

**Universidade do Vale do Paraíba**  
**Colégio Técnico Antônio Teixeira Fernandes**  
**Disciplina Introdução a Computação Gráfica**  
**(ICG)**  
***Material I***

*Primitivas Gráficas em 2D, Conceito de pontos pixel, Sistema de cores, linhas retas, Algoritmo DDA, Bresenham, Linhas cores e espessuras, Traçando círculos e elipses, Coordenadas polares, preenchimento de áreas, transformações geométricas em duas dimensões, Tratamento Imagens BMP (Mapa de Bits)*

Site : <http://www1.univap.br/~wagner>

***Prof. Responsável***

***Wagner Santos C. de Jesus***

Prof. Wagner Santos C. de Jesus  
[wagner@univap.br](mailto:wagner@univap.br)

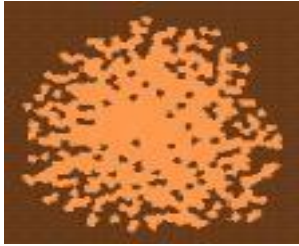


# Computação Gráfica (CG)

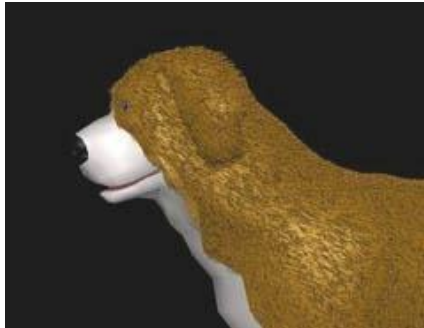
- Vem a ser a forma de se fazer uma representação da realidade graficamente na resolução de computadores.
- A CG vem auxiliando nas mais diversas áreas do conhecimento; facilitando a visualização e simulação de eventos sociais, culturais e científicos.

## Algumas áreas de atuação da computação gráfica

- Engenharia – Simulação e CAD
- Bioengenharia – Simulação de crescimento tumoral
- Medicina – Sistemas de eletrocardiograma e Tomografias
- Arte e Cinema – (Efeitos especiais e personagens).  
Representação de quadros e esculturas
- Fotografia – Processamento de imagens
- Geografia – Sistema informações geográficas
- Arquitetura – Sistemas especializados em plantas de edificações.
- Entretenimento e cultura – Jogos e vídeo games.
- Matemática, Física e estatística.



Simulação de crescimento tumoral



Criação de texturas de pelos de animais



Criação de realidade virtual  
simulação de deslocamento humano.

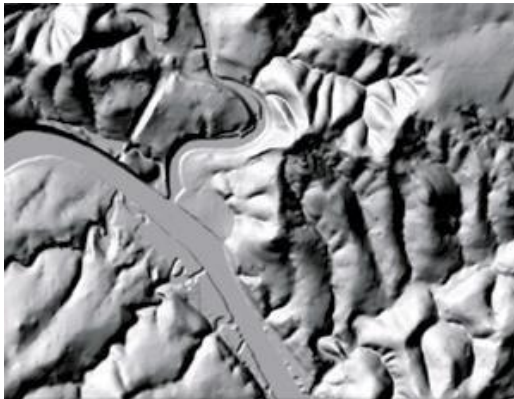


Imagem do leito de um rio  
em relevo

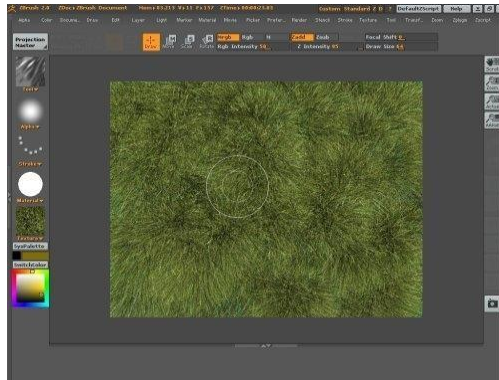


Simulador de Gestaç o



Figura 4.4 - Primeira modelo virtual contratada por uma agência, a Elite, para desfiles e comerciais. (Fonte: <http://www.optidigit.com/stevens>)

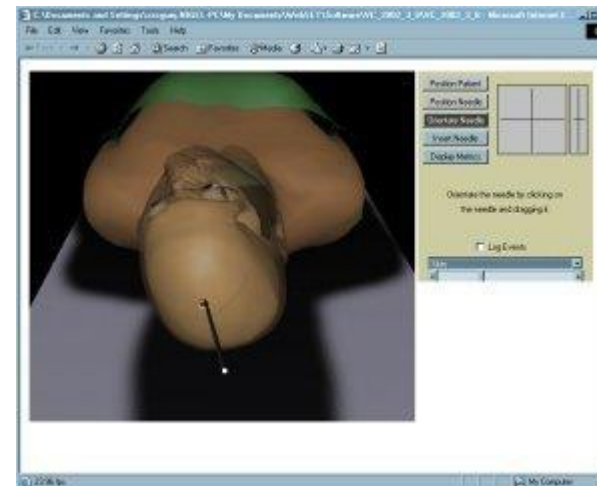
Criação de modelagem orgânica  
digital



Sistema de informações  
geográficas



Cena de Praça pública (CCRV)



Simulador Neuro  
Cirúrgico

Auto-CAD – Arquitetura e  
Edificações (AutoDesk)





## Vista Lateral (Praça Pública CCRV)

Prof. Wagner Santos C. de Jesus

[wagner@univap.br](mailto:wagner@univap.br)

# Entretenimento (Jogos em 2D)



Tiro ao Alvo



Corrida

## Estatística e Cinema

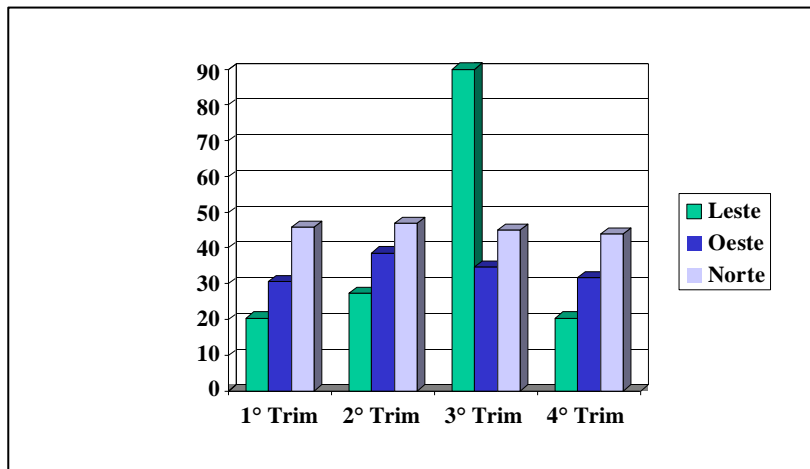
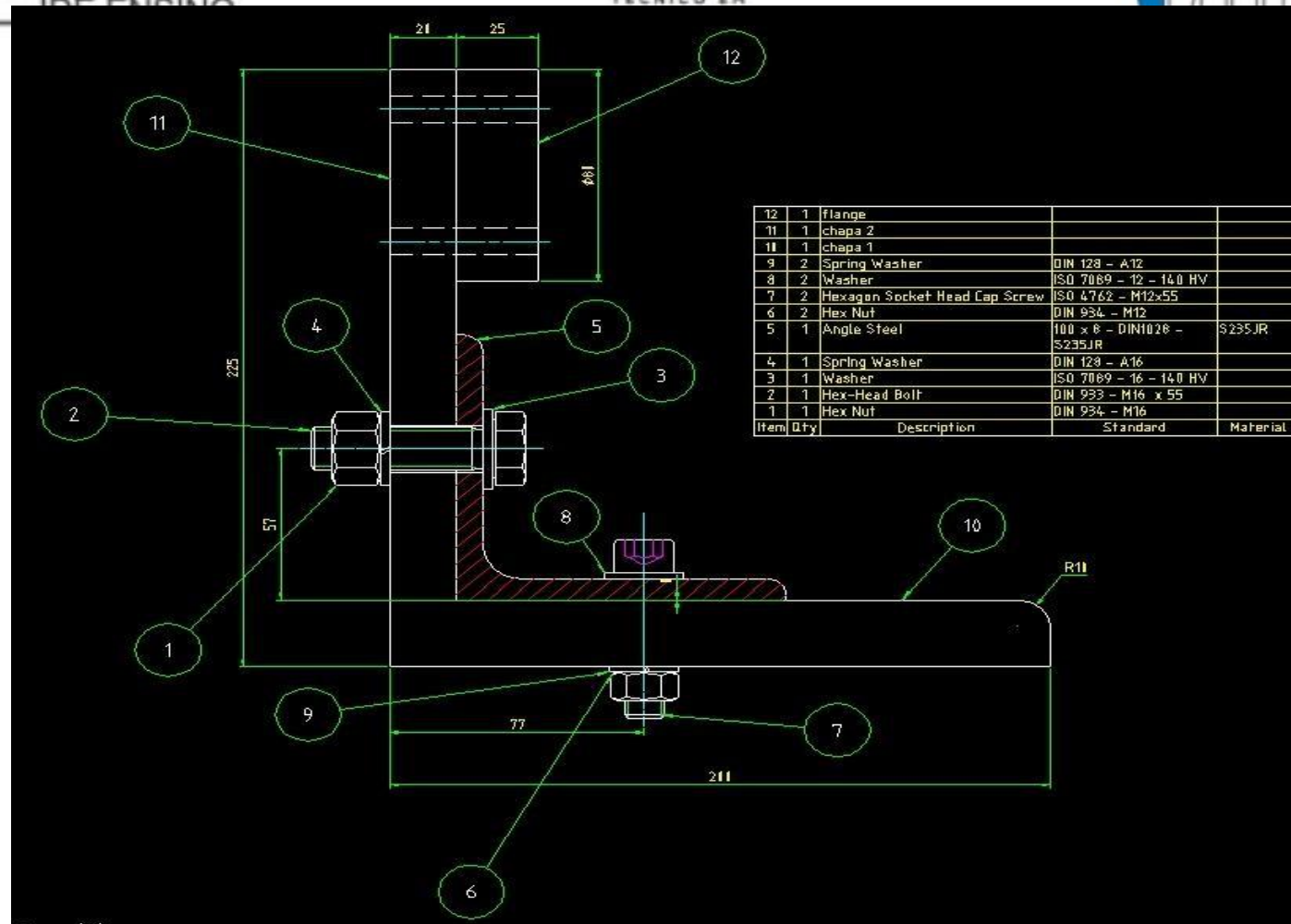


Gráfico (Estatística)



Cinema cena do filme (toyStoy - 1995) -Estúdio: Walt Disney

Pictures / Pixar Animation  
Studios



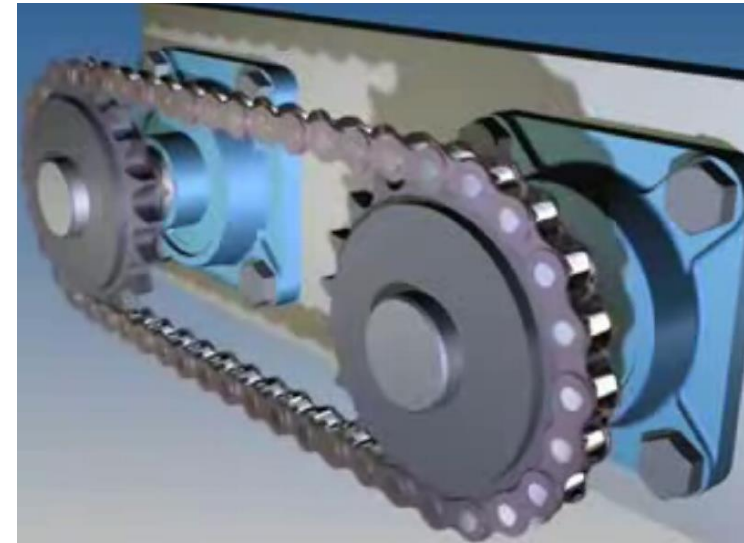
Desenho confeccionado em AutoCad 2D usando conceitos de primitivas básicas.



**2004 - Produtora(s):** DreamWorks SKG, Pacific Data Images (PDI)

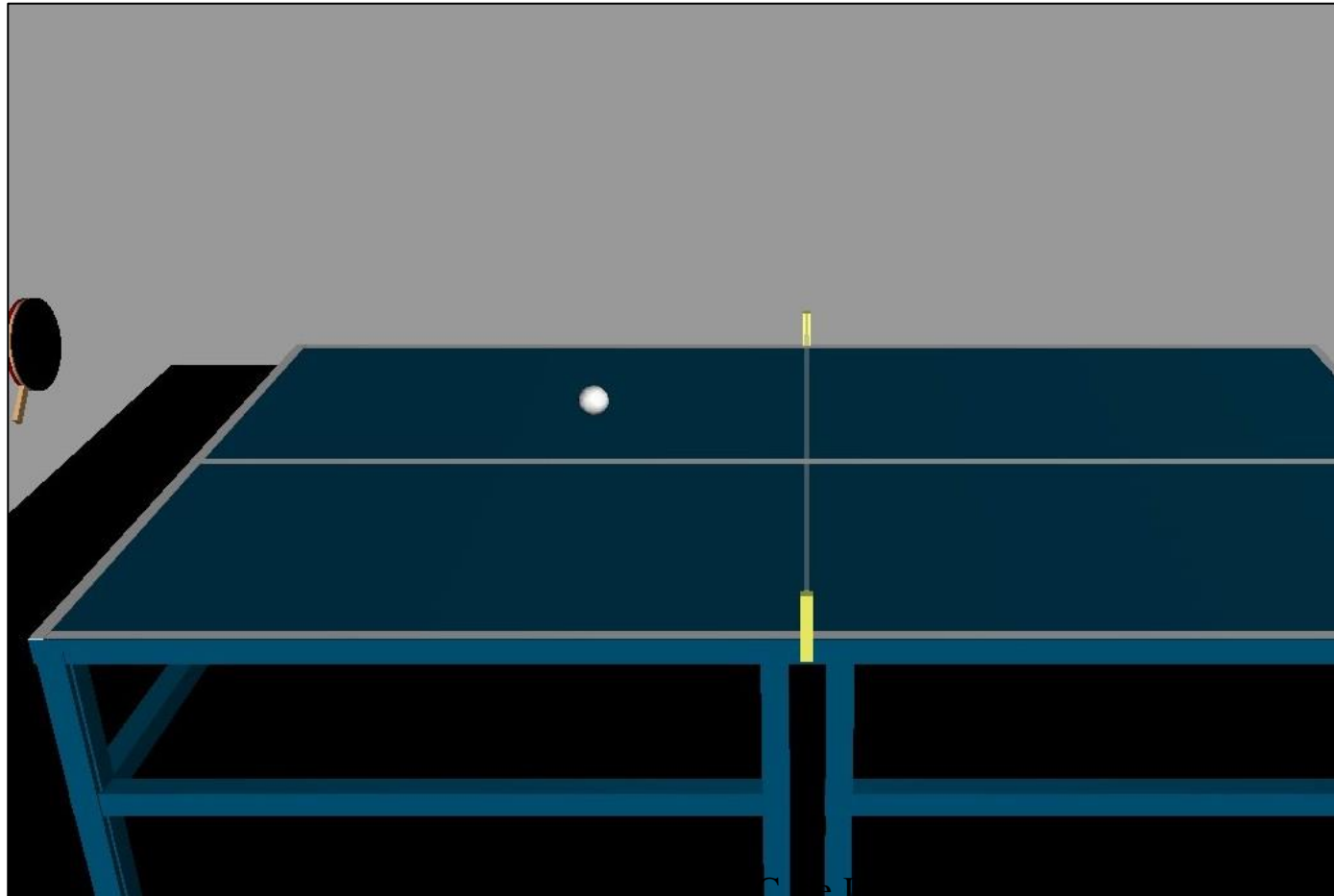


Projeto trem de pouso  
aeronave



Simulação de Funcionamento de  
Máquinas usando modelo Virtual  
AutoDesk.

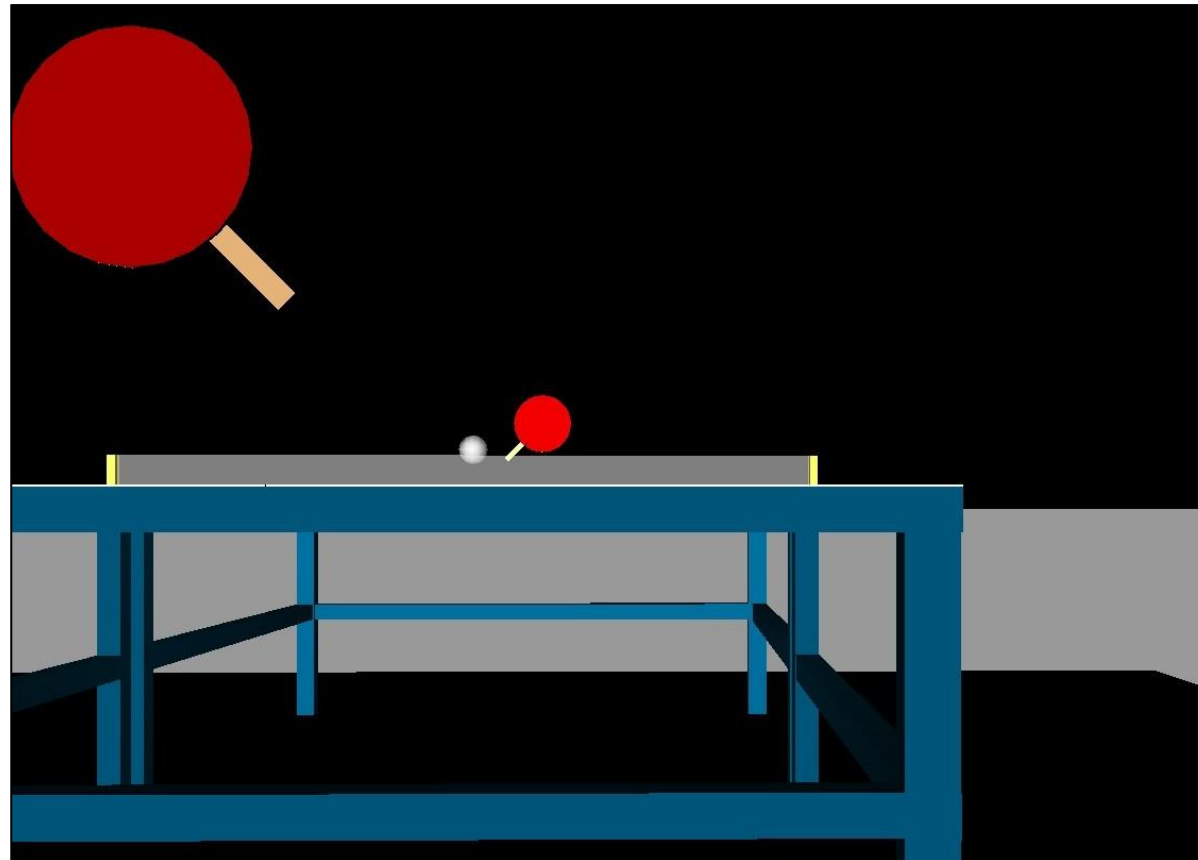
# Cena confeccionada pelos alunos André e Rafael – 2008



Prof. Wagner Santos Cruz de Jesus

[wagner@univale.br](mailto:wagner@univale.br) CRV

# Cena vista de outro ângulo

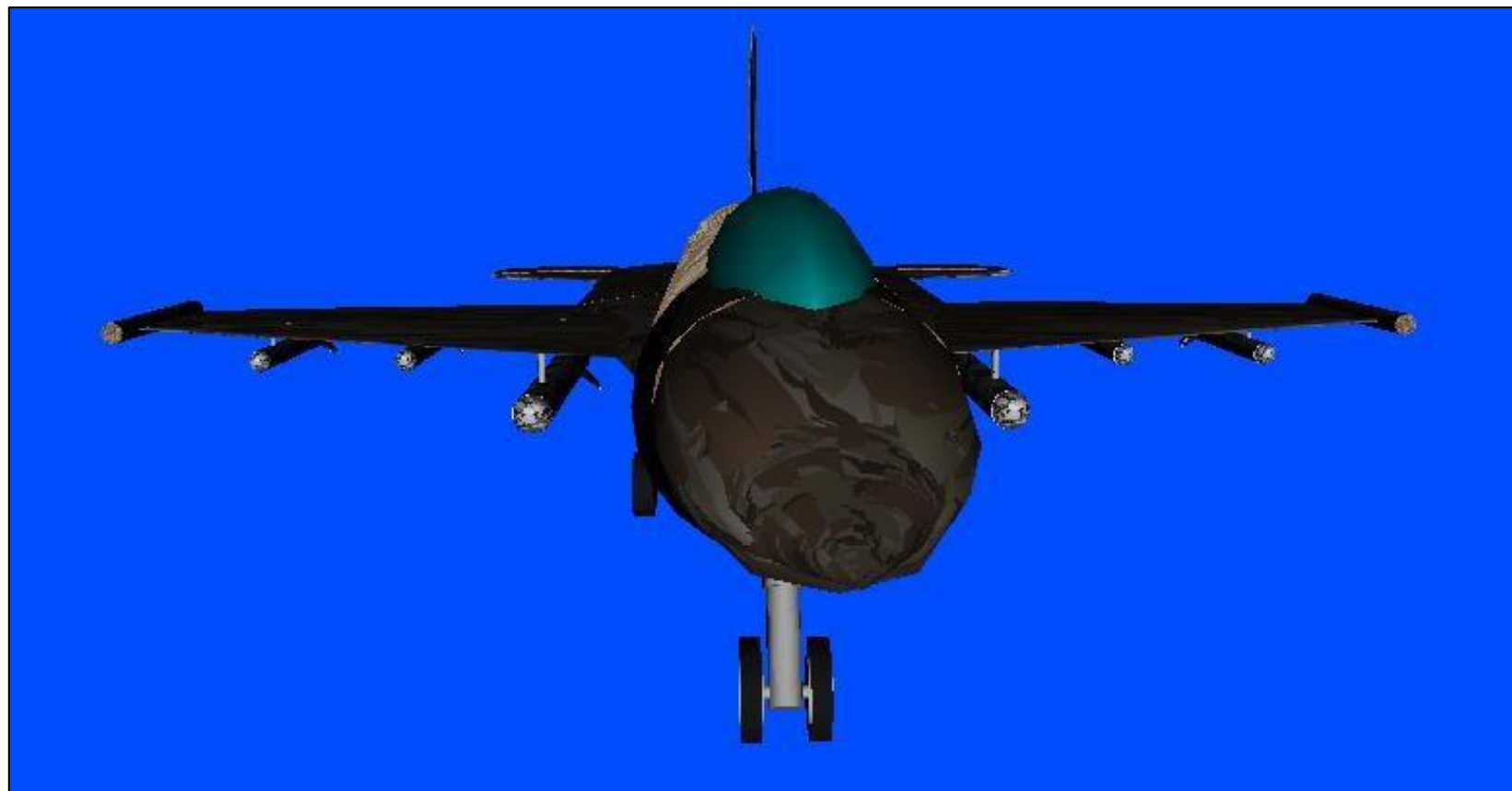


Software - CCRV

Prof. Wagner Santos C. de Jesus

[wagner@univap.br](mailto:wagner@univap.br)

# Caça F16 2009



# Caça F16 2009 (Francisco)





# Introdução aos métodos de Computação Gráfica em modo 2D.

## Primitivas Gráficas – 2D

Chamamos de primitivas gráficas os comandos e funções que manipulam e alteram os elementos gráficos de uma imagem. Também entram na definição os elementos básicos de gráficos a partir dos quais são construídos outros, mais complexos.(Hetem,2006).

