Preview for next few lectures

Defer next math topic (transformations) for a few lectures

Quick overview of display technology (The creation of a dot).

Drawing lines

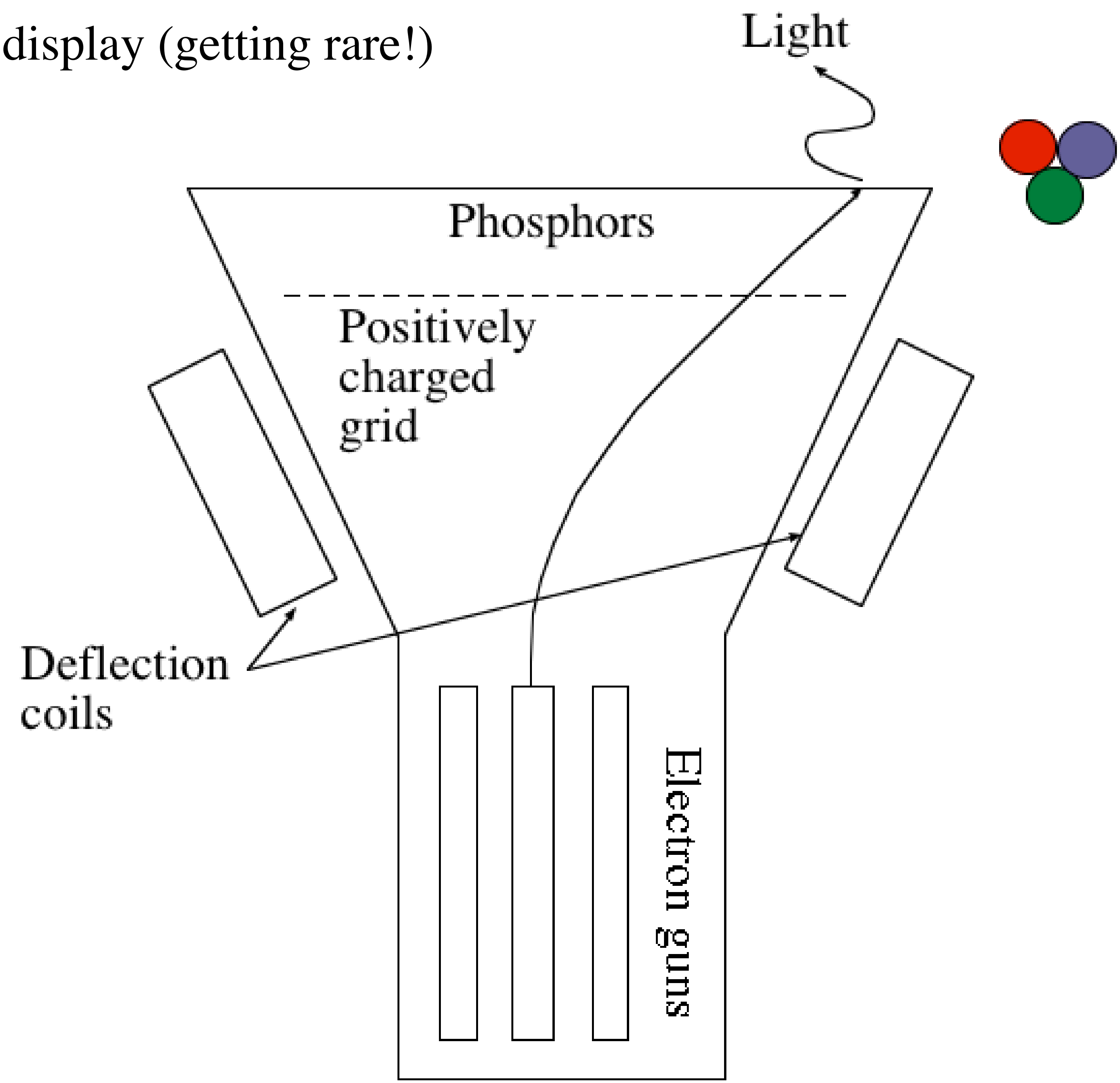
Aliasing

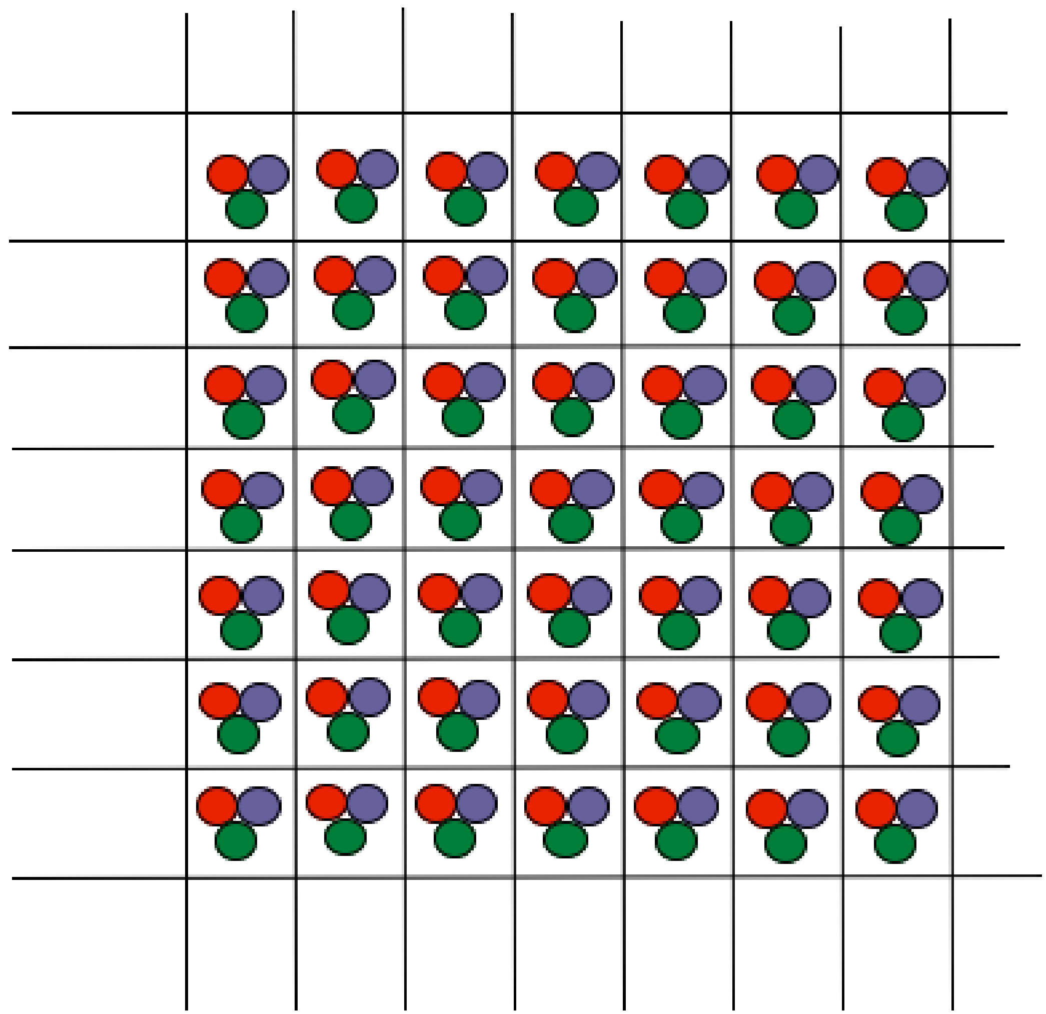
Drawing polygons

Clipping

2D Transformations

CRT display (getting rare!)



