# 6th Sem BCA

## Paper: BCACsT6.7: COMPUTER GRAPHICS

3Hrs /Week Total: 48Hrs

**Chapter 1: Graphics Systems and Output Primitives** **12Hrs**

Application for CG, CG classification-Graphics software-CRT functioning-Factors affecting CRT- Raster and Random scan monitors-Shadow mask method, display processor with raster system- Raster co-ordinate system-color mapping- Instruction set and Raster system applications.
Output Primitives: Line drawing methods-Direct, DDA and Bresenhams, line attributes, Circle drawing – Direct and midpoint circle drawing – ellipse drawing-Bresenhams ellipse algorithm-Area filling-scanline area filling and character attributes

**Chapter 2: Two-dimensional Transformation** **8 Hrs**

Basic Transformation, Translation, Rotation-Rotation with arbitrary point, Scaling-Scaling with fixed point, Reflection and Shear, matrix representation- homogeneous co-ordinates, Composite transformation-Raster methods for transformation.

**Chapter 3: Windowing and Clipping** **6 Hrs**

Window, viewport, viewing transformations, clipping process, point clipping, line clipping- Cohen Sutherland line clipping algorithm, midpoint subdivision algorithm, Area clipping, Sutherland and Hodgeman polygon clipping algorithm, text clipping, Window to view port transformation, Blanking.

**Chapter 4: 3D-Graphics and Segments** **12Hrs**

3D-coordinate system, 3D displays technique-Parallel Projection, Perspective Projection, Intensity Cueing. 3D Transformations-Translation, Scaling, Rotation, Reflection, Shearing, polygon surfaces, polygon tables, curves, Bezier curves, Octree, Fractals, Hidden line and surface removal algorithm-Depth buffer, Back-face and scan-line. Constructive Solid Geometry method- Union, Intersection, Difference. Sweep representation.
Introduction to segments, functions for segmenting, display file, segment attributes, display file compilation.

**Chapter 5: Graphical Input devices and Input Techniques** **10Hrs**

Input Devices: Keyboard, Mouse, Joystick, Touch panels, Track ball, Light pen, Graphic tablets.
Predefined Graphics functions.
Positioning techniques, Grid, Constraints, Dynamic manipulation, Gravity field, Rubber band, Dragging, Selection technique, Menu, Pointing and selection by naming.

## Source:

1. **Tutorials point**
2. **Computer graphics by Godse.**
3. **Google…**

**Chapter 1:** **Graphics Systems and Output Primitives**

Computer graphics is an art of drawing pictures, lines, charts, etc using computers with the help of programming. Computer graphics is made up of number of pixels. Pixel is the smallest graphical picture or unit represented on the computer screen. Basically there are two types of computer graphics namely.

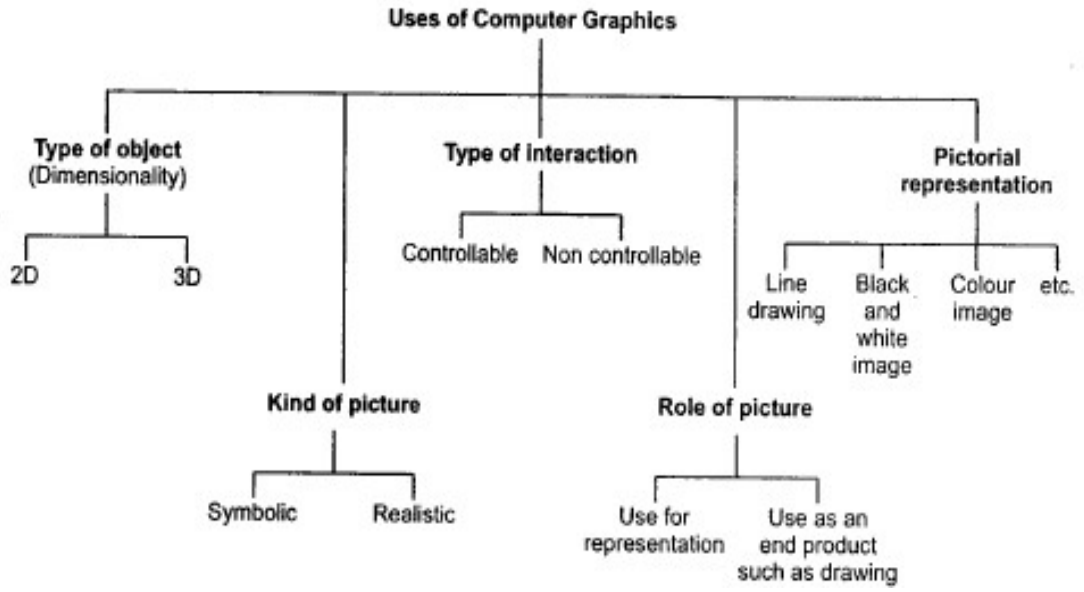
1.**Interactive Computer Graphics:** Interactive Computer Graphics involves a two way communication between computer and user. Here the observer is given some control over the image by providing him with an input device for example the video game controller of the ping pong game. This helps him to signal his request to the computer.

2. **Non Interactive Computer Graphics**: In non interactive computer graphics otherwise known as passive computer graphics. it is the computer graphics in which user does not have any kind of control over the image. Image is merely the product of static stored program and will work according to the instructions given in the program linearly. The image is totally under the control of program instructions not under the user. Example: screen savers.

## Application of Computer Graphics

1. Education and Training: Computer-generated model of the physical, financial and economic system is often used as educational aids. Model of physical systems, physiological system, population trends or equipment can help trainees to understand the operation of the system.
2. Use in Biology: Molecular biologist can display a picture of molecules and gain insight into their structure with the help of computer graphics.
3. Computer-Generated Maps: Town planners and transportation engineers can use computer-generated maps which display data useful to them in their planning work.
4. Architect: Architect can explore an alternative solution to design problems at an interactive graphics terminal. In this way, they can test many more solutions that would not be possible without the computer.
5. Presentation Graphics: Example of presentation Graphics are bar charts, line graphs, pie charts and other displays showing relationships between multiple parameters. Presentation Graphics is commonly used to summarize
   - Financial Reports
   - Statistical Reports
   - Mathematical Reports
   - Scientific Reports
   - Economic Data for research reports
   - Managerial Reports
   - Consumer Information Bulletins
   - And other types of reports
6. Computer Art: Computer Graphics are also used in the field of commercial arts. It is used to generate television and advertising commercial.
7. Entertainment: Computer Graphics are now commonly used in making motion pictures, music videos and television shows.
8. Visualization: It is used for visualization of scientists, engineers, medical personnel, business analysts for the study of a large amount of information.
9. Educational Software: Computer Graphics is used in the development of educational software for making computer-aided instruction.
10. Printing Technology: Computer Graphics is used for printing technology and textile design.

## Classification of CG




**Graphics softwares:**

1. LOGO
2. COREL DRAW
3. AUTO CAD
4. 3D STUDIO
5. CORE
6. GKS (Graphics Kernel System)
7. PHIGS
8. CAM (Computer Graphics Metafile)
9. CGI (Computer Graphics Interface)

## Cathode Ray Tube (CRT)

**Definition**: The CRT is a display screen which produces images in the form of the video signal. It is a type of vacuum tube which displays images when the electron beam through electron guns are strikes on the phosphorescent surface. In other Words, the CRT generates the beams, accelerates it at high velocity and deflect it for creating the images on the phosphorous screen so that the beam becomes visible.

A CRT is an evacuated glass tube. An electron gun at the rear of the tube produces a beam of electrons which is directed towards the front of the tube (screen). The inner side of the screen is coated with phosphor substance which gives off light when it is stroked by electrons. This is illustrated in Fig. 1.14. It is possible to control the point at which the electron beam strikes the screen, and therefore the position of the dot upon the screen, by deflecting the electron beam. The Fig. 1.15 shows the electrostatic deflection of the electron beam in a CRT.

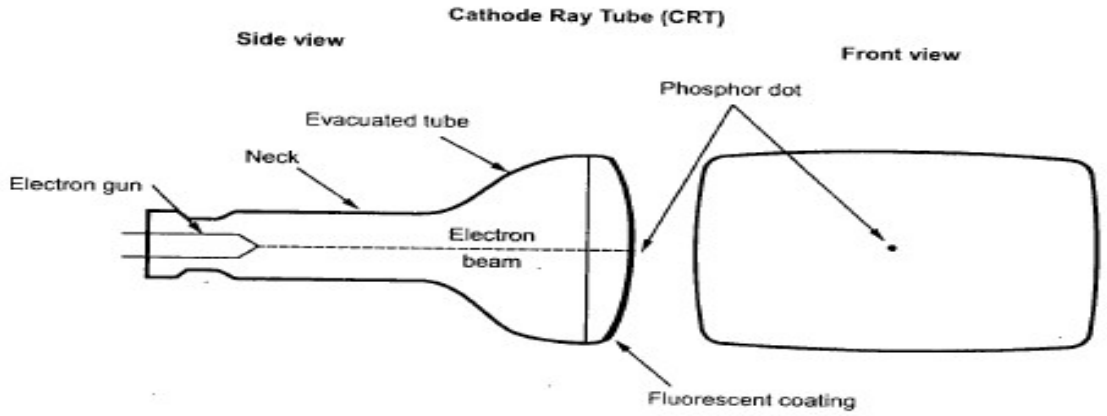



**Fig. 1.14 Simplified representation of CRT**

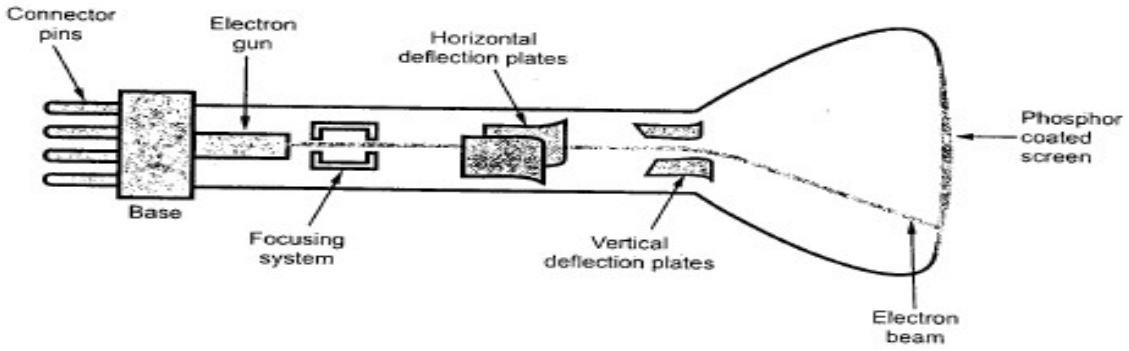



**Fig. 1.15 Cathode Ray Tube**

The deflection system of the cathode-ray-tube consists of two pairs of parallel plates, referred to as the vertical and horizontal deflection plates. The voltage applied to vertical plates controls the vertical deflection of the electron beam and voltage applied to the horizontal deflection plates controls the horizontal deflection of the electron beam. There are two techniques used for producing images on the CRT screen : Vector scan/random scan and Raster scan.

**Random Scan (Vector 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, orcalligraphic 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.

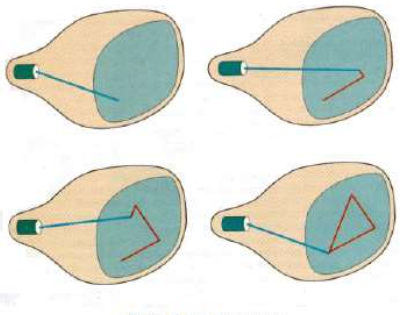


Figure: Random Scan

### 1.7.1.2 Vector Scan/Random Scan Display

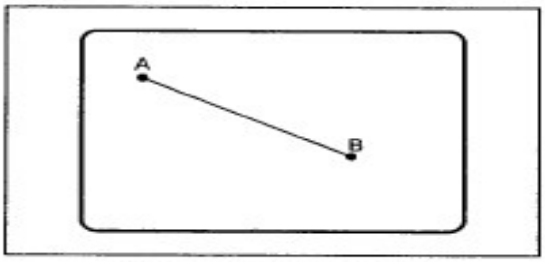



Fig. 1.16 Vector scan CRT

As shown in Fig. 1.16, vector scan CRT display directly traces out only the desired lines on CRT i.e. If we want a line connecting point A with point B on the vector graphics display, we simply drive the beam deflection circuitry, which will cause beam to go directly from point A to B. If we want to move the beam from point A to point B without showing a line between points, we can blank the beam as we move it. To move the beam across the CRT, the information about both, magnitude and direction is required. This information is generated with the help of vector graphics generator.

The Fig. 1.17 shows the typical vector display architecture. It consists of display controller, Central Processing Unit (CPU), display buffer memory and a CRT. A display controller is connected as an I/O peripheral to the central processing unit (CPU). The display buffer memory stores the computer produced display list or display program. The program contains point and line plotting commands with (x, y) or (x, y, z) end point coordinates, as well as character plotting commands. The display controller interprets commands for plotting points, lines and characters and sends digital and point coordinates to a vector generator. The vector generator then converts the digital coordinate values to analog voltages for beam-deflection circuits that displace an electron beam writing on the CRT's phosphor coating.

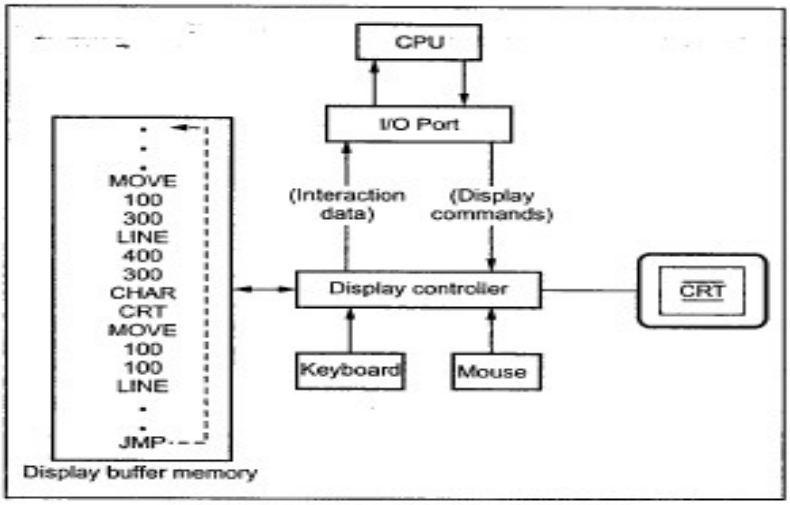
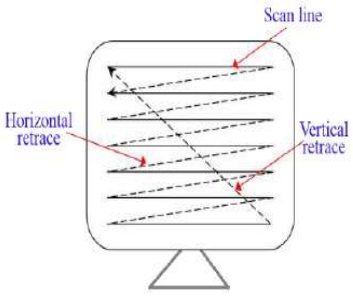



Fig. 1.17 Architecture of a vector display

In vector displays beam is deflected from end point to end point, hence this technique is also called **random scan**. We know as beam, strikes phosphor it emits light. But phosphor light decays after few milliseconds and therefore it is necessary to repeat through the display list to refresh the phosphor at least 30 times per second to avoid flicker. As display buffer is used to store display list and it is used for refreshing, the display buffer memory is also called **refresh buffer**.

## Raster Scan

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.

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

### 1.7.1.3 Raster Scan Display

The Fig. 1.18 shows the architecture of a raster display. It consists of display controller, central processing unit (CPU), video controller, refresh buffer, keyboard, mouse and the CRT.

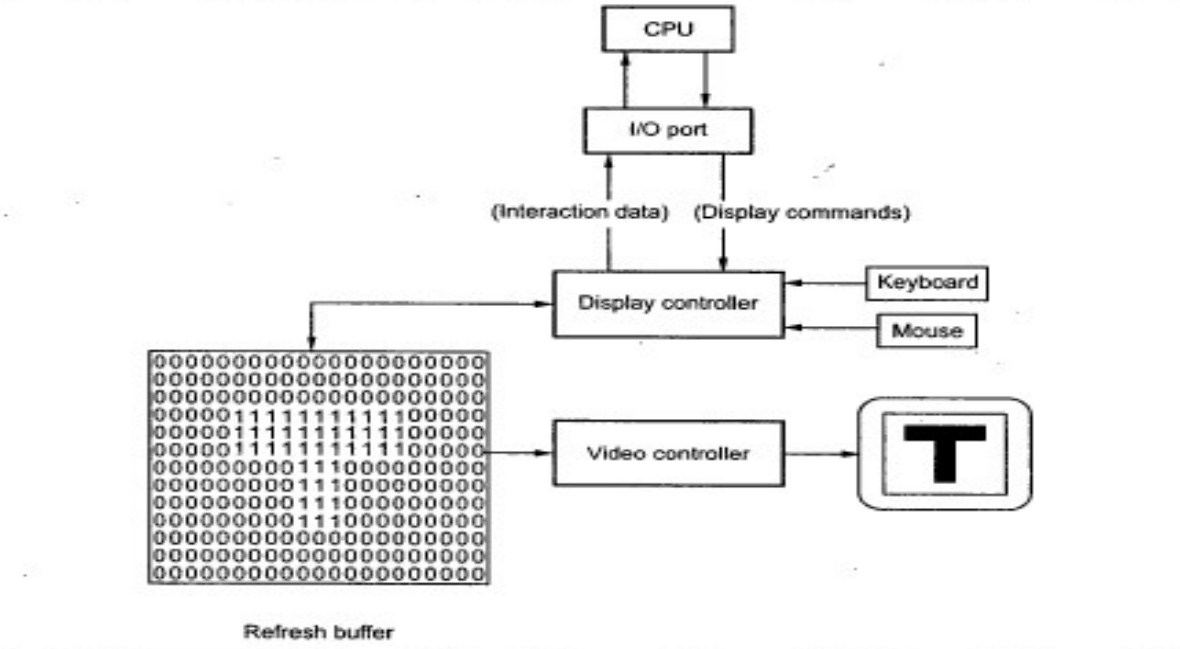



**Fig. 1.18 Architecture of a raster display**

As shown in the Fig. 1.18, the display image is stored in the form of 1s and 0s in the refresh buffer. The video controller reads this refresh buffer and produces the actual image on the screen. It does this by scanning one scan line at a time, from top to bottom and then back to the top, as shown in the Fig. 1.18.

Raster scan is the most common method of displaying images on the CRT screen. In this method, the horizontal and vertical deflection signals are generated to move the beam all over the screen in a pattern shown in the Fig. 1.19

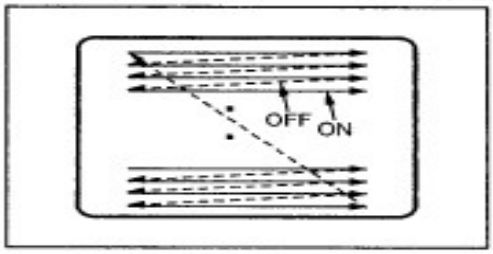



**Fig. 1.19 Raster scan CRT**

Here, the beam is swept back and forth from the left to the right across the screen. When the beam is moved from the left to the right, it is ON. The beam is OFF, when it is moved from the right to the left as shown by dotted line in Fig. 1.19

When the beam reaches the bottom of the screen, it is made OFF and rapidly retraced back to the top left to start again. A display produced in this way is called **raster scan display**. Raster scanning process is similar to reading different lines on the page of a book. After completion of scanning of one line, the electron beam flies back to the start of next line and process repeats. In the raster scan display, the screen image is maintained by repeatedly scanning the same image. This process is known as **refreshing of** screen.

