Chapter 1

Introduction

Definition:

The term of Graphics comes from Greek "*graphikos"* which means 'something written' e.g. autograph. So, graphics are visual images or designs on some surface, such as a wall, canvas, screen, paper, or stone to inform, illustrate, or entertain.

Computer Graphics is a field that is concerned with all aspects of producing pictures or images using a computer with the help of data structure, graphics algorithm and programming. i.e.

Computer graphics = Data Structure + Graphics Algorithm + Language

Computer Graphics includes the creation; storage and manipulation of images of objects those come from diverse fields such as physical, mathematical engineering, architectural abstract structures and natural phenomenon.

We use different input and display devices to generate & perform the modification on the digital images. Some such devices are keyboard, mouse, or touch sensitive panel, monitors etc.

Some considerable terminologies on the computer graphics are:

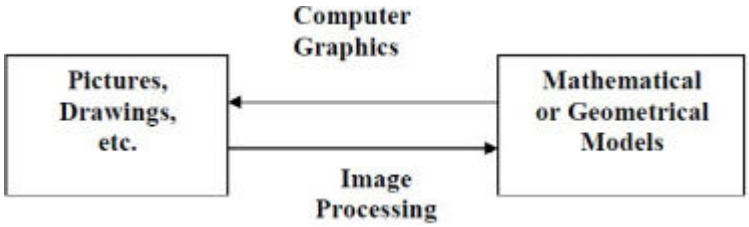
- ✗ Modeling: creating and representing the geometry of objects in the 3D world
- ✗ Rendering: generating 2D images of the 3D objects
- ✗ Animation: describing how objects change in time

History of Computer Graphics

The history of the computer graphics illustrates a acute development of hardware and software. The past principals and techniques are still applicable in present and future computer graphics technology. The evolutions of graphics can be explained under following points.

Year	**Major Achievements**
1950	First Graphics Images were created
1951	CRT monitors on Main Frame computer were instroduced
1959	CAD was used to design cars
1961	First video game named "Space War"developed
1963	First Hidden Line and Hidden surface removal algorithms developed.
1965	DDA algorithm developed by Jack Bresenham
1973	First use of 2D animations
1982	AutoCAD was released.
2001	First digital film name "The Spirits Within"with digital actors
2006	Google acquired sketechup
2015	"Big Data" used for constructing animation
2016	With enough preparation, Real Time source can be animated

Differentiate between Computer Graphics and Image Processing



Computer Graphics	Image Processing
Computer Graphics involves in generating images from standard graphical models	Image Processing involves in analyzing the images to generate standard graphical models.
It includes the creation storage and manipulation of images or objects.	It is the part of computer graphics that handles image manipulation or interpretation.
E.g., drawing a picture	Eg: Making blur image visible

Interactive and Non-Interactive Computer Graphics

Interactive Computer Graphics:

Interactive Computer Graphics involves a two-way communication between computer and user. The observer is given some control over the image by providing him with an input device for example the video game controller of the ping pong game.

Different parts of Interactive Computer Graphics are:

- Inputs
- Processing
- Outputs

Non-Interactive Computer Graphics:

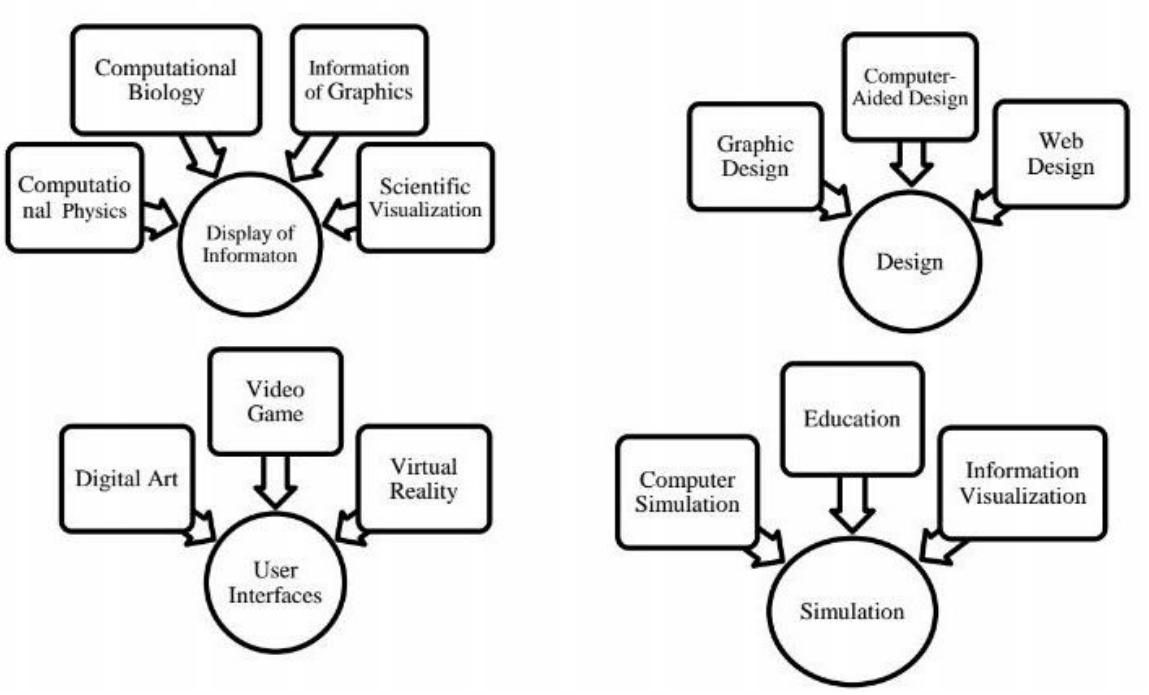
Non-interactive computer graphics or passive computer graphics is the computer graphics in which user does not have any kind of control over the image. Image is simply the product of static stored program and the image is totally under the control of program instructions not under the user. Example: screen savers.

Application of Computer Graphics

There is no area in which graphical displays can't be used to some advantage. Today almost every computer can do some graphics, and people have even come to expect to control their computer through icons and pictures rather than just by typing.

We can classify applications of computer graphics into four main areas:

- Display of information
- Design
- User interfaces
- Simulation



1. **Computational Biology:** Computational biology is a field that applies the techniques of computer science, applied mathematics and statistics to address biological problems. The main focus lies on developing mathematical modeling and computational simulation techniques.
2. **Computational Physics:** Computational physics is the study and implementation of numerical algorithm to solve problems in physics for which a quantitative theory already exists.
3. **Information of Graphics:** Information graphics are visual representations of information, data or knowledge. These graphics are used where complex information needs to be explained quickly and clearly, such as in signs, maps, journalism, technical writing, and education.
4. **Scientific Visualization:** Scientific visualization focuses on the use of computer graphics to create visual images which aid in understanding of complex, often massive numerical representation of scientific concepts or results.
5. **Graphic Design:** Graphic design can refer to a number of artistic and professional disciplines which focus on visual communication and presentation. Various tools like Coral Draw, In Design, photo shop are used.
6. **Computer-aided Design:** Computer-aided design (CAD) is the use of computer technology for the design of objects, real or virtual. CAD is also widely used to produce computer animation for special effects in movies, advertising, technical manuals.
7. **Web Design:** The process of designing Web pages, Web sites, Web applications or multimedia for the Web may utilize multiple disciplines, such as animation, communication design, corporate identity, graphic design, interaction design, marketing, photography, search engine optimization etc.
8. **Digital Art:** The impact of digital technology has transformed traditional activities such as painting and drawing.
9. **Video Games:** A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a raster display device. The electronic systems used to play video games are known as platforms. This platform creates through graphics.
10. **Virtual Reality:** Virtual reality (VR) is a technology which allows a user to interact with a

computer-simulated environment. The simulated environment can be similar to the real world, for example, simulations for pilot or combat training, or it can differ significantly from reality, as in VR games.

11. **Computer Simulation:** A computer simulation, a computer model or a computational model is a computer program, or network of computers, that attempts to simulate an abstract model of a particular system.
12. **Education:** Different computer graphics and images are used in schools and many training centers for the better understanding the subject of interest.
13. **Information Visualization:** Information visualization is the study of the visual representation and the use of graphical techniques to help people understand and analyze data

Chapter 2

Graphics Hardware

Input and Input Devices

- Keyboard
- Mouse
- Light Pen
- Touch panel
- Barcode Readers
- Data Gloves

Output and Output Devices

1. Hard Copy Devices
2. Display Devices

Basic Terms:

➢ **Image**

An image is a visual representation of something. It is a 2 dimensional light intensity function f(x,y) where (x,y) are the spatial co-ordinate and f(x, y) is proportional to brightness or intensity or gray value of the image at that point.

Types of Images:

- Raster Image
- Vector Image

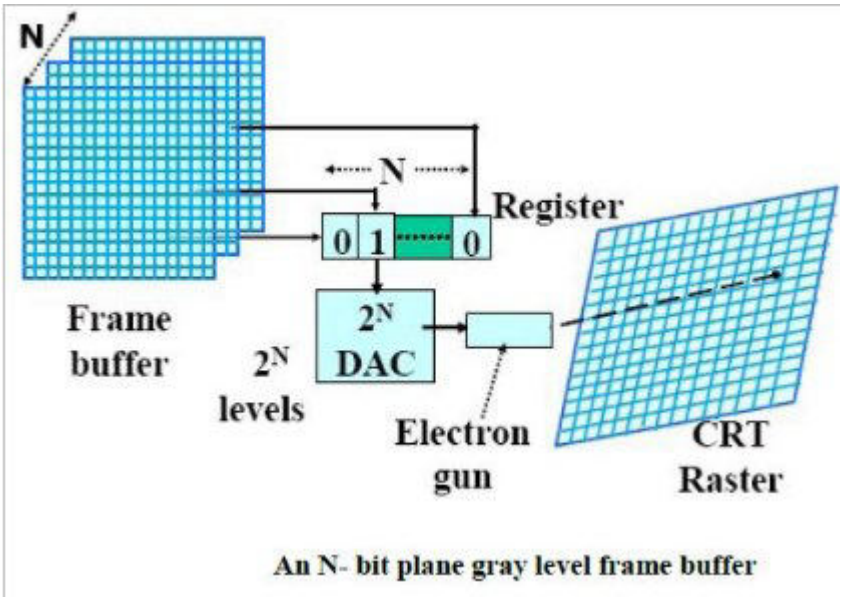
➢ **Pixel (or Pel)**

Computer graphics consists pictures or images which are collection of discrete picture elements called pixels.

➢ **Frame (or Refresh) Buffer**

➢ It is a large piece of memory into which the intensity values for all pixels are placed.

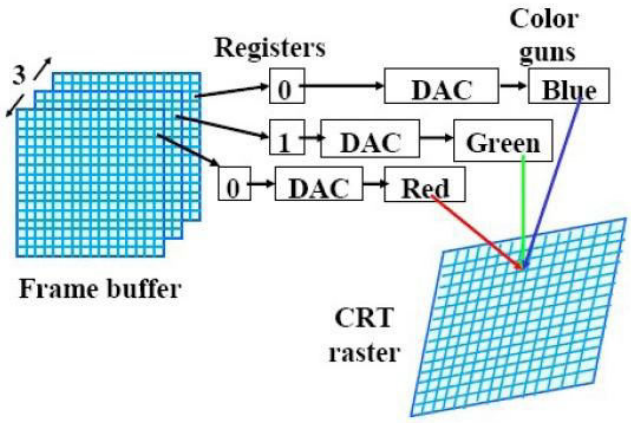
➢ At a minimum there is one memory bit for each pixel called a bit plane which stores the internal representation of image called frame buffer.



An N- bit plane gray level frame buffer

Figure given above illustrates a system with N=3 bit planes for a total of 8 (2^3) intensity levels. The frame buffer is a digital device and the CRT is an analog device. Therefore, a conversion from a digital representation to an analog signal must take place when information is read from the frame

buffer and displayed on the raster CRT graphics device. For this we can use a digital to analog converter (DAC). This is converted into an analog voltage between 0 and the maximum voltage of the electron gun by the DAC. Each pixel in the frame buffer must be accessed and converted before it is visible on the raster CRT.



Simple Color buffer

Because there are three primary colors, a simple color frame buffer is implemented with three bit planes, one for each primary color. Each bit plane drives an individual color gun for each of the three primary colors used in color video. These three primaries (red, green, and blue) are combined at the CRT to yield eight colors.

- **Aliasing**
  - Aliasing is distortion that appear in any display system when the sampling the continuous object to discrete integer pixel position.
  - This is because, lines, polygon, circle etc. are continuous but a raster device is discrete.
- **Bit Map and Pixel Map**
  - If a pixel has only two-color values (i.e. black and white), it can be encoded by a 1 bit of information. On a black and white system with one bit per pixel, the frame buffer is called bitmap.
  - An image of more than two colors is called pix map.
- **Bit Depth (Or Color Depth)**

  It is defined as number of bits assigned to each pixel in the image.
- **Resolution**
  - It is defined as the ratio of width and height of output of the display device. So the number of pixels in horizontal direction and vertical direction on the screen is called resolution.
  - Example: For a resolution of 1024 * 768, 1024 pixels are going horizontally and 768 pixels are going vertically.
- **Aspect Ratio**
  - It is defined as the ratio of horizontal points to the vertical points required to produce equal length lines in both the direction of the screen.
  - It is given by the following formula.

    $$\text{Aspect Ratio} = \frac{No\ of\ horizontalpoints}{No\ of\ verticalPoints}$$
  - For example: (800 * 600) pixels in the display has the aspect ratio 800/600 = 4/3 i.e. 4:3
  - The main difference between resolution and aspect ratio is that aspect ratio is the shape that the picture takes while the resolution is the number of pixels in that shape.
- **Phosphorescence**

  It is a process in which energy absorbed by a substance is released relatively slowly in the form of light.
- **Persistence**

A phosphor's persistence is the time for the emitted light to decay to 10 % of the initial intensity. The persistence may be varied with different phosphors. The phosphors used for graphics display usually have persistence of 10 to 60 microseconds

- **Refresh Rate**
  - It is defined as the number of times per second the pixels are recharged so that the image doesn't flicker. It is measured in hertz(hz).
  - It is also called as frame rate, horizontal scan rate, vertical frequency or vertical scan rate.
  - Normally the refresh rate varies from **60 to 80hz**.
  - A refresh rate of 75 hz means that the image is redrawn 75 times a second.

Cathode Ray Tube(CRT)

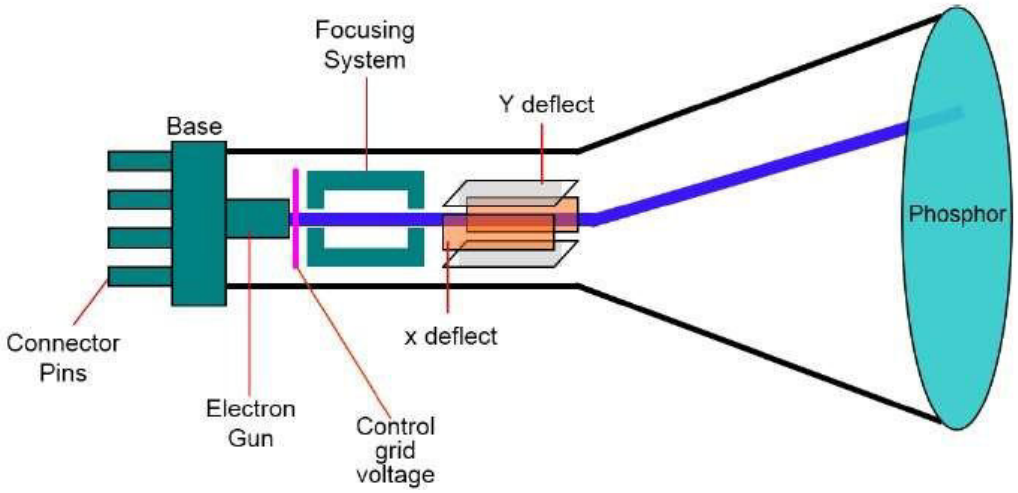


Fig: CRT Block diagram

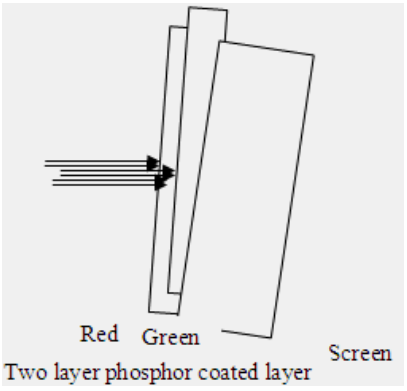
The image on a CRT display is created by firing electrons from the back of the tube to phosphors located towards the front of the display. Once the electrons hit the phosphors, they light up and are projected on the screen. The color you see on the screen is produced by a blend of red, blue, and green light, often referred to as RGB.

- Types of CRT

a) Beam Penetration Method

The beam penetration method is for the random scan monitor display where two different layers of phosphor coating are used, Red (outer) and Green (inner) coated on the CRT screen.

In this method, only four colors possible and hence the poor picture quality



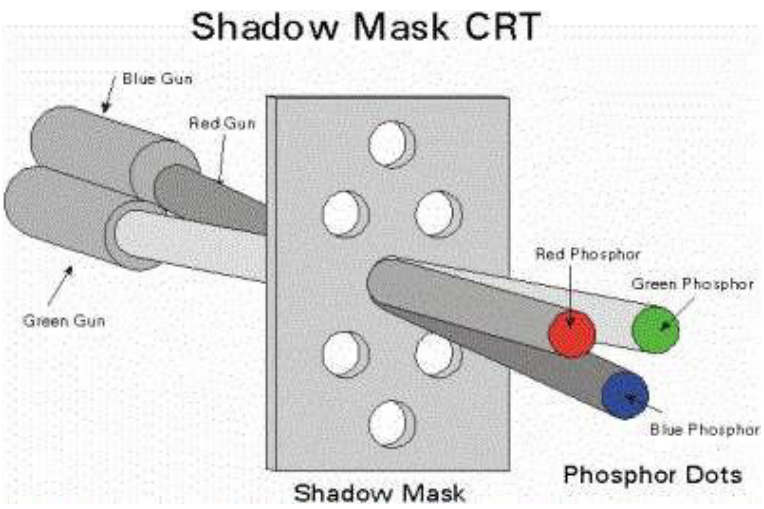
i. A beam of slow electrons excites only the outer red layer
ii. A beam of very fast electrons penetrates through the red phosphor and excites the inner green layer.
iii. Intermediate is a combination of red and green so two additional colors orange and yellow color.

- When quantity of red is more than green then color appears as orange
- When quantity of green is more than red then color appears as yellow

b) Shadow Mask Method

The shadow mask CRT, instead of using one electron gun, uses 3 different guns placed one by the side of the other to form a triangle or a "Delta" as shown. Each pixel point on the screen is also made up of 3 types of phosphors to produce red, blue and green colors. Just before the phosphor screen is a metal screen, called a "shadow mask".

This plate has holes placed strategically, so that when the beams from the three electron guns are focused on a particular pixel, they get focused on particular color producing pixel only



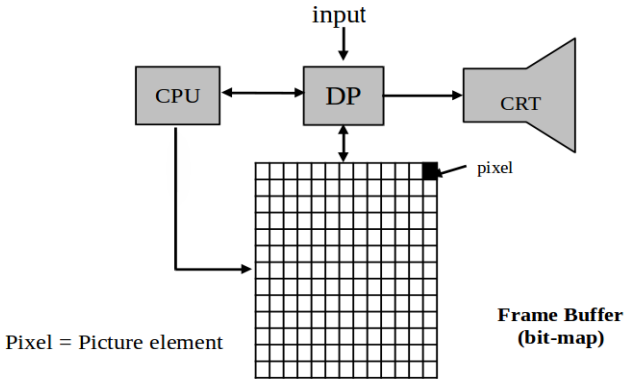
Raster and Vector Graphics

A raster image is made of up pixels, each a different color, arranged to display an image where as a vector image is made up of paths, each with a mathematical formula (vector) that tells the path how it is shaped and what color it is bordered with or filled by.

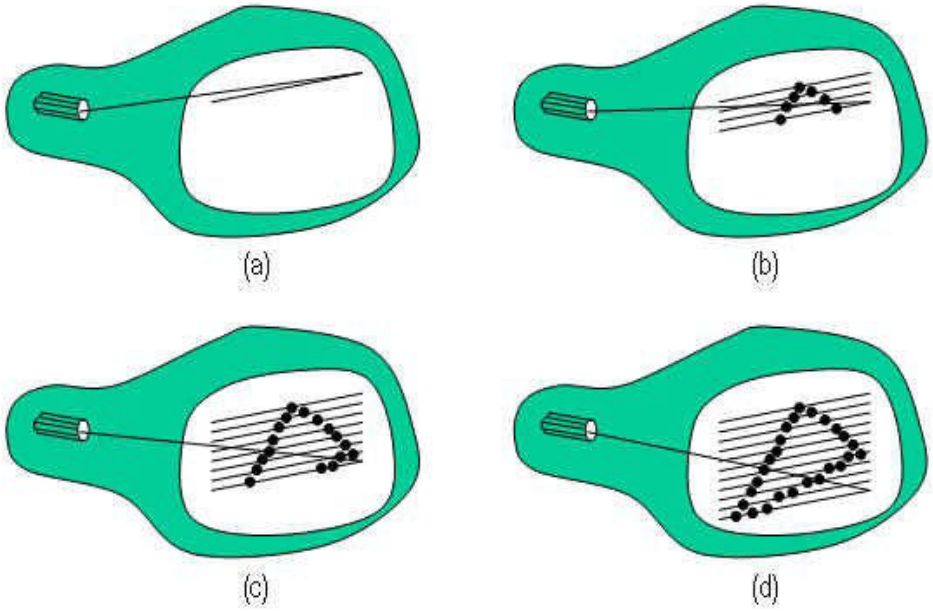
Raster Graphics	Vector Graphics
Raster Graphics are composed of pixels	Vector graphics are composed of paths.
Raster image pixels do not keep on their appearance as size increases - when you blow a photograph up, it becomes blurry for this reason	Vector images keep on appearance regardless of size, since the mathematical formulas dictate how the image is rendered
Normally they have the file extension of .gif, .jpg	They have the extension of .eps

Refresh or Raster Scan Display System

- A raster consists of pixels organized into rows and columns (or a grid) where each cell contains a value of information. In this display system, raster points are used as basic drawing primitives.
- Home television (CRT) and printers are example of systems using raster scan method.

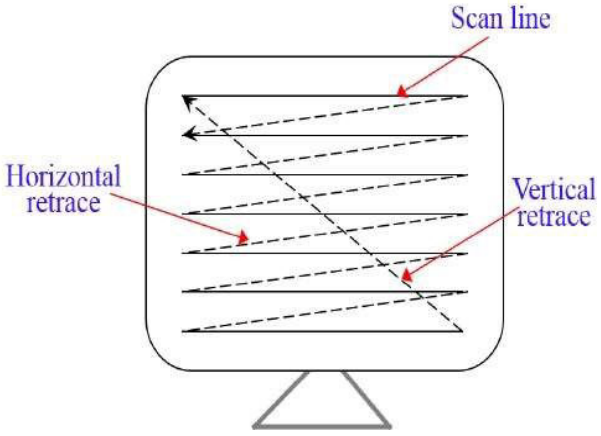


- In a raster scan system, the electron beam is swept across the screen, one row at a time from top to bottom. As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.
- Picture definition is stored in memory area called the **Refresh Buffer** or **Frame Buffer**. This memory area holds the set of intensity values for all the screen points. Stored intensity values are then retrieved from the refresh buffer and "painted" on the screen one row (scan line) at a time as shown in the following illustration.
- Refreshing on raster-scan displays is carried out at the rate of 60 to 80 frames per second. Refreshing must be done because 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 some point. The no of times/sec the image is redrawn to give a feeling of non-flickering pictures is called **refresh-rate**.



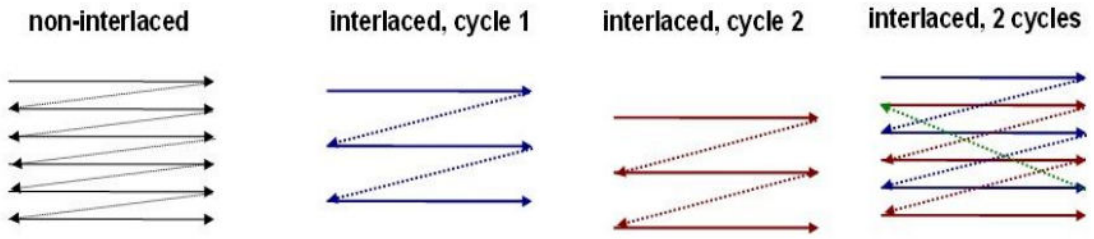
- Each scan line in display device consists of two retrace
  i. **Retrace procedure**

  At the end of each scan line in raster scan display, the electron beam returns to the left side of the screen to begin displaying the next scan line. The return to the left of the screen, after refreshing each scan line is called the horizontal retrace of the electron beam. And at the end of each frame the electron beam returns to the top left corner of the screen to begin the next frame which is called vertical retrace.



ii. Interlaced refresh procedure

On some raster scan systems each frame is displayed in two passes using an interlaced refresh procedure so that the whole picture should displaced in half time. Here, the first scan does the even lines 0, 2, 4,... then the second scan does the odd lines 1, 3, 5,....



➢ Architecture of Raster Scan Display System

- A special purpose processor, called *video controller* or *display controller* is used to control the operation of the display device.
- The below figure shows the commonly used raster system organization.

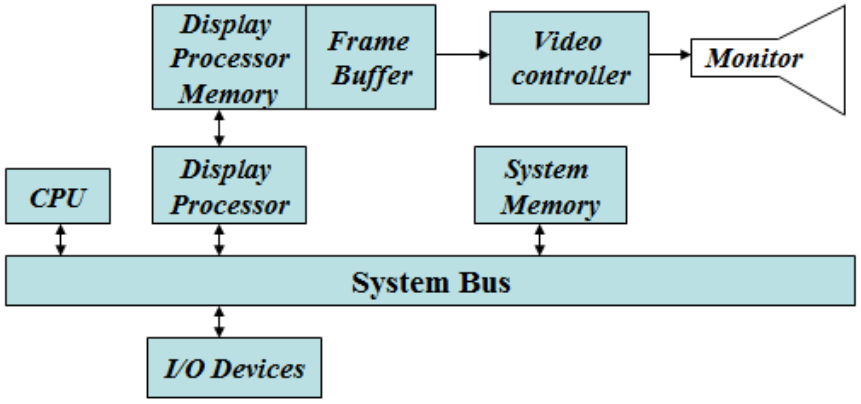
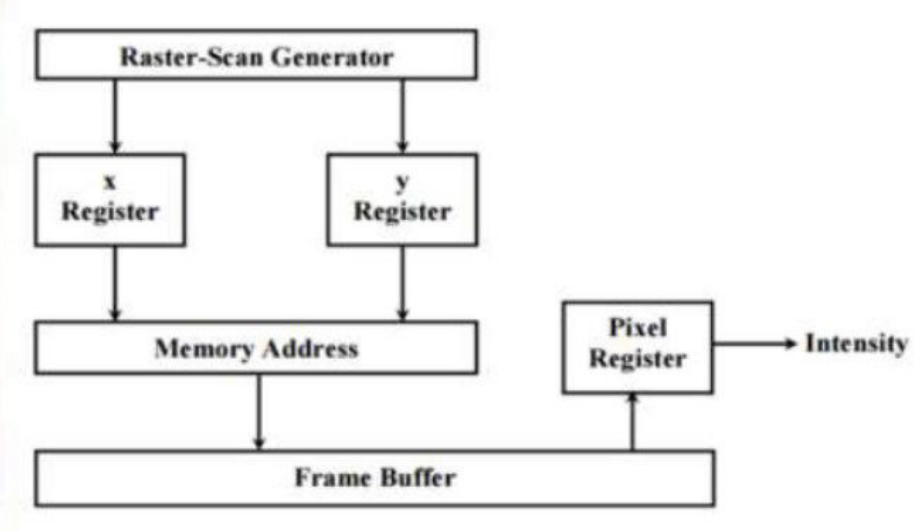


Fig. Architecture of Raster display system with display processor

- When a particular command is called by the application program, the graphics subroutine package sets the appropriate pixels in the frame buffer. The video controller then cycles through the frame buffer, one scan line at a time, typically 50 times per second. It will bring a value of each pixel contained in the frame buffer and uses it to control the intensity of the CRT electron beam. So there exists a one to one relationship between the pixel in frame buffer and that on the CRT screen.
- Frame buffer locations, and the corresponding screen position are referenced in Cartesian co-ordinates. For most of the system, the coordinate origin is referenced at the lower left corner of the screen, with positive X value increasing to the right and positive y value increasing from bottom to top. However, in some PC, the coordinate origin is referenced at the upper left corner of the screen.

Video Controller

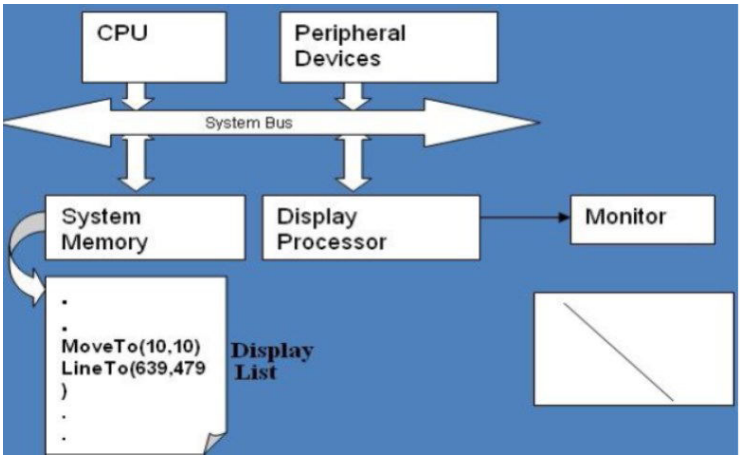


Two registers are used to store the coordinates of the screen pixels; x register is used to store x coordinate and y register is used to store y-coordinate.

The raster scan generator produces the deflection signals that generates the raster scan and also controls x and y registers. Initially, the x-axis is incremented by 1 up to x_{max}. Each pixel value is fetched and used to control the intensity. After the first scan line, the x-address is set to 0 and y address is incremented by 1 and the process continues till y_{max}.

Vector Display System or Random Scan

- In this technique, the electron beam is directed only to the part of the screen where the picture is to be drawn rather than scanning from left to right and top to bottom as in raster scan. It is also called **vector display, stroke-writing display,** or **calligraphic display**.
- Picture definition is stored as a set of line-drawing commands in an area of memory referred to as the **refresh display file**. To display a specified picture, the system cycles through the set of commands in the display file, drawing each component line in turn. After all the line-drawing commands are processed, the system cycles back to the first line command in the list.
- Random-scan displays are designed to draw all the component lines of a picture 30 to 60 times each second.
- **Architecture for Random Scan System**



- The graphics command in the application program are translated by the graphics package into a display list stored in system memory. The display list is accessed by the display processor to refresh the screen. The display processor cycles through each command in the display list once during each refresh cycle.
- Graphics pattern are drawn by directing the electron beam along the component lines of the picture. Lines are defined by the values for their co-ordinate endpoints.
-A scene is then drawn one line at a time by positioning beam to fill in the line between specified endpoints.

Raster Model	Vector Model
Advantages: ➢ Simple data structure ➢ Easy and efficient overlaying ➢ Compatible with Remote Sensing images ➢ High spatial variability is efficiently represented	Advantages: ➢ Compact data structure ➢ Efficient for network analysis ➢ Efficient projection transformation ➢ Accurate map output
Disadvantages ➢ Inefficient use of computer storage ➢ errors in perimeter, area, and shape ➢ Difficult network analysis ➢ Less accurate maps	Disadvantages ➢ Difficult overlay operations ➢ High spatial variability is inefficiently represented ➢ Not compatible with RS imagery

Flat Panel Display

The term flat–panel displays refers to a class of video devices that have reduced volume, weight, and power requirements compared to a CRT. A significant feature of flat-panel displayed is that they are thinner than CRTs, and we can hang them on walls or wear them on our wrists.

We can separate flat panel displays into two categories

1. Emissive Display
2. Non-Emissive Display

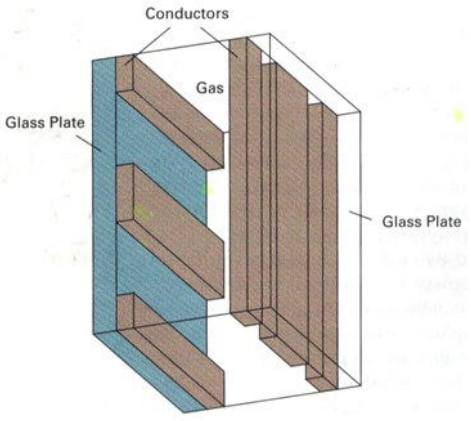
Emissive Display

Emissive display or emitter change electrical energy into light energy. Plasma boards and light discharging diode are cases of emissive display.

