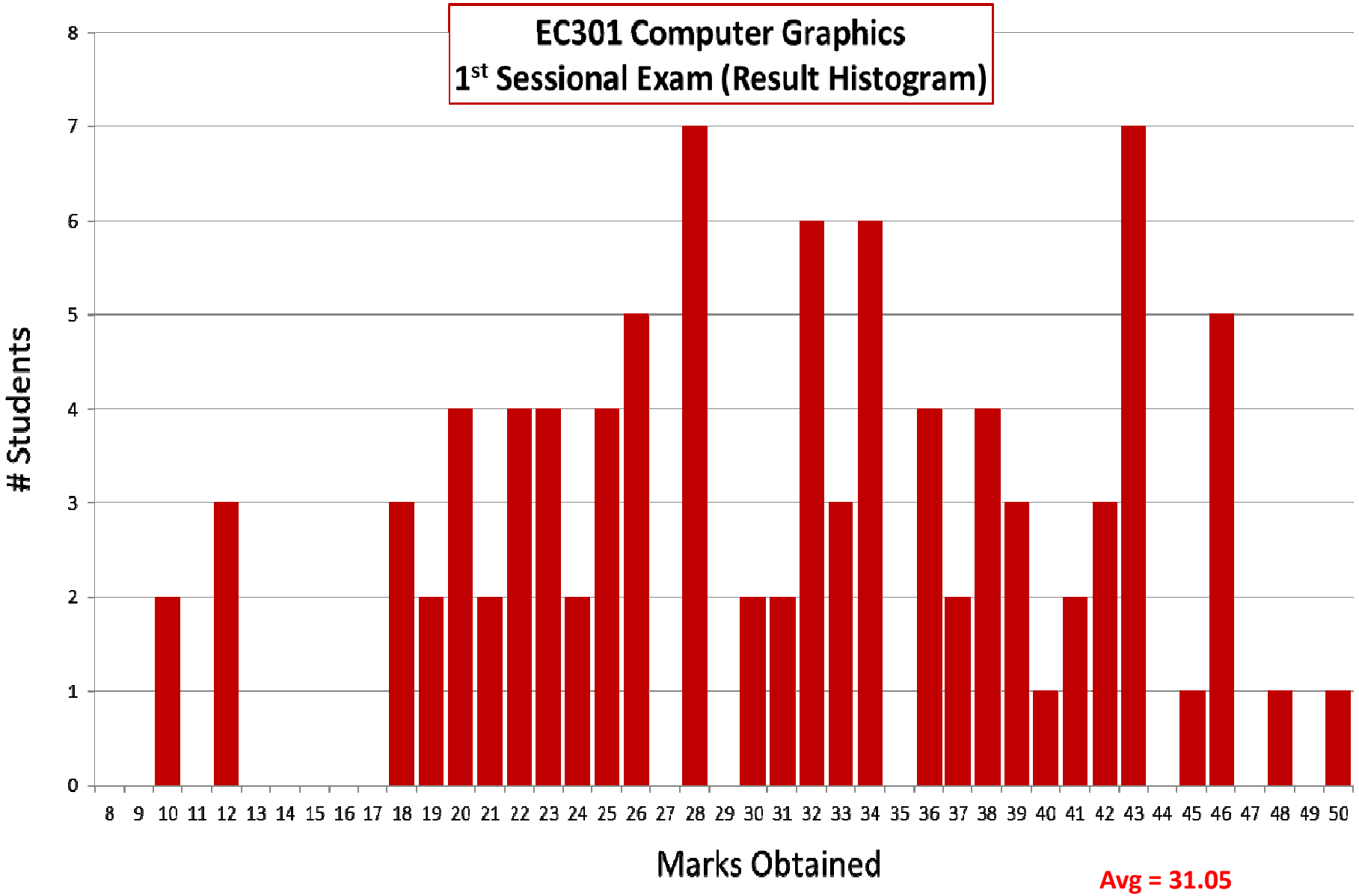
EC-301 Computer Graphics

Lecture Slides- Wk10:

- **1st Sessional Exam- Result Histogram**
- Revision of Lecture- Wk9
- Parallel Line Drawing Algorithms


EC301 Computer Graphics
1st Sessional Exam (Result Histogram)
Students
0 1 2 3 4 5 6 7 8
8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
Marks Obtained
Avg = 31.05

Graphics Output Primitives

- Functions in a graphics package used to describe various picture components
  - **geometric primitives** describe the geometry of objects (point positions, straight line segments, circles, cones, curves etc)
  - most graphics systems provide functions for displaying character strings
- This lecture is about output primitives available in OpenGL, and device-level algorithms for implementing the primitives

Coordinate Reference Frames

- World-coordinate frame and objects' positions
- Scene description: coordinate positions, color and coordinate extents (*bounding box* or *rectangle*)
- Objects displayed by passing scene info to the viewing routines

- **Screen Coordinates**
  - Locations on video monitors- integer screen coordinates, correspond to pixel positions in frame buffer
  - *Scan line number* and *column number* (0 – max)
  - Pixel positions wrt the top-left corner of the screen
  - Software commands can be used to set up any convenient reference frame (e.g., origin at lower-left)