In raster scan displays a special area of memory is dedicated to graphics only. This memory area is called **frame buffer**. It holds the set of intensity values for all the screen points. The stored intensity values are retrieved from frame buffer and displayed on the screen one row (scan line) at a time. Each screen point is referred to as a **pixel** or **pel** (shortened forms of picture element). Each pixel on the screen can be specified by its row and column number. Thus by specifying row and column number we can specify the **pixel position** on the screen.

| Random Scan | Raster Scan |
|---|---|
| 1. It has high Resolution | 1. Its resolution is low. |
| 2. It is more expensive | 2. It is less expensive |
| 3. Any modification if needed is easy | 3.Modification is tough |
| 4. Solid pattern is tough to fill | 4.Solid pattern is easy to fill |
| 5. Refresh rate depends or resolution | 5. Refresh rate does not depend on the picture. |
| 6. Only screen with view on an area is displayed. | 6. Whole screen is scanned. |
| 7. Beam Penetration technology come under it. | 7. Shadow mark technology came under this. |
| 8. It does not use interlacing method. | 8. It uses interlacing |
| 9. It is restricted to line drawing applications | 9. It is suitable for realistic display. |

### Shadow Mask Technique

The shadow mask technique produces a much wider range of colours than the beam penetration technique. Hence this technique is commonly used in raster-scan displays including colour TV. In a shadow mask technique, CRT has three phosphor colour dots at each pixel position. One phosphor dot emits a red light, another emits a green light, and the third emits a blue light. The Fig. 1.28 shows the shadow mask CRT. It has three electron guns, one for each colour dot, and a shadow mask grid just behind the phosphor coated screen.

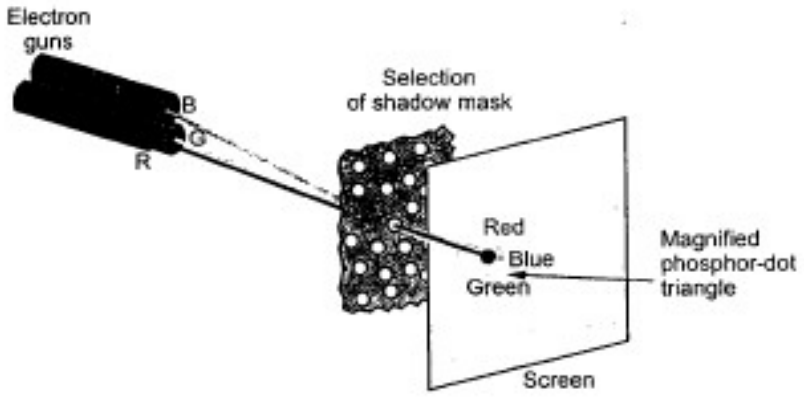



**Fig. 1.28**

The shadow mask grid consists of series of holes aligned with the phosphor dot pattern. As shown in the Fig. 1.28, three electron beams are deflected and focused as a group onto the shadow mask and when they pass through a hole in the shadow mask, they excite a dot triangle. A dot triangle consists of three small phosphor dots of red, green and blue colour. These phosphor dots are arranged so that each electron beam can activate only its corresponding colour dot when it passes through the shadow mask. A dot triangle when activated appears as a small dot on the screen which has colour of combination of three small dots in the dot triangle. By varying the intensity of the three electron beams we can obtain different colours in the shadow mask CRT.

## Display processor with raster system

In the raster scan display systems, the purpose of display controller is to free the CPU from the graphics routine task. Here, display controller is provided with separate memory area as shown in the Fig. 1.26. The main task of display controller is to digitize a picture definition given in an application program into a set of pixel-intensity values for storage in the frame buffer. This digitization process is known as **scan conversion**.

Display controller are also designed to perform a number of additional operations. These operations include

- Generating various line styles (dashed, dotted, or solid)
- Display colour areas
- Performing certain transformations and
- Manipulations on displayed objects

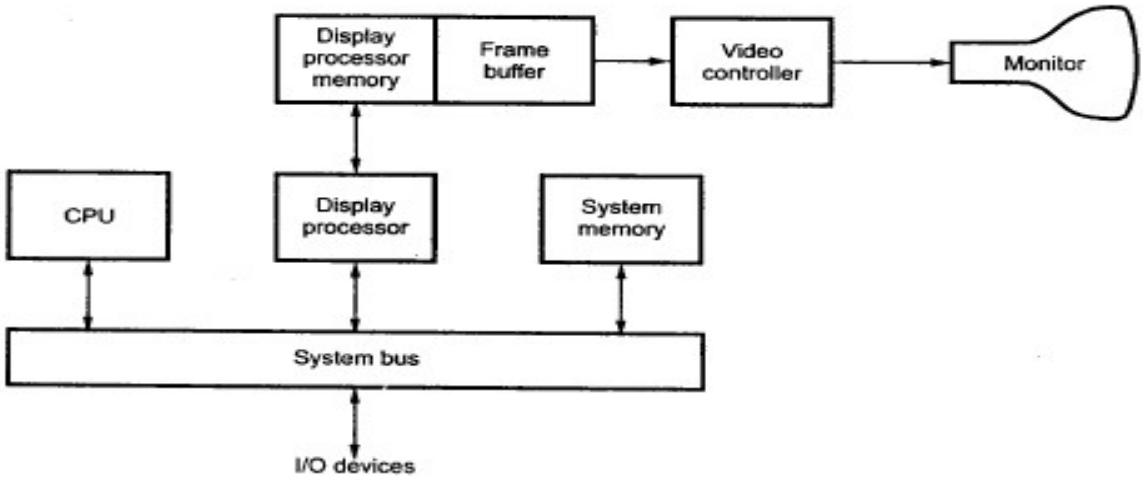



**Fig. 1.27 Raster scan system with a display processor**

## Display processor with random system

In some graphics systems a separate computer is used to interpret the commands in the display file. Such computer is known as display controller. Display controller access display file and it cycles through each command in the display file once during every refresh cycle Fig. 1.26 shows the vector scan system with display controller.

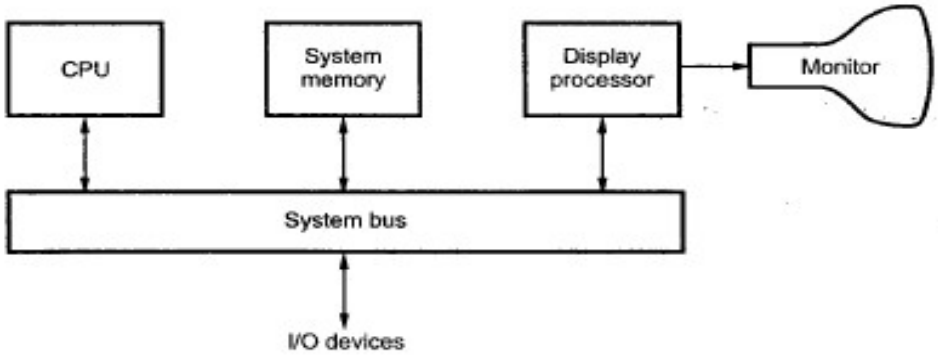
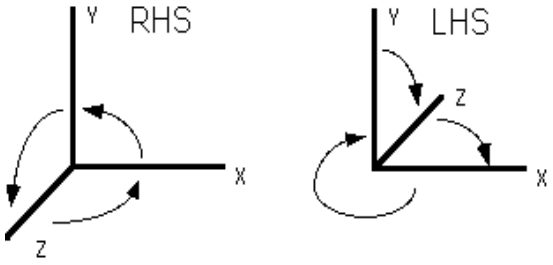



**Fig. 1.26 Vector scan system**

## Raster co-ordinate system

In a 2-D coordinate system the X axis generally points from left to right, and the Y axis generally points from bottom to top. When we add the third coordinate, Z, we have a choice as to whether the Z-axis points into the screen or out of the screen:




**Right Hand Coordinate System (RHS)**

Z is coming out of the page

Counter clockwise rotations are positive
if we rotate about the X axis : the rotation Y->Z is positive
if we rotate about the Y axis : the rotation Z->X is positive
if we rotate about the Z axis : the rotation X->Y is positive

**Left Hand Coordinate System (LHS)**

Z is going into the page

Clockwise rotations are positive
if we rotate about the X axis : the rotation Y->Z is positive
if we rotate about the Y axis : the rotation Z->X is positive
if we rotate about the Z axis : the rotation X->Y is positive

The important thing to note is coordinate system is being used by the package we are handling with, both for the creation of models and the displaying. Also we can note that if the two packages use different coordinate systems, then the model need to be inverted when they are loaded in for viewing.

## Color mapping

A *color map* (often called a *color table* or a *palette*) is an array of colors used to map pixel data (represented as indexes into the color table) to the actual color values. Some graphic displays, especially older ones, cannot show an arbitrary color for every pixel. Instead, each pixel is stored as a small number that's used as an index into the color table.

For example, it was common for VGA, in some modes, to be able to display up to 256 unique colors at one time. But since the screen had many more than 256 pixels, you couldn't set an arbitrary RGB value for each of them. Instead, you'd store a single byte per pixel, which would be used to look up the actual color in the color table.

Color tables are less relevant today, but some graphics file formats still represent as a color table and pixel data the consists of indexes into the table, rather than directly encoding the pixels as colors in some colorspace, like RGB.

## Instruction set

The **instruction set**, also called **ISA** (**instruction set architecture**), is part of a computer that pertains to programming, which is basically machine language. The instruction set provides commands to the processor, to tell it what it needs to do. The instruction set consists of addressing modes, instructions, native data types, registers, memory architecture, interrupt, and exception handling, and external I/O.

An example of an instruction set is the x86 instruction set, which is common to find on computers today. Different computer processors can use almost the same instruction set while still having very different internal design. Both the Intel Pentium and AMD Athlon processors use nearly the same x86 instruction set. An instruction set can be built into the hardware of the processor, or it can be emulated in software, using an interpreter. The hardware design is more efficient and faster for running programs than the emulated software version.

**Examples of instruction set**

- **ADD** - Add two numbers together.
- **COMPARE** - Compare numbers.
- **IN** - Input information from a device, e.g., keyboard.
- **JUMP** - Jump to designated RAM address.
- **JUMP IF** - Conditional statement that jumps to a designated RAM address.
- **LOAD** - Load information from RAM to the CPU.
- **OUT** - Output information to device, e.g., monitor.
- **STORE** - Store information to RAM.

## 1.3 The Advantages of Interactive Graphics

Let us discuss the advantages of interactive graphics.

- Today, a high quality graphics displays of personal computer provide one of the most natural means of communicating with a computer.
- It provides tools for producing pictures not only of concrete, "real-world" objects but also of abstract, synthetic objects, such as mathematical surfaces in 4D and of data that have no inherent geometry, such as survey results.
- It has an ability to show moving pictures, and thus it is possible to produce animations with interactive graphics.
- With interactive graphics use can also control the animation by adjusting the speed, the portion of the total scene in view, the geometric relationship of the objects in the scene to one another, the amount of detail shown and so on.
- The interactive graphics provides tool called **motion dynamics**. With this tool user can move and tumble objects with respect to a stationary observer, or he can make objects stationary and the viewer moving around them. A typical example is walk throughs made by builder to show flat interior and building surroundings. In many case it is also possible to move both objects and viewer.
- The interactive graphics also provides facility called **update dynamics**. With update dynamics it is possible to change the shape, colour or other properties of the objects being viewed.
- With the recent development of digital signal processing (DSP) and audio synthesis chip the interactive graphics can now provide audio feedback alongwith the graphical feedbacks to make the simulated environment even more realistic.

## Output Primitives: Line drawing methods

### Direct line drawing algorithm:

Considering the assumptions made in the previous section most line drawing algorithms use incremental methods. In these methods line starts with the starting point. Then a fix increment is added to the current point to get the next point on the line. This is continued till the end of line. Let us see the incremental algorithm.

**Incremental Algorithm**

1. CurrPosition = Start
   Step = Increment
2. if ( | CurrPosition – End | < Accuracy) then go to step 5
   [ This checks whether the current position is reached upto approximate end point. If yes, line drawing is completed. ]
   if (CurrPosition < End) then go to step 3
   [Here start < End]
   if (CurrPosition > End) then go to step 4
   [Here start > End]
3. CurrPosition = CurrPosition + Step
   go to step 2
4. CurrPosition = CurrPosition – Step
   go to step 2
5. Stop.

**DDA Line Algorithm**

1. Read the line end points $(x_1, y_1)$ and $(x_2, y_2)$ such that they are not equal.
   [ if equal then plot that point and exit]
2. $\Delta x = |x_2 - x_1|$ and $\Delta y = |y_2 - y_1|$
3. if $(\Delta x \geq \Delta y)$ then
   length = $\Delta x$
   else
   length = $\Delta y$
   end if
4. $\Delta x = (x_2 - x_1) / \text{length}$
   $\Delta y = (y_2 - y_1) / \text{length}$
   [This makes either $\Delta x$ or $\Delta y$ equal to 1 because length is either $|x_2 - x_1|$ or $|y_2 - y_1|$. Therefore, the incremental value for either x or y is one.]
5. $x = x_1 + 0.5 * \text{Sign}(\Delta x)$
   $y = y_1 + 0.5 * \text{Sign}(\Delta y)$
   [Here, Sign function makes the algorithm work in all quadrant. It returns – 1, 0, 1 depending on whether its argument is < 0, = 0, > 0 respectively. The factor 0.5 makes it possible to round the values in the integer function rather than truncating them.]
6. i = 1 [Begins the loop, in this loop points are plotted]
   While ( i ≤ length)
   {
   Plot (Integer (x), Integer (y) )
   x = x + $\Delta x$
   y = y + $\Delta y$
   i = i + 1
   }

7.stop

**Bresenham's Line Algorithm**

1. Read the line end points $(x_1, y_1)$ and $(x_2, y_2)$ such that they are not equal.
   [ if equal then plot that point and exit ]
2. $\Delta x = |x_2 - x_1|$ and $\Delta y = |y_2 - y_1|$
3. [Initialize starting point]
   $x = x_1$
   $y = y_1$
4. $e = 2 * \Delta y - \Delta x$
   [Initialize value of decision variable or error to compensate for nonzero intercepts]
5. i = 1 [Initialize counter]
6. Plot (x, y)
7. while ( $e \geq 0$)
   {
   $y = y + 1$
   $e = e - 2 * \Delta x$
   }
   $x = x + 1$
   $e = e + 2 * \Delta y$
8. $i = i + 1$
9. if ( $i \leq \Delta x$) then go to step 6.
10. Stop

## Circle drawing algorithms

## 2.8 Representation of a Circle

There are two standard methods of mathematically representing a circle centered at the origin.

### 2.8.1 Polynomial Method

In this method circle is represented by a polynomial equation.

$$x^2 + y^2 = r^2$$

where x : the x coordinate
y : the y coordinate
r : radius of the circle

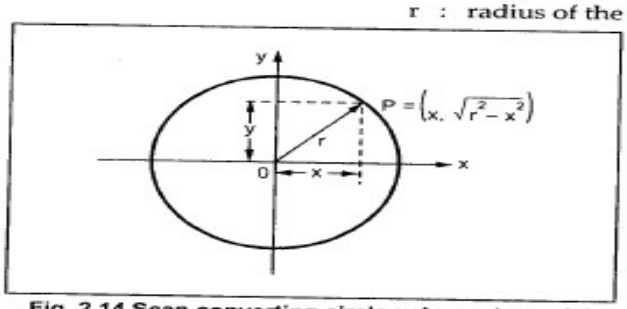



Fig. 2.14 Scan converting circle using polynomial method

Here, polynomial equation can be used to find y coordinate for the known x coordinate. Therefore, the scan converting circle using polynomial method is achieved by stepping x from 0 to $r\sqrt{2}$, and each y coordinate is found by evaluating $\sqrt{r^2 - x^2}$ for each step of x. This generates the 1/8 portion (90° to 45°) of the circle. Remaining part of the circle can be generated just by reflection.

The polynomial method of circle generation is an inefficient method. In this method for each point both x and r must be squared and $x^2$ must be subtracted from $r^2$; then the square root of the result must be found out.

### 2.8.2 Trigonometric Method

In this method, the circle is represented by use of trigonometric functions

$$x = r\cos\theta \text{ and } y = r\sin\theta$$

where $\theta$ : current angle
r : radius of the circle
x : the x coordinate
y : the y coordinate

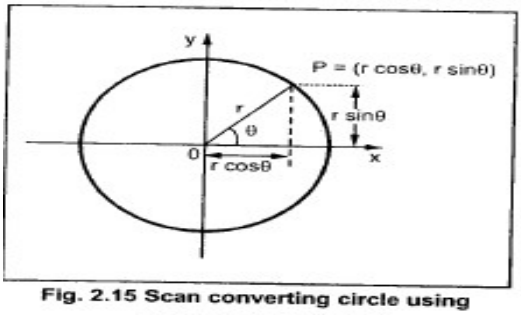



Fig. 2.15 Scan converting circle using trigonometric method

The scan converting circle using trigonometric method is achieved by stepping $\theta$ from 0 to $\pi/4$ radians, and each value of x and y is calculated. However, this method is more inefficient than the polynomial method because the computation of the values of $\sin\theta$ and $\cos\theta$ is even more time-consuming than the calculations required in the polynomial method.

## Mid-point circle drawing algorithm

**Algorithm**

1. Read the radius (r) of the circle
2. Initialize starting position as
   x = 0
   y = r
3. Calculate initial value of decision parameter as
   P = 1.25 − r
4. do
```
{   plot (x, y)
    if (d < 0)
        {   x = x + 1
            y = y
            d = d + 2x + 1
        }
    else
        {   x = x + 1
            y = y − 1
            d = d + 2x + 2y + 1
}
    while (x < y)
```
5. Determine symmetry points
6. Stop.

## Ellipse drawing algorithm

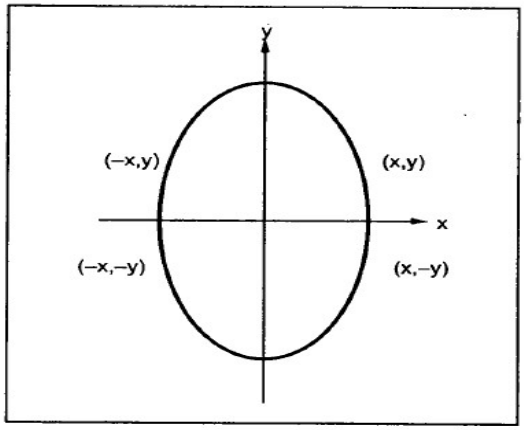



Fig. 2.20 Four way symmetry of ellipse

The midpoint ellipse drawing algorithm uses the four way symmetry of the ellipse to generate it. The Fig. 2.20 shows the four-way symmetry of ellipse. This approach is similar to that used in displaying a raster circle. Here, the quadrant of the ellipse is divided into two regions. The Fig. 2.21 shows the division of the first quadrant according to the slope of an ellipse with $r_x < r_y$. As ellipse is drawn from 90° to 0°, the x moves in the positive direction and y moves in the negative direction, and ellipse passes through two regions. It is important to note that while processing first quadrant we have to take steps in the x direction where the slope of the curve has a magnitude less than 1 (for region 1) and to take steps in the y direction where the slope has a magnitude greater than 1 (for region 2).

## Mid point or Bresenham's ellipse drawing algorithm

**ALGORITHM:**

1. Start
2. Initialize the graphics mode.
3. Get the center point as (x1,y1)
4. Get the length of semi-major, Semi-minor axes as
   r1 & r2
5. calculate t=pi/180
6. Initialize i=0
7. compute d= i*t
8. compute x=x1+y1*sin(d), y=y1+r2cos(d)
9. plot(x,y)
10. increment i by 1
11. Repeat steps 6 to 9 until i<360

## Area filling

Polygon is an ordered list of vertices as shown in the following figure. For filling polygons with particular colors, you need to determine the pixels falling on the border of the polygon and those which fall inside the polygon.

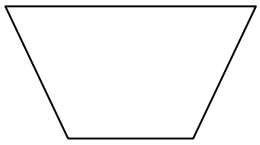
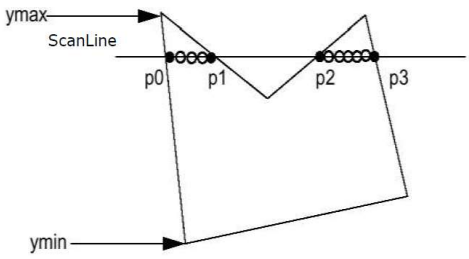


**Figure: A Polygon**

## Scan Line Algorithm

This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections. The following steps depict how this algorithm works.

