EAD

EDUCAÇÃO A DISTÂNCIA

UNIDADE I

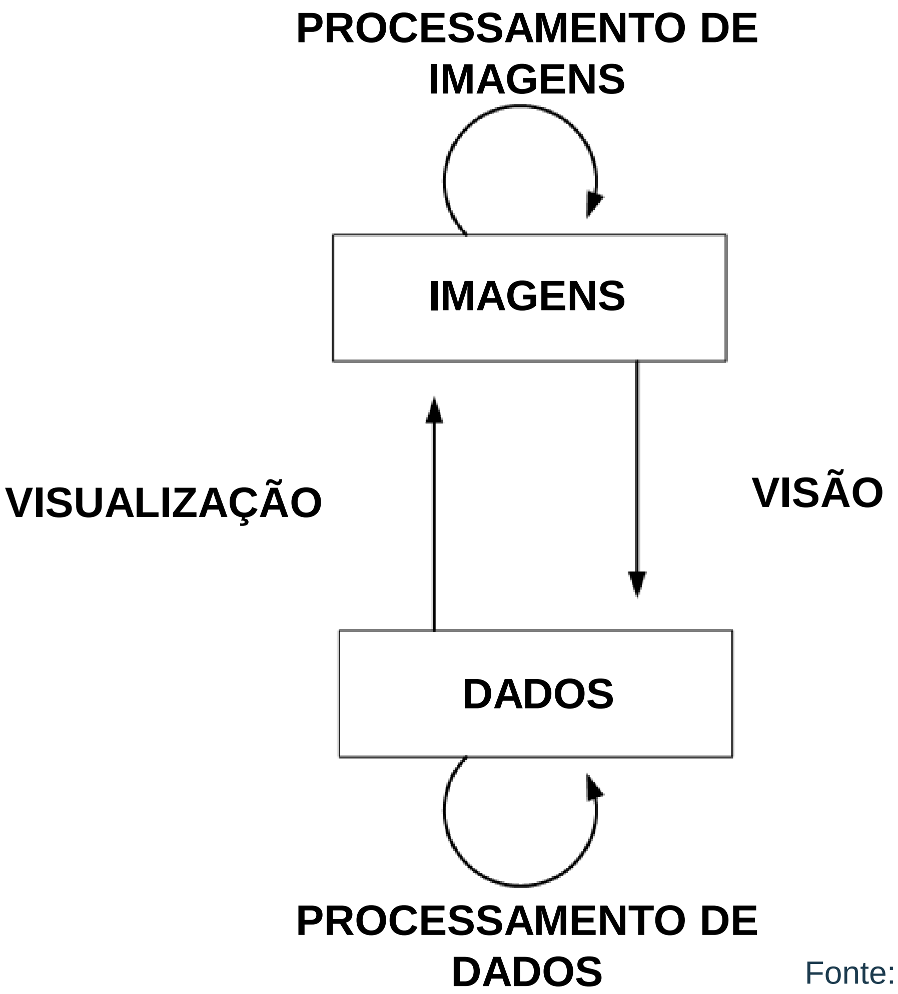
Computação Gráfica

Prof. Hugo Insua

Conceitos básicos e terminologia

- **Computação Gráfica**: a área da Ciência da Computação que estuda a transformação dos dados em imagem. Essa aplicação se estende à recriação visual do mundo real por intermédio de fórmulas matemáticas e algoritmos complexos.

- **Estrutura principal da computação gráfica**.



Fonte: autoria própria.

Origens da computação gráfica

A escala temporal nos ajuda a identificar oportunidades e direções de investigação e aplicação. Algumas das fundações que merecem destaque são:

- **Euclides [300-250 a.C.]** – desenvolveu toda a geometria que norteou seu desenvolvimento até o século XVIII.

- **Brunelleschi [1377-1446]** – arquiteto e escultor italiano que usou de forma criativa a noção de percepção visual e criou em 1425 a perspectiva.

- **Descartes [1596-1650]** – matemático e filósofo francês que formulou a geometria analítica e os sistemas de coordenadas 2D e 3D.

Origens da computação gráfica

- **Euler [1707-1783]** – o mais produtivo matemático do século XVIII, que, entre outros, criou o conceito de senos, tangentes, a expressão que relaciona o número de vértices, arestas e faces de poliedros etc.

- **Monge [1746-1818]** – matemático francês que desenvolveu a geometria descritiva como um ramo da geometria.

- **Sylvester [1814-1897]** – matemático inglês que inventou as matrizes e a notação matricial, uma das ferramentas mais comuns da computação gráfica.

- **Hermite [1822-1901]** – matemático francês que provou a transcendência do número e (usado como base para os logaritmos naturais) desenvolveu funções elípticas e curvas, entre outros.

Arquitetura de sistemas gráficos (o *hardware* gráfico)

- Dispositivos de entrada: como teclado, *mouse*, *joystick*, dispositivos de rastreamento de movimento e câmeras, que permitem ao usuário interagir com o sistema.
- Unidade de processamento gráfico (GPU): é o componente principal que realiza as operações matemáticas necessárias para gerar imagens em tempo real. A GPU é otimizada para processamento paralelo e é capaz de executar muitos cálculos simultaneamente.
- APIs gráficas: são interfaces de programação de aplicativos que permitem que os desenvolvedores interajam com a GPU para criar imagens em tempo real. Algumas das APIs gráficas mais populares incluem OpenGL, DirectX e Vulkan.
- Bibliotecas de *software*: como bibliotecas de física, bibliotecas de animação e bibliotecas de inteligência artificial, que ajudam a criar uma experiência mais imersiva para o usuário.
- *Display* de saída: inclui monitores, projetores e dispositivos de realidade virtual e aumentada, que exibem as imagens geradas pelo sistema.

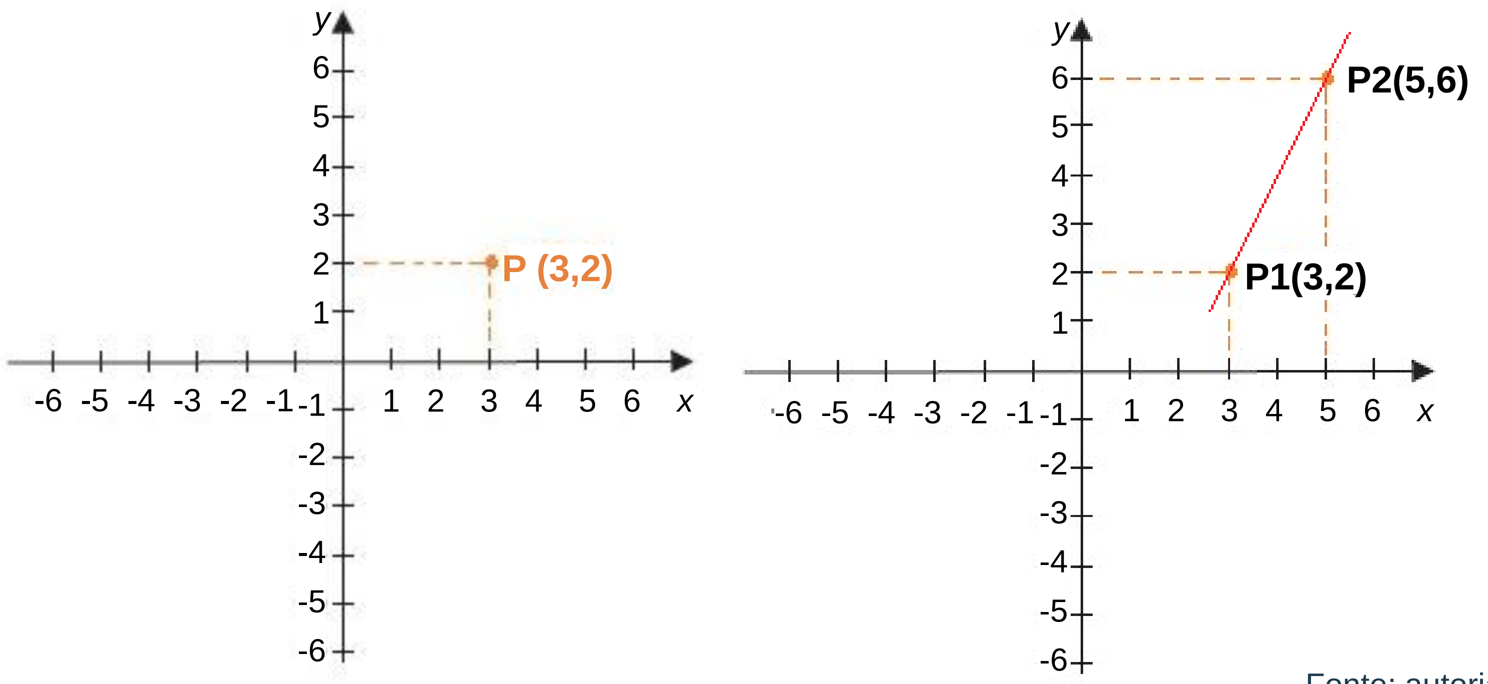
Primitivas como elementos básicos do desenho

Ponto

- Ponto é uma primitiva gráfica que não tem dimensão e é definido por um par de coordenadas (x, y). A representação matemática de um ponto é (x, y).