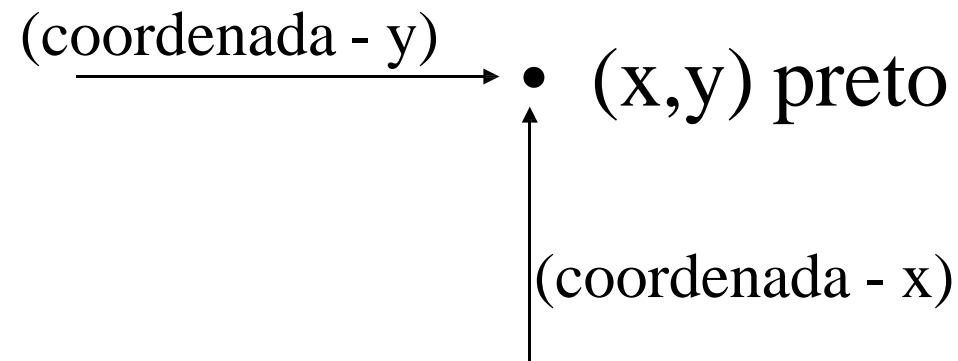
# Pontos

O ponto é a unidade gráfica fundamental e também pode ser chamada de pixel.

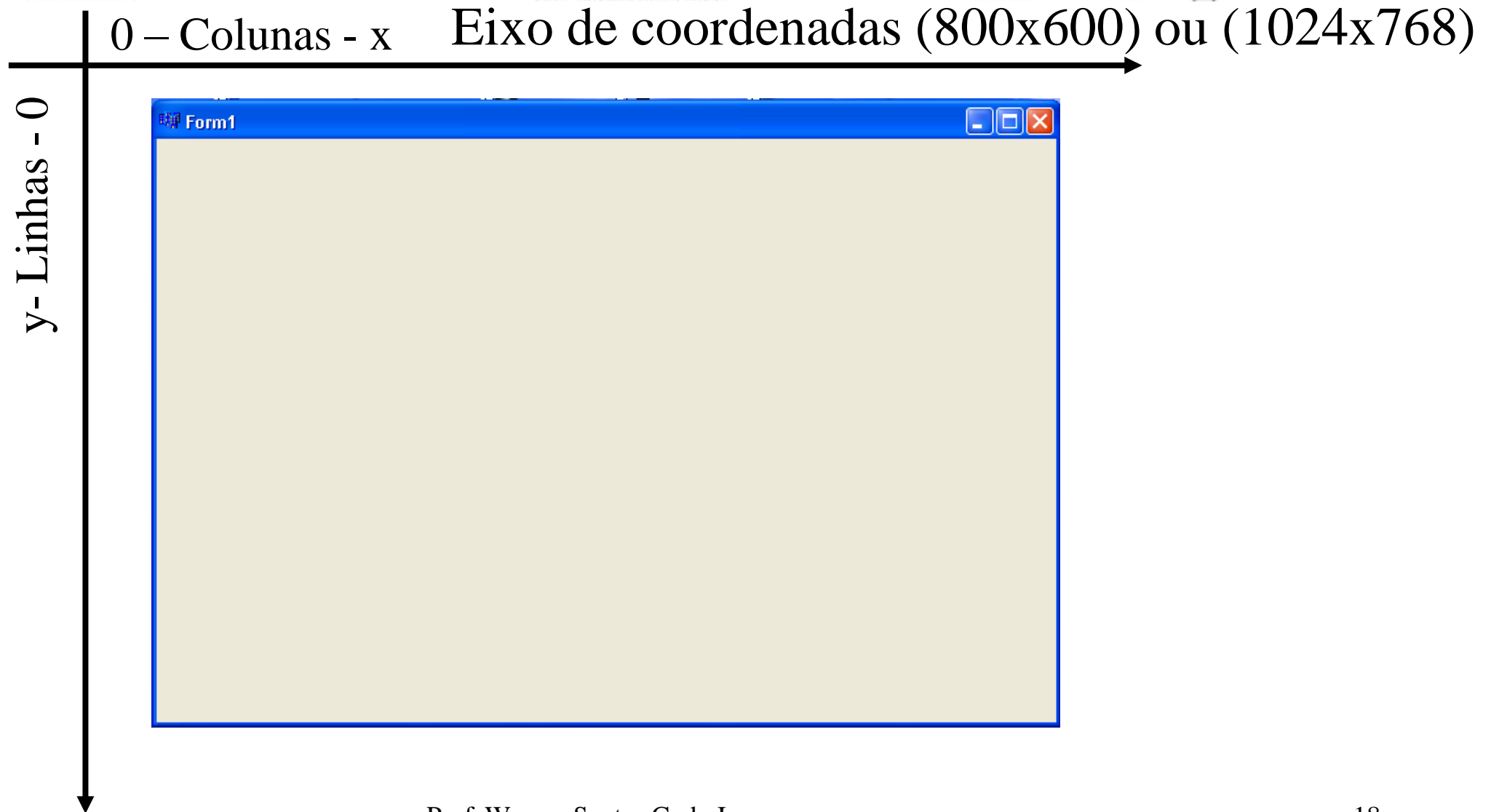
Propriedades de um pixel :

- Posição no plano gráfico (x,y)
- Cor

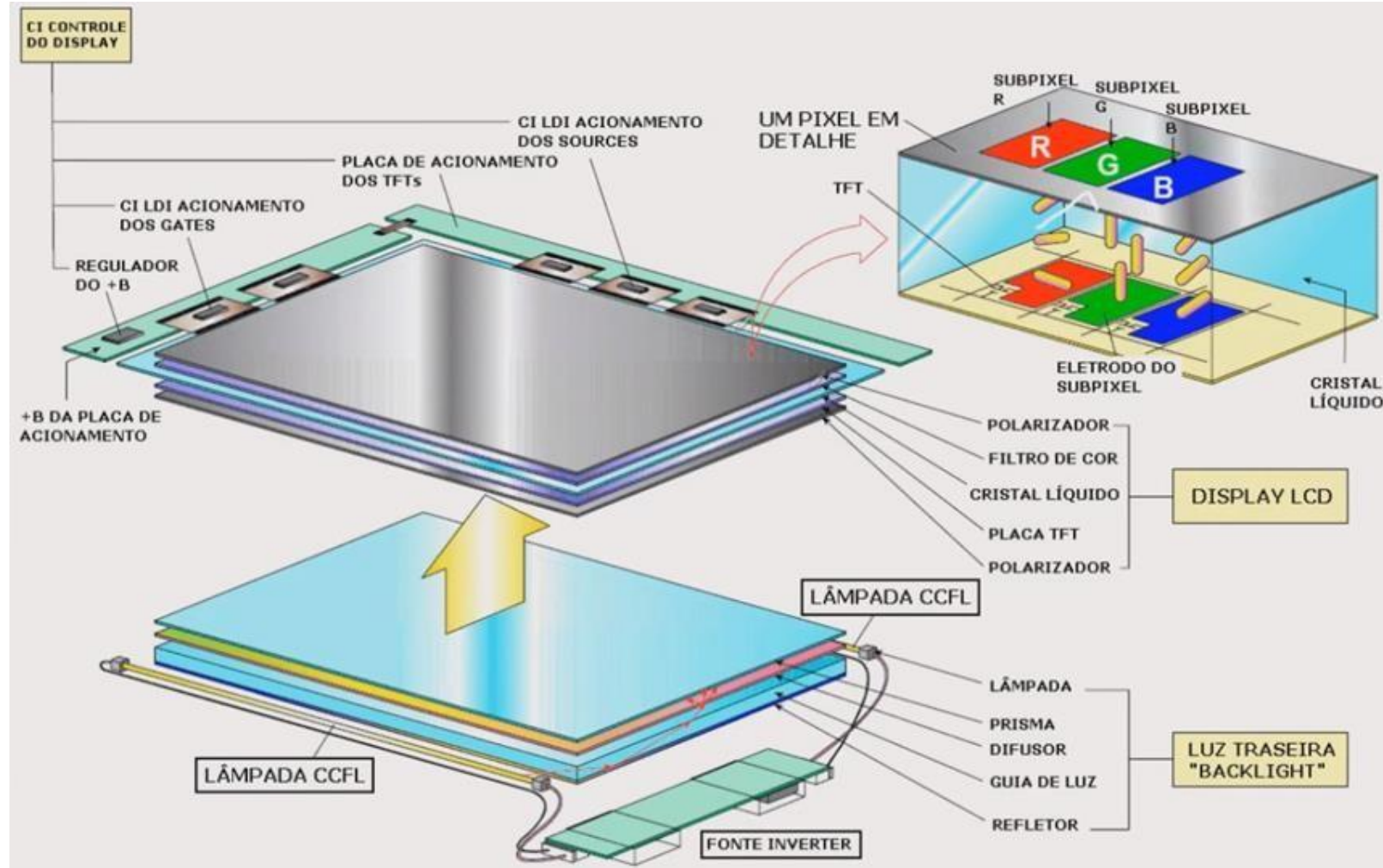
# Representação de pixel



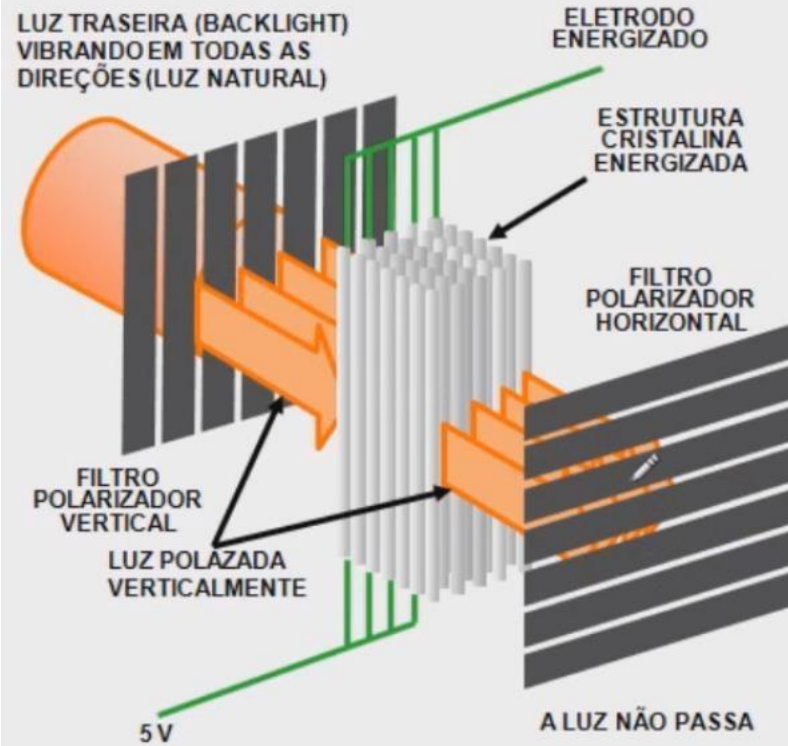
*Para se obter um pixel é necessário informar o par ordenado (x,y), que possibilita as coordenadas de linha e coluna onde será pintada a grade do vídeo; de acordo com a resolução especificada no sistema operacional.*



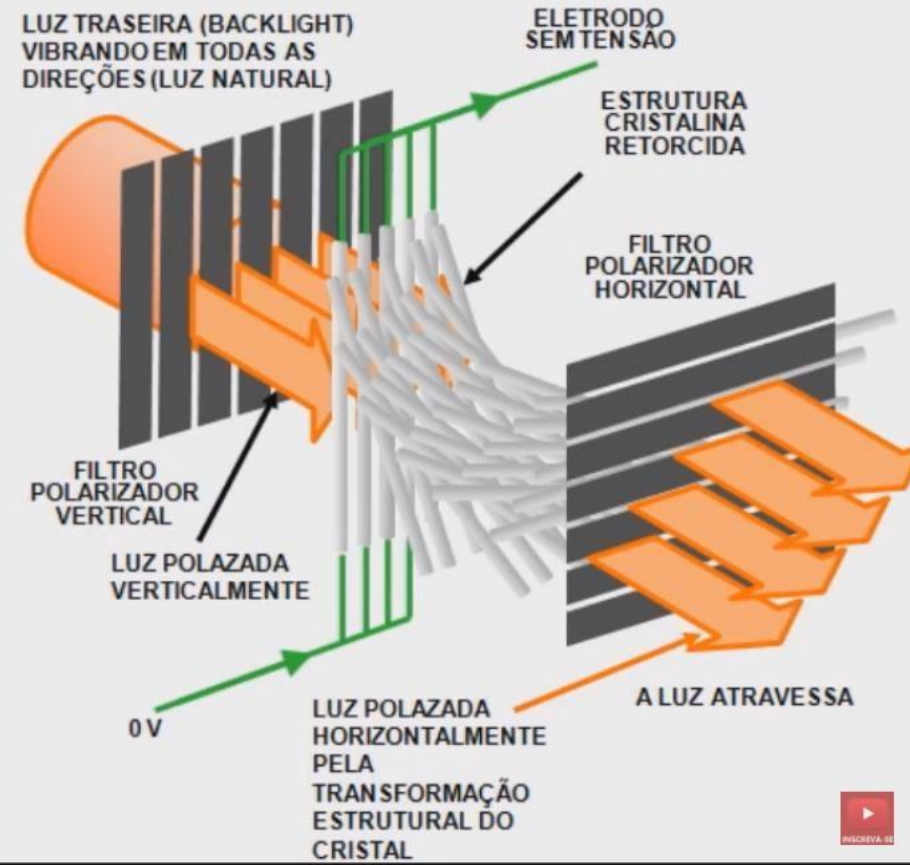


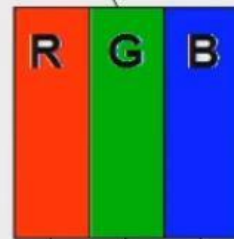
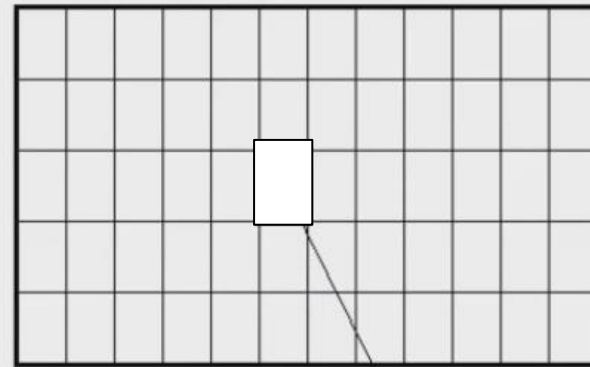


## PIXEL APAGADO



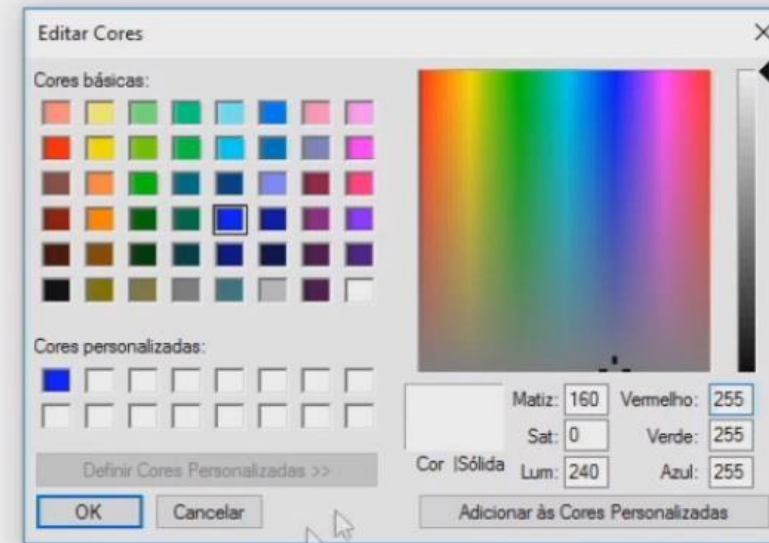
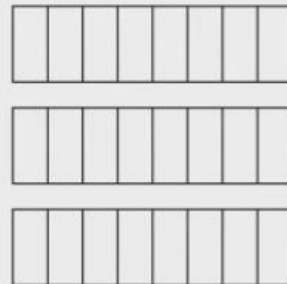
## PIXEL ACESO





PIXEL

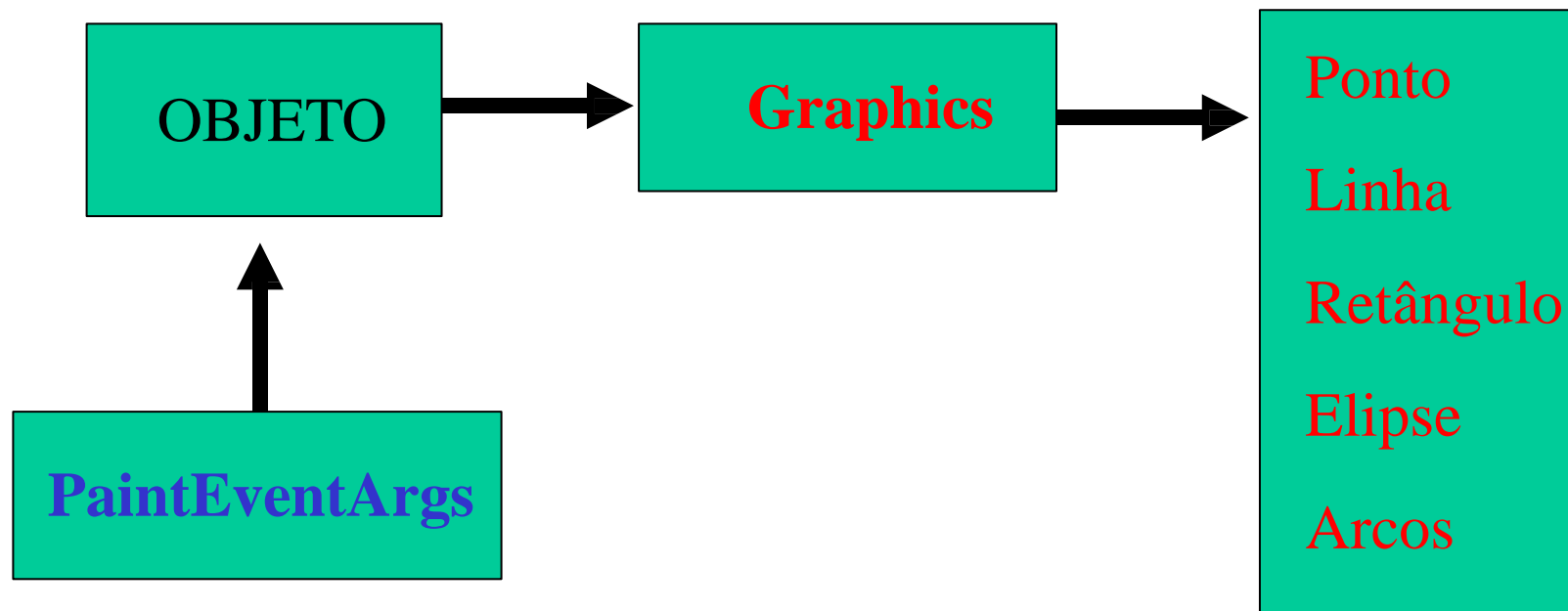
128 64 32 16 8 4 2 1



$$24 \text{ bits} = 3 \text{ bytes} \Rightarrow 2^8 = 256$$

## Pintando um pixel na prática usando C#

Para se tratar elementos gráficos podemos usar um objeto denominado **Graphics** que possibilita usar primitivas gráficas.



