Paper: Computer Graphics
Code: CS501
Contacts: 3L
Credits: 3
Total Lectures: 36

Objective(s)

- **Acquire a basic knowledge of computer graphics, input output devices and application of computer graphics**
- **Analyze different scan conversion algorithms**
- **Understand different Transformations techniques**
- **. Explain various clipping algorithms,3D projections and current models for surfaces**
- **Make use of the color and transformation techniques for various applications**
-

• Course Outcomes (CO) :

- **After successful completion of this course, the students should be able to**

CO1:	**. Acquire a basic knowledge of computer graphics, input output devices and application of computer graphics.** Understand Definition, different terms related to computer graphics. Understand different Hardware requirements for computer graphics. Understand different display units. Understand different color models Application of computer graphics.
CO2:	**Analyze different scan conversion algorithms.** Understand the concept of scanning. Design different drawing algorithms. Analyze different numerical problems. Understand polygon filling algorithms.
CO3:	**Understand different Transformations techniques.** Understand different basic transformations. Understand other transformations like reflection and shearing.

	Understands composite transformations. Understand window port, view port and window to view port transformations. Understand 3D transformations.
CO4:	. **Explain various clipping algorithms,3D projections and current models for surfaces** Understand and analyze different clipping algorithms. Understand 3D projections. Understand and design different curves. Understand and design different surfaces
CO5:	**Make use of the color and transformation techniques for various applications** Understand Light & color model. shading model and Texture. Understand Ray-tracing

Prerequisites:

1. **Basic co-ordinate geometry.**
2. **Matrix operation.**

Module-1: [6 L]

Introduction to computer graphics [3L]

Overview of computer graphics, Basic Terminologies in Graphics, direct coding, lookup table, 3D viewing devices, Plotters, printers, digitizers, Light pens etc.; Active & Passive graphics devices; Computer graphics software

Display [3L]

Light &Color models, Raster refresh displays, CRT basics, video basics, Flat panel displays, interpolative shading model; Texture

Module-2: [8 L]

Scan conversion: [8L]

Points & lines, Line drawing algorithms; DDA algorithm, Bresenham's line algorithm, Circle generation algorithm

Ellipse generating algorithm; scan line polygon, fill algorithm, boundary fill algorithm, flood fill algorithm

Module-3: [16 L]

2D and 3D Transformation [12L]

Basic transformations: translation, rotation, scaling ; Matrix representations & homogeneous coordinates, transformations between coordinate systems; reflection shear; Transformation of points, lines, parallel lines, intersecting lines

3D transformations: translation, rotation, scaling & other transformations. Rotation about an arbitrary axis in space, reflection through an arbitrary plane; general parallel projection transformation

2D and 3D Viewing & Clipping [4L]

Viewing pipeline, Window to viewport co-ordinate transformation, clipping operations, point clipping, line clipping, clipping circles, polygons & ellipse. Viewport clipping, 3D viewing.

Module-4: [6 L]

Curves [3L]

Curve representation, surfaces, designs, Bezier curves, B-spline curves, end conditions for periodic B-spline curves, rational B-spline curves

Hidden Surface Removal [3L]

Depth comparison, Z-buffer algorithm, Back face detection, BSP tree method, the Printer's algorithm, scan-line algorithm; Hidden line elimination, wire frame methods , fractal - geometry

MODULE 1:

Introduction to Computer Graphics.

LECTURE1:

Overview of computer graphics:

Computer graphics is an art of drawing pictures on computer screens with the help of programming. It involves computations, creation, and manipulation of data. In other words, we can say that computer graphics is a rendering tool for the generation and manipulation of images.

Basically Computer Graphics= Data structure + Graphics algorithm + Languages

1.1. Basic Terminologies in Graphics:

a) PIXEL:

Pixel or picture element is defined as the smallest addressable screen element .So it is the smallest unit of a display unit .It is one spot in a rectilinear grid of thousands of such spots that are painted to form an image .Each pixel has its own intensity, name or address .It is a measure of screen resolution.

Each pixel there are three sub-pixels, to produce different colors .The color of sub-pixels are red ,blue ,green(RGB).

b) **FRAME BUFFER:**

It is a large contigious piece of memory into which intensity values for all pixels are placed. It is an arrary and graphical display devices can access this array.

c) **BITMAP, PIXEL MAP AND APIXMAP:**

An image of two colors is called as a bitmap.On a black-white system with one bit per pixel , the frame buffer is called as a bitmap.

An image of more than two colors is called a pixel map.

For systems with multiple bits per pixel , the frame buffer is called a apixmap.

d) **DOT PITCH**:

It is defined as the measurement of the diagonal distance between two liked-colored pixels on a display screen.

e) RESOLUTION:

In computers, **resolution** is the number of pixels (individual points of color) contained on a display monitor, expressed in terms of the number of pixels on the horizontal axis and the number on the vertical axis. The sharpness of the image on a display depends on the **resolution** and the size of the monitor.

f) ASPECT RATIO:

In **computer graphics**,the relative horizontal and vertical sizes. For example, if a**graphic** has an **aspect ratio** of 2:1, it means that the width is twice as large as the height. When resizing **graphics**, it is important to maintain the **aspect ratio** to avoid stretching the **graphic** out of proportion.

NUMERICAL PROBLEMS:

1) **Suppose a system using a 8 inch X 8 inch screen with a resolution of 100 pixels per inch in each direction.If we want to store 6 bits per pixels in frame buffer then what is the size of frame buffer? Also find out the aspect ratio.**

SOLUTION: The resolution in pixels is given by-
=(8X100) x (10X100)
=800X1000
Frame buffer size-
=800x1000x6 bits
$=6 \times 10^{5\text{Bytes}}$
Aspect ratio=800/1000=4:5

1.2. CHARACTERISTICS OF COMPUTER GRAPHICS:

a) It is interactive in nature.

b) It should be user friendly.

c) It provides a tool called as motion dynamics which allows users to move objects.

d) It provides an update dynamics facility which allows users to change the shape , color , and other properties of objects.

e) It has the ability to show the moving pictures (animations).

1.3. COMPONENTS OF COMPUTER GRAPHICS:

a) **Software level**: Involves three components –
   i) Application program.
   ii) Application model

iii) Graphics system.

b) **Hardware level**:
   i) Interactive devices
   ii) Display devices.

Typical graphics system comprices of a host computer with support of fast processor , large memory , frame buffer etc.

LECTURE 2:

1.4.3D viewing devices.

a) **PLOTTER**:
The **plotter** is a **computer** printer for printing vector **graphics**. In the past, **plotters** were used in applications such as **computer**-aided design, though they have generally been replaced with wide-format conventional printers. A **plotter** gives a hard copy of the output. It draws pictures on a paper using a pen.

b) **PRINTER:**
A **printer** is a device that accepts text and **graphic** output from a **computer** and transfers the information to paper, usually to standard size sheets of paper.Early impact **printers** worked something like an automatic typewriter, with a key striking an inked impression on paper for each printed character.

c) **DIGITIZER:**
A **graphic** tablet (also known as a digitizer, drawing tablet, 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.

d) **LIGHT PEN:**
A **light pen** is a **computer** input device in the form of a **light**-sensitive wand used in conjunction with a **computer's** CRT display. It allows the user to point to **displayed objects or draw on the screen in a similar way to a touchscreen but with** greater positional accuracy.

1.5. ACTIVE &PASSIVE COMPUTER GRAPHICS SYSTEM.

In case of non –interactive or passive computer graphics, the user has no control over picture. For example a staticwebsite , a TV system etc .

In case of active systems , the user controls the display with the help of a GUI ,using an input device. For examole videogames.

1.6.APPLICATION OF COMPUTER GRAPHICS:

Typical application areas are-

a) GUI.
b) Desktop publishing
c) Plotting in business.
d) Plotting in science and technology.
e) Advertisements.
f) Entertainment(movie ,TV , games)
g) Medical field.
h) Multimedia.

LECTURE 3:

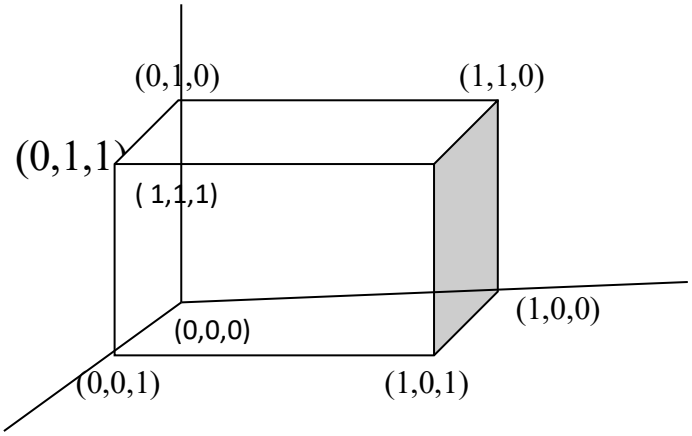
1.7.Color Models.

a) RGB Color Model-

In this model the primary colors are Red ,Green and Blue.It is an additive model , in which adding components produces colors , with white having all colors and being the absence of any color.This is the model used for active displays ,such as television and computer screen.

The RGB model is usually represented by a unit cube with one corner located at the origin of the three dimensional color co-ordinate system.The origin(0,0,0) is considered as black and the diagonally opposite corner (1,1,1) is called white.The line joining black to white represents a gray scale and has equql components of R,G,B.

Vertices of the cube on the axes represent the primary colors i,e; red,green and blue and remaining vertices represent the complementary color for each of the primary colors.The primary color intensities are added to produce other color.



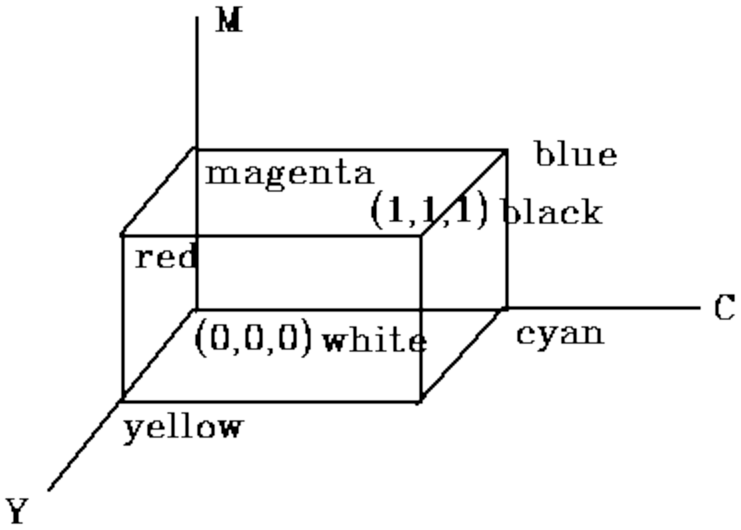
The magenta vertex is obtained by adding red and blue vertices.
Similarlycyan by adding green and blu :

.

b) CMY Color Model-

In this model Cyan ,Magenta and Yellow are the primary colors and red ,green and blue are the secondary colors. This is subtractive model. This model is useful for describing color output to hard copy devices.

Here origin (0,0,0) represents white , whereas point (1,1,1) represents black.



Remember that cyan = green + blue, so light reflected from a cyan pigment has no red component, i.e., the red is absorbed by cyan. Similarly magenta subtracts green and yellow subtracts blue. Printers usually use four colors: cyan, yellow, magenta and black. This is because cyan, yellow, and magenta together produce a dark gray rather than a true black.

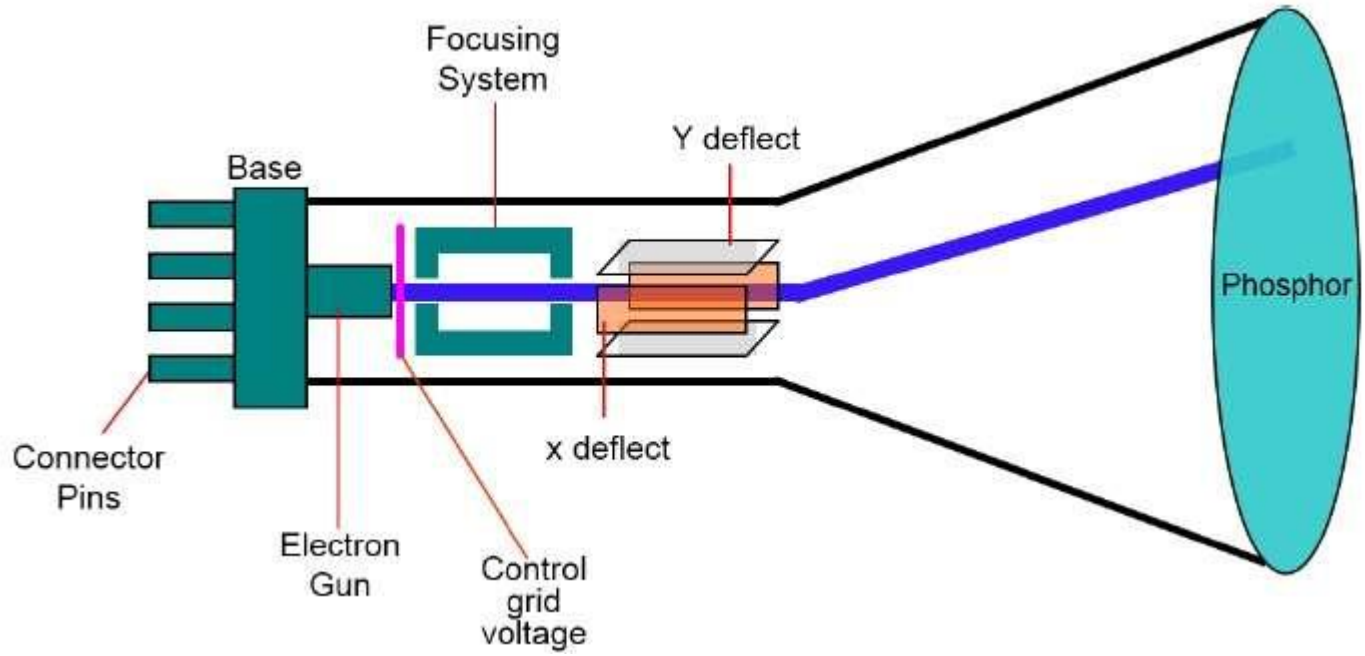
LECTURE 4:

1.8. CRT BASICS:

.The primary output device in a graphical system is the video monitor. The main element of a video monitor is the **Cathode Ray Tube (CRT),** shown in the following illustration.

The operation of CRT is very simple −

- The electron gun emits a beam of electrons (cathode rays).
- The electron beam passes through focusing and deflection systems that direct it towards specified positions on the phosphor-coated screen.
- When the beam hits the screen, the phosphor emits a small spot of light at each position contacted by the electron beam.
- It redraws the picture by directing the electron beam back over the same screen points quickly.



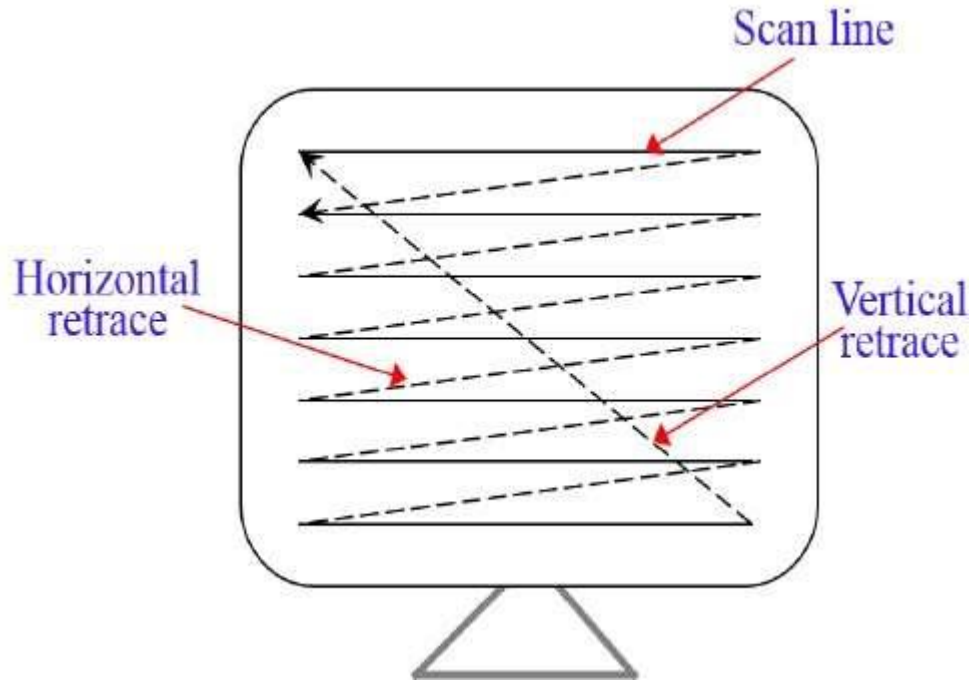
There are two ways (Random scan and Raster scan) by which we can display an object on the screen.

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.

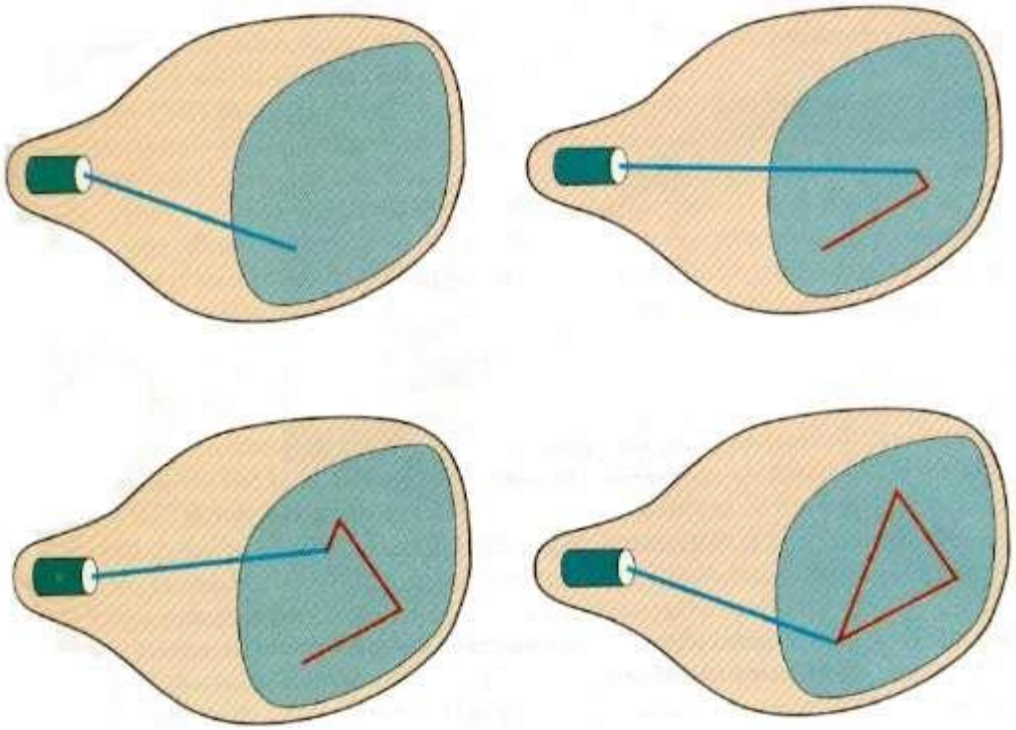


Random Scan (Vector Scan)

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,** or **calligraphic display**.