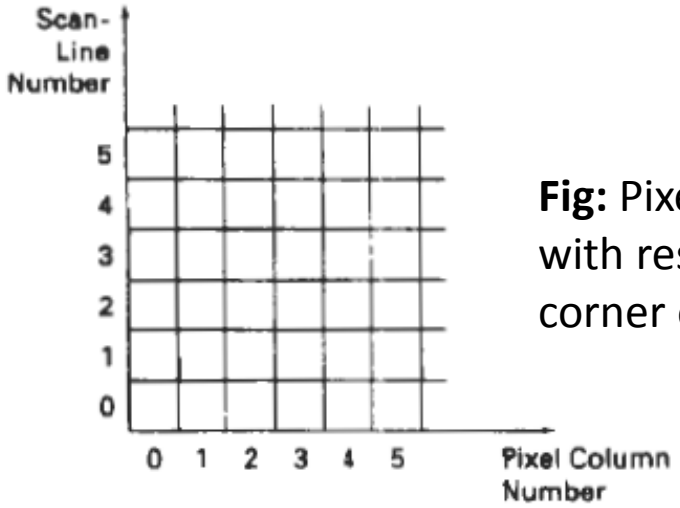


Fig: Pixel positions referenced with respect to the lower-left corner of a screen area.

- We could use non-integer Cartesian values for picture description
- Coordinate values used to describe geometry of scene are converted by the viewing routine to integer pixel positions
- Scan-line algorithms set rules to determine which pixels to be displayed (example: line path from endpoint coordinates)
- Store and retrieve frame-buffer setting for a pixel location:

```
setPixel (x, y);
getPixel (x, y, color);
```

- Additional screen-coordinate info needed for 3D scenes

- **Absolute and Relative Coordinate Specifications**
  - *Absolute Coordinate:* values specified are actual positions within the coordinate system in use
  - *Relative Coordinate:* position specified as an offset from the last position referenced (used in drawings with pen plotters, artist's drawing and painting systems)
  - Example: for location (3, 8) being last ref position, a rel position (2, -1) would correspond to abs position of (5, 7)

OpenGL: 2D Reference Frame

- **gluOrtho2D** function to set up ref frame (*x* and *y* coordinate limits for picture as arguments)
- Identity matrix used as the projection matrix before defining coordinate range (to avoid errors)

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ( );
gluOrtho2D (xmin, xmax, ymin, ymax);
```

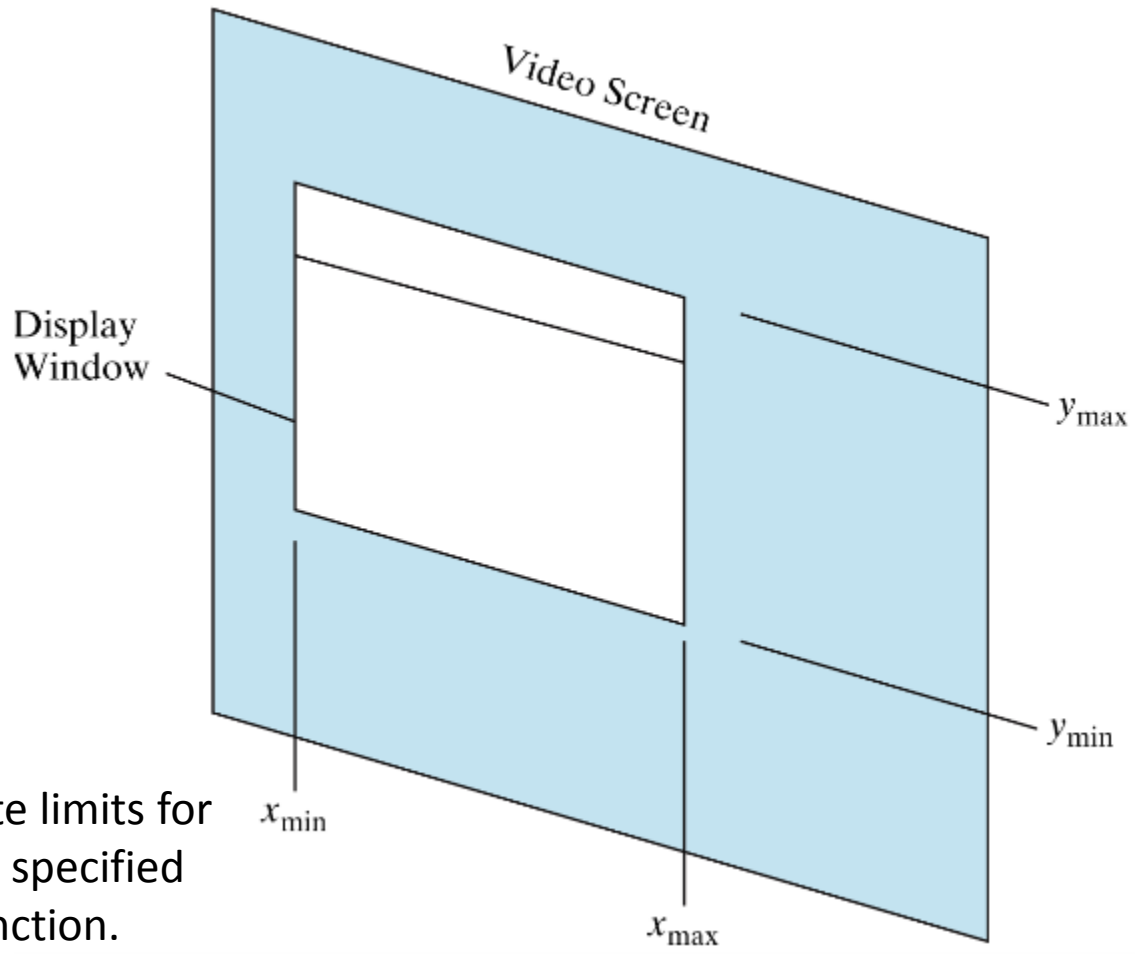


Fig: World-coordinate limits for a display window, as specified in the **glOrtho2D** function.

- Objects that are specified within these coordinate limits will be displayed within the OpenGL window

OpenGL Point Functions

- OpenGL primitives' displayed with default size and colour: one screen pixel, and white
- **glVertex** function to specify coordinates for any point position

