Computer Graphics and Multimedia Systems SCS1302

Unit 1

Syllabus

SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY FACULTY OF COMPUTING

SCS1302	COMPUTER GRAPHICS AND MULTIMEDIA SYSTEMS	L	T	P	Credits	Total Marks
		3	0	0	3	100

COURSE OBJECTIVES

- To gain knowledge to develop, design and implement two and three dimensional graphical structures.
- To enable students to acquire knowledge of Multimedia compression and animations.
- To learn creation, Management and Transmission of Multimedia objects.

UNIT 1 BASICS OF COMPUTER GRAPHICS **9 Hrs.**

Output Primitives: Survey of computer graphics - Overview of graphics systems - Line drawing algorithm - Circle drawing algorithm - Curve drawing algorithm - Attributes of output primitives - Anti-aliasing.

UNIT 2 2D TRANSFORMATIONS AND VIEWING **8 Hrs.**

Basic two dimensional transformations - Other transformations - 2D and 3D viewing - Line clipping - Polygon clipping - Logical classification - Input functions - Interactive picture construction techniques.

UNIT 3 3D CONCEPTS AND CURVES **10 Hrs.**

3D object representation methods - B-REP , sweep representations, Three dimensional transformations. Curve generation - cubic splines, Beziers, blending of curves- other interpolation techniques, Displaying Curves and Surfaces, Shape description requirement, parametric function. Three dimensional concepts – Introduction - Fractals and self similarity- Successive refinement of curves, Koch curve and peano curves.

Syllabus

UNIT 4 METHODS AND MODELS **8 Hrs.**

Visible surface detection methods - Illumination models - Halftone patterns - Dithering techniques - Polygon rendering methods - Ray tracing methods - Color models and color applications.

UNIT 5 MULTIMEDIA BASICS AND TOOLS **10 Hrs.**

Introduction to multimedia - Compression & Decompression - Data & File Format standards - Digital voice and audio - Video image and animation. Introduction to Photoshop - Workplace - Tools - Navigating window - Importing and exporting images - Operations on Images - resize, crop, and rotate - Introduction to Flash - Elements of flash document - Drawing tools - Flash animations - Importing and exporting - Adding sounds - Publishing flash movies - Basic action scripts - GoTo, Play, Stop, Tell Target

Max. 45 Hours

TEXT / REFERENCE BOOKS

1. Donald Hearn, Pauline Baker M., "Computer Graphics", 2nd Edition, Prentice Hall, 1994.
2. Tay Vaughan ,"Multimedia", 5th Edition, Tata McGraw Hill, 2001.
3. Ze-Nian Li, Mark S. Drew ,"Fundamentals of Multimedia", Prentice Hall of India, 2004.
4. D. McClelland, L.U.Fuller ,"Photoshop CS2 Bible", Wiley Publishing, 2005.
5. James D. Foley, Andries van Dam, Steven K Feiner, John F. Hughes, "Computer Graphics Principles and Practice, 2nd
6. Edition in C, Audison Wesley, ISBN - 981 -235-974-5
7. William M. Newman, Roberet F. Sproull, " Principles of Interactive Computer Graphics", Second Edition, Tata McGraw-Hill Edition.

Course Objective(CO)

CO1: Construct lines and circles for the given input.

CO2: Apply 2D transformation techniques to transform the shapes to fit them as per the picture definition.

CO3: Construct splines, curves and perform 3D transformations

CO4: Apply colour and transformation techniques for various applications.

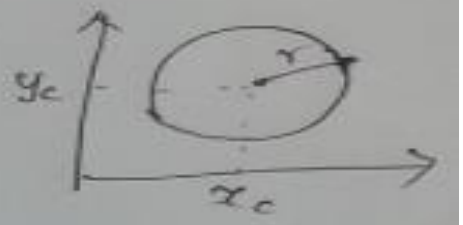
CO5: Analyse the fundamentals of animation, virtual reality, and underlying technologies.

CO6: Develop photo shop applications

CIRCLE DRAWING ALGORITHM

Circle Generating Algorithm

*) A circle is defined as the set of points that are all at a given distance, r, from a center position (x_c, y_c).



*) This distance relationship is expressed by the pythagorean cartesian coordinates as,

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

$$(y - y_c)^2 = r^2 - (x - x_c)^2$$

$$y - y_c = \pm \sqrt{r^2 - (x - x_c)^2}$$

$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2}$$

*) In parametric polar form:

$$x = x_c + r\cos\theta$$

$$y = y_c + r\sin\theta$$

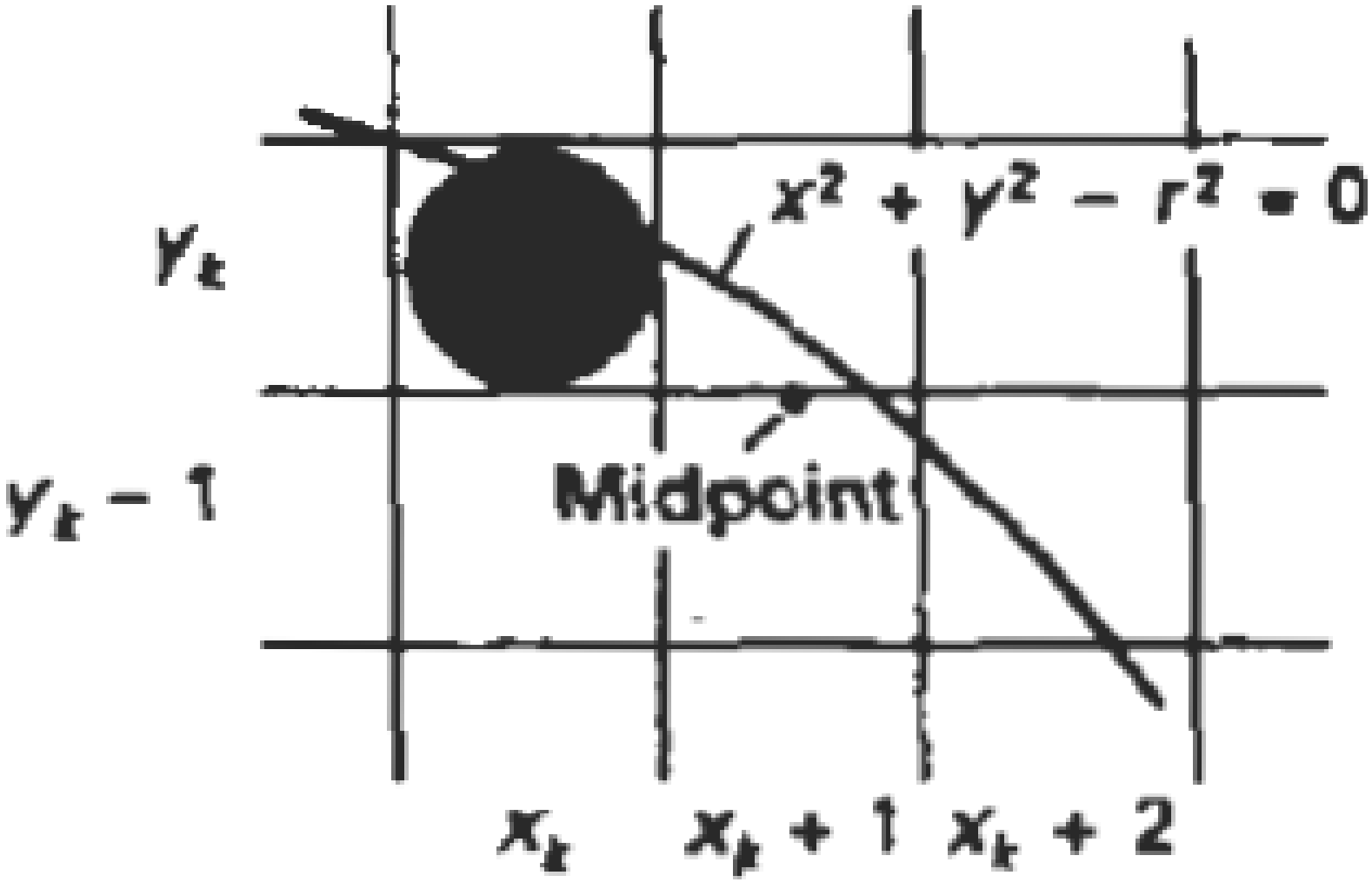
*) Midpoint Circle Algorithm:

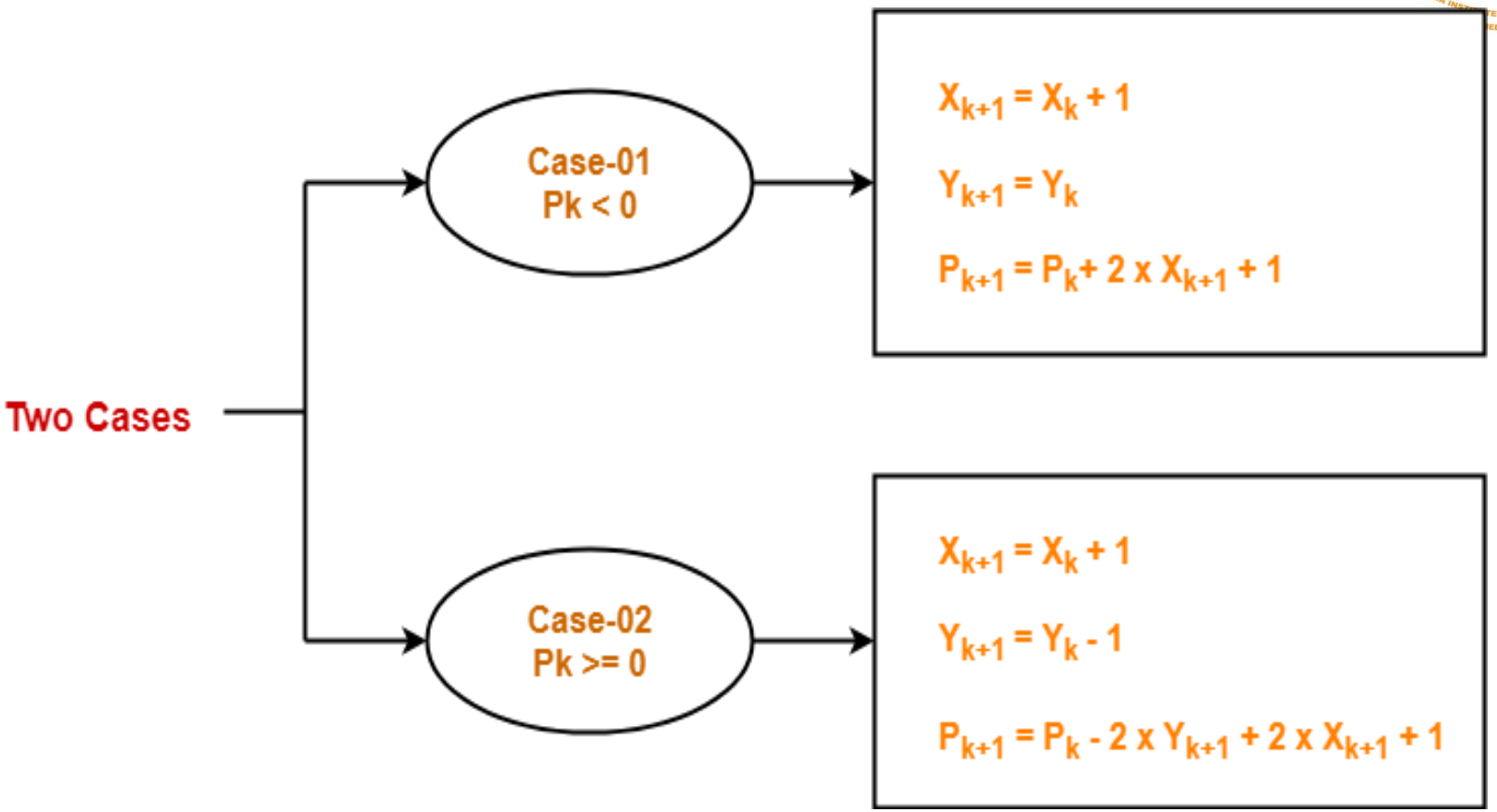
circle function :-

$$f_{circle}(x,y) = x^2 + y^2 - r^2$$

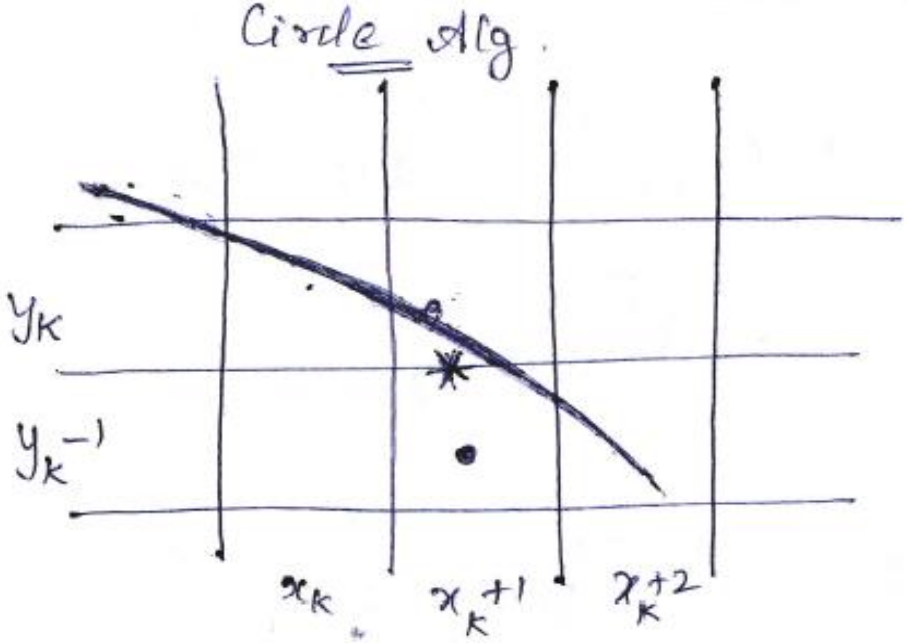
$$f_{circle}(x,y) \begin{cases} < 0, & \text{if } (x,y) \text{ is inside the circle boundary} \\ = 0, & \text{if } (x,y) \text{ is on the circle boundary} \\ > 0, & \text{if } (x,y) \text{ is outside the circle boundary} \end{cases}$$