➢ Plasma Display:

Plasma Display is a emissive display common to large TV displays beyond 30 inches or larger. They are called plasma displays because they use small cell containing electrically charged ionized gas.

Plasma Panel are developed by filling area between two glass plates with a blend of gas.



Advantages:

- Wider viewing angles than of LED.

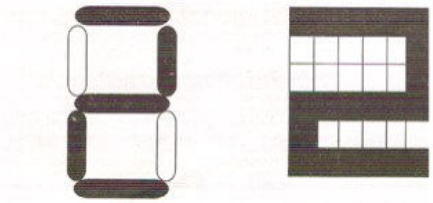
- Less visible motion blur.

Disadvantages:

- does not work at high altitude above 2km due to pressure differential between gas.
- Radio Frequency Interface can be irritating.

➢ Light Emitting Diodes

A matrix of diodes is arranged to form the pixel positions in the display, and picture definition is stored in refresh buffer. As in scan- line refreshing of a CRT, information is read from the refresh buffer and converted to voltage levels that are applied to the diodes to produce the light patterns in the display.



a. Seven Segment Display b. Matrix of diode

Q. How does the Light Emitting Diode work?

LEDs are diodes. A diode has a P-N junction across which charge carriers like electrons and holes pass when current flows through the diode. When forward biased, the electrons from N region flows to the P region and holes from P region towards N region. Some of the electrons recombine with the holes at the junction and their energy is radiated outward. By proper design using suitable materials like indium phosphide or gallium arsenide, we can create a junction that radiates maximum energy as visible light.

The wavelength of the radiation and hence the color of the light depends on the materials used

Non-Emissive Display

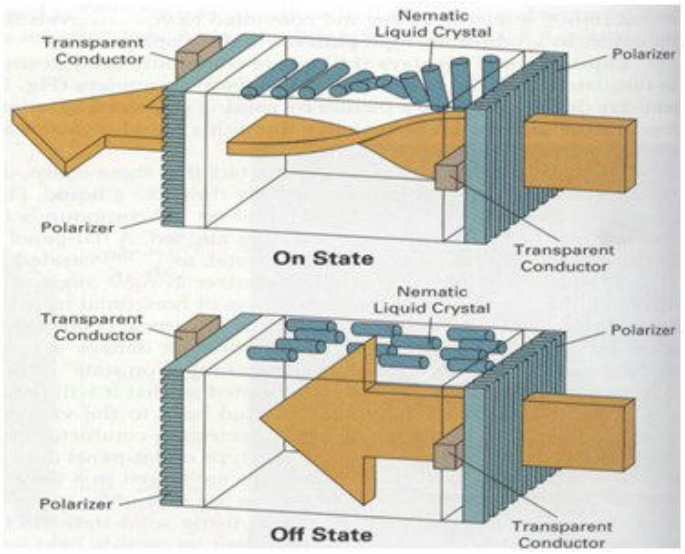
Non-emissive displays (or **no emitters**) use optical effects to convert sunlight or light from some other source into graphics patterns. The most important example of a non-emissive flat-panel display is a liquid- crystal device.

➢ LCD

Standing for "Liquid Crystal Display", it is a flat panel display technology commonly used in TVs and computer monitors. It is also used in screens for mobile devices, such as laptops, tablets, and smart phones.

How LCD works

- ✗ A very small electric field is required to excite the crystals into their liquid state.
- ✗ Most of the energy used by an LCD display system is due to the back lighting.
- ✗ Thus the crystals spend most of their time in intermediate states, being neither "On" or "Off". This behavior is indicative of passive displays
- ✗ An LCD has back light that provides light to individual pixels arranged in a rectangular grid. Each pixel has a red, green, and blue RGB sub-pixel that can be turned on or off. When all of a pixel's sub-pixels are turned off, it appears black. When all the sub-pixels are turned



on 100%, it appears white. By adjusting the individual levels of red, green, and blue light, millions of color combinations are possible.

Advantage:

- ✗ Low power utilization.
- ✗ Low weight
- ✗ LCD's are level.
- ✗ Small size.

Disadvantages:

- ✗ LCD's can be seen just from a restricted point
- ✗ LCD's are temperature subordinate (0-70ºC).
- ✗ LCD's don't transmit light

Q. Difference between LCD and plasma

	LCD	Plasma
Cost	Expensive than Plasma	Cheaper than LED
Viewing	Upto 165º, Picture suffer from the side	Looks same from almost any angle
Backlight	Yes	No

Solved Problems

1. **There is a system with 24 bits per pixel and resolution of 1024 by 1024. Calculate the size of frame buffer (in Megabytes)**

8bit= 1 byte
1024byte = 1 kilobyte
1024kilobyte= 1Megabyte
1024Megabyte= 1 Gigabyte

Solution:

Resolution = 1024 * 1024
Total number of pixel = 1024 * 1024 = 1048576 pixels
Bits per pixels storage = 24 bits
Therefore, total storage required in frame buffer = 1048576 * 24
= 25165824bits
=25165824 / 8 Byte
= 25165824 / (8 * 1024) Kb
= 25165824 / (8 * 1024 * 1024) Mb
= 3 Mb

2. **How Many k bytes does a frame buffer needs in a 600 x 400 pixel?**

Suppose, n bits are required to store 1 pixel. Then,
the size of frame buffer = Resolution * bits per pixel
= (600 * 400) * n bits
= 240000 n bits
= 240000 n kb / (8 * 1024)
= 29.30 n k bytes.

3. **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 * 10 inch

   Pixels Per inch (Resolution) = 100

   Then total number of pixels = 8 * 100 * 10 * 100 = 800000 pixels

   Bits per pixels' storage =8

   Therefore, total storage required in frame buffer = 800000 * 8 bits

   = 6400000bits

   = 6400000/8 bytes

   = 800000Bytes

4. **Find out the aspect ratio of the raster system using 8 x 10 inches' screen and 100 pixels/inch.**

   Solution:

   We know that, Aspect ratio = Width / Height

   = (8 x 100) / (10 * 100)

   = 4 / 5

   So, aspect ratio = 4: 5

5. **What is the time required to display a pixel on the monitor of size 1024 * 768 with refresh rate of 60 Hz?**

   Solution:

   Refresh Rate = 60Hz → 60 frames per second

   Total number of pixel in one frame = 1024 * 768 = 786432 pixels.

   60 frames need 1 second

   1 frame need 1 / 60 second → 786432 pixels need 1/ 60 second

   → 1 pixels need 1 / (60 * 786432) second

   → 10^9 / (60 * 786432) ns

   → 21.19 ns

6. **If the total intensity achievable for a pixel is 256 and the screen resolution is 640 * 480. What will be the size of frame buffer?**

   Solution:

   1 pixel = 256 different intensity level

   Let; x be the number of bits required to represent 256 different intensity level

   Then, $2^X = 256$

   Therefore, x= 8 bits

   Resolution = 640 * 480

   Hence, number of bits required for the screen = 640 * 480 * 8 = 2457600 bits.

7. **If a pixel is accessed from the frame buffer with an average access time of 300ns then will this rate produce an un-flickering effect for the screen size of 640 * 480.**

   Solution:

   Size of screen = 640 * 480

   Total Number of pixels = 640 * 480 = 307200

Average access time of one pixel = 300ns

Therefore, total time required to access entire pixels of image in the screen

$= 307200 * 300$

$= 92160000$ ns

$= 92160000/10^9$

$= 0.09216$ seconds

i.e. 1 cycle take 0.09216 second

Now, Number of cycles per second i.e. Refresh Rate =?

0.09216 seconds = 1 cycle

1 second = 1/ 0.09216

= 10.86

Therefore, Refresh Rate= 10.86 cycles per second

Since the minimum refresh rate for unflicker image is 60 frames per second, hence we can say the monitor produces flickering effect.

8. **Consider a raster scan system having 12-inch by 12-inch screen with a resolution of 100 pixels per inch in each direction. If display controller of this system refreshes the screen at the rate of 50 frames per second, how many pixels could be accessed per second and what is the access time per pixel of the system.**

Solution:

Size of Screen = 12 inch * 12 inches

Resolution = 100 pixels per inch

Therefore, total number of pixels in one frame = 12 * 100 * 12 * 100

Refresh Rate = 50 frames per second → 50 frames can be accessed in 1 second

Therefore, total number of pixels accessed in 1 second = 50 * 12 * 100 * 12 * 100

= 72000000 pixels.

Again,

Since, 50 frames can be accessed in 1 second

1 frame can be accessed in 1/50 second

(12 * 100 * 12 * 100) pixels can be accessed in 1/50 second

then 1 pixel can be accessed in 1 / (50 * 12 * 100 * 12 * 100) second

$= 10^9 / (50 * 12 * 100 * 12 * 100)$ ns

=13.88 ns

Hence, Access time per pixel = 13.88ns/pixel.

9. **How much time is spent scanning across each row of pixels during screen refresh on a raster system with resolution 1280 * 1024 and refresh rate of 60 frames per second?**

Solution:

Resolution = 1280 * 1024 i.e. One frame contains 1024 scan line and each scan line consists of 1280 pixels.

Refresh Rate = 60 frames per second

i.e. 60 frame take 1 second

1 frame take 1/60 second

i.e. 1024 scan line take 1/60 second i.e. 0.0166 second

1 scan line take 0.0166/1024 = 0.016 Ms.

Chapter 3

Two Dimensional Algorithms

Output Primitives

Output primitives are the geometric structure such as straight-line segments and polygon areas used to describe the shapes and color of the objects.

In case of the two-dimensional algorithm, we mostly deal with the line, circle, ellipse etc. as they are the basic output primitives.

Line Scan Conversion

The slope-intercept equation of a straight line is: $y = mx + b$

where m = slope of line and, b = y-intercept.

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

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

$\therefore b = y - \frac{y_2 - y_1}{x_2 - x_1} x$ from above equation

At any point (x_k, y_k)

$y_k = mx_k + b$ ……..1

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

$y_{k+1} = mx_{k+1} + b$…….2

subtracting 1 from 2 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.

$\therefore \Delta y = m.\Delta x$ ……3

or $m = \frac{\Delta y}{\Delta x}$ ……4

When the value of m is calculated using equation 4, test for three cases can be performed as

Case I: For |m| < 1,

△x can be set to small increment (Generally 1) and the corresponding △y calculated from equation 3.

Case II: For |m| > 1,

△y can be set to small increment (Generally 1) and the corresponding △x calculated from equation 3.

Case III: For |m| = 1,

△y=△x, x and y pixels both are equal.

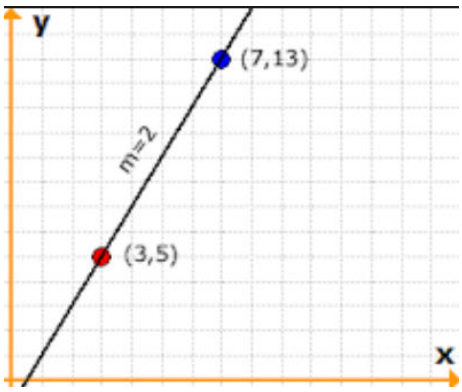
DDA Algorithm(Digital differential analyzer)

It is a scan conversion line algorithm based on calculation either △x or △y using equation m=△y/△x.

We sample the line at unit interval in one direction (x if △x is greater than △y, otherwise in y direction) and determine the corresponding integer values nearest the line path for the other coordinate.

➢ **Algorithm**

Consider a line with positive slope as shown in the figure below



The equation of the line is given,

$y = mx+b$ ……(i)

$m = (y2-y1)/(x2-x1)$ ……(ii)

For any interval $\triangle x$, corresponding interval is given by $\triangle y = m.\triangle x$.

Case I:

If $|m| \leq 1$, we sample at unit x interval i.e $\triangle x = 1$.

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

Then we compute each successive y-values, by setting $\triangle y = m$

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

The calculated y value must be rounded to the nearest integer

Case II:

If $|m| > 1$, we sample at unit y-interval i.e $\triangle y = 1$ and compute each successive x-values.

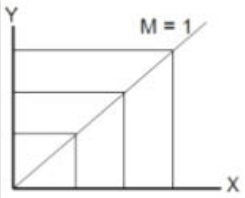
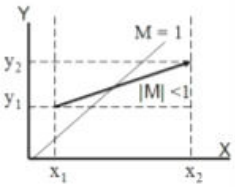
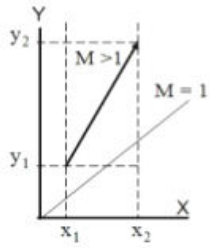
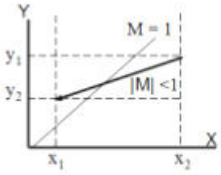
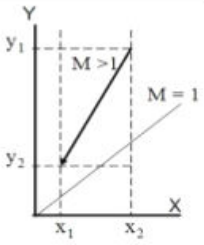
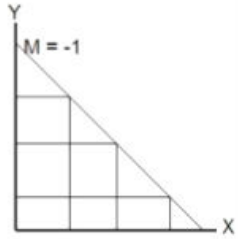
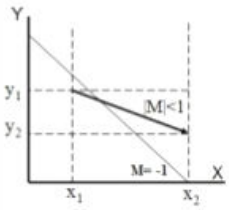
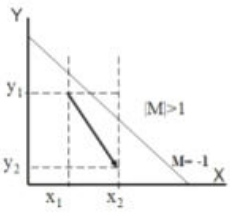
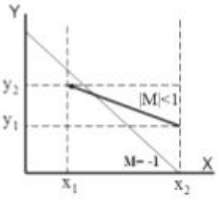
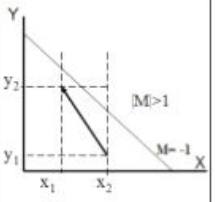
$y_{k+1} = y_k + 1$

Therefore, $1 = m.\triangle x$, and $\triangle x = 1/m$ (since, $m=\triangle y/\triangle x$ and $\triangle y=1$).

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

The above equations are under the assumption that lines are to be processed from left endpoints (x_k, y_k) to right endpoints (x_{k+1}, y_{k+1}).

- **Advantage of DDA**
  a. It is simple to understand.
  b. It is faster method than direct use of line equation y=mx+c.
     (Removes multiplication operation and only involve increments operation of x or y direction)
  c. It requires no special skills for implementation
- **Disadvantages of DDA**
  a. A floating point addition is still needed in determining each successive point which is time consuming.
  b. The accumulation of round off error in successive addition of the floating-point increments may cause the calculated pixel position to drift away from the true line path for long line segment.
- The other possible combinations can be as:

	Moving Left to Right		Moving Right to Left	
	Slope (m) < 1	Slope (m) > 1	Slope (m) < 1	Slope (m) > 1
Positive Slope  *Fig. Positive Slope*	 $x_{k+1} = x_k + 1$ $y_{k+1} = y_k + m$	 $x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k + 1$	 $x_{k+1} = x_k - 1$ $y_{k+1} = y_k - m$	 $x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k - 1$
Negative Slope  *Fig. Negative Slope*	 $x_{k+1} = x_k + 1$ $y_{k+1} = y_k - m$	 $x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k - 1$	 $x_{k+1} = x_k - 1$ $y_{k+1} = y_k + m$	 $x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k + 1$

Numerical Examples for DDA

Q. Consider a line from (2,1) to (8,3). Using simple DDA algorithm, rasterize this line.

Given,

Starting Point :(x1, y1)= (2,1)

Ending Point: (x2,y2) = (8,3)

Now,

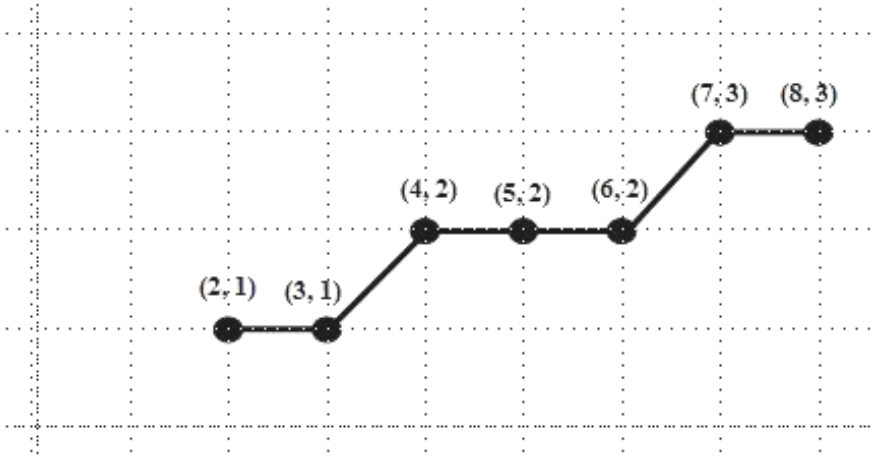
Slope m=(y2-y1)/(x2-x1)= (3-1)/(8-2) =(1/3) = 0.333

Since |m|<1, From DDA algorithm we have

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k + m$

X	Y	X-Plot	Y-Plot	(X,Y)
2	1	2	1	(2,1)
3 (2+1)	1.33 (1+0.33)	3	1	(3,1)
4 (3+1)	1.66 (1.33+0.33)	4	2	(4,2)
5	1.993	5	2	(5,2)
6	2,326	6	2	(6,2)
7	2.659	7	3	(7,3)
8	2.999	8	3	(8,3)



Q. Digitized the line with end points (0,0) and (4,5) using DDA.

Given Starting Point :(x1, y1)= (0,0) and Ending Point: (x2,y2) = (4,5)

Now Slope m=(y2-y1)/(x2-x1)= (5-0)/(5-0) =(5/4) = 1.25

Since |m|>1, From DDA algorithm we have

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

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

X	Y	X-Plot	Y-Plot	(X,Y)
0	0	0	0	(0,0)
0.8	1	1	1	(1,1)
1.6	2	2	2	(2,2)
2.4	3	2	3	(2,3)
3.2	4	3	4	(3,4)
4	5	4	5	(4,5)

Q. Digitized the line with end points (3,7) and (8,3) using DDA.

Given Starting Point :(x1, y1)= (3,7) and Ending Point: (x2,y2) = (8,3)

Now Slope m=(y2-y1)/(x2-x1)= (3-7)/(8-3) =(-4/5) = -0.8

Since |m|<1, From DDA algorithm we have

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k + m$

X	Y	X-Plot	Y-Plot	(X,Y)
3	7	3	7	(3,7)
4	7-0.8=6.2	4	6	(4,6)
5	5.4	5	5	(5,5)
6	4.6	6	5	(6,5)
7	3.8	7	4	(7,4)
8	3	8	3	(8,3)

Q. Consider a line from (0,0) to (5,5). Using simple DDA algorithm, rasterize this line.

Solution:

Given Starting Point :(x1, y1)= (0,0) and Ending Point: (x2,y2) = (5,5)

Now Slope m=(y2-y1)/(x2-x1)= (5-0)/(5-0) =(/55) = 1
Since |m|<=1, From DDA algorithm we have
$X_{k+1} = X_k + 1$
$Y_{k+1} = Y_k + m$

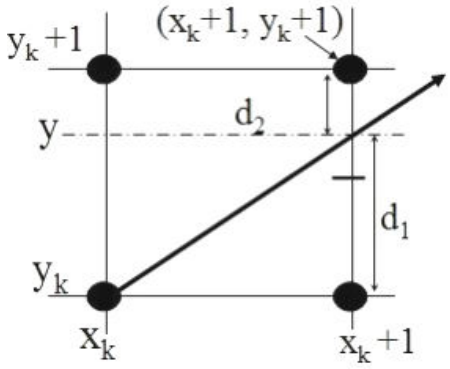
X	Y	X-Plot	Y-Plot	(X,Y)
0	0	0	0	(0,0)
1	1	1	1	(1,2)
2	2	2	2	(2,2)
3	3	3	3	(3,3)
4	4	4	4	(4,4)
5	5	5	5	(5,5)

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 (|m| ≤ 1)

Pixel position along the line path are 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 k^{th} 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 path as d_1 and d_2 as 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 $d_1 - d_2 = [m(x_k + 1) + b - y_k] - [(y_k + 1) - m(x_k + 1) - b]$
$= m(x_k + 1) + b - y_k - (y_k + 1) + m(x_k + 1) + b$
$= 2m(x_k + 1) - (y_k + 1) - y_k + 2b$
$= 2m(x_k + 1) - 2y_k + 2b - 1$

A decision parameter p_k for the k^{th} step in the line algorithm can be obtained by substituting $m = \frac{\Delta y}{\Delta x}$ in above eq^n and defining

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

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).

At step k+1, p_{k+1} is evaluated as:

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

We get,

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

Since we are sampling at X-direction, we have $x_{k+1} = x_k + 1$

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

The term y_k is either 0 or 1 depending upon the sign of p_k.

So,

if p_k is negative

$y_{k+1} = y_k$ and $P_{k+1} = p_k + 2\Delta y$

otherwise

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

Q. If the starting point is (x_0 , y_0), then what will be the value of initial decision parameter (P_0) when |m| ≤ 1?

We have $P_k = \Delta x(d_1 - d_2)$

$P_k = \Delta x\,(2m(x_k + 1) - 2y_k + 2b - 1)$

$P_k = \Delta x\,(2mx_k + 2m + 2b - 2y_k - 1)$

$P_k = \Delta x\,\{2(mx_k + b - y_k) + 2m - 1)$

The line y = mx + b must passes through the initial point (x_0, y_0) , $(mx_k + b - y_k) = 0$, we have

$P_0 = \Delta x\,(2m - 1)$

$P_0 = \Delta x\,(2\,\Delta y/\Delta x - 1)$

$P_0 = 2\Delta y - \Delta x$, which is the initial decision parameter.

Conclusion for Bresenham's Line drawing Algorithm for slope, |m| ≤ 1

a) Calculate the starting value of the decision parameter: $P_0 = 2\Delta y - \Delta x$.

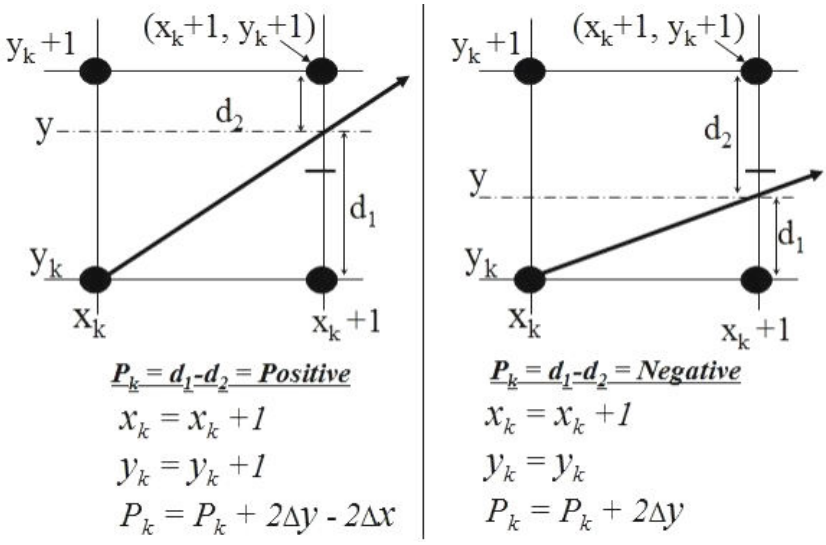
b) At each x_k along the line, starting at k = 0, perform the following test.

If $P_k > 0$,

$P_{k+1} = P_k + 2\Delta y - 2\Delta x$, and $y_{k+1} = y_k + 1$ & $x_{k+1} = x_k + 1$

Else If $P_k < 0$,

$P_{k+1} = P_k + 2\Delta y$, and $y_{k+1} = y_k$ & $x_{k+1} = x_k + 1$



Like wise,

Conclusion for Bresenham's Line drawing Algorithm for slope, |m| >1

a) Calculate the starting value of the decision parameter: $P_0 = 2\Delta x - \Delta y$

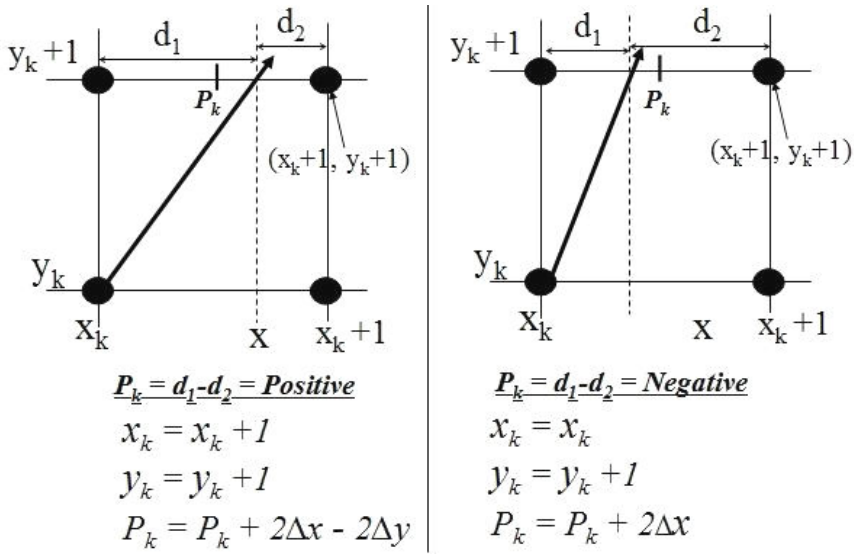
b) At each y_k along the line, starting at k = 0, perform the following test.

If $P_k > 0$,

$P_{k+1} = P_k + 2\Delta x - 2\Delta y$, and $x_{k+1} = x_k + 1$ & $y_{k+1} = y_k + 1$

Else If $P_k < 0$,

$P_{k+1} = P_k + 2\Delta x$, and $x_{k+1} = x_k$ & $y_{k+1} = y_k + 1$



Q. Digitize the line with end points (20, 10) and (30, 18) using BLA.

Solution

Here, Starting point of line = (x1, y1) = (20, 10)

Ending point of line = (x2, y2) = (30, 18)

Thus, slope of line,

$m = \Delta y / \Delta x = y2-y1 / x2-x1 = (18-10) / (30-20) = 8/10$

As the given points, it is clear that the line is moving left to right,

$|m| = 0.8 \leq 1$

Thus,

The initial decision parameter (P_0) = $2\Delta y - \Delta x = 2*8 - 10 = 6$

Since, for the Bresenham's Line drawing Algorithm of slope, $|m| \leq 1$, we have

If $P_k > 0$,

$P_{k+1} = P_k + 2\Delta y - 2\Delta x$, and $y_{k+1} = y_k + 1$ & $x_{k+1} = x_k + 1$

Else If $P_k < 0$,

$P_{k+1} = P_k + 2\Delta y$, and $y_{k+1} = y_k$ & $x_{k+1} = x_k + 1$

k	P_k	X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1})
0.	6	20+1 = 21	11	(21, 11)
1.	6 + 2*8 -2*10 = 2	21+1 = 22	12	(22, 12)
2.	2 + 2*8 -2*10 = -2	22+1 = 23	12	(23, 12)
3.	-2 + 2*8 = 14	23+1 = 24	13	(24, 13)
4.	14 + 2*8 -2*10 = 10	24+1 = 25	14	(25, 14)
5.	10 + 2*8 -2*10 = 6	25+1 = 26	15	(26, 15)
6.	6 + 2*8 -2*10 = 2	26+1 = 27	16	(27, 16)
7.	2 + 2*8 -2*10 = -2	27+1 = 28	16	(28, 16)
8.	-2 + 2*8 = 14	28+1 = 29	17	(29, 17)
9.	14 + 2*8 -2*10 = 10	29+1 = 30	18	(30, 18)

Q. Apply Bresenham's algorithm to draw a line from (20,15) and end point is (30,30)

Solution

Starting Point (x_0,y_0) = (20, 15)

Ending Point (x_n, y_n) = (30, 30) such that scanning takes place from left to right

Slope m = (30-15)/ (30-20) = 1.5 i.e. |M|>1

Here: Δx= 30-20 = 10

Δy= 30-15 = 15

2Δx = 2 * 10 = 20

2Δy = 2 * 15 = 30

Now from Bresenhams Algorithm for |M|>1, and scanning from left to right

Initial Decision Parameter:

p0 = 2Δx – Δy = 20-15= 5

if(pk<0) then

Xk+1 = Xk

Yk+1 = Yk+1

Pk+1 = Pk + 2Δx

otherwise

Xk+1 = Xk+1

Yk+1 = Yk+1

Pk+1 = Pk + 2Δx - 2Δy

K	P_k	(X_{k+1}, Y_{k+1})
0	5	(21,16)
1	-5	(21,17)
2	15	(22,18)
3	5	(23,19)
4	-5	(23,20)
5	15	(24,21)
6	5	(25,22)
7	-5	(25,23)
8	15	(26,24)
9	5	(27,25)
10	-5	(27,26)
11	15	(28,27)
12	5	(29,28)

13	-5	(29,29)
14	15	(30,30)

Q. Apply Bresenham's algorithm to draw a line from (-3,0) and end point is (4,4)

Solution

Starting Point $(x_0,y_0) = (-3, 0)$

Ending Point $(x_n, y_n) = (4,4)$ such that scanning takes place from left to right

Slope m = (4-0)/ (4+3) = 4/ 5 = 0.57 i.e. |M|<1

Here:

$\Delta x = 4 + 3 = 7$

$\Delta y = 4 - 0 = 4$

$2\Delta x = 2 * 7 = 14$

$2\Delta y = 2 * 4 = 8$

Now from Bresenhams Algorithm for |M|<1, and scanning from left to right

Initial Decision Parameter: $P_0 = 2\Delta y – \Delta x$

if(p_k<0) then

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k$

$P_{k+1} = P_k + 2\Delta y$

otherwise

$X_{k+1} = X_k+1$ and

$Y_{k+1} = Y_k+1$ and

$P_{k+1} = P_k + 2\Delta y - 2\Delta x$

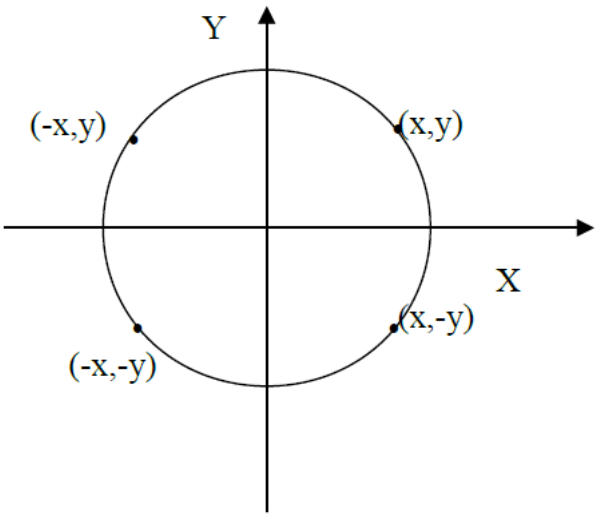
K	P_k	(X_{k+1}, Y_{k+1})
0	1	(-2,1)
1	-5	(-1,1)
2	3	(0,2)
3	-3	(1,2)
4	5	(2,3)
5	-1	(3,3)
6	7	(4,4)

Advantage of BLA over DDA

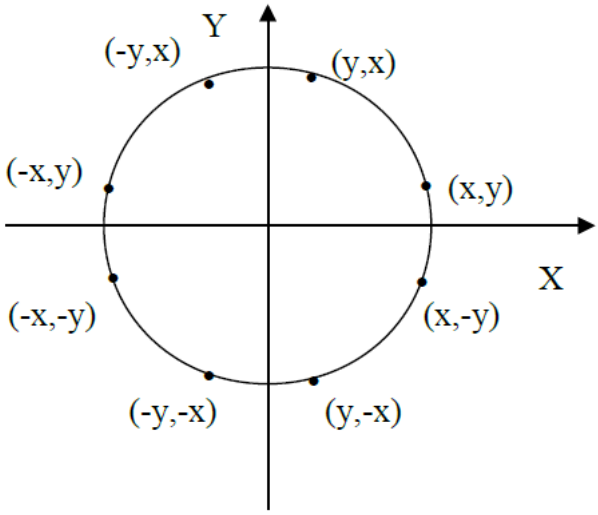
- ➢ In DDA algorithm each successive point is computed in floating point, so it requires more time and more memory space. While in BLA each successive point is calculated in integer value or whole number. So it requires less time and less memory space.
- ➢ In DDA, since the calculated point value is floating point number, it should be rounded at the end of calculation but in BLA it does not need to round, so there is no accumulation of rounding error.
- ➢ Due to rounding error, the line drawn by DDA algorithm is not accurate, while in BLA line is accurate.
- ➢ DDA algorithm cannot be used in other application except line drawing, but BLA can be implemented in other application such as circle, ellipse and other curves.

