Computer Graphics & Multimedia (5CS4-04)

Computer Science and Engineering Department

Vision of the Department

To become renowned Centre of excellence in computer science and engineering and make competent engineers & professionals with high ethical values prepared for lifelong learning.

Mission of the Department

- To impart outcome based education for emerging technologies in the field of computer science and engineering.
- To provide opportunities for interaction between academia and industry.
- To provide platform for lifelong learning by accepting the change in technologies.
- To develop aptitude of fulfilling social responsibilities.

Program Outcomes (PO):

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

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

- **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.

- **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.

- **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.

- **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.

- **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.

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

- **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

- **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.

- **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 and in multidisciplinary environments.

- **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 change.

Program Educational Objectives (PEO):

PEO1: To provide students with the fundamentals of Engineering Sciences with more emphasis in computer science and engineering by way of analyzing and exploiting engineering challenges.

PEO2: To train students with good scientific and engineering knowledge so as to comprehend, analyze, design, and create novel products and solutions for the real life problems.

PEO3: To inculcate professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, entrepreneurial thinking and an ability to relate engineering issues with social issues.

PEO4: To provide students with an academic environment aware of excellence,leadership, written ethical codes and guidelines, and the self-motivated life-long learning needed for a successful professional career.

PEO5: To prepare students to excel in Industry and Higher education by educating Students along with High moral values and Knowledge.

Program Specific Outcome (PSO):

PSO: Ability to interpret and analyze network specific and cyber security issues, automation in real word environment.
PSO2: Ability to Design and Develop Mobile and Web-based applications under realistic constraints.

Course Outcome (CO):

CO1: Implement geometric images using graphical input techniques

CO2: Design and develop images with the help of 2D & 3D transformations.

CO3: Identify visible surfaces for generation of realistic graphics display and curves representation.

CO4: Analyse multimedia and animation techniques.

CO-PO Mapping												
Computer Graphics & Multimedia Techniques 5CS4-04												
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO1 : Implement geometric images using graphical input techniques	3	2	3	2	2	1	1	1	1	2	1	3
CO2: Design and develop images with the help of 2D & 3D transformations.	3	2	3	2	2	2	1	1	1	2	1	3
CO3: Identify visible surfaces for generation of realistic graphics display and curves representation.	3	3	3	3	2	2	1	2	1	2	1	3
CO4: Analyse multimedia and animation techniques.	3	3	3	3	3	2	2	1	1	2	2	3

CO-PSO Mapping		
CO's	**PSO1**	**PSO2**
CO1: Implement geometric images using graphical input techniques	2	2
CO2: Design and develop images with the help of 2D & 3D transformations.	2	2
CO3: Identify visible surfaces for generation of realistic graphics display and curves representation.	2	2
CO4: Analyse multimedia and animation techniques.	2	2

RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Syllabus

III Year-V Semester: B.Tech. Computer Science and Engineering

5CS4-04: Computer Graphics & Multimedia

Credit: 3 **Max. Marks: 150(IA:30, ETE:120)**

3L+0T+0P **End Term Exam: 3 Hours**

SN	Contents	Hours
1	**Introduction:** Objective, scope and outcome of the course.	01
2	**Basic of Computer Graphics:**Basic of Computer Graphics, Applications of computer graphics, Display devices, Random and Raster scan systems, Graphics input devices, Graphics software and standards	06
3	**Graphics Primitives:**Points, lines, circles and ellipses as primitives, scan conversion algorithms for primitives, Fill area primitives including scan-line polygon filling, inside-outside test, boundary and flood-fill, character generation, line attributes, area-fill attributes, character attributers. Aliasing, and introduction to Anti Aliasing (No anti aliasing algorithm).	07
4	**Two Dimensional Graphics:**Transformations (translation, rotation, scaling), matrix representation, homogeneous coordinates, composite transformations, reflection and shearing, viewing pipeline and coordinates system, window-to-viewport transformation, clipping including point clipping, line clipping (cohen-sutherland, liang- bersky, NLN), polygon clipping	08
5	**Three Dimensional Graphics:**3D display methods, polygon surfaces, tables, equations, meshes, curved lies and surfaces, quadric surfaces, spline representation, cubic spline interpolation methods, Bazier curves and surfaces, B-spline curves and surfaces.3D scaling, rotation and translation, composite transformation, viewing pipeline and coordinates, parallel and perspective transformation, view volume and general (parallel and perspective) projection transformations.	08
6	**Illumination and Colour Models:**Light sources – basic illumination models – halftone patterns and dithering techniques; Properties of light – Standard primaries and chromaticity diagram; Intuitive colour concepts – RGB colour model – YIQ colour model – CMY colour model – HSV colour model – HLS colour model; Colour selection.	06
7	**Animations &Realism:**Design of Animation sequences – animation function – raster animation – key frame systems – motion specification – morphing – tweening. **ComputerGraphics Realism:** Tiling the plane – Recursively defined curves – Koch curves – C curves – Dragons – space filling curves – fractals – Grammar based models – fractals – turtle graphics – ray tracing.	06
	Total	42

Lecture Plan of Computer Graphics & Multimedia Techniques (CGMT) 5CS4-04

Unit No./ Total lec. Req.	**Topics**	**Lect. Req.**
Unit-1(1)	Introduction: Objective, scope and outcome of the course.	1
Unit-2 (6)	Basic of Computer Graphics	1
	Applications of computer graphic, Display devices	2
	Random and Raster scan systems	1
	Graphics input devices, Graphics software and standards	2
Unit-3(7)	Points,lines,circles and ellipses as primitives, scan conversion algorithms for primitives	3
	Fill area primitives including scan- line polygon filling, inside-outside test, boundary and flood-fill	2
	character generation, line attributes, area-fill attributes, character attributers. Aliasing, and introduction to Anti Aliasing (No anti aliasing algorithm)	2
Unit-4(8)-	Transformations (translation, rotation, scaling), matrix representation	1
	homogeneous coordinates, composite transformations, reflection and shearing	1
	viewing pipeline and coordinates system	2
	window-to-viewport transformation, clipping including point clipping	2
	line clipping (cohen-sutherland,liang- bersky, NLN), polygon clipping	2
Unit-5(8)	3D display methods, polygon surfaces, tables, equations, meshes, curved lies and surfaces	2
	quadric surfaces, spline representation, cubic spline interpolation methods, Bazier curves and surfaces	2
	B-spline curves and surfaces.3D scaling, rotation and translation, composite transformation	2
	viewing pipeline and coordinates, parallel and perspective transformation	1
	view volume and general (parallel and perspective) projection transformations.	1
Unit-6(6)	Light sources – basic illumination models – halftone patterns and dithering techniques	1

	Properties of light – Standard primaries and chromaticity diagram	2
	Intuitive colour concepts – RGB colour model – YIQ colour model – CMY colour model – HSV colour model – HLS colour model	2
	Colour selection.	1
Unit-7(6)	Design of Animation sequences – animation function – raster animation – key frame systems – motion specification – morphing – tweening	3
	Tiling the plane – Recursively defined curves – Koch curves – C curves – Dragons – space filling curves –fractals – Grammar based models – fractals – turtle graphics – ray tracing.	3
	Total No. of Lecture	42

This schedule is tentative and is subject to minimal changes during teaching.

Points and Lines

- Point plotting is done by converting a single coordinate position furnished by an application program into appropriate operations for the output device in use.
- Line drawing is done by calculating intermediate positions along the line path between two specified endpoint positions.
- The output device is then directed to fill in those positions between the end points with some color.
- For some device such as a pen plotter or random scan display, a straight line can be drawn smoothly from one end point to other.
- 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 intensity is loaded in frame buffer at the corresponding pixel positions.
- 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.
- For example line position of $(12.36, 23.87)$ would be converted to pixel position $(12, 24)$.
- This rounding of coordinate values to integers causes lines to be displayed with a stair step appearance ("the jaggies"), as represented in fig 2.1.

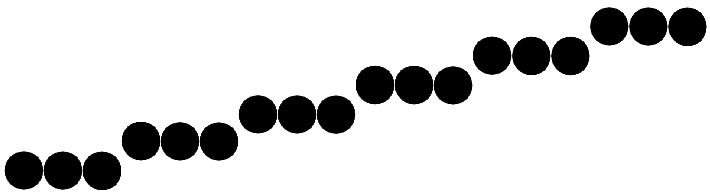


Fig. 2.1: - Stair step effect produced when line is generated as a series of pixel positions.

- The stair step shape is noticeable in low resolution system, and we can improve their appearance somewhat by displaying them on high resolution system.
- More effective techniques for smoothing raster lines are based on adjusting pixel intensities along the line paths.
- For raster graphics device-level algorithms discuss here, object positions are specified directly in integer device coordinates.
- Pixel position will referenced according to scan-line number and column number which is illustrated by following figure.

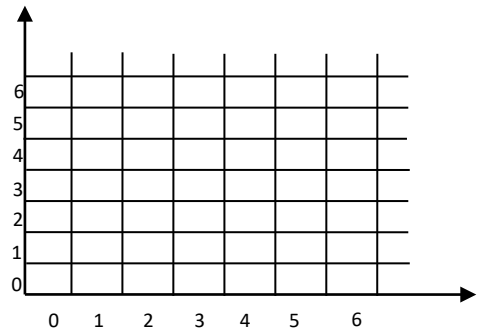


Fig. 2.2: - Pixel positions referenced by scan-line number and column number.

- To load the specified color into the frame buffer at a particular position, we will assume we have available low-level procedure of the form $setpixel(x, y)$.

- Similarly for retrieve the current frame buffer intensity we assume to have procedure $getpixel(x,y)$.

Line Drawing Algorithms

- The Cartesian slop-intercept equation for a straight line is "$y = mx + b$" with 'm' representing slop and 'b' as the intercept.
- The two endpoints of the line are given which are say $(x1, y1)$ and $(x2,y2)$.

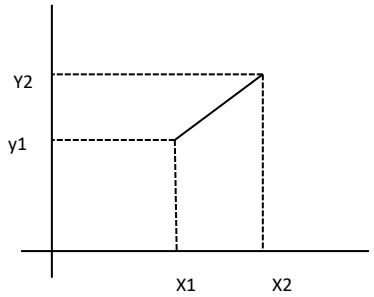


Fig. 2.3: - Line path between endpoint positions.

- We can determine values for the slope m by equation:
  $m = (y2 - y1)/(x2 - x1)$
- We can determine values for the intercept b by equation:
  $b = y1 - m * x1$
- For the given interval Δx along a line, we can compute the corresponding y interval Δy as:
  $\Delta y = m * \Delta x$
- Similarly for Δx:
  $\Delta x = \Delta y/m$
- For line with slop $|m| < 1$, Δx can be set proportional to small horizontal deflection voltage and the corresponding vertical deflection voltage is then set proportional to Δy which is calculated from above equation.
- For line with slop $|m| > 1$, Δy can be set proportional to small vertical deflection voltage and the corresponding horizontal deflection voltage is then set proportional to Δx which is calculated from above equation.
- For line with slop $m = 1$, $\Delta x = \Delta y$ and the horizontal and vertical deflection voltages are equal.

DDA Algorithm

- Digital differential analyzer (DDA) is scan conversion line drawing algorithm based on calculating either Δy or Δx using above equation.
- We sample the line at unit intervals in one coordinate and find corresponding integer values nearest the line path for the other coordinate.
- Consider first a line with positive slope and slope is less than or equal to 1:
  We sample at unit x interval $(\Delta x = 1)$ and calculate each successive y value as follow:
  $y = m * x + b$
  $y_k = m * (x + 1) + b$
  In general $y_k = m * (x + k) + b$, &
  $y_{k+1} = m * (x + k + 1) + b$
  Now write this equation in form:
  $y_{k+1} - y_k = (m * (x + k + 1) + b) - (m * (x + k) + b)$
  $y_{k+1} = y_k + m$
  So that it is computed fast in computer as addition is fast compare to multiplication.

