UNIT I  INTRODUCTION

Survey of computer graphics, Overview of graphics systems – Video display devices, Raster scan systems, Random scan systems, Graphics monitors and Workstations, Input devices, Hard copy Devices, Graphics Software; Output primitives – points and lines, line drawing algorithms, loading the frame buffer, line function; circle and ellipse generating algorithms; Pixel addressing and object geometry, filled area primitives.

1.1 Survey of Computer graphics

- Computer graphics is the pictorial representation of information using a computer program.

- Graphics are visual presentations on some surface such as wall, canvas, computer screen, paper, etc.,

- In computer graphics, pictures as graphical subjects are represented as a collection of discrete picture element called pixels. Pixel is smallest addressable screen element.

Computer Graphics

- The computer is an information processing machine. It is a tool for storing, manipulating and correlating data. There are many ways to communicate the processed information to the user.

- The computer graphics is one of the most effective and commonly used way to communicate the processed information to the user.

- The computer graphics is the use of computers to display and manipulate information in graphical or pictorial form, either on a visual-display unit or via a printer or plotter.

Classification of Computer Graphics

Based on type of object (Dimensionality)

- 2D Graphics
- 3D Graphics

Based on kind of picture

- Symbolic Graphics
- Realistic Graphics

Based on type of interaction

- Controllable
- Non-Controllable
Based on role of picture

- Use for representation
- Use as an end product such as drawing

Based on type of pictorial representation

- Line Drawing
- Black and White Image
- Colour Image

Application/Strength of Computer Graphics

- User Interfaces, Office Automation, Simulation and Animation
- Presentation graphics software
- Animation software
- CAD software, Cartography
- Desktop Publishing
- Weather Maps, Satellite Imaging, Photo Enhancement
- Medical Imaging
- Engineering drawings
- Typography, Architecture, Art

2D Computer Graphics

- A Picture that has height and weight but no depth is two-dimensional (2-D).
- 2D Computer graphics are the computer-based generation of digital images mostly from two-dimensional models such as 2D geometric models, text, and digital images and by techniques specific to them.
- 2D Graphics are mainly used in applications that were originally developed upon traditional printing and drawing technologies such as typography, cartography, technical drawing, advertising, etc.,

3D Computer Graphics

- A picture that has height, weight and depth is three-dimensional (3D).
- 3D Graphics compared to 2D Graphics are graphics that use a three-dimensional representation of geometric data.
- 3D computer graphics are essential to realistic computer games, simulators and objects.

Computer Animation

- Computer Animation is the art of creating moving images via the use of computers. It is a subfield of computer graphics and animation.
- It is also referred to as CGI (Computer-Generated Imagery or Computer-Generated Imaging), especially when used in films.

- To create the illusion of movement, an image is displayed on the computer screen then quickly replaced by a new image that is similar to the previous image but shifted slightly. This technique is identical to the illusion of movement in television and motion pictures.

**Digital Image**

A Digital image is a representation of a two-dimensional image using ones and zeros (binary). Depending on whether or not the image resolution is fixed, it may be of vector or raster type.

**Pixel**

- In computer graphics, pictures or graphics objects are presented as a collection of discrete picture elements called **pixels**.
- The pixel is the smallest addressable screen element. It is the smallest piece of the display screen which we can control.
- The control is achieved by setting the intensity and colour of the pixel which compose the screen.

![Pixel display area of 4 x 3](image)

**Overview of Graphics Systems**

- It consists of input and output devices, graphics systems, application program and application model.

- A computer receives input from input devices and output images to a display device.

- The input and output devices are called the hardware components of the conceptual framework.

- There are three software components of conceptual framework. These are,
  - **Application model** – The application model captures all the data and objects to be pictured on the screen. It also captures the relationship
among the data and objects these relationship are stored in the database called application database and referred by the application programs.

- **Application Program** – It creates the application model and communicates with it to receive and store the data and information of objects’ attributes. The application program also handles user input.

- **Graphics System** – It is an intermediately between the application program and the display hardware. It accepts the series of graphics output commands from application program.

  The output commands contain both a detailed geometric description of what is to be viewed and the attributes describing how the objects should appear.

### 1.2 Video Display Devices:

- Basically Display Devices are the Devices or else known as output devices for showing the information in visual form. So Video Display Devices are nothing but Display or output devices which present Videos in various forms.

- Most of the display devices are based on the standard CRT design. There are several Video Display Devices in Computer Graphics. Important Video display Devices are provided below. They are,

  i) **Cathode Ray Tube (CRT):**

     We often use Computer Monitors and Televisions in our regular life. So CRT technology is employed in these output devices. These CRT Displays are used mostly by Graphic professionals. Because CRT Technology offers good vibrant and very good accurate colour.

  ii) **Monitor or Display:**

     Monitor is also referred as Visual Display Unit. Monitor is an electronic output device where we can see entire output through this device.

  iii) **Random Scan Display:**

     Random Scan Display is formed by using various geometrical primitives. So they are formed using Curves, Points, Lines, and Polygons

  iv) **Raster Scan Display:**

     In the name Raster Scan Display, the Raster word means a rectangular array of points. or else array of dots. So Group of dots are known as Pixels, and image is sub grouped in to a sequence of stips. These are called Scan lines, and these scan lines are further divided in to Pixels.
v) Flat Panel Display:

This kind of Flat Panel Display surrounds number of various electronics visual display techniques or methods. So that it obviously become lighter and very thinner about 100mm. These kind of Flat Panel Displays are used in many portable devices. They are,

Ex: Digital Cameras, Phone, Laptops, Compact Cameras and Camcorders like wise

1.2.1 Refresh Cathode Ray Tube: (Refresh CRT)

- A beam of Cathode rays discharged by an Electron gun moves through some deflection systems.
- So these deflection systems which direct the beam towards a described specified position on the Screen.
- This Screen is Phosphor coated, and this Phosphor discharges a mark of light at each position where it is contacted by the Electron beam.
- To maintain the screen picture we need some process, as the light discharged by the phosphor fades with very high speed.
- That method we require is “To keep up the phosphor glowing is to draw up again (redraw) the picture repeatedly by quickly directing this electron beam back on to the same points which were resulted in first time path. This kind of Display is called “Refresh CRT”

![Diagram of a magnetic-deflection CRT](image)

**Working:**

Electron Gun Components:

- i) Heated Metal Cathode
- ii) Control Grid
- Cathode gets heated by directing “Current” using wire coil, which is also called Filament
- So obviously this causes electrons to be boiled off the very hot cathode surface. In the Cathode ray tube vacuum, already negative charged electrons are then accelerated towards the Coating by positive voltage.

- Accelerating voltage will be generated with a positively charged metal which is coated on the inside CRT envelope near the Phosphor screen.

- On the Phosphor screen, various marks or spots of light are produced on the screen by the transfer of CRT beam energy to screen.