```
glBegin(GL_POINTS);
  glVertex2i (50, 100);
  glVertex2i (75, 150);
  glVertex2i (100, 200);
glEnd ( );
```

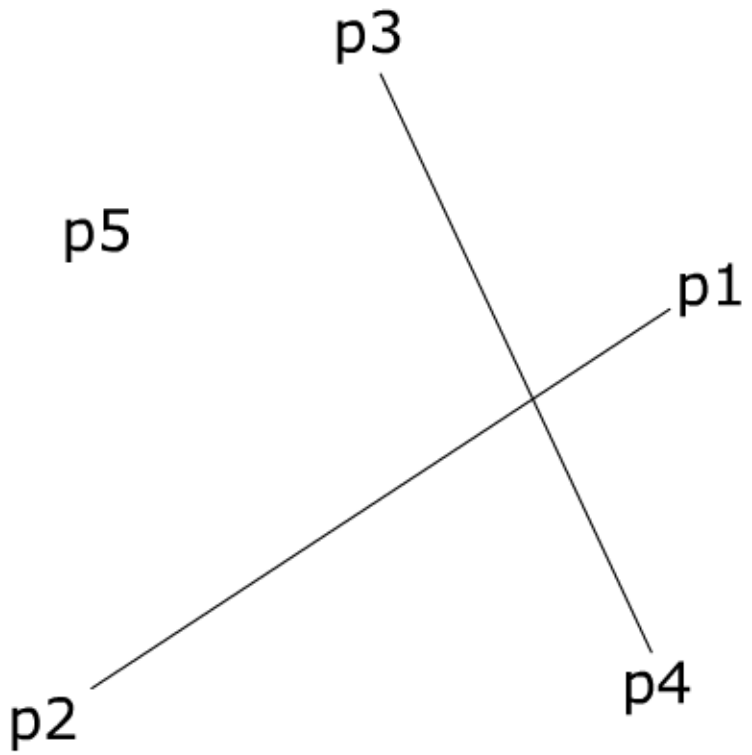
– Alternatively, we could specify the coordinate values in arrays