- In above equation k takes integer values starting from 1 and increase by 1 until the final endpoint is reached.
- As m can be any real number between 0 and 1, the calculated y values must be rounded to the nearest integer.
- Consider a case for a line with a positive slope greater than 1:
  We change the role of x and y that is sample at unit y intervals $(\Delta y = 1)$ and calculate each succeeding x value as:
  $x = (y - b)/m$
  $x_1 = ((y + 1) - b)/m$
  In general $x_k = ((y + k) - b)/m$, &
  $x_{k+1} = ((y + k + 1) - b)/m$
  Now write this equation in form:
  $x_{k+1} - x_k = (((y + k + 1) - b)/m) - (((y + k) - b)/m)$
  $x_{k+1} = x_k + 1/m$
- Above both equations are based on the assumption that lines are to be processed from left endpoint to the right endpoint.
- If we processed line from right endpoint to left endpoint than:
  If $\Delta x = -1$ equation become:
  $y_{k+1} = y_k - m$
  If $\Delta y = -1$ equation become:
  $x_{k+1} = x_k - 1/m$
- Above calculated equations also used to calculate pixel position along a line with negative slope.
- **Procedure for DDA line algorithm.**

```
Void lineDDA (int xa, int ya, int xb, int yb)
{
    int dx = xb – xa, dy = yb – ya, steps, k;
    float xincrement, yincrement, x = xa, y = ya;
    if (abs(dx)>abs(dy))
    {
            Steps = abs (dx);
    }
    else
    {
            Steps = abs (dy);
    }
    xincrement = dx/(float) steps;
    yincrement = dy/(float) steps;

    setpixel (ROUND (x), ROUND (y));
    for(k=0;k<steps;k++)
    {
            x += xincrement;
            y += yincrement;
            setpixel (ROUND (x), ROUND (y));
    }
}
```

Advantages of DDA algorithm

- It is faster algorithm.
- It is simple algorithm.

Disadvantage of DDA algorithm

- Floating point arithmetic is time consuming.
- Poor end point accuracy.

Bresenham's Line Algorithm

- An accurate and efficient raster line-generating algorithm, developed by Bresenham which scan converts line using only incremental integer calculations that can be modified to display circles and other curves.
- Figure shows section of display screen where straight line segments are to be drawn.

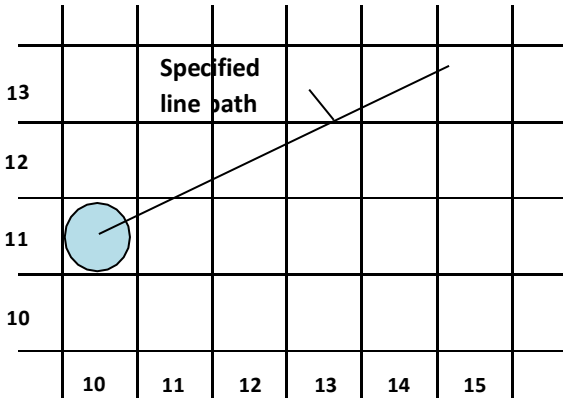


Fig. 2.4: - Section of a display screen where a straight line segment is to be plotted, starting from the pixel at column 10 on scan line 11.

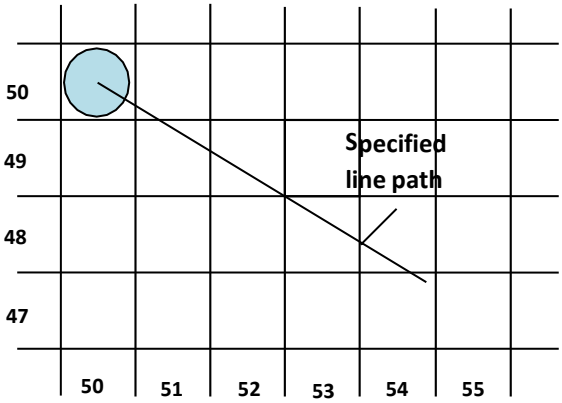


Fig. 2.5: - Section of a display screen where a negative slope line segment is to be plotted, starting from the pixel at column 50 on scan line 50.

- The vertical axes show scan-line positions and the horizontal axes identify pixel column.
- Sampling at unit x intervals in these examples, we need to decide which of two possible pixel position is closer to the line path at each sample step.
- To illustrate bresenham's approach, we first consider the scan-conversion process for lines with positive slope less than 1.
- Pixel positions along a line path are then determined by sampling at unit x intervals.
- Starting from left endpoint (x_0, y_0) of a given line, we step to each successive column and plot the pixel whose scan-line y values is closest to the line path.
- Assuming we have determined that the pixel at (x_k, y_k) is to be displayed, we next need to decide which pixel to plot in column $x_k + 1$.
- Our choices are the pixels at positions $(x_k + 1, y_k)$ and $(x_k + 1, y_k + 1)$.
- Let's see mathematical calculation used to decide which pixel position is light up.
- We know that equation of line is:
  $y = mx + b$
  Now for position $x_k + 1$.
  $y = m(x_k + 1) + b$
- Now calculate distance bet actual line's y value and lower pixel as d_1 and distance bet actual line's y value and upper pixel as d_2.
  $d_1 = y - y_k$

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

$d_2 = (y_k + 1) - y$

$d_2 = (y_k + 1) - m(x_k + 1) - b$ ..... (2)

- Now calculate $d_1 - d_2$ from equation (1) and (2).

  $d_1 - d_2 = (y - y_k) - ((y_k + 1) - y)$

  $d_1 - d_2 = \{m(x_k + 1) + b - y_k\} - \{(y_k + 1) - m(x_k + 1) - b\}$

  $d_1 - d_2 = \{mx_k + m + b - y_k\} - \{y_k + 1 - mx_k - m - b\}$

  $d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1$ ..... (3)

- Now substitute $m = \Delta y/\Delta x$ in equation (3)

  $d_1 - d_2 = 2\left(\frac{\Delta y}{\Delta x}\right)(x_k + 1) - 2y_k + 2b - 1$ ..... (4)

- Now we have decision parameter p_k for k^{th} step in the line algorithm is given by:

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

  $p_k = \Delta x(2\Delta y/\Delta x(x_k + 1) - 2y_k + 2b - 1)$

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

  $p_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + 2\Delta x b - \Delta x$ ..... (5)

  $p_k = 2\Delta y x_k - 2\Delta x y_k + C\ (Where\ Constant\ C = 2\Delta y + 2\Delta x b - \Delta x)$ ..... (6)

- The sign of p_k is the same as the sign of $d_1 - d_2$, since $\Delta x > 0$ for our example.
- Parameter c is constant which is independent of pixel position and will eliminate in the recursive calculation for p_k.
- Now if p_k is negative then we plot the lower pixel otherwise we plot the upper pixel.
- So successive decision parameters using incremental integer calculation as:

  $p_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + C$

- Now Subtract pk from p_{k+1}

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

  $p_{k+1} - p_k = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + C - 2\Delta y x_k + 2\Delta x y_k - C$

  But $x_{k+1} = x_k + 1$, so that $(x_{k+1} - x_k) = 1$

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

- Where the terms $y_{k+1} - y_k$ is either 0 or 1, depends on the sign of parameter p_k.
- This recursive calculation of decision parameters is performed at each integer x position starting at the left coordinate endpoint of the line.
- The first decision parameter p_0 is calculated using equation (5) as first time we need to take constant part into account so:

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

  $p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x b - \Delta x$

  Now $Substitute\ b = y_0 - mx_0$

  $p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x(y_0 - mx_0) - \Delta x$

  $Now\ Substitute\ m = \Delta y/\Delta x$

  $p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x(y_0 - (\Delta y/\Delta x)x_0) - \Delta x$

  $p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x y_0 - 2\Delta y x_0 - \Delta x$

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

- Let's see Bresenham's line drawing algorithm for $|m| < 1$
  1. Input the two line endpoints 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 $(x_k + 1, y_k)$ and
   $$p_{k+1} = p_k + 2\Delta y$$
   Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and
   $$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$
5. Repeat step-4 Δx times.

- Bresenham's algorithm is generalized to lines with arbitrary slope by considering symmetry between the various octants and quadrants of the xy plane.
- For lines with positive slope greater than 1 we interchange the roles of the x and y directions.
- Also we can revise algorithm to draw line from right endpoint to left endpoint, both x and y decrease as we step from right to left.
- When $d_1 - d_2 = 0$ we choose either lower or upper pixel but once we choose lower than for all such case for that line choose lower and if we choose upper the for all such case choose upper.
- For the negative slope the procedure are similar except that now one coordinate decreases as the other increases.
- The special case handle separately. Horizontal line $(\Delta y = 0)$, vertical line $(\Delta x = 0)$ and diagonal line with $|\Delta x| = |\Delta y|$ each can be loaded directly into the frame buffer without processing them through the line plotting algorithm.

Parallel Execution of Line Algorithms

- The line-generating algorithms we have discussed so far determine pixel positions sequentially.
- With parallel computer we can calculate pixel position along a line path simultaneously by dividing work among the various processors available.
- One way to use multiple processors is partitioning existing sequential algorithm into small parts and compute separately.
- Alternatively we can go for other ways to setup the processing so that pixel positions can be calculated efficiently in parallel.
- Important point to be taking into account while devising parallel algorithm is to balance the load among the available processors.
- Given $\boldsymbol{n_p}$ number of processors we can set up parallel Bresenham line algorithm by subdividing the line path into $\boldsymbol{n_p}$ partitions and simultaneously generating line segment in each of the subintervals.
- For a line with slope $0 < m < 1$ and left endpoint coordinate position (x_0, y_0), we partition the line along the positive x direction.
- The distance between beginning x positions of adjacent partitions can be calculated as:
  $\Delta x_p = (\Delta x + n_p - 1)/n_p$
  Were Δx is the width of the line. And value for partition with Δx_p is computed using integer division.
- Numbering the partitions and the processors, as $0, 1, 2$, up to $\boldsymbol{n_p - 1}$, we calculate the starting x coordinate for the k^{th} partition as:
  $x_k = x_0 + k\Delta x_p$
- To apply Bresenham's algorithm over the partitions, we need the initial value for the y coordinate and the initial value for the decision parameter in each partition.
- The change Δy_p in the y direction over each partition is calculated from the line slope m and partition width Δx_p:
- $\Delta y_p = m\Delta x_p$

- At the k^{th} partition, the starting y coordinate is then
- $y_k = y_0 + round(k\Delta y_p)$
- The initial decision parameter for Bresenham's algorithm at the start of the k^{th} subinterval is obtained from Equation(6):

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

  $p_k = 2\Delta y(x_0 + k\Delta x) - 2\Delta x(y_0 + round(k\Delta y_p)) + 2\Delta y + 2\Delta x(y_0 - \frac{\Delta y}{\Delta x}x_0) - \Delta x$

  $p_k = 2\Delta y x_0 - 2\Delta y k\Delta x_p - 2\Delta x y_0 - 2\Delta x round(k\Delta y_p) + 2\Delta y + 2\Delta x y_0 - 2\Delta y x_0 - \Delta x$

  $p_k = 2\Delta y k\Delta x_p - 2\Delta x round(k\Delta y_p) + 2\Delta y - \Delta x$
- Each processor then calculates pixel positions over its assigned subinterval.
- The extension of the parallel Bresenham algorithm to a line with slope greater than 1 is achieved by partitioning the line in the y direction and calculating beginning x values for the positions.
- For negative slopes, we increment coordinate values in one direction and decrement in the other.

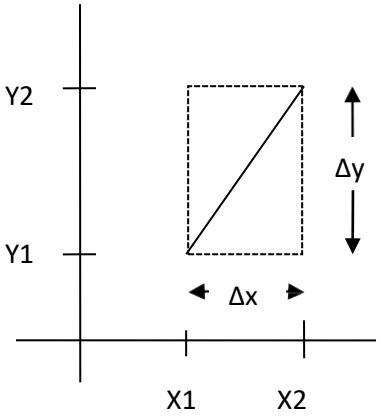


Fig. 2.6: - Bounding box for a line with coordinate extents Δx and Δy.

- Another way to set up parallel algorithms on raster system is to assign each processor to a particular group of screen pixels.
- With sufficient number of processor we can assign each processor to one pixel within some screen region.
- This approach can be adapted to line display by assigning one processor to each of the pixels within the limit of the bounding rectangle and calculating pixel distance from the line path.
- The number of pixels within the bounding rectangle of a line is $\Delta x \times \Delta y$.
- Perpendicular distance d from line to a particular pixel is calculated by:

  $d = Ax + By + C$

  Where

  $A = -\Delta y / linelength$

  $B = -\Delta x / linelength$

  $C = (x_0\Delta y - y_0\Delta x)/linelength$

  With

  $linelength = \sqrt{\Delta x^2 + \Delta y^2}$
- Once the constant A, B, and C have been evaluated for the line each processors need to perform two multiplications and two additions to compute the pixel distance d.
- A pixel is plotted if d is less than a specified line thickness parameter.
- Instead of partitioning the screen into single pixels, we can assign to each processor either a scan line or a column a column of pixels depending on the line slope.

- Each processor calculates line intersection with horizontal row or vertical column of pixels assigned to that processor.
- If vertical column is assign to processor then x is fix and it will calculate y and similarly is horizontal row is assign to processor then y is fix and x will be calculated.
- Such direct methods are slow in sequential machine but we can perform very efficiently using multiple processors.