- So soon after electrons in the beam collide with the phosphor coating, they are stopped. Now kinetic energy (K.E) is absorbed by the Phosphor. And some part of the beam energy is converted to heat energy because of Friction takes place. And the remaining energy causes electrons in the phosphor atoms to move up to the various higher of its quantum energy levels.

- After some time, the excited phosphor electrons come back to their previous stable ground state.

- Make note of this, the Frequency of the light emitted by the Phosphor is directly proportional to the energy difference between “excited quantum state” and “the Ground previous state”.

### 1.2.2 Raster Scan Display

- In a raster scan system, the electron beam is swept across the screen, one row at a time from top to bottom. As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.

- Picture definition is stored in memory area called the Refresh Buffer or Frame Buffer. This memory area holds the set of intensity values for all the screen points. Stored intensity values are then retrieved from the refresh buffer and “painted” on the screen one row (scan line) at a time.

- Each screen point is referred to as a pixel (picture element) or pel. At the end of each scan line, the electron beam returns to the left side of the screen to begin displaying the next scan line.
1.2.3 Random Scan Display

- In this technique, the electron beam is directed only to the part of the screen where the picture is to be drawn rather than scanning from left to right and top to bottom as in raster scan. It is also called vector display, stroke-writing display, or calligraphic display.

- Picture definition is stored as a set of line-drawing commands in an area of memory referred to as the refresh display file. To display a specified picture, the system cycles through the set of commands in the display file, drawing each component line in turn.

- After all the line-drawing commands are processed, the system cycles back to the first line command in the list.

- Random-scan displays are designed to draw all the component lines of a picture 30 to 60 times each second.

1.3 Output primitives

- A picture is completely specified by the set of intensities for the pixel positions in the display.

- Shapes and colors of the objects can be described internally with pixel arrays into the frame buffer or with the set of the basic geometric – structure such as straight line segments and polygon color areas. To describe structure of basic object is referred to as output primitives.

- Each output primitive is specified with input co-ordinate data and other information about the way that objects is to be displayed. Additional output primitives that can be used to constant a picture include circles and other conic sections, quadric surfaces, Spline curves and surfaces, polygon floor areas and character string.
Points and Lines

- **Point plotting** is accomplished by converting a single coordinate position furnished by an application program into appropriate operations for the output device. With a CRT monitor, for example, the electron beam is turned on to illuminate the screen phosphor at the selected location.

- **Line drawing** is accomplished by calculating intermediate positions along the line path between two specified end points positions. An output device is then directed to fill in these positions between the end points.

- Digital devices display a straight line segment by plotting discrete points between the two end points. Discrete coordinate positions along the line path are calculated from the equation of the line. For a raster video display, the line color (intensity) is then loaded into the frame buffer at the corresponding pixel coordinates. Reading from the frame buffer, the video controller then plots “the screen pixels”.

- Pixel positions are referenced according to scan-line number and column number (pixel position across a scan line). Scan lines are numbered consecutively from 0, starting at the bottom of the screen; and pixel columns are numbered from 0, left to right across each scan line.

To load an intensity value into the frame buffer at a position corresponding to column x along scan line y,

```
setpixel(x, y)
```

To retrieve the current frame buffer intensity setting for a specified location we use a low level function

```
getpixel(x, y)
```

### 1.4 Line Drawing Algorithms

- Digital Differential Analyzer (DDA) Algorithm
- Bresenham’s Line Algorithm
- Parallel Line Algorithm

The Cartesian *slope-intercept equation* for a straight line is

\[ y = m \cdot x + b \]  \hspace{1cm} (1)

Where \( m \) as slope of the line

\( b \) as the y intercept

Given that the two endpoints of a line segment are specified at positions \((x_1,y_1)\) and \((x_2,y_2)\) as in figure we can determine the values for the slope \( m \) and y intercept \( b \) with the following calculations.
Line Path between endpoint positions \((x_1, y_1)\) and \((x_2, y_2)\)

\[
m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{(2)}
\]

\[
b = y_1 - m \cdot x_1 \quad \text{(3)}
\]

For any given \(\Delta x\) interval along a line, we can compute the corresponding \(\Delta y\) interval as

\[
\Delta y = m \Delta x \quad \text{(4)}
\]

We can obtain the \(\Delta x\) interval corresponding to a specified \(\Delta y\) as

\[
\Delta x = \frac{\Delta y}{m} \quad \text{(5)}
\]

- For lines with slope magnitudes \(|m| < 1\), \(\Delta x\) can be set proportional to a small horizontal deflection voltage and the corresponding vertical deflection is then set proportional to \(\Delta y\) as calculated from Eq (4).

- For lines whose slopes have magnitudes \(|m| > 1\), \(\Delta y\) can be set proportional to a small vertical deflection voltage with the corresponding horizontal deflection voltage set proportional to \(\Delta x\), calculated from Eq (5).

- For lines with \(m = 1\), \(\Delta x = \Delta y\) and the horizontal and vertical deflections voltage are equal.
1.4.1 Digital Differential Analyzer (DDA) Algorithm

- The digital differential analyzer (DDA) is a scan-conversion line algorithm based on calculation either ∆y or ∆x.
- The line at unit intervals in one coordinate and determine corresponding integer values nearest the line path for the other coordinate.
- A line with positive slope, if the slope is less than or equal to 1, at unit x intervals (∆x=1) and compute each successive y values as
  \[ y_{k+1} = y_k + m \] (6)
  - Subscript k takes integer values starting from 1 for the first point and increases by 1 until the final endpoint is reached. \( m \) can be any real number between 0 and 1 and, the calculated y values must be rounded to the nearest integer.
- For lines with a positive slope greater than 1 we reverse the roles of x and y, (∆y=1) and calculate each succeeding x value as
  \[ x_{k+1} = x_k + \frac{1}{m} \] (7)
  - Equation (6) and (7) are based on the assumption that lines are to be processed from the left endpoint to the right endpoint.
- If this processing is reversed, ∆x=-1 that the starting endpoint is at the right
  \[ y_{k+1} = y_k - m \] (8)
  When the slope is greater than 1 and ∆y = -1 with
  \[ x_{k+1} = x_k - \frac{1}{m} \] (9)
  - If the absolute value of the slope is less than 1 and the start endpoint is at the left, we set ∆x = 1 and calculate y values with Eq. (6)
  - When the start endpoint is at the right (for the same slope), we set ∆x = -1 and obtain y positions from Eq. (8). Similarly, when the absolute value of a negative slope is greater than 1, we use ∆y = -1 and Eq. (9) or we use ∆y = 1 and Eq. (7).