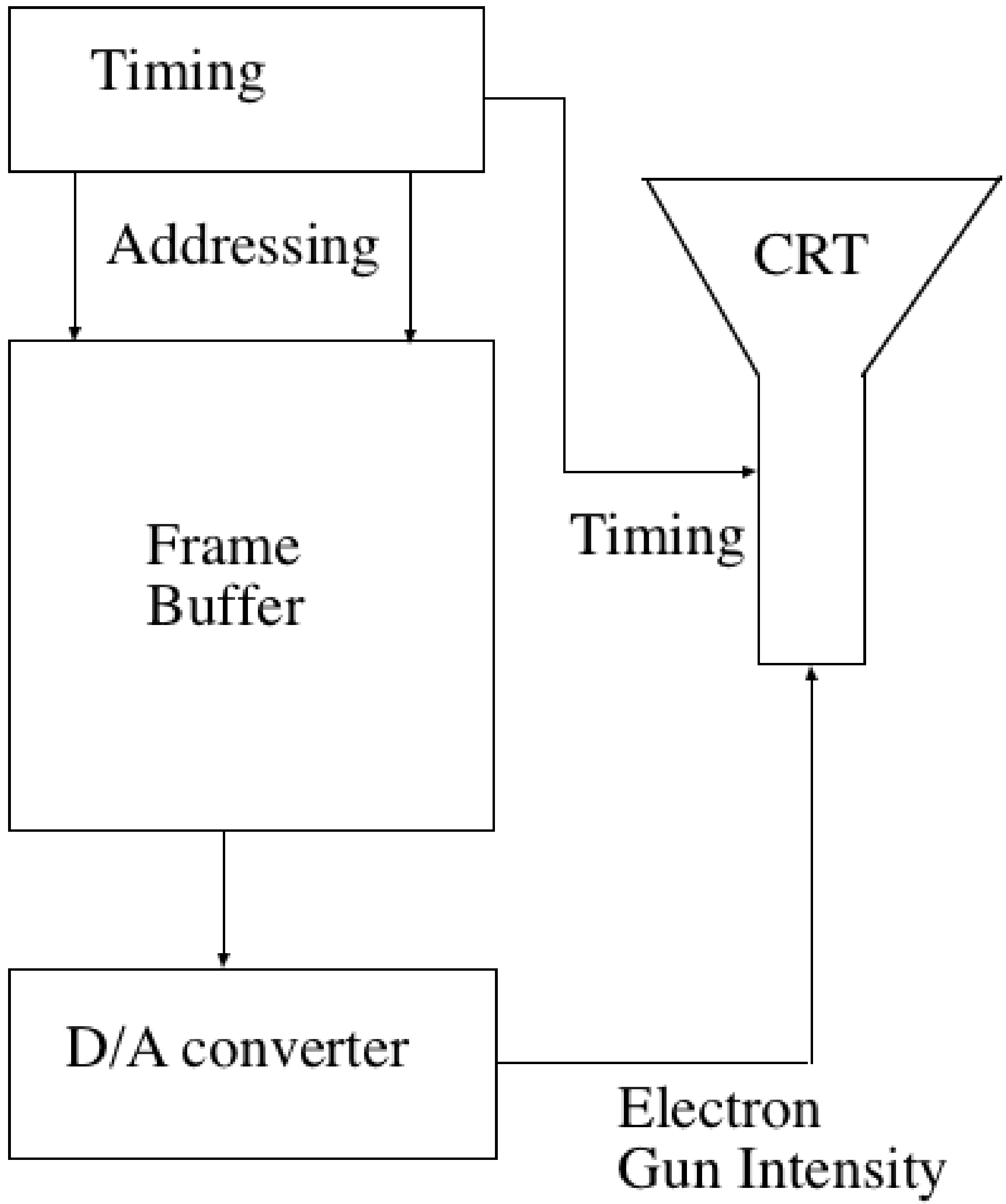


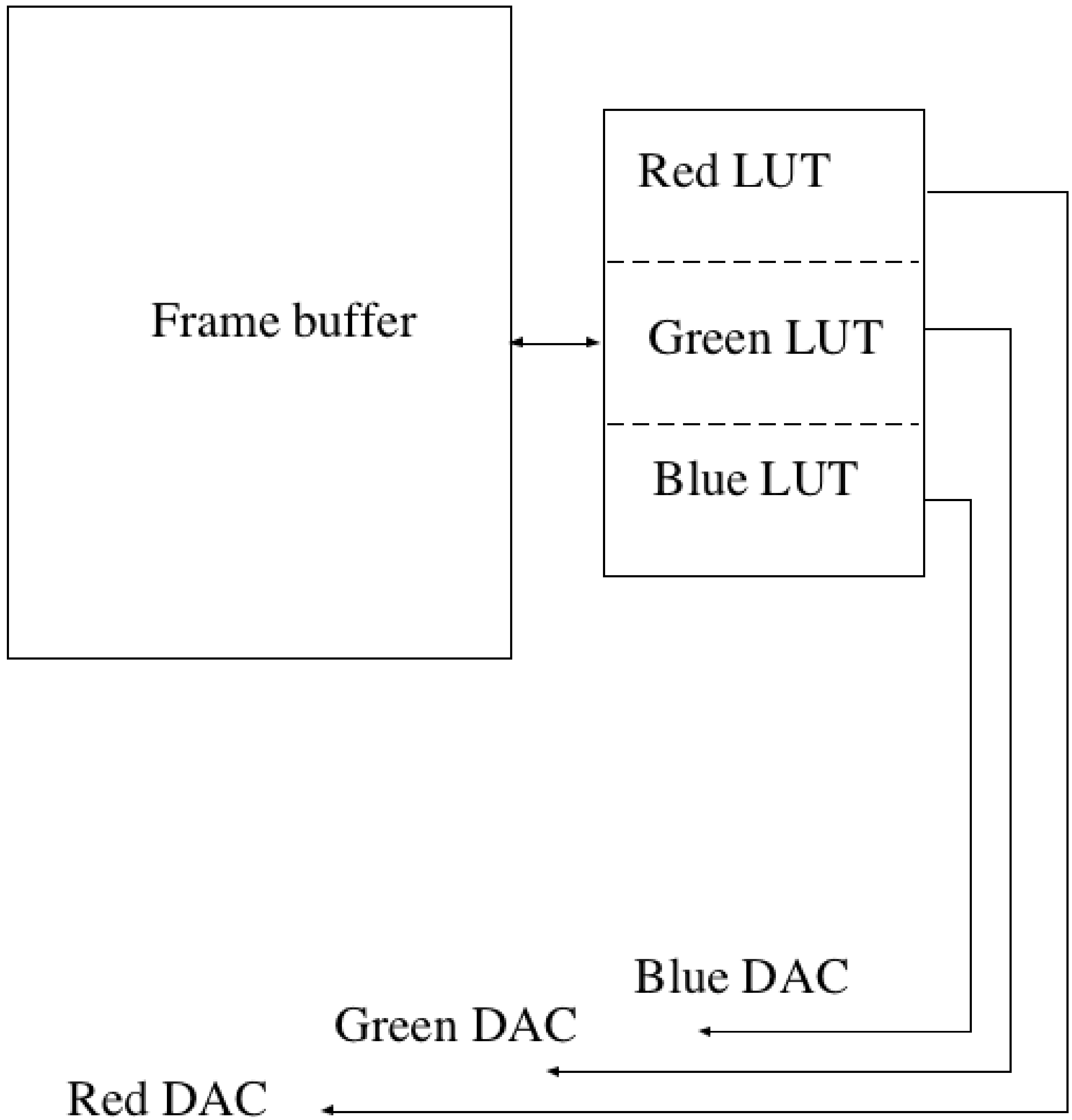
CRT Displays

- Phosphors glow when hit by electron beam.
- 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.
- Timing is used to make screen elements correspond to memory elements.

