Administrative

Please do “Apply”--it is needed for CAT card access to graphics lab.

(Console ccess may not yet be available, use Debian boxes if you need a console in the interim; remote access should work: graphics01 to graphics11 @cs.arizona.edu

I need your E-mail--check it on the list; if you are not on the list because your paperwork has not yet perculated through the system, add your name and E-mail at the bottom of the list.

Administrative

Unix versus windows: Graphics machines are now all Red Hat. If you want to reboot in windows, use the next available one with the highest number--i.e., make “graphics11” windows before “graphics10”. If “graphics11” is being used, then use “graphics10”, etc.

If you develop on windows, you must check that your code compiles and runs on linux.

Check the class page regularly for announcements. (http://www.cs.arizona.edu/classes/cs433/fall02/index.html)

Assignment

Revised versions will be put online as the English and the specifications get debugged.

I have reworded the instructions on how bright to make the line. It now says:

> Lines should be drawn so that their brightness is normalized to (200, 200, 200) times the output of the weighting filter (this will be explained further in class).

Assignment (cont)

Also, I have added:

> Note: A non-aliased, or differently aliased line is better than no line at all. If you are unable to achieve the cone filter result, then drawing the line some other way will earn partial credit (note the method used in the README file).

Assignment (cont)

Also, I have changed one hint to:

> Modest inefficiencies are OK, especially in the anti-aliasing code, but you should not compute the brightness of pixels which are obviously too far from the line to be anything other than black.

Assignment (cont)

