Unit 1
Overview of graphics systems and output primitives

Computer Graphics

Introduction

Computer Graphics is a field related to the generation of graphics using computers. It includes the creation, storage, and manipulation of images of objects. These objects come from diverse fields such as physical, mathematical, engineering, architectural, abstract structures and natural phenomenon. Computer graphics today is largely interactive, i.e. the user controls the contents, structure, and appearance of images of the objects by using input devices, such as keyboard, mouse, or touch-sensitive panel on the screen.

In short, Computer graphics refer different things in different contexts:

– **Pictures**, scenes that are generated by a computer.
– **Tools** used to make such pictures, software and hardware, input/output devices.
– The **whole field of study** that involves these tools and the pictures they produce.

Until the early 1980's computer graphics was a small, specialized field, largely because the hardware was expensive and graphics-based application programs that were easy to use and cost-effective were few. Then, personal computers with built-in raster graphics displays-such as the Xerox Star, Apple Macintosh and the IBM PC- popularized the use of bitmap graphics for user-computer interaction. A bitmap is an ones and zeros representation of the rectangular array points on the screen. Each point is called a pixel, short for "Picture Elements". Once bitmap graphics became affordable, and explosion of easy-to-use and inexpensive graphics-based applications soon followed. Graphics-based user interfaces allowed millions of new users to control simple, low-cost application programs, such as word-processors, spreadsheets, and drawing programs.

The concepts of a "desktop" now became a popular for organizing screen space. By means of a window manager, the user could create position and resize rectangular screen areas called windows. This allowed user to switch among multiple activities just by pointing and clicking at the desired window, typically with a mouse. Besides windows, icons which represent data files, application program, file cabinets, mailboxes, printers, recycle bin, and so on, made the user-computer interaction more effective. By pointing and clicking the icons, users could activate the corresponding programs or objects, which replaced much of the typing of the commands used in earlier operating systems and computer applications.

Today, almost all interactive application programs, even those for manipulating text (e.g. word processor) or numerical data (e.g. spreadsheet programs), use graphics extensively in the user interface and for visualizing and manipulating the application-specific objects.
Even people who do not use computers encounter computer graphics in TV commercials and as cinematic special effects. Computer graphics is no longer a rarity. It is an integral part of all computer user interfaces, and is indispensable for visualizing 2D, 3D objects in diverse areas such as education, science, engineering, medicine, commerce, the military, advertising, and entertainment.

Historical background

Guys, Its quite descriptive but interesting one, read thoroughly.

Prehistory
The foundations of computer graphics can be traced to artistic and mathematical ``inventions,'' for example,

- Euclid (circa 300 - 250 BC) who's formulation of geometry provides a basis for graphics concepts.
- Filippo Brunelleschi (1377 - 1446) architect, goldsmith, and sculptor who is noted for his use of perspective.
- Rene Descartes' (1596-1650) who developed analytic geometry, in particular coordinate systems which provide a foundation for describing the location and shape of objects in space.
- Gottfried Wilhelm Leibniz (1646 - 1716) and Isaac Newton (1642 - 1727) who co-invented calculus that allow us to describe dynamical systems.
- James Joseph Sylvester (1814 - 1897) who invented matrix notation. A lot of graphics can be done with matrices.
- I. Schoenberg who discovered splines, a fundamental type of curve.
- J. Presper Mauchly (1919 - 1995) and John William Mauchly (1907 - 1980) who build the ENIAC computer.

Early History
History of computer graphics dates from the Whirlwind Project and the SAGE computer system, which were designed to support military preparedness. The Whirlwind Project started as an effort to build a flight simulator and SAGE was to provide a air defense system in the United States to guard against the threat of a nuclear attack. The SAGE workstation had a vector display and light pens that operators would use pinpoint planes flying over regions of the United States. We can see a SAGE workstation at the Boston Computer Museum.
Besides the being the age of the first vacuum tube computers, the 1940's were when the transistor was invented at Bell Labs (1947). In 1956, the first transistorized computer was built at MIT.

The Age of Sutherland
In the early 1960's IBM, Sperry-Rand, Burroughs and a few other computer companies existed. The computers of the day had a few kilobytes of memory, no operating systems to speak of and no graphical display monitors. The peripherals were Hollerith punch cards, line printers, and roll-paper plotters. The only programming languages supported were assembler, FORTRAN, and Algol. Function graphs and "Snoopy" calendars were about the only graphics done.
In 1963 Ivan Sutherland presented his paper *Sketchpad* at the Summer Joint Computer Conference. Sketchpad allowed interactive design on a vector graphics display monitor with a light pen input device. Most people mark this event as the origins of computer graphics.

The Middle to Late '60's
Software and Algorithms
Jack Bresenham theorized line drawing algorithm on a raster device. He later extended this to circles. Anti-aliased lines and curve drawing is a major topic in computer graphics. Larry Roberts pointed out the usefulness of homogeneous coordinates, 4x4 matrices and hidden line detection algorithms. Steve Coons introduced parametric surfaces and developed early computer aided geometric design concepts. The earlier work of Pierre Bezier on parametric curves and surfaces also became public. Author Appel at IBM developed hidden surface and shadow algorithms that were pre-cursors to ray tracing. The fast Fourier transform was discovered by Cooley and Tukey. This algorithm allows us to better understand signals and is fundamental for developing antialiasing techniques. It is also a precursor to wavelets.
Hardware and Technology
Doug Englebart invented the mouse at Xerox PARC. The Evans & Sutherland Corporation and General Electric started building flight simulators with real-time raster graphics. The floppy disk was invented at

IBM and the microprocessor was invented at Intel. The concept of a research network, the ARPANET, was developed.

The Early '70's
The state of the art in computing was an IBM 360 computer with about 64 KB of memory, a Tektronix 4014 storage tube, or a vector display with a light pen (but these were very expensive).
Software and Algorithms
Rendering (shading) were discovered by Gouraud and Phong at the University of Utah. Phong also introduced a reflection model that included specular highlights. Keyframe based animation for 3-D graphics was demonstrated. Xerox PARC developed a ``paint'' program. Ed Catmull introduced parametric patch rendering, the *z*-buffer algorithm, and texture mapping. BASIC, C, and UNIX were developed at Dartmouth and Bell Labs.
Hardware and Technology
An Evans & Sutherland Picture System was the high-end graphics computer. It was a vector display with hardware support for clipping and perspective. Xerox PARC introduced the Altos personal computer, and an 8 bit computer was invented at Intel.

The Middle to Late '70's
Software and Algorithms
Turned Whitted developed recursive ray tracing and it became the standard for photorealism, living in a pristine world. Pascal was the programming language everyone learned.
Hardware and Technology
The Apple I and II computers became the first commercial successes for personal computing. The DEC VAX computer was the mainframe (mini) computer of choice. Arcade games such as Pong and Pac Mac became popular. Laser printers were invented at Xerox PARC.

The Early '80's
Software and Algorithms
No notable progress.
Hardware and Technology
The IBM PC was marketed in 1981 The Apple MacIntosh started production in 1984, and microprocessors began to take off, with the Intel x86 chipset, but these were still toys. Computers with a mouse, bitmapped (raster) display, and Ethernet became the standard in academic and science and engineering settings.

The Middle to Late '80's
Software and Algorithms
Jim Blinn introduces blobby models and texture mapping concepts. Binary space partitioning (BSP) trees were introduced as a data structure, but not many realized how useful they would become. Loren Carpenter started exploring fractals in computer graphics. Postscript was developed by John Warnock and Adobe was formed. Steve Cook introduced stochastic sampling to ray tracing. Character animation became the goal for animators. Radiosity was introduced by the Greenberg and folks at Cornell. Photoshop was marketed by Adobe. Video arcade games took off, many people/organizations started publishing on the desktop. UNIX and X windows were the platforms of choice with programming in C and C++, but MS-DOS was starting to rise. Remarkably, the PHIGS (programmers hierarchical Interactive Graphics System) standard came into play, which is later dominated by openGL in 90’s.
Hardware and Technology

Sun workstations, with the Motorola 680x0 chipset became popular as advanced workstation a in the mid 80's. The Video Graphics Array (VGA) card was invented at IBM. Silicon Graphics (SGI) workstations that supported real-time raster line drawing and later polygons became the computer graphicists desired. The data glove, a precursor to virtual reality, was invented at NASA. VLSI for special purpose graphics processors and parallel processing became hot research areas.

The Early '90's

The computer to have now was an SGI workstation with at least 16 MB of memory, at 24-bit raster display with hardware support for Gouraud shading and *z*-buffering for hidden surface removal. Laser printers and single frame video recorders were standard. UNIX, X and Silicon Graphics GL were the operating systems, window system and application programming interface (API) that graphicist used. Shaded raster graphics were starting to be introduced in motion pictures. PCs started to get decent, but still they could not support 3-D graphics, so most programmer's wrote software for scan conversion (rasterization) used the painter's algorithm for hidden surface removal, and developed ``tricks'' for real-time animation.

Software and Algorithms

Mosaic, the first graphical Internet browser was written by Marc Andreessen and Eric Bina at the University of Illinois, National Center for Scientific Applications (NCSA). MPEG standards for compressed video began to be promulgated. Dynamical systems (physically based modeling) that allowed animation with collisions, gravity, friction, and cause and effects were introduced. In 1992 OpenGL (Open Graphics Library) became the standard for graphics APIs In 1993; the World Wide Web took off. Surface subdivision algorithms were rediscovered. Wavelets begin to be used in computer graphics.

Hardware and Technology

Hand-held computers were invented at Hewlett-Packard about 1991. Zip drives were invented at Iomega. The Intel 486 chipset allowed PC to get reasonable floating point performance. In 1994, Silicon Graphics produced the Reality Engine: It had hardware for real-time texture mapping. The Ninetendo 64 game console hit the market providing Reality Engine-like graphics for the masses of games players. Scanners were introduced.

The Middle to Late '90's

The PC market erupts and supercomputers begin to wane. Microsoft grows, Apple collapses, but begins to come back, SGI collapses, and lots of new startups enter the graphics field.

Software and Algorithms

Image based rendering became the area for research in photo-realistic graphics. Linux and open source software become popular.

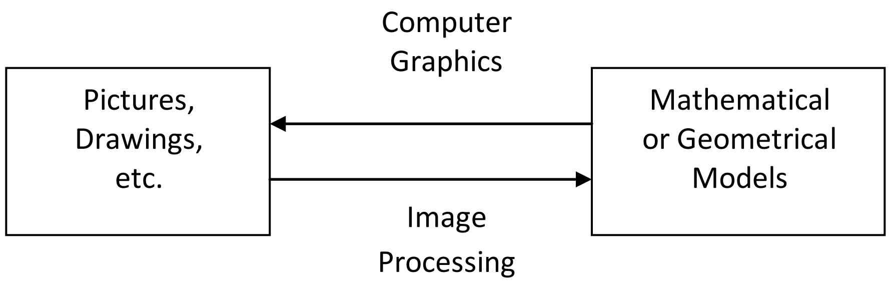
Hardware and Technology

PC graphics cards, for example 3dfx and Nvidia, were introduced. Laptops were introduced to the market. The Pentium chipset makes PCs almost as powerful as workstations. Motion capture, begun with the data glove, becomes a primary method for generating animation sequences. 3-D video games become very popular: DOOM (which uses BSP trees), Quake, Mario Brothers, etc. Graphics effects in movies become pervasive: Terminator 2, Jurassic Park, Toy Story, Titanic, Star Wars I. Virtual reality and the Virtual Reality Meta (Markup) Language (VRML) become hot areas for research. PDA's, the Palm Pilot, and flat panel displays hit the market.

The '00's

Today most graphicist want an Intel PC with at least 256 MB of memory and a 10 GB hard drive. Their display should have graphics board that supports real-time texture mapping. A flatbed scanner, color laser printer, digital video camera, DVD, and MPEG encoder/decoder are the peripherals one wants. The environment for program development is most likely Windows and Linux, with Direct 3D and OpenGL, but Java 3D might become more important. Programs would typically be written in C++ or Java.
What will happen in the near future -- difficult to say, but high definition TV (HDTV) is poised to take off (after years of hype). Ubiquitous wireless computing should become widespread, and audio and gestural input devices should replace some of the functionality of the keyboard and mouse.
You should expect 3-D modeling and video editing for the masses, computer vision for robotic devices and capture facial expressions, and realistic rendering of difficult things like a human face, hair, and water. With any luck C++ will fall out of favor.

