UNIT – 2

Scan Conversion Techniques and Image Representation

Unit-02/Lecture-01

Scan Conversion **[RGPV/DEC-2008(10)]**

Scan conversion or scan rate converting is a technique for changing the vertical / horizontal scan frequency of video signal for different purposes and applications. The device which performs this conversion is called scan converter. The application of scan conversion is wide and covers video projectors, cinema equipment, TV and video capture cards, standard and HDTV televisions, LCD monitors and many different aspects of picture and video processing.

Scan conversion process, not only needs to make changes in synchronization (sync) frequencies, changes in picture information data rate are also mandatory in most cases. There are two distinct methods for doing the process:

- **Analog Methods** (Non retentive, memory-less or real time method)
  This conversion is done using large numbers of delay cells and is appropriate for analog video.
- **Digital methods** (Retentive or buffered method)
  In this methods , picture is stored in a line or frame buffer with n1 speed (data rate) and is read with n2 speed, several picture processing techniques are applicable when the picture is stored in buffer memory including kinds of interpolation from simple to smart high order comparisons, motion detection and to improve the picture quality and prevent the conversion artifacts.

A picture is completely specified by the set of intensities for the pixel positions in the display. Shapes and colors of the objects can be described internally with pixel arrays into the frame buffer or with the set of the basic geometric ‐ structure such as straight line segments and polygon color areas. To describe structure of basic object is referred to as output primitives.

OUTPUT PRIMITIVES: POINTS AND LINES

Graphics programming packages provide functions to describe a scene in terms of these basic geometric structures, referred to as Output Primitives, and to group sets of output primitives into more complex structures.

Each output primitive is specified with input coordinate data and other information about the way that objects is to be displayed.

Points and Straight Line segments are the simplest geometric components of pictures. Additional output primitives that can be used to construct a picture include circles and other conic sections, quadric surfaces, spline curves and surfaces, polygon color areas, and character strings.

Points and Lines:

Point plotting is accomplished by converting a single coordinate position furnished by an application program into appropriate operations for the output device.

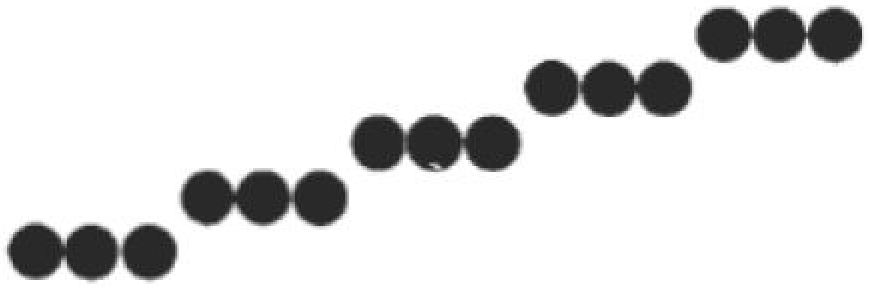
A Random-Scan (Vector) System stores point-plotting instructions in the display list, and coordinate values in these instructions are converted to deflection voltages that position the electron beam at the screen locations to be plotted during each refresh cycle.

For a black-and-white raster system, a point is plotted by setting the bit value corresponding to a specified screen position within the frame buffer to 1. Then, as the electron beam sweeps across each horizontal scan line, it emits a burst of electrons (plots a point) whenever a value of 1 is encountered in the frame buffer.

With an RGB system, the frame buffer is loaded with the color codes for the intensities that are to be displayed at the screen pixel positions. Line drawing is accomplished by calculating intermediate positions along the line path between two specified endpoint positions. An output device is then directed to fill in these positions between the endpoints.

For a raster video display, the line color (intensity) is then loaded into the frame buffer at the corresponding pixel coordinates. Reading from the frame buffer, the video controller then "plots" the screen pixels. Screen locations are referenced with integer values, so plotted positions may only approximate actual Line positions between two specified endpoints.

For example, a computed line position is (10.48, 20.51), it is rounded to (10, 21). This rounding of coordinate values to integers causes lines to be displayed with a stairstep appearance ("the jaggies"), as represented below. This stairstep shape is noticeable in low resolution systems.

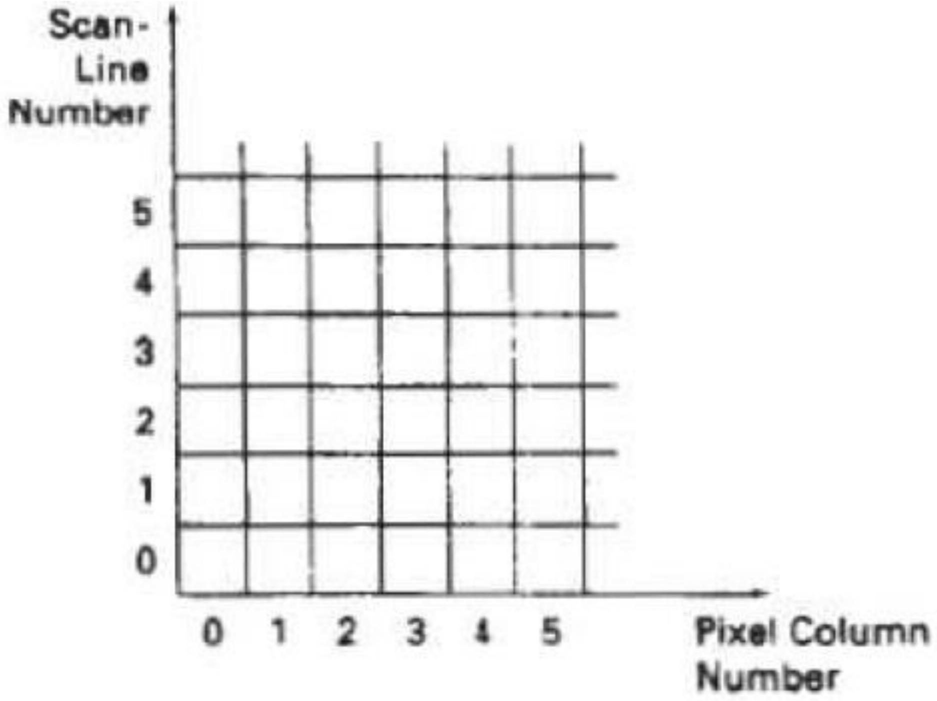


For the raster-graphics device-level algorithms, object positions are specified directly in integer device coordinates.To load a specified color into the frame buffer at a position corresponding to column x along scan line y, we will assume we have available a low-level procedure of the form

setPixel (x, y)

Sometimes we want to retrieve the current frame-buffer intensity setting for a specified location. We accomplish this with the low-level function. We use,

getPixel (x, y)



S.NO	RGPV QUESTIONS	Year	Marks
1.	What is scan conversion?	Dec, 2012	5
2.	What do you mean by scan-conversion techniques? What methods are adopted to remove the side effects of scan conversion?	Dec, 2011	10

Line drawing Algorithm

Line Drawing Algorithms

- Digital Differential Analyzer (DDA) Algorithm
- Bresenham‟s Line Algorithm
- Parallel Line Algorithm

