UNIT- 1

Overview of Computer Graphics

Application of Computer Graphics

Computer-Aided Design for engineering and architectural systems etc.
Objects maybe displayed in a wireframe outline form. Multi-window environment is also favored for producing various zooming scales and views.
Animations are useful for testing performance.

Presentation Graphics
To produce illustrations which summarize various kinds of data. Except 2D, 3D graphics are good tools for reporting more complex data.

Computer Art
Painting packages are available. With cordless, pressure-sensitive stylus, artists can produce electronic paintings which simulate different brush strokes, brush widths, and colors. Photorealistic techniques, morphing and animations are very useful in commercial art. For films, 24 frames per second are required. For video monitor, 30 frames per second are required.

Entertainment
Motion pictures, Music videos, and TV shows, Computer games

Education and Training
Training with computer-generated models of specialized systems such as the training of ship captains and aircraft pilots.

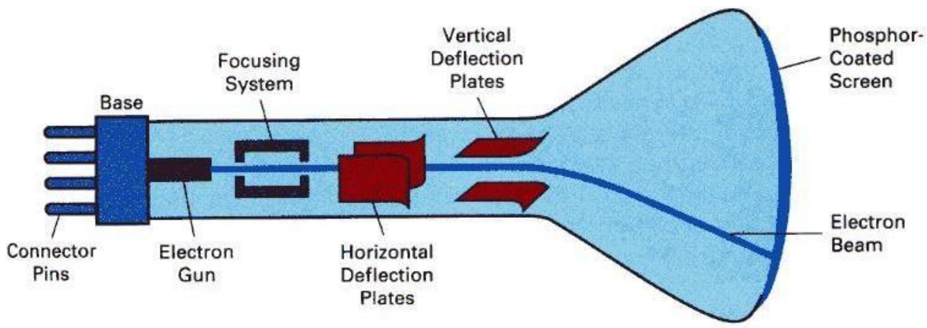
Visualization
For analyzing scientific, engineering, medical and business data or behavior. Converting data to visual form can help to understand mass volume of data very efficiently.

Image Processing
Image processing is to apply techniques to modify or interpret existing pictures. It is widely used in medical applications.

Graphical User Interface
Multiple window, icons, menus allow a computer setup to be utilized more efficiently.

Video Display devices

Cathode-Ray Tubes (CRT) - still the most common video display device presently



Electrostatic deflection of the electron beam in a CRT

An electron gun emits a beam of electrons, which passes through focusing and deflection systems and hits on the phosphor-coated screen. The number of points displayed on a CRT is referred to as r**esolutions** (eg. 1024x768). Different phosphors emit small light spots of different colors, which can combine to form a range of colors. A common methodology for color CRT display is the **Shadow-mask** meth

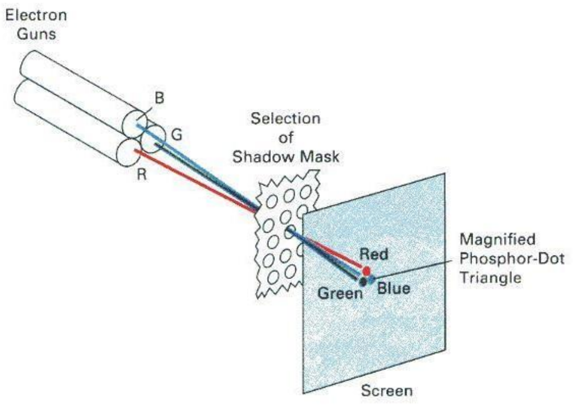
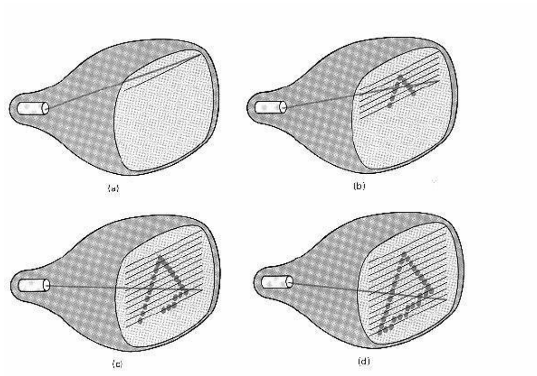


Illustration of a shadow-mask CRT

The light emitted by phosphor fades very rapidly, so it needs to redraw the picture repeatedly. There are 2 kinds of redrawing mechanisms: Raster-Scan and Random-Scan

Raster-Scan



The electron beam is swept across the screen one row at a time from top to bottom. As it moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots. This scanning process is called refreshing. Each complete scanning of a screen is normally called a **frame**.