Algorithm:

```c
#define ROUND(a) ((int)(a+0.5))
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));
}

<table>
<thead>
<tr>
<th>K</th>
<th>x Increment</th>
<th>Y Increment</th>
<th>Plotting points (Rounded to Integer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0+0.66=0.66</td>
<td>0+1=1</td>
<td>(1,1)</td>
</tr>
<tr>
<td>1</td>
<td>0.66+0.66=1.32</td>
<td>1+1=2</td>
<td>(1,2)</td>
</tr>
<tr>
<td>2</td>
<td>1.32+0.66=1.98</td>
<td>2+1=3</td>
<td>(2,3)</td>
</tr>
<tr>
<td>3</td>
<td>1.98+0.66=2.64</td>
<td>3+1=4</td>
<td>(3,4)</td>
</tr>
<tr>
<td>4</td>
<td>2.64+0.66=3.3</td>
<td>4+1=5</td>
<td>(3,5)</td>
</tr>
<tr>
<td>5</td>
<td>3.3+0.66=3.96</td>
<td>5+1=6</td>
<td>(4,6)</td>
</tr>
</tbody>
</table>

**Algorithm Description:**

Step 1: Accept Input as two endpoint pixel positions

Step 2: Horizontal and vertical differences between the endpoint positions are assigned to parameters dx and dy (Calculate dx=xb-xa and dy=yb-ya).

Step 3: The difference with the greater magnitude determines the value of parameter steps.

Step 4: Starting with pixel position (xa, ya), determine the offset needed at each step to generate the next pixel position along the line path.

Step 5: loop the following process for steps number of times

a. Use a unit of increment or decrement in the x and y direction

b. if xa is less than xb the values of increment in the x and y directions are 1 and m

c. if xa is greater than xb then the decrements -1 and – m are used.

**Example: Consider the line from (0,0) to (4,6)**

1. xa=0, ya =0 and xb=4 yb=6
2. dx=xb-xa = 4-0 = 4 and dy=yb-ya=6-0= 6
3. x=0 and y=0
4. 4 > 6 (false) so, steps=6
5. Calculate xnIncrement = dx/steps = 4 / 6 = 0.66 and yIncrement = dy/steps=6/6=1
6. Setpixel(x,y) = Setpixel(0,0) (Starting Pixel Position)
7. Iterate the calculation for xnIncrement and yIncrement for steps(6) number of times
8. Tabulation of the each iteration
Result:

Advantages of DDA Algorithm
1. It is the simplest algorithm
2. It is a faster method for calculating pixel positions

Disadvantages of DDA Algorithm
1. Floating point arithmetic in DDA algorithm is still time-consuming
2. End point accuracy is poor

1.4.2 Bresenham’s Line Algorithm

- An accurate and efficient raster line generating algorithm developed by Bresenham, that uses only incremental integer calculations.
- In addition, Bresenham’s line algorithm can be adapted to display circles and other curves.
- 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 the left endpoint \((x_0,y_0)\) of a given line, we step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path.
- To determine the pixel \((x_k,y_k)\) is to be displayed, next to decide which pixel to plot the column \(x_{k+1}=x_k+1,(x_{k+1},y_{k+1})\) and \(x_{k+1},y_{k+1}\). At sampling position \(x_{k+1}\), we label vertical pixel separations from the mathematical line path as \(d_1\) and \(d_2\). The y coordinate on the mathematical line at pixel column position \(x_{k+1}\) is calculated as

\[
y = m(x_{k+1}) + b \quad \text{(1)}
\]

Then

\[
d_1 = y - y_k = m(x_{k+1}) + b - y_k
\]
\[
d_2 = (y_{k+1}) - y = y_{k+1} - m(x_{k+1}) - b
\]
To determine which of the two pixel is closest to the line path, efficient test that is based on the difference between the two pixel separations

\[ d_1 - d_2 = 2m(x_{k+1}) - 2y_k + 2b - 1 \]  

(2)

A decision parameter \( P_k \) for the \( k^{th} \) step in the line algorithm can be obtained by rearranging equation (2). By substituting \( m = \Delta y / \Delta x \) where \( \Delta x \) and \( \Delta y \) are the vertical and horizontal separations of the endpoint positions and defining the decision parameter as

\[ p_k = \Delta x (d_1 - d_2) = 2\Delta y x_k - 2\Delta x y_k + c \]  

(3)

The sign of \( p_k \) is the same as the sign of \( d_1 - d_2 \), since \( \Delta x > 0 \). Parameter \( C \) is constant and has the value \( 2\Delta y + \Delta x(2b - 1) \) which is independent of the pixel position and will be eliminated in the recursive calculations for \( P_k \).

If the pixel at \( y_k \) is “closer” to the line path than the pixel at \( y_{k+1} \) (\( d_1 < d_2 \)) than decision parameter \( P_k \) is negative. In this case, plot the lower pixel, otherwise plot the upper pixel.

Coordinate changes along the line occur in unit steps in either the x or y directions.

To obtain the values of successive decision parameters using incremental integer calculations. At steps \( k+1 \), the decision parameter is evaluated from equation (3) as

\[ P_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + c \]

Subtracting the equation (3) from the preceding equation

\[ P_{k+1} - P_k = 2\Delta y (x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k) \]

But \( x_{k+1} = x_k + 1 \) so that

\[ P_{k+1} = P_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k) \]  

(4)

Where the term \( y_{k+1} - y_k \) is either 0 or 1 depending on the sign of parameter \( P_k \)

This recursive calculation of decision parameter is performed at each integer x position, starting at the left coordinate endpoint of the line.

The first parameter \( P_0 \) is evaluated from equation at the starting pixel position \( (x_0, y_0) \) and with \( m \) evaluated as \( \Delta y / \Delta x \)

\[ P_0 = 2\Delta y - \Delta x \]  

(5)

Bresenham’s line drawing for a line with a positive slope less than 1 in the following outline of the algorithm.

The constants \( 2\Delta y \) and \( 2\Delta y - 2\Delta x \) are calculated once for each line to be scan converted.
Bresenham’s line Drawing Algorithm for \(|m| < 1\)

1. Input the two line endpoints and store the left end point in \((x_0, y_0)\)
2. Load \((x_0, y_0)\) into frame buffer, i.e. Plot the first point.
3. Calculate the constants \(\Delta x\), \(\Delta y\), \(2\Delta y\) 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
   i. If \(P_k < 0\), the next point to plot is \((x_{k+1}, y_k)\) and
      1. \(P_{k+1} = P_k + 2\Delta y\)
   ii. Otherwise, the next point to plot is \((x_{k+1}, y_{k+1})\) and
      1. \(P_{k+1} = P_k + 2\Delta y - 2\Delta x\)
5. Perform step 4 \(\Delta x\) times.

Implementation of Bresenham Line drawing Algorithm