The Difference between Computer Graphics and Image Processing:



- ***Computer Graphics***: Synthesize pictures from mathematical or geometrical models.
- ***Image Processing***: analyze pictures to derive descriptions (often in mathematical or geometrical forms) of objects appeared in the pictures.

Applications of Computer Graphics

Computer graphics is used today in many different areas of science, engineering, industry, business, education, entertainment, medicine, art and training. All of these are included in the following categories.

1. User interfaces

Most applications have user interfaces that rely on desktop windows systems to manage multiple simultaneous activities, and on point-and click facilities to allow users to select menu items, icons and objects on the screen. These activities fall under computer graphics. Typing is necessary only to input text to be stored and manipulated. For example, Word processing, spreadsheet, and desktop-publishing programs are the typical examples where user-interface techniques are implemented.

2. Plotting

Plotting 2D and 3D graphs of mathematical, physical, and economic functions use computer graphics extensively. The histograms, bar, and pie charts; the task-scheduling charts are the most commonly used plotting. These all are used to present meaningfully and concisely the trends and patterns of complex data.

3. Office automation and electronic publishing

Computer graphics has facilitated the office automation and electronic publishing which is also popularly known as desktop publishing, giving more power to the organizations to print the meaningful materials in-house. Office automation and electronic publishing can produce both traditional printed (Hardcopy) documents and electronic (softcopy) documents that contain text, tables, graphs, and other forms of drawn or scanned-in graphics.

4. Computer Aided Drafting and Design

One of the major uses of computer graphics is to design components and systems of mechanical, electrical, electrochemical, and electronic devices, including structures such as buildings, automobile bodies, airplane and ship hulls, very large scale integrated (VLSI) chips, optical systems and telephone and computer networks. These designs are more frequently used to test the structural, electrical, and thermal properties of the systems.

5. Scientific and business Visualization

Generating computer graphics for scientific, engineering, and medical data sets is termed as scientific visualization whereas business visualization is related with the non scientific data sets such as those obtained in economics. Visualization makes easier to understand the trends and patterns inherent in the huge amount of data sets. It would, otherwise, be almost impossible to analyze those data numerically.

6. Simulation and modeling

Simulation is the imitation of the conditions like those, which is encountered in real life. Simulation thus helps to learn or to feel the conditions one might have to face in near future without being in danger at the beginning of the course. For example, astronauts can exercise the feeling of weightlessness in a simulator; similarly a pilot training can be conducted in flight simulator. The military tank simulator, the naval simulator, driving simulator, air traffic control simulator, heavy-duty vehicle simulator, and so on are some of the mostly used simulator in practice. Simulators are also used to optimize the system, for example the vehicle, observing the reactions of the driver during the operation of the simulator.

7. Entertainment

Disney movies such as Lion Kings and The Beauty of Beast, and other scientific movies like Jurassic Park, The lost world etc are the best example of the application of computer graphics in the field of entertainment. Instead of drawing all necessary frames with slightly changing scenes for the production of cartoon-film, only the key frames are sufficient for such cartoon-film where the in between frames are interpolated by the graphics system dramatically decreasing the cost of production while maintaining the quality. Computer and video games such FIFA, Doom, Pools are few to name where graphics is used extensively.

8. **Art and commerce**

Here computer graphics is used to produce pictures that express a message and attract attention such as a new model of a car moving along the ring of the Saturn. These pictures are frequently seen at transportation terminals supermarkets, hotels etc. The slide production for commercial, scientific, or educational presentations is another cost effective use of computer graphics. One of such graphics packages is a PowerPoint.

9. Cartography

Cartography is a subject, which deals with the making of maps and charts. Computer graphics is used to produce both accurate and schematic representations of geographical and other natural phenomena from measurement data. Examples include geographic maps, oceanographic charts, weather maps, contour maps and population-density maps. Surfer is one of such graphics packages, which is extensively used for cartography.

Graphics Hardware Systems

Video display devices

Typically, primary output device in a graphics system is video monitor whose operation is based mostly on standard **cathode-ray tube (CRT)** design.

Cathode Ray Tube (CRT)

- CRTs are the most common display devices on computer today. A CRT is an evacuated glass tube, with a heating element on one end and a phosphor-coated screen on the other end.
- When a current flows through this heating element (filament) the conductivity of metal is reduced due to high temperature. These cause electrons to pile up on the filament.
- These electrons are attracted to a strong positive charge from the outer surface of the focusing anode cylinder.
- Due to the weaker negative charge inside the cylinder, the electrons head towards the anode forced into a beam and accelerated towards phosphor-coated screen by the high voltage in inner cylinder walls.
- The forwarding fast electron beam is called Cathode Ray. A cathode ray tube is shown in figure below.

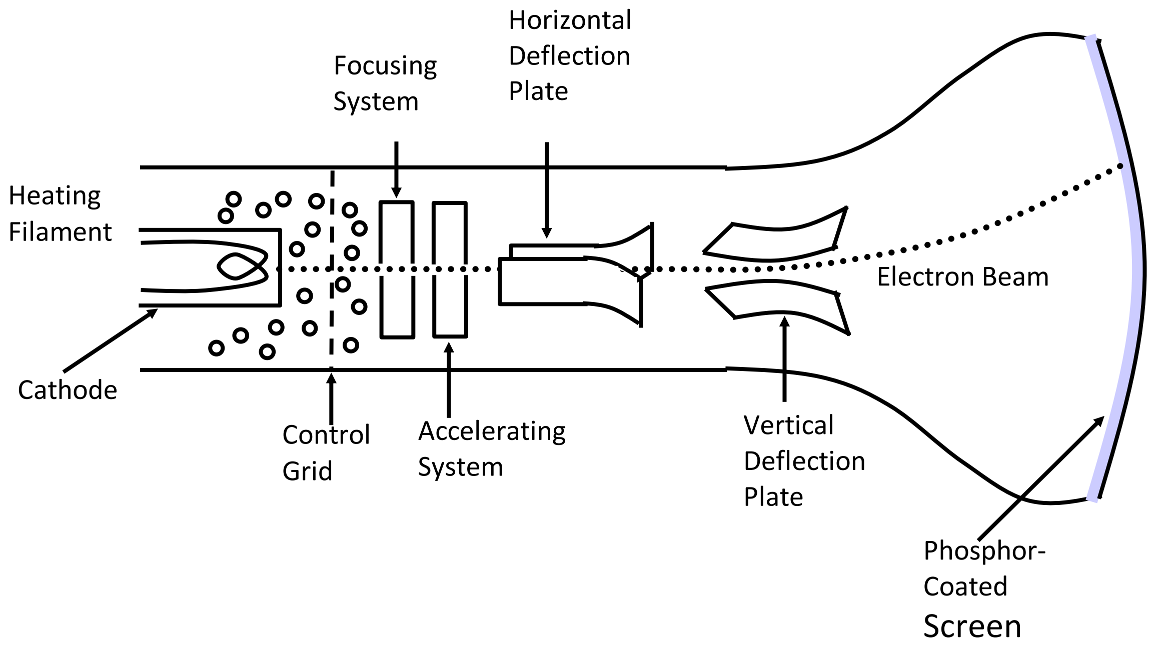


Fig: Cathode-ray tube (CRT)

- There are two sets of weakly charged deflection plates with oppositely charged, one positive and another negative. The first set displaces the beam up and down and the second displaces the beam left and right.
- The electrons are sent flying out of the neck of bottle (tube) until the smash into the phosphor coating on the other end.
- When electrons strike on phosphor coating, the phosphor then emits a small spot of light at each position contacted by electron beam. The glowing positions are used to represent the picture in the screen.

- The amount of light emitted by the phosphor coating depends on the no of electrons striking the screen. The brightness of the display is controlled by varying the voltage on the control grid.

<u>Persistence</u>:
How long a phosphor continues to emit light after the electron beam is removed?

- Persistence of phosphor is defined as **time** it takes for emitted light to decay to 1/10 (10%) of its original intensity. Range of persistence of different phosphors can react many seconds.
- Phosphors for graphical display have persistence of 10 to 60 microseconds. Phosphors with low persistence are useful for animation whereas high persistence phosphor is useful for highly complex, static pictures.

<u>Refresh Rate</u>:

- Light emitted by phosphor fades very rapidly, so to keep the drawn picture glowing constantly; it is required to redraw the picture repeatedly and quickly directing the electron beam back over the some point. The no of times/sec the image is redrawn to give a feeling of non-flickering pictures is called refresh-rate.
- If Refresh rate decreases, flicker develops.
- Refresh rate above which flickering stops and steady it may be called as critical fusion frequency (CFF).

<u>Resolution</u>:
Maximum number of points displayed horizontally and vertically without overlap on a display screen is called resolution. More precise definition of resolution is no of dots per inch (dpi/pixel per inch) that can be plotted horizontally and vertically.

Display technologies

A. Raster-Scan Display

- The most common type of graphics monitor employing a CRT is the raster-scan display, based on television technology.
- In raster-scan the electron beam is swept across the screen, one row at a time from top to bottom. No of scan line per second is called horizontal scan rate.
- As electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.
- **<u>Picture definition</u>** is stored in a memory called **frame buffer or refresh buffer**. Frame buffer holds all the intensity value for screen points.
- Stored intensity values are then retrieved from the frame buffer and “painted” on the screen one row (scan line) at a time.
- Each screen point is referred to as a **pixel** or **pel** (picture element).
- Availability of frame buffer makes raster-scan display well suited for the realistic display.
- Example: Monitors, Home television, printers.

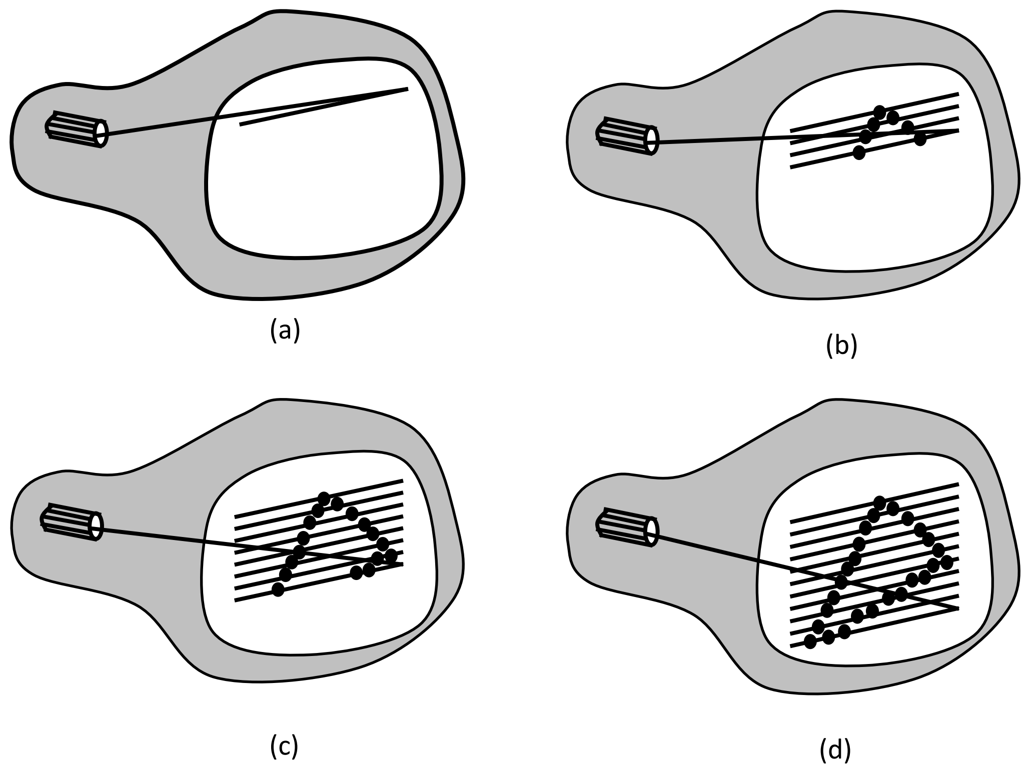


Figure: A raster-scan system displays an object as a set of points across each screen scan line