# Método Paint()

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    // Instruções gráficas
    // Chamadas de funções
}
```

O método Paint(), permite acessar o controlador gráfico do hardware, permitindo Reproduzir, Pontos, Linhas, Retas etc.



# Criando um ponto

Para se desenhar um ponto usa-se a propriedade **Graphics** do objeto (**PaintEventArgs**).

Exemplo:

```
e.Graphics.DrawLine(<ObjPen>, x, y, x+1, y);
```

Onde x coordenada da coluna; e y coordenada da linha a função RGB em ObjPen determina a cor do pixel que será plotado na tela.

# Objeto da Classe Color

```
Color cor = new Color();
```

```
cor = Color.FromArgb(<Alfa>, <R>, <G>, <B>);
```

Cana Alfa determina a transparência do ponto, os valores RGB indicam as intensidades.

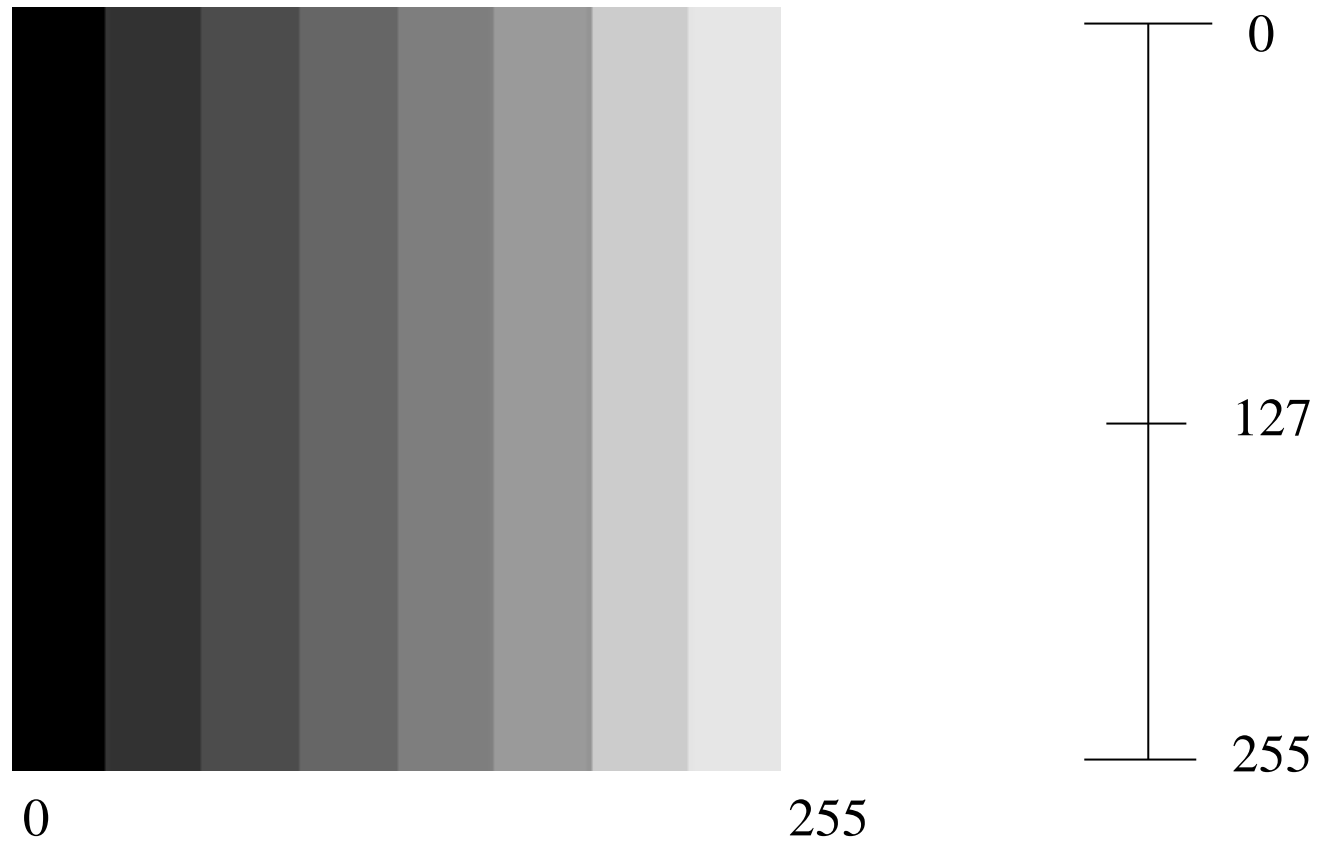
## Objeto da classe Pen

Define parâmetros da caneta que reproduz os pontos no vídeo.

Pen <Objeto> = new Pen(<cor>,<ExpN>);

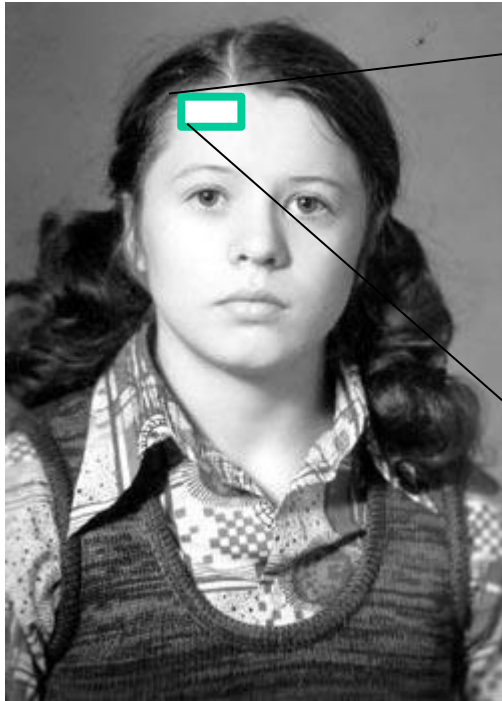
<ExpN> - Determina a espessura.

# Tons monocromáticos



Faixa de tons de cinza  $\Rightarrow C$ , portanto  $0 \leq C \leq 255$

## Intensidade – Imagens Monocromáticas



	0	1	2	3	4
0	240	240	241	240	241
1	240	240	241	241	242
2	240	241	240	240	240
3	239	241	240	240	239
4	239	240	241	240	240
5	240	241	241	241	240
6	241	241	241	241	242
7	243	242	241	241	242
8	242	242	242	241	241
9	243	242	242	241	243



## Demonstração da Tabela RGB abaixo.

### RGB(0,0,0)

R(Vermelho)	G(Verde)	B(Blue)	Cor
0	0	0	Preto
255	0	0	Vermelho
0	255	0	Verde
0	0	255	Azul
255	255	255	Branco

```
Color cor = new Color();
cor = Color.FromArgb(<ExpA>,<ExpN1>,<ExpN2>,<ExpN3>);
```

<ExpA> - Determina canal Alfa (Transparência do ponto).

<ExpN1> - Determina a Intensidade no canal R.

<ExpN2> - Determina a Intensidade no canal G.

<ExpN3> - Determina a Intensidade no canal B.

# Exemplo da criação de um ponto no vídeo - algoritmo

Pintap(Inteiro : x,y,cor)

inicio

cor = Caneta(cor);

Desenhapixels(x,y,cor);

fim

# Linhas Retas

- Para se desenhar uma reta é necessário usar primitivas gráficas, onde as primitiva vem a ser um conjunto de algoritmos para se realizar elementos gráficos mais complexos.

# Conceito Matemático de reta

$$y = mx + b$$

Onde ( $m$ ) Coeficiente angular em relação ao eixo  $x$ .

se  $m \leq 1$

*Ângulo entre  $0^\circ$  e  $45^\circ$  com eixo  $x$ , o Coeficiente linear  $b$  dá o valor do eixo  $y$ .*

se  $m \geq 1$

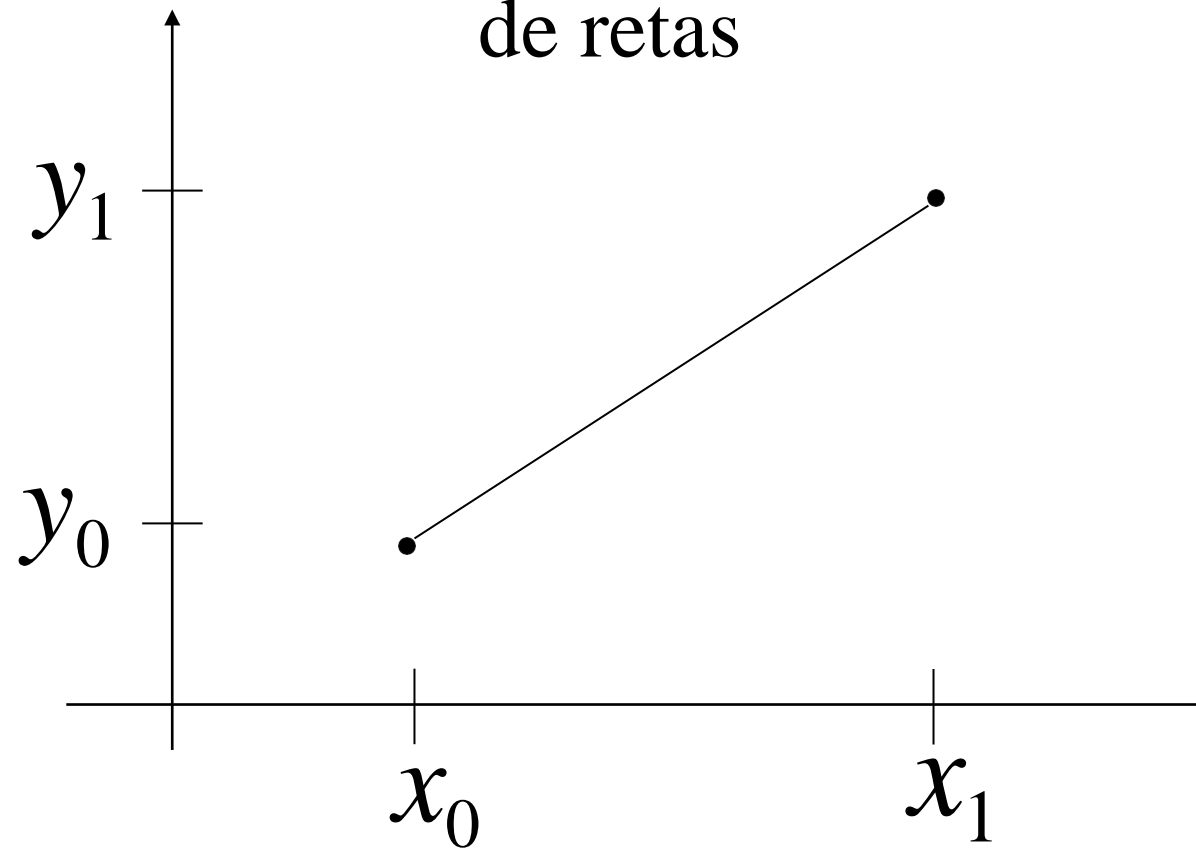
*Ângulo entre  $45^\circ$  e  $90^\circ$  com eixo  $x$ .*

Dados dois pontos no plano **P1** e **P2**, pode-se obter  $m$  e  $b$  da seguinte maneira.

$$m = \frac{y_1 - y_0}{x_1 - x_0} \quad (1)$$

$$b = y_1 - mx_1 \quad (2)$$

As formulas (1) e (2) serão base para construir os algoritmos  
de retas



## DDA – Digital Diferencial Analyser (Analizador Diferencial Digital)

Trata-se de um algoritmo que respeita as equações de coeficiente angular e linear mostrados anteriormente; Porém esse algoritmo torna-se ineficiente em alguns casos mostrando uma certa descontinuidade nas retas desenhadas.



## Algoritmo DDA (codificado)

**retaDDA( Inteiro : x0, y0, x1, y1)**

**Inicio**

**dx <- x1 - x0**

**dy <- y1 - y0**

**x. <- x0**

**y <- y0**

**s <- 0**

**se (dx > dy) entao**

**s <- dx**

**senao**

**s <- dy**

**fim\_se**

**xi. <- dx/s**

**yi <- dy/s**

**Pintap( x,y, cor)**

**Para i de 0 ate s faca**

**x <- x + xi**

**y <- y + yi**

**Pintap( x,y, cor)**

**fim\_para**

**fim**

Exemplos valores (x0,y0)-(x1,y1)

(100,100) ..... (500,100)

(100,200)

(300,600)

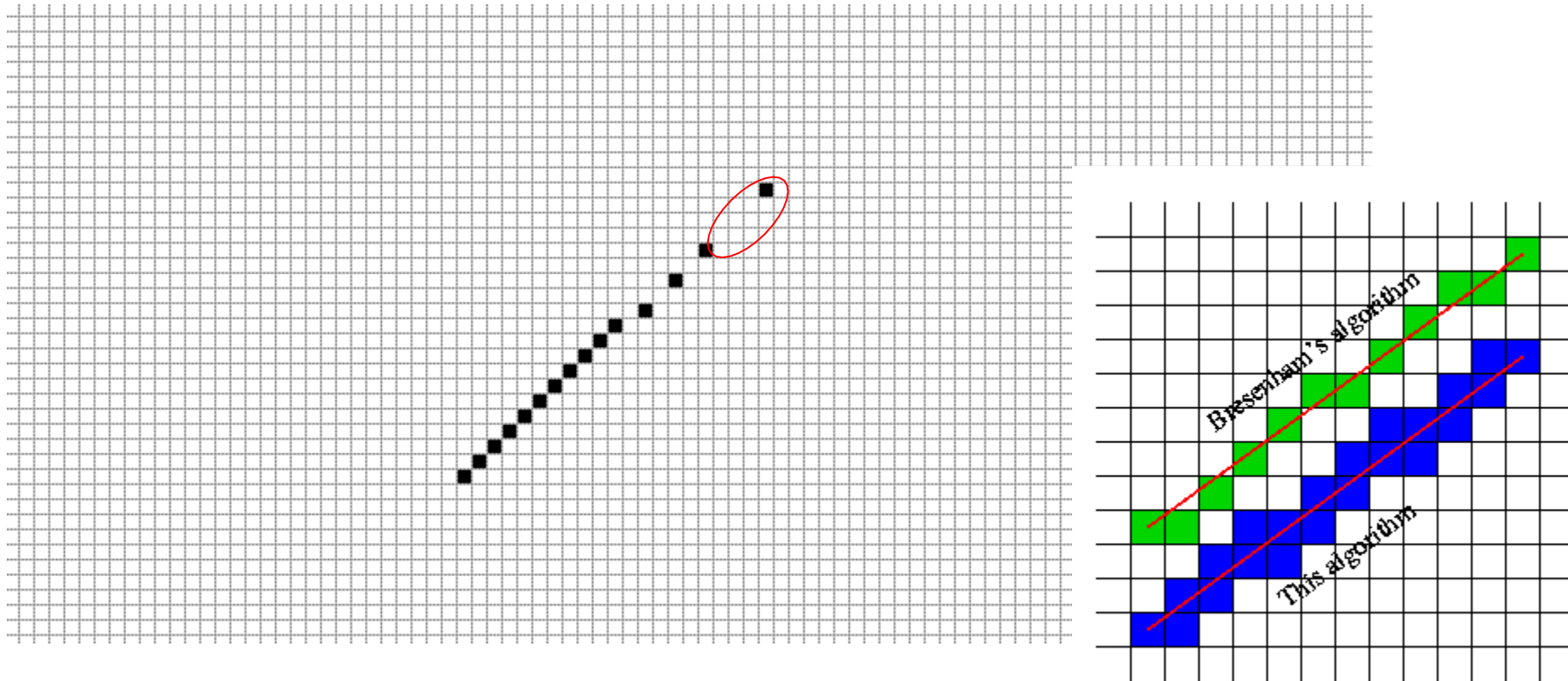
# Breseham

Esse algoritmo baseia-se no argumento de que um segmento de reta, ao ser plotado, deve ser contínuo, os pixels que compõem o segmento devem ser vizinhos; Isso fará com que os pontos das retas sejam próximos não havendo separação entre os pixels pintados.

# C#, Breseham

Em linguagens modernas esse algoritmo já se encontra implementado e melhorado. Não havendo necessidade de uma nova implementação.

# Problemas do algoritmo DDA



**retaBreseham( Inteiro : x1, y2, x2, y2)**

**x <- x1**

**y <- y1**

**dx <- x2 – x1**

**dy <- y2 – y1**

**s <- dx > dy ? dx : dy**

**Para i de 1 ate s faça**

**pintaP(x, y)**

**se(dx ≠ 0) x <- x + 1**

**se(dy ≠ 0) y <- y + 1**

**Fim-Para**

# Algoritmo

retaBreseham (inteiro:x0,y0,x1,y1,cor)

Inicio

Cor = Intensidade(Alfa,cor);

Caneta(cor,0);

DesenhaLinha(cor,x0,y0,x1,y1);

fim



# Implementação do Segmento de Reta Breseham C#

# Métodos para segmentos de retas

DrawLine() : Determinar o ponto Inicial e ponto Final para o traçado de um segmento de reta.

Exemplo :

DrawLine(<Cor>, x0,y0,x1,y1);

# Exemplo Prático

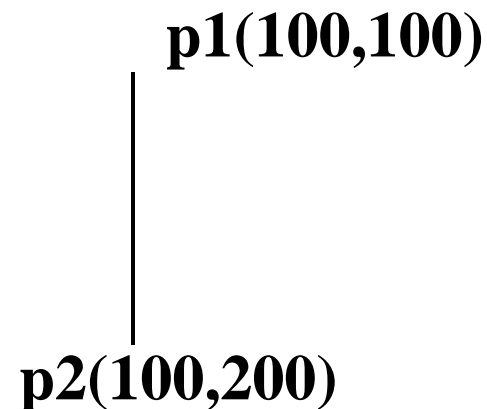
$$p_1 \in |Z|, p(x_0, y_0)$$

$$p_2 \in |Z|, p(x_1, y_1)$$

$$\left\{ \begin{array}{l} x_0 \vee x_1 > C, x_0 - 1 \\ y_0 \vee y_1 > L, y_0 - 1 \end{array} \right.$$

Onde  $C \Rightarrow$  representa o número de colunas disponíveis na tela.

$L \Rightarrow$  representa o número de linhas disponíveis na tela.



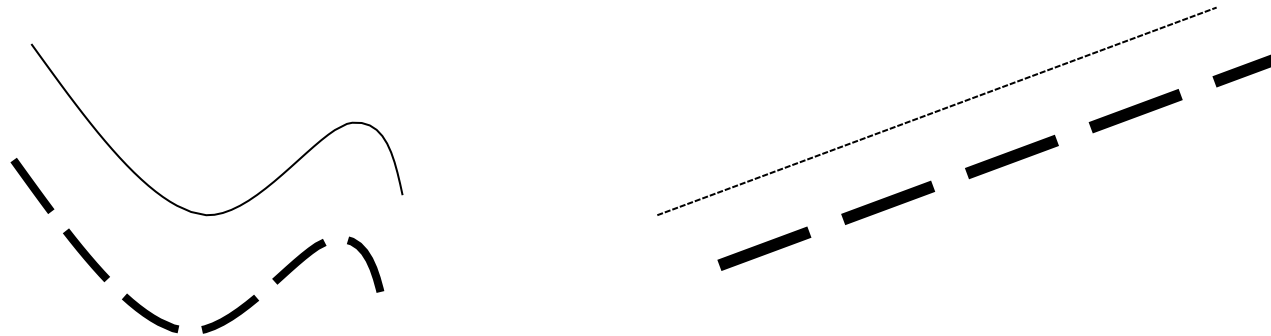
# Método: Invalidate()

Permite que seja, realizada, uma nova chamada para o método Paint().

Sintaxe:

Invalidate()

# Padrões de Multilinha



## Espessura de uma linha

Classe: **Pen** Permite ativar a caneta gráfica; determinando a espessura dada por um número inteiro <ExpN> do ponto.

Exemplo :

```
Pen caneta = new Pen(Cor, <ExpN>);
```

←  
Espessura



# Exemplo de Espessura de Linhas com cor.

## Exemplo-1:

```
Color cor = new Color();  
cor = Color.FromArgb(0, 0, 255);  
Pen pen = new Pen(cor, 0);
```

## Exemplo-2:

```
Pen pen = new Pen(Color.Black, 0);
```

# Tipos de linhas

Para usar os tipos de linhas usando objeto da classe Pen implementado em C#, usa-se as propriedades **DashPattern**; determina o tipo de linha que será usada em uma figura.

Usa-se multiplicação por escalar;  $E = \lambda v$



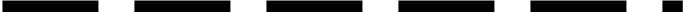


$E$  = Estilo da linha

$\lambda$  = Espessura  $\Rightarrow \lambda \in \mathbb{Z} > 0$

$v$  = Vetor com valores de seguimentos e interrupções,

$v \in \mathbb{Z} > 0$

# Estilos da Linha

Tipo	Linha
SOLID	
DASH	
DOT	
DASHDOT	
DASHDOTDOT	
NULL	

# Exemplo Prático