```c
void lineBres (int xa, int ya, int xb, int yb)
{
    int dx = abs(xa - xb), dy = abs(ya - yb);
    int p = 2 * dy - dx;
    int twoDy = 2 * dy, twoDyDx = 2 * (dy - dx);
    int x, y, xEnd;
    /* Determine which point to use as start, which as end */
    if (xa > xb)
    {
        x = xb;
        y = yb;
        xEnd = xa;
    }
    else
    {
        x = xa;
        y = ya;
        xEnd = xb;
    }
}
```
setPixel(x,y);
while(x<xEnd)
{
    x++;
    if (p<0)
        p+=twoDy;
    else
    {
        y++;
        p+=twoDyDx;
    }
    setPixel(x,y);
}

Example: Consider the line with endpoints (20,10) to (30,18)

The line has the slope \( m = \frac{18-10}{30-20} = \frac{8}{10} = 0.8 \)

\[ \Delta x = 10 \quad \Delta y = 8 \]

The initial decision parameter has the value

\[ p_0 = 2\Delta y - \Delta x = 6 \]

and the increments for calculating successive decision parameters are

\[ 2\Delta y = 16 \quad 2\Delta y - 2\Delta x = -4 \]

We plot the initial point \((x_0, y_0) = (20,10)\) and determine successive pixel positions along the line path from the decision parameter as

**Tabulation**

<table>
<thead>
<tr>
<th>k</th>
<th>( p_k )</th>
<th>((x_{k+1}, y_{k+1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>(21,11)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>(22,12)</td>
</tr>
<tr>
<td>2</td>
<td>-2</td>
<td>(23,12)</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>(24,13)</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>(25,14)</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>(26,15)</td>
</tr>
</tbody>
</table>
Advantages

- Algorithm is Fast
- Uses only integer calculations

Disadvantages

- It is meant only for basic line drawing.

1.4.3 Line Function

- The two dimension line function is Polyline(n,wcPoints) where n is assigned an integer value equal to the number of coordinate positions to be input and wcPoints is the array of input world-coordinate values for line segment endpoints.
- polyline function is used to define a set of n – 1 connected straight line segments
- To display a single straight-line segment we have to set n=2 and list the and y values of the two endpoint coordinates in wcPoints.

Example: following statements generate 2 connected line segments with endpoints at (50, 100), (150, 250), and (250, 100)