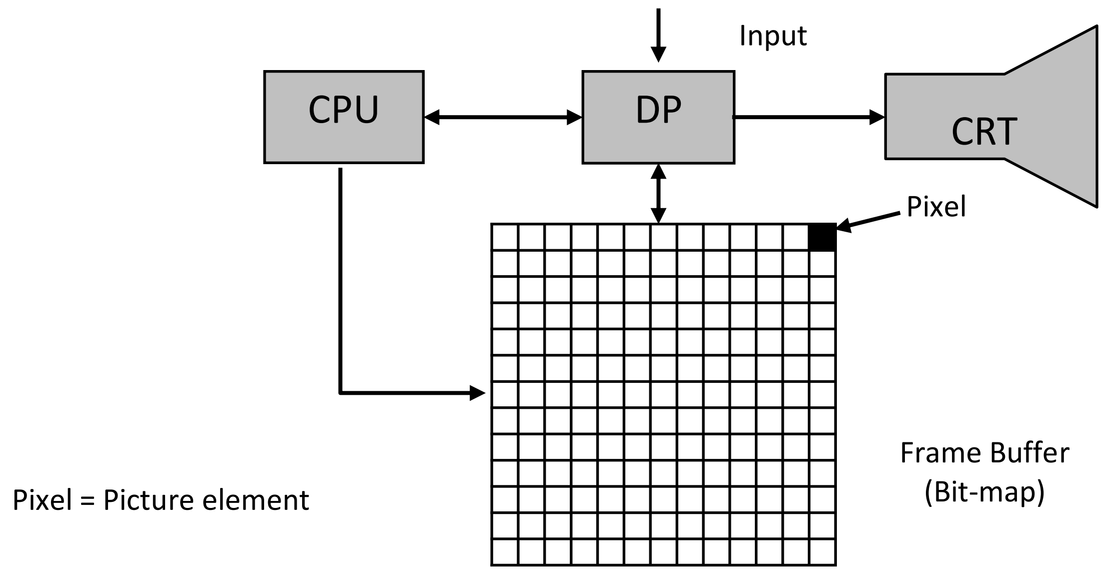


Figure: Raster Scan display system

- Intensity range for pixel position depends on capability of raster system. For **B/W system** each point on screen is either on or off, so only one bit per pixel is needed to control the pixel intensity. **To display color** with varying intensity level, additional bits are needed. Up to 24 to 32 bit per pixel are included in high quality systems, which require more space of storage for the frame buffer, depending upon the resolution of the system.
- A system with 24 bit pixel and screen resolution 1024 $\times$ 1024 require 3 megabyte of storage in frame buffer.

  1024*1024 pixels = 1024*1024*24 bits = 3 MB (using 24-bit per pixel)

- The frame butter in B/W system stores a pixel with one bit per pixel so it is termed as **bitmap**. The frame buffer in multi bit per pixel storage is called **pixmap**.
- Refreshing on Raster-Scan display is carried out at the rate of 60 or higher frames per second. Sometimes refresh rates are described in units of cycles per second or hertz (Hz), where cycle corresponds to one frame.
- Returning of electron beam from right end to left end after refreshing each scan line is called **horizontal retrace** of electron beam. At the end of each frame, the electron beam returns to the top left corner to begin next frame called **vertical retrace**.

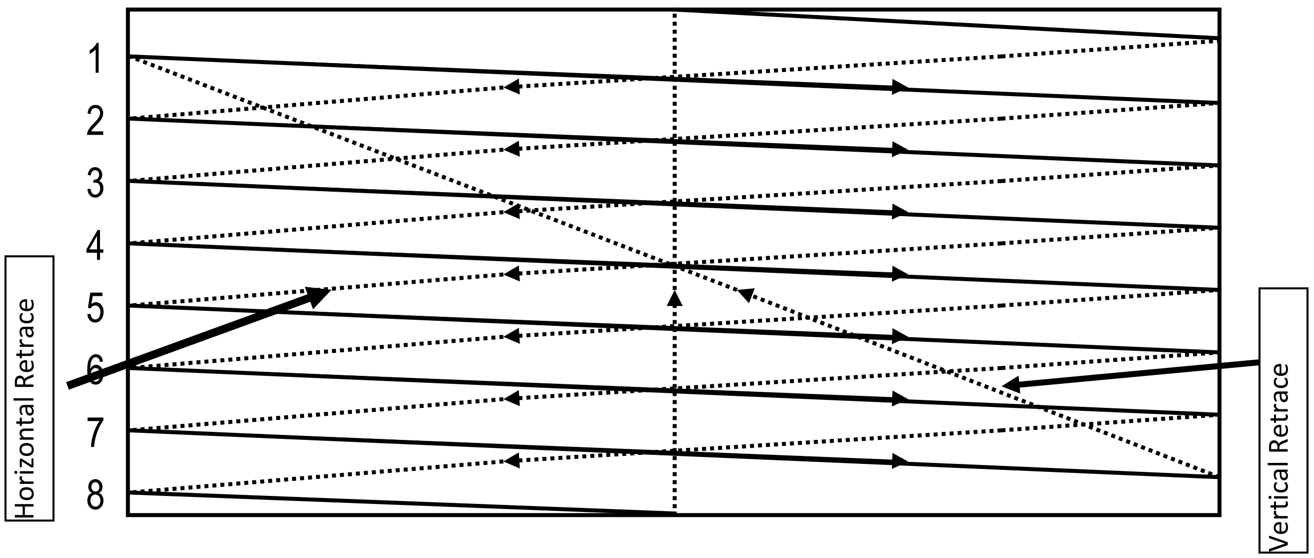


Figure: Horizontal retrace and Vertical retrace

Interlaced vs. **non-interlaced** scan (refresh procedure)

- In interlaced scan, each frame is displayed in two passes. First pass for odd scan lines and another for even ones.
- In non-interlaced refresh procedure, electron beam sweeps over entire scan lines in an frame from top to bottom in one pass.

***Question**: Consider a RGB raster system is to be designed using 8 inch by 10 inch screen with a resolution of 100 pixels per inch in each direction. If we want to store 8 bits per pixel in the frame buffer, how much storage (in bytes) do we need for the frame buffer?*

Solution: Size of screen = 8 inch $\times$ 10 inch.
Pixel per inch (Resolution) = 100.
Then, Total no of pixels = $(8\times100) \times (10 \times 100)$ pixels = (800×1000) pixels
Per pixel storage = 8 bits

Therefore, Total storage required in frame buffer = $(800 \times 1000\times8)$ bits
= $(800 \times1000\times8)/8$ Bytes
= 800000 Bytes

B. Random scan (Vector) display

- In random scan system, the CRT has the electron beam that is directed only to the parts of the screen where the picture is to be drawn. It draws a picture one line at a time, so it is also called **vector display** (or stroke writing or calligraphic display). The component lines of a picture are drawn and refreshed by random scan system in any specified order.

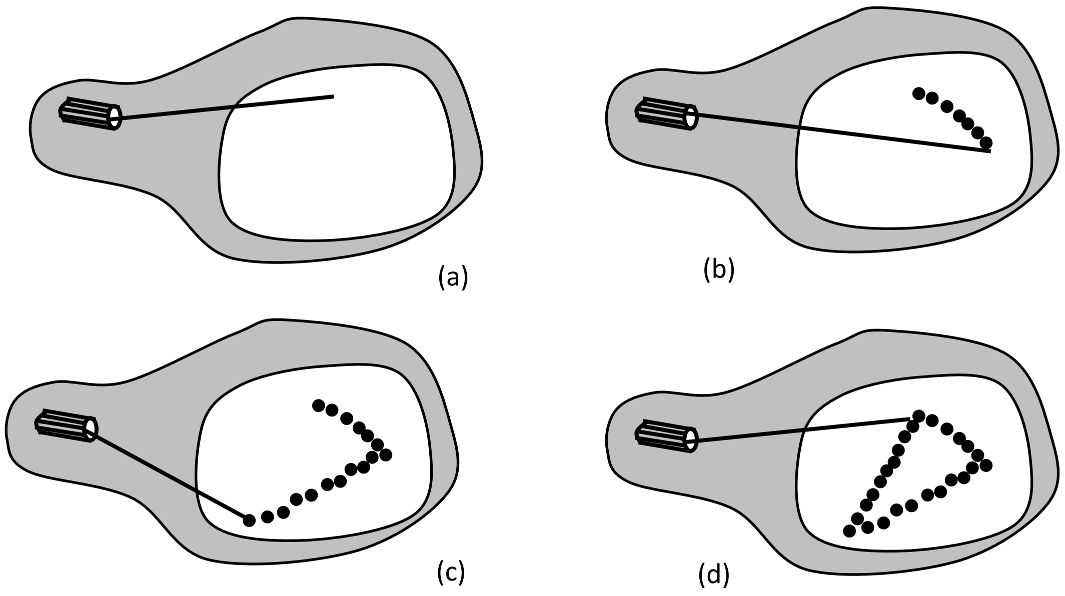


Figure: Random Scan Display

- The refresh rate of vector display depends upon the no of lines to be displayed for any image.
- **Picture definition** is stored as a set of line drawing instructions in an area of memory called the **refresh display file** (Display list or display file).
- To display a picture, the system cycles through the set of commands (line drawing) in the display file. After all commands have been processed, the system cycles back to the first line command in the list.
- Random scan systems are designed for drawing all component lines 30 to 60 times per second. Such systems are designed for line-drawing applications and can not display realistic shaded scenes. Since CRT beam directly follows the line path, the vector display system produce smooth line.

C. Color CRT

A CRT monitor displays color pictures by using a combination of phosphors that emit different-colored light. By combining the emitted light from the different phosphors, a range of colors can be generated. Two basic techniques for producing color displays with CRT are:

1. Beam-penetration method
2. Shadow-mask method

<u>Beam Penetration method</u>

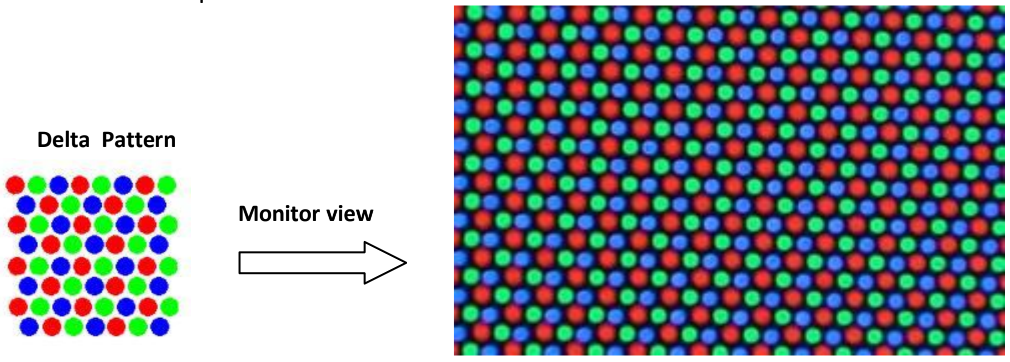
This method is commonly used for random scan display or vector display. In random scan display CRT, the two layers of phosphor usually red and green are coated on CRT screen. Display color depends upon how far electrons beam penetrate the phosphor layers.

