Unit 3
Raster Scan Graphics

Circle Generation derivation

vertically downward. These are labeled m_H, m_D, m_V, respectively, in Fig. 2–12. The algorithm chooses the pixel which minimizes the square of the distance between one of these pixels and the true circle, i.e., the minimum of

$$m_H = |(x_i + 1)^2 + (y_i)^2 - R^2|$$
$$m_D = |(x_i + 1)^2 + (y_i - 1)^2 - R^2|$$
$$m_V = |(x_i)^2 + (y_i - 1)^2 - R^2|$$

The difference between the square of the distance from the center of the circle to the diagonal pixel at $(x_i + 1, y_i - 1)$ and the distance to a point on the circle R^2 is

$$\Delta_i = (x_i + 1)^2 + (y_i - 1)^2 - R^2$$

If $\Delta_i < 0$, then the diagonal point $(x_i + 1, y_i - 1)$ is inside the actual circle, i.e., case 1 or 2 in Fig. 2–13. It is clear that either the pixel at $(x_i + 1, y_i)$, i.e., m_H, or that at $(x_i + 1, y_i - 1)$, i.e., m_D, must be chosen. To decide which, first consider case 1 by examining the difference between the squares of the distance from the actual circle to the pixel at m_H and the distance from the actual circle to the pixel at m_D, i.e.,

$$\delta = \left|(x_i + 1)^2 + (y_i)^2 - R^2\right| - \left|(x_i + 1)^2 + (y_i - 1)^2 - R^2\right|$$

If $\delta < 0$, then the distance from the actual circle to the diagonal pixel, m_D, is greater than that to the horizontal pixel, m_H. Conversely, if $\delta > 0$, then the distance to the horizontal pixel, m_H, is greater; thus, if

$$\begin{aligned} \delta \leq 0 \quad & \text{choose } m_H \text{ at } (x_i + 1, y_i) \\ \delta > 0 \quad & \text{choose } m_D \text{ at } (x_i + 1, y_i - 1) \end{aligned}$$

The horizontal move is selected when $\delta = 0$, i.e., when the distances are equal.

The work involved in evaluating δ is reduced by noting that for case 1

$$(x_i + 1)^2 + (y_i)^2 - R^2 \geq 0$$
$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 < 0$$

because the diagonal pixel at $(x_i + 1, y_i - 1)$ is always inside the circle, and the horizontal pixel at $(x_i + 1, y_i)$ is always outside the circle. Thus, δ can be evaluated as

$$\delta = (x_i + 1)^2 + (y_i)^2 - R^2 + (x_i + 1)^2 + (y_i - 1)^2 - R^2$$

Completing the square for the $(y_i)^2$ term by adding and subtracting $-2y_i + 1$ yields

$$\delta = 2\left[(x_i + 1)^2 + (y_i - 1)^2 - R^2\right] + 2y_i - 1$$

Using the definition for Δ_i gives

$$\delta = 2(\Delta_i + y_i) - 1$$

If $\Delta_i > 0$, then the diagonal point $(x_i + 1, y_i - 1)$ is outside the actual circle, i.e., case 3 or 4 in Fig. 2–13. Here, it is clear that either the pixel at $(x_i + 1, y_i - 1)$, i.e., m_D, or that at $(x_i, y_i - 1)$, i.e., m_V, must be chosen. Again, the decision criteria is obtained by first considering case 3 and examining the difference between the squares of the distance from the actual circle to the diagonal pixel at m_D, and the distance from the actual circle to the pixel at m_V, that is

$$\delta' = |(x_i + 1)^2 + (y_i - 1)^2 - R^2| - |(x_i)^2 + (y_i - 1)^2 - R^2|$$

If $\delta' < 0$, then the distance from the actual circle to the vertical pixel at $(x_i, y_i - 1)$ is larger and the diagonal move m_D to the pixel at $(x_i + 1, y_i - 1)$ is chosen. Conversely, if $\delta' > 0$, then the distance from the actual circle to the diagonal pixel is greater, and the vertical move to the pixel at $(x_i, y_i - 1)$ is chosen. Thus, if

$$\begin{aligned} \delta' \leq 0 \quad & \text{choose } m_D \text{ at } (x_i + 1, y_i - 1) \\ \delta' > 0 \quad & \text{choose } m_V \text{ at } (x_i, y_i - 1) \end{aligned}$$

Here, the diagonal move is selected when $\delta' = 0$, that is, when the distances are equal.

Again, examination of the components of δ' shows that

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 \geq 0$$
$$(x_i)^2 + (y_i - 1)^2 - R^2 < 0$$

because the diagonal pixel at $(x_i + 1, y_i - 1)$ is outside the actual circle, while the vertical pixel at $(x_i, y_i - 1)$ is inside the actual circle for case 3. This allows δ' to be written as

$$\delta' = (x_i + 1)^2 + (y_i - 1)^2 - R^2 + (x_i)^2 + (y_i - 1)^2 - R^2$$

Completing the square for the $(x_i)^2$ term by adding and subtracting $2x_i + 1$ yields

$$\delta' = 2\left[(x_i + 1)^2 + (y_i - 1)^2 - R^2\right] - 2x_i - 1$$

Using the definition of Δ_i then yields

$$\delta' = 2(\Delta_i - x_i) - 1$$

It remains only to examine case 5 of Fig. 2–13, which occurs when the diagonal pixel at $(x_i + 1, y_i - 1)$ lies on the actual circle, i.e., for $\Delta_i = 0$. Examining the components of δ shows that

$$(x_i + 1)^2 + (y_i)^2 - R^2 > 0$$
$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 = 0$$

Hence, $\delta > 0$; and the diagonal pixel at $(x_i + 1, y_i - 1)$ is selected. Similarly, the components of δ' are

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 = 0$$
$$(x_i)^2 + (y_i - 1)^2 - R^2 < 0$$

and $\delta' < 0$, which is the condition for selecting the correct diagonal move to $(x_i + 1, y_i - 1)$. Thus, the case of $\Delta_i = 0$ is satisfied by the same criteria used for $\Delta_i < 0$ or for $\Delta_i > 0$.

Summarizing these results yields

$$
\begin{array}{lll}
\Delta_i < 0 & & \\
\delta \leq 0 & \text{choose the pixel at } (x_i + 1, y_i) & \longrightarrow \quad m_H \\
\delta > 0 & \text{choose the pixel at } (x_i + 1, y_i - 1) & \longrightarrow \quad m_D
\end{array}
$$

$$
\begin{array}{lll}
\Delta_i > 0 & & \\
\delta' \leq 0 & \text{choose the pixel at } (x_i + 1, y_i - 1) & \longrightarrow \quad m_D \\
\delta' > 0 & \text{choose the pixel at } (x_i, y_i - 1) & \longrightarrow \quad m_V \\
\Delta_i = 0 & \text{choose the pixel at } (x_i + 1, y_i - 1) & \longrightarrow \quad m_D
\end{array}
$$

Simple recursion relationships which yield an incremental implementation of the algorithm are easily developed. First, consider the horizontal movement, m_H, to the pixel at $(x_i + 1, y_i)$. Call this next pixel location $(i + 1)$. The coordinates of the new pixel and the value of Δ_i are then

$$
\begin{aligned}
x_{i+1} &= x_i + 1 \\
y_{i+1} &= y_i \\
\Delta_{i+1} &= (x_{i+1} + 1)^2 + (y_{i+1} - 1)^2 - R^2 \\
&= (x_{i+1})^2 + 2x_{i+1} + 1 + (y_i - 1)^2 - R^2 \\
&= (x_i + 1)^2 + (y_i - 1)^2 - R^2 + 2x_{i+1} + 1 \\
&= \Delta_i + 2x_{i+1} + 1
\end{aligned}
$$

Similarly, the coordinates of the new pixel and the value of Δ_i for the move m_D to $(x_i + 1, y_i - 1)$ are

$$x_{i+1} = x_i + 1$$

$$y_{i+1} = y_i - 1$$

$$\Delta_{i+1} = \Delta_i + 2x_{i+1} - 2y_{i+1} + 2$$

Those for the move m_V to $(x_i, y_i - 1)$ are

$$x_{i+1} = x_i$$

$$y_{i+1} = y_i - 1$$

$$\Delta_{i+1} = \Delta_i - 2y_{i+1} + 1$$

Bresenham's incremental circle algorithm *for the first quadrant*

all variables are assumed integer

initialize the variables

$x_i = 0$
$y_i = R$
$\Delta_i = 2(1 - R)$
$\text{Limit} = 0$
while $y_i \geq \text{Limit}$
    call setpixel (x_i, y_i)
    determine if case 1 or 2, 4 or 5, or 3
    if $\Delta_i < 0$ **then**
        $\delta = 2\Delta_i + 2y_i - 1$
        determine whether case 1 or 2
        if $\delta \leq 0$ **then**
            call $\text{mh}(x_i, y_i, \Delta_i)$

```
            else
                call md(x_i, y_i, Δ_i)
            end if
        else if Δ_i > 0 then
            δ' = 2Δ_i − 2x_i − 1
            determine whether case 4 or 5
            if δ' ≤ 0 then
                call md(x_i, y_i, Δ_i)
            else
                call mv(x_i, y_i, Δ_i)
            end if
        else if Δ_i = 0 then
            call md(x_i, y_i, Δ_i)
        end if
    end while
finish
```

move horizontally

subroutine mh(x_i, y_i, Δ_i)

$x_i = x_i + 1$

$\Delta_i = \Delta_i + 2x_i + 1$

end sub

move diagonally

subroutine md(x_i, y_i, Δ_i)

$x_i = x_i + 1$

$y_i = y_i - 1$

$\Delta_i = \Delta_i + 2x_i - 2y_i + 2$

end sub

move vertically

subroutine mv(x_i, y_i, Δ_i)

$y_i = y_i - 1$

$\Delta_i = \Delta_i - 2y_i + 1$

end sub

To illustrate the circle generation algorithm, consider the origin-centered circle of radius 8. Only the first quadrant is generated.

initial calculations

$x_i = 0$
$y_i = 8$
$\Delta_i = 2(1-8) = -14$
Limit $= 0$

Incrementing through the main loop yields

$y_i >$ Limit
continue
setpixel $(0, 8)$
$\Delta_i < 0$
$\delta = 2(-14) + 2(8) - 1 = -13$
$\delta < 0$
call mh(0,8,−14)
$x = 0 + 1 = 1$
$\Delta_i = -14 + 2(1) + 1 = -11$

$y_i >$ Limit
continue
setpixel $(1, 8)$
$\Delta_i < 0$
$\delta = 2(-11) + 2(8) - 1 = -7$
$\delta < 0$
call mh(1,8,−11)
$x = 1 + 1 = 2$
$\Delta_i = -11 + 2(2) + 1 = -6$

$y_i >$ Limit
continue
setpixel $(2, , 8)$
.
.
.
continue

setpixel	Δ_i	δ	δ'	x	y
	-14			0	8
(0, 8)					
	-11	-13		1	8
(1, 8)					
	-6	-7		2	8
(2, 8)					
	-12	3		3	7
(3, 7)					
	-3	-11		4	7
(4, 7)					
	-3	7		5	6
(5, 6)					
	1	5		6	5
(6, 5)					
	9		-11	7	4
(7, 4)					
	4		3	7	3
(7, 3)					
	18		-7	8	2
(8, 2)					
	17		19	8	1
(8, 1)					
	18		17	8	0
(8, 0)					

Scan Conversion Definition

- It is a process of representing graphics objects a collection of pixels. The graphics objects are continuous. The pixels used are discrete. Each pixel can have either on or off state.

- The circuitry of the video display device of the computer is capable of converting binary values (0, 1) into a pixel on and pixel off information. 0 is represented by pixel off. 1 is represented using pixel on. Using this ability graphics computer represent picture having discrete dots.

Run-length encoding (RLE)

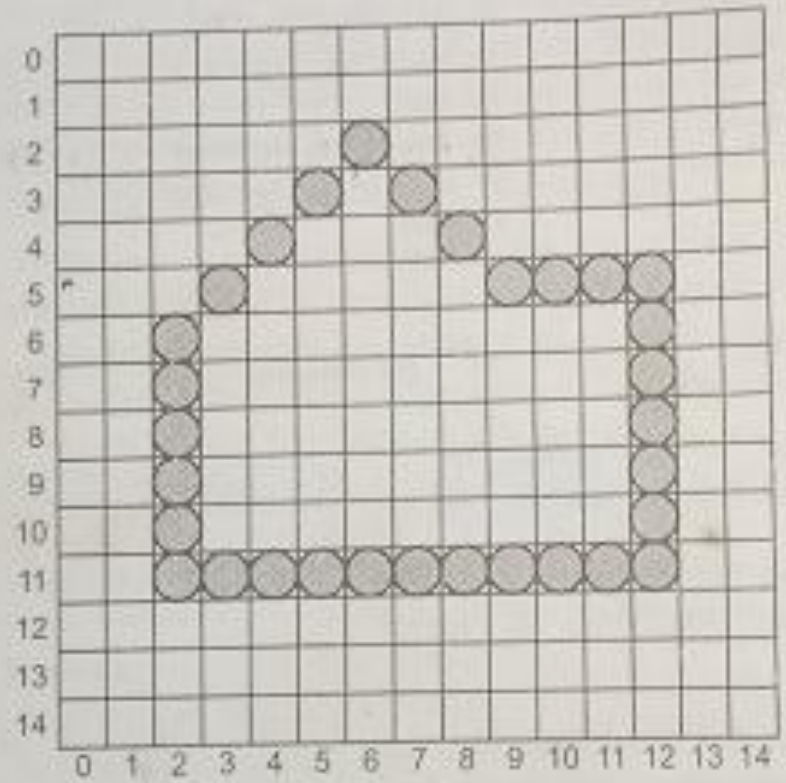
- **Run-length encoding** (**RLE**) is a form of lossless data compression in which *runs* of data (sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run.
- Consider a screen containing plain black text on a solid white background. There will be many long runs of white pixels in the blank space, and many short runs of black pixels within the text. A hypothetical scan line, with B representing a black pixel and W representing white, might read as follows:

- WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW
- With a run-length encoding (RLE) data compression algorithm applied to the above hypothetical scan line, it can be rendered as follows
- 12W1B12W3B24W1B14W

3.8.2 Run Length Encoding

In many pictures, we observe that the large number of pixels in the picture have the same intensity or colour. Considering this fact the picture can be defined by pixel intensity and number of successive pixels on given scan line with the same intensity. Such a representation of a picture is called **run length encoding**.

The Fig. 3.18 shows a simple picture on 15 × 15 raster and associated encoding for each scan line. It can be easily noticed that encoded data is in two groups : Intensity (I) and Run length (RL)



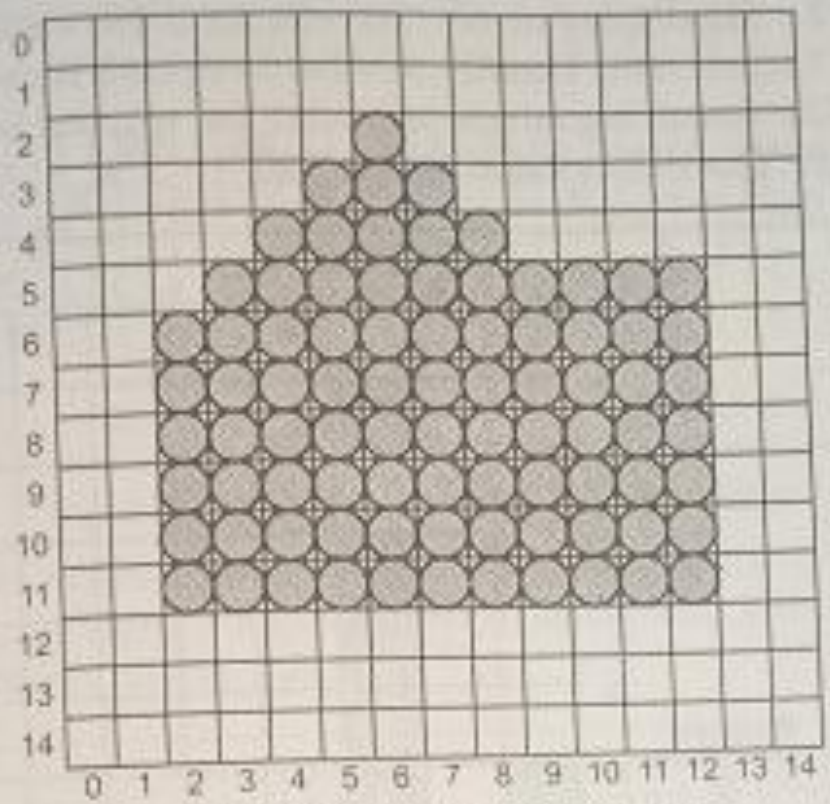
(a) Picture on 15 x 15 rasters

	I	RL	I	RL	I	RL	I	RL	I	RL
0	0	15								
1	0	15								
2	0	6	1	1	0	8				
3	0	5	1	1	0	1	1	1	0	7
4	0	4	1	1	0	3	1	1	0	6
5	0	3	1	1	0	5	1	4	0	2
6	0	2	1	1	0	9	1	1	0	2
7	0	2	1	1	0	9	1	1	0	2
8	0	2	1	1	0	9	1	1	0	2
9	0	2	1	1	0	9	1	1	0	2
10	0	2	1	1	0	9	1	1	0	2
11	0	2	1	11	0	2				
12	0	15								
13	0	15								
14	0	15								

(b) Run length encoded picture

The first number is the intensity, and the second is the number of successive pixels on that scan line with same intensity. In pixel-by-pixel storage, i.e., one piece of information for each pixel would require 225 intensity values for the 15 × 15 raster. However, with run length encoding the complete picture is encoded with only 102 numbers. Thus the data compression using run length encoding in this case is nearly 2.2 : 1.

The run length encoding technique can also be used for solid pictures. The Fig. 3.19 shows the solid picture and corresponding run length encoding.



(a) Solid picture on 15 x 15 rasters

I	RL	I	RL	I	RL
0	15				
0	15				
0	6	1	1	0	8
0	5	1	3	0	7
0	4	1	5	0	6
0	3	1	10	0	2
0	2	1	11	0	2
0	2	1	11	0	2
0	2	1	11	0	2
0	2	1	11	0	2
0	2	1	11	0	2
0	2	1	11	0	2
0	15				
0	15				
0	15				

(b) Run length encoded solid picture

Fig. 3.19

The run length encoding technique can also be extended to colour pictures. For colour pictures, the intensity of each of the red, green and blue colours is stored followed by the number of successive pixels having same colour combination on that scan line.

Therefore, in colour pictures the data is encoded in four groups : Red Intensity (RI), Green Intensity (GI), Blue Intensity (BI), and Run Length (RL).

Advantages

1. Picture is stored in compressed form with compression ratio approaching 10 : 1.
2. Less memory is required to store picture.
3. Saves storage space for computer generated animated sequences or films.
4. Saves the transmission time.

Disadvantages

1. Since the run lengths are stored sequentially, adding or deleting lines or text from the picture is difficult and time consuming.
2. It requires extra overhead in encoding and decoding the picture.
3. For short runs, expansion may result instead of compression.

