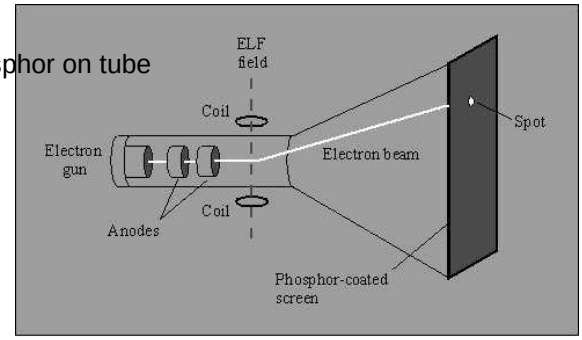
CSC418 Computer Graphics

- **Display Technology**
- **Drawing lines**

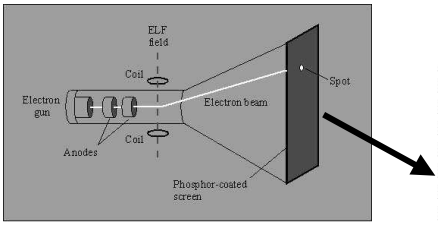
Display Technology

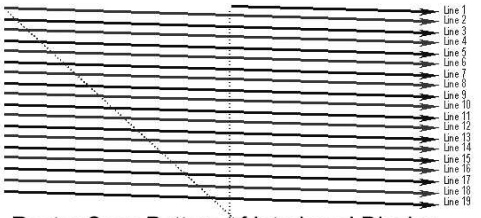
Raster Displays (CRT)

- Common display device today
- Evacuated glass bottle
- Electrons attracted to anode
- Deflection plates
- Beam strikes phosphor on tube