Linha

- Linha é um elemento gráfico definido por dois pontos, também conhecido como segmento de reta. Uma linha é uma primitiva gráfica definida por dois pontos (x1, y1) e (x2, y2).

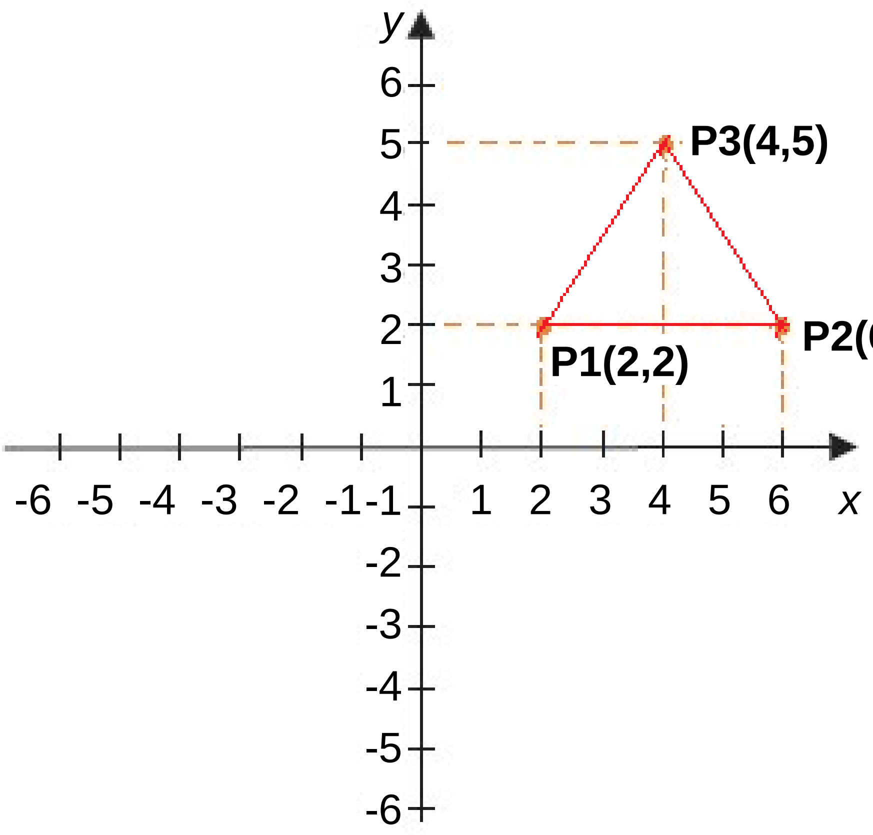


Fonte: autoria própria.

Primitivas como elementos básicos do desenho

Polígono

- Um polígono é uma forma geométrica fechada e plana definida por uma sequência de pontos conectados por linhas retas. Um polígono com n vértices é chamado de n-gon. A representação matemática de um polígono é dada por uma lista de pares ordenados que representam seus vértices, ou seja, P = [(x1, y1), (x2, y2), ..., (xn, yn)].

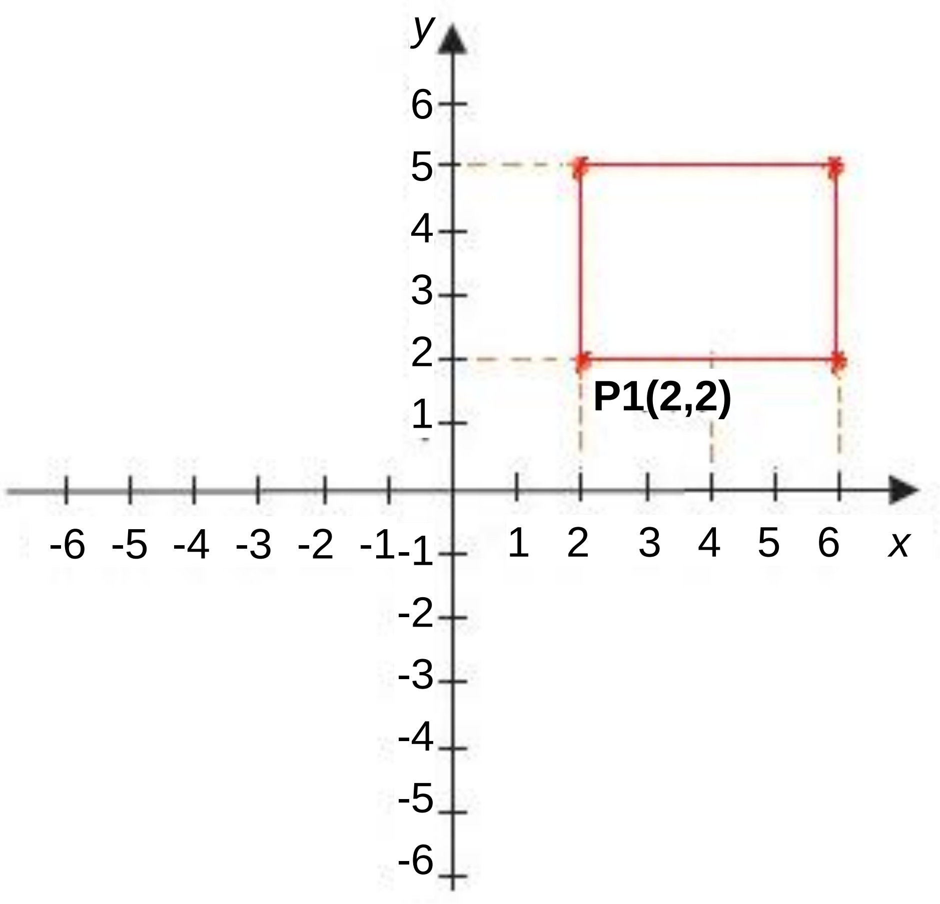


Fonte: autoria própria.

Primitivas como elementos básicos do desenho

Retângulo

- Um retângulo é um polígono com quatro lados, em que os lados opostos são paralelos e iguais. Um retângulo é definido por suas coordenadas (x, y), sua largura e altura. A representação matemática de um retângulo é R = (x, y, w, h), em que w é a largura e h é a altura do retângulo.