Polygon filling

Scan-line polygon-fill algorithms

- For each scan-line that crosses the polygon, the edge intersections are sorted from left to right, and then pixel positions between and including each intersection pair are set to the specified fill color

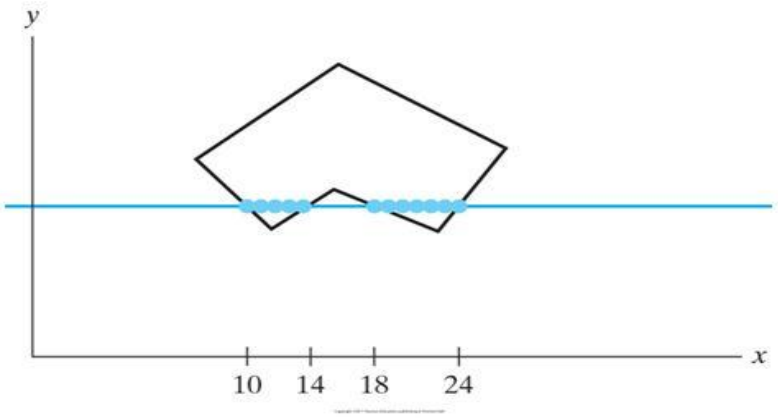
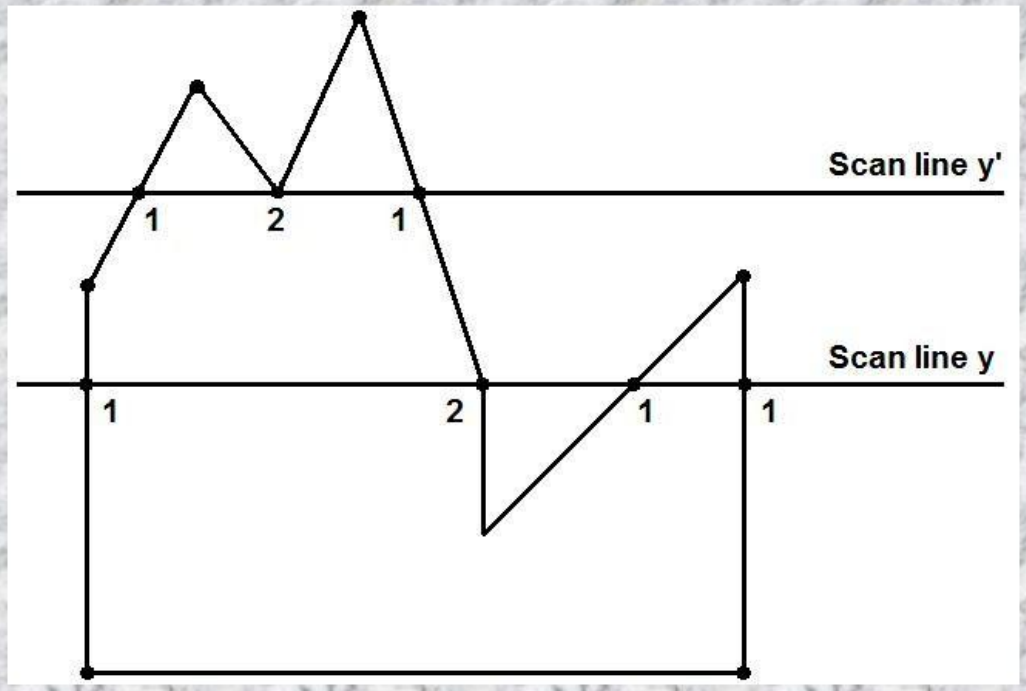


Figure 6-46 Interior pixels along a scan line passing through a polygon fill area.

The Scan-Line Polygon Fill Algorithm

Dealing with vertices



2–15 Seed Fill Algorithms

The algorithms discussed in the previous sections fill the polygon in scan line order. A different approach is used in the seed fill algorithms. The seed fill algorithms assume that at least one pixel interior to a polygon or region is known. The algorithm then attempts to find and color or fill all other pixels interior to the region. Regions may be either interior- or boundary-defined. If a region is interior-defined, then all the pixels in the interior of the region are one color or value, and all the pixels exterior to the region are another, as shown in Fig. 2–48. If a region is boundary-defined, then all the pixels on the region boundary are a unique value or color, as shown in Fig. 2–49. None of the pixels interior to the region can have this unique value. However, pixels exterior to the boundary can have the boundary value. Algorithms that fill interior-defined regions are referred to as flood fill algorithms, and those that fill boundary-defined regions are known as boundary fill algorithms. The discussion in this section concentrates on boundary fill algorithms. However, the companion flood fill algorithms are developed in an analogous manner.

Interior- or boundary-defined regions are either 4-connected or 8-connected. If a region is 4-connected, then every pixel in the region can be reached by a combination of moves in only four directions: left, right, up, down. For an 8-connected region, every pixel in the region is reached by a combination of moves in the two horizontal, two vertical and four diagonal directions. An 8-connected algorithm fills a 4-connected region, but a 4-connected algorithm does not fill an 8-connected region. Simple examples of 4-, and 8-connected interior-defined regions are shown in Fig. 2–50. Although each of the subregions of the 8-connected region shown in Fig. 2–50b is 4-connected, passage from one subregion to the other requires an 8-connected algorithm. However, if each of the subregions is a separate 4-connected region, each to be filled with a separate color or value, then use of an 8-connected algorithm causes both regions to be incorrectly filled with a single color or value.

A Simple Seed Fill Algorithm

A simple seed fill algorithm for a boundary-defined region can be developed using a stack. A stack is simply an array, or other storage space, into which values are sequentially placed, or from which they are sequentially removed. As new values are added to or pushed onto the stack, all previously stored values are pushed down one level. As values are removed or popped from the stack, previously stored values float or pop up one level. Such a stack is referred to as a first in, last out (FILO), or push-down, stack. A simple seed fill algorithm is then:

Simple seed fill algorithm using a stack:

Push the seed pixel onto the stack

while the stack is not empty

Pop a pixel from the stack

Set the pixel to the required value

For each of the 4-connected pixels adjacent to the current pixel, check if it is a boundary pixel, or if it has already been set to the

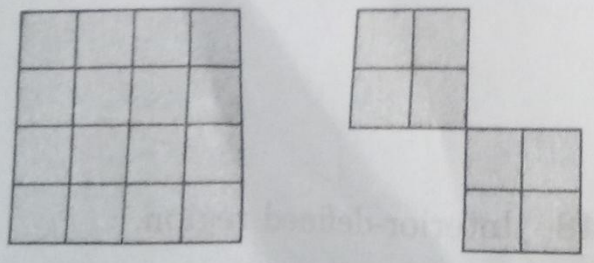


Figure 2–50 Four- and 8-connected interior-defined regions. (a) 4-connected; (b) 8-connected.

simple seed fill algorithm *for 4-connected boundary-defined regions*

```
Seed(x, y) is the seed pixel
Push is a function for placing a pixel on the stack
Pop is a function for removing a pixel from the stack
Pixel(x, y) = Seed(x, y)
initialize stack
Push Pixel(x, y)
while (stack not empty)
    get a pixel from the stack
    Pop Pixel(x, y)
    if Pixel(x, y) < > New value then
        Pixel(x, y) = New value
    end if
    examine the surrounding pixels to see if they should be placed
      onto the stack
    if (Pixel(x + 1, y) < > New value and
        Pixel(x + 1, y) < > Boundary value) then
        Push Pixel(x + 1, y)
    end if
    if (Pixel(x, y + 1) < > New value and
        Pixel(x, y + 1) < > Boundary value) then
        Push Pixel(x, y + 1)
    end if
    if (Pixel(x − 1, y) < > New value and
        Pixel(x − 1, y) < > Boundary value) then
        Push Pixel(x − 1, y)
```

```
        end if
        if (Pixel(x, y − 1) < > New value and
            Pixel(x, y − 1) < > Boundary value) then
            Push Pixel(x, y − 1)
        end if
    end while
```

Antialising

- In computer graphics, the process by which smooth curves and other lines become jagged because the resolution of the graphics device or file is not high enough to represent a smooth curve.

- In the line drawing algorithms, we have seen that all rasterized locations do not match with the true line and we have to select the optimum raster locations to represent a straight line.

- This problem is severe in low resolution screens. In such screens line appears like a stair-step, as shown in the figure below. This effect is known as **aliasing**. It is dominant for lines having gentle and sharp slopes

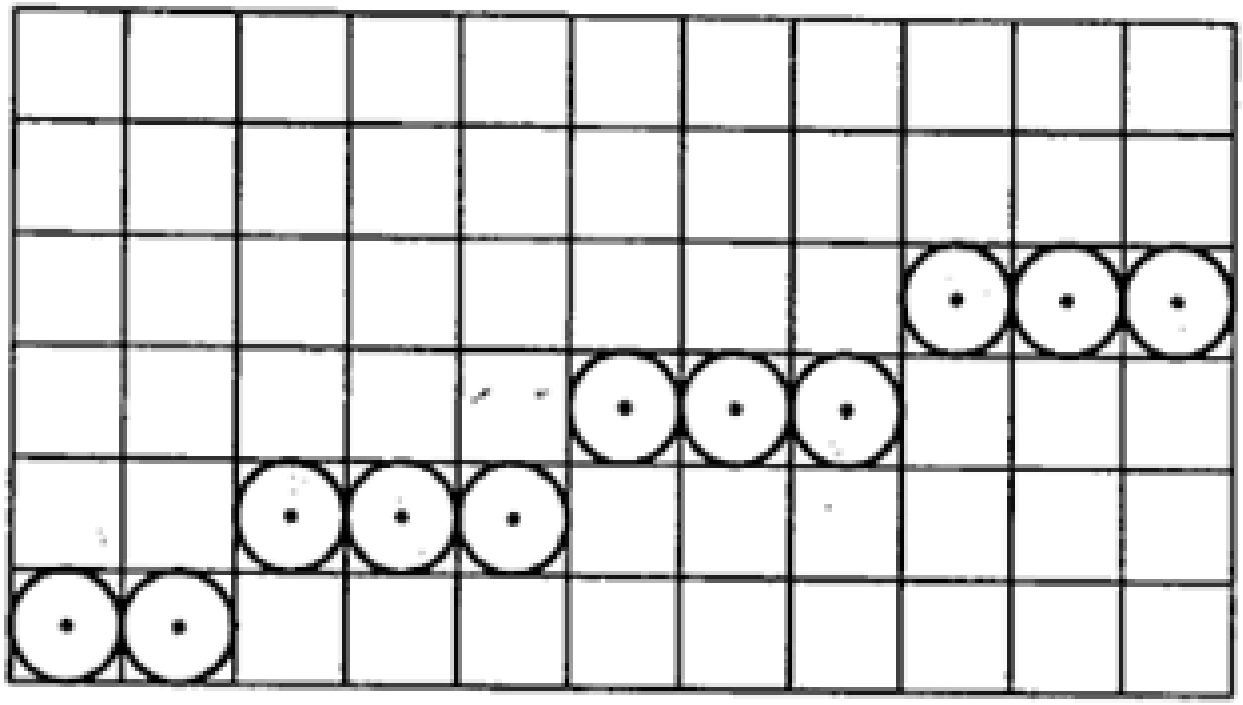


Fig. Aliasing effect

- The **aliasing effect** can be reduced by adjusting intensities of the pixels along the line. The process of adjusting intensities of the pixels along the line to minimize the effect of aliasing is called **antialiasing**.

- In computer graphics, *antialiasing* is a software technique for diminishing *jaggies* - stair step-like lines that should be smooth.

- Jaggies occur because the output device, the monitor or printer, doesn't have a high enough resolution to represent a smooth line.

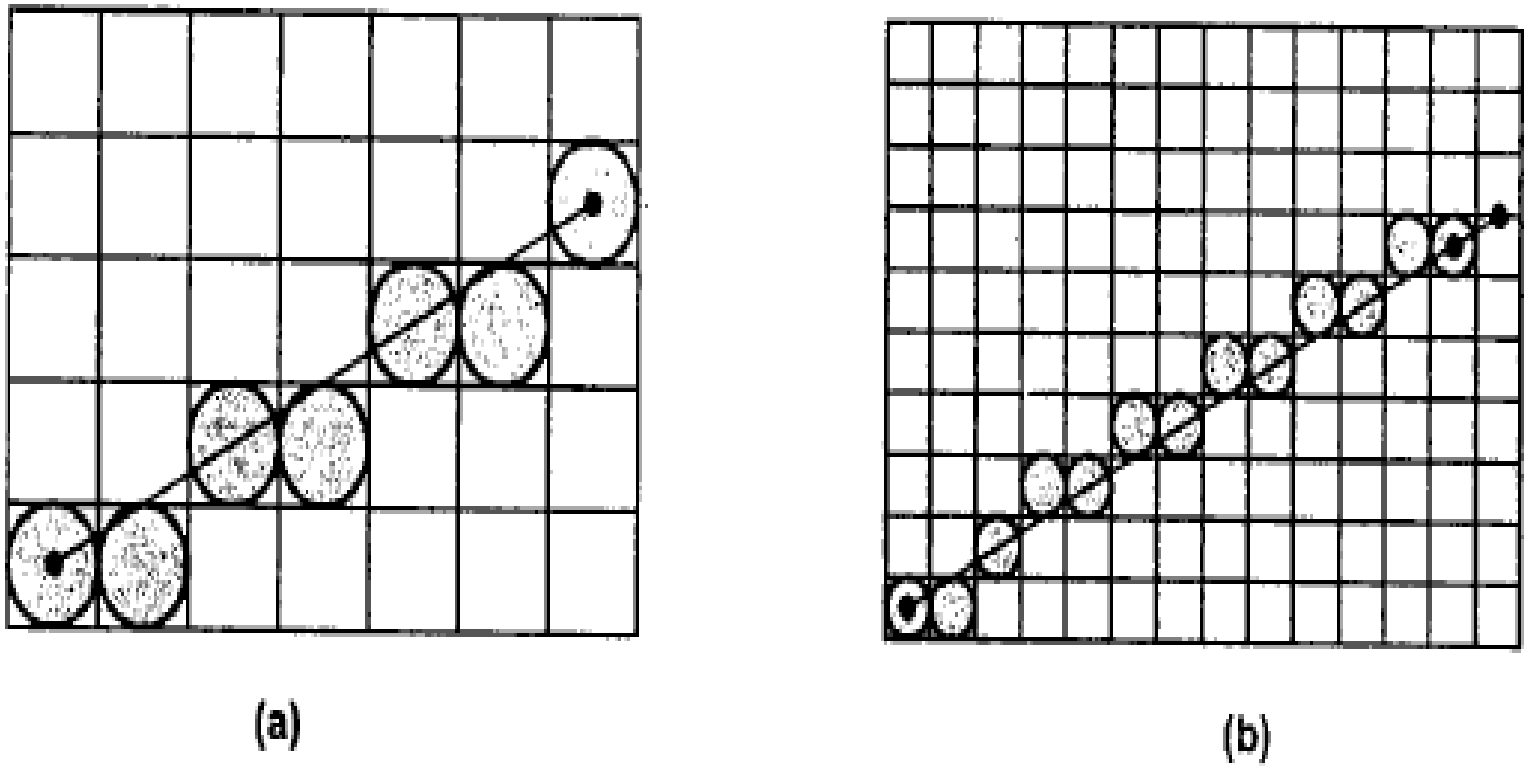
- Antialiasing reduces the prominence of jaggies by surrounding the stair steps with intermediate shades of gray or color (for color devices).

- Although this reduces the jagged appearance of the lines, it also makes them fuzzier.

- Another method for reducing jaggies is called *smoothing*, in which the printer changes the size and horizontal alignment of dots to make curves smoother.

- The aliasing effect can be minimized by increasing resolution of the raster display.

- By increasing resolution and making it twice the original one, the line passes through twice as many column of pixels and therefore has twice as many **jags**, but each jag is half as large in x and in y direction.



(a) (b)

Fig. Effect on aliasing with increase in resolution

Antialiasing methods

- **Using high-resolution display:**
- One way to reduce aliasing effect and increase sampling rate is to simply display objects at a higher resolution. Using high resolution, the jaggies become so small that they become indistinguishable by the human eye. Hence, jagged edges get blurred out and edges appear smooth.

- **Practical applications:**
  For example retina displays in Apple devices, OLED displays have high pixel density due to which jaggies formed are so small that they blurred and indistinguishable by our eyes.

- **Supersampling or Postfiltering:-**
- In this method, we are increasing the sampling resolution by treating the screen as if it's made of a much more fine grid, due to which the effective pixel size is reduced.
- But the screen resolution remains the same. Now, intensity from each subpixel is calculated and average intensity of the pixel is found from the average of intensities of subpixels.
- Thus we do sampling at higher resolution and display the image at lower resolution or resolution of the screen, hence this technique is called supersampling.
- This method is also known as post filtration as this procedure is done after generating the rasterized image.

- **Practical applications:**
  In gaming, SSAA (Supersample Antialiasing) or FSAA (full-scene antialiasing) is used to create best image quality. It is often called the pure AA and hence is very slow and has a very high computational cost

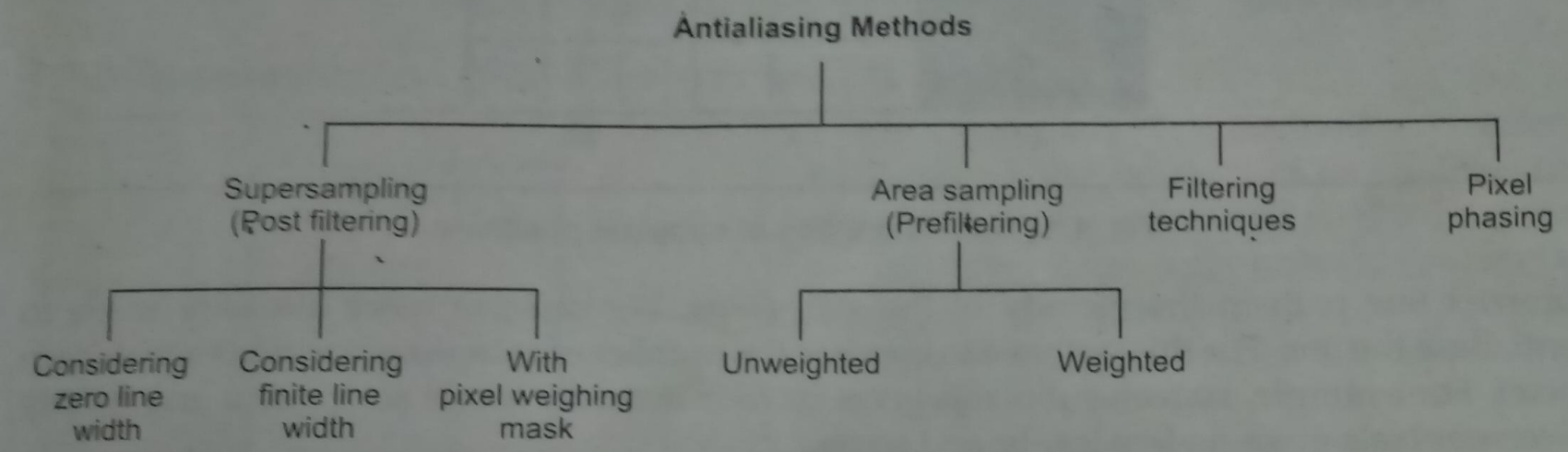


Fig. 1.15 Classification of antialiasing methods

1.7.1 Supersampling Considering Zero Line Width

Super sampling of straight lines with zero line width can be performed in several ways. For the gray scale display of a straight-line segment, we can divide each pixel into a number of subpixels and count the number of subpixels that are along the line path. The intensity level for each pixel is then set to a value that is proportional to this subpixel count. This is illustrated in Fig. 1.16. Here each pixel area is divided into nine equal-sized square subpixels. The black regions show the subpixels that would be selected by Bresenham's algorithm. In this example, the pixel at position (20, 30) has 3 subpixels along the line path, the pixel at position (21, 31) has 2 subpixels along the line path and the pixel at position (22, 32) has 1 subpixel along the line path. Since pixel at position (20, 30) has 3 (maximum)

subpixels its intensity level is kept highest (level 3). The intensity level at pixel position (21, 31) is kept to the medium level (level 2) because pixel has 2 subpixels along the line path, and the intensity at pixel position (22, 32) is kept to the lowest level above zero intensity because it has only 1 subpixel along the line path. Thus,the line intensity is spread out over a greater number of pixels, and the stair-step effect is smoothed by displaying a somewhat