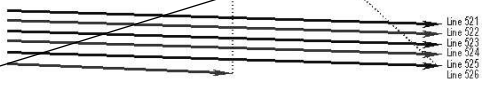


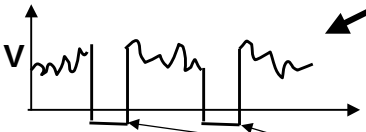
Raster Displays I





Raster Scan Pattern of Interlaced Display



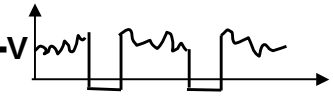


time

Electron beam Intensity

Raster Displays II

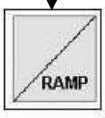
Gamma correction



time

Electron beam Intensity

Sample Input to Monitor



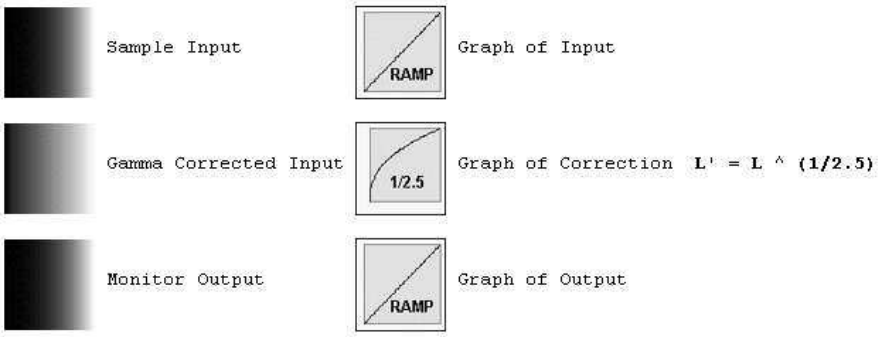
Graph of Input

Output from Monitor

Graph of Output $L = V^{2.5}$

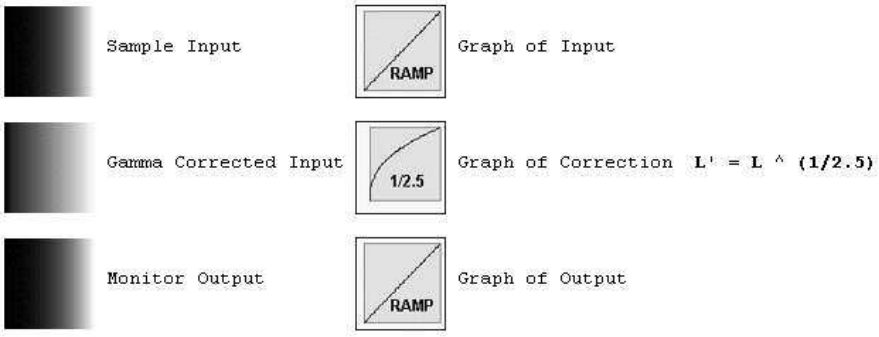
Raster Displays II

Gamma correction



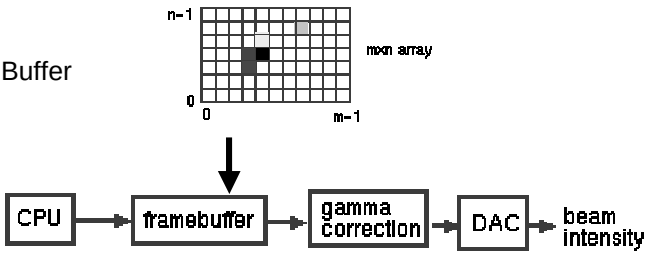
Raster Displays II

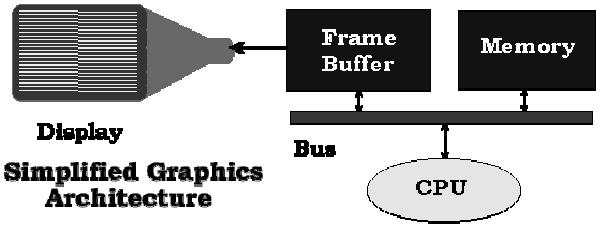
Gamma correction



Display Architecture

Frame Buffer

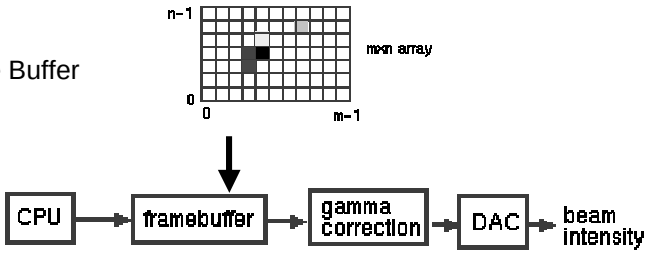


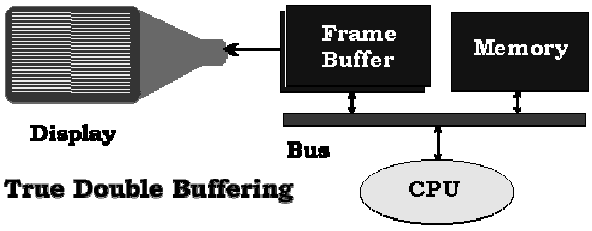


Simplified Graphics Architecture

Display Architecture

Double Buffer

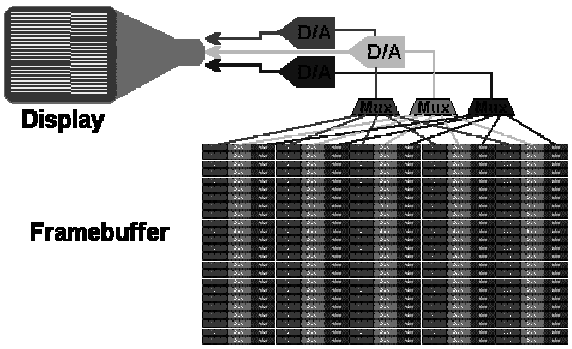




True Double Buffering

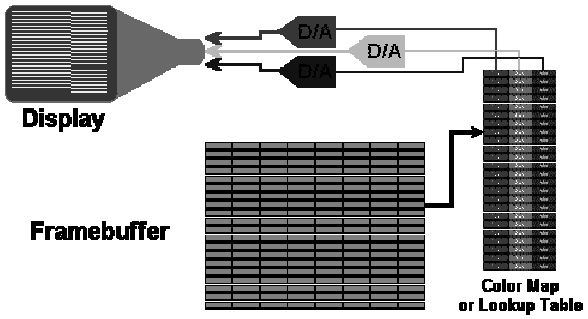
Display Architecture II

