When the start endpoint is at the right (for the same slope), we ... obtain y positions from Eq. 3-8. Similarly, when the absolute v... slope is greater than 1, we use $\Delta y = -1$ and Eq. 3-9 or we use Δy ...

This algorithm is summarized in the following procedure, w... input the two endpoint pixel positions. Horizontal and vertical d... tween the endpoint positions are assigned to parameters dx and d... ence with the greater magnitude determines the value of parameter ... ing with pixel position (x_a, y_a), we determine the offset needed a... generate the next pixel position along the line path. We loop through steps times. If the magnitude of dx is greater than the magnitude of ... is less than xb, the values of the increments in the x and y directions a... respectively. If the greater change is in the x direction, but xa is grea... then the decrements -1 and $-m$ are used to generate each new point o... Otherwise, we use a unit increment (or decrement) in the y direction an... crement (or decrement) of $1/m$. We assume that points are to be plo... bilevel intensity system, so that the call to setPixel with an intensity ...

```
procedure lineDDA (xa, ya, xb, yb : integer);
var
  dx, dy, steps, k : integer;
  xIncrement, yIncrement, x, y : real;
begin
  dx := xb - xa;
  dy := yb - ya;
  if abs(dx) > abs(dy) then steps := abs(dx)
  else steps := abs(dy);
  xIncrement := dx / steps;
  yIncrement := dy / steps;
  x := xa;
  y := ya;
  setPixel (round(x), round(y), 1);
  for k := 1 to steps do
    begin
      x := x + xIncrement;
      y := y + yIncrement;
      setPixel (round(x), round(y), 1)
    end
end;  {lineDDA}
```

The DDA algorithm is a faster method for calculating pix... the direct use of Eq. 3-1. It eliminates the multiplication in Eq. 3-1 b... of raster characteristics, so that appropriate increments are applied ... direction to step to pixel positions along the line path. The acc... roundoff error in successive additions of the floating-point increme... can cause the calculated pixel positions to drift away from the true ... long line segments. Furthermore, the rounding operations and ... arithmetic in procedure lineDDA are still time-consuming. We can ... performance of the DDA algorithm by separating the increments m... integer and fractional parts so that all calculations are reduced to ... tions. A method f... ...or steps...

... we digitize the line with endpoints (20, 10) and ... of 0.8, with

$$\Delta x = 10, \qquad \Delta y = 8$$

The initial decision parameter has the value

$$p_0 = 2\Delta y - \Delta x$$
$$= 6$$

P1

and the increments for calculating successive decision parameters are

$$2\Delta y = 16, \qquad 2\Delta y - 2\Delta x = -4$$

We plot the initial point $(x_0, y_0) = (20, 10)$, and determine successive pixel positions along the line path from the decision parameter as

k	p_k	(x_{k+1}, y_{k+1})	k	p_k	(x_{k+1}, y_{k+1})
0	6	(21, 11)	5	6	(26, 15)
1	2	(22, 12)	6	2	(27, 16)
2	−2	(23, 12)	7	−2	(28, 16)
3	14	(24, 13)	8	14	(29, 17)
4	10	(25, 14)	9	10	(30, 18)

6 +16 -20

P2 :

A plot of the pixels generated along this line path is shown in Fig. 3-9.

An implementation of Bresenham line drawing for slopes in the range $0 < m < 1$ is given in the following procedure. Endpoint pixel positions for the line are passed to this procedure, and pixels are plotted from the left endpoint to the right endpoint. The call to `setPixel` loads the intensity value 1 into the frame buffer at the specified (x, y) pixel position.

```
procedure lineBres (xa, ya, xb, yb : integer);
var
  dx, dy, x, y, xEnd, p : integer;
begin
  dx := abs(xa - xb);
  dy := abs(ya - yb);
  p := 2 * dy - dx;
  { determine which point to use as start, which as end }
  if xa > xb then
    begin
      x := xb;
      y := yb;
      xEnd := xa
    end { if xa > xb }
  else
    begin
```

```
      x := xa;
      y := ya;
      xEnd := xb
    end;
  setPixel (x, y, 1);
  while x < xEnd do
    begin
      x := x + 1;
      if p < 0 then p := p + 2 * dy
      else
        begin
          y := y + 1;
          p := p + 2 * (dy - dx)
        end;
      setPixel (x, y, 1)
    end
end; { lineBres }
```

Bresenham's algorithm is generalized to lines with arbitrary slop sidering the symmetry between the various octants and quadrants plane. For a line with positive slope greater than 1, we interchange th the x and y directions. That is, we step along the y direction in unit steps culate successive x values nearest the line path. Also, we could revise gram to plot pixels starting from either endpoint. If the initial position with positive slope is the right endpoint, both x and y decrease as we right to left. To ensure that the same pixels are plotted regardless of th endpoint, we always choose the upper (or the lower) of the two candid whenever the two vertical separations from the line path are equal (d negative slopes, the procedures are similar, except that now one coor creases as the other increases. Finally, special cases can be handled Horizontal lines ($\Delta y = 0$), vertical lines ($\Delta x = 0$), and diagonal lines wi $|\Delta y|$ each can be loaded directly into the frame buffer without proce hrough the line-plotting algorithm.

'arallel Line Algorithms

The line-generating algorithms we have discussed so far determine ions sequentially. With a parallel computer, we can calculate pix

```
procedure circleMidpoint (xCenter, yCenter,
var
  p, x, y : integer;
  procedure plotPoints;
  begin
    setPixel (xCenter + x, yCenter + y, 1);
    setPixel (xCenter - x, yCenter + y, 1);
    setPixel (xCenter + x, yCenter - y, 1);
    setPixel (xCenter - x, yCenter - y, 1);
    setPixel (xCenter + y, yCenter + x, 1);
    setPixel (xCenter - y, yCenter + x, 1);
    setPixel (xCenter + y, yCenter - x, 1);
    setPixel (xCenter - y, yCenter - x, 1)
  end;  plotPoints
begin
  x := 0;
  y := radius;
  plotPoints;
  p := 1 - radius;
  while x < y do
    begin
      if p < 0 then
        x := x + 1
      else
        begin
          x := x + 1;
          y := y - 1
        end;
      if p < 0 then
        p := p + 2 * x + 1
      else
        p := p + 2 * (x - y) + 1;
      plotPoints
    end;
end;  { circleMidpoint }
```