

CSI 422/502: Lecture 04 Calculation problems:

1. How many bytes of memory is used by the frame buffer of a 1152x864 display when the depth is 32 bits?

$$1152 \times 864 \times (32\text{bits}) / (8\text{bits/byte})$$

$$= 3,981,312 \text{ in other units, } 3,981,312 / (1024 * 1024) = 3.797 \text{ Megabytes}$$
2. How many megabytes per second are transmitted from the frame buffer to the display when the refresh rate is 80Hz?

$$(3.797\text{MB}) \times (80\text{frames/sec}) = 303.7\text{MBytes/second}$$

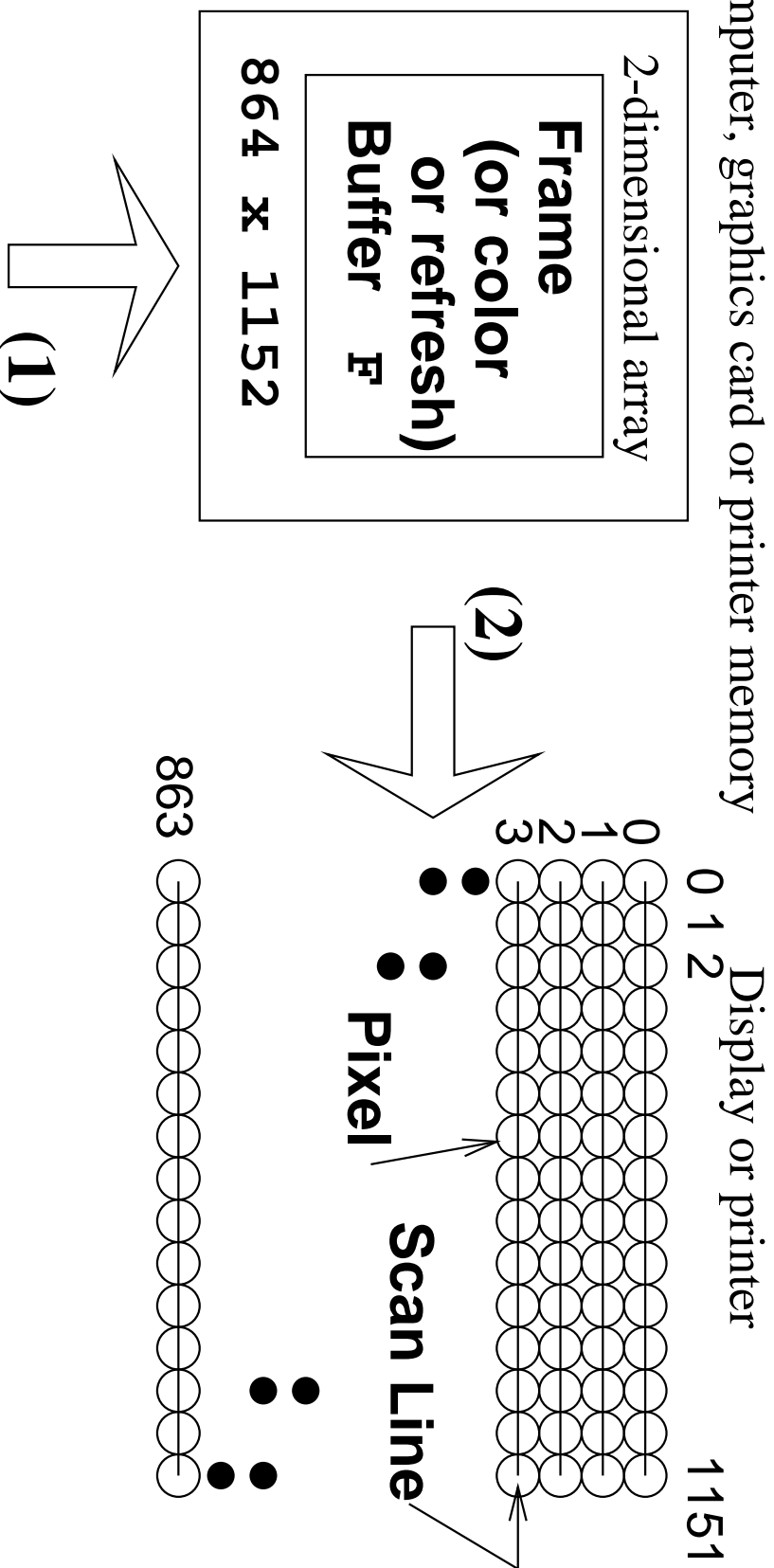
3. Suppose 10 nanoseconds ($1\text{ns}=10^{-9}=1 \text{ Billionth sec.}$) avg. time is used to transfer 32 bits from a CPU to the frame buffer. How long will it take to fill the above frame buffer? What is the maximum number of refills per second, approximately?

$$1152 \times 864 \times (10\text{nanoseconds}) = 9,953,280\text{ns.} = 0.010\text{sec./refill}$$

$$\text{Number of refills/second} = 1 / (0.010\text{sec./refill}) = 100.\text{refills/sec.}$$

But this is very optimistic. Suppose the video card is on a (old) 100MHz PCI bus, so at least 10,000 ns. are needed per 32 bit transfer. Then a full refill takes $(0.01 \times 1000) = 10 \text{ seconds}$.

Computer, graphics card or printer memory



(1) How do graphics applications store data into the frame buffer?(SW+(network?)+graph. accelerator HW.)