True Color Frame Buffer : 8 bits per pixel RGB



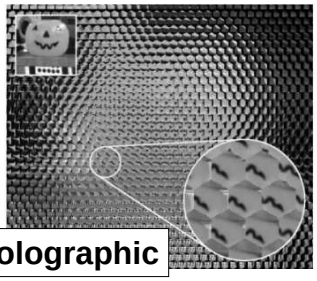
Display Architecture II

Indexed Color Frame Buffer : 8 bit index to color map



Display Devices II

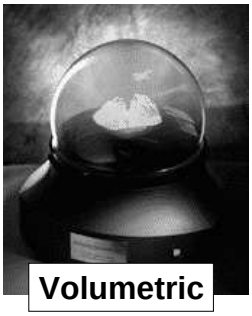
Plasma



Holographic

Immersive

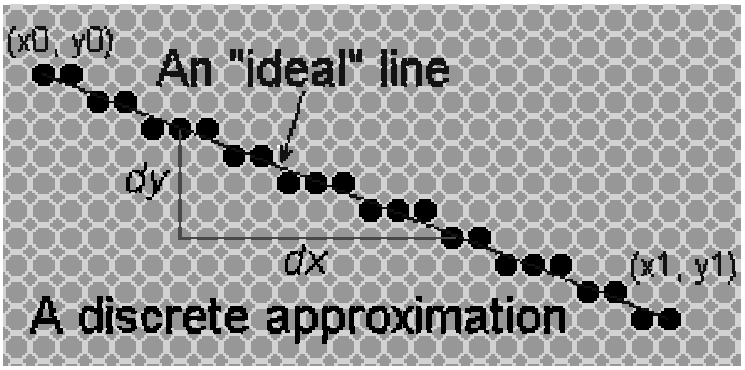
Head-mounted



Volumetric

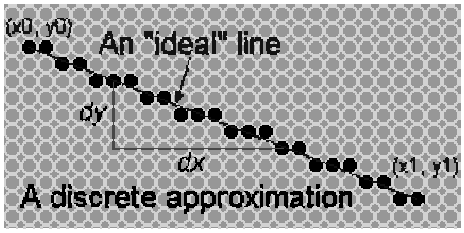
Line Drawing

What is the best line line we can draw?



Line Drawing

What is the best line line we can draw?



The best we can do is a discrete approximation of an ideal line.

Important line qualities:

- **Continuous appearence**
- **Uniform thickness and brightness**
- **Accuracy (Turn on the pixels nearest the ideal line)**
- **Speed (How fast is the line generated)**

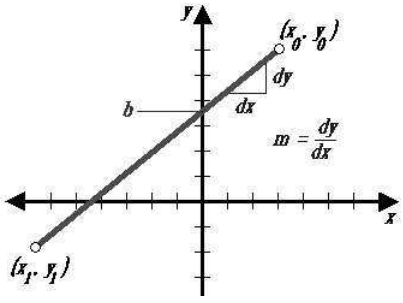
Equation of a Line

Explicit : $y = mx + b$

Parametric :

$x(t) = x_0 + (x_1 - x_0)*t$

$y(t) = y_0 + (y_1 - y_0)*t$



$P = P_0 + (P_1 - P_0)*t$

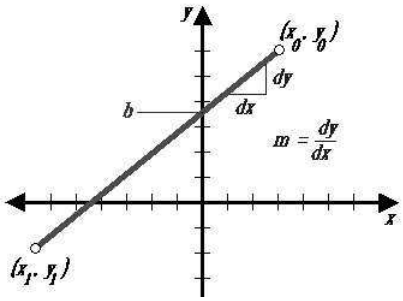
$P = P_0*(1-t) + P_1*t$ **(weighted sum)**