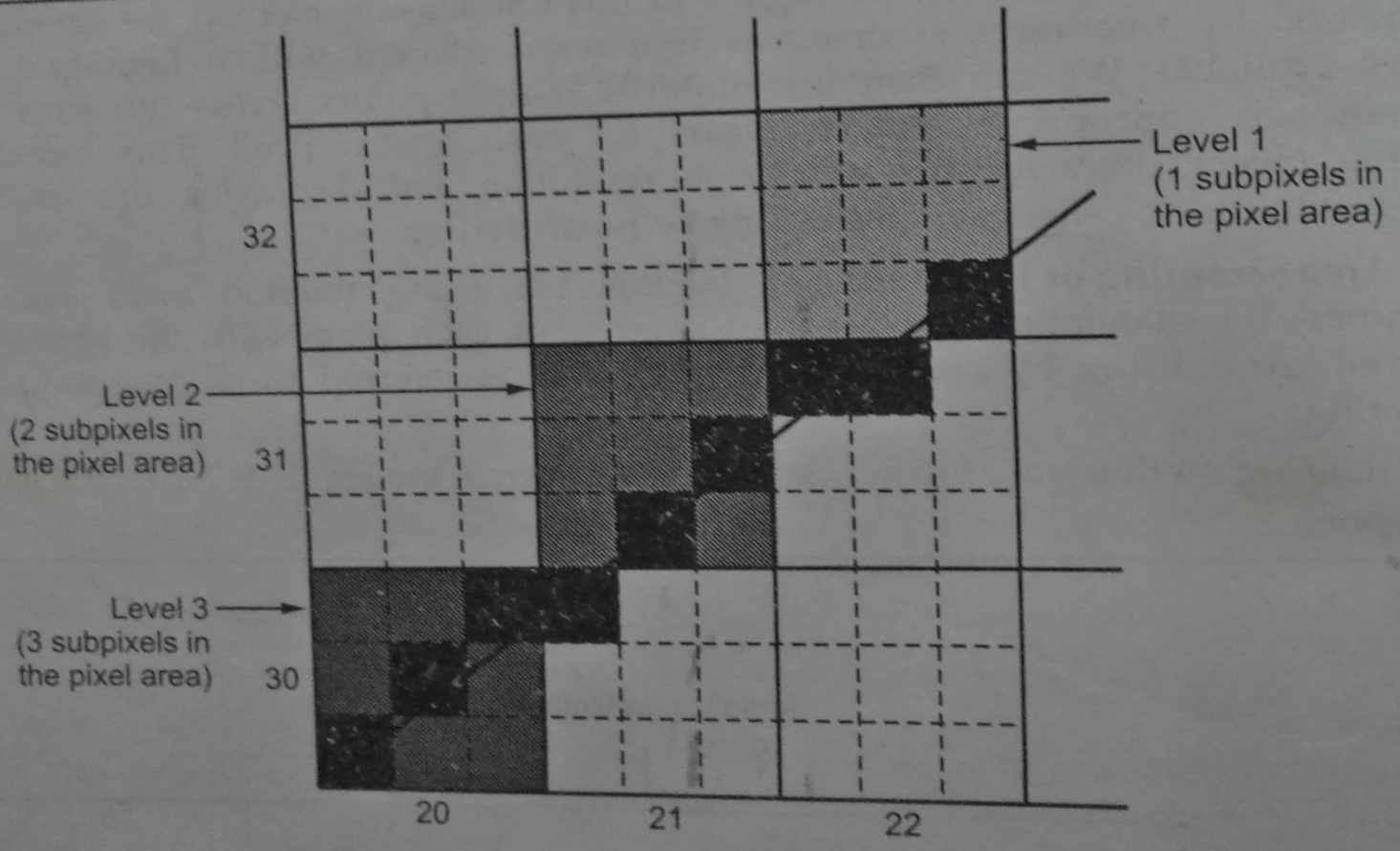
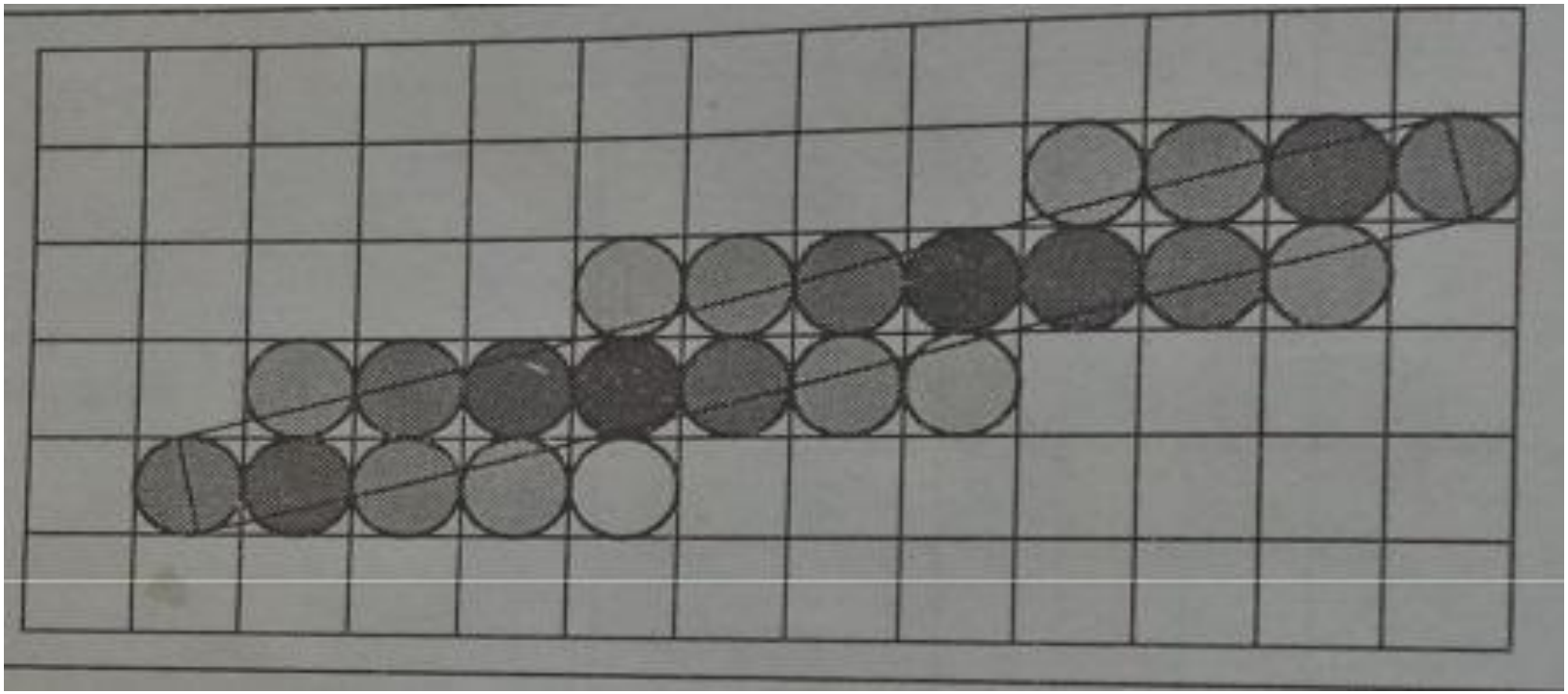


Fig. 1.16 Supersampling of subpixel positions

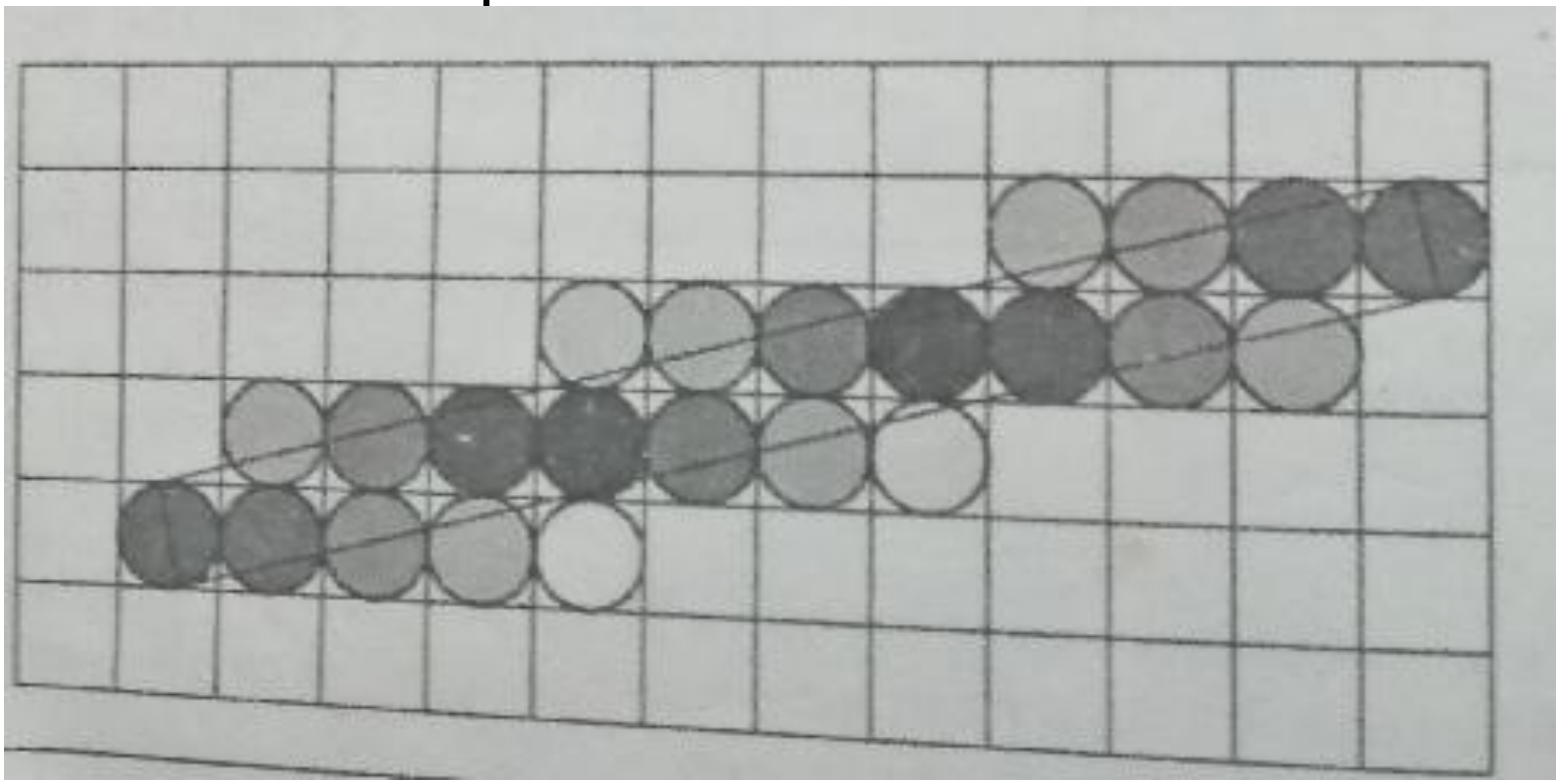
- **Area sampling or Prefiltering:-**
- In area sampling, pixel intensities are calculated proportional to areas of overlap of each pixel with objects to be displayed. Here pixel color is computed based on the overlap of scene's objects with a pixel area.

  For example: Suppose, a line passes through two pixels. The pixel covering bigger portion(90%) of line displays 90% intensity while less area(10%) covering pixel displays 10-15% intensity. If pixel area overlaps with different color areas, then the final pixel color is taken as an average of colors of the overlap area. This method is also known as pre-filtering as this procedure is done BEFORE generating the rasterized image. It's done using some graphics primitive algorithms.

- Unweighted area sampling:-
- In this area sampling, the intensity of pixel is proportional to the amount of line area occupied by the pixel. This technique produces noticeably better results than does setting pixels either to full intensity or to zero intensity



- Weighted area sampling:-
- In this area sampling equal areas contribute unequally i.e. A small area closer to the pixel center has greater intensity than does one at a greater distance. Thus in weighted area sampling the intensity of the pixel is dependent on the line area occupied and the distance of area from the pixels center.



Halftoning

- The process of generating a binary pattern of black and white dots from an image is termed as halftoning.

- A halftone, or halftone image, is an image comprised of discrete dots rather than continuous tones.

- When viewed from a distance, the dots blur together, creating the illusion of continuous lines and shapes.

- By halftoning an image (converting it from a bitmap to a halftone), it can be printed using less ink.

- Therefore, many newspapers and magazines use halftoning to print pages more efficiently.

- Originally, halftoning was performed mechanically by printers that printed images through a screen with a grid of holes. During the printing process, ink passed through the holes in the screen, creating dots on the paper.

- For monochrome images, only one pass was needed to create an image. For multicolor images, several passes or "screens" were required.

- Today's printers are more advanced and typically do not contain physical screens. Instead, the halftone images are generated by a computer and the resulting image is printed onto the paper

- By using a process called dithering, modern printers can randomize the dot patterns, creating a more natural appearance.
- Dithering is the process by which we create illusions of the color that are not present actually. It is done by the random arrangement of pixels.

- This produces realistic images using far less ink than fully saturated ones.

- Like a standard bitmap, the quality of a halftone image depends largely on the its resolution.

- A halftone with a high resolution (measured in LPI), will have greater detail than a halftone with a low resolution.

- While the goal of halftoning is typically to create a realistic image, sometimes low resolutions are used for an artistic effect.

- Halftone images are surprisingly simple in practice.

- Originally, they were designed to create images that only required one color of ink (usually black), but still needed to offer a good range of tones.

- For example, black and white photography consists of only one color of ink: true black.

- This means that when viewed under a microscope, there will only be two colors present in any black and white picture: black and white.

- The range of different grays we see is actually produced by varying the size of individual dots of ink, and these dots are aptly named halftone dots.

- Halftone dots are small dots of ink — sometimes circular, elliptical, or even square — that are spaced evenly across areas of a black and white image.
- By changing the size of the dot, you change the saturation of ink, and therefore can lighten or darken the tones as needed.
- Smaller dots produce lighter grays and off-whites, while larger dots can lead to dark gray or black areas in an image.
- Not only does this save on ink and reduce the number of expensive colors of ink, it's actually interesting from an optical illusion standpoint.
- Because the human eye can only see tiny dots up close, halftone images from a distance appear to have a rich depth of tones from true white to true black, as well as many shades of gray in between

- Like most things, the original halftone images were of lower quality, but thanks to advancements in technology, halftone images can provide a rich viewing experience.

- Printers today actually use the same concept when it comes to printing in colors.
- By using the CYMK color palette which consists of only four inks – cyan, yellow, magenta, and key (black) – printers can layer together different colors of dots to produce millions of colors and a rich, crisp image.

- Halftone printing can be tailored to fit the needs of the budget, design, or preference of the person printing.

- By using smaller, less densely spaced halftone dots, you can provide enough visual interest in an image without using a lot of inks which can be expensive, therefore cutting the cost dramatically.

- This is a common method used in newspaper, diagram and map printing.

- By increasing the DPI, or dots per inch, you can create rich and detailed images, which can be used for photographs and artwork displayed in newspapers, brochures, or on fliers

Dithering

- Dithering is the process by which we create illusions of the color that are not present actually. It is done by the random arrangement
- Full-color photographs may contain an almost infinite range of color values. Dithering is the most common means of reducing the color range of images down to the 256 (or fewer) colors seen in 8-bit GIF images.

- Dithering is the process of pairing pixels of two colors to create the illusion that a third color is present.

- A simple example is an image with only black and white in the color palette. By combining black and white pixels in complex patterns a graphics program like Adobe Photoshop can create the illusion of gray values:


Full-color image
dithered
to two colors

The same process softens the effect of reducing the number of colors in full-color images

Original full-color photograph

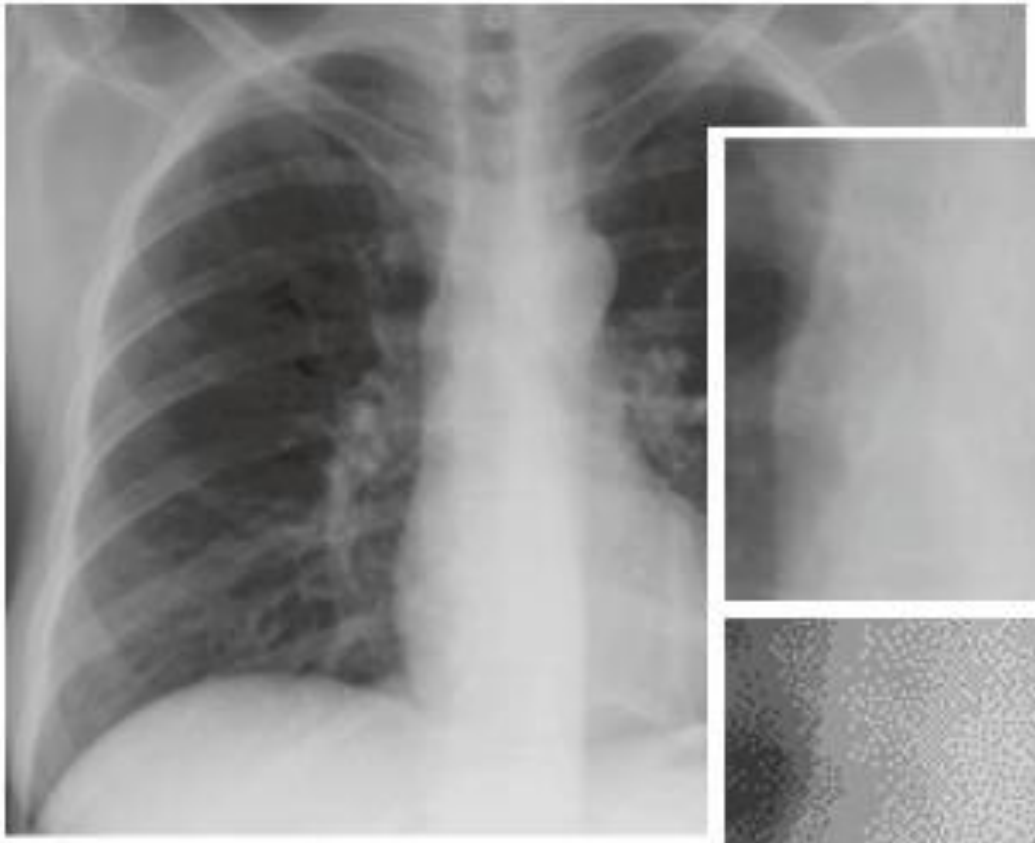
Dithered to 256 colors

- Most images are dithered in a diffusion or randomized pattern to diminish the harsh transition from one color to another.

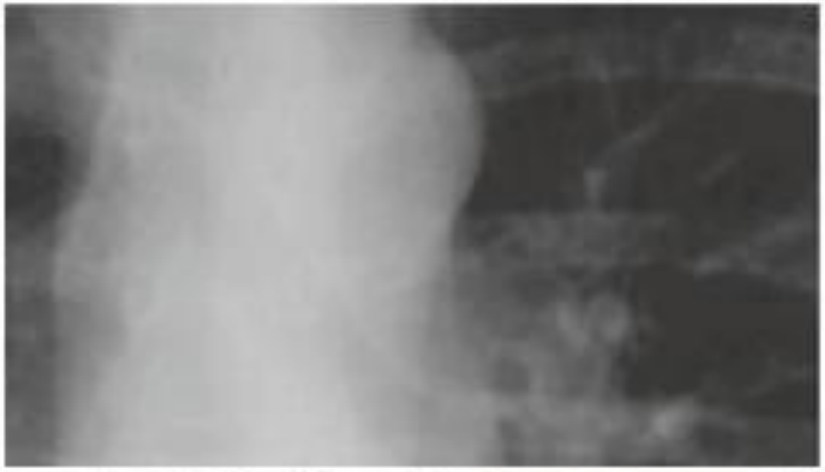
- But dithering also reduces the overall sharpness of an image, and it often introduces a noticeable grainy pattern in the image.

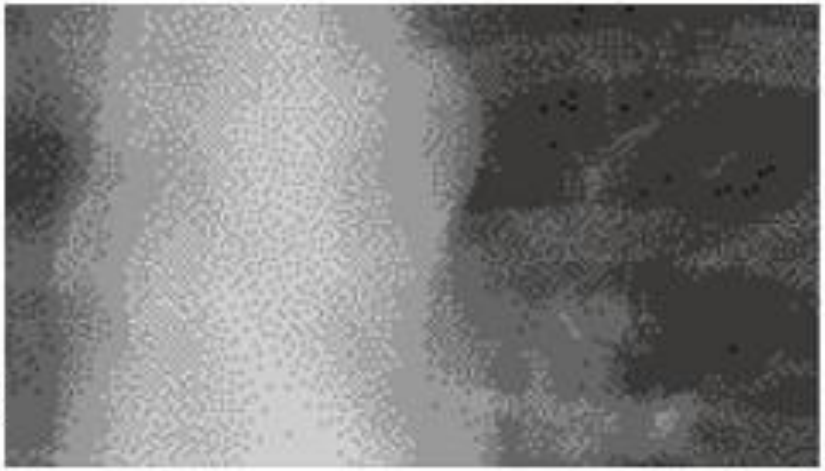
- This loss of image detail is especially apparent when full-color photos are dithered down to the 216-color browser-safe palette:.