**Step 1**: Find out the Ymin and Ymax from the given polygon.




**Step 2**: ScanLine intersects with each edge of the polygon from Ymin to Ymax. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

**Step 3**: Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).

**Step 4**: Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

## Character attributes

**Character Attributes:**

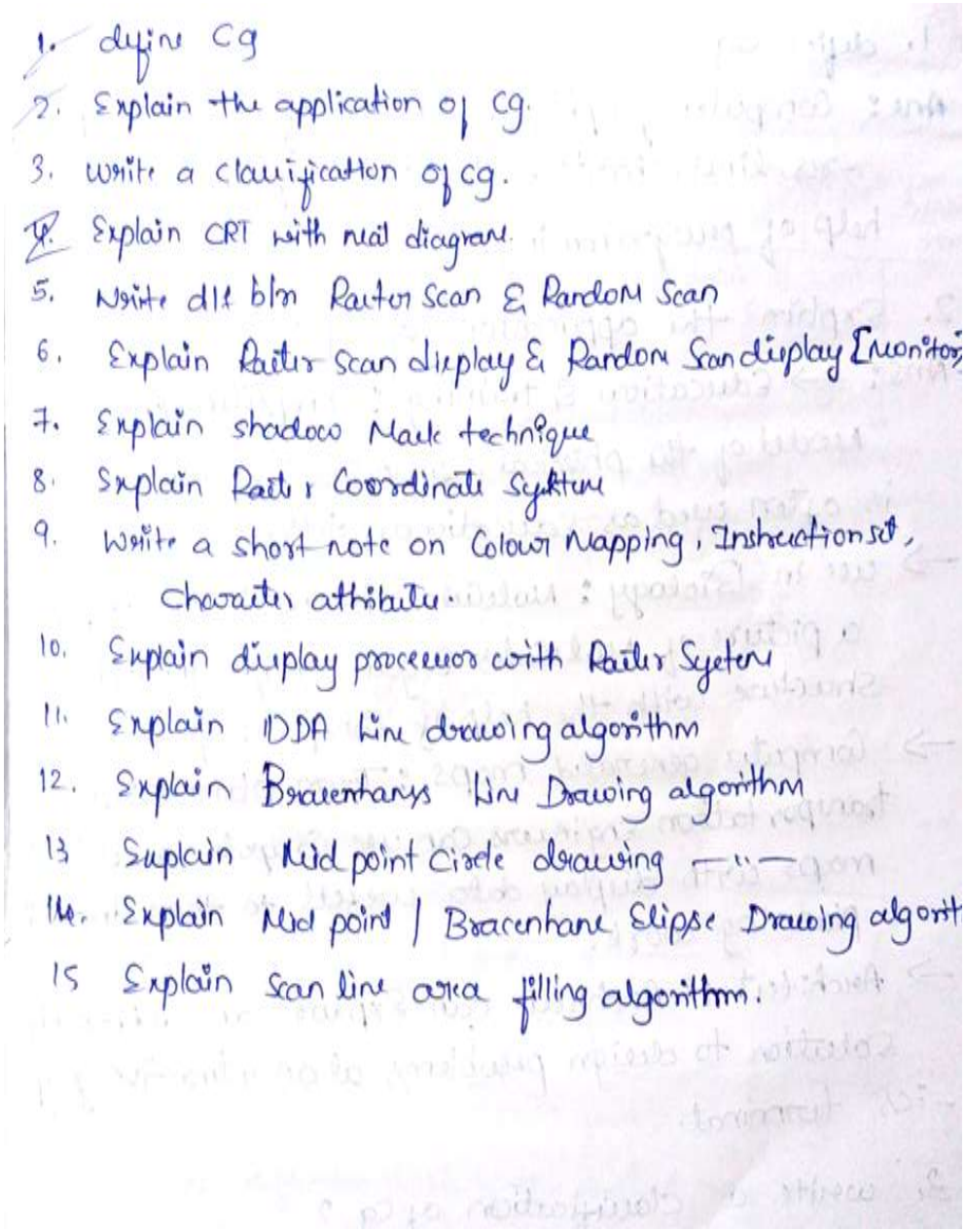
The appearance of displayed characters is controlled by attributes such as font, size, colour and orientation. Attributes can be set both for entire character strings and for individual characters, known as Marker symbols.

1. **Text Attributes:** There are many text options available, such as font, colour, size, spacing, and orientation.
2. **Text Style:** The characters in a selected font can also be displayed in various underlining styles (solid, dotted, dashed, double), in **bold**, in italics, shadow style, etc. Font options can be made available as predefined sets of grid patterns or as character sets designed with lines and curves.
3. **Text Colour:** Colour settings for displayed text are stored in the system attribute list and transferred to the frame buffer by character loading functions. When a character string is displayed, the current colour is used to set pixel values in the frame buffer corresponding to the character shapes and position.
4. **Text Size:** We can adjust text size by changing the overall dimensions, i.e., width and height, of characters or by changing only the width. Character size is specified in **Points**, where 1 point is 0.013837 inch or approximately 1/72 inch. Point measurements specify the size of the **Character**

**Body**. Different fonts with the same point specifications can have different character sizes depending upon the design of the font.

5. **Text Orientation:** The text can be displayed at various angles, known as orientation. A procedure for orienting text rotates characters so that the sides of character bodies, from baseline to topline at aligned at some angle. Character strings can be arranged vertically or horizontally.
6. **Text Path:** In some applications the character strings are arranged vertically or horizontally. This is known as Text Path. Text path can be right, left, up or down.
7. **Text Alignment:** Another attribute for character strings is alignment. This attribute specifies how text is to be positioned with respect to the start coordinates. Vertical alignment can be top, cap, half, base and bottom. Similarly, horizontal alignment can be left, centre and right.

## Questions


1. dfine cg
2. Explain the application of cg.
3. Write a classification of cg.
4. Explain CRT with neat diagram.
5. Write diff b/w Raster Scan & Random Scan
6. Explain Raster Scan display & Random Scan display Monitor
7. Explain shadow Mask technique
8. Explain Raster Coordinate System
9. Write a short note on Colour Mapping, Instruction set, Character attribute.
10. Explain display processor with Raster System
11. Explain DDA Line drawing algorithm
12. Explain Bresenhams Line Drawing algorithm
13. Explain Mid point Circle drawing —"—
14. Explain Mid point / Bresenham Ellipse Drawing algorithm
15. Explain Scan line area filling algorithm.

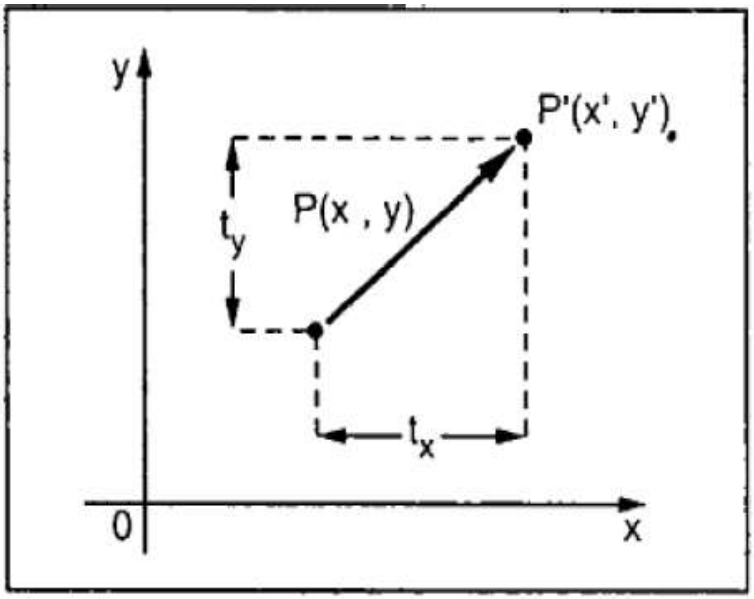
**Chapter 2:** **Two-dimensional Transformation**

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.

**Translation:**

A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate ($t_x$, $t_y$) to the original coordinate (X, Y) to get the new coordinate (X', Y').




From the above figure, you can write that:

$$X' = X + t_x$$

$$Y' = Y + t_y$$

The pair ($t_x$, $t_y$) is called the translation vector or shift vector. The above equations can also be represented using the column vectors.

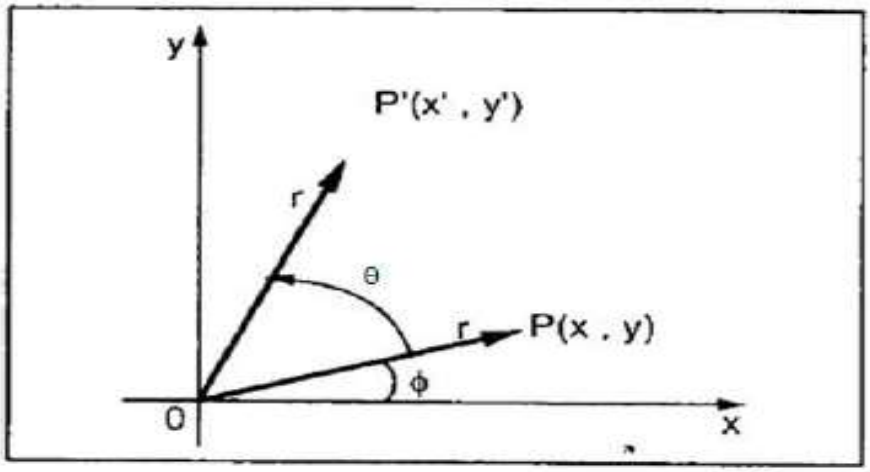
$$P = \begin{bmatrix} X \\ Y \end{bmatrix} \quad P' = \begin{bmatrix} X' \\ Y' \end{bmatrix} \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

We can write it as:

$$P' = P + T$$

**Rotation with arbitrary point**

In rotation, we rotate the object at particular angle θ (theta) from its origin. From the following figure, we can see that the point P(X, Y) is located at angle Φ from the horizontal X coordinate with distance r from the origin. If we want to rotate it at the angle θ. After rotating it to a new location, you will get a new point P' (X', Y').




Using standard trigonometric the original coordinate of point P(X, Y) can be represented as:

$$X = r \cos \Phi \quad \ldots\ldots (1)$$

$$Y = r \sin \Phi \quad \ldots\ldots (2)$$

Same way we can represent the point P' (X', Y') as:

$$X' = r \cos (\Phi + \theta) = r \cos \Phi \cos \theta - r \sin \Phi \sin \theta \quad \ldots\ldots (3)$$

$$Y' = r \sin (\Phi + \theta) = r \cos \Phi \sin \theta + r \sin \Phi \cos \theta \quad \ldots\ldots (4)$$

Substituting equation (1) & (2) in (3) & (4) respectively, we will get

$$X' = X \cos \theta - Y \sin \theta$$

$$Y' = X \sin \theta + Y \cos \theta$$

Representing the above equation in matrix form,

$$[X' \quad Y'] = [X \quad Y] \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

OR

$$P' = P \cdot R$$

Where R is the rotation matrix

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

The rotation angle can be positive and negative.

For positive rotation angle, we can use the above rotation matrix. However, for negative angle rotation, the matrix will change as shown below:

$$R = \begin{bmatrix} \cos(-\theta) & \sin(-\theta) \\ -\sin(-\theta) & \cos(-\theta) \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \qquad (\because \cos(-\theta) = \cos\theta \text{ and } \sin(-\theta) = -\sin\theta)$$

## Scaling with the fixed point

To change the size of an object, scaling transformation is used. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.

Let us assume that the original coordinates are (X, Y), the scaling factors are ($S_X$, $S_Y$), and the produced coordinates are (X', Y'). This can be mathematically represented as shown below:

$$X' = X \cdot S_X \qquad \text{and} \qquad Y' = Y \cdot S_Y$$

The scaling factor $S_X$, $S_Y$ scales the object in X and Y direction respectively. The above equations can also be represented in matrix form as below:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} X \\ Y \end{bmatrix} \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

OR

$$P' = P \cdot S$$

Where S is the scaling matrix. The scaling process is shown in the following figure.

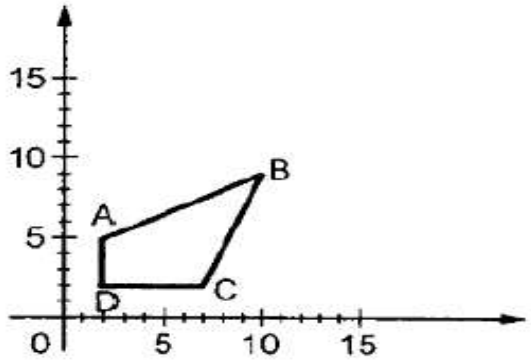



**Figure: Before scaling process**

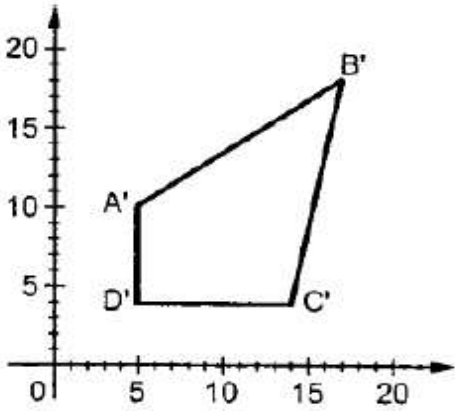



**Figure: After Scaling Process**

If we provide values less than 1 to the scaling factor S, then we can reduce the size of the object. If we provide values greater than 1, then we can increase the size of the object.

## Reflection

Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180°. In reflection transformation, the size of the object does not change.

The following figures show reflections with respect to X and Y axes, and about the origin respectively.

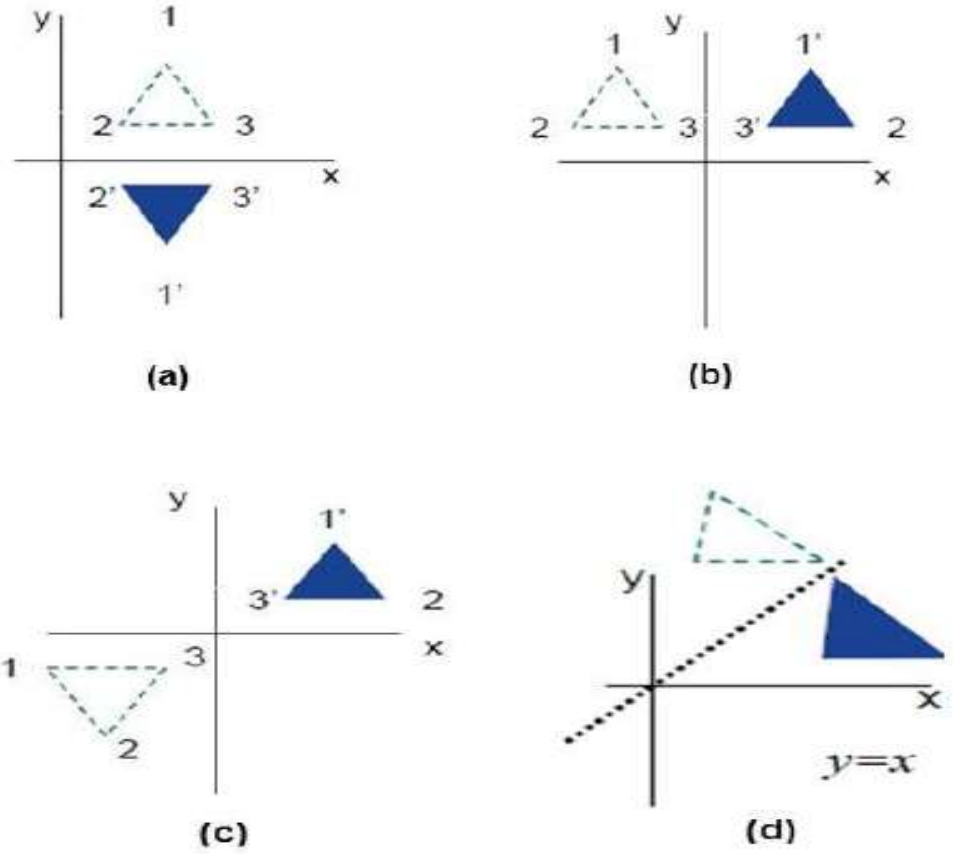


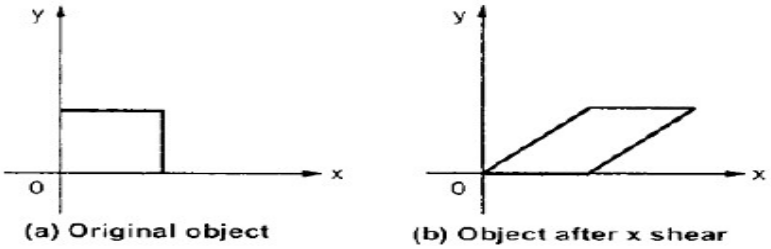

**Figure: Reflection about line y=x**

## Shear

A transformation that slants the shape of an object is called the shear transformation. There are two shear transformations **X-Shear** and **Y-Shear**. One shifts X coordinates values and other shifts Y coordinate values. However, in both the cases, only one coordinate changes its coordinates and other preserves its values. Shearing is also termed as **Skewing**.

**X-Shear**

The X-Shear preserves the Y coordinate and changes are made to X coordinates, which causes the vertical lines to tilt right or left as shown in below figure.



The transformation matrix for X-Shear can be represented as:

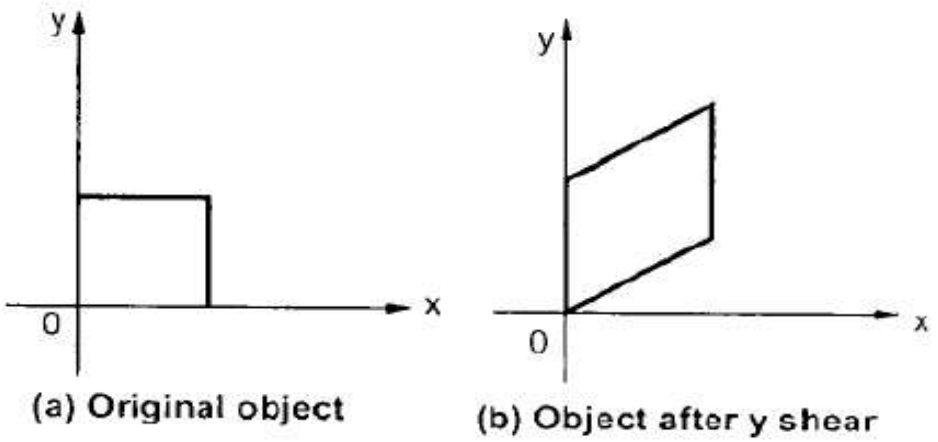
$$X_{Sh} = \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$X' = X + Sh_x \cdot Y$$

$$Y' = Y$$

## Y-Shear

The Y-Shear preserves the X coordinates and changes the Y coordinates which causes the horizontal lines to transform into lines which slopes up or down as shown in the following figure.




(a) Original object (b) Object after y shear

The Y-Shear can be represented in matrix from as:

$$Y_{Sh} = \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Y' = Y + Sh_y \cdot X$$

$$X' = X$$

## Matrix representation

- Translation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Rotation:

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

- Scaling:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

**Homogenous Coordinates**

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

1. Translate the coordinates,
2. Rotate the translated coordinates, and then
3. 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 coordinate W. By this we can represent the point by 3 numbers instead of 2 numbers called as **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).

**Composite Transformation**

If a transformation of the plane T1 is followed by a second plane transformation T2, then the result itself may be represented by a single transformation T which is the composition of T1 and T2 taken in that order. This is written as T = T1·T2.

Composite transformation can be achieved by concatenation of transformation matrices to obtain a combined transformation matrix.