(2) How does hardware retrieve, translate and transmit frame buffer data to the display or printing device? (1 whole frame per refresh).

Begin to answer (1)... Given OpenGL line primitive, what happens?

```
glBegin(GL_LINES);  
    glVertex2i( 180, 15 );  
    glVertex2i( 10, 145 );  
glEnd( );  
glFlush( );
```

1. Transform **world coordinates** to **device** (or **pixel coordinates**, for each endpoint.
2. (In a modern system, transmit endpoint data to graphics device.)
3. **Rastorize** the line: Compute which pixels to modify.

Mathematical functions, two uses:

1. **Transform** or **map** something: $F(n) = 2n + 1$ *transforms* each integer into a corresponding *odd* integer.
2. **Constrain** or **Classify** something: With $F(x, y) = (Mx + B) - y$,
 - The truth of $F(x, y) = 0$, equivalently, $y = Mx + B$, determine which points (x, y) are on a line.
 - $F(x, y) < 0$ classifies (x, y) *above* the line; $F(x, y) > 0$ classifies

(x, y) below the line

Introductory example of **Transformations**

We refer to HB 6-1, 6-2 and 6-3 but use formulas without matrices for now—matrices are in **YOUR** future!

We go directly from:

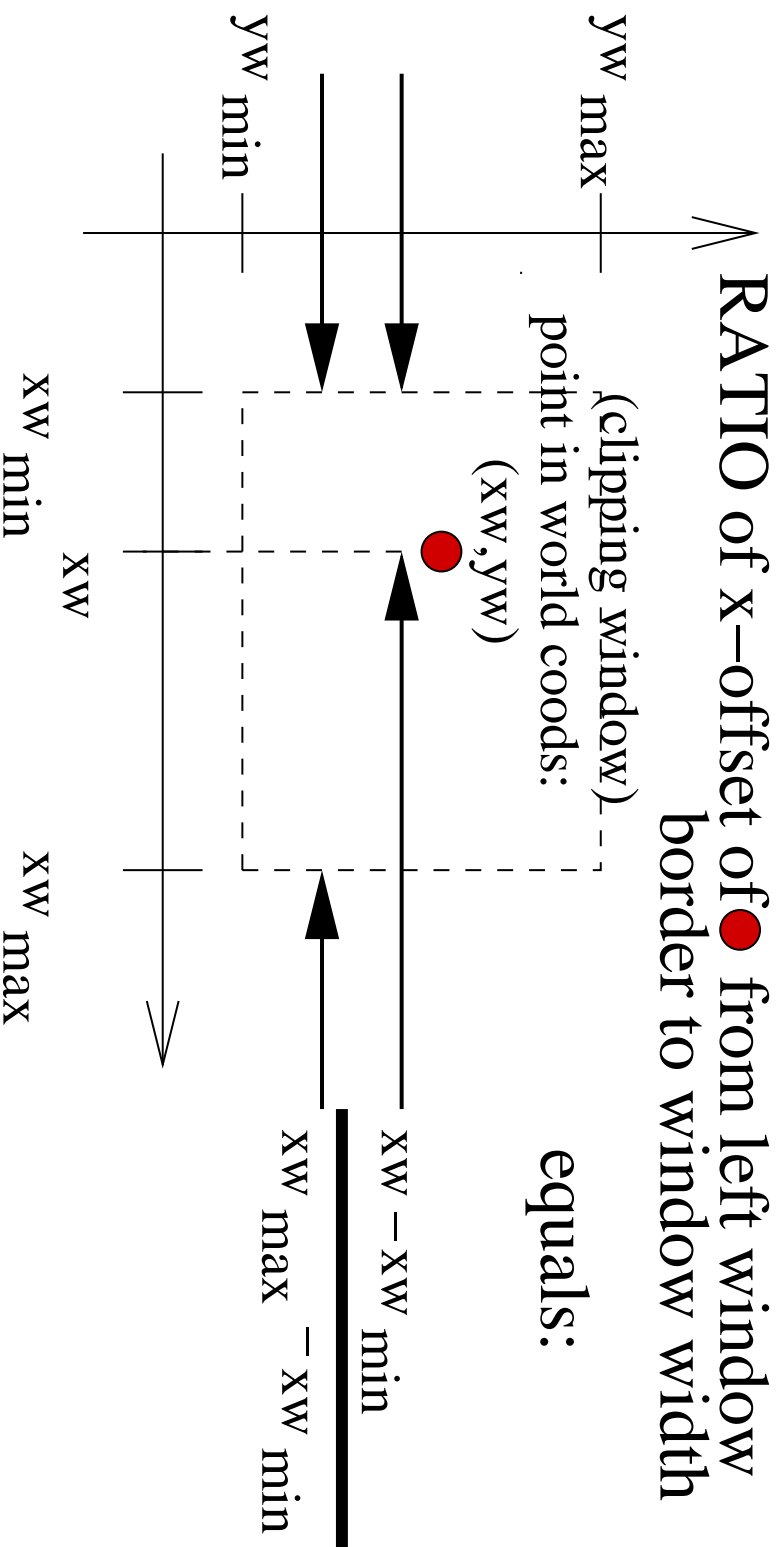
1. A **clipping window** in *world coordinates*
.....**TO**.....
2. A **screen viewport** in *screen coordinates*

In our first OpenGL example,

```
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
gluOrtho2D(0.0, 200.0, 0.0, 150.0);
```

will determine the clipping window in the world coordinate system.