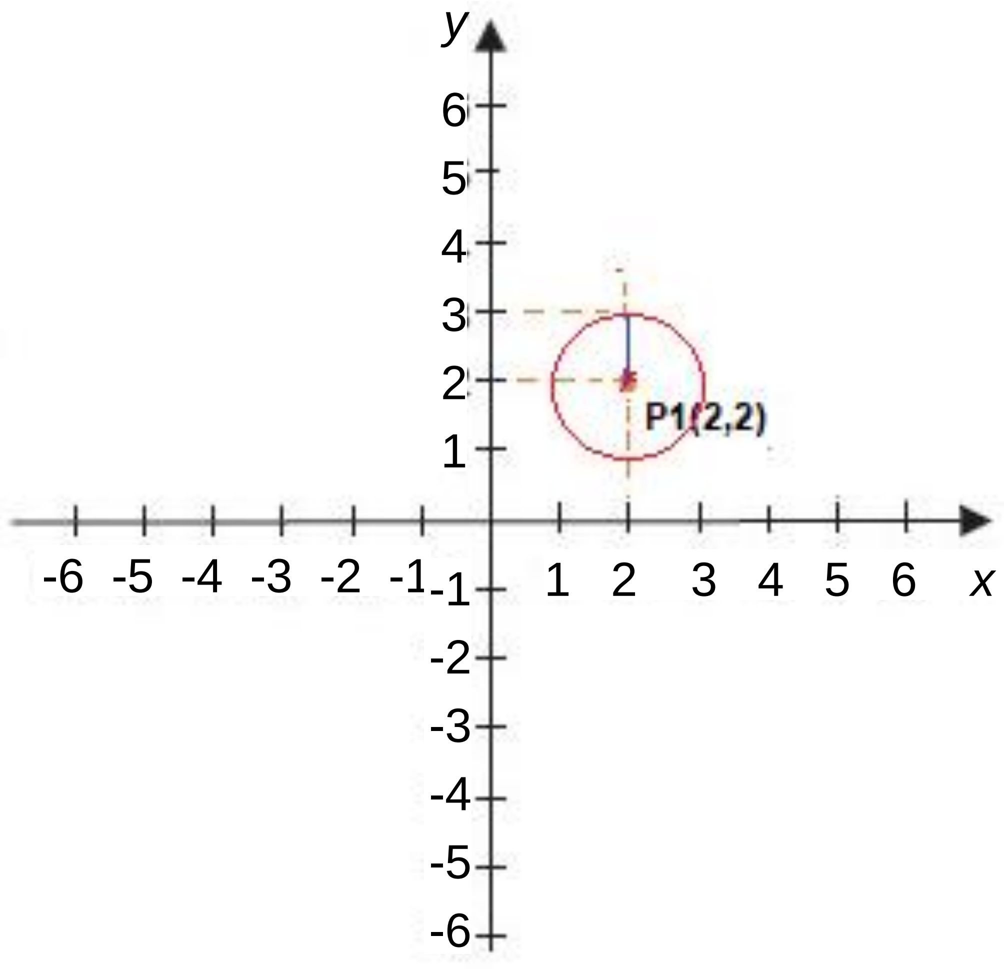


Fonte: autoria própria.

Primitivas como elementos básicos do desenho

Círculo

- Um círculo é uma forma geométrica definida por um ponto central (xc, yc) e um raio r. A representação matemática de um círculo é $(x - xc)^2 + (y - yc)^2 = r^2$.



Fonte: autoria própria.

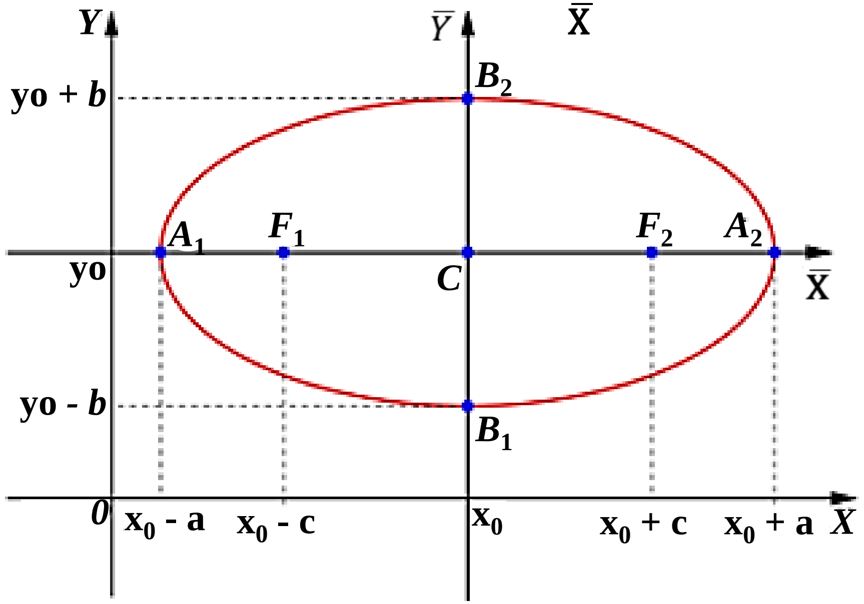
Primitivas como elementos básicos do desenho

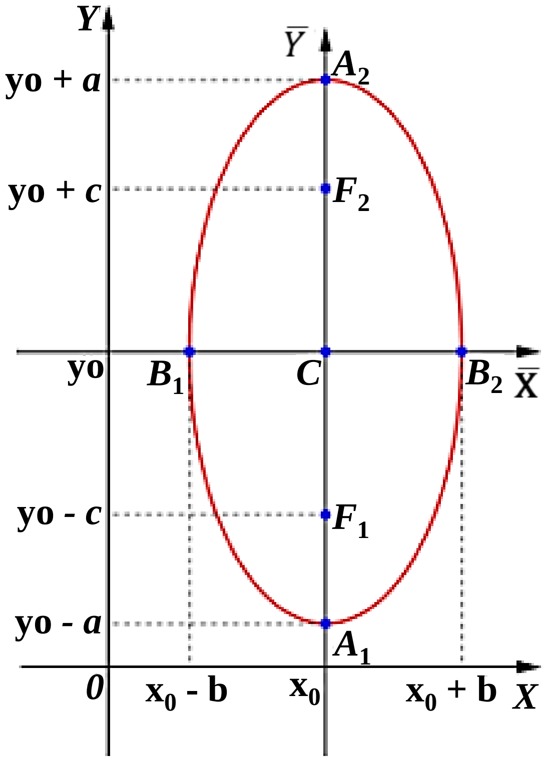
Elipse

Uma elipse é uma forma geométrica que se assemelha a um círculo achatado ou esticado. Os elementos principais são os focos F1 e F2, o eixo maior e o eixo menor. A equação matemática que representa uma elipse é dada por:

$$\frac{(x - x_c)^2}{a^2} + \frac{(y - y_c)^2}{b^2} = 1$$

- ‘a’ é a distância do centro até um dos focos.
- ‘b’ é a metade da dimensão do eixo menor.





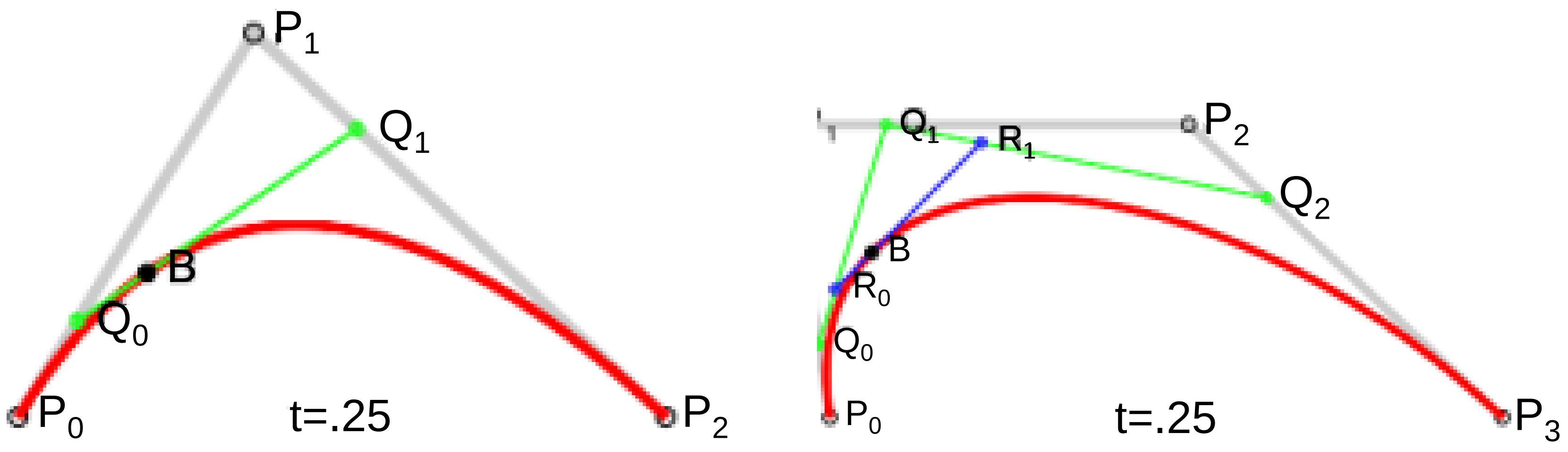
Fonte: autoria própria.

Primitivas como elementos básicos do desenho

Curva de Bézier

A curva de Bézier é uma curva definida por um conjunto de pontos de controle que determinam sua forma. Uma curva de Bézier é uma primitiva gráfica usada para criar curvas suaves em um plano 2D ou em um espaço 3D. A curva de Bézier é uma função paramétrica que descreve a posição do ponto da curva para cada valor do parâmetro t. A representação matemática de uma curva de Bézier é dada por:

$$C(t) = \sum_{i=0}^{N} P_i B_i^n(t)$$



Fonte: adaptado de: Machado (2013, p. 7).