Midpoint Circle drawing algorithm




Two Cases
Case-01
Pk < 0
$X_{k+1} = X_k + 1$
$Y_{k+1} = Y_k$
$P_{k+1} = P_k + 2 \times X_{k+1} + 1$
Case-02
Pk >= 0
$X_{k+1} = X_k + 1$
$Y_{k+1} = Y_k - 1$
$P_{k+1} = P_k - 2 \times Y_{k+1} + 2 \times X_{k+1} + 1$

Midpoint Circle drawing algorithm



2018 – 2022 (1)

III) ABCD

plotted at : (x_k, y_k)

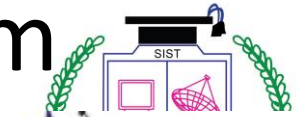
whether to plot the next pixel at,

(x_k+1, y_k) (or)

(x_k+1, y_k-1)

{is closer to circle}.

Midpoint Circle drawing algorithm



The decision parameter is the circle function, evaluated at the midpoint between these two pixels.

$$P_k = f_{circle}\left(x_k + 1, y_k - \frac{1}{2}\right) \longrightarrow (A)$$

$$= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$$

$$= x_k^2 + 2x_k + 1 + y_k^2 + \frac{1}{4} - y_k - r^2 \longrightarrow (1)$$

if $P_k < 0$, then this midpoint is inside the circle & y_k is closer to circle boundary.

$P_k > 0$, then the midpoint is outside the circle & $y_k - 1$ is closer.

Midpoint Circle drawing algorithm

(2)

$$P_{k+1} = f_{circle}\left(x_{k+1}+1, y_{k+1}-\frac{1}{2}\right)$$

$$= \left[(x_k+1)+1\right]^2 + \left(y_{k+1}-\frac{1}{2}\right)^2 - r^2$$

$$= (x_k+2)^2 + \left(y_{k+1}-\frac{1}{2}\right)^2 - r^2$$

$$= x_k^2 + 4 + 4x_k + y_{k+1}^2 + \frac{1}{4} - 2\frac{1}{2}y_{k+1} - r^2$$

$\longrightarrow$ (2)

(2) − (1),

Midpoint Circle drawing algorithm

(2) − (1),

$$P_{k+1} - P_k = x_k^2 + 4 + 4x_k + y_{k+1}^2 + \frac{1}{4} - y_{k+1} - r^2$$
$$- x_k^2 - 2x_k - 1 - y_k^2 - \frac{1}{4} + y_k + r^2$$
$$= 2x_k + 3 + y_{k+1}^2 - y_{k+1} - y_k^2 + y_k$$
$$= 2(x_k + 1) + 1 + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k)$$
$$= P_k + 2(x_k + 1) + 1 + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k)$$

→ (3)

Midpoint Circle drawing algorithm

(i) if P_k is -ve, then sub $y_{k+1} = y_k$ (in ③)

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

$$= 2(x_k+1) + 1$$

$$= 2x_{k+1} + 1$$

Midpoint Circle drawing algorithm

(ii) if P_k is +ve, then sub $y_{k+1} = y_k - 1$ (3)

$$= 2(x_k+1)+1 + \left((y_k-1)^2 - y_k^2\right) - (y_k - 1 - y_k)$$ in (3)

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

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

$$= 2x_k + 2 + 1 - 2y_k + 2$$

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

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

Midpoint Circle drawing algorithm

Starting Position :- $(0, r) \Leftrightarrow (0, 2r)$

Initial Decision Parameter :

$(x_0, y_0) = (0, r)$ Sub $(0, r)$ in (A),

$P_0 = f_{circle}\left(1, r - \frac{1}{2}\right)$

Apply $f_{circle}(x, y) = x^2 + y^2 - r^2$

$= 1 + \left(r - \frac{1}{2}\right)^2 - r^2$

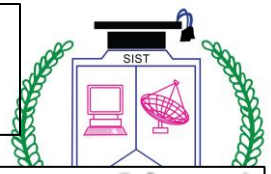
$= 1 + r^2 + \frac{1}{4} - 2 \cdot \frac{1}{2} r - r^2$

$= 1 + \frac{1}{4} - r$

$P_0 = \frac{5}{4} - r$

$\simeq \boxed{P_0 = 1 - r}$

Midpoint Circle drawing algorithm




```
void circleMidpoint (int xCenter, int yCenter, int radius)
{
   int x = 0;
   int y = radius;
   int p = 1 - radius;
   void circlePlotPoints (int, int, int, int);

   /* Plot first set of points */
   circlePlotPoints (xCenter, yCenter, x, y);

   while (x < y) {
      x++;
      if (p < 0)
         p += 2 * x + 1;
      else {
         y--;
         p += 2 * (x - y) + 1;
      }
      circlePlotPoints (xCenter, yCenter, x, y);
   }
}
```

Midpoint Circle drawing algorithm


```
void circlePlotPoints (int xCenter, int yCenter, int x, int y)
{
  setPixel (xCenter + x, yCenter + y);
  setPixel (xCenter - x, yCenter + y);
  setPixel (xCenter + x, yCenter - y);
  setPixel (xCenter - x, yCenter - y);
  setPixel (xCenter + y, yCenter + x);
  setPixel (xCenter - y, yCenter + x);
  setPixel (xCenter + y, yCenter - x);
  setPixel (xCenter - y, yCenter - x);
}
```

Midpoint Circle drawing algorithm


```
void circleMidpoint (int xCenter, int yCenter, int radius)
{
   int x = 0;
   int y = radius;
   int p = 1 - radius;
   void circlePlotPoints (int, int, int, int);

   /* Plot first set of points */
   circlePlotPoints (xCenter, yCenter, x, y);

   while (x < y) {
      x++;
      if (p < 0)
         p += 2 * x + 1;
      else {
         y--;
         p += 2 * (x - y) + 1;
      }
      circlePlotPoints (xCenter, yCenter, x, y);
   }
}

void circlePlotPoints (int xCenter, int yCenter, int x, int y)
{
   setPixel (xCenter + x, yCenter + y);
   setPixel (xCenter - x, yCenter + y);
   setPixel (xCenter + x, yCenter - y);
   setPixel (xCenter - x, yCenter - y);
   setPixel (xCenter + y, yCenter + x);
   setPixel (xCenter - y, yCenter + x);
   setPixel (xCenter + y, yCenter - x);
   setPixel (xCenter - y, yCenter - x);
}
```

Midpoint Circle drawing algorithm

- Example:

  Given (xc,yc)=(0,0), x=0 and radius, r=10.