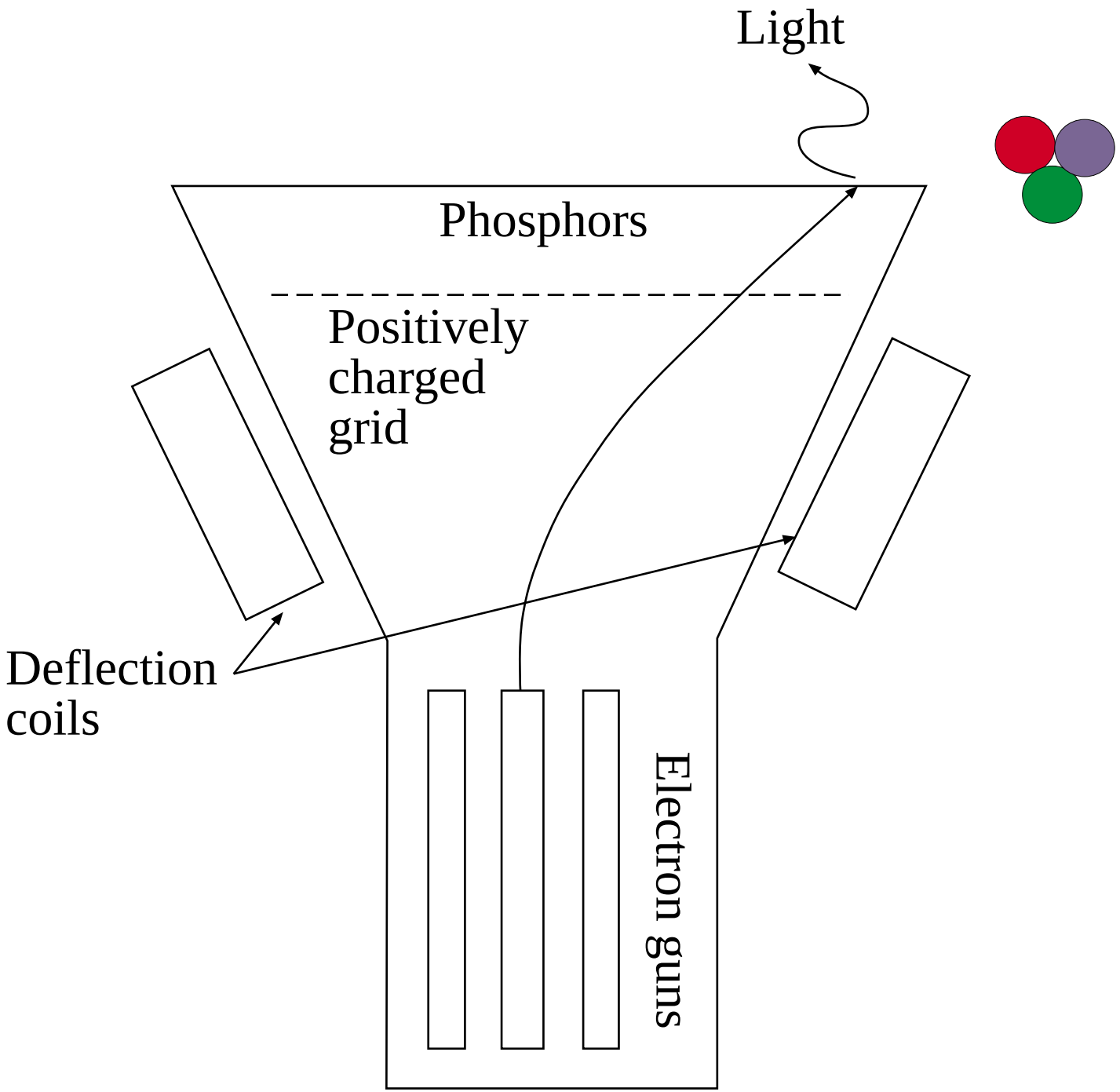
Issues

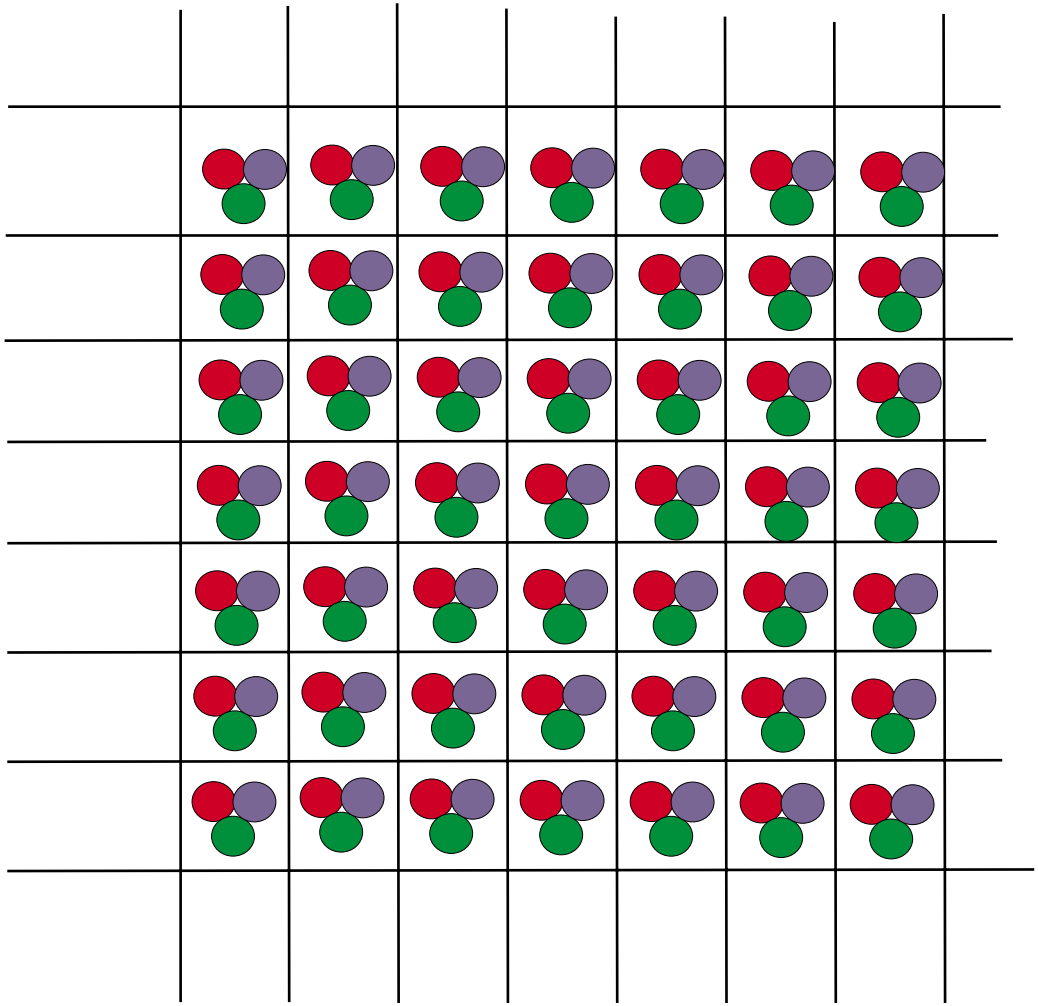
Credit: 12 points (Relative, and roughly absolute weighting)

