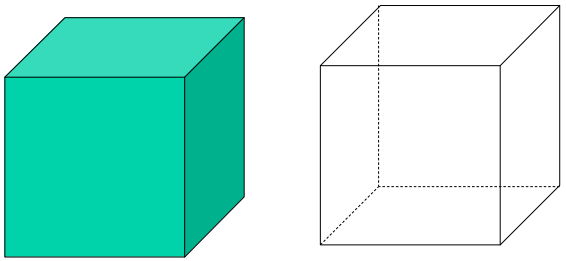
## Homework One Issues

- Making movies
  - The intention of assignment one is that the "movie" capability only exists while reading the input file.
  - You can implement this by calling the drawing callback yourself.
  - After EOF, then enter event loop via
    glutMainLoop();
  - If you want to do animation after glutMainLoop() has been called (maybe useful for A2 extensions), have a look at:
    glutIdleFunc();
    glutTimerFunc();
- Other issues?

## Drawing in 2D*



*That's all there really is!

## Displaying lines

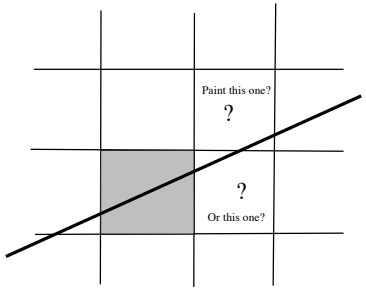
- Assume for now:
  - lines have integer vertices
  - lines all lie within the displayable region of the frame buffer
- Other algorithms will take care of these issues.
- Consider lines of the form y=m x + c, where 0<m<1
- Other cases follow by symmetry
- (Boundary cases, e.g. m=0, m=1 also work in what follows, but are often considered separately, because they can be done very quickly as special cases).

## 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
- What if we don't assume m<1?
  - our lines can have holes in them!

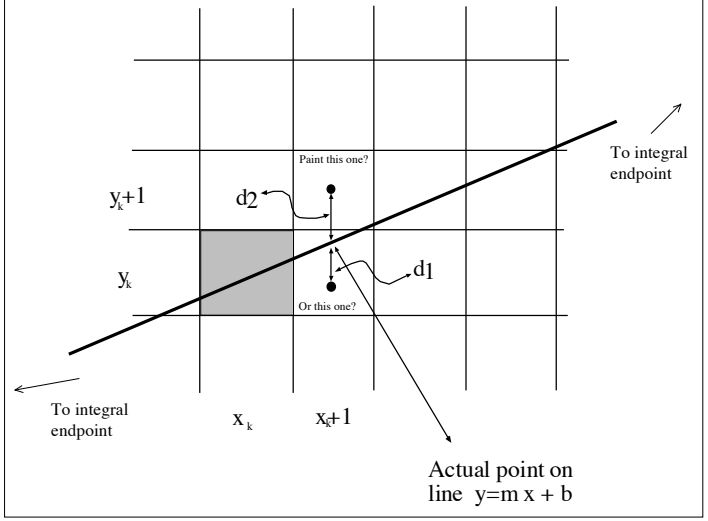
## Bresenham's algorithm [ H&B, pp 95-99]

- Plot the pixel whose y-value is closest to the line




## Bresenham's algorithm [ H&B, pp 95-99]

- 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 $0<m<1$
- We can derive a "decision parameter" for this choice that is easy to update and cheap to compute (no floating point operations if endpoints are integral).
- "decision parameter" == "determiner"

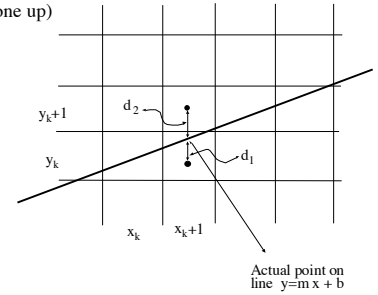



## Bresenham's algorithm

- Decision parameter is $d_1 - d_2$

  $d_1 - d_2 < 0$ => plot at $y_k$ (same level as previous)

  otherwise => plot at $y_k+1$ (one up)



Current integral point is $(x_k, y_k)$

Line goes through $(x_k + 1, y)$

Is $y$ closer to $y_k$ or $y_{k+1}$ ?

$$d_1 = y - y_k \quad \textit{and} \quad d_2 = (y_k + 1) - y$$

So $d_1 - d_2 = (y - y_k) - ((y_k + 1) - y)$

Plugging in $y = m(x_k + 1) + b$

Gives:

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

## Avoiding Floating Point

From the previous slide

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

Recall that,

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

So, for integral endpoints we can avoid division (and floating point ops) if we scale by a factor of dx. Use determiner $P_k$.

$$\begin{aligned} p_k &= (d_1 - d_2)dx \\ &= (2m(x_k + 1) - 2y_k + 2b - 1)dx \\ &= 2(x_k + 1)dy - 2y_k(dx) + 2b(dx) - dx \\ &= 2(x_k)dy - 2y_k(dx) + 2(dy) + 2b(dx) - dx \\ &= 2(x_k)dy - 2y_k(dx) + \text{constant} \end{aligned}$$

(No division)

## Incremental Update

From previous slide

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

Express the next determiner in terms of the previous.