Symmetry in circle scan conversion

Symmetry simply means the exact correspondence on either side of a dividing line, plane , center or axis. In circle, we can reduce the time required for circle generation by using the symmetries. 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

Mid-point circle Algorithm

Let us consider a circle centered at origin. Assume that (x, y) is any point on the circle we can easily compute other seven points on the circle. We can generate all pixel positions around a circle by calculating only the points within the first octants & positions in the other seven octants are obtained by symmetry.

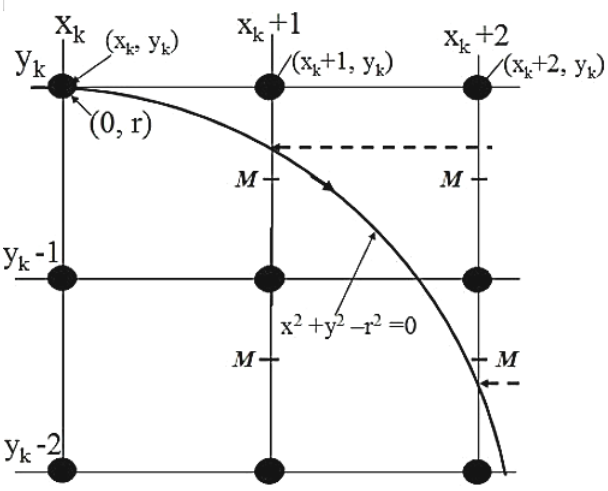
For a given radius r, we have to calculate all the pixel position from (0, r).

To apply the midpoint method, we define a circle function:

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

The relative position of any point (x, y) can be determined by checking the sign of the circle function as: -

i. $f_{circle}(x, y) < 0$, the point (x, y) is inside the circle boundary.
ii. $f_{circle}(x, y) = 0$, the point (x, y) is on the circle boundary.
iii. $f_{circle}(x, y) > 0$, the point (x, y) is outside the circle boundary.



Suppose, (x_k, y_k) is the pixel plotted, then the next pixel (x_{k+1}, y_{k+1}) will be either (x_k+1, y_k) or (x_k+1, y_k-1).

Here,

the mid-point (m) = $(x_k+1, y_k-1/2)$.

Now, decision parameter $(P_k) = f_{circle}(x_k+1, y_k-1/2) = (x_k+1)^2 + (y_k-1/2)^2 - r^2$, is the circle function evaluated at midpoint.

Also, the next pixel to plot will either be $(x_{k+1}+1, y_{k+1})$ or $(x_{k+1}+1, y_{k+1}-1)$

Also, the midpoint here is $(x_{k+1}+1, y_{k+1}-1/2)$

And the next decision parameter $P_{k+1} = f_{circle}(x_{k+1}+1, y_{k+1}-1/2)$

$$P_{k+1} = (x_{k+1}+1)^2 + (y_{k+1}-1/2)^2 - r^2$$

$$P_{k+1} = \{(x_k+1)+1\}^2 + (y_{k+1}-1/2)^2 - r^2$$

$$P_{k+1} = (x_k+1)^2 + 2(x_k+1) + 1 + (y_{k+1}-1/2)^2 - r^2$$

Now, subtracting P_{k+1} and P_k, we have

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

$$= 2(x_k+1) + 1 + y_{k+1}^2 - y_{k+1} + \tfrac{1}{2}^2 - y_k^2 + y_k - \tfrac{1}{2}^2$$

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

If $P_k < 0$, the midpoint is inside the circle and the pixel on the scan line y_k closer to the circle boundary. Otherwise, the midpoint is outside or on the circle boundary, and we select the pixel on scan line $y_k - 1$.

Case $P_k < 0$:

$$x_{k+1} = x_k + 1$$
$$y_{k+1} = y_k$$
$$P_{k+1} = P_k + 2(x_k+1) + 1$$
$$P_{k+1} = P_k + 2x_{k+1} + 1$$

Case $P_k >= 0$:

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

Q. Calculate the initial decision parameter (P_0) for circle

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

Here, the next pixel will either be (1, r) or (1, r-1) where the midpoint is (1, r- ½). Thus, the initial decision parameter is given by:

$$P_0 = f_{circle}(1, r-\tfrac{1}{2})$$
$$= 1^2 + (r - \tfrac{1}{2})^2 - r^2$$
$$= 5/4 - r$$
$$\approx 1 - r$$

Thus, $P_0 = 1 - r$

Conclusion:

1. Calculate the initial decision parameter $(P_0) = 1 - r$
2. If $P < 0$
   Plot (x_k+1, y_k)
   $P_{k+1} = P_k + 2x_{k+1} + 1$
   Else ($P >= 0$)
   Plot $(x_k+1, y_k - 1)$
   $P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$

Q. Digitized the circle with radius 10

Solution:

Initial Point = $(X_o, Y_o) = (0, r) = (0, 10)$ and origin = (0,0)

Initial decision parameter $(p_0) = 1 - r$
$= 1 - 10$
$= -9$

From midpoint circle algorithm we have

If $P < 0$
Plot (x_k+1, y_k)
$P_{k+1} = P_k + 2x_{k+1} + 1$

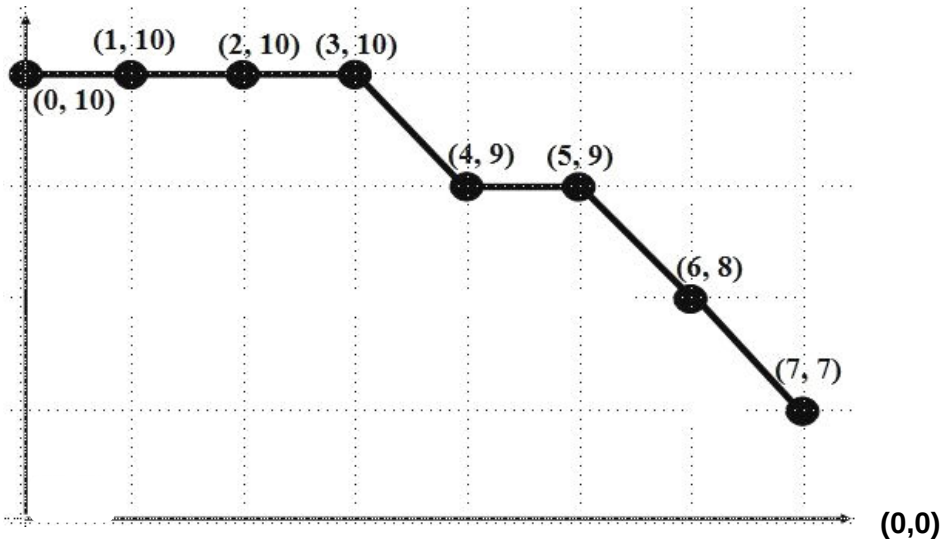
Else (P >= 0)

Plot (x_k +1, y_k - 1)

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

Now, the successive decision parameter values and position along the circle path are determined as follow.

K	P_k	X_{k+1}	Y_{K+1}	(X_{K+1}, Y_{k+1})
0	-9	1	10	(1,10)
1	-9*2*1+1=-6	2	10	(2,10)
2	-6 + 2*2+1=-1	3	10	(3,10)
3	-1 + 2 * 3 + 1 = 6	4	9	(4,9)
4	6 + 2 * 4 – 2 * 9 + 1 =-3	5	9	(5,9)
5	-3 + 2 * 5 + 1= 8	6	8	(6,8)
6	8 + 2 * 6 – 2 * 8 + 1 = 5	7	7	(7,7)
7	5+ 2 * 7-2 * 7 + 1 =6	8	6	[stop and discard this point since x>y



Q. Digitized a circle with radius r=12 and centered at (250,300)

Solution:

> Note: When center is not origin, we first calculate the octants points of the circle in the same way as the center at origin then add the given circle center on each calculated pixels.

Here, Initial Point $(x_0,y_0) = (0,r) = (0,12)$ and origin=(0,0)

Initial decision parameter $(p_0) = 1 - 12 = 1 - 12 = -11$

From midpoint circle algorithm we have

If P < 0

Plot (x_k +1, y_k)

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

Else (P >= 0)

Plot (x_k +1, y_k – 1)

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

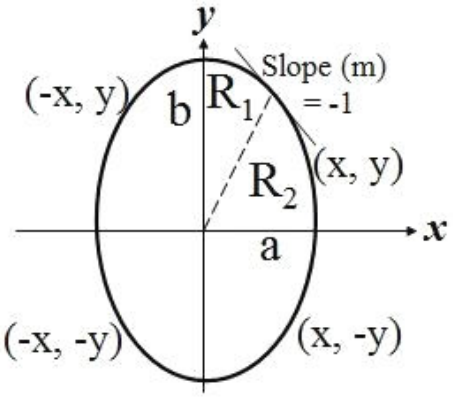
Now, the successive decision parameter values and position along the circle path are determined as follow.

K	P_k	X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1}) at (0,0)	(X_{k+1}, Y_{k+1}) at (250,300)
0	-11	1	12	(1,12)	(250+1, 300+12)=(251,312)
1	-11 + 2* 1 + 1 = -8	2	12	(2,12)	(250+2,300+12) =(252,312)
2	-8 + 2 * 2 + 1 = -3	3	12	(3,12)	(250+3,300+12) =(253,312)
3	-3 + 2 * 3 + 1=4	4	11	(4,11)	(250+4,300+11) =(254,311)
4	4 + 2 * 3 – 2 * 11 + 1= -9	5	11	(5,11)	(250+5,300+11) =(255,311)
5	-9 + 2 * 5 + 1 = 2	6	10	(6,10)	(250+6,300+10) =(256,310)

6	2 + 2*6 – 2* 10 + 1 = -5	7	10	(7,10)	(250+7,300+10) =(257,310)
7	-5 + 2 * 7 + 1 = 10	8	9	(8,9)	(250+8,300+9) =(258,309)
8	10+ 16 – 18 + 1= 9	9	8	(Discard and stop since X>Y)	

Midpoint Ellipse Algorithm

Our approach here is similar to that used in displaying a raster circle but the ellipse has 4-way symmetry. The midpoint ellipse method is applied throughout the first quadrant in two parts or region as shown in figure.



The region-1 just behaves as the circular property and the region-2 is slightly straight curve.

We have,

The equation of ellipse, whose centre at (0, 0) is

$$x^2 / a^2 + y^2 / b^2 = 1$$

Hence, we define the ellipse function for centre at origin (0, 0) as:

$$F_{ellipse}(x, y) = x^2 b^2 + y^2 a^2 - a^2 / b^2$$

This has the following properties:

$F_{ellipse}(x, y)$
- < 0, if (x, y) is inside the ellipse boundary
- = 0, if (x, y) is on the ellipse boundary
- > 0, if (x, y) is outside the ellipse boundary

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

$$2 x b^2 + 2 y a^2 * dy / dx = 0$$

Or

$$dy / dx = - 2xb^2 / 2ya^2$$

At the boundary between region 1 and region 2, $dy / dx = - 1$ and $2 x b^2 = 2 y a^2$. Therefore, we move out of region 1 whenever $2xb^2 >= 2ya^2$

For Region – 1: Condition ($2xb^2 < 2ya^2$)

Assuming that the position (x_k , y_k) has been plotted, we determine next position (x_{k+1} , y_{k+1}) as either (x_k+1 , y_k) or (x_k+1 , y_k-1) by evaluating the decision parameter $P1_k$ as:

$$P1_k = F_{ellipse}(x_k +1, y_k – ½)$$
$$= b^2(x_k +1)^2 + a^2 (y_k – ½)^2 – a^2b^2$$

At next sampling position, the decision parameter will be

$$P1_{k+1} = F_{ellipse}(x_{k+1} +1, y_{k+1} – ½)$$
$$= b^2(x_{k+1} +1)^2 + a^2 (y_{k+1} – ½)^2 – a^2b^2$$
$$= b^2\{(x_k +1) +1\}^2 + a^2(y_{k+1} – ½)^2 – a^2b^2$$
$$= b^2\{(x_k +1)^2 + 2(x_k +1) + 1\} + a^2(y_{k+1} – ½)^2 – a^2b^2$$

Now, $P1_{k+1} – P1_k = \left(b^2\{(x_k +1)^2 + 2(x_k +1) + 1\} + a^2(y_{k+1} – ½)^2 – a^2b^2\right) - \left(b^2(x_k +1)^2 + a^2 (y_k – ½)^2 – a^2b^2\right)$

$$= \left(b^2 (x_k +1)^2 + 2b^2 (x_k +1) + b^2 + a^2(y_{k+1})^2 – a^2 y_{k+1} + a^2 ¼ – a^2b^2\right) - \left(b^2(x_k +1)^2 + a^2 y_k^2 – a^2 y_k + a^2 ¼ – a^2b^2\right)$$
$$= 2b^2 (x_k +1) + b^2 + a^2\{((y_{k+1})^2 – y_k^2) - (y_{k+1} – y_k)\}$$

If $P1_k < 0$, the midpoint is inside the ellipse and the pixel on the scan line y_k closer to the ellipse boundary. Otherwise, the midpoint is outside or on the ellipse boundary, and we select the pixel on scan line $y_k -1$.

Case: $P1_k < 0$

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

$$P1_{k+1} = P1_k + 2b^2(x_{k+1}) + b^2$$

Case I: $P1_k >= 0$

$$x_{k+1} = x_k + 1$$
$$y_{k+1} = y_k - 1$$
$$P1_{k+1} = P1_k + 2b^2(x_k+1) + b^2 + a^2\{((y_{k+1})^2 - y_k^2) - (y_{k+1} - y_k)\}$$
$$= P1_k + 2b^2(x_k+1) + b^2 + a^2((y_k - 1)^2 - y_k^2) - a^2(y_k - 1 - y_k)\}$$
$$= P1_k + 2b^2(x_k+1) + a^2[(y_k^2 - 2y_k + 1 - y_k^2)] - a^2(y_k - 1 - y_k) + b^2$$
$$= P1_k + 2b^2(x_k+1) - 2a^2 y_k + a^2 + a^2 + b^2$$
$$= P1_k + 2b^2(x_k+1) - 2a^2(y_k - 1) + b^2$$
$$= P1_k + 2b^2 x_{k+1} - 2a^2 y_{k+1} + b^2$$

Initial decision parameter ($P1_0$) for region-1 of ellipse

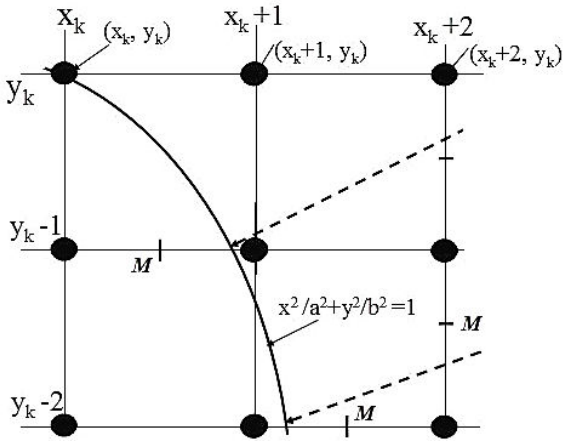
The initial decision parameter is obtained by evaluating the ellipse function at the start position $(x_0, y_0)=(0, b)$.

Here, the next pixel will either be (1, b) or (1, b-1) where the midpoint is (1, b- ½). Thus, the initial decision parameter is given by:

$$P1_0 = F_{ellipse}(1, b - ½)$$
$$= b^2 + a^2(b - ½)^2 - a^2 b^2$$
$$= b^2 - a^2 b + a^2/4$$

For Region – 2: Condition ($2xb^2 >= 2ya^2$)

Assuming that the position (x_k, y_k) has been plotted, we determine next position (x_{k+1}, y_{k+1}) as either (x_k+1, y_k-1) or (x_k, y_k-1) by evaluating the decision parameter $P2_k$ as:



$$P2_k = F_{ellipse}(x_k + ½, y_k - 1)$$
$$= b^2(x_k + ½)^2 + a^2(y_k - 1)^2 - a^2b^2$$

At next sampling position, the decision parameter will be

$$P2_{k+1} = F_{ellipse}(x_{k+1} + ½, y_{k+1} - 1)$$
$$= b^2(x_{k+1} + ½)^2 + a^2(y_{k+1} - 1)^2 - a^2b^2$$

Now, subtracting $P2_k$ from $P2_{K+1}$, we have

$$P2_{k+1} - P2_k = \{b^2(x_{k+1} + ½)^2 + a^2(y_{k+1}-1)^2 - a^2b^2\} - \{b^2(x_k + ½)^2 + a^2(y_{k-1})^2 - a^2b^2\}$$
$$= \{b^2(x_{k+1} + ½)^2 + a^2(y_{k+1}-1)^2 - a^2b^2 - b^2(x_k + ½)^2 - a^2(y_{k-1})^2 + a^2b^2\}$$
$$= \{b^2(x_{k+1} + ½)^2 + a^2(y_{k+1}-1)^2 - b^2(x_k + ½)^2 - a^2(y_{k-1})^2\}$$
$$= b^2(x_{k+1} + ½)^2 + a^2\{(y_k-1)-1\}^2 - b^2(x_k + ½)^2 - a^2(y_{k-1})^2$$
$$= b^2x_{k+1}^2 + b^2x_{k+1} + b^2/4 + a^2(y_k-1)^2 - 2a^2(y_k-1) + a^2 - b^2x_k^2 - b^2x_k - b^2/4 - a^2(y_{k-1})^2$$
$$= a^2 + b^2(x_{k+1}^2 - x_k^2) - 2a^2(y_k-1) + b^2(x_{k+1} - x_k)$$

If $P2_k<0$, the midpoint is inside the ellipse and the pixel on the scan line x_k is closer to the ellipse boundary. Otherwise, the midpoint is outside or on the ellipse boundary, and we select the pixel on scan line x_k+1.

Case : $P2_k \le 0$

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

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

$P2_{k+1} = P2_k + b^2 [(x_{k+1}^2 - x_k^2) + (x_{k+1} - x_k)] - 2a^2(y_k - 1) + a^2$

$= P2_k + b^2 [(x_k^2 + 2x_k + 1 - x_k^2) + (x_k + 1 - x_k)] - 2a^2(y_k - 1) + a^2$

$= P2_k + b^2 [2x_k + 1 + 1] - 2a^2(y_k - 1) + a^2$

$= P2_k + 2b^2(x_k + 1) - 2a^2(y_k - 1) + a^2$

$= P2_k + 2b^2 x_{k+1} - 2a^2 y_{k+1} + a^2$

Case $P2_k > 0$

$x_{k+1} = x_k$

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

$P2_{k+1} = P2_k + b^2 [(x_{k+1}^2 - x_k^2) + (x_{k+1} - x_k)] - 2a^2(y_k - 1) + a^2$

$= P2_k - 2a^2(y_k - 1) + a^2$

$= P2_k - 2a^2 y_{k+1} + a^2$

Initial decision parameter ($P2_0$) for region-2 of ellipse

When we enter region 2, the initial position (x_o, y_0) is taken as the last position selected in region 1 and the initial derision parameter in region 2 is given by:

$P2_0 = F_{ellipse}(x_0 + 1/2, y_0 - 1)$

$= b^2 (x_0 + 1/2)^2 + a^2 (y_0 - 1)^2 - a^2 b^2$

Q. Example: Given input ellipse parameters $r_x = a = 8$ and $r_y = b = 6$, we illustrate the steps in the midpoint ellipse algorithm by determining raster positions along the ellipse path in the first quadrant.

Solution:

Given a = 8 and b= 6

For Region 1

The initial point for the ellipse on the origin $(x_o, y_o) = (0, b) = (0,6)$

Calculation the initial decision parameter

$P1_0 = b^2 - a^2 b + a^2/4$

$= 6^2 - 6 * (8)^2 + (8^2 / 4)$

$= -332$

From midpoint algorithm, for Region 1 we know,

If($P1_k < 0$) then

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

$y_{k+1} = y_k$

$P1_{k+1} = P1_k + 2b^2 x_{k+1} + b^2$

Else (i.e. $P1_k >= 0$)

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

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

$P1_{k+1} = P1_k + 2b^2 x_{k+1} - 2a^2 y_{k+1} + b^2$

And stop when $2x_{k+1}b^2 >= 2y_{k+1}a^2$

Now successive decision parameter values and positions along the ellipse path are calculated using midpoint method as

K	$P1_k$	(X_{k+1}, Y_{K+1})	$2b^2(x_{k+1})$	$2a^2 y_{k+1}$
0	-332	(1, 6)	72	768
1	-224	(2, 6)	144	768
2	-44	(3, 6)	216	768

3	208	(4, 5)	288	640
4	-108	(5, 5)	360	640
5	288	(6, 4)	432	512
6	244	(7, 3)	504	384

Since $2x_{k+1}b^2 >= 2y_{k+1}a^2$ so we stop here for region 1.

[note: (7, 3) is the starting point for region 2]

Now for Region-2,

From midpoint algorithm, for Region 2 we know,

Initial Decision Parameter:

$$P2_0 = b^2 (x_0 + 1/2)^2 + a^2 (y_0 - 1)^2 - a^2 b^2$$

If(P_k<0) then

$X_{k+1} = X_k + 1$

$Y_{k+1} = Y_k - 1$

$P2_{k+1} = P2_k + 2b^2 x_{k+1} - 2a^2 y_{k+1} + a^2$

Else (i.e. P_k >=0)

$X_{k+1} = X_k$

$Y_{k+1} = Y_k - 1$

$P2_{K+1} = P2_k - 2a^2 y_{k+1} + a^2$

So, the initial point for region 2 is (x_0, y_0) = (7, 3) and the initial decision parameter is

$$P2_0 = b^2 (x_0 + 1/2)^2 + a^2 (y_0 - 1)^2 - a^2 b^2$$

$$= 36 * (7 + 0.5)^2 + 64 * (3-1)^2 - 64 * 36$$

$$= -23$$

The remaining pixels in region two for first quadrant are than calculated as

K	$P2_k$	(X_{k+1}, Y_{K+1})
0	-23	(8,2)
1	361	(8,1)
2	297	(8,0)

So remaining points in other quadrants can be calculated using the symmetry property of ellipse.

Filled Area Primitive

A standard output primitive in general graphics 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. The main idea behind the 2D or 3D object filling procedure is that it provides us more realism on the object of interest.

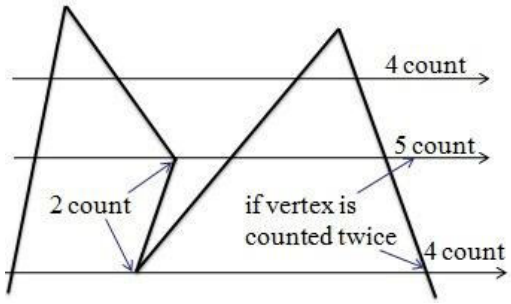
There are two basic approaches to area filling in raster systems.

- ➢ One way to fill an area is to determine the overlap intervals for scan lines that crosses 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 this method, the scan line must be even. At two edge connecting point called the vertex, we count the scan line two to handling the problem for scan line passing through vertex, but this consideration also may create problem for some instant as shown in figure. To handle such problem shown by count-5, we keep the vertex blank and hence the count become 1, so that overall count in that scan line become even as shown in figure



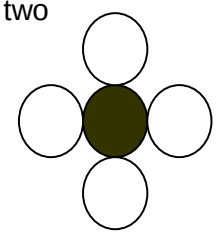
Boundary-fill Algorithm:

In Boundary filling algorithm starts at a point called seed pixel 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.

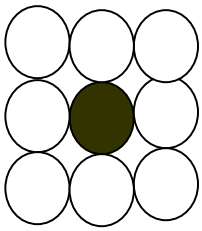
Starting from (x,y), the procedure tests neighboring positions to determine whether they are of boundary color. If not, they are painted with the fill color, and their neighbors are tested. This process continues until all pixel up to the boundary color area have tested.

The neighboring pixels from current pixel are proceeded by two method:

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



4- Connected



8- Connected

Algorithm for 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_fill4(x-1,y, b_color, fill_color);
                Boundary_fill4(x+1,y, b_color, fill_color);
                Boundary_fill4(x,y-1, b_color, fill_color);
                Boundary_fill4(x,y+1, b_color, fill_color);
        }
}
```

Algorithm for 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 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;
        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);
       }
}
```

Boundary Fill Algorithm	Flood Fill Algorithm
Area filling is started inside a point with in a boundary region and fill the region with in the specified color until it reaches the the boundary.	Area filling is started from a point and it replaces the old color with the new color
It is used in interactive packages where we can specify the region boundary	It is used when we cannot specify the region boundary
It is less time consuming	It consumes more time
It searches for boundary.	It searches for old color

Chapter 4

Two Dimensional Transformation

In many cases a complex picture can always be treated as a combination of straight line, circles, ellipse etc., and if we are able to generate these basic figures, we can also generate combinations of them. Once we have drawn these pictures, the need arises to transform these pictures.

Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, etc. When a transformation takes place on a 2D plane, it is called **2D transformation.**

Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

The three basic transformations are

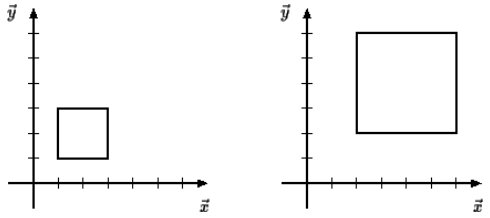
a) Translation

b) Rotation

c) Scaling.

Other transformation includes reflection and shear.

For each cases of transformation, we consider the reference / pivot point is origin, so if we have to do these transformations for a point P(x, y) about any arbitrary point (x_r, y_r), then we have to first shift the given point to the origin and then perform required transformation and finally shift to that arbitrary point position.



Geometric transformations involve taking a **preimage** and transforming it in some way to produce an **image**.

There are three basic classes of transformations:

1. **Rigid body**

- Preserves distance and angles.
- Examples: translation and rotation.

2. **Conformal**

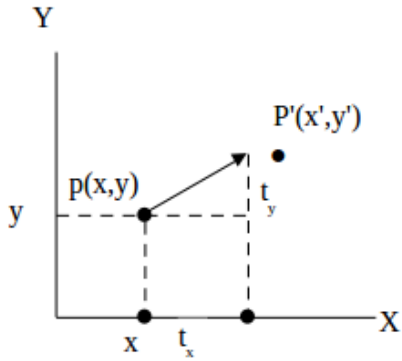
- Preserves angles.
- Examples: translation, rotation, and uniform scaling.

3. **Affine**

- Preserves parallelism. Lines remain lines.
- Examples: translation, rotation, scaling, shear, and reflection

a) Translation:

Repositioning of object along a straight-line path from one coordinate location to another is called translation. We translate a two-dimensional point by adding translation distances t_x , and t_y , to the original coordinate position (x, y) to move the point to a new position (x ' , y') as:



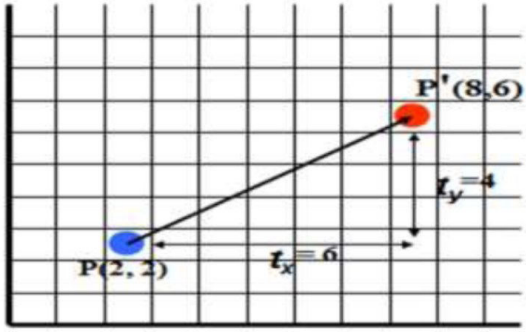
$$x' = x + t_x$$
$$y' = x + t_y$$

The translation distance pair (t_x,t_y) is known as translation vector or shift vector. We can express translation equations as matrix representations as

P= $\begin{bmatrix} x \\ y \end{bmatrix}$ P'= $\begin{bmatrix} x' \\ y' \end{bmatrix}$ T= $\begin{bmatrix} t_x \\ t_y \end{bmatrix}$ ∴P'=P+T

For Example:

Given a point P with original position (2,2). Then after performing translation operation with t_x = 6 and t_y = 4, we get new transformed coordinate P' with coordinate (8,6).



b) Rotation

A two-dimensional rotation is applied to an object by repositioning it along a circular path in the xy plane. To generate a rotation, we specify a rotation angle 'θ' and the position (x_r , y_r ,) of the rotation point (or pivot point) about which the object is to be rotated.

- ➢ + Value for 'θ' define counter-clockwise rotation about a point
- ➢ - Value for 'θ' defines clockwise rotation about a point

Let (x, y) is the original point, 'r' the constant distance from origin, & 'Φ' the original angular displacement from x-axis. Now the point (x, y) is rotated through angle 'θ' in a counter clock wise direction

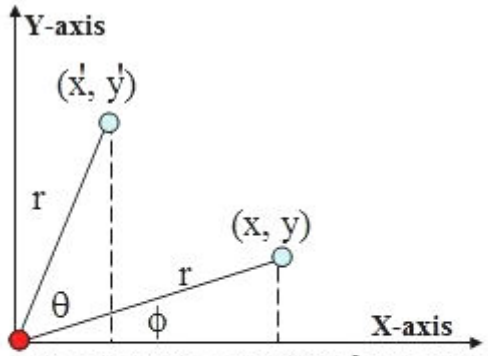


Fig. Rotating a point from position (x, y) to position (x', y') through an angle θ about rotation point (0, 0). The original angular displacement of the point from the x axis is ϕ.

we can express the transformed coordinates in terms of 'Φ' and 'θ' as

x' = r cos(Φ + θ) = r cosΦ . cosθ - r sin Φ. sinθ ....(i)

y' = r sin(Φ + θ) = r cosΦ . sinθ + r sinΦ . cosθ ....(ii)

We know that original coordinates of point in polar coordinates are

x = r cos Φ

y = r sin Φ

Substituting these values in (i) and (ii), we get,

x' = x cosθ - y sinθ

y' = x sinθ + y cosθ

So using column vector representation for coordinate points the matrix form would be P' = R . P

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} \text{Cos}\theta & -\text{Sin}\theta \\ \text{Sin}\theta & \text{Cos}\theta \end{pmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Rotation of a point about an arbitrary pivot position

- Translate the point (x_r,y_r) and P(x,y) by translation vector $(-x_r , -y_r)$ which translates the pivot to origin and P(x,y) to $(x-x_r , y-y_r)$.
- Now apply the rotation equations when pivot is at origin to rotate the translated point $(x-x_r , y-y_r)$ as:

$$x_1 = (x - x_r)\cos\theta - (y - y_r)\sin\theta$$

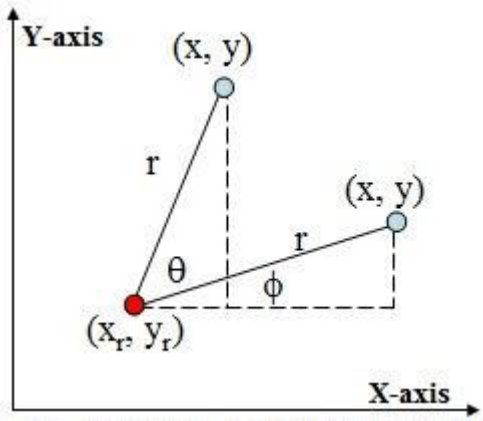
$$y_1 = (x - x_r)\sin\theta + (y - y_r)\cos\theta$$

- Re- translate the rotated point (x_1,y_1) with translation vector (x_r,y_r) which is reverse translation to original translation. Finally we get the equation after successive transformation as

$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$ 1

$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$ 2

Which are actually the equations for rotation of (x,y) from the pivot point (x_r,y_r)



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 1-\cos\theta & -\sin\theta \\ -\sin\theta & 1-\cos\theta \end{bmatrix} \begin{bmatrix} x_r \\ y_r \end{bmatrix}$$

c) Scaling

A scaling transformation alters the size of an object. This operation can be carried out for polygons by multiplying the coordinate values (x, y) of each vertex by scaling factors s_X and s_Y to produce the transformed coordinates (x', y').

- s_X scales object in 'x' direction
- s_Y scales object in 'y' direction

Thus, for equation form,

$x' = x . s_X$ and $y' = y . s_Y$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Values greater than 1 for s_X and s_Y produce enlargement
- Values less than 1 for s_X s_Y reduce size of object
- $s_X = s_Y = 1$ leaves the size of the object unchanged
- When s_X , s_Y are assigned the same value $s_X = s_Y = 4$ then a Uniform Scaling is produced.

- Scaling About arbitrary point

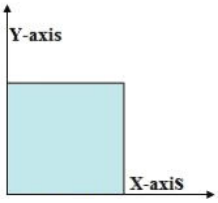
If p(x,y) be scaled to a point p'(x', y') by s_x time in X-units and s_y times in y-units about arbitrary point (x_f, y_f) then the equation for scaling is given as

$x' = x_f + s_x .(x-x_f)$

$y' = y_f + s_y .(y-y_f)$

d. Shearing

A transformation that distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other is called shear.