The Cartesian slope-intercept equation for a straight line is

$$y = mx + b \qquad (1)$$

with m representing the slope of the line and b as the y intercept. Given that the two endpoints of a line segment are specified at positions (x1, y1) and (x2, y2).
We can determine the slope m and y intercept b with the following calculations:

$$m = \frac{(y2-y1)}{(x2-x1)} \qquad (2)$$

$$b = (y1 - m * x1) \qquad (3)$$

Algorithms for displaying straight lines are based on the line equations (1) and the calculations given in equations (2) and (3).
For any given x interval Δx along a line, we can compute the corresponding y interval Δy from equation (2) as,

$$y = m\Delta x \qquad (4)$$

Similarly, we can obtain the x interval Δx corresponding to a specified ?y as

$$\Delta x = \frac{\Delta y}{m} \qquad (5)$$

These equations form the basis for determining deflection voltages in analog devices.
For lines with slope magnitudes | m | < 1, Δx can be set proportional to a small horizontal deflection voltage and the corresponding vertical deflection is then set proportional to Δy as calculated from Equation 4.
For lines whose slopes have magnitudes | m | > 1, Δy can be set proportional to a small vertical deflection voltage with the corresponding horizontal deflection voltage set proportional to Δx, calculated from Equation 5.
For lines with m = 1, Δx = Δy and the horizontal and vertical deflections voltages are equal.
In each case, a smooth line with slope m is generated between the specified endpoints.

Digital Differential Analyzer (DDA) Algorithm

The Digital Differential Analyzer (DDA) is a Scan-Conversion line algorithm based calculating either Δy orΔx using equations (4) and (5).
Consider first a line with positive slope, less than or equal to 1, we sample at unit intervals (Δx=1) and compute each successive y value as

$$y_{k+1} = y_k + m \qquad (6)$$

subscript k takes integer values starting from 1, for the first point, and increases by 1 until the final endpoints is reached.
Since m can be any real number between 0 & 1, the calculated y values must be rounded to the nearest integer.
For lines with a positive slope greater than 1, we reserve the roles of x & y. That is, we

sample at unit y intervals (Δy=1) and calculate each succeeding x value as

$$x_{k+1} = x_k + \frac{1}{m} \qquad (7)$$

Equations (6) and (7) are based on the assumption that lines are to be processed from the left endpoint to the right endpoint.

If this processing is reversed, so that the starting endpoint is that, then either we have Δx= -1 and

$$y_{k+1} = y_k - m \qquad (8)$$

or (when the slope is greater than 1) we have y = -1 with

$$x_{k+1} = x_k - \frac{}{m} \qquad (9)$$

Equations (6), (7), (8) and (9) can also be used to calculate pixel positions along a line with negative slope. If the absolute value of the slope is less than 1 and the start endpoint is at the left, we set Δx=1 and calculate y values with equation (6).

When the start endpoint is at the right (for the same slope), we set Δx= -1 and obtain y positions from equation (8). Similarly, when the absolute value of a negative slope is greater than 1, we use Δy= -1 and equation (9) or we use Δy=1 and equation (7).

Algorithm:

```
void lineDDA (int xa, int ya, int xb, int yb)
{

int dx = xb - xa, dy = yb - ya, steps, k;
float xIncrement, yIncrement, x = xa, y = ya;
if (abs (dx) > abs (dy))
        steps = abs (dx) ;
        else

        steps = abs (dy);
        xIncrement = dx / (float) steps;
        yIncrement = dy / (float) steps;
        setpixel (ROUND(x), ROUND(y)) ;
        for (k=O; k<steps; k++)
        {

        x += xIncrment; y += yIncrement;
        setpixel (ROUND(x), ROUND(y));
        }
        }
```

The DDA algorithm is a faster method for calculating pixel position than the direct use of equation

- We can improve the performance of the DDA algorithm by separating the increments m and 1/m into integer and fractional parts so that all calculations are reduced to integer operations.

Algorithm Description:

- Step 1 : Accept Input as two endpoint pixel positions
- Step 2: Horizontal and vertical differences between the endpoint positions are assigned to parameters dx and dy (Calculate dx=xb-xa and dy=yb-ya).
- Step 3: The difference with the greater magnitude determines the value of parameter steps.
- Step 4 : Starting with pixel position (xa, ya), determine the offset needed at each step to generate the next pixel position along the line path.
- Step 5: loop the following process for steps number of times
  1.) Use a unit of increment or decrement in the x and y direction
  2.)if xa is less than xb the values of increment in the x and y directions are 1 and m
  3.)if xa is greater than xb then the decrements -1 and – m are used.

Example : Consider the line from (0,0) to (4,6)

- xa=0, ya =0 and xb=4 yb=6
- dx=xb-xa = 4-0 = 4 and dy=yb-ya=6-0= 6
- x=0 and y=0
- 4 > 6 (false) so, steps=6
- Calculate xIncrement = dx/steps = 4 / 6 = 0.66 and yIncrement = dy/steps =6/6=1
- Setpixel(x,y) = Setpixel(0,0) (Starting Pixel Position)
- Iterate the calculation for xIncrement and yIncrement for steps(6) number of times
- Tabulation of the each iteration

k	**x**	**Y**	**Plotting points (Rounded to Integer)**
0	0+0.66=0.66	0+1=1	(1,1)
1	0.66+0.66=1.32	1+1=2	(1,2)
2	1.32+0.66=1.98	2+1=3	(2,3)
3	1.98+0.66=2.64	3+1=4	(3,4)
4	2.64+0.66=3.3	4+1=5	(3,5)
5	3.3+0.66=3.96	5+1=6	(4,6)

Advantages of DDA Algorithm

- It is the simplest algorithm
- It is a is a faster method for calculating pixel positions

Disadvantages of DDA Algorithm

- Floating point arithmetic in DDA algorithm is still time-consuming
- End point accuracy is poor

S.NO	RGPV QUESTIONS	Year	Marks
1.	Give the drawback of DDA algorithm and discuss Bresenhams Line Generation algorithm in detail.	June, 2012	10