```
int pt1 [ ] = (50, 100);
int pt2 [ ] = (75, 150);
int pt3 [ ] = (100, 200);
glBegin(GL_POINTS);
  glVertex2iv (pt1);
  glVertex2iv (pt2);
  glVertex2iv (pt3);
glEnd ( );
```

OpenGL Line Functions

- Three symbolic constants in OpenGL
  - **GL_LINES** results in a set of unconnected lines

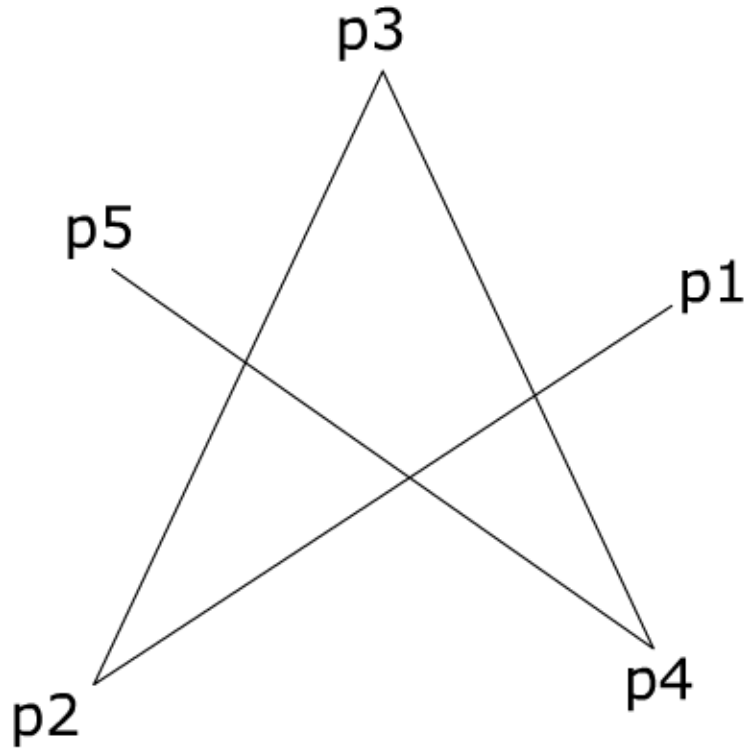
```
glBegin(GL_LINES);
  glVertex2iv (p1);
  glVertex2iv (p2);
  glVertex2iv (p3);
  glVertex2iv (p4);
  glVertex2iv (p5);
glEnd ( );
```



OpenGL Line Functions (Cont.)

- **GL_LINE_STRIP** gives sequence of connected line segments between the first and last endpoints (polyline)

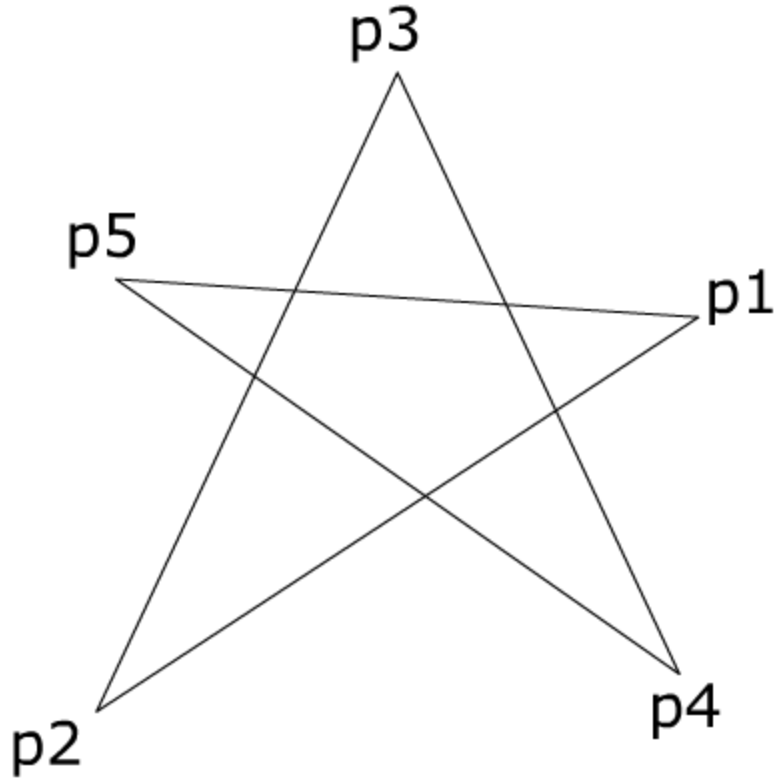
```
glBegin(GL_LINE_STRIP);
  glVertex2iv (p1);
  glVertex2iv (p2);
  glVertex2iv (p3);
  glVertex2iv (p4);
  glVertex2iv (p5);
glEnd ( );
```



OpenGL Line Functions (Cont.)

- **GL_LINE_LOOP** produces a closed polyline (additional line to connect first and last endpoints)