Circle

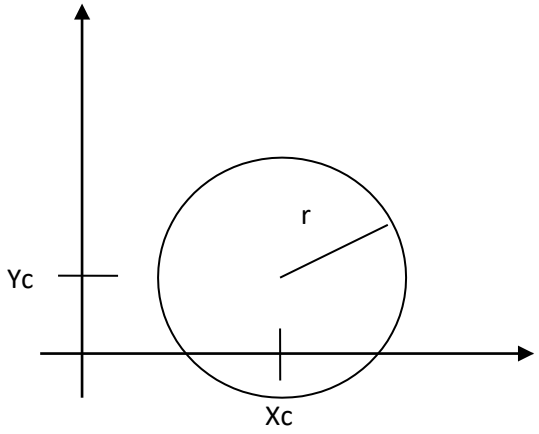


Fig. 2.7: - Circle with center coordinates (x_c, y_c) and radius r.

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

Properties of Circle

- The distance relationship is expressed by the Pythagorean theorem in Cartesian coordinates as:
  $(x - x_c)^2 + (y - y_c)^2 = r^2$
- We could use this equation to calculate circular boundary points by incrementing 1 in x direction in every steps from $x_c - r$ to $x_c + r$ and calculate corresponding y values at each position as:
  $(x - x_c)^2 + (y - y_c)^2 = r^2$
  $(y - y_c)^2 = r^2 - (x - x_c)^2$
  $(y - y_c) = \pm\sqrt{r^2 - (x_c - x)^2}$
  $y = y_c \pm \sqrt{r^2 - (x_c - x)^2}$
- But this is not best method for generating a circle because it requires more number of calculations which take more time to execute.
- And also spacing between the plotted pixel positions is not uniform as shown in figure below.

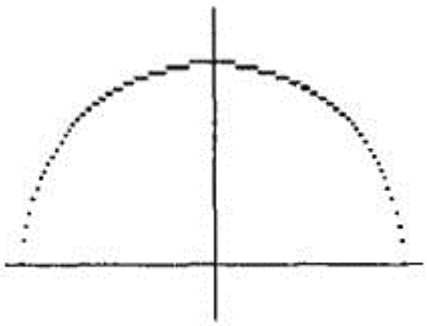


Fig. 2.8: - Positive half of circle showing non uniform spacing bet calculated pixel positions.

- We can adjust spacing by stepping through y values and calculating x values whenever the absolute value of the slop of the circle is greater than 1. But it will increases computation processing requirement.
- Another way to eliminate the non-uniform spacing is to draw circle using polar coordinates 'r' and 'θ'.
- Calculating circle boundary using polar equation is given by pair of equations which is as follows.
  $x = x_c + r\cos\theta$
  $y = y_c + r\sin\theta$
- When display is produce using these equations using fixed angular step size circle is plotted with uniform spacing.
- The step size 'θ' is chosen according to application and display device.
- For a more continuous boundary on a raster display we can set the step size at $1/r$. This plot pixel position that are approximately one unit apart.
- Computation can be reduced by considering symmetry city property of circles. The shape of circle is similar in each quadrant.
- We can obtain pixel position in second quadrant from first quadrant using reflection about y axis and similarly for third and fourth quadrant from second and first respectively using reflection about x axis.
- We can take one step further and note that there is also symmetry between octants. Circle sections in adjacent octant within one quadrant are symmetric with respect to the 45^0 line dividing the two octants.
- This symmetry condition is shown in figure below where point (x, y) on one circle sector is mapped in other seven sector of circle.

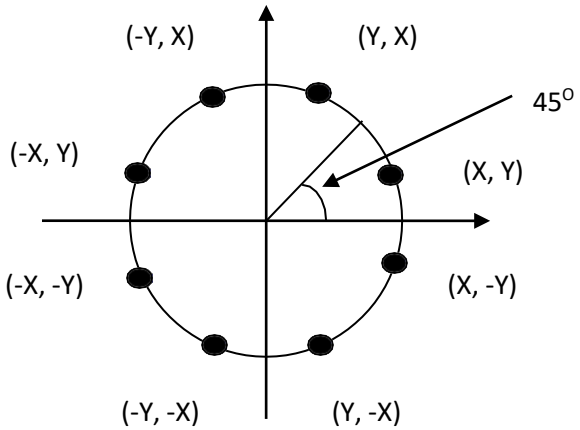


Fig. 2.9: - symmetry of circle.

- Taking advantage of this symmetry property of circle we can generate all pixel position on boundary of circle by calculating only one sector from $x = 0$ to $x = y$.
- Determining pixel position along circumference of circle using any of two equations shown above still required large computation.
- More efficient circle algorithm are based on incremental calculation of decision parameters, as in the Bresenham line algorithm.
- Bresenham's line algorithm can be adapted to circle generation by setting decision parameter for finding closest pixel to the circumference at each sampling step.
- The Cartesian coordinate circle equation is nonlinear so that square root evaluations would be required to compute pixel distance from circular path.
- Bresenham's circle algorithm avoids these square root calculation by comparing the square of the pixel separation distance.

- A method for direct distance comparison to test the midpoint between two pixels to determine if this midpoint is inside or outside the circle boundary.
- This method is easily applied to other conics also.
- Midpoint approach generates same pixel position as generated by bresenham's circle algorithm.
- The error involve in locating pixel positions along any conic section using midpoint test is limited to one-half the pixel separation.

Midpoint Circle Algorithm

- Similar to raster line algorithm we sample at unit interval and determine the closest pixel position to the specified circle path at each step.
- Given radius 'r' and center (x_c, y_c)
- We first setup our algorithm to calculate circular path coordinates for center (0, 0). And then we will transfer calculated pixel position to center (x_c, y_c) by adding x_c to x and y_c to y.
- Along the circle section from $x = 0$ to $x = y$ in the first quadrant, the slope of the curve varies from 0 to -1 so we can step unit step in positive x direction over this octant and use a decision parameter to determine which of the two possible y position is closer to the circular path.
- Position in the other seven octants are then obtain by symmetry.
- For the decision parameter we use the circle function which is:
  $f_{circle}(x, y) = x^2 + y^2 - r^2$
- Any point which is on the boundary is satisfied $f_{circle}(x, y) = 0$ if the point is inside circle function value is negative and if point is outside circle the function value is positive which can be summarize as below.

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

- Above equation we calculate for the mid positions between pixels near the circular path at each sampling step and we setup incremental calculation for this function as we did in the line algorithm.
- Below figure shows the midpoint between the two candidate pixels at sampling position $x_k + 1$.

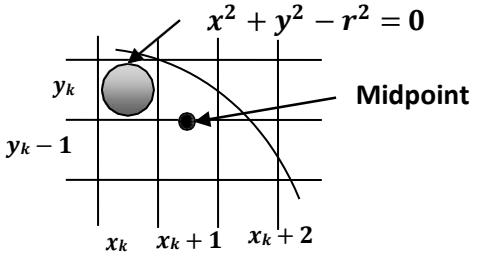


Fig. 2.10: - Midpoint between candidate pixel at sampling position $x_k + 1$ along circle path.

- Assuming we have just plotted the pixel at (x_k, y_k) and next we need to determine whether the pixel at position '$(x_k + 1, y_k)$' or the one at position' $(x_k + 1, y_k - 1)$' is closer to circle boundary.
- So for finding which pixel is more closer using decision parameter evaluated at the midpoint between two candidate pixels as below:

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

  $p_k = (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2$
- If $p_k < 0$ this midpoint is inside the circle and the pixel on the scan line y_k is closer to circle boundary. Otherwise the midpoint is outside or on the boundary and we select the scan line $y_k - 1$.
- Successive decision parameters are obtain using incremental calculations as follows:

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

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

- Now we can obtain recursive calculation using equation of p_{k+1} and p_k as follow.

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

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

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

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

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

- In above equation y_{k+1} is either y_k or $y_k - 1$ depending on the sign of the p_k.
- Now we can put $2x_{k+1} = 2x_k + 2$ and when we select $y_{k+1} = y_k - 1$ we can obtain $2y_{k+1} = 2y_k - 2$.
- The initial decision parameter is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$ as follows.

$p_0 = f_{circle}(0 + 1, r - \frac{1}{2})$

$p_0 = 1 + (r - \frac{1}{2})^2 - r$

$p_0 = 1 + r^2 - r + \frac{1}{4} - r^2$

$p_0 = \frac{5}{4} - r$

Algorithm for Midpoint Circle Generation

1. Input 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
$$p_0 = \frac{5}{4} - r$$
3. At each x_k position, starting at $k = 0$, perform the following test:
If $p_k < 0$, the next point along the circle centered on $(0, 0)$ is $(x_k + 1, y_k)$ &
$$p_{k+1} = p_k + 2x_{k+1} + 1$$
Otherwise, the next point along the circle is $(x_k + 1, y_k - 1)$ &
$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$
Where $2x_{k+1} = 2x_k + 2$, & $2y_{k+1} = 2y_k - 2$.
4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x, y) onto the circular path centered on (x_c, y_c) and plot the coordinate values:
$$x = x + x_c, y = y + y_c$$
6. Repeat steps 3 through 5 until $x \geq y$.

Ellipse

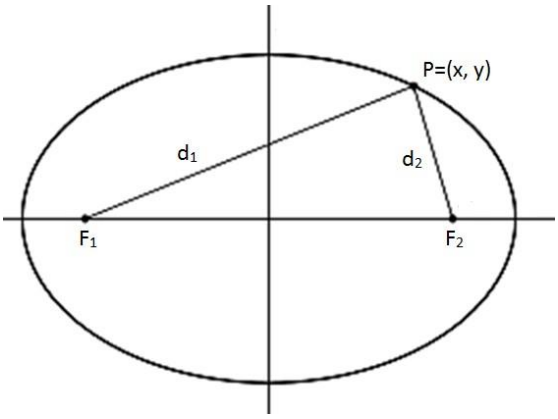


Fig. 2.11: - Ellipse generated about foci f_1 and f_2.

- AN ellipse is defined as the set of points such that the sum of the distances from two fixed positions (**foci**) is same for all points.

Properties of Ellipse

- If we labeled distance from two foci to any point on ellipse boundary as d_1 and d_2 then the general equation of an ellipse can be written as follow.
  $d_1 + d_2 = Constant$
- Expressing distance in terms of focal coordinates $f_1 = (x_1, y_1)$ and $f_2 = (x_2, y_2)$ we have
  $\sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2} = Constant$
- An interactive method for specifying an ellipse in an arbitrary orientation is to input two foci and a point on the ellipse boundary.
- With this three coordinates we can evaluate constant in equation:
- $\sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2} = Constant$
- We can also write this equation in the form
  $Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$
- Where the coefficients $A, B, C, D, E,$ and F are evaluated in terms of the focal coordinates and the dimensions of the major and minor axes of the ellipse.
- Major axis of an ellipse is straight line segment passing through both foci and extends up to boundary on both sides.
- The minor axis spans shortest dimension of ellipse, it bisect the major axis at right angle in two equal half.
- Then coefficient in $Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$ can be evaluated and used to generate pixels along the elliptical path.
- Ellipse equation are greatly simplified if we align major and minor axis with coordinate axes i.e. $x - axis$ and $y - axis$.
- We can say ellipse is in standard position if their major and minor axes are parallel to $x - axis$ and $y - axis$ which is shown in below figure.

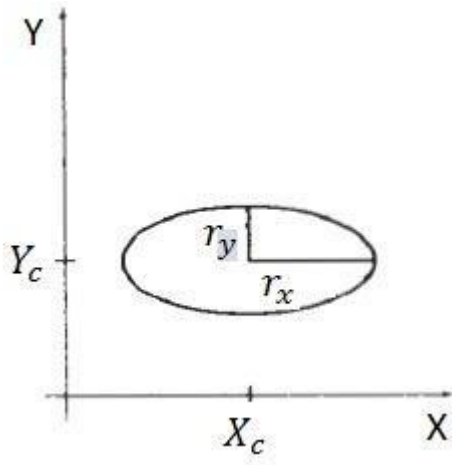


Fig. 2.12: - Ellipse centered at (x_c, y_c) with semi major axis r_x and semi minor axis r_y are parallel to coordinate axis.

