/dx
if|m|<=1, dx>dy, yk+1=yk + m
if |m|>1, dy>dx, xk+1= xk + 1/m
if dx =-1, then yk+1 = yk - m
if dy = -1, then, xk+1 = xk – 1/m

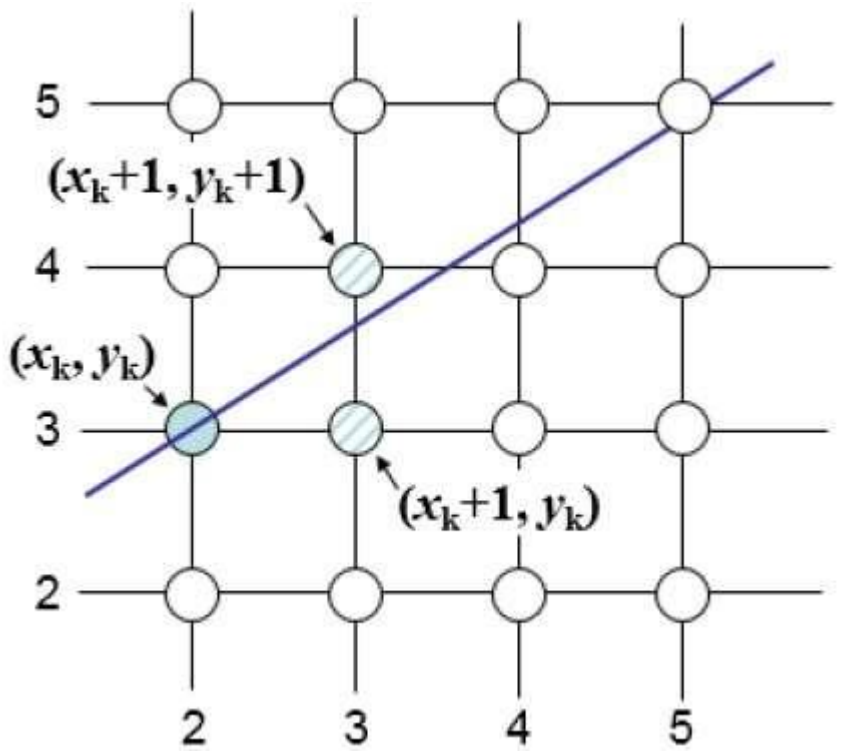
P2=1+1,1+1=2,2

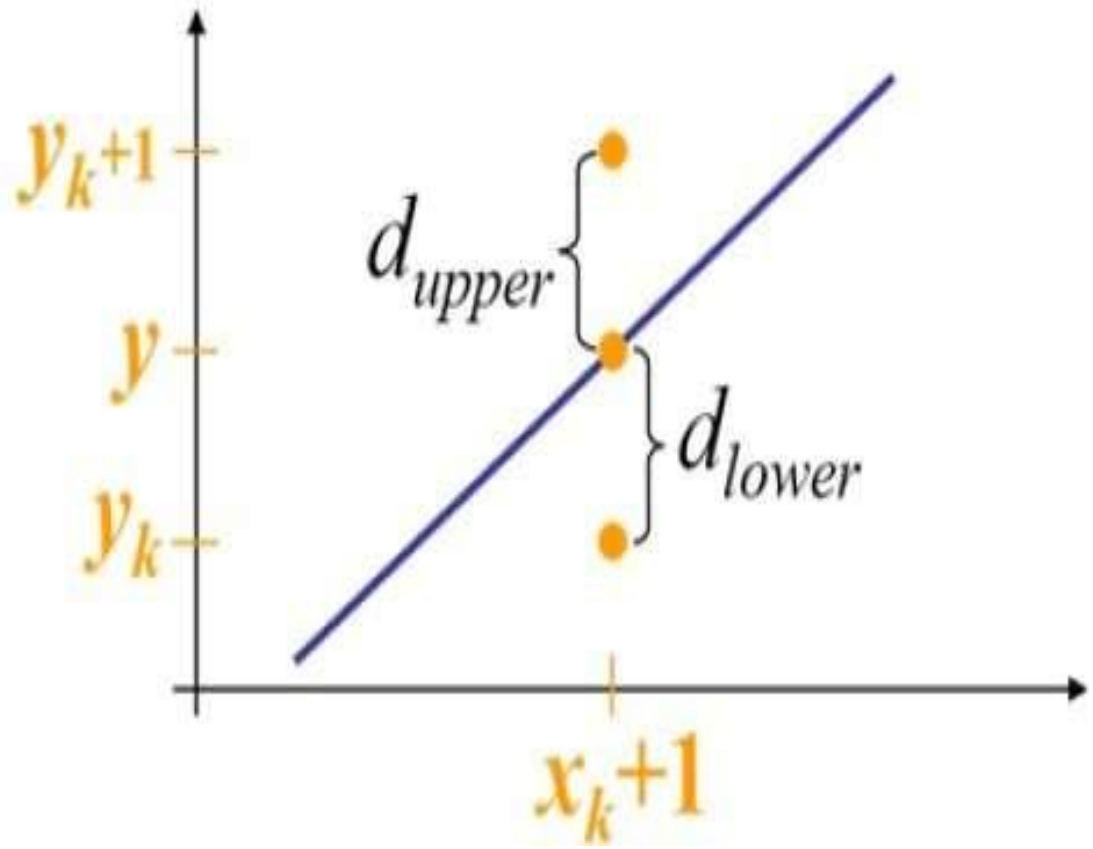
Unit-III: Graphics Primitives

Bresenham's Line Drawing Algorithm:

- An accurate and efficient raster line-generating algorithm, developed by Bresenham, scan converts lines using only incremental integer calculations that can be adapted to display circles and other curves.

Bresenham's Line Drawing Algorithm





Unit-III: Graphics Primitives

Bresenham's Line Drawing Algorithm:

- The Bresenham algorithm is another incremental scan conversion algorithm.
- The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates.
- For example, as shown in the following illustration, from position 2,3, we need to choose between 3,3 and 3,4. we would like the point that is closer to the original line.

Unit-III: Graphics Primitives

Bresenham's Line Drawing Algorithm:

$y = mx + b$

at x_k+1, equation of line will be

$y = m(x_k+1) + b$

then $d_{lower}\ (d_1) = y - y_k$

$d_1 = m(x_k+1) + b - y_k$

similarly $d_{upper}\ (d_2) = y_k+1 - y$

$d_2 = y_k + 1 - m(x_k+1) - b$

Unit-III: Graphics Primitives

Bresenham's Line Drawing Algorithm:

The difference between two separation is : $d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1$ ----------(1)

A decision parameter ***pk for the kth step in the line algorithm can be obtained*** by rearranging Eq. 1 so that it involves only integer calculations. We accomplish this by substituting **$m = \Delta y/\Delta x$, where Δy and Δx are the vertical and** horizontal separations of the endpoint positions, and defining:

decision parameter (p_k) = $\Delta x(d_1 - d_2)$

from Eq.1 $d1 - d2 = 2(\Delta y/\Delta x) (x_k + 1) - 2y_k + 2b - 1$

$$\Delta x (d_1-d_2) = 2\Delta y(x_k +1) - \Delta x\, 2y_k + \Delta x\, 2b - \Delta x$$

$$= 2\Delta y x_k + 2\Delta y - \Delta x\, 2y_k + \Delta x\, 2b - \Delta x$$

$$p_k = 2\Delta y.x_k - 2\Delta x.y_k + C \quad \text{where } C = 2\Delta y + 2b\Delta x - \Delta x \text{ ----------(2)}$$

Unit-III: Graphics Primitives

Bresenham's Line Drawing Algorithm:

- The sign of p_k is the same as sign of (d_1-d_2) as $\Delta x > 0$, parameter C is constant, which is independent of pixel position and will be eliminated in the recursive calculation of p_k.
- If the pixel at y_k is closer to the line path than the pixel at y_{k+1} $(d_1 < d_2)$, then decision parameter p_k is negative, in that case we plot the lower pixel, otherwise we plot the upper pixel.
- Calculate changes along the line occur in unit steps in either x or y directions.

Unit-III: Graphics Primitives

Bresenham's Line Drawing Algorithm:

- Therefore at step k+1, the decision parameter is evaluated from Eq.2 as

$$p_{k+1} = 2\Delta y.\, x_{k+1} - 2\Delta x.\, y_{k+1} + C \text{ ----------(3)}$$

- Subtracting Eq.2 from Eq.3, we have

$$p_{k+1} - p_k = 2\Delta y.\, (x_{k+1} - x_k) - 2\Delta x.\, (y_{k+1} - y_k)$$

but $x_{k+1} = xk + 1$, so that

$$p_{k+1} - p_k = 2\Delta y.\, (x_k + 1 - x_k) - 2\Delta x.\, (y_{k+1} - y_k)$$

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x.\, (y_{k+1} - y_k) \text{ ----------(4)}$$

where term $(y_{k+1} - y_k)$ is either 0 or 1, depending on the sigh of p_k.

Unit-III: Graphics Primitives

Bresenham's Line Drawing Algorithm:

- This recursive calculation of decision parameters is performed at each integer ***x*** position, starting at the left coordinate endpoint of the line. The first parameter, p_0, is evaluated from Eq.2, at the starting pixel (x_0, y_0), $m = \Delta y/\Delta x$

$$p_0 = 2\Delta y - \Delta x$$

Unit-III: Graphics Primitives

Bresenham's Line Drawing Algorithm for |m| < 1:

1. Input the two endpoints of line and store the left endpoint in (x_0, y_0).
2. Load (x_0, y_0) into the frame buffer; that is, plot the first point.
3. Calculate constants Δx, Δy, $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as:

$$p_0 = 2\Delta y - \Delta x$$

4. At each x_k along the line, starting at k = 0, perform the following test:

If $P_k < 0$, the next point to plot is (x_{k+1}, y_k)

$$p_{k+1} = p_k + 2\Delta y$$

Unit-III: Graphics Primitives

Bresenham's Line Drawing Algorithm for |m| < 1:

Otherwise, the next point to plot is (x_{k+1}, y_{k+1}) and

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x \quad p0+1= p0+16-20 = p1 = 6-4=2$$

5. Repeat step 4 , Δx times.

For a line with positive slope greater than 1, we interchange the roles of the x and y directions. That is, we step along the y direction in unit steps and calculate successive x values nearest the line path.

Unit-III: Graphics Primitives

Bresenham's Line Drawing Algorithm for |m| < 1:

Example:

X1= 20 **x2 = 30** **$\Delta x = 30-20=10$**

Y1 = 10 **y2 = 18** **$\Delta y = 18 - 10 = 8$** **m = 8/10 = 0.8**

$2\Delta y = 16$, $2\Delta y - 2\Delta x = -4$

1. **Calculate initial decision parameter: $p_0 = 2\Delta y - \Delta x = 16 - 10 = 6$**
2. **Plot initial value (x1, y1) = (20, 10), and determine successive values along the line path from the decision parameter such as**

Unit-III: Graphics Primitives

Bresenham's Line Drawing Algorithm for |m| < 1:

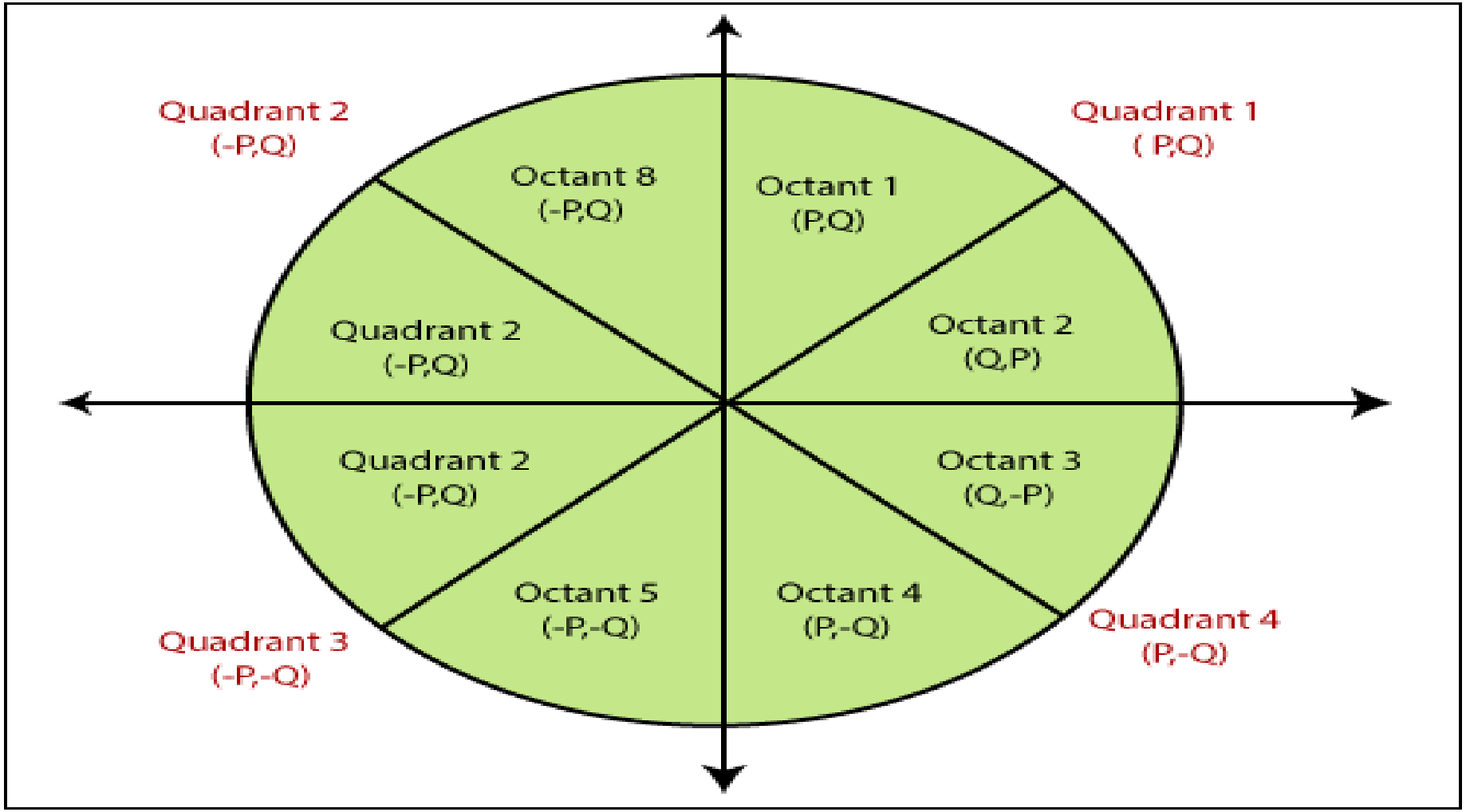
Example:

k	p_k	X_{k+1}, y_{k+1}	k	p_k	X_{k+1}, y_{k+1}
0	6	(21, 11)	5	6	(26, 15)
1	2	(22, 12)	6	2	(27, 16)
2	-2	(23, 12)	7	-2	(28, 16)
3	14	(24, 13)	8	14	(29, 17)
4	10	(25, 14)	9	10	(30, 18)

Unit-III: Graphics Primitives

Mid-Point Circle Drawing Algorithm

- The **mid-point** circle drawing algorithm is an algorithm used to determine the points needed for rasterizing a circle.
- We use the **mid-point** algorithm to calculate all the perimeter points of the circle in the **first octant** and then print them along with their mirror points in the other octants. This will work because a circle is symmetric about it's centre.


Quadrant 2
(-P,Q)
Quadrant 1
(P,Q)
Octant 8
(-P,Q)
Octant 1
(P,Q)
Quadrant 2
(-P,Q)
Octant 2
(Q,P)
Quadrant 2
(-P,Q)
Octant 3
(Q,-P)
Octant 5
(-P,-Q)
Octant 4
(P,-Q)
Quadrant 3
(-P,-Q)
Quadrant 4
(P,-Q)

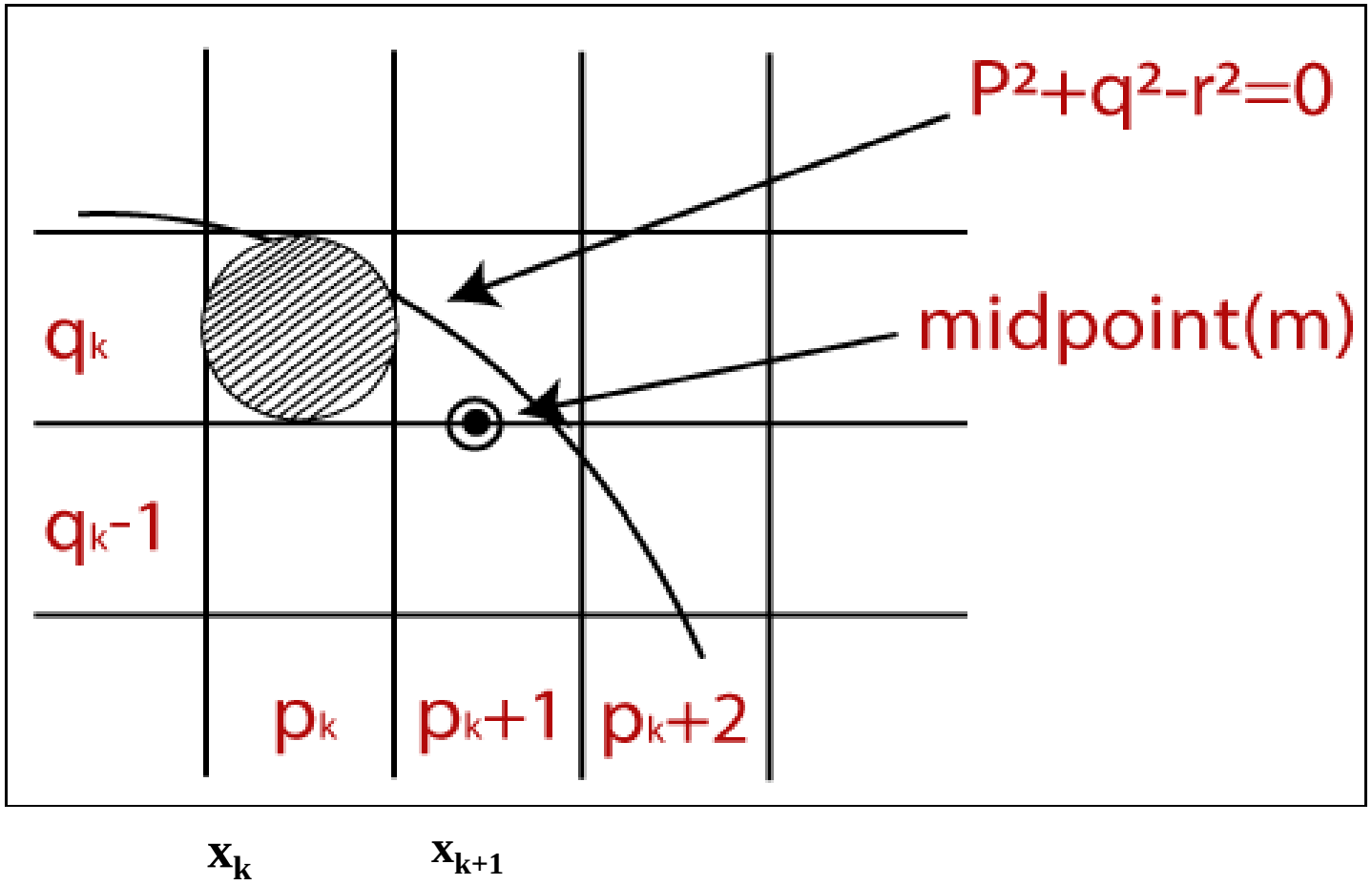
Mid Point Circle

$f(x,y)=x^2 + y^2 - r^2$

If f(x, y) < 0
Point lies inside the circle boundary

If f(x, y) > 0
Point lies outside the circle boundary

If f(x, y) = 0
Point lies on the circle boundary



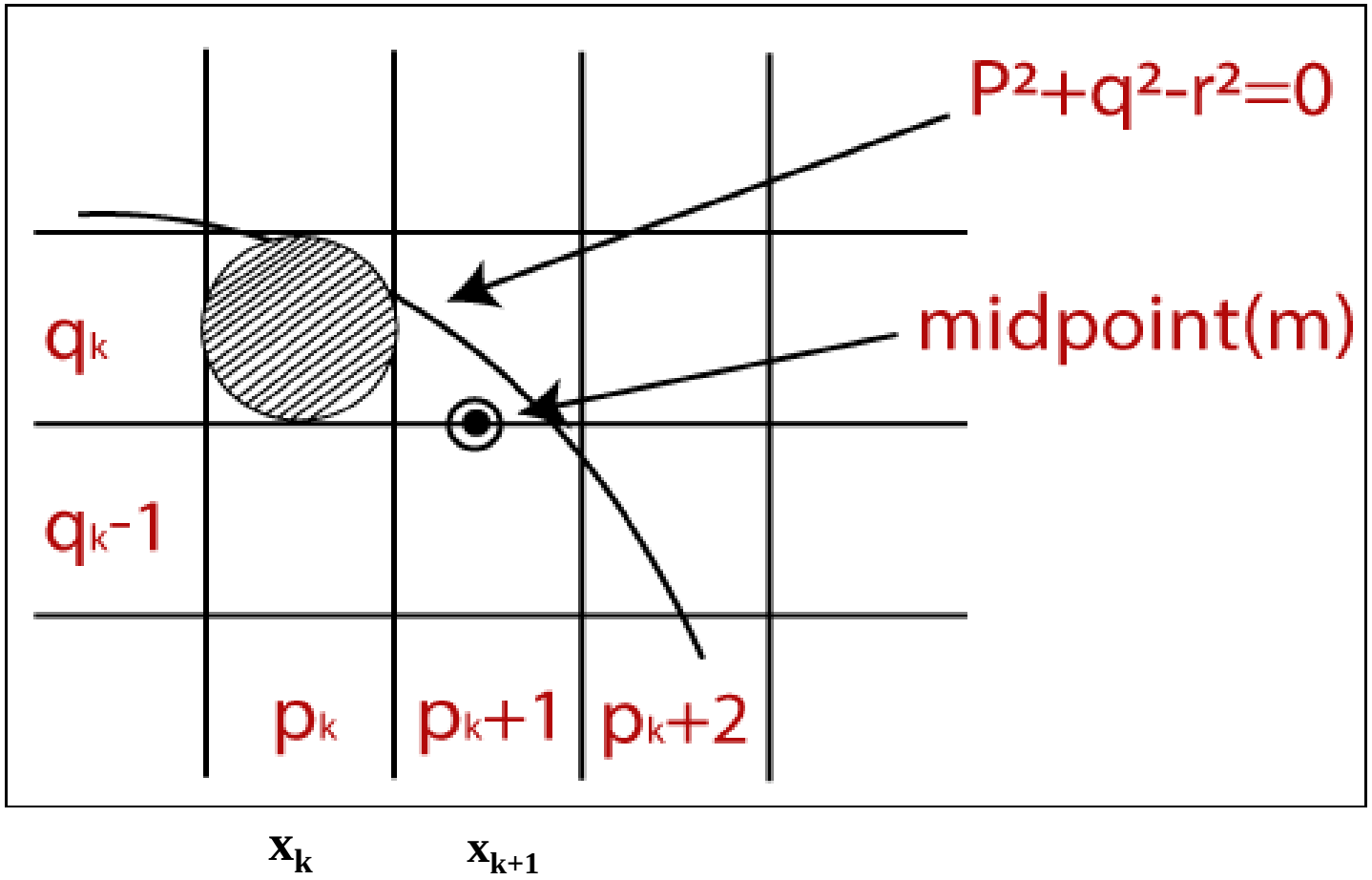
Mid Point Circle

$f(x,y)=x^2 + y^2 - r^2$

Objective is to find next value of y

y_k **or** y_{k-1}

(K+k-1)/2
2k-1/2
K-1/2



Unit-III: Graphics Primitives

Mid-Point Circle Drawing Algorithm:

Decision parameter: $p_k = f_{circle}(x_{k+1}, y_k - 1/2)$

$$= (x_k + 1)^2 + (y_k - 1/2)^2 - r^2 \text{ --------(1)}$$

If $p_k < 0$, this midpoint is inside the circle and the pixel on scan line $\mathbf{y_k}$ is closer to the circle boundary. Otherwise, the midpoint is outside of the circle boundary, and we select the pixel on scan line $\mathbf{y_k - 1}$

Unit-III: Graphics Primitives

Mid-Point Circle Drawing Algorithm:

Successive decision parameters are obtained using incremental calculations.

We obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position $x_{k+1} + 1 = x_k + 2$

$$p_{k+1} = f_{circle}(x_{k+1} + 1, y_{k+1} - 1/2)$$

$$= [(x_k + 1) + 1)]^2 + (y_{k+1} - 1/2)^2 - r^2 \quad \text{----------(2)}$$

Unit-III: Graphics Primitives

Mid-Point Circle Drawing Algorithm:

subtract equation 1 from equation 2:

$$p_{k+1} - p_k = [(x_k + 1) + 1)]^2 + (y_{k+1} - 1/2)^2 - r^2$$

$$- [(x_k + 1)^2 + (y_k - 1/2)^2 - r^2]$$

$$p_{k+1} - p_k = (x_k + 1)^2 + 1 + 2(x_k + 1) + y_{k+1}^2 + 1/4 - y_{k+1} - r^2$$

$$- (x_k + 1)^2 - (y_k - 1/2)^2 + r^2$$

$$p_{k+1} = p_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

Unit-III: Graphics Primitives

Mid-Point Circle Drawing Algorithm:

Find initial decision parameter:

Initial point will be (0, r)

So $p_0 = f_{circle}(1, r-1/2)$

$= 1 + (r-1/2)^2 - r^2$

$= 1 + r^2 + 1/4 - r - r^2$

$p_0 = 5/4 - r = 1 - r$

Unit-III: Graphics Primitives

Mid-Point Circle Drawing Algorithm:

As in Bresenham's line algorithm, the midpoint method calculates pixel positions along the circumference of a circle using integer additions and subtractions, assuming that the circle parameters are specified in integer screen coordinates.

Unit-III: Graphics Primitives

Mid-Point Circle Drawing Algorithm:

We can summarize the steps in the midpoint circle algorithm as follows.

1. **In**put radius r and circle center $(x_c, y_c,)$, and obtain the first point on the circumference of a circle centered on the origin as

   $(x_0, y_0) = (0, r)$

2. Calculate the initial value of the decision parameter as

$$\mathbf{p_0 = 5/4 - r = 1 - r}$$

Unit-III: Graphics Primitives

Mid-Point Circle Drawing Algorithm:

3. **At** each x_k position, starting at k = ***0,*** perform the following test:

If pk < 0, the next point along the circle centered on **(0,O)** is

(x_k, y_k) and

$$\mathbf{p_{k+1} = p_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1}$$

$$\mathbf{= p_k + 2(x_k + 1) + 0 - 0 + 1}$$

$$\mathbf{= p_k + 2(x_k + 1) + 1 \text{ or } p_k + 2x_{k+1} + 1}$$

Unit-III: Graphics Primitives

Mid-Point Circle Drawing Algorithm:

Otherwise, **the** next point along the circle is $(x_{k+1}, y_k - 1)$

$$p_{k+1} = p_k + 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

$$= p_k + 2(x_k + 1) + [(y_k - 1)^2 - y_k)] - (y_k - 1 - y_k) + 1$$

$$= p_k + 2(x_k + 1) + [y_k^2 + 1 - 2y_k - y_k^2] + 1 + 1$$

$$= p_k + 2(x_k + 1) + 1 - 2y_k - 2$$

$$= p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

where $2x_{k+1} = 2x_k + 2$ **and** $2y_{k+1} = 2y_k - 2$

Unit-III: Graphics Primitives

Mid-Point Circle Drawing Algorithm:

4. Determine symmetry points in the other seven octants.

5. Move each calculated pixel position (x, y) onto the circular path cen**tered** on (xc, yc) and plot the coordinate values:

   x=x + xc, y=y + yc

6. Repeat steps 3 through 5 until ***x*** >= y.

Unit-III: Graphics Primitives

Mid-Point Circle Drawing Algorithm:

Example: r = 10 initial point will be (0,r) = (0, 10)

$p_0 = 1 - r = 1 - 10 = -9$

k	p_k	x_{k+1}, y_{k+1}	$2x_{k+1}$	$2y_{k+1}$
0	-9	1, 10	2	20
1	-6	2, 10	4	20
2	-1	3, 10	6	20
3	6	4, 9	8	18
4	-3	5, 9	10	18
5	8	6, 8	12	16
6	5	7, 7	14	14

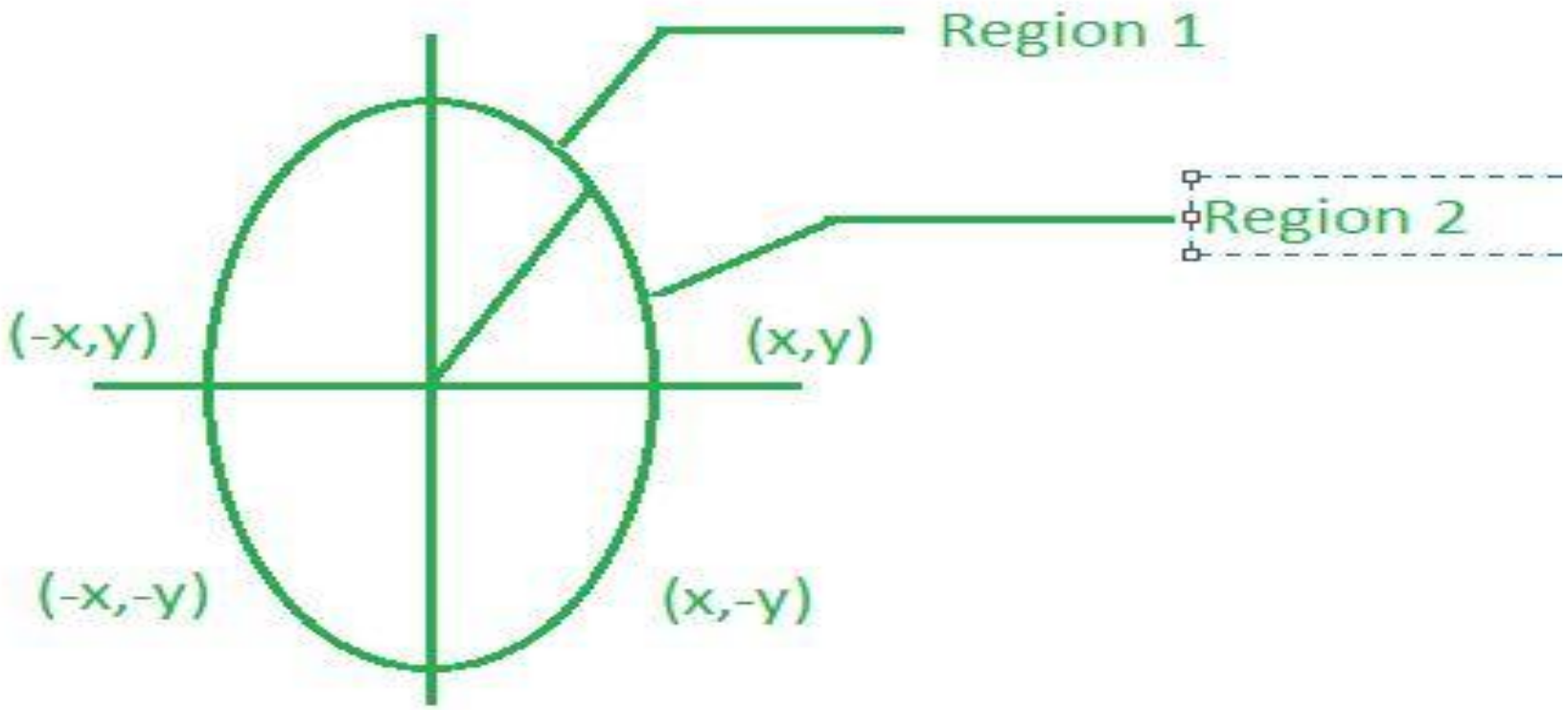
Unit-III: Graphics Primitives

Mid-Point Ellipse Drawing Algorithm

- The Mid-point Ellipse algorithm is used to draw an ellipse in computer graphics.
- Midpoint ellipse algorithm plots(finds) points of an ellipse on the first quadrant by dividing the quadrant into two regions. Each point(x, y) is then projected into other three quadrants (-x, y), (x, -y), (-x, -y) i.e. it uses 4-way symmetry.

Unit-III: Graphics Primitives

Mid-Point Ellipse Drawing Algorithm



Unit-III: Graphics Primitives

Mid-Point Ellipse Drawing Algorithm

$f_{ellipse}(x, y)=r_y^2x^2+r_x^2y^2-r_x^2r_y^2$

$f_{ellipse}(x, y)<0$ then (x, y) is inside the ellipse.

$f_{ellipse}(x, y)>0$ then (x, y) is outside the ellipse.

$f_{ellipse}(x, y)=0$ then (x, y) is on the ellipse.

Unit-III: Graphics Primitives

Mid-Point Ellipse Drawing Algorithm:

Decision parameter: $p1_k = f_{ellipse}(x_{k+1}, y_k - 1/2)$

$$= r_y^2 (x_k + 1)^2 + r_x^2 (y_k - 1/2)^2 - r_x^2 r_y^2 \text{ --------(1)}$$

If $p1_k < 0$, this midpoint is inside the ellipse and the pixel on scan line y_k is closer to the ellipse boundary. Otherwise, the midpoint is outside of the ellipse boundary, and we select the pixel on scan line $y_k - 1$

Unit-III: Graphics Primitives

Mid-Point Ellipse Drawing Algorithm:

$$p1_{k+1} = f_{ellipse}(x_{k+1} + 1, y_{k+1} - 1/2)$$

$$= r_y^2 [(x_k + 1) + 1)]^2 + r_x^2 (y_{k+1} - 1/2)^2 - r_x^2 r_y^2 \quad \text{--------(2)}$$

Mid-Point Circle Drawing Algorithm:

subtract equation 1 from equation 2:

$$p1_{k+1} - p1_k = r_y^2 [(x_k + 1) + 1)]^2 + r_x^2 (y_{k+1} - 1/2)^2 - r_x^2 r_y^2$$
$$- r_y^2 (x_k + 1)^2 - r_x^2 (y_k - 1/2)^2 + r_x^2 r_y^2$$

$$p1_{k+1} - 1p_k = r_y^2 [(x_k + 1)^2 + 1 + 2(x_k + 1)] + r_x^2 (y_{k+1}^2 - y_{k+1} + 1/4) - r_x^2 r_y^2$$
$$- r_y^2 (x_k + 1)^2 - r_x^2 (y_k^2 + 1/4 - y_k) + r_x^2 r_y^2$$

$$p1_{k+1} - 1p_k = r_y^2 + 2r_y^2 (x_k + 1) + r_x^2 (y_{k+1}^2 - y_{k+1}) - r_x^2 (y_k^2 - y_k)$$

$$p1_{k+1} = p1_k + r_y^2 + 2r_y^2 (x_k + 1) + r_x^2 (y_{k+1}^2 - y_k^2) - r_x^2 (y_{k+1} - y_k)$$

Mid-Point Circle Drawing Algorithm:

if $p1_k < 0$, next value of y will be y_k

$$p1_{k+1} = p1_k + r_y^2 + 2r_y^2(x_k + 1)$$

If $p_1k > 0$, next value of y will be $y_k - 1$

$$p1_{k+1} = p1_k + r_y^2 + 2r_y^2(x_k+1) + r_x^2(1 - 2y_k) + r_x^2$$

$$= p1_k + r_y^2 + 2r_y^2(x_k+1) + 2r_x^2 - 2r_x^2 y_k$$

$$= p1_k + r_y^2 + 2r_y^2(x_k+1) - 2r_x^2 (y_k - 1)$$

$$= p1_k + r_y^2 + 2r_y^2 x_{k+1} + 2r_x^2 y_{k+1}$$

Unit-III: Graphics Primitives

Mid-Point Ellipse Drawing Algorithm:

Find initial decision parameter:

Initial point will be $(0, r_y)$

So $p1_0 = f_{ellipse}(1, r_y - 1/2)$

$$= r_y^2 (1)^2 + r_x^2 (r_y - 1/2)^2 - r_x^2 r_y^2$$

$$= r_y^2 + r_x^2 (r_y^2 + 1/4 - r_y) - r_x^2 r_y^2$$

$$\mathbf{p1_0 = r_y^2 + 1/4\, r_x^2 - r_x^2 r_y}$$

Unit-III: Graphics Primitives

Mid-Point Ellipse Drawing Algorithm:

Decision parameter for region 2:

$$p2_k = f_{ellipse}(x_k + 1/2, y_k - 1)$$

$$= r_y^2 (x_k + 1/2)^2 + r_x^2 (y_k - 1)^2 - r_x^2 r_y^2 \text{ -------(3)}$$

If $p2_k < 0$, this midpoint is inside the ellipse and the pixel on scan line $\mathbf{x_k}$ **+1** is closer to the ellipse boundary. Otherwise, the midpoint is outside of the ellipse boundary, and we select the pixel on scan line $\mathbf{x_k}$

Unit-III: Graphics Primitives

Mid-Point Ellipse Drawing Algorithm:

$$p2_{k+1} = f_{ellipse}(x_{k+1} + 1/2, y_{k+1} - 1)$$

$$= r_y^2 [(x_{k+1} + 1/2)]^2 + r_x^2 ((y_k - 1) - 1)^2 - r_x^2 r_y^2 \quad \text{--------(4)}$$

