Rasterization

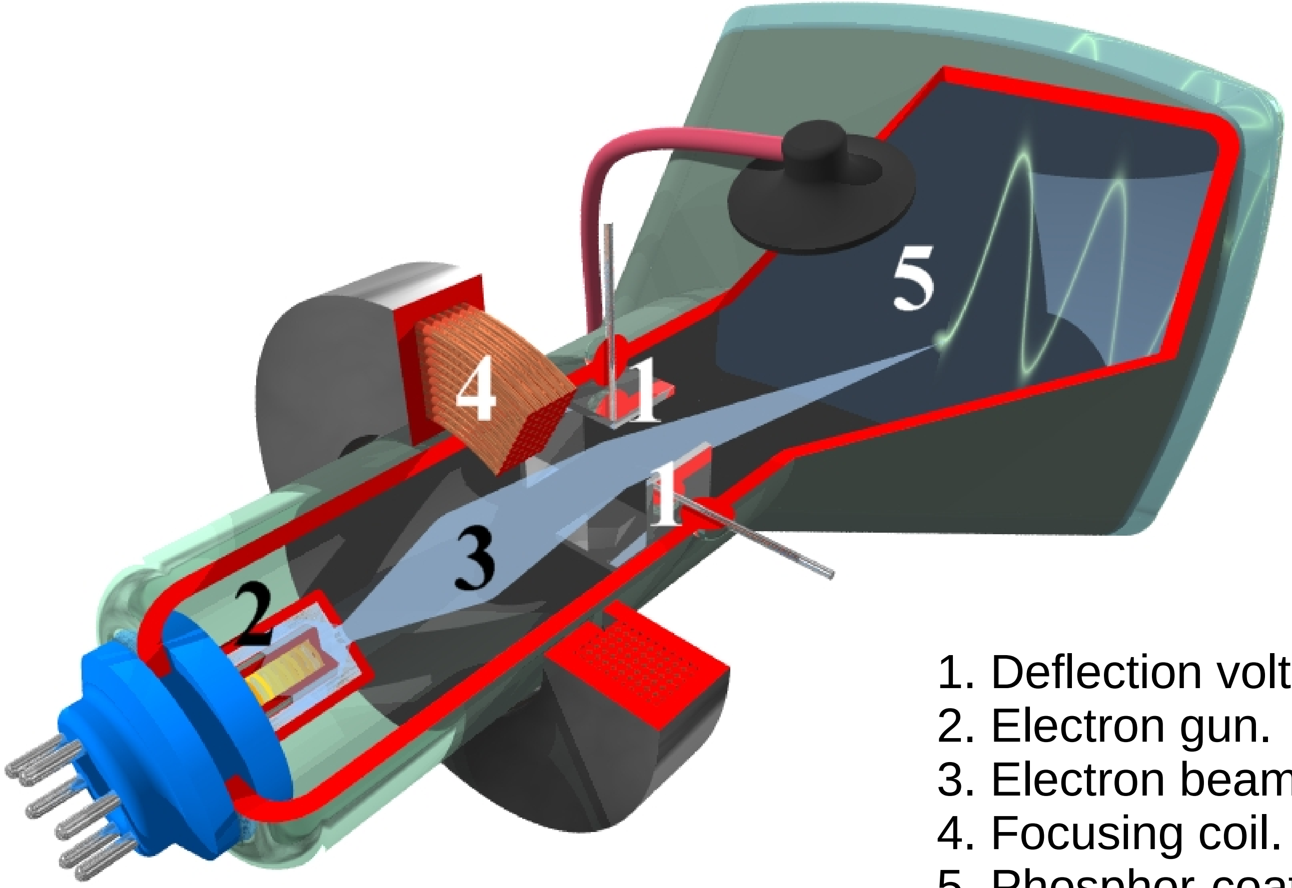
Lecture 2

1107190 - Introdução à Computação Gráfica

Prof. Christian Azambuja Pagot
CI / UFPB

Universidade Federal da Paraíba
Centro de Informática

Vector Display



1. Deflection voltage eletrode.
2. Electron gun.
3. Electron beam.
4. Focusing coil.
5. Phosphor-coated inner side of the screen.

Søren Peo Pedersen (Wikipedia)

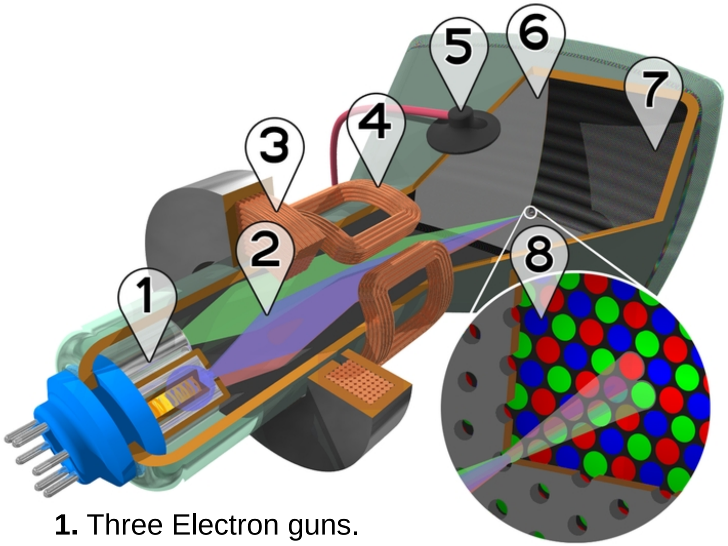
Vector Graphics

Vectrex video game (video)

Blakespot (Flickr)

Raster Displays

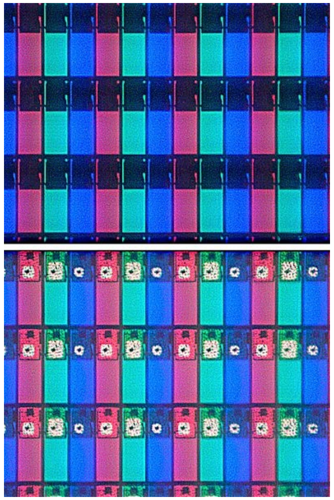
CRT Display



1. Three Electron guns.
2. Electron beams.
3. Focusing coils.
4. Deflection coils.
5. Anode connection.
6. Mask for separating beams.
7. Phosphor layer.
8. Close-up of the phosphor-coated inner side of the screen.

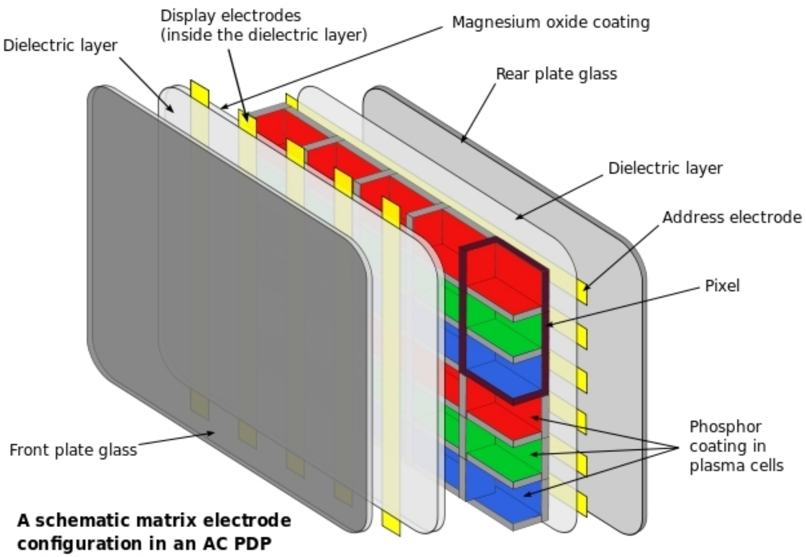
Søren Peo Pedersen (Wikipedia)