```
glBegin(GL_LINE_LOOP);
  glVertex2iv (p1);
  glVertex2iv (p2);
  glVertex2iv (p3);
  glVertex2iv (p4);
  glVertex2iv (p5);
glEnd ( );
```



Scan Conversion

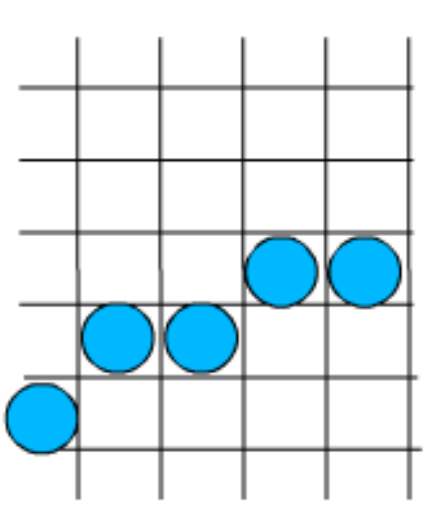
- Converting output primitives into frame buffer updates. Choose which pixel contain which intensity value (task of *graphics controller*)

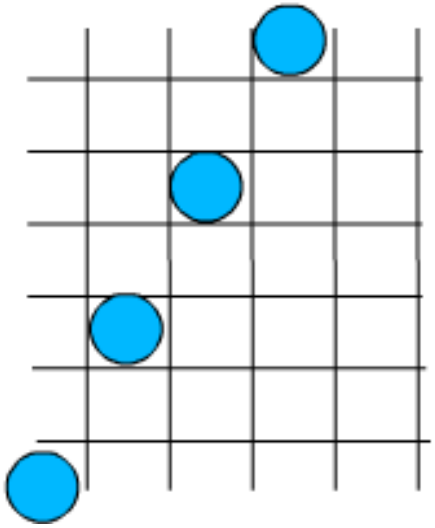
h

- Constraints
  - Straight lines should appear as a straight line
  - Primitives should start and end accurately
  - Primitives should have consistent brightness along their length
  - They should be drawn rapidly

Line Drawing Algorithms

- Project endpoints to integer screen coordinates and determine the nearest pixel positions along the line path between the two endpoints
- A computed line position is rounded to pixel position, resulting into stair-step appearance (“the jaggies”)





– Pixel positions along a straight-line path are determined from the geometric properties of line

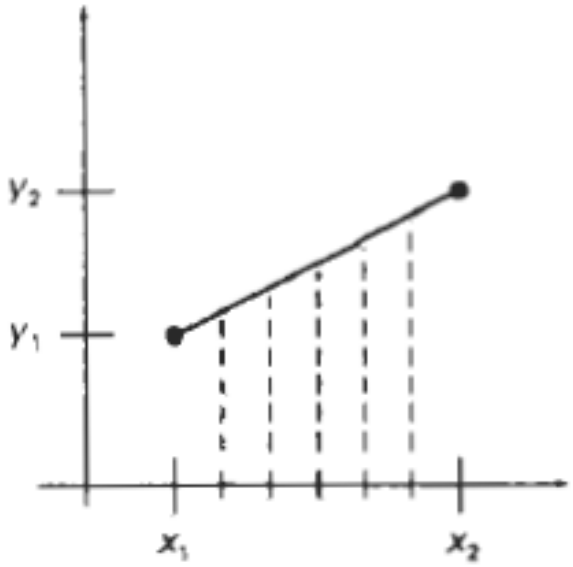
$$y = m \cdot x + b$$

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

$$b = y_0 - m \cdot x_0$$

$$\Rightarrow \quad \delta y = m \cdot \delta x$$

$$\delta x = \frac{\delta y}{m}$$



$$\delta y = m.\delta x, \;\&\; \delta x = \frac{\delta y}{m}$$

- These Eqs form basis for determining deflection voltages in analog displays, such as vector-scan systems
- For lines with slope magnitudes $|m| < 1$, δx can be set proportional to a small horizontal deflection voltage
- For lines with slope magnitudes $|m| > 1$, δy can be set proportional to a small vertical deflection voltage
- On Raster systems, lines are plotted with pixels, and step sizes are constrained by pixel separations

DDA Algorithm

- Digital Differential Analyzer (DDA) is a scan-conversion line algorithm (based on $\delta y \; or \; \delta x$)
- A line is sampled at unit intervals in one coordinate and corresponding integer values nearest the line path are determined for the other coordinate

$$\delta y = m \,.\delta x$$

$$\rightarrow \quad y_{k+1} = y_k + m \quad for \quad \delta x = 1,\ 0 < m < 1$$

$$\delta x = \frac{\delta y}{m}$$

$$\rightarrow \quad x_{k+1} = x_k + \frac{1}{m} \quad for \quad \delta y = 1,\ m \geq 1$$

- DDA is faster than directly implementing ($y=mx+b$). No floating point multiplications. We have floating point additions at each step.
- But what about round off errors?
- Can we get rid of floating point operations completely?

Bresenham's Line Algorithm

- An accurate and efficient raster line-generating algorithm which uses only integer calculations