Two **Ratios** determine the location of a point inside the clipping window, here within the *world coordinate system*.



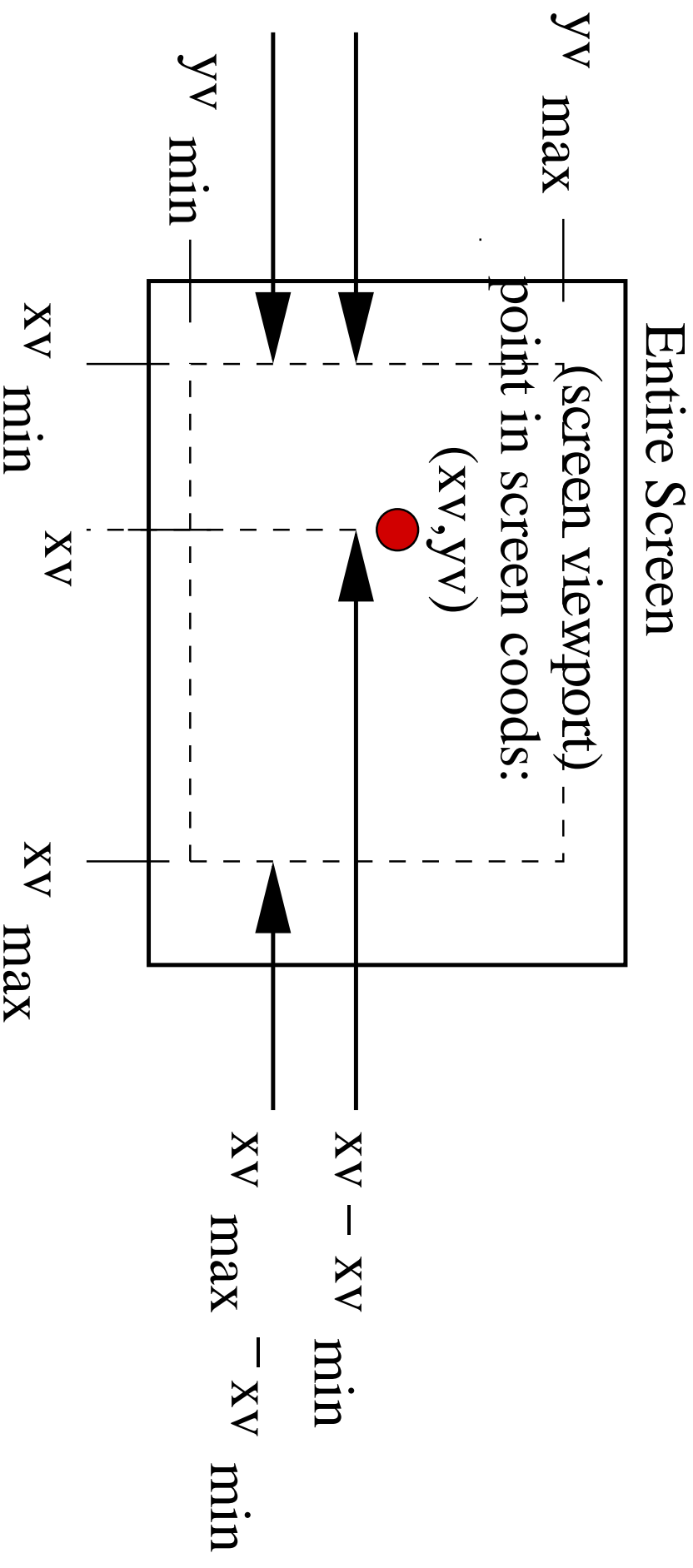
The analogous **ratio** for the y position is $(yw - yw_{min}) / (yw_{max} - yw_{min})$.