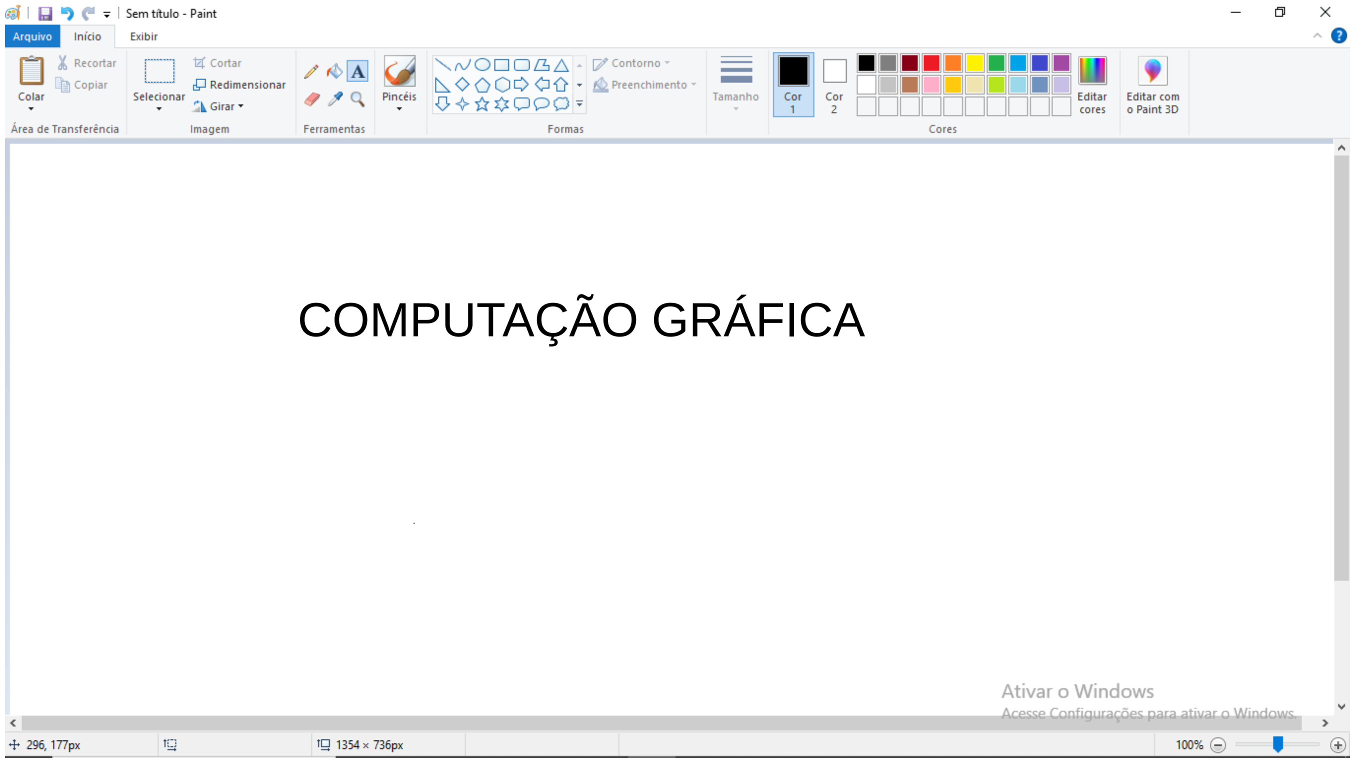
Primitivas com funções de linguagem

- Um *pixel* é a menor unidade de uma imagem digital. É uma abreviação para “elemento de imagem” em inglês (*picture element*) e é comumente usado para medir a resolução de uma imagem digital. Um *pixel* é uma pequena unidade quadrada que pode conter uma cor específica.

Fonte: https://upload.wikimedia.org/wikipedia/commons/a/a5/Pixel_Aspect_Ratio_2zu1.jpg
Acesso: 20/03/2023

Primitivas com funções de linguagem

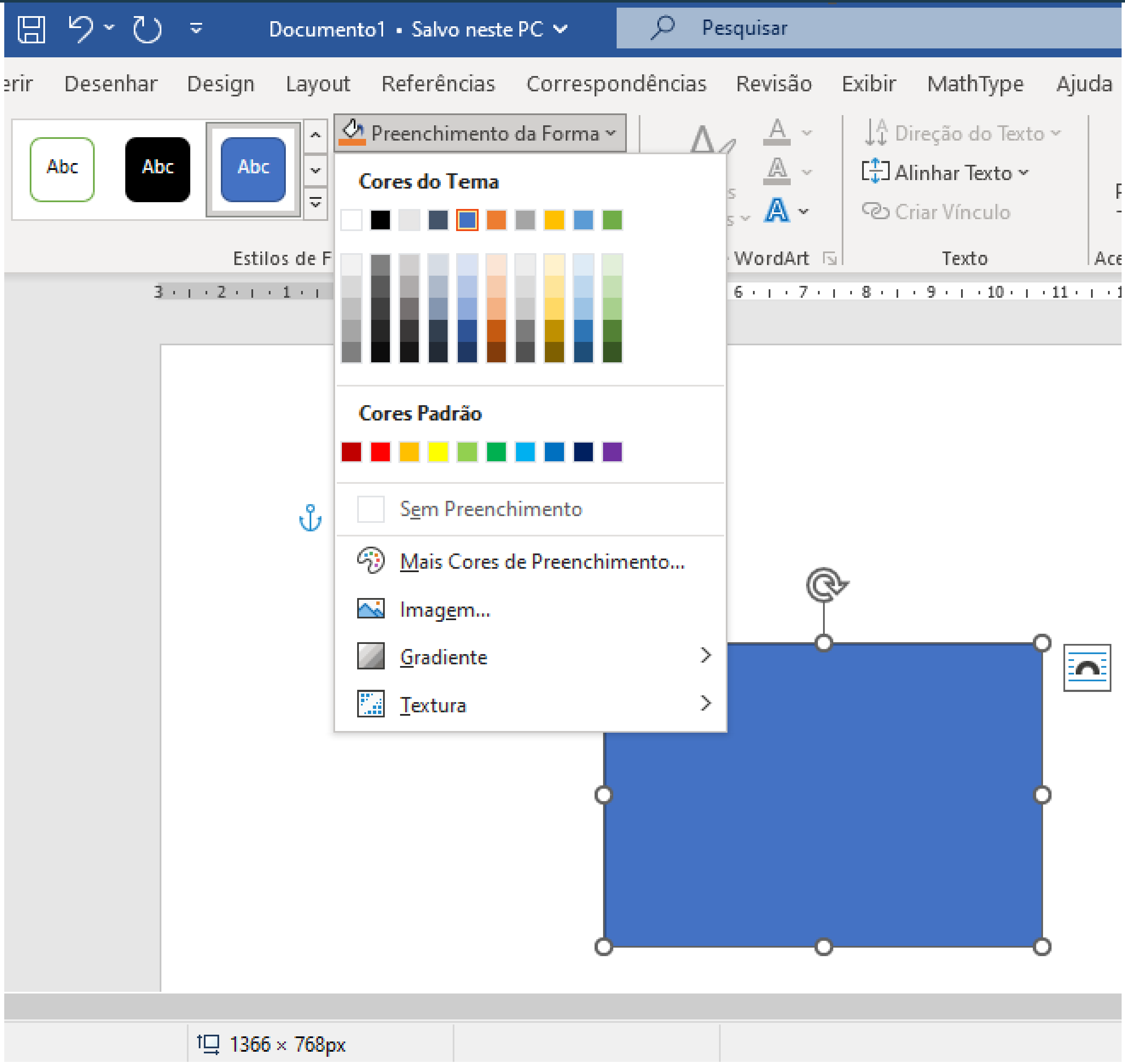
- As primitivas gráficas são uma das principais ferramentas utilizadas na computação gráfica para desenhar objetos e imagens na tela ou no dispositivo de saída. Elas são a base para a criação de algoritmos mais complexos que permitem criar formas mais elaboradas e efeitos visuais.
- **Texto**: é uma primitiva gráfica que permite escrever palavras e frases na tela. A representação matemática do texto é dada pela posição do cursor e pelos caracteres que são desenhados na tela.



Fonte: autoria própria.

Primitivas com funções de linguagem

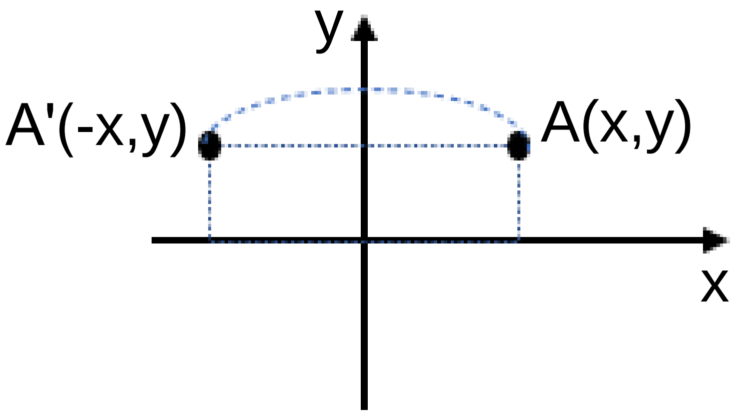
- **Preenchimento de cor**: o preenchimento de cor é uma primitiva gráfica que permite preencher uma área com uma cor sólida. A representação matemática do preenchimento de cor é dada pela posição do ponto de partida e pela cor que é usada para preencher a área.