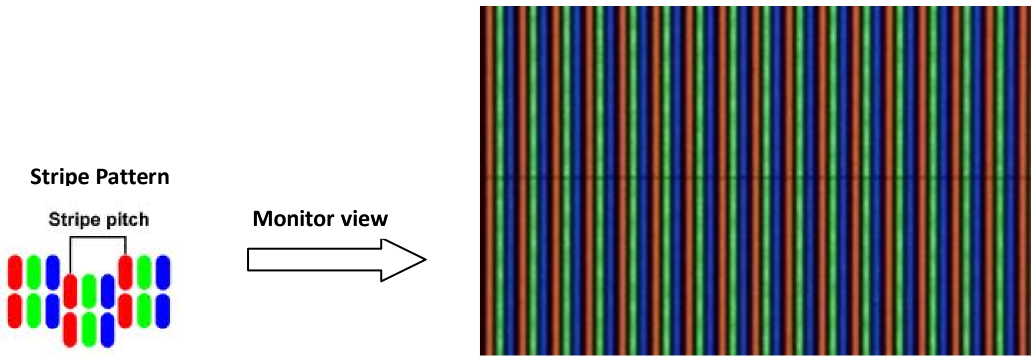
- **Slow electrons** excite only red layer so that we can see red color displayed on the screen pixel where the beam strikes.
- **Fast electrons** beam excite green layer penetrating the red layer and we can see the green color displayed at the corresponding position.
- At **Intermediate** beam speeds, combinations of red and green light are emitted to show two additional colors - orange and yellow.
- The speed of the electrons and hence the screen color at any point, is controlled by the beam-acceleration voltage.
- Beam-penetration has an inexpensive way to produce color in random-scan monitors, but quality of pictures is not as good as other methods since only 4 colors are possible.

<u>Shadow Mask Method</u>

Shadow mask method is used for raster-scan systems because they can produce wide range of colors than beam-penetration method. In shadow mask CRT has three phosphor color dots at each pixel position. The phosphor on the face of the screen is laid out in a precise geometric pattern. There are two primary variations.

1. The stripe pattern of inline tube
2. The delta pattern of delta tube





- In color CRT, there are **three electron guns**, one for each red, green and blue color. In phosphor coating there may be either strips one for each primary color, for a single pixel or there may be three dots one for each pixel in delta fashion.
- Special metal plate called a **shadow mask** is placed just behind the phosphor coating to cover front face.
- The mask is aligned so that it simultaneously allow each electron beam to see only the phosphor of its assigned color and block the phosphor of other two color.

Depending on the pattern of coating of phosphor, two types of raster scan color CRT are commonly used using shadow mask method.

a) Delta-Delta CRT

In delta-delta CRT, three electron beams one for each R, G, and B colors are deflected and focused as a group onto shadow mask, which contains a series of holes aligned with the phosphor dots.

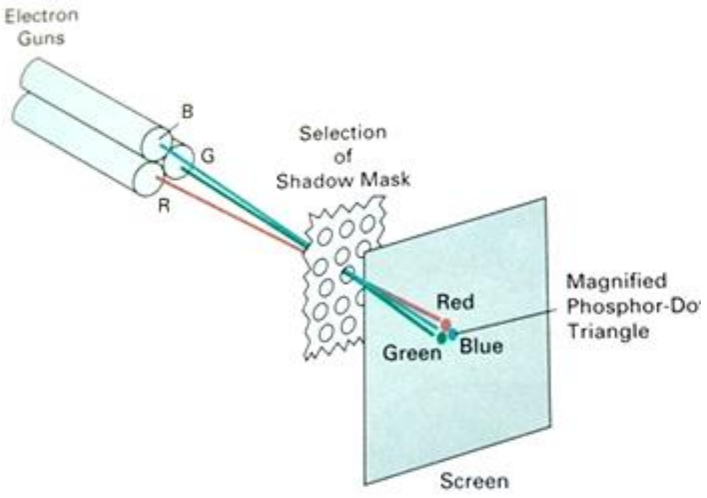


Figure: Shadow mask in Delta-Delta CRT

- Inner side of viewing has several groups of closely spaced red ,green and blue phosphor dot called triad in delta fashion.
- Thin metal plate adjusted with many holes near to inner surface called shadow mask which is mounted in such a way that each hole aligned with respective triad.
- Triads are so small that is perceived as a mixture of colors. When three beams pass through a hole in shadow mask, they activate the dot triangle to illuminate a small spot colored on the screen.
- The color variation in shadow mask CRT can be obtained by varying the intensity level of the three electron guns.

Drawback: Difficulties for the alignment of shadow mask whole and respective triads.

b) Precision inline CRT

This CRT uses strips pattern instead of delta pattern. Three strips one for each R, G, and B colors are used for a single pixel along a scan line so called **inline**. This eliminates the drawbacks of delta-delta CRT at the cost of slight reduction of image sharpness at the edge of the tube.

- Normally 1000 scan lines are displayed in this method. Three beams simultaneously expose three inline phosphor dots along scan line.

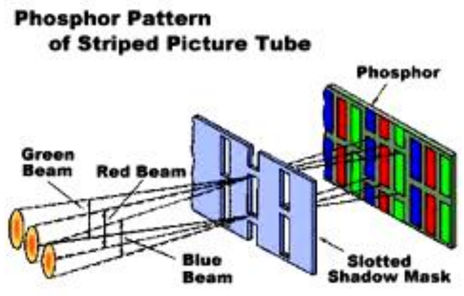


Figure: Inline CRT

D. Direct-view Storage tubes (DVST)

This is alternative method for method maintaining a screen image to store picture information inside the CRT **instead of refreshing** the system.

- DVST stores the picture information as a charge distribution just behind the phosphor-coated screen.
- Two electron guns used: primary gun – to store picture pattern and flood gun – maintains the picture display.
- Pros: Since no refreshing is needed complex pictures can be displayed in high-resolution without flicker.
- Cons: Ordinarily do not display color and that selected parts of picture can not be erased. To eliminate a picture section, entire screen must be erased and modified picture redrawn, which may take several seconds for complex picture.

E. Flat panel Displays

Flat-panel display refers to a class of video devices that have reduced volume (thinner), weight and power consumption compared to CRT. These emerging display technologies tend to replace CRT monitors. Current uses of flat-panel displays include TV monitors, calculators, pocket video games, laptops, displays in airlines and ads etc.

Two categories of flat-panel displays:

a) Emissive displays: convert electrical energy into light. Example: Plasma panels, electroluminescent displays and light-emitting diodes.
b) Non-emissive displays: use optical effects to convert sunlight or light from other sources into graphics patterns. Example: liquid-crystal displays.

Hey! For details of flat displays, read page no. 65-67 of book “Computer graphics C version”, Hearn & Baker.

Architecture of Raster-Scan System

The raster graphics systems typically consist of several processing units. CPU is the main processing unit of computer systems. Besides CPU, graphics system consists of a special purpose processor called video controller or display processor (DP). The display processor controls the operation of the display device. The organization of raster system is as shown below:

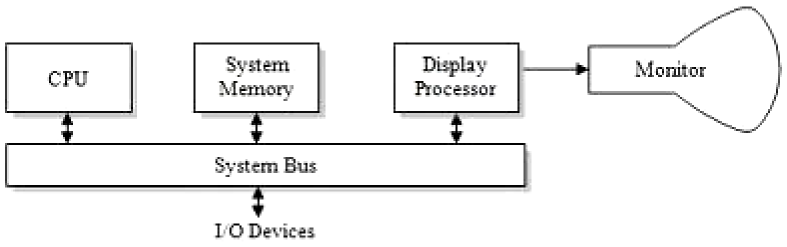


Fig: Architecture of simple raster-graphics system

- A fixed area of system memory is reserved for the frame buffer. The video controller has the direct access to the frame buffer for refreshing the screen.
- The video controller cycles through the frame buffer, one scan line at a time, typically at 60 times per second or higher. The contents of frame buffer are used to control the CRT beam's intensity or color.

The video controller

The video controller is organized as in figure below. The raster-scan generator produces deflection signals that generate the raster scan and also controls the X and Y address registers, which in turn defines memory location to be accessed next. Assume that the frame buffer is addressed in X from 0 to X_{max} and in Y from 0 to Y_{max} then, at the start of each refresh cycle, X address register is set to 0 and Y register is set to 0 (top scan line).

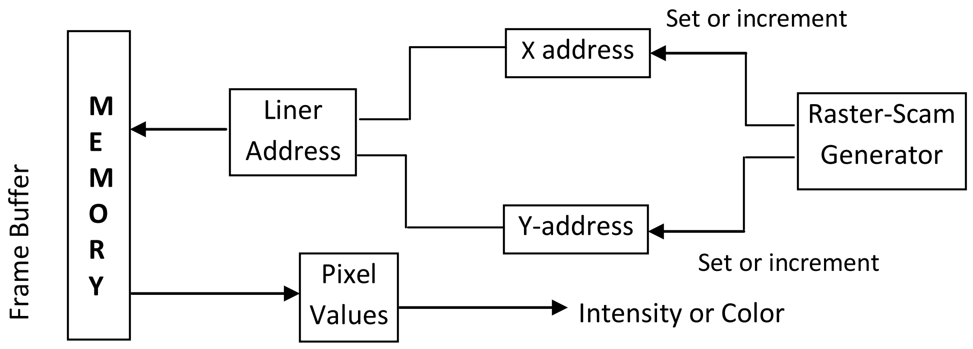


Fig: Basic video-controller refresh-operation

As first scan line is generated, the X address is incremented up to X_{max}. Each pixel value is fetched and used to control the intensity of CRT beam. After first scan line, X address is reset to 0 and Y address is incremented by 1. The process is continued until the last scan line ($Y=Y_{max}$) is generated.

Raster-Scan Display Processor

The raster scan with a peripheral display processor is a common architecture that avoids the disadvantage of simple raster scan system. It includes a separate graphics processor to perform graphics functions such as scan conversion and raster operation and a separate frame buffer for image refresh. The display processor has its own separate memory called display processor memory.

- System memory holds data and those programs that execute on the CPU, and the application program, graphics packages and OS.
- The display processor memory holds data plus the program that perform scan conversion and raster operations.
- The frame buffer stores displayable image created by scan conversion and raster operations.

The organization is given below in figure:

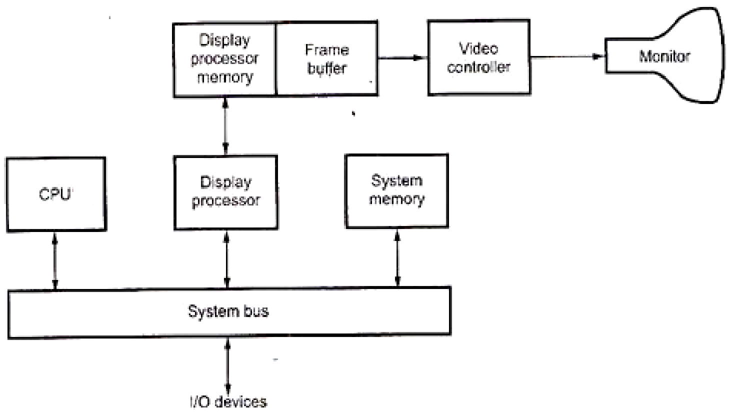


Fig: Architecture of a raster-graphics system with a display processor

Architecture of Random-scan (Vector) Systems

The organization of simple vector system shown in the figure below:

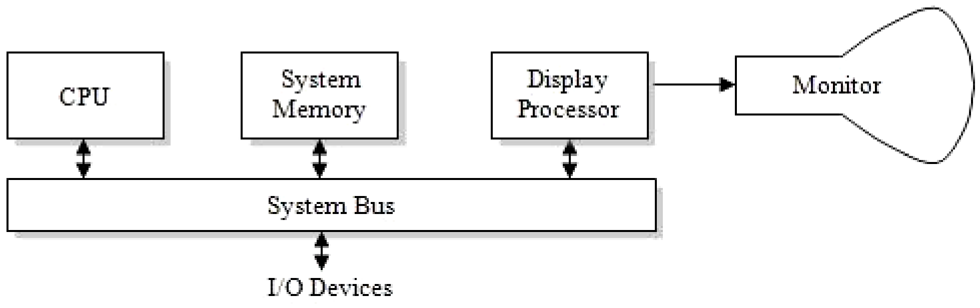


Fig: Architecture of Vector Display System

- Vector display system consists of several units along with peripheral devices. The display processor is also called as graphics controller.
- Graphics package creates a display list and stores in systems memory (consists of points and line drawing commands) called display list or display file.
- Vector display technology is used in monochromatic or beam penetration color CRT.
- Graphics are drawn on a vector display system by directing the electron beam along component line.