- Equation of ellipse shown in figure 2.12 can be written in terms of the ellipse center coordinates and parameters r_x and r_y as.

$$\left(\frac{x - x_c}{r_x}\right)^2 + \left(\frac{y - y_c}{r_y}\right)^2 = 1$$

- Using the polar coordinates r and θ, we can also describe the ellipse in standard position with the parametric equations:

$$x = x_c + r_x \cos\theta$$
$$y = y_c + r_y \sin\theta$$

- Symmetry considerations can be used to further reduced computations.
- An ellipse in standard position is symmetric between quadrants but unlike a circle it is not symmetric between octant.
- Thus we must calculate boundary point for one quadrant and then other three quadrants point can be obtained by symmetry as shown in figure below.

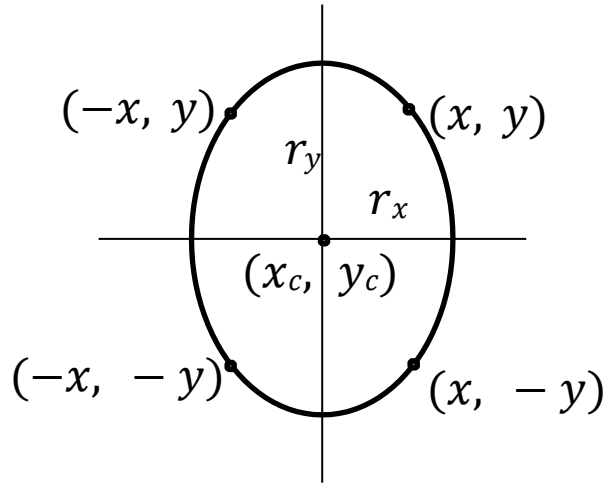


Fig. 2.13: - symmetry of an ellipse.

Midpoint Ellipse Algorithm

- Midpoint ellipse algorithm is a method for drawing ellipses in computer graphics. This method is modified from Bresenham's algorithm.

- The advantage of this modified method is that only addition operations are required in the program loops.
- This leads to simple and fast implementation in all processors.
- Given parameters r_x, r_y and (x_c, y_c) we determine points (x, y) for an ellipse in standard position centered on the origin, and then we shift the points so the ellipse is centered at (x_c, y_c).
- If we want to display the ellipse in nonstandard position then we rotate the ellipse about its center to align with required direction.
- For the present we consider only the standard position.
- In this method we divide first quadrant into two parts according to the slope of an ellipse as shown in figure below.

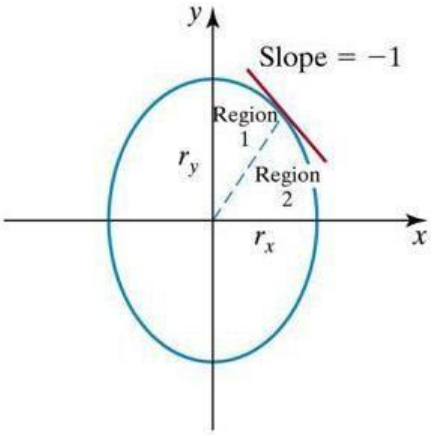


Fig. 2.14: - Ellipse processing regions. Over the region 1 the magnitude of ellipse slope is < 1 and over the region 2 the magnitude of ellipse slope > 1.

- We take unit step in x direction if magnitude of slope is less than 1 in that region otherwise we take unit step in y direction.
- Boundary divides region at $slope = -1$.
- With $r_x < r_y$ we process this quadrant by taking unit steps in x direction in region 1 and unit steps in y direction in region 2.
- Region 1 and 2 can be processed in various ways.
- We can start from $(0, r_y)$ and step clockwise along the elliptical path in the first quadrant shifting from unit step in x to unit step in y when slope becomes less than -1.
- Alternatively, we could start at $(r_x, 0)$ and select points in a counterclockwise order, shifting from unit steps in y to unit steps in x when the slope becomes greater than -1.
- With parallel processors, we could calculate pixel positions in the two regions simultaneously.
- Here we consider sequential implementation of midpoint algorithm. We take the start position at $(0, r_y)$ and steps along the elliptical path in clockwise order through the first quadrant.
- We define ellipse function for center of ellipse at $(0, 0)$ as follows.

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

- Which has the following properties:

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

- Thus the ellipse function serves as the decision parameter in the midpoint ellipse algorithm.
- At each sampling position we select the next pixel from two candidate pixel.

- Starting at $(0, r_y)$ we take unit step in x direction until we reach the boundary between region 1 and 2 then we switch to unit steps in y direction in remaining portion on ellipse in first quadrant.
- At each step we need to test the value of the slope of the curve for deciding the end point of the region-1.
- The ellipse slope is calculated using following equation.
$$\frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y}$$
- At boundary between region 1 and 2 $slope = -1$ and equation become.
$$2r_y^2 x = 2r_x^2 y$$
- Therefore we move out of region 1 whenever following equation is false
$$2r_y^2 x \leq 2r_x^2 y$$
- Following figure shows the midpoint between the two candidate pixels at sampling position $x_k + 1$ in the first region.

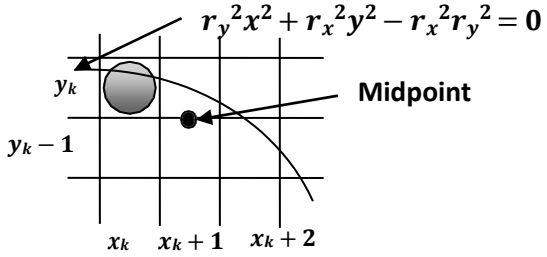


Fig. 2.15: - Midpoint between candidate pixels at sampling position $x_k + 1$ along an elliptical path.

- Assume we are at (x_k , y_k) position and we determine the next position along the ellipse path by evaluating decision parameter at midpoint between two candidate pixels.
$$p1_k = f_{ellipse}\left(x_k + 1, y_k - \frac{1}{2}\right)$$
$$p1_k = r_y^2 (x_k + 1)^2 + r_x^2 \left(y_k - \frac{1}{2}\right)^2 - r_x^2 r_y^2$$
- If $p1_k < 0$, the midpoint is inside the ellipse and the pixel on scan line y_k is closer to ellipse boundary otherwise the midpoint is outside or on the ellipse boundary and we select the pixel $y_k - 1$.
- At the next sampling position decision parameter for region 1 is evaluated as.
$$p1_{k+1} = f_{ellipse}\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$
$$p1_{k+1} = r_y^2 [(x_k + 1) + 1]^2 + r_x^2 \left(y_{k+1} - \frac{1}{2}\right)^2 - r_x^2 r_y^2$$
- Now subtract $p1_k$ from $p1_{k+1}$
$$p1_{k+1} - p1_k = r_y^2 [(x_k + 1) + 1]^2 + r_x^2 \left(y_{k+1} - \frac{1}{2}\right)^2 - r_x^2 r_y^2 - r_y^2 (x_k + 1)^2 - r_x^2 \left(y_k - \frac{1}{2}\right)^2$$
$$p1_{k+1} - p1_k = r_y^2 [(x_k + 1) + 1]^2 + r_x^2 \left(y_{k+1} - \frac{1}{2}\right)^2 - r_y^2 (x_k + 1)^2 - r_x^2 \left(y_k - \frac{1}{2}\right)^2$$
$$p1_{k+1} - p1_k = r_y^2 (x_k + 1)^2 + 2r_y^2 (x_k + 1) + r_y^2 + r_x^2 \left(y_{k+1} - \frac{1}{2}\right)^2 - r_y^2 (x_k + 1)^2 - r_x^2 \left(y_k - \frac{1}{2}\right)^2$$
$$p1_{k+1} - p1_k = 2r_y^2 (x_k + 1) + r_y^2 + r_x^2 \left[\left(y_{k+1} - \frac{1}{2}\right)^2 - \left(y_k - \frac{1}{2}\right)^2\right]$$
- Now making $p1_{k+1}$ as subject.

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

- Here y_{k+1} is either y_k or $y_k - 1$, depends on the sign of $p1_k$
- Now we calculate the initial decision parameter $p1_0$ by putting $(x_0, y_0) = (0, r_y)$ as follow.

$$p1_0 = f_{ellipse}(0 + 1, r_y - \frac{1}{2})$$

$$p1_0 = r_y^2(1) + r_x^2(r_y - \frac{1}{2})^2 - r_x^2 r_y^2$$

$$p1_0 = r_y^2 + r_x^2(r_y - \frac{1}{2})^2 - r_x^2 r_y^2$$

$$p1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

- Now we similarly calculate over region 2 by unit stepping in negative y direction and the midpoint is now taken between horizontal pixels at each step as shown in figure below.

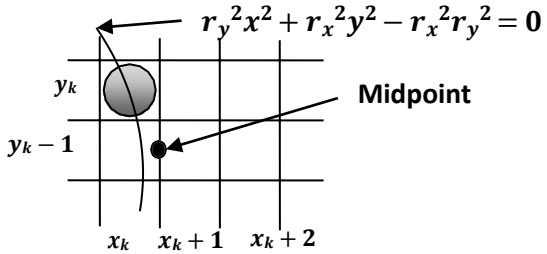


Fig. 2.16: - Midpoint between candidate pixels at sampling position $y_k - 1$ along an elliptical path.

- For this region, the decision parameter is evaluated as follows.

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

$$p2_k = r_y^2(x_k + \frac{1}{2})^2 + r_x^2(y_k - 1)^2 - r_x^2 r_y^2$$

- If $p2_k > 0$ the midpoint is outside the ellipse boundary, and we select the pixel at x_k.
- If $p2_k \leq 0$ the midpoint is inside or on the ellipse boundary and we select $x_k + 1$.
- At the next sampling position decision parameter for region 2 is evaluated as.

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

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

- Now subtract $p2_k$ from $p2_{k+1}$

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

$$p2_{k+1} - p2_k = r_y^2(x_{k+1} + \frac{1}{2})^2 + r_x^2(y_k - 1)^2 - 2r_x^2(y_k - 1) + r_x^2 - r_y^2(x_k + \frac{1}{2})^2 - r_x^2(y_k - 1)^2$$

$$p2_{k+1} - p2_k = r_y^2(x_{k+1} + \frac{1}{2})^2 - 2r_x^2(y_k - 1) + r_x^2 - r_y^2(x_k + \frac{1}{2})^2$$

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

- Now making $p2_{k+1}$ as subject.

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

- Here x_{k+1} is either x_k or $x_k + 1$, depends on the sign of $p2_k$.
- In region 2 initial position is selected which is last position of region one and the initial decision parameter is calculated as follows.

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

$$p2_0 = r_y^2 (x_0 + \frac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

- For simplify calculation of $p2_0$ we could select pixel position in counterclockwise order starting at $(r_x, 0)$.
- In above case we take unit step in the positive y direction up to the last point selected in region 1.

Algorithm for Midpoint Ellipse Generation

1. Input r_x, r_y and ellipse center (x_c, y_c), and obtain the first point on an ellipse centered on the origin as
$$(x_0, y_0) = (0, r_y)$$
2. Calculate the initial value of the decision parameter in region 1 as
$$p1_0 = r_y^2 - r_x^2 r_y^2 + \frac{1}{4} r_x^2$$
3. At each x_k position in region 1, starting at $k = 0$, perform the following test:

   If $p1_k < 0$, the next point along the ellipse centered on $(0, 0)$ is (x_{k+1}, y_k) and
   $$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2$$

   Otherwise, the next point along the ellipse is $(x_{k+1}, y_k - 1)$ and
   $$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} + r_y^2 - 2r_x^2 y_{k+1}$$
   With
   $$2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2$$
   $$2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2$$
   And continue until $2r_y^2 x \leq 2r_x^2 y$
4. Calculate the initial value of the decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as
$$p2_0 = r_y^2 (x_0 + \frac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$
5. At each y_k position in region-2, starting at $k = 0$, perform the following test:

   If $p2_k > 0$, the next point along the ellipse centered on $(0, 0)$ is $(x_k, y_k - 1)$ and
   $$p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2$$
   Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and
   $$p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_x^2 + 2r_y^2 x_{k+1}$$
   Using the same incremental calculations for x and y as in region 1.
6. Determine symmetry points in the other three quadrants.
7. Move each calculated pixel position (x, y) onto the elliptical path centered on (x_c, y_c) and plot the coordinate values:
$$x = x + x_c$$
$$y = y + y_c$$
   Repeat the steps for region 2 until $y_k \geq 0$.

Filled-Area Primitives

- In practical we often use polygon which are filled with some color or pattern inside it.

- 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 line that cross the area.
- Another method is to fill the area is to start from a given interior position and paint out wards from this point until we encounter boundary.

Scan-Line Polygon Fill Algorithm

- Figure below shows the procedure for scan-line filling algorithm.