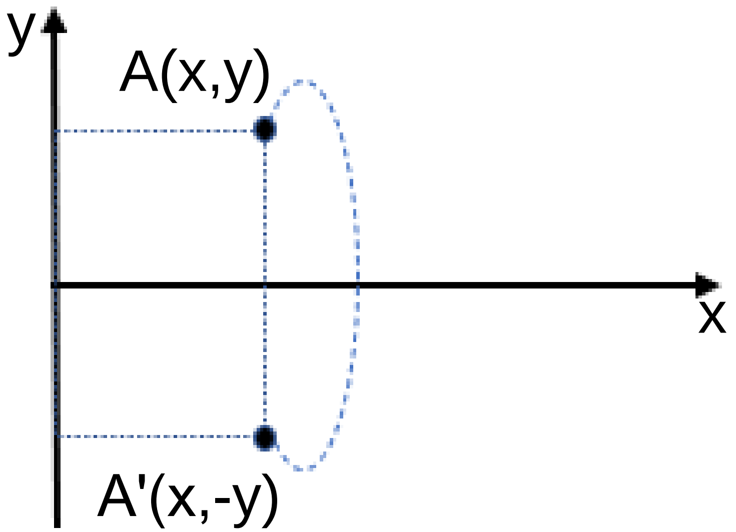
Fonte: autoria própria.

Primitivas com funções de linguagem

- **Transformações geométricas**: as transformações geométricas são primitivas gráficas que permitem mover, rotacionar, redimensionar e espelhar objetos na tela. A representação matemática dessas transformações é dada por matrizes de transformação.



Reflexão em torno do eixo y

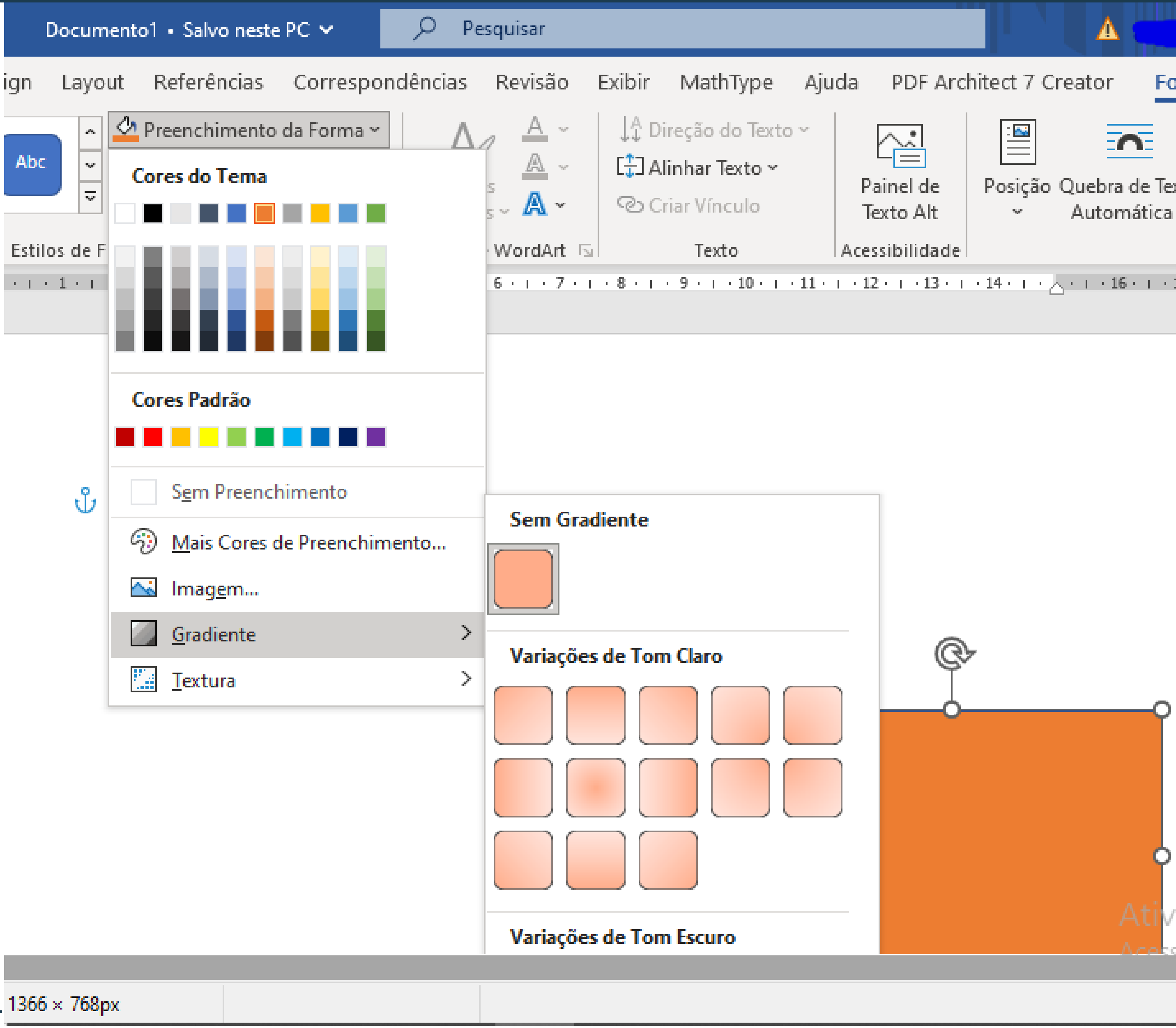


Reflexão em torno do eixo x

Fonte: autoria própria.

Primitivas com funções de linguagem

- **Gradientes**: são primitivas gráficas que permitem criar transições suaves entre duas ou mais cores. A representação matemática dos gradientes é dada pela posição do ponto de partida e do ponto de término, além das cores que são usadas na transição.



Fonte: autoria própria.

Pacotes gráficos e bibliotecas principais

Os pacotes gráficos são conjuntos de ferramentas de *software* para criação, edição e renderização de imagens digitais. Eles geralmente incluem uma ampla variedade de recursos, como modelagem 3D, animação, texturização, iluminação e efeitos especiais. Algumas das principais opções de pacotes gráficos incluem:

- Autodesk Maya
- Autodesk 3ds Max
- Blender
- Cinema 4D
- Houdini
- Unity

Pacotes gráficos e bibliotecas principais

As bibliotecas principais são conjuntos de código pré-escrito que fornecem funcionalidades específicas para a criação de gráficos em um determinado ambiente de programação. Essas bibliotecas podem ser usadas para criar aplicativos de jogos, visualização de dados, interfaces de usuário e muito mais. Algumas das principais bibliotecas gráficas incluem:

- OpenGL (para gráficos 3D)
- DirectX (para jogos em plataformas Windows)
- Vulkan (para jogos em plataformas multiplataforma)
- WebGL (para gráficos 3D em navegadores da *web*)
- Cairo (para gráficos 2D em várias plataformas)
- OpenCV (para processamento de imagens em tempo real)

Interatividade

Não são objetos de estudo da Computação Gráfica:

a) A síntese e o processamento de imagens.
b) A análise de imagens.
c) Visão computacional.
d) Desenvolvimento de sistemas operacionais e gerenciadores de tarefas.
e) Desenvolvimento de tecnologias para a síntese de imagens computadorizadas.

Resposta

<u>Não</u> são objetos de estudo da Computação Gráfica:

a) A síntese e o processamento de imagens.
b) A análise de imagens.
c) Visão computacional.
d) Desenvolvimento de sistemas operacionais e gerenciadores de tarefas.
e) Desenvolvimento de tecnologias para a síntese de imagens computadorizadas.

Primitivas gráficas em duas dimensões

- As primitivas gráficas são ambos elementos básicos de gráficos/desenhos, a partir dos quais são construídos outros objetos, mais complexos, e também os comandos e funções que manipulam e alteram os elementos gráficos de uma imagem.
- No que diz respeito à primeira definição, a modelagem de objetos por meio de primitivas dá-se por instanciação: uma linha é uma sequência de pontos, uma polilinha é uma sequência de linhas etc. Já uma circunferência PODE ser vista como uma polilinha fechada, em que o número de linhas define a qualidade da curva e assim por diante.