Picture definition is stored as a set of line-drawing commands in an area of memory referred to as the **refresh display file**. To display a specified picture, the system cycles through the set of commands in the display file, drawing each component line in turn. After all the line-drawing commands are processed, the system cycles back to the first line command in the list.

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



LECTURE 5:

1.9. Interpolative Shading Model:

The idea of interpolative shading is to avoid computing the full lighting equation at each pixel by interpolating quantites at the vertices of the faces

.

There are two methods used for interpolative shading:

Gouraud Shading:

The radiance values are computed at the vertices and then linearly interpolated within each triangle.

PhongShading:

The normal values at each vertex are linearly interpolated within each triangle, and the radiance is computed at each pixel.

MODULE 2
Scan conversion

LECTURE 6:

Introduction:

On our raster systems, we can generate images by turning the pixels ON or OFF. So what we want is to represent an object which is represented as a collection of discrete pixels.This process of conversion of the rasterized picture stored in a frame buffer to the rigid display pattern of video is called a scan conversion. It is a continuous to discrete transformation.

2.1. Point & line:

We can specify a point with an ordered pair of numbers(x,y) where x is the horizontal distance from origin and y is the vertical distance.

To specify a line we need two such points (x1,y1) and (x2,y2). The equation for the line is-

y-y1/x-x1=y2-y1/x2-x1.

Analternate form of the straight line is-
Y=mx+c
Where m is the slope of the line.
m= y2-y1/x2-x
and c may be found as:
y1-mx1 or y2-mx2

2.2. Vector Generation:

The process of turning on the pixels for a line segment is called vector generation.Each pixel has certain characteristics and they are listed below-

a) Each pixel is accessed by positive co –ordinates only.

b) The floating point value of pixel co-ordinates are rounded off to its nearest integer.

c) Each pixel has a particular color and intensity values.

2.3.Line Drawing Algorithms:

Before vgoing further to the specific line drawing algorithm , let us discuss some characteristics of a good line drawing algorithms-

a) Line should appear straight.

b) Should have constant density.

c) Should start and terminate accurately.

Most line drawing algorithms use incremental methods. In this method , a line starts with the starting point , then a fixed increment is added to the current point on the line. This is continued till the end of the line or end point.

a) DDA algorithm:

A line connects two points. It is a basic element in graphics. To draw a line, we need two points between which we can draw a line.We refer the one point of line as(X_0,Y_0) and the second point of line as (X_1,Y_1)

Digital Differential Analyzer (DDA) algorithm is the simple line generation algorithm which is explained step by step here.

Step 1 − Get the input of two end points (X_0,Y_0) and (X_1,Y_1).

Step 2 − Calculate the difference between two end points.

m= Y1-Y0/X1-X0
Step 3 – If m<1
Then $(x_{k+1},y_{k+1}) =(x_k +1 ,y_k +m)$

Step 4 – If m>1
Then $(x_{k+1},y_{k+1}) =(x_k +1/m ,y_k +1)$

Step 5 – If m=1
Then $(x_{k+1},y_{k+1}) =(x_k +1 ,y_k +1)$

LECTURE 7:

Problem on DDA algorithm:

Implement the DDA algorithm to draw a line from (0,0) to (4,5).

Solution:

Here starting point (0,0)

Ending point (4,5)

So m= 5-0/4-0 = 5/4 >1

So we have to use-

– If m>1

Then $(x_{k+1},y_{k+1}) = (x_k + 1/m , y_k + 1)$

x	y	x-plot	y-plot	(x,y)
0	**0**	**0**	**0**	**(0,0)**
0.8	**1**	**1**	**1**	**(1,1)**
1.6	**2**	**2**	**2**	**(2,2)**
2.4	**3**	**2**	**3**	**(2,3)**
3.2	**4**	**3**	**3**	**(3,4)**
4	**5**	**4**	**4**	**(4,5)**

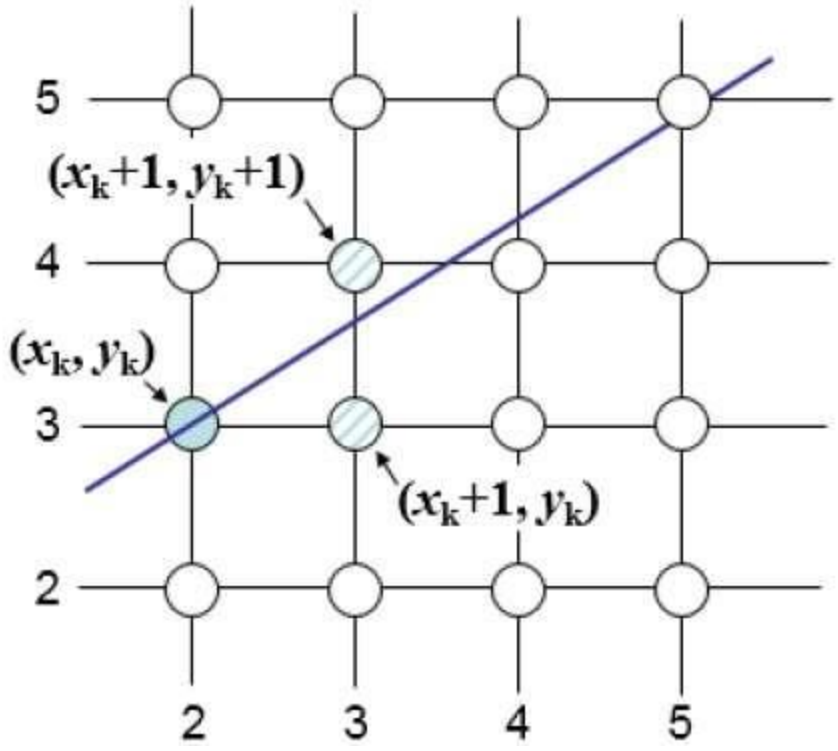
So the intermediate points are (0,0) , (1,1) , (2,2) , (2,3) , (3,4) , (4,5)

DDA algorithm is very simple. But we seen that every iteration we are using round function.So in order to avoid such round function we are moving to the another line drawing algorithm called Bresenham line drawing algo.

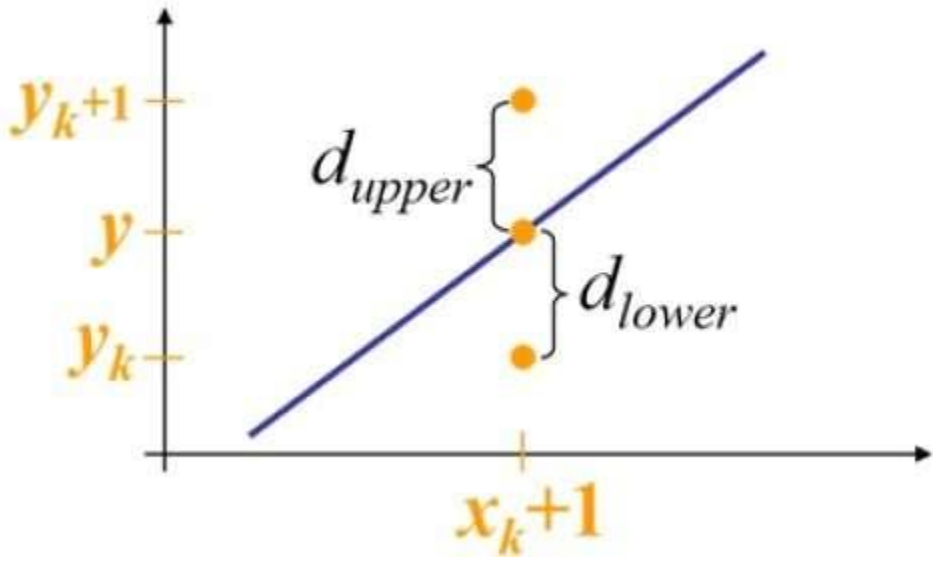
b) Bresenham's Line Generation:

The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates.

For example, as shown in the following illustration, from position (2, 3) you need to choose between (3, 3) and (3, 4). You would like the point that is closer to the original line.



At sample position Xk+1,Xk+1, the vertical separations from the mathematical line are labelled as dupperdupper and dlowerdlower.



From the above illustration, the y coordinate on the mathematical line at xk+1xk+1 is −

Y = m(XkXk+1) + b

So, dupperdupper and dlowerdlower are given as follows −

$$Dlower=y-ykdlower=y-yk$$

$$=m(Xk+1)+b-Yk=m(Xk+1)+b-Yk$$

and

$$dupper=(yk+1)-ydupper=(yk+1)-y$$

$$=Yk+1-m(Xk+1)-b=Yk+1-m(Xk+1)-b$$

You can use these to make a simple decision about which pixel is closer to the mathematical line. This simple decision is based on the difference between the two pixel positions.

$$dlower-dupper=2m(xk+1)-2yk+2b-1dlower-dupper=2m(xk+1)-2yk+2b-1$$

Let us substitute *m* with *dy/dx* where *dx* and *dy* are the differences between the end-points.

$$dx(dlower-dupper)=dx(2dydx(xk+1)-2yk+2b-1)dx(dlower-dupper)=dx(2dydx(xk+1)-2yk+2b-1)$$

$$=2dy.xk-2dx.yk+2dy+2dx(2b-1)=2dy.xk-2dx.yk+2dy+2dx(2b-1)$$

$$=2dy.xk-2dx.yk+C=2dy.xk-2dx.yk+C$$

So, a decision parameter Pk for the *k*th step along a line is given by −

$$pk=dx(dlower-dupper)pk=dx(dlower-dupper)$$

$$=2dy.xk-2dx.yk+C=2dy.xk-2dx.yk+C$$

The sign of the decision parameter Pk is the same as that of dlower−dupperdlower−dupper.

If pk is negative, then choose the lower pixel, otherwise choose the upper pixel.

Remember, the coordinate changes occur along the x axis in unit steps, so you can do everything with integer calculations. At step k+1, the decision parameter is given as −

$$pk+1=2dy.xk+1-2dx.yk+1+Cpk+1=2dy.xk+1-2dx.yk+1+C$$

Subtracting pk from this we get −

$$pk+1-pk=2dy(xk+1-xk)-2dx(yk+1-yk)pk+1-pk=2dy(xk+1-xk)-2dx(yk+1-yk)$$

But, xk+1xk+1 is the same as (xk)+1(xk)+1. So −

$$pk+1=pk+2dy-2dx(yk+1-yk)pk+1=pk+2dy-2dx(yk+1-yk)$$

Where, Yk+1–YkYk+1–Yk is either 0 or 1 depending on the sign of Pk.

The first decision parameter p0p0 is evaluated at (x0,y0)(x0,y0) is given as −

$$p0=2dy−dxp0=2dy−dx$$

Now, keeping in mind all the above points and calculations, here is the Bresenham algorithm for slope m < 1 −

Step 1 − Input the two end-points of line, storing the left end-point in (x0,y0).

Step 2 − Plot the point (x0,y0).

Step 3 − Calculate the constants dx, dy, 2dy, and (2dy – 2dx) and get the first value for the decision parameter as −

$$p0=2dy−dxp0=2dy−dx$$

Step 4 − At each Xk along the line, starting at k = 0, perform the following test −

If pk < 0, the next point to plot is (xk+1,yk)(xk+1,yk) and

$$pk+1=pk+2dypk+1=pk+2dy$$

Otherwise,

$$(xk,yk+1)(xk,yk+1)$$

$$pk+1=pk+2dy−2dxpk+1=pk+2dy−2dx$$

Step 5 − Repeat step 4 (dx – 1) times.

For m > 1, find out whether you need to increment x while incrementing y each time.

After solving, the equation for decision parameter Pk will be very similar, just the x and y in the equation gets interchanged

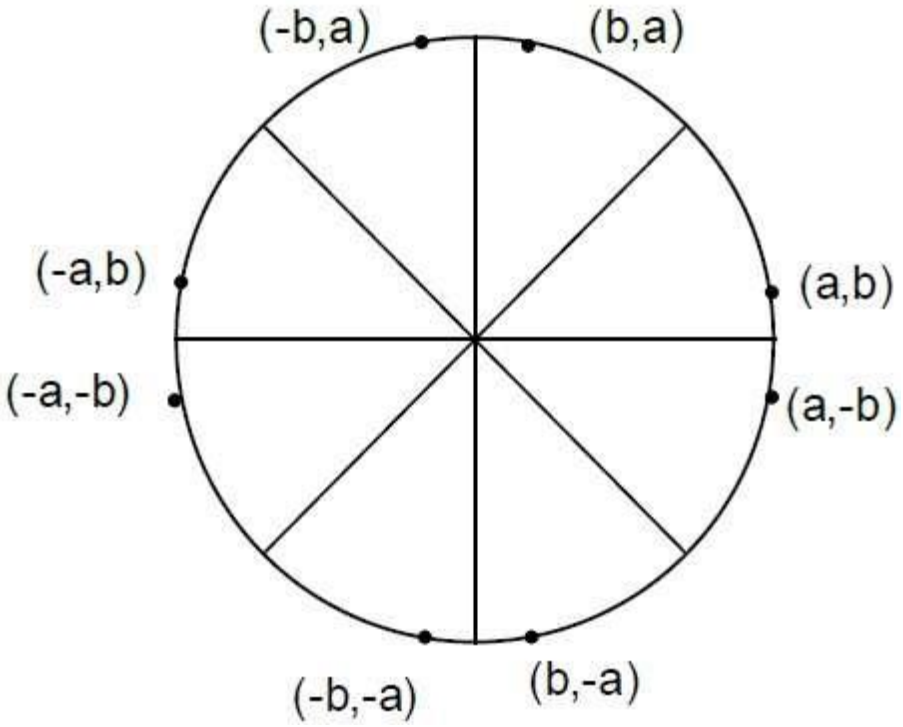
.

LECTURE 8:

2.4. Circle generation algorithm

Drawing a circle on the screen is a little complex than drawing a line. There are two popular algorithms for generating a circle − **Bresenham'sAlgorithm**and **Midpoint Circle Algorithm**. These algorithms are based on the idea of determining the subsequent points required to draw the circle. Let us discuss the algorithms in detail −

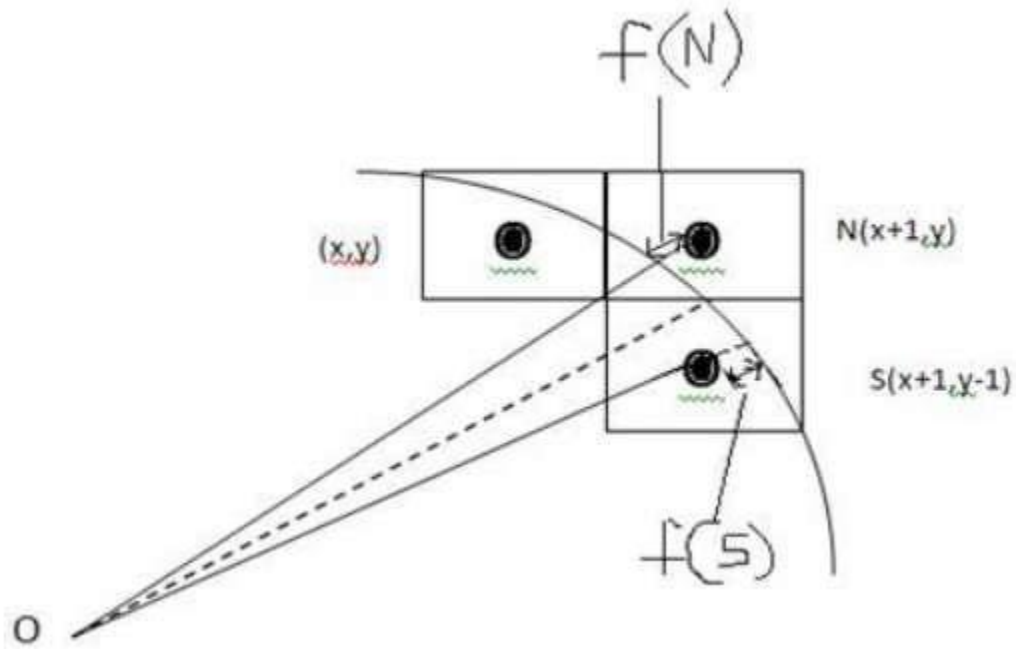
The equation of circle is $X2+Y2=r2$,X2+Y2=r2, where r is radius.



a) **Bresenham's Algorithm**:

We cannot display a continuous arc on the raster display. Instead, we have to choose the nearest pixel position to complete the arc.

From the following illustration, you can see that we have put the pixel at (X, Y) location and now need to decide where to put the next pixel − at N (X+1, Y) or at S (X+1, Y-1).



This can be decided by the decision parameter **d**.

- If d <= 0, then N(X+1, Y) is to be chosen as next pixel.
- If d > 0, then S(X+1, Y-1) is to be chosen as the next pixel.

Algorithm

Step 1 − Get the coordinates of the center of the circle and radius, and store them in x, y, and R respectively. Set P=0 and Q=R.

Step 2 − Set decision parameter D = 3 – 2R.

Step 3 − Repeat through step-8 while P ≤ Q.

Step 4 − Call Draw Circle (X, Y, P, Q).

Step 5 − Increment the value of P.

Step 6 − If D < 0 then D = D + 4P + 6.

Step 7 − Else Set R = R - 1, D = D + 4(P-Q) + 10.