- Can be adopted to display circles and other curves
- **Basic Idea:**
  - All lines can be placed in one of four categories:
    A. Steep positive slope (m > 1)
    B. Gradual positive slope (0 < m <= 1)
    C. Steep negative slope (m < -1)
    D. Gradual negative slope (0 >= m >= -1)
  - In each case, there are only 2 choices for the next pixel to be plotted!

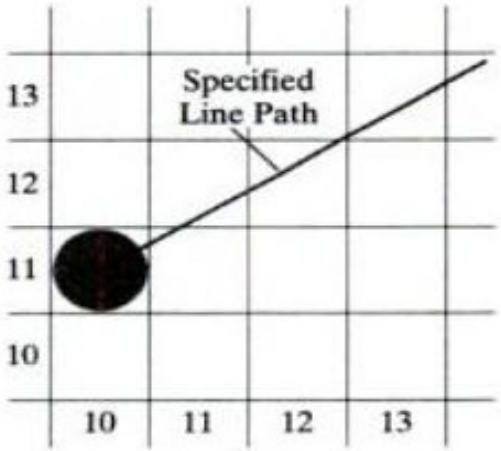


FIGURE 3-8 A section of a display screen where a straight-line segment is to be plotted, starting from the pixel at column 10 on scan line 11.

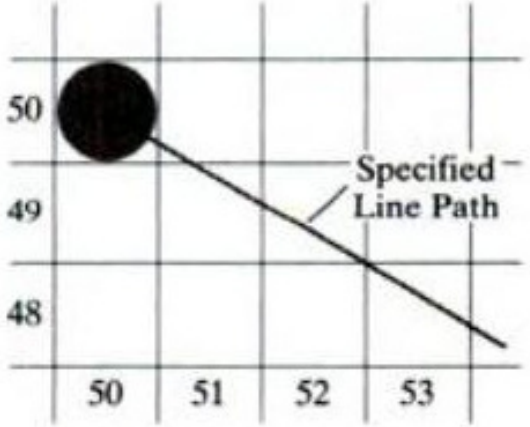
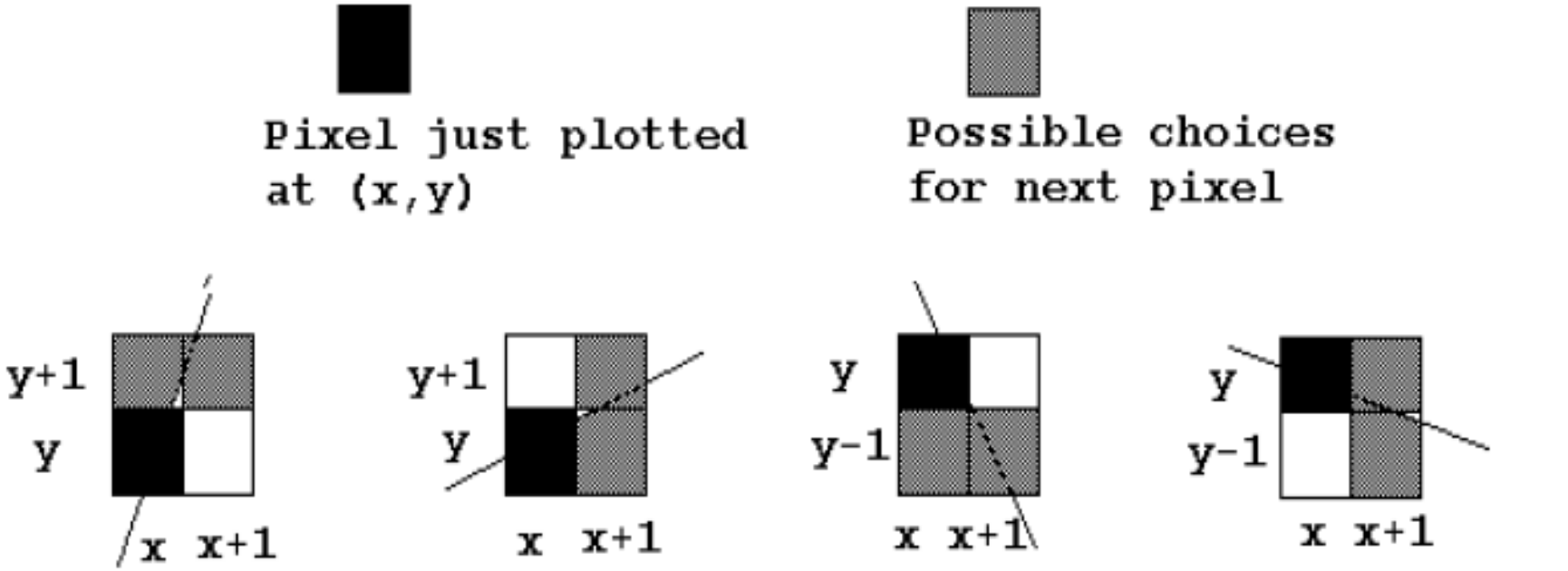


FIGURE 3-9 A section of a display screen where a negative slope line segment is to be plotted, starting from the pixel at column 50 on scan line 50.

– **The Four Bresenham Cases:**



A. steep +m
next point:
(x,y+1)
or
(x+1,y+1)

B. gradual +m
next point:
(x+1,y)
or
(x+1,y+1)

C. steep -m
next point:
(x,y-1)
or
(x+1,y-1)

D. gradual -m
next point:
(x+1,y)
or
(x+1,y-1)

- Algorithm selects the next pixel position by testing the sign of an integer parameter whose value is proportional to the difference between the vertical separations of the two pixels from the actual line path

- Considering the scan-conversion procedure for lines with positive slope less than 1 (+m *gradual*):
  - Sample at unit "**x**" intervals
  - Start from the left endpoint (x_0, y_0) of a given line
  - Step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path

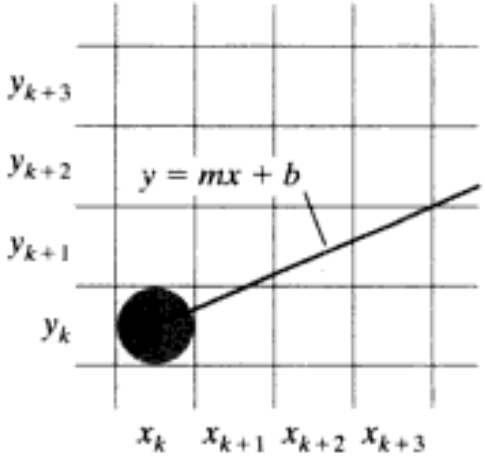


FIGURE 3-10 A section of the screen showing a pixel in column x_k on scan line y_k that is to be plotted along the path of a line segment with slope $0 < m < 1$.

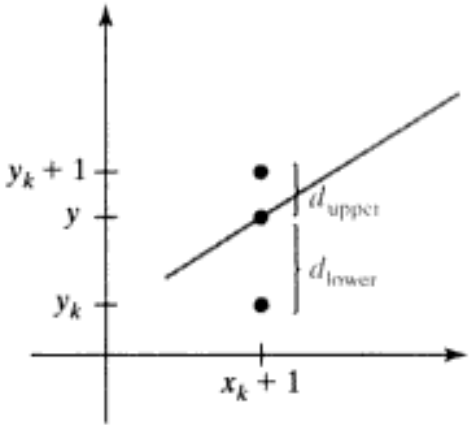


FIGURE 3-11 Vertical distances between pixel positions and the line y coordinate at sampling position $x_k + 1$.

- Assume we have determined that the pixel at (x_k, y_k) is to be displayed:
  - Our choices are the pixels at positions (x_{k+1}, y_k) and (x_{k+1}, y_{k+1})
  - At $(x_k + 1 = x_{k+1})$, label vertical pixel separations from the mathematical line path as d_{lower} & d_{upper}
  - The y coordinate on the mathematical line at pixel column position $x_k + 1$ is calculated as:

$$y = m\,(x_k + 1) + b \rightarrow (1)$$

Then

$$d_{lower} = y - y_k$$
$$= m\,(x_k + 1) + b - y_k \rightarrow (2)$$

and

$$d_{upper} = (y_k + 1) - y$$
$$= y_k + 1 - m\,(x_k + 1) - b \rightarrow (3)$$

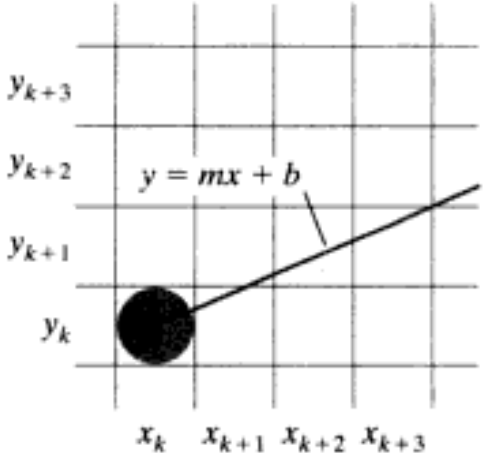


FIGURE 3-10 A section of the screen showing a pixel in column x_k on scan line y_k that is to be plotted along the path of a line segment with slope $0 < m < 1$.