  Generate a circle using Midpoint Circle drawing algorithm.

- Given xc=yc=0, radius, r=10.

Solution:

x=0; y=10;

p=1-r

=1-10

=-9

circlepoints(0,0,0,10)

{

Setpixel(0+0,0+10) =(0,10)

Setpixel(0-0,0+10) =(0,10)

Setpixel(0+0,0-10) =(0,-10)

Setpixel(0-0,0-10) =(0,-10)

Setpixel(0+10,0+0) =(10,0)

Setpixel(0-10,0+0) = (-10,0)

Setpixel(0+10,0-0) =(10,0)

Setpixel(0-10,0-0) = (-10,0)

}

While(0<10)

{x=1

If(-9 <0)

P=p+2*x+1 = -9+2*1+1 = -9+2+1

=-6

}

circlepoints(0,0,1,10)

{

Setpixel(0+1,0+10) =(1,10)

Setpixel(0-1,0+10) =(-1,10)

Setpixel(0+1,0-10) =(1,-10)

Setpixel(0-1,0-10) =(-1,-10)

Setpixel(0+10,0+1) =(10,1)

Setpixel(0-10,0+1) = (-10,1)

Setpixel(0+10,0-1) =(10,-1)

Setpixel(0-10,0-1) = (-10,-1)

}

Example: Midpoint Circle drawing algorithm

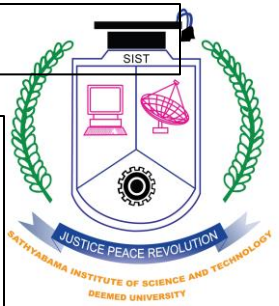
Iteration 2:

While(1<10)
{x=2
If(-6 <0)
P=p+2*x+1 = -6+2*2+1 = -6+4+1
=-1
}
circlepoints(0,0,2,10)
{
Setpixel(0+2,0+10) =(2,10)
Setpixel(0-2,0+10) =(-2,10)
Setpixel(0+2,0-10) =(2,-10)
Setpixel(0-2,0-10) =(-2,-10)
Setpixel(0+10,0+2) =(10,2)
Setpixel(0-10,0+2) = (-10,2)
Setpixel(0+10,0-2) =(10,-2)
Setpixel(0-10,0-2) = (-10,-2)
}

Iteration 3:

While(2<10)
{x=3
If(-1 <0)
P=p+2*x+1 = -1+2*3+1 = -1+6+1
=6
}
circlepoints(0,0,3,10)
{
Setpixel(0+3,0+10) =(3,10)
Setpixel(0-3,0+10) =(-3,10)
Setpixel(0+3,0-10) =(3,-10)
Setpixel(0-3,0-10) =(-3,-10)
Setpixel(0+10,0+3) =(10,3)
Setpixel(0-10,0+3) = (-10,3)
Setpixel(0+10,0-3) =(10,-3)
Setpixel(0-10,0-3) = (-10,-3)
}

Example: Midpoint Circle drawing algorithm



Iteration4:

While(3<10)

{x=4

If(6 <0)

Else

y=y-1 = 9

P=p+2*(x-y)+1 = 6+2*(4-9)+1 = 6 +2(-5) +1 = 6 -10 +1

=-3

}

circlepoints(0,0,4,9)

{

Setpixel(0+4,0+9) =(4,9)

Setpixel(0-4,0+9) =(-4,9)

Setpixel(0+4,0-9) =(4,-9)

Setpixel(0-4,0-9) =(-4,-9)

Setpixel(0+9,0+4) =(9,4)

Setpixel(0-9,0+4) = (-9,4)

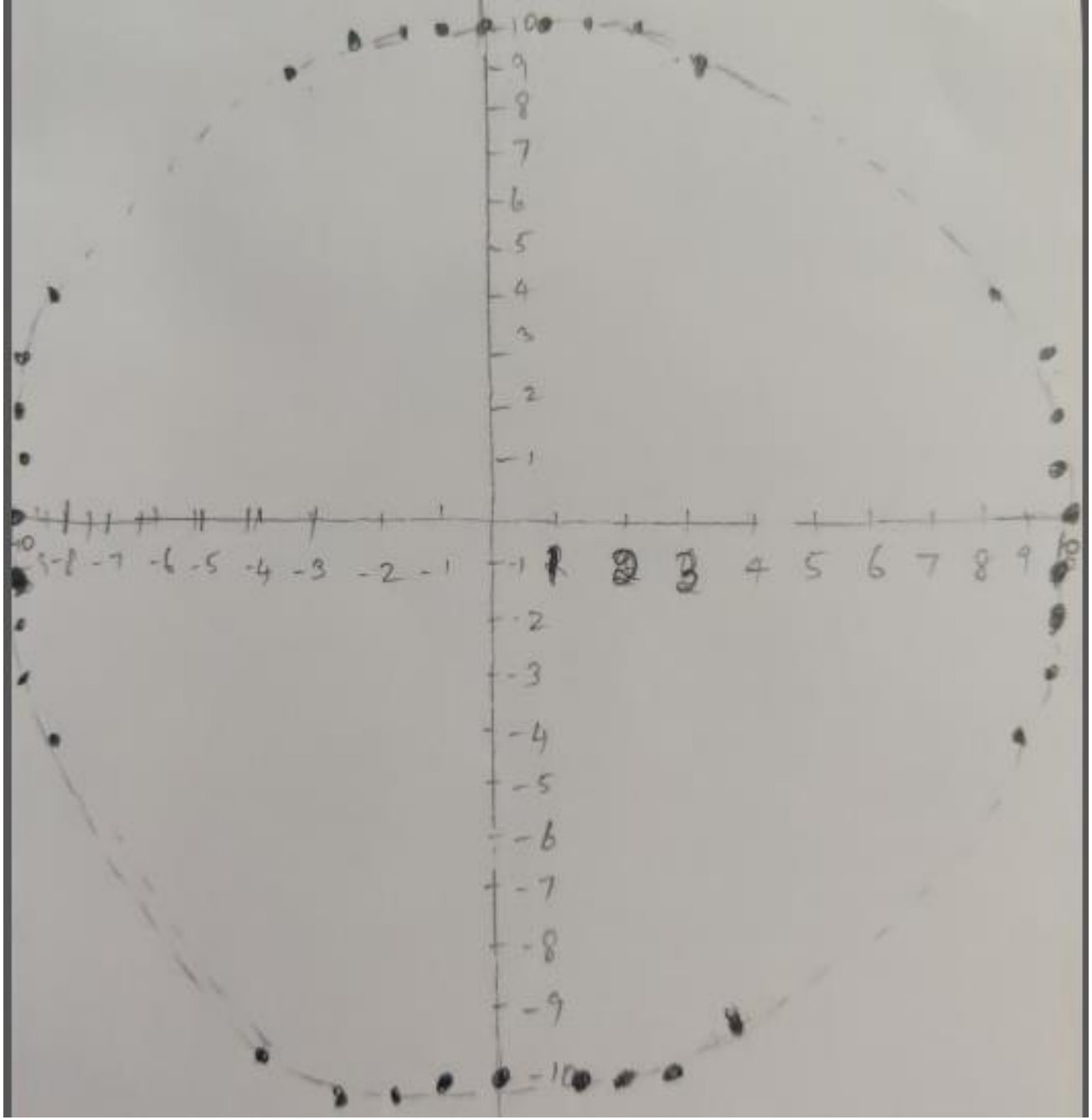
Setpixel(0+9,0-4) =(9,-4)