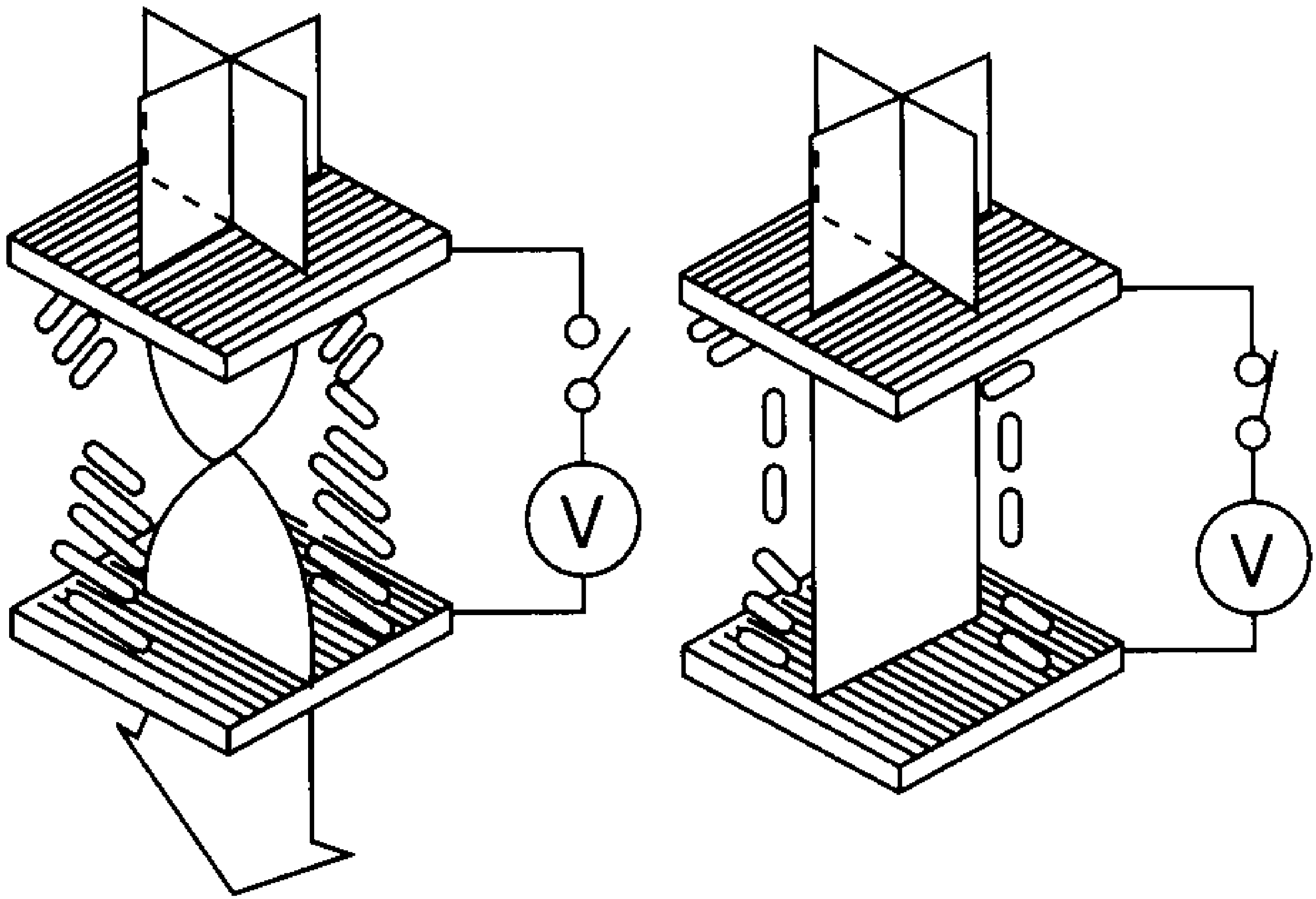
CRT Displays

- Typical refresh rate is 75 per second
- May have many phosphor dots corresponding to one memory element (old stuff), but more usually one per phosphor trio.
- Memory elements called pixels
- Refresh method creates architectural and *programming* issues (e.g. double buffering), defines "real time" in animation.