LCD Display



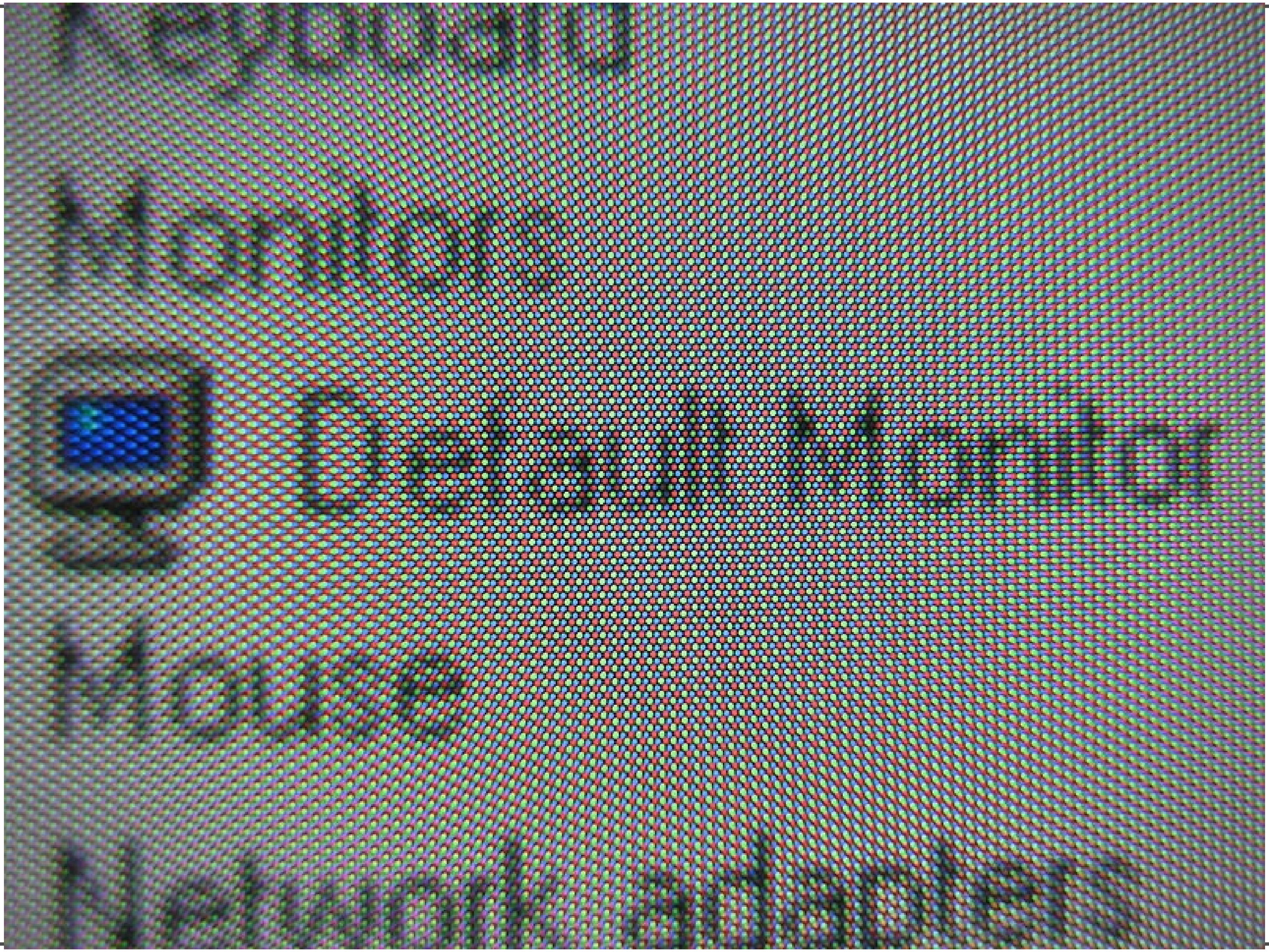
Gabelstaplerfahrer (Wikipedia)

Plasma Display



Jari Laamanen (Wikipedia)

Raster Display

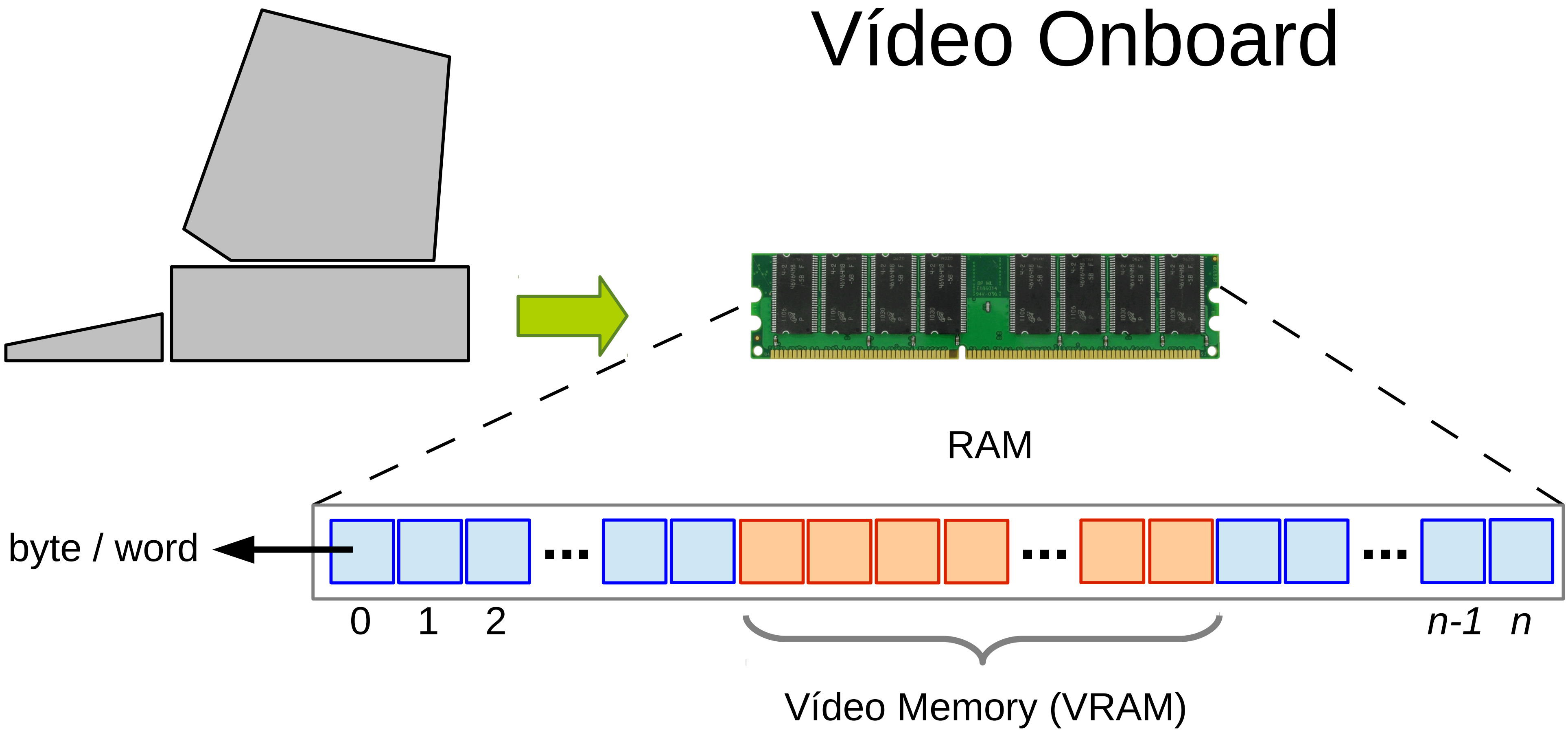


Gona.eu (Wikipedia)

Raster Graphics

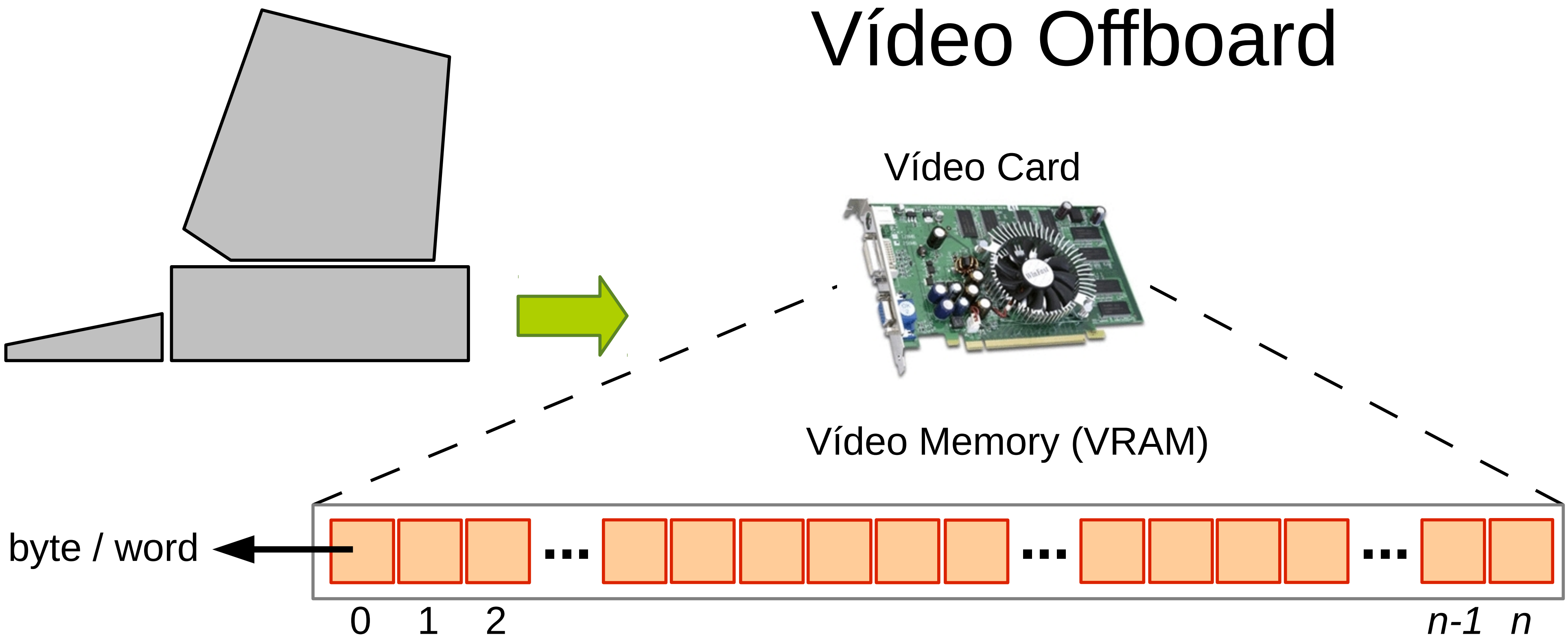
Jet 2 (1987) (video)

Video Memory



Video Memory

Vídeo Offboard



How about colors?