Advantages:

- Can produce output with high resolutions.
- Better for animation than raster system since only end point information is needed.

Disadvantages:

- Cannot fill area with pattern and manipulate bits.
- Refreshing image depends upon its complexity.

Hey! For the knowledge of various **input and hard-copy devices** employed in computer graphics, plz read the section through page no. 80-94 of book “Computer graphics C version”, Hearn & Baker.

Output primitives

Output primitives are the geometric structures such as straight line segments (pixel array) and polygon color areas, used to describe the shapes and colors of the objects. Points and straight line segments are the simplest geometric components of pictures. Additional output primitive includes: circles and other conic sections, quadric surfaces, spline curves and surfaces, polygon color areas and character strings. Here, we discuss picture generation algorithm by examining device-level algorithms for displaying two-dimensional output primitives, with emphasis on **scan-conversion methods** for raster graphics system.

Points and Lines

- Point plotting is done in CRT monitor by turning on the electron beam to illuminate at the screen phosphor at the selected location.
    - Random-scan systems: stores point plotting instructions in the display list and co-ordinate values in these instructions are converted into deflection voltages that position the electron beam at selected location.
    - B/W raster system: With in frame buffer, bit value is set to 1 for specified screen position. Electron beam then sweeps across each horizontal scan line, it emits a burst of electrons (plots a point) whenever value of 1 is encountered in the frame buffer.
    - RGB raster system: Frame buffer is loaded with the color codes for the intensities that are to be displayed at the screen pixel positions.
- Line drawing is accomplished by calculating intermediate positions along the line path between two specified endpoint positions. An output device is then directed to fill in these positions between the endpoints.
    - For analog devices (vector-pen plotter and random-scan display), a straight line can be drawn smoothly between two points. [**Reason**: linearly varying horizontal and vertical deflection voltages are generated that are proportional to the required changes in the x and y directions]
    - Digital devices display a straight line segment by plotting discrete points between two end-points. Discrete integer coordinates are calculated from the equation of the line. Since **rounding of coordinate values** occur [viz. (4.48, 48.51) would be converted to (4, 49)], line is displayed with stairstep appearance.

<u>Line Drawing Algorithms</u>

The Cartesian slope-intercept equation of a straight line is:

$$y = mx + b \quad \text{.........................} \ (1)$$

Where m = slope of line and b = y-intercept.

For any two given points (x_1, y_1) and (x_2, y_2)

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$\therefore$ (1) becomes,

$$b = y - \frac{y_2 - y_1}{x_2 - x_1} x$$

At any point (x_k, y_k) ,

$$y_k = mx_k + b \quad \text{.........................} \ (2)$$

At (x_{k+1}, y_{k+1}) ,

$y_{k+1} = mx_{k+1} + b$ ……………………… (3)

Subtracting (2) from (3) we get,

$y_{k+1} - y_k = m(x_{k+1} - x_k)$

Here ($y_{k+1} - y_k$) is increment in y as corresponding increment in x.

$\therefore \Delta y = m.\Delta x$

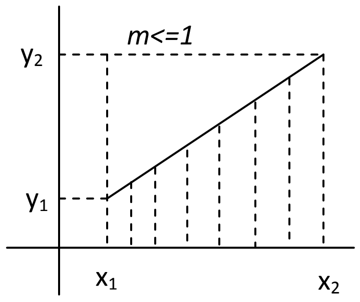
or $m = \dfrac{\Delta y}{\Delta x}$

<u>DDA line Algorithm (Incremental algorithm)</u>

The digital differential analyzer (DDA) is a scan conversion line drawing algorithm based on calculating either Δx or Δy form the equation,

$$\Delta y = m.\Delta x$$

We sample the line at unit intervals in one co-ordinate and determine the corresponding integer values nearest to the line path for the other co-ordinates.

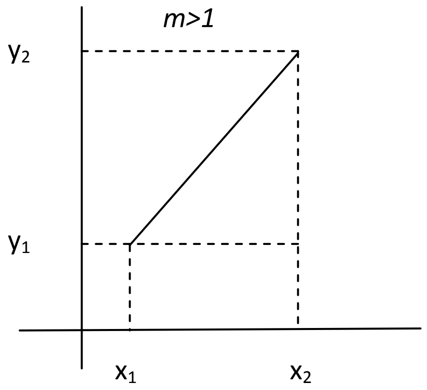


Consider first the line with positive slope.

If m<=1, we sample x co-ordinate. So $\Delta x = 1$ and compute each successive *y* value as:

$$y_{k+1} = y_k + m \quad \because m = \frac{\Delta y}{\Delta x}, \Delta x = 1$$

Here *k* takes value from starting point and increase by 1 until final end point. *m* can be any real value between 0 and 1.



For line with positive slope greater than 1, we sample $\Delta y = 1$ and calculate corresponding x values as

$$x_{k+1} = x_k + \frac{1}{m} \quad \because m = \frac{\Delta y}{\Delta x}, \Delta y = 1$$

The above equations are under the assumption that the lines are processed from leπ τo right i.e. left end point is starting. If the processing is from right to left, we can sample $\Delta y = -1$ for line |m|<1

$$\therefore y_{k+1} = y_k - m$$

If |m|>1, $\Delta y = -1$ and calculate

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

Problem: Floating point multiplication & addition

C function for DDA algorithm

```
void lineDDA (in x1, int y1, int x2, int y2)
{
        int dx, dy, steps, k;
        float incrx, incry, x,y;
        dx = x2-x1;
        dy = y2-y1;
        if (abs(dx) > abs(dy))
                steps = abs(dx);
        else
                steps = abs(dy);
        incrx = dx/steps;
        incry = dy/steps;
        x = x1;    /* first point to plot */
        y = y1;
        putpixel(round(x), round(y),1);         //1 is color parameter
        for (k = 1;k <= steps;k++)
        {
                x = x + incrx;
                y = y + incry;
                putpixel(round(x),round(y),1);
        }
}
```

The DDA algorithm is faster method for calculating pixel position but it has problems:

- *m* is stored in floating point number.
- round of error
- Error accumulates as we precede line.
- so line will move away from actual line path for long line

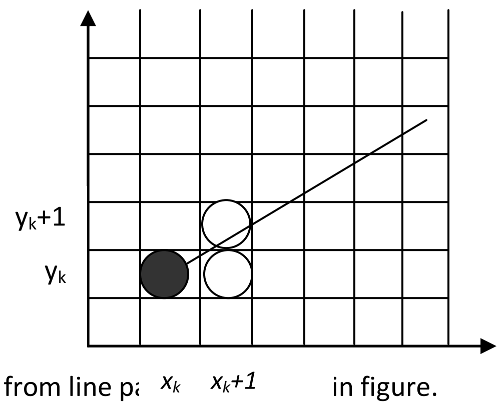
Bresenham's Line algorithm

An accurate and efficient line generating algorithm, developed by Bresenham that scan converts lines only using integer calculation to find the next *(x, y)* position to plot. It avoids incremental error accumulation.

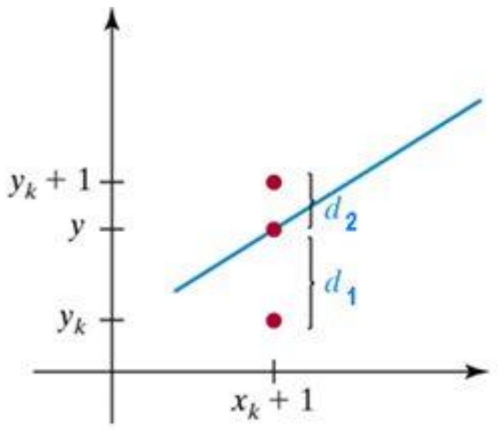
Line with positive slope less than 1 (0<m<1)

Pixel position along the line path is determined by sampling at unit x intervals. Starting from left end point, we step to each successive column and plot the pixel closest to line path.

Assume that (x_k, y_k) is pixel at kth step then next point to plot may be either (x_k+1, y_k) or (x_k+1, y_k+1).



At sampling position x_k+1, we label vertical pixel separation from line p in figure.



The y-coordinate on the mathematical line path at pixel column x_k+1 is $y = m(x_k+1)+b$.

Then $d_1 = y - y_k = m(x_k + 1) + b - y_k$
$d_2 = (y_k + 1) - y = (y_k + 1) - m(x_k + 1) - b$

Now $d_1 - d_2 = 2m(x_k + 1) - (y_k + 1) - y_k + 2b = 2m(x_k + 1) - 2y_k + 2b - 1$
A decision parameter p_k for the kth step in the line algorithm can be obtained by rearranging above equation so that it involves only integer calculations. We accomplish this by substituting $m = \frac{\Delta y}{\Delta x}$ in above eqn and defining

$$p_k = \Delta x(d_1 - d_2) = \Delta x[2\frac{\Delta y}{\Delta x}(x_k + 1) - 2y_k + 2b - 1] = 2\Delta y.x_k - 2\Delta x.y_k + c$$

Where the constant $c = 2\Delta y - \Delta x(2b - 1)$ which is independent of the pixel position. Also, sign of p_k is same as the sign of d_1-d_2.

If decision parameter p_k is negative i.e. $d_1<d_2$, pixel at y_k is closer to the line path than pixel at y_k+1. In this case we plot lower pixel *(x_k+1, y_k), other wise plot upper pixel (x_k+1, y_k+1).*

Co-ordinate change along the line occur in unit steps in either x, or y direction. Therefore we can obtain the values of successive decision parameters using incremental integer calculations.
At step *k+1,* decision parameter p_{k+1} *is* evaluated as.

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

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

Since $x_{k+1} = x_k + 1$

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

The term $y_{k+1} - y_k$ is either 0 or 1 depending upon the sign of p_k.

The first decision parameter p_0 is evaluated as.

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

and successively we can calculate decision parameter as

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

So if p_k is negative, $y_{k+1} = y_k$ so $p_{k+1} = p_k + 2\Delta y$

Otherwise *y_{k+1} = y_k+1, then p_{k+1} = p_k+ 2* Δy -2 Δx

<u>Algorithm:</u>

1. Input the two line endpoint and store the left endpoint at (x_o, y_o)
2. Load (x_o, y_o) in to frame buffer, i.e. Plot the first point.
3. Calculate constants $2\Delta x, 2\Delta y$ calculating $\Delta x, \Delta y$ and obtain first decision parameter value as

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

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

   if $p_k < 0,$ next point is $(x_k + 1, y_k)$

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

   otherwise

   next point to plot is $(x_k + 1, y_k + 1)$

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

5. Repeat step 4 Δx times.

<u>C Implementation</u>

```
void lineBresenham (int x1, int y1, int x2, int y2){
        int dx = abs(x2-x1), dy=abs(y2-y1);
        int  pk, xEnd;
        pk=2*dy-dx;
        //determine which point to use as start, which as end
        if(x1>x2){
                x = x2;
                y = y2;
                xEnd = x1;
        }
         else {
                x = x1;
                y = y1;
                xEnd = x2;
        }
        putixel (x,y,1);
        while (x < xEnd)
        {
                x++;
                if(pk<0)
                        pk=pk+2*dy;
                else
                {
                        Y++;
                        pk= pk+2*dy-2*dx
                 }
                putpixel (x,y,1);
         }
}
```

Bresenham's algorithm is generalized to lines with **arbitrary slope** by considering the symmetry between the various octants & quadrants of xy-plane.

Line with positive slope greater than 1 (m>1)

Here, we simply interchange the role of x & y in the above procedure i.e. we step along the y-direction in unit steps and calculate successive x values nearest the line path.

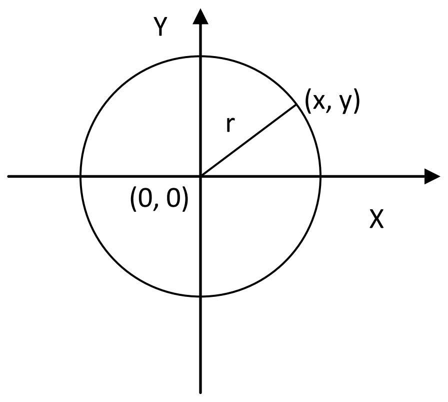
Circle generating algorithms

Circle is a frequently used component in pictures and graphs, a procedure for generating circular arcs or full circles is included in most graphics packages.