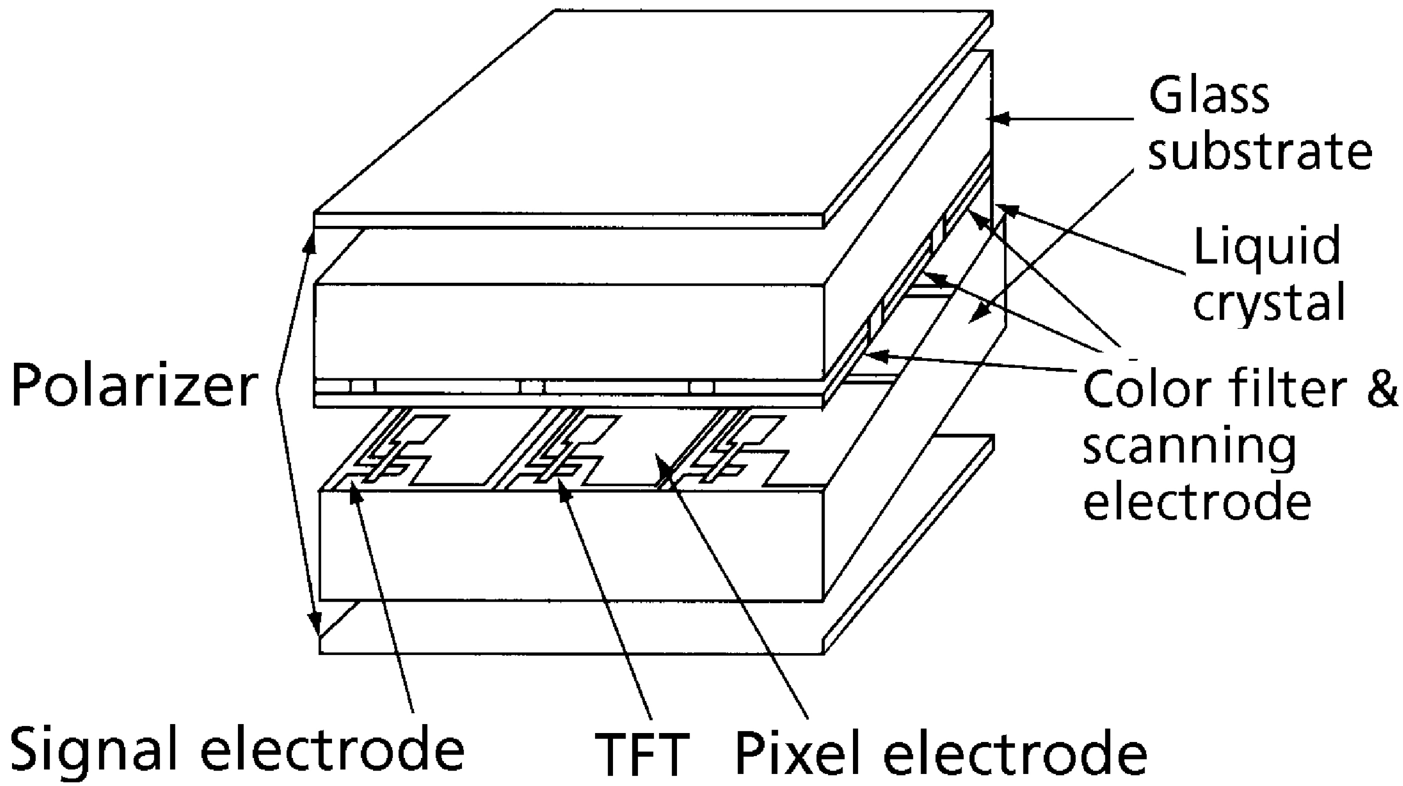

Timing
Addressing
CRT
Timing
Frame
Buffer
D/A converter
Electron
Gun Intensity


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

Flat Panel TFT Displays



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



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

3D displays

Use some scheme to control what each eye sees

Color, temporal + shutter glasses, polarization + glasses

Displaying lines

- Assume for now:
  - lines have integer vertices
  - lines all lie within the displayable region of the frame buffer
- Other algorithms will take care of these issues.

Displaying lines

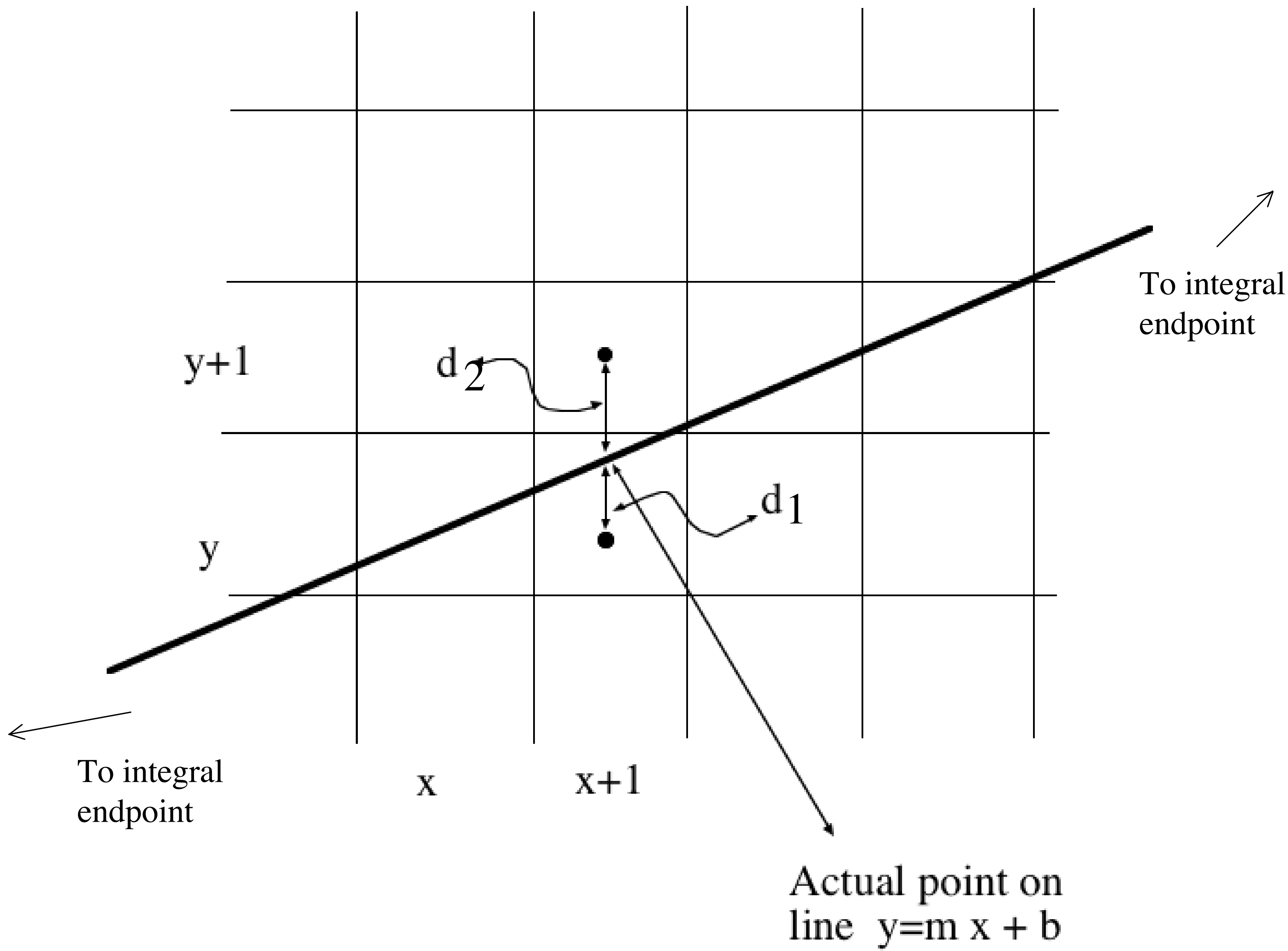
- Assume for now:
  - lines have integer vertices
  - lines all lie within the displayable region of the frame buffer
- Other algorithms will take care of these issues.
- Consider lines of the form $y=m\,x + c$, where $0<m<1$
- Other cases follow by symmetry