In our first OpenGL example,

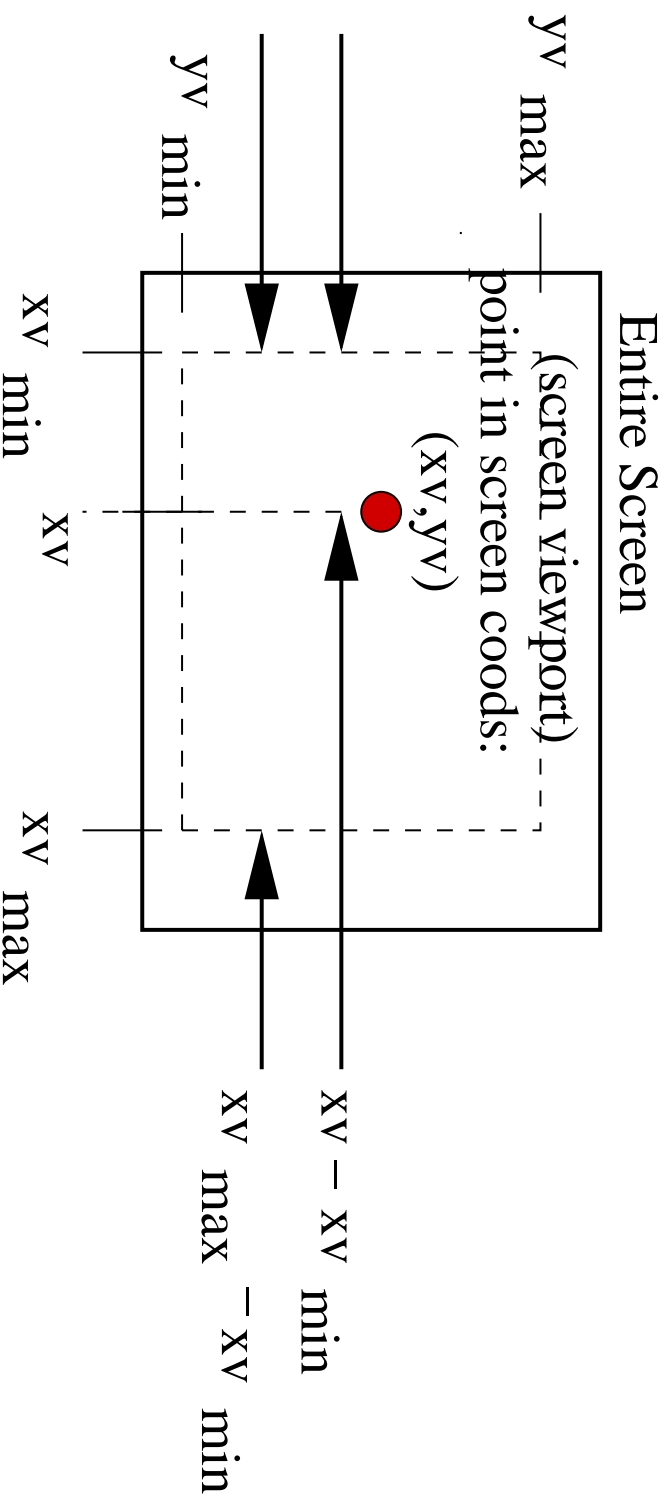
```
glutInitWindowSize(400, 300);
```

determines the screen viewport as

$$xv_{min} = 0$$
$$xv_{max} = 399$$
$$yv_{min} = 0$$
$$yv_{max} = 299$$



Goal: Figure out the screen coordinates of the point so the *position* *determining ratios* for the point in the world **clipping window** are the same as the corresponding ratios for the point in the screen viewport.



Let's compute the screen coordinates (x^v, y^v) of this point by

transforming the world coordinates (x^w, y^w). Problem: With *known* values $x^w, y^w, x^{wmin}, x^{wmax}, y^{wmin}, y^{wmax}, x^{vmin}, x^{vmax}$ and y^{vmax} , figure out (unknowns) x^v and y^v , to solve equations:

$$\frac{x^v - x^{vmin}}{x^{vmax} - x^{vmin}} = \frac{x^w - x^{wmin}}{x^{wmax} - x^{wmin}}; \quad \frac{y^v - y^{vmin}}{y^{vmax} - y^{vmin}} = \frac{y^w - y^{wmin}}{y^{wmax} - y^{wmin}}$$

Figure out x^v and y^v to make corresponding ratios be equal.

$\frac{x^v - x^{vmin}}{x^{vmax} - x^{vmin}}$ means offset of x^v screen coordinate relative to width of screen viewport, both measured in pixel units.

$\frac{xw - xw_{min}}{xw_{max} - xw_{min}}$ means offset of xw world coordinate relative to width of clipping window within the world coordinate system, both measured in world coordinate units.

Write equation that says these ratios are equal, and solve for xv . (ditto for yv):

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

Problem: Given equation with unknown variable xv ,

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

solve the equation for xv :

$$xv - xv_{min} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}} \times (xv_{max} - xv_{min})$$

(We multiplied both sides by $(xv_{max} - xv_{min})$)

$$xv = xv_{min} + \frac{xw - xw_{min}}{xw_{max} - xw_{min}} \times (xv_{max} - xv_{min})$$

(We added xv_{min} to both sides.)

Geometric significance of the solution formula:

$$xv = xv_{min} + \frac{xw - xw_{min}}{xw_{max} - xw_{min}} \times (xv_{max} - xv_{min})$$

(1) $V_1 = xw - xw_{min}$ **transforms** position xw by **moving** it distance xw_{min} to the left, to get value V_1 .