Setpixel(0-9,0-4) = (-9,-4)

}

RESULTANT POINTS

K	P_k	(X_{K+1}, Y_{k+1})
0	-9	(1,10)
1	-6	(2,10)
2	-1	(3,10)
3	6	(4,9)
4	-3	(5,9)
5	8	(6,8)
6	5	(7,7)



Disadvantages of Mid Point Circle Drawing Algorithm-

The disadvantages of Mid Point Circle Drawing Algorithm are-

- Accuracy of the generating points is an issue in this algorithm.
- The circle generated by this algorithm is not smooth.
- This algorithm is time consuming.

Advantages of Mid Point Circle Drawing Algorithm-

- The advantages of Mid Point Circle Drawing Algorithm are-
- It is a powerful and efficient algorithm.
- The entire algorithm is based on the simple equation of circle $X^2 + Y^2 = R^2$.
- It is easy to implement from the programmer's perspective.
- This algorithm is used to generate curves on raster displays.

Attributes of Output Primitives

Any parameter that affects the way a primitive is to be displayed is referred to as an attribute parameter. Example attribute parameters are color, size etc. A line drawing function for example could contain parameter to set color, width and other properties.

1. Line Attributes
2. Curve Attributes
3. Color and Grayscale Levels
4. Area Fill Attributes
5. Character Attributes
6. Bundled Attributes

Line Attributes

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

* Line Type
* Line Width
* Pen and Brush Options
* Line Color

Line type

Possible selection of line type attribute includes solid lines, dashed lines and dotted lines. To set line type attributes in a **PHIGS** application program, a user invokes the function

setLinetype (lt)

Where parameter lt is assigned a positive integer value of 1, 2, 3 or 4 to generate lines that are solid, dashed, dash dotted respectively. Other values for line type parameter it could be used to display variations in dot-dash patterns.

Line width

Implementation of line width option depends on the capabilities of the output device to set the line width attributes.

setLinewidthScaleFactor (lw)

Line Cap

We can adjust the shape of the **line** ends to give them a better appearance by adding line caps.

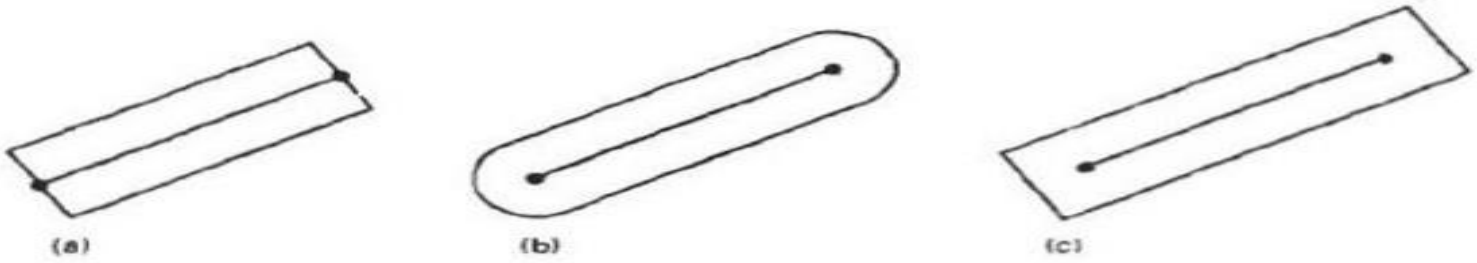
There are three types of line cap. They are

* Butt cap
* Round cap
* Projecting square cap

Butt cap obtained by adjusting the end positions of the component parallel lines so that the thick line is displayed with square ends that are perpendicular to the line path.

Round cap obtained by adding a filled semicircle to each butt cap. The circular arcs are centered on the line endpoints and have a diameter equal to the line thickness.

Projecting square cap extend the line and add butt caps that are positioned one-half of the line width beyond the specified endpoints.



Thick lines drawn with (a) butt caps, (b) round caps, and (c) projecting square caps.

Three possible methods for smoothly joining two line segments

* Mitter Join
* Round Join
* Bevel Join

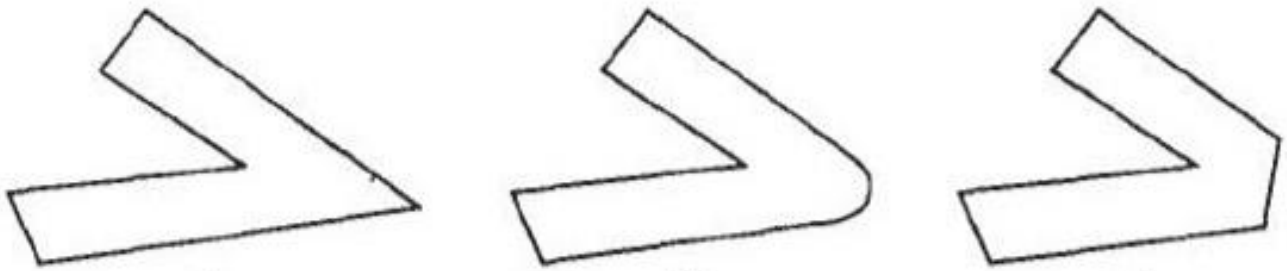
A miter join accomplished by extending the outer boundaries of each of the two lines

until they meet.

A round join is produced by capping the connection between the two segments with a circular boundary whose diameter is equal to the width.

A bevel join is generated by displaying the line segment with but caps and filling in tri

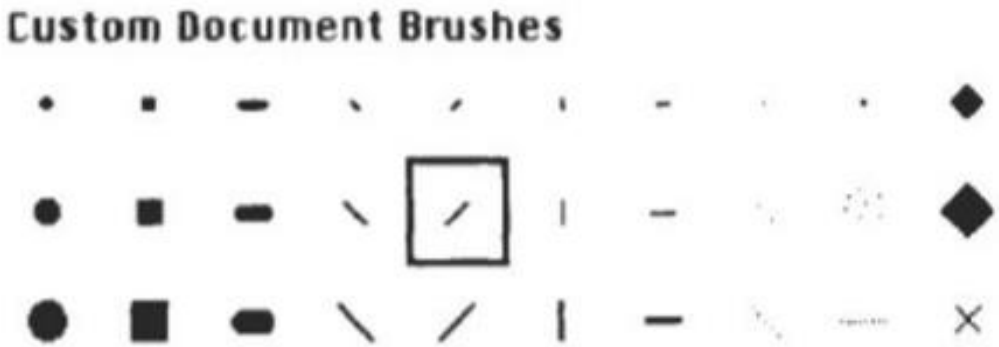
angular gap where the segments meet.