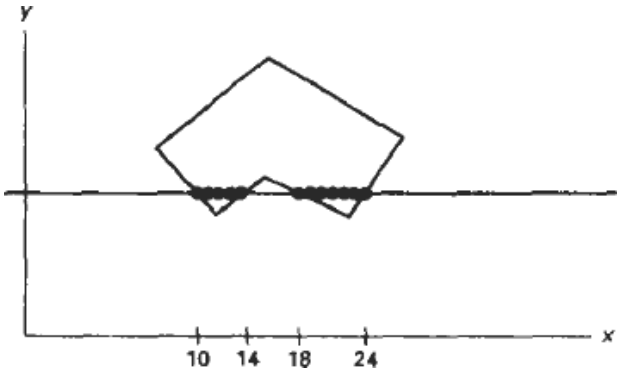


Fig. 2.17: - Interior pixels along a scan line passing through a polygon area.

- For each scan-line crossing a polygon, the algorithm locates the intersection points are of scan line with the polygon edges.
- This intersection points are stored from left to right.
- Frame buffer positions between each pair of intersection point are set to specified fill color.
- Some scan line intersects at vertex position they are required special handling.
- For vertex we must look at the other endpoints of the two line segments of the polygon which meet at this vertex.
- If these points lie on the same (up or down) side of the scan line, then that point is counts as two intersection points.
- If they lie on opposite sides of the scan line, then the point is counted as single intersection.
- This is illustrated in figure below

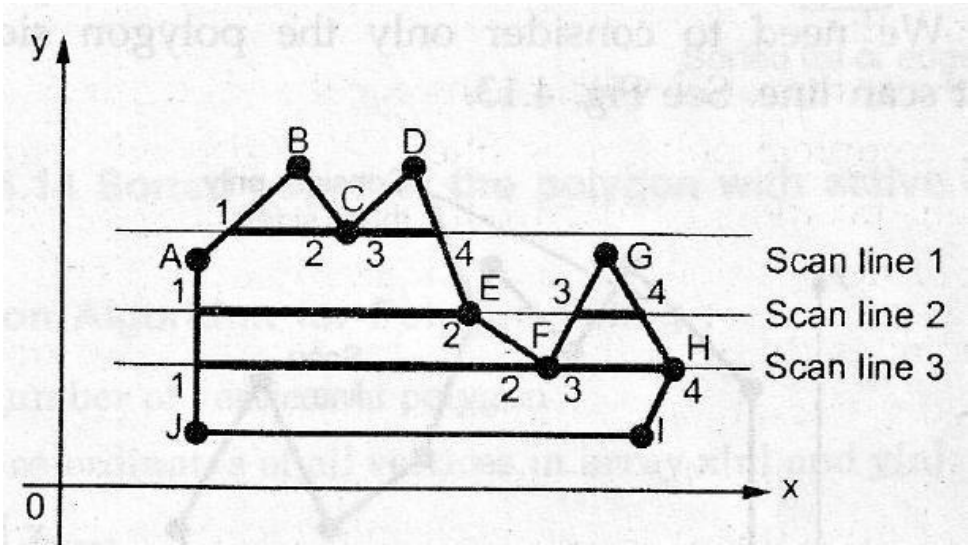


Fig. 2.18: - Intersection points along the scan line that intersect polygon vertices.

- As shown in the Fig. 2.18, each scan line intersects the vertex or vertices of the polygon. For scan line 1, the other end points (B and D) of the two line segments of the polygon lie on the same side of the scan

line, hence there are two intersections resulting two pairs: 1 -2 and 3 - 4. Intersections points 2 and 3 are actually same Points. For scan line 2 the other endpoints (D and F) of the two line segments of the Polygon lie on the opposite sides of the scan line, hence there is a single intersection resulting two pairs: l - 2 and 3 - 4. For scan line 3, two vertices are the intersection points"

- For vertex F the other end points E and G of the two line segments of the polygon lie on the same side of the scan line whereas for vertex H, the other endpoints G and I of the two line segments of the polygon lie on the opposite side of the scan line. Therefore, at vertex F there are two intersections and at vertex H there is only one intersection. This results two pairs: 1 - 2 and 3 - 4 and points 2 and 3 are actually same points.
- Coherence methods often involve incremental calculations applied along a single scan line or between successive scan lines.
- In determining edge intersections, we can set up incremental coordinate calculations along any edge by exploiting the fact that the slope of the edge is constant from one scan line to the next.
- Figure below shows three successive scan-lines crossing the left edge of polygon.

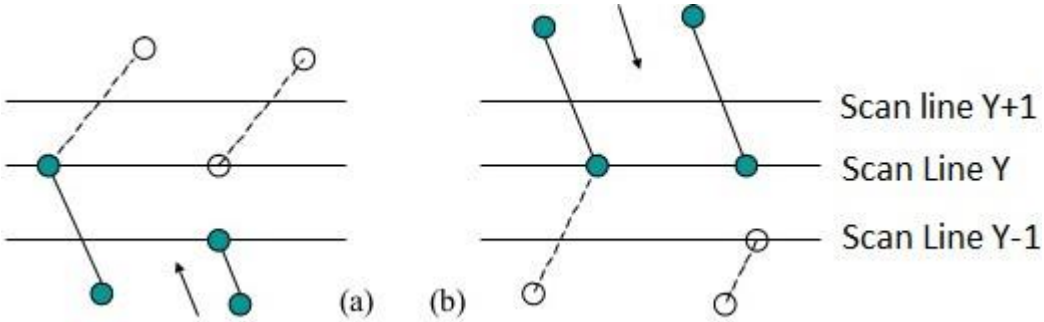


Fig. 2.18: - adjacent scan line intersects with polygon edge.

- For above figure we can write slope equation for polygon boundary as follows.

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

- Since change in y coordinates between the two scan lines is simply

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

- So slope equation can be modified as follows

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

$$m = \frac{1}{x_{k+1} - x_k}$$

$$x_{k+1} - x_k = \frac{1}{m}$$

$$x_{k+1} = x_k + \frac{1}{m}$$

- Each successive x intercept can thus be calculated by adding the inverse of the slope and rounding to the nearest integer.
- For parallel execution of this algorithm we assign each scan line to separate processor in that case instead of using previous x values for calculation we use initial x values by using equation as.

$$x_k = x_0 + \frac{k}{m}$$

- Now if we put $m = \frac{\Delta y}{\Delta x}$ in incremental calculation equation $x_{k+1} = x_k + \frac{1}{m}$ then we obtain equation as.

$$x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$$

- Using this equation we can perform integer evaluation of x intercept by initializing a counter to 0, then incrementing counter by the value of Δx each time we move up to a new scan line.
- When the counter value becomes equal to or greater than Δy, we increment the current x intersection value by 1 and decrease the counter by the value Δy.
- This procedure is seen by following figure.

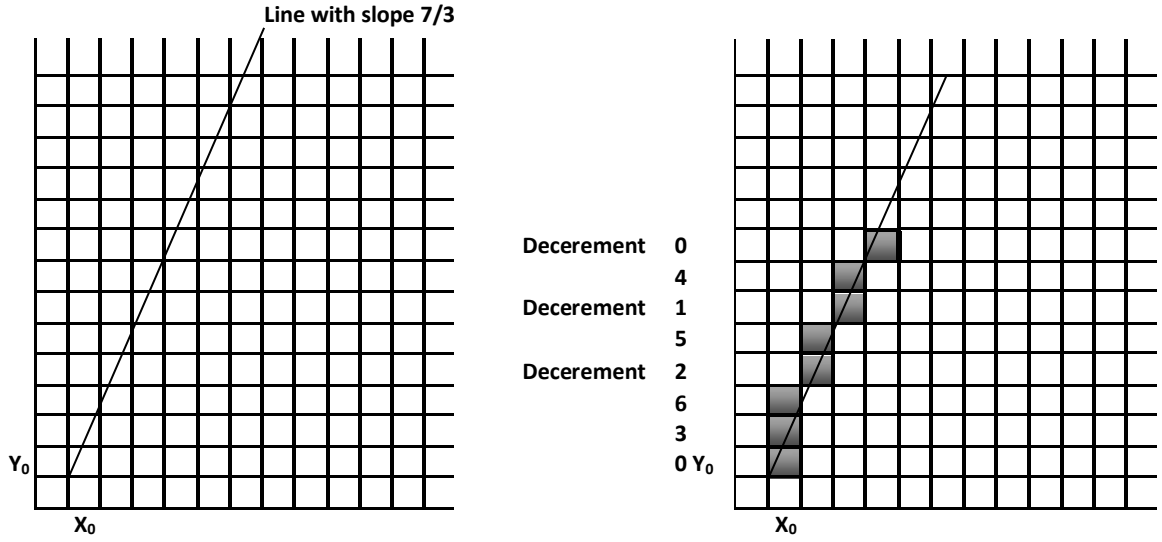


Fig. 2.19: - line with slope 7/3 and its integer calculation using equation $x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$

- Steps for above procedure

1. Suppose m = 7/3
2. Initially, set counter to 0, and increment to 3 (which is Δx).
3. When move to next scan line, increment counter by adding Δx
4. When counter is equal or greater than 7 (which is Δy), increment the x-intercept (in other words, the x-intercept for this scan line is one more than the previous scan line), and decrement counter by 7(which is Δy).

- To efficiently perform a polygon fill, we can first store the polygon boundary in a sorted edge table that contains all the information necessary to process the scan lines efficiently.
- We use bucket sort to store the edge sorted on the smallest y value of each edge in the correct scan line positions.
- Only the non-horizontal edges are entered into the sorted edge table.
- Figure below shows one example of storing edge table.

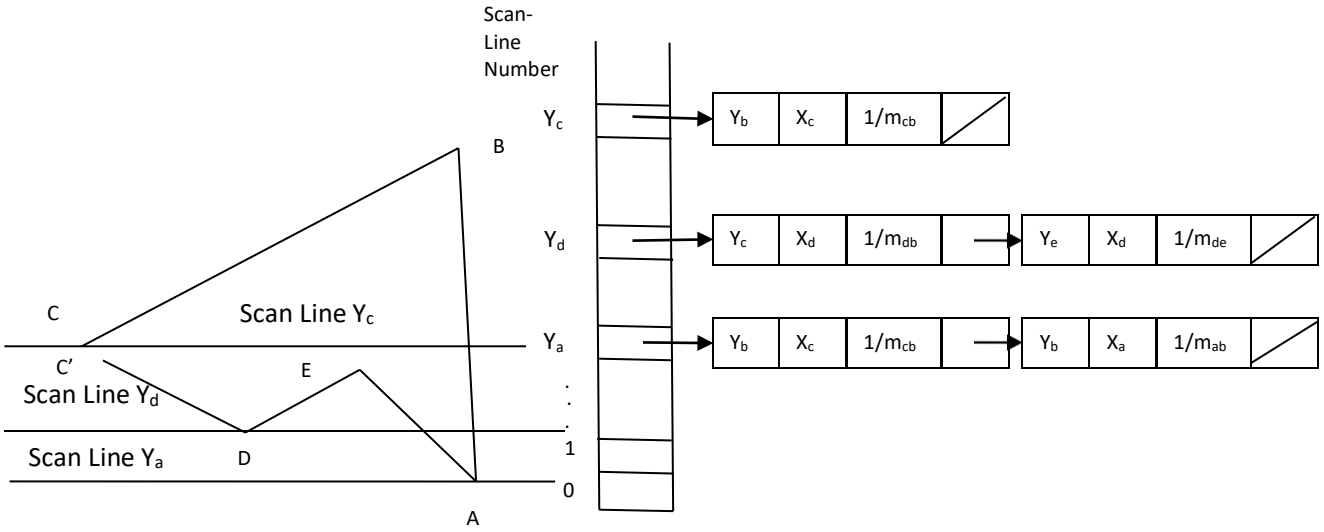


Fig. 2.20: - A polygon and its sorted edge table.

- Each entry in the table for a particular scan line contains the maximum y values for that edges, the x intercept value for edge, and the inverse slope of the edge.
- For each scan line the edges are in sorted order from left to right.
- Than we process the scan line from the bottom to top for whole polygon and produce active edge list for each scan line crossing the polygon boundaries.
- The active edge list for a scan line contains all edges crossed by that line, with iterative coherence calculation used to obtain the edge intersections.

Inside-Outside Tests

- In area filling and other graphics operation often required to find particular point is inside or outside the polygon.
- For finding which region is inside or which region is outside most graphics package use either odd even rule or the nonzero winding number rule.

Odd Even Rule

- It is also called the odd parity rule or even odd rule.
- By conceptually drawing a line from any position p to a distant point outside the coordinate extents of the object and counting the number of edges crossing by this line is odd, than p is an interior point. Otherwise p is exterior point.
- To obtain accurate edge count we must sure that line selected is does not pass from any vertices.
- This is shown in figure 2.21(a).