X-direction Shear:

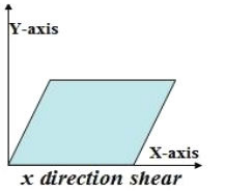
An X-direction shear relative to x-axis is produced with transformation matrix equation.

In 'x' direction,

$x' = x + s_{hx} \cdot y$

$y' = y$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & s_{hx} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Y-direction Shear:

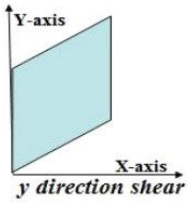
An y-direction shear relative to y-axis is produced by following transformation equations.

In 'y' direction,

$x' = x$

$y' = y + s_{hy} \cdot x$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ s_{hy} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Both direction Shear:

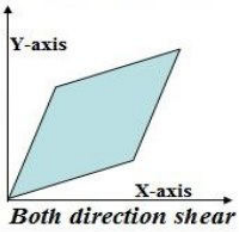
Both directions share relative to x-axis and y-axis is produced by following transformation equations

In both directions,

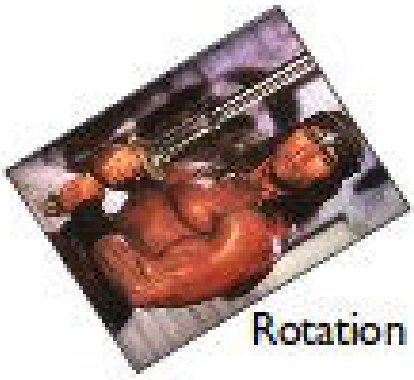
$x' = x + s_{hx} \cdot y$

$y' = y + s_{hy} \cdot x$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & s_{hx} \\ s_{hy} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Original



Rotation

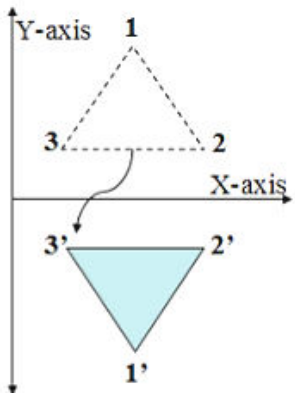
Uniform Scale

Shear

Nonuniform Scale

e. Reflection

A reflection is a transformation that produce a mirror image of an object. In 2D-transformation, reflection is generated relative to an axis of reflection. The reflection of an object to an relative axis of reflection , is same as 180^o rotation about the reflection axis.



i. **Reflection about x axis or about line y = 0**

Keeps 'x' value same but flips y value of coordinate points

$x' = x$

$y' = -y$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

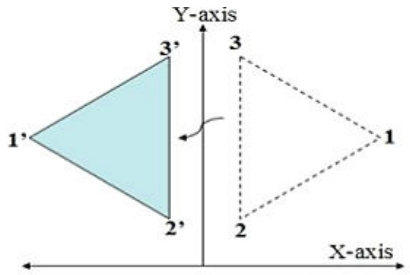
ii. **Reflection about y axis or about line x = 0**

Keeps 'x' value same but flips y value of coordinate points

$x' = -x$

$y' = y$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



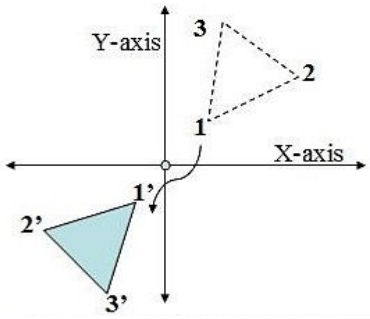
iii. **Reflection about origin**

Flip both 'x' and 'y' coordinates of a point

$x' = -x$

$y' = -y$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Q. What is Arbitrary axis?

Arbitrary axis simply means the axis chosen in any way we like within the co-ordinate system. We choose any line in space, and then put a pin into our object along that line, and transform the object around the pin.

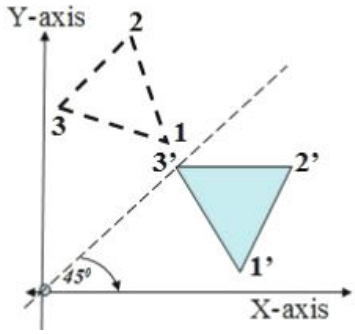
iv. **Reflection on an arbitrary axis**

The reflection on any arbitrary axis of reflection can be achieved by sequence of transformation and co-ordinate axes reflection matrices.

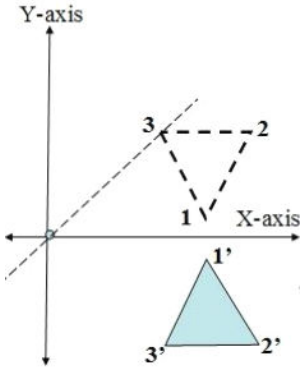
Example:

Reflection about line y = x ($\theta=45^0$)

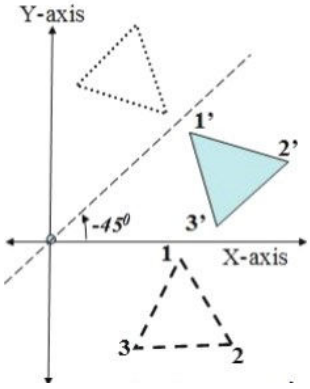
- Rotate about origin in clockwise direction by 45 degree which rotates line y = x to x-axis



- Take reflection against x-axis



- Rotate in anti-clockwise direction by same angle



Here, we have multiple transformation at once, So when more than one transformations are applied for performing a task then such transformation is called **composite transformation.**

So the composite transformation required for reflecting the given object about y=x axis is

$$T = R_{\theta=45} R_x R_{\theta=-45}$$

Forming products of transformation matrices is often referred to as a concatenation, or composition, of matrices.

Q. What is the basic purpose of composite transformation?

The basic purpose of composing transformations is to gain efficiency by applying a single

composed transformation to a point, rather than applying a series of transformation, one after another.

Q. Translate the given points (2,5) by the translating value (3,3).

Solution:

Given Point P (2, 5) and translation distance Tx= 3, and Ty= 3
We have From Translation Matrix

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

i.e.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix} + \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 8 \end{bmatrix}$$

Therefore, P(2,5) is translated to new point P'(5,8).

Q. Translate the given square having coordinate A (0,0), B (3,0), and C (3,3) and D (0, 3) by the translating value 2 in both directions.

Solution:

Given points A (0,0), B (3,0), and C (3,3) and D (0, 3) and translating value 2 (t_x=t_y=2)

We have From Translation Matrix

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

i.e.

A'= $\begin{bmatrix} 0 \\ 0 \end{bmatrix}$+ $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$= $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$

B'= $\begin{bmatrix} 3 \\ 0 \end{bmatrix}$+ $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$= $\begin{bmatrix} 5 \\ 2 \end{bmatrix}$

C'= $\begin{bmatrix} 3 \\ 3 \end{bmatrix}$+ $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$= $\begin{bmatrix} 5 \\ 5 \end{bmatrix}$

D'= $\begin{bmatrix} 0 \\ 3 \end{bmatrix}$+ $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$= $\begin{bmatrix} 2 \\ 5 \end{bmatrix}$

Hence the final co-ordinate are A' (2,2), B'(5,2), C'(5,5) and D'(2,5)

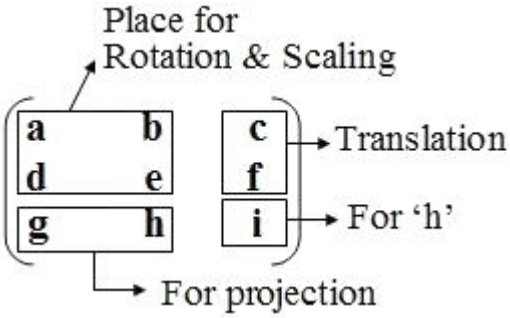
Homogeneous Co-ordinates

To perform a sequence of transformation such as translation followed by rotation and scaling, we need to follow a sequential process −

- Translate the coordinates,
- Rotate the translated coordinates, and then
- Scale the rotated coordinates to complete the composite transformation.

To shorten this process, we have to use 3×3 transformation matrix instead of 2×2 transformation matrix. To convert a 2×2 matrix to 3×3 matrix, we have to add an extra **dummy coordinate**. So, we can represent the point by 3 numbers instead of 2 numbers, which is called Homogenous Coordinate system.
In this system, we can represent all the transformation equations in matrix multiplication.

Any Cartesian point P(X, Y) can be converted to homogenous coordinates by P' (X_h, Y_h, h). The 'h' is normally set to 1. If the value of 'h' is more the one value then all the co-ordinate values are scaled by this value.



Coordinates of a point are represented as three element column vectors; transformation operations are written as 3 x 3 matrices.

For translation

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

With T (t_x, t_y) as translation matrix, inverse of this translation matrix is obtained by representing t_x, t_y with $-t_x$, $-t_y$.

For rotation

a. Counter Clock Wise (CCW):

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

b. Clock Wise (CCW):

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

For Shearing

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} S_{hx} & 0 & 0 \\ 0 & S_{hy} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Inverse shearing matrix is obtained with $1 / s_{hx}$ and $1 / s_{hy}$.

For Reflection

a) Reflection about x-axis

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

b) Reflection about y-axis

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

c) Reflection about y=x-axis

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

d) Reflection about y=-x-axis

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

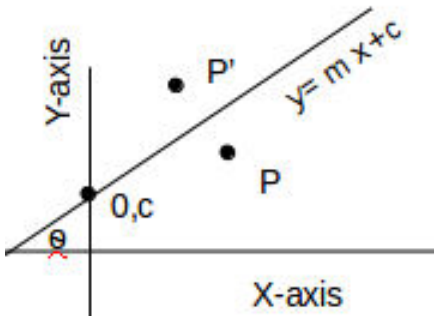
e) Reflection about any line y=mx+c

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{-(m^2-1)}{(m^2+1)} & \frac{2m}{(m^2+1)} & \frac{2mc}{(m^2+1)} \\ \frac{2m}{(m^2+1)} & \frac{(m^2-1)}{(m^2+1)} & \frac{2c}{(m^2+1)} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Q. Derive the composite matrix for reflecting an object about any arbitrary line y=mx+c.

In order to reflect an object about any line y=mx+c, we need to perform composite transformation as below

$$T = T_{(0,c)} \cdot R_{\Theta} \cdot R_{x} \cdot R_{-\Theta} \cdot T_{(0,-c)}$$



And

Slope m= tan Θ,

Also we have,

$$\cos^2\Theta = \frac{1}{\tan^2\theta+1}$$

$$= \frac{1}{m^2+1}$$

$$\cos\Theta = \frac{1}{\sqrt{m^2+1}}$$

Also we have,

$$\sin^2\Theta + \cos^2\Theta = 1$$

$$\sin^2\Theta = 1 - \cos^2\Theta$$

$$\sin^2\Theta = 1 - \frac{1}{m^2+1} = \frac{m^2+1-1}{m^2+1}$$

$$\sin\Theta = \frac{m}{\sqrt{m^2+1}}$$

So,

$$T = T_{(0,c)} \cdot R_{\Theta} \cdot R_{x} \cdot R_{-\Theta} \cdot T_{(0,-c)}$$

$$= \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & c\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & -c\\ 0 & 0 & 1\end{bmatrix}$$

$$= \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & c\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta & \sin\theta & -c\sin\theta\\ -\sin\theta & \cos\theta & -c\cos\theta\\ 0 & 0 & 1\end{bmatrix}$$

$$= \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & c\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta & \sin\theta & -c\sin\theta\\ \sin\theta & -\cos\theta & c\cos\theta\\ 0 & 0 & 1\end{bmatrix}$$

$$= \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & c\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{(m^2+1)}} & \frac{-m}{\sqrt{(m^2+1)}} & 0\\ \frac{-m}{\sqrt{(m^2+1)}} & \frac{1}{\sqrt{(m^2+1)}} & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{(m^2+1)}} & \frac{m}{\sqrt{(m^2+1)}} & \frac{-cm}{\sqrt{(m^2+1)}}\\ \frac{m}{\sqrt{(m^2+1)}} & \frac{-1}{\sqrt{(m^2+1)}} & \frac{c}{\sqrt{(m^2+1)}}\\ 0 & 0 & 1\end{bmatrix}$$

$$= \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & c\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & \frac{-2cm}{m^2+1}\\ \frac{2m}{m^2+1} & \frac{m^2-1}{m^2+1} & \frac{c-cm^2}{m^2+1}\\ 0 & 0 & 1\end{bmatrix}$$

$$= \begin{bmatrix}\frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & \frac{-2mc}{m^2+1}\\ \frac{2m}{m^2+1} & \frac{m^2-1}{m^2+1} & \frac{2c}{m^2+1}\\ 0 & 0 & 1\end{bmatrix}$$

Q. Rotate a triangle A(0,0) , B(2,2) , C(4,2) about the origin by the angle of 45 degree.

Solution:

The given triangle ABC can be represented by a matrix, formed from the homogeneous coordinates of the vertices.

$$\begin{bmatrix}0 & 2 & 4\\ 0 & 2 & 2\\ 1 & 1 & 1\end{bmatrix}$$

Also, we have

$$R_{45^0} = \begin{bmatrix}cos45^0 & -sin45^0 & 0\\ sin45^0 & cos45^0 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0\\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0\\ 0 & 0 & 1\end{bmatrix}$$

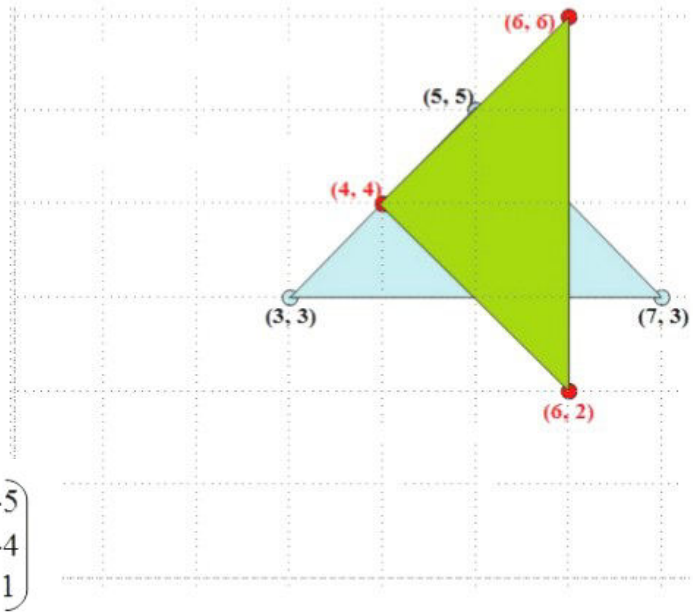
So the coordinates of the rotated triangle ABC are

$$R_{45^0}[ABC] = \begin{bmatrix}\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0\\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0\\ 0 & 0 & 1\end{bmatrix} \cdot \begin{bmatrix}0 & 2 & 4\\ 0 & 2 & 2\\ 1 & 1 & 1\end{bmatrix} = \begin{bmatrix}0 & 0 & \sqrt{2}\\ 0 & 2\sqrt{2} & 3\sqrt{2}\\ 1 & 1 & 1\end{bmatrix}$$

Hence the final co-ordinate points are A'(0,0), B'(0, 2√2) and C'(√2, 3√2).

Q. Rotate the triangle (5, 5), (7, 3), (3, 3) about fixed point (5, 4) in counter clockwise (CCW) by 90 degree.

Solution



The required steps are:

1. Translate the fixed point to origin.
2. Rotate about the origin by 90 degree.
3. Reverse the translation as performed earlier.

Thus, the composite matrix is given by

$$Com = T_{(xf, yf)} \, R_{\theta} \, T_{(-xf, -yf)}$$

$$= \begin{pmatrix} 1 & 0 & 5 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} Cos90 & -Sin90 & 0 \\ Sin90 & Cos90 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -5 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 5 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -5 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 5 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -1 & 4 \\ 1 & 0 & -5 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & -1 & 9 \\ 1 & 0 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

Now, the required co-ordinate can be calculated as:

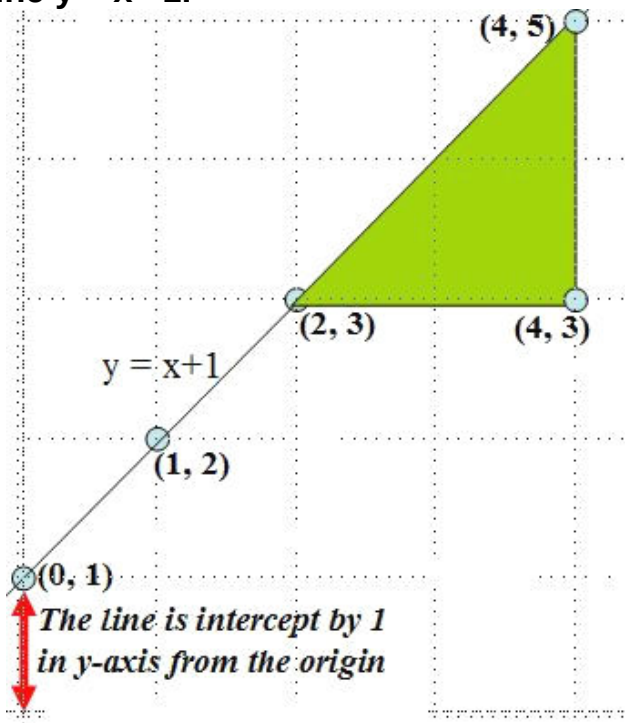
P' = Com * P

$$= \begin{pmatrix} 0 & -1 & 9 \\ 1 & 0 & -1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 & 7 & 3 \\ 5 & 3 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 4 & 6 & 6 \\ 4 & 6 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

Hence, the new required coordinate points are (4, 4), (6, 6) & (6, 2).

Q. Reflect an object (2, 3), (4, 3), (4, 5) about line y = x +1.



Here,

The given line is y = x +1.

Thus,

When x = 0, y=1

When x = 1, y=2

When x = 2, y=3

Also,

The slope of the line (m) = 1

Thus, the rotation angle (θ) = $Tan^{-1}(m) = Tan^{-1}(1)=45^0$

Here, the required steps are:

1. Translate the line to origin by decreasing the y-intercept with one.
2. Rotate the line by angle 45 0 in clockwise direction so that the given line must overlap x-axis.
3. Reflect the object about the x-axis.
4. Reverse rotate the line by angle -45 0 in counter-clockwise direction.
5. Reverse translate the line to original position by adding the y-intercept with one

Thus, the composite matrix is given by:

Com = $\mathbf{T' R_\theta' R_x R_\theta T}$

$$= \underbrace{\begin{pmatrix}1&0&0\\0&1&1\\0&0&1\end{pmatrix}}_{\text{Addition y-intercept}} \underbrace{\begin{pmatrix}\text{Cos45}&-\text{Sin45}&0\\\text{Sin45}&\text{Cos45}&0\\0&0&1\end{pmatrix}}_{\text{CCW Rotation}} \underbrace{\begin{pmatrix}1&0&0\\0&-1&0\\0&0&1\end{pmatrix}}_{\text{Reflection about x-axis}} \underbrace{\begin{pmatrix}\text{Cos45}&\text{Sin45}&0\\-\text{Sin45}&\text{Cos45}&0\\0&0&1\end{pmatrix}}_{\text{CW Rotation}} \underbrace{\begin{pmatrix}1&0&0\\0&1&-1\\0&0&1\end{pmatrix}}_{\text{Reduce y-intercept}}$$

$$= \begin{pmatrix}1&0&0\\0&1&1\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&-1/\sqrt{2}&0\\1/\sqrt{2}&1/\sqrt{2}&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&0\\0&-1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&1/\sqrt{2}&0\\-1/\sqrt{2}&1/\sqrt{2}&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&-1\\0&0&1\end{pmatrix}$$

$$= \begin{pmatrix}1&0&0\\0&1&1\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&-1/\sqrt{2}&0\\1/\sqrt{2}&1/\sqrt{2}&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&0\\0&-1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&1/\sqrt{2}&-1/\sqrt{2}\\-1/\sqrt{2}&1/\sqrt{2}&-1/\sqrt{2}\\0&0&1\end{pmatrix}$$

$$= \begin{pmatrix}1&0&0\\0&1&1\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&-1/\sqrt{2}&0\\1/\sqrt{2}&1/\sqrt{2}&0\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&1/\sqrt{2}&-1/\sqrt{2}\\1/\sqrt{2}&-1/\sqrt{2}&1/\sqrt{2}\\0&0&1\end{pmatrix}$$

$$= \begin{pmatrix}1&0&0\\0&1&1\\0&0&1\end{pmatrix}\begin{pmatrix}0&1&-1\\1&0&0\\0&0&1\end{pmatrix} = \begin{pmatrix}0&1&-1\\1&0&1\\0&0&1\end{pmatrix}$$

Now, the required co-ordinate can be calculated as:

$$P' = Com * P$$

$$= \begin{pmatrix} 0 & 1 & -1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & 4 \\ 3 & 3 & 5 \\ 1 & 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 2 & 4 \\ 3 & 5 & 5 \\ 1 & 1 & 1 \end{pmatrix}$$

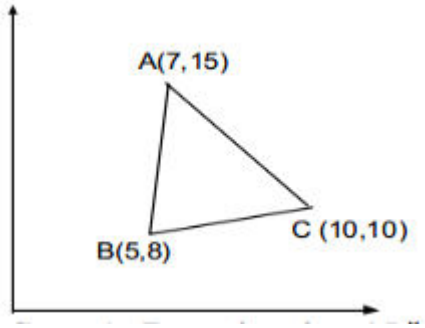
Hence, the final coordinates are (2, 3), (2, 5) & (4, 5)

Note: Composite Matrix can also be calculated as:

$$\begin{pmatrix} \frac{-(m^2-1)}{(m^2+1)} & \frac{2m}{(m^2+1)} & \frac{-2mc}{(m^2+1)} \\ \frac{2m}{(m^2+1)} & \frac{(m^2-1)}{(m^2+1)} & \frac{2c}{(m^2+1)} \\ 0 & 0 & 1 \end{pmatrix}$$

where m= 1, c=1.

Q. Rotate triangle ABC by 450 clockwise about origin and scale it by (2,3) about origin.



Here,

The steps required are:

1. Rotate by 45 clockwise
2. scale by t_x= 2 and t_y= 3

Thus the composite matrix is given by

$$com = S(2,3).R_{-45}$$

$$= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45 & Sin45 & 0 \\ -Sin45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2/\sqrt{2} & 2/\sqrt{2} & 0 \\ -3/\sqrt{2} & 3/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Now, the required co-ordinate can be calculated as:

A' = com * A

$$= \begin{bmatrix} 2/\sqrt{2} & 2/\sqrt{2} & 0 \\ -3/\sqrt{2} & 3/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 7 \\ 15 \\ 1 \end{bmatrix} =$$

B' = com * B

$$= \begin{bmatrix} 2/\sqrt{2} & 2/\sqrt{2} & 0 \\ -3/\sqrt{2} & 3/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 8 \\ 1 \end{bmatrix} =$$

C' = com * C

$$= \begin{bmatrix} 2/\sqrt{2} & 2/\sqrt{2} & 0 \\ -3/\sqrt{2} & 3/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 \\ 10 \\ 1 \end{bmatrix} =$$

Hence, the final co-ordinates are A'( ), B' ( ) and C'( ).

Q. Rotate the △ABC by 90° anti clockwise about (5,8) and scale it by (2,2) about (10,10)

Q. A mirror is placed such that it passes through (0,10) (10,0). Find the mirror image of an object (6,7), (7,6), (6,9).

Q. Show that Successive translations are additive

If two successive translations are applied then they are additive.

Proof:

If two successive translation vectors (t_{x1}, t_{y1}) and (t_{x2}, t_{y2}) are applied to a coordinate position P, the final transformed location P' is calculated with the following composite transformation as:

$$T' = T_{(x2,y2)} \cdot T_{(x1,y1)}$$

$$= \begin{bmatrix} 1 & 0 & Tx2 \\ 0 & 1 & Ty2 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & Tx1 \\ 0 & 1 & Ty1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & Tx1 + Tx2 \\ 0 & 1 & Ty1 + Ty2 \\ 0 & 0 & 1 \end{bmatrix}$$

Hence, $T_{(tx2,\ tx2)}.T_{(tx1,ty1)} = T_{(tx1+ty1,\ tx2+ty2)}$ which demonstrates that two successive translation are additive

Q. Show that Successive rotation are additive

If two successive rotations are applied, then they are additive.

Proof:

Let P be point anticlockwise rotated by angle θ_1 to point P' and again let P' be rotated by angle θ_2 to point P'', then the combined transformation can be calculated with the following composite matrix as:

$$T = R(\theta 2)\, R(\theta 1)$$

$$= \begin{bmatrix} \cos\theta 2 & -\sin\theta 2 & 0 \\ \sin\theta 2 & \cos\theta 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta 1 & -\sin\theta 1 & 0 \\ \sin\theta 1 & \cos\theta 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta 2 * cos\theta 1 - \sin\theta 2 * sin\theta 1 & -\cos\theta 2 * sin\theta 1 - sin\theta 2 * cos\theta 1 & 0 \\ \sin\theta 2 * cos\theta 1 + cos\theta 2 * sin\theta 1 & -\sin\theta 2 * sin\theta 1 + cos\theta 2 * cos\theta 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta 1 + \theta 2) & -\sin(\theta 1 + \theta 2) & 0 \\ \sin(\theta 1 + \theta 2) & \cos(\theta 1 + \theta 2) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

i.e. $\mathbf{R}_{\theta 2}.\mathbf{R}_{\theta 1} = R(\theta_1 + \theta_2)$, which demonstrates that two successive rotations are additive.

Q. Show that Successive Scaling are multiplication

If two successive Scaling are applied, then they are multiplicative/commutative.

Proof:

Let point P is first scaled with scaling factor Sx1, Sx2 to Point P' and again let P' be scaled by scaling factor Sx2,Sy2 to Point P'', then the combined transformation can be calculated with the following composite matrix

$$T = S(S_{x2}, S_{y2})\, S(S_{x1}, S_{y1})$$

$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x1}s_{x2} & 0 & 0 \\ 0 & s_{y1}s_{x2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

i.e. $S(s_{x2,sy2})\, S(s_{x2,sy2}) = S(s_{x1.sx2},\ s_{y1.sy2})$, which demonstrates that two successive scaling are multiplicative.

Multiple Coordinate Systems in a Graphics Program

A coordinate system is a reference system used to represent the object along with its features within a common co-ordinate framework.

In a typical graphics program, we may need to deal with a number of different coordinate systems, and a good part of the work is the conversion of coordinates from one system to another. The list of some of the coordinate systems are:

a) **Modeling co-ordinate system**

The Model Coordinate System is simply the coordinate system where the model was created. It is used to define coordinates that are used to construct the shape of individual parts (objects) of a 2D scene

b) **World co-ordinate system**

A Model Coordinate System is the unique coordinate space of the model. Two distinct models, each with their own coordinate systems can't interact with each other. There needs to be a universal coordinate system that allows each model to interact with each other. This universal system is called *World Coordinate System*. For interaction to occur, the coordinate system of each model is transformed into the World Coordinate System

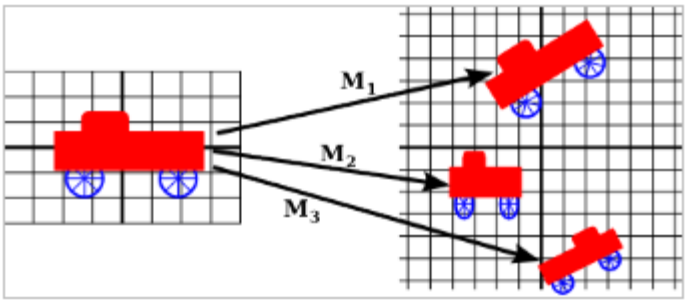


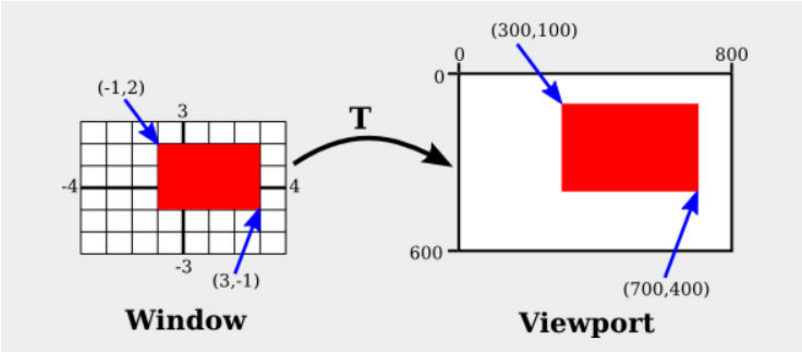
Fig: Modeling Coordinate Fig: World Coordinates

c) **Viewing co-ordinate system**

A world coordinate area selected for display is called **window**. Window is an area of picture that is selected for viewing. An area on display device to which a window is mapped is called **view port**. View port is the part of computer screen.

Viewing co-ordinate system are used to define particular view of a 2D scene. Translation, scaling, and rotation of the window will generate a different view of the scene.

For a 2-D picture, a view is selected by specifying a sub area (window) of the total picture area.



d) **Normalized Viewing Co-ordinates**

NVC's are used to make the viewing process independent of the output device (monitor, mobile, hard copy devices). Normally, the value of NVC is 0 to 1.

e) **Device Co-ordinates**

Device co-ordinate are used to define coordinates in an output device. They are integers within the range (0, 0) to (x_{max}, y_{max}) for a particular output device.

Two Dimensional Viewing

The process of mapping the world co-ordinate scent to device co-ordinate is called viewing transformation or window to view port transformation or windowing transformation.

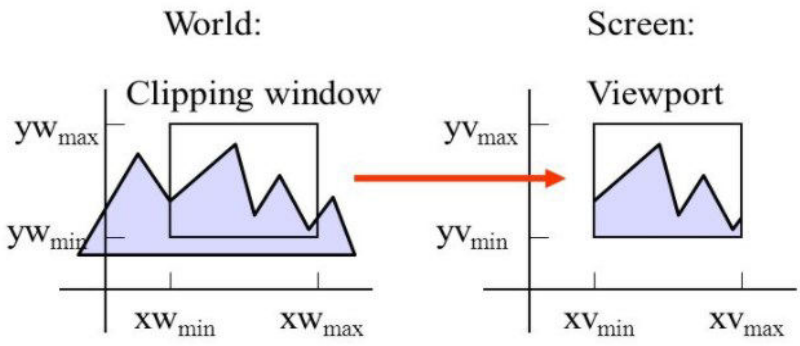


Fig: Mapping of picture section falling under rectangular Window onto a designated rectangular view port

The window defines what is to be viewed whereas the view port defines where it is to be displayed. Window can have any shape (circle, polygon) however some graphics package provide window and view port operations on standard rectangle only. Window deals with object space whereas view port deals with image space.

Transformations from world to device coordinates involve translation, rotation and scaling operations, as well as procedures for deleting those parts of the picture that are outside the limits of a selected display area i.e. clipping.

To make the viewing process independent of the requirements of any output device, graphics systems convert object descriptions to normalized coordinates.