Original full-tone image



Detail of original image





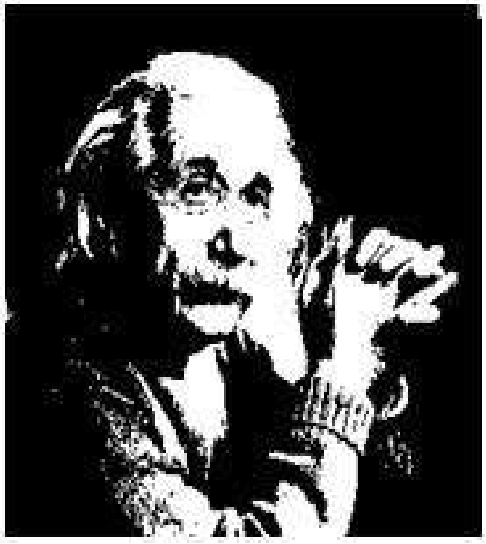
Dithered image shows loss of tone and loss of image detail

Performing dithering

First of all, we will work on threholding. Dithering is usually working to improve thresholding. During threholding, the sharp edges appear where gradients are smooth in an image.

In thresholding, we simply choose a constant value. All the pixels above that value are considered as 1 and all the value below it are considered as 0.

We got this image after thresholding.



Since there is not much change in the image, as the values are already 0 and 1 or black and white in this image.

Now we perform some random dithering to it. Its some random arrangement of pixels.

We got an image that gives slighter of the more details, but its contrast is very low.

So we do some more dithering that will increase the contrast. The image that we got is this:

Now we mix the concepts of random dithering, along with threshold and we got an image like this.

Now you see, we got all these images by just re-arranging the pixels of an image. This re-arranging could be random or could be according to some measure.

Clipping

Clipping is the process of extracting a portion of a data base or identifying elements of a scene or picture inside or outside a specified region, called the clipping region. Clipping is fundamental to several aspects of computer graphics. In addition to its more typical use in selecting only the specific information required to display a particular scene or view from a larger environment, Chapter 2 showed

3–1 Two-dimensional Clipping

Figure 3–1 shows a two-dimensional scene and a regular clipping window. It is defined by left (L), right (R), top (T) and bottom (B) two-dimensional edges. A regular clipping window is rectangular, with its edges aligned with those of the object space or display device. The purpose of a clipping algorithm is to determine which points, lines or portions of lines lie within the clipping window. These points, lines or portions of lines are retained for display; all others are discarded.

Because large numbers of points or lines must be clipped for a typical scene or picture, the efficiency of clipping algorithms is of particular interest. In many cases, the large majority of points or lines are either completely interior to or completely exterior to the clipping window. Therefore, it is important to be able to quickly accept a line like ab or a point like p, or reject a line like ij or a point like q in Fig. 3–1.

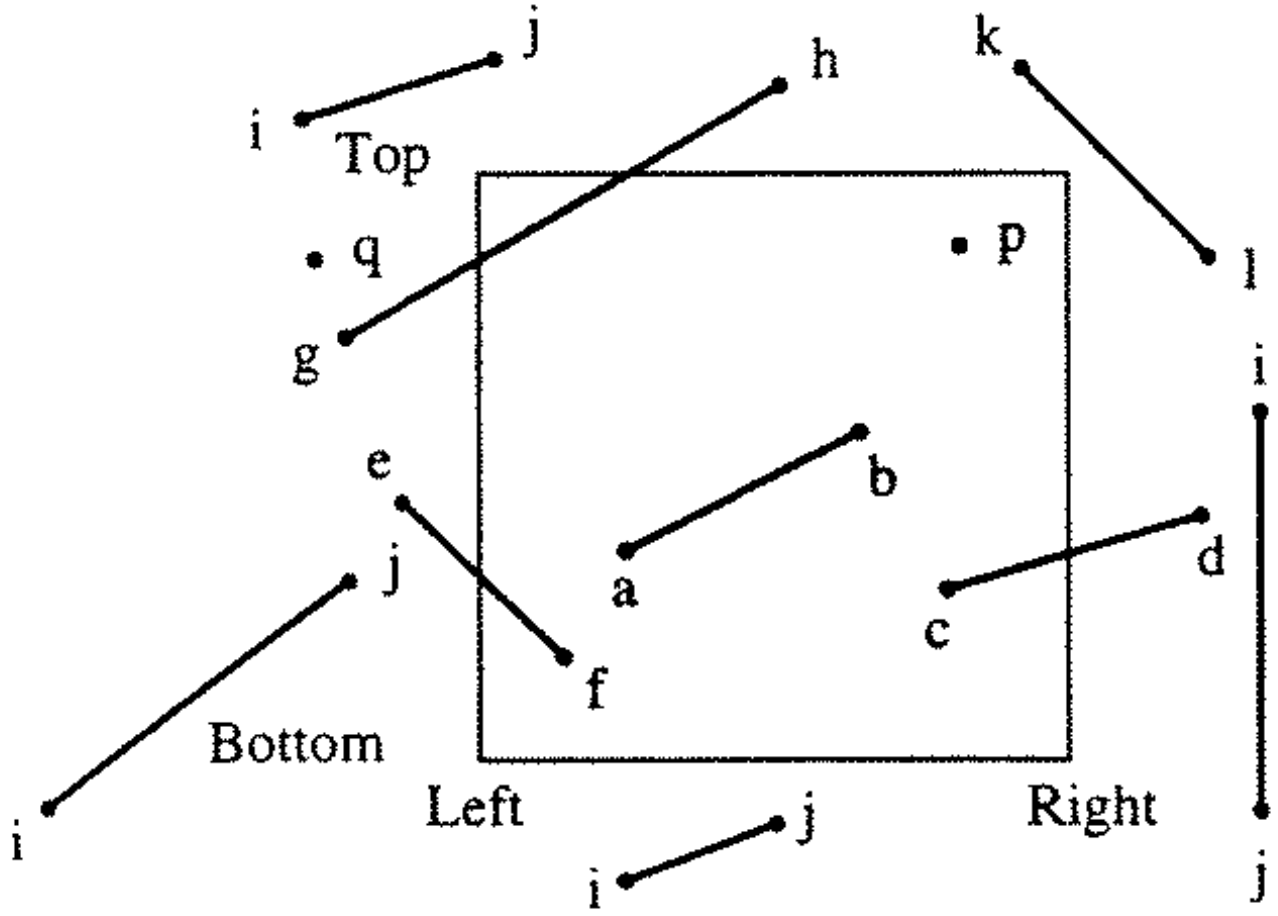


Figure 3–1 Two-dimensional clipping window.

A Simple Visibility Algorithm

Points are interior to the clipping window provided that

$$x_L \leq x \leq x_R \qquad \text{and} \qquad y_B \leq y \leq y_T$$

The equal sign indicates that points on the window boundary are included within the window.

Lines are interior to the clipping window and hence visible if both end points are interior to the window, e.g., line *ab* in Fig. 3–1. However, if both end points of a line are exterior to the window, the line is not necessarily completely exterior to the window, e.g., line *gh* in Fig. 3–1. If both end points of a line are completely to the right of, completely to the left of, completely above or completely below the window, then the line is completely exterior to the window and hence invisible. This test eliminates all the lines labeled *ij* in Fig. 3–1. It does not eliminate either line *gh*, which is partially visible, or line *kl*, which is totally invisible.

If *a* and *b* are the end points of a line, then an algorithm for identifying completely visible and most invisible lines is:

simple visibility algorithm

a *and* b *are the end points of the line, with components* x *and* y

for each line

Visibility = True

check for totally invisible lines

if both end points are left, right, above or below the window, the line is trivially invisible

if $x_a < x_L$ **and** $x_b < x_L$ **then** Visibility = False
if $x_a > x_R$ **and** $x_b > x_R$ **then** Visibility = False
if $y_a > y_T$ **and** $y_b > y_T$ **then** Visibility = False
if $y_a < y_B$ **and** $y_b < y_B$ **then** Visibility = False

if Visibility <> False **then** *avoid the totally visible calculation*

check if the line is totally visible

if any coordinate of either end point is outside the window, then the line is not totally visible

```
        if x_a < x_L or x_a > x_R then Visibility = Partial
        if x_b < x_L or x_b > x_R then Visibility = Partial
        if y_a < y_B or y_a > y_T then Visibility = Partial
        if y_b < y_B or y_b > y_T then Visibility = Partial
    end if
    if Visibility = Partial then
        the line is partially visible or diagonally crosses a corner invisibly
        determine the intersections and the visibility of the line
    end if
    if Visibility = True then
        line is totally visible — draw line
    end if
    line is invisible
    next line
finish
```

End Point Codes

The tests for totally visible lines and the region tests given above for totally invisible lines are formalized using a technique due to Dan Cohen and Ivan Sutherland. The technique uses a 4 bit (digit) code to indicate which of nine regions contains the end point of a line. The 4 bit codes are shown in Fig. 3–2. The rightmost bit is the first bit. The bits are set to 1, based on the following scheme:

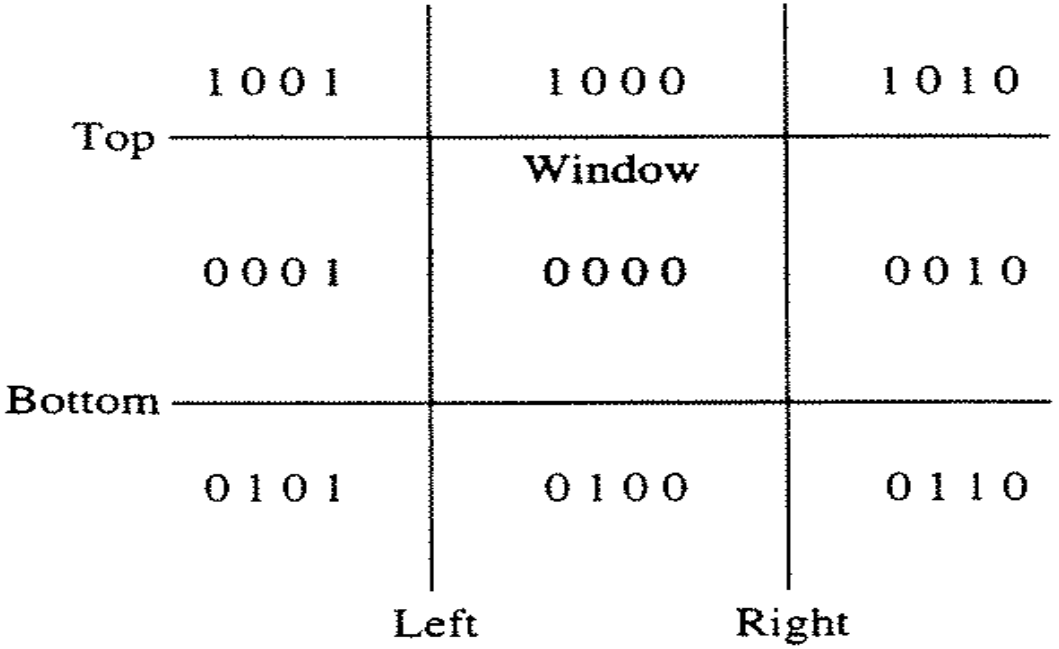
First-bit set	—	if the end point is to the left of the window
Second-bit set	—	if the end point is to the right of the window
Third-bit set	—	if the end point is below the window
Fourth-bit set	—	if the end point is above the window

Otherwise, the bit is set to zero. From this it is obvious that, if both end point codes are *zero*, then both ends of the line lie inside the window; and the line is visible. The end point codes are also used to trivially reject totally invisible lines. Consider the truth table for the logical **and** operator:

True and False	→	False			1 and 0	→ 0
			False = 0			
False and True	→	False			0 and 1	→ 0
			→			
False and False	→	False			0 and 0	→ 0
			True = 1			
True and True	→	True			1 and 1	→ 1

end point code algorithm

P_1 *and* P_2 *are the end points of the line*



x_L, x_R, y_T, y_B *are the left, right, top and bottom window coordinates*

calculate the end point codes

put the codes for each end into 1 × 4 arrays called P1code *and* P2code

first end point: P_1

if $x_1 < x_L$ **then** P1code(4) = 1 **else** P1code(4) = 0
if $x_1 > x_R$ **then** P1code(3) = 1 **else** P1code(3) = 0
if $y_1 < y_B$ **then** P1code(2) = 1 **else** P1code(2) = 0
if $y_1 > y_T$ **then** P1code(1) = 1 **else** P1code(1) = 0

second end point: P_2

if $x_2 < x_L$ **then** P2code(4) = 1 **else** P2code(4) = 0
if $x_2 > x_R$ **then** P2code(3) = 1 **else** P2code(3) = 0
if $y_2 < y_B$ **then** P2code(2) = 1 **else** P2code(2) = 0
if $y_2 > y_T$ **then** P2code(1) = 1 **else** P2code(1) = 0

finish

Table 3–1 End Point Codes

Line (see Fig. 3–1)	End point codes (see Fig. 3–2)		Logical and	Comments
ab	0000	0000	0000	Totally visible
ij	0010	0110	0010	Totally invisible
ij	1001	1000	1000	Totally invisible
ij	0101	0001	0001	Totally invisible
ij	0100	0100	0100	Totally invisible
cd	0000	0010	0000	Partially visible
ef	0001	0000	0000	Partially visible
gh	0001	1000	0000	Partially visible
kl	1000	0010	0000	Totally invisible

The intersection between two lines can be determined either parametrically or nonparametrically. Explicitly, the equation of the infinite line through $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ is

$$y = m(x - x_1) + y_1 \qquad \text{or} \qquad y = m(x - x_2) + y_2$$

where

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

is the slope of the line. The intersections with the window edges are given by

Left:	$x_L, y = m(x_L - x_1) + y_1$	$m \neq \infty$
Right:	$x_R, y = m(x_R - x_1) + y_1$	$m \neq \infty$
Top:	$y_T, x = x_1 + (1/m)(y_T - y_1)$	$m \neq 0$
Bottom:	$y_B, x = x_1 + (1/m)(y_B - y_1)$	$m \neq 0$

Example 3–1 Explicit Line–Window Intersections

Consider the clipping window and the lines shown in Fig. 3–3. For the line from $P_1(3/2, 1/6)$ to $P_2(1/2, 3/2)$ the slope is

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{3/2 - 1/6}{1/2 - (-3/2)} = \frac{2}{3}$$

and the intersections with the window edge are

$$\begin{aligned}
\text{Left:} \quad & x = -1 \quad & y &= 2/3\,[-1 - (-3/2)] + 1/6 \\
& & &= 1/2 \\
\text{Right:} \quad & x = 1 \quad & y &= 2/3\,[1 - (-3/2)] + 1/6 \\
& & &= 11/6 \\
\text{Top:} \quad & y = 1 \quad & x &= -3/2 + 3/2\,[1 - 1/6] \\
& & &= -1/4 \\
\text{Bottom:} \quad & y = -1 \quad & x &= -3/2 + 3/2\,[-1 - 1/6] \\
& & &= -13/4
\end{aligned}$$

Similarly, for the line from $P_3(-3/2, -1)$ to $P_4(3/2, 2)$

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{2 - (-1)}{3/2 - (-3/2)} = 1$$

and

$$\begin{array}{lll}
\text{Left:} & x = -1 & y = (1)[-1 - (-3/2)] + (-1) \\
 & & \quad = -1/2 \\
\text{Right:} & x = 1 & y = (1)[1 - (-3/2)] + (-1) \\
 & & \quad = 3/2 \\
\text{Top:} & y = 1 & x = -3/2 + (1)[1 - (-1)] \\
 & & \quad = 1/2 \\
\text{Bottom:} & y = -1 & x = -3/2 + (1)[-1 - (-1)] \\
 & & \quad = -3/2
\end{array}$$

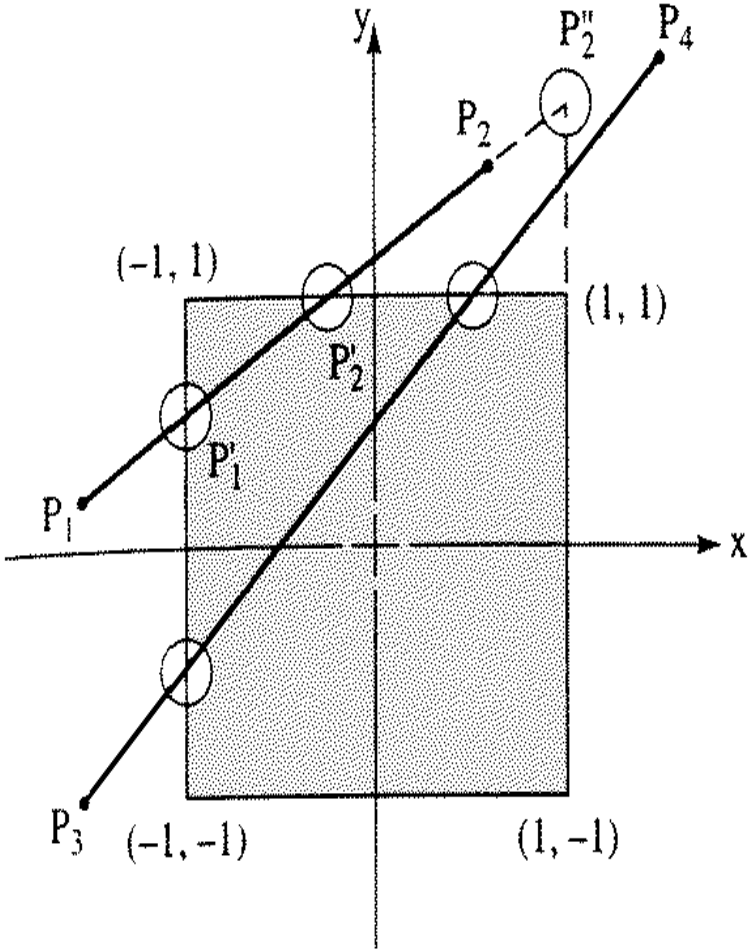


Figure 3–3 Two-dimensional parametric clipping.

Sutherland Cohen Line clipping Algorithm

For the line P_1P_2, determine if the line is totally visible or can be trivially rejected as invisible.

If P_1 is outside the window, continue; otherwise, swap P_1 and P_2.

Replace P_1 with the intersection of P_1P_2 and the window edge.

Example 3–2 Cohen–Sutherland Clipping

Again consider the line P_1P_2 clipped against the window shown in Fig. 3–3. The end point codes for $P_1(-3/2, 1/6)$ and $P_2(1/2, 3/2)$ are (0001) and (1000), respectively. The end point codes are not simultaneously zero, and the logical and of the end point codes is zero. Consequently, the line is neither totally visible nor trivially invisible. Comparing the first bits of the end point codes, the line crosses the left edge; and P_1 is outside the window.

The intersection with the left edge ($x = -1$) of the window is $P_1'(-1, 1/2)$. Replace P_1 with P_1' to yield the new line, $P_1(-1, 1/2)$ to $P_2(1/2, 3/2)$.

The end point codes for P_1 and P_2 are now (0000) and (1000), respectively. The line is neither totally visible nor trivially invisible.

Comparing the second bits, the line does not cross the right edge; skip to the bottom edge.

The end point codes for P_1 and P_2 are still (0000) and (1000), respectively. The line is neither totally visible nor trivially invisible.

Comparing the third bits, the line does not cross the bottom edge. Skip to the top edge.

The end point codes for P_1 and P_2 are still (0000) and (1000), respectively. The line is neither totally visible nor trivially invisible.

Comparing the fourth bits, the line crosses the top edge. P_1 is not outside. Swap P_1 and P_2 to yield the new line, $P_1(1/2, 3/2)$ to $P_2(-1, 1/2)$.

The intersection with the top edge ($y = 1$) of the window is $P_1'(-1/4, 1)$. Replace P_1 with P_1' to yield the new line, $P_1(-1/4, 1)$ to $P_2(-1, 1/2)$.

The end point codes for P_1 and P_2 are (0000) and (0000), respectively. The line is totally visible.

The procedure is complete.

Draw the line ($P_1'P_2'$ in Fig. 3–3).