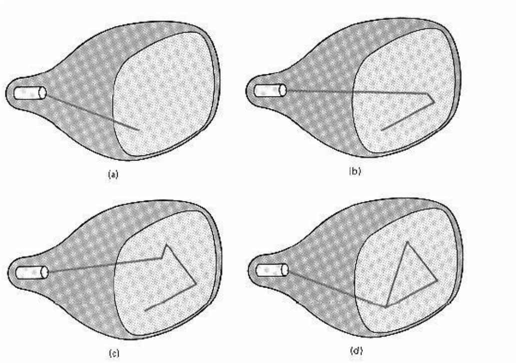
The refreshing rate, called the **frame rate**, is normally 60 to 80 frames per second, or described as 60 Hz to 80 Hz.

Picture definition is stored in a memory area called the **frame buffer**. This frame buffer stores the intensity values for all the screen points. Each screen point is called a **pixel** (picture element).

On black and white systems, the frame buffer storing the values of the pixels is called a **bitmap**. Each entry in the bitmap is a 1-bit data which determine the on (1) and off (0) of the intensity of the pixel.

On color systems, the frame buffer storing the values of the pixels is called a **pixmap** (Though nowadays many graphics libraries name it as bitmap too). Each entry in the pixmap occupies a number of bits to represent the color of the pixel. For a true color display, the number of bits for each entry is 24 (8 bits per red/green/blue channel, each channel 2^8=256 levels of intensity value, ie. 256 voltage settings for each of the red/green/blue electron guns).

Random-Scan (Vector Display)



The CRT's electron beam is directed only to the parts of the screen where a picture is to be drawn. The picture definition is stored as a set of line-drawing commands in a refresh display file or a refresh buffer in memory.

Random-scan generally have higher resolution than raster systems and can produce smooth line drawings, however it cannot display realistic shaded scenes.

Display Controller

For a raster display device reads the frame buffer and generates the control signals for the screen, ie. the signals for horizontal scanning and vertical scanning. Most display controllers include a **color map** (or video look-up table). The major function of a color map is to provide a mapping between the input pixel value to the output color.

Anti-Aliasing

On dealing with integer pixel positions, jagged or stair step appearances happen very usually. This distortion of information due to under sampling is called aliasing. A number of ant aliasing methods have been developed to compensate this problem.

One way is to display objects at higher resolution. However there is a limit to how big we can make the frame buffer and still maintaining acceptable refresh rate.

Drawing a Line in Raster Devices

DDA Algorithm

In computer graphics, a hardware or software implementation of a digital differential analyzer (DDA) is used for linear interpolation of variables over an interval between start and end point. DDAs are used for rasterization of lines, triangles and polygons. In its simplest implementation the DDA Line drawing algorithm interpolates values in interval [(xstart, ystart), (xend, yend)] by computing for each xi the equations $x_i = x_{i-1} + 1/m$, $y_i = y_{i-1} + m$, where $\Delta x = x_{end} - x_{start}$ and $\Delta y = y_{end} - y_{start}$ and $m = \Delta y/\Delta x$.

The dda is a scan conversion line algorithm based on calculating either dy or dx. A line is sampled at unit intervals in one coordinate and corresponding integer values nearest the line path

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

Subscript k takes integer values starting from 0, for the 1st point and increases by until endpoint is reached. y value is rounded off to nearest integer to correspond to a screen pixel.
For lines with slope greater than 1, we reverse the role of x and y i.e. we sample at dy=1 and calculate consecutive x values as

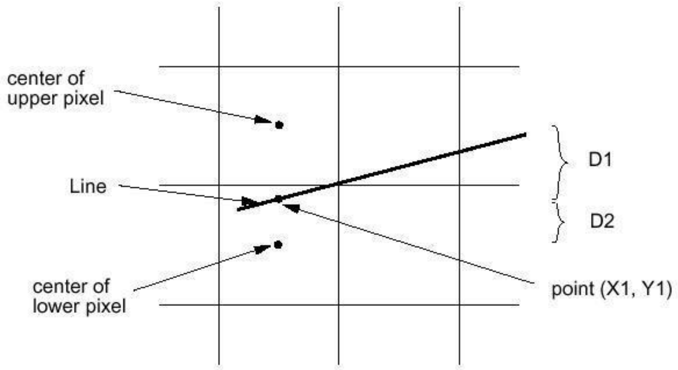
Similar calculations are carried out to determine pixel positions along a line with negative slope. Thus, if the absolute value of the slope is less than 1, we set dx=1 if i.e. the starting extreme point is at the left.