Thick line segments connected with (a) miter join, (b) round join, and (c) bevel join.

Pen and Brush Options

With some packages, lines can be displayed with pen or brush selections. Options in this category include shape, size, and pattern. Some possible pen or brush shapes are given in Figure



Line color

A poly line routine displays a line in the current color by setting this color value in the frame buffer at pixel locations along the line path using the set pixel procedure.

We set the line color value in PHIGS with the function

setPolylineColourIndex (lc)

Curve attributes

Parameters for curve attribute are same as those for line segments. Curves displayed with varying colors, widths, dot – dash patterns and available pen or brush options

Color and Grayscale Levels

Various color and intensity-level options can be made available to a user, depending on the capabilities and design objectives of a particular system

A user can set color-table entries in a PHIGS applications program with the function

setColourRepresentation (ws, ci, colorptr)

Parameter **ws** identifies the workstation output device; parameter ci specifies the color index, which is the color-table position number (**0** to **255**) and parameter colorptr points to a trio of RGB color values (**r, g, b**) each specified in the range from **0** to 1

Grayscale

With monitors that have no color capability, color functions can be used in an application program to set the shades of gray, or grayscale, for displayed primitives. Numeric values over the range from 0 to 1 can be used to specify grayscale levels, which are then converted to appropriate binary codes for storage in the raster.

INTENSITY CODES FOR A FOUR-LEVEL GRAYSCALE SYSTEM

Intensity Codes	*Stored Intensity Values In The Frame Buffer (Binary Code)*		*Displayed Grayscale*
0.0	0	(00)	Black
0.33	1	(01)	Dark gray
0.67	2	(10)	Light gray
1.0	3	(11)	White

Area fill Attributes

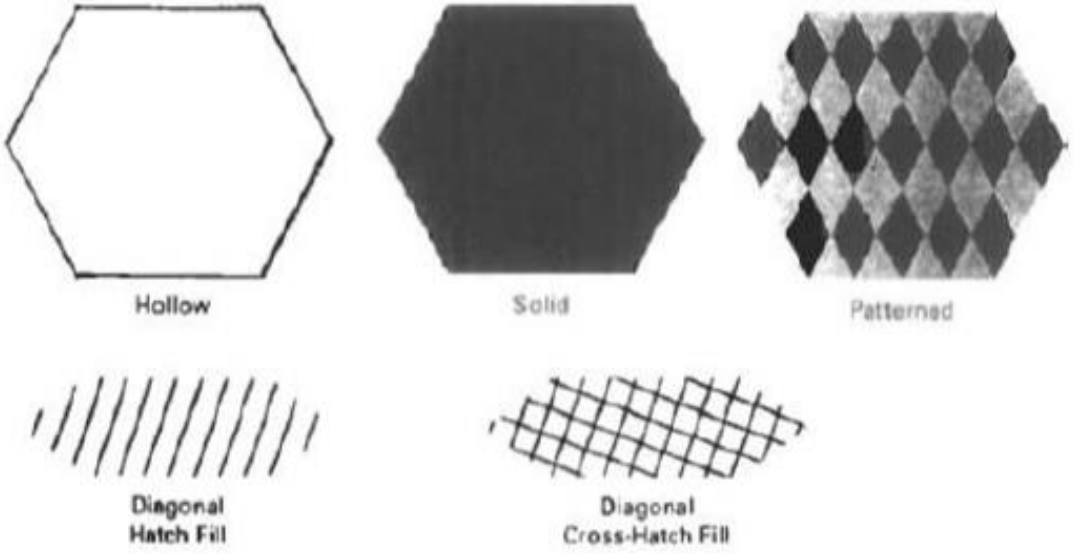
Options for filling a defined region include a choice between a solid color or a pattern fill and choices for particular colors and patterns

Fill Styles

Areas are displayed with three basic fill styles: hollow with a color border, filled with a solid color, or filled with a specified pattern or design. A basic fill style is selected in a PHIGS program with the function

setInteriorStyle (fs)

Values for the fill-style parameter fs include hollow, solid, and pattern. Another value for fill style is hatch, which is used to fill an area with selected hatching patterns-parallel lines or crossed lines



SetInteriorStyle(pattern)

SetInteriorStyleIndex(2);

Fill area (n, points)

Index (pi)	*Pattern (cp)*
1	$\begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$
2	$\begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 2 \end{bmatrix}$

Character Attributes

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

Text Attributes

The choice of font or type face is set of characters with a particular design style as courier, Helvetica, times roman, and various symbol groups.

The characters in a selected font also be displayed with styles. (solid, dotted, double) in **bold face** in **italics**, and in **outline** or shadow styles.

A particular font and associated stvle is selected in a PHIGS program by setting an

integer code for the text font parameter tf in the function

setTextFont (tf)

Control of text color (or intensity) is managed from an application program with

setTextColourIndex (tc)

Where text color parameter tc specifies an allowable color code.

Text size can be adjusted without changing the width to height ratio of characters with

setCharacterHeight (ch)

Height 1

Height 2

Height 3

Parameter ch is assigned a real value greater than 0 to set the coordinate height of capital letters

The width only of text can be set with function.

setCharacterExpansionFactor (cw)

Where the character width parameter cw is set to a positive real value that scales the body width of character

width 05

width 1.0

width 2.0

Spacing between characters is controlled separately with

setCharacterSpacing (cs)

Where the character-spacing parameter cs can he assigned any real value

Spacing 0.0

Spacing 0.5

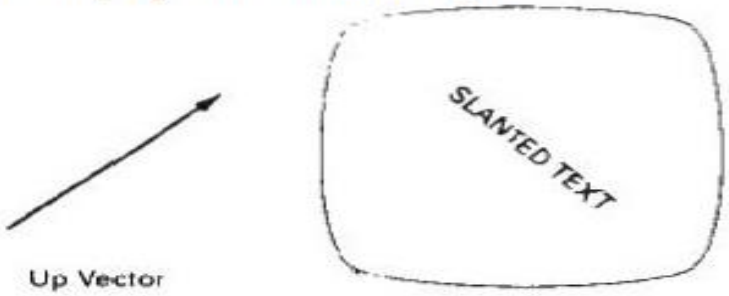
Spacing 1.0

The orientation for a displayed character string is set according to the direction of the character up vector

setCharacterUpVector (upvect)

Parameter upvect in this function is assigned two values that specify the x and y

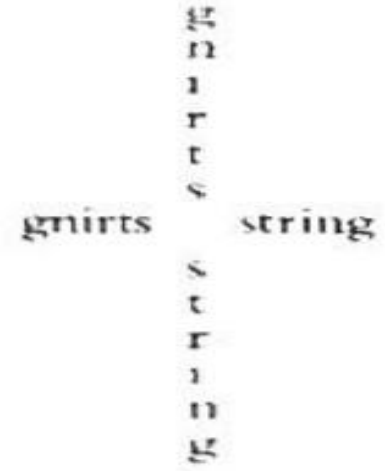
vector components. For example, with upvect = (1, 1), the direction of the up vector is 45o and text would be displayed as shown in Figure.