Curves and surfaces

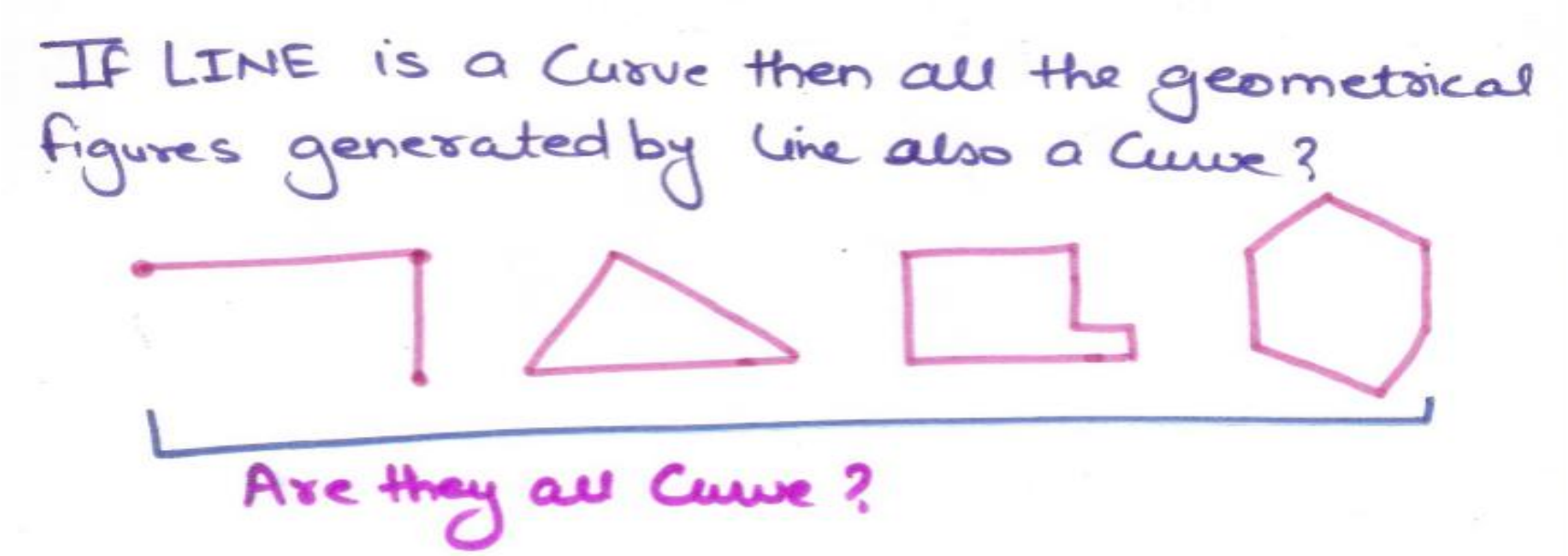
introduction

- In computer graphics, we often need to draw different types of objects onto the screen. Objects are not flat all the time and we need to draw curves many times to draw an object.
- There are lots of definitions for curve but we will focus on main two of them.

- 1..When set of points finite or infinite are joined together then what we get is called as curve

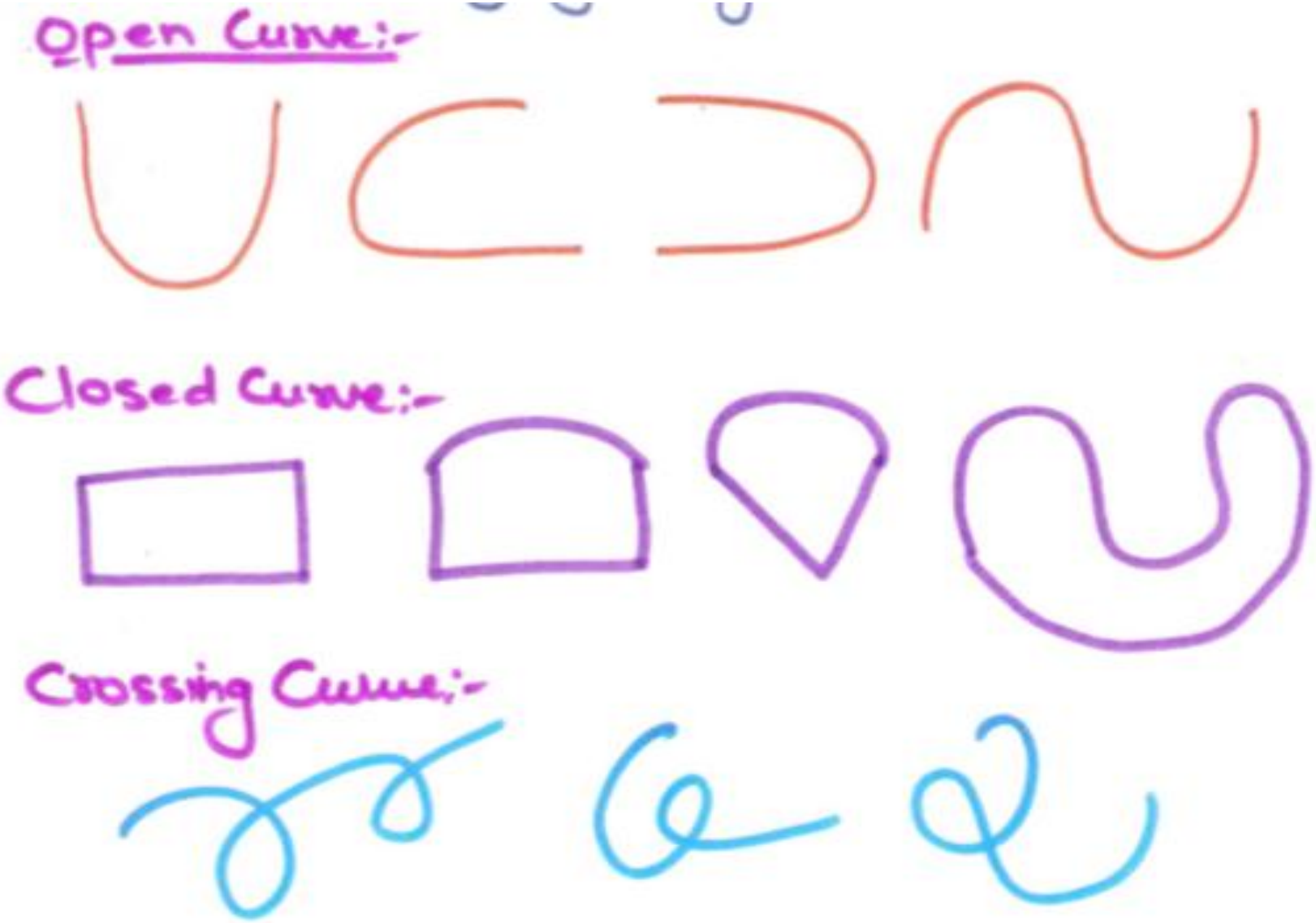
- 2.. When we start from a point for drawing a geometrical figure and end at some other point without any gap then what we get is called as curve

- As per definition is line a curve?
- Yes, mathematically line is also a curve...



- Circle is also a curve....

- There are many types of curves....



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

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

- **Ex.. $Y= 5x^2+2x+1$**

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

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

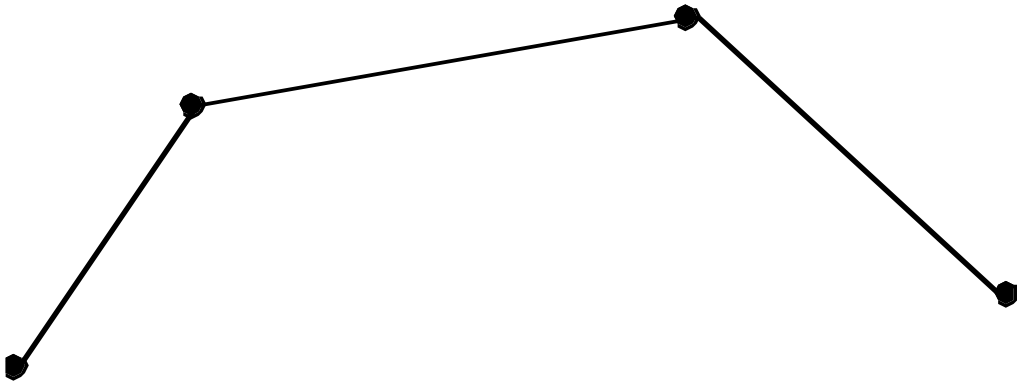
- It can represent multivalued curves (multiple y values for an x value). A common example is the circle, whose implicit representation is

$$\mathbf{x^2 + y^2 - R^2 = 0}$$

- An explicit function is one which is given in terms of the independent variable.
- Example.. $X^2+y^2-1=0$, $y^4+x^3+18=0$

- **Parametric Curves**
- Curves having parametric form are called parametric curves.
- There are many curves which we cannot write down as a single equation in terms of only x and y.
- Instead of defining y in terms of x (y=f(x)) or x in terms of y (x=f(y)), we define both the x and y in terms of a third variable called a parameter.
- example.. x= fx(u) y=fy(u)
- Line parametric equation is
- $X= (1-u)x_0 + ux_1$
- $Y= (1-u)\, y_0 + uy_1$

“Computers can’t draw curves.”

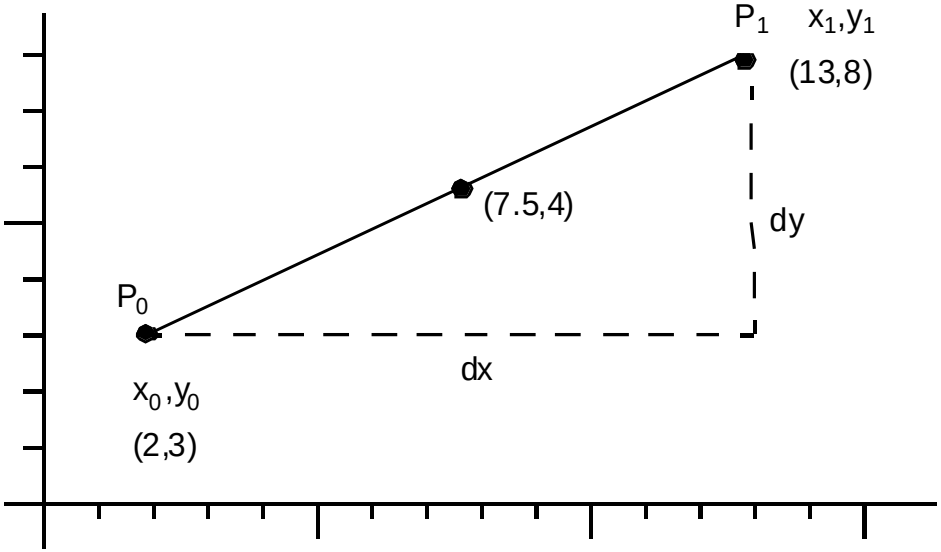


The more points/line segments that are used, the smoother the curve.

Why have curves ?

- Representation of “irregular surfaces”
- Example: Auto industry (car body design)
  - Artist’s representation
  - Clay / wood models
  - Digitizing
  - Surface modeling (“body in white”)
  - Scaling and smoothening
  - Tool and die Manufacturing

Parametric Equations - linear



Walk from P_0 to P_1 at a constant speed.

Start from P_0 at t=0

Arrive at P_1 at t=1

$dx = x_1 - x_0$

$dy = y_1 - y_0$

Where are you at a general time t?

Equation(s) of a straight line as a function of an arbitrary parameter t.

$x(t) = x_0 + t.dx$

$y(t) = y_0 + t.dy$

- Multiple techniques are available for drawing and designing curves manually
- A wide verity of pencils, pens, brushes, along with straight edges, French curves, compasses, templates.
- Each tool has its own function

Curve generation

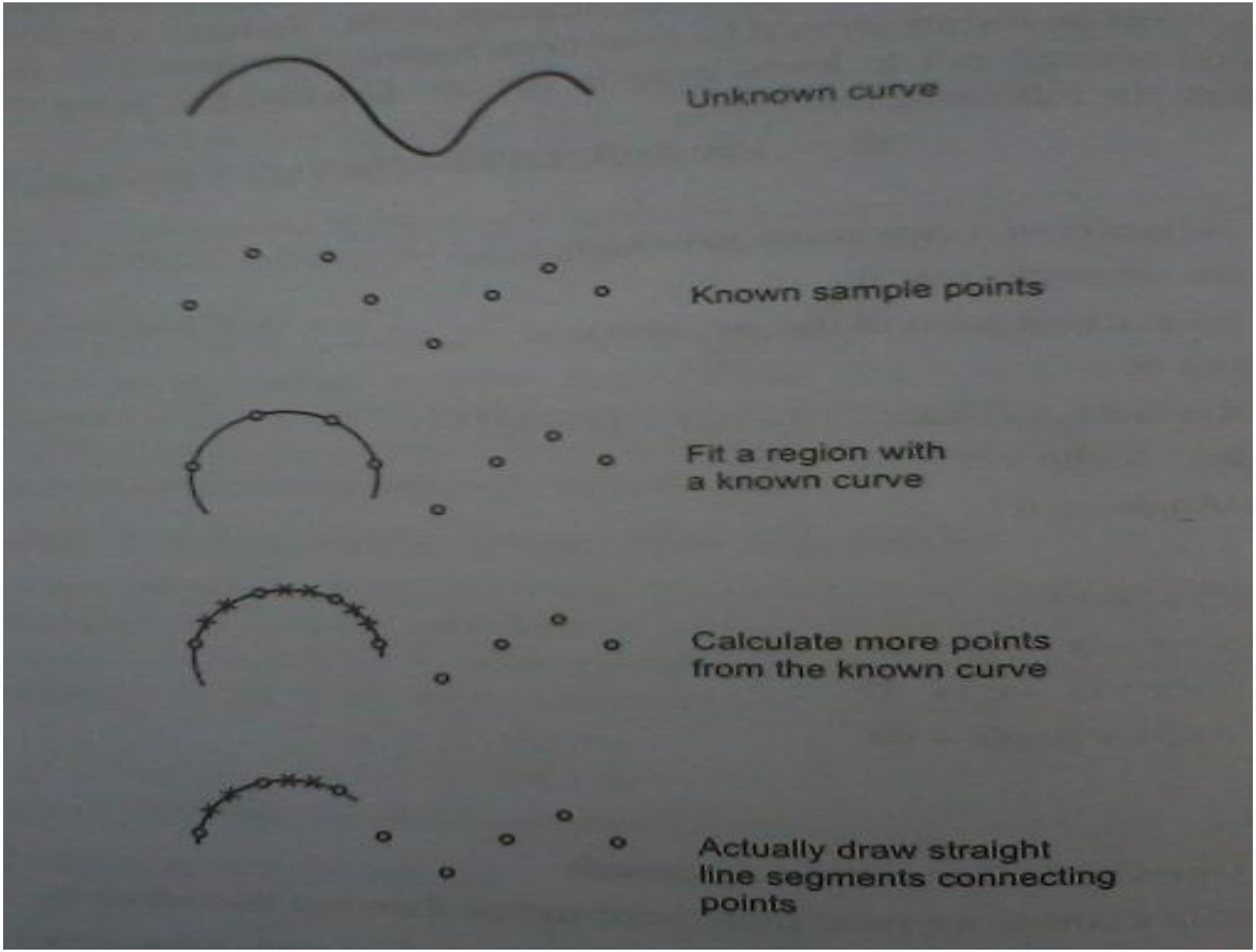
- A curve may be represented as a collection of points.
- Provided the points are properly spaced, connection of the points by short straight line segments yields an adequate visual representation of the curve.
- Connection of points by short straight line segment yields a poor representation of the curve.

- The representation is especially poor where the radius of curvature is small.
- Increasing the point density in these regions improves the representation

Interpolation

- Complex curves can be generated using approximation methods.
- If we have set of sample points which lies on the required curve, then we can draw the required curve by filling portion of curve with the piece of known curves which pass through nearby sample points.
- The gap between the sample points can be filled by finding the co-ordinates of the points along the known approximating curve and connecting these points with the line segments.

interpolation



- Curves can be described mathematically by nonparametric or parametric equations.
- Nonparametric equations can be explicit or implicit
- For a nonparametric curve, the coordinates *y* and *z* of a point on the curve are expressed as two separate functions of the third coordinate *x* as the independent variable
- This curve representation is known as the **nonparametric explicit form.**

Nonparametric and parametric curves

Nonparametric curves:-

mathematically either parametric or a nonparametric

Form is used to represent a curve.

nonparametric representation is either explicit or implicit

for a plane curve an explicit form is given by

$$y=f(x)$$

e.g. equation of a straight line y= mx+b

- In this form, for each x value only one y value is obtained.
- Closed or multivalue curves for eg. Circle cannot be represented explicitly.

- The general Implicit equation is written as

  $ax^2+2bxy+cy^2+2dx+2ey+f=0$
- It provides variety of two dimensional curve forms called conic sections.
- Three types of conic sections are the parabola, the hyperbola and the ellipse.
- The circle is a special case of ellipse.
- by defining the constant coefficients a,b,c,d,e and f, several different types of conic sections are produced.
- If c=1.0 in the general equation, then to define a curve segment between two points, five independent conditions must be specified to determine the values of a,b,d,e and f
- One choice is to specify the slope of curve at both the end points and an intermediate point through which the curve must pass.

- If instead b=0 and c=1.0, then curve is described by only four additional conditions.
- E.g., the two end points and two end slopes.
- An even simpler curve is defined by first setting a=1.0, b=0 and c=1.0 then the form of the curve is

  $x^2+y^2+2dx+2ey+f=0$
- The three conditions required to fix d, e and f are, e.g., the two end points and the slope at the beginning or at the end of the curve segment.
- The alternate choice is to specify two end points and a third internal point through which the curve must pass.
- A straight line is obtained by setting a=b=c=0
- The equation is the

  dx+ey+f=0

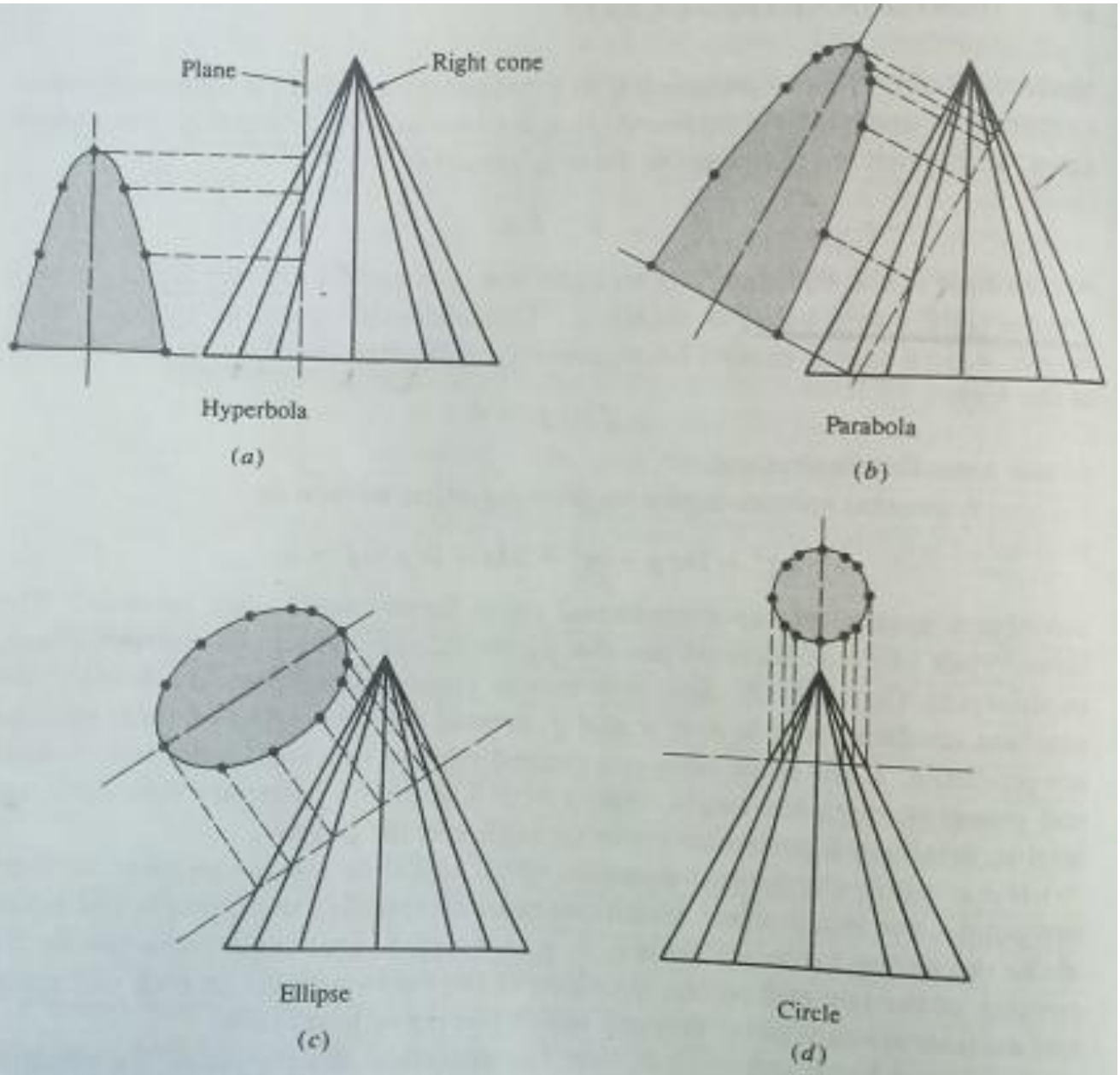


Figure 4–2 Conic sections.