A combined matrix:

[T][X] = [X] [T1] [T2] [T3] [T4] …. [Tn]

Where [Ti] is any combination of

1. Translation
2. Scaling
3. Shearing
4. Rotation
5. Reflection

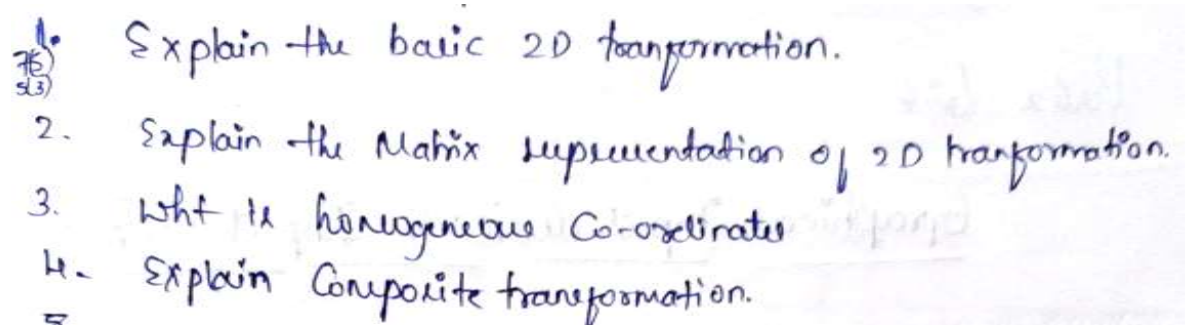
The change in the order of transformation would lead to different results, as in general matrix multiplication is not cumulative, that is [A] · [B] ≠ [B] · [A] and the order of multiplication. The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformation, one after another.

For example, to rotate an object about an arbitrary point (Xp, Yp),

we have to carry out three steps:

1. Translate point (Xp, Yp) to the origin.
2. Rotate it about the origin.
3. Finally, translate the center of rotation back where it belonged.

**Questions:**



## Chapter 3: Windowing and Clipping

### Window:

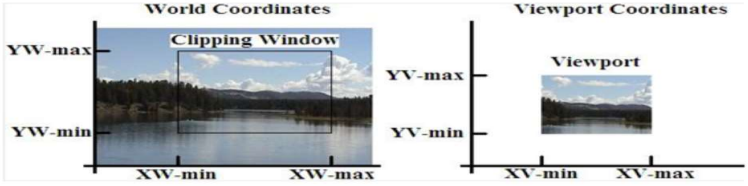
1. A world-coordinate area selected for display is called a window.
2. In computer graphics, a window is a graphical control element.
3. It consists of a visual area containing some of the graphical user interface of the program it belongs to and is framed by a window decoration.
4. A window defines a rectangular area in world coordinates. You define a window with a GWINDOW statement. You can define the window to be larger than, the same size as, or smaller than the actual range of data values, depending on whether you want to show all of the data or only part of the data.

### Viewport:

1. An area on a display device to which a window is mapped is called a viewport.
2. A viewport is a polygon viewing region in computer graphics. The viewport is an area expressed in rendering-device-specific coordinates, e.g. pixels for screen coordinates, in which the objects of interest are going to be rendered.
3. A viewport defines in normalized coordinates a rectangular area on the display device where the image of the data appears. You define a viewport with the GPORT command. You can have your graph take up the entire display device or show it in only a portion, say the upper-right part.

### Window to viewport transformation:

1. Window-to-Viewport transformation is the process of transforming a two-dimensional, world-coordinate scene to device coordinates.
2. In particular, objects inside the world or clipping window are mapped to the viewport. The viewport is displayed in the interface window on the screen.
3. In other words, the clipping window is used to select the part of the scene that is to be displayed. The viewport then positions the scene on the output device.
4. **Example:**



## 5.2 Viewing Transformation

We know that the picture is stored in the computer memory using any convenient cartesian coordinate system, referred to as **world coordinate system** (WCS). However, when picture is displayed on the display device it is measured in **physical device coordinate system** (PDCS) corresponding to the display device. Therefore, displaying an image of a picture involves mapping the coordinates of the points and lines that form the picture into the appropriate physical device coordinate where the image is to be displayed. This mapping of coordinates is achieved with the use of coordinate transformation known as **viewing transformation**.

The viewing transformation which maps picture coordinates in the WCS to display coordinates in PDCS is performed by the following transformations :

- Normalization transformation (N) and
- Workstation transformation (W)

### Normalisation transformation

The interpreter uses a simple linear formula to convert the normalized device coordinates to the actual device coordinates.

$$x = x_n \times X_W \quad ..$$

$$y = y_n \times Y_H \quad ..$$

where

x : Actual device x coordinate

y : Actual device y coordinate

$x_n$ : Normalized x coordinate

$y_n$ : Normalized y coordinate

$X_W$ : Width of actual screen in pixels

$Y_H$ : Height of actual screen in pixels.

The transformation which maps the world coordinate to normalized device coordinate is called **normalization transformation**. It involves scaling of x and y, thus it is also referred to as **scaling transformation**.

### Workstation Transformation

The transformation which maps the normalized device coordinates to physical device coordinates is called workstation transformation.

The viewing transformation is the combination of normalization transformation and workstation transformations as shown in the Fig. 5.4. It is given as

$$V = N \cdot W$$

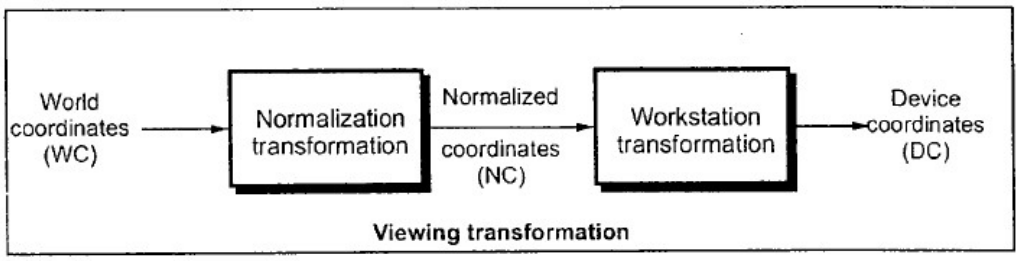



Fig. 5.4 Two dimensional viewing transformation

## Clipping process

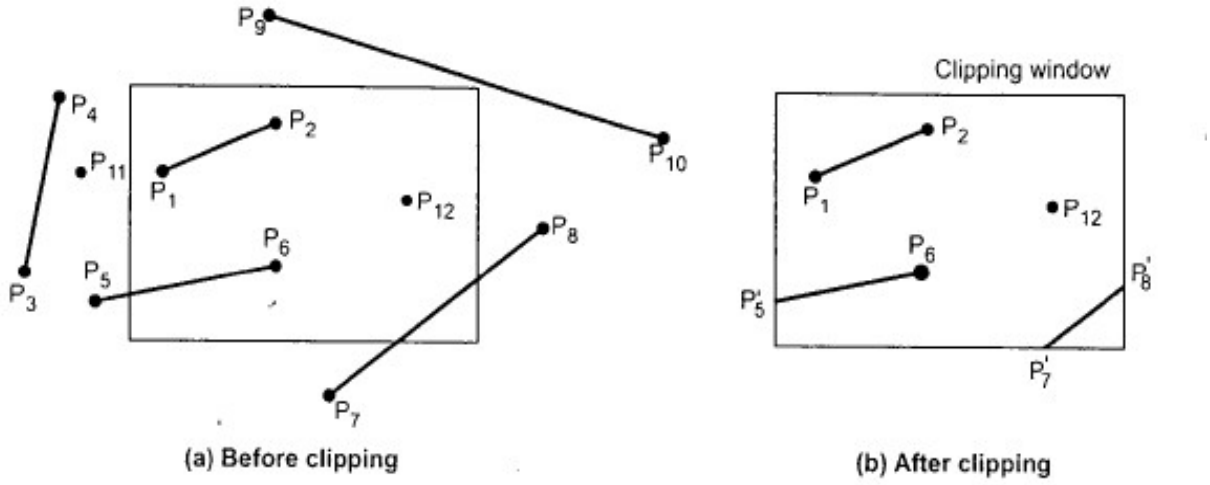



Fig. 5.8

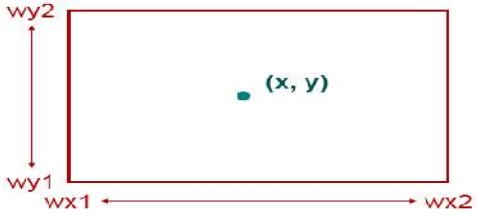
The procedure that identifies the portions of a picture that are either inside or outside of a specified region of space is referred to as clipping. The region against which an object is to be clipped is called a **clip window** or **clipping window.** It usually is in a rectangular shape, as shown in the Fig. 5.8.

The clipping algorithm determines which points, lines or portions of lines lie within the clipping window. These points, lines or portions of lines are retained for display. All others are discarded.

### Point Clipping

Clipping a point from a given window is very easy. Consider the following figure, where the rectangle indicates the window. Point clipping tells us whether the given point (X, Y) is within the given window or not; and decides whether we will use the minimum and maximum coordinates of the window.

The X-coordinate of the given point is inside the window, if X lies in between Wx1 ≤ X ≤ Wx2. Same way, Y coordinate of the given point is inside the window, if Y lies in between Wy1 ≤ Y ≤ Wy2.

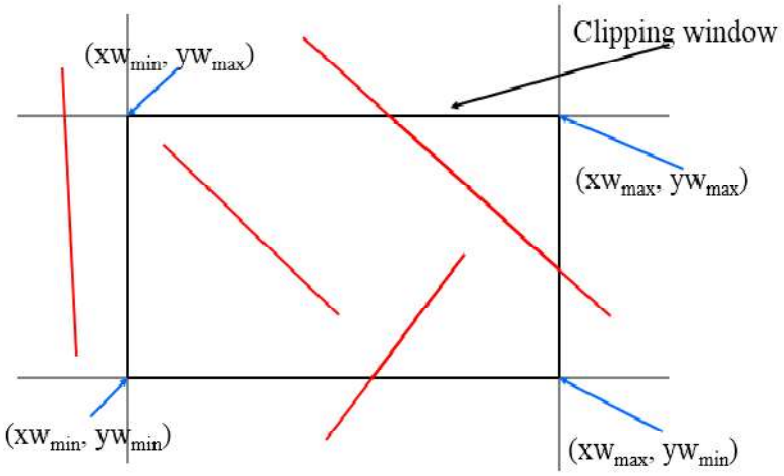



### Line Clipping

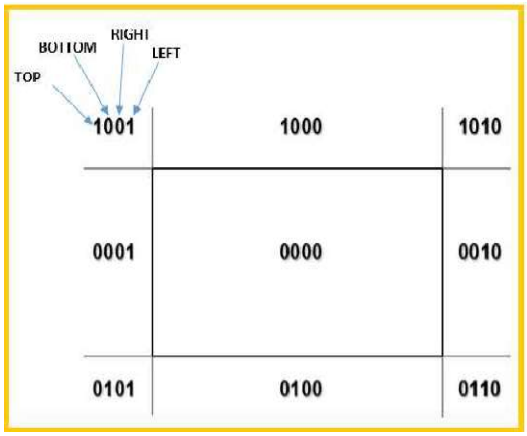
The concept of line clipping is same as point clipping. In line clipping, we will cut the portion of line which is outside of window and keep only the portion that is inside the window.

## Cohen-Sutherland Line Clipping algorithm

This algorithm uses the clipping window as shown in the following figure. The minimum coordinate for the clipping region is (XWmin, YWmin) and the maximum coordinate for the clipping region is (XWmax, YWmax).




We will use 4-bits to divide the entire region. These 4 bits represent the Top, Bottom, Right, and Left of the region as shown in the following figure. Here, the **TOP** and **LEFT** bit is set to 1 because it is the **TOP-LEFT** corner.




There are 3 possibilities for the line:
1. Line can be completely inside the window (This line should be accepted).
2. Line can be completely outside of the window (This line will be completely removed from the region).
3. Line can be partially inside the window (We will find intersection point and draw only that portion of line that is inside region).

**Algorithm**

**Step 1**: Assign a region code for each endpoints.
**Step 2**: If both endpoints have a region code **0000** then accept this line.
**Step 3**: Else, perform the logical **AND** operation for both region codes.
- **Step 3.1**: If the result is not **0000**, then reject the line.
- **Step 3.2**: Else you need clipping.
  - **Step 3.2.1**: Choose an endpoint of the line that is outside the window.
  - **Step 3.2.2**: Find the intersection point at the window boundary (base on region code).
  - **Step 3.2.3**: Replace endpoint with the intersection point and update the region code.
  - **Step 3.2.4**: Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.

**Step-4**: Repeat step 1 for other lines.

## Midpoint subdivision algorithm

Like previous algorithm, initially the line is tested for visibility. If line is completely visible it is drawn and if it is completely invisible it is rejected. If line is partially visible then it is subdivided in two equal parts. The visibility tests are then applied to each half. This subdivision process is repeated until we get completely visible and completely invisible line segments. This is illustrated in Fig. 5.14. (see on next page)

As shown in the Fig. 5.14, line $P_1 P_2$ is partially visible. It is subdivided in two equal parts $P_1 P_3$ and $P_3 P_2$ (see Fig. 5.14(b)). Both the line segments are tested for visibility and found to be partially visible. Both line segments are then subdivided in two equal parts to get midpoints $P_4$ and $P_5$ (see Fig. 5.14 (c)). It is observed that line segments $P_1P_4$ and $P_5P_2$ are completely invisible and hence rejected. However, line segment $P_3P_5$ is completely visible and hence drawn. The remaining line segment $P_4P_3$ is still partially visible. It is then subdivided to get midpoint $P_6$. It is observed that $P_6P_3$ is completely visible whereas $P_4P_6$ is partially visible. Thus $P_6P_3$ line segment is drawn and $P_4P_6$ line segment is further subdivided into equal parts to get midpoint $P_7$. Now, it is observed that line segment $P_4P_7$ is completely invisible and line segment $P_7P_6$ is completely visible (see Fig. 5.14 (f)), and there is no further partially visible segment.

**Midpoint Subdivision Algorithm :**

1. Read two endpoints of the line say $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$.
2. Read two corners (left-top and right-bottom) of the window, say ($Wx_1$, $Wy_1$ and $Wx_2$, $Wy_2$).
3. Assign region codes for two end points using following steps :

   Initialize code with bits 0000
   Set Bit 1 - if ($x < Wx_1$)
   Set Bit 2 - if ($x > Wx_2$)
   Set Bit 3 - if ($y < Wy_1$)

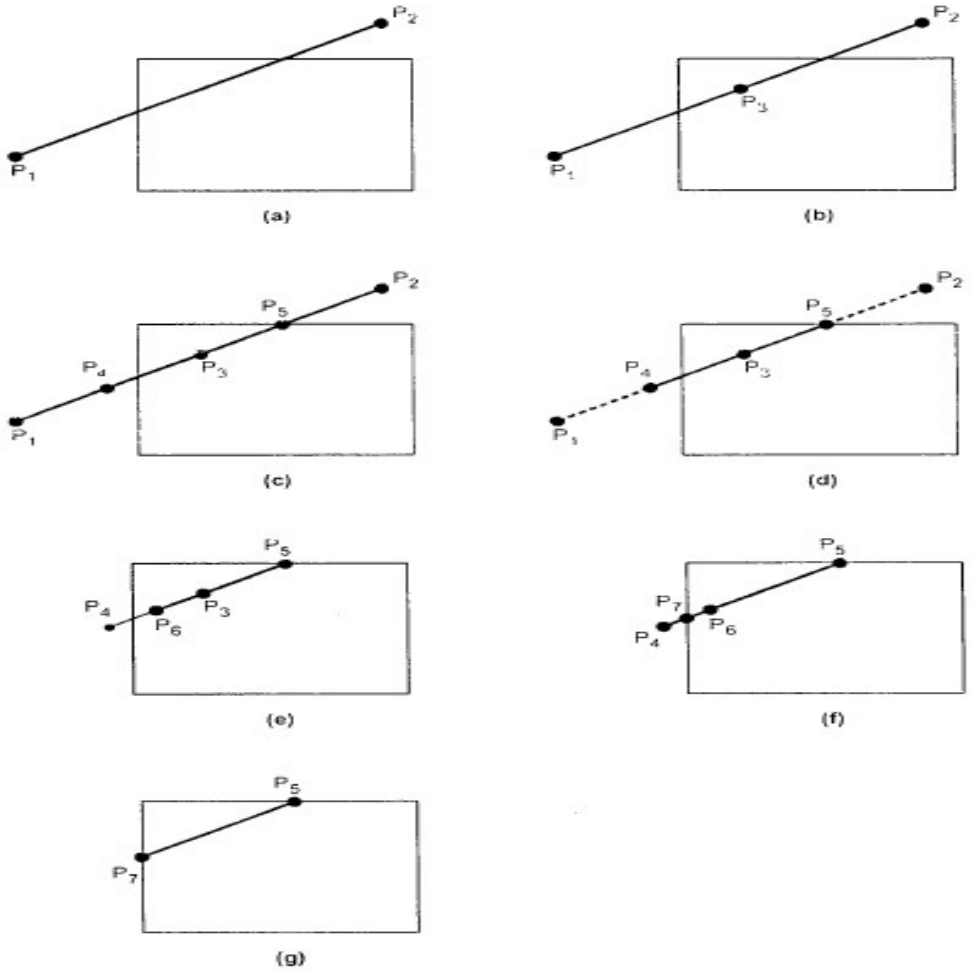



Fig. 5.14 Clipping line with midpoint subdivision algorithm

4. Check for visibility of line
   a) If region codes for both endpoints are zero then the line is completely visible. Hence draw the line and go to step 6.
   b) If region codes for endpoints are not zero and the logical ANDing of them is also nonzero then the line is completely invisible, so reject the line and go to step 6.
   c) If region codes for two endpoints do not satisfy the conditions in 4a) and 4b) the line is partially visible.
5. Divide the partially visible line segment in equal parts and repeat steps 3 through 5 for both subdivided line segments until you get completely visible and completely invisible line segments.
6. Stop.

## Polygon Clipping

A polygon can also be clipped by specifying the clipping window here all the vertices of the polygon are clipped against each edge of the clipping window. First the polygon is clipped against the left edge of the polygon window to get new vertices of the polygon. These new vertices are used to clip the polygon against right edge, top edge, bottom edge, of the clipping window as shown in the following figure.

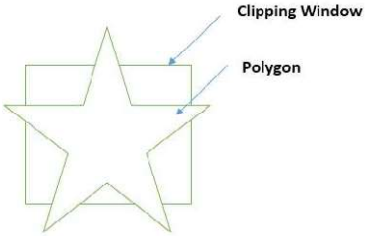



**Figure: Polygon before Filling**

While processing an edge of a polygon with clipping window, an intersection point is found if edge is not completely inside clipping window and the a partial edge from the intersection point to the outside edge is clipped. The following figures show left, right, top and bottom edge clippings:

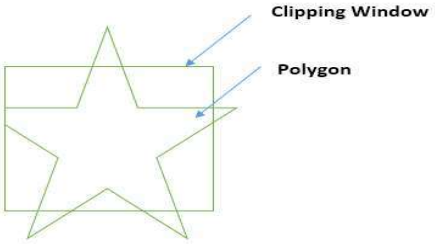



**Figure: Clipping Left Edge**

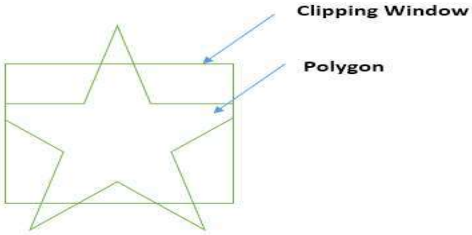



**Figure: Clipping Right Edge**

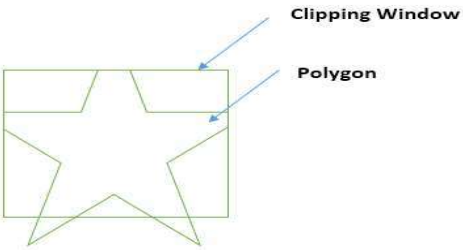



**Figure: Clipping Top Edge**

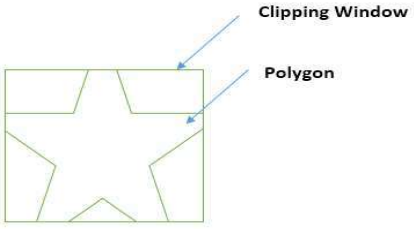



**Figure: Clipping Bottom Edge**

### Sutherland-Hodgeman Polygon Clipping Algorithm

1. Read coordinates of all vertices of the polygon.
2. Read coordinates of the clipping window
3. Consider the left edge of the window
4. Compare the vertices of each edge of the polygon, individually with the clipping plane

5. Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary discussed earlier.
6. Repeat the steps 4 and 5 for remaining edges of the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window.
7. Stop.