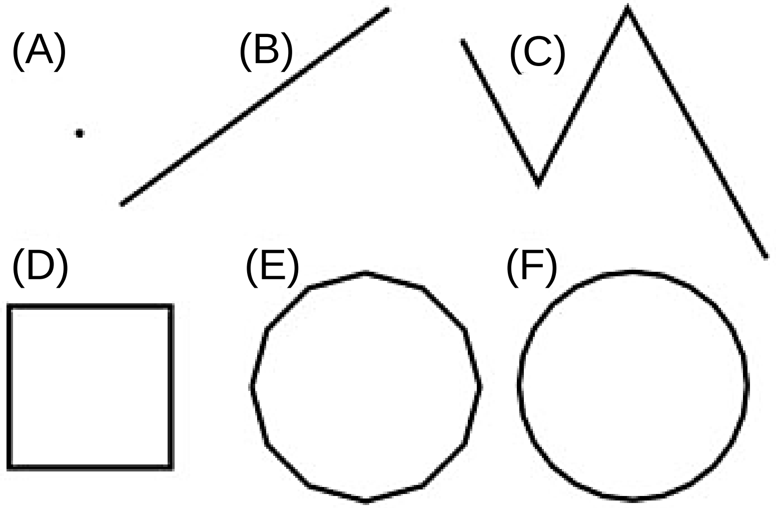


Figura 1. Primitivas Gráficas 2D: (A) ponto, (B) linha, (C) polilinha, (D) retângulo, (E) dodecágono, (F) circunferência (polilinha com 24 lados ou linhas).

Fonte: autoria própria.

Primitivas gráficas em duas dimensões

- Atributo da primitiva gráfica PONTO.

- Em CG, o PONTO matemático, sem dimensão, transforma-se no *PIXEL*, a menor unidade gráfica manipulável (*PIXEL*: de *Picture Element*).

- Ao contrário do PONTO, o *PIXEL* tem forma, dimensão, mas esses atributos não são, em geral, manipuláveis diretamente por meio de funções e dependem do *hardware*.

Primitivas gráficas em duas dimensões

- Por outro lado, em CG, tanto o PONTO quanto o *PIXEL* são objetos de um desenho que se inscreve em um espaço bidimensional. No caso do ponto matemático, tal espaço pertence ao R^2. Já o *pixel* é um elemento de uma matriz de tamanho W x H (*Width*-largura e *Height*-altura, em *pixels*, da matriz do dispositivo gráfico de saída), cujas coordenadas pertencem ao N^2. (R é o conjunto dos números reais e N é o conjunto dos naturais).

- Assim, podemos atribuir ao *pixel* apenas duas propriedades fundamentais: COR e POSIÇÃO.

Primitivas gráficas em duas dimensões

- Como as coordenadas do PONTO, que formam um VETOR com valores no espaço real, e as coordenadas do *PIXEL* que são elementos de uma MATRIZ, possuindo coordenadas inteiras e positivas, devemos converter os valores adequadamente.

- Esse cálculo é denominado de MAPEAMENTO WINDOW-TO-VIEWPORT. Podemos chamá-lo de RASTERIZAÇÃO do ponto, já que o termo rasterização (*rastering*) é o processo de converter vetores para *pixels* em dispositivos *raster* (lembre-se de que as coordenadas de um ponto são grandezas vetoriais).

Primitivas gráficas em duas dimensões

- O MAPEAMENTO *WINDOW-TO-VIEWPORT*

- Sejam XR e YR as coordenadas (reais) de um ponto pertencente à R2 e XP e YP (o P se refere a *pixel*) às coordenadas do *pixel* correspondente (inteiros positivos) na matriz gráfica (tela ou canvas).

- As coordenadas reais (XR, YR) são, muitas vezes, referidas como COORDENADAS DO UNIVERSO (ou do MUNDO) e estão descritas em um SISTEMA DE REFERÊNCIA DO UNIVERSO (SRU), ou sistema de referências do mundo. São, em última análise, as coordenadas que usamos normalmente na modelagem de objetos matemáticos.

Primitivas gráficas em duas dimensões

- Por outro lado, as coordenadas inteiras (XP, YP) são descritas no SISTEMA DE REFERÊNCIA DO DISPOSITIVO gráfico de saída (SRD). Quando elas representam diretamente os valores das LINHAS e COLUNAS no dispositivo gráfico, portanto, valores inteiros que correspondem aos índices dos elementos da matriz de *pixel*, serão referidas como coordenadas do dispositivo. Caso essas coordenadas expressem os VALORES REAIS de seus respectivos pontos no SRU, serão referidas como COORDENADAS LÓGICAS.

- Uma vez que o espaço disponível para desenho no dispositivo gráfico de saída é limitado pela matriz de *pixels*, devemos especificar os valores mínimos e máximos de uma JANELA DE VISUALIZAÇÃO (*viewport*) na qual visualizaremos o nosso modelo matemático (desenho, função etc.).

Primitivas gráficas em duas dimensões

- Portanto, a *viewport* corresponde a uma área no interior da matriz de *pixels* do dispositivo de saída gráfica. Ela pode corresponder a toda a matriz, como pode haver várias *viewports* no dispositivo, sobrepostas ou não...

- Já o plano cartesiano onde localizamos as coordenadas reais (XR, YR) é infinito. Para representarmos qualquer modelo matemático desse espaço no interior da *viewport*, devemos definir um DOMÍNIO que contém os objetos a serem representados. Esse domínio será chamado de *WINDOW* por falta de um nome melhor. A *window* tem o mesmo papel de uma janela no sentido comum, por meio da qual vemos uma cena que queremos representar. Cada elemento primitivo descrito no interior da *window* será, então, MAPEADO na *viewport*.

Primitivas gráficas em duas dimensões

- Frequentemente, os objetos geométricos são descritos (modelados) em um sistema próprio, denominado SISTEMA DE REFERÊNCIA DO OBJETO (SRO).
- O mapeamento *window-to-viewport* ocorre diretamente entre os SRU e o SRD, mas podemos usar as coordenadas lógicas como coordenadas intermediárias no mapeamento.
- Outro sistema de referência utilizado na modelagem de objetos é o SISTEMA DE REFERÊNCIA NORMALIZADO (SRN). O SRN nada mais é do que o SRO limitado no intervalo [0, 1] ou [-1, 1] em cada uma de suas coordenadas.

Fonte: autoria própria.

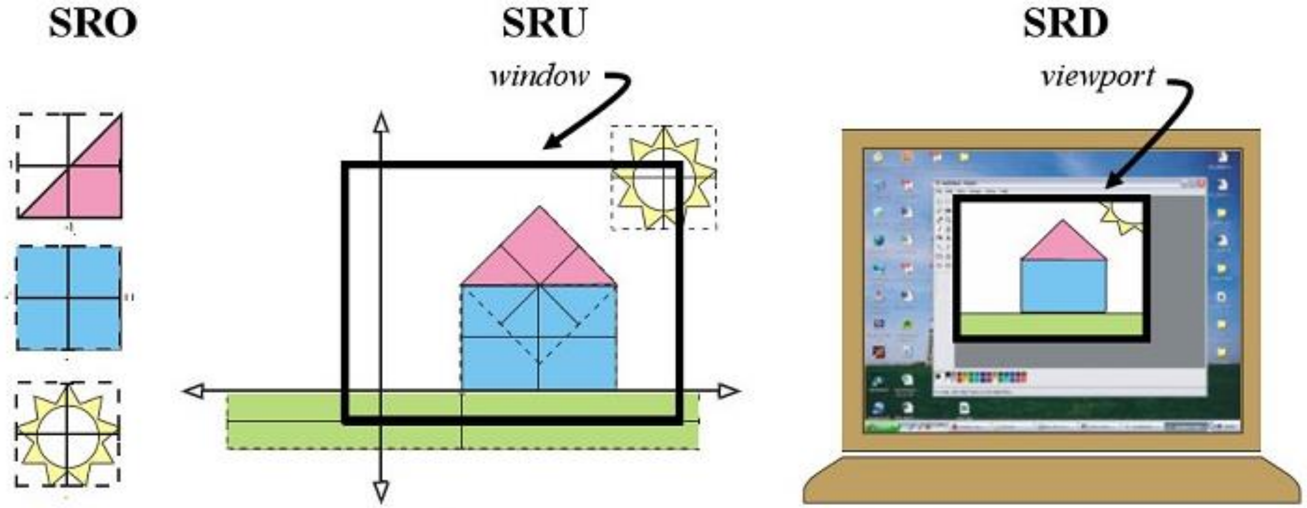


Figura 2. Esquema SRO → SRU → SRD. À esquerda, três primitivas gráficas (triângulo, quadrado, sol) modeladas no sistema de referência do objeto (SRO), normalizado no intervalo [-1,1]. Ao centro, uma composição com essas primitivas, devidamente transformadas (rotação, escala e translação) inseridas no sistema de referência do universo (SRU) com a respectiva *window*. À direita, um exemplo de dispositivo de saída gráfica com uma *viewport* exibindo a cena contida na *window*. Observe as distorções e recortes da cena. O piso é formado pela mesma primitiva quadrada repintada de verde.