Colors (Quick view)

- Electromagnetic spectrum

Visible spectrum

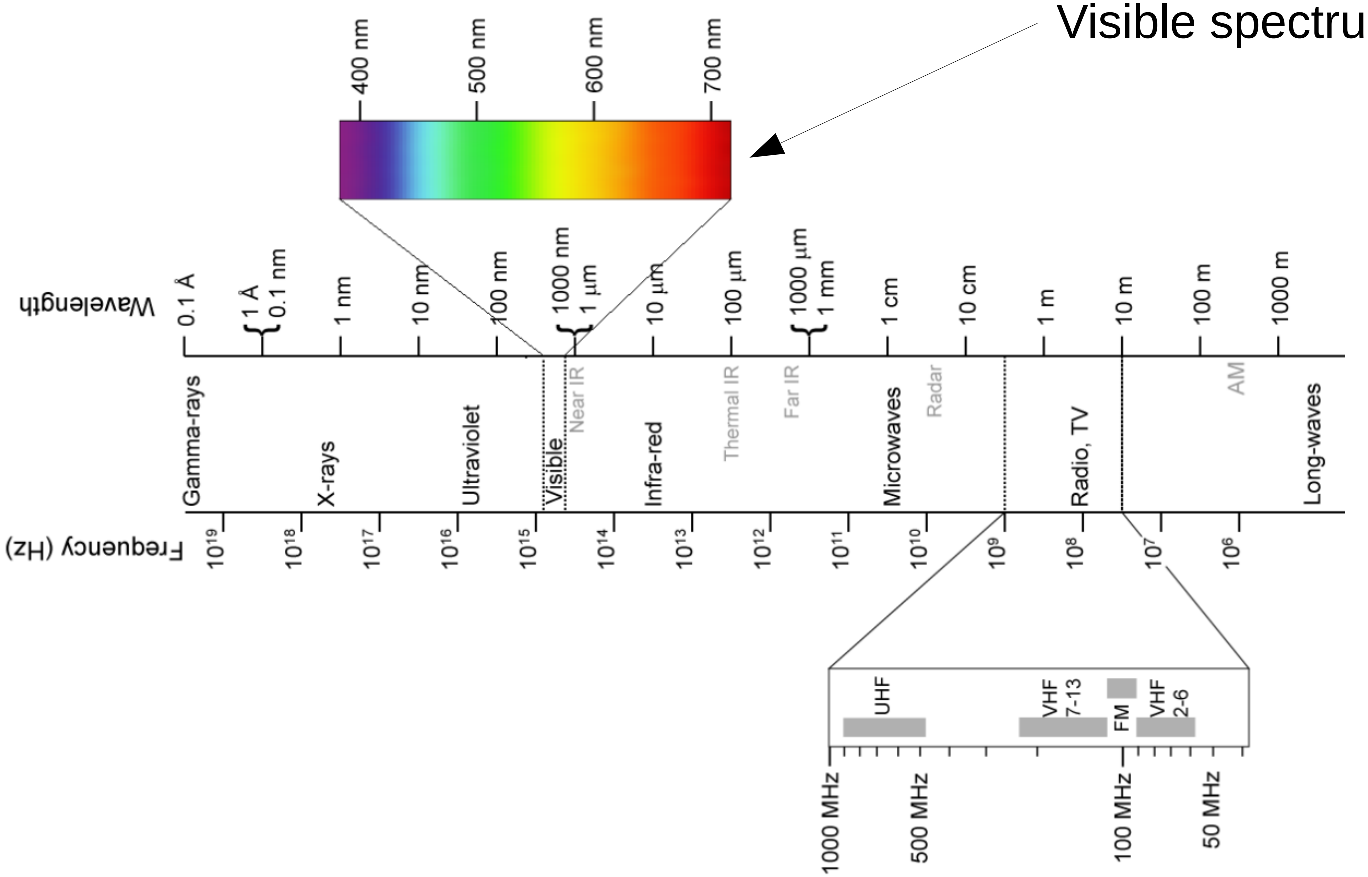


Image by penubag (Wikipedia)

Colors (Quick view)

- CIE Color Space (1931)

- CIE RGB Color Space

Normally used in computers!

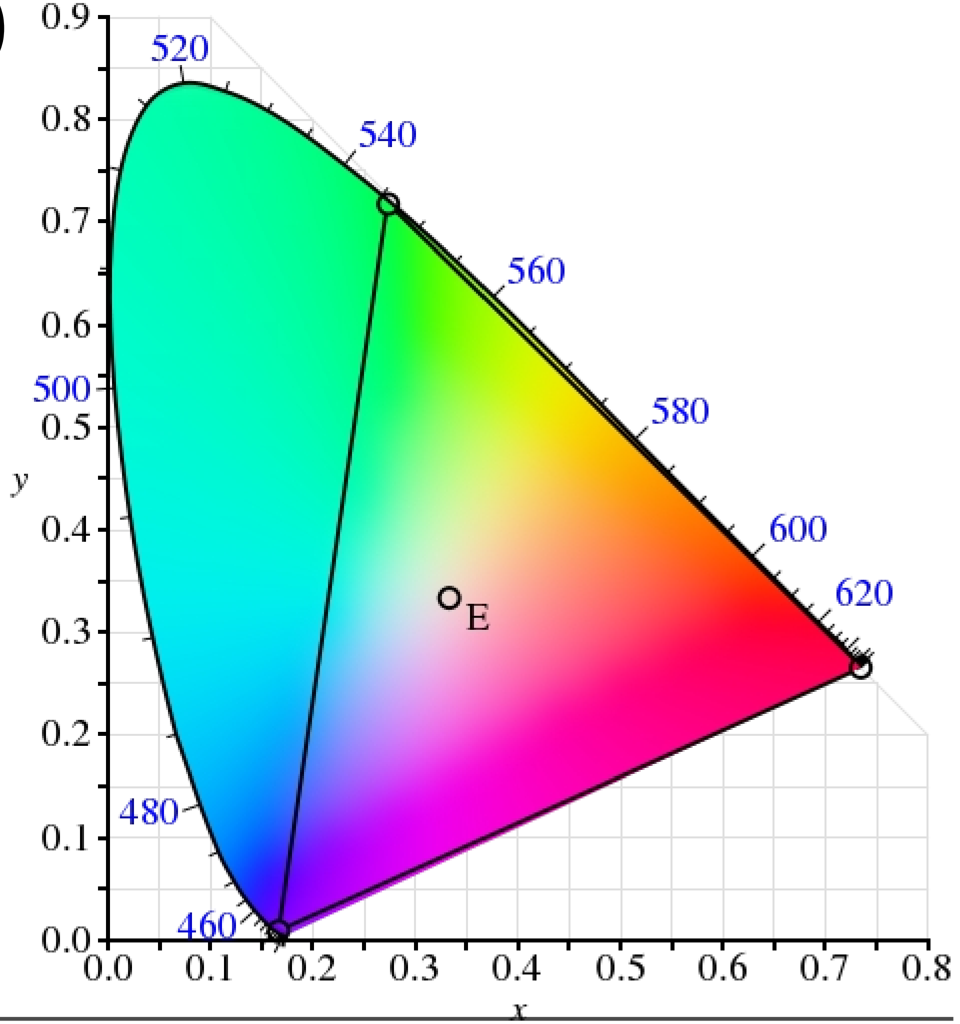


Image by BenRG (Wikipedia)

RGB Representation

- The **intensity** of each **component** is represented by a **number**.
- Usually, **8 bits** are reserved for each **component** (**channel**). Thus, **each component** can present up to **256 levels** of intensity (256^3 = **~16 millions of colors**).
- An additional channel (**alpha**) can be used for **transparency** (RGB**A**).
- It's not uncommon to have **different** number of **bits** per **channel**.