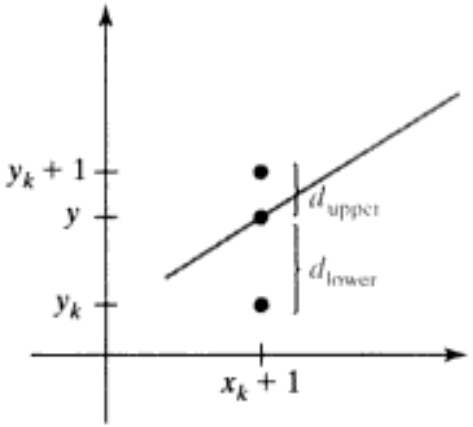


FIGURE 3-11 Vertical distances between pixel positions and the line y coordinate at sampling position $x_k + 1$.

- Test to determine which pixel is closest to the line path:

$$d_{lower} - d_{upper} = 2m(x_k + 1) - 2y_k + 2b - 1 \rightarrow (4)$$

- A decision parameter p_k for the k^{th} step:

  substituting $m = \Delta y / \Delta x$ in (4),

$$p_k = \Delta x\,(d_{lower} - d_{upper})$$
$$= 2\Delta y.x_k - 2\Delta x.y_k + c \rightarrow (5)$$

- If the pixel at y_k is "closer" to the line path than the pixel at $y_k + 1$ (that is, $d_{lower} < d_{upper}$), then decision parameter p_k is negative. In that case, we plot the lower pixel; otherwise we plot the upper pixel.

– Values of successive decision parameters:

$$p_{k+1} = 2\Delta y.x_{k+1} - 2\Delta x.y_{k+1} + c$$

Subtracting Eq (5) from the preceding Eq:

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k) \rightarrow (6)$$

But $x_{k+1} = x_k + 1$, so that

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k) \rightarrow (7)$$

Where the term $y_{k+1} - y_k$ is either 0 or 1, depending on the sign of parameter p_k.

– The first parameter, $\mathbf{p_0}$, is evaluated from Eq (5) at the starting pixel position $\mathbf{(x_0, y_0)}$ and with **m** evaluated as **Δy / Δx**:

$$\mathbf{p_0 = 2\Delta y - \Delta x} \rightarrow \mathbf{(8)}$$

– The constants **2Δy** and **2Δy –2Δx** are calculated once for each line to be scan converted, so the arithmetic involves only integer addition and subtraction of these two constants.

Bresenham's Line-Drawing Algorithm for $|m| < 1.0$

1. Input the two line endpoints and store the left endpoint in (x_0, y_0).
2. Set the color for frame-buffer position (x_0, y_0); i.e., plot the first point.
3. Calculate the constants Δx, Δy, $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

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

4. At each x_k along the line, starting at $k = 0$, perform the following test. If $p_k < 0$, the next point to plot is $(x_k + 1, y_k)$ and

$$p_{k+1} = p_k + 2\Delta y$$

Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

5. Perform step 4 $\Delta x - 1$ times.

EXAMPLE 3-1 Bresenham Line Drawing

To illustrate the algorithm, we digitize the line with endpoints (20, 10) and (30, 18). This line has a slope of 0.8, with

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

The initial decision parameter has the value:

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

$$= 6$$

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})