Mid-Point Circle Drawing Algorithm:

subtract equation 3 from equation 4:

$$p2_{k+1} - p2_k = r_y^2 [(x_{k+1} + 1/2)]^2 + r_x^2 ((y_k - 1) - 1)^2 - r_x^2 r_y^2$$

$$- r_y^2 (x_k + 1/2)^2 - r_x^2 (y_k - 1)^2 + r_x^2 r_y^2$$

$$p2_{k+1} - p2_k = r_y^2 [(x_{k+1} + 1/2)]^2 + r_x^2 ((y_k - 1)^2 + 1 - 2(y_k - 1)) - r_x^2 r_y^2$$

$$- r_y^2 (x_k + 1/2)^2 - r_x^2 (y_k - 1)^2 + r_x^2 r_y^2$$

$$p2_{k+1} = p2_k + r_y^2 [(x_{k+1} + 1/2)^2 - (x_k + 1/2)^2] + r_x^2 (1 - 2(y_k - 1))$$

Mid-Point Circle Drawing Algorithm:

if $p2_k > 0$, next value of x will be x_k

$$p2_{k+1} = p2_k + r_y^2 [(x_{k+1} + 1/2)^2 - (x_k + 1/2)^2] + r_x^2 (1 - 2(y_k - 1))$$

$$= p2_k + r_y^2 [(x_k + 1/2)^2 - (x_k + 1/2)^2] + r_x^2 (1 - 2(y_k - 1))$$

$$= p2_k - 2r_x^2 y_{k+1} + r_x^2$$

Mid-Point Circle Drawing Algorithm:

if $p2_k < 0$, next value of x will be $x_k + 1$

$$p2_{k+1} = p2_k + r_y^2 [(x_k + 1/2) + 1)^2 - (x_k + 1/2)^2] + r_x^2 (1 - 2(y_k - 1))$$

$$= p2_k - 2r_x^2 y_{k+1} + r_x^2 + 2r_y^2 x_{k+1}$$

Unit-III: Graphics Primitives

Mid-Point Ellipse Drawing Algorithm:

Find initial decision parameter for region 2:

Initial point will be (x_0, y_0)

So $p2_0 = f_{ellipse}(x_0 + 1/2, y_0 - 1)$

$$= r_y^2 (x_0 + 1/2)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

Unit-III: Graphics Primitives

Mid-Point Ellipse Drawing Algorithm:

Example: rx = 8 ry = 6 initial point will be (0,ry) = (0, 6)

$p1_0 = -332$

k	p_k	x_{k+1}, y_{k+1}	$2r_y^2x_{k+1}$	$2r_x^2y_{k+1}$
0	-332	1,6	72	768
1	-224	2,6	144	768
2	-44	3,6	216	768
3	208	4,5	288	640
4	-108	5,5	360	640
5	288	6,4	432	512
6	244	7,3	504	384
			Here $2r_y^2x_{k+1} > 2r_x^2y_{k+1}$	

Unit-III: Graphics Primitives

Mid-Point Ellipse Drawing Algorithm:

So for region2 (x_0, y_0) will be (7,3)

$P2_0$ = -151

k	p_k	x_{k+1}, y_{k+1}	$2r_y^2x_{k+1}$	$2r_x^2y_{k+1}$
0	-151	8, 2	576	256
1	233	8, 1	576	128
2	745	8, 0		
3				

Unit-III: Graphics Primitives

Filled Area Primitives: There are two basic approaches to area filling on raster systems.

- One way to fill an area is to determine the overlap intervals for scan lines that cross the area.
- Another method for area filling is to start from a given interior position and paint outward from this point until we encounter the specified boundary conditions.

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Approach

The scan-line approach is typically used in general graphics packages to fill polygons, circles, ellipses, and other simple curves. All methods starting from an interior point are useful with more complex boundaries and in interactive painting systems.

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Approach

The scan-line approach is typically used in general graphics packages to fill polygons, circles, ellipses, and other simple curves. All methods starting from an interior point are useful with more complex boundaries and in interactive painting systems.

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm

- Scan-line filling is basically filling up of polygons using horizontal lines or scan-lines.
- The purpose of the SLPF algorithm is to fill (color) the interior pixels of a polygon given only the vertices of the figure.
- This algorithm works by intersecting scan-line with polygon edges and fills the polygon between pairs of intersections.

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm

F: 7,1
E: 13,5
M= 5-1/13-7=2/3

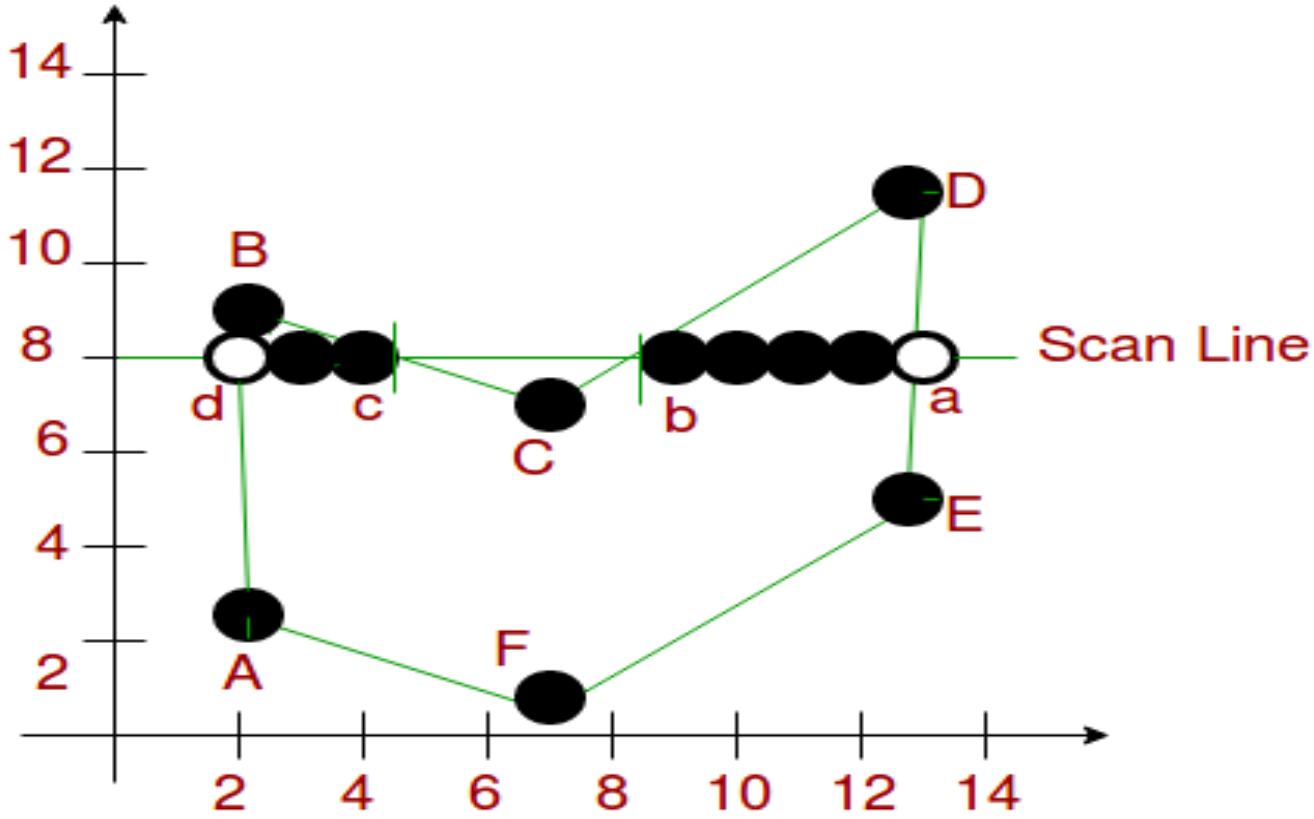


Figure 1

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm

To efficiently perform a polygon fill,

1. First store the polygon boundary in a ***sorted edge*** table that contains a11 the information necessary to process the scan lines efficiently. Proceeding around the edges in either a clockwise or a counterclockwise order.
2. Use bucket sort to store the edges, sorted on the smallest y value of each edge, in the correct scan-line positions.
3. Only non horizontal edges are entered into the sorted edge table.

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm

4. Each entry in the table for a particular scan line contains the maximum y value for that edge, the x-intercept value (at the lower vertex) for the edge, and the inverse slope of the edge.
5. For each scan line, the edges are in sorted order from left to right.

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm

6. Next, process the scan lines from the bottom of the polygon to its top, producing an active edge list for each scan line crossing the polygon boundaries.
7. The active edge list for a scan line contains all edges crossed by that scan line, with iterative coherence calculations used to obtain the edge intersections.

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm

- Many edges intersected by scan-line i will also be intersected by scan-line i+1
- Formula for scan-line s is y = s, for an edge is y = mx + b

There intersection is: $s = mx_s + b$

$$x_s = (s-b)/m \text{ ----------(1)}$$

For scan-line s+1: $x_{s+1} = (s+1-b)/m$ **----------(2)**

$$x_{s+1} = x_s + 1/m \text{ ----------(3)}$$

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm

For each scan line:

1. Find the intersections of the scan line with all edges of the polygon.
2. Sort the intersections by increasing x-coordinate.
3. Fill in all pixels between pairs of intersections.

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm

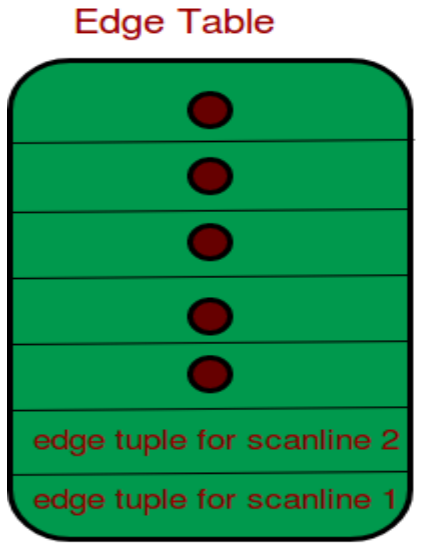
For scan line number 8 as shown in Figure 1.

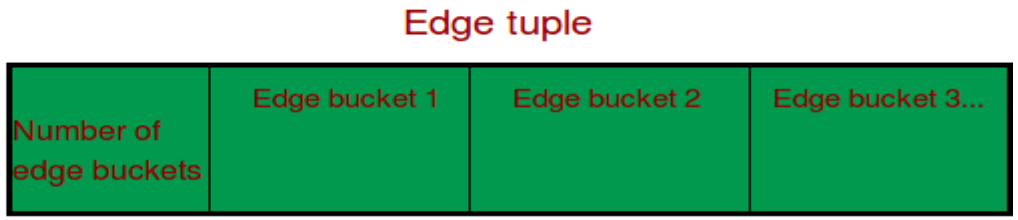
- The sorted list of x-coordinates is (2,4,9,13)
- Therefore fill pixels with x-coordinates 2-4 and 9-13.

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm





ymax - max y - coordinate of edge
xofymin - x-coordinate of lowest edge point, updated at every scanline while scanline filling
slopeinverse - inverse of edge slope (horizontal lines are not stored)

Scanline Filling

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm

Components of Polygon fill:

Edge Buckets: It contains an edge's information. The entries of edge bucket vary according to data structure.

In the example , there are three edge buckets namely ymax,xofymin, slopinverse

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm

Components of Polygon fill:

Edge Table: It consists of several edge lists -> holds all of the edges that compose the figure. When creating edges, the vertices of the edge need to be ordered from left to right and the edges are maintained in increasing yMin order.

Filling is complete once all of the edges are removed from the ET.

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm

Components of Polygon fill:

Active List: IT maintains the current edges being used to fill in the polygon.

Edges are pushed into the AL from the Edge Table

when an edge's yMin is equal to the current scan-line being processed.

The Active List will be re-sorted after every pass.