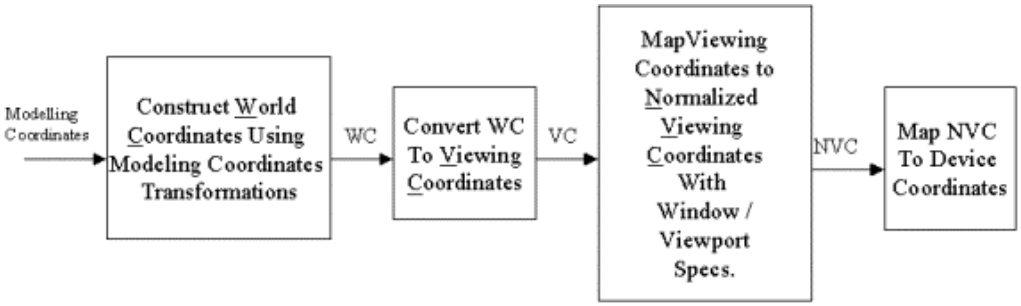


Fig: The 2D Viewing Transformation Pipeline

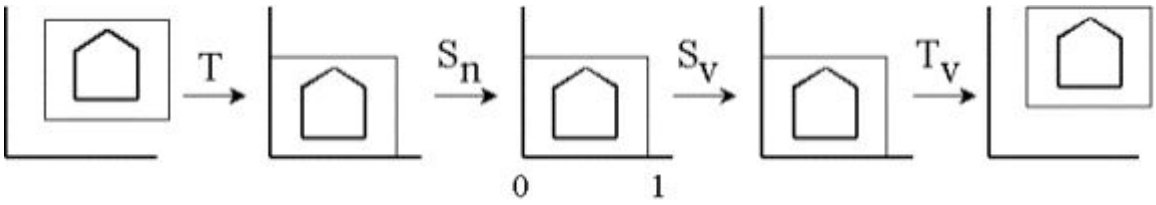
Procedure for to transform a window to the view port we have to perform the following steps:

Step1: The object together with its window is translated until the lower left corner of the window is at the origin

Step2: The object and window are scaled until the window has the dimensions of the view port

Step3: Again translate to move the view port to its correct position on the screen

(Setup Window,Translate window, Scale to normalize, Scale to view port, Translate to View port)

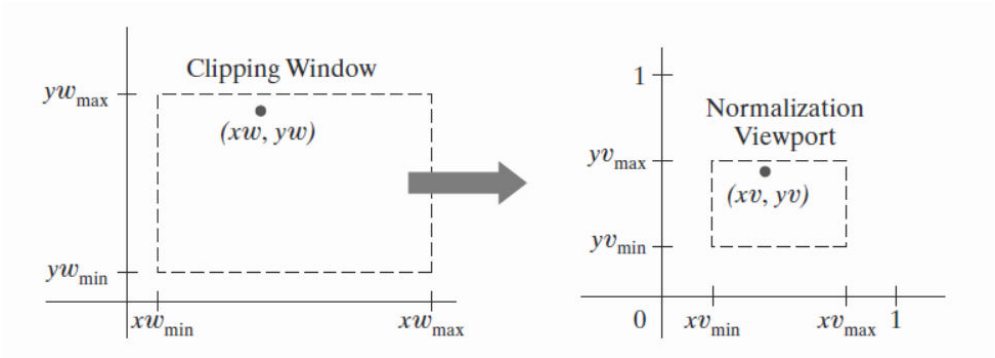


Application

i. By changing the position of the view port, we can view objects at different positions on the display area of an output device.
ii. By varying the size of view ports, we can change size of displayed objects.
iii. Zooming effects can be obtained by successively mapping different-sized windows on a fixed-sized view port
iv. Panning effects (Horizontal Scrolling) are produced by moving a fixed-size window across the various objects in a scene.

Window to View port Coordinate Transformation

A window can be specified by four world coordinates: xw_{min}, xw_{max}, yw_{min} and yw_{max}. Similarly, a view port can be described by four device coordinates: xv_{min}, xv_{max}, yv_{min} and yv_{max}



The following proportional ratios must be equal.

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

Solving these expressions, we get

$xv = xv_{min} + (xw - xw_{min})S_x$

$yv = yv_{min} + (yw - yw_{min})S_y$

where,

$$s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$

$$s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

Q. Window port is given by (100,100,300,300) and view port is given by (50,50,150,150). Convert the window port coordinate (200,200) to the view port coordinate.

Solution:

$(xw_{min}, yw_{min}) = (100, 100)$

$(xw_{max}, xw_{max}) = (300, 300)$

$(xv_{min}, yv_{min}) = (50, 50)$

$(xv_{max}, xv_{max}) = (150, 150)$

(xw,yw)=(200,200)

Then, we have

$$s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} = (150 - 50)/(300 - 100) = 0.5$$

$$s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} = (150 - 50)/(300 - 100) = 0.5$$

The equations for mapping window co-ordinate to view port coordinate is given by

$xv = xv_{min} + (xw - xw_{min})S_x$

$yv = yv_{min} + (yw - yw_{min})S_y$

Hence,

$xv = 50 + (200 - 100) * 0.5 = 100$

$yv = 50 + (200 - 100) * 0.5 = 100$

The transformed view port coordinate is (100,100).

Q. Find the normalization transformation matrix for window to view port which uses the rectangle whose lower left corner is at (2,2) and upper right corner is at (6,10) as a window and the view port that has lower left corner at (0,0) and upper right corner at (1,1)

The composite transformation matrix for transforming the window co-ordinate to viewport coordinate is given as

$$T = S(s_x, s_y)T(-2, -2)$$

Now we know,

$$s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$

$$s_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

$S_x = (1\text{-}0)/(6 - 2) = 0.25$
$S_y = (1\text{-}0)/(10 - 2) = 0.125$

Then, transformation matrix,

$$T = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

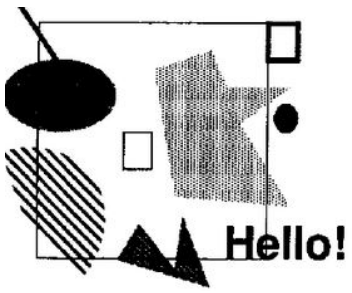
$$= \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

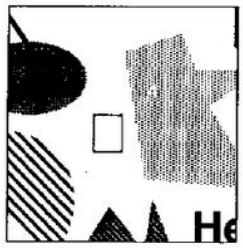
$$= \begin{bmatrix} 0.25 & 0 & -0.5 \\ 0 & 0.125 & -0.25 \\ 0 & 0 & 1 \end{bmatrix}$$

Clipping

The process of discarding those parts of a picture which are outside of a specified region or window is called **clipping**. The procedure using which we can identify whether the portions of the graphics object is within or outside a specified region or space is called **clipping algorithm**.

The region or space which is used to see the object is called window and the region on which the object is shown is called **view port**.





Clipping is necessary to remove those portions of the objects which are not necessary for further operation's. excludes unwanted graphics from the screen. So there are three cases

i. The object may be completely outside the viewing area defined by the window port.
ii. The object may be seen partially in the window port.
iii. The object may be seen completely in the window port

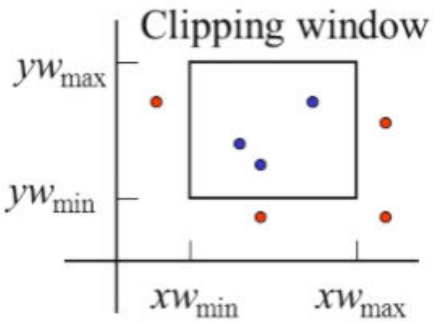
For case i and ii, clipping operation is necessary but not for case iii.

Applications of Clipping

i. Extracting part of a defined scene for viewing.
ii. Identifying visible surfaces in three-dimensional views.
iii. Anti aliasing line segments or object boundaries.
iv. Drawing and painting operations that allow parts of a picture to be selected for copying, moving erasing, or duplicating.

Types **of Clipping**

i. Point Clipping
ii. Line Clipping (straight-line segments)
iii. Area Clipping (polygons)
iv. Curve Clipping
v. Text Clipping



Point Clipping

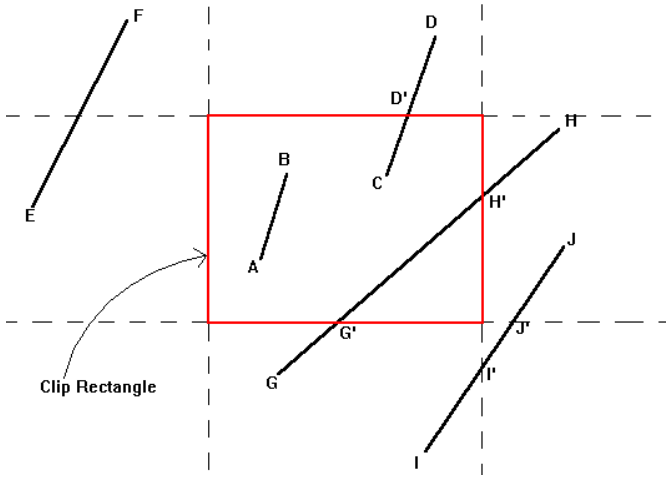
Point clipping is the process of removing all those points that lies outside a given region or window. Let xw_{min} and xw_{max} be the edge of the boundaries of the clipping window parallel to Y axis and yw_{min} and yw_{max} be the edge of the boundaries of the clipping window parallel to X axis as shown in figure below. So any point P(x,y) of world coordinate can be saved for display if the following conditions are satisfied

$$xw_{min} \leq x \leq xw_{max}$$
$$yw_{min} \leq y \leq yw_{max}$$

Line Clipping

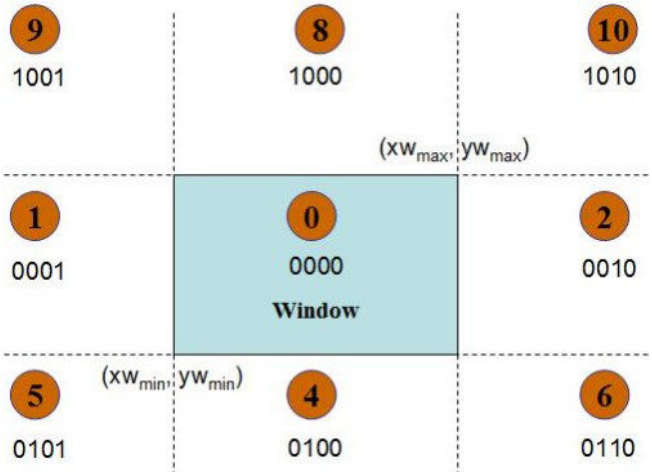
In line clipping, a line or part of line is clipped if it is outside the window port. All the line segment falls into one of the following clipping cases.

a. The line is totally outside the window port so is directly clipped.
b. The line is partially clipped if a part of the line lies outside the window port.
c. The line is not clipped if all whole line lied within the window port.

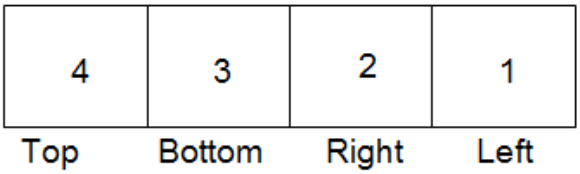


So in the above figure, line FE require total clipping, CD, GH, IJ require partial clipping and AB requires no clipping.

Cohen-Sutherland Line Clipping Algorithm

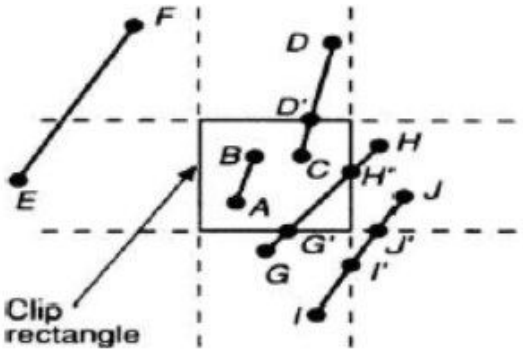


- ➢ Divide 2D space into 3x3 = 9-regions.
- ➢ Middle region is the clipping window.
- ➢ Each region is assigned a 4-bit code.
- ➢ Bit 1 is set to 1 if the region is to the left of the clipping window, otherwise.
  Similarly we deal for bits 2, 3 and 4.



- ➢ Any point inside the clipping window has a region code 0000.
- ➢ Any endpoint (x, y) of a line segment, the code can be determined as follows:
  - If $x < xw_{min}$, first bit is 1, (Point lies to left of window(Left)) (0th bit) Otherwise 0.
  - If $x > xw_{max}$, second bit is 1, (Point lies to right of window(Right)) (1st bit), otherwise 0.
  - If $y < yw_{min}$, third bit is 1, (Point lies to below window(Bottom)) (2nd bit), otherwise 0.
  - If $y > yw_{max}$, fourth bit is 1, (Point lies to above window(Top)) (3rd bit), otherwise 0.

Example:



Line	Code 1	Code 2
A⇒B	0000	0000
C⇒D	0000	1000
E⇒F	0001	1001
G⇒H	0100	0010
I⇒J	0100	0010

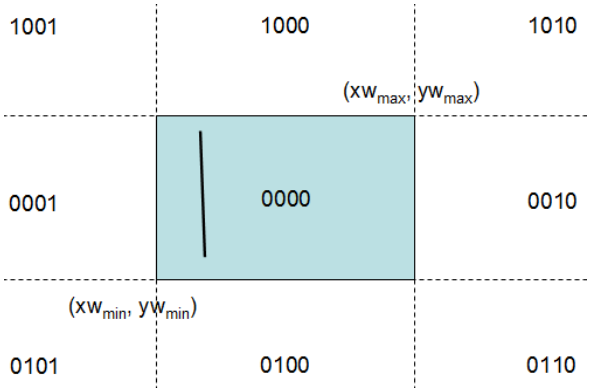
Algorithm:

a) Given a line segment with end points P1=(x1,y1) and P2(x2,y2), compute 4 bit region code for each end point.

b) To clip a line, first we have to find out which regions its two endpoints lie in.

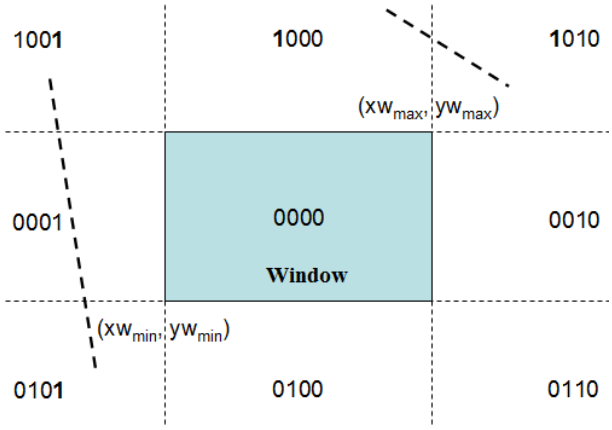
Case I:

If both end point code is 0000, then the line segment is completely inside the window. i.e. if bitwise OR of the codes yields 0000, then the line segment is accepted for display.



Case II:

If the two end point region code both have a 1 in the same bit position, then the line segment lies completely outside the window, so discarded. i.e. if bitwise AND of the codes not equal to 0000, then the line segment is rejected.



Case III:

If lines cannot be identified as completely inside or outside we have to do some more calculations.

Here we find the intersection points with a clipping boundary using the slope intercept form of the line equation

Here, to find the visible surface, the intersection points on the boundary of window can be determined as:

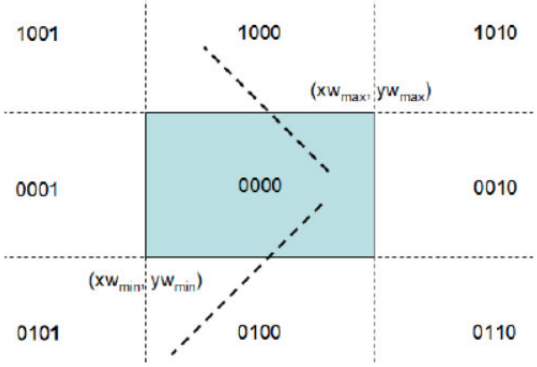
$$y - y_1 = m\,(x - x_1)$$

where $m = (y_2 - y_1)/(x_2 - x_1)$

i. If the intersection is with horizontal boundary

$x = x_1 + (y - y_1)/m$

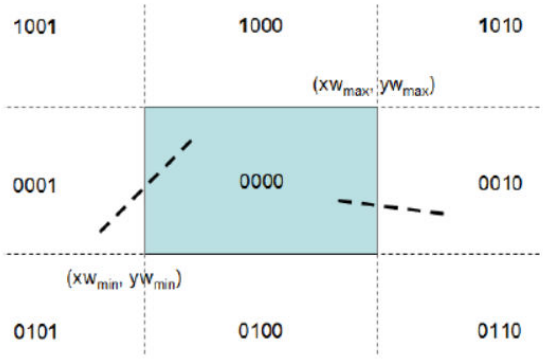
Where y is set to either yw_{min} or yw_{max}



ii. If the intersection is with vertical boundary

$y = y_1 + m\,(x - x_1)$

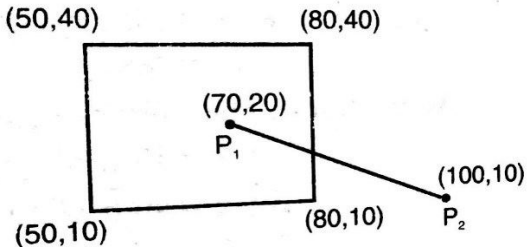
Where x is set to either Xw_{min} or Xw_{max}



c) Assign a new four-bit code to the intersection point and repeat until either cas1 or case2 are satisfied.

Q. Use the Cohen –Sutherland algorithm to clip the line P1(70,20) and P2(100,10) against a window lower left hand corner (50,10) and upper right hand corner (80,40)

Solution:



Assigning 4 bit binary code to the two end point

P1 = 0000

P2 = 0010

Finding bitwise OR:

P1 | P2 = 0000 | 0010 = 0010

Since P1 | P2 ! = 0000,hence the two point doesn't lie completely inside the window.

Finding bitwise AND:

P1 & P2 = 0000 & 0010 = 0000

since P1 & P2 = 0000, hence line is partially visible.

Now, For finding the intersection of P1 and P2 with the boundary of Window,
We have,
P1(x1,y1) = (70,20)
P2(x2,y2) = (100,10)
Slope m = (10-20)/(100-70) = -1/3
We have to find the intersection with right edge of window, here x=80, y=?
We have

y = y2 + m(x-x2)
= 10 +(-1/3)(80-100)
= 10 + 6.67
= 16.67

Thus the intersection point P3 = (80,16.66), So discarding the line segment that lie outside the boundary i.e P3P2, we get new line P1P3 with co-ordinate P1(70, 20) and P3(80,16.67)

Polygon Clipping: Sutherland – Hodgeman

The Sutherland–Hodgeman algorithm is used for clipping polygons. A single polygon can actually be split into multiple polygons. The algorithm clips a polygon against all edges of the clipping region in turn.

This algorithm is actually quite general — the clip region can be any convex polygon in 2D, or any convex polyhedron in 3D.

There are four possible cases for any given edge of given polygon against clipping edge.

1. **Both vertices are inside** :

Only the second vertex is added to the output list

2. **First vertex is outside while second one is inside** :

Both the point of intersection of the edge with the clip boundary and the second vertex are added to the output list

3. **First vertex is inside while second one is outside** :

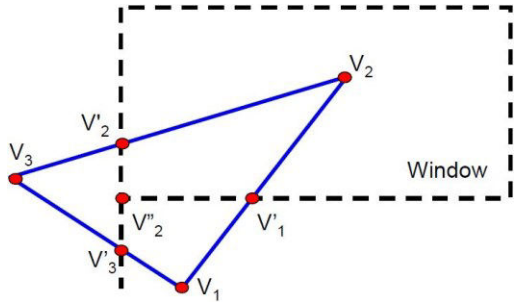
Only the point of intersection of the edge with the clip boundary is added to the output list

4. **Both vertices are outside** :

No vertices are added to the output list

Case	1st vertex	2nd vertex	output
1	inside	inside	2nd vertex
2	inside	outside	intersection
3	outside	outside	none
4	outside	inside	2nd and intersection

Example:



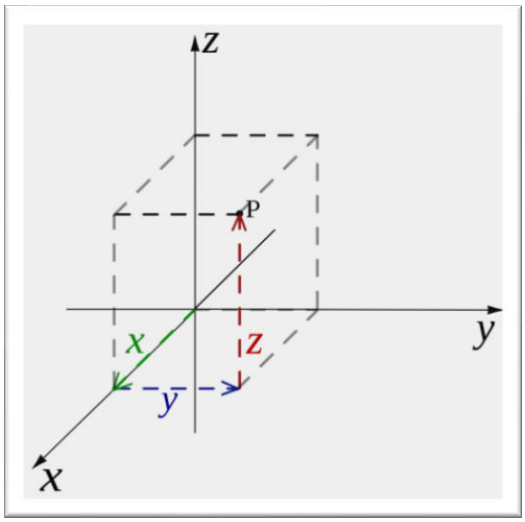
From	To	1st point	2nd point	Case	Output list
V_1	V_2	Outside	Inside	4	V'_1 and V_2
V_2	V_3	Inside	Outside	2	V'_2
V_3	V_1	Outside	Outside	3	

Chapter 5

3D Graphics System

Three-dimensional Space

Three-dimensional space is a geometric 3-parameters model of the physical universe in which all known matter exists. These three dimensions can be labeled by a combination of length, breadth, and depth. Any three directions can be chosen, provided that they do not all lie in the same plane.



3-Dimensional Object

An object that has height, width and depth, like any object in the real world is a 3-dimensional object.

Types of objects: Trees, terrains, clouds, rocks, glass, hair, furniture, human body, flowers, rubber etc

3D Graphics

3D computer graphics or three-dimensional computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images.

2D is "flat", using the horizontal and vertical (X and Y) dimensions, the image has only two dimensions. 3D adds the depth (Z) dimension. This third dimension allows for rotation and visualization from multiple perspectives. It is essentially the difference between a photo and a sculpture.

We can perform different transformation by specifying the three-dimensional transformation vector, however the 3d Transformation is more complex than 2D transformation.

Q. What are the issue in 3D that makes it more complex than 2D?

When we model and display a three-dimensional scene, there are many more considerations we must take into account besides just including coordinate values as 2D, some of them are:

- Relatively more co-ordinate points are necessary.
- Object boundaries can be constructed with various combinations of plane and curved surfaces.
- Consideration of projection (dimension change with distance) and transparency.
- Many considerations on visible surface detection and remove the hidden surfaces.

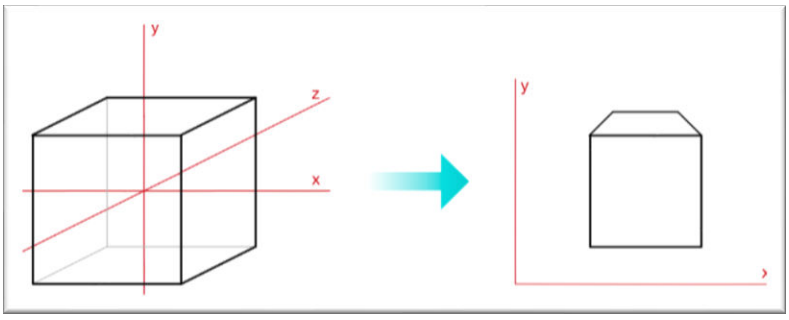
Pseudo-3D and true 3D

Pseudo-3D is a term used to describe either 2D graphical projections and similar techniques used to cause images or scenes to simulate the appearance of being three-dimensional (3D) when in fact they are not.

By contrast, games using 3D computer graphics without such restrictions are said to use true 3D.

3D display method

Three-dimensional viewing coordinates must be transformed onto two-dimensional device coordinates to view the scene.



To obtain a display of a three-dimensional scene that has been modeled in world coordinates, we must first set up a coordinate reference for the "camera". This coordinate reference defines the position and orientation for the plane of the camera film, which is the plane we want to use to display a view of the objects in the scene. Object descriptions are then transferred to the camera reference coordinates and projected onto

the selected display plane.

We can then apply lighting and surface-rendering techniques to shade the visible surfaces.

3D Geometric Transformation

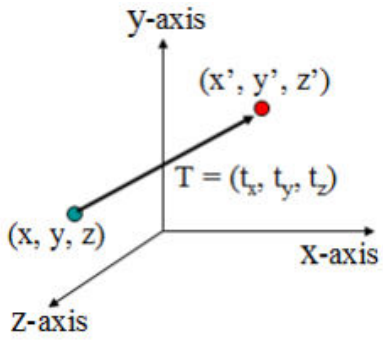
Methods for geometric transformations and object modeling in three dimensions are extended from two-dimensional methods by including considerations for the z coordinate. We now translate an object by specifying a three-dimensional translation vector, which determines how much the object is to be moved in each of the three coordinate directions.

2D transformations can be represented by 3 x 3 matrices using homogeneous coordinates, so 3D transformations can be represented by 4 x 4 matrices, providing we use homogeneous coordinate representations of points in 2 spaces as well. Thus instead of representing a point as (x, y, z), we represent it as (x, y, z, H), where two these quadruples represent the same point it one is a non-zero multiple of the other the quadruple (0, 0, 0, 0) is not allowed. A standard representation of a point (x,y, z, H) with H not zero is given by (x/H, y/H, z/H, 1). Transforming the point to this form is called homogenizing.

1. **Translation**

A point is translated from position P=(x, y, z) to position P'= (x', y', z') with the matrix operation as:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



Parameters t_x, t_y, t_z specify translation distances for the coordinate directions x, y and z. This

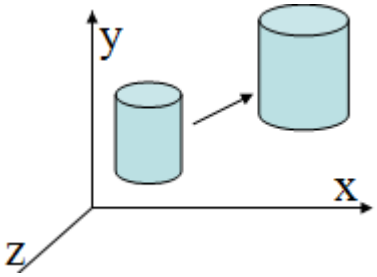
matrix representation is equivalent to three equations:

$$x' = x + t_x$$
$$y' = y + t_y$$
$$z' = z + t_z$$

2. **Scaling**

Matrix expression for scaling transformation of a position P = (x, y, z) relative to the coordinate origin can be written as :

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



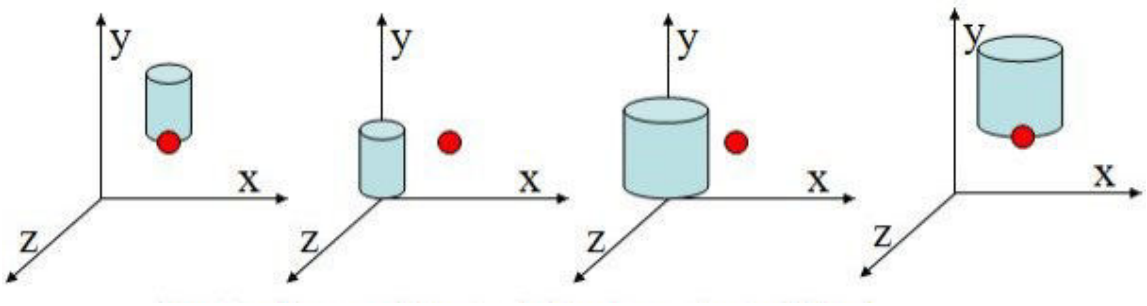
Scaling with respect to a selected fixed point (x_f, y_f, z_f)

- ➢ Translate fixed point to the origin
- ➢ Scale object relative to the coordinate origin
- ➢ Translate fixed point back to its original position

i.e $\mathbf{T}_{(x, y, z)} \cdot \mathbf{S}_{(x, y, z)} \cdot \mathbf{T}_{(-x, -y, z)}$

$$= \begin{bmatrix} 1 & 0 & 0 & x_f \\ 0 & 1 & 0 & y_f \\ 0 & 0 & 1 & z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_f \\ 0 & 1 & 0 & -y_f \\ 0 & 0 & 1 & -z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

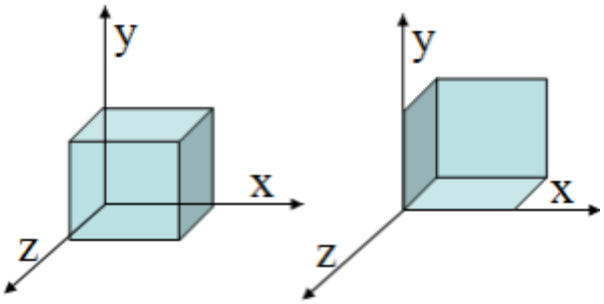


3. **Shearing**

Shearing transformations are used to modify object shapes.

a) **Z axis Shearing**

This transformation alters x and y co-ordinate values by amount that is proportional to the z value while leaving z co-ordinate unchanged.



$x' = x + S_{hx}.Z$

$z' = y + S_{hy}.Z$

$z' = z$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & S_{hx} & 0 \\ 0 & 1 & S_{hy} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Parameters S_{hx} and S_{hy} can be assigned any real values

b) **X axis Shearing**

This transformation alters Y and Z coordinate values by an amount that is proportional to the X value while leaving the X value unchanged i.e.

$x' = x$

$y' = y + S_{hy}x$

$z' = z + s_{hz}X$

c) **Y axis shearing**

This transformation alters Y and Z coordinate values by an amount that is proportional to the X value while leaving the X value unchanged i.e.

$x' = x + S_{hx}.y$

$y' = y$

$z' = z + S_{hy}X$

4. **Reflection**

In 3D-reflection the reflection takes place about a plane. The matrix in case of pure reflections, along basic planes, viz. *X-Y plane, Y-Z plane and Z-X plane* are given below:

a) **X-Y plane**

This transformation changes the sign of the z coordinates, leaving the x and y coordinate values unchanged.

$$RFz = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

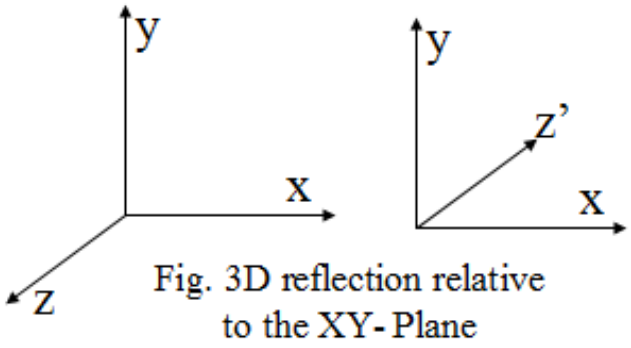


Fig. 3D reflection relative to the XY- Plane

b) **Y-Z plane**

This transformation changes the sign of the x coordinates, leaving the y and z coordinate values unchanged.

$$RFx = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

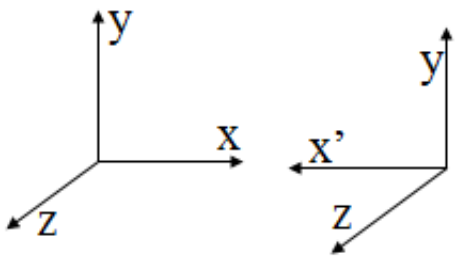


Fig. 3D reflection relative to the YZ- Plane

c) **Z-X plane**

This transformation changes the sign of the y coordinates, leaving the x and z coordinate values unchanged.

$$RFy = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

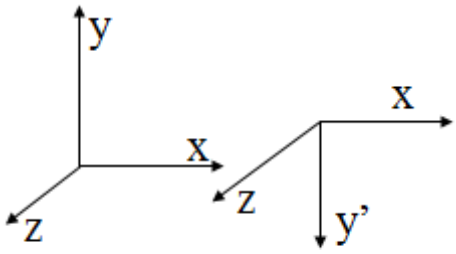


Fig. 3D reflection relative to the ZX- Plane

d) **Reflection about any axis parallel to one of the coordinate axes**

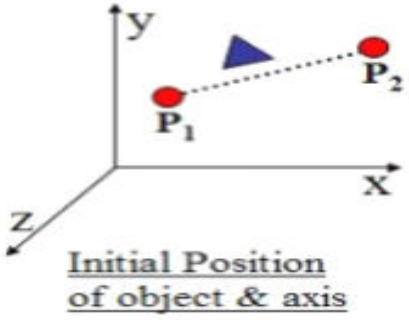
Steps:

- Translate object so that reflection axis coincides with the parallel coordinate axis.
- Perform specified reflection about that axis
- Translate object back to its original Position

$$P' = [\, T^{-1} \,.\, R \,.\, T \,] \,.\, P$$

e) **Reflection about an arbitrary axis**

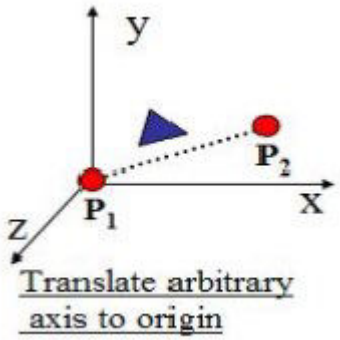
A reflection matrix for any axis that does not coincide with a coordinate axis can be set up as a composite transformation involving combinations of translation, the coordinate-axes rotation and reflection.



Initial Position of object & axis

Step-1: Translate the arbitrary axis so that it passes through origin.