0	6	(21, 11)
1	2	(22, 12)
2	−2	(23, 12)
3	14	(24, 13)
4	10	(25, 14)

k	p_k	(x_{k+1}, y_{k+1})
5	6	(26, 15)
6	2	(27, 16)
7	−2	(28, 16)
8	14	(29, 17)
9	10	(30, 18)

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k) \rightarrow (7)$$

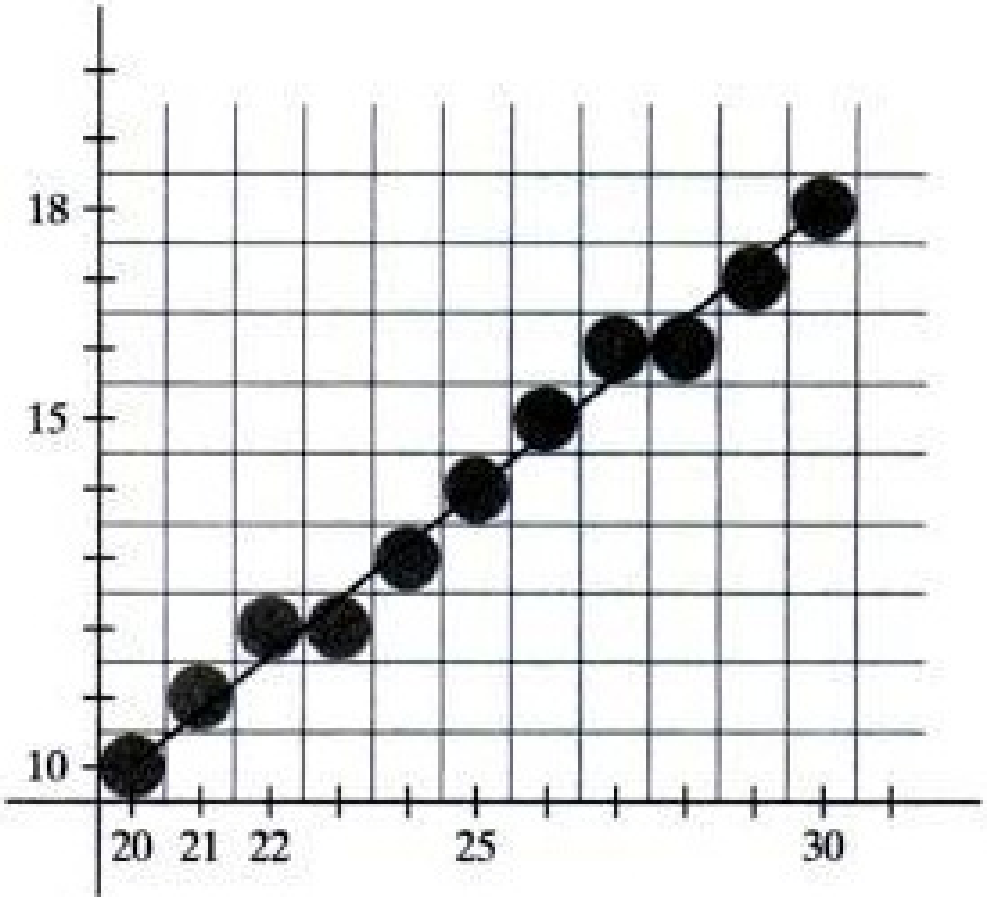


FIGURE 3-12 Pixel positions along the line path between endpoints (20, 10) and (30, 18), plotted with Bresenham's line algorithm.

- For a line with positive slope **m>1**, interchange the role of x and y directions
- We could revise the program to plot pixels starting from either endpoint
- To ensure same pixels are plotted regardless of the starting side, always chose the upper (or lower) of two candidate pixels when $\mathbf{d_{lower} = d_{upper}}$.
- Negative slopes: only change is one coordinate decreases as the other increases
- Special cases: **Δy=0**, **Δx=0**, diagonal lines (|**Δx**|=|**Δy**|) each be loaded directly to frame buffer.

Parallel Line Algorithms

- Line-generating algorithms discussed so far (DDA & Bresenham's Algo) work sequentially
- How to calculate multiple pixel positions along a line path simultaneously?

- Given n_p processors, we can set up a parallel Bresenham's line algorithm:
  - Subdivide the line path into n_p partitions, and
  - Simultaneously generate line segments in each subinterval
  - Distance between beginning x-positions of adjacent partitions can be calculated as $(0 < m < 1)$:

$$\Delta x_p = \frac{\Delta x + n_p - 1}{n_p}$$

  - Starting x-coordinate for k^{th} partition:

$$x_k = x_0 + k\Delta x_p$$

Example: If we have n_p = 4 processors, with Δx = 15, the width of the partition is 4 and starting x values of the partitions are x_0, x_0 + 4, x_0 + 8, and x_0 +12.

- To apply Bresenham's algorithm over partitions, we need for each partition:
  - the initial value for the y coordinate, and
  - Initial value for the decision parameter

- The change $\Delta \boldsymbol{y_p}$ in the y direction over each partition:

$$\Delta y_p = m \Delta x_p$$

- At the k^{th} partition, the starting y coordinate:

$$y_k = y_0 + round \quad (k \Delta y_p)$$

- The initial decision parameter for Bresenham's algorithm at the start of the k^{th} subinterval:

$$p_k = (k\Delta x_p)(2\Delta y) - round\,(k\Delta y_p)(2\Delta x) + 2\Delta y - \Delta x$$

  - How to extend the parallel Bresenham's algorithm to a line with slope greater than 1.0?
  - How about the cases where lines have negative slopes?

- Another way to set up parallel algorithms on raster systems is to assign each processor to a particular group of screen pixels.