Implicit : $(x-x_0)dy - (y-y_0)dx = 0$

Algorithm I

Explicit form:

$y = dy/dx * (x-x_0) + y_0$

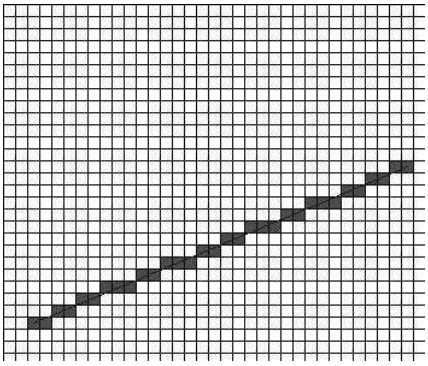



```
float y;
int x;
for ( x=x0; x<=x1; x++)
{
   y= y0 + (x-x0)*(y1-y0)/(x1-x0);
   setpixel (x, round(y));
 }
```

Algorithm I

Explicit form:

$y = dy/dx * (x-x_0) + y_0$

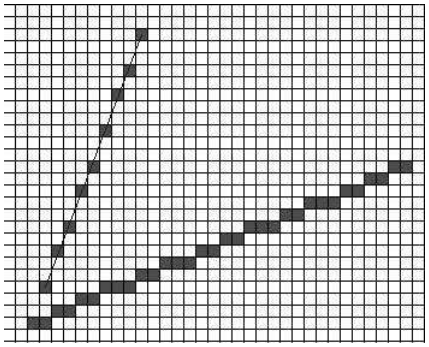


```
float y;
int x;
dx = x1-x0; dy = y1 – y0;
m = dy/dx;
y= y1 + 0.5;
for ( x=x0; x<=x1; x++)
{
   setpixel (x, floor(y));
   y= y + m;
 }
```

Algorithm I

DDA (Digital Differential Analyzer)

```
float y;
int x;
dx = x1-x0; dy = y1 – y0;
m = dy/dx;
y= y1 + 0.5;
for ( x=x0; x<=x1; x++)
{
   setpixel (x, floor(y));
   y= y + m;
 }
```



Algorithm II

Bresenham Algorithm

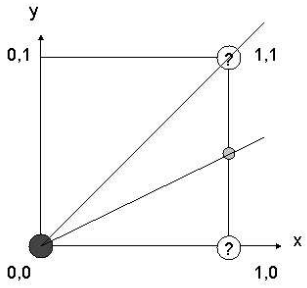
- **Assume line slope <1 (first quadrant)**
- **Slope is rational (ratio of two integers). m = (y1 - y0) / (x1 - x0)**
- **The incremental part of the algorthim never generates a new y that is more than one unit away from the old one (because the slope is always less than one) $y_{i+1} = y_i + m$**
- **If we maintained only the fractional part of *y*, we could modify the DDA and draw a line by noting when this fraction exceeded one. If we initialize fraction with 0.5, then rounding off is handled. fraction += m; if (fraction >= 1) { y = y + 1; fraction -= 1; }**

Algorithm II

Bresenham Algorithm Geometric Interpretation

```
fraction = m/2 – 1/2

if m <= 0  then fraction >= 1
{
        Plot(1,1)
        fraction += m –1;
}
else
{
        Plot(1,0)
        fraction += m;
}
```



Algorithm II

Bresenham Algorithm

Implicit View

F(x,y) = (x-x0)dy - (y-y0)dx
F(x+1,y + 0.5) = F(x,y) + dy -0.5 dx

2 F(x+1,y+ 0.5) = d = 2F(x,y) + 2dy -dx

F(x+1,y) = F(x,y) + dy d' = d + 2dy
F(x+1,y+1) = F(x,y) +dy-dx d' = d + 2dy - 2dx

CSC418 Computer Graphics

Next Lecture….

- **Simple Camera model**
- **Display techniques**
  - **Z Buffer**
  - **Ray Tracing**
- **Scan conversion**