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

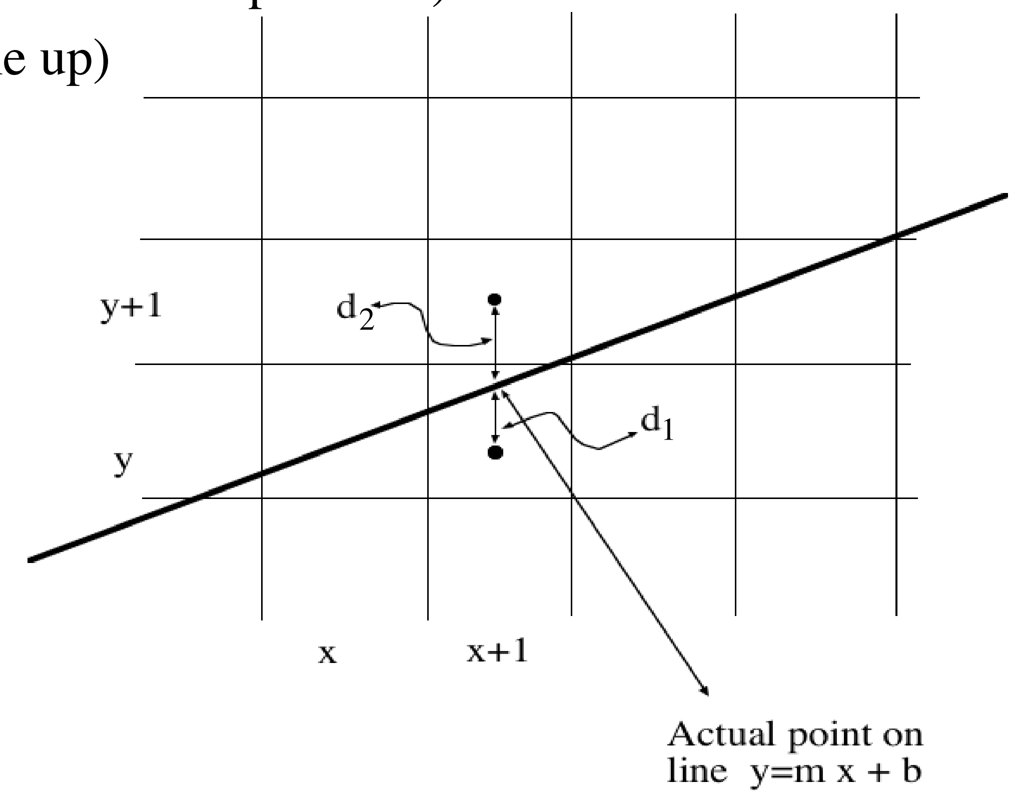
- Plot the pixel whose y-value is closest to the line
- Given (x_k, y_k), must **choose** from either (x_k+1, y_k+1) or (x_k+1, y_k)---recall we are working on case $0<m<1$
- Idea: compute value that will determine this choice that is easy to update and cheap to compute (no floating point operations if endpoints are integral).



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

Bresenham's algorithm

- Determiner is $d_1 - d_2$
  - $d_1 < d_2$ => plot at y_k (same level as previous)
  - $d_1 > d_2$ => plot at y_k+1 (one up)



(Current point is, (x_k, y_k) line goes through $(x_k + 1, y)$)

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

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.

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

From previous slide

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

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

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

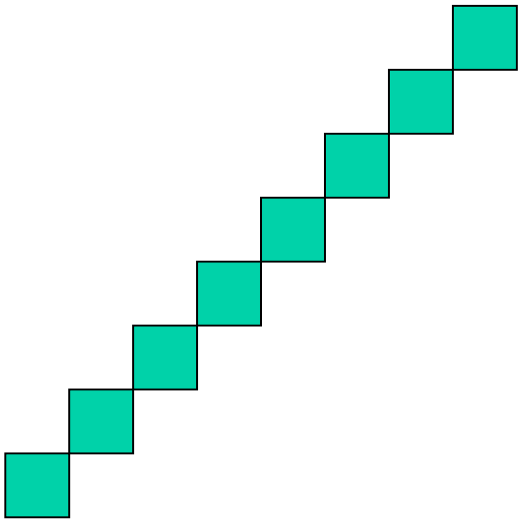
Bresenham (continued)

- $p_{k+1} = p_k + 2\,dy - 2\,dx\,(y_{k+1} - y_k)$
- 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.