Step 8 − Call Draw Circle (X, Y, P, Q).

```
Draw Circle Method(X, Y, P, Q).

Call Putpixel (X + P, Y + Q).
Call Putpixel (X - P, Y + Q).
Call Putpixel (X + P, Y - Q).
Call Putpixel (X - P, Y - Q).
Call Putpixel (X + Q, Y + P).
Call Putpixel (X - Q, Y + P).
Call Putpixel (X + Q, Y - P).
Call Putpixel (X - Q, Y - P).
```

LECTURE 9:

b) Mid Point Algorithm :

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

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

Step 2 − Calculate the initial value of decision parameter as

$P_0P0 = 5/4 – r$ (See the following description for simplification of this equation.)

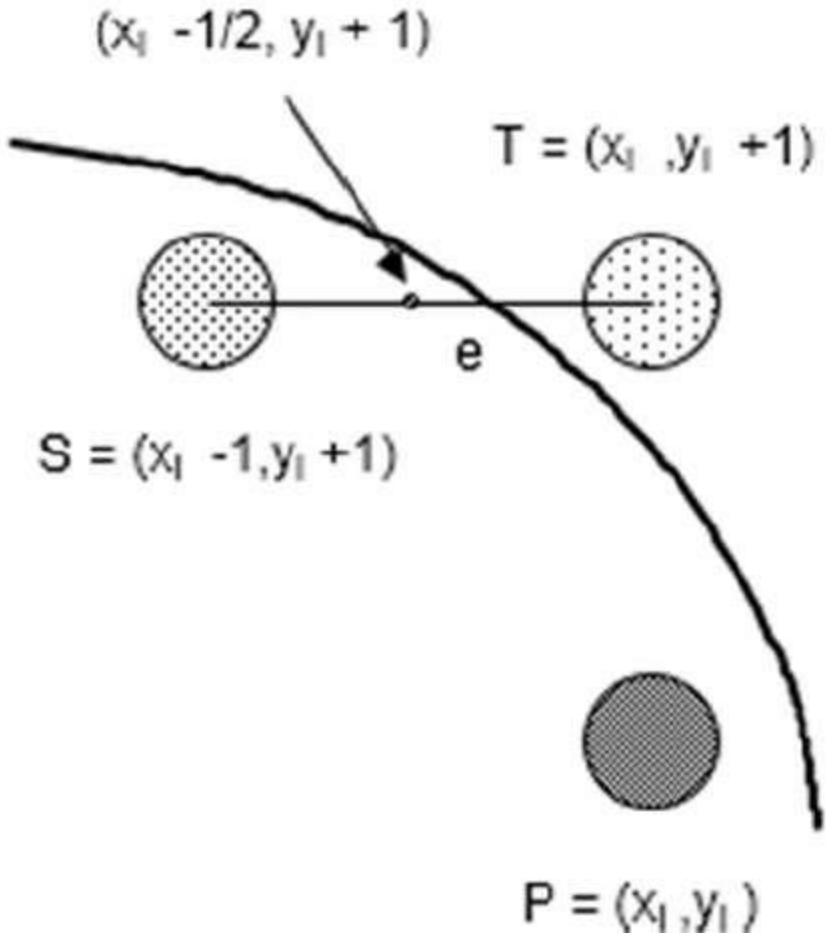
$$f(x, y)= x^2+ y^2- r^2=0$$

$$f(x_i-1/2+ e, y_i+1)$$

$$=(x_i-1/2+ e)^2+(y_i+1)^2- r^2$$

$$=(x_i-1/2)^2+(y_i+1)^2- r^2+2(x_i-1/2)e+ e^2$$

$$=f(x_i-1/2,y_i+1)+2(x_i-1/2)e + e^2=0$$



Let $d_i=f(x_i-1/2,y_i+1)=-2(x_i-1/2)e - e^2$

Thus,

If e <0then di >0 so choose point S =(x_i-1,y_i+1).

$d_{i+1}= f(x_i-1-1/2,y_i+1+1)=((x_i-1/2)-1)^2+((y_i+1)+1)^2- r^2$

$= d_i-2(x_i-1)+2(y_i+1)+1$

$= d_i+2(y_{i+1}- x_{i+1})+1$

If e >=0then di <=0 so choose point T =(x_i,y_i+1)

$d_{i+1}= f(x_i-1/2,y_i+1+1)$

$= d_i+2y_{i+1}+1$

The initial value of di is

$d_0 = f(r - 1/2, 0+1) = (r - 1/2)^2 + 1^2 - r^2$

$= 5/4 - r$ {1-r can be used if r is an integer}

When point S $= (x_i - 1, y_i + 1)$ is chosen then

$d_{i+1} = d_i + -2x_{i+1} + 2y_{i+1} + 1$

When point T $= (x_i, y_i + 1)$ is chosen then

$d_{i+1} = d_i + 2y_{i+1} + 1$

Step 3 − At each X_K position starting at K=0, perform the following test –

If $P_K < 0$ then next point on circle (0,0) is (X_{K+1}, Y_K) and

$P_{K+1} = P_K + 2X_{K+1} + 1$

Else

$P_{K+1} = P_K + 2X_{K+1} + 1 - 2Y_{K+1}$

Where, $2X_{K+1} = 2X_{K+2}$ and $2Y_{K+1} = 2Y_{K-2}$.

Step 4 − Determine the symmetry points in other seven octants.

Step 5 − Move each calculate pixel position (X, Y) onto the circular path centered on (X_C, Y_C) and plot the coordinate values.

$X = X + X_C, \quad Y = Y + Y_C$

Step 6 − Repeat step-3 through 5 until X >= Y.

LECTURE 10:

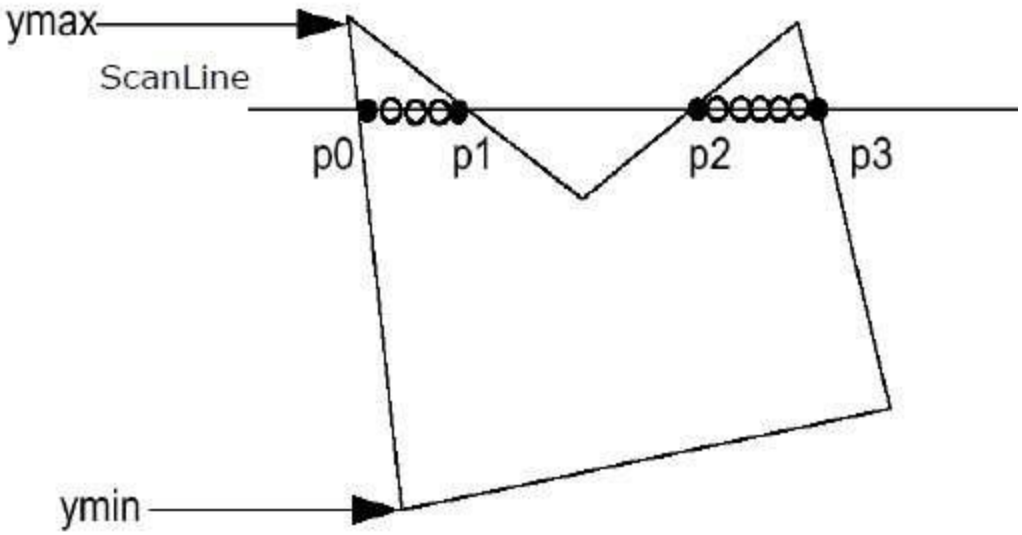
2.5. Polygon Filling Algorithm:

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. In this chapter, we will see how we can fill polygons using different techniques.

a) 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.

b) Flood Fill Algorithm:

Sometimes we come across an object where we want to fill the area and its boundary with different colors. We can paint such objects with a specified interior color instead of searching for particular boundary color as in boundary filling algorithm.

Instead of relying on the boundary of the object, it relies on the fill color. In other words, it replaces the interior color of the object with the fill color. When no more pixels of the original interior color exist, the algorithm is completed.

Once again, this algorithm relies on the Four-connect or Eight-connect method of filling in the pixels. But instead of looking for the boundary color, it is looking for all adjacent pixels that are a part of the interior.

LECTURE11.

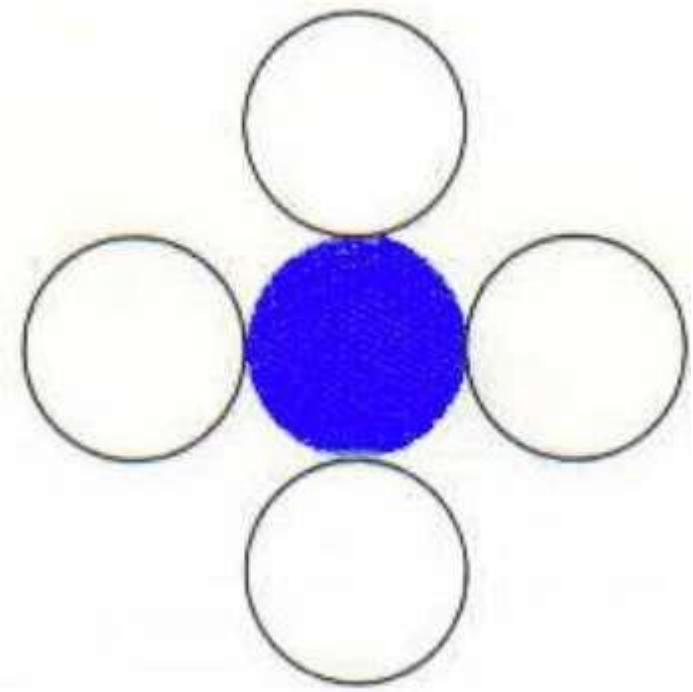
2.6. Boundary Fill Algorithm

The boundary fill algorithm works as its name. This algorithm picks a point inside an object and starts to fill until it hits the boundary of the object. The color of the boundary and the color that we fill should be different for this algorithm to work.

In this algorithm, we assume that color of the boundary is same for the entire object. The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

4-Connected Polygon:

In this technique 4-connected pixels are used as shown in the figure. We are putting the pixels above, below, to the right, and to the left side of the current pixels and this process will continue until we find a boundary with different color.



Algorithm

Step 1 − Initialize the value of seed point (seedx, seedy), fcolor and dcol.

Step 2 − Define the boundary values of the polygon.

Step 3 − Check if the current seed point is of default color, then repeat the steps 4 and 5 till the boundary pixels reached.

```
Ifgetpixel(x, y)=dcolthen repeat step 4and5
```

Step 4 − Change the default color with the fill color at the seed point.

```
setPixel(seedx, seedy, fcol)
```

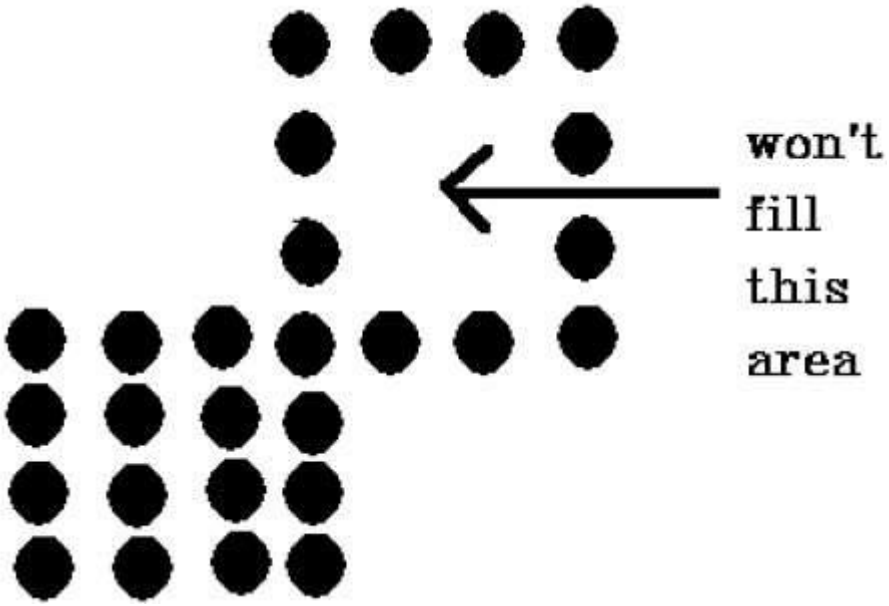
Step 5 − Recursively follow the procedure with four neighborhood points.

```
FloodFill (seedx – 1, seedy, fcol, dcol)
FloodFill (seedx + 1, seedy, fcol, dcol)
FloodFill (seedx, seedy - 1, fcol, dcol)
```

```
FloodFill (seedx – 1, seedy + 1, fcol, dcol)
```

Step 6 − Exit

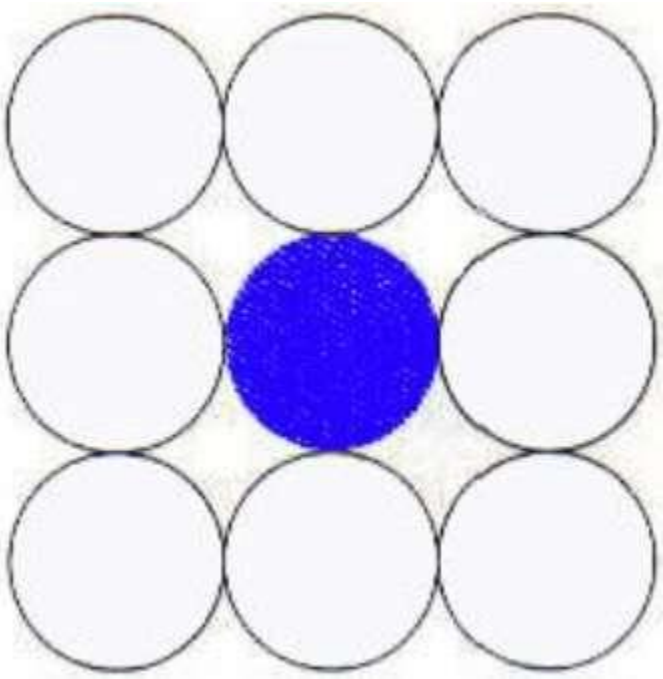
There is a problem with this technique. Consider the case as shown below where we tried to fill the entire region. Here, the image is filled only partially. In such cases, 4-connected pixels technique cannot be used.



8-Connected Polygon

In this technique 8-connected pixels are used as shown in the figure. We are putting pixels above, below, right and left side of the current pixels as we were doing in 4-connected technique.

In addition to this, we are also putting pixels in diagonals so that entire area of the current pixel is covered. This process will continue until we find a boundary with different color.



Algorithm

Step 1 − Initialize the value of seed point (seedx, seedy), fcolor and dcol.

Step 2 − Define the boundary values of the polygon.

Step 3 − Check if the current seed point is of default color then repeat the steps 4 and 5 till the boundary pixels reached

```
If getpixel(x,y) = dcol then repeat step 4 and 5
```

Step 4 − Change the default color with the fill color at the seed point.

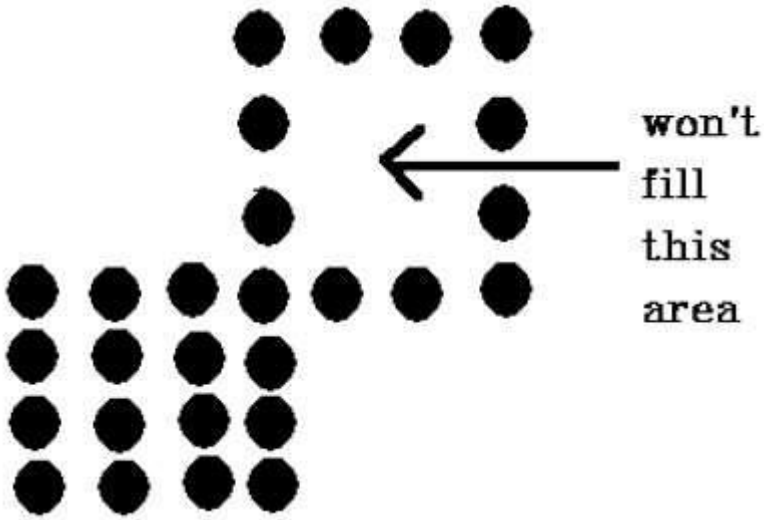
```
setPixel(seedx, seedy, fcol)
```

Step 5 − Recursively follow the procedure with four neighbourhood points

```
FloodFill (seedx – 1, seedy, fcol, dcol)
FloodFill (seedx + 1, seedy, fcol, dcol)
FloodFill (seedx, seedy - 1, fcol, dcol)
FloodFill (seedx, seedy + 1, fcol, dcol)
FloodFill (seedx – 1, seedy + 1, fcol, dcol)
FloodFill (seedx + 1, seedy + 1, fcol, dcol)
FloodFill (seedx + 1, seedy - 1, fcol, dcol)
FloodFill (seedx – 1, seedy - 1, fcol, dcol)
```

Step 6 − Exit

The 4-connected pixel technique failed to fill the area as marked in the following figure which won't happen with the 8-connected technique.



LECTURE12

Inside-outside Test

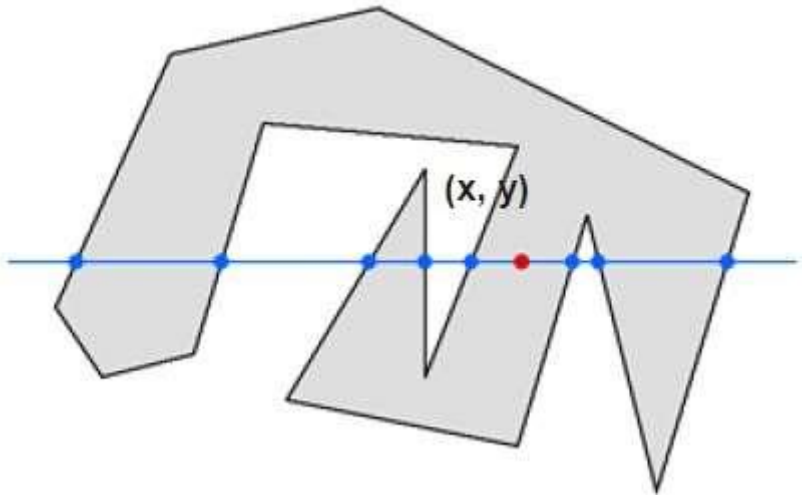
This method is also known as **counting number method**. While filling an object, we often need to identify whether particular point is inside the object or outside it. There are two methods by which we can identify whether particular point is inside an object or outside.

- Odd-Even Rule
- Nonzero winding number rule

Odd-Even Rule

In this technique, we will count the edge crossing along the line from any point (x,y) to infinity. If the number of interactions is odd, then the point (x,y) is an interior point; and if the number of

interactions is even, then the point (x,y) is an exterior point. The following example depicts this concept.