Parametric curves:-

- here each coordinate of a point on a curve is represented as function of a single parameter.
- For two dimensional curve with t as the parameter, the Cartesian coordinate of a point on curve are

  x=x(t)

  y=y(t) t is a parameter 0<=t<=1

  So position vectors of a point on curve is

  P(t)=[x(t) y(t)]

- The nonparametric form is obtained from the parametric form by eliminating the parameter to obtain a single equation in term of x and y.
- The parametric form is suitable for representing closed and multiple valued curves.
- The tangent vector on a parametric curve is given by

$$\mathbf{P'(t)=[x'(t)\ \ y'(t)]}$$

- where ‘ denotes differentiation with respect to the parameter.
- The slope of the curve , dy/dx is

- $dy/dx = (dy/dt)/(dx/dt) = y'(t)/x'(t)$ ----- slope

- Since a point on a parametric curve is specified by a single value of the parameter, the parametric form is axis independent.
- The curve end points and length are fixed by the parameter range.

- The simplest parametric ‘curve’ representation is for a straight line. For two position vectors p1 and p2, a parametric representation of the straight line segment between them is

$$P(t)=P1 + (P2 – P1)t \qquad 0<=t<=1$$

- Since P(t) is a position vector, each of the component of P(t) has a parametric representation x(t) and y(t) between P1 and P2. ie.,

$$x(t)=x1 + (x2 – x1)t \qquad 0<=t<=1$$

$$y(t)=y1 + (y2 – y1)t \qquad 0<=t<=1$$

- Example from book page 212.
- For the position vectors P1[1 2] and P2[4 3] determine the parametric representation of the line segment between them. Also determine the slope and tangent vector of the line segment.

A comparison of non-parametric and parametric representations for a circle in the first quadrant

- The nonparametric representation of the unit circle in the first quadrant is given by

  $y = +\sqrt{1-x^2} \quad 0 <= x <= 1$ shown in figure a.

- Equal increments in x were used to obtain the points on the arc.
- Notice that the resulting arc lengths along the curve are unequal.
- A poor visual representation of the circle results.
- The standard parametric form for a unit circle is

$x = \cos\Theta \qquad 0 <= \Theta <= 2\prod$

$y = \sin\Theta$

Or

$P(\Theta) = [x\ y] = [\cos\Theta \quad \sin\Theta] \qquad 0 <= \Theta <= 2\prod$

- where the parameter Θ is associated with the angle measured counterclockwise from the positive x axis.
- Since for this parametric representation, equal parameter increments produce equal arc lengths along the circumference of the circle, the appearance is quite good. However computation of the trigonometric functions is expensive.

- There is no unique parametric representation for a curve.
- For example,

$$P(t)= \begin{matrix} (1-t^2) & 2t \\ (1+t^2) & (1+t^2) \end{matrix} \qquad 0<=t<=1$$

Also represents the unit arc in the first quadrant as shown in figure c.

The correlation between the parametric representation and the standard parametric representation is given by

$$x= \cos\Theta = \begin{matrix} (1-t^2) \\ (1+t^2) \end{matrix} \qquad 0<= \Theta <= \prod/2 \qquad 0<=t<=1$$

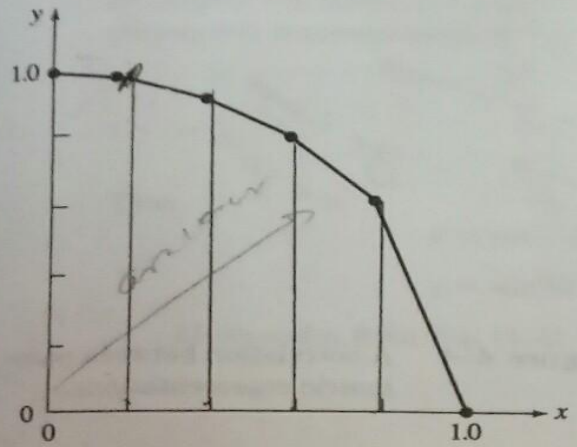
$y = \sin\Theta = \frac{2t}{(1+t^2)}$ $\quad 0 <= \Theta <= \prod/2 \quad$ $0<=t<=1$

- So

  $r^2 = x^2 + y^2 =$ ---------- $= 1$

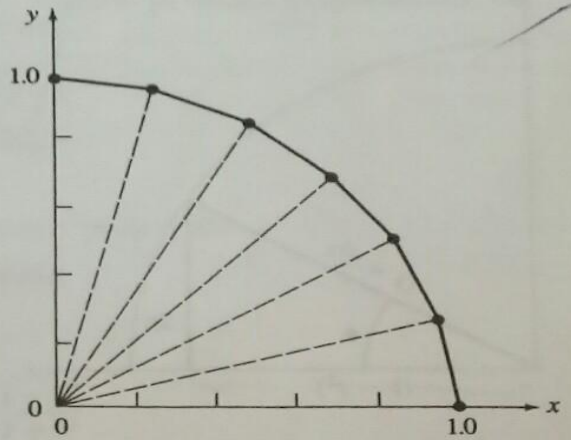
Where r is the unit radius.

Refer example from book……. Page 214



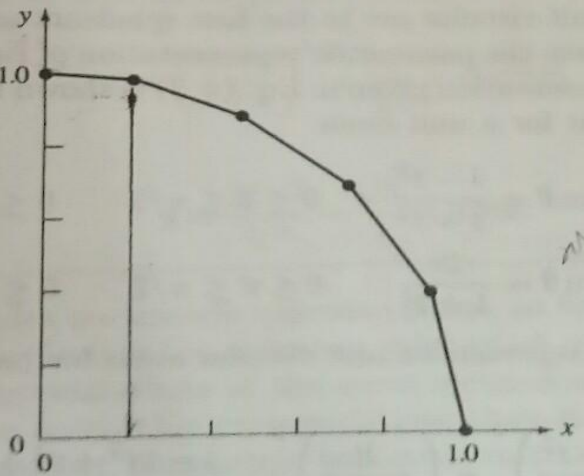
$$y = +\sqrt{1 - x^2} \quad 0 \le x \le 1$$

(a)



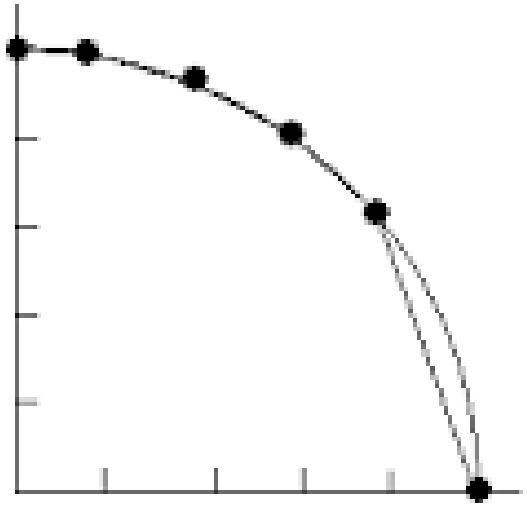
$$x = \cos\theta \quad y = \sin\theta \quad 0 \le \theta \le \pi/2$$

(b)

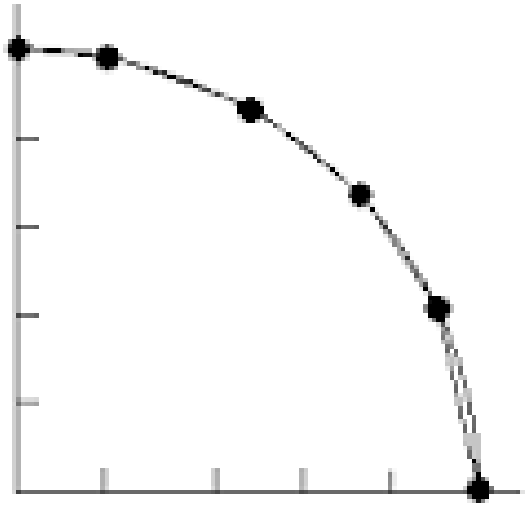


$$x = \frac{1 - t^2}{1 + t^2} \quad y = \frac{2t}{1 + t^2} \quad 0 \le t \le 1$$

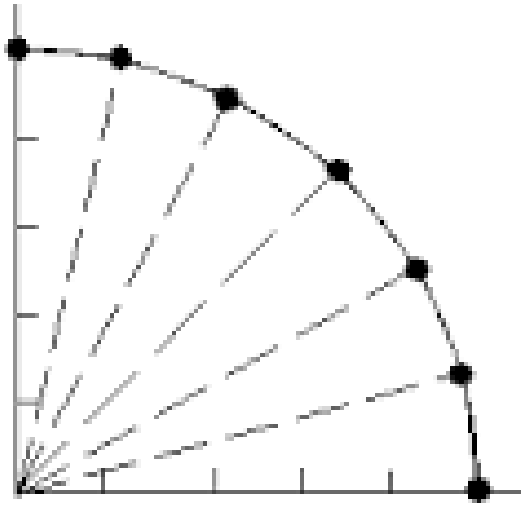
(c)



Explicit form



Discussed form



Standard parametric form

Bezier curves

- Various techniques used for curve generation constrained the curves to pass through existing data points are curve fitting techniques.
- Bezier curve is another approach for the construction of the curve.
- A Bezier curve is determined by a defining polygon.
- Bezier curve have a no. of properties that makes them highly useful and convenient for curve and surface design.

Bezier Curves

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

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

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

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

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

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

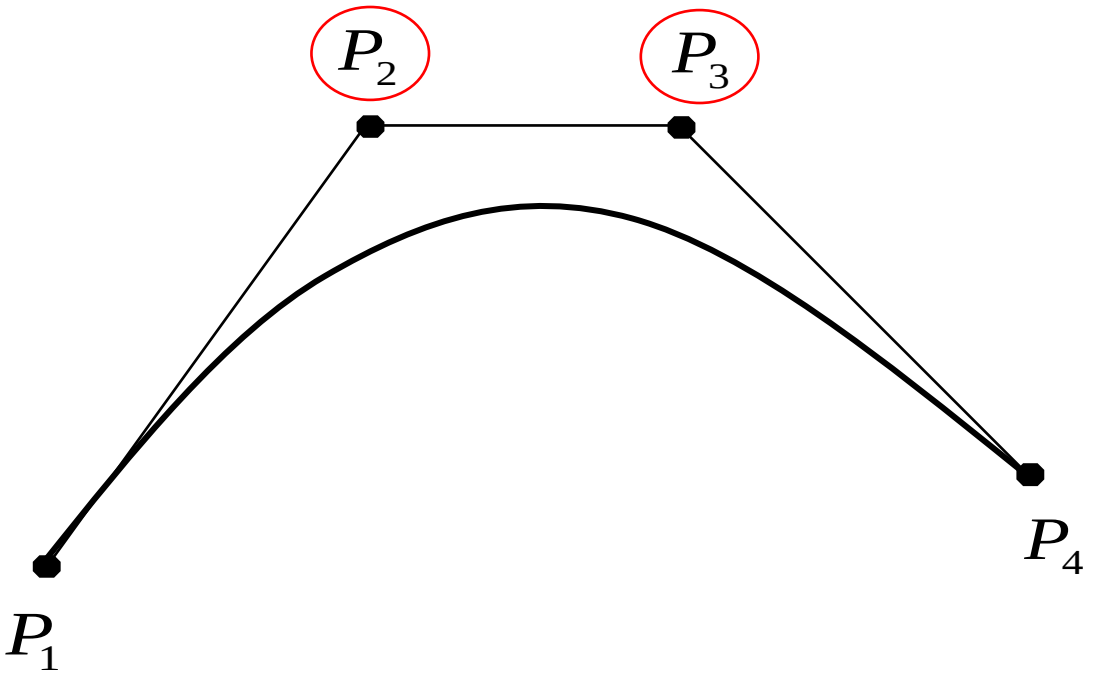
Properties of Bezier Curves

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.

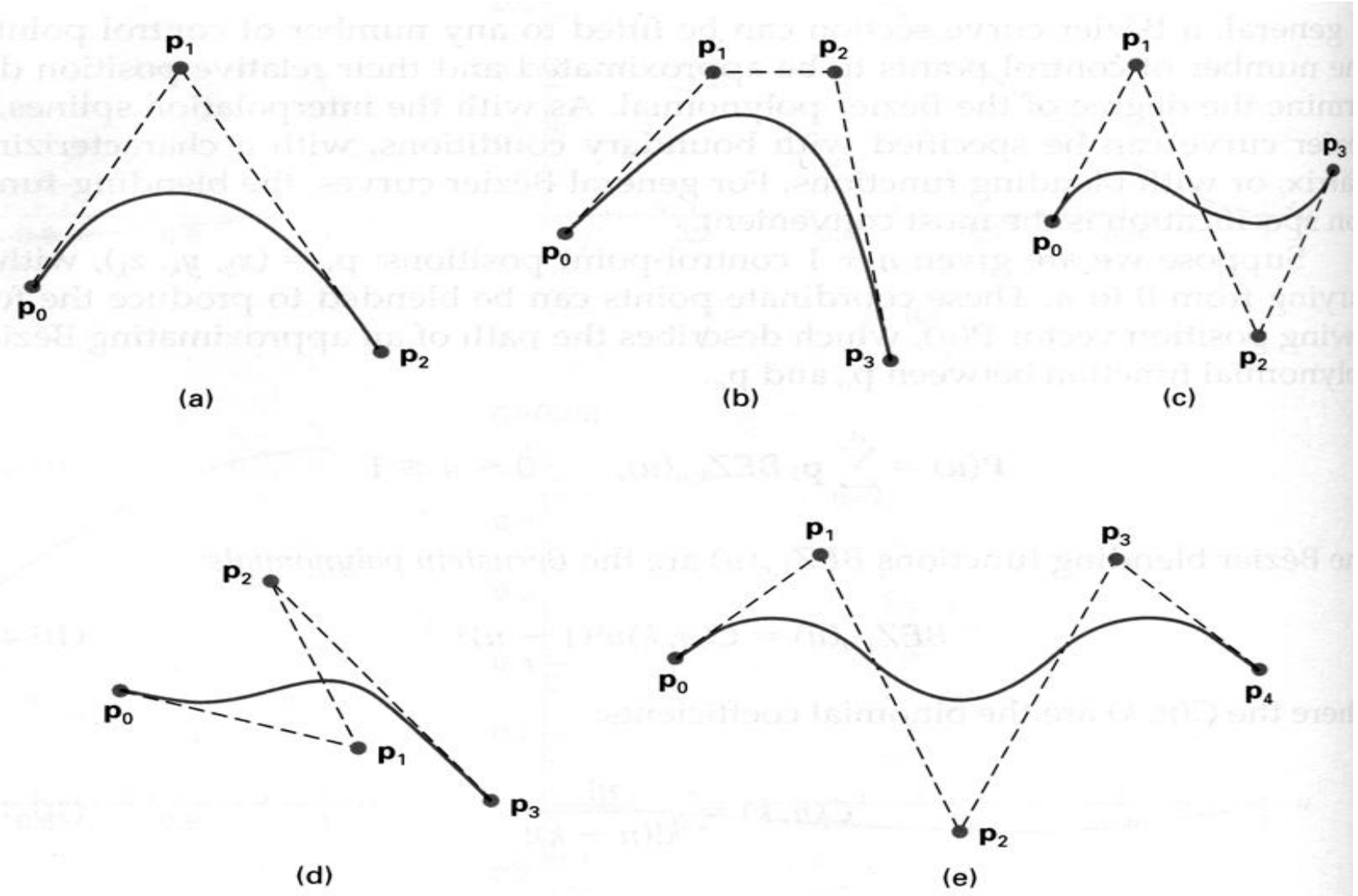
Properties of Bezier curve

- The basic functions are real
- The degree of the polynomial defining the curve segment is one less than the number of defining polygon points
- Bezier curve always passes through the first and last control points.
- The curve generally follows the shape of the defining polygon
- The curve lies entirely within the convex hull formed by four control points



- Simple Bezier curve with a polygon having 4 points.

- Several four point bezier polygon and resulting cubic curves are shown below.



- Mathematically a Bezier curve is defined by
-

$$P(t) = \sum_{i=0}^{n} B_i \, J_{n,i}(t) \qquad 0 \le t \le 1$$

Where the Bezier blending function is

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

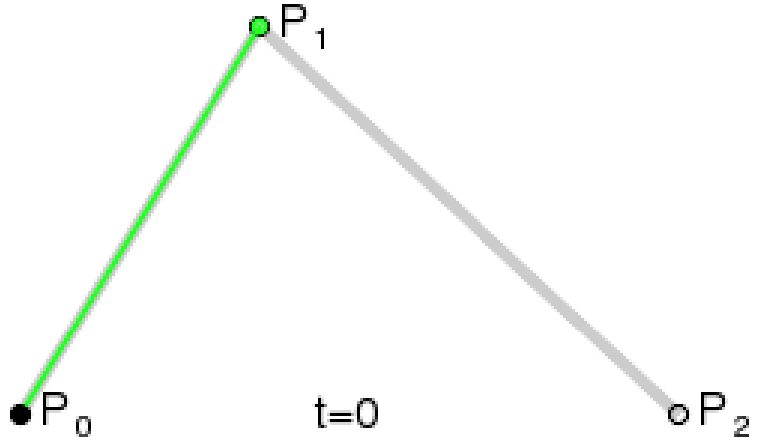
With

$$(ni) = \frac{n!}{i!(n-i)!}$$

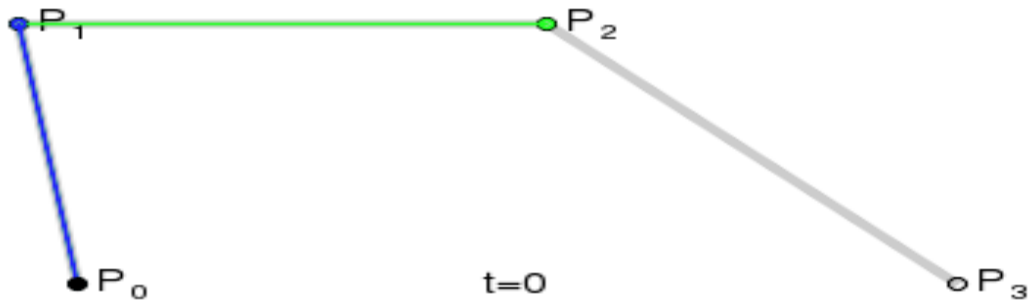
Jn,i(t) is the i^{th} nth-order Bernstein basis function.

n is the degree of the defining Bernstein function.