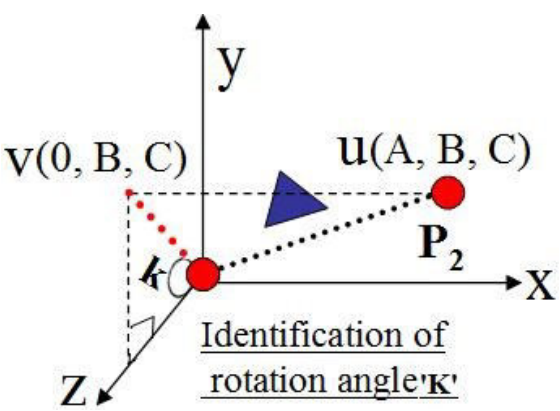
$$T = \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

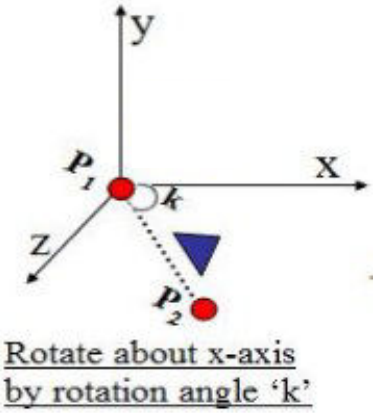


Step-2: Rotate the object so that the arbitrary axis coincides with one of the coordinate axes. Usually the z axis is preferred

To coincide the axis of rotation to z axis we have to first perform rotation about x axis to bring it into X-Z plane and then perform rotation about y axis to coincide it with z axis.

a) **Rotation about x-axis by angle 'k' in clockwise direction so that the arbitrary axis lies on X-Z-plane**





Here,

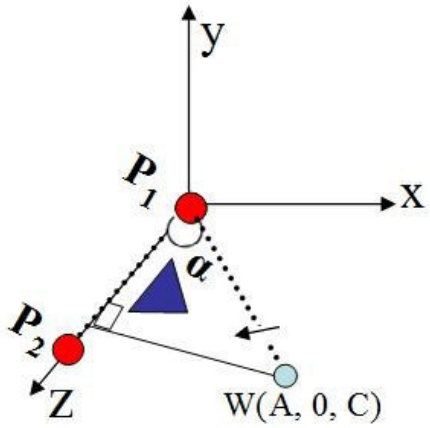
Sin k = B / V Where, $V=\sqrt{(B^2 + C^2)}$

Cos k = C / V

Now, the translation matrix is given by:

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \text{Cos k} & \text{-Sin k} & 0 \\ 0 & \text{Sin k} & \text{Cos k} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & C/V & -B/V & 0 \\ 0 & B/V & C/V & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

b) **Rotation about y-axis by angle 'α' in clockwise direction so that the arbitrary axis align with z-axis**



Here,

Sin α= A / W &

Cos α= C / W

Now, the translation matrix is given by:

$$R_y = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} C/W & 0 & -A/W & 0 \\ 0 & 1 & 0 & 0 \\ A/W & 0 & C/W & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Step-3: Reflect the object about xy-plane or z-axis

$$RF_Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Step-4: Perform inverse rotation about y-axis & then x-axis

Step-5: Perform inverse translation

Hence,

Composite matrix = $\mathbf{T' R_x 'R_y ' RF_Z R_y R_x T}$

f) Reflection about any arbitrary plane in 3D Space

The reflection about any arbitrary plane perform same operation as the reflection about any arbitrary line, the only difference is that we have to characterize the rotation by any normal vector 'N' in that plane.

Step 1: Translate the reflection plane to the origin of the coordinate system

Step 2: Perform appropriate rotations to make the normal vector of the reflection plane at the origin until it coincides with the z-axis.

Step 3: After that reflect the object through the z= 0 coordinate plane.

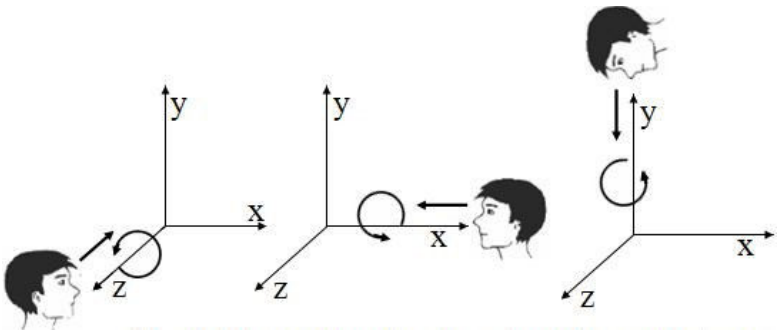
Step 4: Perform the inverse of the rotation transformation

Step 5: Perform the inverse of the translation

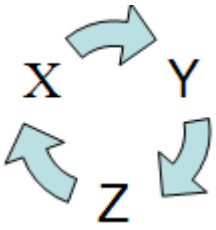
5. **Rotation**

To generate a rotation transformation for an object in 3D space, we require the following:

- ➢ Angle of rotation.
- ➢ Pivot point
- ➢ Direction of rotation
- ➢ Axis of rotation



Cyclic permutation of the coordinate parameters x, y and z are used to get transformation equations for rotations about the coordinates:



Taking origin as the center of rotation, when a point P(x, y, z) is rotated through an angle about any one of the axes to get the transformed point P'(x', y', z'), we have the following equation for each.

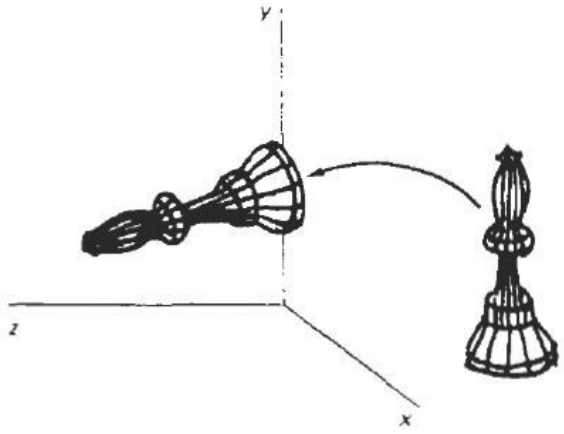
a) **3D x-axis rotation ($P' = R_{x(\theta)} . P$)**

$x' = x$

$y' = y\cos\theta - z\sin\theta$

$z' = y\sin\theta + z\cos\theta$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



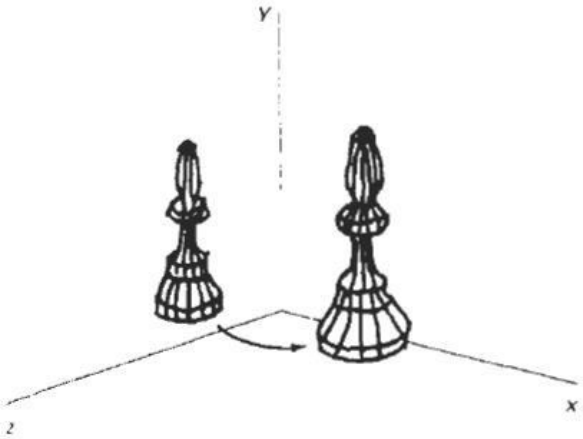
b) **3D y-axis rotation ($P' = R_{y(\theta)} . P$)**

$x' = z\sin\theta + x\cos\theta$

$y' = y$

$z' = z\cos\theta - x\sin\theta$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



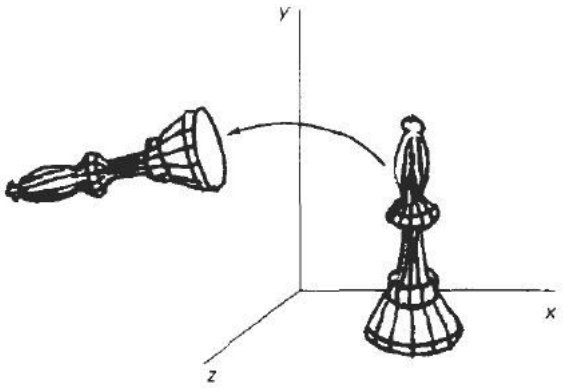
c) **3D z-axis rotation ($P' = R_{z(\theta)} . P$)**

$x' = x\cos\theta - y\sin\theta$

$y' = x\sin\theta + y\cos\theta$

$z' = z$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

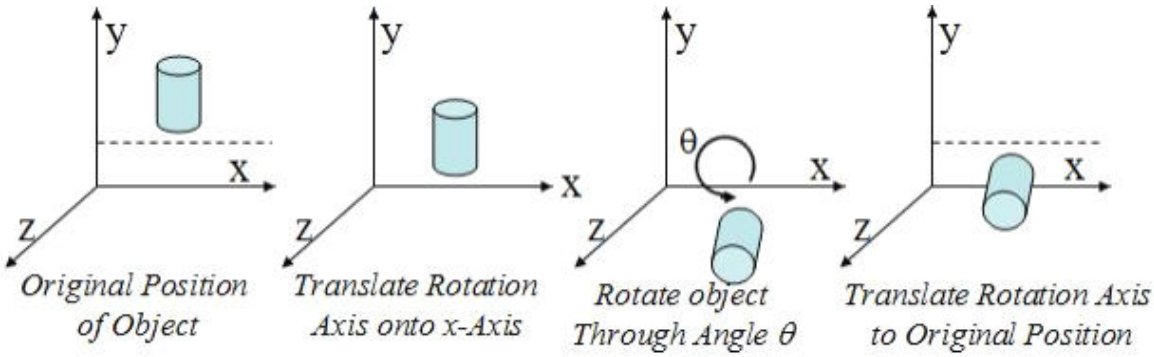


d) **Rotation about an axis parallel to one of the coordinate axes**

Steps:

- Translate object so that rotation axis coincides with the parallel coordinate axis.
- Perform specified rotation about that axis
- Translate object back to its original Position

$$P' = [\, T^{-1} . R_{(\theta)} . T \,] . P$$



e) **Rotation about an arbitrary axis**

A rotation matrix for any axis that does not coincide with a coordinate axis can be set up as a composite transformation involving combinations of translation and the coordinate-axes rotations.

Step-1: Translate the arbitrary axis so that it passes through origin.

Step-2: Align the arbitrary axis on any major co-ordinate axis (z-axis)

Step-3: Rotate the object about yz-plane or z-axis

Step-4: Perform inverse rotation about y-axis & then x-axis

Step-5: Perform inverse translation

Hence,

Composite matrix = $\mathbf{T' R_x 'R_y ' R_Z R_y R_x T}$

f) **Rotation about any arbitrary plane in 3D Space**

The rotation about any arbitrary plane perform same operation as the rotation about any arbitrary line, the only difference is that we have to characterize the rotation by any normal vector 'N' in that plane.

Step 1: Translate the rotation plane to the origin of the coordinate system

Step 2: Perform appropriate rotations to make the normal vector of the rotation plane at the origin

until it coincides with the z-axis.

Step 3: After that rotate the object through the z= 0 coordinate plane.

Step 4: Perform the inverse of the rotation transformation

Step 5: Perform the inverse of the translation

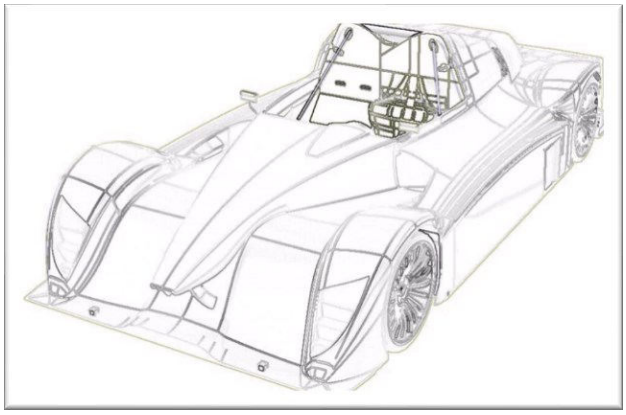
3D Representation

Graphics scenes can contain many different kinds of objects like trees, flowers, clouds, rocks, water etc. These cannot be describe with only one methods but obviously require large precisions such as polygon & quadratic surfaces, spline surfaces, procedural methods, volume rendering, visualization techniques etc.

Representation schemes for solid objects are often divided into two broad categories:

- **Boundary representations (B-reps)**: Often abbreviated as **B-rep** or **BREP,** boundary representation is a method for representing shapes (a set of surfaces that separate the object interior from the environment) using the limits.

A solid is represented as a collection of connected surface elements, the boundary between solid and non-solid.



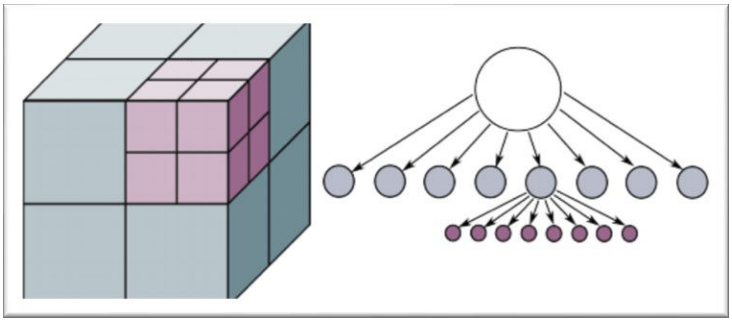
For examples: polygon surfaces and spline patches.

Boundary representation of models are composed of two parts: topology and geometry (surfaces, curves and points). The main topological items are: *faces*, *edges* and *vertices*. A face is a bounded portion of a surface; an edge is a bounded piece of a curve and a vertex lies at a point.

Other elements are the *shell* (a set of connected faces), the *loop* (a circuit of edges bounding a face) and *loop-edge links* (also known as *winged edge links* or *half-edges*) which are used to create the edge circuits. The edges are like the edges of a table, bounding a surface portion.

- **Space-partitioning representations**: are used to describe interior properties, by partitioning the spatial region containing an object into a set of small, non-overlapping, contiguous solids (usually cubes).

Space-partitioning systems are often hierarchical, meaning that a space (or a region of space) is divided into several regions, and then the same space-partitioning system is recursively applied to each of the regions thus created. The regions can be organized into a tree, called a space-partitioning tree.



For example: Octree representation.

The visual realism in 3D object can be maintained by maintaining the transparency, projection, lighting & shading effect on each portion. The representation of 3D object include following three stages:

1. Represent the objects with multiple polygons i.e. wired-frame representation.
2. Fill all the polygons either with flat filling or smooth filling.
3. Give the lightning or shading effect and the coloring effect.

Polygon Surfaces

A set of polygon surfaces are used to represent object boundary that enclose the object interior in 3D graphics. This simplifies and speeds up the surface rendering and display of objects, since all surfaces are described with linear equations of plane: Ax + By + Cz + D = 0. For this reason, polygon descriptions are often referred to as "standard graphics objects."

The wire frame representations are common in design & solid modeling application because they can be displayed quickly to of a give a general indication of the surface structure.

A. Polygon Tables

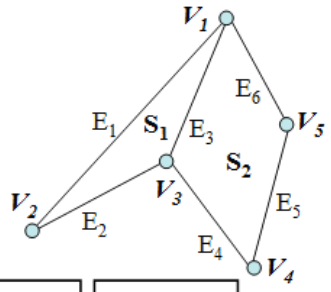
To specify a polygon surface, a set of vertex coordinates and associated attribute parameters are placed into tables & that are used in the subsequent processing, display, error checking and manipulation of the objects in a scene.

Polygon data tables can be organized into two groups:

a) **Geometric tables**

Geometric data tables contain vertex coordinates and parameters to identify the spatial orientation of the polygon surfaces. A convenient organization for storing geometric data is to create three lists: a **vertex table, an edge table, and a polygon table.**

- ➢ Coordinate values for each vertex in the object are stored in the vertex table.
- ➢ The edge table contains pointers back into the vertex table to identify the vertices for each polygon edge.
- ➢ The polygon table contains pointers back into the edge table to identify the edges for each polygon.



Vertex Table
V_1: x_1, y_1, z_1
V_2: x_2, y_2, z_2
V_3: x_3, y_3, z_3
V_4: x_4, y_4, z_4
V_5: x_5, y_5, z_5

Edge Table
E_1: V_1, V_2
E_2: V_2, V_3
E_3: V_3, V_1
E_4: V_3, V_4
E_5: V_4, V_5
E_6: V_5, V_1

Polygon Table
S_1: E_1, E_2, E_2
S_2: E_3, E_4, E_5, E_6

b) **Attribute tables**

Attribute information for an object includes parameters specifying the degree of transparency of the object and its surface reflectivity and texture characteristics. The above three table also include the polygon attribute according to their pointer information.

B. Plane Equations

Plane equation method is the method for representing polygon surface for 3D object. The information about spatial orientation of object is described by its individual surface which is obtained by vertex coordinate values & equation of each plane gives a description of each surface. The equation for plane surface can be expressed in the form of:

$$Ax + By + Cz + D = 0$$

Where (x,y,z) is any point on plane, & A, B, C, D are constants describing spatial properties of the plane. The values of A, B, C, D can be obtained by solving set of 3 plane equations using coordinate values of three non-collinear points on plane.

Let (x_1,y_1,z_1), (x_2,y_2,z_2) , (x_3,y_3,z_3) are three such points on points on the plane. Then

$$Ax_1 + By_1 + Cz_1 + D = 0 \ldots\ldots\ldots\ldots (1)$$

$$Ax_2 + By_2 + Cz_2 + D = 0 \ldots\ldots\ldots\ldots (2)$$

$$Ax_3 + By_3 + Cz_3 + D = 0 \ldots\ldots\ldots\ldots (3)$$

The solution of these equations can be obtained in the determinant form using Cramer's rule as:

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \qquad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$$

$$C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad D = -\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

For any point (x, y, z)

if $Ax + By + Cz + D \neq 0$ then (x, y, z) is not on the plane

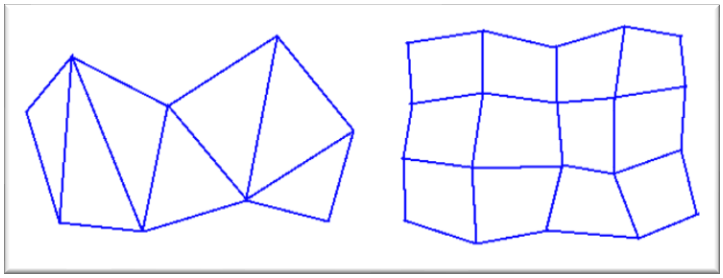
if $Ax + By + Cz + D < 0$, the point (x, y, z) is inside plane

i.e. invisible side

if $Ax + By + Cz + D > 0$ the point lies outside the surface.

C. Polygon Meshes

A polygon mesh is a collection of edges, vertices and polygons connected such that each edge is shared by at most two polygons & hence bounding the planer surface.



One type of polygon mesh is the triangle strip that produces n–2 connected triangles, given the coordinates for n vertices.

Another similar functions the quadrilateral mesh that generates a mesh of (n-1).(m-1) quadrilaterals, given the coordinates for an n by m array of vertices.

Cubic Spline and Bezier Curve

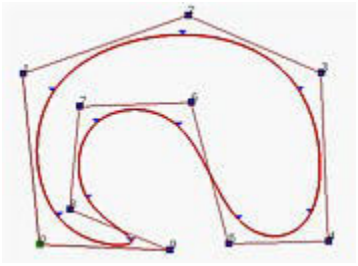
Splines are the smooth curves passing through a set of given points. By varying the number and position of the lead weights, the shape of the spline is changed so that it passes through some designated set of points.

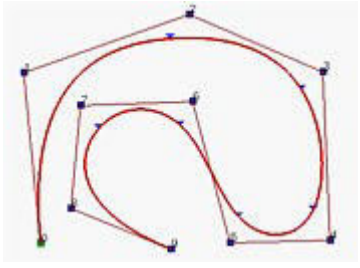
In computer graphics, continuous curve that are formed with polynomial section with certain boundary conditions are called spline curve.

Splines are of two types

a. Closed Splines: the generated curve will touch the first and last legs of the control

b. Open Splines : the generated curve will not touch the first and last legs of the control





A cubic spline is a spline constructed of piecewise third-order polynomials which pass through a set *m* of control points.

Splines are used:

- ➢ To design curve and surface shapes in graphics applications,
- ➢ To digitize drawings for computer storage
- ➢ To specify animation paths for the objects or image.
- ➢ Typical CAD applications for splines include the design of automobile bodies, aircraft and spacecraft surfaces, and ship hulls.

A. Control points

We specify a spline curve by giving a set of coordinate positions, called control points, which indicates the general shape of the curve These, control points are then fitted with piecewise continuous parametric polynomial functions in one of two ways.

- **Interpolation curve**: The polynomial sections are fitted by passing the curve through each control points. Interpolation curves are commonly used to digitize drawings or to specify animation paths.
- **Approximation curve**: The polynomials are fitted to the general control-point path without necessarily passing through any control point. Approximation curves are primarily used as design tools to structure object surfaces.

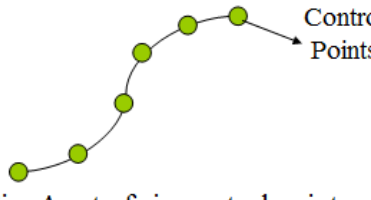


Fig. A set of six control points ***interpolated*** with piecewise continuous p

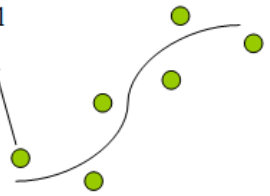


Fig. A set of six control points ***approximated*** with piecewise

ions

B. Convex hulls

The convex polygon boundary that encloses the set of control points is called convex hull. Convex hulls provide the measure for deviation of a curve or surface from a region bounding the control points.

Some splines are bounded by convex hull, thus ensuring that polynomials smoothly follow control points

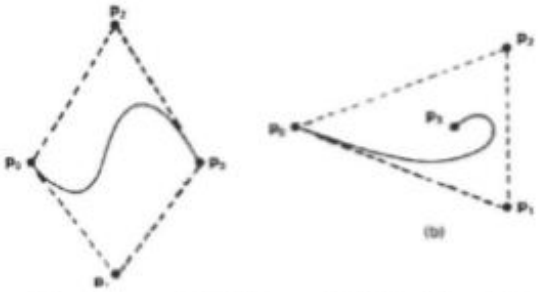
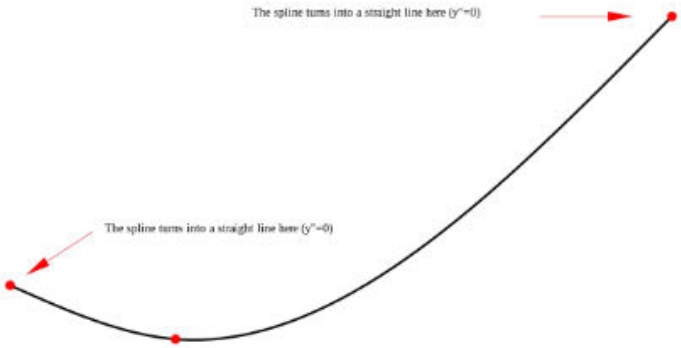


Fig. Convex hull shapes (dashed lines) for two sets of control points.

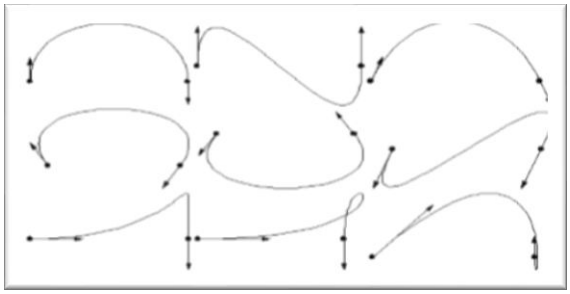
C. Natural Cubic Splines

One of first spline curves to be developed for graphics applications is a natural cubic spline. This interpolation curve is the mathematical representation of a original drafting spline. We formulate natural cubic spline by requiring that 2 adjacent curve sections have same first & 2nd parametric derivatives at their common boundary. Thus, natural cubic splines have C-2 continuity. Although natural cubic splines are the mathematical model for drafting spline, they have major disadvantage. If a position of any one control point is altered, then the entire curve is affected. Thus, natural cubic splines allow for no "local control", so that we cannot restructure part of curve without specifying an entirely new set of control points.



D. Hermite Curves

Hermite form is defined by 2 endpoints & 2 tangent vectors at these endpoints. Hermite spline is an interpolating piece-wise cubic polynomial with specified tangent at each control point. Unlike natural cubic splines, Hermite splines can be adjusted locally because each curve section is only dependent on endpoint constraints.

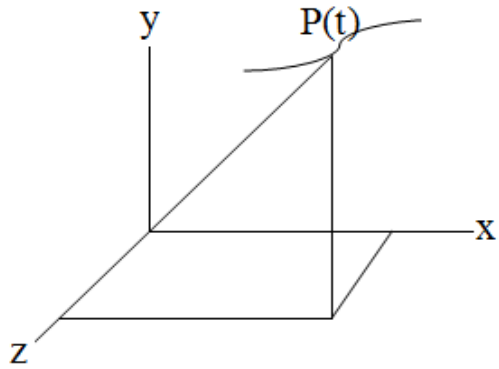


E. Parametric Cubic Curve

A parametric cubic curve is defined as

$P(t) = \sum_{i=0}^{3} a_i t^i \quad 0 <= t <= 1$ ------ (i)

Where, P (t) is a point on the curve



Expanding equation (i) yield

$P(t) = a_3t^3 + a_2t^2 + a_1t + a_0$ ----------(ii)

This equation is separated into three components of P (t)

$x(t) = a_{3x}t^3 + a_{2x}t^2 + a_{1x}t + a_{0x}$

$y(t) = a_{3y}t3 + a_{2y}t^2 + a_{1y}t + a_{0y}$

$z(t) = a_{3z}t3 + a_{2z}t^2 + a_{1z}t + a_{0z}$ ----------(iii)

F. Bezier Curve

A Bezier curve is a mathematically defined curve used in two-dimensional graphic applications. The curve is defined by four points: the initial position and the terminating position (which are called "anchors") and two separate middle points (which are called "handles"). The shape of a Bezier curve can be altered by moving the handles. The mathematical method for drawing curves was created by Pierre Bézier in the late 1960's for the manufacturing of automobiles**.**

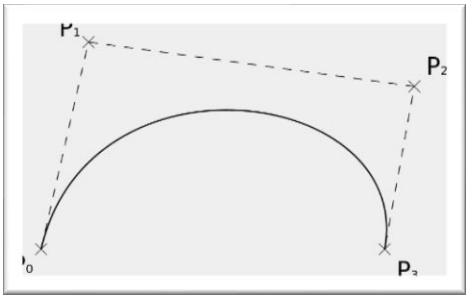


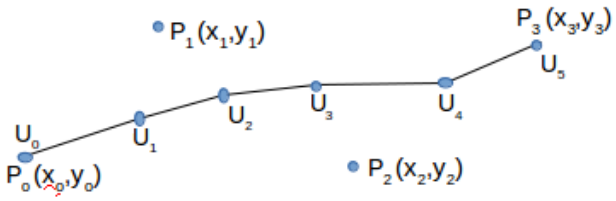
Fig: Bezier curve with four control points

In general, a Bezier curve can be fitted to any number of control points. The number of control points to be approximated and their relative position determine the degree of the Bezier polynomial. As with the interpolation splines, a Bezier curve can be specified with boundary conditions, with a characterizing matrix, or with blending functions.

The Bezier curve has two important properties:

a) It always passes through the first and last control points.

b) It lies within the convex hull (convex polynomial boundary) of the control points.



Let, P be the control points and N_{seg} is number of segments in a curve segment.

i.e $u = \frac{i}{N_{seg}}$; where i = 0 to N_{seg} i.e $0 \leq u \leq 1$

we have,

$x(u) = \sum_{j=0}^{n} x_j BeZ_{(j,n)}(u)$ where n= no. of control points.

$= x_0BeZ_{0,3}(u) + x_1BeZ1_{,3}(u) + x_2BeZ_{2,3}(u) + x_3BeZ_{3,3}(u)$

$y(u) = \sum_{j=0}^{n} y_j BeZ_{(j,n)}(u)$

$= y_0BeZ_{0,3}(u) + y_1BeZ1_{,3}(u) + y_2BeZ_{2,3}(u) + y_3BeZ_{3,3}(u)$

The blending function $BeZ_{j,n}(u)$ is defined as a Bernstein polynomial

$$BeZ_{j,n}(u) = \frac{n!}{j!(n-j)!} u^j (1-u)^{n-j}$$

Also we have

$$Q(u) = P_0\, BeZ_{0,3}(u) + P_1\, BeZ_{1,3}(u) + P_2 BeZ_{2,3}(u) + P_3\, BeZ_{3,3}(u)$$

Calculating

$$BeZ_{0,3}(u) = \frac{3!}{0!(3-0)!} u^0 (1-u)^{3-0} = (1-u)^3$$

$$BeZ_{1,3}(u) = \frac{3!}{1!(3-1)!} u^1 (1-u)^{3-1} = 3u(1-u)^2$$

$$BeZ_{2,3}(u) = \frac{3!}{2!(3-2)!} u^2 (1-u)^{3-2} = 3u^2(1-u)$$

$$BeZ_{3,3}(u) = \frac{3!}{3!(3-3)!} u^3 (1-u)^{3-3} = u^3$$

Substituting the values, we get

$$Q(u) = P_0 (1-u)^3 + P_1\, 3u(1-u)^2 + P_2 3u^2(1-u) + P_3\, u^3$$

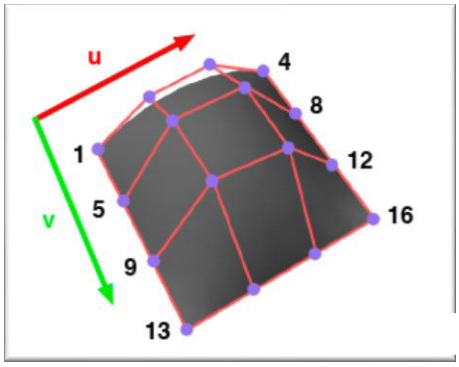
Q. Calculate the co-ordinates of Beizer curve described by 4 control points $P_0(0,0)$, $P_1(1,2)$, $P_2(3,3)$, $P_3(4,0)$ and approximate with 5 lines segments.

G. B-Spline

While drawing the bezier curve, each section of the curve is connected to the next section at a sample point but the slopes of these two sections need not match at this point. This means that there can be corners at the sample points and that we may not have a complete smooth curve. To solve this problems, we force the curve to pass through the sample point but rather pull it to the neighborhood of the sample point. A set of blending functions which take this approach are called **B-Splines**

H. Bezier Non-planar Surfaces

Like Bezier curve, Bézier surface is defined by a set of control points. Similar to interpolation in many respects, a key difference is that the surface does not pass through the central control points; rather, it is "stretched" toward them as though each were an attractive force.



A two-dimensional Bézier surface can be defined as a parametric surface where the position of a point p as a function of the parametric coordinates u, v is given by:

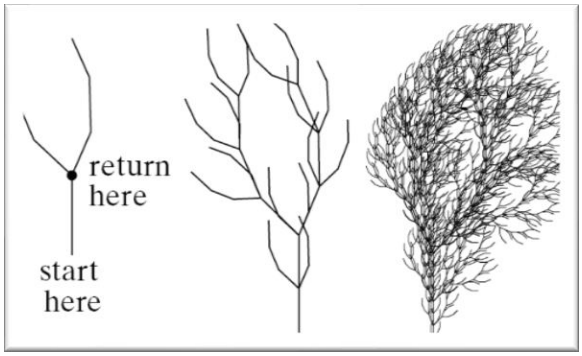
$$\mathbf{p}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(u)\, B_j^m(v)\, \mathbf{k}_{i,j}$$

evaluated over the unit square, where

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}$$

Fractal Geometry Method

A fractal is a mathematical set that typically displays self-similar patterns or recursive operations, which means it is "the same from near as from far".



Natural objects, such as mountains and clouds, have irregular or fragmented features, and these are described with fractal-geometry methods.

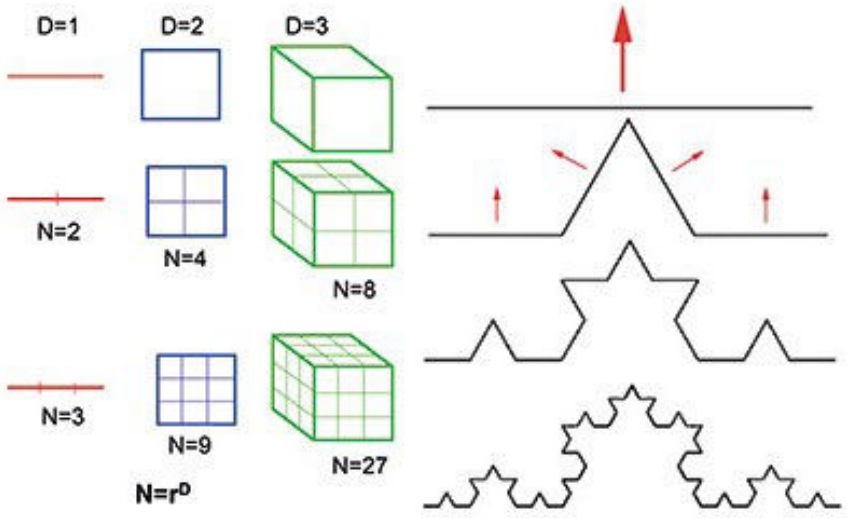
Fractional object describes and explains the features of natural phenomena with procedures or function in various dimensions that theoretically repeat an infinite number of times but finite number of steps for graphics display.

For fractional-Generation procedure, If $P_0 = (x_0 , y_0 , z_0)$ is a selected initial point, each iteration of a transformation function F generates successive levels of detail with the calculations are:

$P_1 = F(P_0)$,

$P_2 = F(P_2)$, …

We can describe the amount of variation in the object detail with a number called the fractal dimension.