Primitivas gráficas em duas dimensões

Sejam, portanto, XRMIN e XRMAX os valores mínimos e máximos horizontais da janela no espaço cartesiano (*window*) e XPMIN e XPMAX os valores correspondentes em *pixel* (*viewport*). YRMIN, YRMAX, YPMIN e YPMAX serão os valores limites na vertical (cf. figura 3). A transformação (XR,YR) à (YP,YP) é obtida por meio de um simples cálculo de PROPORCIONALIDADE (regra de três simples!). Para a coordenada horizontal (coordenada x), a transformação é:

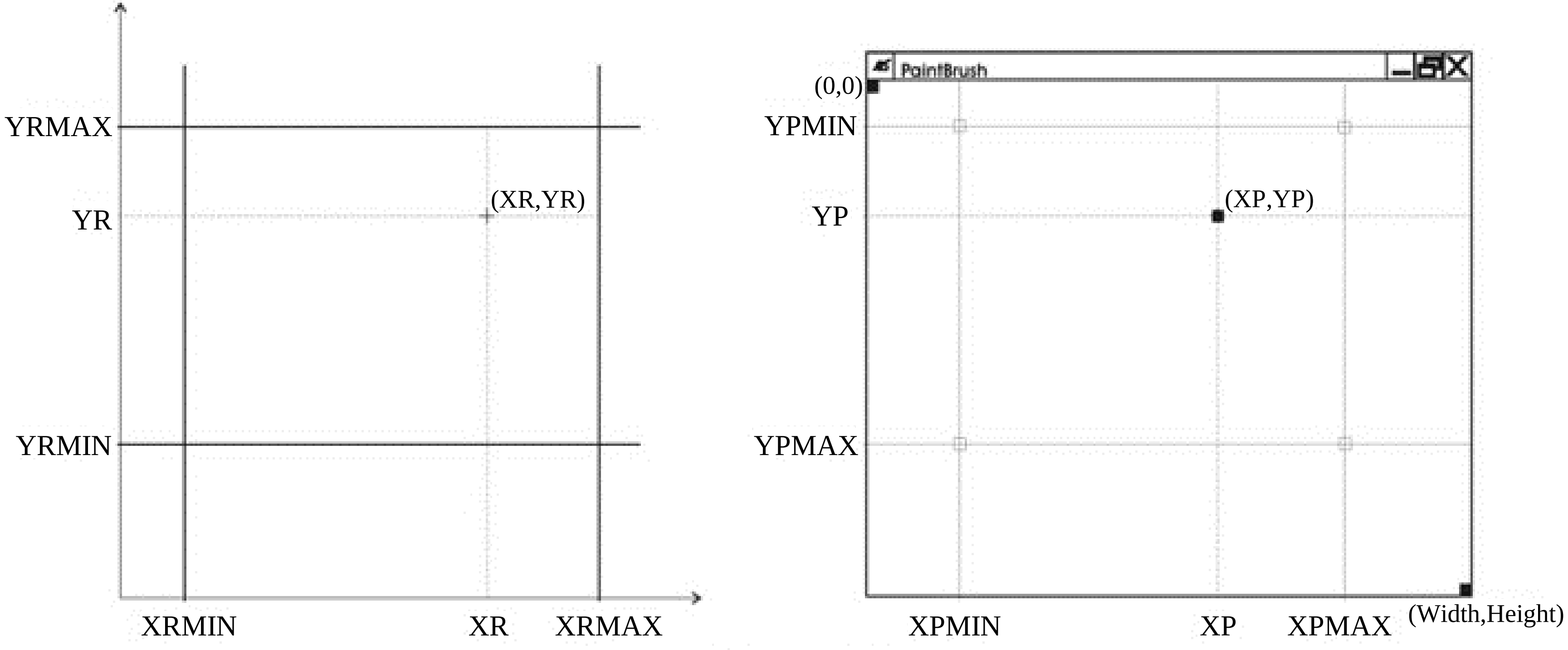


Figura 3. Rasterização de ponto

Fonte: autoria própria.

Primitivas gráficas em duas dimensões

Resolvendo XP em função de XR:

$$\frac{XPMAX - XP}{XR - XRMIM} = \frac{XPMAX - XPMIM}{XRMAX - XRMIN}$$

Denominando-se por $SX = \frac{XPMAX - XPMIM}{XRMAX - XRMIN}$ como fator de escala horizontal, temos:

$$\frac{XPMAX - XP}{XR - XRMIM} = SX$$

$$-XP = SX(XR - XRMIN)$$

$$\mathbf{XP = SX(XR + XRMIN)}$$

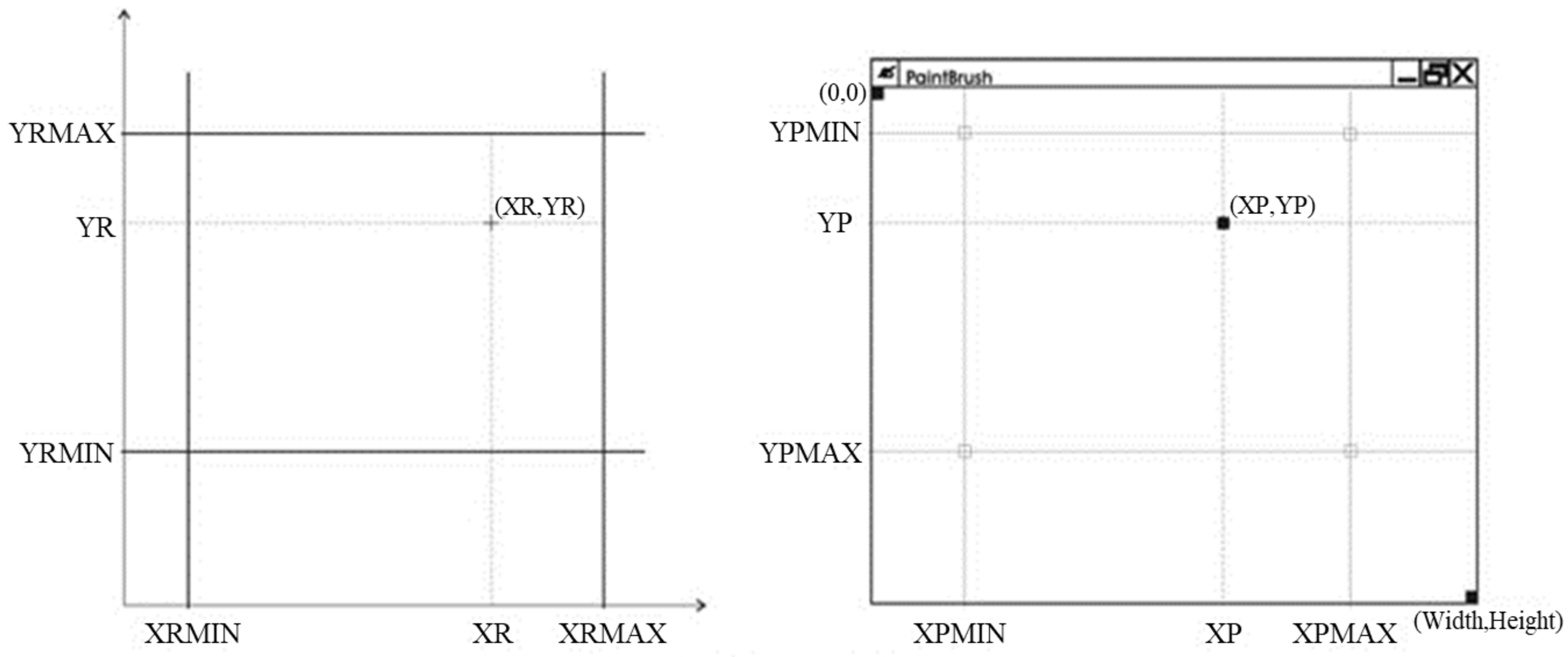


Figura 3. Rasterização de ponto

Fonte: autoria própria.