Bresenham Line Algorithm

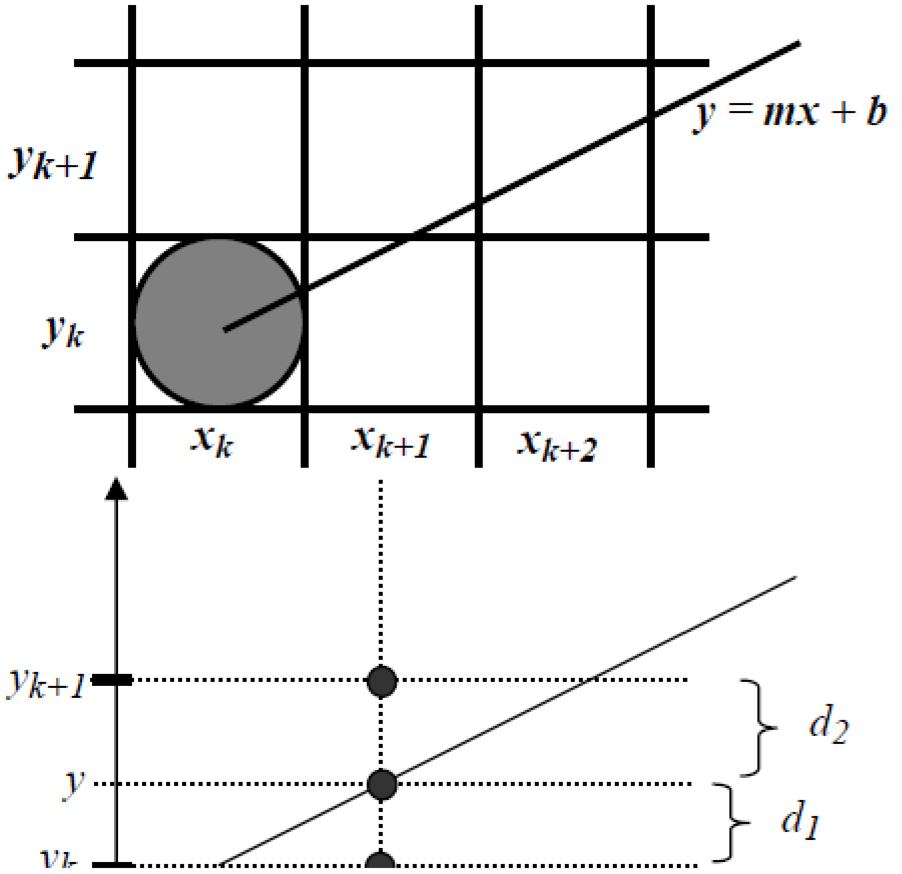
An accurate and efficient raster line-generating algorithm, developed by Bresenham, scans converts lines using only incremental integer calculations that can be adapted to display circles and other curves.

We first consider the scan-conversion process for lines with positive slope less than 1. Pixel positions along a line path are then determined by sampling at unit x intervals. Starting from the left endpoint (x0, y0) of a line, we step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path.

The above figure demonstrates the kth step in this process. Assuming we have determined that the pixel at (xk, yk) is to be displayed, we next need to decide which pixel to plot in column xk+1. Our choices are the pixels at positions

(xk+1, yk) and (xk+1, yk+1).

At sampling position xk+1, we label vertical pixel separations from the mathematical line path as d1 and d2 shown in the diagram. The y coordinate on the mathematical line at pixel column position xk+1 is calculated as



We can summarize Bresenham line drawing for a line with a positive slope less than 1 in the following listed steps:

1. Input 2 endpoints, store left endpoint in (x0, y0).
2. Load (x0, y0) into frame buffer, i.e. plot the first point.
3. Calculate constants Δx, Δy, 2Δy, 2Δy – 2 Δx, and initial value of decision parameter:

   p0 = 2Δy -Δx
4. At each xk along the line, start at k=0,
   test: if pk < 0, plot (xk+1, yk) and

```
                    pk+1 = pk + 2Δy
            else plot (xk+1, yk+1) and
                    pk+1 = pk + 2Δy - 2Δx
     5.  Repeat step (4) Δx times.
```

Implementation of Bresenham Line drawing Algorithm

```
void lineBres (int xa,int ya,int xb, int yb)
 {
int dx = abs( xa – xb) , dy = abs (ya - yb);
 int p = 2 * dy – dx;
 int twoDy = 2 * dy, twoDyDx = 2 *(dy - dx);
 int x , y, xEnd;
/* Determine which point to use as start, which as end * /
 if (xa > x b )
{
x = xb;
y = yb;
 xEnd = xa;
 }
else
{
 x = xa;
 y = ya;
 xEnd = xb;
 }
 setPixel(x,y);
while(x<xEnd)
 {
 x++;
if (p<0)
 p+=twoDy;
else
{
 y++;
p+=twoDyDx;
 }
setPixel(x,y);
 }
 }
```

Example : Consider the line with endpoints (20,10) to (30,18)

The line has the slope m= (18-10)/(30-20)=8/10=0.8 Δx = 10 Δy=8

The initial decision parameter has the value p0 = 2Δy- Δx = 6

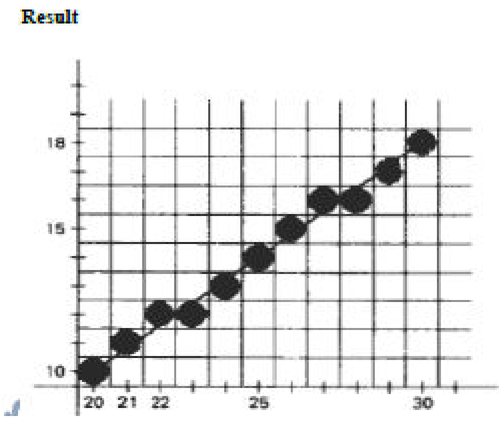
and the increments for calculating successive decision parameters are 2Δy=16 2Δy-2 Δx= -4

We plot the initial point (x0,y0) = (20,10) and determine successive pixel positions along the line path from the decision parameter as

Tabulation

k	pk	(xk+1, yK+1)
0	6	(21,11)
1	2	(22,12)
2	-2	(23,12)
3	14	(24,13)
4	10	(25,14)
5	6	(26,15)
6	2	(27,16)

7	-2	(28,16)
8	14	(29,17)
9	10	(30,18)



Advantages

- Algorithm is Fast
- Uses only integer calculations

Disadvantages

- It is meant only for basic line drawing.

S.NO	RGPV QUESTIONS	Year	Marks
1	Write an algorithm for drawing circle in third quadrant in anticlockwise Direction, using Bresenhams algorithm. And also draw the flowchart.	Dec,2011	5
2	Explain the Brersenham's line algorithm for drawing a line with a slope less than 1 and greater than 0.	Dec,2012	10
3	Rasterize the line from (-1,1) to (5,-8) using Bresenhams line drawing Algorithm.	Dec,2013	10

Unit-02/Lecture-04

Circle-Generating Algorithms

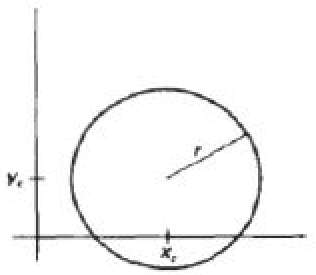
In general, a single procedure can be provided to display either
Circular or Elliptical Curves.
Properties of Circles:
A Circle is defined as the set of points that are all at a given distance r from a center position (xc, yc).

$$(x - xc)2 + (y - yc)2 = r2 \qquad (1)$$

This distance relationship is expressed by the Pythagorean theorem in Cartesian coordinates as

$$y = yc +(-)\ (r2 - (xc - x)2)1/2 \qquad (2)$$

But this is not the best method for generating a circle. One problem with this approach is that it involves considerable computation at each step. Moreover, the spacing between plotted pixel positions is not uniform, as shown in the following figure.