```c
typedef struct myPt{int x, y;};
myPt wcPoints[3];
wcPoints[0].x = 50; wcPoints[0].y = 100;
```
wcPoints[1].x = 150; wcPoints[1].y = 50;
wcPoints[2].x = 250; wcPoints[2].y = 100;
polyline (3, wcpoints);

1.5 Circle-Generating Algorithms

General function is available in a graphics library for displaying various kinds of curves, including circles and ellipses.

Properties of a circle

A circle is defined as a set of points that are all the given distance \((x_c,y_c)\).

This distance relationship is expressed by the pythagorean theorem in Cartesian coordinates as

\[(x - x_c)^2 + (y - y_c)^2 = r^2\]  \hspace{1cm} (1)

Use above equation to calculate the position of points on a circle circumference by stepping along the x axis in unit steps from \(x_c-r\) to \(x_c+r\) and calculating the corresponding y values at each position as

\[y = y_c + \left( r^2 - (x_c - x)^2 \right)^{1/2} \] \hspace{1cm} (2)

This is not the best method for generating a circle for the following reason

- Considerable amount of computation
- Spacing between plotted pixels is not uniform

To eliminate the unequal spacing is to calculate points along the circle boundary using polar coordinates \(r\) and \(\theta\). Expressing the circle equation in parametric polar form yields the pair of equations

\[x = x_c + r \cos \theta \quad y = y_c + r \sin \theta \]

When a display is generated with these equations using a fixed angular step size, a circle is plotted with equally spaced points along the circumference. To reduce calculations use a large angular separation between points along the circumference and connect the points with straight line segments to approximate the circular path.
• Set the angular step size at $1/r$. This plots pixel positions that are approximately one unit apart. The shape of the circle is similar in each quadrant.

• To determine the curve positions in the first quadrant, to generate the circle section in the second quadrant of the $xy$ plane by noting that the two circle sections are symmetric with respect to the $y$ axis and circle section in the third and fourth quadrants can be obtained from sections in the first and second quadrants by considering symmetry between octants.

• Circle sections in adjacent octants within one quadrant are symmetric with respect to the $45^0$ line dividing the two octants. Where a point at position $(x, y)$ on a one-eighth circle sector is mapped into the seven circle points in the other octants of the $xy$ plane.

• To generate all pixel positions around a circle by calculating only the points within the sector from $x=0$ to $y=0$, the slope of the curve in this octant has an magnitude less than or equal to 1.0. at $x=0$, the circle slope is 0 and at $x=y$, the slope is -1.0.

• Bresenham’s line algorithm for raster displays is adapted to circle generation by setting up decision parameters for finding the closest pixel to the circumference at each sampling step. Square root evaluations would be required to computer pixel sitances from a circular path.

• Bresenham’s circle algorithm avoids these square root calculations by comparing the squares of the pixel separation distances. It is possible to perform a direct distance comparison without a squaring operation.

• In this approach is to test the halfway position between two pixels to determine if this midpoint is inside or outside the circle boundary. This method is more easily applied to other conics and for an integer circle radius the midpoint approach generates the same pixel positions as the Bresenham circle algorithm.
For a straight line segment the midpoint method is equivalent to the
bresenham line algorithm. The error involved in locating pixel positions along
any conic section using the midpoint test is limited to one half the pixel
separations.

1.5.1 Midpoint circle Algorithm:

- In the raster line algorithm at unit intervals and determine the closest pixel
  position to the specified circle path at each step for a given radius r and screen
center position \((x_c, y_c)\) set up our algorithm to calculate pixel positions around a
circle path centered at the coordinate position by adding \(x_c\) to \(x\) and \(y_c\) to \(y\).

To apply the midpoint method we define a circle function as

\[
f_{\text{circle}}(x, y) = x^2 + y^2 - r^2
\]

- Any point \((x, y)\) on the boundary of the circle with radius \(r\) satisfies the equation
  \(f_{\text{circle}}(x, y) = 0\). If the point is in the interior of the circle, the circle function is
  negative. And if the point is outside the circle the, circle function is positive

  \[
f_{\text{circle}}(x, y) < 0, \text{ if } (x, y) \text{ is inside the circle boundary}
  = 0, \text{ if } (x, y) \text{ is on the circle boundary}
  > 0, \text{ if } (x, y) \text{ is outside the circle boundary}
\]

- The tests in the above eqn are performed for the midposition sbtewen pixels near
  the circle path at each sampling step. The circle function is the decision parameter
  in the midpoint algorithm.

- Midpoint between candidate pixels at sampling position \(x_{k+1}\) along a circular path.
  Fig -1 shows the midpoint between the two candidate pixels at sampling position
  \(x_{k+1}\). To plot the pixel at \((x_k, y_k)\) next 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 circular to the circle.

  Our decision parameter is the circle function evaluated at the midpoint between
  these two pixels

  \[
P_k = f_{\text{circle}}(x_{k+1}, y_{k-1}/2) = (x_{k+1})^2 + (y_{k-1}/2)^2 - r^2
\]

- If \(P_k < 0\), this midpoint is inside the circle and the pixel on scan line \(y_k\) is closer to
  the circle boundary. Otherwise the mid position is outside or on the circle
  boundary and select the pixel on scan line \(y_{k-1}\).

- Successive decision parameters are obtained using incremental calculations. To
  obtain a recursive expression for the next decision parameter by evaluating the
  circle function at sampling position \(x_{k+1+1} = x_{k+2}\)

  \[
P_k = f_{\text{circle}}(x_{k+1+1}, y_{k+1-1}/2) = [(x_{k+1})+1]^2 + (y_{k+1-1}/2)^2 - r^2
\]
or

\[ P_{k+1} = P_k + 2(x_{k+1}) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1 \]

Where \( y_{k+1} \) is either \( y_k \) or \( y_{k-1} \) depending on the sign of \( P_k \).

Increments for obtaining \( P_{k+1} \) are either \( 2x_{k+1} + 1 \) (if \( P_k \) is negative) or

\[ 2x_{k+1} + 1 - 2y_{k+1}. \]

Evaluation of the terms \( 2x_{k+1} \) and \( 2y_{k+1} \) can also be done incrementally as

\[ 2x_{k+1} = 2x_{k+2} \]

\[ 2y_{k+1} = 2y_{k-2} \]

- At the Start position \((0,r)\) these two terms have the values 0 and \( 2r \) respectively. Each successive value for the \( 2x_{k+1} \) term is obtained by adding 2 to the previous value and each successive value for the \( 2y_{k+1} \) term is obtained by subtracting 2 from the previous value.

- The initial decision parameter is obtained by evaluating the circle function at the start position \((x_0,y_0) = (0,r)\)

\[ P_0 = f_{\text{circle}}(1,r-1/2) = 1 + (r-1/2)^2 - r^2 \]

or

\[ P_0 = (5/4) - r \]

If the radius \( r \) is specified as an integer

\[ P_0 = 1 - r \text{ (for } r \text{ an integer)} \]

**Algorithm: Midpoint circle Algorithm**

1. Input radius \( r \) and circle center \((x_c,y_c)\) and obtain the first point on the circumference of the circle centered on the origin as

\[(x_0,y_0) = (0,r)\]

2. Calculate the initial value of the decision parameter as \( P_0 = (5/4) - r \)

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 \((x_{k+1},y_k)\) and \( P_{k+1} = P_k + 2x_{k+1} + 1 \)