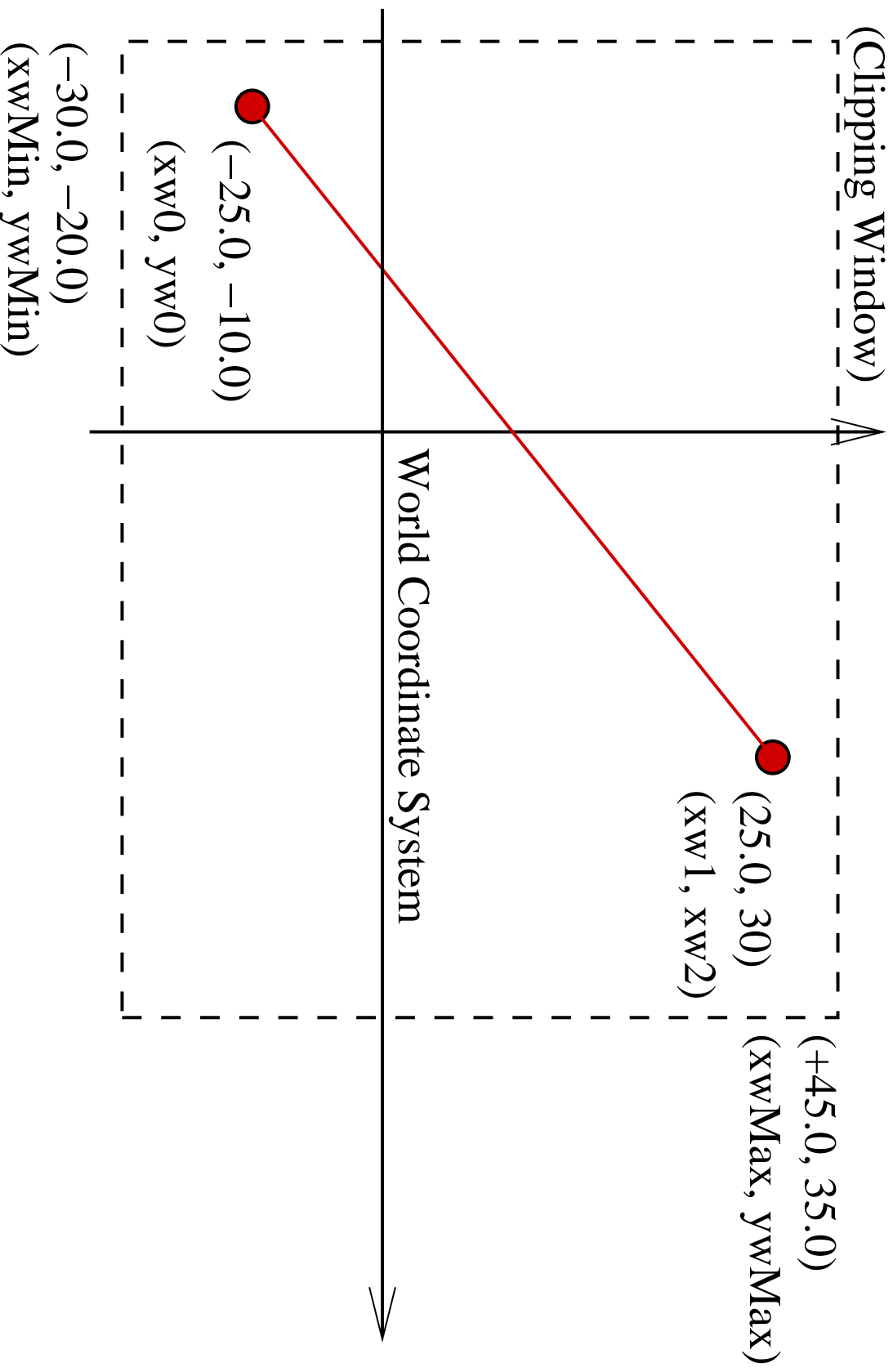
(2) $V_2 = V_1 / (xw_{max} - xw_{min})$ **transforms** value V_1 by **scaling** it down by the clipping window width: The result is the ratio which is between 0 and 1.

(3) $V_3 = V_2 \times (xv_{max} - xv_{min})$ **scales** V_2 up by the factor of *screen viewport width*.

(4) $xv = V_3 + xv_{min}$ **moves** position V_3 to the right, so the position becomes V_3 pixel units right of the left viewport boundary.

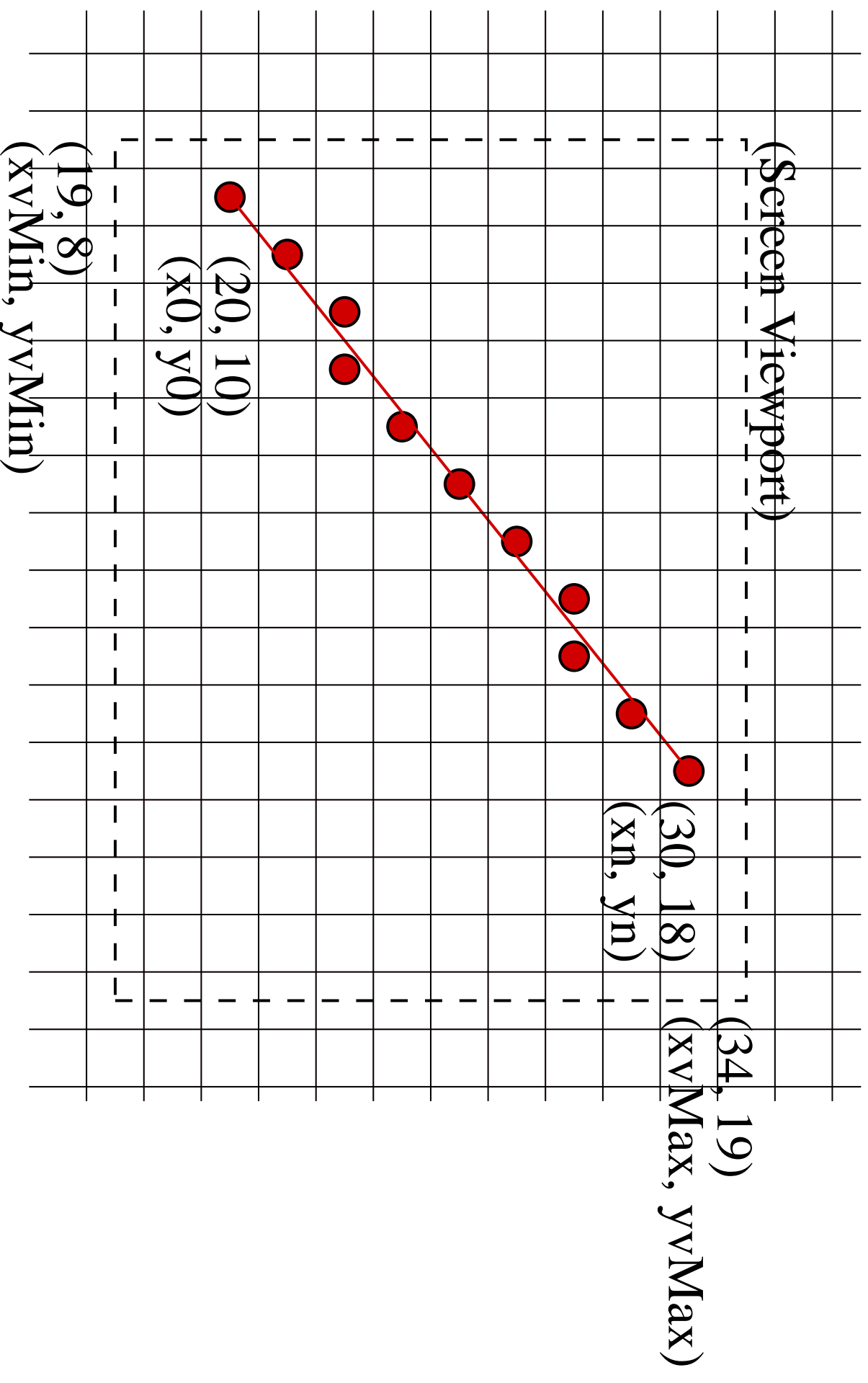
Technically, *move* here means **translate**, which means *move something without changing its size or orientation*.