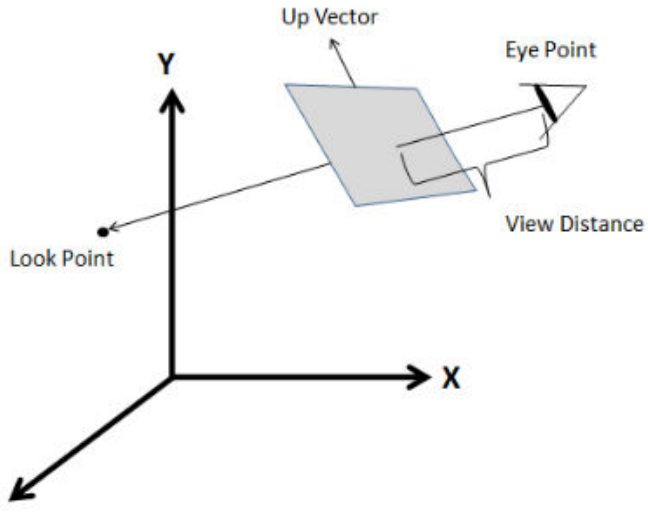


3D Viewing Transformation

The basic idea of the 3D viewing transformation is similar to the 2D viewing transformation. That is, a viewing window is defined in world space that specifies how the viewer is viewing the scene. A corresponding view port is defined in screen space, and a mapping is defined to transform points from world space to screen space based on these specifications. The view port portion of the transformation is the same as the 2D case.

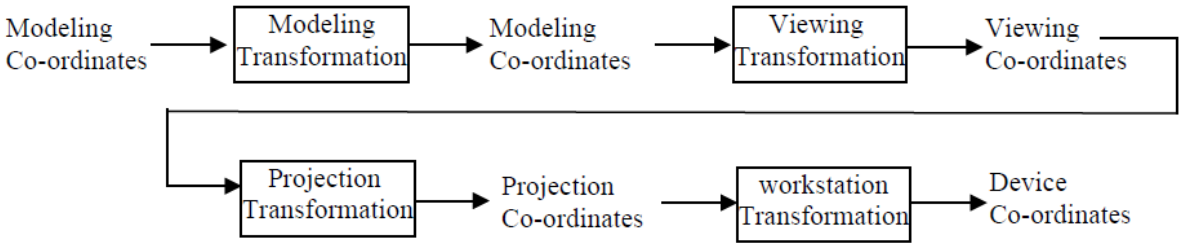
Specification of the window, however, requires additional information and results in a more complex mapping to be defined. Defining a viewing window in world space coordinates is exactly like it sounds; sufficient information needs to be provided to define a rectangular window at some location and orientation. The usual viewing parameters that are specified are:

Eye Point	The position of the viewer in world space
Look Point	The point that the eye is looking at
View Distance	The distance that the window is from the eye
Window Size	The height and width of the window in world space coordinates Up Vector which direction represents "up" to the viewer, this parameter is sometimes specified as an angle of rotation about the viewing axis



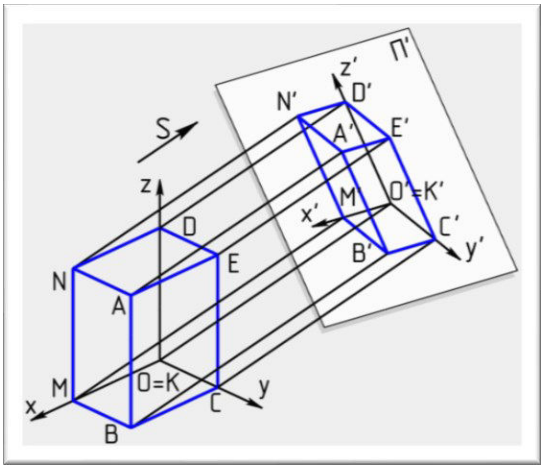
3D Viewing Pipeline

The following figure shows general processing steps for modeling and converting a world-coordinate description of a scene to device coordinates. Once the scene has been modeled, world-coordinate positions are converted to viewing coordinates. The viewing-coordinate system is used in graphics packages as a reference for specifying the observer viewing position and the position of the projection plane, which we can think of in analogy with the camera film plane. Next, projection operations are performed to convert the viewing-coordinate description of the scene to coordinate positions on the projection plane, which will then be mapped to the output device. Objects outside the specified viewing limits are clipped from further consideration, and the remaining objects are processed through visible-surface identification and surface-rendering procedures to produce the display within the device view port.



Projection

3D projection is any method of mapping three-dimensional points to a two-dimensional plane. In general, a projection transforms a N-dimension points to N-1 dimensions. Ideally, an an object is projected by projecting each of its endpoints. But an object has infinite number of points. So we cannot project all those points. What we do is that we project only the corner points of an object on a 2D plane and we will join these projected points by a straight line in a 2D plane.



Basic points in projection:

- **Center of projection**: Point from where projection is taken.
- **Projection plane**: the plane on which the projection of the object is formed.
- **Projectors**: Lines emerging from the center of projection and hitting the projection plane
- **Parallel projection**: In this projection, the co-ordinate positions are transformed to the view plane along parallel lines.
- **Perspective Projection**: In this projection, the co-ordinate positions are transformed to the view plane along lines that converges to a point called as center of projection.

The two types of projections are:

a) Parallel projection

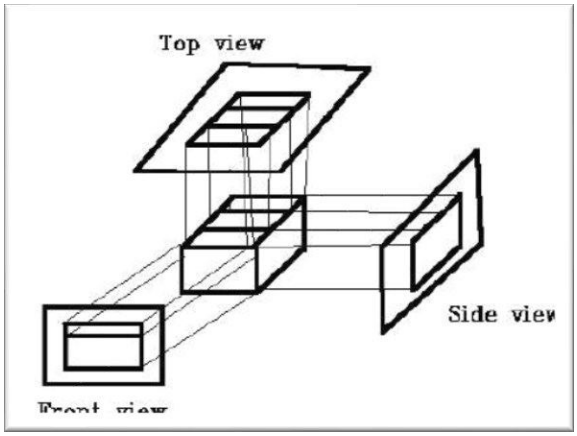
- In this projection, the co-ordinate positions are transformed to the view plane along parallel lines.
- Preserves relative proportions of objects so that accurate views of various sides of an object are obtained but doesn't give realistic representation of the 3D object.
- Can be used for exact measurements so parallel lines remain parallel
- There are two different types of parallel projections

a. Orthographic parallel projection

In orthographic projection, the center of projection lies at infinity and the projections are perpendicular to the view plane. So, a true size and shape of a single face of object is obtained.

In orthographic projection the direction of projection is normal to the projection of the plane. There are three types of orthographic projections −

- Front Projection
- Top Projection
- Side Projection

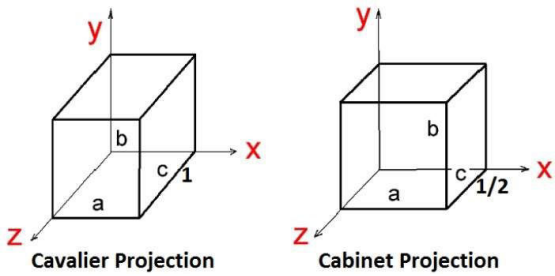


b. Oblique parallel Projection

A projection in which the angle between the projectors and the plane of plane of projection is not equal to 90^0 is called oblique projection.

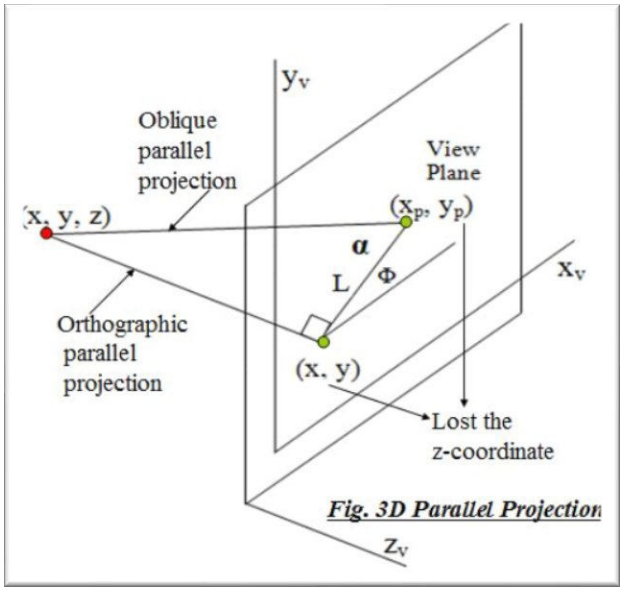
An oblique projection is formed by parallel projections from a center of projection at infinity that intersects the plane of projection at an oblique angle.

There are two types of oblique projections − **Cavalier** and **Cabinet**.



The Cavalier projection makes 45° angle with the projection plane. The projection of a line perpendicular to the view plane has the same length as the line itself in Cavalier projection. In a cavalier projection, the foreshortening factors for all three principal directions are equal.

The Cabinet projection makes 63.4° angle with the projection plane. In Cabinet projection, lines perpendicular to the viewing surface are projected at ½ their actual length.



- Point (x, y, z) is projected to position (x_p , y_p) on the view plane.
- Orthographic projection coordinated on the plane are (x, y). Oblique projection line from (x, y, z) to (x_p , y_p) makes an angle with the line on the projection plane that joins (x_p , y_p) and (x , y). This line of length L is at an angle Φ with the horizontal direction in the projection plane.
- Expressing projection coordinates in terms of x, y, L and Φ as

$$x_p = x + L\cos\Phi$$

$$y_p = y + L\sin\Phi$$

L depends on the angle α and z coordinate of point to be projected

$$\tan\alpha = z/L$$

Thus,

$$L = z / \tan\alpha$$

$$= z\,L_1, \text{ where } L_1 \text{ is the inverse of } \tan\alpha$$

So the oblique projection equations are:

$$x_p = x + z(L_1\cos\Phi)$$

$$y_p = y + z(L\,1\sin\Phi)$$

- The transformation matrix for producing any parallel projection onto the x v y v -plane can be written as:

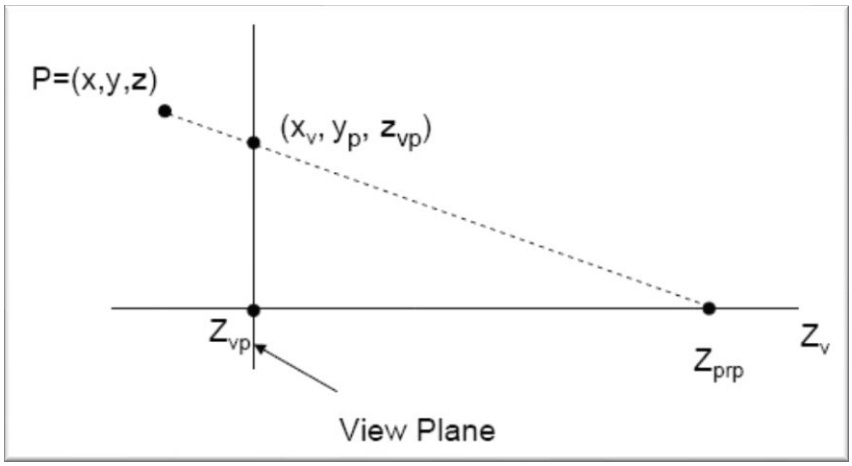
$$M_{parallel} = \begin{pmatrix} 1 & 0 & L_1\cos\Phi & 0 \\ 0 & 1 & L_1\sin\Phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Orthographic projection is obtained when $L_1 = 0$ (occurs at projection angle α of 90 degree) Oblique projection is obtained with non-zero values for L_1.

b) Perspective Projection

In this projection, the co-ordinate positions are transformed to the view plane along lines that converges to a point called as center of projection.

The projection is perspective if the distance is finite or has fixed vanishing point or the incoming rays converge at a point. Thus the viewing dimension of object is not exact i.e. seems large or small according as the movement of the view plane from the object.



The perspective projection perform the operation as the scaling (i.e. zoom in & out) but here the object size is only maximize to the size of real present object i.e. only size reduces not enlarge. This operation is possible here only the movement of the view plane towards or far from the object position.

- We can write equations describing co-ordinates positions along this prospective projection line in parametric form as

  $x' = x - xu$

  $y' = y - yu$

  $z' = z - (z - z_{prp})u$

  Where u= 0 to 1.

  If u=0,we are at position P=(x,y,z).If u=1, we have projection reference point $(0,0,z_{prp})$. On the view plane, $z' = z_{vp}$ then

  $$u = \frac{z_{vp} - z}{z_{prp} - z}$$

  Substituting these value in eq^n for x', y'.

  $$x_p = x - x\left(\frac{z_{vp} - z}{z_{prp} - z}\right) = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) = x\left(\frac{dp}{z_{prp} - z}\right)$$

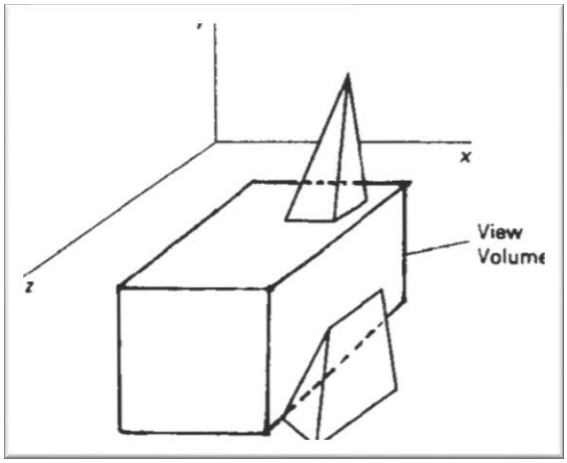
- Similarly,

  $$y_p = y\left(\frac{z_{prp} - z_{vp}}{z - z_{prp}}\right) = y\left(\frac{dp}{z_{prp} - z}\right)$$

  Where $dp = z_{prp} - z_{vp}$ is the distance of the view plane from projection reference point.

Clipping in 3D

Clipping in three dimensions can be accomplished using extensions of two-dimensional clipping methods. Instead of clipping against straight-line window boundaries, we now clip objects against the boundary planes of the view volume. *(The view volume defines the three-dimensional volume in space that, once projected, is to fit within the view port. There are two parts to the view volume: the view plane rectangle and the near and far clipping planes)*



An algorithm for three-dimensional clipping identifies and saves all surface segments within the view volume for display on the output device. All parts of objects that are outside the view volume are discarded. View-volume clipping boundaries are planes whose orientations depend on the type of projection, the projection window, and the position of the projection reference point.

To clip a polygon surface, we can clip the individual polygon edges. To clip a line segment against the view volume, we would need to test the relative position of the line using the view volume's boundary plane equations. An endpoint (x, y, z) of a line segment is

- ➢ Outside a boundary plane: $Ax + By + Cz + D > 0$,

where A, B, C, and D are the plane parameters for that boundary.

- ➢ Inside the boundary if $Ax + By + Cz + D < 0$

Lines with both endpoints outside a boundary plane are discarded, and those with both endpoints inside all boundary planes are saved.

The intersection of a line with a boundary is found using the line equations along with the plane equation. Intersection coordinates (x_1, y_1, z_1) are values that are on the line and that satisfy the plane equation $Ax_1 + By_1 + Cz_1 + D = 0$.

To clip a polygon surface, we can clip the individual polygon edges.

As in two-dimensional viewing, the projection operations can take place before the view-volume clipping or after clipping. All objects within the view volume map to the interior of the specified projection window. The last step is to transform the window contents to a two-dimensional view port, which specifies the location of the display on the output device.

Z-clipping

Z-clipping, or depth clipping, refers to techniques that selectively render certain scene objects based on their depth relative to the screen. Most graphics toolkits allow the programmer to specify a "near" and "far" clip depth, and only portions of objects between those two planes are displayed.

Importance of clipping in video games

- ➢ Maximize the game's frame rate and visual quality
- ➢ Speed up the rendering of the current scene
- ➢ Economizing the use of renderer time and memory within the hardware's capability

Chapter 6

Visible Surface Detection

Visible surface detection or Hidden surface removal is major concern for realistic graphics for identifying those parts of a scene that are visible from a chosen viewing position. Several algorithms have been developed. Some require more memory, some require more processing time and some apply only to special types of objects.

Visible surface detection methods are broadly classified according to whether they deal with objects or with their projected images.

➢ **Object-Space Methods (OSM):**

An object-space method compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible.

E.g. Back-face detection method

➢ **Image-Space Methods (ISM):**

In an image-space algorithm, visibility is decided point by point at each pixel position on the projection plane. Most visible-surface algorithms use image-space methods.

E.g. Depth-buffer method, Scan-line method, Area-subdivision method

➢ **List Priority Algorithms**

This is a hybrid model that combines both object and image precision operations. Here, depth comparison & object splitting are done with object precision and scan conversion (which relies on ability of graphics device to overwrite pixels of previously drawn objects) is done with image precision.

E.g. Depth-Sorting method, BSP-tree method

A. Back Face Detection Method (Plane Equation method)

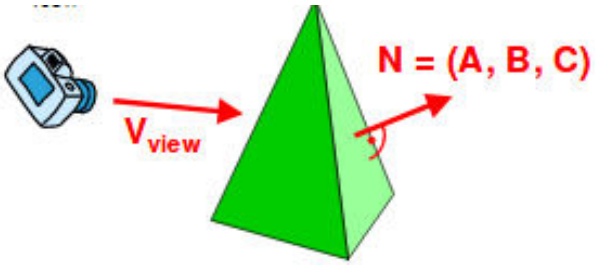
In a solid object, there are surfaces which are facing the viewer (front faces) and there are surfaces which are opposite to the viewer (back faces). These back faces contribute to approximately half of the total number of surfaces. Since we cannot see these surfaces anyway, to save processing time, we can remove them before the clipping process with a simple test.

Each surface has a normal vector. If this vector is pointing in the direction of the center of projection, it is a front face and can be seen by the viewer. If it is pointing away from the center of projection, it is a back face and cannot be seen by the viewer.

The test is very simple, if the z component of the normal vector is positive, then, it is a back face. If the z component of the vector is negative, it is a front face.

Principle:

- ➢ Remove all surfaces pointing away from the viewer
- ➢ Eliminate the surface if it is completely obscured by other surfaces in front of it
- ➢ Render only the visible surfaces facing the viewer



Back facing and front facing faces can be identified using the sign of V • N where V is the view vector and N is normal vector.

- ➢ If V • N > 0, back face
- ➢ if V • N < 0 front face
- ➢ if V • N= 0 on line of view

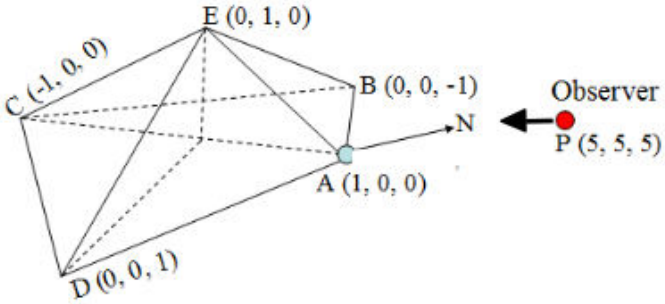
Numerical

Q. Find the visibility for the surface AED where an observer is at P (5, 5, 5).

Here,

AE = (0-1) i + (1-0) j + (0-0) k = - i + j

AD= (0-1) i + (0-0) j + (1-0) k = - i + k



Step-1: Normal vector N for AED

Thus, N = AE x AD = $\begin{pmatrix} i & j & k \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$ $= i(1-0) - j(-1+0) + k(0+1)$
$= i + j + k$

Step-2: If observer at P(5, 5, 5) so we can construct the view vector V from view point A(1,0, 0) as:

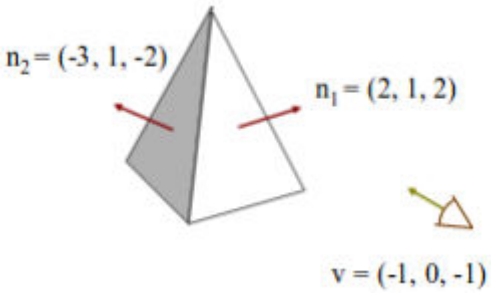
V = PA = (1-5) i + (0-5) j + (0-5) k = -4i - 5j - 5k

Step-3: To find the visibility of the object, we use dot product of view vector V and normal vector N as:

V • N = (-4i - 5j – 5k) • (i + j + k) = -4-5-5 = -14< 0

This shows that the surface is visible for the observer.

Q. Find the visibility of n1 and n2 from V.



n1 • v = (2, 1, 2) ·(-1, 0, -1) = -2 – 2 = -4,

i.e n1 • v< 0

So, n1 front facing polygon

n2 • v = (-3, 1, -2) ·(-1, 0, -1) = 3 + 2 = 5

i.e n2 • v > 0

so n2 back facing polygon

B. Depth Buffer (Z-Buffer) Method

This method is developed by Edwin Catmull and is an image-space approach. The basic idea is to test the Z-depth of each surface to determine the closest (visible) surface.

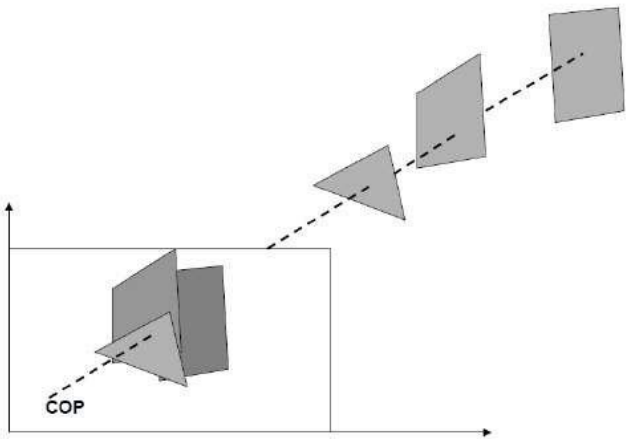
In this method each surface is processed separately one pixel position at a time across the surface. The depth values for a pixel are compared and the closest (smallest z) surface determines the color to be displayed in the frame buffer.

It is applied very efficiently on surfaces of polygon. Surfaces can be processed in any order. To override the closer polygons from the far ones, two buffers named frame buffer and depth buffer, are used.

- ➢ Depth buffer is used to store depth values for (x, y) position, as surfaces are processed (0 ≤ depth ≤ 1).
- ➢ The frame buffer is used to store the intensity value of color value at each position (x, y).

The z-coordinates are usually normalized to the range [0, 1]. The 0 value for z-coordinate indicates back

clipping pane and 1 value for z-coordinates indicates front clipping pane.



Algorithm

Step-1 − Set the buffer values −

- ➢ Depthbuffer (x, y) = 0
- ➢ Framebuffer (x, y) = background color

Step-2 − Process each polygon (One at a time)

- ➢ For each projected (x, y) pixel position of a polygon, calculate depth z.
- ➢ If Z > depthbuffer (x, y)
- Compute surface color,
- set depthbuffer (x, y) = z,
- framebuffer (x, y) = surfacecolor (x, y)

Advantages

- It is easy to implement.
- It reduces the speed problem if implemented in hardware.
- It processes one object at a time.

Disadvantages

- It requires large memory.
- It is time consuming process.

C. Scan-Line Method

This image-space method for removing hidden surfaces is an extension of the scan-line algorithm for filling polygon interiors where, we deal with multiple surfaces rather than one. Each scan line is processed with calculating the depth for nearest view for determining the visible surface of intersecting polygon. When the visible surface has been determined, the intensity value for that position is entered into the refresh buffer.

To facilitate the search for surfaces crossing a given scan line, we can set up an **active list** of edges from information in the edge table that contain only edges that cross the current scan line, sorted in order of increasing x. In addition, we define a **flag** for each surface that is set on or off to indicate whether a position along a scan line is inside or outside of the surface. Scan lines are processed from left to right. At the leftmost boundary of a surface, the surface flag is turned on; and at the rightmost boundary, it is turned off.

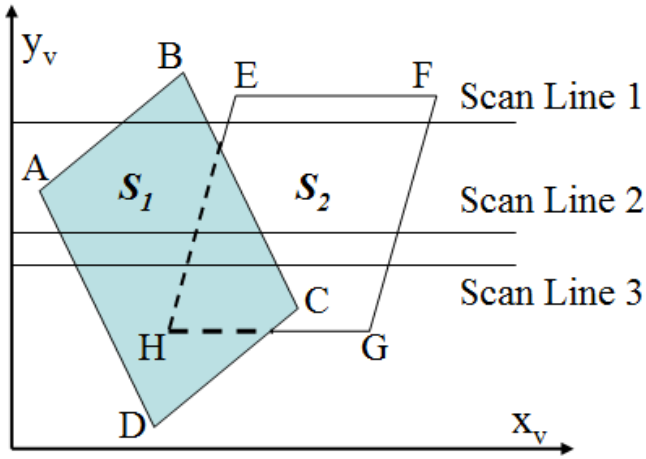


Fig. Scan lines crossing the projection of two surfaces, S_1 and S_2 in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

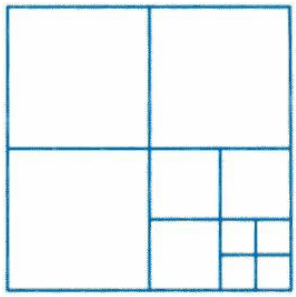
The active list for scan line -1 contains information from the edge table for edges AB, BC, EH, and FG. For positions along this scan line between edges AB and BC, only the flag for surface S1 is on. Therefore, no depth calculations are necessary, and intensity information for surface S1 , is entered from the polygon table into the refresh buffer. Similarly, between edges EH and FG, only the flag for surface S2 is on. NO other positions along scan line-1 intersect surfaces, so the intensity values in the other areas are set to the background intensity.

For scan lines 2 and 3, the active edge list contains edges AD, EH, BC, and FG. Along scan line 2 from

edge AD to edge EH, only the flag for surface S1 , is on. But between edges EH and BC, the flags for both surfaces are on. In this interval, depth calculations must be made using the plane coefficients for the two surfaces. For this example, the depth of surface S1 is assumed to be less than that of S2 , so intensities for surface S1 are loaded into the refresh buffer until boundary BC is encountered. Then the flag for surface S1 goes off, and intensities for surface S2 are stored until edge FG is passed.

D. Area-Subdivision Method

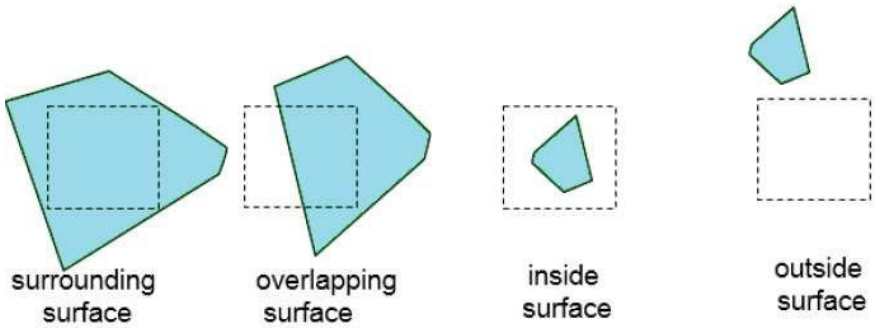
The area-subdivision method takes advantage by locating those view areas that represent part of a single surface.



Divide the total viewing area into smaller and smaller rectangles until each small area is the projection of part of a single visible surface or no surface at all.

Continue this process until the subdivisions are easily analyzed as belonging to a single surface or until they are reduced to the size of a single pixel. An easy way to do this is to successively divide the area into four equal parts at each step. There are four possible relationships that a surface can have with a specified area boundary.

- Surrounding surface − One that completely encloses the area.
- Overlapping surface − One that is partly inside and partly outside the area.
- Inside surface − One that is completely inside the area.
- Outside surface − One that is completely outside the area.



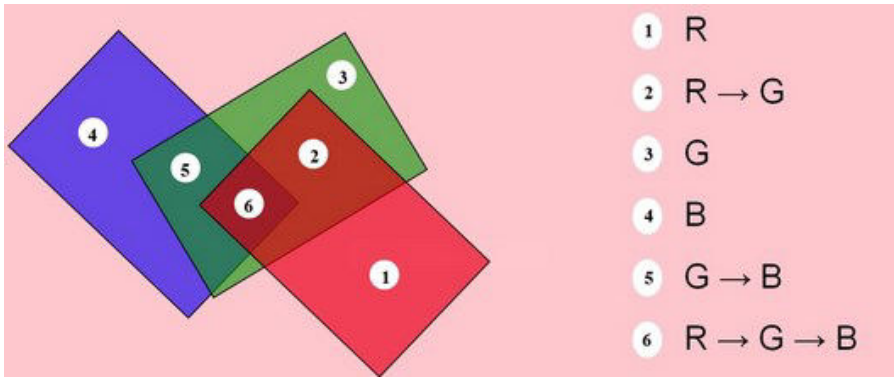
The tests for determining surface visibility within an area can be stated in terms of these four classifications. No further subdivisions of a specified area are needed if one of the following conditions is true −

- All surfaces are outside surfaces with respect to the area.
- Only one inside, overlapping or surrounding surface is in the area.
- A surrounding surface obscures all other surfaces within the area boundaries.

E. A-Buffer Method

The A-buffer method is an extension of the depth-buffer method. The A-buffer method is a visibility detection method developed at Lucas film Studios for the rendering system Renders Everything You Ever Saw (REYES).

The A-buffer expands on the depth buffer method to allow transparencies. The key data structure in the A-buffer is the accumulation buffer.

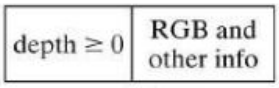


Each position in the A-buffer has two fields −

- Depth field − It stores a positive or negative real number
- Intensity field − It stores surface-intensity

information or a pointer value

- If depth >= 0, the number stored at that position is the depth of a single surface overlapping the corresponding pixel area.



- If depth < 0, it indicates multiple-surface contributions to the pixel intensity. The intensity field then stores a pointer to a linked list of surface data.

- The surface buffer in the A-buffer includes −
  - RGB intensity components
  - Opacity Parameter
  - Depth
  - Percent of area coverage
  - Surface identifier

The algorithm proceeds just like the depth buffer algorithm. The depth and opacity values are used to determine the final color of a pixel.

F. Depth Sorting Method

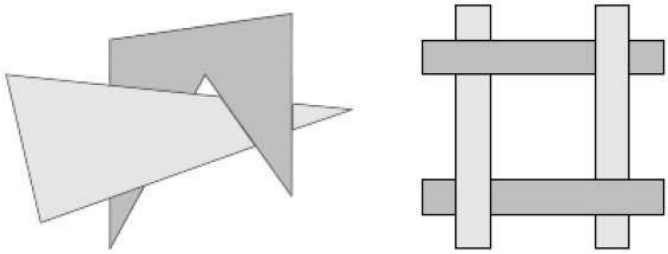
Depth sorting method uses both image space and object-space operations. The depth-sorting method performs two basic functions −

- First, the surfaces are sorted in order of decreasing depth.
- Second, the surfaces are scan-converted in order, starting with the surface of greatest depth.

The scan conversion of the polygon surfaces is performed in image space. This method for solving the hidden-surface problem is often referred to as the **painter's algorithm.**

The algorithm begins by sorting by depth.

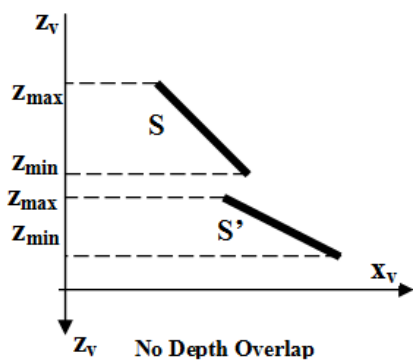
- Sort all surfaces according to their distances from the view point.
- Render the surfaces to the image buffer one at a time starting from the farthest surface.
- Surfaces close to the view point will replace those which are far away.

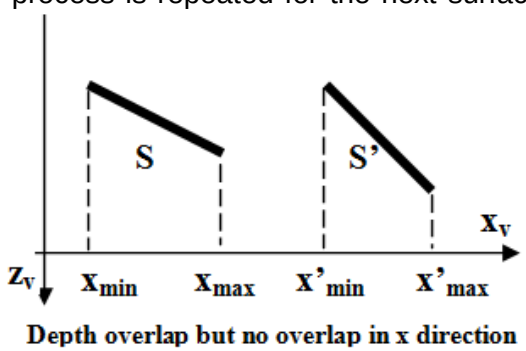


- After all surfaces have been processed, the image buffer stores the final image.

Example:

Assuming we are viewing along the z axis. Surface S with the greatest depth is then compared to other surfaces in the list to determine whether there are any overlaps in depth. If no depth overlaps occur, S can be scan converted. This process is repeated for the next surface in the list. However, if depth overlap is detected, we need to make some additional comparisons to determine whether any of the surfaces should be reordered.