Otherwise the next point along the circle is \((x_{k+1},y_{k-1})\) and \( P_{k+1} = P_k + 2x_{k+1} + 1 - 2y_{k+1} \)

Where \( 2x_{k+1} = 2x_{k+2} \) and \( 2y_{k+1} = 2y_{k-2} \)

4. Determine symmetry points in the other seven octants.

5. Move each calculated pixel position \((x,y)\) onto the circular path centered at \((x_c,y_c)\) and plot the coordinate values.

\[ x = x + x_c \quad y = y + y_c \]

6. Repeat step 3 through 5 until \( x > y \).
Example: Midpoint Circle Drawing

Given a circle radius \( r=10 \)

The circle octant in the first quadrant from \( x=0 \) to \( x=y \). The initial value of the decision parameter is \( P_0=1-r = -9 \)

For the circle centered on the coordinate origin, the initial point is \((x_0,y_0)=(0,10)\) and initial increment terms for calculating the decision parameters are

\[
2x_0=0 \quad , \quad 2y_0=20
\]

Successive midpoint decision parameter values and the corresponding coordinate positions along the circle path are listed in the following table.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( p_k )</th>
<th>((x_{k+1},y_{k-1}))</th>
<th>( 2x_{k+1} )</th>
<th>( 2y_{k+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-9</td>
<td>(1,10)</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>-6</td>
<td>(2,10)</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>(3,10)</td>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>(4,9)</td>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>(5,9)</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>(6,8)</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>(7,7)</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>

Implementation of Midpoint Circle Algorithm

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

1.6  Ellipse-Generating Algorithms

An ellipse is an elongated circle. Therefore, elliptical curves can be generated by modifying circle-drawing procedures to take into account the different dimensions of an ellipse along the major and minor axes.
Properties of ellipses

- An ellipse can be given in terms of the distances from any point on the ellipse to two fixed positions called the **foci** of the ellipse. The sum of these two distances is the same values for all points on the ellipse.

- If the distances to the two focus positions from any point \(p=(x,y)\) on the ellipse are labeled \(d_1\) and \(d_2\), then the general equation of an ellipse can be stated as

\[
d_1 + d_2 = \text{constant}
\]

- Expressing distances \(d_1\) and \(d_2\) in terms of the focal coordinates \(F_1=(x_1, y_1)\) and \(F_2=(x_2, y_2)\)

\[
\sqrt{(x-x_1)^2+(y-y_1)^2} + \sqrt{(x-x_2)^2+(y-y_2)^2} = \text{constant}
\]

- By squaring this equation isolating the remaining radical and squaring again. The general ellipse equation in the form

\[
Ax^2 + By^2 + Cxy + Dx + Ey + F = 0
\]

- 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.

- The major axis is the straight line segment extending from one side of the ellipse to the other through the foci. The minor axis spans the shorter dimension of the ellipse, perpendicularly bisecting the major axis at the halfway position (ellipse center) between the two foci.

- An interactive method for specifying an ellipse in an arbitrary orientation is to input the two foci and a point on the ellipse boundary.

- Ellipse equations are simplified if the major and minor axes are oriented to align with the coordinate axes. The major and minor axes oriented parallel to the \(x\) and \(y\) axes parameter \(r_x\) for this example labels the semi major axis and parameter \(r_y\) labels the semi minor axis

\[
\frac{(x-x_c)^2}{r_x^2} + \frac{(y-y_c)^2}{r_y^2} = 1
\]
• Using polar coordinates $r$ and $\theta$, to describe the ellipse in Standard position with the parametric equations

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

• Angle $\theta$ called the eccentric angle of the ellipse is measured around the perimeter of a bounding circle.

• We must calculate pixel positions along the elliptical arc throughout one quadrant, and then we obtain positions in the remaining three quadrants by symmetry.

Midpoint ellipse Algorithm

• The midpoint ellipse method is applied throughout the first quadrant in two parts. The below figure show the division of the first quadrant according to the slope of an ellipse with $r_x < r_y$. 
• In the x direction where the slope of the curve has a magnitude less than 1 and unit steps in the y direction where the slope has a magnitude greater than 1.

Region 1 and 2 can be processed in various ways

1. Start at position \((0, r_y)\) and step clockwise along the elliptical path in the first quadrant shifting from unit steps in x to unit steps in y when the slope becomes less than -1.

2. Start at \((r_x, 0)\) and select points in a counter clockwise order.
   2.1 Shifting from unit steps in y to unit steps in x when the slope becomes greater than -1.0.
   2.2 Using parallel processors calculate pixel positions in the two regions simultaneously.

3. Start at \((0, r_y)\) step along the ellipse path in clockwise order throughout the first quadrant ellipse function \((x_c, y_c) = (0, 0)\)

   \[ f_{\text{ellipse}}(x, y) = ry^2 + rx^2 - rx^2 ry \]

   which has the following properties:

   \[ f_{\text{ellipse}}(x, y) < 0, \text{ if } (x, y) \text{ is inside the ellipse boundary} \]
   \[ = 0, \text{ if } (x, y) \text{ is on ellipse boundary} \]
   \[ > 0, \text{ if } (x, y) \text{ is outside the ellipse boundary} \]

   Thus, the ellipse function \(f_{\text{ellipse}}(x, y)\) serves as the decision parameter in the midpoint algorithm.

Starting at \((0, r_y)\):

Unit steps in the x direction until to reach the boundary between region 1 and region 2. Then switch to unit steps in the y direction over the remainder of the curve in the first quadrant.

At each step to test the value of the slope of the curve. The ellipse slope is calculated

\[ \frac{dy}{dx} = -(2ry^2x/2rx^2) \]

At the boundary between region 1 an region 2

\[ \frac{dy}{dx} = -1.0 \text{ and } 2ry^2x=2rx^2y \]

to more out of region 1 whenever

\[ 2ry^2x\geq 2rx^2y \]
The following figure shows the midpoint between two candidate pixels at sampling position \( x_{k+1} \) in the first region.

- To determine the next position along the ellipse path by evaluating the decision parameter at this mid point

\[
P_{1k} = f_{\text{ellipse}} (x_{k+1}, y_{k-1}/2) = ry^2 (x_{k+1})^2 + rx^2 (y_{k-1}/2)^2 - rx^2 ry^2
\]

- If \( P_{1k} < 0 \), the midpoint is inside the ellipse and the pixel on scan line \( y_k \) is closer to the ellipse boundary. Otherwise the midpoint is outside or on the ellipse boundary and select the pixel on scan line \( y_{k-1} \)

- At the next sampling position \( (x_{k+1}+1=x_{k+2}) \) the decision parameter for region 1 is calculated as

\[
p_{1k+1} = f_{\text{ellipse}}(x_{k+1}+1, y_{k+1} - \frac{1}{2}) = ry^2 [(x_k + 1) + 1]^2 + rx^2 (y_{k+1} - \frac{1}{2})^2 - rx^2 ry^2
\]

Or

\[
p_{1k+1} = p_{1k} + 2 ry^2 (x_k + 1) + ry^2 + rx^2 [(y_{k+1} - \frac{1}{2})^2 - (y_k - \frac{1}{2})^2]
\]

Where \( y_{k+1} \) is \( y_k \) or \( y_{k-1} \) depending on the sign of \( P_{1k} \).

Decision parameters are incremented by the following amounts

\[
\text{increment} = \begin{cases} 
2 ry^2 (x_k + 1) + ry^2 & \text{if } p_{1k} < 0 \\
2 ry^2 (x_k + 1) + ry^2 - 2rx^2 y_{k+1} & \text{if } p_{1k} \geq 0
\end{cases}
\]

- Increments for the decision parameters can be calculated using only addition and subtraction as in the circle algorithm.

- The terms \( 2ry^2 x \) and \( 2rx^2 y \) can be obtained incrementally. At the initial position \((0,ry)\) these two terms evaluate to

\[