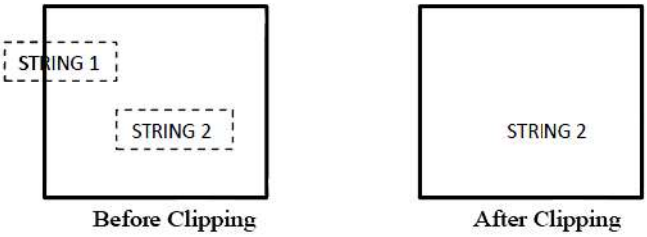
(Refer the above figure)

## Text Clipping

Various techniques are used to provide text clipping in a computer graphics. It depends on the methods used to generate characters and the requirements of a particular application. There are three methods for text clipping which are listed below:

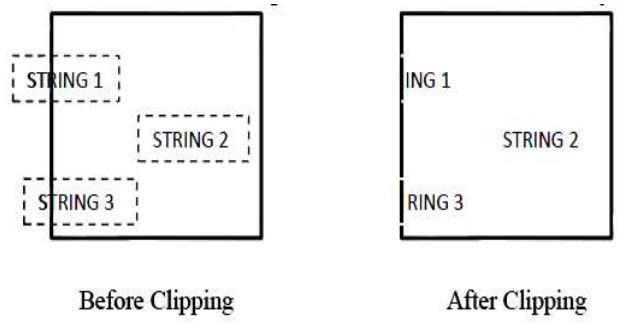
1. All or none string clipping
2. All or none character clipping
3. Text clipping

The following figure shows all or none string clipping:




In all or none string clipping method, either we keep the entire string or we reject entire string based on the clipping window. As shown in the above figure, STRING2 is entirely inside the clipping window so we keep it and STRING1 being only partially inside the window, we reject.

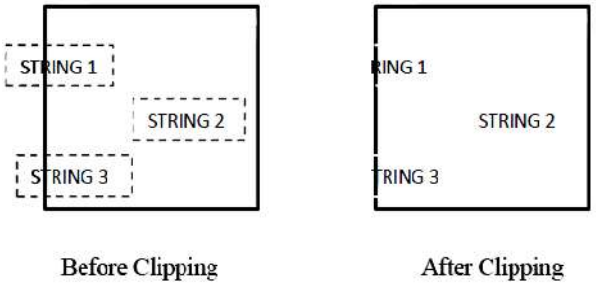
The following figure shows all or none character clipping:




This clipping method is based on characters rather than entire string. In this method if the string is entirely inside the clipping window, then we keep it. If it is partially outside the window, then:

- You reject only the portion of the string being outside
- If the character is on the boundary of the clipping window, then we discard that entire character and keep the rest string.

The following figure shows text clipping:




This clipping method is based on characters rather than the entire string. In this method if the string is entirely inside the clipping window, then we keep it. If it is partially outside the window, then

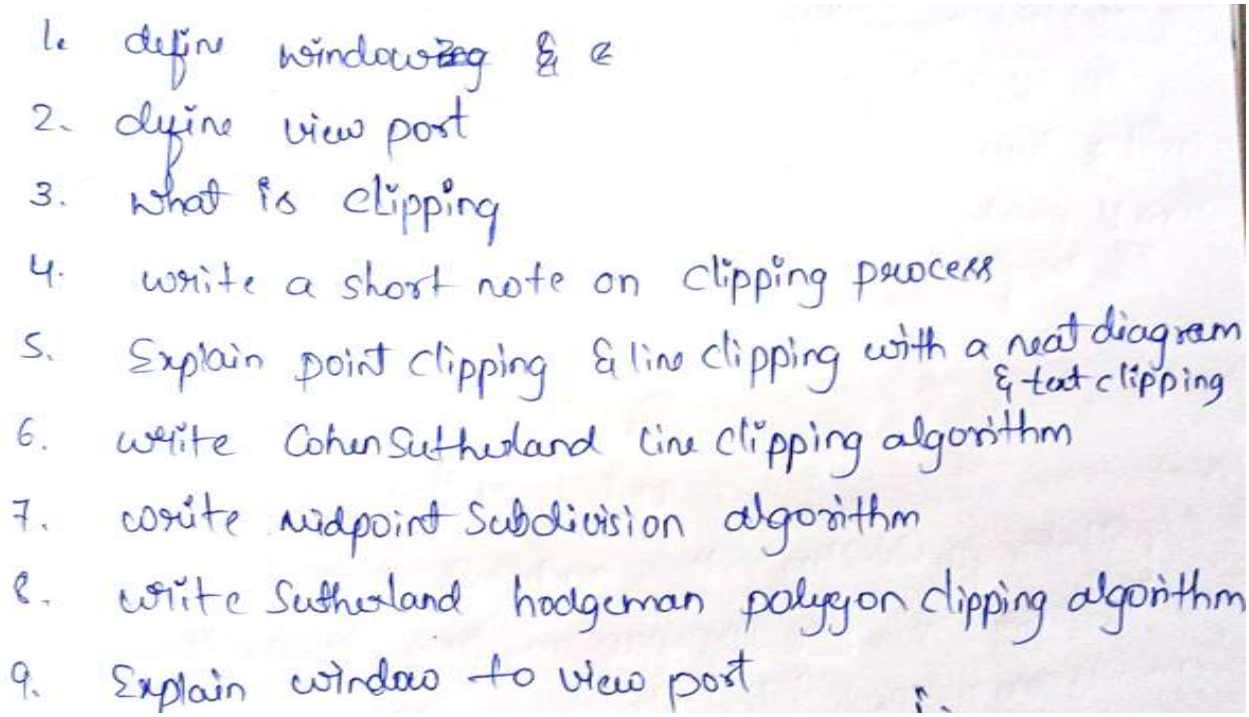
- You reject only the portion of string being outside.
- If the character is on the boundary of the clipping window, then we discard only that portion of character that is outside of the clipping window.

**Blanking**:

In raster scan equipment, an image is built up by scanning an electron beam from left to right across a screen to produce a visible trace of one scan line, reducing the brightness of the beam to zero (horizontal blanking), moving it back as fast as possible to the left of the screen at a slightly lower position (the next scan line), restoring the brightness, and continuing until all the lines have been displayed and the beam is at the bottom right of the screen. Its intensity is then reduced to zero again (vertical blanking), and it is rapidly moved to the top left to start again, creating the next Film frame.

In television, in particular, the vertical blanking interval is long to accommodate the slow equipment available at the time the standard was set. Fast modern electronics allows digital information to be encoded into the signal during the vertical blanking interval; it is not displayed on screen as the beam is blanked, but can be processed by appropriate circuitry.

**Questions:**



**Chapter 4:** **3D-Graphics and Segments**

## 3D Co-ordinate system:

A three-dimensional Cartesian coordinate system is formed by a point called the origin (denoted by O) and a basis consisting of three mutually perpendicular vectors. These vectors define the three coordinate axes: the x−, y−, and z−axis. They are also known as the abscissa, ordinate and applicate axis, respectively. The coordinates of any point in space are determined by three real numbers: x, y, z.

## 3D-Displays technique:

### 1.Parallel Projection

Parallel projection discards z-coordinate and parallel lines from each vertex on the object are extended until they intersect the view plane. In parallel projection, we specify a direction of projection instead of center of projection. In parallel projection, the distance from the center of projection to project plane is infinite. In this type of projection, we connect the projected vertices by line segments which correspond to connections on the original object. Parallel projections are less realistic, but they are good for exact measurements. In this type of projections, parallel lines remain parallel and angles are not preserved.

Various types of parallel projections are shown in the following hierarchy.

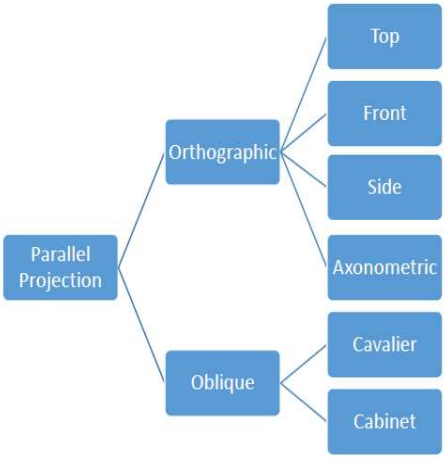
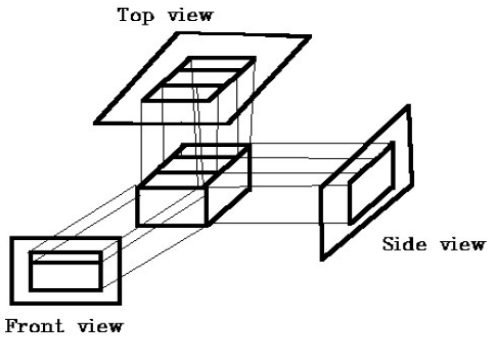



**Figure: Types of Parallel Projection**

### Orthographic Projection

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

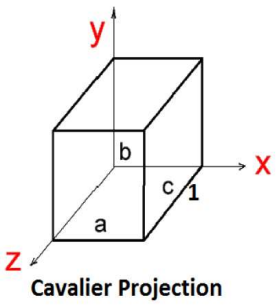
1. Front Projection
2. Top Projection
3. Side Projection



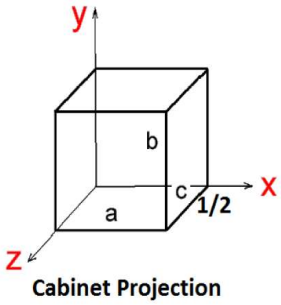

### Oblique Projection

In orthographic projection, the direction of projection is not normal to the projection of plane. In oblique projection, we can view the object better than orthographic projection.

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. Both the projections are shown in the above figure:




**Cavalier Projection**




**Cabinet Projection**

**Perspective Projection**

In perspective projection, the distance from the center of projection to project plane is finite and the size of the object varies inversely with distance which looks more realistic. The distance and angles are not preserved and parallel lines do not remain parallel. Instead, they all converge at a single point called **center of projection** or **projection reference point**. There are 3 types of perspective projections which are shown in the following chart.

- **One point** perspective projection is simple to draw.
- **Two point** perspective projection gives better impression of depth.
- **Three point** perspective projection is most difficult to draw.

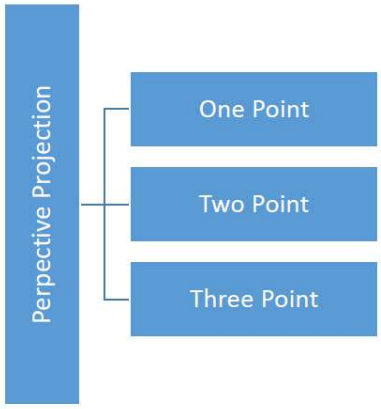



**Figure: Types of Perspective Projections**

The following figure shows all the three types of perspective projection:

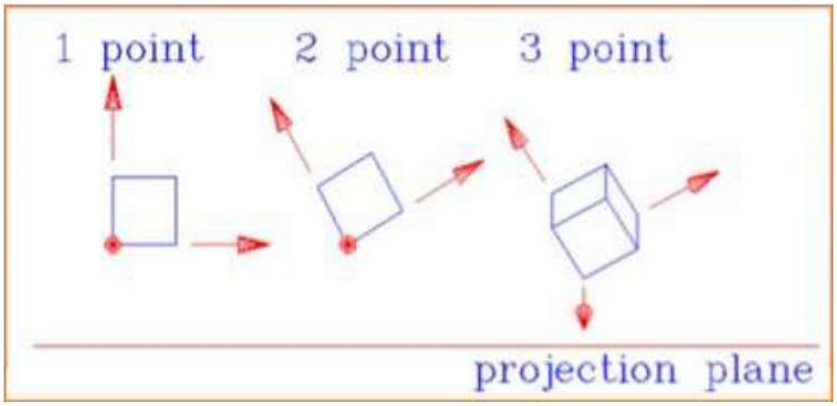



**Figure: 1-point, 2-point, 3-point perspective projection**

**Intensity cueing or Depth cueing**

Depth cueing can be used for many rendering effects such as fog, haze, mist but can also be used to render some sorts of water. It can also be used for the effect that objects farther away from the viewer get darker with increasing distance. In this document I'm gonna explain the latter one, the other ones can be rendered pretty easy using an inverse logic (atmospherical effects can be rendered by making use of the fact that in fog objects farther away are getting more and more "milky", that means are getting brighter with increasing distance). The water effect mentioned before refers to rendering murky or cloudy water (or more general each liquid with visibility reduced to a few meters). Generally depth cueing means that objects get darker, brighter or whatever with increasing distance.

**3D-transformations:**

**1.Translation**

In 3D translation, we transfer the Z coordinate along with the X and Y coordinates. The process for translation in 3D is similar to 2D translation. A translation moves an object into a different position on the screen.

The following figure shows the effect of translation:

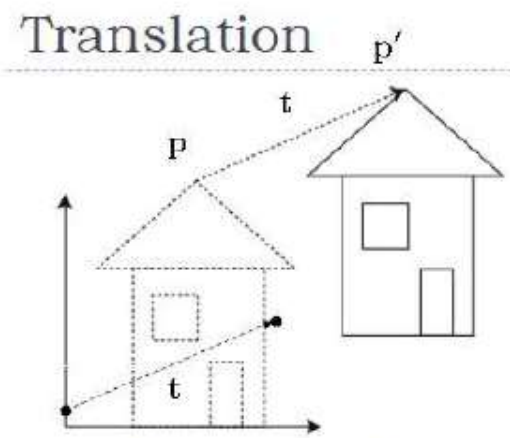



**Figure: 3D Translation**

A point can be translated in 3D by adding translation coordinate ($t_x$, $t_y$, $t_z$) to the original coordinate (X, Y, Z) to get the new coordinate (X′, Y′, Z′).

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

$P' = P \cdot T$

$$[X' \quad Y' \quad Z' \quad 1] = [X \quad Y \quad Z \quad 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

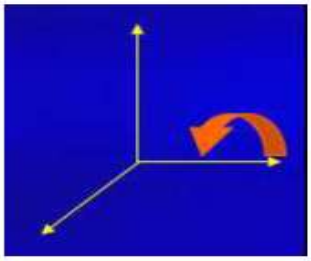
$$= [X + t_x \quad Y + t_y \quad Z + t_z \quad 1]$$

**Rotation**

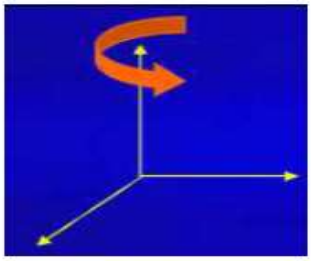
3D rotation is not same as 2D rotation. In 3D rotation, we have to specify the angle of rotation along with the axis of rotation. We can perform 3D rotation about X, Y, and Z axes. They are represented in the matrix form as below:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

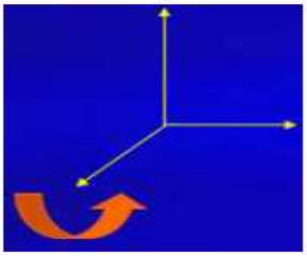
The following figure explains the rotation about various axes:



**Rotation about x-axis**



**Rotation about y-axis**



**Rotation about z-axis**

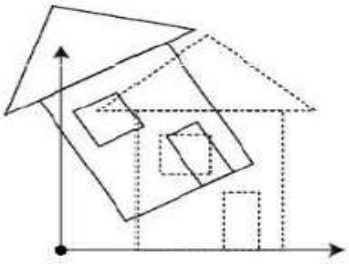


**Figure: 3D Rotation**

**Scaling**

You can change the size of an object using scaling transformation. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result. The following figure shows the effect of 3D scaling:

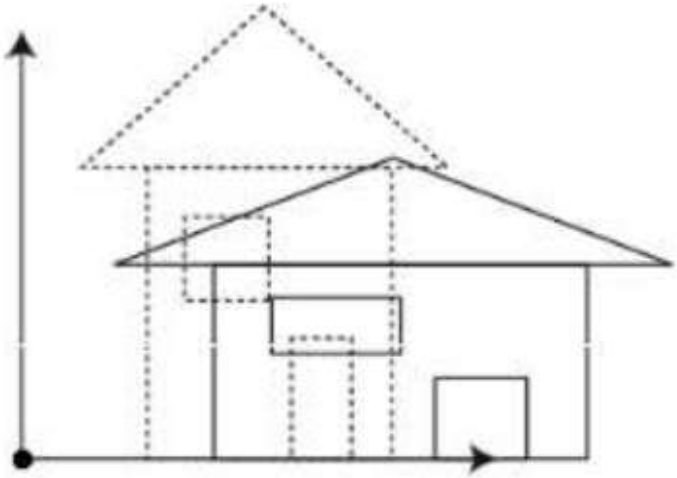


**Figure: 3D Scaling**

In 3D scaling operation, three coordinates are used. Let us assume that the original coordinates are (X, Y, Z), scaling factors are ($S_X$, $S_Y$, $S_Z$) respectively, and the produced coordinates are (X', Y', Z'). This can be mathematically represented as shown below:

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P' = P \cdot S$$

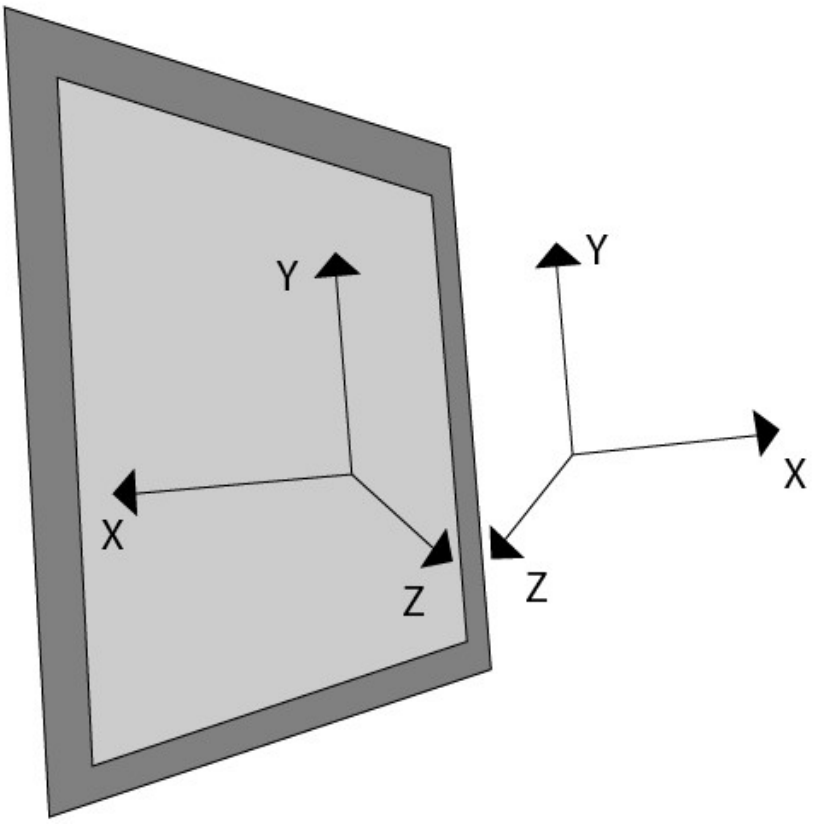
$$[X' \quad Y' \quad Z' \quad 1] = [X \quad Y \quad Z \quad 1] \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= [X \cdot S_x \quad Y \cdot S_y \quad Z \cdot S_z \quad 1]$$

**Reflection**

It is also called a mirror image of an object. For this reflection axis and reflection of plane is selected. Three-dimensional reflections are similar to two dimensions. Reflection is 180° about the given axis. For reflection, plane is selected (xy,xz or yz). Following matrices show reflection respect to all these three planes.