The basic concept is:

- A line can be specified in the form:

$$y = mx + c$$

- Let m be between 0 to 1, then the slope of the line is between 0 and 45 degrees.
- For the x-coordinate of the left end point of the line, compute the corresponding y value according to the line equation. Thus we get the left end point as (x1,y1), where y1 may not be an integer.
- Calculate the distance of (x1,y1) from the center of the pixel immediately above it and call it D1
- Calculate the distance of (x1,y1) from the center of the pixel immediately below it and call it D2
- If D1 is smaller than D2, it means that the line is closer to the upper pixel than the lower pixel, then, we set the upper pixel to on; otherwise we set the lower pixel to on.
- Then increatement x by 1 and repeat the same process until x reaches the right end point of the line.
- This method assumes the width of the line to be zero



Bresenham's Line Algorithm

This algorithm is very efficient since it use only incremental integer calculations. Instead of calculating the non-integral values of D1 and D2 for decision of pixel location, it computes a value, p, which is defined as:

p = (D2-D1)* horizontal length of the line
if p>0, it means D1 is smaller than D2, and we can determine the pixel location accordingly

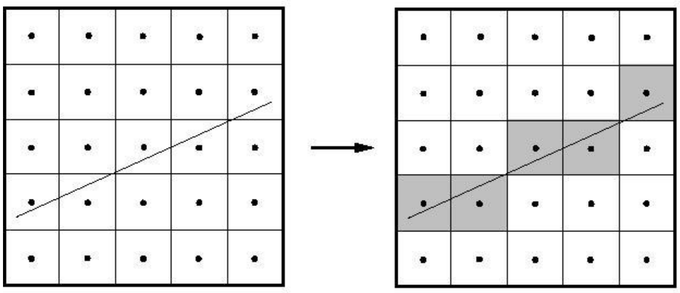
However, the computation of p is very easy:
The initial value of p is 2 * vertical height of the line - horizontal length of the line.

At succeeding x locations, if p has been smaller than 0, then, we increment p by 2 * vertical height of the line, otherwise we increment p by 2 * (vertical height of the line - horizontal length of the line)

All the computations are on integers. The incremental method is applied to

```
void BresenhamLine(int x1, int y1, int x2, int y2)
{ int x, y, p, const1, const2; /* initialize
    variables */ p=2*(y2-y1)-(x2-x1);
    const1=2*(y2-y1); const2=2*((y2-
    y1)-(x2-x1));

    x=x1;
    y=y1;
    SetPixel(x,y);
    while (x<xend) {
    x++;
        if (p<0)
        {  p=p+const1;
        }
        else
        { y++;
            p=p+const2;
        }
        SetPixel(x,y);
    }
}
```



Bitmap

- A graphics pattern such as an icon or a character may be needed frequently, or may need to be re-used.
- Generating the pattern every time when needed may waste a lot of processing time.
- A bitmap can be used to store a pattern and duplicate it to many places on the image or on the screen with simple copying operations.