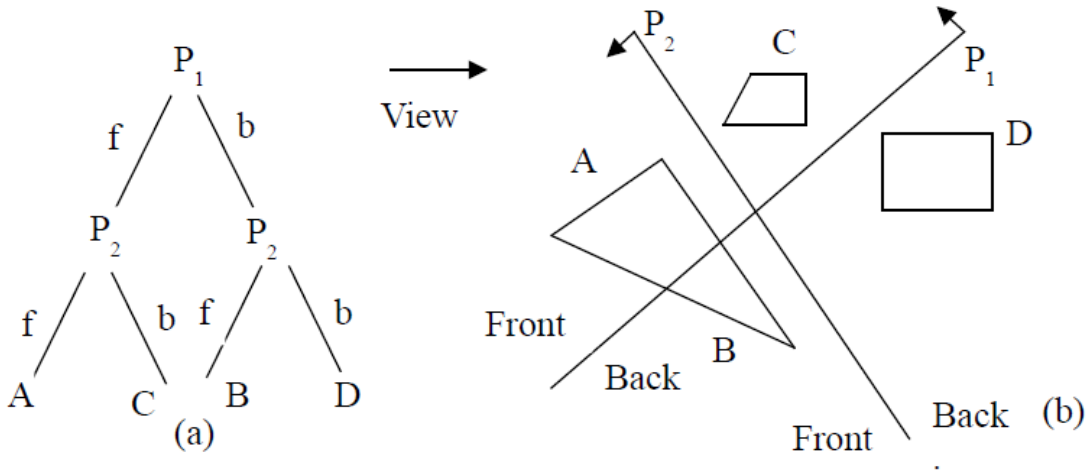
G. BSP Tree Method

A binary space partitioning (BSP) tree is an efficient method for determining object visibility by painting surfaces onto the screen from back to front as in the painter's algorithm.

The BSP tree is particularly useful when the view reference point changes, but object in a scene are at fixed position.

Applying a BSP tree to visibility testing involves identifying surfaces that are "inside" or "outside" the partitioning plane at each step of space subdivision relative to viewing direction. It is useful and efficient for calculating visibility among a static group of 3D polygons as seen from an arbitrary viewpoint.

In the following figure,



Here plane P_1 partitions the space into two sets of objects, one set of objects is back and one set is in front of partitioning plane relative to viewing direction. Since one object is intersected by plane P_1, we divide that object into two separate objects labeled A and B. Now object A&C are in front of P_1, B and D are 4 back of P_1.

We next partition the space with plane P_2 and construct the binary free as fig (a). In this tree, the objects are represented as terminal nodes, with front object as left branches and behind object as right branches.

When BSP tree is complete, we process the tree by selecting surface for displaying in order back to front. So fore ground object is painted over back ground objects.

Chapter 7

Illumination and Shading

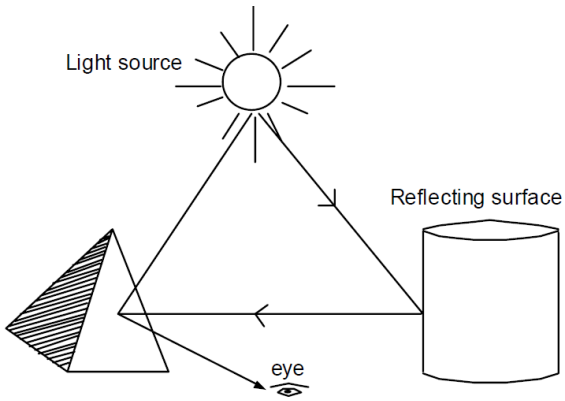
Illumination, an observable property and effect of light. The illumination of the subject of a drawing or painting is a key element in creating an artistic piece, and the interplay of light and shadow is a valuable method. The placement of the light sources can make a considerable difference in the type of message that is being presented. Multiple light sources can wash out any wrinkles in a person's face, for instance, and give a more youthful appearance. In contrast, a single light source, such as harsh daylight, can serve to highlight any texture or interesting features.

An **illumination model**, also called a lighting model and sometimes referred to as a shading model, is used to calculate the intensity of light that we should see at a given point on the surface of an object.

Surface rendering means a procedure for applying a lighting model to obtain pixel intensities for all the projected surface positions in a scene. A surface-rendering algorithm uses the intensity calculations from an illumination model to determine the light intensity for all projected pixel positions for the various surfaces in a scene. Surface rendering can be performed by applying the illumination model to every visible surface point.

Light Sources

When we view an opaque non-luminous object, we see reflected light from the surfaces of the object. The total reflected light is the sum of the contributions from a single light source or illuminated nearby objects.



a) Point source:

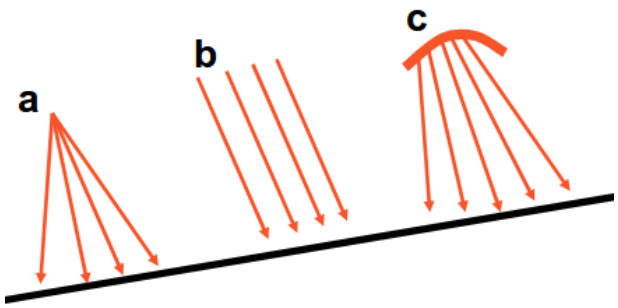
   All light rays originate at a point and radially diverge

b) Parallel source

   Light rays are all parallel. May be modeled as a point source at infinite distance (the sun)

c) Distributed source

   All light rays originate at a finite area in space. It models a nearby source, such as a fluorescent light.



Illumination models:

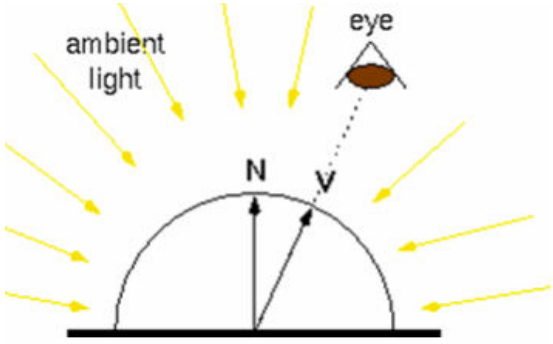
Illumination models are used to calculate light intensities that we should see at a given point on the surface of an object. Lighting calculations are based on the optical properties of surfaces, the background lighting conditions and the light source specifications. All light sources are considered to be point sources, specified

with a co-ordinate position and an intensity value (color).

Some illumination models are:

1. Ambient Light

Ambient light means the light that is already present in a scene, before any additional lighting is added. It usually refers to natural light, either outdoors or coming through windows etc. It can also mean artificial lights such as normal room lights**.**



The equation expressing this simple model is

$$I = K_a$$

Where I is the resulting intensity and $K_a$is the object's intrinsic intensity.

If we assume that ambient light impinges equally on all surface from all direction, then

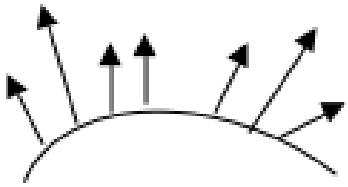
$$I = I_a K_a$$

Where $I_a$is intensity of ambient light. The amount of light reflected from an object's surface is determined byK_a, the ambient-reflection coefficient, ranges from 0 to 1.

2. Diffuse reflection:

Objects illuminated by ambient light are uniformly illuminated across their surfaces even though light are more or less bright in direct proportion of ambient intensity. . The object's brightness varies form one part to another, depending on the direction of and distance to the light source.

Diffuse reflections are constant over each surface in a scene, independent of the viewing direction.



Diffuse lighting of an object approximates how photons will bounce off of the object's surface in many different directions. Assuming diffuse reflections from the surface are scattered with equal intensity in all directions, independent of the viewing direction (surface called. "Ideal diffuse reflectors") also called Lambertian reflectors and governed by Lambert's cosine law.

$$I_{diff} = K_d I_l \cos\Theta$$

Where $I_l$is the intensity of the point light source.

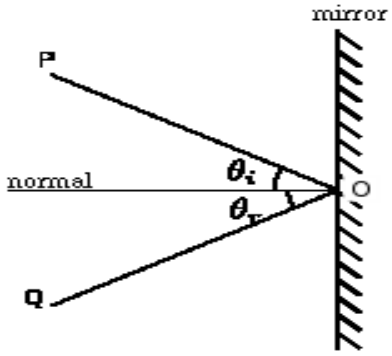
If N is unit vector normal to the surface & L is unit vector in the direction to the point slight source then

$$I_l,\text{diff} = K_d I_l (N.L)$$

In addition, many graphics packages introduce an ambient reflection coefficient $K_a$to modify the ambient-light intensity I_a

$$\text{Idiff} = K_a I_a + K_d I_l (N.L)$$

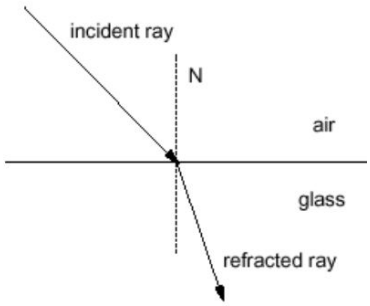
3. Specular reflection



Specular reflection, also known as regular reflection, is the mirror-like reflection of waves, such as light, from a surface. In this process, each incident ray is reflected at the same angle to the surface normal as the incident ray, but on the opposing side of the surface normal in the plane formed by incident and reflected rays. The result is that an image reflected by the surface is reproduced in mirror-like (*specular*) fashion

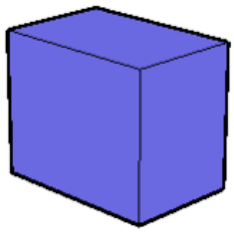
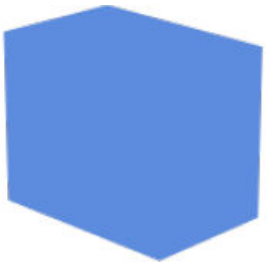
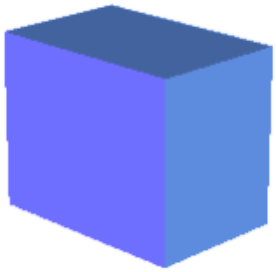
Light refraction

Refraction occurs when light travels through transparent or semi-transparent objects of different densities (for example, from air to glass).



Surface Shading Method

In computer graphics, shading refers to the process of altering the color of an object/surface/polygon in the 3D scene, based on things like (but not limited to) the surface's angle to lights, its distance from lights, its angle to the camera and material properties (e.g. bidirectional reflectance distribution function) to create a photo realistic effect. Shading is performed during the rendering process by a program called a shader.

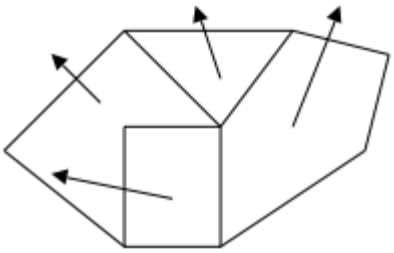
		
Rendered image of a box. This image has no shading on its faces, but uses edge lines to separate the faces.	This is the same image with the edge lines removed.	This is the same image rendered with shading of the faces to alter the colors of the 3 faces based on their angle to the light sources.

A shading model is used in computer graphics to simulate the effects of light shining on a surface. The intensity that we see on a surface is dependent upon

- The type of light sources.
- The surface characteristics (eg. Shining, matte, dull, and opaque or transparent).

Types of Shading

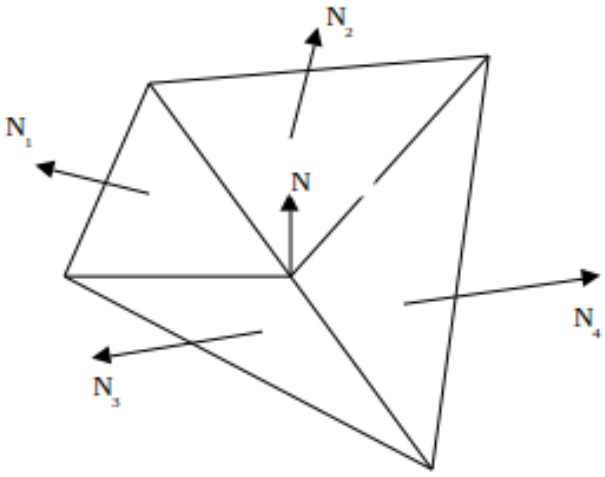
1. Constant Intensity Shading (Flat Shading)



- ➢ The intensity value for the whole polygon is calculated once and the whole polygon will be shaded with the same intensity value.
- ➢ Each rendered polygon has a single normal vector; shading for the entire polygon is constant across the surface of the polygon.
- ➢ Fast and simple but cannot model specular reflection.

2. Gouraud Shading

- ➢ The intensity value is calculated once for each vertex of a polygon.
- ➢ Each polygon has one normal vector per vertex, the color of each vertex is computed and then interpolated across the surface of the polygon.
- ➢ The interpolation of color values can cause bright or dark intensity streaks, called the Mach-bands, to appear on the surface
- ➢ Calculation for each polygon surfaces



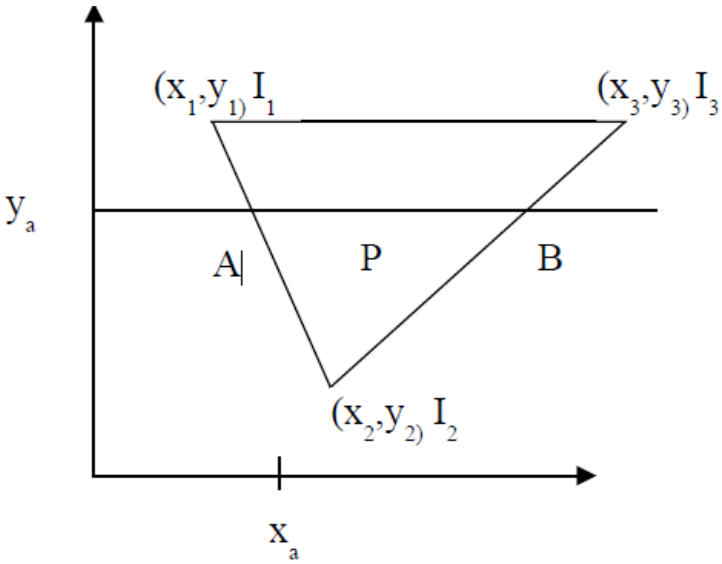
- Determine the average unit normal vector at each vertex. At each polygon vertex, we obtain a normal vertex by averaging the surface normals of all polygons sharing the vertex as:

$$N_v = \frac{N_1+N_2+N_3+N_4}{|N_1+N_2+N_3+N_4|}$$

Where N_v is normal vector at a vertex sharing Four surfaces as in figure.

- Apply illumination model to calculate each vertex intensity.
- Linearly interpolate the vertex intensity over the surface of the polygon.

Once N_v is known, intensity at the vertices can obtain from lighting model.



- Here in figure, the intensity of vertices I_1 , I_2, I_3 are obtained by averaging normals of each surface sharing the vertices and applying a illumination model.

- For each scan line , intensity at intersection of line with Polygon edge are linearly interpolated from the intensities at the edge end point.

So Intensity at intersection point A, I_a is obtained by linearly interpolating intensities of I_1 and I_2 as`

$$I_a = \frac{y_a-y_2}{y_1-y_2} I_1 + \frac{y_1-y_a}{y_1-y_2} I_2$$

Similarly, the intensity at point B is obtained by linearly interpolating intensities at I_2 and I_3 as

$$I_b = \frac{y_a - y_2}{y_3 - y_2} I_3 + \frac{y_3 - y_a}{y_3 - y_2} I_2$$

The intensity of a point P in the polygon surface along scan-line is obtained by linearly interpolating intensities at I_a and I_b as,

$$I_p = \frac{x_p - x_a}{x_b - x_a} I_b + \frac{x_b - x_p}{x_b - x_a} I_a$$

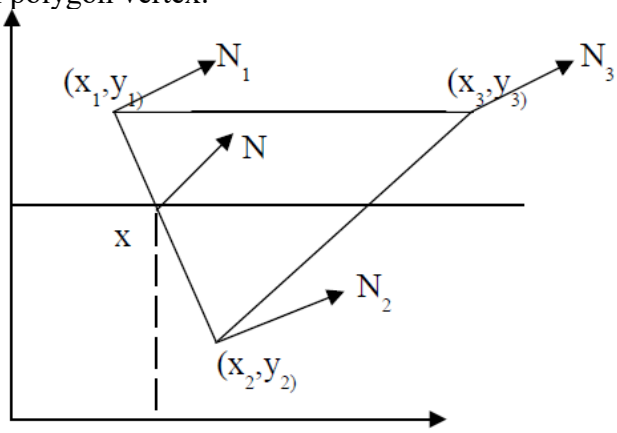
Advantages: Removes intensity discontinuities at the edge as compared to constant shading.

Disadvantages: Highlights on the surface are sometimes displayed with anomalous shape and linear intensity interpolation can cause bright or dark intensity streak called mach-bands.

3. Phong Shading

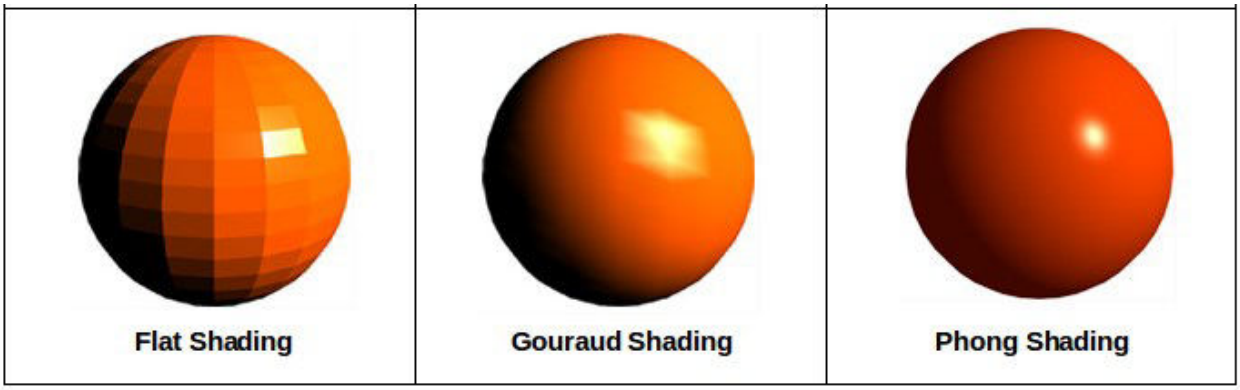
- ➢ Each rendered polygon has one normal vector per vertex; shading is performed by interpolating the vectors across the surface and computing the color for each point of interest.
- ➢ Interpolating the normal vectors gives a reasonable approximation to a smoothly-curved surface while using a limited number of polygons
- ➢ This method greatly reduces the Mach-band problem but it requires more computational time
- ➢ A polygon surface is rendered with Phong shading by carrying out following calculations.

- Determine the average normal unit vectors at each polygon vertex.
- Linearly interpolate vertex normals over the surface of polygon.
-Apply illumination model along each scan line to calculate the pixel intensities for the surface point.



In figure, N_1, N_2, N_3 are the normal unit vectors at each vertex of polygon surface. For scan-line that intersect an edge, the normal vector N can be obtained by vertically interpolating normal vectors
of the vertex on that edge as.

$$N = \frac{y - y_2}{y_1 - y_2} N_1 + \frac{y_1 - y}{y_1 - y_2} N_2$$



4. Fast Phong Shading:

Fast Phong shading approximates the intensity calculations using a Taylor series expansion and Triangular surface patches. Since Phong shading interpolates normal vectors from vertex normals , we can express the surface normal N at any point (x,y) over a triangle as

$$N = Ax + By + C$$

Where A,B,C are determined from the three vertex equations.

$$N_k = Ax_k + By_k + C, \quad k = 1,2,3 \text{ for } (x_k , y_k) \text{ vertex.}$$

Omitting the reflectivity and attenuation parameters

$$I_{\text{diff}}(x,y) = \frac{L.N}{|L|.|N|} = \frac{L.(Ax+By+C)}{|L|.|Ax+By+C|} = \frac{(L.A)x+(L.B)y+(L.C)}{|L|.|Ax+By+C|}$$

Mach bands

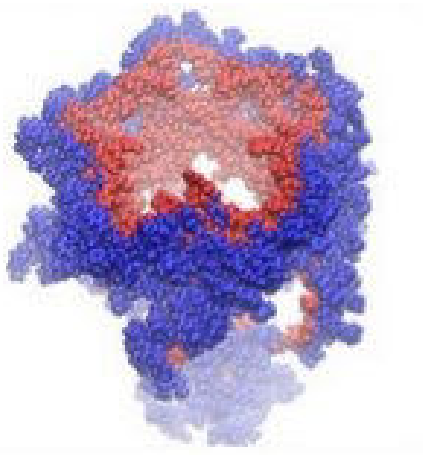
Mach bands are an optical illusion where a band of gradients will appear in places to be lighter or darker than they actually are.

When looking at Mach bands, one sees a central band of a light to dark gradient surrounded on one side by the lightest color and on the opposite side by the darkest color.

Mach bands, as well as numerous other visual and perceptual illusions, help scientists study the way the eye and brain process visual information.

Depth Cueing

It is the process of rendering distant objects at a lower intensity than near ones, hence giving them the appearance of depth**.**



Depth cuing indicates the process in which the lines closest to the viewing position are displayed with the highest intensities, and lines farther away are displayed with decreasing intensities. Depth cueing is applied by choosing maximum and minimum intensity (or color) values and a range of distances over which the intensities are to vary.

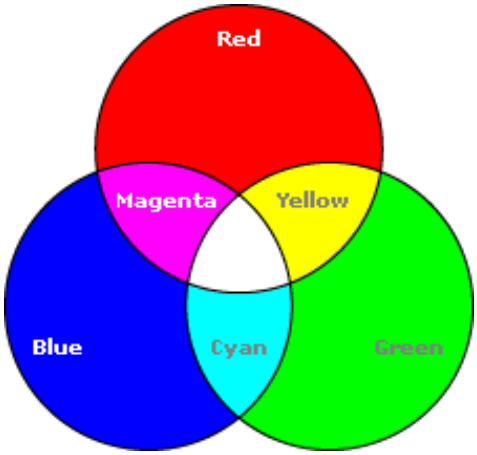
Computer Color Models

A color model is simply a way to define color. A model describes how color will appear on the computer screen or on paper. Three popular color models are:

1. RGB (Red, Green, Blue)
2. CMYK (Cyan, Magenta, Yellow, Black)

1. RGB (Red, Green, Blue)

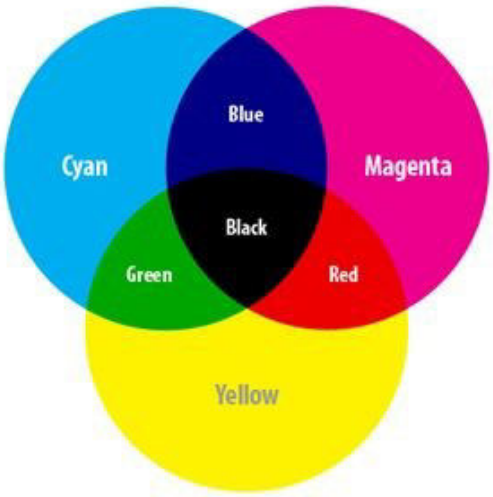
The RGB model is used when working with screen based designs. A value between 0 and 255 is assigned to each of the light colors, Red, Green and Blue. So for example, if we wanted to create a purely blue color, Red would have a value of 0, Green would have a value of 0 and Blue would have a value of 255 (pure blue). To create black, Red, Green and Blue would each have a value of 0 and to create white, each would have a value of 255. RGB is known as an "additive" model and is the opposite of the subtractive color model.



2. CMYK (Cyan, Magenta, Yellow, Black)

The CMYK model is used for print work and it describes colors based on their percentage of Cyan, Magenta, Yellow and Black. These four colors are used by commercial printers and bureaus and you may also find that your home printer uses these colors too. These four colors are needed to reproduce full color artwork in magazines, books and brochures. By combining Cyan, Magenta, Yellow and Black on paper in varying percentages, the illusion of lots of colors is created.

CMYK is known as a "subtractive" color model. White is the natural color of the paper or other background, while black results from a full combination of colored inks.



Chapter 8

Graphical Language

Machine languages are the first generation of programming languages.

A machine language is a machine-dependent, low-level language that uses binary code to interact with a specific computer system. A machine-dependent language works only on a specific computer system and its components.

A Machine Independent language is one that can run on any machine. An example of this would be Java.

Machine Independent language can take the compiled code for any given machine and run it on the machine we are attempting to run it on. A TRULY machine-independent language would produce exactly the same output no matter which computer it was run on.

Graphics Software

Graphics software refers to a program or collection of programs that enable a person to manipulate images or models visually on a computer.

There are two general classifications for graphical Softwares.

A. General Programming Packages:

General programming packages provides an extensive set of graphics functions that can be used in a high-level programming language, such as C or FORTRAN. E.g. GL (Graphics Library) system on Silicon Graphics equipment, which includes basic functions for generating picture components (straight lines, polygons, circles, and other figures), setting colors and intensity values, selecting views, and applying transformations

B. Special Purpose Application Packages:

Special purpose applications packages are, in contrast designed for non programmers, so that users can generate displays without worrying about how graphics operations work.

The interface to the graphics routines in such packages allows users to communicate with the programs in their own terms. For example: the artist's painting programs and various business, medical, and Computer-Aided Design (CAD) systems.

There are many graphics software available in market; they can be categories as:

1) **Paint program:** Paint program works with bit map images.
2) **Photo manipulation program:** It works with bit map images and is widely used to edit digitized photographs.
3) **Computer Aided Design program:** CAD software is used in technical design fields to create models of objects that will be built or manufactured. CAD software allows user to design objects in 3 dimensions and can produce 3-D frames and solid models.
4) **3-D Modelling Programs:** It is used to create visual effects. 3-D modeling program works by creating objects like surface, solid, polygon etc.
5) **Animation:** Computer are used to create animation for use in various fields, including games, and movies composing to allow game makers and film makers to add characters and objects to scenes that did not originally contain them.

Software standards

The primary goal of standardized graphics software is portability. When packages are designed with standard graphics functions, software can be moved easily from one hardware system to another and used in different implementations and applications. Without standards, programs designed for one hardware system often cannot be transferred to another system without extensive rewriting of the programs.

A) **General Kernel System (GKS)**

The Graphical Kernel System (GKS) was the first ISO standard for low-level computer graphics, introduced in 1977. The main purpose of GKS is the production and manipulation of 2D pictures in a way that does not depend on the system of graphical device used.

In GKS, pictures are constructed from a number of basic building blocks. These basic building blocks are called as primitives. There are five types of primitives in GKS.

- Polyline – draws sequence of connected lines

- Polymaker – marks a sequence of points with same symbol

- Fill Area – displays a specificed area

- Text – draws a string of characters

- Cell Array – displays an image composed of a variety of colors or gray scales.

B) **PHIGS (Programmer's Hierarchical Interactive Graphics System)**

PHIGS, short for the Programmer's Hierarchical Interactive Graphics System, is basically a library of about 400 functions that allow the user to display and interact with 2-D and 3-D graphics. It is an international standard, being created by the International Organization for Standardization (ISO). PHIGS hides hardware-dependent details from the user; so, for example, it allows an application draw on a plotter the same way it draws on a computer screen.

PHIGS provides a set of familiar graphics objects called *primitives*, each with attributes that control its location, orientation, color, and appearance.

PHIGS Primitives

- **polyline:** which draws a sequence of connected line segments;

- **polymarker:** which marks a sequence of points with a symbol;

- **fill area:** which defines the boundary of an area to be displayed;

- **fill area set:** which defines the boundaries of a set of areas to be displayed as one;

- **text:** which draws a sequence of characters;

- **annotation text:** which draws a sequence of characters to annotate a drawing;

- **cell array:** which displays an image;

Over View of Graphical File Formats

A) JPEG

Joint Photographic Experts Group) is a lossy compression method; JPEG-compressed images are usually stored in the JFIF (JPEG File Interchange Format) file format. The JPEG/JFIF filename extension is JPG or JPEG. Nearly every digital camera can save images in the JPEG/JFIF format, which supports eight-bit grayscale images and 24-bit color images (eight bits each for red, green,

and blue). JPEG applies lossy compression to images, which can result in a significant reduction of the file size.

B) TIFF

The TIFF (Tagged Image File Format) format is a flexible format that normally saves eight bits or sixteen bits per color (red, green, blue) for 24-bit and 48-bit totals, respectively, usually using either the TIFF or TIF filename extension. TIFF image format is not widely supported by web browsers.

C) GIF

GIF (Graphics Interchange Format) is in normal use limited to an 8-bit palette, or 256 colors (while 24-bit color depth is technically possible). GIF is most suitable for storing graphics with few colors, such as simple diagrams, shapes, logos, and cartoon style images, as it uses LZW lossless compression, which is more effective when large areas have a single color, and less effective for photographic images.

D) BMP

The BMP file format (Windows bitmap) handles graphic files within the Microsoft Windows OS. Typically, BMP files are uncompressed, and therefore large and lossless; their advantage is their simple structure and wide acceptance in Windows programs.

E) PNG

The PNG (Portable Network Graphics) file format was created as a free, open-source alternative to GIF. The PNG file format supports eight-bit palleted images (with optional transparency for all palette colors) and 24-bit true color (16 million colors) or 48-bit true color with and without alpha channel - while GIF supports only 256 colors and a single transparent color.

Data structure in Computer Graphics

A) Triangle mesh: A **triangle mesh** is a type of polygon mesh in computer graphics. It comprises a set of triangles (typically in three dimensions) that are connected by their common edges or corners. The data structure representing the mesh provides support for two basic operations, inserting triangles and removing triangles.

B) Quad-edge: A **quad-edge** data structure is a computer representation of the topology of a two-dimensional or three-dimensional map, that is, a graph drawn on a (closed) surface. It represents simultaneously both the map, its dual and mirror image

C) Polygon Mesh: A **polygon mesh** is a collection of vertices, edges and faces that defines the shape of a polyhedral object in 3D computer graphics and solid modeling. The faces usually consist of triangles (triangle mesh), quadrilaterals, or other simple convex polygons, since this simplifies rendering, but may also be composed of more general concave polygons, or polygons with holes.

D) Octree: An **octree** is a tree data structure in which each internal node has exactly eight children. Octrees are most often used to partition a three-dimensional space by recursively subdividing it

Open GL

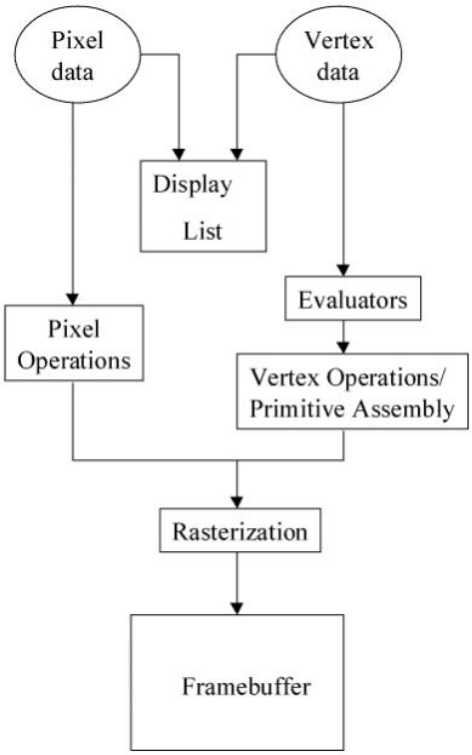
OpenGL is a low-level graphics library specification. It makes available to the programmer a small set of geometric primitives - points, lines, polygons, images, and bitmaps. OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives,

together with commands that control how these objects are rendered (drawn).

Since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), the OpenGL Utility Toolkit (GLUT) has been created to aid in the development of more complicated three-dimensional objects such as a sphere, a torus, and even a teapot. GLUT may not be satisfactory for full-featured OpenGL applications, but it is a useful starting point for learning OpenGL.

Rendering Pipeline

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline.



Although this is not a strict rule of how OpenGL is implemented, it provides a reliable guide for predicting what OpenGL will do. Geometric data (vertices, line, and polygons) follow a path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images and bitmaps) are treated differently for part of the process. Both types of data undergo the same final step before the final pixel data is written to the frame buffer

Libraries

OpenGL provides a powerful but primitive set of rendering command, and all higher-level drawing must be done in terms of these commands. There are several libraries that allow you to simplify your programming tasks, including the following:

- OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections and rendering surfaces.
- OpenGL Utility Toolkit (GLUT) is a window-system-independent toolkit, written by Mark Kilgard, to hide the complexities of differing window APIs.