**Reflection relative to XY plane**




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

**Reflection relative to YZ plane**

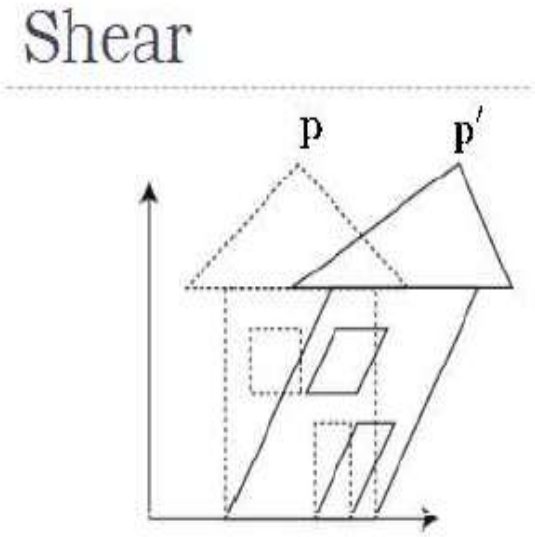
$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**Reflection relative to ZX plane**

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

**Shear**

A transformation that slants the shape of an object is called the **shear transformation**. Like in 2D shear, we can shear an object along the X-axis, Y-axis, or Z-axis in 3D.




As shown in the above figure, there is a coordinate P. You can shear it to get a new coordinate P', which can be represented in 3D matrix form as below:

$$Sh = \begin{bmatrix} 1 & Sh_x^y & Sh_x^z & 0 \\ Sh_y^x & 1 & Sh_y^z & 0 \\ Sh_z^x & Sh_z^y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P' = P \cdot Sh$$

$$X' = X + Sh_x^y\, Y + Sh_x^z\, Z$$

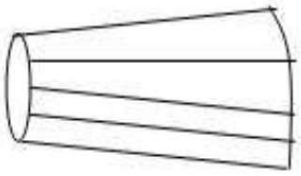
$$Y' = Sh_y^x\, X + Y + Sh_y^z\, Z$$

$$Z' = Sh_z^x\, X + Sh_z^y\, Y + Z$$

**Polygon Surfaces:** Polygon Surfaces are the collection of surfaces with multiple sides. Objects are represented as a collection of surfaces. 3D object representation is divided into two categories.

- **Boundary Representations (B-reps)**: It describes a 3D object as a set of surfaces that separates the object interior from the environment.
- **Space–partitioning representations:** It is used to describe interior properties, by partitioning the spatial region containing an object into a set of small, non-overlapping, contiguous solids (usually cubes).

The most commonly used boundary representation for a 3D graphics object is a set of surface polygons that enclose the object interior. Many graphics system use this method. Set of polygons are stored for object description. This simplifies and speeds up the surface rendering and display of object since all surfaces can be described with linear equations.



A 3D object represented by polygons

The polygon surfaces are common in design and solid-modeling applications, since their **wireframe display** can be done quickly to give general indication of surface structure. Then realistic scenes are produced by interpolating shading patterns across polygon surface to illuminate.

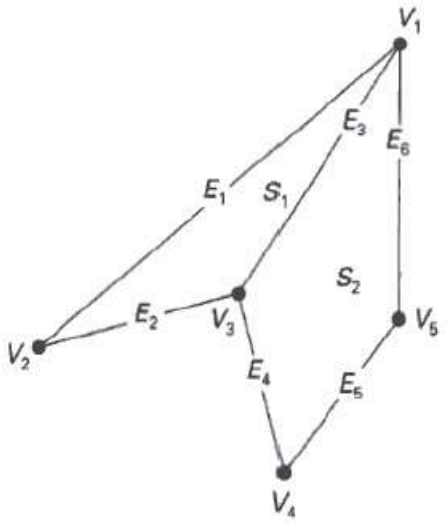
**Polygon Tables:**

In this method, the surface is specified by the set of vertex coordinates and associated attributes. As shown in the following figure, there are five vertices, from v1 to v5.

1.Each vertex stores x, y, and z coordinate information which is represented in the table as v1: x1, y1, z1.

2.The Edge table is used to store the edge information of polygon. In the following figure, edge E1 lies between vertex v1 and v2 which is represented in the table as E1: v1, v2.

3.Polygon surface table stores the number of surfaces present in the polygon. From the following figure, surface S1 is covered by edges E1, E2 and E3 which can be represented in the polygon surface table as S1: E1, E2, and E3.




| 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-SURFACE TABLE | |
|---|---|
| $S_1$: | $E_1, E_2, E_3$ |
| $S_2$: | $E_3, E_4, E_5, E_6$ |

**Figure: Polygon Table**

**Curves**

A curve is an infinitely large set of points. Each point has two neighbors except endpoints. Curves can be broadly classified into three categories: **explicit**, **implicit**, and **parametric curves**.

**Implicit Curves:**Implicit curve representations define the set of points on a curve by employing a procedure that can test to see if a point in on the curve. Usually, an implicit curve is defined by an implicit function of the form:

$$f(x, y) = 0$$

It can represent multivalued curves (multiple y values for an x value). A common example is the circle, whose implicit representation is

$$x2 + y2 - R2 = 0$$

**Explicit Curves:**A mathematical function y = f(x) can be plotted as a curve. Such a function is the explicit representation of the curve. The explicit representation is not general, since it cannot represent vertical lines and is also single-valued. For each value of x, only a single value of y is normally computed by the function.

**Parametric Curves:** Curves having parametric form are called parametric curves. The explicit and implicit curve representations can be used only when the function is known. In practice the parametric curves are used. A two-dimensional parametric curve has the following form:

$$P(t) = f(t), g(t) \text{ or } P(t) = x(t), y(t)$$

The functions f and g become the (x, y) coordinates of any point on the curve, and the points are obtained when the parameter t is varied over a certain interval [a, b], normally [0, 1].

## Bezier Curves

Bezier curve is discovered by the French engineer **Pierre Bézier**. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as:

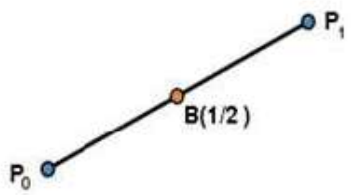
$$\sum_{k=0}^{n} P_i B_i^n(t)$$

Where $P_i$ is the set of points and $B_i^n(t)$ represents the Bernstein polynomials which are given by:

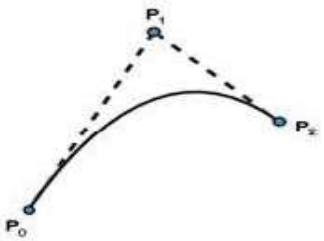
$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i} t^i$$

Where **n** is the polynomial degree, **i** is the index, and **t** is the variable.

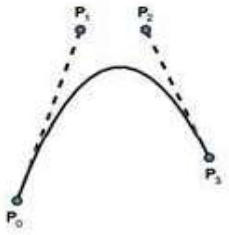
The simplest Bézier curve is the straight line from the point $P_0$ to $P_1$. A quadratic Bezier curve is determined by three control points. A cubic Bezier curve is determined by four control points.




Simple Bezier Curve




Quadratic Bazier Curve




Cubic Bazier Curve

**Properties of Bezier Curves**

Bezier curves have the following properties:

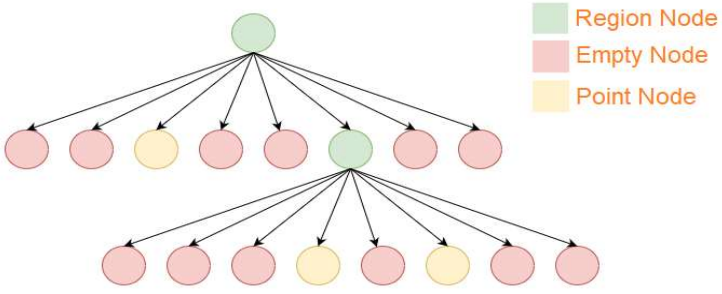
- They generally follow the shape of the control polygon, which consists of the segments joining the control points.
- They always pass through the first and last control points.
- They are contained in the convex hull of their defining control points.
- The degree of the polynomial defining the curve segment is one less that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.
- A Bezier curve generally follows the shape of the defining polygon.
- The direction of the tangent vector at the end points is same as that of the vector determined by first and last segments.
- The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
- No straight line intersects a Bezier curve more times than it intersects its control polygon.
- They are invariant under an affine transformation.
- Bezier curves exhibit global control means moving a control point alters the shape of the whole curve.
- A given Bezier curve can be subdivided at a point t=t0 into two Bezier segments which join together at the point corresponding to the parameter value t=t0.

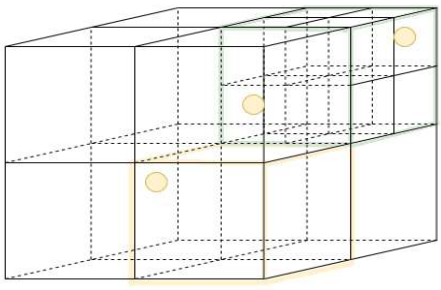
**Octree**:

Octree is a tree data structure where each internal node has 8 children. An octree is generally used to represent relation between objects in a 3-dimensional space. It is used in 3D computer graphics. Octrees are also used for nearest neighbor search which can be done easily in logarithmic time. Like a binary tree divides a 1-dimensional space into two segments, an octree subdivides the 3D space into 8 octants where each octant is represented by a node.

An octree save a large amount of space when storing points in a 3D space, especially if the space is sparsely populated. If there are k points in a 3D cube of dimension N, then space used by the tree is $O(klog_2N)$

The image below shows how an octree represents points in a space.



**Algorithm**

Three types of nodes are used in octree:

1. **Point node**: Used to represent of a point. Is always a leaf node.
2. **Empty node**: Used as a leaf node to represent that no point exists in the region it represent.
3. **Region node**: This is always an internal node. It is used to represent a 3D region or a cuboid. A region node always have **4 children** nodes that can either be a point node or empty node.

**Fractals:**

Fractals are very complex pictures generated by a computer from a single formula. They are created using iterations. This means one formula is repeated with slightly different values over and over again, taking into account the results from the previous iteration. Fractals are used in many areas such as:

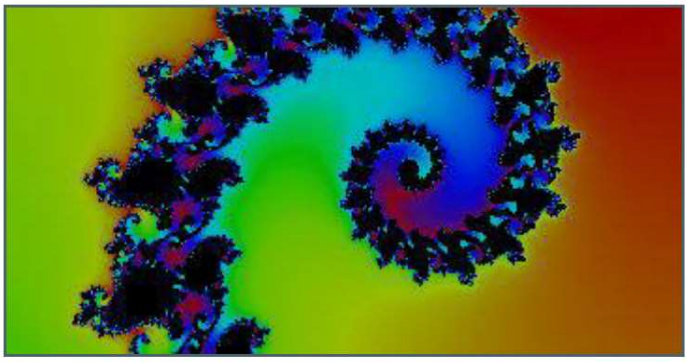


**Figure: Fractals**

- **Astronomy:** For analyzing galaxies, rings of Saturn, etc.
- **Biology/Chemistry:** For depicting bacteria cultures, Chemical reactions, human anatomy, molecules, plants,
- **Others:** For depicting clouds, coastline and borderlines, data compression, diffusion, economy, fractal art, fractal music, landscapes, special effect, etc.

**Generation of Fractals:**

Fractals can be generated by repeating the same shape over and over again as shown in the following figure. In figure (a) shows an equilateral triangle. In figure (b), we can see that the triangle is repeated to create a star-like shape. In figure (c), we can see that the star shape in figure (b) is repeated again and again to create a new shape.

We can do unlimited number of iteration to create a desired shape. In programming terms, recursion is used to create such shapes.

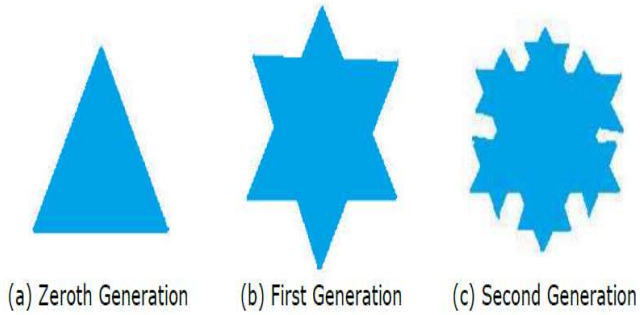



**Figure: Generation of Fractals**

**Geometric Fractals**

Geometric fractals deal with shapes found in nature that have non-integer or fractal dimensions. To geometrically construct a deterministic (nonrandom) self-similar fractal, we start with a given geometric shape, called the **initiator**. Subparts of the initiator are then replaced with a pattern, called the **generator**.

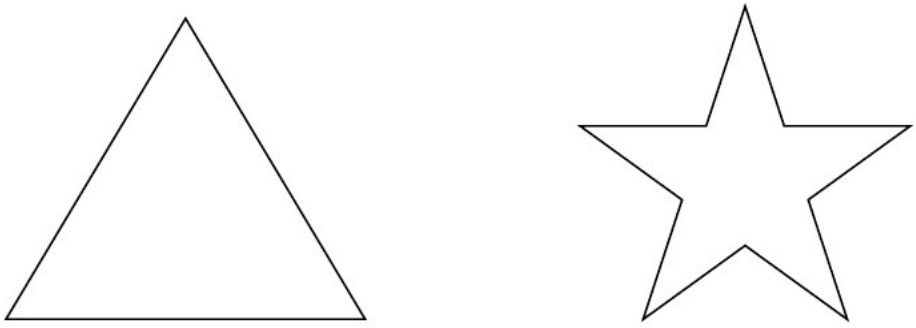
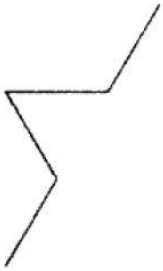



**Figure: Initiator and Generator for fractals**

As an example, if we use the initiator and generator shown in the above figure, we can construct good pattern by repeating it. Each straight-line segment in the initiator is replaced with four equal-ength line segments at each step. The scaling factor is 1/3, so the fractal dimension is D = ln 4/ln 3 ≈ 1.2619.

Also, the length of each line segment in the initiator increases by a factor of 4/3 at each step, so that the length of the fractal curve tends to infinity as more detail is added to the curve as shown in the following figure:

| Segment Length = 1 | Segment Length = 1/3 | Segment Length = 1/9 |
|---|---|---|
|  |  |  |
| Length = 1 | Length = 4/3 | Length = 16/9 |

## Hidden line and surface removal algorithm-

### Depth Buffer (Z-Buffer) Method

This method is developed by Cutmull. It 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)

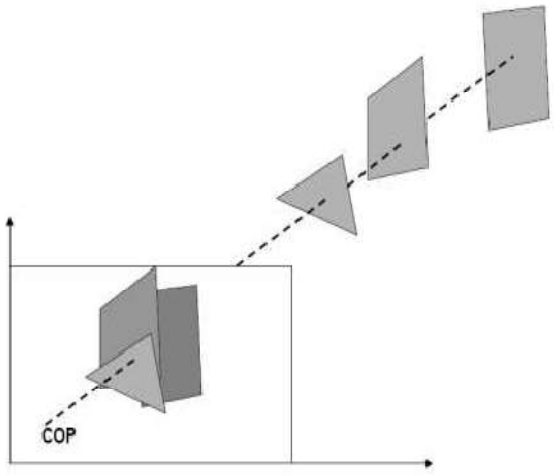
For each projected (x, y) pixel position of a polygon, calculate depth z.

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.

**Back Face Removal Algorithm**

It is used to plot only surfaces which will face the camera. The objects on the back side are not visible. This method will remove 50% of polygons from the scene if the parallel projection is used. If the perspective projection is used then more than 50% of the invisible area will be removed. The object is nearer to the center of projection, number of polygons from the back will be removed.

It applies to individual objects. It does not consider the interaction between various objects. Many polygons are obscured by front faces, although they are closer to the viewer, so for removing such faces back face removal algorithm is used.

When the projection is taken, any projector ray from the center of projection through viewing screen to object pieces object at two points, one is visible front surfaces, and another is not visible back surface.

This algorithm acts a preprocessing step for another algorithm. The back face algorithm can be represented geometrically. Each polygon has several vertices. All vertices are numbered in clockwise. The normal M1 is generated a cross product of any two successive edge vectors. M1represent vector perpendicular to face and point outward from polyhedron surface

N1=(v2-v1 )(v3-v2)
If N1.P≥0 visible
N1.P<0 invisible

**Advantage**

- It is a simple and straight forward method.
- It reduces the size of databases, because no need of store all surfaces in the database, only the visible surface is stored.

**Back Face Removal Algorithm**

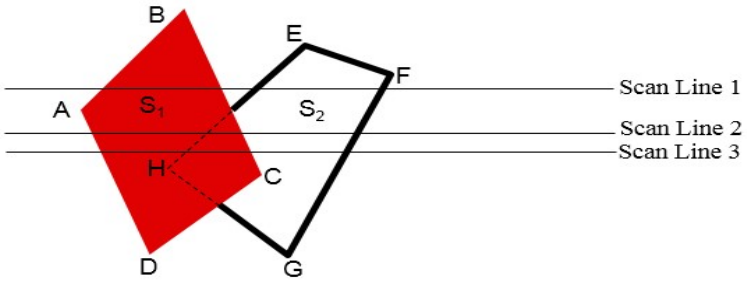
Repeat for all polygons in the scene.
1.Do numbering of all polygons in clockwise direction i.e.
v1 v2 v3.....vz
2.Calculate normal vector i.e. N1
N1=(v2-v1 )*(v3-v2)
3.Consider projector P, it is projection from any vertex
Calculate dot product
Dot=N.P
4.Test and plot whether the surface is visible or not.
If Dot ≥ 0 then
surface is visible
else
Not visible

**Scan-Line Method**

It is an image-space method to identify visible surface. This method has a depth information for only single scan-line. In order to require one scan-line of depth values, we must group and process all polygons intersecting a given scan-line at the same time before processing the next scan-line. Two important tables, **edge table** and **polygon table**, are maintained for this.

**The Edge Table:** It contains coordinate endpoints of each line in the scene, the inverse slope of each line, and pointers into the polygon table to connect edges to surfaces.

**The Polygon Table:** It contains the plane coefficients, surface material properties, other surface data, and may be pointers to the edge table.




To facilitate the search for surfaces crossing a given scan-line, an active list of edges is formed. The active list stores only those edges that cross the scan-line in order of increasing x. Also a flag is set for each surface to indicate whether a position along a scan-line is either inside or outside the surface.

Pixel positions across each scan-line are processed from left to right. At the left intersection with a surface, the surface flag is turned on and at the right, the flag is turned off. You only need to perform depth calculations when multiple surfaces have their flags turned on at a certain scan-line position.

**Constructive solid geometry (CSG)**

Constructive solid geometry (CSG) formerly called computational binary solid geometry) is a technique used in solid modelling. Constructive solid geometry allows a modeller to create a complex surface or object by using Boolean operators to combine simpler objects, potentially generating visually complex objects by combining a few primitive ones.

In 3D computer graphics and CAD, CSG is often used in procedural modelling. CSG can also be performed on polygonal meshes, and may or may not be procedural and/or parametric.

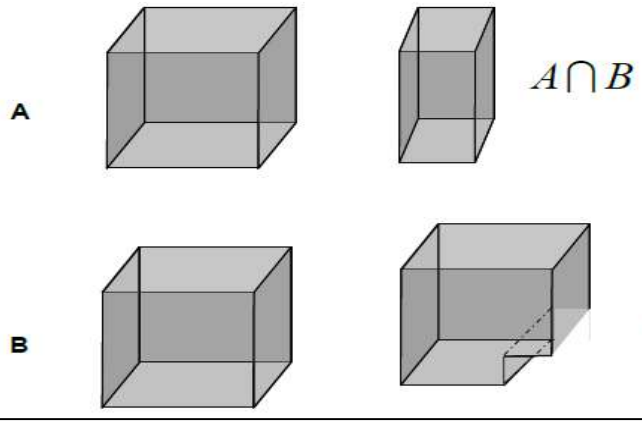
Contrast CSG with polygon mesh modelling and box modelling.