To arrange character strings vertically or horizontally

setTextPath (tp)

can be assigned the value: right, left, up, or down



Another handy attribute for character strings is alignment. This attribute specifies how text is to be positioned with respect to the $tart coordinates. Alignment attributes are set with

setTextAlignment (h,v)

where parameters h and v control horizontal and vertical alignment. Horizontal alignment is set by assigning h a value of left, center, or right. Vertical alignment is set by assigning v a value of top, cap, half, base or bottom.

A precision specification for text display is given with

setTextPrecision (tpr)

tpr is assigned one of values string, char or stroke.

Marker Attributes

A marker symbol is a single character that can he displayed in different colors and in different sizes. Marker attributes are implemented by procedures that load the chosen character into the raster at the defined positions with the specified color and size. We select a particular character to be the marker symbol with

setMarkerType (mt)

where marker type parameter mt is set to an integer code. Typical codes for marker type are the integers 1 through 5, specifying, respectively, a dot (.) a vertical cross (+), an asterisk (*), a circle (o), and a diagonal cross (X).

We set the marker size with

setMarkerSizeScaleFactor (ms)

with parameter marker size ms assigned a positive number. This scaling parameter is applied to the nominal size for the particular marker symbol chosen. Values greater than 1 produce character enlargement; values less than 1 reduce the marker size.

Marker color is specified with

setPolymarkerColourIndex (mc)

A selected color code parameter mc is stored in the current attribute list and used to display subsequently specified marker primitives

Bundled Attributes

The procedures considered so far each function reference a single attribute that specifies exactly how a primitive is to be displayed these specifications are called individual attributes.

A particular set of attributes values for a primitive on each output device is chosen by specifying appropriate table index. Attributes specified in this manner are called bundled attributes. The choice between a bundled or an unbundled specification is made by setting a switch called the aspect source flag for each of

these attributes

setIndividualASF(attributeptr, flagptr)

where parameter attributer ptr points to a list of attributes and parameter flagptr points to the corresponding list of aspect source flags. Each aspect source flag can be assigned a value of individual or bundled.

Bundled line Attributes

Entries in the bundle table for line attributes on a specified workstation are set with the function

setPolylineRepresentation (ws, li, lt, lw, lc)

Parameter ws is the workstation identifier and line index parameter li defines the bundle table position. Parameter lt, lw, tc are then bundled and assigned values to set the line type, line width, and line color specifications for designated table index.

Example

setPolylineRepresentation (1, 3, 2, 0.5, 1)

setPolylineRepresentation (4, 3, 1, 1, 7)

A poly line that is assigned a table index value of 3 would be displayed using dashed lines at half thickness in a blue color on work station 1; while on workstation 4, this same index generates solid, standard-sized white lines

Bundle area fill Attributes

Table entries for bundled area-fill attributes are set with

setInteriorRepresentation (ws, fi, fs, pi, fc)

Which defines the attributes list corresponding to fill index fi on workstation ws. Parameter fs, pi and fc are assigned values for the fill style pattern index and fill color.

Bundled Text Attributes

setTextRepresentation (ws, ti, tf, tp, te, ts, tc)

Bundles values for text font, precision expansion factor size an color in a table position for work station ws that is specified by value assigned to text index parameter ti.

Bundled marker Attributes

setPolymarkerRepresentation (ws, mi, mt, ms, mc)

That defines marker type marker scale factor marker color for index mi on workstation ws.

Inquiry functions

Current settings for attributes and other parameters as workstations types and status in the system lists can be retrieved with inquiry functions.

inquirePolylineIndex (lastli)

and

inquireInteriorcColourIndex (lastfc)

Copy the current values for line index and fill color into parameter lastli and lastfc.

SUMMARY OF ATTRIBUTES

Output Primitive Type	*Associated Attributes*	*Attribute-Setting Functions*	*Bundled-Attribute Functions*
Line	Type	setLinetype	setPolylineIndex
	Width	setLineWidthScaleFactor	setPolylineRepresentation
	Color	setPolylineColourIndex	
Fill Area	Fill Style	setInteriorStyle	setInteriorIndex
	Fill Color	setInteriorColorIndex	setInteriorRepresentation
	Pattern	setInteriorStyleIndex	
		setPatternRepresentation	
		setPatternSize	
		setPatternReferencePoint	
Text	Font	setTextFont	setTextIndex
	Color	setTextColourIndex	setTextRepresentation
	Size	setCharacterHeight	
		setCharacterExpansionFactor	
	Orientation	setCharacterUpVector	
		setTextPath	
		setTextAlignment	
Marker	Type	setMarkerType	setPolymarkerIndex
	Size	setMarkerSizeScaleFactor	setPolymarkerRepresentation
	Color	setPolymarkerColourIndex	

Antialiasing

- Antialiasing is the smoothing of the image or sound roughness caused by aliasing .
- With images, approaches include adjusting pixel positions or setting pixel intensities so that there is a more gradual transition between the color of a line and the background color.
- With sound, aliases are removed by eliminating frequencies above half the sampling frequencies.

Side effects of Scan Conversion

- The most common side effects when working with raster devices are:
  - Unequal Intensity
  - Overstrike
  - Aliasing
- **Unequal Intensity**
- Human perception of light is dependent on density and intensity of light source.
- **Solution:** By increasing the number of pixels on diagonal lines.
- **Overstrike:**
- The same pixel is written more than once.
- This results in intensified pixels in case of photographic media, such as slide or transparency.
- **Solution:** Check each pixel to see whether it has already been written to prior to writing a new point.

Aliasing

- The effect created when rasterization is performed over a discrete series of pixels.
- In particular, when lines or edges do not necessarily align directly with arrow or column of pixels, that line may appear unsmooth and have a stair-step edge appearance.
- Jagged appearance of curves or diagonal lines on a display screen, which is caused by low screen resolution.
- Refers to the plotting of a point in a location other than its true location in order to fit the point into the raster.

- Consider equation Y=mX+b, For m=0.5, b=1, X=3

  Y would be 2.5. So the point (3, 2.5) is plotted at alias location (3,3).

Anti-Aliasing

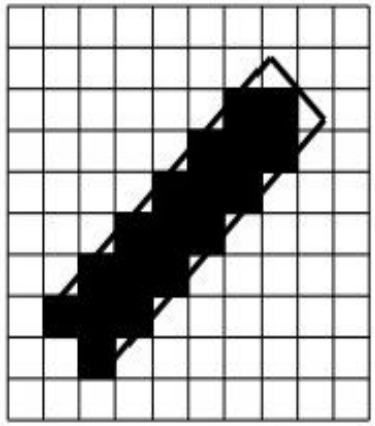
- The image on the right shows the result of anti-aliasing through the use of high r resolution.