Image Storage

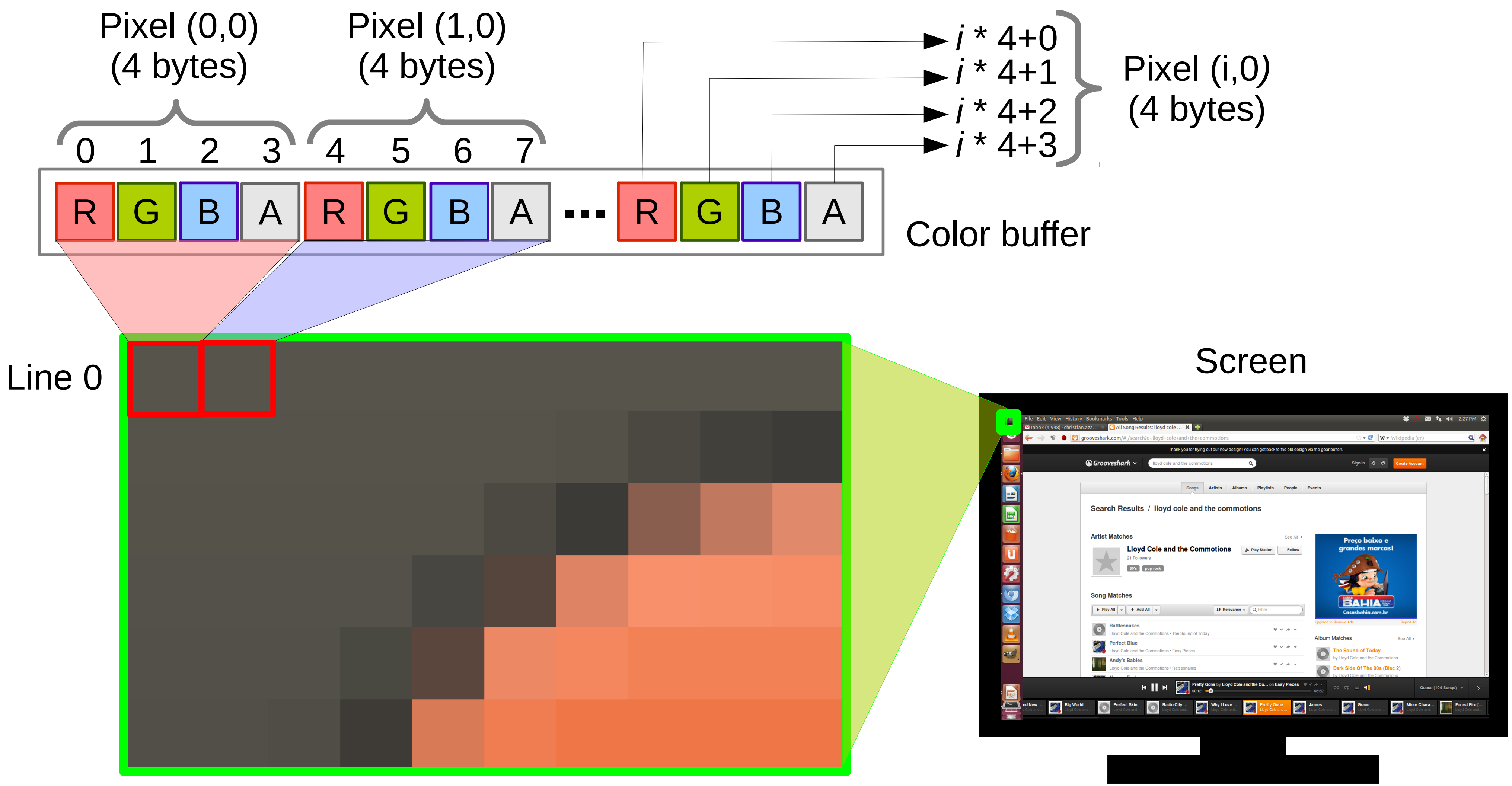
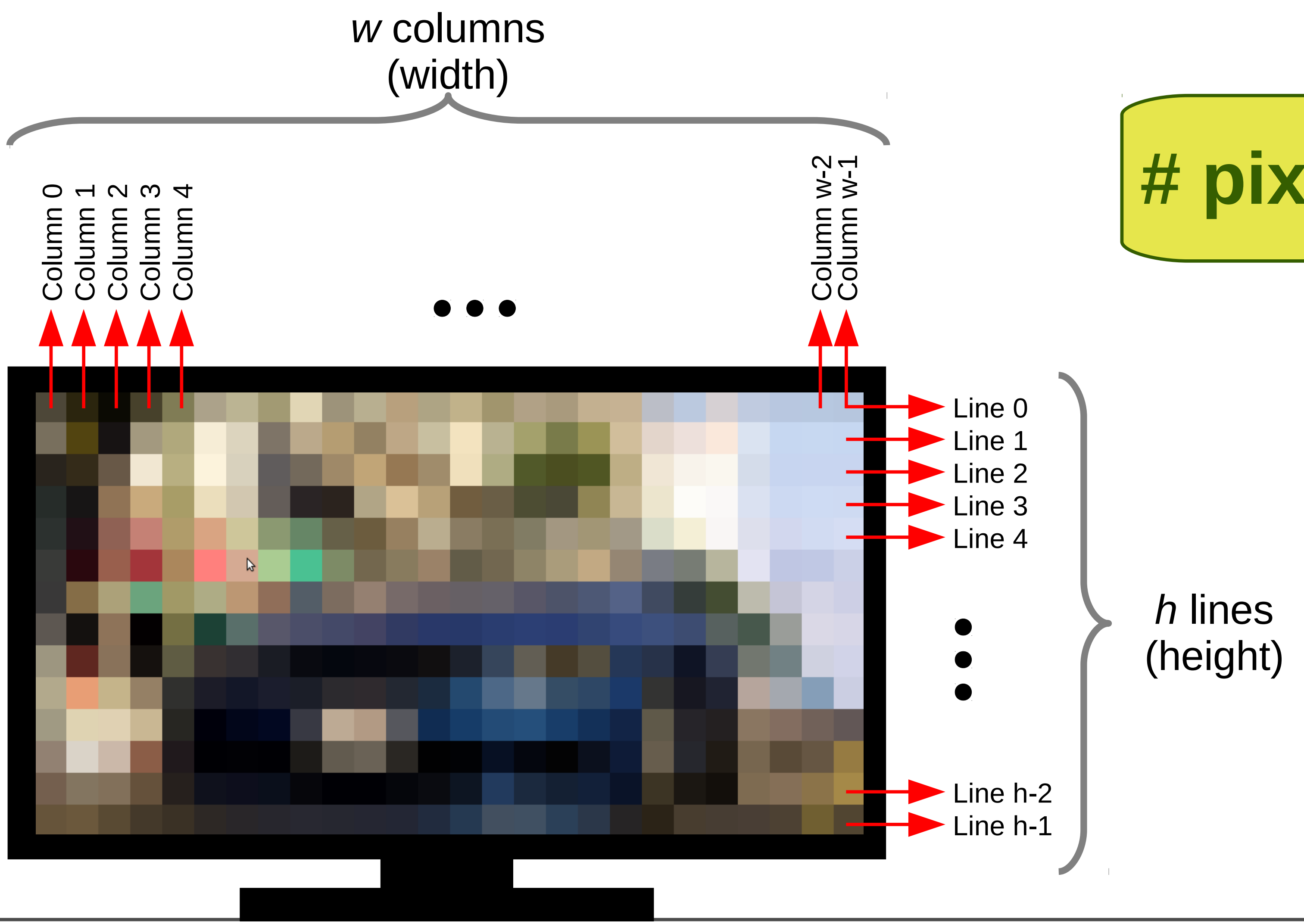
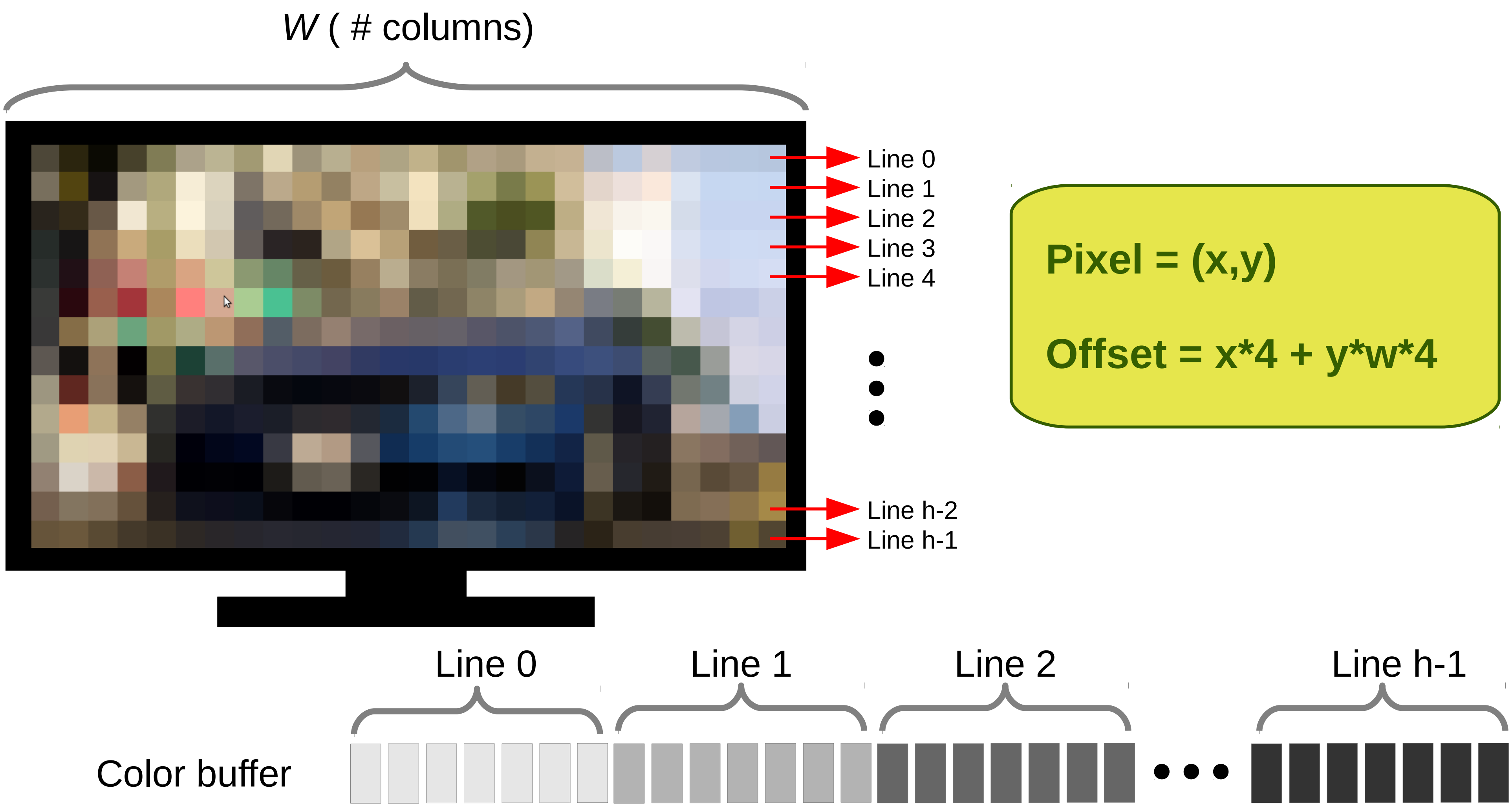