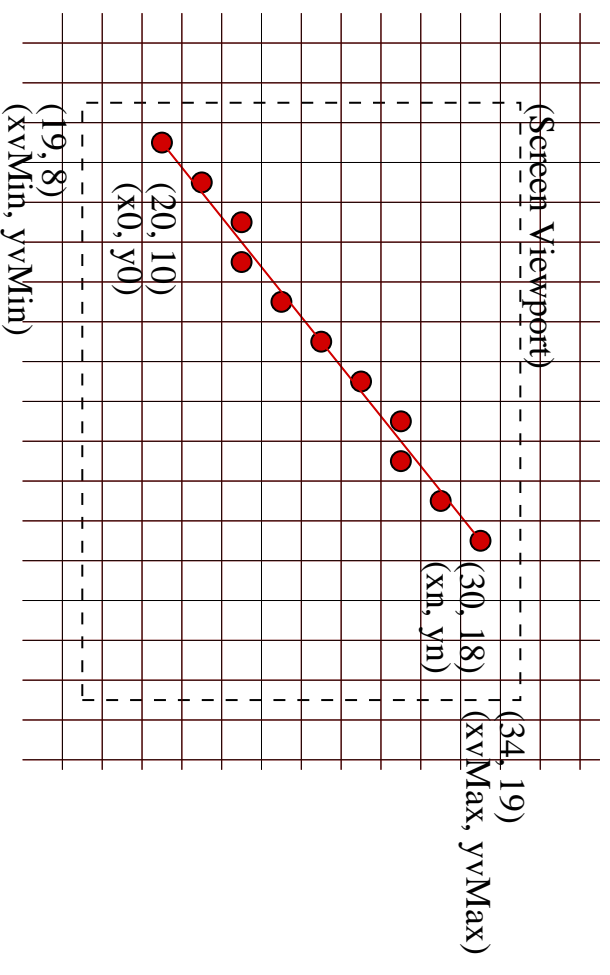
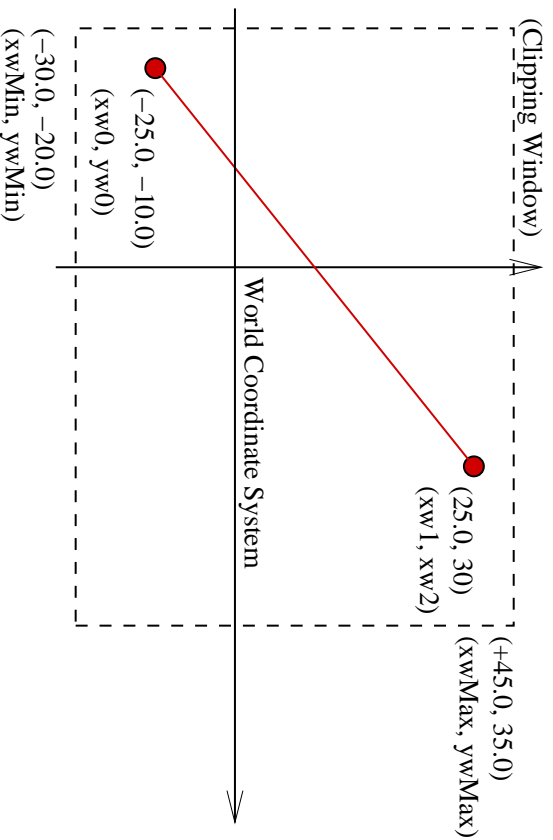
Basic Line Drawing: Begin with a clipping window plus one line primitive in world coordinates,



and given a viewport in screen coordi-

nates, transform the line endpoints (x_0, y_0) , (x_n, y_n) into screen coordinates:





$$xv = xv_{min} + \frac{xw - xw_{min}}{xw_{max} - xw_{min}} \times (xv_{max} - xv_{min}) \text{ etc. for } yv$$

$$xv = 19 + \frac{xw - (-30.0)}{45.0 - (-30.0)} \times (34 - 19) ; yv = 8 + \frac{yw - (-20.0)}{35.0 - (-20.0)} \times (19 - 8)$$