Window resizing

The grid

Displays


Light
Phosphors
Positively
charged
grid
Deflection
coils
Electron guns

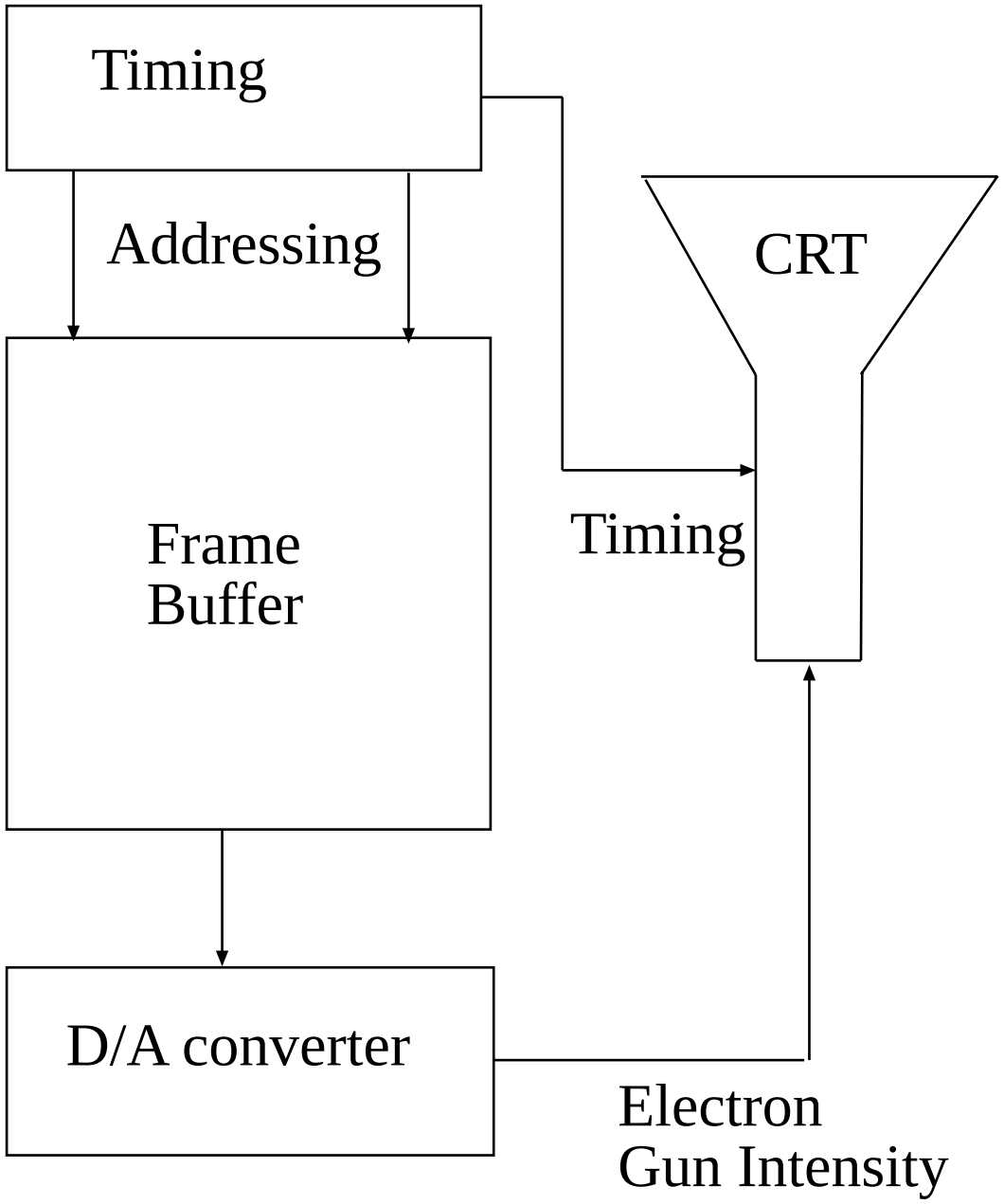


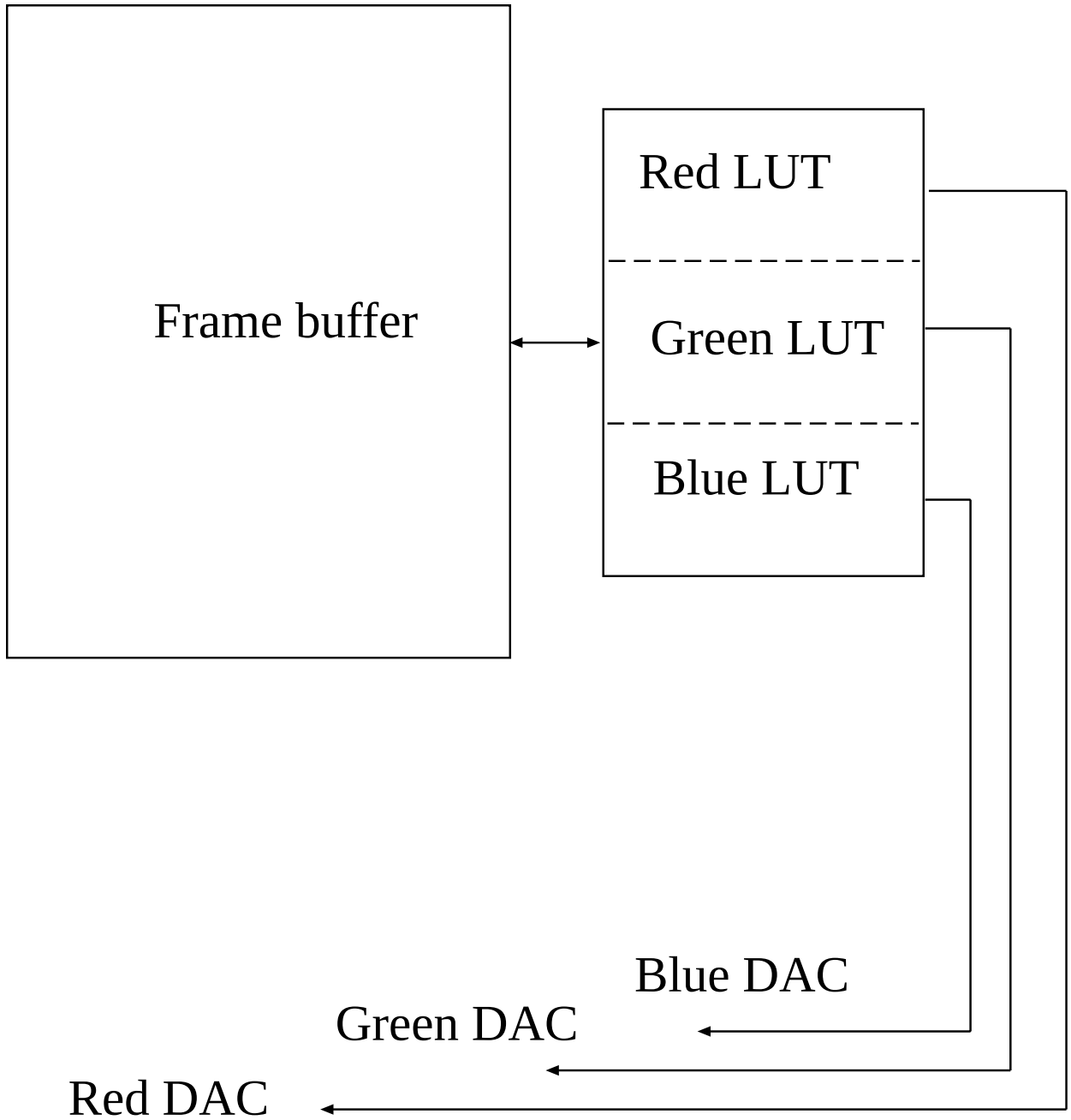
CRT Displays

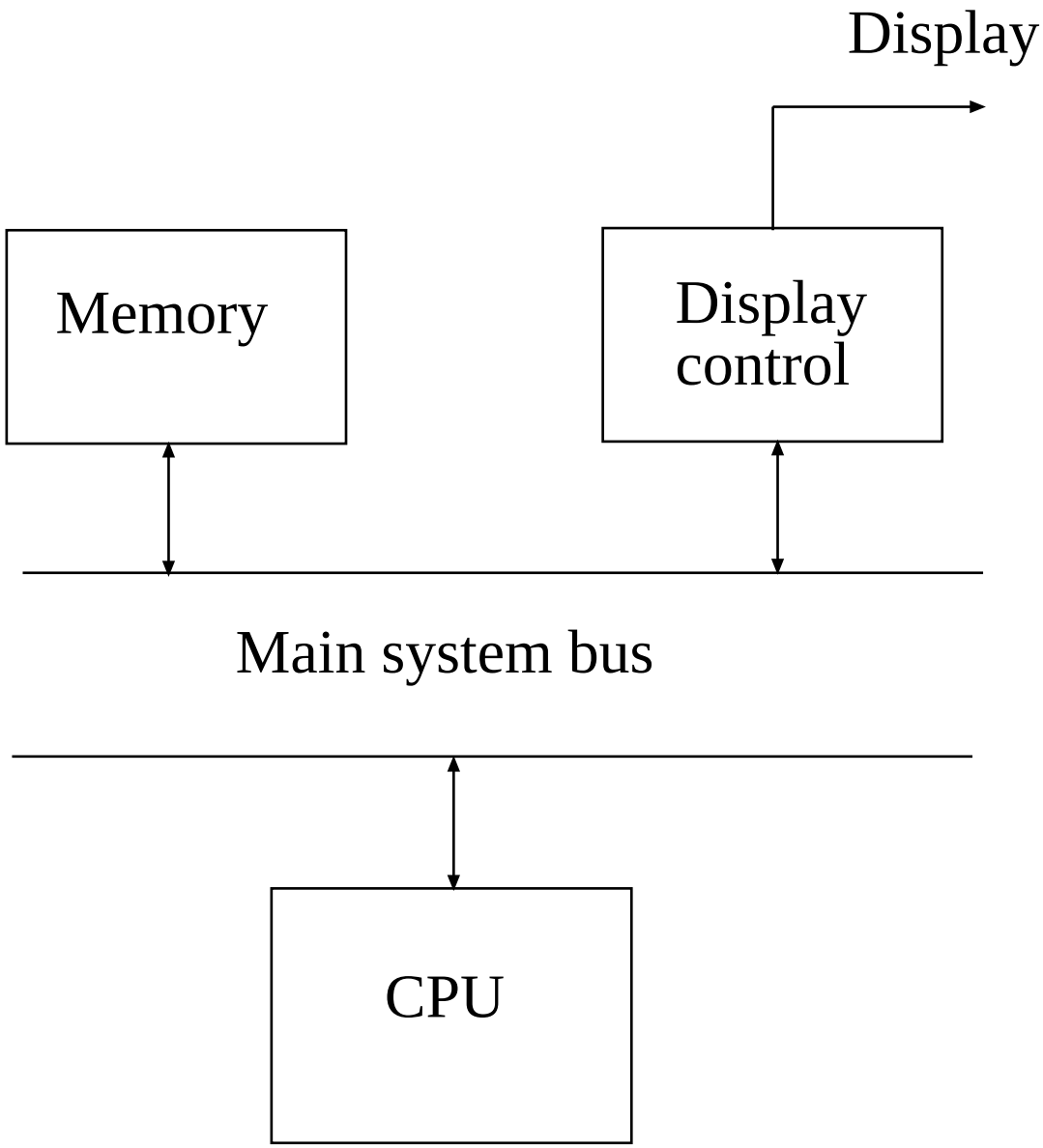
- Phosphurs glow when hit by electron beem.
- Color is adjusted via intensity of beam delivered to each of R,G, and B phosphor
- CRT display phosphors glow for limited time--need to be refreshed
- Raster displays refresh by scanning from top to bottom in left right order.
- Use timing to construct a correspondence between memory elements and screen elements.