Image Storage



 # pixels = *w* * *h*

Image Storage

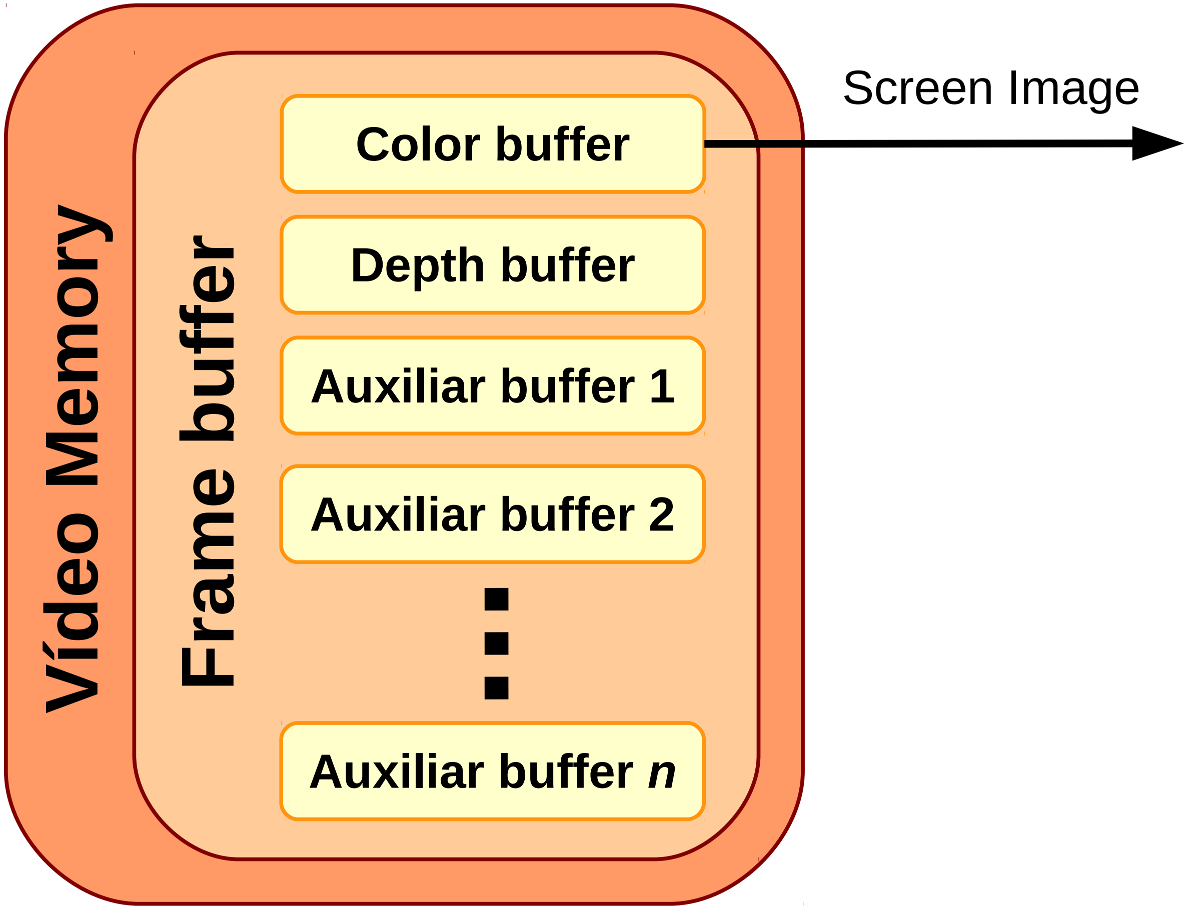


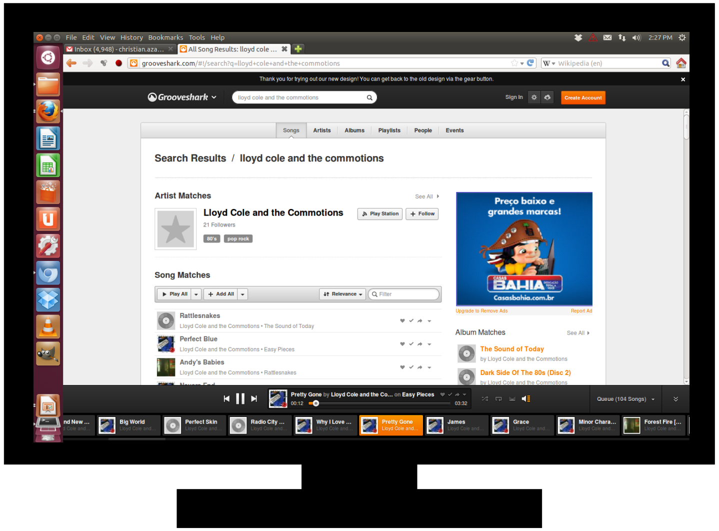
Pixel = (x,y)

Offset = x*4 + y*w*4

Image Storage

- Vídeo memory screen image footprint:





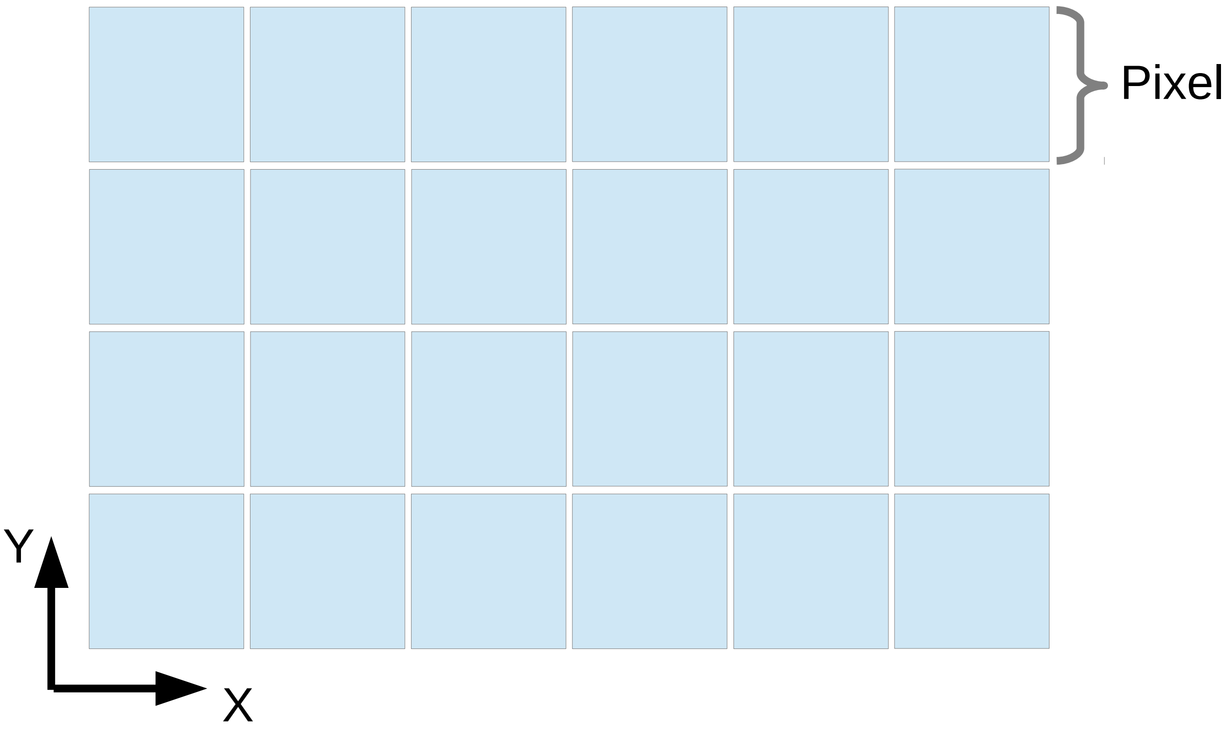
Rasterization

- “Approximation of mathematical ('ideal') primitives, described in terms of vertices on a Cartesian grid, by sets of pixels of the appropriate intensity of gray or color.”