Simple Algorithm

The equation of circle centered at origin and radius r is given by $x^2 + y^2 = r^2$

$\Rightarrow y = \pm\sqrt{r^2 - x^2}$



- Increment x in unit steps and determine corresponding value of y from the equation above. Then set pixel at position *(x,y).*
- The steps are taken from *–r* to *+r*.
- In computer graphics, we take origin at upper left corner point on the display screen i.e. first pixel of the screen. So any visible circle drawn would be centered at point other than (0,0). If center of circle is *(xc, yc)* then the calculated points from origin center should be moved to pixel position by *(x+xc, y+yc)*.

In general the equation of circle centered at *(xc, yc)* and radius r is

$(x - xc)^2 + (y - yc)^2 = r^2$

$\Rightarrow y = yc \pm \sqrt{r^2 - (x - xc^2}$ .... (1)

We use this equation to calculate the position of points on the circle. Take unit step from xc-r to xc+r for x value and calculate the corresponding value of y-position for pixel position (x, y). This algorithm is simple but,

- Time consuming – square root and squares computations.
- Non-uniform spacing, due to changing slope of curve. If non-uniform spacing is avoided by interchanging x and y for slope |m|>y, this leads to more computation.

Following program demonstrates the simple computation of circle using the above equation (1)

```
//program for circle (simple algorithm)
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<graphics.h>
#define SQUARE(x) ((x)*(x))
void drawcircle(int ,int,int);
void main()
```

```
{
        int gd,gm,err;
        int xc,yc,r;
        gd=DETECT;
        initgraph(&gd,&gm,"\\tc\\bgi");
        err=graphresult();
        if(err!=0)
        {
                printf("ERROR:%s",grapherrormsg(err));
                printf("\nPress a key..");
                getch();
                exit(1);
        }
        xc=getmaxx()/2;
        yc=getmaxy()/2;
        r=50;
        drawcircle(xc,yc,r);
        getch();
        closegraph();
}//end main
void drawcircle(int xc,int yc,int r)
{
        int i,x,y,y1;
        for(i=xc-r;i<=xc+r;i++)
        {
                x=i;
                y=yc+sqrt(SQUARE(r)-SQUARE(x-xc));
                y1=yc-sqrt(SQUARE(r)-SQUARE(x-xc));
                putpixel(x,y,1);
                putpixel(x,y1,1);
        }
```

<u>Drawing circle using polar equations</u>

If (x,y) be any point on the circle boundary with center (0,0) and radius r, then

$$x = r\cos\theta$$
$$y = r\sin\theta$$

i.e. $(x, y) = (r\cos\theta, r\sin\theta)$

To draw circle using these co-ordinates approach, just increment angle starting from 0 to 360. Compute (x,y) position corresponding to increment angle. Which draws circle centered at origin, but the circle centered at origin is not visible completely on the screen since (0, 0) is the starting pixel of the screen. If center of circle is given by (xc, yc) then the pixel position (x, y) on the circle path will be computed as

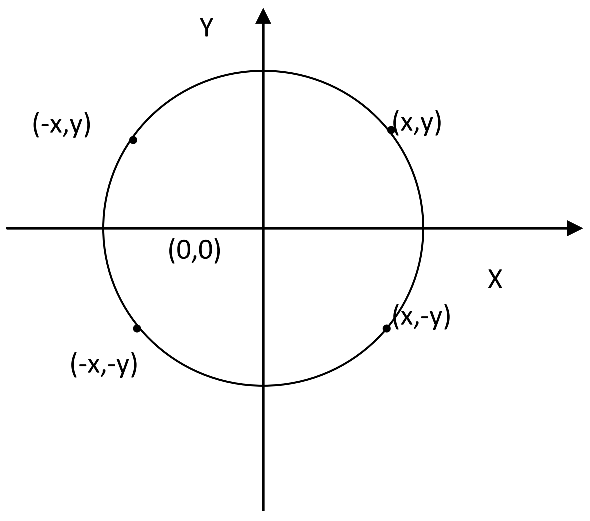
$$x = xc + r\cos\theta$$
$$y = yc + r\sin\theta$$

polarcircle() Function to draw circle using the polar transformation:

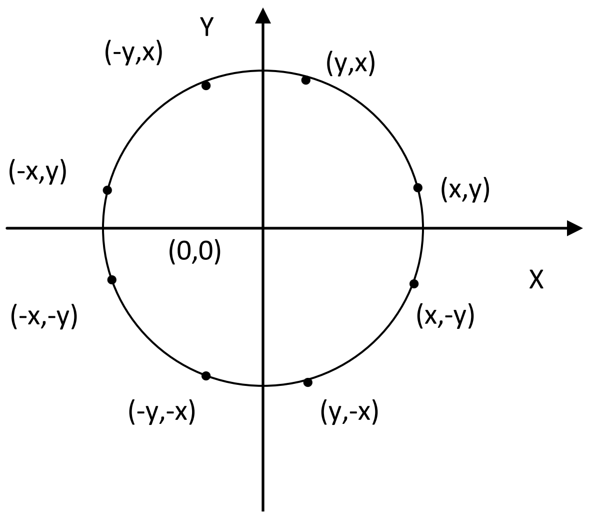
```
void polarcircle(int xc, int yc, int r)
{
        int x,y;
        float theta;
        const float PI=3.14;

        for(theta=0.0;theta<=360;theta+=1)
        {
                x= xc+r*cos(theta*PI/180.0);
                y= yc+r*sin(theta*PI/180.0);
                putpixel(x,y,1);

        }
}
```

Symmetry in circle scan conversion

We can reduce the time required for circle generation by using the symmetries in a circle e.g. 4-way or 8-way symmetry. So we only need to generate the points for one quadrant or octants and then use the symmetry to determine all the other points.



4-way symmetry



8-way symmetry

Problem of computation still persists using symmetry since there are square roots; trigonometric functions are still not eliminated in above algorithms.

Mid point circle Algorithm

In mid point circle algorithm, we sample 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 (xc, yc), we can first set up our algorithm to calculate pixel positions around a circle path centered at (0, 0) and then each calculated pixel position (x, y) is moved to its proper position by adding xc to x and yc to y

i.e. x = x + xc, y = y + yc.

To apply the mid point method, we define a circle function as:

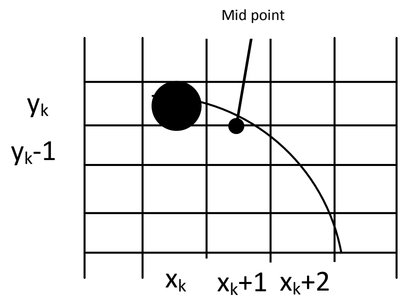
$$f_{circle} = x^2 + y^2 - r^2$$

To summarize the relative position of point (x, y) by checking sign of f_{circle} function,

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

The circle function tests are performed for the mid positions between pixels near the circle path at each sampling step. Thus the circle function is decision parameter in mid point algorithm and we can set up incremental calculations for this functions as we did in the line algorithm.

The figure, shows the mid point between the two candidate pixel at sampling position $x_k + 1$, Assuming we have just plotted the pixel (x_k, y_k), we next need to determine whether the pixel at position $(x_k + 1, y_k) or (x_k + 1, y_k - 1)$ is closer to the circle.



Our decision parameter is circle function evaluated at the mid point

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

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

If $p_k < 0$, then mid-point lies inside the circle, so point at y_k is closer to boundary otherwise, $y_k - 1$ closer to choose next pixel position.

Successive decision parameters are obtained by incremental calculation. The decision parameter for next position is calculated by evaluating circle function at sampling position $x_{k+1} + 1$ i.e. $x_k + 2$ as

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

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

$$= (x_{k+1})^2 + 2x_{k+1} + 1 + (y_{k+1})^2 - (y_{k+1}) + \frac{1}{4} - r^2$$

Now, $p_{k+1} - p_k = 2x_{k+1} + ({y^2}_{k+1} - {y_k}^2) - (y_{k+1} - y_k) + 1$

i.e. $p_{k+1} = p_k + 2x_{k+1} + ({y^2}_{k+1} - {y_k}^2) - (y_{k+1} - y_k) + 1$

Where y_{k+1} is either y_k or $y_k - 1$ depending upon sign of p_k. and $x_{k+1} = x_k + 1$

If p_k is negative, $y_{k+1} = y_k$ so we get,

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

If p_k is positive, $y_{k+1} = y_k - 1$ so we get,

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

Where $2x_{k+1} = 2x_k + 2$
$2y_{k+1} = 2y_k - 2$

At the start position, (0, r), these two terms have the values 0 and 2r, respectively. Each successive values are obtained by adding 2 to the previous value of 2x and subtracting 2 from previous value of 2y.

The initial decision parameter is obtained by evaluating the circle function at starting position $(x_0, y_0) = (0, r)$.

$$p_0 = f_{circle}(1, r - 1/2)$$
$$= 1 + (r - 1/2)^2 - r^2$$
$$= 1 + r^2 - r + \frac{1}{4} - r^2$$
$$= \frac{5}{4} - r$$

If p_0 is specified in integer,

$$p_0 = 1 - r.$$

<u>Steps of Mid-point circle algorithm</u>

1. Input radius r and circle centre (x_c, y_c) and obtain the first point on circle centered at origin as.
   $$(x_0, y_0) = (0, r).$$
2. Calculate initial decision parameter
   $$p_0 = 5/4 - r$$
3. At each x_k position, starting at $k = 0$, perform the tests:
   If $p_k < 0$ next point along the circle centre at $(0,0)$ is $(x_k + 1, y_k)$
   $$p_{k+1} = p_k + 2x_{k+1} + 1)$$
   Otherwise, the next point along circle is $(x_k + 1, y_k - 1)$
   $$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$
   Where $2x_{k+1} = 2x_k + 2.$ and $2y_{k+1} = 2y_k - 2.$
4. Determine symmetry point on the other seven octants.

5. Move each calculated pixels positions (x, y) in to circle path centered at (x_c, y_c) as
   $$x = x + x_c, y = y + y_c$$
6. Repeat 3 through 5 until $x \geq y$.

```
//mid point circle algorithm

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
void drawpoints(int,int,int,int);
void drawcircle(int,int,int);

void main(void)
{
  /* request auto detection */
  int gdriver = DETECT, gmode, errorcode;
  int xc,yc,r;
 /* initialize graphics and local
    variables */
  initgraph(&gdriver, &gmode, "\\tc\\bgi");

  /* read result of initialization */
  errorcode = graphresult();
  if (errorcode != grOk)  /* an error
     occurred */
  {
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1); /* terminate with an error
                    code */
  }
 printf("Enter the center co-ordinates:");
 scanf("%d%d",&xc,&yc);
 printf("Enter the radius");
 scanf("%d",&r);
 circlemidpoint(xc,yc,r);
 getch();
 closegraph();
}
void drawpoints(int x,int y, int xc,int yc)
{
        putpixel(xc+x,yc+y,1);
        putpixel(xc-x,yc+y,1);
        putpixel(xc+x,yc-y,1);
        putpixel(xc-x,yc-y,1);
```

```
            putpixel(xc+y,yc+x,1);
            putpixel(xc-y,yc+x,1);
            putpixel(xc+y,yc-x,1);
            putpixel(xc-y,yc-x,1);
}
void circlemidpoint(int xc,int yc,int r)
{
            int x = 0, y=r, p = 1-r;
            drawpoints(x,y,xc,yc);
            while(x<y)
            {
                        X++;
                        if(p<0)
                                    p += 2*x+1;
                        else
                        {
                                    y--;
                                    p += 2*(x-y)+1;
                        }
                        drawpoints(x,y,xc,yc);
            }
}
```

Ellipse Algorithm generating algorithm

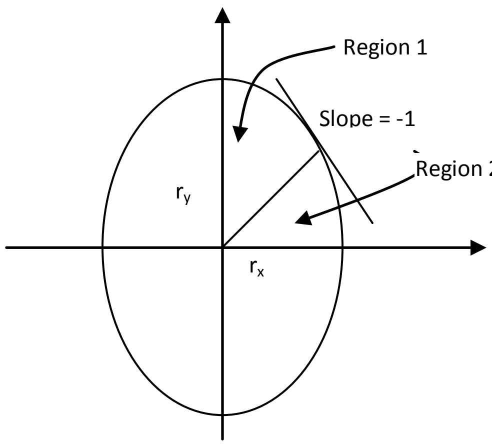
Direct Method

An ellipse is an elongated circle therefore the basic algorithm for drawing ellipse is same as circle computing x and y position at the boundary of the ellipse from the equation of ellipse directly.