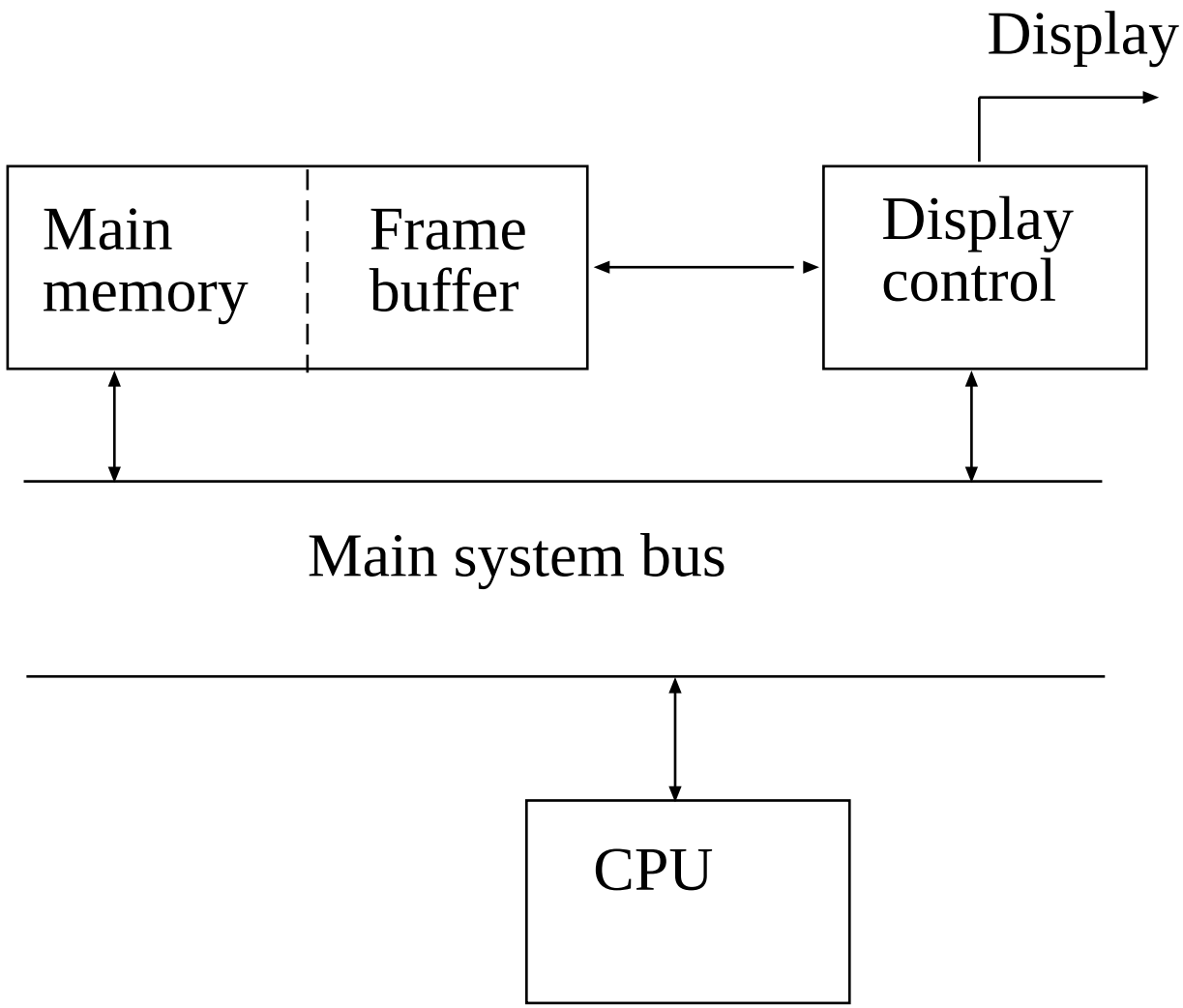
CRT Displays

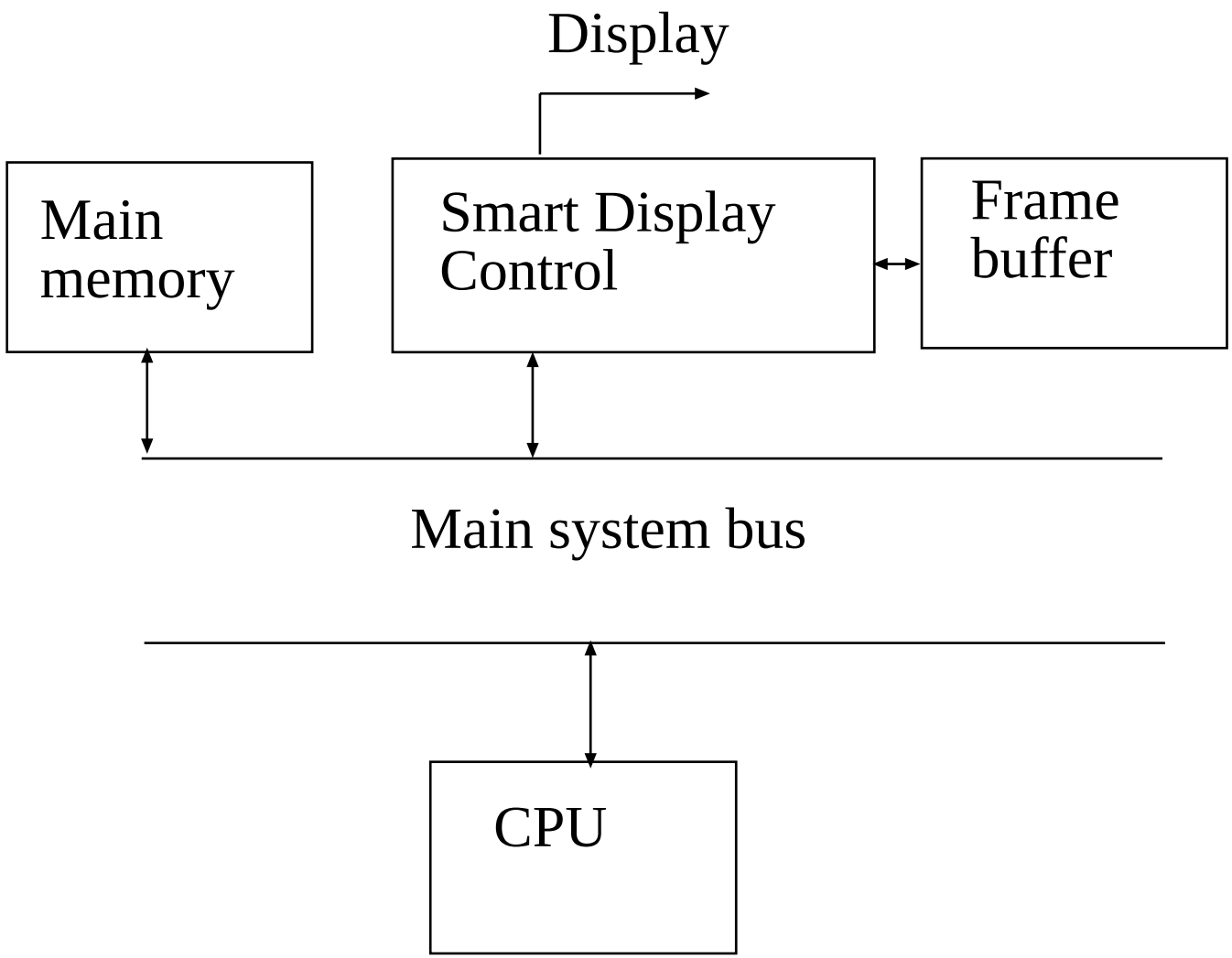
- Typical numbers:
  - 1280 by 1024 points
  - 75 refreshes per second
- May have many phosphor dots corresponding to one memory element (but generally one per phosphor trio).
- Memory elements called pixels
- Refresh method creates architectural and *programming* issues (e.g. double buffering), defines “real time” in animation.