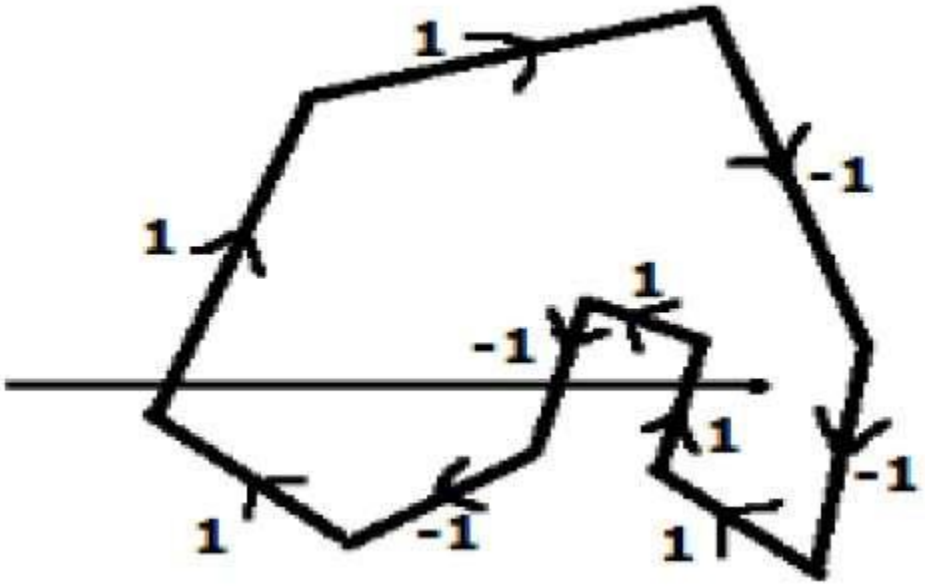


From the above figure, we can see that from the point (x,y), the number of interactions point on the left side is 5 and on the right side is 3. From both ends, the number of interaction points is odd, so the point is considered within the object.

Nonzero Winding Number Rule

This method is also used with the simple polygons to test the given point is interior or not. It can be simply understood with the help of a pin and a rubber band. Fix up the pin on one of the edge of the polygon and tie-up the rubber band in it and then stretch the rubber band along the edges of the polygon.

When all the edges of the polygon are covered by the rubber band, check out the pin which has been fixed up at the point to be test. If we find at least one wind at the point consider it within the polygon, else we can say that the point is not inside the polygon.



In another alternative method, give directions to all the edges of the polygon. Draw a scan line from the point to be test towards the left most of X direction.

- Give the value 1 to all the edges which are going to upward direction and all other -1 as direction values.
- Check the edge direction values from which the scan line is passing and sum up them.
- If the total sum of this direction value is non-zero, then this point to be tested is an **interior point,** otherwise it is an **exterior point**.
- In the above figure, we sum up the direction values from which the scan line is passing then the total is 1 – 1 + 1 = 1; which is non-zero. So the point is said to be an interior point.

MODULE 3

2D and 3D Transformation.

Lecture 13

Introduction: The process of changing the size, position or orientations of objects by a mathematical operation is called as transformation**.This process is accomplished by two methods-**

a) **Geometric transformation:** In this case an object itself is transformed relative to stationary co –ordinate system**.**

b) **Co-ordinate transformation:** In this case an object is held stationary while the co-ordinate system is transformed relative to the object.

3.1: Point or Object Representation:

In a 2D co-ordinate system , any point is represented as (x,y). This point can be converted into the matrix form in two ways-

a) Row-major: (x,y)=$[x\ y\ ]$

b) Column-major: (x,y)= $\begin{bmatrix} x \\ y \end{bmatrix}$

These matrices are called as position vectors. Now for example , say we want to represent a rectangle in matrix form.If (x1,y1) and (x2,y2) are the opposite vertices of a rectangle its matrix representation will be-

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} X1 & Y1 \\ X2 & Y1 \\ X2 & Y2 \\ X1 & Y2 \end{bmatrix}$$

To perform transformation on any object, the original object matrix [x] is multiplied by the transformation matrix [T] to get a new transformed object matrix i.e [x*] = [x][T].

3.2; Basic 2D transformation:

The basic transformations are-

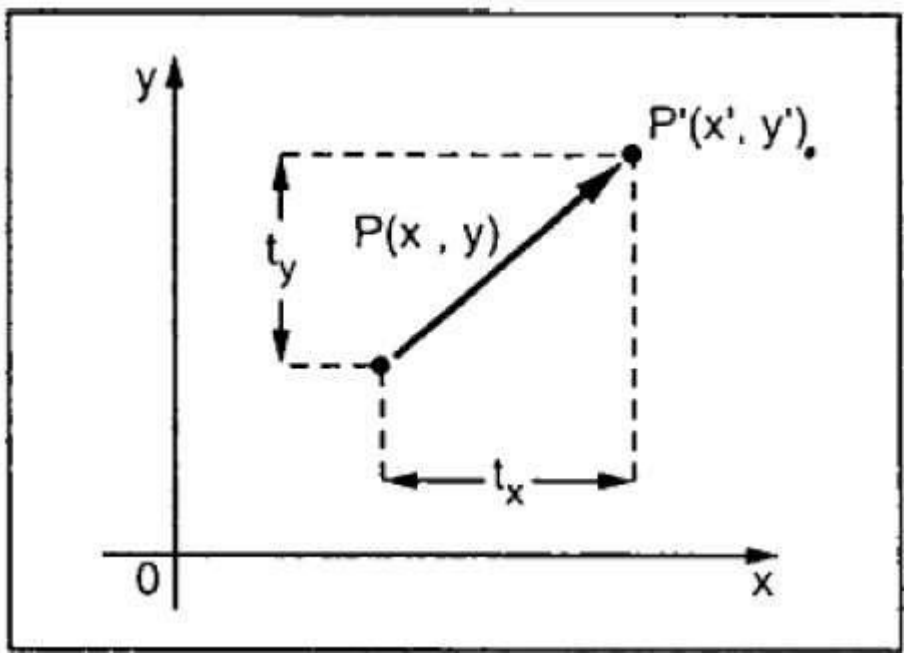
a) Translation b) Rotation c) Scaling

The other transformations are-

a) Reflection
b) Shearing

3.3:2D Translation :

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 −

$$\mathbf{X' = X + t_x}$$

$$\mathbf{Y' = Y + t_y}$$

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.

We can write it as −

P' = P + T

Now the translation matrix-

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

3.3.1: Translation of an object:

Here we translate each point individually and connect them together.Actually translation of a rigid body means moves the body without any deformation.

Suppose for a square , four corner pointa are A, B , C and D.

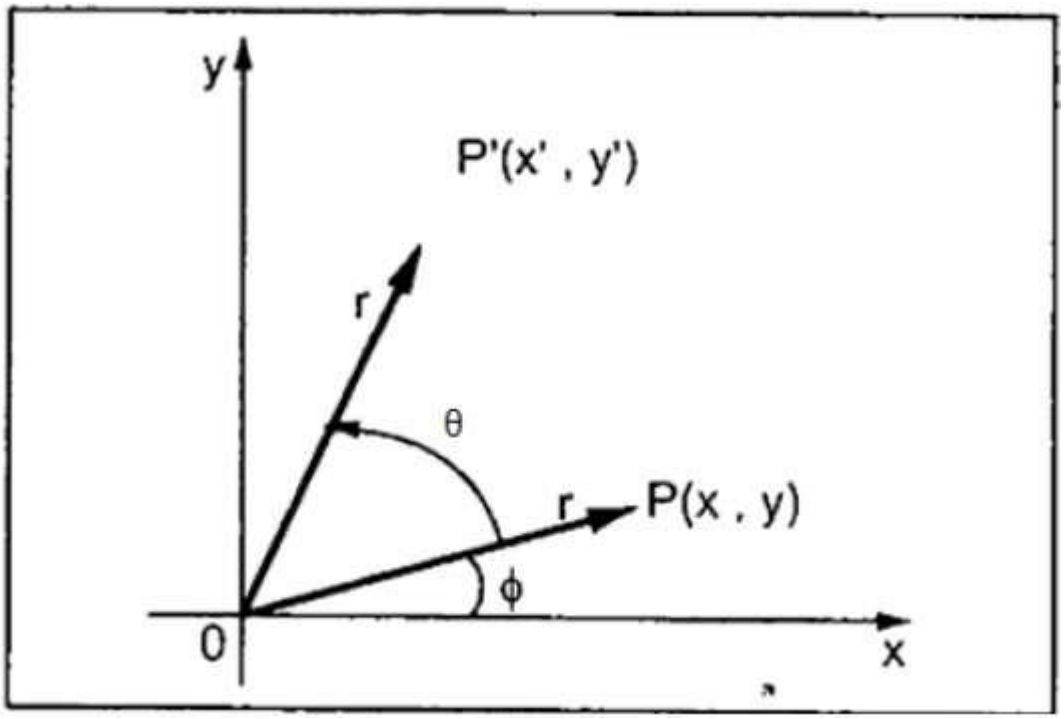
We can write-

A' =T.A ; B' =T.B ; C' =T.C and D' =T.D.

Lecture 14:

3.4: 2D Rotation: In rotation, we rotate the object at particular angle θ 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.

Let us suppose you 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 −

$$X=r\cos\phi \quad .....(1)$$
$$Y=r\sin\phi \quad .....(2)$$

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

$$x'=r\cos(\phi+\theta)=r\cos\phi\cos\theta-r\sin\phi\sin\theta \quad .....(3)$$
$$y'=r\sin(\phi+\theta)=r\cos\phi\sin\theta+r\sin\phi\cos\theta \quad .....(4)$$

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,

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

Notation

P' = R(θ).P

Where R is the rotation matrix

The rotation angle can be positive and negative.

For anticlockwise rotation θ is positive and for clockwise rotation θ is negative.

3.5: 2D Scaling:

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 −

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

The scaling factor S_X, S_Y scales the object in X and Y direction respectively.

Scaling factor>1 means scale up and <1 means scale down.

Now 2D Scaling matrix-

$$S=\begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix}$$

The above equations can also be represented in matrix form as below –

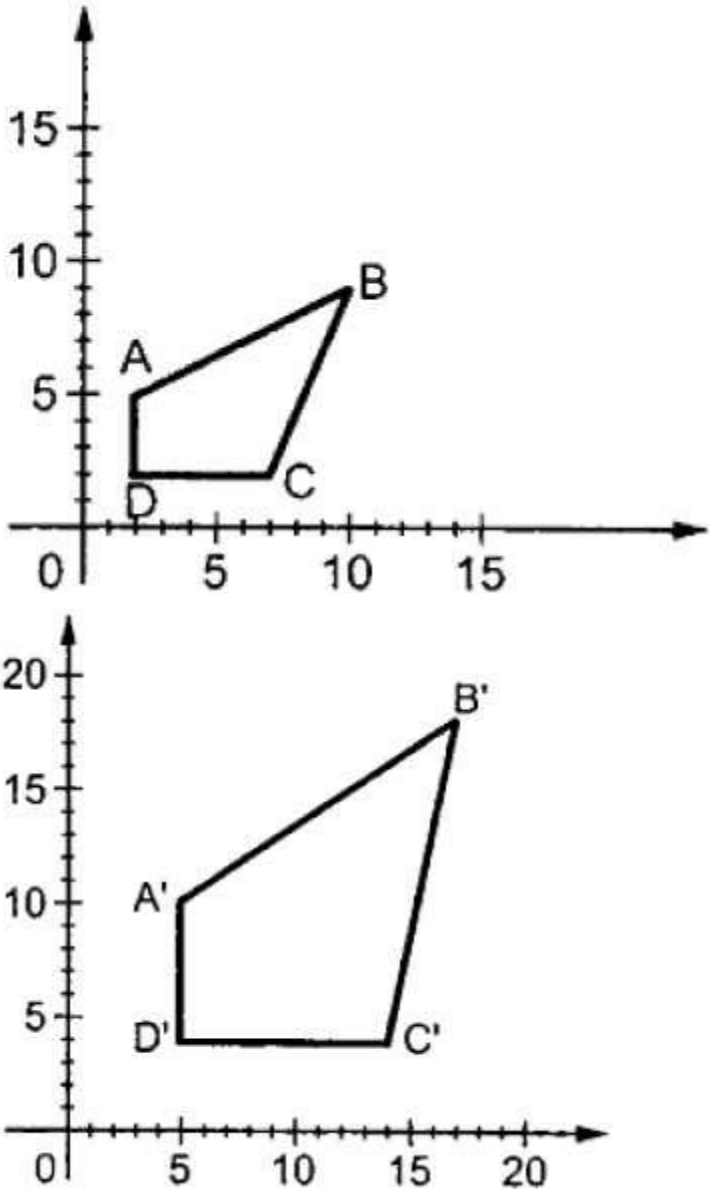
$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$$

OR

P' = P . S

Where S is the scaling matrix.

The scaling process is shown in the following figure.



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.

Lecture 15:

3.6: 2D 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.

There are four types of reflection-

a) Reflection about x axis.
b) Reflection about y axis.
c) Reflection about x-y axis.
d) Reflection about x=y axis.

a) Reflection about x axis:

P'=R.P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

b) Reflection about y axis:

P'=R.P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

c) Reflection about x-y axis:

P'=R.P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

d) Reflection about x=y axis:

P'=R.P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

NUMERICALS:

Consider a point (2,1). Perform all type of reflection.

e) Reflection about x axis:

P'=R.P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$$

So the new location (2,-1)

f) Reflection about y axis:

P'=R.P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \\ 1 \end{bmatrix}$$

So the new location (-2,1)

g) Reflection about x-y axis:

P'=R.P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \\ 1 \end{bmatrix}$$

So the new location (-2,-1)

h) Reflection about x=y axis:

P'=R.P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

So the new location (1,2)

Lecture 16

3.6: 2D shearing:

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**.

There are two types of shearing-

a) Shear about x axis
b) Shear about y axis.

a) Shear about x axis

P'=S(x).p

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & S(x) & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

b) **Shear about y axis**.

P'=S(y).p

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ S(y) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

3.7:Homogenous Coordinates:

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

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

To shorten this process, we have to use 3×3 transformation matrix instead of 2×2 transformation matrix. To convert a 2×2 matrix to 3×3 matrix, we have to add an extra dummy coordinate W.

In this way, we can represent the point by 3 numbers instead of 2 numbers, which is called **Homogenous Coordinate** system. In this system, we can represent all the transformation equations in matrix multiplication. Any Cartesian point P(X, Y) can be converted to homogenous coordinates by P' (X_h, Y_h, h).

Lecture 17

3.8: Composite Transformation.

Composite transformation means **two or more successive transformations.**

3.8.1: Two successive translations:

Let T1(tx1 ,ty1) and T2(tx2 ,ty2) be two translations applied on a point P(x,y) in succession.

The notation –

P'= T2(tx2 ,ty2).(tx1 ,ty1).P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx2 \\ 0 & 1 & ty2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & tx1 \\ 0 & 1 & ty1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & tx2 + tx1 \\ 0 & 1 & ty2 + ty1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

So two successive translations are additive in nature.

3.8.2: Two successive Rotations:

Suppose first rotation R1(θ1) and second rotation R2(θ2).

The notation-

P' = R2(θ2). R1(θ1).P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta 2 & -sin\theta 2 & 0 \\ sin\theta 2 & cos\theta 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} cos\theta 1 & -sin\theta 1 & 0 \\ sin\theta 1 & cos\theta 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

So two successive rotations are additive in nature.

3.8.3: Two successive Scalings:

Suppose first scaling S1(Sx1 ,Sy1) and second scaling S2(Sx2 ,Sy2).

The notation-

P' = S2(Sx2 ,Sy2). S1(Sx1 ,Sy1).P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx2 & 0 & 0 \\ 0 & Sy2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Sx1 & 0 & 0 \\ 0 & Sy1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} Sx1.Sx2 & 0 & 0 \\ 0 & Sy1.Sy2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

So two successive scalings are multiplicative in nature.

Lecture 18

3.9: General pivot point Rotation:

For rotating object about the arbitrary point called pivot point, we need to apply following sequence of transformation.

1. Translate the object so that the pivot-point coincides with the coordinate origin.
2. Rotate the object about the coordinate origin with specified angle.
3. Translate the object so that the pivot-point is returned to its original position (i.e. Inverse of step-1).

Let's find matrix equation for this

$$P' = T(x_r, y_r) \cdot [R(\theta) \cdot \{T(-x_r, -y_r) \cdot P\}]$$
$$P' = \{T(x_r, y_r) \cdot R(\theta) \cdot T(-x_r, -y_r)\} \cdot P$$
$$P' = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$
$$P' = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$
$$P' = R(x_r, y_r\theta) \cdot P$$

Here P' and are the column vector of final and initial point coordinate respectively and (x_r, y_r) are the coordinates of General Pivot-point.

Example: – Locate the new position of the triangle [A (5, 4), B (8, 3), C (8, 8)] after its rotation by 90° clockwise about the centroid.

$$x_r = \frac{5+8+8}{3} = 7, \qquad y_r = \frac{4+3+8}{3} = 5$$

As rotation is clockwise we will take $\theta = -90^{\circ}$.

$$P' = R_{(x_r, y_r, \theta)} \cdot P$$
$$P' = \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r \sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r \sin\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$
$$P' = \begin{bmatrix} \cos(-90) & -\sin(-90) & 7(1-\cos(-90)) + 5\sin(-90) \\ \sin(-90) & \cos(-90) & 5(1-\cos(-90)) - 7\sin(-90) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$
$$P' = \begin{bmatrix} 0 & 1 & 7(1-0) - 5(1) \\ -1 & 0 & 5(1-0) + 7(1) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$
$$P' = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 12 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 & 8 & 8 \\ 4 & 3 & 8 \\ 1 & 1 & 1 \end{bmatrix}$$

Final coordinates after rotation are [A' (11, 7), B' (13, 4), C' (18, 4)]

3.10: General pivot point Scaling

For scaling object with the position of one General Pivot point called fixed point will remain same, we need to apply following sequence of transformation.

1. Translate the object so that the fixed-point coincides with the coordinate origin.
2. Scale the object with respect to the coordinate origin with specified scale factors.
3. Translate the object so that the fixed-point is returned to its original position (i.e. Inverse of step-1).