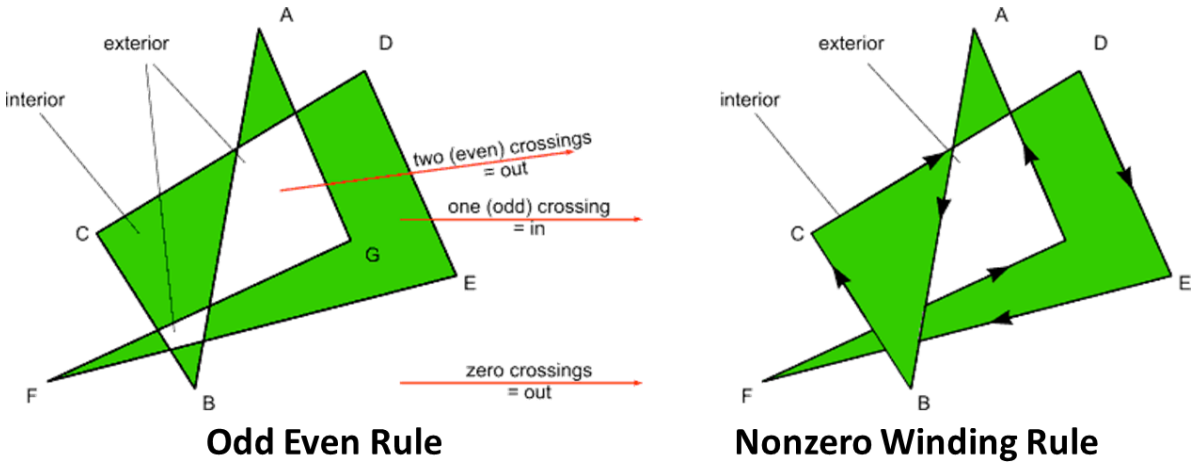


Fig. 2.21: - Identifying interior and exterior region for a self-intersecting polygon.

Nonzero Winding Number Rule

- This method counts the number of times the polygon edges wind around a particular point in the counterclockwise direction. This count is called the winding number, and the interior points of a two-dimensional object are defined to be those that have a nonzero value for the winding number.
- We apply this rule by initializing winding number with 0 and then draw a line for any point p to distant point beyond the coordinate extents of the object.
- The line we choose must not pass through vertices.
- Then we move along that line we find number of intersecting edges and we add 1 to winding number if edge cross our line from right to left and subtract 1 from winding number if edges cross from left to right.
- The final value of winding number is nonzero then the point is interior and if winding number is zero the point is exterior.
- This is shown in figure 2.21(b).
- One way to determine directional edge crossing is to take the vector cross product of a vector U along the line from p to distant point with the edge vector E for each edge that crosses the line.
- If z component of the cross product$U \times E$ for a particular edge is positive that edge is crosses from right to left and we add 1 to winding number otherwise the edge is crossing from left to right and we subtract 1 from winding number.

Comparison between Odd Even Rule and Nonzero Winding Rule

- For standard polygons and simple object both rule gives same result but for more complicated shape both rule gives different result which is illustrated in figure 2.21.

Scan-Line Fill of Curved Boundary Areas

- Scan-line fill of region with curved boundary is more time consuming as intersection calculation now involves nonlinear boundaries.
- For simple curve such as circle or ellipse scan line fill process is straight forward process.
- We calculate the two scan line intersection on opposite side of the curve.
- This is same as generating pixel position along the curve boundary using standard equation of curve.
- Then we fill the color between two boundary intersections.
- Symmetry property is used to reduce the calculation.
- Similar method can be used for fill the curve section.

Boundary Fill Algorithm/ Edge Fill Algorithm

- In this method, edges of the polygons are drawn. Then starting with some seed, any point inside the polygon we examine the neighbouring pixels to check whether the boundary pixel is reached.
- If boundary pixels are not reached, pixels are highlighted and the process is continued until boundary pixels are reached.
- Boundary defined regions may be either 4-cormected or 8-connected. as shown in the Figure below

(a) Four connected region (b) Eight connected region

Fig. 2.22: - Neighbor pixel connected to one pixel.

- If a region is 4-connected, then every pixel in the region may be reached by a combination of moves in only four directions: left, right, up and down.
- For an 8-connected region every pixel in the region may be reached by a combination of moves in the two horizontal, two vertical, and four diagonal directions.
- In some cases, an 8-connected algorithm is more accurate than the 4-connected algorithm. This is illustrated in Figure below. Here, a 4-connected algorithm produces the partial fill.

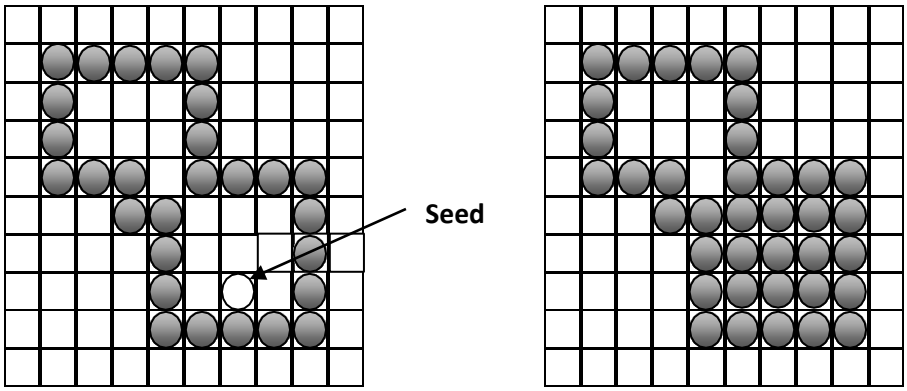


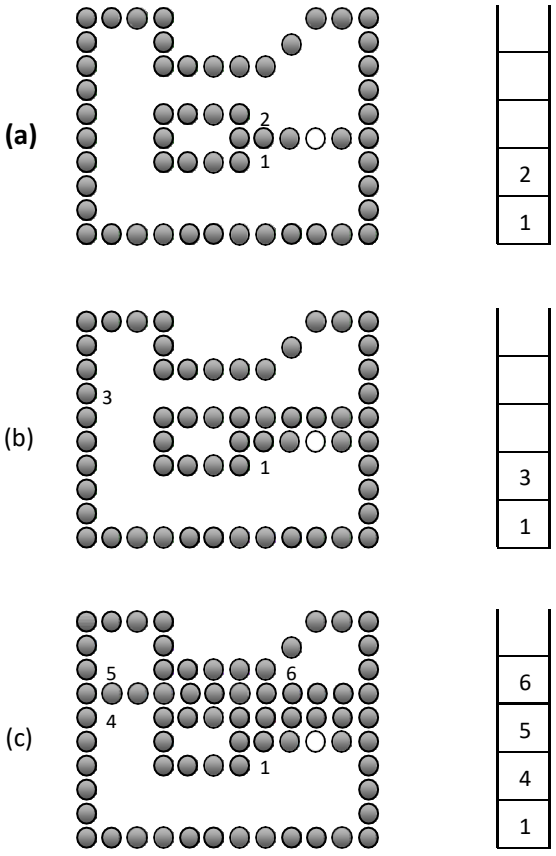
Fig. 2.23: - partial filling resulted due to using 4-connected algorithm.

- The following procedure illustrates the recursive method for filling a 4-connected region with color specified in parameter fill color (f-color) up to a boundary color specified with parameter boundary color (b-color).
- Procedure :

```
boundary-fill4(x, y, f-color, b-color)
{
    if(getpixel (x,y) ! = b-color&&gepixel (x, y) ! = f-color)
    {
            putpixel (x, y, f-color)
            boundary-fill4(x + 1, y, f-color, b-color);
```

```
            boundary-fill4(x, y + 1, f-color, b-color);
            boundary-fill4(x - 1, y, f-color, b-color);
            boundary-fill4(x, y - l, f-color, b-color);
        }
    }
```

- Note: 'getpixel' function gives the color of .specified pixel and 'putpixel' function draws the pixel with specified color.
- Same procedure can be modified according to 8 connected region algorithm by including four additional statements to test diagonal positions, such as $(x + 1, y + 1)$.
- This procedure requires considerable stacking of neighbouring points more, efficient methods are generally employed.
- This method fill horizontal pixel spans across scan lines, instead of proceeding to 4 connected or 8 connected neighbouring points.
- Then we need only stack a beginning position for each horizontal pixel span, instead of stacking all unprocessed neighbouring positions around the current position.
- Starting from the initial interior point with this method, we first fill in the contiguous span of pixels on this starting scan line.
- Then we locate and stack starting positions for spans on the adjacent scan lines, where spans are defined as the contiguous horizontal string of positions bounded by pixels displayed in the area border color.
- At each subsequent step, we unstack the next start position and repeat the process.
- An example of how pixel spans could be filled using this approach is illustrated for the 4-connected fill region in Figure below.



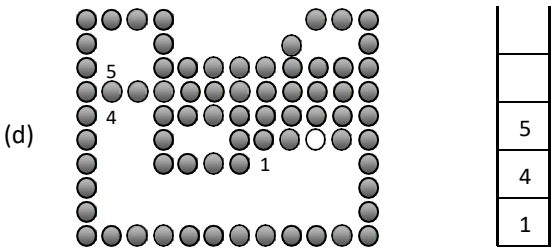


Fig. 2.24: - Boundary fill across pixel spans for a 4-connected area.

Flood-Fill Algorithm

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

```
flood-fill4(x, y, new-color, old-color)
{
    if(getpixel (x,y) = = old-color)
    {
            putpixel (x, y, new-color)
            flood-fill4 (x + 1, y, new-color, old -color);
            flood-fill4 (x, y + 1, new -color, old -color);
            flood-fill4 (x - 1, y, new -color, old -color);
            flood-fill4 (x, y - l, new -color, old-color);
    }
}
```

- Note: 'getpixel' function gives the color of .specified pixel and 'putpixel' function draws the pixel with specified color.

Character Generation

- We can display letters and numbers in variety of size and style.
- The overall design style for the set of character is called typeface.
- Today large numbers of typefaces are available for computer application for example Helvetica, New York platino etc.
- Originally, the term font referred to a set of cast metal character forms in a particular size and format, such as 10-point Courier Italic or 12- point Palatino Bold. Now, the terms font and typeface are often used interchangeably, since printing is no longer done with cast metal forms.
- Two different representations are used for storing computer fonts.

Bitmap Font/ Bitmapped Font

- A simple method for representing the character shapes in a particular typeface is to use rectangular grid patterns.
- Figure below shows pattern for particular letter.

1	1	1	1	1	1	0
0	1	1	0	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	1	0
0	1	1	0	0	1	1
0	1	1	0	0	1	1
1	1	1	1	1	1	0

Fig. 2.25: - Grid pattern for letter B.

- When the pattern in figure 2.25 is copied to the area of frame buffer, the 1 bits designate which pixel positions are to be displayed on the monitor.
- Bitmap fonts are the simplest to define and display as character grid only need to be mapped to a frame-buffer position.
- Bitmap fonts require more space because each variation (size and format) must be stored in a font cache.
- It is possible to generate different size and other variation from one set but this usually does not produce good result.

Outline Font

- In this method character is generated using curve section and straight line as combine assembly.
- Figure below shows how it is generated.

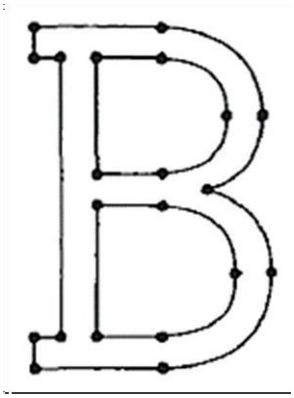


Fig. 2.26: - outline for letter B.

- To display the character shown in figure 2.26 we need to fill interior region of the character.
- This method requires less storage since each variation does not required a distinct font cache.
- We can produce boldface, italic, or different sizes by manipulating the curve definitions for the character outlines.
- But this will take more time to process the outline fonts, because they must be scan converted into the frame buffer.

Stroke Method

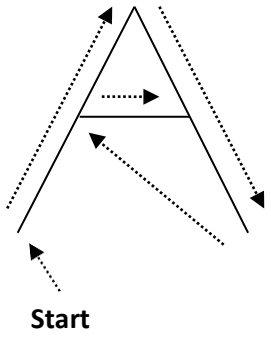


Fig. 2.27: - Stroke Method for Letter A.