**Solid is bound by surfaces. So need to also define the polygons of vertices, which form the solid. It must also be a valid representation.**

**Regularized Boolean Set Operations**

**Operators (*op*):**
**Union:** ∪
**Intersection:** ∩
**Difference:** −

**Boolean intersections of, say cubes, may produce solids, planes, lines, points and null object. Two examples:**



**Sweep Representations:-**

Sweep representations are used to construct three dimensional objects from two dimensional shape .There are two ways to achieve sweep: **Translational sweep** and **Rotational sweep**. In translational sweeps, the 2D shape is swept along a linear path normal to the plane of the area to construct three dimensional object. To obtain the wireframe representation we have to replicate the 2D shape and draw a set of connecting lines in the direction of shape, as shown in the figure (8)

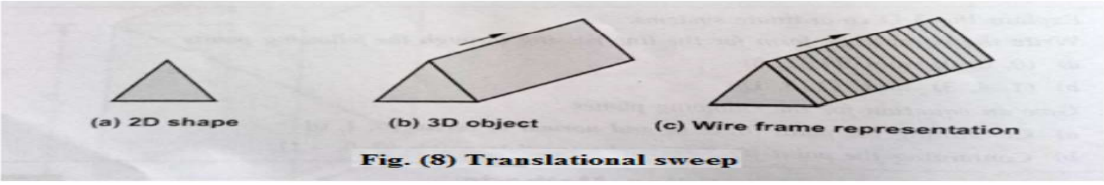



**Fig. (8) Translational sweep**

In rotational sweeps, the 2D shape is rotated about an a axis of rotation specified in the plane of 2D shape to produce three dimensional object. This is illustrated in figure (9).

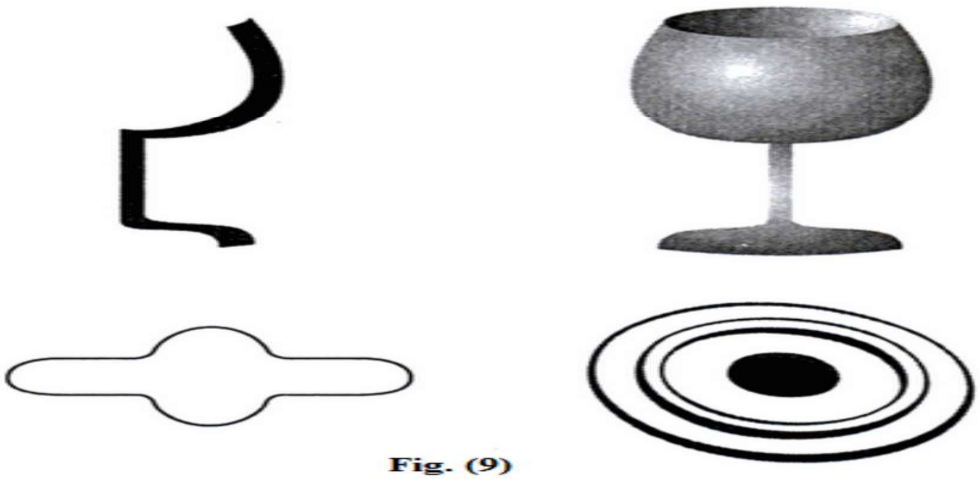


**Fig. (9)**

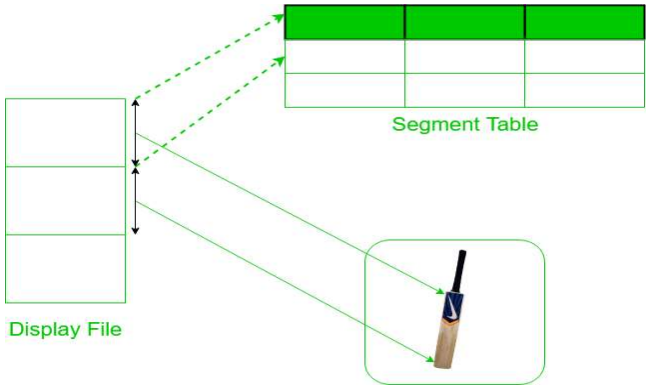
In general we can specify sweep constructions using any path. For translation we can vary the shape or size of the original 2D shape along the sweep path. For rotational sweeps, we can move along a circular path through any angular distance from 0° to 360°. These sweeps whose generating area or volume changes in size, shape or orientation as they are swept and that follow an arbitrary curved trajectory are called general sweeps .General sweeps are difficult to model efficiently for example, the trajectory and object shape may make the swept object intersect itself, making volume calculations complicated. Further more, general sweeps do not always generate solids. For example, sweeping a 2D shape in its own plane generates another 2D shape.

**Segments in Computer Graphics**

A segment is a defined portion or section of something larger such as a database, geometric object, or network. The term is used in database management, graphics, and communications.

To view an entire image or a part of image with various attributes, we need to organize image information in a particular manner since existing structure of display file does not satisfy our requirements of viewing an image. To achieve this display, file is divided into **Segments**. Each segment corresponds to a component and is associated with a set of attributes and image transformation parameters like scaling, rotation. Presence of Segment allows :

- Subdivision of picture.
- Visualization of particular part of picture.
- Scaling, rotation and translation of picture.




**Types of Segments :**

- **Posted Segment :** When visible attribute of segment is set to 1, it is called Posted segment. This is included in active segment list.
- **Unposted Segment :** When visible attribute of segment is set to 0, it is called Unposted segment. This is not included in active segment list.

**Functions for Segmenting the display :**

1. **Segment Creation :** Segment must be created or opened when no other segment is open, since two segments can't be opened at the same time because it's difficult to assign drawing instruction to particular segment. The segment created must be given a name to identify it which must be a **valid** one and there should be no segment with the same name. After this, we initialize items in segment table under our segment name and the first instruction of this segment is allocated at next free storage in display file and attributes of segments are initialized to default.

   **Algorithm :**

   1. If any segment is open, give error message : ***"Segment is still open"*** and go to step 8.
   2. Read the name of the new segment.
   3. If the segment name is not valid, give error message : ***"Segment name not a valid name"*** and go to step 8.
   4. If given segment name already exists, give error message : ***"Segment name already exists in name list"*** and go to step 8.
   5. Make next free storage area in display file as start of new segment.
   6. Initialize size of new segment to 0 and all its attributes to their default values.
   7. Inform that the new segment is now open.
   8. Stop.

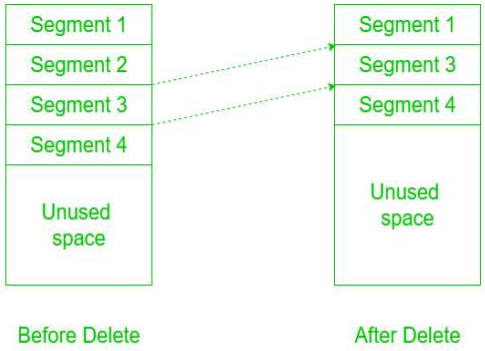
2. **Closing a Segment :** After completing entry of all display file instructions, the segment needs to be closed for which it has to be renamed, which is done by changing the name of currently open segment as 0. Now the segment with name 0 is open i.e. unnamed segment is open and if two unnamed segments are present in display file one needs to be deleted.

   **Algorithm :**

   1. If any segment is not open, give error message : ***"No segment is open now"*** and go to step 6.

2. Change the name of currently opened segment to any unnamed segment, lets say 0.
3. Delete any other unnamed segment instruction which may have been saved and initialize above unnamed segment with no instructions.
4. Make the next free storage area available in display file as start of the unnamed segment.
5. Initialize size of unnamed segment to 0.
6. Stop.

3. **Deleting a Segment :** To delete a particular segment from display file, we must just delete that one segment without destroying or reforming the entire display and recover space occupied by this segment. Use this space for some other segment. The method to achieve this depends upon the data structure used to represent display file. In case of arrays, the gap left by deleted segment is filled by shifting up all the segments following it.




**Display file contents before and after deleting Segment 2**

**Algorithm :**

1. Read the name of the segment to be deleted.
2. If segment name is not valid, give error message : ***"Segment name is not a valid name"*** and go to step 8.
3. If the segment is open, give error message : ***"Can't delete an open segment"*** and go to step 8.
4. If size of segment is less than 0, no processing is required and go to step 8.
5. The segments which follow the deleted segment are shifted by its size.
6. Recover deleted space by resetting index of next free instruction.
7. The starting position of shifted segments is adjusted by subtracting the size of deleted segment from it.
8. Stop.

4. **Renaming a Segment :** This is done to achieve **Double Buffering** i.e. the idea of storing two images, one to show and other to create, alter and for animation.

**Algorithm :**

1. If both old and new segment names are not valid, give error message : ***"Segment names are not valid names"*** and go to step 6.
2. If any of two segments is open, give error message : ***"Segments are still open"*** and go to step 6.
3. If new segment name given already exists in the display list, give error message : ***"Segment name already exists"*** and go to step 6.
4. The old segment table entry are copied into new position.
5. Delete the old segment.
6. Stop.

**Advantages of using segmented display :**

- Segmentation allows to organize display files in sub-picture structure.
- It allows to apply different set of attributes to different portions of image.
- It makes it easier to the picture by changing/replacing segments.
- It allows application of transformation on selective portions of image.

**SEGMENT ATTRIBUTES:**

**1.Segment Transformations:** The segment transformation is a transformation operating on all the coordinates in the segment definition. When a segment is created, the segment transformation is set to the null, or identity, transformation. The segment transformation can be subsequently changed by invoking the function:

**2.Segment Transformation and Clipping:**The segment transformation is actually applied after the normalization transformation (which, it will be recalled, maps world coordinates into normalized device coordinates) but before any clipping. Coordinates stored in segments are in fact normalized device coordinates. It will be recalled that the elements of the transformation matrix which define the shift to be applied are expressed in normalized device coordinates.

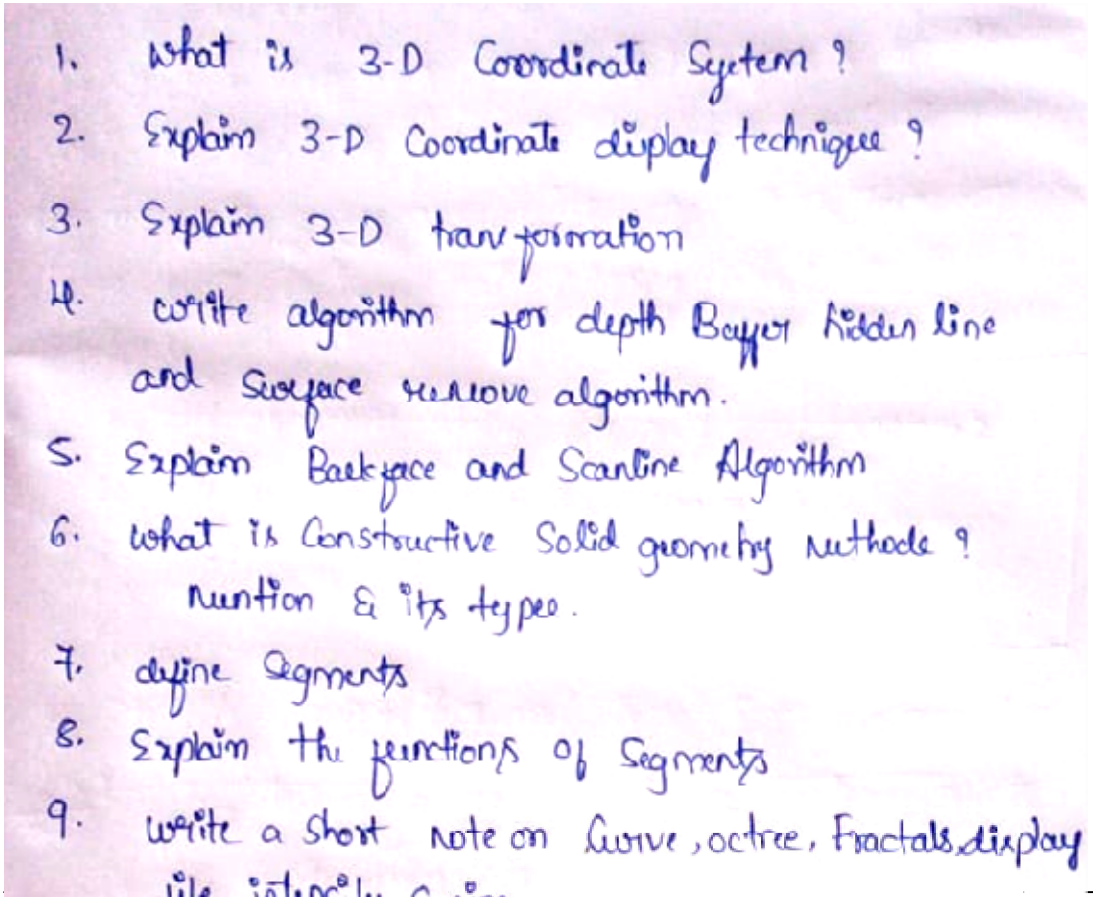
**3 Segment Visibility:** The segment visibility attribute determines whether a segment will be displayed or not. By default when a segment is created it is visible

**4 Segment Highlighting:** Most vector refresh display hardware has the capability of highlighting a segment, for example by causing it to blink. The principal use of this facility is in drawing the operator's attention to some facet of the display.

**Display file:** A display file is a set of uncorrelated data, such as a histogram array or bivariate array. The arrays are filled event by event from a list data in order to create a display. The saved arrays usually take up far less disk space, but can the data can no longer be gated.

**Display file compilation:** includes the display file which is used by the display processor to refresh the screen.

**Questions:**



**Chapter 5:** **Graphical Input devices and Input Techniques**

# Graphics input devices

## Keyboards

- Keyboards are used as entering text strings. It is efficient devices for inputting such a non-graphics data as picture label.
- Cursor control key's & function keys are common features on general purpose keyboards.
- Many other application of key board which we are using daily used of computer graphics are commanding & controlling through keyboard etc.

## Mouse

- Mouse is small size hand-held box used to position screen cursor.
- Wheel or roller or optical sensor is directing pointer on the according to movement of mouse.
- Three buttons are placed on the top of the mouse for signaling the execution of some operation.
- Now a day's more advance mouse is available which are very useful in graphics application for example Z mouse.

## Trackball and Spaceball

- Trackball is ball that can be rotated with the finger or palm of the hand to produce cursor movement.
- Potentiometer attached to the ball, measure the amount and direction of rotation.
- They are often mounted on keyboard or Z mouse.
- Space ball provide six-degree of freedom i.e. three dimensional.
- In space ball strain gauges measure the amount of pressure applied to the space ball to provide input for spatial positioning and orientation as the ball is pushed or pulled in various directions.
- Space balls are used in 3D positioning and selection operations in virtual reality system, modeling, animation, CAD and other application.

## Joysticks

- A joy stick consists of small vertical lever mounted on a base that is used to steer the screen cursor around.
- Most joy sticks selects screen positioning according to actual movement of stick (lever).
- Some joy sticks are works on pressure applied on sticks.
- Sometimes joy stick mounted on keyboard or sometimes used alone.
- Movement of the stick defines the movement of the cursor.
- In pressure sensitive stick pressure applied on stick decides movement of the cursor. This pressure is measured using strain gauge.
- This pressure sensitive joy sticks also called as isometric joy sticks and they are non movable sticks.

## Touch Panels

- As name suggest Touch Panels allow displaying objects or screen-position to be selected with the touch or finger.
- A typical application is selecting processing option shown in graphical icons.

- Some system such as a plasma panel are designed with touch screen
- Other system can be adapted for touch input by fitting transparent touch sensing mechanism over a screen.
- Touch input can be recorded with following methods.
  1. Optical methods
  2. Electrical methods
  3. Acoustical methods

### Optical method

- Optical touch panel employ a line of infrared LEDs along one vertical and one horizontal edge.
- The opposite edges of the edges containing LEDs are contain light detectors.
- When we touch at a particular position the line of light path breaks and according to that breaking line coordinate values are measured.
- In case two line cuts it will take average of both pixel positions.
- LEDs operate at infrared frequency so it cannot be visible to user.

### Electrical method

- An electrical touch panel is constructed with two transparent plates separated by small distance.
- One is coated with conducting material and other is coated with resistive material.
- When outer plate is touch it will come into contact with internal plate.
- When both plates touch it creates voltage drop across the resistive plate that is converted into coordinate values of the selected position.

### Acoustical method

- In acoustical touch panel high frequency sound waves are generated in horizontal and vertical direction across a glass plates.
- When we touch the screen the waves from that line are reflected from finger.
- These reflected waves reach again at transmitter position and time difference between sending and receiving is measure and converted into coordinate values.

## Light pens

- Light pens are pencil-shaped device used to select positions by detecting light coming from points on the CRT screen.
- Activated light pens pointed at a spot on the screen as the electron beam lights up that spot and generate electronic pulse that causes the coordinate position of the electron beam to be recorded.

**Graphics tablet**

A **graphics tablet** (also known as a **digitizer**, **drawing tablet**, **drawing pad**, **digital drawing tablet**, **pen tablet**, or **digital art board**) is a computer input device that enables a user to hand-draw images, animations and graphics, with a special pen-like stylus, similar to the way a person draws images with a pencil and paper. These tablets may also be used to capture data or handwritten signatures. It can also be used to trace an image from a piece of paper which is taped or otherwise secured to the tablet surface. Capturing data in this way, by tracing or entering the corners of linear poly-lines or shapes, is called digitizing.

**Predefined graphics functions:** The interface which provides access to a simple graphics library that makes it possible to draw lines, rectangles, ovals, arcs, polygons, images, and strings on a graphical window is called predefined graphics functions.

**Functions**

| | |
|---|---|
| **initGraphics()**<br>**initGraphics(width, height)** | Creates the graphics window on the screen. |
| **drawArc(bounds, start, sweep)**<br>**drawArc(x, y, width, height, start, sweep)** | Draws an elliptical arc inscribed in a rectangle. |
| **fillArc(bounds, start, sweep)**<br>**fillArc(x, y, width, height, start, sweep)** | Fills a wedge-shaped area of an elliptical arc. |
| **drawImage(filename, pt)**<br>**drawImage(filename, x, y)**<br>**drawImage(filename, bounds)**<br>**drawImage(filename, x, y, width, height)** | Draws the image from the specified file with its upper left corner at the specified point. |
| **getImageBounds(filename)** | Returns the bounds of the image contained in the specified file. |
| **drawLine(p0, p1)**<br>**drawLine(x0, y0, x1, y1)** | Draws a line connecting the specified points. |
| **drawPolarLine(p0, r, theta)**<br>**drawPolarLine(x0, y0, r, theta)** | Draws a line of length **r** in the direction **theta** from the initial point. |
| **drawOval(bounds)**<br>**drawOval(x, y, width, height)** | Draws the frame of a oval with the specified bounds. |
| **fillOval(bounds)**<br>**fillOval(x, y, width, height)** | Fills the frame of a oval with the specified bounds. |
| **drawRect(bounds)**<br>**drawRect(x, y, width, height)** | Draws the frame of a rectangle with the specified bounds. |
| **fillRect(bounds)**<br>**fillRect(x, y, width, height)** | Fills the frame of a rectangle with the specified bounds. |
| **drawPolygon(polygon)**<br>**drawPolygon(polygon, pt)**<br>**drawPolygon(polygon, x, y)** | Draws the outline of the specified polygon. |
| **fillPolygon(polygon)**<br>**fillPolygon(polygon, pt)**<br>**fillPolygon(polygon, x, y)** | Fills the frame of the specified polygon. |
| **drawString(str, pt)**<br>**drawString(str, x, y)** | Draws the string **str** so that its baseline origin appears at the specified point. |

| | |
|---|---|
| **getStringWidth(str)** | Returns the width of the string **str** when displayed in the current font. |
| **setFont(font)** | Sets a new font. |
| **getFont()** | Returns the current font. |
| **setColor(color)** | Sets the color used for drawing. |
| **getColor()** | Returns the current color as a string in the form **"#rrggbb"**. |
| **saveGraphicsState()** | Saves the state of the graphics context. |
| **restoreGraphicsState()** | Restores the graphics state from the most recent call to **saveGraphicsState()**. |
| **getWindowWidth()** | Returns the width of the graphics window in pixels. |
| **getWindowHeight()** | Returns the height of the graphics window in pixels. |
| **repaint()** | Issues a request to update the graphics window. |
| **pause(milliseconds)** | Pauses for the indicated number of milliseconds. |
| **waitForClick()** | Waits for a mouse click to occur anywhere in the window. |
| **setWindowTitle(title)** | Sets the title of the primary graphics window. |
| **getWindowTitle()** | Returns the title of the primary graphics window. |
| **exitGraphics()** | Closes the graphics window and exits from the application without waiting for any additional user interaction. |

**Positioning technique**

the positioning technique refers to position the item on the screen to a new position, i.e., the old current position. The user indicates a position on the screen with an input device, and this position is used to insert a symbol.