2 ry^2 x = 0
\]
\[2rx^2y = 2rx^2 \cdot ry\]

- \(x\) and \(y\) are incremented updated values are obtained by adding \(2ry^2\) to the current value of the increment term and subtracting \(2rx^2\) from the current value of the increment term. The updated increment values are compared at each step and more from region 1 to region 2, when the condition 4 is satisfied.

- In region 1 the initial value of the decision parameter is obtained by evaluating the ellipse function at the start position

\[(x_0, y_0) = (0, ry)\]

- In region 2 at unit intervals in the negative \(y\) direction and the midpoint is now taken between horizontal pixels at each step for this region the decision parameter is evaluated as

\[p_{10} = f_{\text{ellipse}}(1, ry - \frac{1}{2})\]

\[= ry^2 + rx^2 (ry - \frac{1}{2})^2 - rx^2 ry^2\]

Or

\[p_{10} = ry^2 - rx^2 ry + \frac{1}{4} rx^2\]

- Over region 2, we sample at unit steps in the negative \(y\) direction and the midpoint is now taken between horizontal pixels at each step. For this region, the decision parameter is evaluated as

\[p_{2k} = f_{\text{ellipse}}(x_k + \frac{1}{2}, y_k - 1)\]

\[= ry^2 (x_k + \frac{1}{2})^2 + rx^2 (y_k - 1)^2 - rx^2 ry^2\]

1. If \(P_{2k} > 0\), the midpoint position is outside the ellipse boundary, and select the pixel at \(x_k\).
2. If \(P_{2k} \leq 0\), the midpoint is inside the ellipse boundary and select pixel position \(x_{k+1}\).

- To determine the relationship between successive decision parameters in region 2 evaluate the ellipse function at the sampling step: \(y_{k+1} - 1 = y_{k-2}\).

\[P_{2k+1} = f_{\text{ellipse}}(x_{k+1} + \frac{1}{2}, y_{k+1} - 1)\]

\[= ry^2(x_{k+1} + \frac{1}{2})^2 + rx^2 [(y_{k+1} - 1) - 1]^2 - rx^2 ry^2\]

Or

\[p_{2k+1} = p_{2k} - 2 rx^2(y_k - 1) + rx^2 + ry^2 [(x_{k+1} + \frac{1}{2})^2 - (x_k + \frac{1}{2})^2]\]
With x_{k+1} set either to x_k or x_{k+1}, depending on the sign of P_{2k}. When we enter region 2, the initial position (x_0, y_0) is taken as the last position. Selected in region 1 and the initial decision parameter in region 2 is then

\[ P_0 = r_y^2(x_0 + \frac{1}{2})^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2 \]

To simplify the calculation of P_0, select pixel positions in counter clockwise order starting at (r_x, 0). Unit steps would then be taken in the positive y direction up to the last position selected in region 1.

Algorithm: Midpoint Ellipse Algorithm

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
   \[ P_{10} = r_y^2 r_x^2 r_y + (1/4) r_x^2 \]

3. At each x_k position in region 1 starting at k=0 perform the following test. If P_{1k}<0, the next point along the ellipse centered on (0,0) is \((x_{k+1}, y_k)\) and
   \[ P_{1k+1} = P_{1k} + 2 r_y^2 x_{k+1} + r_y^2 \]
   Otherwise the next point along the ellipse is \((x_{k+1}, y_{k+1})\) and
   \[ P_{1k+1} = P_{1k} + 2 r_y^2 x_{k+1} - 2 r_x^2 y_{k+1} + r_y^2 \]
   with
   \[ 2 r_y^2 x_{k+1} + 1 = 2 r_y^2 x_k + 2 r_y^2 \]
   \[ 2 r_x^2 y_k + 1 = 2 r_x^2 y_k + 2 r_x^2 \]
   And continue until \(2 r_y^2 x \geq 2 r x^2 y\)

4. Calculate the initial value of the decision parameter in region 2 using the last point \((x_0, y_0)\) is the last position calculated in region 1.
   \[ P_{20} = r_y^2(x_0 + 1/2)^2 + r_x^2(y_0 - 1)^2 - r_x^2 r_y^2 \]

5. At each position y_k in region 2, starting at k=0 perform the following test. If \(P_{2k}>0\) the next point along the ellipse centered on (0,0) is \((x_k, y_{k+1})\) and
   \[ P_{2k+1} = P_{2k} - 2 r_x^2 y_{k+1} + r_x^2 \]
   Otherwise the next point along the ellipse is \((x_{k+1}, y_{k+1})\) and
   \[ P_{2k+1} = P_{2k} + 2 r_y^2 x_{k+1} - 2 r_x^2 y_{k+1} + r_x^2 \]
   Using the same incremental calculations for x any y as in region 1.

6. Determine symmetry points in the other three quadrants.
7. Move each calculate pixel position \((x, y)\) onto the elliptical path centered on \((x_c, y_c)\) and plot the coordinate values

\[ x = x + x_c, \quad y = y + y_c \]

8. Repeat the steps for region 1 unit \(2r_x^2 x > 2r_y^2 y\)

**Example: Mid point ellipse drawing**

Input ellipse parameters \(r_x = 8\) and \(r_y = 6\) the mid point ellipse algorithm by determining raster position along the ellipse path is the first quadrant. Initial values and increments for the decision parameter calculations are

\[ 2r_y^2 x = 0 \quad (\text{with increment } 2r_y^2 = 72) \]
\[ 2r_x^2 y = 2r_x^2 r_y \quad (\text{with increment } -2r_x^2 = -128) \]

For region 1 the initial point for the ellipse centered on the origin is \((x_0, y_0) = (0, 6)\) and the initial decision parameter value is

\[ p_1^0 = r_y^2 - r_x^2 r_y^2 + 1/4r_x^2 = -332 \]

Successive midpoint decision parameter values and the pixel positions along the ellipse are listed in the following table.

<table>
<thead>
<tr>
<th>K</th>
<th>(p_1_k)</th>
<th>(x_{k+1}, y_{k+1})</th>
<th>(2r_y^2 x_{k+1})</th>
<th>(2r_x^2 y_{k+1})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-332</td>
<td>(1, 6)</td>
<td>72</td>
<td>768</td>
</tr>
<tr>
<td>1</td>
<td>-224</td>
<td>(2, 6)</td>
<td>144</td>
<td>768</td>
</tr>
<tr>
<td>2</td>
<td>-44</td>
<td>(3, 6)</td>
<td>216</td>
<td>768</td>
</tr>
<tr>
<td>3</td>
<td>208</td>
<td>(4, 5)</td>
<td>288</td>
<td>640</td>
</tr>
<tr>
<td>4</td>
<td>-108</td>
<td>(5, 5)</td>
<td>360</td>
<td>640</td>
</tr>
<tr>
<td>5</td>
<td>288</td>
<td>(6, 4)</td>
<td>432</td>
<td>512</td>
</tr>
<tr>
<td>6</td>
<td>244</td>
<td>(7, 3)</td>
<td>504</td>
<td>384</td>
</tr>
</tbody>
</table>

Move out of region 1, \(2r_y 2x > 2r_x^2 y\).

For a region 2 the initial point is \((x_0, y_0) = (7, 3)\) and the initial decision parameter is

\[ p_2^0 = f_{\text{ellipse}}(7 + 1/2, 2) = -151 \]

The remaining positions along the ellipse path in the first quadrant are then calculated as

<table>
<thead>
<tr>
<th>K</th>
<th>(p_2_k)</th>
<th>(x_{k+1}, y_{k+1})</th>
<th>(2r_y^2 x_{k+1})</th>
<th>(2r_x^2 y_{k+1})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-151</td>
<td>(8, 2)</td>
<td>576</td>
<td>256</td>
</tr>
<tr>
<td>1</td>
<td>233</td>
<td>(8, 1)</td>
<td>576</td>
<td>128</td>
</tr>
<tr>
<td>2</td>
<td>745</td>
<td>(8, 0)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Implementation of Midpoint Ellipse drawing

#define Round(a) ((int)(a+0.5))

void ellipseMidpoint (int xCenter, int yCenter, int Rx, int Ry)
{
    int Rx2=Rx*Rx;
    int Ry2=Ry*Ry;
    int twoRx2 = 2*Rx2;
    int twoRy2 = 2*Ry2;
    int p;
    int x = 0;
    int y = Ry;
    int px = 0;
    int py = twoRx2* y;
    void ellipsePlotPoints ( int , int , int , int ) ;
    /* Plot the first set of points */
    ellipsePlotPoints (xcenter, yCenter, x,y ) ;

    / * Region 1 */
    p = ROUND(Ry2 - (Rx2* Ry) + (0.25*Rx2));
    while (px < py)
    {
        x++;
        px += twoRy2;
        if (p < 0)
            p += Ry2 + px;
        else
        {
            y -- ;
            py -= twoRx2;
            p += Ry2 + px - py;
        }
    ellipsePlotPoints(xCenter, yCenter,x,y);
/* Region 2 */
p = ROUND (Ry2*(x+0.5)*' (x+0.5)+ Rx2*(y- 1 )* (y- 1 ) - Rx2*Ry2);
while (y > 0 )
{
    y--;
    py -= twoRx2;
    if (p > 0)
        p += Rx2 - py;
    else
    {
        x++;
        px+=twoRy2;
        p+=Rx2-py+px;
    }
ellipsePlotPoints(xCenter, yCenter,x,y);
}

void ellipsePlotPoints(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);
}
1.6 **Pixel Addressing and Object Geometry**

There are two ways to adjust the dimensions of displayed objects in finite pixel areas,

1. The dimensions of displayed objects are adjusted according to the amount of overlap of pixel areas with the object boundaries