We have equation of ellipse centered at origin (0,0) is

$\frac{x^2}{{r_x}^2} + \frac{y^2}{{r_y}^2} = 1$ which gives

$y = \pm \frac{r_y}{r_x} \sqrt{({r_x}^2 - x^2)}$ .... (1)



Stepping unit interval in x direction from $-r_x$ to r_x we can get corresponding y value at each x position which gives the ellipse boundary co-ordinates. Plotting these computed points we can get the ellipse.

If center of ellipse is any arbitrary point *(xc, yc)* then the equation of ellipse can be written as

$$\frac{(x-xc)^2}{r_x^{\ 2}}+\frac{(y-yc)^2}{r_y^{\ 2}}=1$$

i.e. $y = yc \pm \frac{r_y}{r_x}\sqrt{r_x^{\ 2}-(x-xc)^2}$ -------------------(2)

For any point *(x, y)* on the boundary of the ellipse If major axis of ellipse is along X-axis, then algorithm based on the direct computation of ellipse boundary points can be summarized as,

1. Input the center of ellipse *(xc, yc)* , x-radius *xr* and y-radius *yr*.
2. For each x position starting from xc-r and stepping unit interval along x-direction, compute corresponding y positions as
   $y = yc \pm \frac{r_y}{r_x}\sqrt{r_x^{\ 2}-(x-xc)^2}$
3. Plot the point (x, y).
4. Repeat step 2 to 3 until x >= xc+xr.

Computation of ellipse using polar co-ordinates

Using the polar co-ordinates for ellipse, we can compute the (x,y) position of the ellipse boundary using the following parametric equations

$$x = xc + r\cos\theta$$
$$y = yc + r\sin\theta$$

The algorithm based on these parametric equations on polar co-ordinates can be summarized as below.

1. Input center of ellipse *(xc,yc)* and radii xr and yr.
2. Starting θ from angle 0^o step minimum increments and compute boundary point of ellipse as
   $x = xc + r\cos\theta$
   $y = yc + r\sin\theta$
3. Plot the point at position (round(x), round(y))
4. Repeat until θ is greater or equal to 360^o.

C implementation

```
void drawellipse(int xc,int yc,int rx,int ry){
        int x,y;
        float theta;
        const float PI=3.14;
        for(theta=0.0;theta<=360;theta+=1){
                x= xc+rx*cos(theta*PI/180.0);
                y= yc+ry*sin(theta*PI/180.0);
                putpixel(x,y,1);

        }
}
```

The methods of drawing ellipses explained above are not efficient. The method based on direct equation of ellipse must perform the square and square root operations due to which there may be floating point number computation which cause rounding off to plot the pixels. Due to the changing slope of curve along the path of ellipse, there may be un-uniform separation of pixel when slope changes. Although , the method based on polar co-ordinate parametric equation gives the uniform spacing of pixel due to uniform increment of angle but it also take extra computation to evaluate the trigonometric functions. So these algorithms are **not efficient** to construct the ellipse. We have another algorithm called mid-point ellipse algorithm similar to raster mid-point circle algorithm and is **efficient** one.

Mid-Point Ellipse Algorithm

The mid-point ellipse algorithm decides which point near the boundary (i.e. path of the ellipse) is closer to the actual ellipse path described by the ellipse equation. That point is taken as next point.

- This algorithm is applied to the first quadrant in two parts as in fig Region 1 and Region 2. We process by taking unit steps in x-coordinates direction and finding the closest value for y for each x-step in region 1.
- In first quadrant at region 1, we start at position *(0, r_y)* and incrementing x and calculating y closer to the path along clockwise direction. When slope becomes -1 then shift unit step in x to y and compute corresponding x closest to ellipse path at Region 2 in same direction.
- Alternatively, we can start at position *(r_x, 0)* and select point in counterclockwise order shifting unit steps in y to unit step in x when slope becomes greater than -1.

Here, to implement mid-point ellipse algorithm, we take start position at *(0, r_y)* and step along the ellipse path in clockwise position throughout the first quadrant.

We define ellipse function center at origin i.e. *(xc, yc) = (0, 0)* as

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

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

So $f_{ellipse}$ function serves as decision parameter in ellipse algorithm at each sampling position. We select the next pixel position according to the sign of decision parameter.

Starting at (0, r_y) , we take unit step in x-direction until we reach the boundary between the region 1 and region 2. Then we switch unit steps in y over the remainder of the curve in first quadrant. At each step, we need to test the slope of curve. The slope of curve is calculated as;

$$\frac{dy}{dx} = -\frac{2r_y^2 x}{2r_x^2 y}$$

At the boundary between region 1 and region 2,

$\frac{dy}{dx} = -1$ and $2r_y^2 = 2r_x^2$ Therefore , we move out of region 1 when $2r_y^2 x \geq 2r_x^2$

Assuming the position (x_k, y_k) is filled, we move x_{k+1} to determine next pixel. The corresponding y value for x_{k+1} position will be either y_k or $y_k - 1$ depending upon the sign of decision parameter. So the decision parameter for region 1 is tested at mid point of $(x_k + 1, y_k)$ and $(x_k + 1, y_k - 1)$ i.e.

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

or $$p_{1k} = r_y{}^2(x_{k+1})^2 + r_x{}^2(y_k - \frac{1}{2})^2 - r_x{}^2 r_y{}^2$$

or $$p_{1k} = r_y{}^2(x_{k+1})^2 + r_x{}^2 y_k{}^2 - r_x{}^2 y_k + \frac{r_x^2}{4} - r_x{}^2 r_y{}^2 \text{ ....(1)}$$

if $p_{1k} < 0$, the mid point lies inside boundary, so next point to plot is $(x_k + 1, y_k)$ otherwise, next point to plot will be $(x_k + 1, y_k - 1)$

The successive decision parameter is computed as

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

$$= r_y{}^2(x_{k+1} + 1)^2 + r_x{}^2(y_{k+1} - \frac{1}{2})^2 - r_x{}^2 r_y{}^2$$

Or, $p_{1k+1} = r_y{}^2(x_{k+1}^2 + 2x_{k+1} + 1) + r_x{}^2(y_{k+1}^2 - y_{k+1} + \frac{1}{4}) - r_x{}^2 r_y{}^2$

Or, $p_{1k+1} = r_y{}^2 x_{k+1}^2 + 2r_y{}^2 x_{k+1} + r_y^2 + r_x{}^2 y_{k+1}^2 - r_x{}^2 y_{k+1} + \frac{r_x{}^2}{4} - r_x{}^2 r_y{}^2$ ....(2)

Subtracting (2) - (1)

$$p_{1k+1} - p_{1k} = 2r_y{}^2 x_{k+1} + r_y^2 + r_x{}^2(y_{k+1}^2 - y_k^2) - r_x^2(y_{k+1} - y_k)$$

if $p_{1k} < 0$, $y_{k+1} = y_k$ then,

$$\therefore p_{1k+1} = p_{1k} + 2r_y{}^2 x_{k+1} + r_y{}^2$$

Otherwise $y_{k+1} = y_k - 1$ then we get,

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

At the initial position, (0, r$_y$) $2r_y{}^2 x = 0$ and $2r_x{}^2 y = 2r_x^2 r_y$

In region 1, initial decision parameter is obtained by evaluating ellipse function at (0, r$_y$) as

$$p_{10} = f_{ellipse}(1, r_y - \frac{1}{2})$$

Or, $p_{10} = f_{ellipse}(1, r_y - \frac{1}{2})$

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

Similarly, over the region 2, the decision parameter is tested at mid point of $(x_k, y_k - 1)$ and $(x_k + 1, y_k - 1)$ i.e.

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

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

$$\therefore p_{2k} = r_y{}^2 x_k{}^2 + r_y^2 x_k + \frac{r_y^2}{4} + r_x{}^2(y_k - 1)^2 - r_x{}^2 r_y{}^2 \text{ ....(3)}$$

if $p_{2k} > 0,$ the mid point lies outside the boundary, so next point to plot is $(x_k, y_k - 1)$ otherwise, next point to plot will be $(x_k + 1, y_k - 1)$

The successive decision parameter is computed as evaluating ellipse function at mid point of

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

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

Or $p_{2k+1} = r_y^2 x_{k+1}^2 + r_y^2 x_{k+1} + \frac{r_y^2}{4} + r_x^2(y_k - 1)^2 - 2r_x^2(y_k - 1) + r_x^2 - r_x^2 r_y^2$ ....(4)

Subtracting (4)-(3)

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

Or $p_{2k+1} = p_{2k} + r_y^2(x_{k+1}^2 - x_k^2) + r_y^2(x_{k+1} - x_k) - 2r_x^2(y_k - 1) + r_x^2$

if $p_{2k} > 0,$ $x_{k+1} = x_k$ then

$$p_{2k+1} = p_{2k} - 2r_x^2(y_k - 1) + r_x^2$$

Otherwise $x_{k+1} = x_k + 1$ then

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

Or, $p_{2k+1} = p_{2k} + r_y^2(2x_k + 1) + r_y^2 - 2r_x^2(y_k - 1) + r_x^2$

Or, $p_{2k+1} = p_{2k} + r_y^2(2x_k + 2) - 2r_x^2(y_k - 1) + r_x^2$

Or, $p_{2k+1} = p_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$ where $x_{k+1} = x_k + 1$ and $y_{k+1} = y_k - 1$

The initial position for region 2 is taken as last position selected in region 1 say which is (x_0, y_0) then initial decision parameter in region 2 is obtained by evaluating ellipse function at mid point of (x_0,y_0-1) and (x_0+1,y_0-1) as

$$p_{20} = f_{ellipse}(x_0 + \frac{1}{2}, y_0 - 1)$$

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

Now the mid point **ellipse algorithm is summarized** as;

1. Input center (xc, yc) and r_x and r_y for the ellipse and obtain the first point as $(x_0, y_0) = (0, r_y)$
2. Calculate initial decision parameter value in Region 1 as

   $$P_{10} = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^{\ 2}$$

3. At each x_k position, in Region 1, starting at k = 0 , compute $x_{k+1} = x_k + 1$

   If $p_{1k} < 0,$ then the next point to plot is

   $$p_{1k+1} = p_{1k} + 2r_y^{\ 2} x_{k+1} + r_y^{\ 2}$$

   $y_{k+1} = y_k$

   Otherwise next point to plot is

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

   $p_{1k+1} = p_{1k} + 2r_y^{\ 2} x_{k+1} + r_y^{\ 2} - 2r_x^{\ 2} y_{k+1}$ with $x_{k+1} = x_k + 1$ and $y_{k+1} = y_k - 1$

4. Calculate the initial value of decision parameter at region 2 using last calculated point say (x_0,y_0) in region 1 as

   $$p_{20} = r_y^2 (x_0 + \frac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

5. At each y_k position in Region 2 starting at k = 0, perform computation y_{k+1} = y-1;

   if $p_{2k} > 0,$ then

   $x_{k+1} = x_k$

   $$p_{2k+1} = p_{2k} - 2r_x^2 (y_k - 1) + r_x^2$$

   Otherwise

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

   $p_{2k+1} = p_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$ where $x_{k+1} = x_k + 1$ and $y_{k+1} = y_k - 1$

6. Determine the symmetry points in other 3 quadrants.
7. Move each calculated point (x_k, y_k) on to the centered (xc,yc) ellipse path as

   $x_k = x_k + xc$;

   $y_k = y_k + yc$

8. Repeat the process for region 1 until $2r_y^2 x_k \geq 2r_x^2 y_k$ and region until $(x_k, y_k) = (r_x, 0)$

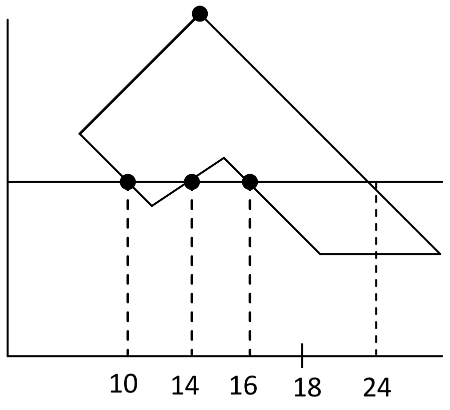
Filled Area primitives

A standard output primitive in general graphics package is solid color or patterned polygon area. Other kinds of area primitives are sometimes available, but polygons are easier to process since they have linear boundaries. There are two basic approaches to area filling in raster systems. One way to fill an area is to determine the overlap intervals for scan lines that cross the area. Another method for area filling is to start from a given interior position and point outward from this until a specified boundary is met.