```
float[] flinha = { 5, 2, 15, 4 };  
Pen caneta = new Pen(Color.Black, 2);  
caneta.DashPattern = flinha;  
e.Graphics.DrawLine(caneta, 100,100,100,200);
```

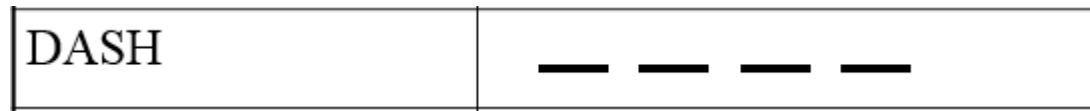
## Exemplo Prático - 2

```
float[] dash = { 5, 1 };
```

```
Pen caneta = new Pen(Color.Black, 1);
```

```
caneta.DashPattern = dash;
```

```
e.Graphics.DrawLine(caneta, 100,100,100,200);
```



## Exemplo Prático - 3

```
float[] dashdot = {5, 2, 1, 2};  
Pen caneta = new Pen(Color.Black, 1);  
caneta.DashPattern = dashdot;  
e.Graphics.DrawLine(caneta, 100,100,100,200);
```





# Criando um retângulo

Método DrawRectangle() : Cria um retângulo com a indicação do ponto e as dimensões.

Exemplo :

DrawRectangle(Pen, <ExpN1>,<ExpN2>,<ExpN3>,<ExpN4>);

<ExpN1>: Corresponde a Coluna da janela.

<ExpN2>: Corresponde a Linha da janela.

<ExpN3>: Corresponde a Largura da janela.

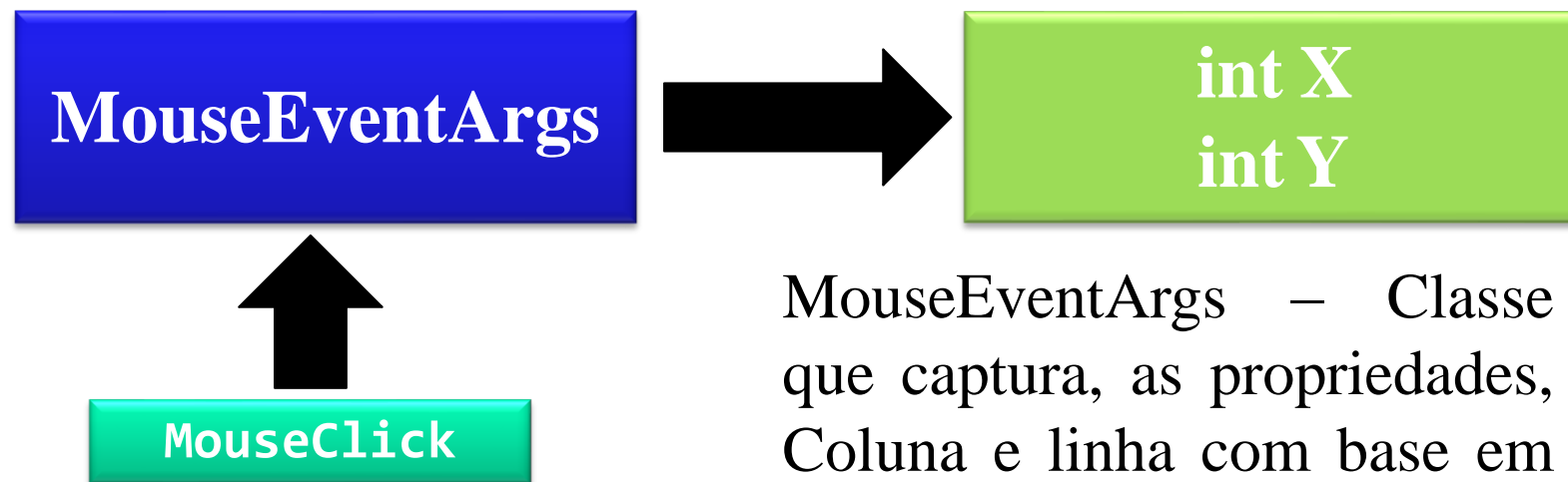
<ExpN4>: Corresponde a Altura da janela.

# Exemplo Prático

```
Color cor = new Color();  
cor = Color.FromArgb(0, 0, 255);  
e.Graphics.DrawRectangle(new Pen(cor,0), 100,50,100,50);
```

# Captura das Coordenadas do Mouse

# Classe MouseEventArgs



`MouseEventArgs` – Classe que captura, as propriedades, Coluna e linha com base em uma posição selecionada do dispositivo Mouse.

# Captura dados do objeto (e)

```
private void Form1_MouseClick(object sender, MouseEventArgs e)
{
    int x = e.X;
    int y = e.Y;
}
```

MouseClick – Executa a ação designada mediante a seleção do Mouse.

# Círculos, Elipses e Arcos

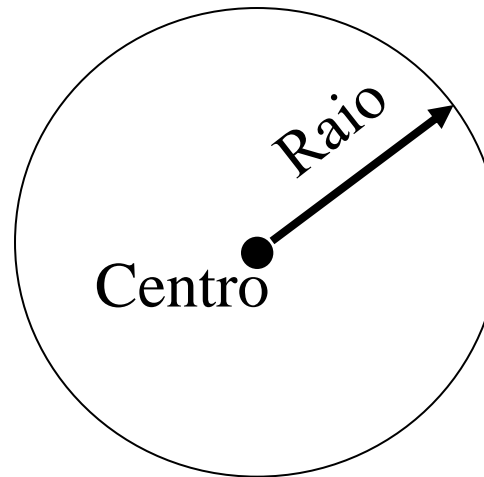
# Círculos

**Traçado de um círculo:**

**Um círculo vem a ser um conjunto de pontos que estão a uma mesma distância de um ponto; essa distância é também chamada de raio. E o ponto distante de todos os outros é o centro.**

Obtemos a seguinte definição.

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$



*Essa definição não tem como ser usada em computação gráfica.*



Definição válida para a computação gráfica.  
 **$y = f(x)$  ou  $x = g(y)$ , Isolando-se as variáveis teremos**

$$\begin{cases} x = x_c \pm \sqrt{r^2 - (y - y_c)^2} & g(y) \\ y = y_c \pm \sqrt{r^2 - (x - x_c)^2} & f(x) \end{cases}$$

Essa definição quando um segmento de círculo fica quase horizontal ou vertical um incremento e x ou y tornará o arco descontínuo.

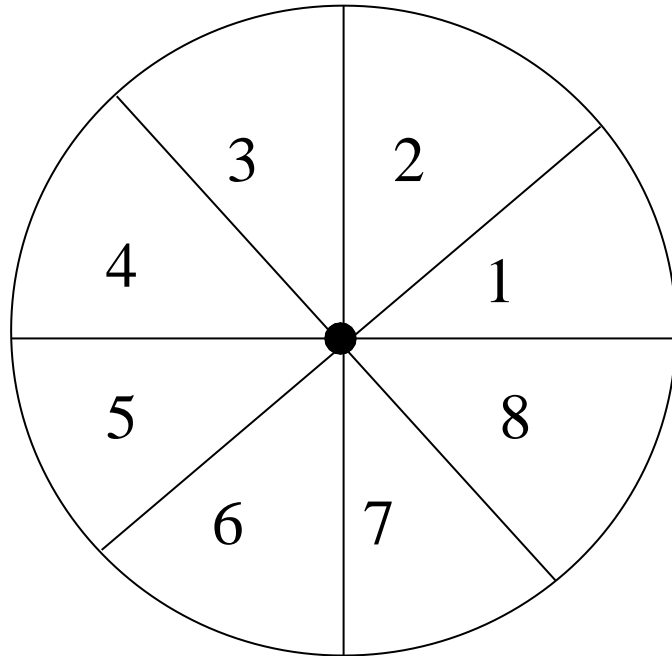
Para podermos criar um algoritmo que desenhe o seguimento circular devemos converte as expressões anteriores em coordenadas polares. Como função de raio e de um ângulo.

Funções trigonométricas:

$$\begin{cases} x = x_c + r \cos \theta \\ y = y_c + r \sen \theta \end{cases}$$

$\theta$  É um ângulo que varia entre 0 a  $2\pi$

# Precisão dependente do raio do círculo $\theta - 0 \text{ até } \pi / 4$



Octante	Xn	Yn
1	x	y
2	y	x
3	y	-x
4	-x	y
5	-x	-y
6	-y	-x
7	-y	x
8	x	-y

## Algoritmo do segmento de arcos e círculos usando funções trigonométricas sentido horário.

dArc(Inteiro: $X_c, Y_c$ , *raio*,  $T_i, T_f$ , *cor*)

Inicio

moverAte( $X_c, Y_c$ )

para teta de  $T_i$  ate  $T_f$  faça

$x \leftarrow X_c + \text{raio} * \cos(\text{teta})$

$y \leftarrow Y_c + \text{raio} * \sin(\text{teta})$

Pintap( $x, y, \text{cor}$ )

fim\_para

fim

$X_c, Y_c$  – Centro Arco ou circulo

Raio – Distância do centro nos pontos que formam o arco  $\theta$ .

$T_i, T_f$  – valores iniciais e finais de  $\theta$ .

Cor : poderá ser três variáveis inteiras para receber valores RGB.

Para forma um circulo  $\theta$  deverá estar entre 0 e  $2\pi$

# Observação

As Variáveis  $T_i$  e  $T_f$ , serão iniciadas com valores em graus que deverão ser transformada em radiano no momento de sua implementação; que será equivalente a teta.

$$\theta = \alpha \frac{\pi}{180^\circ}$$

*O algoritmo apresentado anteriormente conhecido como funções polares; irá apresentar falha de precisão quando se aumentar o raio podem tornar imprecisões no traçado do círculo ou arco; Para resolver esse problema existe um algoritmo conhecido como ponto médio para círculos. Esse algoritmo já se encontra implementado em C#.*

# Elipses

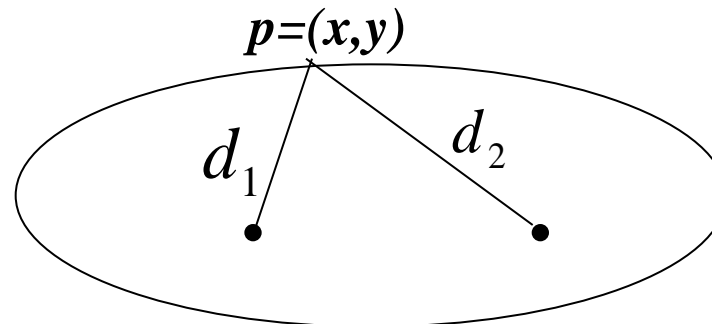
## Traçado de uma Elipse:

Uma elipse é definida como um conjunto de pontos cuja soma das distâncias para dois pontos fixos é constantes. Onde os dois pontos fixos são chamados de foco da elipse. Sua definição matemática é:

$$d_1 + d_2 = \sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2}$$

Onde  $(x_1, y_1)$  e  $(x_2, y_2)$  São as posições dos focos,  $d_1$  e  $d_2$  São as do ponto **P** distância até os focos.

Essa definição não tem como ser usada em computação gráfica.



Para se obter um algoritmo que satisfaça as definições em computação gráfica.

Se faz necessário uma definição  $f(x)$  e  $g(y)$ .

*Coordenada polar*

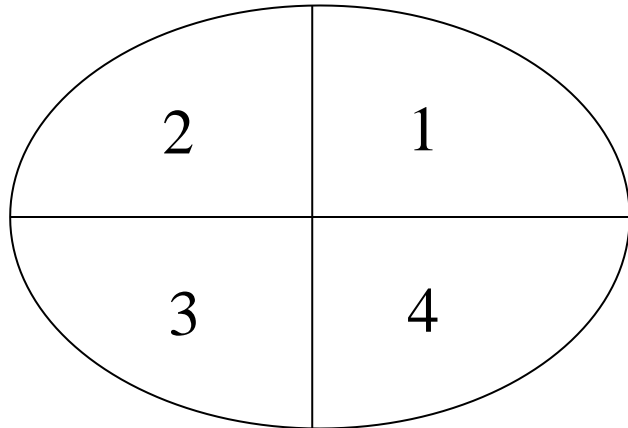
$$\begin{cases} y = y_c + r_x \cos\theta \\ x = x_c + r_y \sen\theta \end{cases}$$

$r_x$  e  $r_y$  referem-se aos raios da elipse na direção x e y,  $(x_c, y_c)$  é o centro da elipse.  $\theta$  é um ângulo que varia entre 0 e  $2\pi$

O número de passos deverá aumentar com o raio.



# Quadrante de simetria numa elipse



Quadrante	$X_n$	$Y_n$
1	$x$	$y$
2	$-x$	$y$
3	$-x$	$-y$
4	$x$	$-y$

Algoritmo do segmento de arcos para confecção da elipse  
usando funções trigonométricas sentido horário.

dEllipse(Inteiro: $X_c, Y_c, R_x, R_y, T_i, T_f, cor$ )

Início

moverAte( $X_c, Y_c$ )

para teta de  $T_i$  ate  $T_f$  faça

$x \leftarrow X_c + R_y * \cos(\text{teta})$

$y \leftarrow Y_c + R_x * \sin(\text{teta})$

Pintap( $x, y, cor$ )

fim\_para

fim

$X_c, Y_c$  – Centro Arco

$R_x, R_y$  – Raio da elipse  
na direção x e y.  $R_x < R_y$

$T_i, T_f$  – valores iniciais e  
finais de  $\theta$

Cor : poderá ser três  
variáveis inteiras para  
receber valores RGB.

Para forma um circulo  $\theta$  deverá estar entre 0 e  $2\pi$

*O algoritmo apresentado anteriormente conhecido como funções polares; irá apresentar falha de precisão quando se aumentar o raio podem tornar imprecisões no traçado do círculo ou arco; Para resolver esse problema existe um algoritmo conhecido como ponto médio para Elipse.*

*Esse algoritmo já se encontra, implementado em C#.*

# Ponto Médio para Elipse

Método : `Ellipse()` desenha uma elipse nas coordenadas especificadas.

Exemplo :

`e.Graphics.DrawEllipse(Pen, Xc, Yc, Lx, Ay);`

Pen: Objeto Caneta

Lx = Largura da região rectangular

Ay = Altura da região rectangular

$Lx = Ax = \text{Cículo}$

$Lx \neq Ax = \text{Elipse}$

# Aplicação de método para desenho de uma Ellipse

Exemplo:

```
Color cor = new Color();
```

```
cor = Color.FromArgb(255, 0, 0);
```

```
e.Graphics.DrawEllipse(new Pen(cor, 0), 100, 100, 200, 100);
```

# Conceito de Preenchimento de Áreas

# Preenchimento de áreas (Sólidas)

Ao se desenhar um objeto além de seu contorno o mesmo poderá possuir cores em seu interior podendo ser visualizadas. Em C# encontramos implementados esse algoritmos o que normalmente se usa é o preenchimento recursivo.

# Propriedade Brush, Color

Permite realizar o preenchimento em um determinado objeto retângulo, círculo etc.

Exemplo :

```
SolidBrush fundo = new SolidBrush(Color.Blue);  
Pen pen = new Pen(Color.Black, 0);  
e.Graphics.DrawEllipse(pen, 100, 100, 200, 100);  
e.Graphics.FillEllipse (fundo, 100, 100, 200, 100);
```



# Preenchimento com Hachura

## Preenchimento Hachura (HatchBrush)

Para se confeccionar o modelo de hachura, usa-se um objeto da classe **HatchBrush**.

Sinaxe:


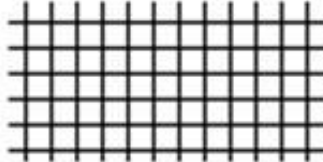
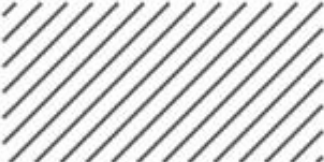
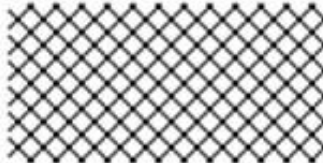


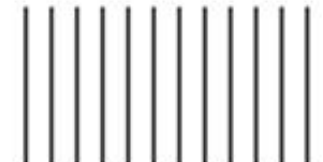
```
HatchBrush preech = new HatchBrush(<TipoH>, <CorL>, <CorF>);
```

<TipoH>: Tipo de Hachura a ser aplicado.

<CorL> : Cor das linhas da Hachura.

<CorF> : Cor de preenchimento sólido de fundo.

# Tabela da propriedade : Hachuras

Value	Pattern	Value	Pattern
Solid	(*) 	Cross	
BackwardDiagonal		DiagonalCross	
ForwardDiagonal		Horizontal	
		Vertical	

Obs: São preenchimentos válidos apenas para o objeto  
Métodos disponíveis no controlador Gráfico.

# Observação

**Para que a classe HatchBrush, funcione corretamente é necessário, acrescentar o pacote.**

**using System.Drawing.Drawing2D;**

**No cabeçalho do programa.**

# Exemplo prático de Aplicação da Hachura

```
HatchBrush hashura = new HatchBrush(  
    HatchStyle.Vertical, // HatchStyle.DarkDownwardDiagonal,  
    Color.Red,  
    Color.FromArgb(128, 255, 255));  
  
e.Graphics.FillEllipse(hashura, 0, 0, 100, 60);
```

# Aplicação de Imagens como Textura

## Usando Textura

Podemos preencher uma figura usando um arquivo com uma imagem ou desenho.

Para realizar essa técnica usa-se um objeto da classe **Bitmap**, encapsulado a classe **TextureBrush**.

Exemplo :

```
Bitmap <ObjBitmap> = new Bitmap(<Caminho>);
```

# Criação do Objeto da Classe TextureBrush

Permite encapsulamento do objeto contendo a imagem que tomará a forma geométrica da figura desenhada.

Exemplo :

```
Bitmap bitmap = new Bitmap("Textura.jpg");  
TextureBrush tBrush = new TextureBrush(bitmap);
```



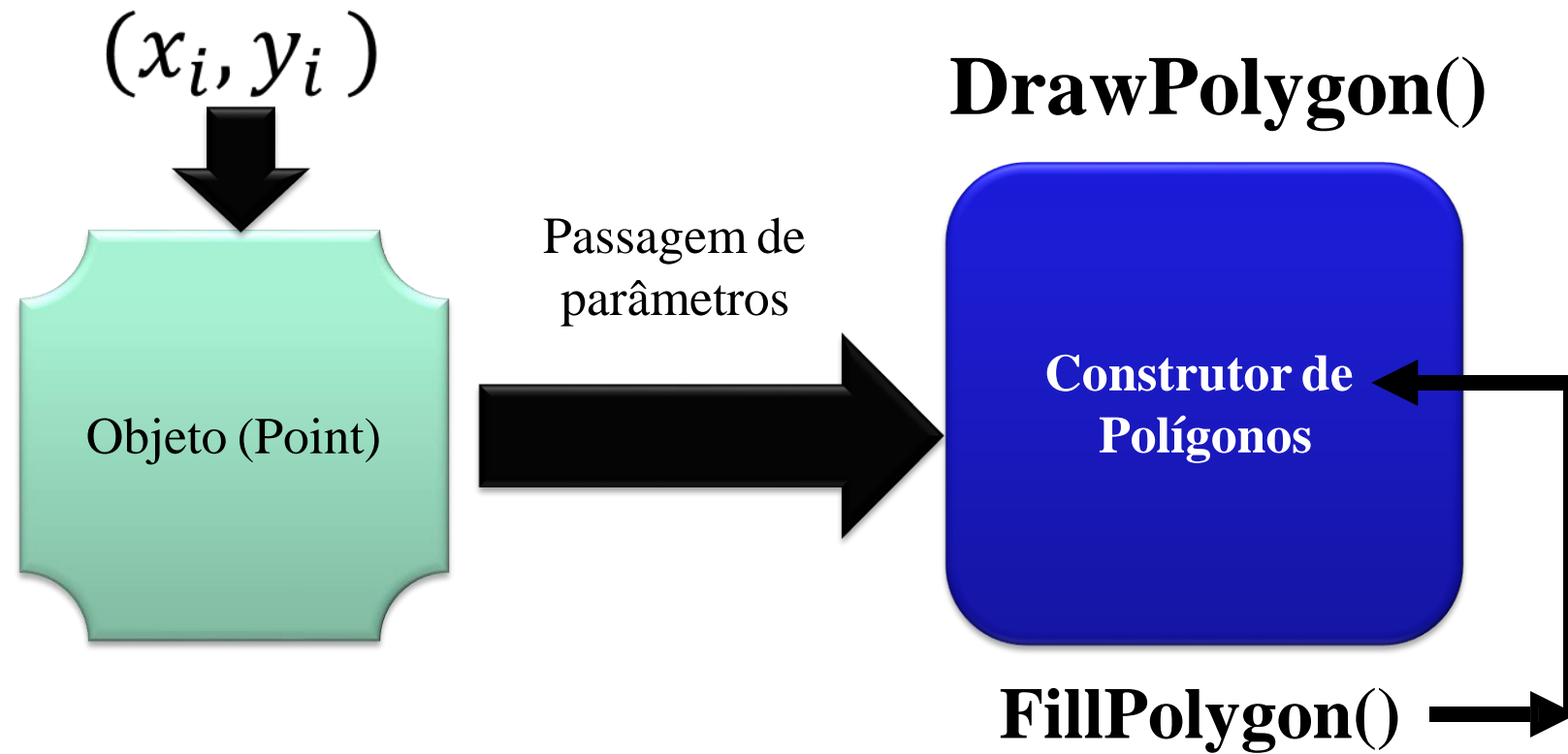
## Exemplo :