- It uses small line segments to generate a character.
- The small series of line segments are drawn like a stroke of a pen to form a character as shown in figure.
- We can generate our own stroke method by calling line drawing algorithm.
- Here it is necessary to decide which line segments are needs for each character and then draw that line to display character.
- It support scaling by changing length of line segment.

Starbust Method

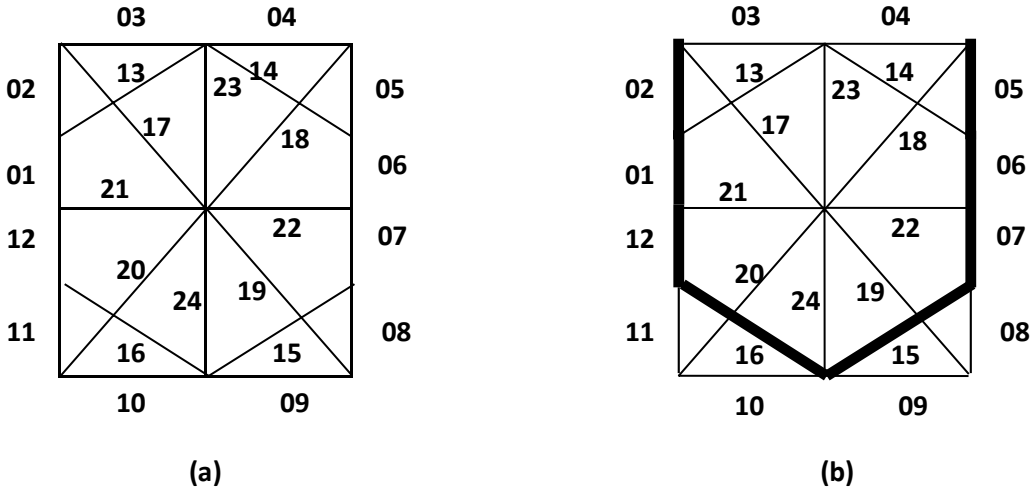


Fig. 2.28: - (a) Starbust Method. (b) Letter V using starbust method

- In this method a fix pattern of line segments are used to generate characters.
- As shown in figure 2.28 there are 24 line segments.
- We highlight those lines which are necessary to draw a particular character.
- Pattern for particular character is stored in the form of 24 bit code. In which each bit represents corresponding line having that number.
- That code contains 0 or 1 based on line segment need to highlight. We put bit value 1 for highlighted line and 0 for other line.
- Code for letter V is
  1 1 0 0 1 1 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0
- This technique is not used now a days because:
  1. It requires more memory to store 24 bit code for single character.

2. It requires conversion from code to character.
3. It doesn't provide curve shapes.

Line Attributes

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

Line Type

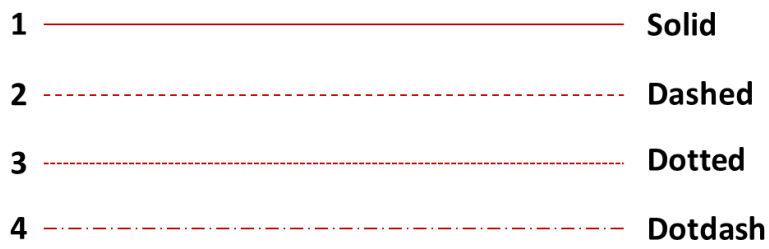


Fig. 2.29: - Line type

- Possible selection for the line-type attribute includes solid lines, dashed lines, and dotted lines etc.
- We modify a line –drawing algorithm to generate such lines by setting the length and spacing of displayed solid sections along the line path.
- A dashed line could be displayed by generating an inter dash spacing that is equal to the length of the solid sections. Both the length of the dashes and the inter dash spacing are often specified as user options.
- 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, 4… etc. to generate lines that are, respectively solid, dashed, dotted, or dash-dotted etc.
- Other values for the line-type parameter It could be used to display variations in the dot-dash patterns. Once the line-type parameter has been set in a PHIGS application program, all subsequent line-drawing commands produce lines with this Line type.
- Raster graphics generates these types by plotting some pixel and some pixel is off along the line path. We can generate different patterns by specifying 1 for on pixel and 0 for off pixel then we can generate 1010101 patter as a dotted line.
- It is used in many application for example comparing data in graphical form.

Line Width

- Implementation of line-width options depends on the capabilities of the output device.
- A heavy line on a video monitor could be displayed as adjacent parallel lines, while a pen plotter might require pen changes.
- To set line width attributes in a PHIGS application program, a user invokes the function:
  setLinewidthScalFactor (lw)
- Line-width parameter lw is assigned a positive number to indicate the relative width of the line to be displayed.
- Values greater than 1 produce lines thicker than the standard line width and values less than the 1 produce line thinner than the standard line width.

- In raster graphics we generate thick line by plotting above and below pixel of line path when slope $|m| < 1$ and by plotting left and right pixel of line path when slope $|m| > 1$ which is illustrated in below figure.

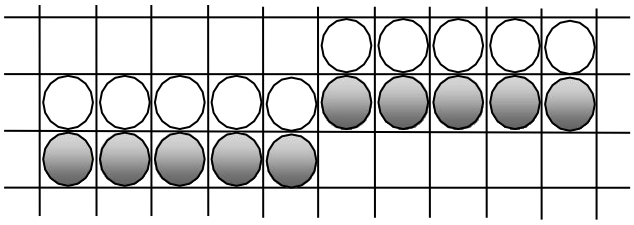


Fig. 2.30: - Double-wide raster line with slope $|m| < 1$ generated with vertical pixel spans.

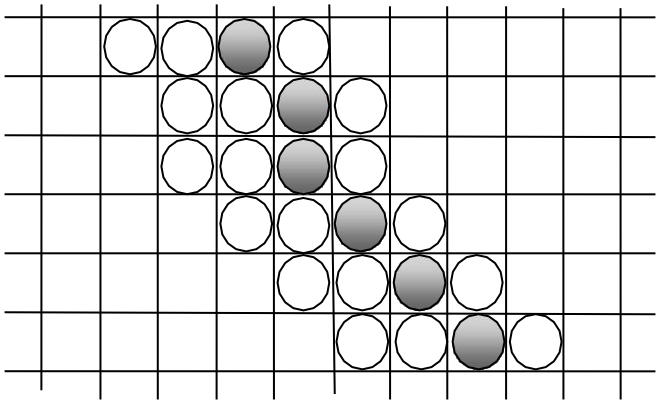


Fig. 2.31: - Raster line with slope $|m| > 1$ and line-width parameter $lw = 4$ plotted with horizontal pixel spans.

- As we change width of the line we can also change line end which are shown below which illustrate all three types of ends.

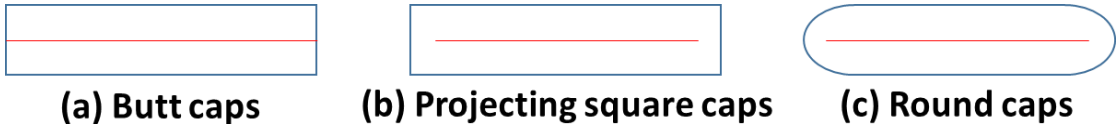


Fig. 2.32: - Thick lines drew with (a) butt caps, (b) projecting square caps, and (c) round caps.

- Similarly we generates join of two lines of modified width are shown in figure below which illustrate all three type of joins.

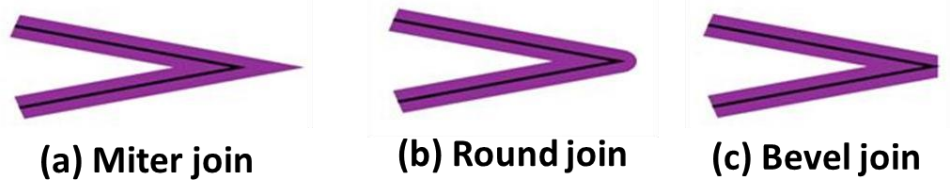


Fig. 2.33: - Thick lines segments connected with (a) miter join, (b) round join, and (c) bevel join.

Line Color

- The name itself suggests that it is defining color of line displayed on the screen.
- By default system produce line with current color but we can change this color by following function in PHIGS package as follows:
  setPolylineColorIndex (lc)
- In this lc is constant specifying particular color to be set.

Pen and Brush Options

- In some graphics packages line is displayed with pen and brush selections.
- Options in this category include shape, size, and pattern.
- Some possible pen or brush are shown in below figure.

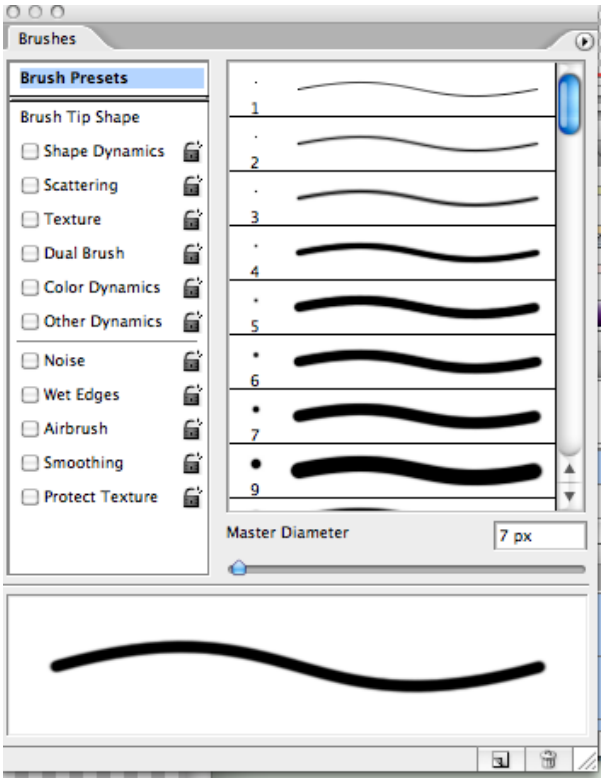


Fig. 2.34: - Pen and brush shapes for line display.

- These shapes can be stored in a pixel mask that identifies the array of pixel positions that are to be set along the line path.
- Lines generated with pen (or brush) shapes can be displayed in various widths by changing the size of the mask.
- Also, lines can be displayed with selected patterns by superimposing the pattern values onto the pen or brush mask.

Color and Greyscale 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.
- General purpose raster-scan systems, for example, usually provide a wide range of colors, while random-scan monitors typically offer only a few color choices, if any.
- In a color raster system, the number of color choices available depends on the amount of storage provided per pixel in the frame buffer
- Also, color-information can be stored in the frame buffer in two ways: We can store color codes directly in the frame buffer, or we can put the color codes in a separate table and use pixel values as an index into this table
- With direct storage scheme we required large memory for frame buffer when we display more color.
- While in case of table it is reduced and we call it color table or color lookup table.

Color Lookup Table

- Color values of 24 bit is stored in lookup table and in frame buffer we store only 8 bit index which gives index of required color stored into lookup table. So that size of frame buffer is reduced and we can display more color.
- When we display picture using this technique on output screen we look into frame buffer where index of the color is stored and take 24 bit color value from look up table corresponding to frame buffer index value and display that color on particular pixel.

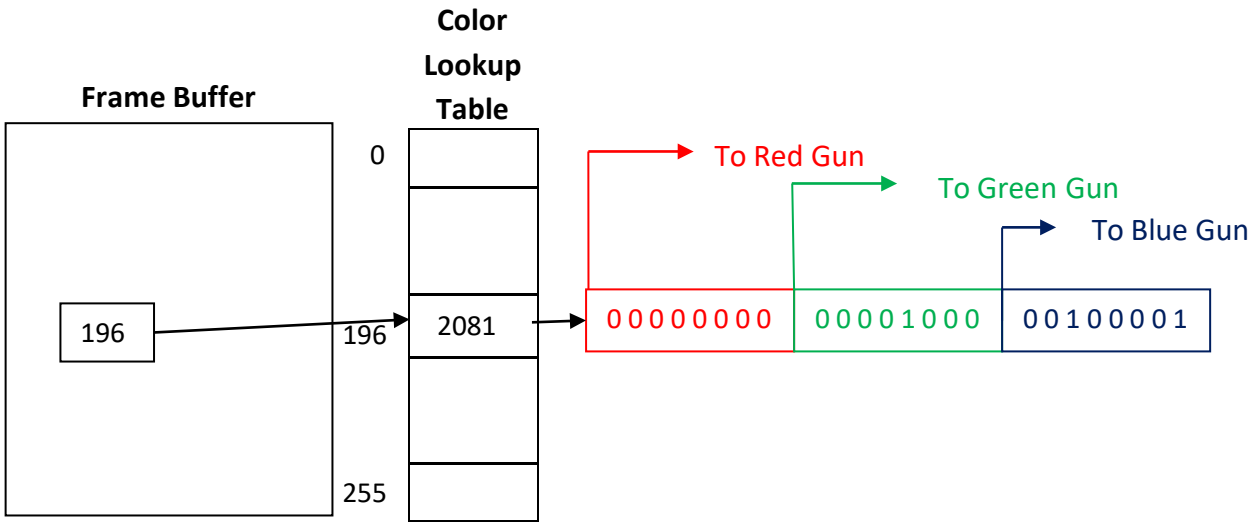


Fig. 2.35: - color lookup table.