2. We align object boundaries with pixel boundaries instead of pixel centers.

**Advantages of pixel-addressing scheme:**

1. It avoids half-integer pixel boundaries
2. It facilitates precise object representations
3. It simplifies the processing involved in many scan-conversion algorithms and in other raster procedures.

**Maintaining Geometric: Properties of displayed objects**

- When we convert geometric descriptions of objects into pixel representations, we transform mathematical points and lines into finite screen areas.
- During this transformation, if we are to maintain the original geometric measurements specified by the geometric descriptions for an object, we need to consider the finite size of pixels.

1.6.1 **Antialiasing and Antialiasing Techniques**

- In the line drawing algorithms, we have seen that all rasterized locations do not match with the true line and we have to select the optimum raster locations to represent a straight line.
- This problem is severe in low resolution screens. In such screen, line appears like a stair-step. This effect is known as **Aliasing**. It is dominant for lines having gentle and sharp slopes.

**Antialiasing:**

- The aliasing effect can be reduced by adjusting intensities of the pixels along the line. The process of adjusting intensities of the pixels along the line to minimize the effect of aliasing is called **antialiasing**.
- The aliasing effect can be reduced by increasing resolution of the raster display. With raster systems that are capable of displaying more than two intensity levels, we can apply antialiasing methods to modify pixel intensities.

- **Antialiasing methods are basically classified as,**
  1. Supersampling or Postfiltering
  2. Area Sampling or Prefiltering
  3. Filtering Techniques
  4. Pixel Phasing
1.7 Filled Area Primitives

Introduction

- A Ployline is a chain of connected line segments.
- It is specified by giving the verties (nodes) P0, P1, P2, ... and so on.
- The first vertex is called the initial or starting point and the last vertex is known as the final or terminal point.
- When starting point and terminal point of any polyline is same, then it is known as polygon.

![Ployline](image1)

![Polygon](image2)

Types of Ploygons

- The classification of polygons is based on where the line segment joining any two points within the polygon is going to be:
  1. Convex Polygon
     
     A Convex Ploygon is a polygon in which the line segment joining any two points within the polygon lies completely inside the polygon.

![Convex Polygon](image3)

  2. Concave Ploygon
     
     A Convex Ploygon is a polygon in which the line segment joining any two points within the polygon may not lie completely inside the polygon.

![Concave Polygon](image4)
Representation of Polygons

- There are three approaches to represent polygons according to the graphics systems:

  1. **Polygon drawing primitive approach**
  2. **Trapezoid primitive approach**
  3. **Line and point approach**

- Some graphics devices support polygon drawing primitive approach and they can directly draw the polygon shapes.
- Some devices support trapezoid primitives. In such devices, trapezoid is formed when two scan lines and two line segments.


![Polygon](image1.png) ![Polygon as a series of trapezoids](image2.png)

- Most of the devices do not provide any polygon support at all. In such devices polygons are represented using lines and points.
- A polygon is represented as a unit and it is stored in the display file.

**Polygon Algorithm:**

1. Read AX and AY of length N
   
   \[ \text{AX and AY are arrays containing the vertices of the polygon and } N \text{ is the number of polygon sides} \]

2. \( i = 0 \)  
   
   \[ \text{Initialize counter to count number of sides} \]

   \[
   \begin{align*}
   \text{DF.OP } [i] &= N \\
   \text{DF.x } [i] &= \text{AX } [i] \\
   \text{DF.y } [i] &= \text{AY } [i] \\
   i &= i + 1
   \end{align*}
   \]
3. do 
   
   { 
   DF_OP[i] = 2
   DF_x[i] = AX[i] 
   DF_y[i] = AY[i] 
   i = i + 1 
   } while (i<N) 
4. DF_OP[i] = 2 
   DF_x[i] = AX[0] 
   DF_y[i] = AY[0] 

5. Stop 

Polygon Filling 
- Fille the polygon means highlighting all pixels which lie inside the polygon with any colour other than background colour. 
- Ploygons are easier to fill since they have linear boundaries. There are two basic approaches to fill the polygon such as, 
  1. Seed Fill Algorithm 
  - One way to fill a polygon is to start from a given 'seed' point known to be inside the polygon and highlight outward from this point that is neighbouring pixels until we encounter the boundary pixels. 
  - This approach is known as seed fill because colour flows from the seed point until reaching the polygon boundary, like water flooding on the surface of the container. 
  - The seed fill algorithm is further classified as Flood Fill Algorithm and Boundary Fill Algorithm. 
  - Algorithms that fill interior-defined regions are called flood-fill algorithms. Those that fill boundary-defined regions are called boundary-fill algorithms or edge-fill algorithms.
2. **Scan-Line Algorithm**

- In this approach, inside test will be applied to check whether the pixel is inside the polygon or outside the polygon and then highlight pixels which lie inside the polygon.
- This approach is known as **scan-line algorithm**. It avoids the need for a seed pixel but it requires some computation.
- This algorithm solves the hidden surface problem while generating display scan line.
- It is used in orthogonal projection and it is non-recursive algorithm.
- In scan line algorithm, we have to stack only a beginning position for each horizontal pixel scan, instead of stacking all unprocessed neighbouring positions around the current position.

**PART-A**

1. Define Computer graphics.
2. What are the video display devices
3. Define refresh buffer/frame buffer.
4. What is meant by scan code?
5. List out the merits and demerits of Penetration techniques?
6. List out the merits and demerits of DVST
7. What do you mean by emissive and non-emissive displays
8. List out the merits and demerits of Plasma panel display
9. What is raster scan and Random scan systems
10. What is pixel?
11. What are the Input devices and Hard copy devices?
12. Define aspect ratio.
13. What is Output Primitive? What is point and lines in the computer graphics system?
14. What is DDA? What are the disadvantages of DDA algorithm?
15. Digitize a line from (10,12) to (15,15) on a raster screen using Bresenham's straight line Algorithm what are the various line drawing algorithms
16. What is loading a frame buffer?
17. What is meant by antialiasing?
18. What is a filled area primitive?
19. What are the various for the Filled area Primitives
20. What is pixel addressing and object addressing

**PART-B**

1. Explain the following Video Displays Devices (a) refresh cathode ray tube(b) raster Scan Displays (c) Random Scan Displays (d) Color SRT Monitors
2. Explain Direct View Storage Tubes(b) Flat Panel Displays (c) Liquid Crystal Displays
3. Explain Raster scan systems and Raster Scan Systems
4. Explain the Various Input Devices
5. Explain (a) Hard Copy devices(b) Graphics Software
6. Explain in detail about the Line drawing DDA scan conversion algorithm?
7. Write down and explain the midpoint circle drawing algorithm. Assume 10 cm as the radius and co-ordinate as the centre of the circle.
8. Calculate the pixel location approximating the first octant of a circle having centre at (4,5) and radius 4 units using Bresenham's algorithm
9. Explain Ellipse generating Algorithm?
10. Explain Boundary Fill Algorithm?