Let's find matrix equation for this

$$\boldsymbol{P' = T(x_f, y_f) \cdot [S(s_x, s_y) \cdot \{T(-x_f, -y_f) \cdot P\}]}$$

$$\boldsymbol{P' = \{T(x_f, y_f) \cdot S(s_x, s_y) \cdot T(-x_f, -y_f)\} \cdot P}$$

$$P' = \begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = \begin{bmatrix} s_x & 0 & x_f(1-s_x) \\ 0 & s_y & y_f(1-s_y) \\ 0 & 0 & 1 \end{bmatrix} \cdot P$$

$$P' = S(x_f, y_f, s_x, s_y) \cdot P$$

Lecture 18

3D Transformation

3.11:3D Translation :

Suppose the original position P(x,y,z)

New position p'(x',y',z')

x'= x+tx

y'= y+ty

z'=z+tz

Where tx ,ty and tz are translational vectors along x ,y and z axis respectively.

The notation-

P'=T.P

Where T is the translational matrix.

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

Now the matrix representation is below-

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

3.12:3D Rotation:

Rotation in 3D is much more complex than in 2D rotation.

There are three co-ordinate axes rotations-

a) X axis rotation
b) Y axis rotation
c) Z axis rotation.

a) X axis rotation:

x'=x

y'=ycosθ-zsinθ

z'=ysinθ+zcosθ

The matrix representation-

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

b) Y axis rotation:

y'=y

x'=zsinθ+xcosθ

z'=zcosθ-xsinθ

The matrix representation-

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

c) Z axis rotation:

Z'=z

x'=xcosθ-ysinθ

y'=xsinθ+ycosθ

The matrix representation-

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

Lecture 20

3.13:3D Scaling:

Suppose the given point P(x ,y ,z)

New position P(x',y',z').

So the 3D scaling matrix-

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Where Sx ,Sy and Sz are called scaling factors along x-axis ,y-axis and z-axis respectively.

3.14:3D Reflection:

There are three types of 3D reflection-

a) Reflection about xy plane.
b) Reflection about yz plane.
c) Reflection about xz plane.

i) Reflection about xy plane:

P'=R.P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ z \\ 1 \end{bmatrix}$$

j) Reflection about yz axis:

P'=R.P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ z \\ 1 \end{bmatrix}$$

k) Reflection about yz axis:

P'=R.P

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ z \\ 1 \end{bmatrix}$$

3.15:3D Shearing:

Here three types of shearing.

a) Shear x.
b) Shear y
c) Shear z.

a) Shear about x axis

P'=S(x).p

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ Sy & 1 & 0 & 0 \\ Sz & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

b) **Shear about y axis**.

P'=S(y).p

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & Sx & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & Sz & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

c) **Shear about z axis**.

P'=S(z).p

Matrix representation-

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & Sx & 0 \\ 0 & 1 & Sy & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

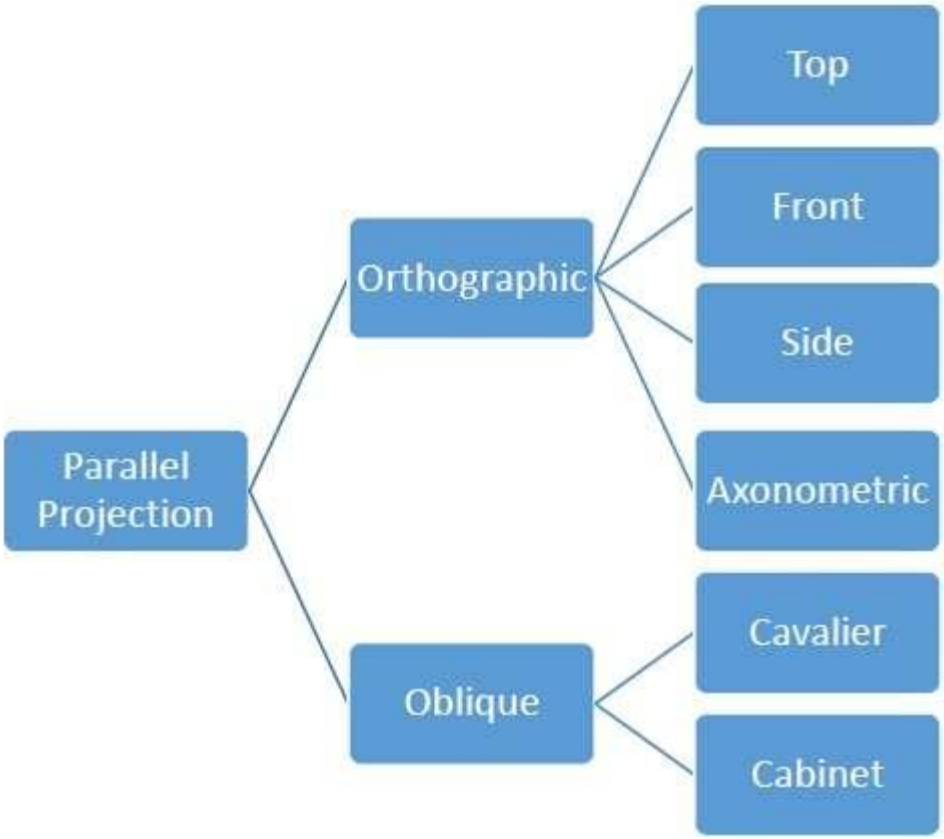
Lecture 21

3.16: 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.

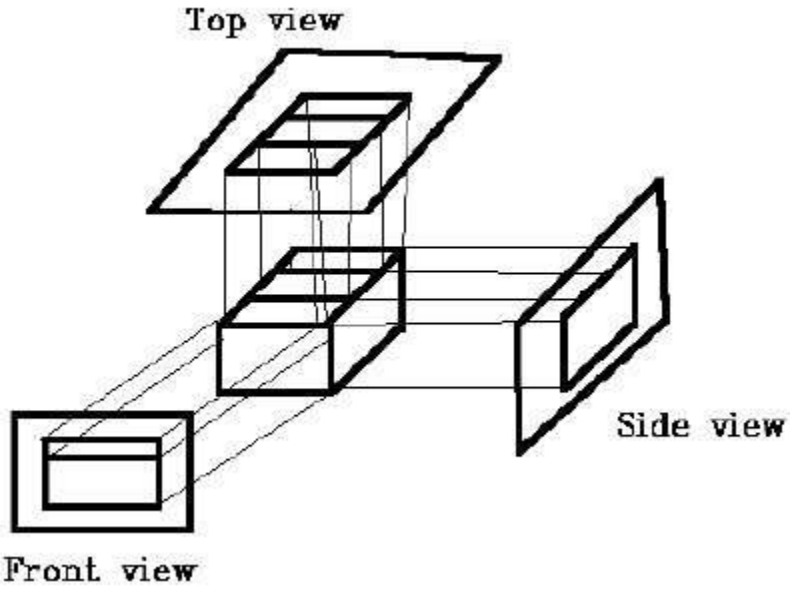
Parallel projections are less realistic, but they are good for exact measurements. In this type of projections, parallel lines remain parallel and angles are not preserved. Various types of parallel projections are shown in the following hierarchy.



Orthographic Projection

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

- Front Projection
- Top Projection
- Side Projection

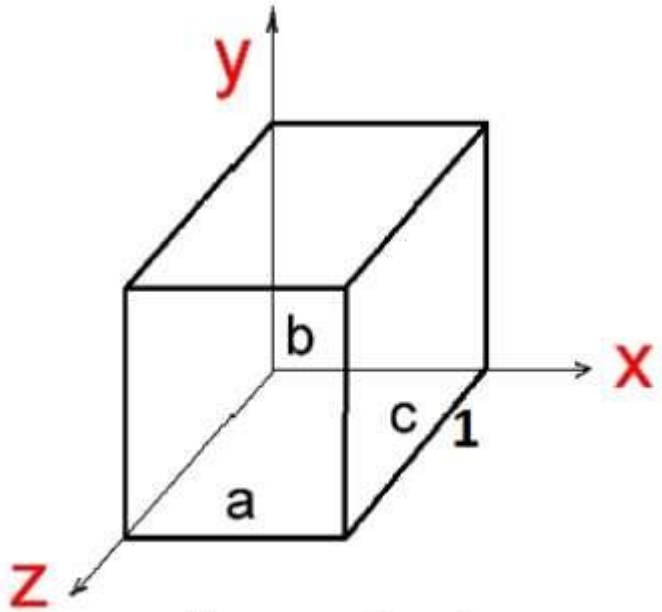


Oblique Projection

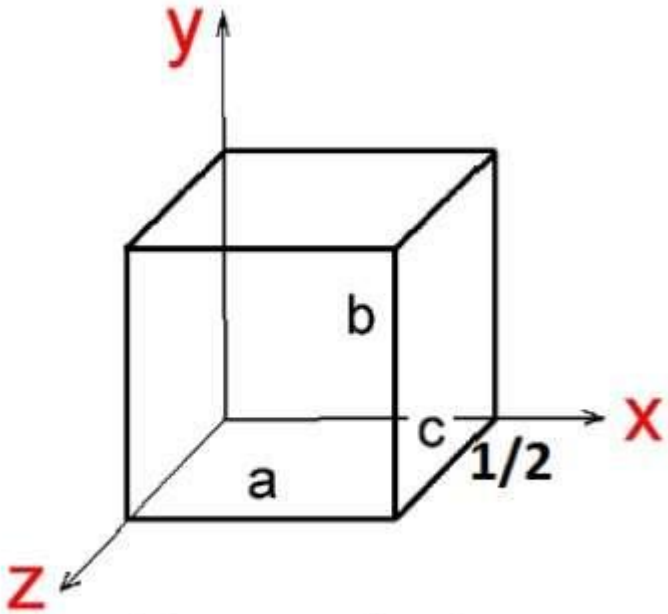
In orthographic projection, the direction of projection is not normal to the projection of plane. In oblique projection, we can view the object better than orthographic projection.

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 following figure −



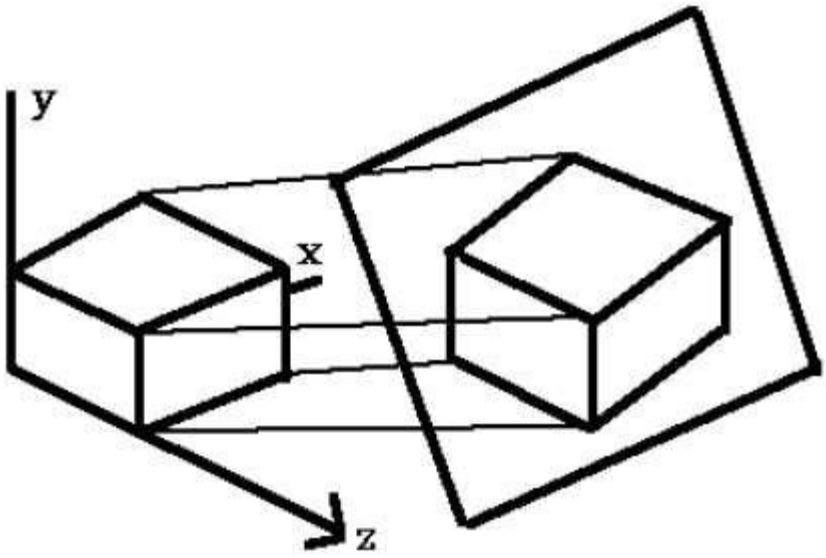
Cavalier Projection



Cabinet Projection

Isometric Projections

Orthographic projections that show more than one side of an object are called **axonometric orthographic projections**. The most common axonometric projection is an **isometric projection** where the projection plane intersects each coordinate axis in the model coordinate system at an equal distance. In this projection parallelism of lines are preserved but angles are not preserved. The following figure shows isometric projection −

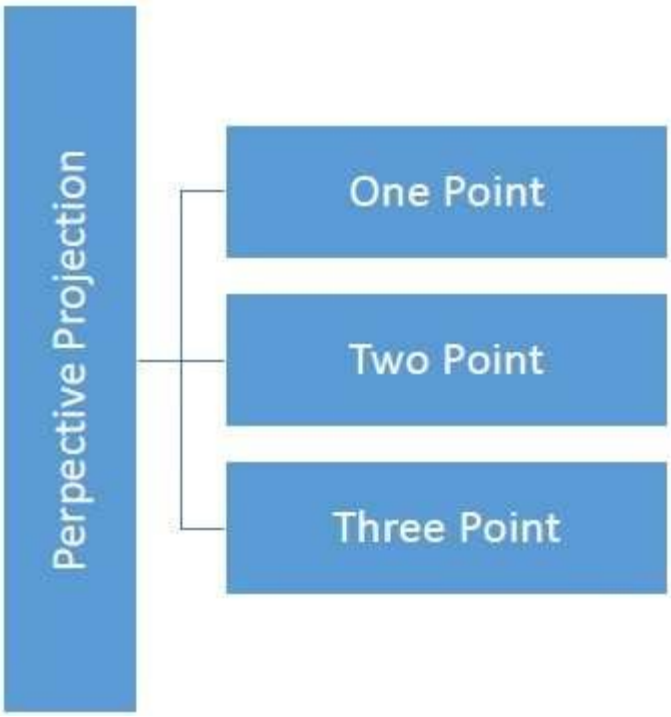


Perspective Projection

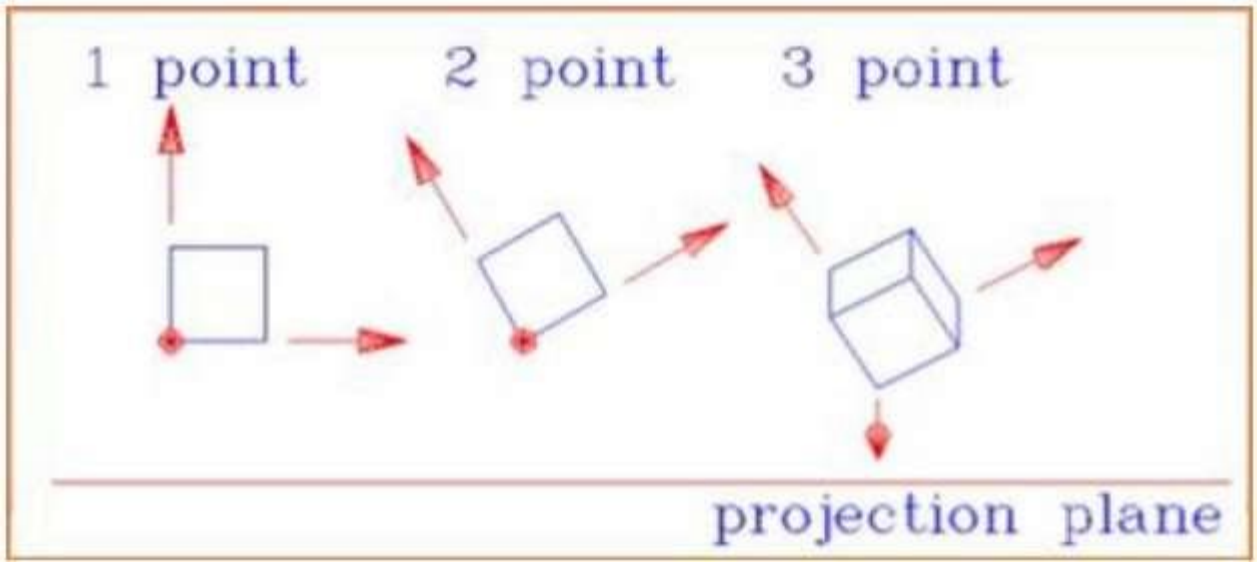
In perspective projection, the distance from the center of projection to project plane is finite and the size of the object varies inversely with distance which looks more realistic.

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.



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



Lecture 22

3.17: 2D and 3D Viewing:

Introduction: Till now we have discussed how to draw objects and how to perform transformations on these objects.Now how to display those 2D objects in display devices? For that we have to discuss about viewing.

Window port and viewport:

Window port:

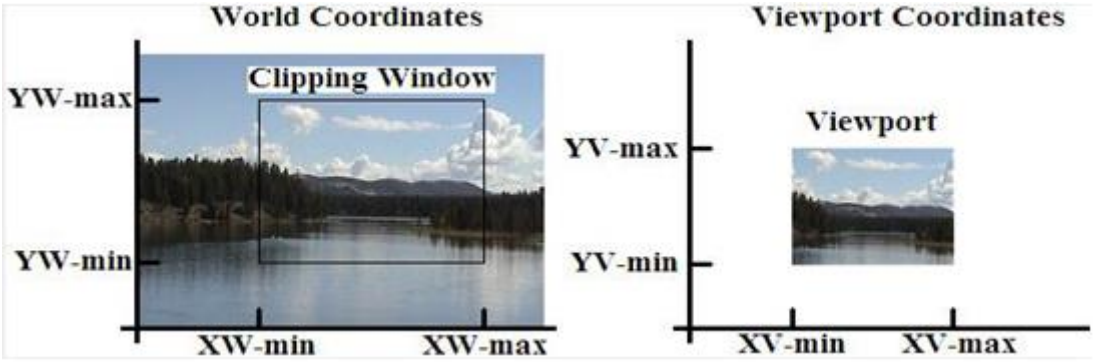
1. A world-coordinate area selected for display is called a window.
2. In computer graphics, a window is a graphical control element.
3. It consists of a visual area containing some of the graphical user interface of the program it belongs to and is framed by a window decoration.
4. A window defines 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:

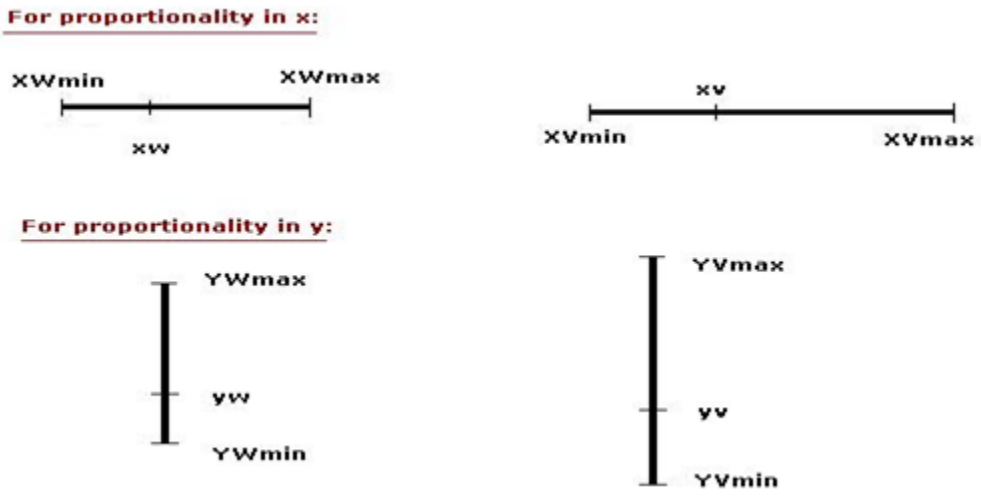
1. An area on a display device to which a window is mapped is called a viewport.
2. A viewport is a polygon viewing region in computer graphics. The viewport is an area expressed in rendering-device-specific coordinates, e.g. pixels for screen coordinates, in which the objects of interest are going to 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:**



1. This transformation involves developing formulas that start with a point in the world window, say (xw, yw).
2. The formula is used to produce a corresponding point in viewport coordinates, say (xv, yv). We would like for this mapping to be "proportional" in the sense that if xw is 30% of the way from the left edge of the world window, then xv is 30% of the way from the left edge of the viewport.
3. Similarly, if yw is 30% of the way from the bottom edge of the world window, then yv is 30% of the way from the bottom edge of the viewport. The picture below shows this proportionality.