Another way to eliminate the unequal spacing is to calculate points along the circular boundary using polar coordinates r and θ. Expressing the circle equation in parametric polar form yields the pair of equations,

$$\begin{aligned} x &= x_c + r\cos\theta \\ y &= y_c + r\sin\theta \end{aligned} \longrightarrow (3)$$

When a display is generated with these equations using a fixed angular step size, a circle is plotted with equally spaced points along the circumference.Computation can be reduced by considering the symmetry of circles. The shape of the circle is similar in each quadrant.

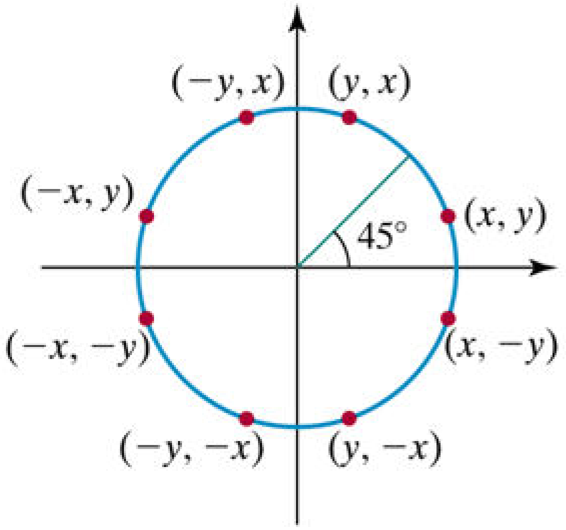
These symmetry conditions are illustrated in the above diagram, where a point at position (x , y) on a one-eighth circle sector is mapped into the seven circle points in the other octants of the xy plane.

Taking advantage of the circle symmetry in this way we can generate all pixel positions around a circle by calculating only the points within the sector from x=0 to x=y.Determining pixel positions along a circle circumference using either equation (1) or equation (3) still requires a good deal of computation time.

Bresenham's line algorithm for raster displays is adapted to circle generation by setting up decision parameters for finding the closest pixel to the circumference at each sampling step.

Bresenham‟s line algorithm for raster displays is adapted to circle generation by setting up decision parameters for finding the closest pixel to the circumference at each sampling step. Square root evaluations would be required to computer pixel siatances from a circular path.

Bresenham’’s circle algorithm avoids these square root calculations by comparing the squares of the pixel separation distances. It is possible to perform a direct distance comparison without a squaring operation. In this approach is to test the halfway position between two pixels to determine if this midpoint is inside or outside the circle boundary. This method is more easily applied to other conics and for an integer circle radius the midpoint approach generates the same pixel positions as the Bresenham circle algorithm. For a straight line segment the midpoint method is equivalent to the bresenham line algorithm. The error involved in locating pixel positions along any conic section using the midpoint test is limited to one half the pixel separation



Unit-02/Lecture-05

Midpoint Circle Algorithm

Midpoint Circle Algorithm

In the raster line algorithm at unit intervals and determine the closest pixel position to the specified circle path at each step for a given radius r and screen center position (xc,yc) set up our algorithm to calculate pixel positions around a circle path centered at the coordinate position by adding xc to x and yc to y.

To apply the midpoint method we define a circle function as

fcircle(x,y) = x2+y2-r2

Any point (x,y) on the boundary of the circle with radius r satisfies the equation fcircle (x,y)=0. If the point is in the interior of the circle, the circle function is negative. And if the point is outside the circle the, circle function is positive

fcircle (x,y) <0, if (x,y) is inside the circle boundary =0,
if (x,y) is on the circle boundary >0,
if (x,y) is outside the circle boundary

The tests in the above eqn are performed for the midposition sbteween pixels near the circle path at each sampling step. The circle function is the decision parameter in the midpoint algorithm. Midpoint between candidate pixels at sampling position xk+1 along a circular path. Fig -1 shows the midpoint between the two candidate pixels at sampling position xk+1. To plot the pixel at (xk,yk) next need to determine whether the pixel at position (xk+1,yk) or the one at position (xk+1,yk-1) is circular to the circle.

Our decision parameter is the circle function evaluated at the midpoint between these two pixels

Pk= fcircle (xk+1,yk-1/2) =(xk+1)2+(yk-1/2)2-r2

If Pk <0, this midpoint is inside the circle and the pixel on scan line yk is closer to the circle boundary. Otherwise the mid position is outside or on the circle boundary and select the pixel on scan line yk -1.

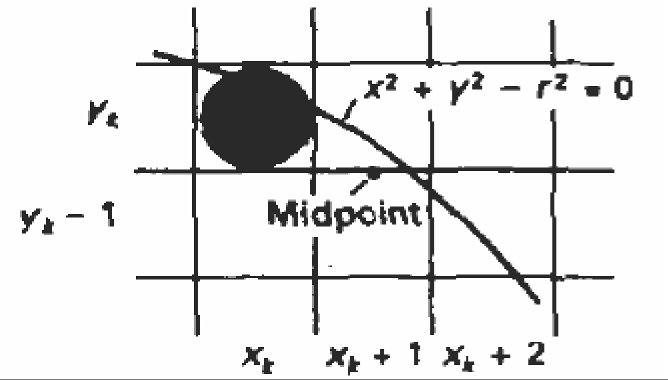
Successive decision parameters are obtained using incremental calculations. To obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position

xk+1+1= xk+2

Pk= fcircle (xk+1+1,yk+1-1/2) =[(xk+1)+1]2+(yk+1-1/2)2-r2

or Pk+1=Pk+2(xk+1)+(y2k+1-y2 k)-(yk+1-yk)+1

Where yk+1 is either yk or yk-1 depending on the sign of Pk .



Increments for obtaining Pk+1 are either 2xk+1+1 (if Pk is negative) or

2xk+1+1-2 yk+1.

Evaluation of the terms 2xk+1 and 2 yk+1 can also be done incrementally as 2xk+1=2xk+2 2 yk+1=2 yk-2 At the Start position (0,r) these two terms have the values 0 and 2r respectively. Each successive value for the 2xk+1 term is obtained by adding 2 to the previous value and each successive value for the 2yk+1 term is obtained by subtracting 2 from the previous value.
The initial decision parameter is obtained by evaluating the circle function at the start position

(x0,y0)=(0,r)

P0= fcircle (1,r-1⁄2) =1+(r-1⁄2)2-r2 or P0=(5⁄4)-r

If the radius r is specified as an integer P0=1-r(for r an integer)

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

Midpoint algorithm

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

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

2. Calculate the initial value of the decision parameter as

$$p_0 = \frac{5}{4} - r$$

3. At each **x_k** position, starting at **k = 0,** perform the following test:
   If **p_k < 0**, the next point along the circle centered on **(0, 0)** is **(x_k +1, y_k)** and

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

   Otherwise, the next point along the circle is **(x_k + 1, y_k - 1)** and

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}, \quad \text{where } 2x_{k+1} = 2x_k + 2 \text{ and } 2y_{k+1} = 2y_k - 2.$$

4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position **(x, y)** onto the circular path centered on **(x_c, y_c)** and plot the coordinate values:

$$\mathbf{x = x + x_c, \qquad y = y + y_c}$$

6. Repeat steps 3 through 5 until **x ≥ y**.

Example : Midpoint Circle Drawing Given a circle radius r=10 The circle octant in the first quadrant from x=0 to x=y.