**Grid**

A grid system is a set of measurements a graphic designer can use to align and size objects within the given format.

A grid system in graphic design uses a two-dimensional framework to align and lay out design elements. Breaking down a single design space into a grid can help position individual components in ways that can catch the eye, create a user flow and make information and visuals more appealing and accessible to audiences.

Many grid systems are composed of boxes, squares and rectangles. Design components can be placed within these boxes to achieve layout goals. Other types of grids include more diverse geometric

shapes, and can be used on asymmetric designs such as a picture of a human or animal. In either case, the grid system utilizes geometric lines to break down the total space into a set of geometric components. Graphic design elements are then applied to each of these areas, to create a unified result.

Another kind of constraint is a grid of rectangular lines displayed in some part of the screen area. When a grid is used, any input coordinate position is rounded to the nearest intersection of two grid lines.

**Constraints:**

Constraint is a rule for altering input coordinates values to produce a specified orientation or alignment of the displayed coordinates. the most common constraint is a horizontal or vertical alignment of straight lines.

The idea of **constraint** programming is to solve problems by stating **constraints** (conditions, properties) which must be satisfied by the solution. A **constraint** is a relation between multiple variables which limits the values these variables can take simultaneously.

Ex: drawing a circle, square etc with some suitable conditions or constraints.

**Dynamic manipulation:**

Dynamically manipulate parameters in graphics by changing the values for the dynamic manipulation.

**Gravity field:**

When it is needed to connect lines at positions between endpoints, the graphics packages convert any input position near a line to a position on the line. The conversion is accomplished by creating a gravity area around the line. Any related position within the gravity field of line is moved to the nearest position on the line. It illustrated with a shaded boundary around the line.

**Rubber Band Techniques**

Rubber banding is a popular technique of drawing geometric primitives such as line, polylines, rectangle, circle and ellipse on the computer screen. It becomes an integral part and de facto standard with the graphical user interface (GUI) for drawing and is almost universally accepted by all windows based applications.

The user specifies the line in the usual way by positioning its two endpoints. As we move from the first endpoint to the second, the program displays a line from the first endpoint to the cursor position, thus he can see the lie of the line before he finishes positioning it. The effect is of an elastic line stretched between the first endpoint and the cursor; hence the name for these techniques.

Consider the different linear structures in fig (a) and fig (d), depending on the position of the cross-hair cursor. The user may move the cursor to generate more possibilities and select the one which suits him for a specific application.

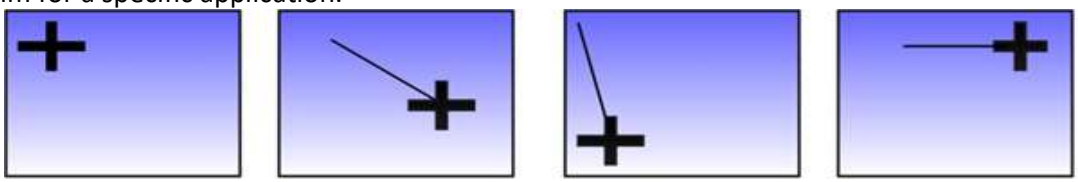
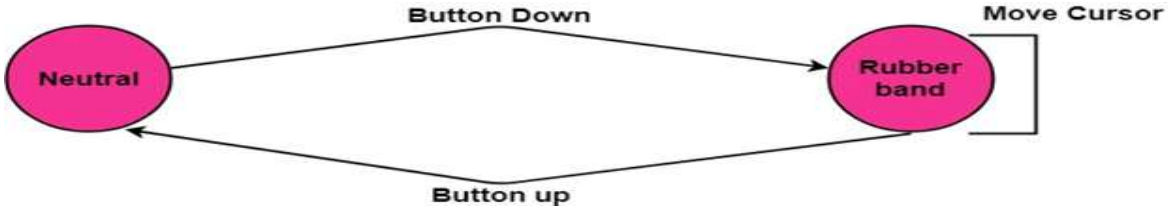


**Fig:Demonstration of rubber banding: look at the cross-hair cursor(a) The start point of the line to be drawn is selected (b) 3 different lines are shown depending on the position of the cursor representing the end-point the user selects the desired line.**

- Selection of Terminal Point of the Line:
- The user moves the cursor to the appropriate position and selects.

- Then, as the cursor is moved, the line changes taking the latest positions of the cursors as the end-point.
- As long as the button is held down, the state of the rubber band is active.
- The process is explained with the state transition diagram of rubber banding in fig:



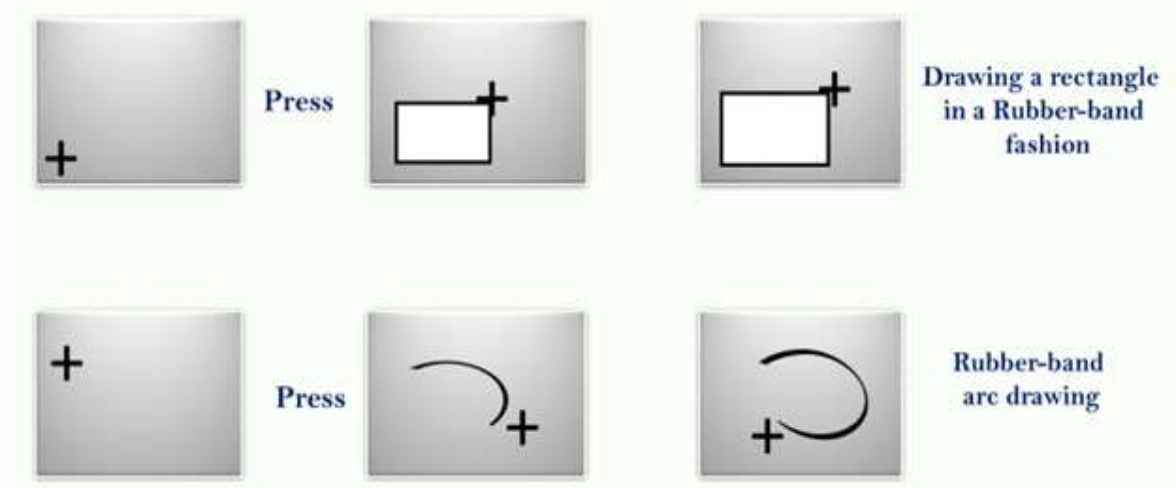

- When the user is happy with the final position, the pressed button is released, and the line is drawn between the start and the last position of the cursor.

**Example:** This is widely followed in MS-Window based Applications like in the case of a paintbrush drawing package.

Other geometric entities can be drawn in a rubber-band fashion:

- Horizontally or vertically constructed lines
- Rectangles
- Arcs of circles

This technique is very helpful in drawing relatively complex entities such as rectangles and arcs.




**Advantage:**

1. It is used for drawing all geometric entities such as line, polygon, circle, rectangle, ellipse, and other curves.
2. It is easy to understand and implement.

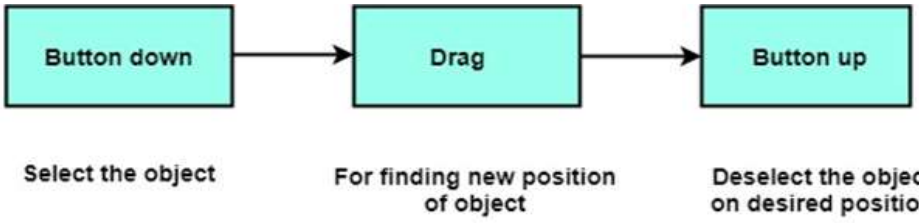
**Disadvantage:**

1. It requires computational resources like software and CPU speed.
2. Expensive

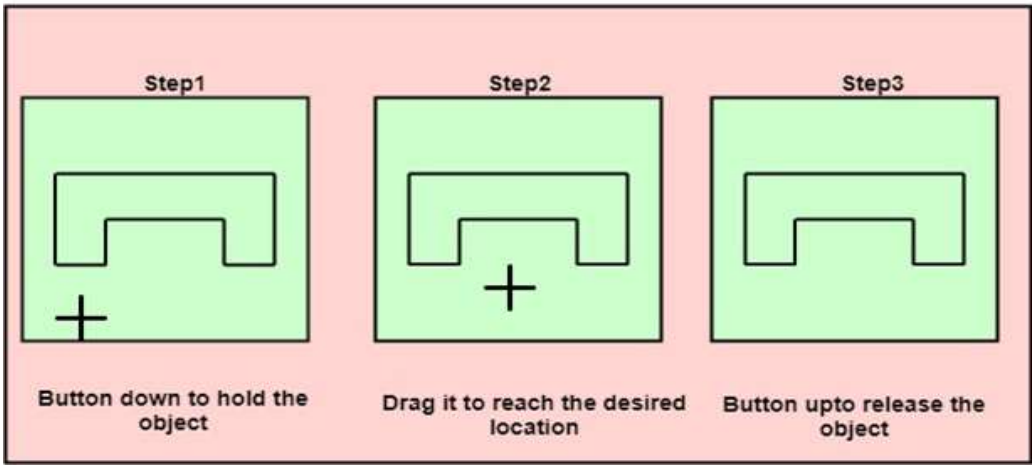
**Dragging:**

Dragging is used to move an object from one position to another position on the computer screen. To drag any other object, first, we have to select the object that we want to move on the screen by holding the mouse button down. As cursor moved on the screen, the object is also moved with the cursor position. When the cursor reached the desired position, the button is released.

**The following diagram represents the dragging procedure:**




Basic Diagram of Dragging




**Selection technique:** is a primary technique used in 3D interaction. Although many techniques has been implemented, few of them can support selection objects in dense area or under occlusion.

Selection = picking which object or objects a mouse click is on.

- Also, picking the objects inside a selection box made by dragging
- Easy to do in 2D - make correspondence between screen coordinates and world coordinates (show formulas)
- Hard to do in 3D:
    - 1 point on screen corresponds to many points in space
    - Converting drawn objects to screen coordinates & pixels done inside OpenGL - we don't have access to the screen coordinates when we draw it.
    - Hidden surfaces - do we want closest 3D object that is at this point?
    - We could do the math ourselves

**Selection by naming**: picking which object or objects by defining a name for it pr else by the name which has been defined already.

**Menu**: a **menu** is a list of options or commands presented to the user of a computer or communications system. A menu may either be a system's entire user interface, or only part of a more complex one.

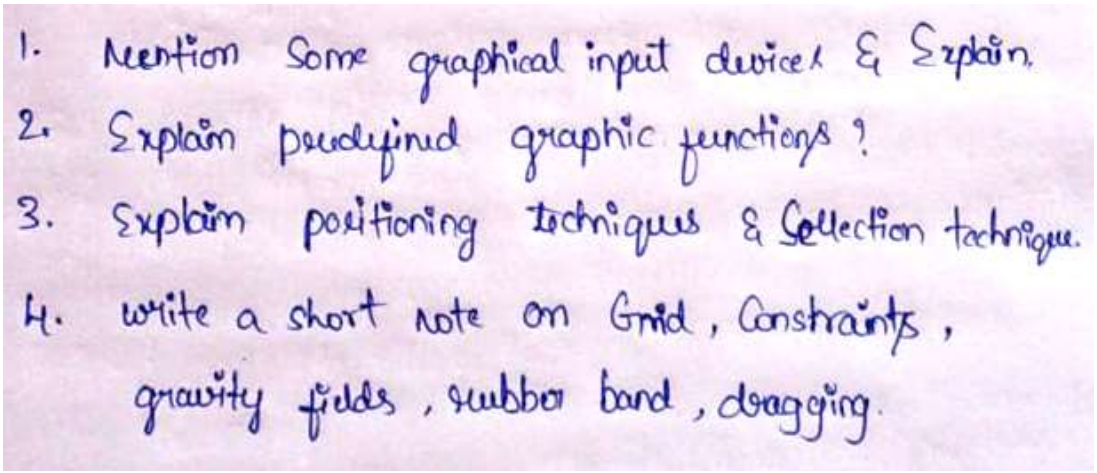
**Pointing and Positioning Techniques**

Pointing technique refers to look at the items already on the screen whereas the positioning technique refers to position the item on the screen to a new position, i.e., the old current position. The user indicates a position on the screen with an input device, and this position is used to insert a symbol.

There are various pointing and positioning devices which are discussed below:

1. Light Pen
2. Mouse
3. Tablet
4. Joystick
5. Trackball and spaceball

**Questions:**




**6thSem BCA**

**Paper: BCACsP6.8: COMPUTER GRAPHICS LAB PRACTICAL**

**3Hrs /Week** **50 Marks**

1. Write a program to draw a straight line using DDA technique.
2. Write a program to draw a straight line using Bresenham's technique.
3. Write a program to draw a circle using DDA technique.
4. Write a program to draw a Circle using Bresenham's technique.
5. Write a program to draw a triangle to perform translation
6. Write a program to draw a triangle to perform scaling
7. Write a program to draw a triangle to perform Rotation
8. Write a program to draw pie chart
9. Write a program to draw Histogram
10. Write a program to clip a triangle against a given window.
11. Write a program to animate a man walking with an umbrella.
12. Write a program to rotate an object from one end of the screen to the other end using the built-in line and circle functions.

**Q.P. Code – 18603**

## Sixth Semester B.C.A. Degree Examination, May/June 2017

*(New Scheme)*

**Computer Science**

## Paper BCA 603 – COMPUTER GRAPHICS

*Time : 3 Hours]* *[Max. Marks : 80*

*Instructions to Candidates : Answer all Sections.*

SECTION – A

I. Answer any **TEN** of the following questions. **(10 × 1 = 10)**

1. What is persistence?
2. What is resolution?
3. Define clipping.
4. What is Kerned Character?
5. What is Dragging?
6. What is the use of control points?
7. What are the basic functions of segments?
8. What are frame buffer?
9. What is intensity cueing?
10. What is Quad tree?
11. Give two examples of Touch Screen.
12. How track ball works?

SECTION – B

II. Answer any **FIVE** of the following. **(5 × 5 = 25)**

1. With neat diagram, explain the working of DVST.
2. Explain color models.
3. Explain window and view port.
4. Write Bresenham's circle drawing algorithm.
5. Explain the different types of line.
6. Explain 3 dimensional rotation.
7. Explain sweep representation.

## Q.P. Code – 18603

SECTION - C

III. Answer any **THREE** of the following questions. **(3 × 15 = 45)**

1. (a) Explain 2 dimensional basic and other transformation.
   (b) Write the difference between parallel and perspective projections. **(10 + 5)**

2. (a) Explain scan line algorithm for area filling.
   (b) Explain depth buffer algorithm.
   (c) Explain the different forms of Text Clipping. **(5 + 5 + 5)**

3. (a) What are polygon surface and polygon tables? Explain.
   (b) Explain Cohen-Sutherland line clipping algorithm. **(8 + 7)**

4. (a) What are the difference between Random Scan Monitors and Raster scan monitors?
   (b) Explain shadow mask CRT, with neat diagram.
   (c) Write the difference between LCD and LED monitors. **(5 + 5 + 5)**

5. (a) Explain segment attributes.
   (b) Explain grids, gravity field, rubber band method.
   (c) Explain any two pointing devices based on motion of an object. **(4 + 6 + 5)**

**Q.P. Code – 18603**

# Sixth Semester B.C.A. (N) Degree Examination, April/May 2018

*(New Scheme)*

## Computer Science

## Paper 603 — COMPUTER GRAPHICS

*Time : 3 Hours]* *[Max. Marks : 80*

*Instructions to Candidates : Answer ALL Sections.*

SECTION – A

I. Answer any **TEN** questions. **(10 × 1 = 10)**

1. Define Resolution.
2. Write the usage of frame buffer.
3. What do you mean by Emissive displays?
4. Mention any two requirements to draw a straight line.
5. Give one difference between Window & Viewport.
6. What do you mean by Uniform Scaling?
7. What is Exterior Clipping?
8. Name primary colors combination to get magenta color.
9. Define Voxel.
10. What are the basic functions of segment?
11. What do you mean by toggle key?
12. How selection by naming can be used in selection Techniques?

SECTION – B

II. Answer any **FIVE** questions. **(5 × 5 = 25)**

13. Write a note on beam penetration method.
14. Write the difference between CRT & LCD Monitors.
15. Draw a straight line using DDA line algorithm for given points (20, 10) & (28, 20).

**Q.P. Code – 18603**

16. Explain 3D – basic transformations.

17. Write a note on perspective projection.

18. What are octrees? How they are used to represent 3D-objects?

19. Explain working of trackball & space ball.

SECTION – C

III. Answer any **THREE** questions. **(3 × 15 = 45)**

20. (a) Explain computer graphics applications in detail.

    (b) Explain Raster scan display monitor with a neat diagram. **(8 + 7)**

21. (a) Write Bresanham's circle drawing algorithm.

    (b) Explain steps involved in fixed point scaling with example. **(8 + 7)**

22. (a) How is window to viewport transformation carried out.

    (b) Write Cohen-Sutherland algorithm for line clipping using region codes. **(7 + 8)**

23. (a) Explain Depth Buffer algorithm for hidden surface removal.

    (b) Write a note on Bezier curves.

    (c) Explain segment attributes. **(5 + 5 + 5)**

24. (a) Write a note on gravity field & rubber band method.

    (b) Explain operations under menu selection in selection techniques. **(8 + 7)**

Q.P. Code – 68604

# Sixth Semester B.C.A. Degree Examination, April/May 2019

*(CBCS Scheme)*

## Computer Science

## Paper 6.7 – COMPUTER GRAPHICS

*Time : 3 Hours]* *[Max. Marks : 90*

*Instructions to Candidates : ALL Sections are compulsory.*

SECTION – A

Answer any **TEN** questions. Each question carries **1** mark. **(10 × 1 = 10)**

1. What is refresh CRT?
2. Define ellipse.
3. What is blanking?
4. What is a viewport?
5. Define reflection.
6. What is composite transformation?
7. What do you mean by device coordinate system?
8. What is parallel projection?
9. What is the use of control points?
10. What is intensity cueing?
11. What is constraints?
12. Mention the combinational keys of keyboard.

**Q.P. Code – 68604**

SECTION – B

Answer any **FIVE** questions. Each question carries **3** marks. **(5 × 3 = 15)**

13. What are the difference between random and raster displays?

14. Write a program to draw a circle using DDA tech.

15. Explain point clipping.

16. Explain homogeneous transformation.

17. Explain uniform scaling transformation with an example.

18. Explain the properties of curves.

19. Explain basic functions of segments.

SECTION – C

Answer any **SIX** questions. Each question carries **5** marks. **(6 × 5 = 30)**

20. Briefly explain color model.

21. Write an algorithm to draw a straight line using Bresenham's tech and trace with 2 end points (20, 10) and (30, 18).

22. Briefly explain character attributes.

23. Explain Cohen and Sutherland line clipping algorithm.

24. (a) Write a note on Shear transformation

    (b) Explain fixed point scaling transformation. **(2 + 3)**

25. Explain 3D rotational transformation.

26. Write a note on polygon table.

27. (a) Explain rubber band method

    (b) Explain light pen. **(3 + 2)**

**Q.P. Code – 68604**

SECTION – D

Answer any **FIVE** questions. Each question carries 7 marks. **(5 × 7 = 35)**

28. Explain ellipse generating algorithm.

29. (a) Write a program to perform scaling transformation. Explain with suitable example.

(b) Explain properties of line. **(4 + 3)**

30. Explain Sutherland and Hodgeman polygon clipping.

31. (a) Explain window to viewport transformation carried out.

(b) Write a program to draw a bar chart. **(3 + 4)**

32. (a) Write a note on projections.

(b) Explain Octree. **(3 + 4)**

33. Explain Z buffer algorithm for hidden surface removal.

34. (a) Explain scan line method.

(b) Write a program to animate man walk with umbrella. **(3 + 4)**