Here scaling factors-
Sx=Xv(max)-Xv(min)/Xw(max)-Xw(min)

Sy=Yv(max)-Yv(min)/Yw(max)-Yw(min)

Xv=Xvmin+(Xw-Xwmin).Sx

Yv=Yvmin+(Yw-Ywmin).Sy

1. The position of the viewport can be changed allowing objects to be viewed at different positions on the Interface Window.
2. Multiple viewports can also be used to display different sections of a scene at different screen positions. Also, by changing the dimensions of the viewport, the size and proportions of the objects being displayed can be manipulated.
3. Thus, a zooming affect can be achieved by successively mapping different dimensioned clipping windows on a fixed sized viewport.
4. If the aspect ratio of the world window and the viewport are different, then the image may look distorted.

Lecture 23

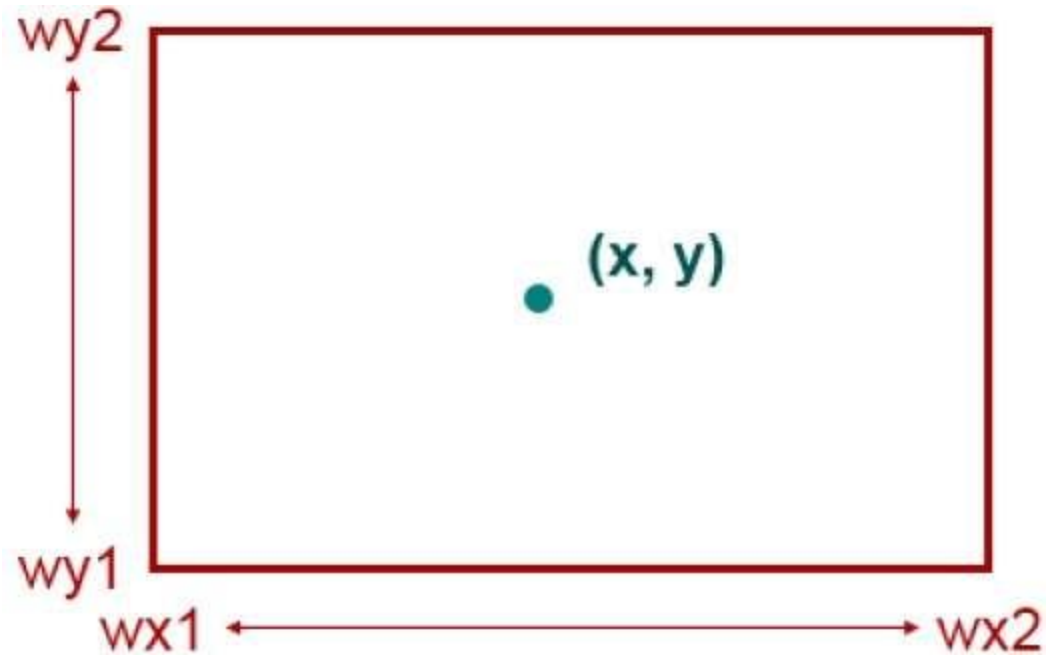
3.17:CLIPPING:

The primary use of clipping in computer graphics is to remove objects, lines, or line segments that are outside the viewing pane. The viewing transformation is insensitive to the position of points relative to the viewing volume − especially those points behind the viewer − and it is necessary to remove these points before generating the view.

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 $Wx1 \leq X \leq Wx2$. Same way, Y coordinate of the given point is inside the window, if Y lies in between $Wy1 \leq Y \leq Wy2$.



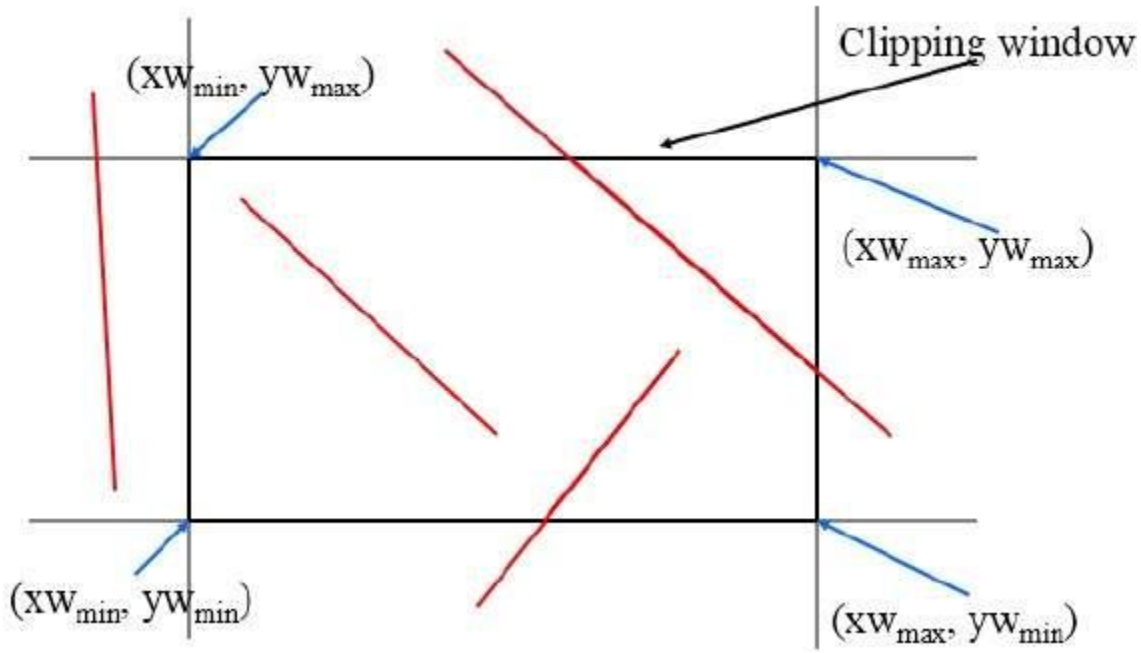
Line Clipping

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.

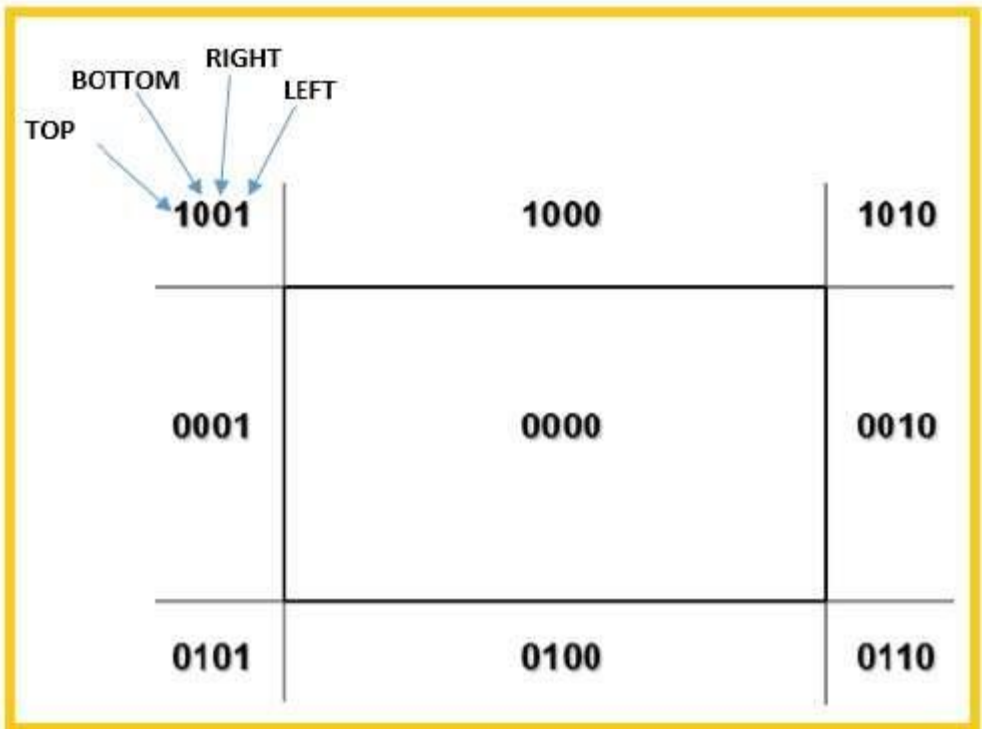
Line clipping algorithm:

a) Cohen-Sutherland Line Clippings
b) Cyrus-Beck Line Clipping Algorithm

3.17.1:Cohen-Sutherland Line Clippings: This algorithm uses the clipping window as shown in the following figure. The minimum coordinate for the clipping region is (XW_{min}, YW_{min})(XWmin,YWmin) and the maximum coordinate for the clipping region is (XW_{max}, YW_{max})(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 −

- Line can be completely inside the window (This line should be accepted).
- Line can be completely outside of the window (This line will be completely removed from the region).
- 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.

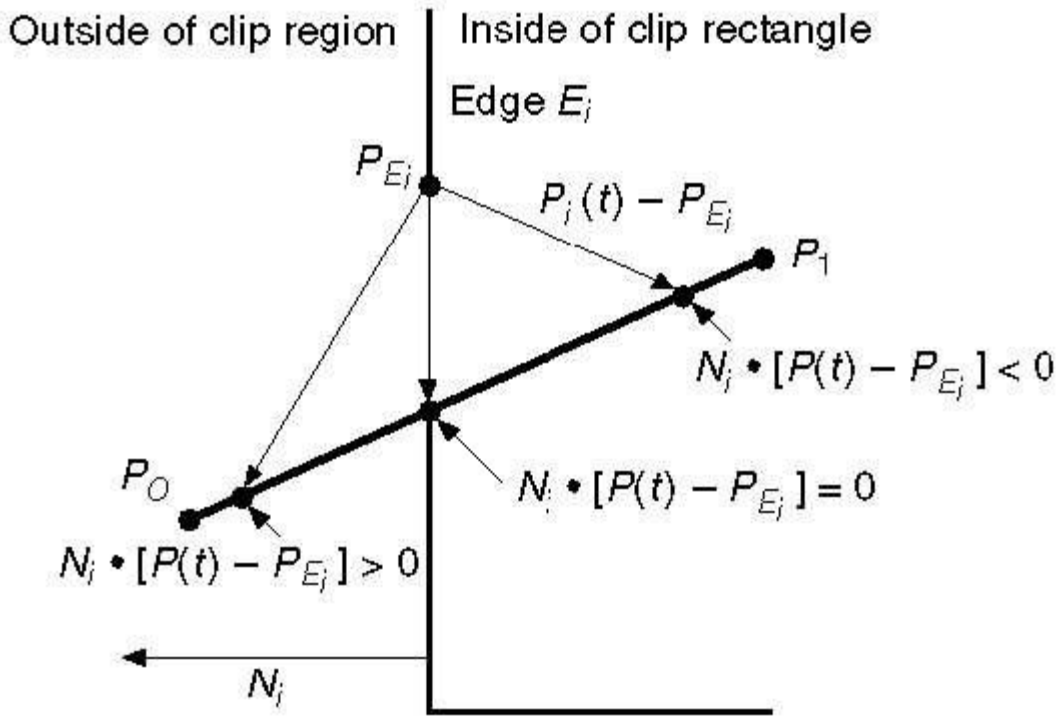
Step 3.2.4 − Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.

Step 4 − Repeat step 1 for other lines.

Lecture 24

3.17.2:Cyrus-Beck Line Clipping Algorithm:

This algorithm is more efficient than Cohen-Sutherland algorithm. It employs parametric line representation and simple dot products.



Parametric equation of line is −

$P_0P_1:P(t) = P_0 + t(P_1 - P_0)$

Let N_i be the outward normal edge E_i. Now pick any arbitrary point P_{Ei} on edge E_i then the dot product $N_i.[P(t) – P_{Ei}]$ determines whether the point P(t) is "inside the clip edge" or "outside" the clip edge or "on" the clip edge.

The point P(t) is inside if $N_i.[P(t) – P_{Ei}] < 0$

The point P(t) is outside if $N_i.[P(t) – P_{Ei}] > 0$

The point P(t) is on the edge if $N_i.[P(t) – P_{Ei}] = 0$ (Intersection point)

$N_i.[P(t) – P_{Ei}] = 0$

$N_i.[P_0 + t(P_1 - P_0) – P_{Ei}] = 0$ (Replacing P(t) with $P_0 + t(P_1 - P_0)$)

$N_i.[P_0 – P_{Ei}] + N_i.t[P_1 - P_0] = 0$

$N_i.[P_0 – P_{Ei}] + N_i \cdot tD = 0$ (substituting D for $[P_1 - P_0]$)

$N_i.[P_0 – P_{Ei}] = - N_i \cdot tD$

The equation for t becomes,

$$t=N_i.[P_o–P_{Ei}]–N_i.Dt=Ni.[Po–PEi]–Ni.D$$

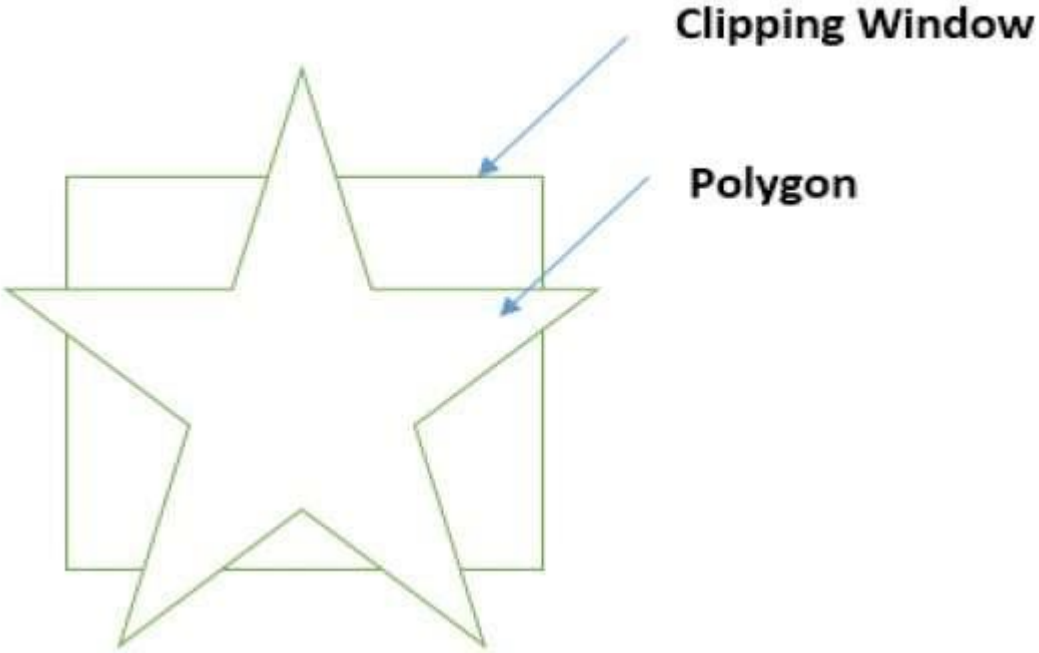
It is valid for the following conditions −

- $N_i \neq 0$ (error cannot happen)
- $D \neq 0$ ($P_1 \neq P_0$)
- $N_i \cdot D \neq 0$ (P_0P_1 not parallel to E_i)

3.18: Polygon Clipping (Sutherland Hodgman Algorithm)

A polygon can also be clipped by specifying the clipping window. Sutherland Hodgeman polygon clipping algorithm is used for polygon clipping. In this algorithm, 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.



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 −

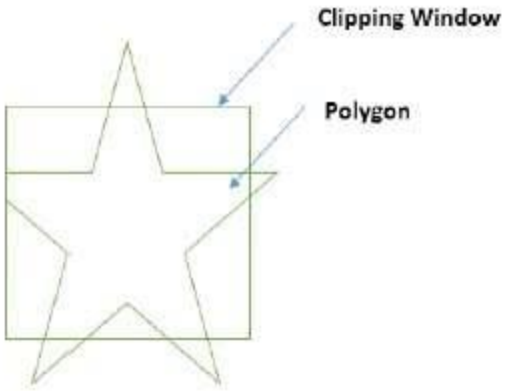


Figure: Clipping Left Edge

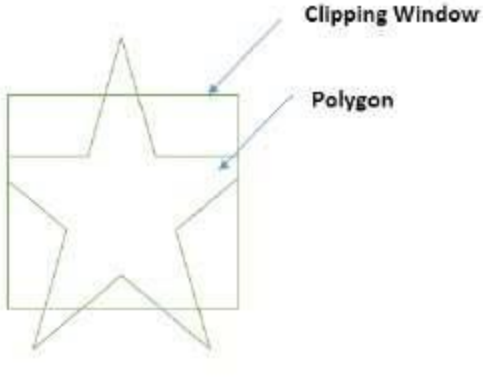


Figure: Clipping Right Edge

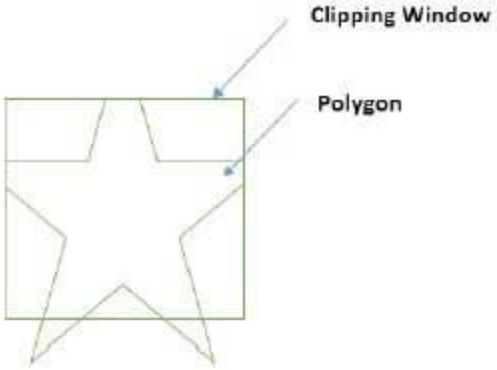


Figure: Clipping Top Edge

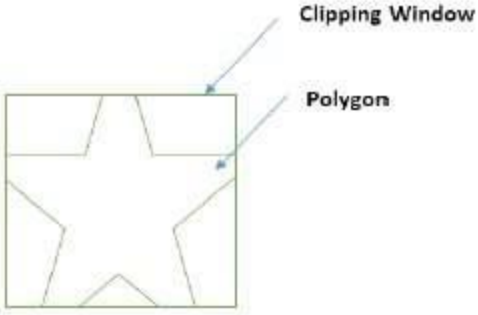


Figure: Clipping Bottom Edge

MODULE 4

Lecture 25

There are **three types of geometric models**, **wireframes**, **surfaces**, and **solids**.