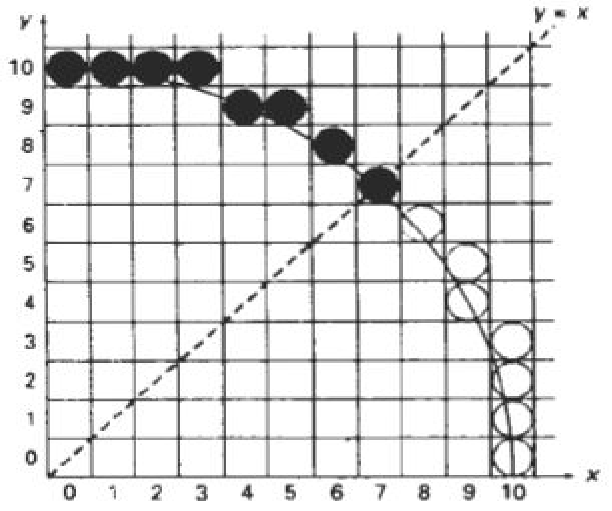
The initial value of the decision parameter is P_0=1-r = - 9

For the circle centered on the coordinate origin, the initial point is (x_0,y_0)=(0,10) and initial increment terms for calculating the decision parameters are $2x_0$=0 , $2y_0$=20

Successive midpoint decision parameter values and the corresponding coordinate positions along the circle path are listed in the following table.

Tabulation

k	pk	(xk+1, yk-1)	2xk+1	2yk+1
0	-9	(1,10)	2	20
1	-6	(2,10)	4	20
2	-1	(3,10)	6	20
3	6	(4,9)	8	18
4	-3	(5,9)	10	18
5	8	(6,8)	12	16
6	5	(7,7)	14	14



Implementation of Midpoint Circle Algorithm

```
void circleMidpoint (int xCenter, int yCenter, int radius)
 {
int x = 0;
 int y = radius;
 int p = 1 - radius;
 void circlePlotPoints (int, int, int, int);
/* Plot first set of points */
circlePlotPoints (xCenter, yCenter, x, y);
 while (x < y)
{      x++ ;
 if (p < 0)
 p +=2*x +1;
else
{       y--;
}
circlePlotPoints(xCenter, yCenter, x, y)
}
 }
 void circlePlotPoInts (int xCenter, int yCenter, int x, int y)
 {
setpixel (xCenter + x, yCenter + y ) ;
 setpixel (xCenter - x. yCenter + y);
 setpixel (xCenter + x, yCenter - y);
setpixel (xCenter - x, yCenter - y ) ;
 setpixel (xCenter + y, yCenter + x);
setpixel (xCenter - y , yCenter + x);
setpixel (xCenter t y , yCenter - x);
setpixel (xCenter - y , yCenter - x);
}
```

S.NO	RGPV QUESTIONS	Year	Marks
1.	Write Bresenham 's Circle algorithm to draw first quadrant of circle with centre(h,k) and radius r.	Dec,2013	10

UNIT 2/LECTURE 6

Midpoint Ellipse drawing Algorithm

An Ellipse is an elongated circle. So, Elliptical Curves can be generated by modifying circle-drawing procedures to take into account the different dimensions along the major and minor axes.

Properties of Ellipses

An Ellipse is defined as the set of points such that the sum of the distances from two fixed positions (foci) is the same for all points.

Our approach here is similar to that used in displaying a raster circle. Given parameters rx, ry, and (xc, yc), we determine points (x, y) for an ellipse in standard position centered on the origin, and then we shift the points so the ellipse is centered at (xc, yc).

The midpoint ellipse method is applied throughout the first quadrant in two parts. The following diagram shows the division of the first quadrant according to the slope of an ellipse with

rx < ry.

Regions 1 and 2 can be processed in various ways. We can start at position (0, ry) and step clockwise along the elliptical path in the first quadrant, shifting from unit steps in x to unit steps in y when the slope becomes less than -1.

Alternatively, we could start at (rx, 0) and select points in a counterclockwise order, shifting from unit steps in y to unit steps in x when the slope becomes greater than -1.

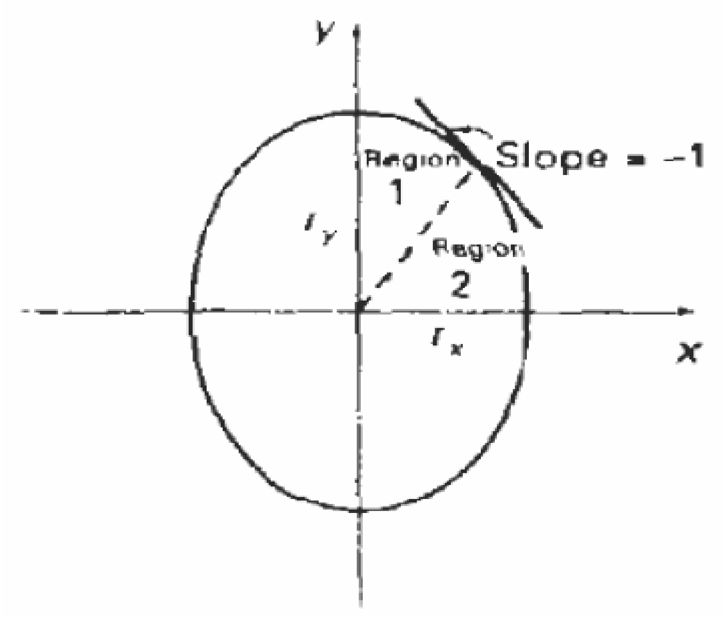
With parallel procssors, we could calculate pixel positions in the two regions simultaneously.

We define an Ellipse Function from equation with (xc,yc) = (0, 0) as

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

which has the following properties:

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



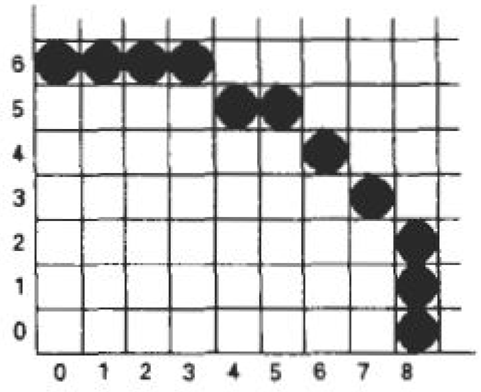
Implementation of Midpoint Ellipse drawing

```
 void ellipseMidpoint (int xCenter, int yCenter, int Rx, int Ry)
 {
 int Rx2=Rx*Rx;
int Ry2=Ry*Ry;
int twoRx2 = 2*Rx2;
int twoRy2 = 2*Ry2;
 int p; int x = 0;
 int y = Ry;
int px = 0;
 int py = twoRx2* y;
 void ellipsePlotPoints ( int , int , int , int ) ;
/* Plot the first set of points */
 ellipsePlotPoints (xcenter, yCenter, x,y ) ;
 / * Region 1 */
 p = ROUND(Ry2 - (Rx2* Ry) + (0.25*Rx2));
 while (px < py)
 {
 x++;
 px += twoRy2;
 i f (p < 0)
 p += Ry2 + px;
else
{
 y - - ;
py -= twoRx2;
 p += Ry2 + px - py;
}

ellipsePlotPoints(xCenter, yCenter,x,y);
 }
/* Region 2 */

 p = ROUND (Ry2*(x+0.5)*' (x+0.5)+ Rx2*(y- l )* (y- l ) - Rx2*Ry2);
while (y > 0 )
 {
y--;
 py -= twoRx2;
 i f (p > 0)
p += Rx2 - py;
else
{
 x++;
 px+=twoRy2;
 p+=Rx2-py+px;
 }
ellipsePlotPoints(xCenter, yCenter,x,y);
 }
}
```

```
 void ellipsePlotPoints(int xCenter, int yCenter,int x,int y)
 {
 setpixel (xCenter + x, yCenter + y);
 setpixel (xCenter - x, yCenter + y);
setpixel (xCenter + x, yCenter - y);
setpixel (xCenter- x, yCenter - y);
}
```



S.NO	RGPV QUESTION	YEAR	MARKS
1.	Write Midpoint Ellipse algorithm ?	Dec 2013,Dec,2011	10

UNIT 2/LECTURE 7

Curves Parametric Equation

Parameters for curve attributes are the same as those for line segments. We can display curves with varying colors, widths, dot dash patterns, and available pen or brush options.
Raster curves of various widths can be displayed using the method of horizontal or vertical pixel spans. Where the magnitude of the curve slope is less than 1, we plot vertical spans; where the slope magnitude is greater than 1, we plot horizontal spans.
Another method for displaying thick curves is to fill in the area between two parallel curve paths, whose separation distance is equal to the desired width. We could do this using the specified curve path as one boundary and setting up the second boundary either inside or outside the original curve path. This approach shifts the original curve path either inward or outward, depending on which direction we choose for the second boundary. We can maintain the original curve position by setting the two boundary curves at a distance of one-half the width on either side of the specified curve path.
Although this method is accurate for generating thick circles, it provides only an approximation to the true area of other thick curves.Curves drawn with pen and brush shapes can be displayed in different sizes and with superimposed patterns or simulated brush strokes.

Parametric equations can be used to generate curves that are more general than explicit equations of the form y=f(x). A quadratic parametric spline may be written as

$$\mathbf{P} = \mathbf{a}_2 t^2 + \mathbf{a}_1 t + \mathbf{a}_0$$

where P is a point on the curve, a0, a1 and a2 are three vectors defining the curve and t is the parameter. The curve passes through three points labelled P0, P1 and P2. By convention the curve starts from pointP0 with parameter value t=0, goes through point P1 when t=t1 (0<t1<1) and finishes at P2 when t=1. Using these conventions we can solve for the three a vectors as follows:

$$\begin{array}{ll} t=0 & \mathbf{P}_0 = \mathbf{a}_0 \\ t=1 & \mathbf{P}_2 = \mathbf{a}_2 + \mathbf{a}_1 + \mathbf{a}_0 \\ t=t_1 & \mathbf{P}_1 = \mathbf{a}_2 t_1^2 + \mathbf{a}_1 t_1 + \mathbf{a}_0 \end{array}$$

and rearranging these equations we get:

$$\mathbf{a}_0 = \mathbf{P}_0$$

$$\mathbf{a}_2 = \frac{(\mathbf{P}_1 - \mathbf{P}_0) - t_1(\mathbf{P}_2 - \mathbf{P}_0)}{t_1(t_1 - 1)}$$

$$\mathbf{a}_1 = \mathbf{P}_2 - \mathbf{P}_0 - \mathbf{a}_2$$

We can now apply this to any set of three points, as shown in the diagram below. It is easy to see the much higher degree of flexibility achieved through the use of parametric equations,

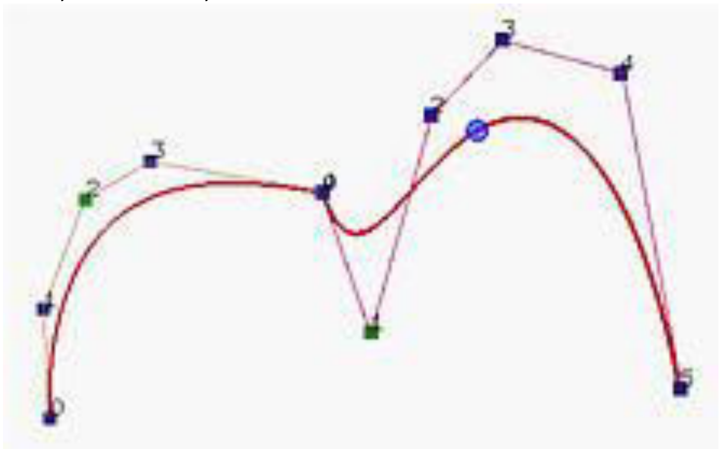
Bézier curve

A Bézier curve is a parametric curve frequently used in computer graphics and related fields. Generalizations of Bézier curves to higher dimensions are called Bézier surfaces, of which the Bézier triangle is a special case.

In vector graphics, Bézier curves are used to model smooth curves that can be scaled indefinitely. "Paths", as they are commonly referred to in image manipulation programs, are combinations of linked Bézier curves. Paths are not bound by the limits of rasterized images and are intuitive to modify.

Bézier curves are also used in the time domain, particularly in animation and user interface design. For example a Bezier curve can be used to specify the velocity over time of an object such as an icon moving from A to B, rather than simply moving at a fixed number of pixels per step. When animators or interface designers talk about the "physics" or "feel" of an operation, they may be referring to the particular Bézier curve used to control the velocity over time of the move in question.

The mathematical basis for Bézier curves – the Bernstein polynomial – has been known since 1912, but its applicability to graphics was understood half a century later. Bézier curves were widely publicized in 1962 by the French engineer Pierre Bézier, who used them to design automobile bodies at Renault. The study of these curves was however first developed in 1959 by mathematician Paul de Casteljau using de Casteljau's algorithm, a numerically stable method to evaluate Bézier curves, at Citroën, another French automaker.



$$\mathbf{C}(u) = \sum_{k=0}^{n} \mathbf{p}_k B_{k,n}(u), \quad 0 \le u \le 1$$

$$B_{k,n}(u) = C(n,k) u^k (1-u)^{n-k}$$

Properties of Bezier Curve

1. The k-th derivative at the start (end) of a Bezier curve depends only on the first (last) (k+1) control points. Two obvious special cases:
    - k=0: The Bezier curve starts at the first control point and stops at the last control point. (In general, it will not pass through any other control point.)
    - k=1: The vector tangent to the Bezier curve at the start (stop) is parallel to the line connecting the first two (last two) control points.
2. A Bezier curve will always be completely contained inside of the Convex Hull of the control points. For planar curves, imagine that each control point is a nail pounded into a board. The shape a rubber band would take on when snapped around the control points is the convex hull. For Bezier curves whose control points do not all lie in a common plane, imagine the control points are tiny balls in space, and image the shape a balloon will take on if it collapses over the balls. This shape is the convex hull in that

case. In any event, a Bezier curve will always lie entirely inside its planar or volumetric convex hull.