$$xv = 19 + \frac{xw + 30.0}{75.0} \times (15) ; yv = 8 + \frac{yw + 20.0}{55.0} \times (11)$$

$$xv = 19 + (xw + 30.0)(0.2) ; yv = 8 + (yw + 20.0)(0.2)$$

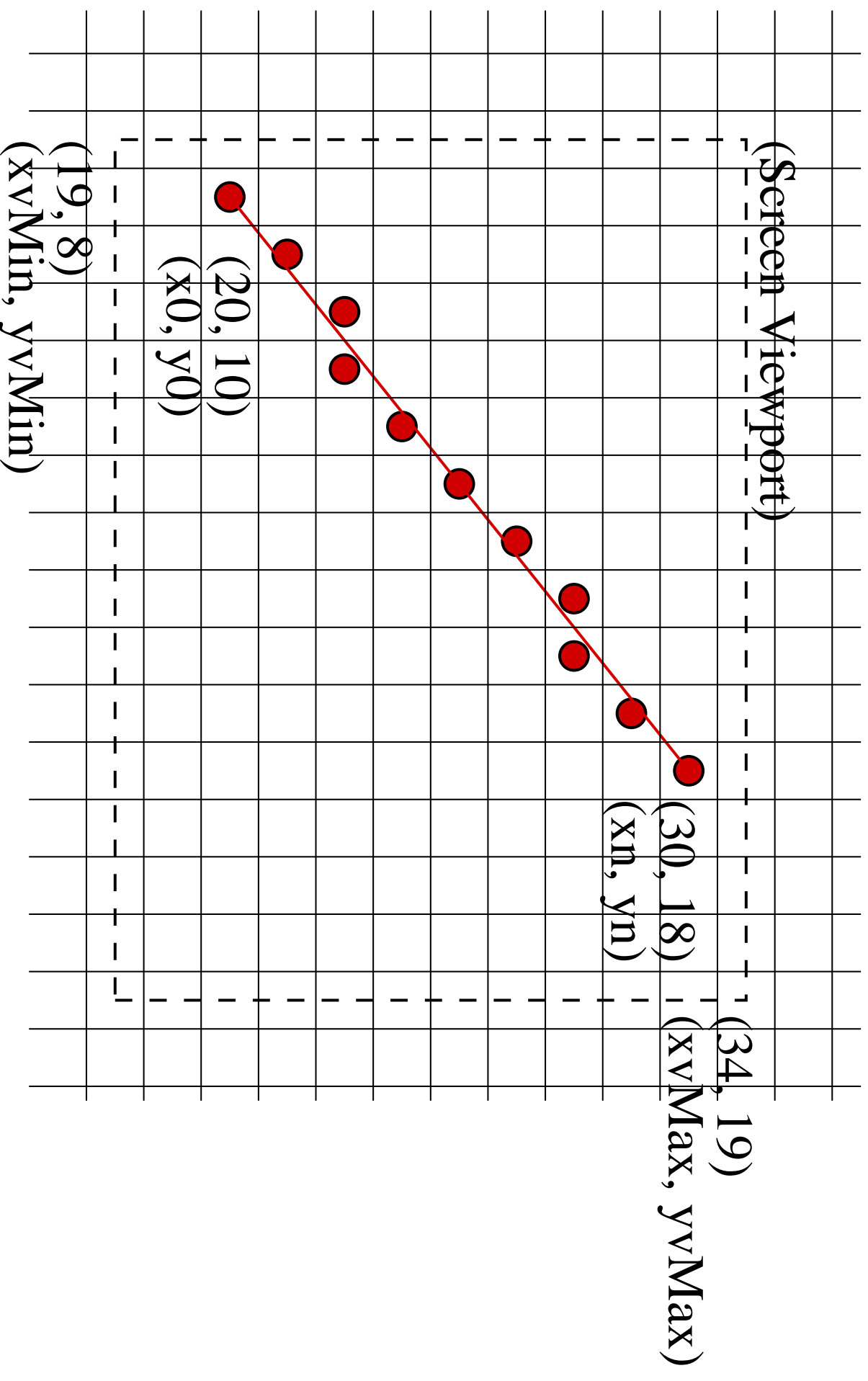
$$xv0 = 19 + (xw0 + 30.0)(0.2)$$

$$xv0 = 19 + ((-25.0) + 30.0)(0.2) = 19 + (5.0)(0.2) = 19 + 1.0 = 20.0$$

$$yv0 = 8 + ((-10) + 20.0)(0.2) = 8 + (10.0)(0.2) = 10.0$$

Line Rasterization Problem: Given pixel coordinates of the endpoints

(x_0, y_0) , (x_n, y_n) of a line, which pixels should be colored in order to draw the line?



Simplified solution (for the $|M| \leq 1$ case):

1. Determine M and B in the line's equation $y = Mx + B$.
2. For $x = x_1, \dots, x_n$, compute $y_i = \text{round}(M \cdot x + B)$.

This *defines* which pixels should be colored, but is a very inefficient way to compute them.

Problem: Rasterize the line from (x_0, y_0) to (x_n, y_n) given these pixel coordinates, solved by **Bresenham's algorithm**:

Details given only for slope between 0 and 1. First make $x_0 < x_n$ (*How??*).

The number of pixels is $x_n - x_0 + 1$.

Differences: $\Delta x = (x_n - x_0)$, $\Delta y = (y_n - y_0)$

Slope: $M = \frac{\Delta y}{\Delta x} = \frac{y_n - y_0}{x_n - x_0}$.