Scan-line Polygon Fill Algorithm

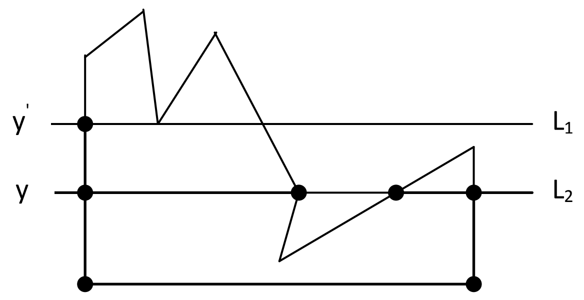
In scan-line polygon fill algorithm, for each scan-line crossing a polygon, it locates the intersection points of the scan line with the polygon edges. These intersection points are then sorted from left to right, and the corresponding frame-buffer positions between each intersection pair are set to the specified color.

In the figure below, the four pixel intersection positions with the polygon boundaries defined two stretches of interior pixel from x=10 to x=14 and from x=16 to x=24.



Some scan-line intersections at polygon vertices require extra special handling. A scan-line passing through a vertex intersects two polygon edges at that position, adding two points to the list of intersection for the scan-line.

This figure shows two scan lines at position y and y' that intersect the edge points. Scan line at y intersects five polygon edges. Scan line at y' intersects 4 (even number) of edges though it passes through vertex.



Intersection points along scan line y' correctly identify the interior pixel spans. But with scan line y, we need to do some additional processing to determine the correct interior points.

For scan line y, the two edges sharing the intersecting vertex are on opposite side of the scan-line. But for scan-line y' the two edges sharing intersecting vertex are on the same side (above) the scan line position.

We can identify these vertices by tracing around the polygon boundary either in clockwise or counter clockwise order and observing the relative changes in vertex y coordinates as we move from one edge to next. If the endpoint y values of two consecutive edges monotonically increases or decrease, we need to count the middle vertex as a **single intersection point** for any scan line passing through that vertex. Otherwise the shared vertex represents a local extremum (minimum or maximum) on the polygon boundary, and the **two edge intersections** with the scan-line passing through that vertex.

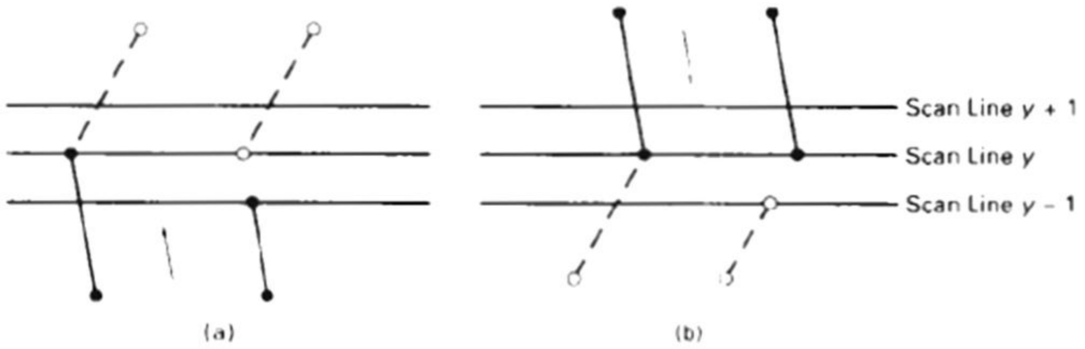


Fig: Adjusting endpoint **y** values for a polygon, as we process edges in order around the polygon perimeter. The edge currently being processed is indicated as a solid line. In **(a),** they coordinate of the upper endpoint of the current edge is decreased by 1. In **(b),** they coordinate of the upper endpoint of the next edge is decreased by 1.

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 two successive scan lines crossing a left edge of a polygon.

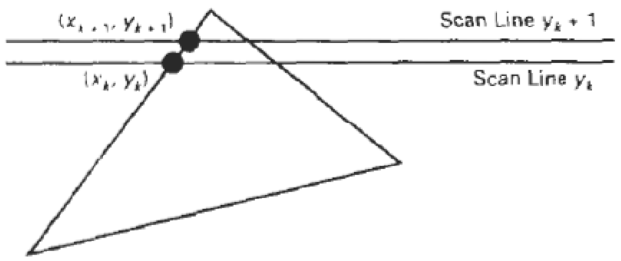


Fig: two successive sc n-lines intersecting a polygon boundary

The slope of this polygon boundary line can be expressed in terms of the scan-line intersection coordinates:

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

Since the change between two scan line in y co-ordinates is 1,

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

The x-intersection value $x_k + 1$, on the upper scan line can be determined from the x-intersection value x_k, on the preceding scan line as

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

Each successive **x**-intercept can thus be calculated by adding the inverse of the slope and rounding to the nearest integer recalling that the slope m is the ratio to two integers

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

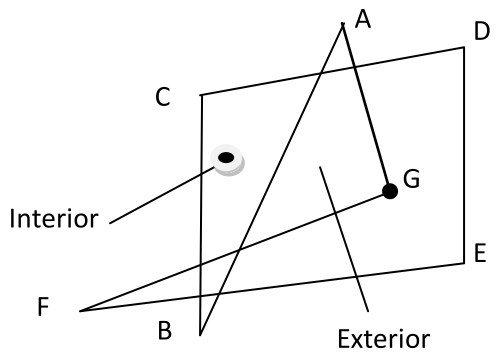
Where $\Delta x \,\&\, \Delta y$ are the differences between the edge endpoint x and y co-ordinate values. Thus incremental calculations of x intercepts along an edge for successive scan lines can be expressed as

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

Inside-Outside Test

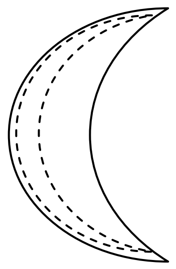
Area filling algorithms and other graphics package often need to identify interior and exterior region for a complex polygon in a plane. Viz. in figure below, it needs to identify interior and exterior region.

We apply add-even rule, also called odd-parity rule. To identify the interior or exterior point, we conceptually draw a line from a point p to a distant point outside the co-ordinate extents of the object and count the number of intersecting edge crossed by this line. If the intersecting edge crossed by this line is odd, P is **interior** otherwise P is **exterior**.



Scan-Line Fill of Curved Boundary Area

It requires more work than polygon filling, since intersection calculation involves nonlinear boundary for simple curves as circle, ellipses, performing a scan line fill is straight forward process. We only need to calculate the two scan-line intersection on opposite sides of the curve. Then simply fill the horizontal spans of pixel between the boundary points on opposite side of curve. Symmetries between quadrants are used to reduce the boundary calculation we can fill generating pixel position along curve boundary using mid point method.

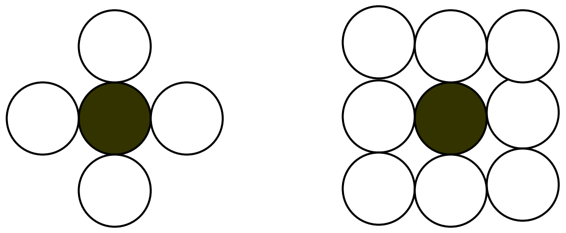


Boundary-fill Algorithm

In Boundary filling algorithm starts at a point inside a region and paint the interior outward the boundary. If the boundary is specified in a single color, the fill algorithm proceeds outward pixel by until the boundary color is reached.

A boundary-fill procedure accepts as input the co-ordinates of an interior point (x, y), a fill color, and a boundary color. Starting from (x, y), the procedure tests neighboring positions to determine whether they are of boundary color. If not, they are painted with the fill color, and their neighbours are tested. This process continues until all pixels up to the boundary color area have tested. The neighbouring pixels from current pixel are proceeded by two methods:

- 4- Connected if they are adjacent horizontally and vertically.
- 8- Connected if they adjacent horizontally, vertically and diagonally.



- Fill method that applies and tests its 4 neighbouring pixel is called **4-connected**.
- Fill method that applies and tests its 8 neighbouring pixel is called **8-connected**.

<u>Algorithm Outline</u>: Recursive procedures

Boundary-Fill 4-Connected

```
void Boundary_fill4(int x, int y, int b_color, int fill_color)
{
            int value = getpixel (x, y);
            if (value! =b_color && value!=fill_color)
            {
                        putpixel (x, y, fill_color);
                        Boundary_fill 4 (x-1, y, b_color, fill_color);
                        Boundary_fill 4 (x+1, y, b_color, fill_color);
                        Boundary_fill 4 (x,y-1, b_color, fill_color);
                        Boundary_fill 4 (x,y+1, b_color, fill_color);
            }
 }
```

Boundary fill 8- connected:

```
void Boundary-fill8(int x,int y,int b_color, int fill_color)
{
            Int current  = getpixel (x, y);
            if (current !=b_color && current!=fill_color)
            (           putpixel (x,y,fill_color);
                        Boundary_fill8(x-1, y, b_color,fill_color);
                        Boundary_fill8(x+1, y, b_color, fill_color);
                        Boundary_fill8(x, y-1, b_color, fill_color);
                        Boundary_fill8(x, y+1, b_color, fill_color);
                        Boundary_fill8(x-1, y-1, b_color,fill_color);
                        Boundary_fill8(x-1,y+1,b_color,fill_color);
                        Boundary_fill8(x+1,y-1,b_color,fill_color);
                        Boundary_fill8(x+1,y+1,b_color,fill_color);
            }
}
```

Recursive boundary-fill algorithm does not fill regions correctly if some interior pixels are already displayed in the fill color. Encountering a pixel with the fill color can cause a recursive branch to terminate, leaving other interior pixel unfilled. To avoid this we can first change the color of any interior pixels that are initially set to the fill color before applying the boundary fill procedure.

Flood-fill Algorithm

Flood-Fill Algorithm is applicable when we want to fill an area that is **not defined within a single color boundary**. If fill area is bounded with different color, we can paint that area by replacing a specified interior color instead of searching of boundary color value. This approach is called **flood fill algorithm**.

We start from a specified interior pixel (x, y) and reassign all pixel values that are currently set to a given interior color with desired fill-color.

Using either 4-connected or 8-connected region recursively starting from input position, the algorithm fills the area by desired color.

Algorithm:

```
void flood_fill4(int x, int y, int fill_color, int old_color)
{
        int current = getpixel (x,y);
        if (current==old_color)
        {
                putpixel (x,y,fill_color);
                flood_fill4(x-1, y, fill_color, old_color);
                flood_fill4(x, y-1, fill_color, old_color);
                flood_fill4(x, y+1, fill_color, old_color);
                flood_fill4(x+1, y, fill_color, old_color);
        }
}
```

Similarly flood fill for 8 connected can be also defined.

We can modify procedure flood_fill4 to reduce the storage requirements of the stack by filling horizontal pixel spans.