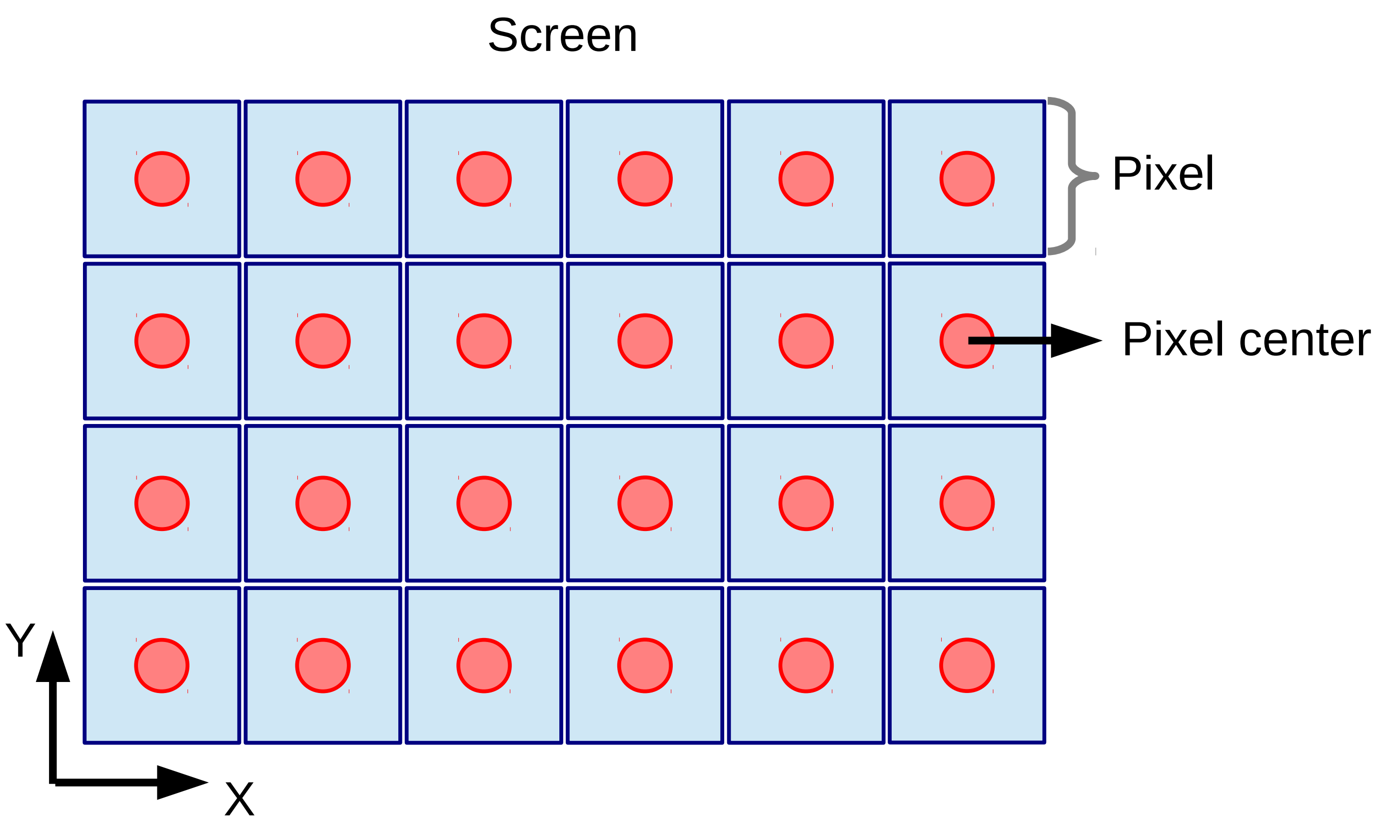
- *Foley et. al*

Rasterization

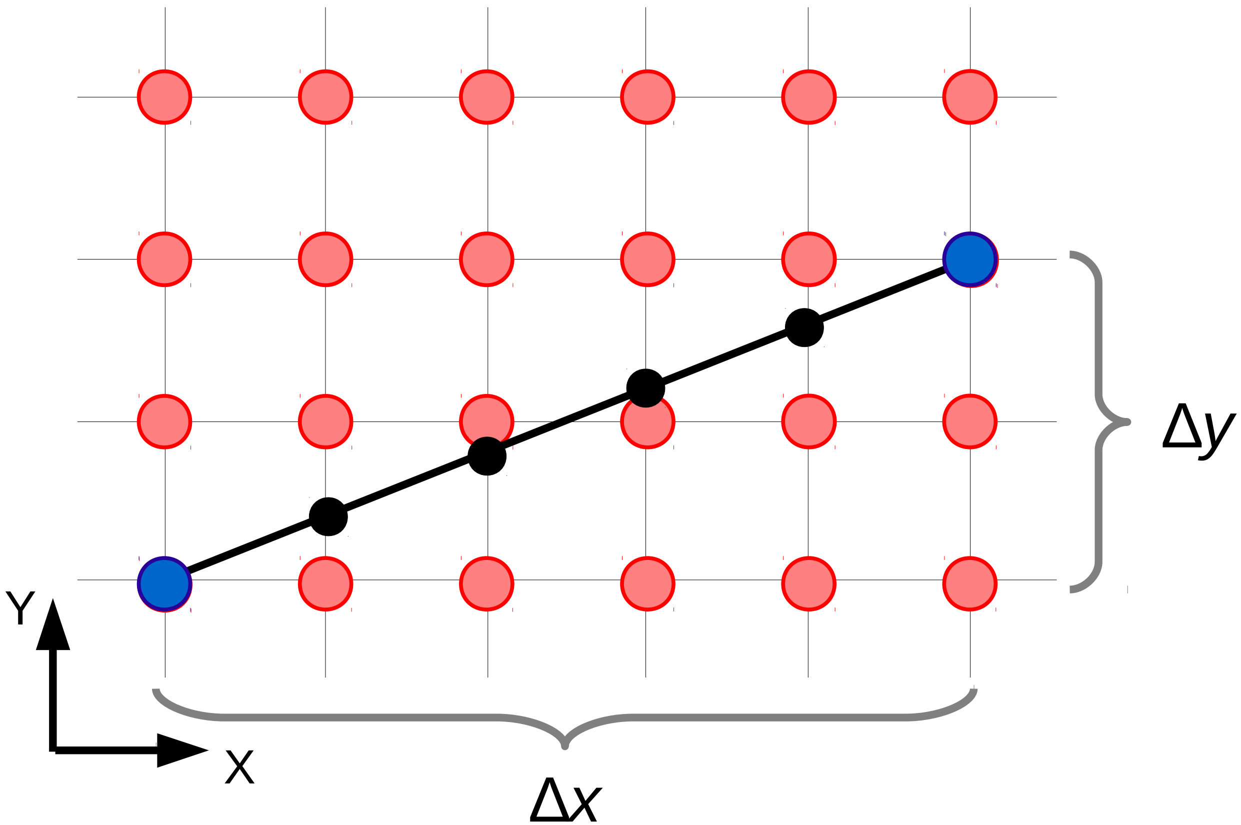
Screen



Rasterization



Rasterizing Lines



Since Δx is greater Δy:

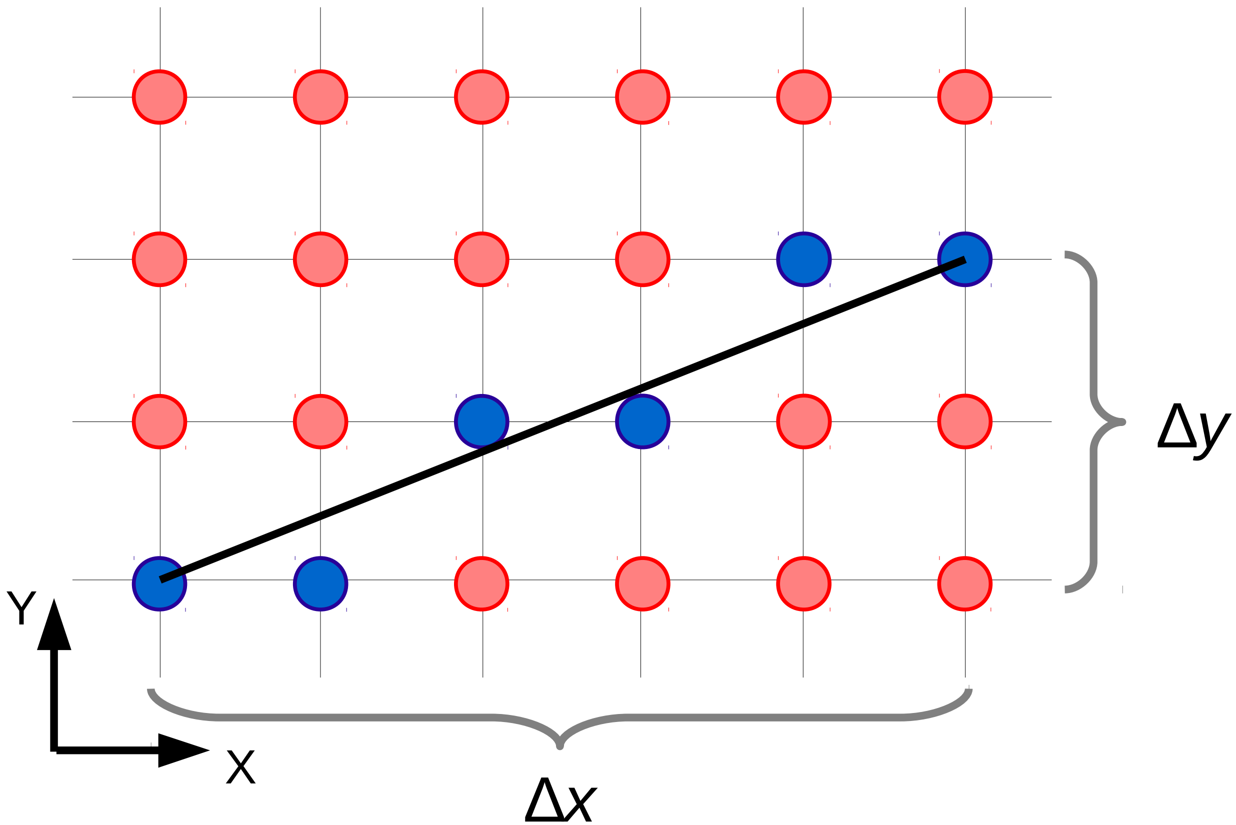
$$m = \frac{\Delta y}{\Delta x}$$

$$y_i = m x_i + b$$

By incrementing *x* by 1, we can compute the corresponding *y*:

1st point: $(x_0, mx_0 + b))$
2nd point: $(x_1, mx_1 + b))$
3rd point: $(x_2, mx_2 + b))$
.
.
.
nth point: $(x_n, mx_n + b))$

Rasterizing Lines



Since Δx is greater Δy:

$$m = \frac{\Delta y}{\Delta x}$$

$$y_i = m x_i + b$$

By incrementing *x* by 1, we can compute the corresponding *y*:

1st point: (x_0, **Round**(mx_0+ b))
2nd point: (x_1, **Round**(mx_1+ b))
3rd point: (x_2, **Round**(mx_2+ b))
.
.
.
nth point: (x_n, **Round**(mx_n+ b))

Rasterizing Lines

- Problems with this approach:
  - At each iteration:
    - A floating point multiplication.
    - A floating point addition.
    - A Round operation.

n^{th} point: $(x_n, \text{Round}(mx_n + b))$

Rasterizing Lines

- Solution:
  - Multiplication can be eliminated:

$$y_{i+1} = m\, x_{i+1} + b$$

$$y_{i+1} = m\left(x_i + \Delta x\right) + b$$

$$y_{i+1} = y_i + m\,\Delta x$$

  - If $\Delta x = 1$:

$$y_{i+1} = y_i + m$$

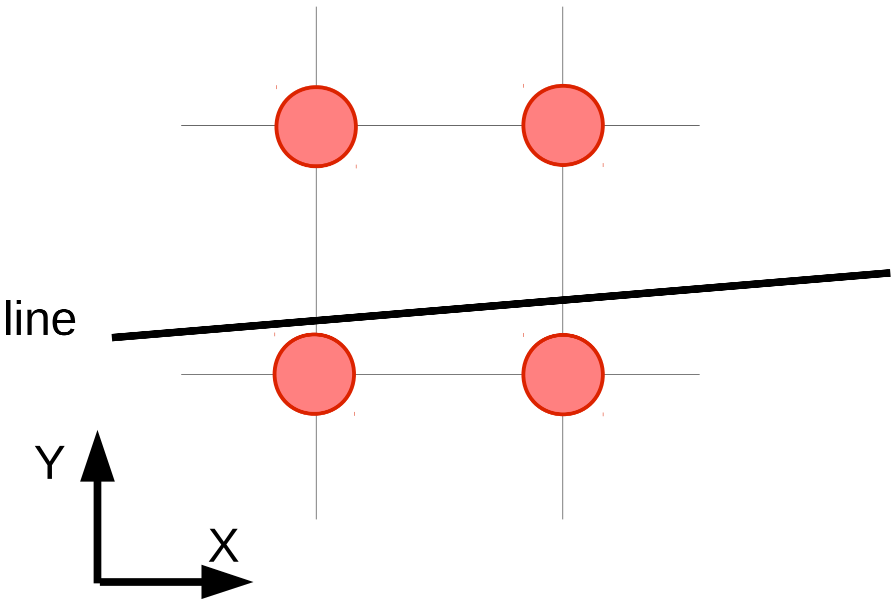
This is an incremental algorithm.

Usually referred as the DDA (digital differential analyzer) algorithm.

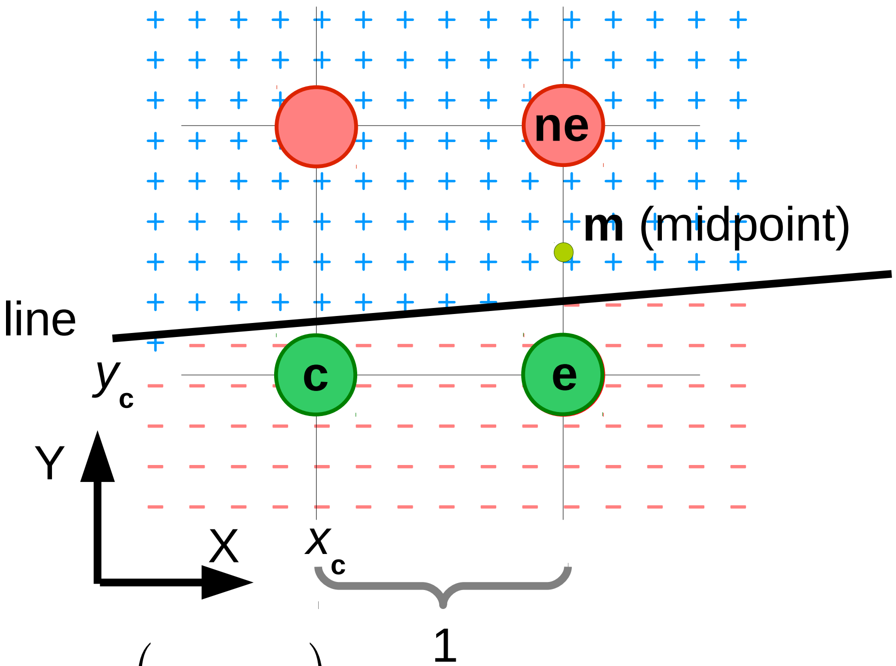
Bresenham Line Algorithm

- Incremental.
- Avoids multiplications and roundings.
- Can be generalized for circles.

Variation of Bresenham's Algor.



Variation of Bresenham's Algor.



$$\mathbf{c} = (x_\mathbf{c}, y_\mathbf{c})$$

$$\mathbf{e} = (x_\mathbf{c}+1, y_\mathbf{c})$$

$$\mathbf{ne} = (x_\mathbf{c}+1, y_\mathbf{c}+1)$$

$$\mathbf{m} = (x_\mathbf{c}+1, y_\mathbf{c}+\frac{1}{2})$$

Assuming that 0 ≤ *m* ≤ 1:

$$y = mx + b$$

$$y = \left(\frac{\Delta y}{\Delta x}\right) x + b$$

$$\begin{array}{ccc} \alpha & = & \Delta y \\ \beta & = & -\Delta x \\ \gamma & = & b \cdot \Delta x \end{array}$$

$$\Phi(x, y) = \alpha x + \beta y + \gamma = 0$$

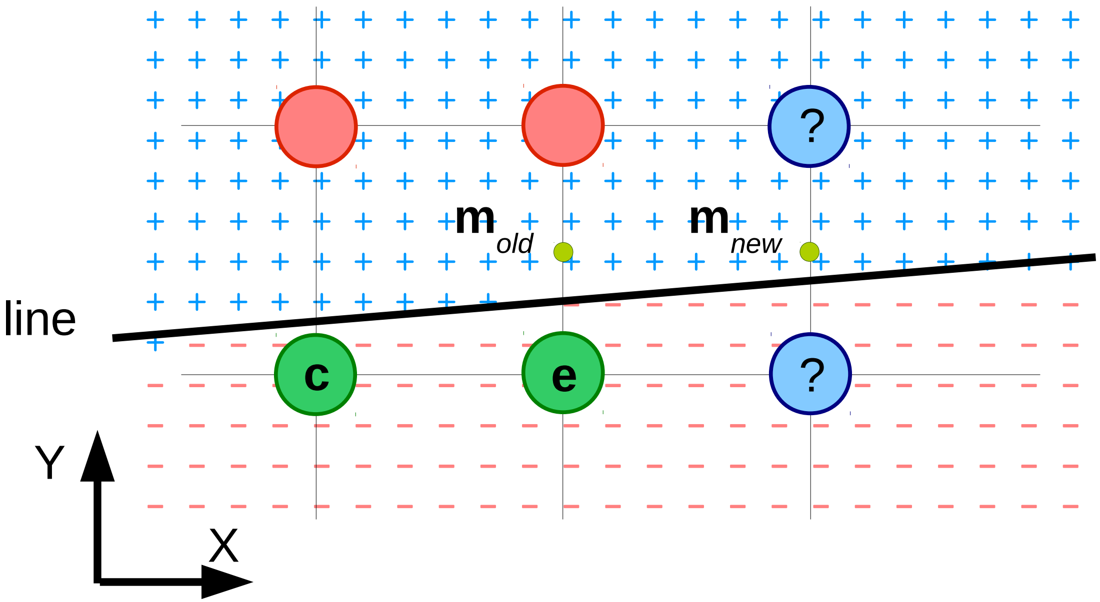
$$\boxed{\Phi(x, y) = \Delta y \cdot x - \Delta x \cdot y + b \cdot \Delta x = 0}$$

$$d = \Phi(\mathbf{m}) \rightarrow \textit{Decision variable}$$

```
if (d < 0)
        next pixel = ne
else
        next pixel = e
```

Will we have to evaluate a **polynomial** every **pixel**?