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm

Example: Figure 1

Scan-Line Number	ymax	xmin	1/m		ymax	xmin	1/m	
12								
11								
10								
9								
8								
7								
6	9	7	-5/3	→	12	7	1	NULL

Unit-III: Graphics Primitives

Filled Area Primitives:

Scan – Line Polygon Filling Algorithm

Example: Figure 1

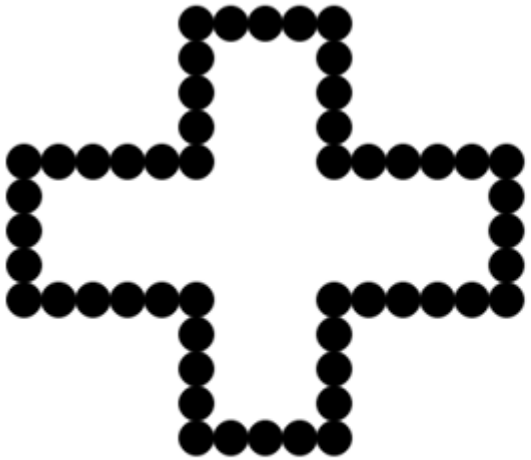
Scan-Line Number	ymax	xmin	1/m		ymax	xmin	1/m	
5	12	5	0	NULL				
4								
3								
2	9	2	0	NULL				
1	2	7	-5	→	5	7	3/2	NULL
0								

Unit-III: Graphics Primitives

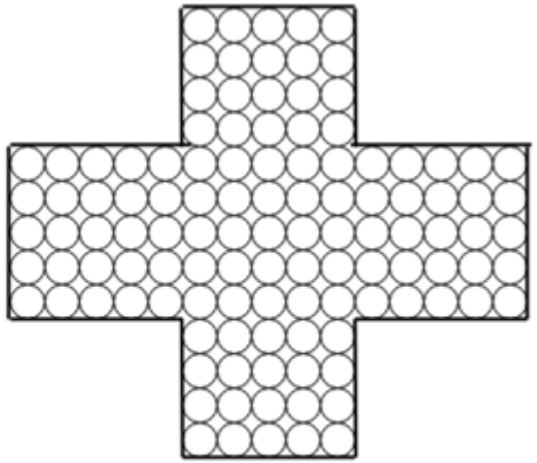
Filled Area Primitives:

Boundary Fill Algorithm

Region filling is the process of filling image or region. Filling can be of boundary or interior region as shown in fig. Boundary Fill algorithms are used to fill the boundary and flood-fill algorithm are used to fill the interior.



Boundary Filled Region



Interior or Flood Filled Region

Unit-III: Graphics Primitives

Filled Area Primitives:

Boundary Fill Algorithm

- It follows an approach where the region filling begins from some extent residing inside the region and paint the inside towards the boundary.
- In case the boundary contains single colour the fill algorithm continues within the outward direction pixel by pixel until the boundary colour is encountered.
- The boundary-fill algorithm is often mainly implemented within the interactive painting packages, where the inside points are easily chosen.

Unit-III: Graphics Primitives

Filled Area Primitives:

Boundary Fill Algorithm

- The functioning of the boundary-fill starts by accepting the coordinates of an indoor point (x, y), a boundary colour and fill colour becomes the input.
- Beginning from the (x, y) the method checks neighbouring locations to spot whether or not they are a part of the boundary colour.
- If they're not from the boundary colour, then they're painted with the fill colour, and their adjacent pixels are tested against the condition.
- The process ends when all the pixels up until the boundary colour for the world are checked.

Unit-III: Graphics Primitives

Filled Area Primitives:

Flood Fill Algorithm

- Flood fill algorithm is also known as a seed fill algorithm. It determines the area which is connected to a given node in a multi-dimensional array.
- This algorithm works by filling or recolouring a selected area containing different colours at the inside portion and therefore the boundary of the image.
- It is often illustrated by a picture having a neighbourhood bordered by various distinct colour regions.
- To paint such regions we replace a specific interior colour instead of discovering a boundary colour value.

Unit-III: Graphics Primitives

Filled Area Primitives:

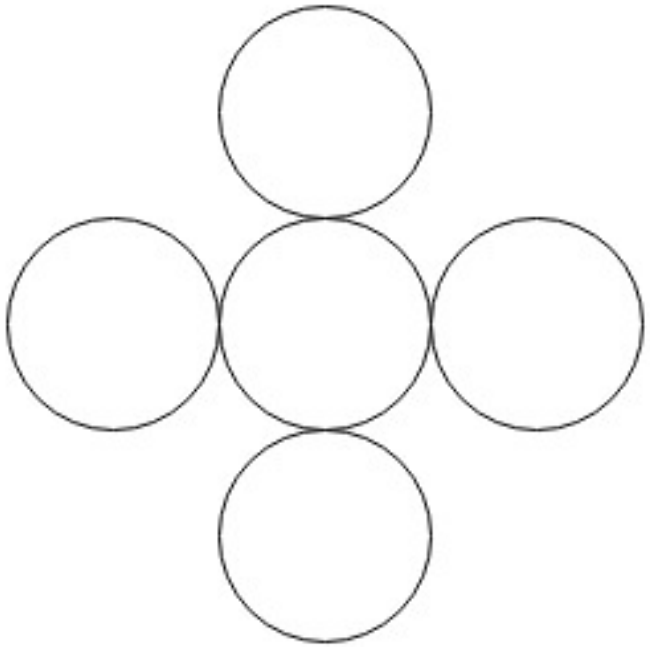
Flood Fill Algorithm

- There are two methods which will be used for creating endless boundary by connecting pixels – 4-connected and 8-connected approach.
- In the 4-connected method, the pixel can have at maximum four neighbours that are positioned at the proper, left, above and below the present pixel.
- On the contrary, in the 8-connected method, it can have eight, and the neighbouring positions are checked against the four diagonal pixels.

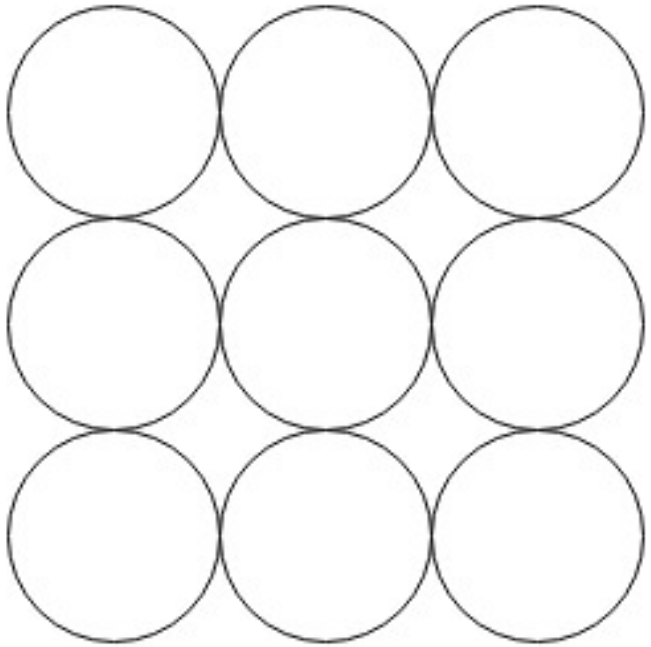
Unit-III: Graphics Primitives

Filled Area Primitives:

Flood Fill Algorithm



Four Connected



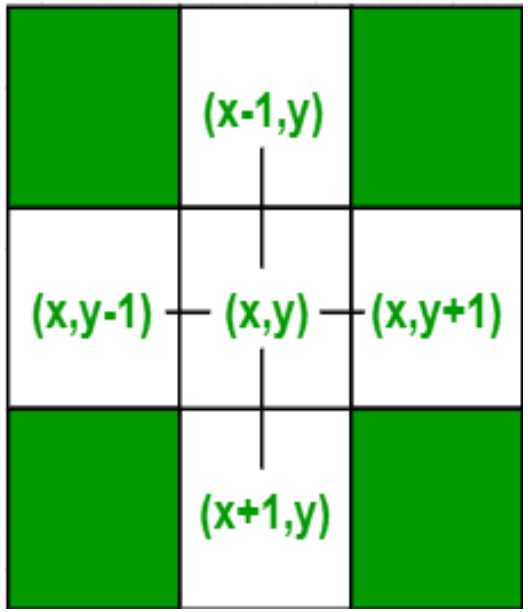
Eight Connected

Unit-III: Graphics Primitives

Filled Area Primitives:

Boundary Fill Algorithm

```
void boundaryFill4(int x, int y, int fill_color,int boundary_color)
{
if(getpixel(x, y) != boundary_color && getpixel(x, y) != fill_color)
 {
putpixel(x, y, fill_color);
 boundaryFill4(x + 1, y, fill_color, boundary_color);
boundaryFill4(x, y + 1, fill_color, boundary_color);
 boundaryFill4(x - 1, y, fill_color, boundary_color);
boundaryFill4(x, y - 1, fill_color, boundary_color);
}
}
```



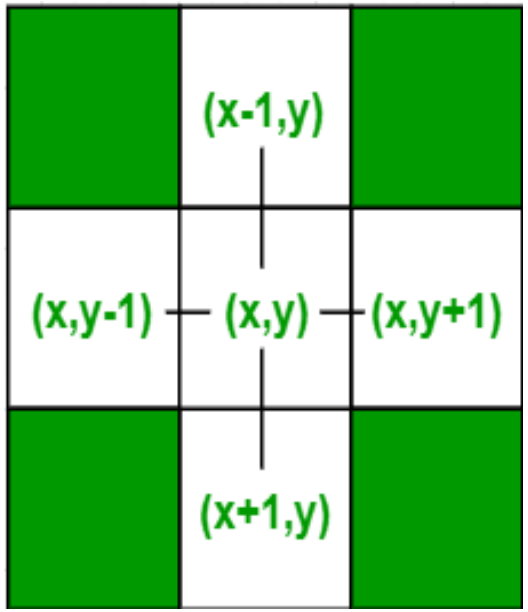
Unit-III: Graphics Primitives

Filled Area Primitives:

Flood Fill Algorithm

floodfill(x, y, newcolor, oldcolor)

1) If x or y is outside the screen, then return.
2) If color of getpixel(x, y) is same as oldcolor, then
3) Recur for top, bottom, right and left.
4) floodFill(x+1, y, newcolor, oldcolor);
5) floodFill(x-1, y, newcolor, oldcolor);
6) floodFill(x, y+1, newcolor, oldcolor);
7) floodFill(x, y-1, newcolor, oldcolor);



Unit-III: Graphics Primitives

Filled Area Primitives:

S.No	Boundary Fill	Flood Fill
1	It can only process the image containing single boundary colour.	It can process the image containing more than one boundary colours.
2	Boundary-fill algorithm is faster than the Flood-fill algorithm.	Flood-fill algorithm is comparatively slower than the Boundary-fill algorithm.
3	In Boundary-fill algorithm Interior points are painted by continuously searching for the boundary colour.	In Flood-fill algorithm a random colour can be used to paint the interior portion then the old one is replaced with a new one.

Unit-III: Graphics Primitives

Filled Area Primitives:

S.No	Boundary Fill	Flood Fill
4	Memory consumption is relatively low in Boundary-fill algorithm.	It requires huge amount of memory.
5	The complexity of Bounndary-fill algorithm is high.	Flood-fill algorithms are simple and efficient.

Unit-III: Graphics Primitives

Filled Area Primitives:

Inside – Outside Test

This method is also known as **counting number method**. While filling an object, we often need to identify whether particular point is inside the object or outside it. There are two methods by which we can identify whether particular point is inside an object or outside.

- **Odd-Even Rule**
- **Nonzero winding number rule**

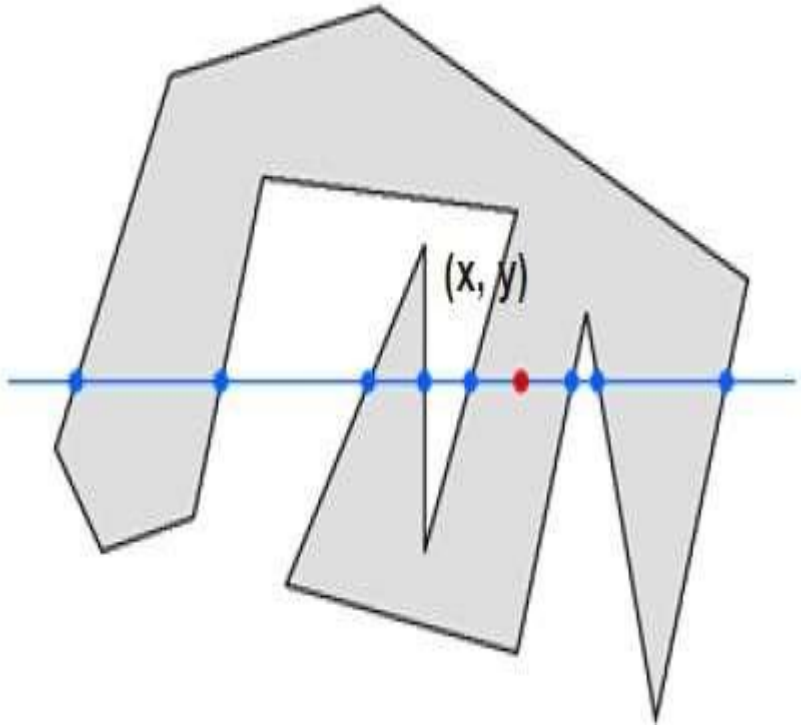
Unit-III: Graphics Primitives

Filled Area Primitives:

Inside – Outside Test

Odd-Even Rule

1. In this technique, we will count the edge crossing along the line from any point (x,y) to infinity.
2. If the number of interactions is odd, then the point (x,y) is an interior point; and if the number of interactions is even, then the point (x,y) s an exterior point.