Timing
Addressing
CRT
Frame
Buffer
Timing
D/A converter
Electron
Gun Intensity

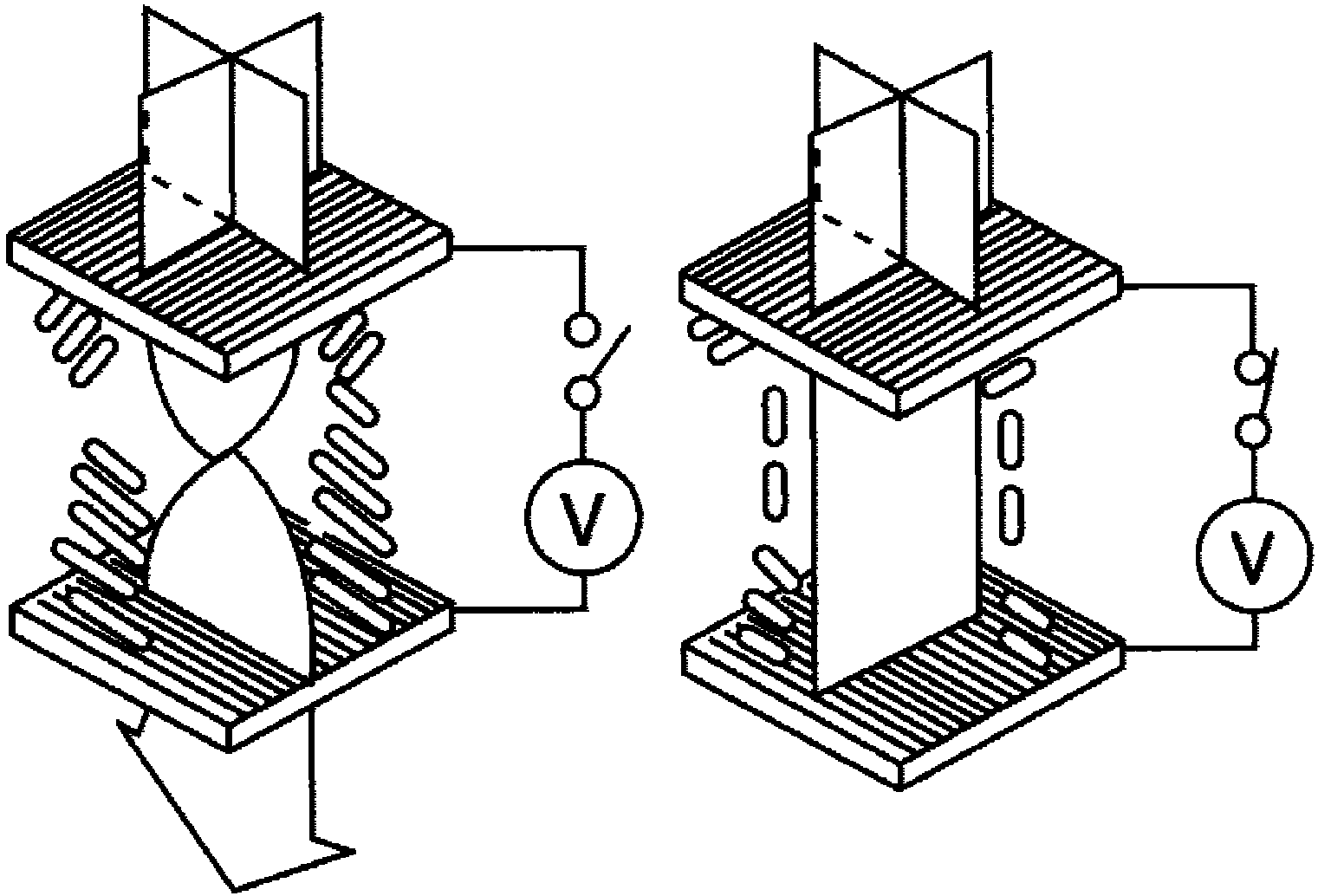

Frame buffer
Red LUT
Green LUT
Blue LUT
Blue DAC
Green DAC
Red DAC


Display
Memory
Display
control
Main system bus
CPU

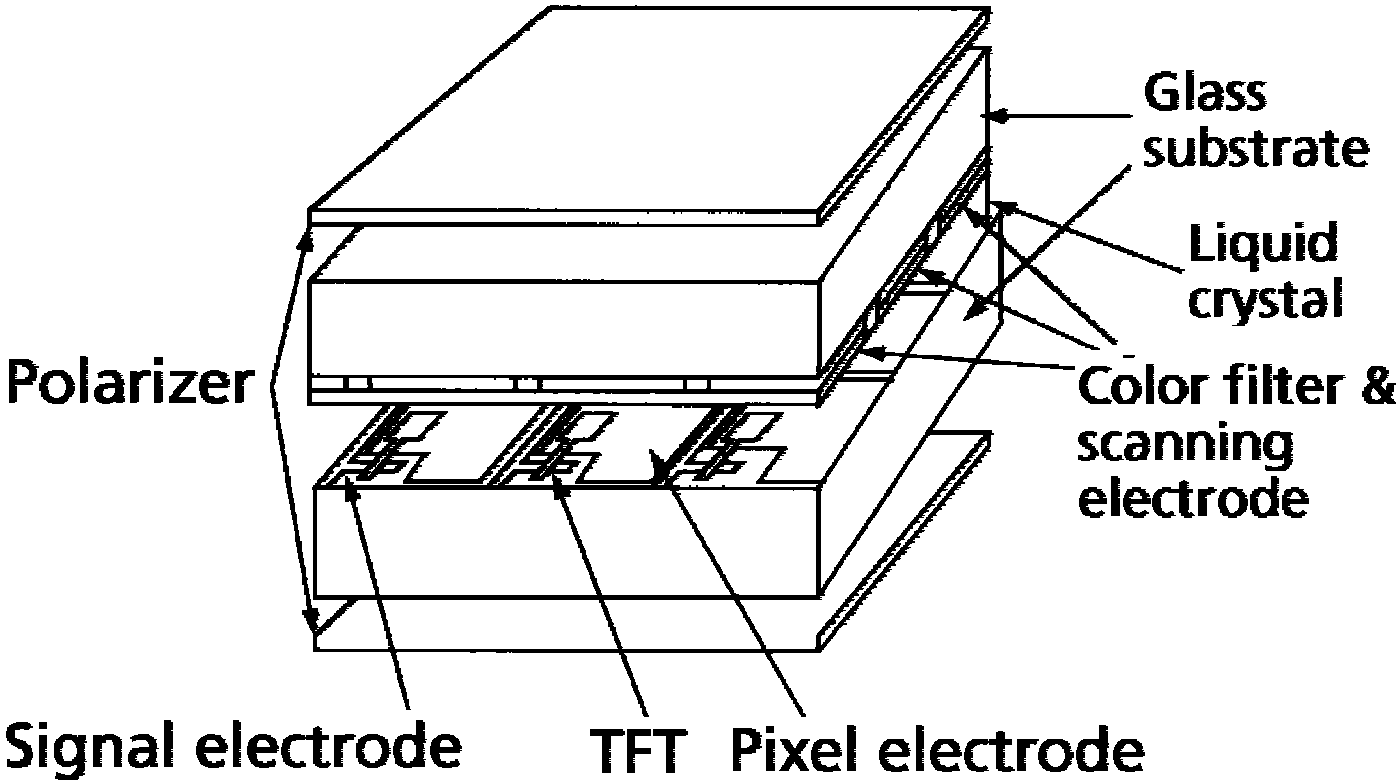

Display
Main memory
Frame buffer
Display control
Main system bus
CPU


Display
Main memory
Smart Display Control
Frame buffer
Main system bus
CPU

Flat Panel LCD TFT Displays



From http://www.atip.or.jp/fpd/src/tutorial



From http://www.atip.or.jp/fpd/src/tutorial

Other devices of interest

- Input devices:
  - mouse, trackpad, trackball,
  - joy-stick
  - keyboard
  - tablet and pen
  - 2D and 3D scanners
- Printers:
  - inkjet
  - laser
  - dye sublimation
  - pen plotters
  - dot-matrix
- Displays:
  - Flat panel
    - plasma
    - lcd
  - 3D - CRT displaying alternate frames to left and right eyes
  - Head mounted

Displaying lines

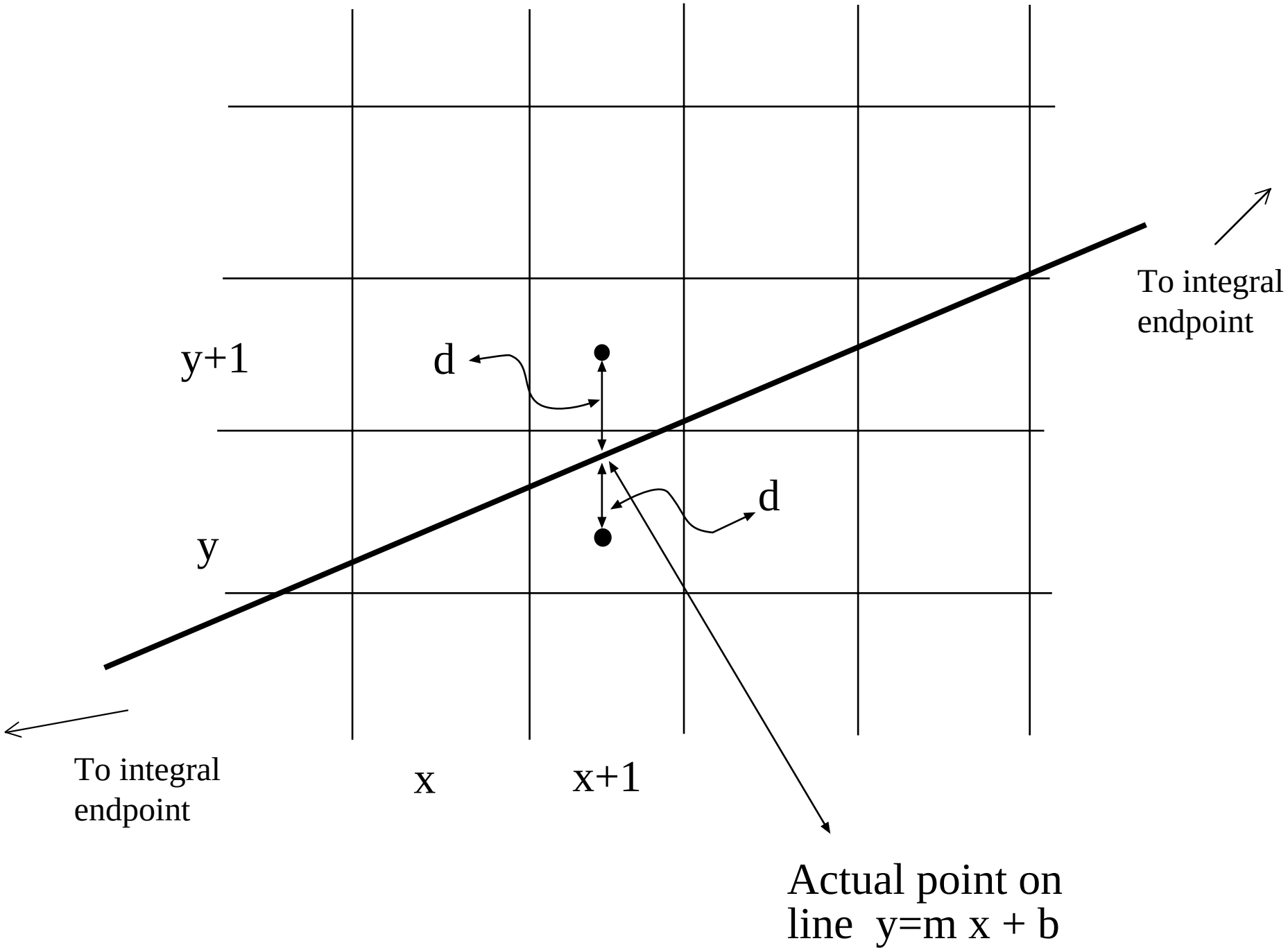
- Assume:
  - lines have integer vertices
  - lines all lie within the displayable region of the frame buffer
- Other algorithms will take care of these issues.
- Consider lines of the form $y=m\,x + c$, where $0<m<1$
- All others follow by symmetry