Wireframe modeling is considered a natural extension of traditional methods of drafting, not requiring any extensive training of users. It forms the basis for surface models. Most of the existing surface algorithms require wireframe entities to generate surfaces.

A wireframe model consists of:

- A finite set of points (vertices)
- Each pair of points is connected by straight lines (edges), or arcs, circles, conics, and curves, such that the three- dimensional form of a solid object can be visualized.

Advantage of wireframe modeling:

- Simplicity to construct, not requiring as much computer time and memory as does surface or solid modeling.

Disadvantages of wireframe modeling:

- Wireframe models are usually confusing representations of real objects and complex designs having many edges and curve surfaces, as shown in the below figure.

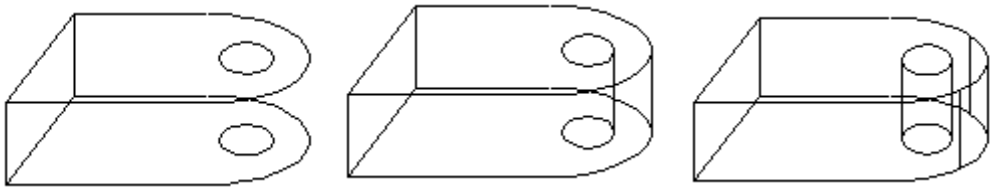


Figure. Displaying holes and curved ends in wireframe models

- Relies heavily on human interpretation
  For example, a wireframe model of a box may represent more than one object depending on which face(s) is assumed to exist.

Parametric Curves

Curves having parametric form are known as parametric curves. 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

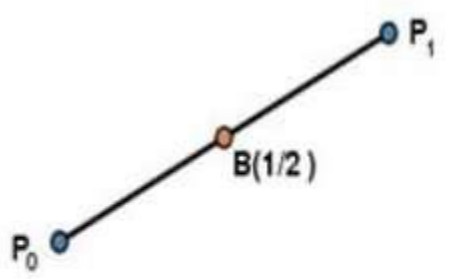
Bezier curve is discovered by the French engineer **Pierre Bézier**. These curves can be generated under the control of other points. Approximate tangents by using control points are used to generate curve. The Bezier curve can be represented mathematically as −

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

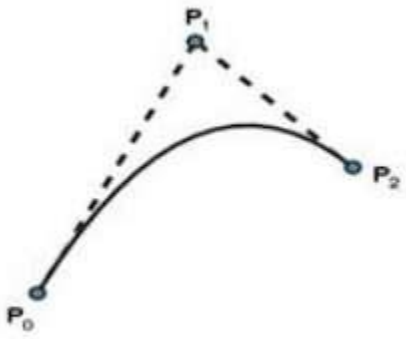
where P_i is the set of points and $B_i^n(t)$ represents the Bernstein polynomials which are given by−

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

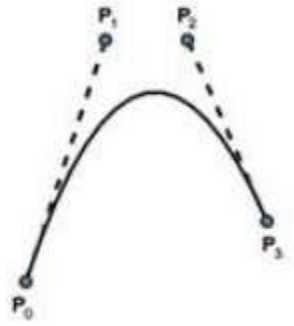
The simplest Bézier curve is the straight line from the point P0P0 to P1P1. A quadratic Bezier curve is determined by three control points. A cubic Bezier curve is determined by four control points.



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.

Lecture 26

B-Spline Curves

The Bezier-curve produced by the Bernstein basis function has limited flexibility.

- First, the number of specified polygon vertices fixes the order of the resulting polynomial which defines the curve.
- The second limiting characteristic is that the value of the blending function is nonzero for all parameter values over the entire curve.

The B-spline basis contains the Bernstein basis as the special case. The B-spline basis is non-global.

A B-spline curve is defined as a linear combination of control points Pi and B-spline basis function Ni,Ni, k (t) given by

$C(t)=\sum ni=0PiNi,k(t),C(t)=\sum i=0nPiNi,k(t)$, $n\geq k-1,n\geq k-1$, $t\epsilon[tk-1,tn+1]t\epsilon[tk-1,tn+1]$

Where,

- {pipi: i=0, 1, 2….n} are the control points
- k is the order of the polynomial segments of the B-spline curve. Order k means that the curve is made up of piecewise polynomial segments of degree k - 1,
- the Ni,k(t)Ni,k(t) are the "normalized B-spline blending functions". They are described by the order k and by a non-decreasing sequence of real numbers normally called the "knot sequence".

$$ti:i=0,...n+Kti:i=0,...n+K$$

The N_i, k functions are described as follows −

$$Ni,1(t)=\{1,0,ifu\epsilon[ti,ti+1)OtherwiseNi,1(t)=\{1,ifu\epsilon[ti,ti+1)0,Otherwise$$

and if k > 1,

$$Ni,k(t)=t-titi+k-1Ni,k-1(t)+ti+k-tti+k-ti+1Ni+1,k-1(t)Ni,k(t)=t-titi+k-1Ni,k-1(t)+ti+k-tti+k-ti+1Ni+1,k-1(t)$$

and

$$t\epsilon[tk-1,tn+1)t\epsilon[tk-1,tn+1)$$

Properties of B-spline Curve

B-spline curves have the following properties −

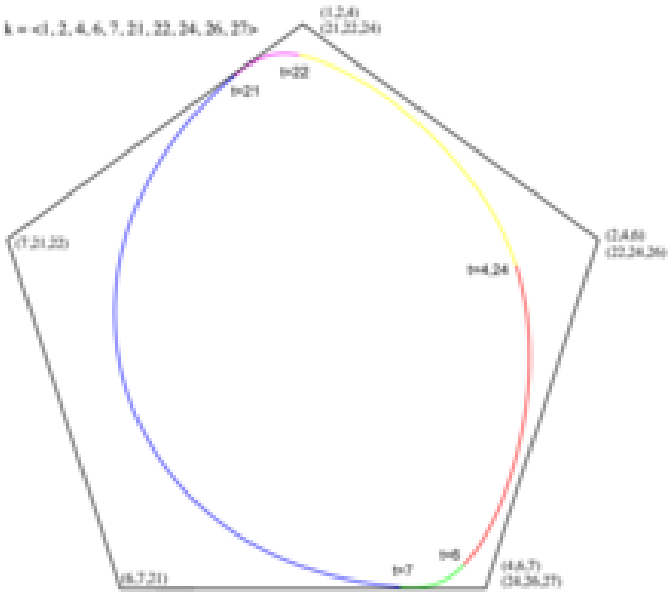
- The sum of the B-spline basis functions for any parameter value is 1.
- Each basis function is positive or zero for all parameter values.
- Each basis function has precisely one maximum value, except for k=1.
- The maximum order of the curve is equal to the number of vertices of defining polygon.

- The degree of B-spline polynomial is independent on the number of vertices of defining polygon.
- B-spline allows the local control over the curve surface because each vertex affects the shape of a curve only over a range of parameter values where its associated basis function is nonzero.
- The curve exhibits the variation diminishing property.
- The curve generally follows the shape of defining polygon.
- Any affine transformation can be applied to the curve by applying it to the vertices of defining polygon.
- The curve line within the convex hull of its defining polygon.

Lecture 27

Periodic B-Spline Curves

A special case of the B-spline is a periodic B-spline. This is simply a B-spline that ends where it begins, making a closed loop. In the image at right, each of the Bézier curves that is generated by the B-spline is colored. But how was this curve generated? There are only 2 constraints on a B-spline curve that must be met in order to turn it into a periodic B-spline. As shown, the knot vector has 10 knots in it. A cubic B-spline curve with 10 knots must have 8 control points, but where are the other three? The first 3 control points of the curve are also the last three control points. This is shown with the polar labels. In addition to control points, the knot vector must be set a certain way. If you take a look at the control points with 2 polar labels, you might notice a pattern with them. At those points, the first polar label and the second polar label have the same knot intervals, that is, they are spaced the same distance apart. For example, the topmost point has polar labels (1,2,4) and (21,22,24). The spacing between these values are 1 and 2, respectively. You might also notice this pattern with the other doubled control points. Although in this example the doubled polar labels are exactly 20 away from each other, this doesn't have to happen. The only thing that matters is the spacing (intervals) between the knots of the polar labels. This can be summed up as follows:



For a degree-*n* B-spline curve to be periodic:

1. The first *n* control points and the last *n* control points must be the same.
2. The first *n+1* knot intervals and the last *n+1* knot intervals of the knot vector must be the same.

As you can see in this case, the knot values themselves aren't necessarily relevant. The knot vector doesn't have to start at 1; it could start at 21, 51 or even 157. All that really matters are the knot intervals. A "regular" B-spline curve can also be described in terms of knot intervals, but it's more intuitive to describe periodic B-splines in terms of knot intervals. It also helps in performing knot insertion.

The curve above described in terms of knot intervals.

The cubic B-spline curve at the right is the same as the curve above and is now described in terms of knot intervals. If you remember, the curve has 10 knots in the knot vector. Discarding the *n-1* knots on either end of the vector yields the 6 parameter values over which the B-spline curve is defined. 5 Bézier curves will span these parameter values, which just happens to be the number of sides on the control polygon. We can assign each side of the control polygon to a Bézier curve and the parameter interval over which the Bézier curve is defined. In the figure at right, the colors show which side corresponds to which curve.

Let's look at what the intervals tell us about the knot vector. If the interval for a curve is 1, the parameter values could go from [0,1], [2,3], or even [156,157]. It doesn't really matter what the values of the knot vectors are, since we know that only the intervals matter. In this way, we can arbitrarily construct a knot vector starting at any value. If we started at 0 and at the topmost control point, then the knot vector would be 2, 4, 5, 18, and 19, if we proceed in a clockwise manner. We must also take into account the end condition knots at either end of the vector. This is just another way of thinking about a B-spline curve. The intervals can be any number, including 0. What happens at a 0 interval? Well, if we had a 0, then that corresponds to a repeated knot. Hopefully, you are now thinking about the de Boor algorithm and wondering how we can get the Bézier curves that a periodic B-spline curve would generate.

Rational B-splines

Rational B-splines have all of the properties of non-rational B-splines plus the following two useful features:

- They produce the correct results under projective transformations (while non-rational B-splines only produce the correct results under affine transformations).
- They can be used to represent lines, conics, non-rational B-splines; and, when generalised to patches, can represents planes, quadrics, and tori.

The antonym of *rational* is *non-rational*. Non-rational B-splines are a special case of rational B-splines, just as uniform B-splines are a special case of non-uniform B-splines. Thus, *non-uniform rational B-splines* encompass almost every other possible 3D shape definition. *Non-uniform rational B-spline* is a bit of a mouthful and so it is generally abbreviated to *NURBS*.

Rational B-splines are defined simply by applying the B-spline equation to homogeneous coordinates, rather than normal 3D coordinates.

Thus our 3D control point, $\mathbf{P}_i = (x_i, y_i, z_i)$, becomes the homogeneous control point, $\mathbf{C}_i = (x_i h_i, y_i h_i, z_i h_i, h_i)$.

A NURBS curve is thus defined as:

$$\mathbf{P}_H(t) = \sum_{i=1}^{n+1} N_{i,k}(t)\mathbf{C}_i, t_{\min} \leq t < t_{\max} \quad (1)$$

Equation 1 is equivalent to:

$$x'(t) = \sum_{i=1}^{n+1} x_i h_i N_{i,k}(t) \quad (2)$$

$$y'(t) = \sum_{i=1}^{n+1} y_i h_i N_{i,k}(t) \quad (3)$$

$$z'(t) = \sum_{i=1}^{n+1} z_i h_i N_{i,k}(t) \quad (4)$$

$$h(t) = \sum_{i=1}^{n+1} h_i N_{i,k}(t) \quad (5)$$

In 3D:

$$x(t) = x'(t)/h(t) \quad (5)$$

$$y(t) = y'(t)/h(t) \quad (6)$$

$$z(t) = z'(t)/h(t) \quad (7)$$

Thus, the 4D to 3D conversion gives us the curve in 3D:

$$\mathbf{P}(t) = \frac{\sum_{i=1}^{n+1} N_{i,k}(t)\mathbf{P}_i h_i}{\sum_{i=1}^{n+1} N_{i,k}(t) h_i}, t_{\min} \leq t < t_{\max} \quad (8)$$

This looks a lot more fierce than Equation 1, but is simply the same thing written a different way.

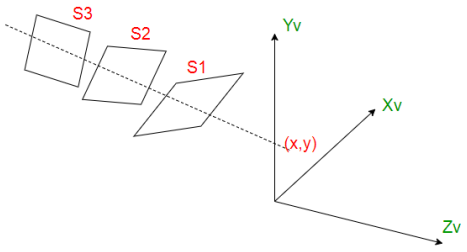
So now, we need to define an additional parameter, h_i, for each control point, $\mathbf{P}_i$. The default is to set $h_i = 1, \forall i$. This results in the denominator of Equation 8 becoming one, and the NURBS equation (Equation 8) therefore reducing to the non-rational B-spline equation .

Increasing h_i pulls the curve closer to point $\mathbf{P}_i$. Decreasing h_i pushes the curve farther from point $\mathbf{P}_i$. Setting h_i=0 means that $\mathbf{P}_i$ has no effect on the curve at all.

Lecture 28

Depth-Buffer method/ Z-Buffer method

This is a commonly used image-space approach to detect visible surfaces. It compares the object depths at each pixel position on the projection plane along the z-axis. Hence, this method is also known as the Z-buffer method. One point at a time is processed across each surface of a scene. The z-buffer algorithm can be implemented in normalized coordinates.



In the above figure, **S1**, **S2**, **S3** are the surfaces and (x,y) is the view-plane position. The surface which is closest to the projection plane is called the visible surface.

The process starts from the first surface and put its value into the buffer. The same will be done for the next surface also. Then, each overlapping pixel of the two surfaces will be checked and the one closer to the projection plane will be determined. Accordingly, the appropriate color will be displayed. As at view-plane position (x, y), surface **S1** has the smallest depth from the view plane, so it is visible at that position.

Depth-buffer method can be used for both polygon and nonplanar surfaces. However, it is usually applied for polygon surfaces, since calculation of depth values is easy.

Z-Buffer algorithm

The steps involved in a Z-buffer algorithm are as follows:

1. Initialization of the depth buffer and refresh buffer for all pixel positions (x,y) on the polygon
   i) depth(x,y) = 0 (minimum depth);
   ii) refresh(x,y) = I_{BI} (Background intensity);
2. For each pixel position (x,y) on each polygon surface,
   i) The depth *z* is calculated
   ii) If *z* > depth(x,y), then
      a) depth(x,y) = z
      b) refresh(x,y) = I_{VSI}(x,y) (Projected Intensity at position (x,y))
3. Repeat Step 2 until all surfaces have been processed

At the completion of the above algorithm, depth buffer and refresh buffer will store the depth value and the corresponding intensity values of the visible surfaces.

Time complexity = Number of pixels X Number of objects

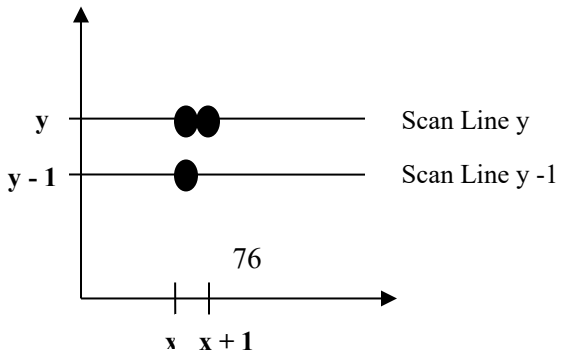
Space complexity = 2 X Number of pixels, for depth buffer and refresh buffer

The space complexity can be reduced by using a smaller depth buffer for processing only one section of the scene at a time.

Calculation of depth value *z*

From the equation of the plane Ax + By + Cz + D = 0,

z for a point A= - (Ax + By + D)/C, C≠0



z' of next point= (-A(x + 1) - By -D)/C = z – A/C

The ratio –(A/C) is constant for each surface. An incremental method is used to find the succeeding depth value.

Lecture 29

Back-Face detection

- Also known as **Plane Equation method**
- An object space method in which objects and parts of objects are compared to find out the visible surfaces.

Let us consider a triangular surface whose visibility needs to be decided.

The idea is to check if the triangle will be facing away from the viewer or not. If it does so, discard it for the current frame and move onto the next one. Each surface has a normal vector. If this normal vector is pointing in the direction of the center of projection, then it is a front face and can be seen by the viewer. If this normal vector is pointing away from the center of projection, then it is a back face and cannot be seen by the viewer.

Algorithm for left-handed system :

1. Compute N for every face of object
2. If (C.(Z component) > 0), then

    It is a back face and don't draw

    else

    It is a front face and draw

The Back-face detection method is very simple. For the left-handed system, if the Z component of the normal vector is positive, then it is a back face. If the Z component of the vector is negative, then it is a front face.

Algorithm for right-handed system :

1. Compute N for every face of object
2. If (C.(Z component) > 0), then

    It is a back face and don't draw

    else

    It is a front face and draw

Thus, for the right-handed system, if the Z component of the normal vector is negative, then it is a back face. If the Z component of the vector is positive, then it is a front face.

Back-face detection can identify all the hidden surfaces in a scene that contain non-overlapping convex polyhedra.

Recalling the polygon surface equation :

$Ax + By + Cz + D < 0$

While determining whether a surface is back-face or front face, also consider the viewing direction. The normal of the surface is given by :

$N = (A, B, C)$

A polygon is a back face if $V_{view}.N > 0$. But it should be kept in mind that after application of the viewing transformation, viewer is looking down the negative Z-axis. Therefore, a polygon is back face if :

$(0, 0, -1).N > 0$

or if $C < 0$

Viewer will also be unable to see surface with $C = 0$, therefore, identifying a polygon surface as a back face if : $C <= 0$.

Considering (a),

$V.N = |V||N|Cos(angle)$

if $0 <= angle$ 0 and $V.N > 0$

Hence, Back-face.

Considering (b),

V.N = |V||N|Cos(angle)

if 90 < angle <= 180, then

cos(angle) < 0 and V.N < 0

Hence, Front-face.

Limitations:

1. This method works fine for convex polyhedra, but not necessarily for concave polyhedra.
2. This method can only be used on solid objects modeled as a polygon mesh.