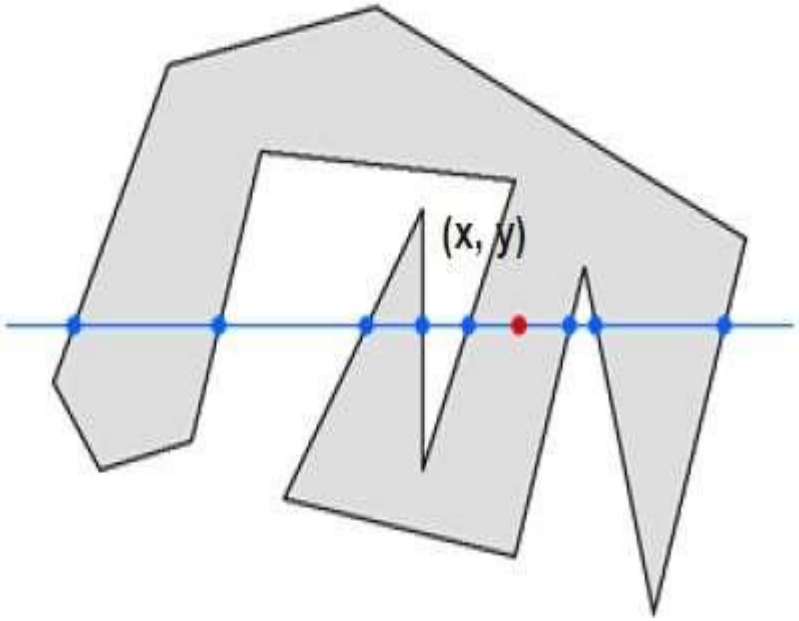
Unit-III: Graphics Primitives

Filled Area Primitives:

Inside – Outside Test

Odd-Even Rule

3. From figure, we can see that from the point (x,y) the number of intersection point on the left side is 5 and on the right side is 3.
4. From both ends, the number of intersection points is odd, so the point is considered within the object.
5. To obtain an accurate edge count, we must be **sure** that the line path we **choose** does not intersect any polygon vertices.



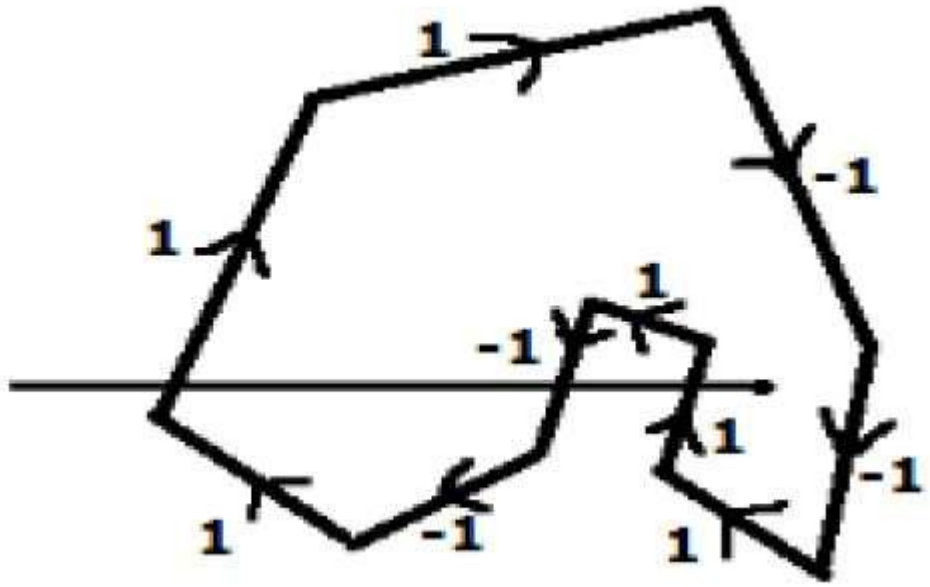
Unit-III: Graphics Primitives

Filled Area Primitives:

Inside – Outside Test

Nonzero Winding Number Rule

1. This method is also used with the simple polygons to test the given point is interior or not
2. Give directions to all the edges of the polygon. Draw a scan line from the point to be test towards the left Most of X direction.



Unit-III: Graphics Primitives

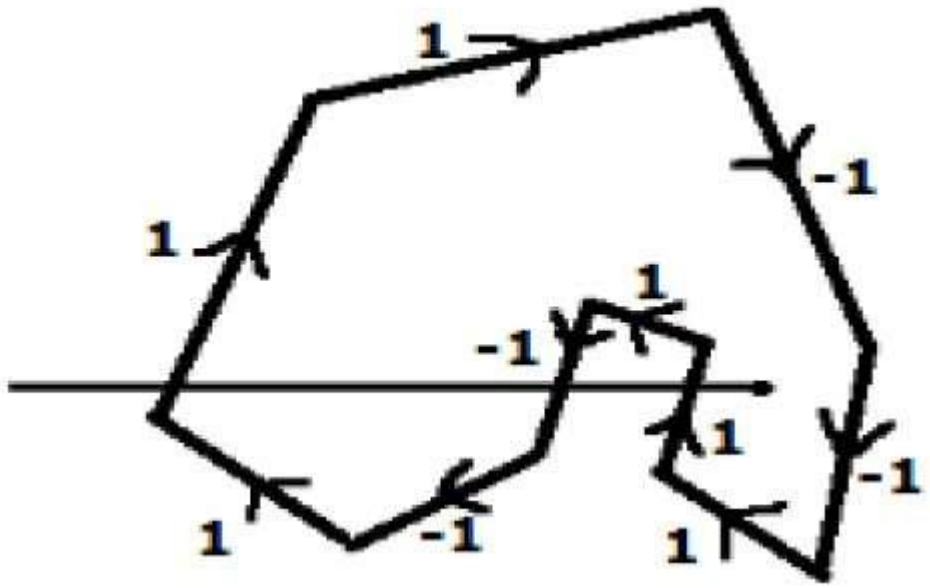
Filled Area Primitives:

Inside – Outside Test

Nonzero Winding Number Rule

a. Give the value 1 to all the edges which are going to upward direction and all other -1 as direction values.

b. Check the edge direction values from which the scan line is passing and sum up them.



Unit-III: Graphics Primitives

Filled Area Primitives:

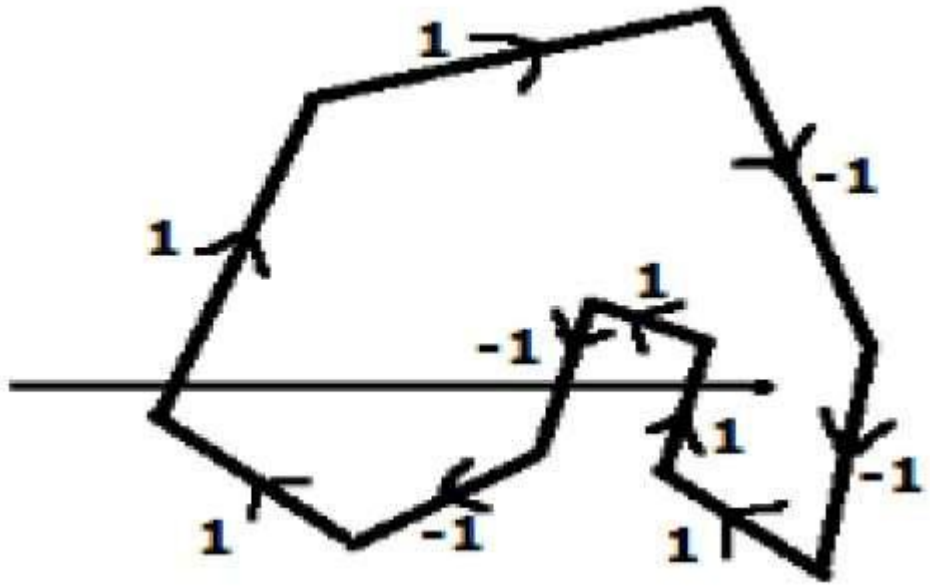
Inside – Outside Test

Nonzero Winding Number Rule

c. If the total sum of this direction value is non-zero, then this be tested is an **interior point,** otherwise it is an **exterior point**

d. In the figure, we sum up the direction values from which the scan line is passing then the total is 1 – 1 + 1 = 1; which is non-zero.

So the point is said to be an interior point.



Unit-III: Graphics Primitives

Character Generation

- Letters, numbers and other characters can be displayed in a variety of sizes and styles. The design style for a set of characters is called a ***Typeface*** or ***Font***.
- Typefaces or Fonts can be divided into two broad groups: ***Serif*** and ***Sans Serif***. Serif type has small lines at the ends of the main character strokes (called accents), while Sans Serif does not have accents.
- Serif type is generally more readable, that is, it is easier to read in large amount of texts. On the other hand, the individual characters in sans-serif type are easier to identify. So, Sans-Serif type is said to be more legible and is good for labelling and short heading.

Unit-III: Graphics Primitives

Character Generation Methods

1. Bitmap Method:

- It is also called dot matrix because in this method characters are represented by an array of dots in the matrix form.
- It is a two dimensional array having columns and rows. An 5x7 array is commonly used to represent characters. However 7x9 and 9x13 arrays are also used. Higher resolution devices such as inkjet printer or laser printer may use character arrays that are over 100x100.

Unit-III: Graphics Primitives

Character Generation Methods

1. Bitmap Method:

- In this, the character shapes in a particular font are represented in form of rectangular grid patterns. The set of characters are then referred to as a Bitmap Font.
- Bitmap fonts are simplest to define and display.
- The character grid only needs to be mapped to a frame-buffer position. But bitmap fonts require more space.
- It is possible to generate different sizes and other variations, such as bold, italics, etc., but usually the results are not good.

Unit-III: Graphics Primitives

Character Generation Methods

2. Stroke Method

- This method uses small line segments to generate a character.
- The small series of line segments are drawn like a stroke of pen to form a character.
- We can build our own stroke method character generator by calls to the line drawing algorithm.
- Here it is necessary to decide which line segments are needed for each character and then drawing these segments using line drawing algorithm.

Unit-III: Graphics Primitives

Character Generation

3. Starbust method:

- In this method a fix pattern of line segments are used to generate characters. Out of these 24 line segments, segments required to display for particular character are highlighted. This method of character generation is called starbust method because of its characteristic appearance.
- The starbust patterns for characters A and M. the patterns for particular characters are stored in the form of 24 bit code, each bit representing one line segment. The bit is set to one to highlight the line segment; otherwise it is set to zero. For example, 24-bit code for Character A is 0011 0000 0011 1100 1110 0001 and for character M is 0000 0011 0000 1100 1111 0011.

Unit-III: Graphics Primitives

Character Generation

Starbust method: Disadvantages

This method of character generation has some disadvantages. They are

1. The 24-bits are required to represent a character. Hence more memory is required
2. Requires code conversion software to display character from its 24-bit code
3. Character quality is poor. It is worst for curve shaped characters.

Unit-III: Graphics Primitives

Attributes

- The features or characteristics of an output primitive are known as ***Attribute***. In other words, any parameter that affects the way a primitive is to be displayed is known as ***Attribute***.
- Some attributes, such as colour and size, are basic characteristics of primitive.
- Some attributes control the basic display properties of primitives. For example, lines can be dotted or dashed, thin or thick. Areas can be filled with one colour or with multiple colours pattern. Text can appear from left to right, slanted or vertical.

Unit-III: Graphics Primitives

Attributes

Line Attributes:

Basic attributes of a straight line are its type, its width, and its colour. In some graphics packages, line can also be displayed using selected pen or brush options.

1. Line Type
2. Line Width
3. Pen and Brush Options
4. Colour

Unit-III: Graphics Primitives

Attributes

1. Line Type:

- The line type attribute includes solid lines, dashed lines, and dotted lines.
- We modify the line drawing algorithm to generate such lines by setting the length and space.
- A dashed line could be displayed by generating spaces that is equal to length of solid part. A dotted line can be displayed by generating very short dashes with spacing equal to or greater than the dash size. Similar methods are used t o produce other line-type variations.