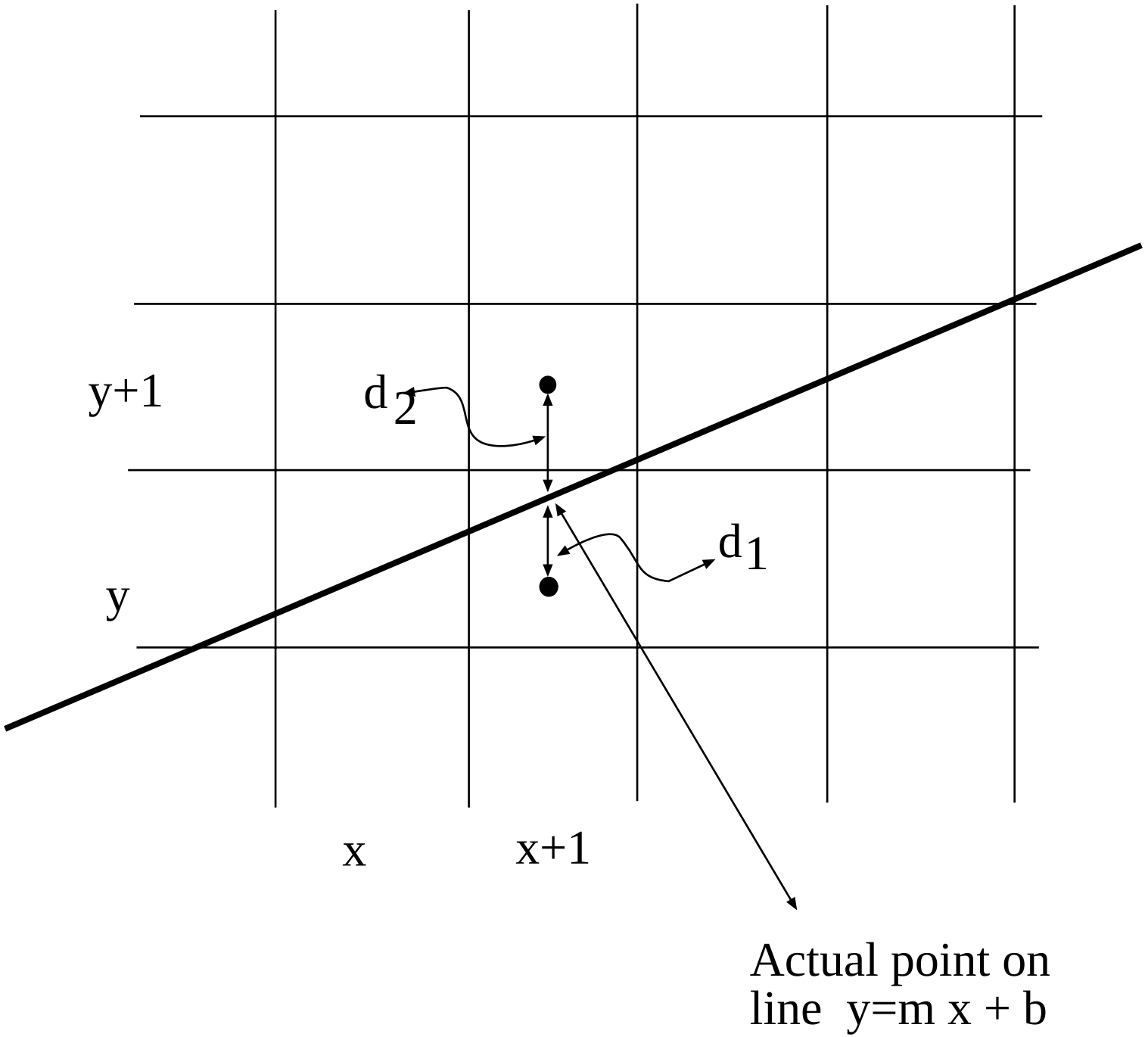
Displaying lines

- Variety of naive (poor) algorithms:
  - step x, compute new y at each step by equation, rounding
  - step x, compute new y at each step by adding m to old y, rounding

Bresenham's algorithm

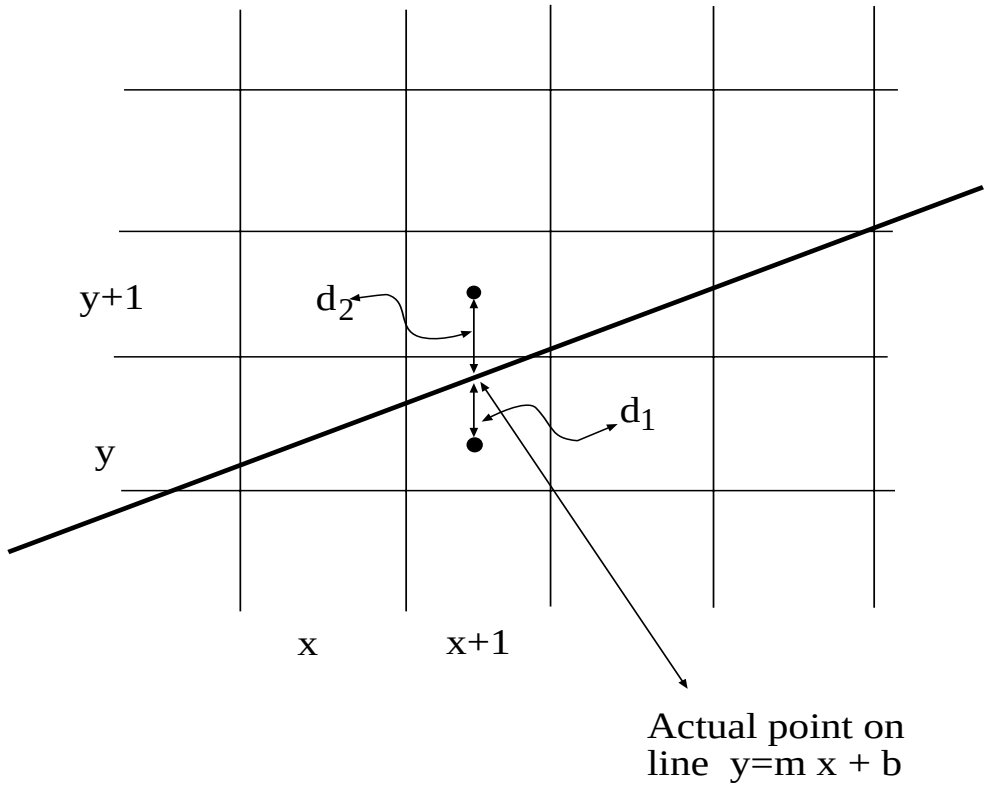
- see text section 3.2.2--midpoint algorithm is Bresnham in simple line case--but note that the text draws the pixels on the gridpoints
- plot the pixel whose y-value is closest to the line
- given (x_k, y_k), must choose from either (x_k+1, y_k+1) or (x_k+1, y_k)
- idea: compute value that will determine this choice, and is easy to update and cheap to compute (no floating point operations if endpoints are integral).