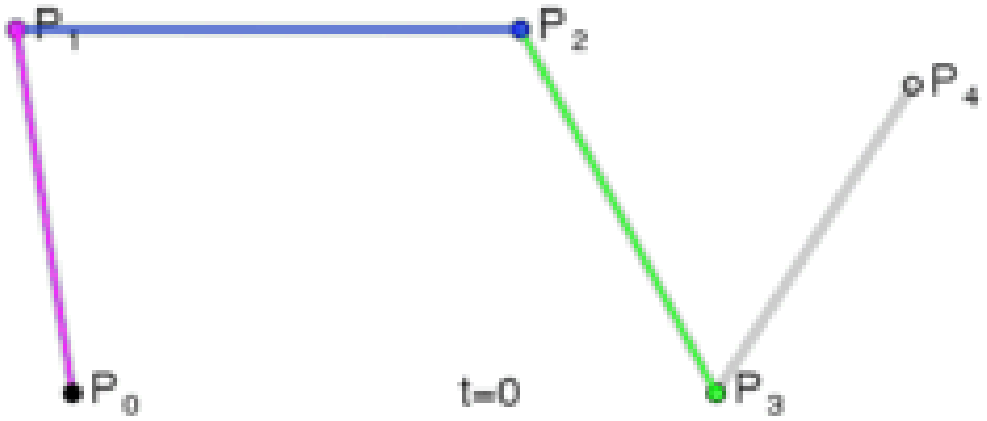
Constructing Bézier curves



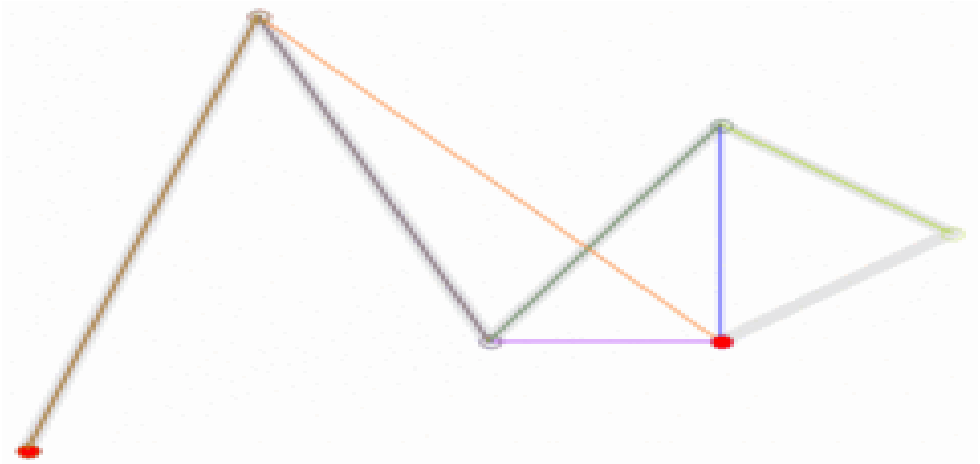
- Linear curves
- The *t* in the function for a linear Bézier curve can be thought of as describing how far **B**(*t*) is from $\mathbf{P}_0$ to $\mathbf{P}_1$. For example, when *t=0.25*, **B**(*t*) is one quarter of the way from point $\mathbf{P}_0$ to $\mathbf{P}_1$. As *t* varies from 0 to 1, **B**(*t*) describes a straight line from $\mathbf{P}_0$ to$\mathbf{P}_1$.



- Quadratic curves
- For quadratic Bézier curves one can construct intermediate points $\mathbf{Q}_0$ and $\mathbf{Q}_1$ such that as t varies from 0 to 1:
- Point $\mathbf{Q}_0(t)$ varies from $\mathbf{P}_0$ to $\mathbf{P}_1$ and describes a linear Bézier curve.
- Point $\mathbf{Q}_1(t)$ varies from $\mathbf{P}_1$ to $\mathbf{P}_2$ and describes a linear Bézier curve.
- Point $\mathbf{B}(t)$ is interpolated linearly between $\mathbf{Q}_0(t)$ to $\mathbf{Q}_1(t)$ and describes a quadratic Bézier curve.



- Higher-order curves
- For higher-order curves one needs correspondingly more intermediate points. For cubic curves one can construct intermediate points $\mathbf{Q}_0$, $\mathbf{Q}_1$, and $\mathbf{Q}_2$ that describe linear Bézier curves, and points $\mathbf{R}_0$ & $\mathbf{R}_1$ that describe quadratic Bézier curves



- Higher-order curves
- or fourth-order curves one can construct intermediate points $\mathbf{Q}_0$, $\mathbf{Q}_1$, $\mathbf{Q}_2$ & $\mathbf{Q}_3$ that describe linear Bézier curves,
  points $\mathbf{R}_0$,$\mathbf{R}_1$ & $\mathbf{R}_2$ that describe quadratic Bézier curves, and points $\mathbf{S}_0$ & $\mathbf{S}_1$ that describe cubic Bézier curves:

- Higher-order curves
- For fifth-order curves, one can construct similar intermediate points

Example:-

- Given Bo[1 1], B1[2 3], B2[4 3] and B3[3 1] the vertices of Bezier polygon, determine seven points on the Bezier curve.

Spline curves

Spline is a flexible strip which was long ago used for designing the ships.

Spline curves:-

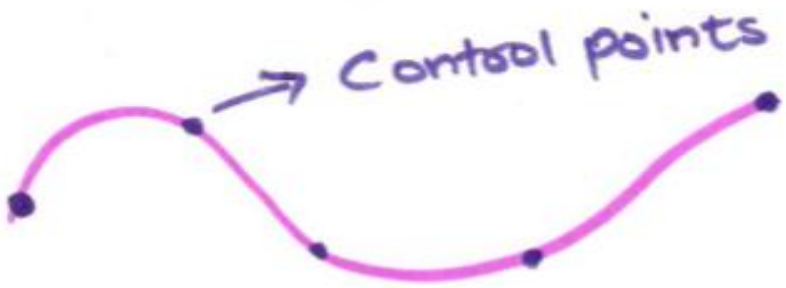
A spline curve is a mathematical representation for which it is easy to build an interface that will allow a user to design and control the shape of complex curves and surfaces.

Spline curves mathematically described with a piecewise cubic polynomial function whose first and second derivatives are continuous across the various curve sections.

- Control points:-
- We specify a spline curve by giving a set of control points which indicates the general shape of the curve. These control points then fitted with piecewise continuous parametric polynomial functions in on two ways.

Interpolate or Interpolation Spline :-

When polynomial sections are fitted so that the curve passes through all control points, then the resulting curve is said to be Interpolate the set of control points.



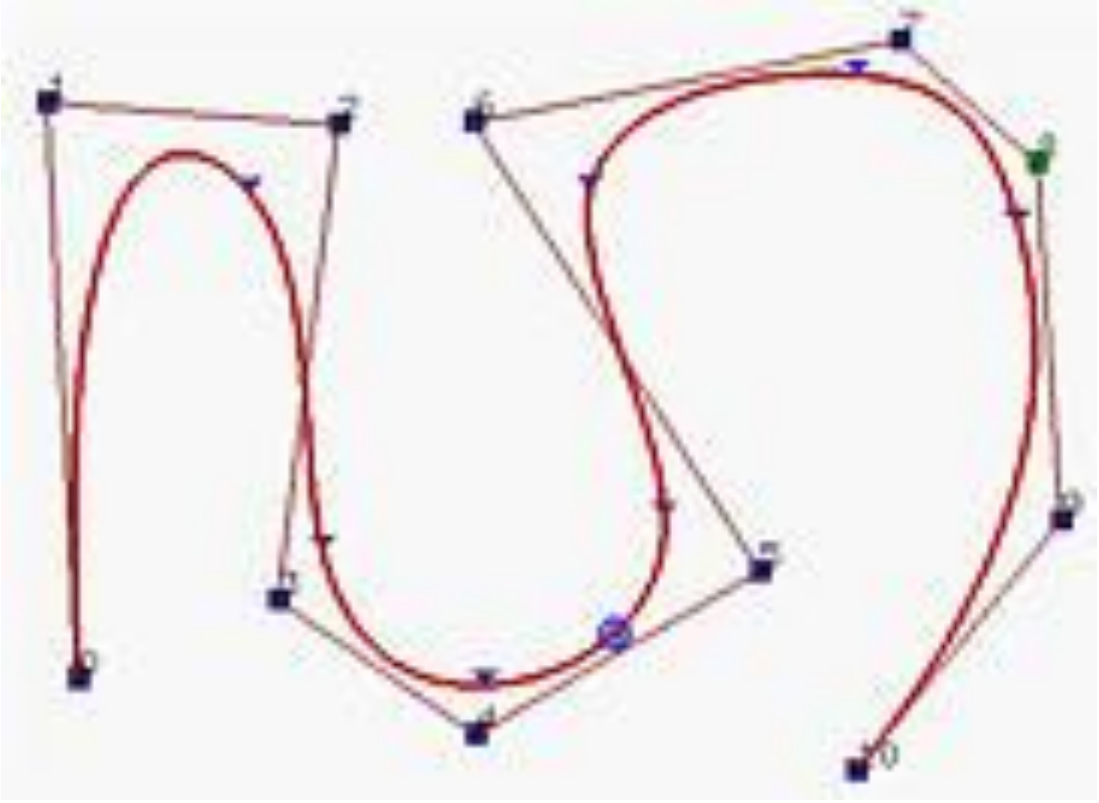
Approximate or Approximation Spline :-

When the polynomials are fitted to the path which is not necessarily passing through all control points, the resulting curve is said to approximate the set of control points.

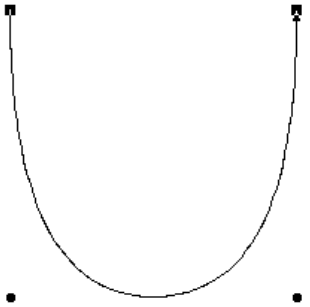
Approximation Spline

B-Spline curves

- A long flexible strips of metal used by draftspersons to lay out the surfaces of airplanes, cars and ships
- Ducks weights attached to the splines were used to pull the spline in different directions
- The metal splines had second order continuity.
- B-Splines
  - Another polynomial curve for modelling curves and surfaces
  - Consists of curve segments whose polynomial coefficients only depend on just a few control points
    - Local control
  - Segments joined at *knots*

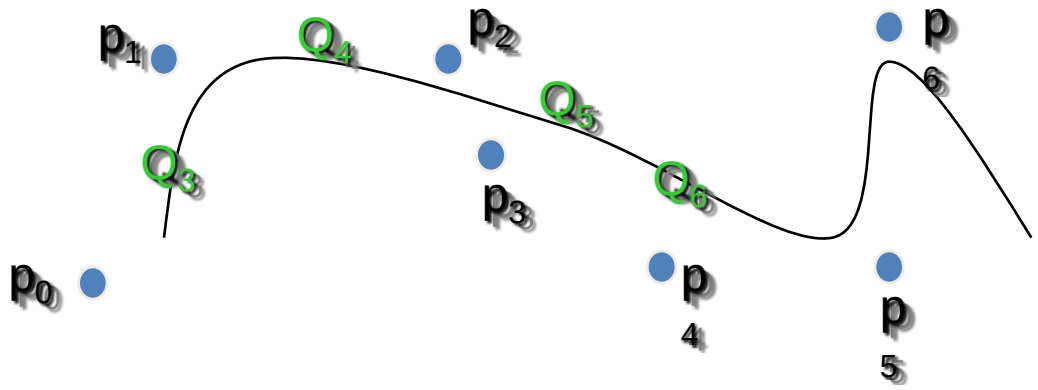


- B-splines
- The curve does not necessarily pass through the control points
- The shape is constrained to the convex hull made by the control points
- Uniform cubic b-splines has C_2 continuity
  - Higher than Bezier curves



B-Spline Curve

- Start with a sequence of control points
- Select four from middle of sequence (p_{i-2}, p_{i-1}, p_i, p_{i+1}) d
  - Bezier and Hermite goes between p_{i-2} and p_{i+1}
  - B-Spline doesn't interpolate (touch) any of them but approximates the going through p_{i-1} and p_i

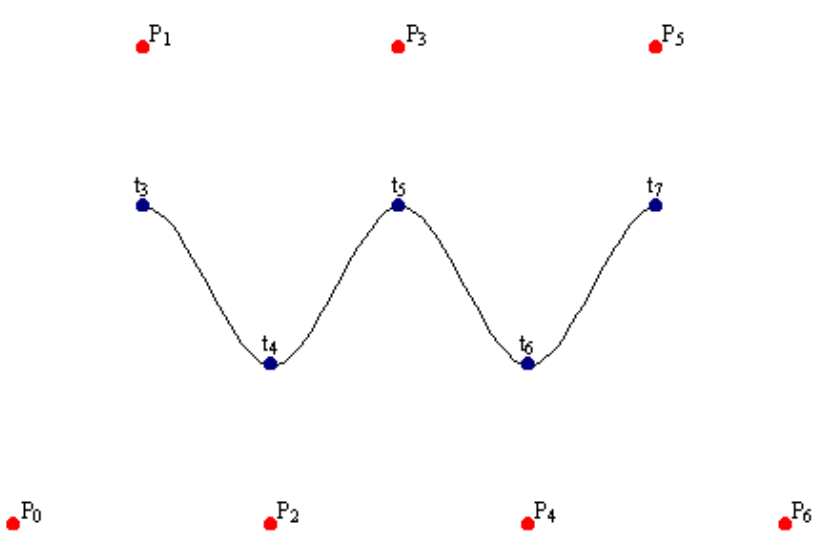


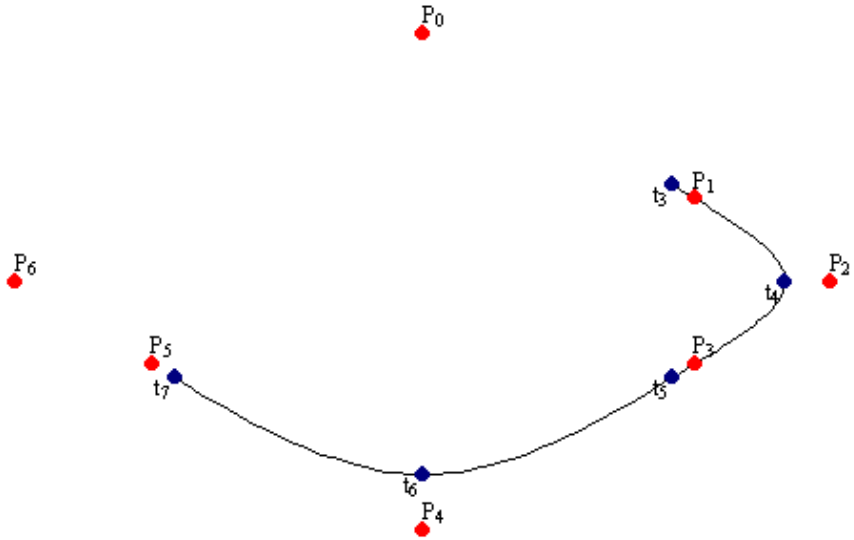
Uniform B-Splines

- First curve segment, Q_3, is defined by first four control points
- Last curve segment, Q_m, is defined by last four control points, P_{m-3}, P_{m-2}, P_{m-1}, P_m
- Each control point affects four curve segments

B-Spline

- By far the most popular spline used C_0, C_1, and C_2 continuous





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.

Hidden surface removal

- In a given set of 3D objects and viewing specification , we wish to determine which line or surface of the objects are visible, so that we can display only the visible line or surface.

- The hidden line or hidden surface algorithm determines the lines, edges or surfaces that are visible or invisible to an observer located at a specific point in a space.

- Two classes of the algorithm
  - Object-space method
  - Image-space method

- Object-space Method:-

  It is implemented in the physical coordinate system in Which objects are described.

  It compares objects and parts of objects to each other within the scene definition to determine which surface we should label as visible.

- Image-space Method:-

  It is implemented in the screen coordinate system in which the objects are viewed. Here the visibility is decided point by point at each pixel position.

Z-Buffer algorithm

- The z-buffer is one of the simplest of the visible surface algorithms.
- Z-buffer is a simple extension of frame buffer idea
- A frame buffer is used to store the intensity of each pixel in image space.
- Z-buffer is a separate depth buffer used to store the z coordinate of every visible pixel in image space.
- Z value of a new pixel to be written in the frame buffer is compared to that pixel stored in the z buffer.

- If comparison indicates that the new pixel is in front of the pixel stored in the frame buffer, then the new pixel is written to the frame buffer and the z- buffer update with the new z value.
- If not, no action is taken
- More precisely this algorithm Is a search over x,y for the largest value of z(x,y).

- Advantages of z buffer
- Simplicity of algorithm
- It handles the visible surface problem & display of complex surface intersections

- Disadvantages of z buffer
- The storage requirement
- If the scene is transformed & clipped to a fixed range of z- coordinates, then a z-buffer of fixed precision can be used.
- Difficulty & expense of implementing ant aliasing, transparency effects.

- Set the frame buffer to the background intensity or color set the z-buffer to the minimum z- value.
- Scan convert each polygon in arbitrary order.
- For each pixel(x,y) in the polygon, calculate the depth z(x,y) at that pixel.
- Compare the depth z(x,y) with the value stored in the z buffer at that location
- If z(x,y) > zbuffer(x,y), then write the polygon attributes (intensity, color) to the frame buffer and replace zbuffer (x,y) with z(x,y)
- Otherwise, no action is taken

- for each polygon P

  for each pixel (x, y) in P

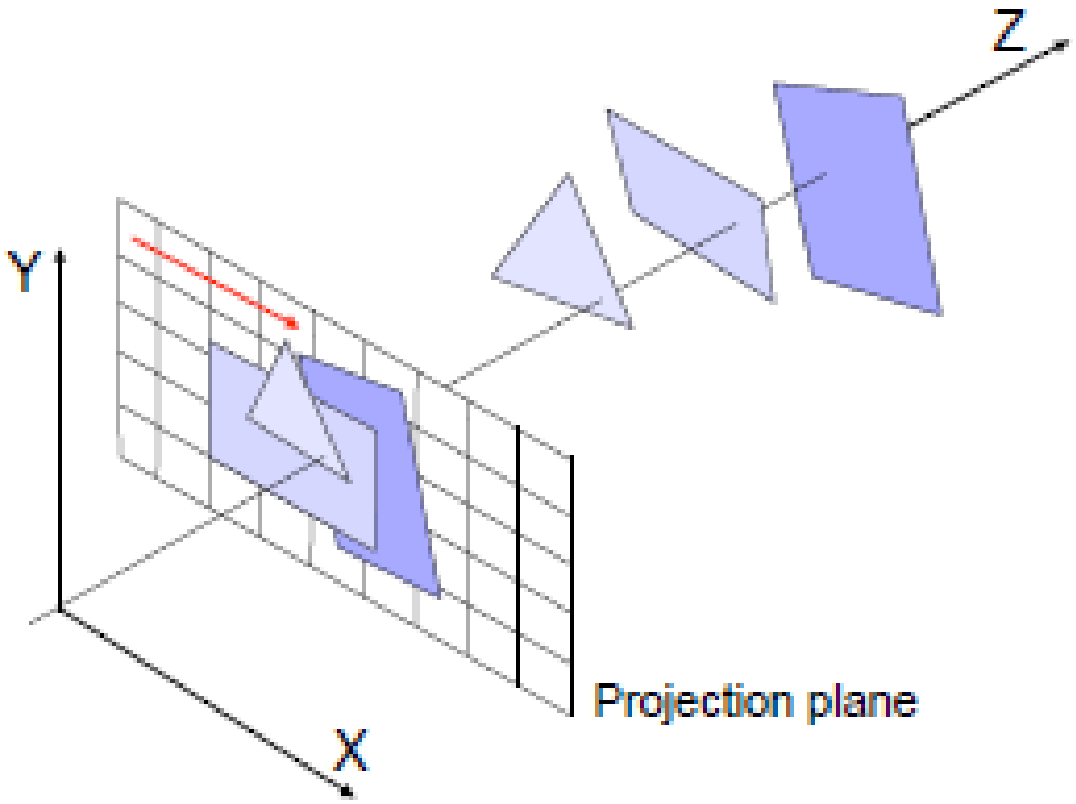
  compute z_depth at x, y

  if z_depth < z_buffer (x, y) then

  set_pixel (x, y, color)

  z_buffer (x, y) = z_depth

Z-buffer algorithm



Warnock's Area subdivision Algorithm

- The area-subdivision method takes advantage by locating those view areas that represent part of a single surface. Divide the total viewing area into smaller and smaller rectangles until each small area is the projection of part of a single visible surface or no surface at all.
- Continue this process until the subdivisions are easily analyzed as belonging to a single surface or until they are reduced to the size of a single pixel.

- An easy way to do this is to successively divide the area into four equal parts at each step. There are four possible relationships that a surface can have with a specified area boundary.

- **1. Surrounding Polygon** - One that completely encloses the (shaded)area of interest (see Fig.h (a))
- **2. Overlapping or Intersecting Polygon** - One that is partly inside and partly outside the area (see Fig. h (b))
- **3. Inside or Contained Polygon** - One that is completely inside the area (see Fig. h (c)).
- **4. Outside or Disjoint Polygon** - One that is completely outside the area (see Fig. h (d)).

Fig. (h) Possible relationship with polygon surfaces and the area of interest.