```
Bitmap bitmap = new Bitmap("Textura.jpg");  
TextureBrush tBrush = new TextureBrush(bitmap);  
e.Graphics.FillEllipse(tBrush, 100, 20, 200, 100);
```



# Criação de Formas usando Conceito de Pontos e Polígonos

# Classe geradora de Polígonos



# Classe - Point

Permite define encapsular uma quantidade de pontos, encapsulando-os em um objeto, permitindo sua utilização posterior.

Exemplo: `Point <objeto> = new Point(<X>,<Y>);`

X – Corresponde a coluna da tela.

Y – Corresponde a linha da tela.

<objeto> - Elemento que encapsula os pontos enviados.

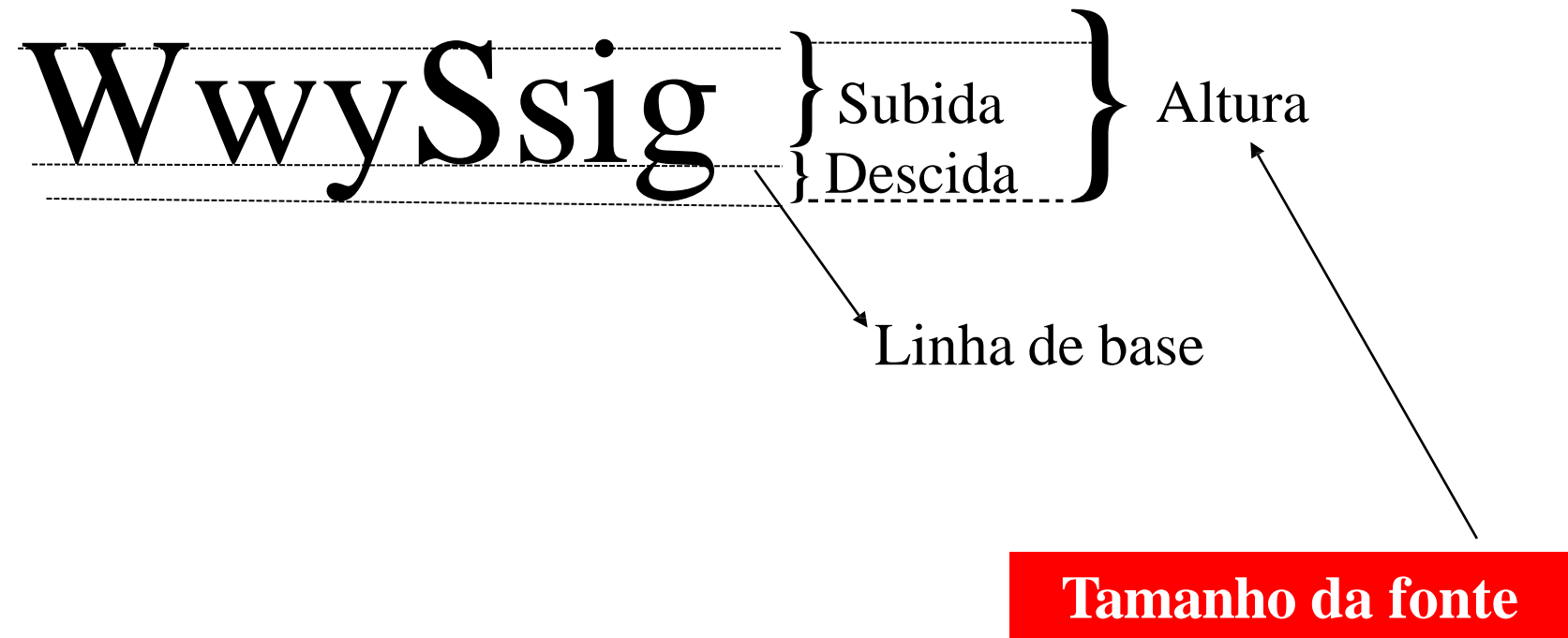
# Exemplo da criação de Polígono

```
Pen caneta = new Pen(Cor,2);  
int[] x = { 100, 200, 200, 100 };  
int[] y = { 100, 100, 300, 300 };  
Point[] pontos = new Point[4];  
for(int i=0;i<=3;i++){  
    Point point1 = new Point(x[i], y[i]);  
    pontos[i] = point1;  
}  
e.Graphics.DrawPolygon(caneta, pontos);
```

# Preenchimento de Figuras

```
SolidBrush fundo = new SolidBrush(cor);  
e.Graphics.FillPolygon(fundo, pontos);
```

# Desenhando Graficamente um texto



# Criação de textos

`DrawString()`: Permite desenhas um texto nas coordenadas especificadas.

Sintaxe: `DrawString(<Texto>,<oFont>,<oCor>,x,y);`

`<Texto>`: String a ser impressa.

`<oFont>`: Objeto com formato da fonte do texto.

`<oCor>`: Objeto com a cor das letras.

x e y: Coordenadas de coluna e linha referentes a posição do texto.



Font(): determina o tipo de fonte para as letras gráficas.

Sintaxe:

Font oFont = new Font(<ExpS>,<ExpN>,<oStyle>);

<ExpS>: Tipo da fonte a ser atribuída a um objeto.

<ExpN>: Tamanho da fonte a ser atribuída.

<oStyle>: Estilo da fonte.

# Tabela de estilo de fonte (FontStyle)

Estilo	Descrição
Bold	Negrito
Italic	Itálico
Regular	Formato tradicional
StrikeOut	Atachado
Underline	Sublinhado

# Exemplo DrawString, Font

```
String str = "Bom Dia";  
Font font = new Font("Arial", 16, FontStyle.Bold );  
SolidBrush corletra = new SolidBrush(Color.Black);  
float x = 150.0F;  
float y = 150.0F;  
e.Graphics.DrawString(str, font, corletra, x,y);
```

## Exemplo de escrita de caracteres em imagens

```
Bitmap img = new Bitmap(imagem);  
Graphics desenha = Graphics.FromImage(img);  
String str = "Bom Dia";  
Font font = new Font("Arial", 16, FontStyle.Bold );  
SolidBrush brush = new SolidBrush(Color.White);  
desenha.DrawString( texto, font, brush, 100, 245);  
img.Save(imagensaida);
```



# Transformação Geométrica

## Transformações Geométricas – 2D

*Chamamos de transformação o ato de levar um objeto de um ponto para outro, num sistema de referências. As transformações mais usadas são :*

- *Translação*
- *Rotação*
- *Escala.*

# Translação

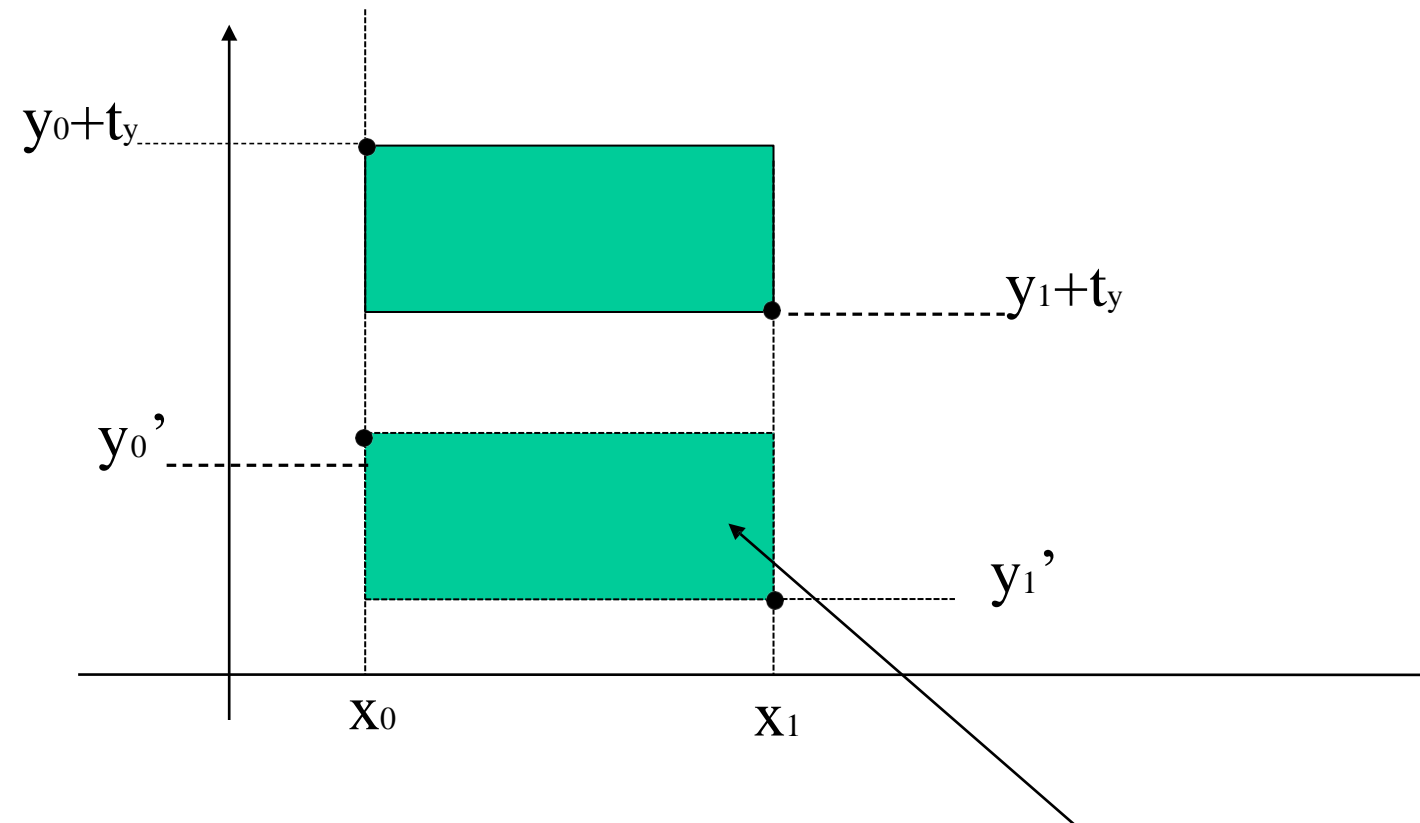
Chamamos de translação o ato de levar um objeto de um ponto para outro, num sistema de referência. Dado um ponto original  $(x,y)$  ; existe um ponto  $(x',y')$  translação, que corresponde ao ponto original dado por:

$$x' = x + t_x$$

$$y' = y + t_y$$

$(t_x, t_y)$  – É chamado de vetor de translação ou vetor de deslocamento  $t_x$  quantos pixels a figura está deslocada na horizontal e  $t_y$  quantos pixels a figura esta deslocada na vertical.

# Aplicação de uma translação



Translação



# Conceito de desenho do retângulo

```
Pen Pen = new Pen(cor, 0);
```

```
e.Graphics.DrawRectangle(Pen, x1,y1,x2,y2);
```

<x1> - Coluna da janela.

<x2> - linha da janela.

<x2> - Largura da figura.

<y2> - Altura da figura.

# Implementação de uma translação

```
int x0 = 10; int y0 = 10;
```

```
int x1 = 100; int y1 = 50;
```

```
ty = 70; ← Vetor de translação horizontal
```

```
Retangulo(x0,y0,x1,y1 );
```

```
Retangulo(x0,y0+ty,x1,y1+ty );
```

# Problema Proposto

Implementar uma função para desenhar um retângulo e em seguida usando conceito de primitiva implementar uma função para desenhar um quadrado.

# Rotação

Dá-se o nome de rotação ao ato de girar um objeto de um ângulo, num sistema de referência.

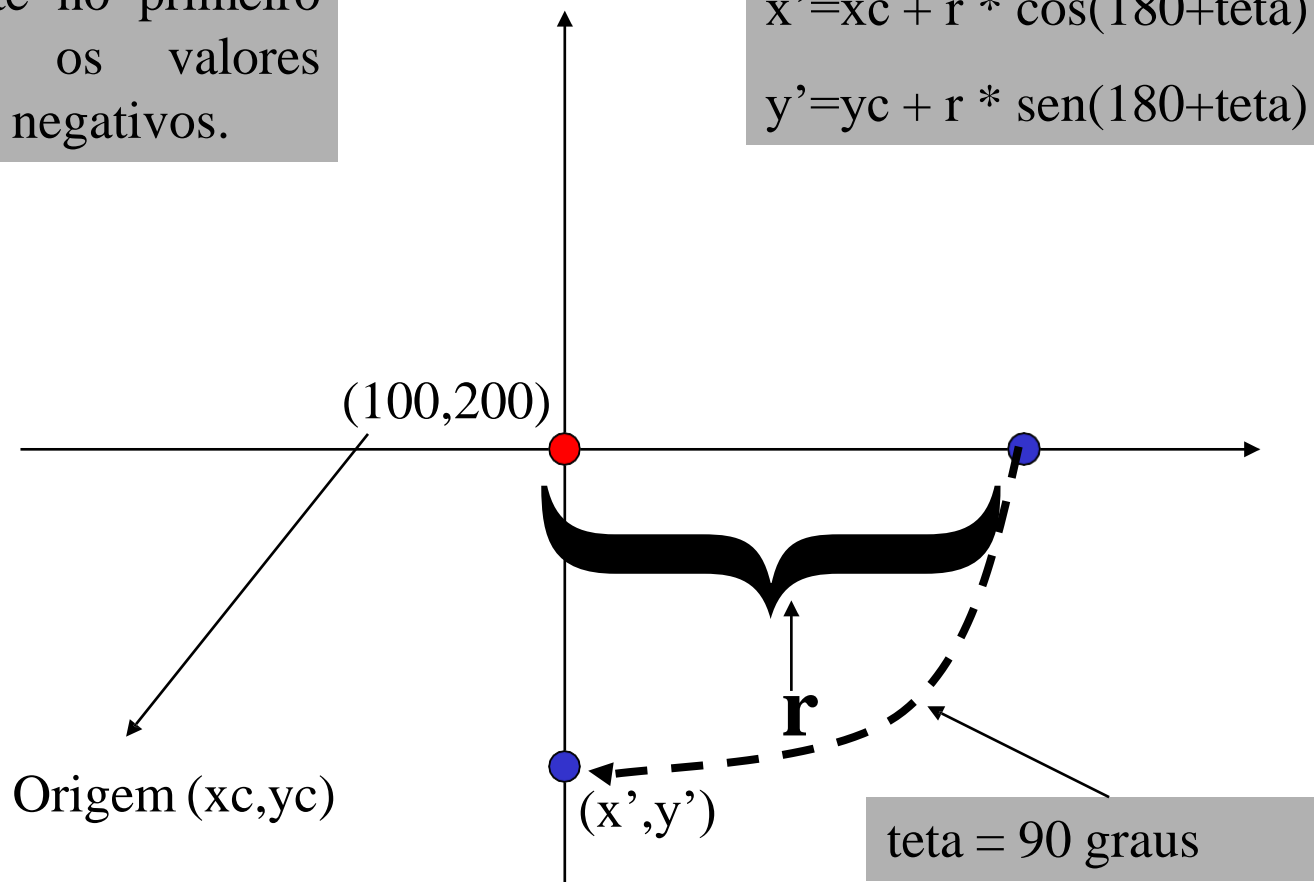
$$\begin{cases} x' = x \cos \theta - y \sin \theta \\ y' = x \sin \theta + y \cos \theta \end{cases}$$

# Exemplo de Aplicação de Rotação em torno de um eixo.

Para ajuste no primeiro quadrante os valores devem ser negativos.

$$x' = x_c + r * \cos(180 + \text{teta})$$

$$y' = y_c + r * \sin(180 + \text{teta})$$



## Implementação em C# de Rotação em torno de um eixo de referência

```
Font font = new Font("Arial", 16, FontStyle.Bold);  
SolidBrush corletra = new SolidBrush(Color.Black);  
float x = 150.0F;  
float y = 150.0F;  
int raio = 50;  
  
e.Graphics.DrawString(".", font, corletra, x, y);  
int x1 = (int)(100 + raio * Math.Sin(180 * 3.14 / 180));  
int y1 = (int)(200 + raio * Math.Cos(180 * 3.14 / 180));  
SolidBrush corletra1 = new SolidBrush(Color.Red);  
e.Graphics.DrawString(".", font, corletra1, x1, y1);
```

## Técnica de rotação usando a transposição de matriz $I^t$

$I(x,y)$

1,1

0,2

1,2

2,2

*Matriz Original*

$I(y,x)^t$

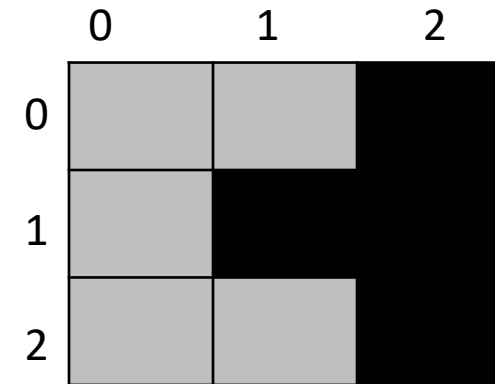
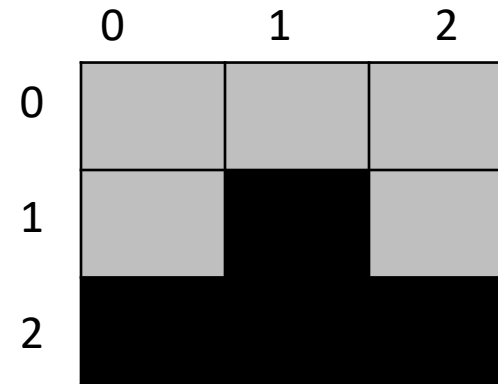
1,1

2,0

2,1

2,2

*Matriz transposta*



# Escala

Quando se aplica uma transformação de escala a um objeto, o resultado é um novo objeto semelhante ao original, mas “esticado” ou “encolhido”. Pode-se aplicar escala com valores diferentes para cada dimensão; por exemplo, dobrar um objeto na direção horizontal e dividir na vertical.



# Formalização da Escala

Uma escala é determinada pelo vetor de escala  $(S_x, S_y)$ . Onde  $S_x$  é a escala aplicada na direção x, e  $S_y$  é a escala aplicada na direção y.

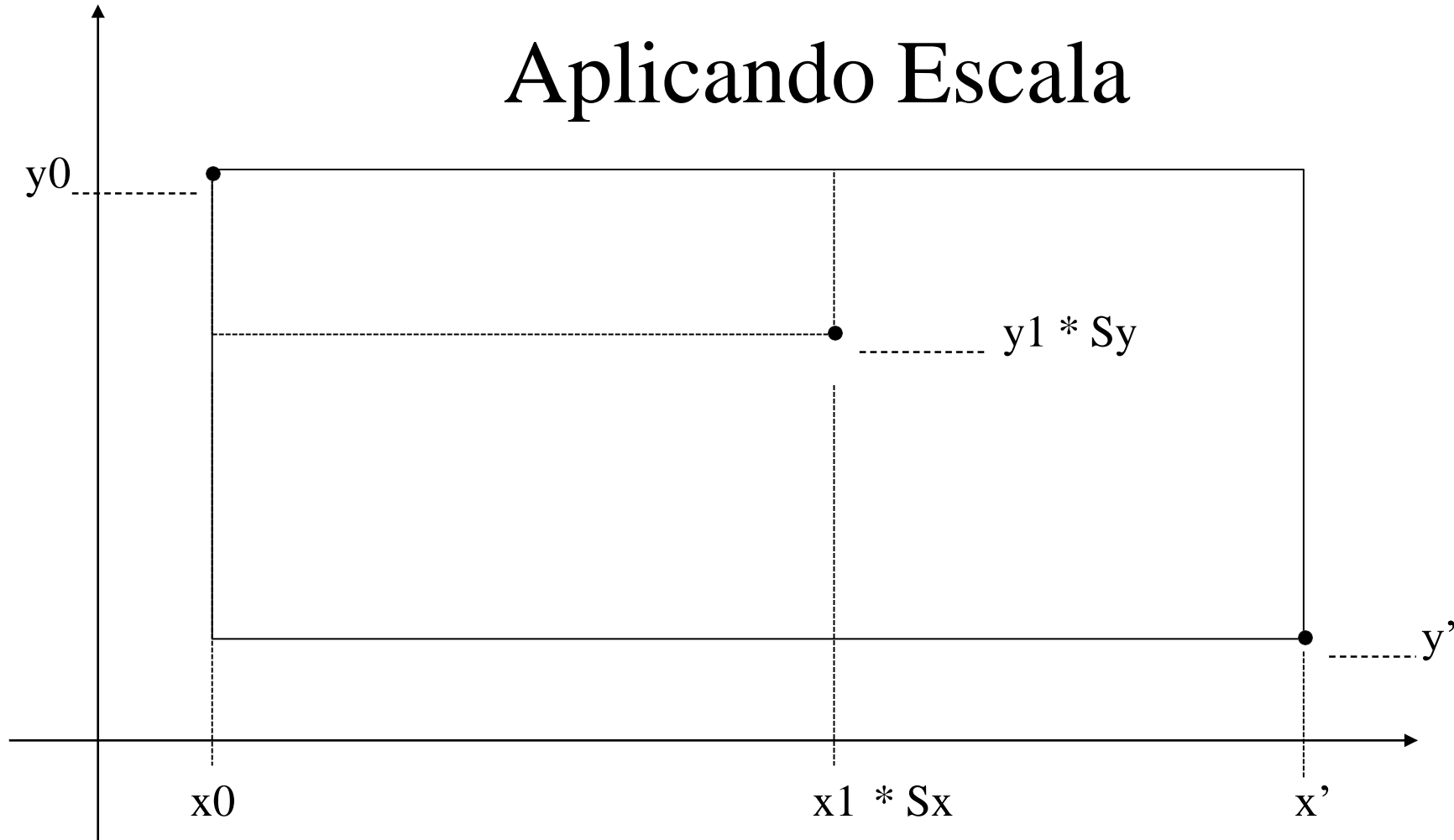
Equação :

$$\begin{cases} x' = xS_x \\ y' = yS_y \end{cases}$$

## Implementação de escala em uma figura.