3. Closely related to the previous is the fact that adjusting the position of a control point changes the shape of the curve in a "predictable manner". Intuitively, the curve "follows" the control point. In the image below, see how a curve defined in terms of four control points (the magenta curve) changes when one of its control points is moved to the right, yielding the modified (cyan) curve.

4. There is no local control of this shape modification. Every point on the curve (with the exception of the first and last) move whenever any interior control point is moved. This property can also be observed in the image shown in the previous item.

5. Also related to property #2 is the fact that Bezier curves exhibit a variation diminishing property. Informally this means that the Bezier curve will not "wiggle" any more than the control polygon does. In other words, the curve will not wiggle unless the designer specifically introduces wiggling in the control polygon. More formally, the variation diminishing property can be stated as follows: any straight line will intersect legs of the control polygon at least as many times as it crosses the Bezier curve itself. See the example below which illustrates the property with a degree 12 Bezier curve.

6. The effect of control point Pi on the curve is at its maximum at parameter value t = i/n. Among other things, this somewhat ameleriorates problems related to the fact that there is no local control (property #4).

7. Bezier curves exhibit a symmetry property: The same Bezier curve shape is obtained if the control points are specified in the opposite order. The only difference will be the parametric direction of the curve. The direction of increasing parameter reverses when the control points are specified in the reverse order.

8. Bezier curves are invariant under affine transformations, but they are not invariant under projective transformations.

9. Bezier curves are also invariant under affine parameter transformations. That is, while the curve is usually defined on the parametric interval [0,1], an affine transformation mapping [0,1] to the interval [a,b], a≠b, yields the same curve.

S.NO	RGPV QUESTION	YEAR	MARKS
1.	State and explain basic properties of Bezier curves.	June 2011	10
2	What is Bezier Curve? Enlist the general characteristics of Bezier curve. Given that A0(1,1), A1(2,3), A2(4,2) and A3(3,1) are the vertices of Bezier polygon , determine seven points of Bezier curve..	June 2013	10

UNIT 2/LECTURE 8

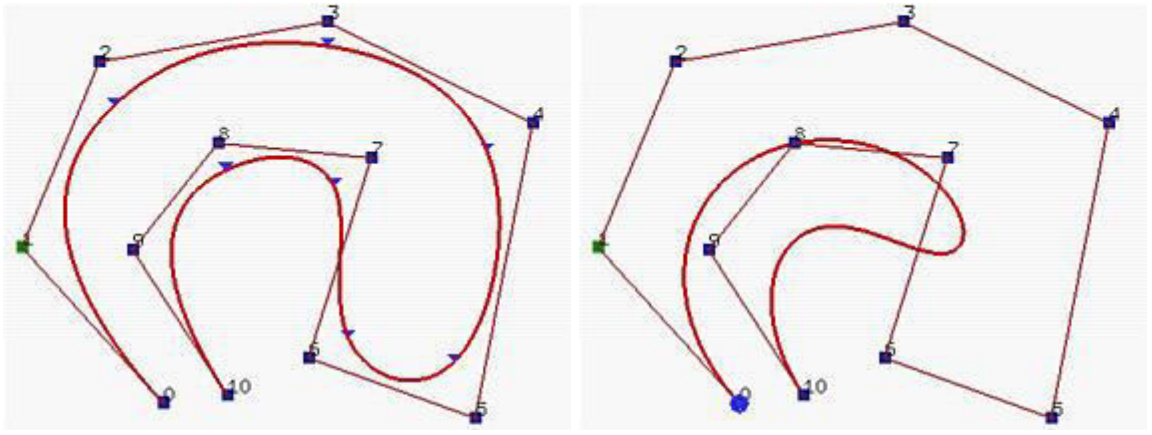
Properties of B-Spline Curve

B-Spline Curve

B-spline curves share many important properties with Bézier curves, because the former is a generalization of the later. Moreover, B-spline curves have more desired properties than Bézier curves. The list below shows some of the most important properties of B-spline curves.

In the following we shall assume a B-spline curve C(u) of degree p is defined by n + 1 control points and a knot vector U = { u0, u1, ..., um } with the first p+1 and last p+1 knots "clamped" (i.e., u0 = u1 = ... = up and um-p = um-p+1 = ... = um).

B-spline curve C(u) is a piecewise curve with each component a curve of degree p.
As mentioned in previous page, C(u) can be viewed as the union of curve segments defined on each knot span. In the figure below, where n = 10, m = 14 and p = 3, the first four knots and last four knots are clamped and the 7 internal knots are uniformly spaced. There are eight knot spans, each of which corresponds to a curve segment. In the left figure below, these knot points are shown as triangles.
This nice property allows us to design complex shapes with lower degree polynomials. For example, the right figure below shows a Bézier curve with the same set of control points. It still cannot follow the control polyline nicely even though its degree is 10.



Properties of B-Spline Curve

1. In general, the lower the degree, the closer a B-spline curve follows its control polyline. The following figures all use the same control polyline and knots are clamped and uniformly spaced. The first figure has degree 7, the middle one has degree 5 and the right figure has degree 3. Therefore, as the degree decreases, the generated B-spline curve moves closer to its control polyline.
2. Equality m = n + p +1must be satisfied. Since each control point needs a basis function and the number of basis functions satisfies m = n + p + 1.
3. Clamped B-spline curve C(u) passes through the two end control points P0 and Pn. Note that basis function N0,p(u) is the coefficient of control point P0 and is non-zero on [u0,up+1). Since u0 = u1 = ... = up = 0 for a clamped B-spline curve, N0,0(u), N1,0(u), ..., Np-1,0(u) are zero and only Np,0(u) is non-zero (recall from the triangular computation scheme). Consequently, if u = 0, then N0,p(0) is 1 and C(0) = P0. A similar discussion can show C(1) = Pn

4. Strong Convex Hull Property: A B-spline curve is contained in the convex hull of its

control polyline. More specifically, if u is in knot span [ui,ui+1), then C(u) is in the convex hul of control points Pi-p, Pi-p+1,..., Pi.