Greyscale

- With monitors that have no color capability, color function can be used in an application program to set the shades of grey, or greyscale for display primitives.
- Numeric values between 0-to-1 can be used to specify greyscale levels.
- This numeric values is converted in binary code for store in raster system.
- Table below shows specification for intensity codes for a four level greyscale system.

Intensity Code	Stored Intensity Values In The		Displayed Greyscale
	Frame Buffer	Binary Code	
0.0	0	00	Black
0.33	1	01	Dark grey
0.67	2	10	Light grey
1.0	3	11	White

Table 2.1: - Intensity codes for a four level greyscale system.

- In this example, any intensity input value near 0.33 would be stored as the binary value 01 in the frame buffer, and pixels with this value would be displayed as dark grey.
- If more bits are available per pixel we can obtain more levels of grey scale for example with 3 bit per pixel we can achieve 8 levels of greyscale.

Area-Fill Attributes

- For filling any area we have choice between solid colors or pattern to fill all these are include in area fill attributes.
- Area can be painted by various burses and style.

Fill Styles

- Area are generally displayed with three basic style hollow with color border, filled with solid color, or filled with some design.
- In PHIGS package fill style is selected by following function.
  setInteriorStyle (fs)
- Value of fs include hollow, solid, pattern etc.
- Another values for fill style is hatch, which is patterns of line like parallel line or crossed line.
- Figure bellow shows different style of filling area.

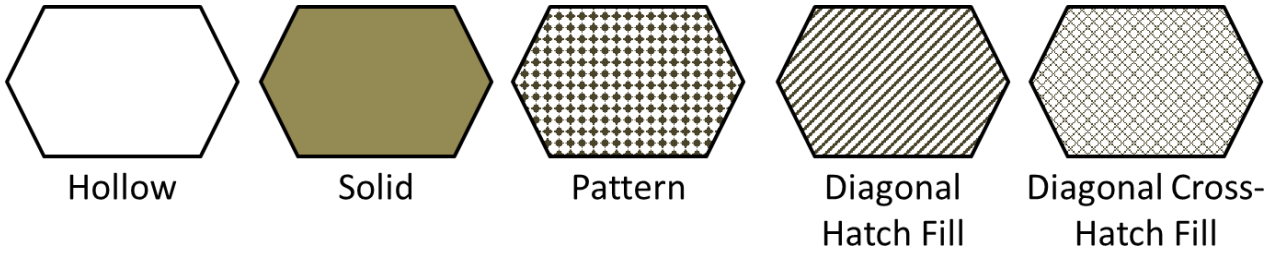


Fig. 2.36: - Different style of area filling.

- For setting interior color in PHIGS package we use:
  setInteriorColorIndex (fc)
- Where fc specify the fill color.

Pattern Fill

- We select the pattern with
  setInteriorStyleIndex (pi)
- Where pattern index parameter pi specifies position in pattern table entry.
- Figure below shows pattern table.

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

Table 2.2: - Pattern table.

- For example, the following set of statements would fill the area defined in the fillArea command with the second pattern type stored in the pattern table:
  SetInteriorStyle(pattern) ;
  setInteriorStyleIndex (2) ;
  fillArea (n, points);
- Separate table can be maintain for hatch pattern and we can generate our own table with required pattern.
- Other function used for setting other style as follows
  setpatternsize (dx, dy)
  setPaternReferencePoint (positicn)
- We can create our own pattern by setting and resetting group of pixel and then map it into the color matrix.

Soft Fill

- Soft fill is modified boundary fill and flood fill algorithm in which we are fill layer of color on back ground color so that we can obtain the combination of both color.
- It is used to recolor or repaint so that we can obtain layer of multiple color and get new color combination.
- One use of this algorithm is soften the fill at boundary so that blurred effect will reduce the aliasing effect.
- For example if we fill t amount of foreground color then pixel color is obtain as:
  $p = tF + (1 - t)B$
- Where F is foreground color and B is background color
- If we see this color in RGB component then:
  $p = (p_r, p_g, p_b)$ $f = (f_r, f_g, f_b)$ $b = (b_r, b_g, b_b)$
- Then we can calculate t as follows:
  $t = \frac{P_k - B_k}{F_k - B_k}$
- If we use more than two color say three at that time equation becomes as follow:
  $p = t_0F + t_1B_1 + (1 - t_0 - t_1)B_2$
- Where the sum of coefficient t_0, t_1, and $(1 - t_0 - t_1)$ is 1.

Character Attributes

- The appearance of displayed characters is controlled by attributes such as font, size, color, and orientation.
- Attributes can be set for entire string or may be individually.

Text Attributes

- In text we are having so many style and design like italic fonts, bold fonts etc.
- For setting the font style in PHIGS package we have one function which is:
  setTextFont (tf)
- Where tf is used to specify text font
- It will set specified font as a current character.
- For setting color of character in PHIGS we have function:
  setTextColorIndex (tc)
- Where text color parameter tc specifies an allowable color code.
- For setting the size of the text we use function.
  setCharacterheight (ch)
- For scaling the character we use function.
  setCharacterExpansionFacter (cw)
- Where character width parameter cw is set to a positive real number that scale the character body width.
- Spacing between character is controlled by function
  setCharacterSpacing (cs)
- Where character spacing parameter cs can be assigned any real value.
- 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.

- Text is then displayed so that the orientation of characters from baseline to cap line is in the direction of the up vector.
- For setting the path of the character we use function:
  setTextPath (tp)
- Where the text path parameter tp can be assigned the value: right, left, up, or down.
- It will set the direction in which we are writing.
- For setting the alignment of the text we use function.
  setTextAlignment (h, v)
- Where parameter h and v control horizontal and vertical alignment respectively.
- For specifying precision for text display is given with function.
  setTextPrecision (tpr)
- Where text precision parameter tpr is assigned one of the values: string, char, or stroke.
- The highest-quality text is produced when the parameter is set to the value stroke.

Marker Attributes

- A marker symbol display single character in different color and in different sizes.
- For marker attributes implementation by procedure that load the chosen character into the raster at defined position with the specified color and size.
- We select marker type using function.
  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). Displayed marker types are centred on the marker coordinates.
- We set the marker size with function.
  SetMarkerSizeScaleFactor (ms)
- Where parameter marker size ms assigned a positive number according to need for scaling.
- For setting marker color we use function.
  setPolymarkerColorIndex (mc)
- Where parameter mc specify the color of the marker symbol.

Aliasing

In signal processing and related disciplines, **aliasing** is an effect that causes different signals to become indistinguishable (or *aliases* of one another) when sampled. It also refers to the distortion or artifact that results when the signal reconstructed from samples is different from the original continuous signal.

Aliasing can occur in signals sampled in time, for instance digital audio, and is referred to as **temporal aliasing**. Aliasing can also occur in spatially sampled signals, for instance moiré patterns in digital images. Aliasing in spatially sampled signals is called **spatial aliasing**.

Aliasing is generally avoided by applying low pass filters or anti-aliasing filters to the input signal before sampling. Suitable reconstruction filters should then be used when restoring the sampled signal to the continuous domain.

Antialiasing methods were developed to combat the effects of aliasing.

There are three main classes of antialiasing algorithms.

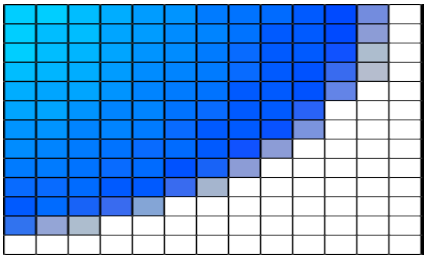
 As aliasing problem is due to low resolution, one easy solution is to increase the resolution , causing sample points to occur more frequently. This increases the cost of image production.

 The image is created at high resolution and then digitally filtered. This method is called supersampling or postfiltering and eliminates high frequencies which are the source of aliases.

 The image can be calculated by considering the intensities over a particular region. This is called prefiltering.

Prefiltering.

Prefiltering methods treat a pixel as an area, and compute pixel color based on the overlap of the scene's objects with a pixel's area. These techniques compute the shades of gray based on how much of a pixel's area is covered by a object.
For example, a modification to Bresenham's algorithm was developed by Pitteway and Watkinson. In this algorithm, each pixel is given an intensity depending on the area of overlap of the pixel and the line. So, due to the blurring effect along the line edges, the effect of antialiasing is not very prominent, although it still exists.
Prefiltering thus amounts to sampling the shape of the object very densely within a pixel region. For shapes other than polygons, this can be very computationally intensive.



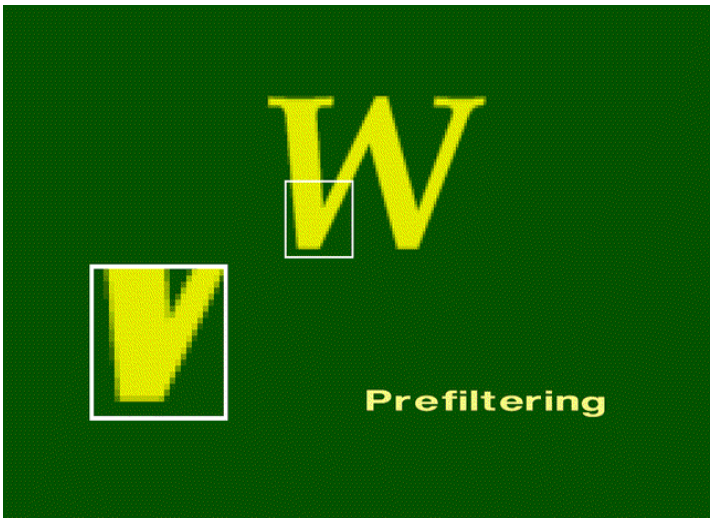
Original Image

Without antialiasing, the jaggies are harshly evident.

Prefiltered image

.

Along the character's border, the colors are a mixture of the foreground and background colors.



Postfiltering.

Supersampling or ***postfiltering*** is the process by which aliasing effects in graphics are reduced by increasing the frequency of the sampling grid and then averaging the results down. This process means calculating a virtual image at a higher spatial resolution than the frame store resolution and then averaging down to the final resolution. It is called postfiltering as the filtering is carried out after sampling.

There are two ***drawbacks*** to this method

- The drawback is that there is a technical and economic limit for increasing the resolution of the virtual image.

- Since the frequency of images can extend to infinity, it just reduces aliasing by raising the Nyquist limit - shift the effect of the frequency spectrum.

Supersampling is basically a three stage process.

- A continuous image I(x,y) is sampled at n times the final resolution. The image is calculated at n times the frame resolution. This is a virtual image.

- The virtual image is then lowpass filtered

- The filtered image is then resampled at the final frame resolution.

Algorithm for supersampling

- To generate the origial image, we need to consider a region in the virtual image. The extent of that region determines the regions involved in the lowpass operation. This process is called ***convolution***.

- After we obtain the virtual image which is at a higher resolution, the pixels of the final image are located over superpixels in the virtual image. To calculate the value of the final image at (Si,Sj), we place the filter over

the superimage and compute the sum of the filter weights and the surrounding pixels. An adjacent pixel of the final image is calculated by moving the filter S superpixels to the right. Thus the step size is same as the scale factor between the real and the virtual image.

 Filters combine samples to compute a pixel's color. The weighted filter shown on the slide combines nine samples taken from inside a pixel's boundary. Each sample is multiplied by its corresponding weight and the products are summed to produce a weighted average, which is used as the pixel color. In this filter, the center sample has the most influence. The other type of filter is an unweighted filter. In an unweighted filter, each sample has equal influence in determining the pixel's color. In other words, an unweighted filter computes an unweighted average.

 The spatial extent of the filter determines the cutoff frequency. The wider the filter, the lower is the cutoff frequency and the more blurred is the image.

The options available in supersampling are

 The value of S - scaling factor between the virtual and the real images.

 The choice of the extents and the weights of the filter

As far the first factor is concerned, higher the value, the better the result is going to be. The compromise to be made is the high storage cost.

Disadvantages

 It is not a context sensitive technique and thereby results in a lot of wasteful computations.