To integral endpoint
y+1
d
d
y
To integral endpoint
x
x+1
Actual point on line y=m x + b


y+1
d2
d1
y
x
x+1
Actual point on
line y=m x + b

Bresenham's algorithm

- (slightly different treatment from book)
- determiner is $d_1 - d_2$
  - $d_1 < d_2$ => plot at y_k
  - $d_1 > d_2$ => plot at y_k+1



$$d_1 - d_2 = (y - y_k) - ((y_k + 1) - y)$$
$$= 2m(x_k + 1) - 2y_k + 2b - 1$$

Recall,

$$m = (y_{end} - y_{start})/(x_{end} - x_{start}) = dy/dx$$

So, for integral endpoints we can avoid floating point if we scale by a factor of dx. Use determiner P_k.

$$p_k = 2(x_k + 1)dy - 2y_k(dx) + 2b(dx) - dx$$
$$= 2(x_k)dy - 2y_k(dx) + \text{constant}$$

Finally, express the next determiner in terms of the previous,

$$\begin{aligned} p_{k+1} &= 2(x_k + 1)dy - 2y_{k+1}(dx) + \text{constant} \\ &= p_k + 2dy - 2(y_{k+1} - y_k) \end{aligned}$$

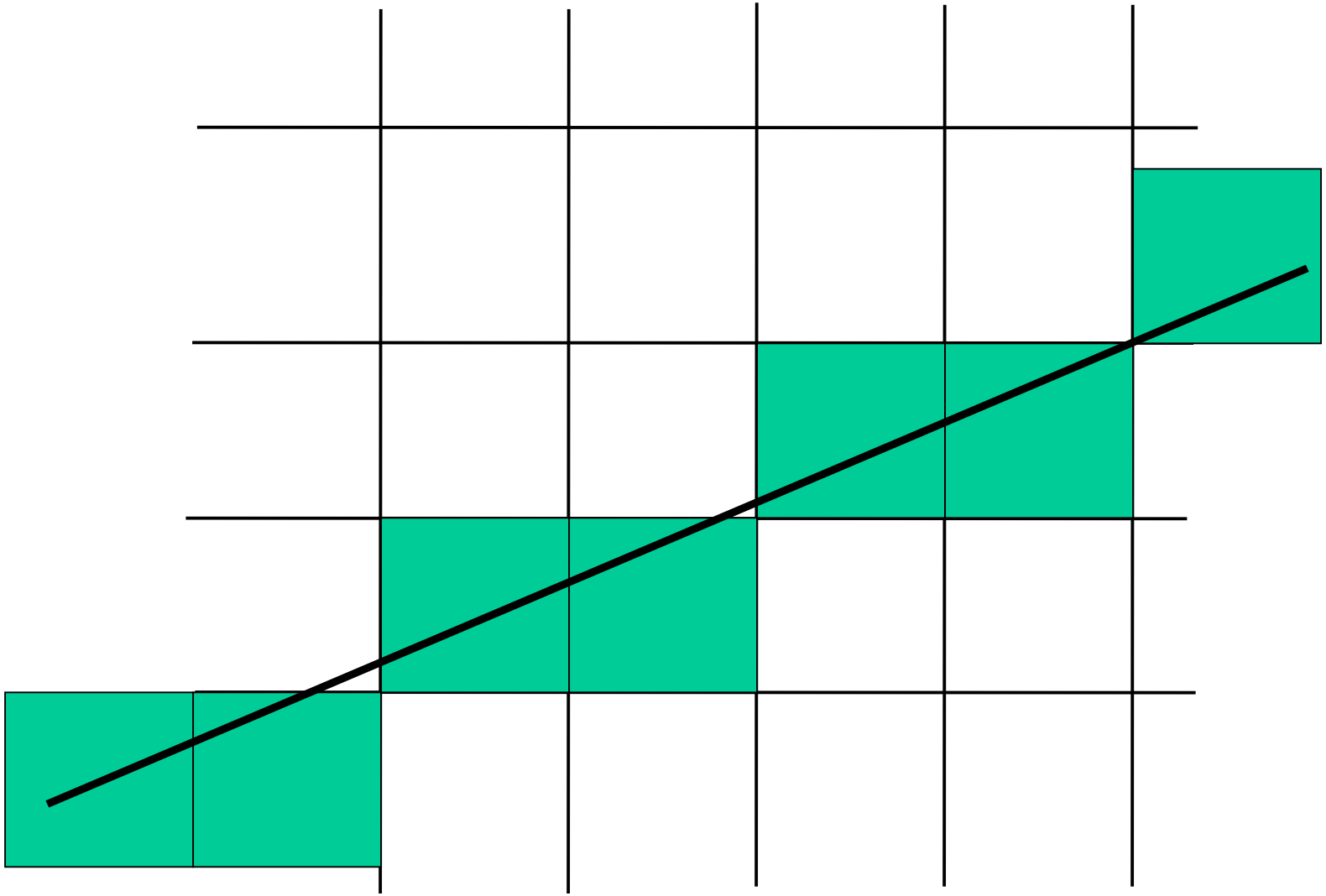
Bresenham (continued)

- $p_{k+1} = p_k + 2\,dy - 2\,dx\,(y_{k+1}-y_k)$
- Exercise: check that $p_0 = 2\,dy - dx$
- Algorithm (for $0<m<1$):
  - x=x_start, y=y_start, p=2 dy - dx, mark (x, y)
  - until x=x+end
    - x=x+1
    - p>0 ? y=y+1, mark (x, y), p=p+2 dy - 2 dx
    - p<0? y=y, mark (x, y), p=p+2 dy
- Some calculations can be done once and cached.

Issues

- End points may not be integral due to clipping (see book, pp. 78-79)
- Brightness is a function of slope (see Figure 3.9, page79).
- Aliasing (related to previous point).

Line drawing--aliasing



Aliasing

- Sampling problem--we are using discrete binary squares to represent perfect mathematical entities
- Points and lines as discussed so far have no width--does this make them invisible?
- Solutions?

Aliasing (cont)

- General approach to reducing aliasing is to exploit ability to draw levels of gray between black and white.
- Example--give the line some width; brightness is proportional to area that pixel shares with line
- We will take a sampling approach.