Unit-III: Graphics Primitives

Attributes

1. Line Type:

- Raster line-algorithms displays line type attribute by plotting pixels. For various dashed, dotted patterns, the line-drawing algorithms outputs part of pixels followed by spaces.
- Plotting dashes with a fixed number of pixels results in unequal-length dashes for different line angles. For example, the length of dash diagonally is more than horizontal dash for same number of pixels.
- For precision drawings, dash length should remain approximately same for any line angle. For this, we can adjust the pixel number according to line slope.

Unit-III: Graphics Primitives

Attributes

2. Line Width:

- A line with more width can be displayed as parallel lines on a video monitor. In raster lines, a standard width line is generated with single pixels at each point.
- Width lines are displayed by plotting additional pixels along next parallel line paths.
- For lines with slope less than 1, we can display thick lines by plotting a vertical length of pixels at each x position along the line.
- Similarly, for lines with slope greater than 1, we can plot thick lines with horizontal widths for each point.

Unit-III: Graphics Primitives

Attributes

2. Line Width:

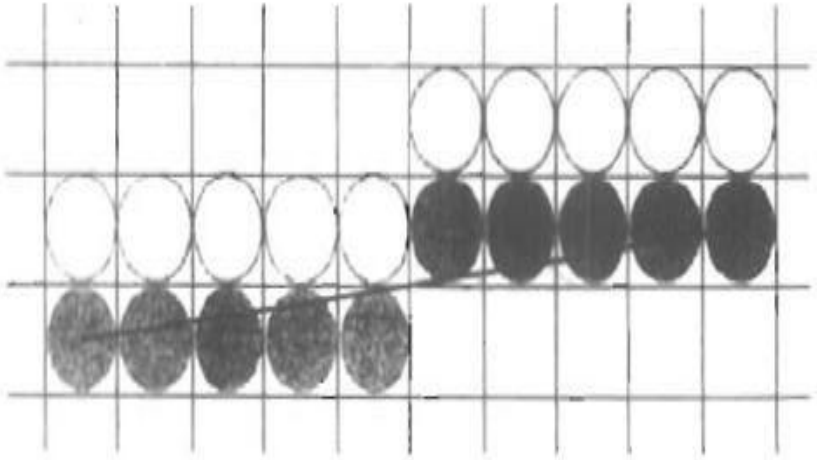


Figure 4-3
Double-wide raster line with slope $|m| < 1$ generated with vertical pixel spans.

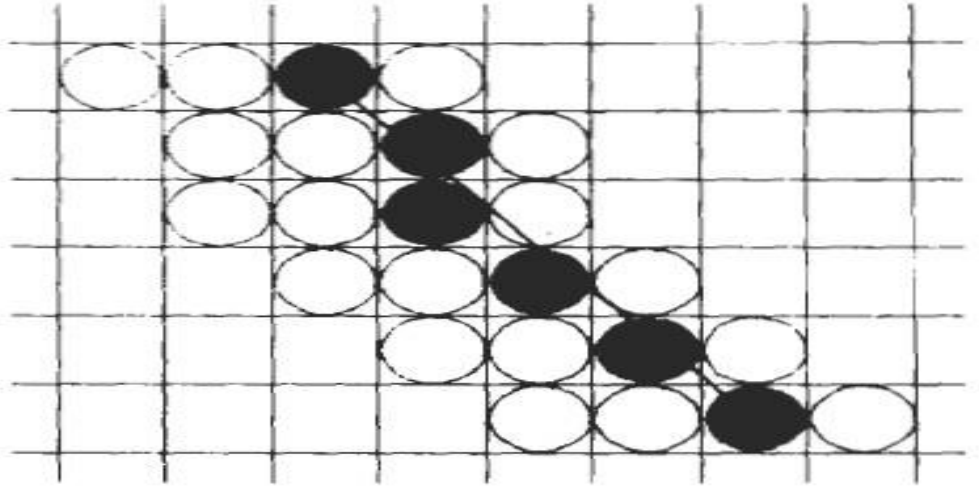


Figure 4-4
Raster line with slope $|m| > 1$ and line-width parameter lw = 4 plotted with horizontal pixel spans.

Unit-III: Graphics Primitives

Attributes

2. Line Width:

- The problem with implementing width options using horizontal and vertical pixel widths is that the width of line is depend on the slope. A 45-degree line will be displayed thinner as compared to vertical or horizontal line plotted with same number of pixel widths.
- Another problem is that it produces lines whose ends are either horizontal or vertical. We can adjust the shape of the line ends by adding ***Line Caps***.

Unit-III: Graphics Primitives

Attributes

2. Line Width:

- **Line Caps**

- One kind of line cap is the ***Butt Cap.*** It is obtained by adjusting the end positions of lines so that the thick line is displayed with square ends that are perpendicular to the line.
- Another line cap is the ***Round Cap*** obtained by adding a filled semicircle to each butt cap.

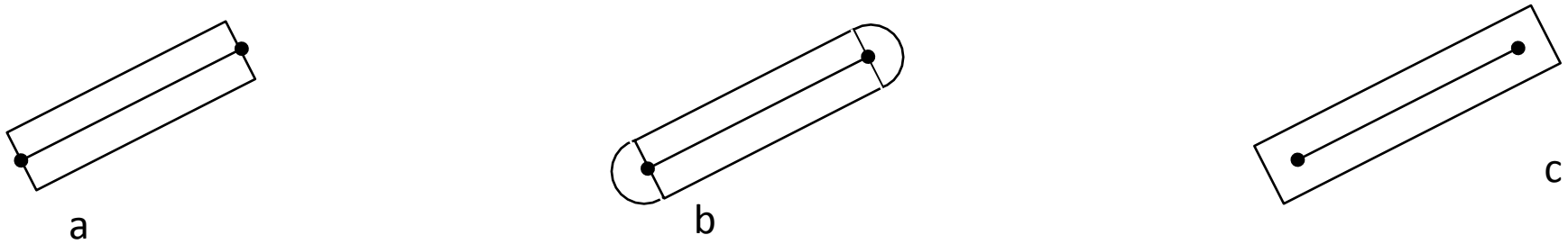
Unit-III: Graphics Primitives

Attributes

2. Line Width:

- **Line Caps**

❑ The third type of line cap is the ***Projecting Square Cap***. Here, the butt cap is extended to half of line width.



Thick Lines drawn with (a) Butt Cap (b) Round Cap and (c) Projecting Square Cap

Unit-III: Graphics Primitives

Attributes

2. Line Width:

- **Line Caps**

- The methods that we have considered for displaying thick lines will not produce smoothly connected line segments. It leaves gaps at the boundaries between lines of different slope.
- There are three possible methods for smoothly joining two line segments.

  1. Miter Join
  2. Round Join
  3. Bevel Join

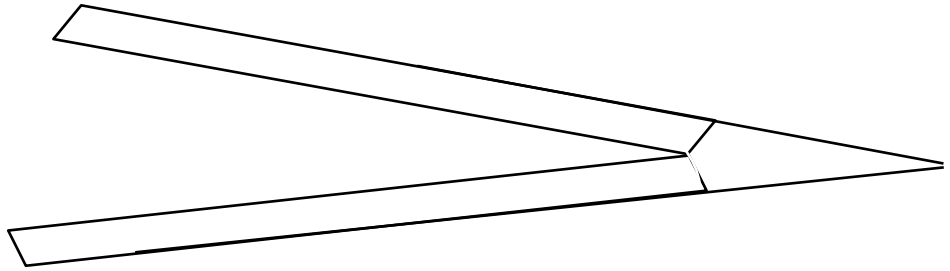
Unit-III: Graphics Primitives

Attributes

2. Line Width:

- **Line Caps**

1. Miter Join: A ***Miter Join*** is obtained by extending the outer boundaries of each of the two lines until they meet.



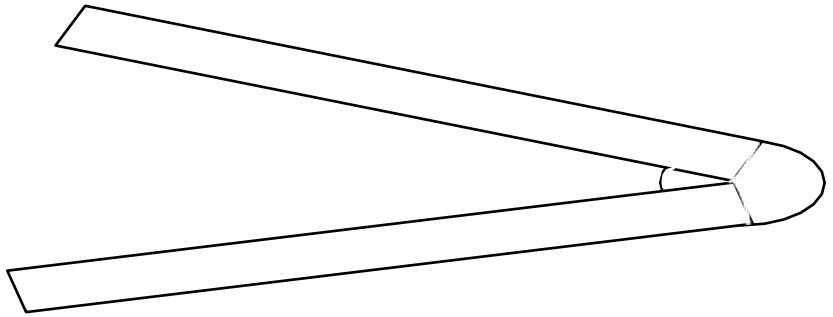
Unit-III: Graphics Primitives

Attributes

2. Line Width:

- **Line Caps**

2. A ***Round Join*** is produced by covering the connection between the two segments with a circular boundary whose diameter is equal to the line width.



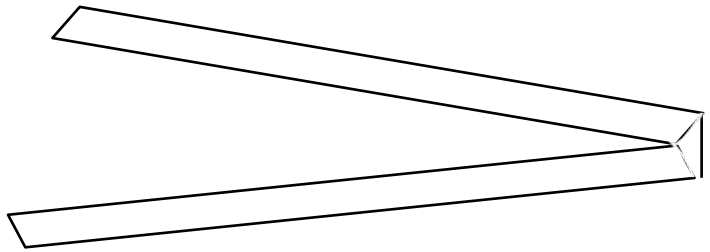
Unit-III: Graphics Primitives

Attributes

2. Line Width:

- **Line Caps**

3. A ***Bevel Join*** is generated by displaying the line with butt caps and filling in the triangular gap where the segments meet.



Unit-III: Graphics Primitives

Attributes

3. Pen and Brush:

- In some graphic packages, lines can be displayed with pen or brush selection. Options in this category include shape, size and pattern.
- These shapes are stored in a ***Pixel Mask*** that identifies the pixel positions that are to be set along the line path.
- Lines generated with pen or brush shaped can be displayed in various widths by changing the size of the mask. Lines can also be displayed with selected patterns.

Unit-III: Graphics Primitives

Attributes

3. Pen and Brush:

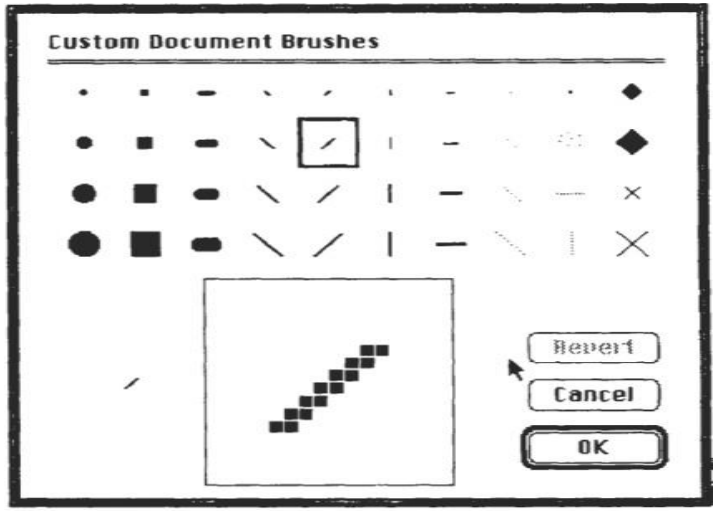


Figure 4-7
Pen and brush shapes for line display.

Figure 4-10
Curved lines drawn with a paint program using various shapes and patterns. From left to right, the brush shapes are square, round, diagonal line, dot pattern, and faded airbrush.

Unit-III: Graphics Primitives

Attributes

4. Line Color:

A system displays a line in the current color by setting the color value in the frame buffer at pixel locations. The number of color choices depends on the number of bits available per pixel in the frame buffer. A line drawn in the background color is invisible.

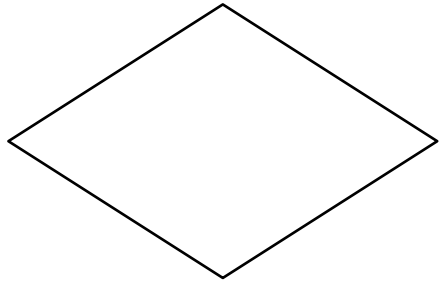
Unit-III: Graphics Primitives

Area Fill Attributes

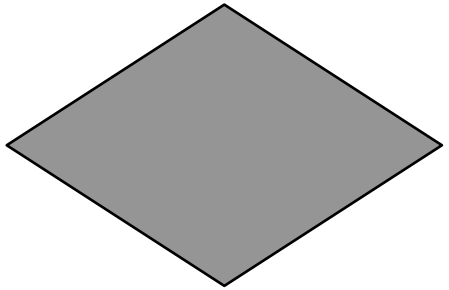
- Options for filling a region include a choice between a solid color and a patterned fill. These options can be applied to polygon regions or regions with curved boundaries.
- Areas can be displayed with various fill styles: hollow, solid, pattern and hatch. Hollow areas are displayed using only the boundary outline, with interior color the same as the background color.
- A Solid fill is displayed in a single color including the borders. Fill style can also be with a specific pattern or design. The Hatch fill is used to fill area with hatching pattern.