- After checking four relationships we can handle each relationship as follows:

- If all the polygons are disjoint from the area, then the background color is displayed in the area.
- If there is only one intersecting or only one contained polygon, then the area is first filled with the background color, and then the part of the polygon contained in the area is filled with color of polygon.
- If there is a single surrounding Polygon, but no intersecting or contained polygons, then the area is filled with the color of the surrounding polygon.
- If there are more than one polygon intersecting, contained in. or surrounding the area then we have to do some more processing.

Warnock's Area Subdivision

(Image Precision)

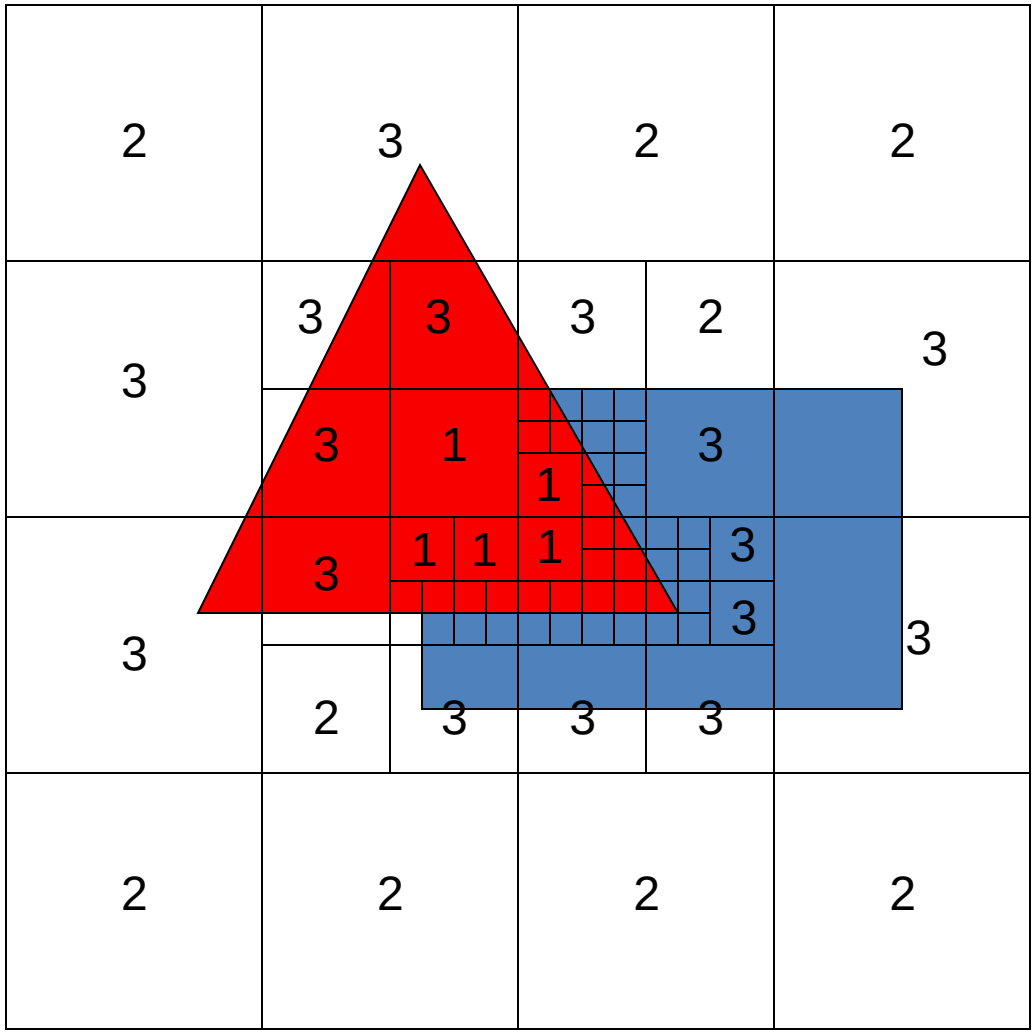
- Start with whole image
- If one of the easy cases is satisfied (previous slide), draw what's in front
- Otherwise, subdivide the region and recurse
- If region is single pixel, choose surface with smallest depth
- Advantages:
  - No over-rendering
  - Anti-aliases well - just recurse deeper to get sub-pixel information
- Disadvantage:
  - Tests are quite complex and slow

- Steps:--.
- 1. Initialize the region.
- 2. Generate list of polygons by sorting them with their z values.
- 3. Remove polygons which are outside the area.
- 4. Identify relationship of each polygon.
- 5. Execute visibility decision analysis:

- a)Fill area with background color if all polygons are disjoint,
- b)Fill entire area with background color and fill part of polygon contained in area with color of polygon if there is only one contained polygon,
- c) If there is a single surrounding polygon but not contained then fill area with color of surrounding polygon.
- d)Set pixel to the color of polygon which is closer to view if region of the pixel (x,y) and if neither of (a) to (d) applies calculate z- coordinate at pixel (x,y) of polygons.

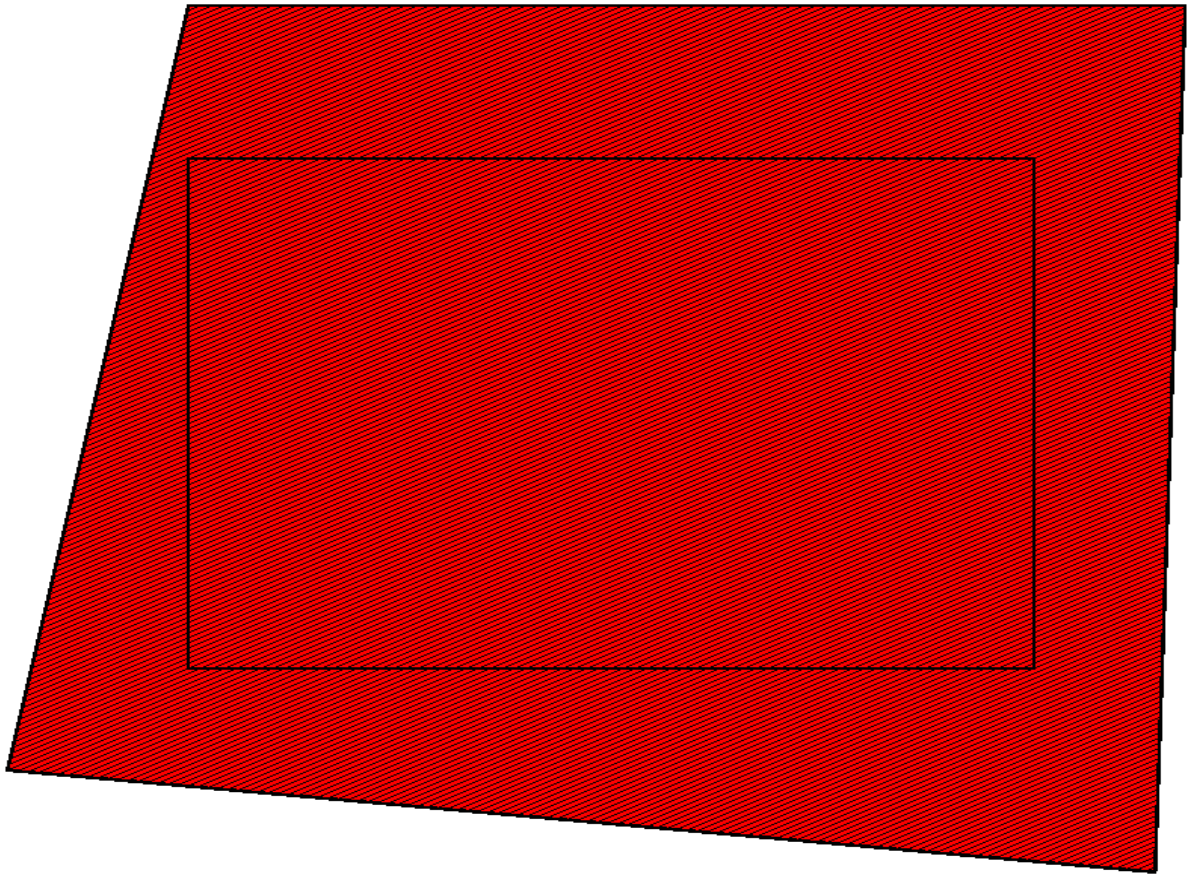
- 6. If none of above is correct then subdivide the area and Go to Step 2.

Warnock's Algorithm

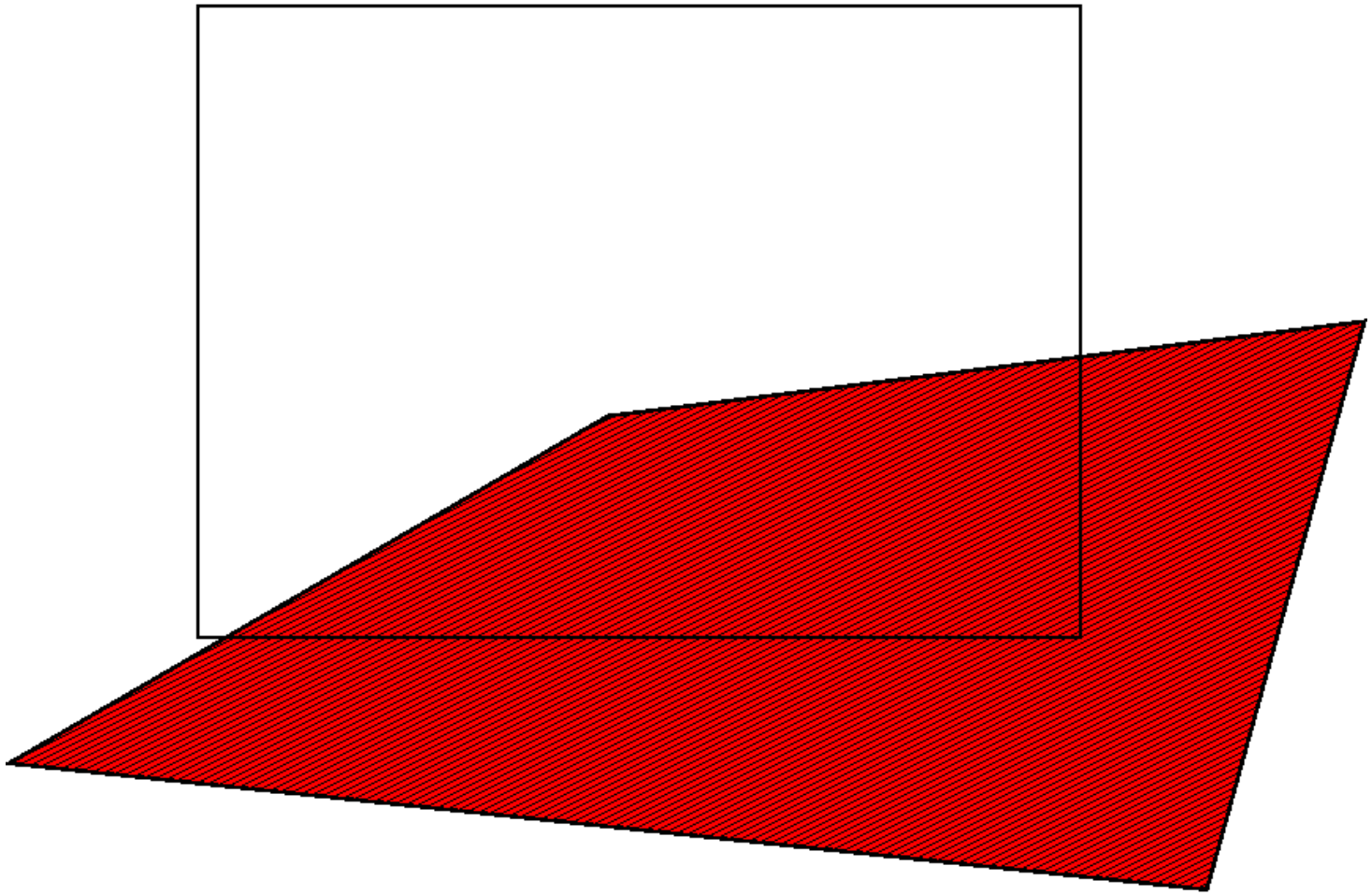


- Regions labeled with case used to classify them:
  1) One polygon in front
  2) Empty
  3) One polygon inside, surrounding or intersecting
- Small regions not labeled
- Note it's a rendering algorithm and a HSR algorithm at the same time
  - Assuming you can draw squares

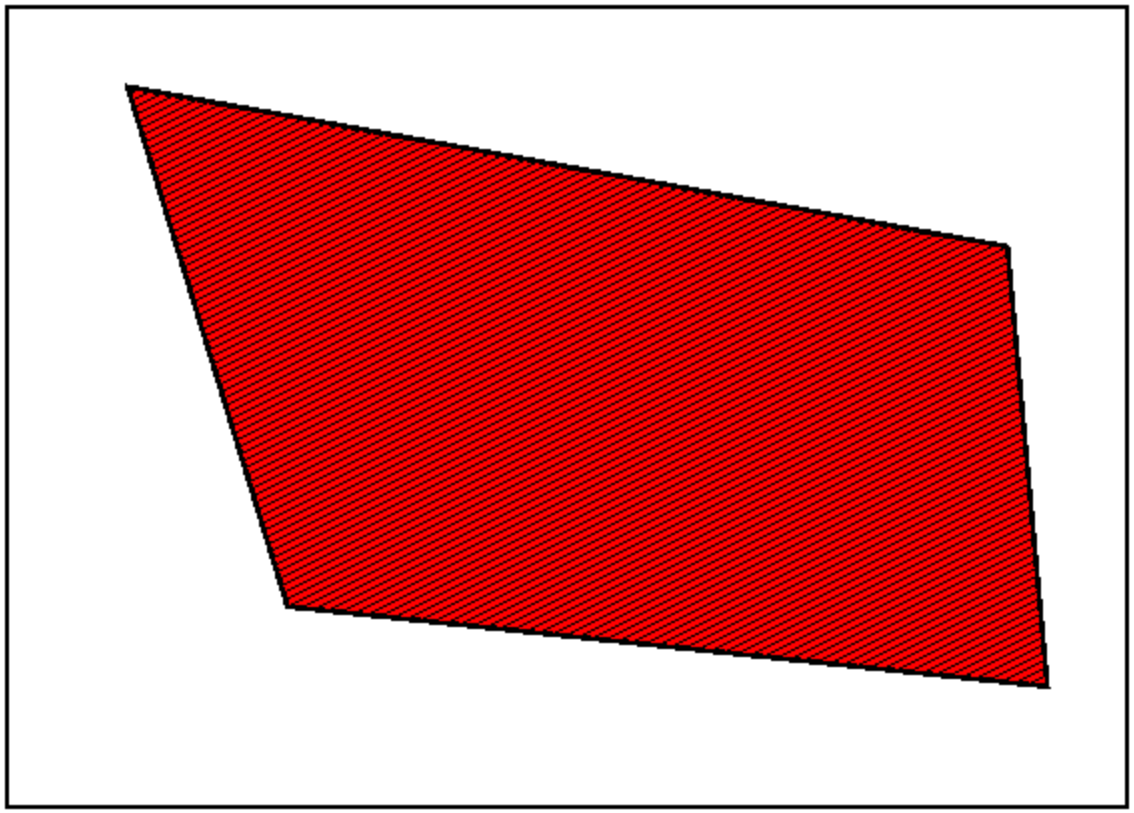
surrounding



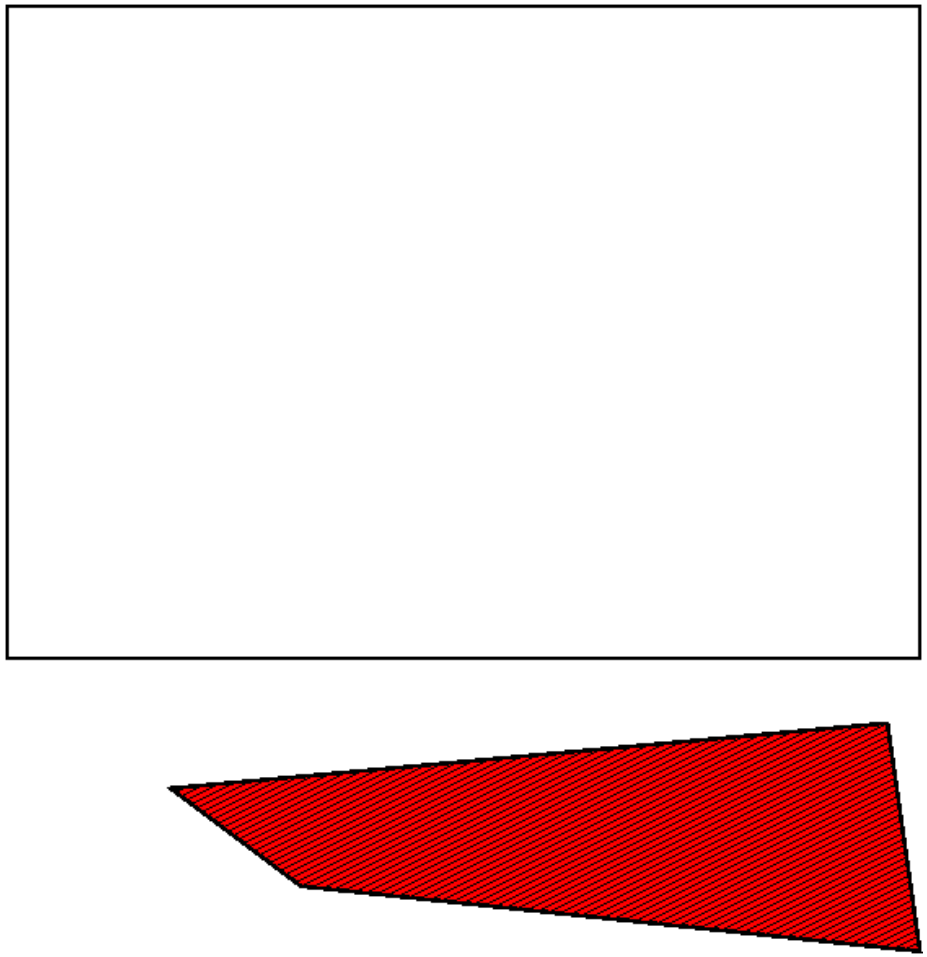
intersecting



inside



outside



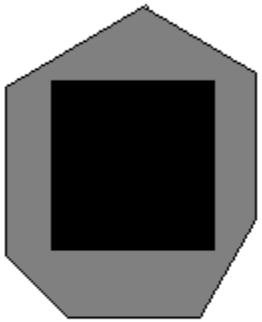
- The classification scheme is used to identify certain trivial cases that are easily handled. These "easy choice tests" and the resulting actions include:
- For polygons outside from the window, set the color/intensity of the window equal to the background color.
- There is only one inside or intersecting polygon. Fill the window area with the background color then render the polygon.
- There is only one surrounding polygon. Fill the window with the polygon's color.
- If more than one polygon intersects, is inside, or surrounds, and at least one is a surrounding polygon.

- Warnock developed area subdivision algorithm which subdivides each area into four equal squares.

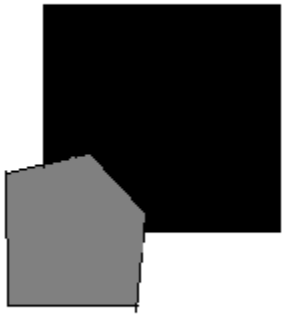
- At each stage in the subdivision process, the relationship between projection of each polygon and the area of interest is checked for four possible relationships

- Surrounding polygon
- Overlapping polygon
- Contained polygon
- Disjoint polygon

- Each of the relationship can be handled as
  - If all polygons are disjoint from the area, the background color is displayed in the area.
  - If there is only one intersecting or only one contained polygon, the area is first filled with the background color, and then the part of the polygon contained in area is filled with the color of polygon
  - If there is a single surrounding polygon, but no intersecting or contained polygon, then the area is filled with the color of surrounding polygon.

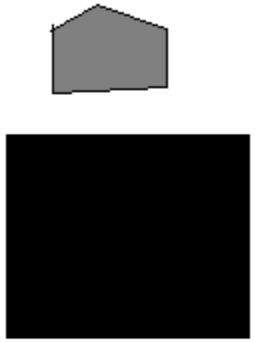


surrounding



overlapping

contained



disjoint