Since M is not an integer (except when $M = 0$ or $+1$) let's use the **multiple** of $(Mx + B) - y$:

$F(x, y) = (2\Delta x)((Mx + B) - y)$.

$F(x, y) = 0$ if and only if (x, y) is *on the line*.

$F(x, y) < 0$ if and only if (x, y) is **ABOVE** the line; *line is BELOW* (x, y) .

$F(x, y) > 0$ if and only if (x, y) is **BELOW** the line; *line is ABOVE* (x, y) .

Algorithm steps and their justification:

1. Store values x_0 and y_0 for future use.
2. Set pixel (x_0, y_0) .
3. Calculate and store for future use: Δx , Δy , $2\Delta y$ and $2\Delta y - 2\Delta x$.

Evaluate and store for future use:

$$F(x_1, y_0 + \frac{1}{2}) = F(x_0 + 1, y_0 + \frac{1}{2})$$

(This helps because the *sign* of this number determines whether the *2nd pixel* is $(x_0 + 1, y_0)$ or $(x_0 + 1, y_0 + 1)$).

$$\begin{aligned} & F(x_0 + 1, y_0 + \frac{1}{2}) \\ &= (2\Delta x)((m(x_0 + 1) + b - (y_0 + \frac{1}{2}))) \\ &= (2\Delta x)((mx_0 + b) + m - (y_0 + \frac{1}{2})) \\ &= (2\Delta x)((y_0) + m - (y_0 + \frac{1}{2})) \\ &= (2\Delta x)(m - \frac{1}{2}) \\ &= (2\Delta x)(\frac{\Delta y}{\Delta x} - \frac{1}{2}) \end{aligned}$$

which we *compute using the simple but equivalent formula*

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

p_0 is called the **initial decision parameter**.

4. At each x_k along the line, starting at $k = 0$, calculate $x_{k+1} = x_k + 1$ and perform the following test and operations conditional on the result:

If $p_k < 0$, then set $y_{k+1} = y_k$ and plot pixel (x_{k+1}, y_{k+1}) and set

$$p_{k+1} = p_k + 2\Delta y. \text{ (We add positive } 2\Delta y \text{ to negative } p_k \text{).}$$

Otherwise, set $y_{k+1} = y_k + 1$ and plot pixel $(x_{k+1}, y_k + 1)$, and set

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x. \text{ (We add a negative value to positive or } 0 \text{ } p_k \text{).}$$

5. Perform step 4. $n - 1 = \Delta x - 1$ more times ($n = \Delta x$ in total).

Justification of step 4: Remember, p_k is the decision parameter for the $(k + 1)$ st pixel, with y coordinate

$$y_{k+1} = \begin{cases} y_k & \text{if } p_k < 0 \\ y_k + 1 & \text{if } p_k \geq 0. \end{cases}$$

The next pixel is at most 1 pixel above the previous because the slope is ≤ 1 .

y_{k+1} affects the decision parameter calculation for the pixel $(k + 2)$:

$$p_{k+1} = F(x_{k+1} + 1, y_{k+1} + \frac{1}{2}) = p_k + \begin{cases} 2\Delta y & \text{if } p_k < 0 \\ 2\Delta y - 2\Delta x & \text{if } p_k \geq 0. \end{cases}$$

You can derive this last formula using algebra with

$$p_k = F(x_{k+1}, y_k + (1/2)).$$

Lazy (smart!) person's way to derive

$$p_{k+1} = F(x_{k+1} + 1, y_{k+1} + \frac{1}{2}) = p_k + \begin{cases} 2\Delta y & \text{if } p_k < 0 \\ 2\Delta y - 2\Delta x & \text{if } p_k \geq 0. \end{cases}$$

Analyze the formulas:

$$F(x, y) = (2\Delta x)((Mx + B) - y) \text{ and } M = \frac{\Delta y}{\Delta x}$$

$$F(x, y) = (2\Delta x)\left(\left(\frac{\Delta y}{\Delta x}\right)x - y + B\right)$$

$$F(x, y) = (2\Delta y)x - (2\Delta x)y + (2\Delta x)B$$

This is a **linear function** of x and y plus a constant.

When x increases by 1, the value of F increases by the x -coefficient which is $(2\Delta y)$

When both x and y each increase by 1, the value of F increases by the x -coefficient *plus* the y -coefficient; a total of $(2\Delta y) - (2\Delta x)$.

Therefore, the case when $p_k < 0$, the F value is for a point obtained by increasing x by 1 but not changing y . To get this new F value (which is p_{k+1} , we therefore add $(2\Delta y)$ to the old value of F , which is p_k .

In the other case, the F value is for the point resulting from increasing x by 1 and increasing y by 1 also. Hence we add $(2\Delta y) - (2\Delta x)$.

We did not have to figure out the constant coefficient $(2\Delta x)B$ to derive Bresenham's rule!

HOMEWORK (to be assigned): Reproduce the calculations of HB fig. 3-12, do another example, set up and demonstrate Bresenham's algorithm for slopes $m > 1$ and for slopes $-1 < m < 0$ (negative slopes).

Such a demonstration may be a quiz problem, and *will* be a midterm problem.

Nowadays, **graphics hardware accelerators** built into graphics cards run Bresenham-like algorithms, so the main CPU doesn't need to. But the mathematical and algorithmic ideas are important. We will apply them to *circle* and other *curve* rasterization.