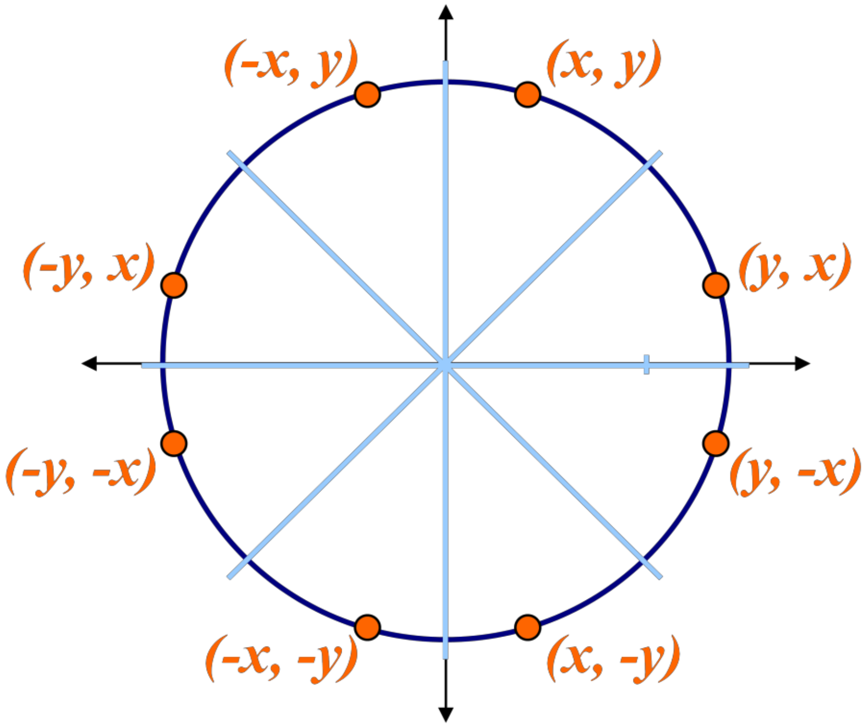
Mid Point circle Algorithm

However, unsurprisingly this is not a brilliant solution!
Firstly, the resulting circle has large gaps where the slope approaches the vertical
Secondly, the calculations are not very efficient

The square (multiply) operations

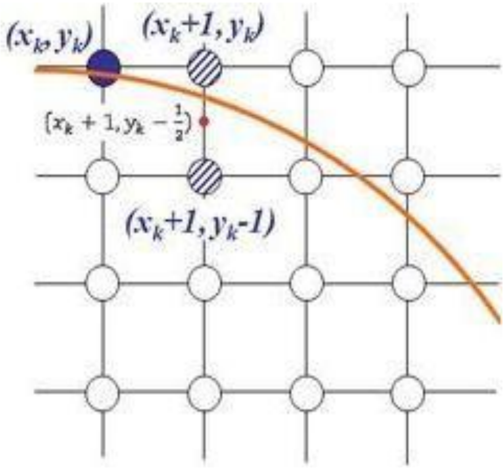
The square root operation – try really hard to avoid these!

We need a more efficient, more accurate solution.
The first thing we can notice to make our circle drawing algorithm more efficient is that circles centred at (*0*, *0*) have *eight-way symmetry*



Similarly to the case with lines, there is an incremental algorithm for drawing circles – the *mid-point circle algorithm*
In the mid-point circle algorithm we use eight-way symmetry so only ever calculate the points for the top right eighth of a circle, and then use symmetry to get the rest of the points

Assume that we have just plotted point (x_k, y_k)
The next point is a choice between (x_k+1, y_k) and (x_k+1, y_k-1)



We would like to choose

the point that is nearest to
the actual circle
So how do we make this choice?

Let's re-jig the equation of the circle slightly to give us:
The equation evaluates as follows:

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

$$f_{circ}(x, y) \begin{cases} < 0, \\ = 0, \\ > 0, \end{cases}$$

<0 if (x, y) is outside thecircleboundary
=0 if (x, y) is on thecircleboundary
>0 if (x, y) is inside thecircleboundary

By evaluating this function at the midpoint between the candidate pixels we can make our decision

Assuming we have just plotted the pixel at (x_k, y_k) so we need to choose between (x_k+1, y_k) and (x_k+1, y_k-1)
Our decision variable can be defined as:

$$p_k = f_{circ}(x_k + 1, y_k - \tfrac{1}{2})$$
$$= (x_k + 1)^2 + (y_k - \tfrac{1}{2})^2 - r^2$$

If $p_k < 0$ the midpoint is inside the circle and and the pixel at y_k is closer to the circle

Otherwise the midpoint is outside and y_k-1 is closer

To ensure things are as efficient as possible we can do all of our calculations incrementally
First consider:

$$p_{k+1} = f_{circ}\left(x_{k+1}+1, y_{k+1} - \tfrac{1}{2}\right)$$

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

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

where y_{k+1} is either y_k or y_k-1 depending on the sign of p_k

The first decision variable is given as:

$$p_0 = f_{circ}(1, r - \tfrac{1}{2})$$
$$= 1 + (r - \tfrac{1}{2})^2 - r^2$$
$$= \tfrac{5}{4} - r$$

Then if $p_k < 0$ then the next decision variable is given as:

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

If $p_k > 0$ then the decision variable is:

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_k + 1$$

Input radius r and circle centre (x_c, y_c), then set the coordinates for the first point on the circumference of a circle centred on the origin as:

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

- Calculate the initial value of the decision parameter as:

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

- Starting with $k = 0$ at each position x_k, perform the following test. If $p_k < 0$, the next point along the circle centred 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:

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

Determine symmetry points in the other seven octants
Move each calculated pixel position (x, y) onto the circular path centred at (x_c, y_c) to plot the coordinate values:

$x = x + x_c \quad y = y + y_c$

Repeat steps 3 to 5 until $x >= y$

To see the mid-point circle algorithm in action lets use it to draw a circle centred at (0,0) with radius 10