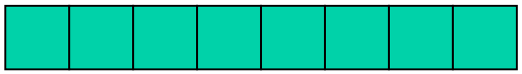
Issues

- End points may not be integral due to clipping
- Brightness is a function of slope.
- Aliasing (related to previous point).

Line drawing--simple line (Bresenham) brightness issues

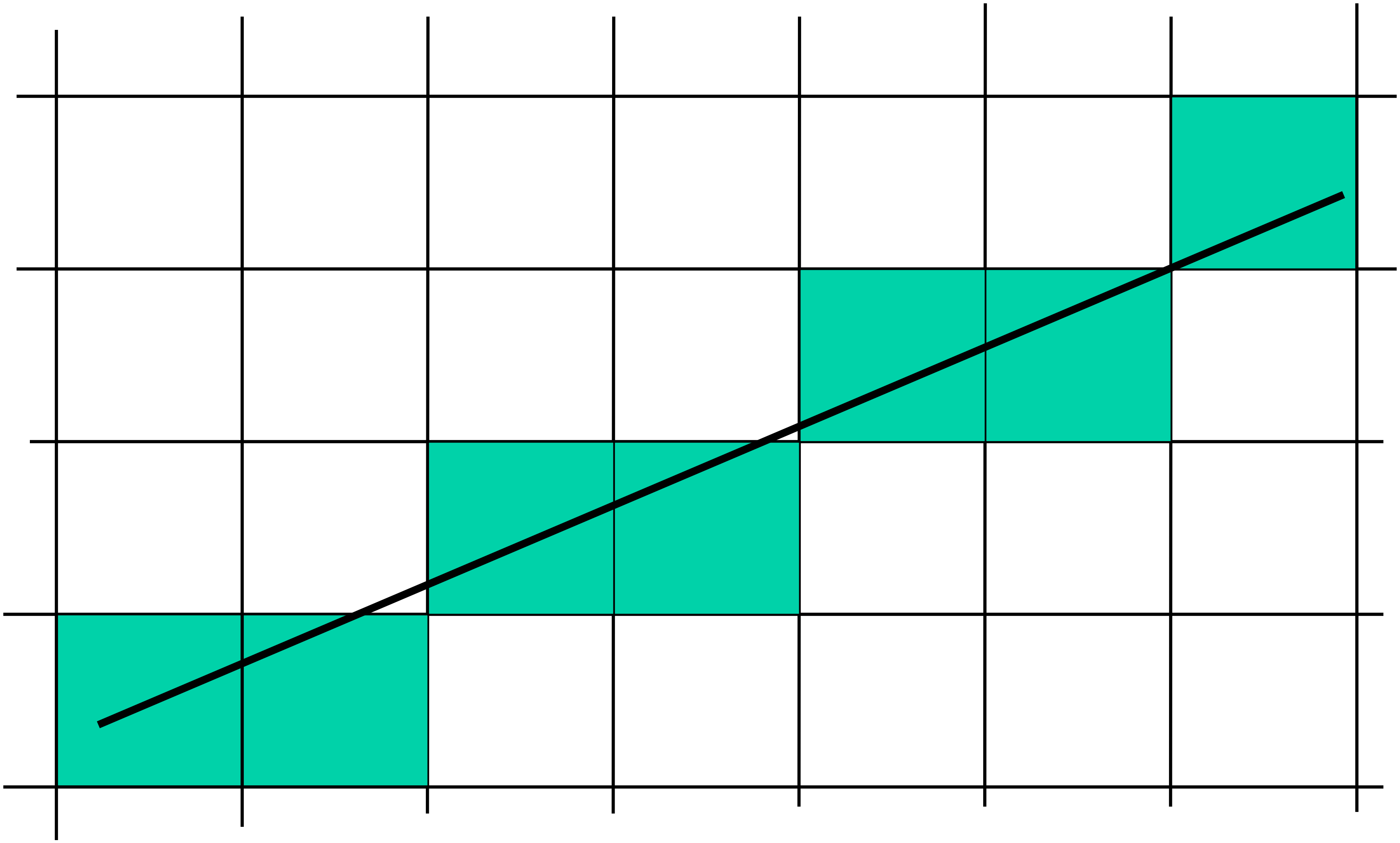


8 pixels per 8*sqrt(2) length



8 pixels for 8 length
(Brighter)

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.

Aliasing via sampling

- Smooth (convolve) the object to be drawn with a filter for each pixel
- This blurs the object, widens the area it occupies
- Now we “sample” the blurred image--i.e., report the value of the blurred function at the (x,y) of interest, and then fill the square with that brightness.