```
int x0=100; int y0=200;  
int x1=300; int y1=250;  
int sx=2; int sy=2;  
Color cor = new Color();  
cor = Color.FromArgb(0, 0, 255);  
retangulo(e,x0,y0,x1,y1,cor);  
MessageBox.Show("ok");  
retangulo(e,x0,y0,x1*sx,y1*sy,cor);
```

# Aplicando Escala



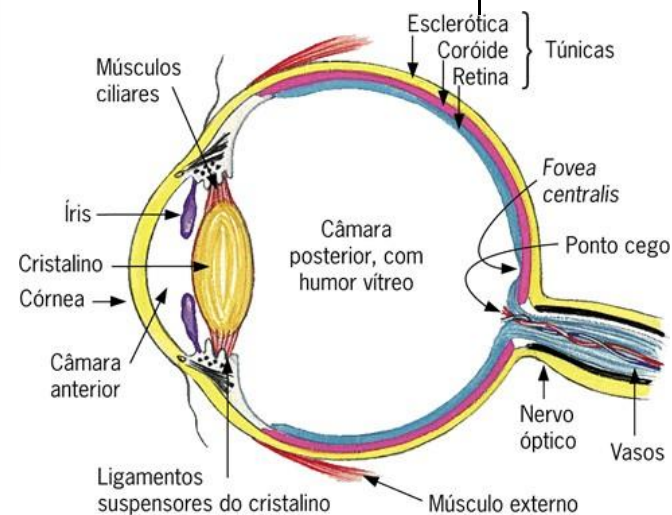
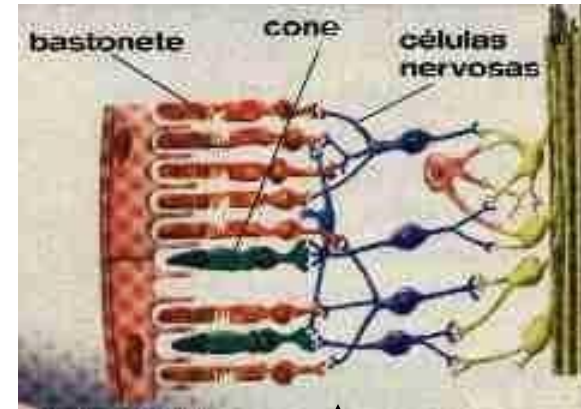
# Introdução ao Processamento de Imagens

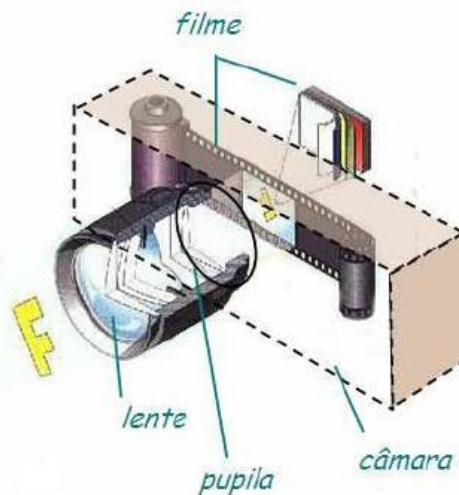
# Introdução : Processando Imagens

## Bitmap

**Processamento de imagem** é qualquer forma de [processamento de dados](#) no qual a entrada e saída são imagens tais como fotografias ou quadros de vídeo.

# Formação da Imagem





## Formação da Imagem

### –Estrutura dos dispositivos de captura de imagens

Exemplos:

- Olhos dos animais, câmeras fotográficas, câmeras de vídeo

Os dispositivos capazes de capturar imagens são organizados estruturalmente de forma similar, possuindo em geral os seguintes elementos:

- Sensor → elemento sensível à energia na forma de radiação eletromagnética, onde se forma a imagem
- Câmara escura + orifício (diafragma, pupila) + lente → servem para filtrar e ajustar a energia eletromagnética que deverá ser direcionada sobre o sensor

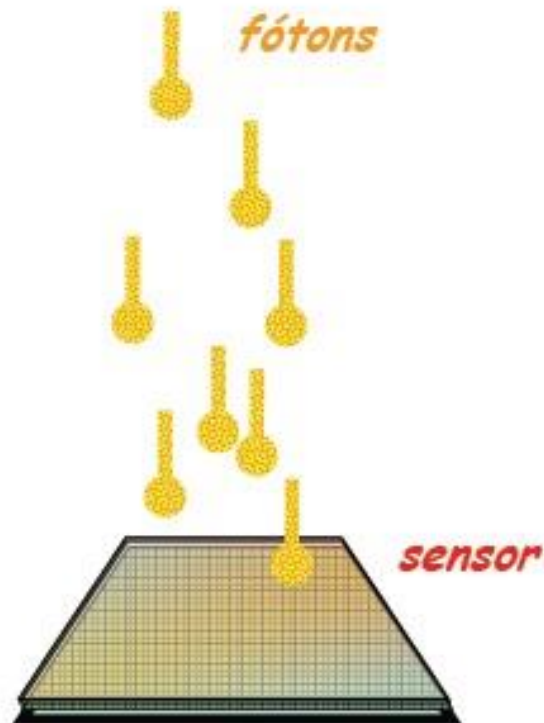
A informação visual é transportada por ondas eletromagnéticas (luz em geral).

As imagens são um registro instantâneo dessa informação.





O sensor de uma câmera digital é formado por um mosaico de pixels. Podemos entender sua operação através de uma alegoria representando os fótons luminosos por gotas de chuva e os pixels do sensor por pequenas bacias usadas para coletar a chuva. Cada bacia coleta uma quantidade de chuva correspondente a quando caiu no ponto em que a bacia está. Assim como cada pixel coleta uma quantidade de fótons correspondente ao quanto chegou da cena na posição em que o pixel se encontra.







## Resolução espacial

### • Exercício

Sabendo-se que:

$X$  = extensão maior da cena

$I$  = extensão maior do sensor

$p$  = extensão do pixel no sentido de  $I$

Calcule o valor da **resolução física  $d$**  = tamanho do menor detalhe da cena visível na sua imagem formada no sensor, relacionando-a aos demais parâmetros acima.

Solução: Tem-se a seguinte proporcionalidade:  $\frac{d}{p} = \frac{X}{I}$

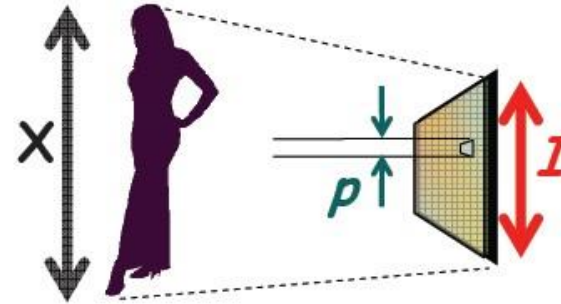
$$\text{portanto} \rightarrow d = p \frac{X}{I}$$

OBSERVAÇÃO → é usual indicar-se a resolução pelas quantidades de pixels existentes nos sentidos vertical e horizontal da imagem.

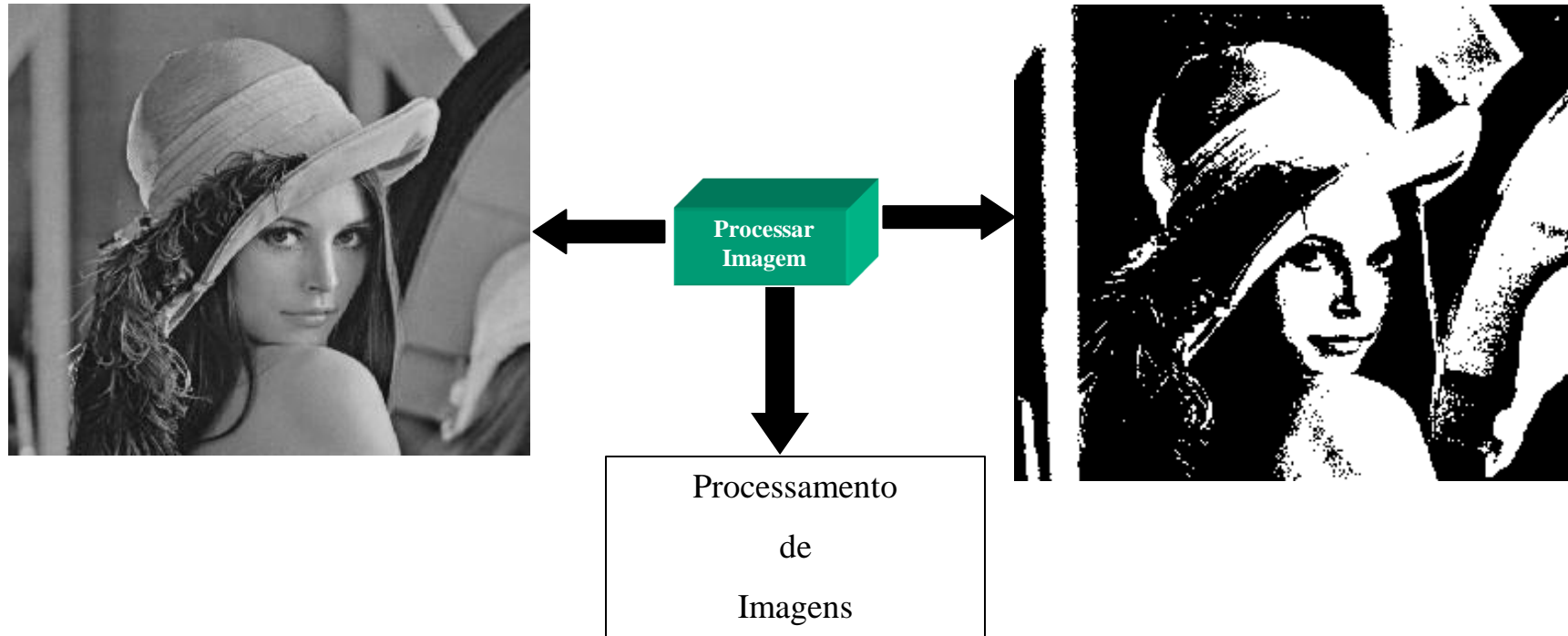
Por exemplo:  $L \times C$  onde  $L$  = quantidade de linhas na imagem e  $C$  quantidades de colunas

Porém, para se usar esses dados corretamente, é preciso conhecer os tamanhos físicos do sensor e do pixel do sensor nessas direções.

$$L = \frac{H}{h} \quad C = \frac{V}{v}$$

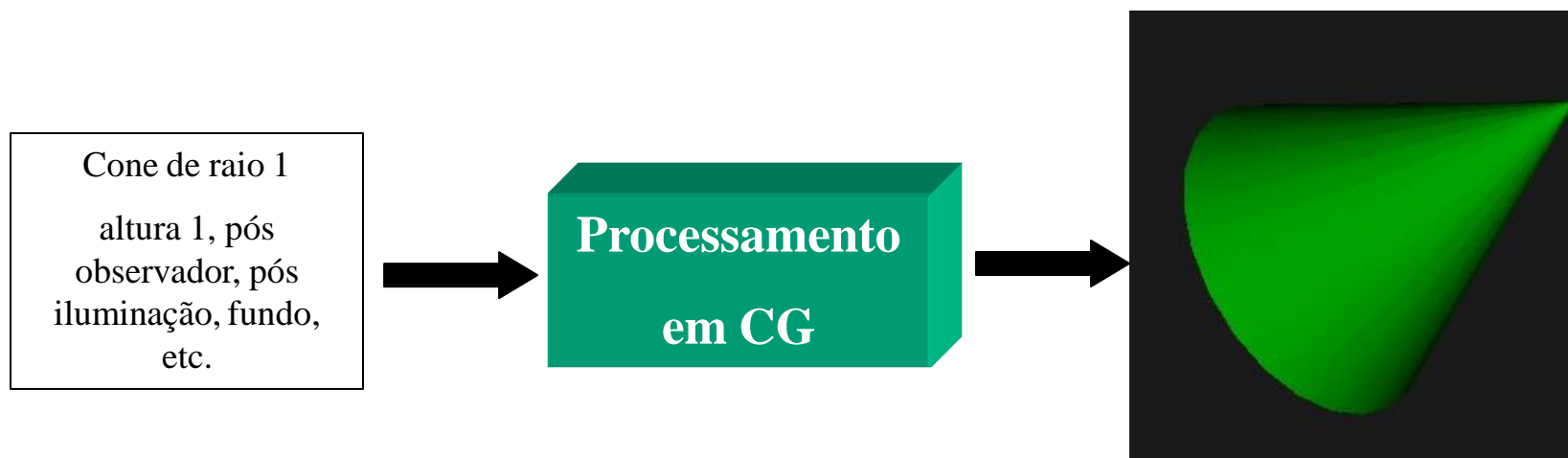


## Processamento computacional envolvendo imagens



**Processamento de Imagens – Estuda as transformações de imagens, que pode ser de uma imagem em outra ou a estimação de parâmetros da imagem.**

# Computação Gráfica (CG)



Estuda os métodos de se criar imagens sintéticas a partir de especificações.

# Tipo Bitmap

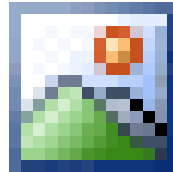
**Device Independent Bitmap (DIB)** ou **Windows Bitmap (BMP)** é um formato de gráficos por mapa de bits (composto por *pixeis*) usado pelos gráficos de subsistema **GDI** (Graphics Device Interface) do **Microsoft Windows**, e é usada geralmente como um formato de gráficos nessa plataforma.

## Importante:

- Um arquivo do tipo BMP é mais seguro para ser manipulado por não ter compressão de informações em seu formato causando a perda de qualidade na imagem ou figura.

# Abrindo um arquivo de imagem.

- Primeiro o objeto da classe Image deve ser instanciado.

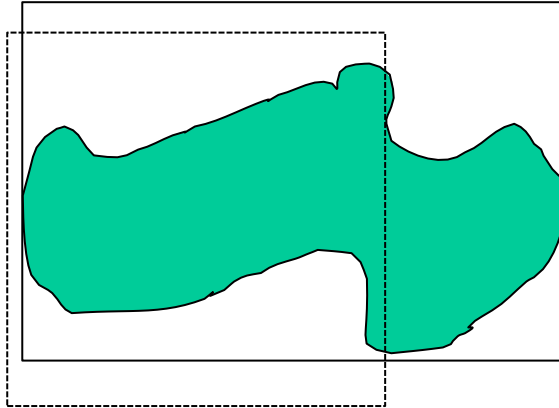


```
pictureBox1.Load("Imagem.bmp");
```

# Fazendo leitura da Imagem

Uma das técnica mais simples para realizar a leitura de uma imagem é chamada de varredura horizontal. Onde pega-se o tamanho da área ocupada pela imagem e realiza-se a leitura dos pixels até o final de cada linha fazendo isso até o final da imagem.



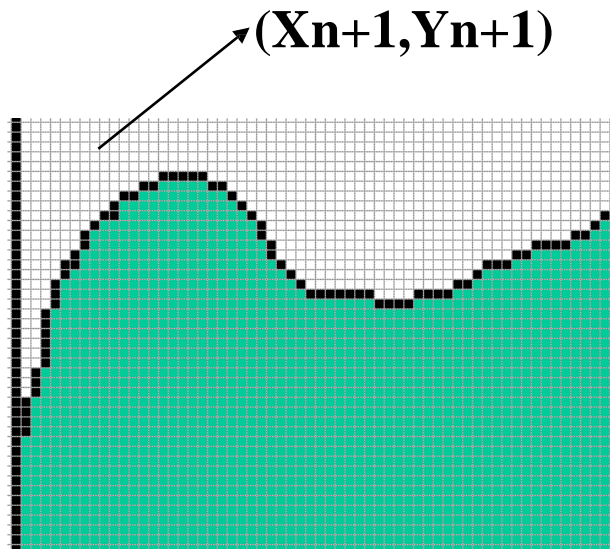


$$\alpha = \text{Largura}(\text{colunas}) \quad x + 1 \leq \alpha$$

$$\beta = \text{Altura}(\text{linhas}) \quad y +$$

## Exemplo : Varredura Horizontal

Largura (width)



Altura  
(height)





# Capturando o tamanho da imagem

## Width e Height

**// Cria um ponteiro do tipo Imagem.**

**// Aponta para o endereço da imagem**

**Bitmap img = new Bitmap("C:\\trabalhos\\vermelho.jpg");**

**// Captura o tamanho da imagem**

**int largura = img.Width;**

**int altura = img.Height;**

**Exemplo :**

```
byte r,g,b;  
r = img.GetPixel(i,j).R;  
g = img.GetPixel(i,j).G;  
b = img.GetPixel(i,j).B;
```

Método	Descrição
GetPixel(i, j).R	Retorna intensidade de cor vermelha.
GetPixel(i, j).G	Retorna intensidade de cor verde.
GetPixel(i, j).B	Retorna intensidade de cor azul.

Método	Descrição
SetPixel(i, j, cor)	Envia Intensidade para Pixel da imagem

### **Exemplo :**

```
cor = Color.FromArgb(r, g, b);  
imgnova.SetPixel(i, j, cor);
```

# Criando a varredura por arranjos simples.

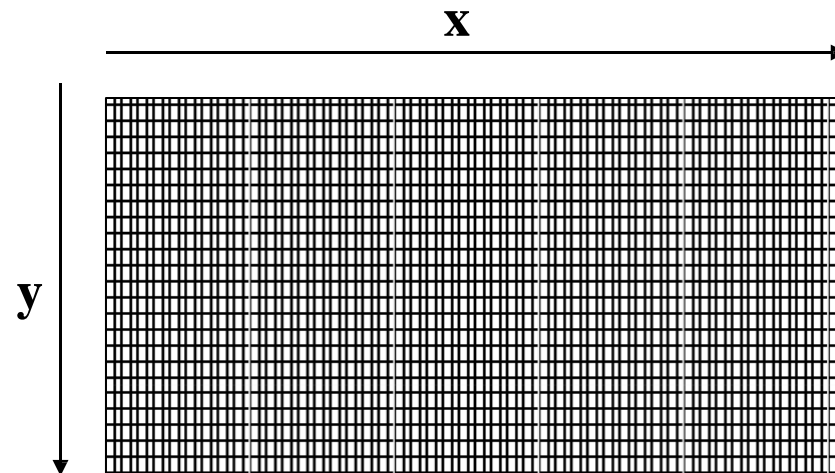
$$S = (\alpha.\beta)$$

S, área retangular da  
imagem.



A, são arranjos simples de S elementos  
tomados 2 a 2.(par ordenado)

Seja: Z  $Z_n = \{x_n, y_{n+1}\}$



```
Bitmap img = new Bitmap("C:\\trabalhos\\aulas\\ICGNOVO-2018\\im1.jpg");
int coluna = img.Width;
int linha = img.Height;
Bitmap imgnova = new Bitmap(coluna, linha);
Color cor = new Color();
for (int i = 0; i <= coluna-1; i++)
{
    for (int j = 0; j <= linha-1; j++)
    {
        int r = img.GetPixel(i, j).R;
        int g = img.GetPixel(i, j).G;
        int b = img.GetPixel(i, j).B;
        if (r == 255 && g == 0 && b == 0)
            cor = Color.FromArgb(0, 255, 0);
        else
            cor = Color.FromArgb(r, g, b);
        imgnova.SetPixel(i, j, cor);
    }
}
```