Unit-III: Graphics Primitives

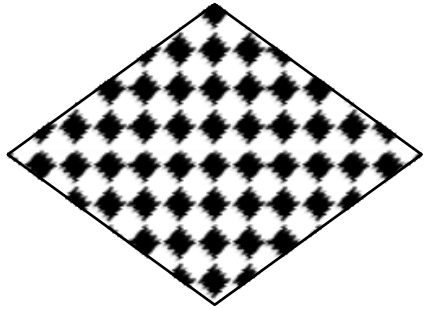
Area Fill Attributes



Hollow

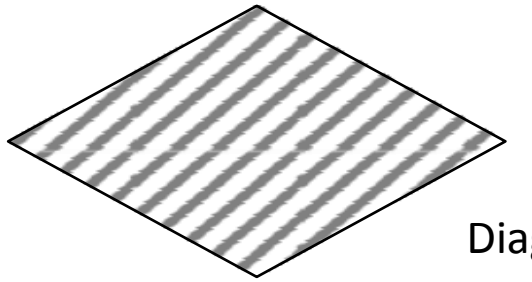


Solid

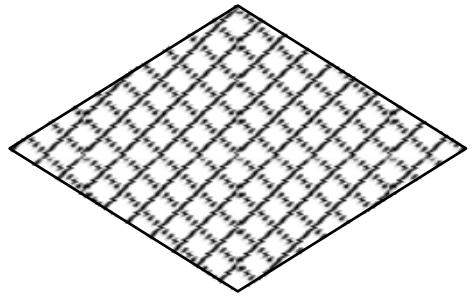


Pattern

Polygon Fill Style



Diagonal Hatch



Diagonal Cross Hatch

Unit-III: Graphics Primitives

Character Attributes

The appearance of displayed characters is controlled by attributes such as font, size, color and orientation. Attributes can be set both for entire character strings and for individual characters, known as Marker symbols.

1. **Text Attributes:** There are many text options available, such as font, color, size, spacing, and orientation.

•**Text Style:** The characters in a selected font can also be displayed in various underlining styles (solid, dotted, dashed, double), in **bold**, in *italics*, shadow style, etc. Font options can be made available as predefined sets of grid patterns or as character sets designed with lines and curves.

Unit-III: Graphics Primitives

Character Attributes

1. **Text Attributes:** There are many text options available, such as font, color, size, spacing, and orientation.

•**Text Color:** Color settings for displayed text are stored in the system attribute list and transferred to the frame buffer by character loading functions. When a character string is displayed, the current color is used to set pixel values in the frame buffer corresponding to the character shapes and position.

Unit-III: Graphics Primitives

Character Attributes

1. **Text Attributes:** There are many text options available, such as font, color, size, spacing, and orientation.

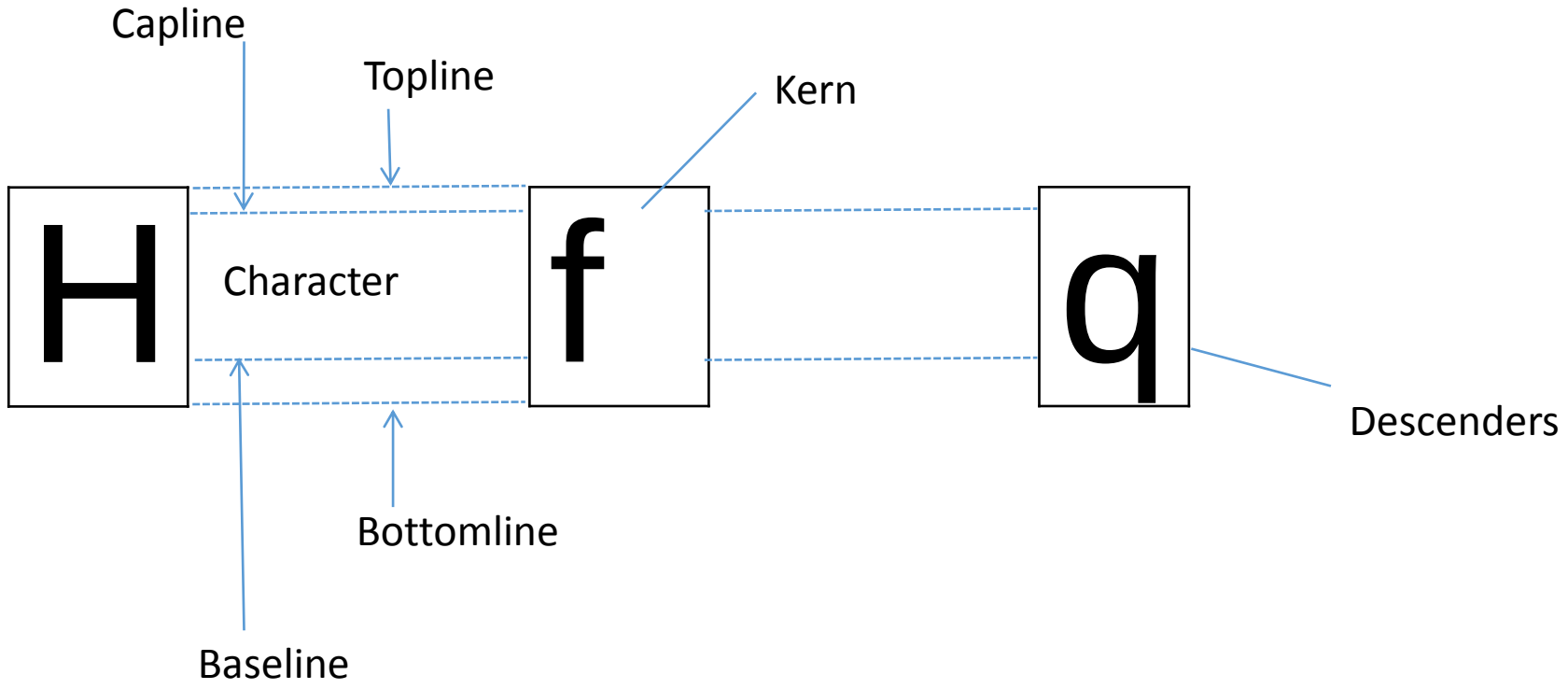
•**Text Size:** We can adjust text size by changing the overall dimensions, i.e., width and height, of characters or by changing only the width. Character size is specified in ***Points***, where 1 point is 0.013837 inch or approximately 1/72 inch. Point measurements specify the size of the ***Character Body***. Different fonts with the same point specifications can have different character sizes depending upon the design of the font.

Unit-III: Graphics Primitives

Character Attributes

1. **Text Attributes:** There are many text options available, such as font, color, size, spacing, and orientation.

- **Text Size:**



Unit-III: Graphics Primitives

Character Attributes

1. **Text Attributes:** There are many text options available, such as font, color, size, spacing, and orientation.

•**Text Size:**

- The distance between ***Topline*** and ***Bottomline*** is same for all characters in a particular size and font, but the width may be different.
- The ***Character Height*** is the distance between the ***Baseline*** and ***Capline*** of characters. Kerned characters, such as f and j, extend beyond the character-width limits. And letters with descenders, such as g, j, p, q, extend below the baseline.

Unit-III: Graphics Primitives

Character Attributes

1. **Text Attributes:** There are many text options available, such as font, color, size, spacing, and orientation.

•**Text Size:**

- **Effect of changing Height, Width and Spacing**

HEIGHT1	WIDTH1	SPACING1
HEIGHT2	**WIDTH2**	**SPACING2**
HEIGHT3	***WIDTH3***	***SPACING3***

Unit-III: Graphics Primitives

Character Attributes

1. **Text Attributes:** There are many text options available, such as font, color, size, spacing, and orientation.

•Text Orientation:

- The text can be displayed at various angles, known as orientation. A procedure for orienting text rotates characters so that the sides of character bodies, from baseline to topline at aligned at some angle. Character strings can be arranged vertically or horizontally.

A text orientated by 45 degrees in anticlockwise and clockwise direction

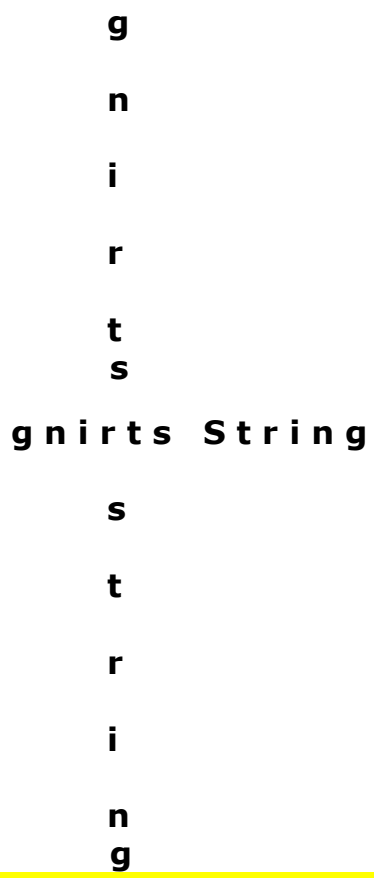
Unit-III: Graphics Primitives

Character Attributes

1. **Text Attributes:** There are many text options available, such as font, color, size, spacing, and orientation.

•**Text Path:**

❑ In some applications the character strings are arranged vertically or horizontally. This is known as Text Path. Text path can be right, left, up or down.



Unit-III: Graphics Primitives

Character Attributes

1. **Text Attributes:** There are many text options available, such as font, color, size, spacing, and orientation.

Text Alignment:

Another attribute for character strings is alignment. This attribute specifies how text is to be positioned with respect to the start coordinates. Vertical alignment can be top, cap, half, base and bottom. Similarly, horizontal alignment can be left, centre and right.

Unit-III: Graphics Primitives

Antialiasing

- **Antialiasing** is a technique used in computer graphics to remove the aliasing effect.
- The aliasing effect is the appearance of jagged edges or “jaggies” in a rasterized image (an image rendered using pixels).
- The problem of jagged edges technically occurs due to distortion of the image when scan conversion is done with sampling at a low frequency, which is also known as Undersampling.
- Aliasing occurs when real-world objects which comprise of smooth, continuous curves are rasterized using pixels.

Unit-III: Graphics Primitives

Antialiasing

Cause of anti-aliasing is **Undersampling**. Undersampling results in loss of information of the picture. Undersampling occurs when sampling is done at a frequency lower than Nyquist sampling frequency. To avoid this loss, we need to have our sampling frequency atleast twice that of highest frequency occurring in the object.

This minimum required frequency is referred to as **Nyquist sampling frequency (fs):**

$f_s = 2 * f_{max}$

Unit-III: Graphics Primitives

Antialiasing

Methods of Antialiasing(AA)

Aliasing is removed using four methods:

1. Using high-resolution display,
2. Post filtering (Supersampling),
3. Pre-filtering (Area Sampling),
4. Pixel phasing.

Unit-III: Graphics Primitives

Antialiasing

1. Using high resolution device:

- One way to reduce aliasing effect and increase sampling rate is to simply display objects at a higher resolution.
- Using high resolution, the jaggies become so small that they become indistinguishable by the human eye.
- Hence, jagged edges get blurred out and edges appear smooth.

Unit-III: Graphics Primitives

Antialiasing

2. Post Filtering (Supersampling)

- In this method, we are increasing the sampling resolution by treating the screen as if it's made of a much more fine grid, due to which the effective pixel size is reduced. But the screen resolution remains the same.
- Now, intensity from each subpixel is calculated and average intensity of the pixel is found from the average of intensities of subpixels.
- Thus we do sampling at higher resolution and display the image at lower resolution or resolution of the screen, hence this technique is called supersampling.
- This method is also known as post filtration as this procedure is done after generating the rasterized image.

Unit-III: Graphics Primitives

Antialiasing

3. Pre Filtering (Area Sampling)

- In area sampling, pixel intensities are calculated proportional to areas of overlap of each pixel with objects to be displayed. Here pixel color is computed based on the overlap of scene's objects with a pixel area.

Unit-III: Graphics Primitives

Antialiasing

4. **Pixel Phasing**

- It's a technique to remove aliasing. Here pixel positions are shifted to nearly approximate positions near object geometry. Some systems allow the size of individual pixels to be adjusted for distributing intensities which is helpful in pixel phasing.

References:

1. Donald Hearn, M. Pauline Baker. Computer Graphics ebook (pdf)

C-Version

Chapter 3: Page. No: 84 to 109, 117 to 131

THANK YOU