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

Either 1 or 0 depending on decision on y

## Bresenham algorithm

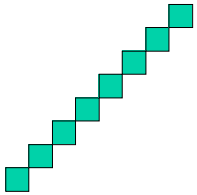
- From previous slide, $p_{k+1} = p_k + 2\,dy - 2\,dx\,(y_{k+1} - y_k)$
- Exercise*: check that $p_0 = 2\,dy - dx$
- Algorithm (for the case that 0<m<1):
  - x=x_start, y=y_start, p=2 dy - dx, **mark** (x, y)
  - until x=x_end
    - x=x+1
    - p>0 ? y=y+1, **mark** (x, y), p=p+2 dy - 2 dx
    - else y=y, **mark** (x, y), p=p+2 dy

*Hint: For $p_0$, $(x_k, y_k)$ is on the line. Use formula for the line and plug into expression for $p_k$ from two slides back: $p_k = (2m(x_k + 1) - 2y_k + 2b - 1)dx$

## Issues

- End points may not be integral due to clipping (or other reasons)
- Brightness is a function of slope.
- Discretization problems “aliasing” (related to previous point).

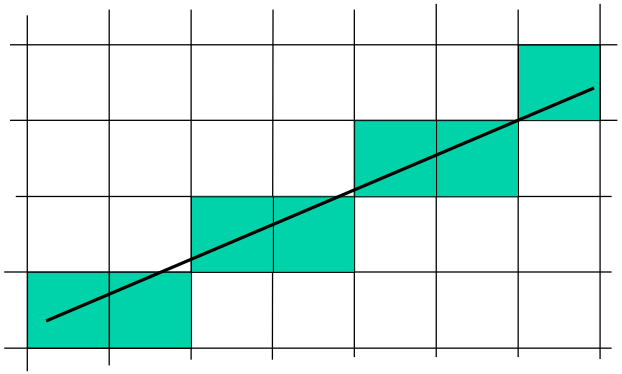
Line drawing--simple line (Bresenham) brightness issues



8 pixels per 8*sqrt(2) length

8 pixels for 8 length
(Brighter)

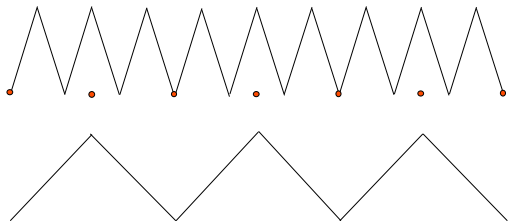
Line drawing--discretization artifacts (often called aliasing)



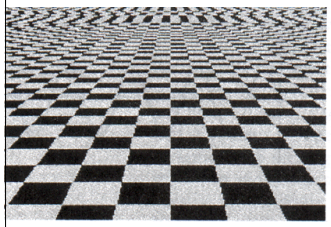
## Aliasing [ H&B, pp 214-221]

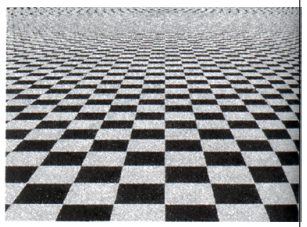
- We are using discrete binary squares to represent perfect mathematical entities
- To get a value for that that square we used a “sample” at a particular discrete location.
- The sample is somewhat arbitrary due to the choice of discretization, leading to the jagged edges.
- Insufficient samples mean that higher frequency parts of the signal can “alias” (masquerade as) lower frequency information.

## Aliasing [ H&B, figure 4-46]



## Aliasing





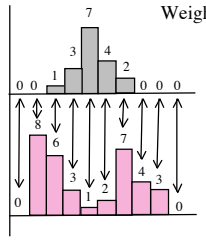
from Watt and Policarpo, The Computer Image

## Aliasing (cont)

- Points and lines as discussed so far have no width. To make them visible we concocted a way to sample them based on which discrete cell was closer
- 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
- A more principled approach (which subsumes the above) is to "filter" before sampling.

## Linear Filters (background)

- General process: Form new image whose pixels are a **weighted sum** of original pixel values, using the same set of weights at each point.




Weights (kernel of filter)

Multiply lined up pairs of numbers and then sum up to get weighted average at the filter location. Then shift the filter and do the same to get the next value.

Signal