```
imgnova.Save("C:\\trabalhos\\aulas\\ICGNOVO-2018\\novaImagem.jpg");
```

## Exemplo : (Troca a imagem de cor)

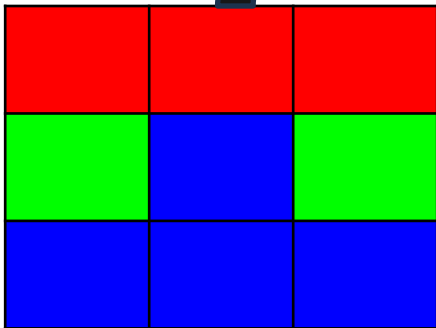
# **Técnicas de Filtro em Imagens Graylevel e Limiarização**

# Graylevel – Conversão para tons de Cinza

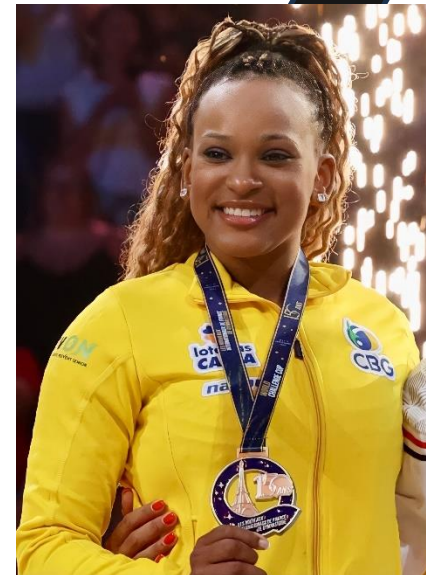
Graylevel, ou nível de cinza, refere-se aos diferentes tons de cinza em uma imagem digital em tons de cinza (grayscale). Em vez de representar cores, uma imagem em tons de cinza representa a intensidade da luz em cada pixel, variando do preto (intensidade mínima) ao branco (intensidade máxima). Cada nível de cinza é representado por um valor numérico que indica a intensidade da luz naquele pixel.

# Aplicação GrayLevel

$$Gray = 0.30 \cdot R + 0.59 \cdot G + 0.11 \cdot B$$



76	76	76
150	28	150
28	28	28



Algoritmo



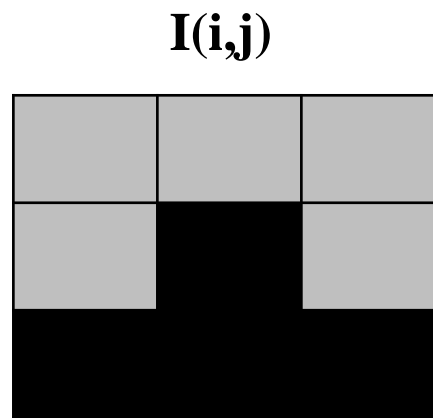


# Limiarização Thresholding

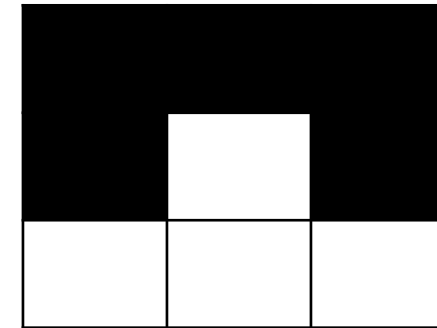
A limiarização (thresholding) é uma técnica de processamento de imagem que converte uma imagem em tons de cinza ou colorida em uma imagem binária, ou seja, uma imagem que tem apenas dois valores possíveis para cada pixel (geralmente preto e branco). Essa técnica é usada para separar objetos do fundo, simplificando a imagem para facilitar a análise e o processamento subsequente.

# Aplicação do Thresholding

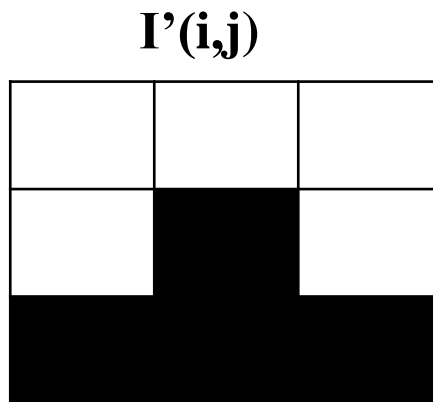
$$I(i,j) \begin{cases} \text{Se } (I(i,j) \leq T), I = 0 \\ \text{Se } (I(i,j) > T), I = 255 \end{cases}$$



$I'(i,j)$



$$I(i,j) \begin{cases} \text{Se } (I(i,j) \leq T), I = 255 \\ \text{Se } (I(i,j) > T), I = 0 \end{cases}$$



# Ajuste de Brilho

# Imagens Digitais

# Conceito de ajuste (Brilho)

Ajustar o brilho de uma imagem no contexto de processamento de imagem envolve manipular os valores dos pixels para tornar a imagem globalmente mais clara ou mais escura. Essa técnica pode ser aplicada tanto a imagens em escala de cinza quanto a imagens coloridas.

# Brilho Imagens em Escala de Cinza

Para uma imagem em escala de cinza, cada pixel tem um único valor de intensidade, geralmente representado em uma escala de 0 (preto) a 255 (branco) para imagens de 8 bits.

Ajuste de Brilho:

Para cada pixel na imagem com intensidade  $I(x,y)$ , o brilho pode ser ajustado adicionando uma constante  $\Delta B$  ao valor de cada pixel:

$$I'(x,y)=I(x,y)+\Delta B$$

$I'(x,y)$  é o valor de intensidade ajustado do pixel na posição  $(x,y)$ .

$\Delta B$  é a constante que determina quanto o brilho deve ser aumentado ou diminuído. Valores positivos aumentam o brilho, enquanto valores negativos diminuem.

### **Clamp dos Valores:**

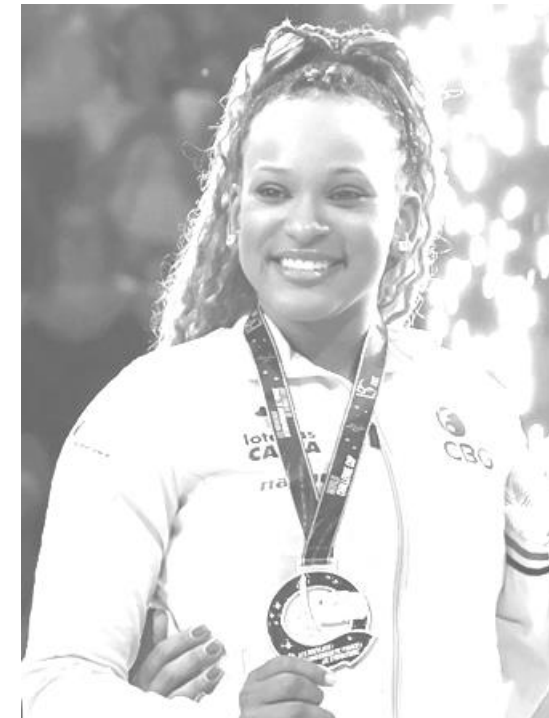
Os novos valores de pixel  $I'(x,y)$  devem ser mantidos dentro dos limites aceitáveis (por exemplo, 0 a 255 para imagens de 8 bits). Isso é feito aplicando:

$$I'(x,y)=\min(\max(I'(x,y),0),255)$$

# Exemplo de Brilho GrayLevel

$$I'(x,y)=I(x,y)+\Delta B$$

$$I'(x,y)=\min(\max(I'(x,y),0),255)$$



# Brilho Imagens Coloridas

Para imagens coloridas, cada pixel tem três componentes de cor: vermelho (R), verde (G) e azul (B). O processo é semelhante ao das imagens em escala de cinza, mas aplicado separadamente a cada componente de cor.

## Ajuste de Brilho:

Para cada pixel na imagem com valores de cor  $R(x,y)$ ,  $G(x,y)$ ,  $B(x,y)$ , o ajuste é feito como:

$$\begin{aligned}R'(x,y) &= R(x,y) + \Delta B \\ G'(x,y) &= G(x,y) + \Delta B \\ B'(x,y) &= B(x,y) + \Delta B\end{aligned}$$

$$R'(x,y), G'(x,y), B'(x,y) = \min(\max(\{R'(x,y), G'(x,y), B'(x,y)\}, 0), 255)$$



# Exemplo de Brilho Colorida

$$I'(x,y)=I(x,y)+\Delta B$$

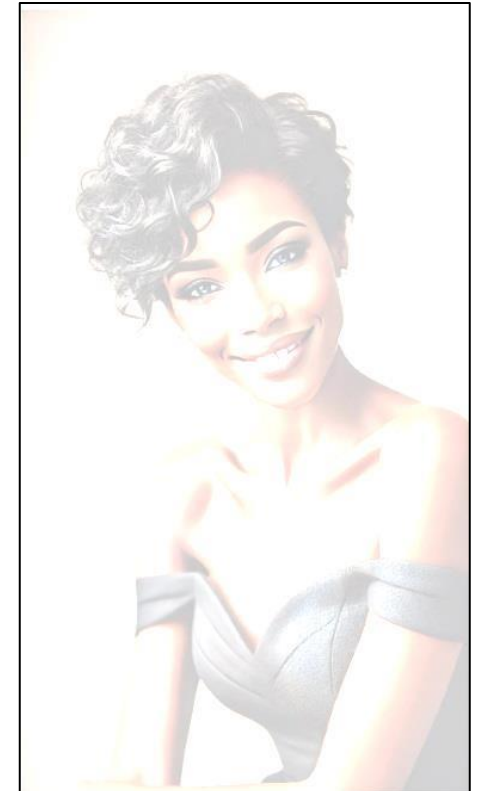
$$R'(x,y)=R(x,y)+\Delta B$$

$$G'(x,y)=G(x,y)+\Delta B$$

$$B'(x,y)=B(x,y)+\Delta B$$



$\Delta B = 100$

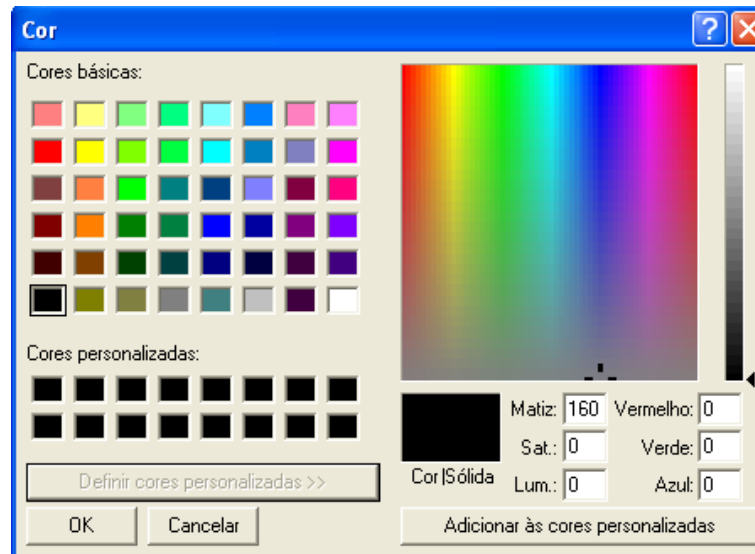


$\Delta B = 200$

$$R'(x,y), G'(x,y), B'(x,y) = \min(\max(\{R'(x,y), G'(x,y), B'(x,y)\}, 0), 255)$$

# Captação de Cores para objetos gráficos

## Capturando cores objeto da classe ColorDialog (Dialog)



Retorna com os tons nas faixas RGB correspondente a cor selecionada.

# Método ColorDialog()

Realiza a instância do objeto da classe ColorDialog.

```
ColorDialog dialogo = new ColorDialog();
```

# ShowDialog()

Exibe a caixa de dialogo, pertencente ao objeto ColorDialog.