Variation of Bresenham's Algor.



$$d_{old} = \Phi(\mathbf{m}_{old}) = \Phi((x_{\mathbf{c}}+1, y_{\mathbf{c}}+\frac{1}{2}))$$

$$d_{new} = \Phi(\mathbf{m}_{new}) = \Phi((x_{\mathbf{c}}+2, y_{\mathbf{c}}+\frac{1}{2}))$$

If E is choosen:

$$d_{old} = \alpha(x_{\mathbf{c}}+1) + \beta(y_{\mathbf{c}}+\frac{1}{2}) + \gamma$$

$$d_{new} = \alpha(x_{\mathbf{c}}+2) + \beta(y_{\mathbf{c}}+\frac{1}{2}) + \gamma$$

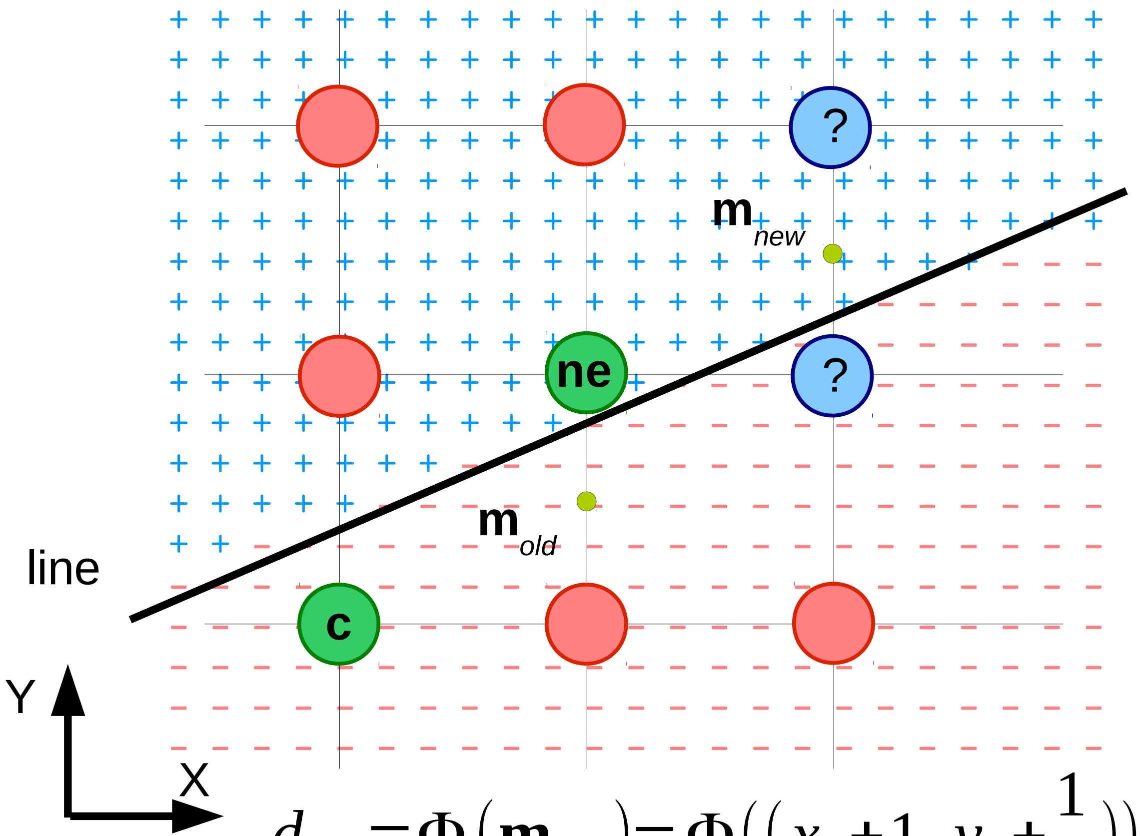
$$d_{new} - d_{old} \rightarrow \boxed{d_{new} = d_{old} + \alpha}$$

Remembering...

$$\Phi(x, y) = \alpha x + \beta y + \gamma$$

$$\begin{array}{ccc} \alpha & = & \Delta y \\ \beta & = & -\Delta x \\ \gamma & = & b \cdot \Delta x \end{array}$$

Variation of Bresenham's Algor.



$$d_{old} = \Phi(\mathbf{m}_{old}) = \Phi((x_{\mathbf{c}}+1, y_{\mathbf{c}}+\frac{1}{2}))$$

$$d_{new} = \Phi(\mathbf{m}_{new}) = \Phi((x_{\mathbf{c}}+2, y_{\mathbf{c}}+\frac{3}{2}))$$

If NE is choosen:

$$d_{old} = \alpha(x_{\mathbf{c}}+1) + \beta(y_{\mathbf{c}}+\frac{1}{2}) + \gamma$$

$$d_{new} = \alpha(x_{\mathbf{c}}+2) + \beta(y_{\mathbf{c}}+\frac{3}{2}) + \gamma$$

$$d_{new} - d_{old} \rightarrow \boxed{d_{new} = d_{old} + \alpha + \beta}$$

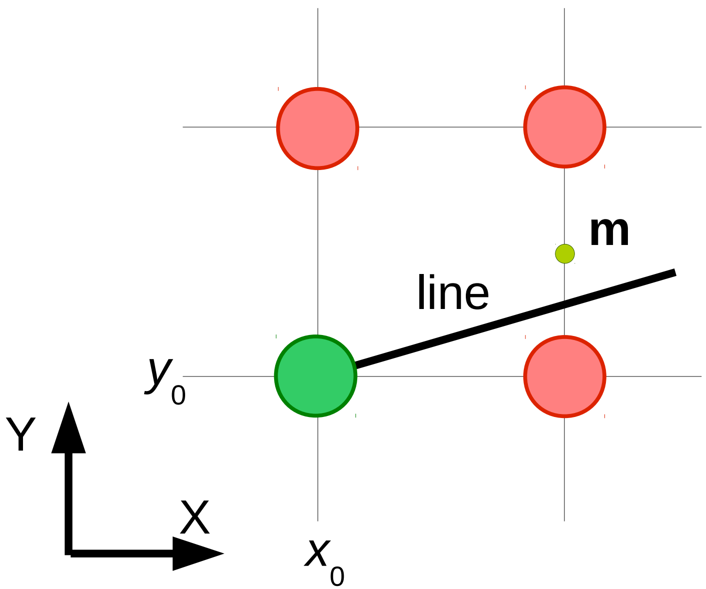
Remembering...

$$\Phi(x, y) = \alpha x + \beta y + \gamma$$

$$\begin{array}{rcl} \alpha & = & \Delta y \\ \beta & = & -\Delta x \\ \gamma & = & b \cdot \Delta x \end{array}$$

Variation of Bresenham's Algor.

- **How about the 1st pixel (there is no D_{old}!)?**



$$d=\Phi(\mathbf{m})$$
$$=\Phi\left(\left(x_0+1, y_0+\frac{1}{2}\right)\right)$$

$$d=\Phi(\mathbf{m})=\alpha(x_0+1)+\beta\left(y_0+\frac{1}{2}\right)+\gamma$$

$$d=\Phi(\mathbf{c})+\alpha+\frac{\beta}{2} \Rightarrow \Phi(\mathbf{c})=0$$

$$d=\alpha+\frac{\beta}{2} \Rightarrow \begin{array}{ccc} \alpha & = & \Delta y \\ \beta & = & -\Delta x \\ \gamma & = & b\cdot\Delta x \end{array}$$

$$d=\Delta y-\frac{\Delta x}{2}$$

$$\boxed{\Phi(x,y)=0=2\cdot 0=2\Phi(x,y)=2(\alpha x+\beta y+\gamma)}$$

$$\Rightarrow \boxed{d=2\Delta y-\Delta x}$$

Variation of Bresenham's Algor.

- **The entire algoritm for 0 < m < 1:**

```
MidPointLine() {
     int dx = x1 – x0;
     int dy = y1 – x0;
     int d = 2 * dy – dx;
     int incr_e = 2 * dy;
     int incr_ne = 2 * (dy – dx);
     int x = x0;
     int y = y0;
     PutPixel(x, y, color)
     while (x < x1) {
          if (d <= 0) {
               d += incr_e;
               x++;
          } else {
               d += incr_ne;
               x++;
               y++;
          }
          PutPixel(x, y, color);
     }
}
```

The **computation** of ***d***, now, **involves** only **addition**!

Slopes **outside** the range **[0,1]** can be **handled** by **symmetry**!

Other Rasterization Issues

- How about?
  - Other primitives:
    - Circles.
    - Ellipses.
    - Triangles.
  - Thick lines.
  - Shape of the endpoints.
  - Antialiasing.
  - Line stile.
  - Filling.
  - Etc.