5. f u is in knot span [ui, ui+1), there are only p+1 basis functions (i.e., Ni,p(u), ... , Ni-p+1,p(u), Ni-p,p(u)) non-zero on this knot span. Since Nk,p(u) is the coefficient of control point Pk, only p+1 control points Pi, Pi-1, Pi-2, .., Pi-p have non-zero coefficients. Since on this knot span the basis functions are non-zero and sum to 1, their "weighted" average, C(u), must lie in the convex hull defined by control points Pi, Pi-1, Pi-2, .., Pi-p. The meaning of "strong" is that while C(u) still lies in the convex hull defined by all control points, it lies in a much smaller one.
6. Local Modification Scheme:
   changing the position of control point Pi only affects the curve C(u) on interval[ui, ui+p+1).This follows from another important property of B-spline basis functions. Recall that Ni,p(u) is non-zero on interval [ui, ui+p+1). If u is not in this interval, Ni,p(u)Pi has no effect in computing C(u) since Ni,p(u) is zero. On the other hand, if u is in the indicated interval, Ni,p(u) is non-zero. If Pi changes its position, Ni,p(u)Pi is changed and consequently C(u) is changed.C(u) is Cp-k continuous at a knot of multiplicity k If u is not a knot, C(u) is in the middle of a curve segment of degree p and is therefore infinitely differentiable. If u is a knot in the non-zero domain of Ni,p(u), since the latter is only Cp-k continuous, so does C(u).
7. Variation Diminishing Property:
   The variation diminishing property also holds for B-spline curves. If the curve is in a plane (resp., space), this means no straight line (resp., plane) intersects a B-spline curve more times than it intersects the curve's control polyline.
8. Bézier Curves Are Special Cases of B-spline Curves.
   If n = p (i.e., the degree of a B-spline curve is equal to n, the number of control points minus 1), and there are 2(p + 1) = 2(n + 1) knots with p + 1 of them clamped at each end, this B-spline curve reduces to a Bézier curve.
9. Affine Invariance :
   The affine invariance property also holds for B-spline curves. If an affine transformation is applied to a B-spline curve, the result can be constructed from the affine images of its control points. This is a nice property. When we want to apply a geometric or even affine transformation to a B-spline curve, this property states that we can apply the transformation to control points, which is quite easy, and once the transformed control points are obtained the transformed B-spline curve is the one defined by these new points. Therefore, we do not have to transform the curve.

The Advantage of Using B-spline Curves

- B-spline curves require more information (i.e., the degree of the curve and a knot vector) and a more complex theory than Bézier curves. But, it has more advantages to offset this shortcoming. First, a B-spline curve can be a Bézier curve.
- Second, B-spline curves satisfy all important properties that Bézier curves have.
- Third, B-spline curves provide more control flexibility than Bézier curves can do. For example, the degree of a B-spline curve is separated from the number of control points.
- More precisely, we can use lower degree curves and still maintain a large number of control points. We can change the position of a control point without globally changing the shape of the whole curve (local modification property). Since B-spline curves satisfy the strong convex hull property, they have a finer shape control.
- Moreover, there are other techniques for designing and editing the shape of a curve such as changing knots.

S.NO	RGPV QUESTION	YEAR	MARKS
1.	State and explain, basic properties of B-spline curves	Dec,2012	10
2.	Determine the blending functions for uniform, periodic B-spline Curve for d=3.	Dec,2014	10

UNIT 2/LECTURE 9

Anti-aliasing

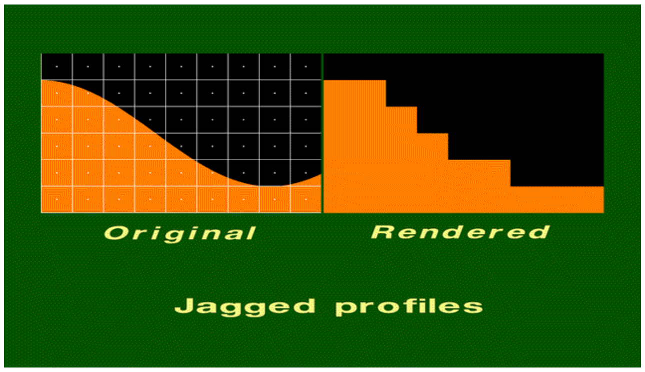
Anti-aliasing is any of a number of techniques to combat the problems of aliasing in a sampled signal such as a digital image or digital audio recording.

- In digital signal processing, anti- aliasing is the technique of minimizing aliasing (jagged or blocky patterns) when representing a high resolution signal at a lower resolution.
- In most cases, anti- aliasing means removing this part data at too high a frequency to represent. When sampling is performed without removing this part of the signal, it causes undesirable artifacts such as the black – and- white noise near the top of figure.

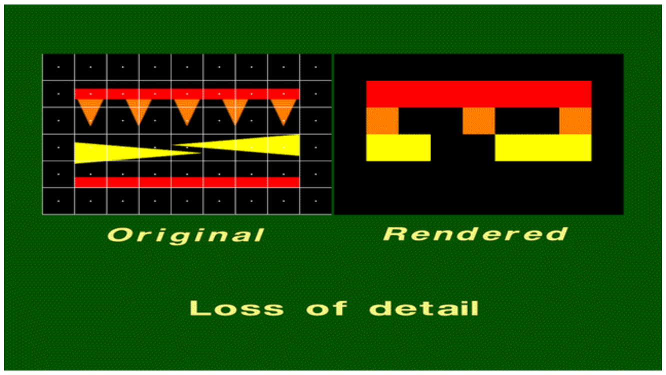
Effects caused by Aliasing:

The errors caused by aliasing are called artefacts. Common aliasing artefacts include jagged profiles, disappearing or improperly rendered fine detail, and disintegrating textures.

1. Jagged Profiles:The picture on the left shows the sampling grid super imposed on the original scene. The picture on the right is the rendered image. A jagged profile is quite evident in the rendered image. Also known as "Jaggies", jagged silhouettes are probably the most familiar effect caused by aliasing. Jaggies are especially noticeable where there is a high contrast between the interior and the exterior of the silhouette.

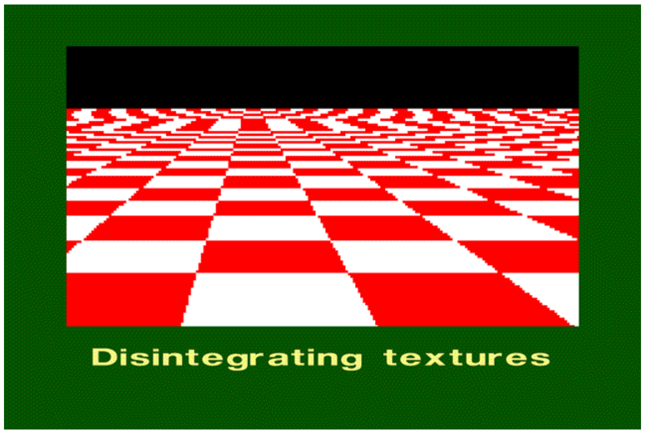


2. Improperly Rendered Detail: The original scene on the left shows a group of small polygons. In the rendered scene, one of the two red rectangles disappears entirely, and the other doubles in width. Two of the orange triangles disappear. Although the two yellow triangles are identical in size, one is larger than the other in the rendered image.



3. Disintegrating textures: This is a checkered texture on a plane. The checkers should become smaller as the distance from the viewer increases. However, the checkers become larger or irregularly shaped when their distance from the viewer becomes too great. Simply increasing the resolution will not remove this artefact. Increasing the

resolution will only move the artefact closer the horizon.



S.NO	RGPV QUESTION	YEAR	MARKS
1	What is anti aliasing effects?	Dec,2013	10