```
if (MyDialog.ShowDialog() == DialogResult.OK)
{
    .....
}
```

# Capturando a cor selecionada.

Byte r = dialogo.Color.R;

Byte g = dialogo.Color.G;

Byte b = dialogo.Color.B;

**Captura as cores escolhidas nas faixas adequadas.**

# Conceito de Realidade Aumentada (RA)

## Conceito (RA)

A virtualidade aumentada ocorre quando o mundo virtual é enriquecido com representações de elementos reais pré-capturados em tempo real, que podem ser manipuladas ou interagir no mundo virtual.





Samsung Gear(a)

Cardboard (b)

Oculus(c)

HTC Vive (d)

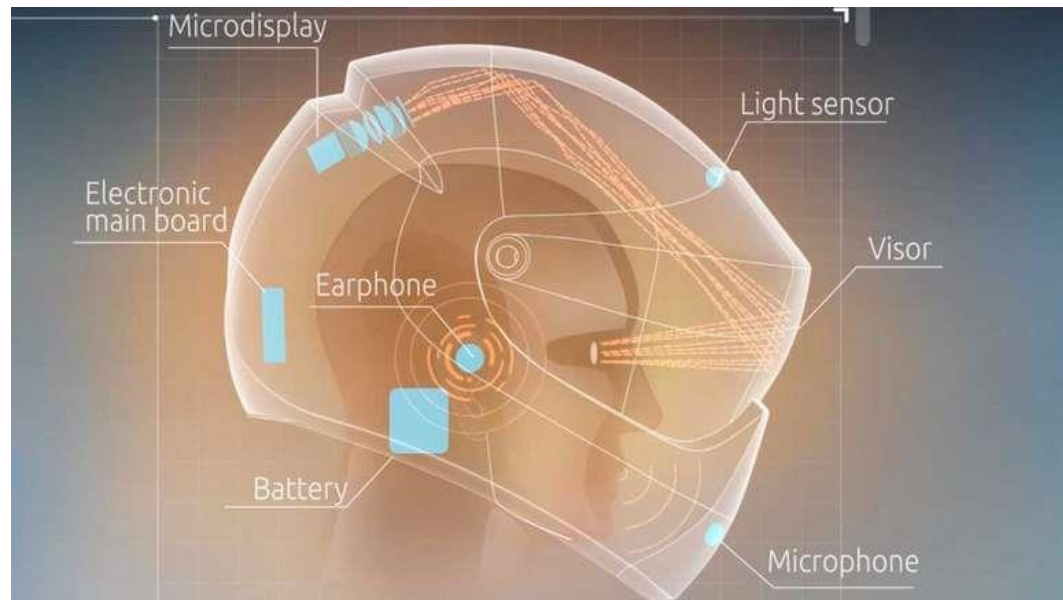
LG (e)

Playstation VR (PSVR) (f)

Hololens (g)

Meta (h)

Moverio BT200 (k)



# Formas de Visualização

- a) Realidade aumentada com monitor (não imersiva) que sobrepõe objetos virtuais no mundo real;
- b) Realidade aumentada com capacete (HMD) com visão óptica direta;
- c) Realidade aumentada com capacete (HMD) com visão de câmera de vídeo montada no capacete;

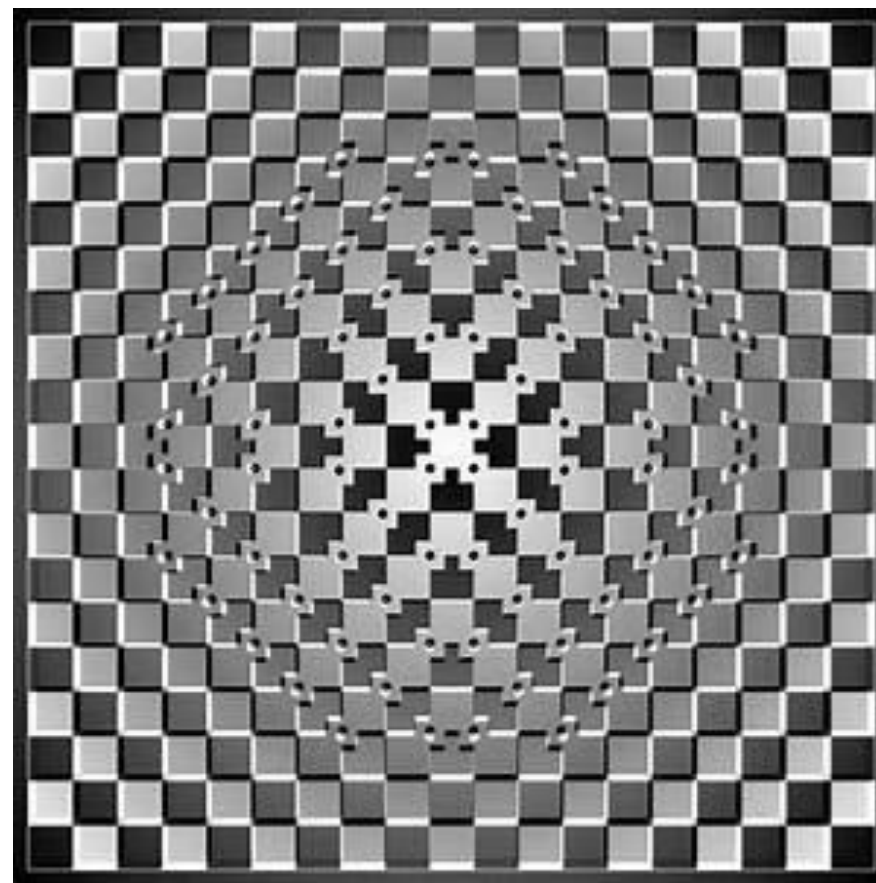
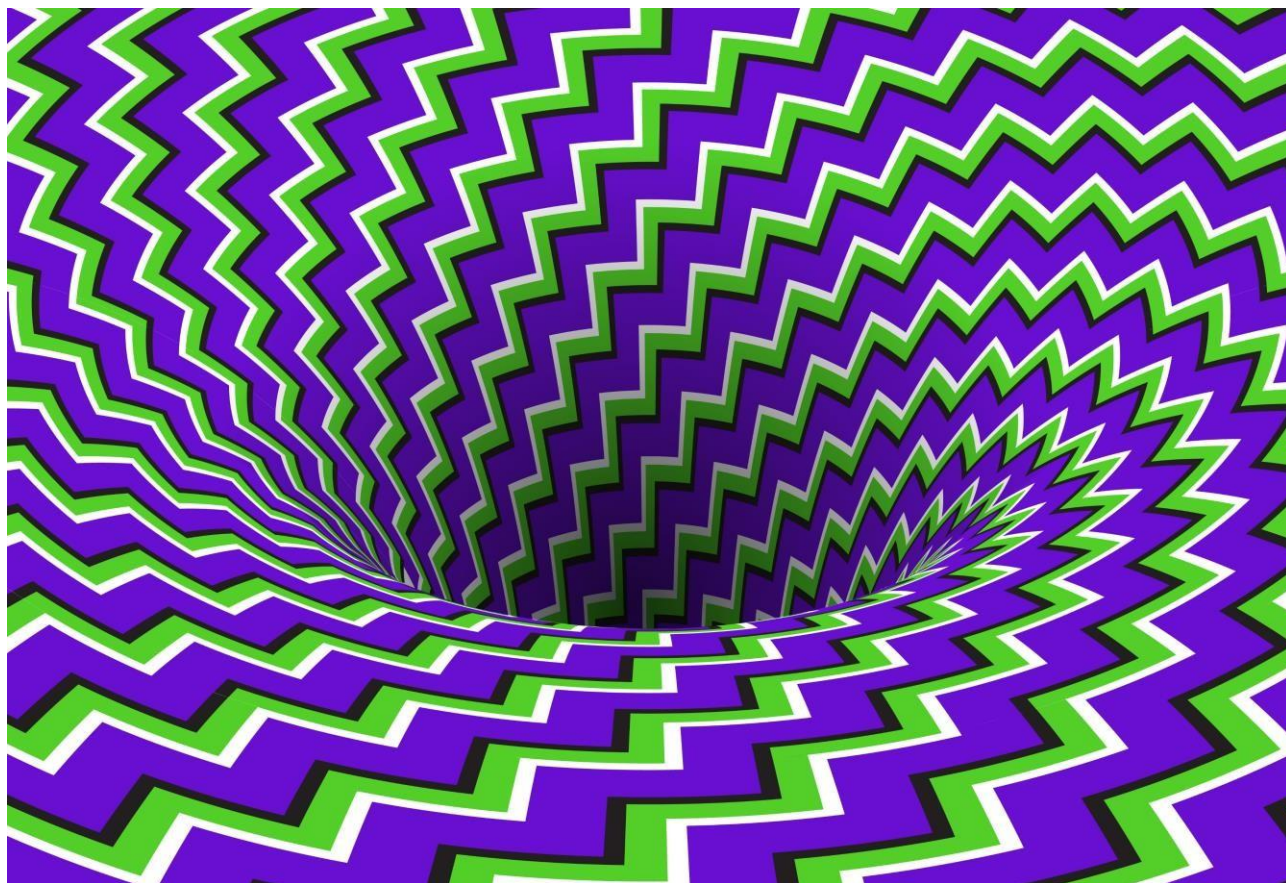
(HMD) - Head Mounted Display  
(Display montado na cabeça)

## Formas de Visualização

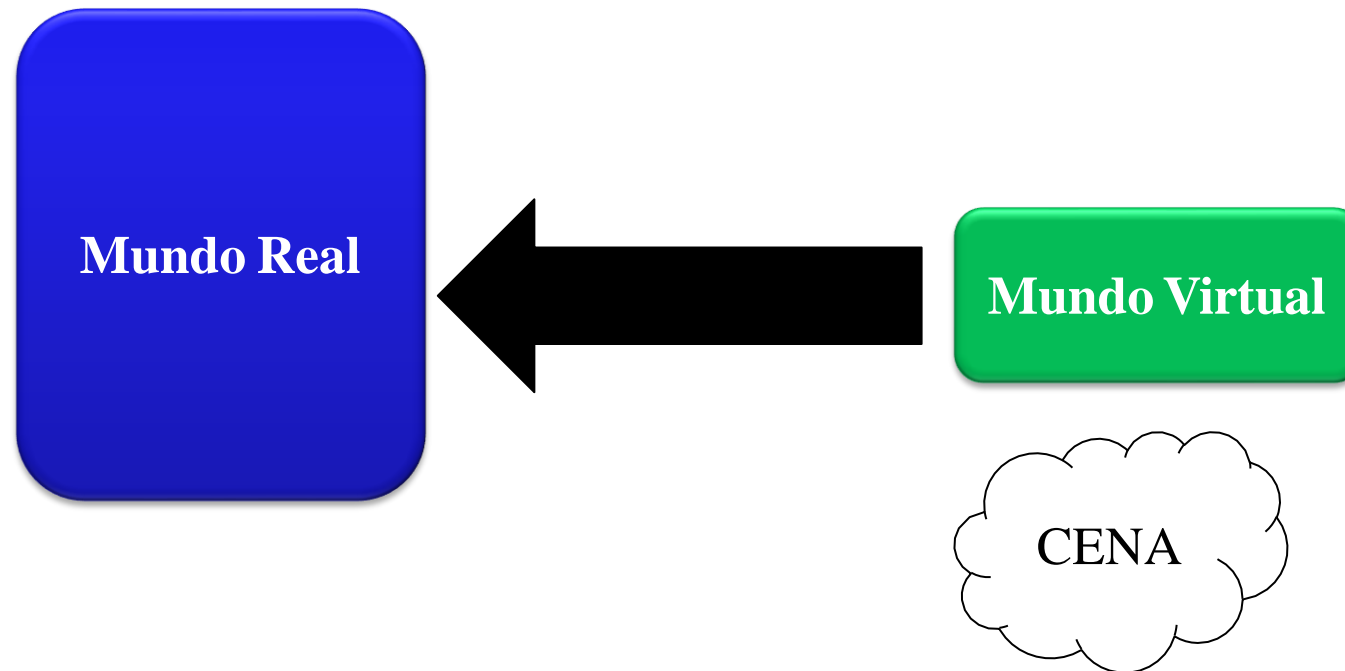
- d) Virtualidade aumentada com monitor, sobrepondo objetos reais obtidos por vídeo ou textura no mundo virtual;
- e) Virtualidade aumentada imersiva ou parcialmente imersiva, baseada em capacete (HMD) ou telas grandes, sobrepondo objetos reais obtidos por vídeo ou textura no mundo virtual;
- d) Virtualidade aumentada parcialmente imersiva com interação de objetos reais, como a mão, no mundo virtual.



# Visão Óptica

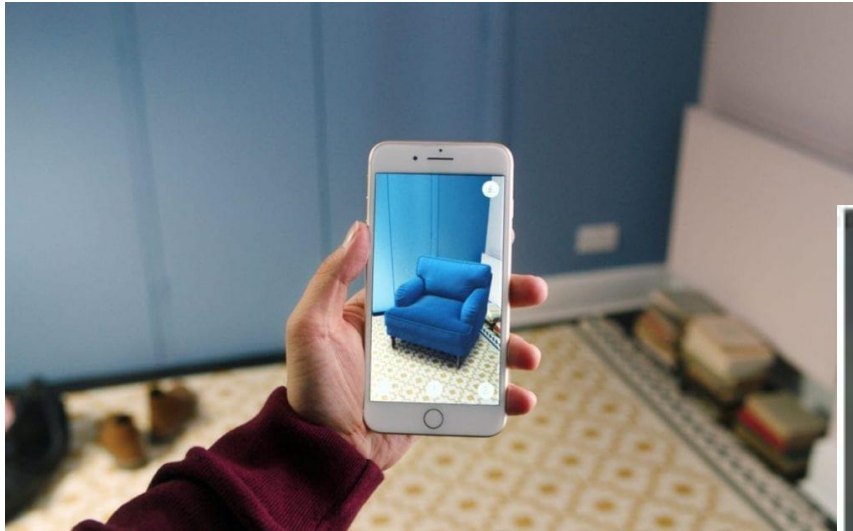


# Realidade Mista

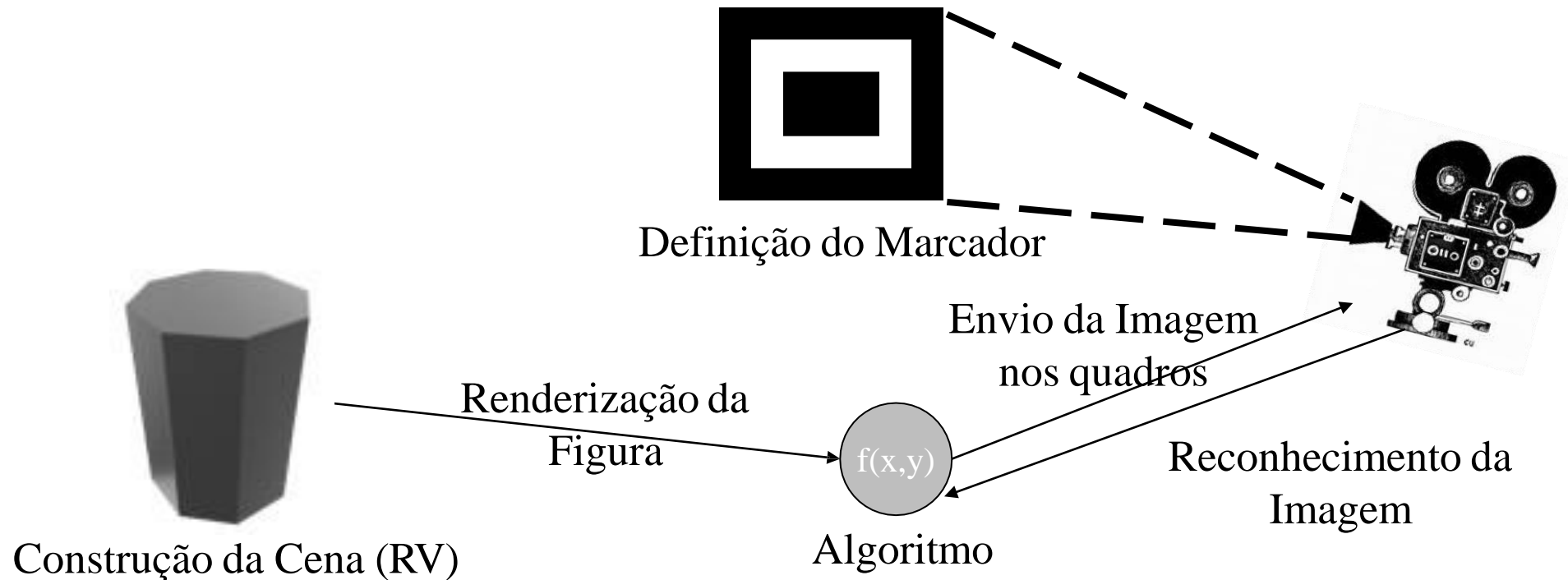


Ou realidade híbrida, vem a ser a junção da realidade virtual com a realidade aumentada.

# Aplicações de Realidade Mista

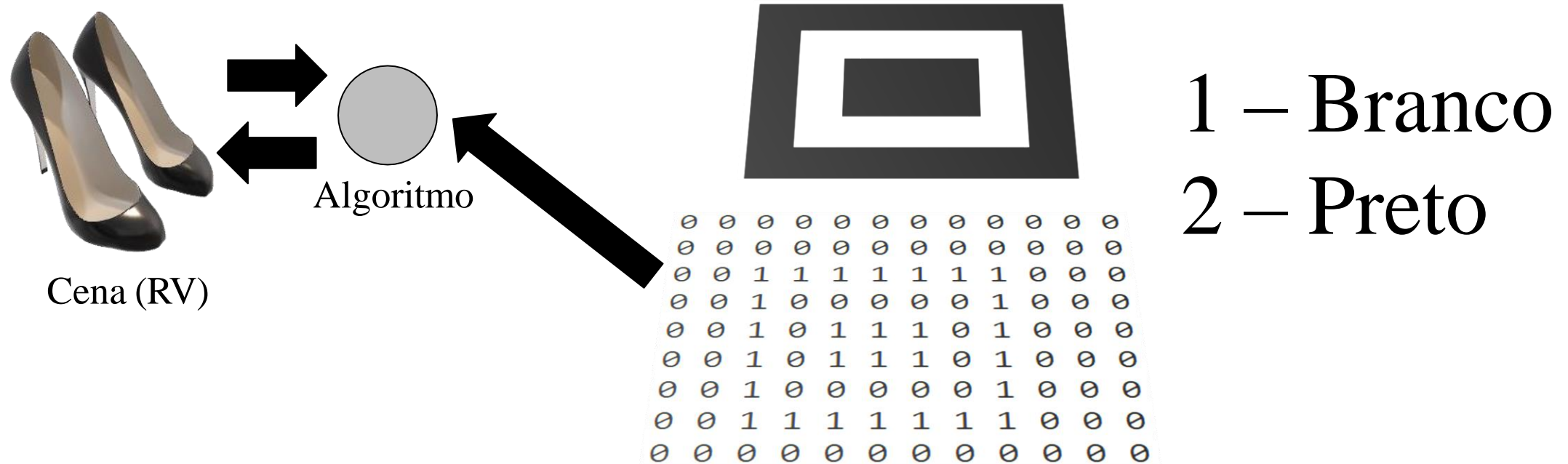


# Definindo Realidade Aumentada





# Marcador (Tag) - Binário

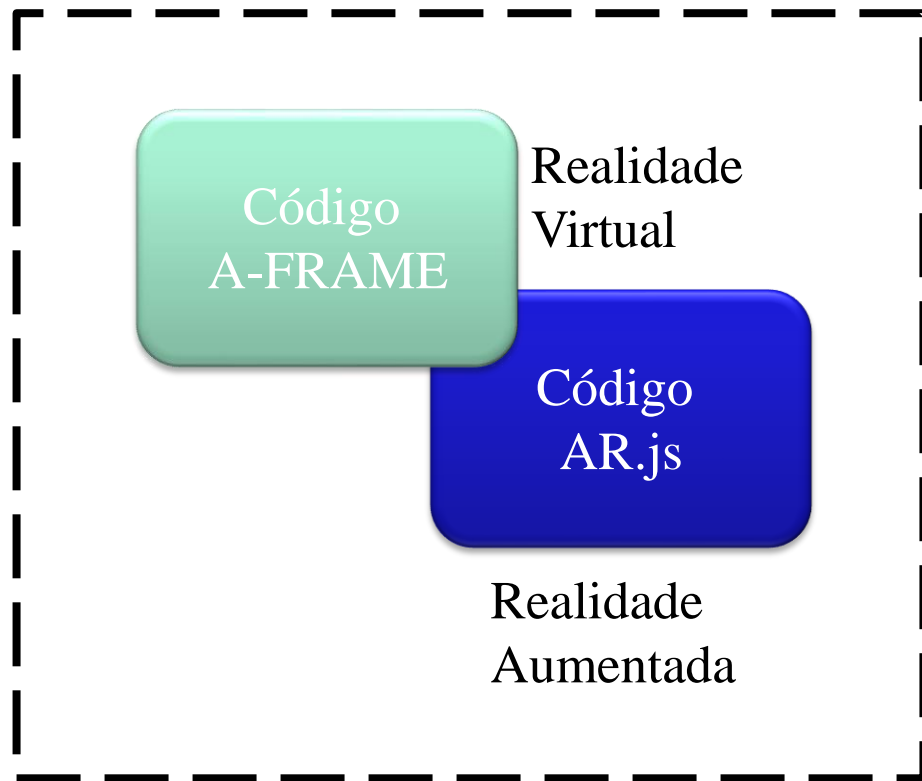


# Ferramentas Realidade Aumentada

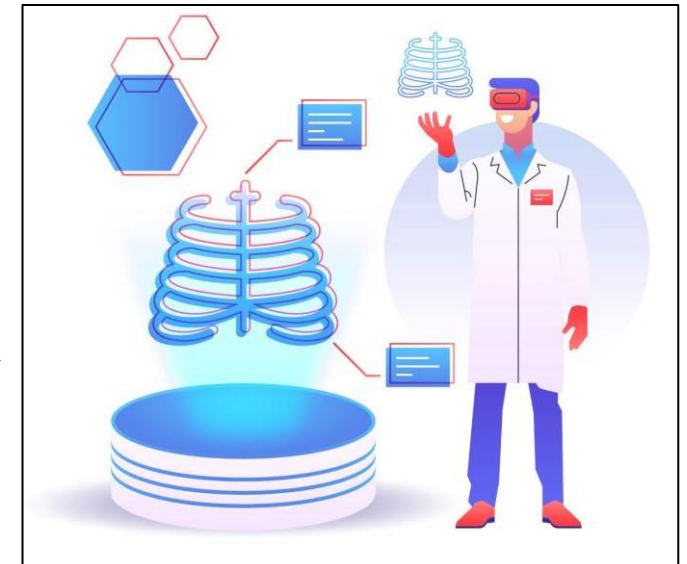
- ArToolkit / JArToolkit
- OpenCV (C++/Python)
- Unity 3D (C#)
- Unreal
- Vuforia Studio
- AR.js (Augmented Reality for the Web)

# Implementação dos Conceitos de RA Usando A-FRAME e AR.js

# Funcionamento da Aplicação



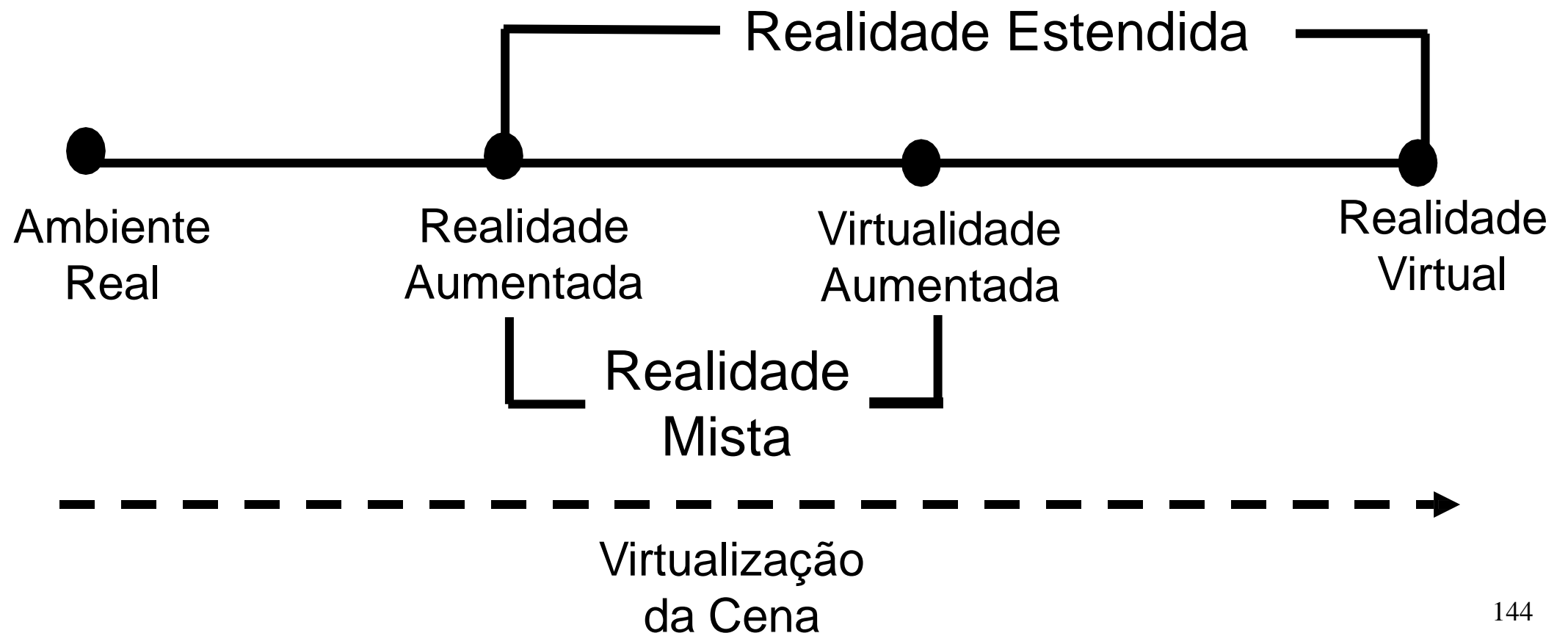
Realidade Virtual + Realidade Aumentada



## CENA

As APIs A-FRAME e AR.js funcionam com a linguagem de marcação de texto HTML5

# Conceito de Realidade Estendida/Mista

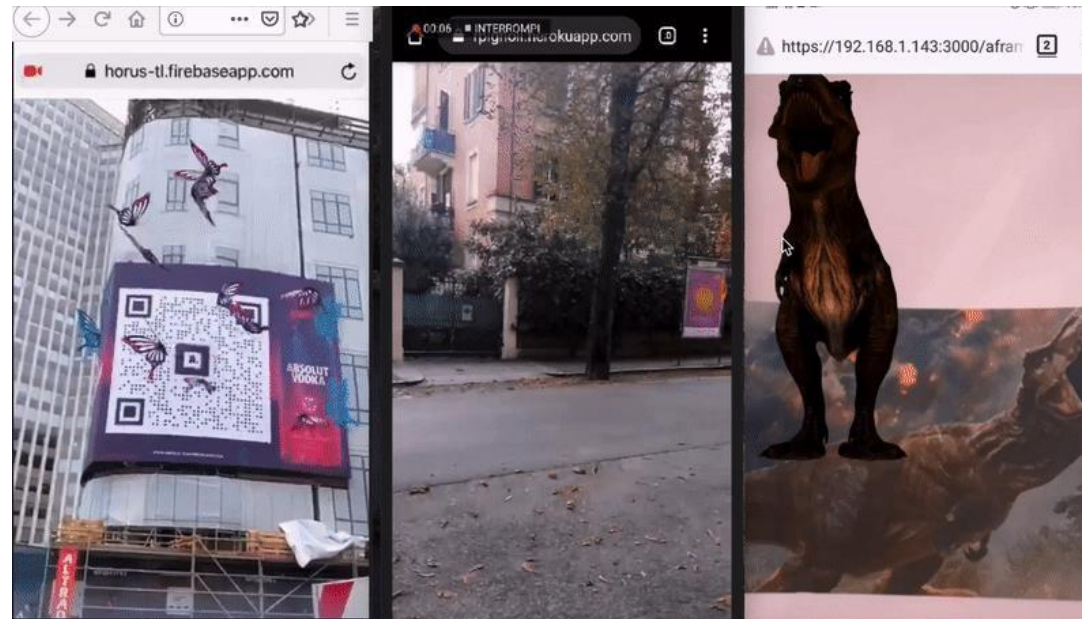


# Exemplo de Realidade Estendida



## Ativando A-FRAME e AR.js

As ferramentas mencionadas são programas em HTML5, permitindo efetuar implementação de código de reconhecimento para as duas bibliotecas.





# Importação das Bibliotecas

1. `<script src="https://aframe.io/releases/1.1.0/aframe.min.js"></script>`
2. `<script src="https://raw.githubusercontent.com/AR-js-org/AR.js/3.3.1/aframe/build/aframe-ar-nft.js"></script>`

1. Insere no código a biblioteca de modelos de primitivas tridimensionais.
2. Insere no código a biblioteca de modelos de realidade aumentada.



## <a-scene>

Cria um cenário para que se adicione elementos gráficos tridimensionais.

Estrutura:

**<a-scene>**

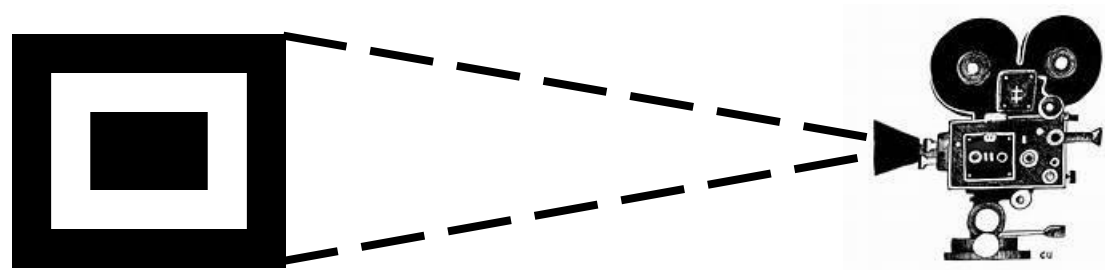
**:**

**:**

**</a-scene>**

<a-scene embedd arjs='sourcetype: webcam;'>

Clausula que incorpora a câmera do dispositivo que possui a câmera conectada.



<a-marker preset= String\_Marcador >

Determina qual será o marcador usado para projetar a imagem nos quadros da cena.



# Ligando cena com marcador

<a-marker preset="hidro">

:

: Cena

:

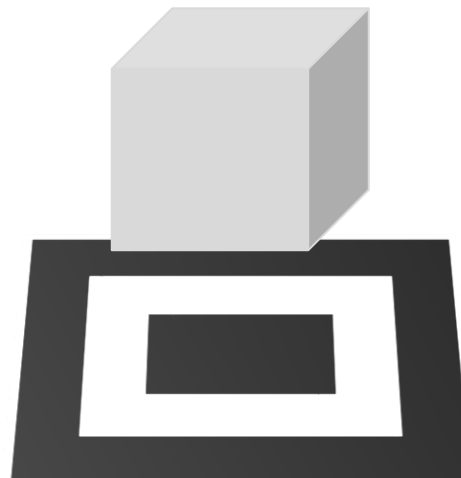
</a-marker>



# Implementação da Cena

<a-box position="-1 0.5 -3" rotation="0 45 0" color="blue"></a-box>

- <a figura> - Figura de realidade virtual apresentada no video devido a leitura de marcador.



# Exemplo Prático de (RA)

**<html>**

**<head>**

**<script src="https://aframe.io/releases/1.1.0/aframe.min.js"></script>**

**<script src="https://raw.githubusercontent.com/AR-js-org/AR.js/3.3.1/aframe/build/aframe-ar-nft.js"></script>**

**</head>**

**<body>**

**<a-scene embedd arjs='sourcetype: webcam;'>**

**<a-marker preset="hidro">**

**<a-box position="-1 0.5 -3" rotation="0 45 0" color="blue"></a-box>**

**</a-marker>**

**</a-scene>**

**</body>**

**</html>**