Painter's Algorithm

The painter's algorithm is based on depth sorting and is a combined object and image space algorithm. It is as follows:

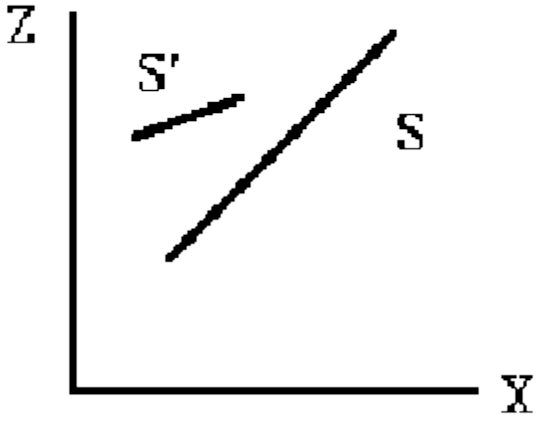
1. Sort all polygons according to z value (object space); Simplest to use maximum z value
2. Draw polygons from back (maximum z) to front (minimum z)

This can be used for wireframe drawings as well by the following:

1. Draw solid polygons using Polyscan (in the background color) followed by Polyline (polygon color).
2. Polyscan erases Polygons behind it then Polyline draws new Polygon.

Problems with simple Painter's algorithm

Look at cases where it doesn't work correctly. *S* has a greater depth than *S′* and so will be drawn first. But *S′* should be drawn first since it is obscured by *S*. We must somehow reorder *S* and *S′*:



A series of tests need to be determined, if two polygons need to be reordered. If the polygons fail a test, then the next test must be performed. If the polygons fail all tests, then they are reordered. The initial tests are computationally cheap, but the later tests are more expensive.

S, let us look at revised algorithm to test for possible reordering

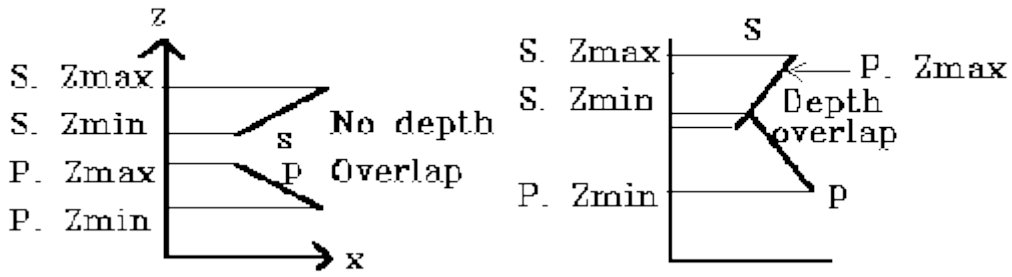
Could store Zmax, Zmin for each Polygon.

- sort on Zmax

- start with polygon with greatest depth (S)

- compare S with all other polygons (P) to see if there is any depth overlap(Test 0)

If S.Zmin <= P.Zmax then have depth overlap (as in above and below figures)



If have depth overlap (failed Test 0) we may need to reorder polygons.

Next (Test 1) check to see if polygons overlap in xy plane (use bounding rectangles)

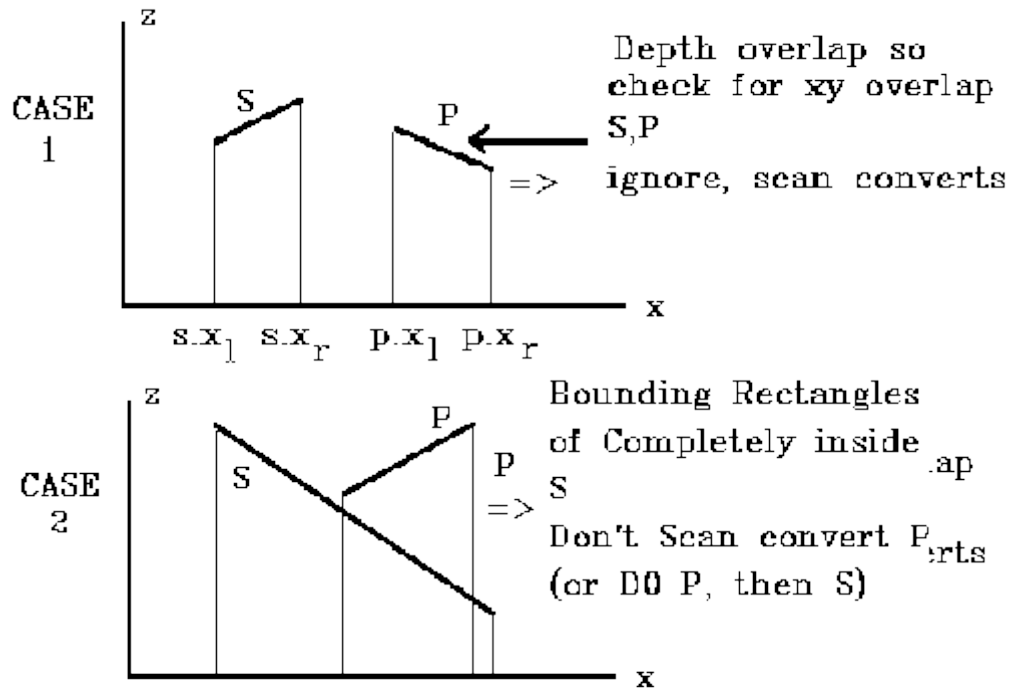
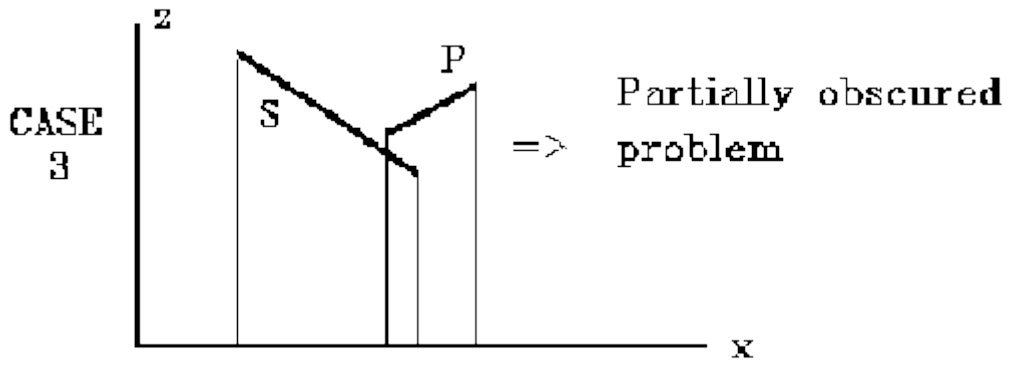


Fig 9 – 9



Do above tests for x and y

If have case 1 or 2 then we are done (passed Test 1) but for case 3 we need further testing failed Test 1)

Next test (Test 2) to see if polygon S is "outside" of polygon P (relative to view plane)

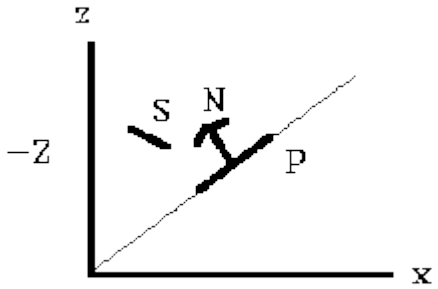
Remember: a point (x, y, z) is "outside" of a plane if we put that point into the plane equation and get:

$Ax + By + Cz + D > 0$

So to test for S outside of P, put all vertices of S into the plane equation for P and check that all vertices give a result that is > 0.

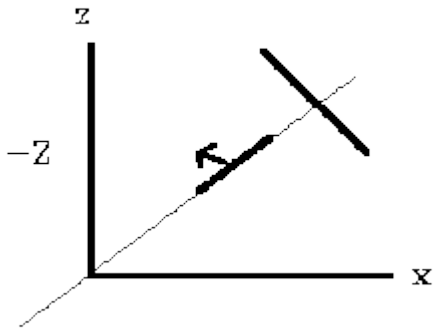
i.e. $Ax' + By' + Cz' + D > 0$ x', y', z' are S vertices

A, B, C, D are from plane equation of P (choose normal away from view plane since define "outside" with respect to the view plane)



S completely "outside" of P so ok -> Scan convert S

Fig 9 – 11



S not outside of P

ert

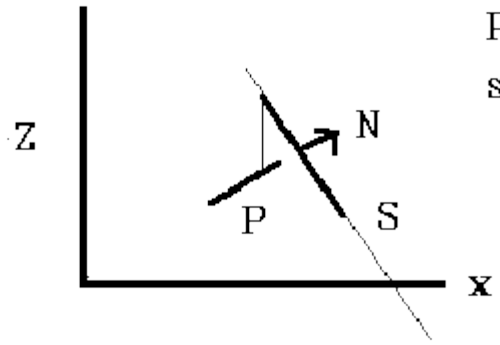
Fig 9 – 12

If the test of S "outside" of P fails, then test to see if P is "inside" of S (again with respect to the view plane) (Test 3).

Compute plane equation of S and put in all vertices of P, if all vertices of P inside of S then P inside.

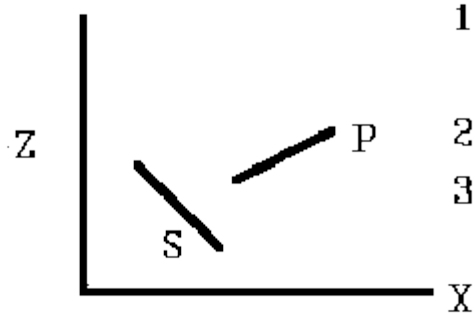
inside test: $Ax' + By' + Cz' + D$ &lt0 where x', y', z' are coordinates of P vertices

so for above case:



P is inside of S
so Scan convert S

Fig 9 – 13



1. bounding rectangles overlap
2. S not inside of P
3. P not inside of S

Fig 9 – 14

Then we do the 4th test and check for overlap for actual projections in xy plane since may have bounding rectangles overlap but not actual overlap

For example: Look at projection of two polygons in the xy plane

Then have two possible cases.

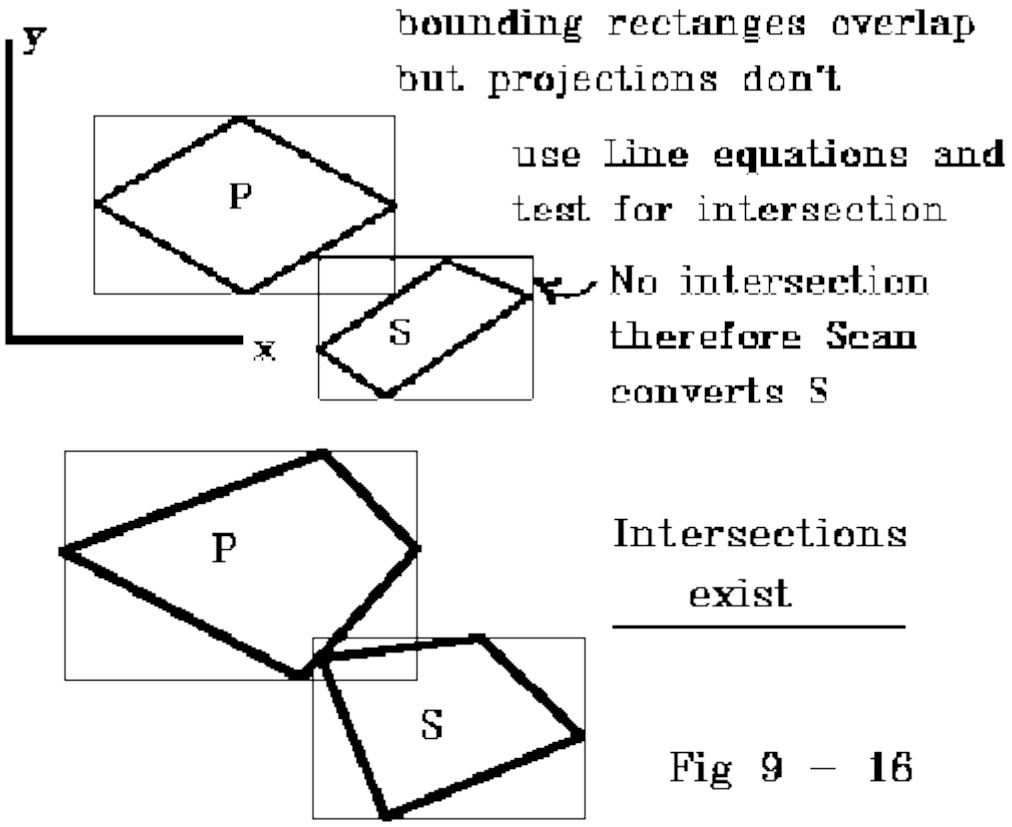
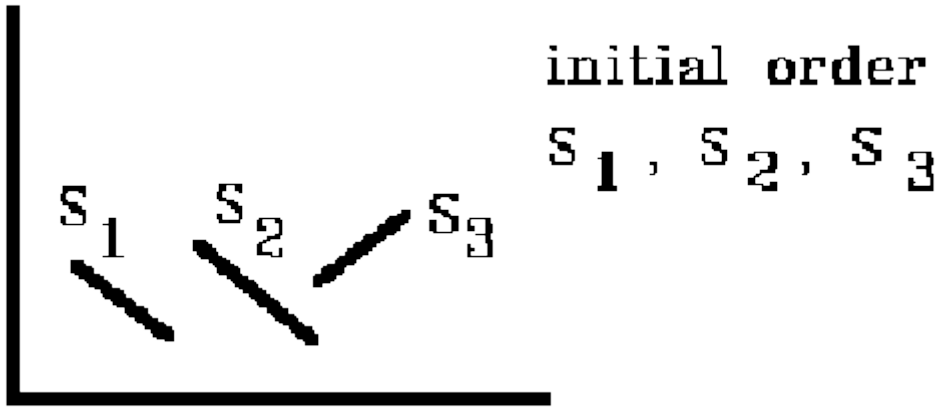


Fig 9 – 16

All 4 tests have failed therefore interchange P and S and scan convert P before S. But before we scan convert P we must test P against all other polygons. Look at an example of multiple interchanges

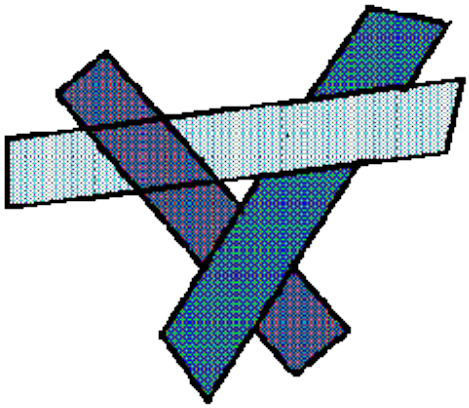


Test S1 against S2 and it fails all tests so reorder: S2, S1, S3

Test S2 against S3 and it fails all tests so reorder: S3, S2, S1

Possible Problem: Polygons that alternately obscure one another. These three polygons will continuously reorder.

One solution might be to flag a reordered polygon and subdivide the polygon into several smaller polygons.

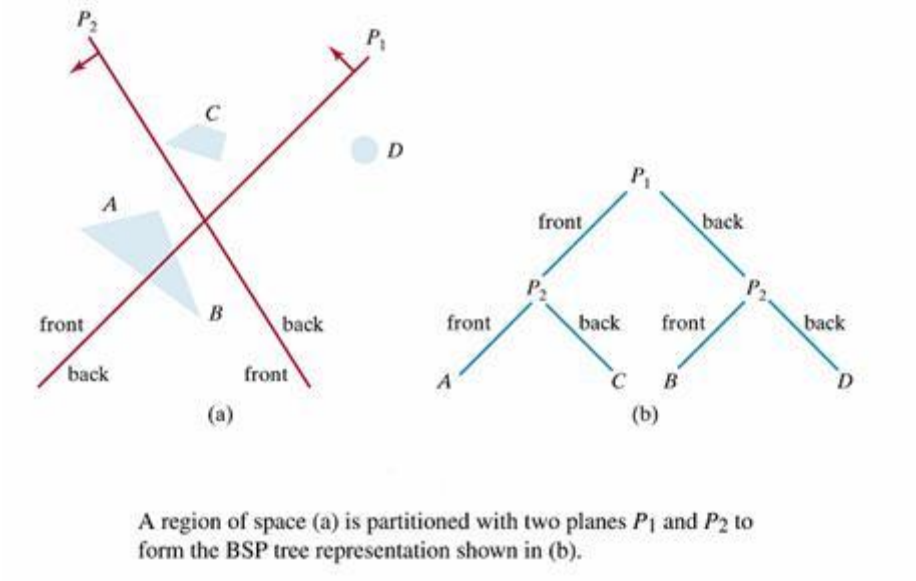


Lecture 30

BSP-Tree Method

- A *binary space partitioning tree* (*bsp–tree*) is a binary tree whose nodes contain polygons.
- Binary space partitioning, or BSP, divides space into distinct sections by building a tree representing that space.
- Used to sort polygons.
- A BSP takes the polygons and divides them into two groups bychoosing a plane, usually taken from the set of polygons, and divides the world into two spaces.
- It decides which side of the plane each polygon is on, or it may also be on the plane.
- If a polygon intersects the splitting plane it must be split into two separate polygons, one on each side of the plane.
- The tree is built by choosing a partitioning plane and dividing the remaining polygons into two or three lists: Front, Back and On lists – done by comparing the normal vector of the plane with that of each polygon.
- For each node in a bsp-tree the polygons in the left subtree lie behind the polygon at the node while the polygons in the right subtree lie in front of the polygon at the node.

- Each polygon has a fixed normal vector, and front and back are measured relative to this fixed normal.
- Once a bsp-tree is constructed for a scene, the polygons are rendered by an in order traversal of the bsp-tree.
- Recursive algorithms for generating a bsp-tree and then using the bsp-tree to render a scene are presented below.



A region of space (a) is partitioned with two planes P_1 and P_2 to form the BSP tree representation shown in (b).

Algorithm

Select any polygon (plane) in the scene for the root.

1. Partition all the other polygons in the scene to the back (left subtree) or the front (right subtree).
2. Split any polygons lying on both sides of the root (see below).
3. Build the left and right subtrees recursively.

BSP-Tree Rendering Algorithm (In order tree traversal)

1. If the eye is in front of the root, then
   a. Display the left subtree (behind)
   b. Display the root
   c. Display the right subtree (front)

2. If eye is in back of the root, then

a. Display the right subtree (front)
b. Display the root
c. Display the left subtree (back)