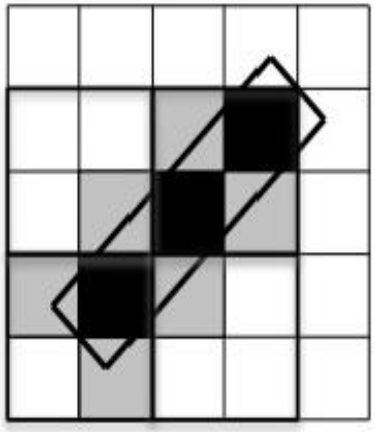
Anti-Aliasing

- Antialiasing utilizes blending techniques to blur the edges of the lines and provide the viewer with the illusion of a smoother line.
- Two general approaches:
- **Super-sampling**
  - samples at higher resolution, then filters down the resulting image
  - Sometimes called post-filtering
  - The prevalent form of anti-aliasing in hardware
- **Area sampling**
  - sample primitives with a box (or Gaussian, or whatever) rather than spikes
  - Requires primitives that have area (lines with width)
  - Sometimes referred to as pre-filtering

Super-sampling

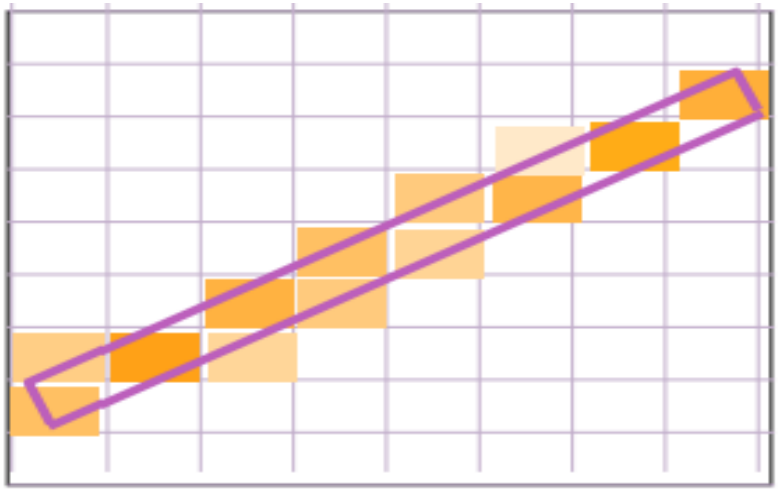
- Sample at a higher resolution than required for display, and filter image down
- 4 to 16 samples per pixel is typical
- Samples might be on a uniform grid, or randomly positioned, or other variants
- Divide each pixel into sub-pixels.
- The number of intensities are the max number of sub-pixels selected on the line segment within a pixel.
- The intensity level for each pixel is proportional to the number of sub-pixels inside the polygon representing the line area.
- Line intensity is distributed over more pixels.





Area Sampling

- Determine the percentage of area coverage for a screen pixel, then set the pixel intensity proportional to this percentage.
- Consider a line as having thickness.
- Consider pixels as little squares.
- Unweighted area sampli by the line.
- eg: Fill pixels according to the proportion of their square covered
- Weighted area sampling: Weight the contribution according to where in the square the primitives falls



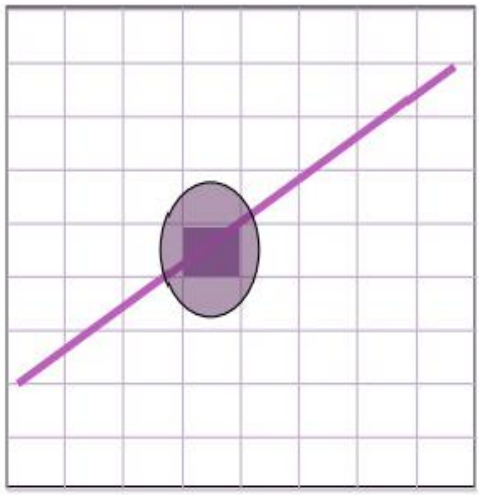
Unweighted Area Sampling

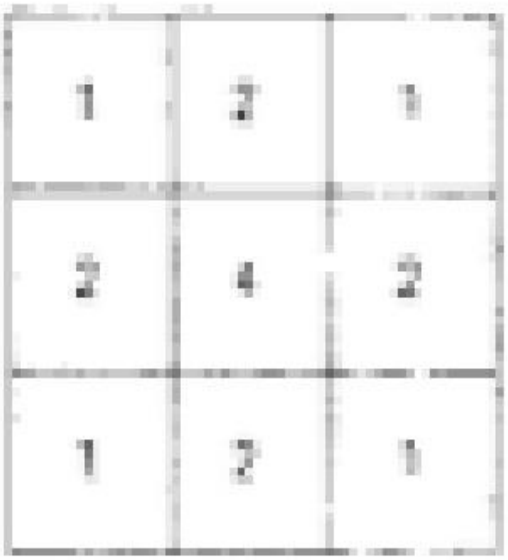
- Primitive cannot affect intensity of pixel if it does not intersect the pixel.
- Equal areas cause equal intensity, regardless of distance from pixel center to area.
- Unweighted sampling colors two pixels identically when the primitive cuts the same area through the two pixels.
- Intuitively, pixel cut through the center should be more heavily weighted than one cut along corner.

0	0	0	1/8	0
0	0	1/4	.914	1/8
0	1/4	.914	1/4	0
1/8	.914	1/4	0	0
0	1/8	0	0	0

Weighted Area Sampling

- weight the subpixel contributions according to position, giving higher weights to the central subpixels.
- weighting function, W(x,y)
  - specifies the contribution of primitive passing through the point (x, y) from pixel center



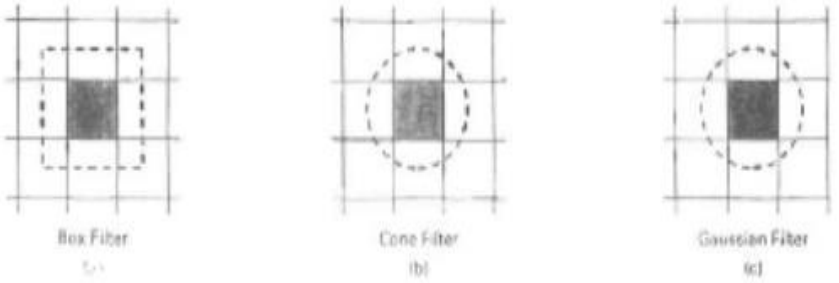


Filtering Techniques

- A continuous weighting surface (or filter function) covering the pixel.
- Applying the filter function by integrating over the pixel surface to obtain the weighted average intensity
- Weighted (Filter) Function

  - Determines the influence on the intensity of a pixel of a given small area dA of a primitive.
  - This function is constant for unweighted and decreases with increasing distance for weighted.
  - Total intensity is the integral of the weighting (filter) function over the area of overlap.
  - Ws is the volume (always between 0 and 1)
  - I=Imax •Ws

- Box, Cone and Gaussiean



- https://developerinsider.co/download-turbo-c-for-windows-7-8-8-1-and-windows-10-32-64-bit-full-screen/
- **developerinsider.in/turbocpp**