Primitivas gráficas em duas dimensões

E XR em função de XP*:*

$$\frac{XR - XRMIN}{XP - XPMIN} = \frac{XRMAX - XRMIM}{XPMAX - XPMIN}$$

Lembrando que $\mathrm{SX} = \frac{\mathrm{XPMAX - XPMIM}}{\mathrm{XRMAX - XRMIN}}$ como fator de escala horizontal, temos:

$$\frac{\mathrm{XR - XRMIN}}{\mathrm{XP - XPMIN}} = \frac{1}{\mathrm{SX}}$$

$$\mathrm{XR - XRMIN} = \frac{\mathrm{XP - XPMIN}}{\mathrm{SX}}$$

$$\mathbf{XR} = \frac{\mathbf{XP - XPMIN}}{\mathbf{SX}} + \mathbf{XRMIN}$$

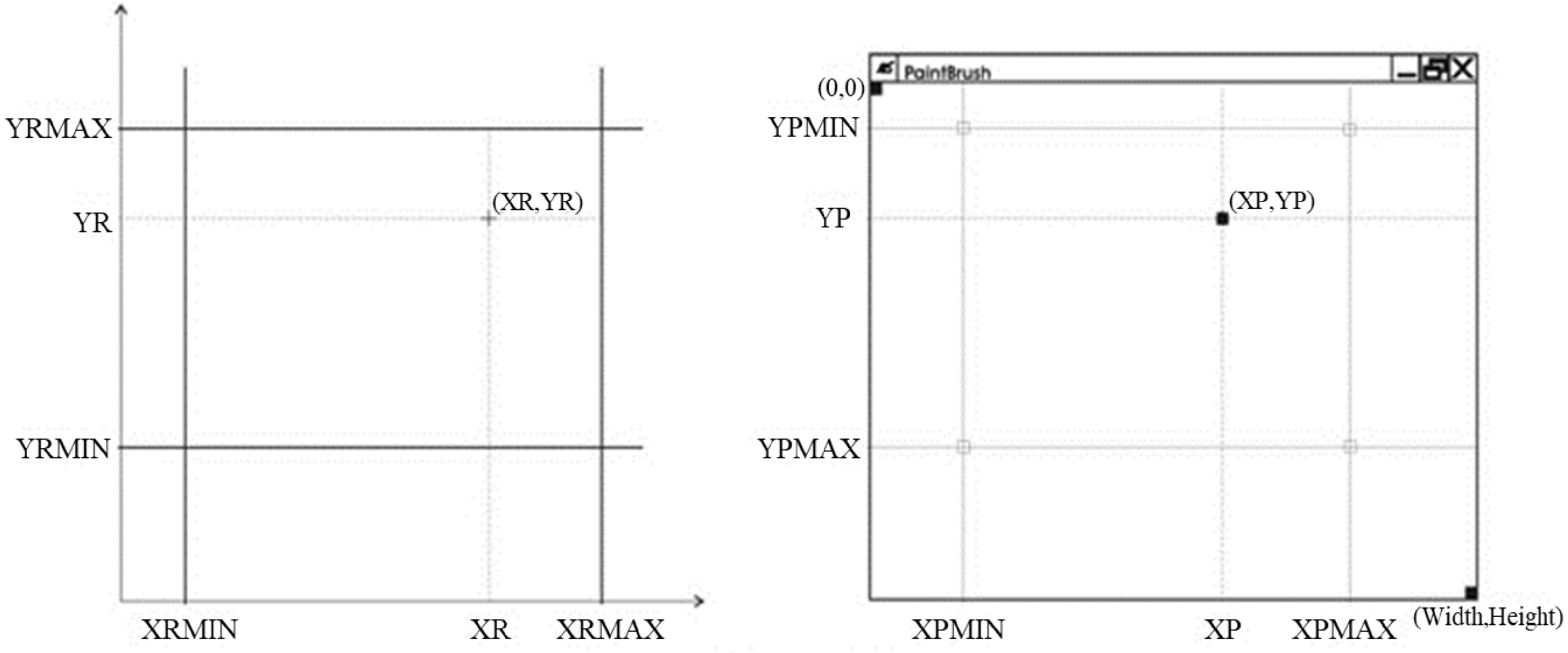


Figura 3. Rasterização de ponto

Fonte: autoria própria.

Primitivas gráficas em duas dimensões

- Expressões análogas são obtidas da mesma forma para a transformação da coordenada vertical (Y).

- Ao implementarmos essas expressões em alguma linguagem de programação, devemos observar os tipos primitivos das variáveis (XR, YR) e (XP, YP). As primeiras são de tipo real, ponto flutuante, e as outras são de tipo inteiro, o que implica em converter adequadamente os tipos.

Primitivas gráficas em duas dimensões

Exemplo: obtenha as relações de YP em função de YR e de YR em função de YP. Considere que os limites da *window* e da *viewport* na vertical são, respectivamente, (YRMIN, YRMAX) e (YPMIN, YPMAX).

$$SY = \frac{YPMAX - YPMIN}{YRMAX - YRMIN}$$

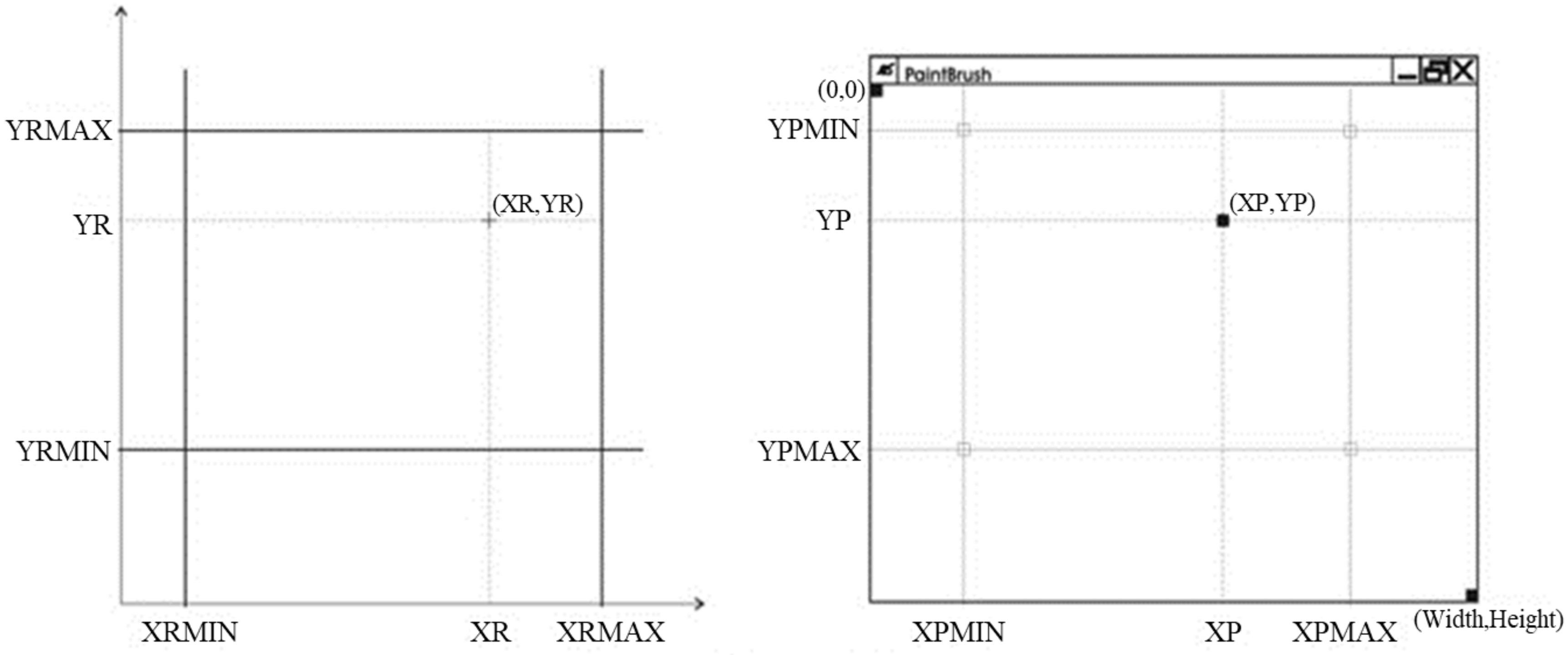


Figura 3. Rasterização de ponto

Fonte: autoria própria.

Primitivas gráficas em duas dimensões

RESOLUÇÃO - YP EM FUNÇÃO DE YR

Dado: $SY = \frac{YPMAX - YPMIM}{YRMAX - YRMIN}$

$$\frac{YP - YPMIN}{YRMAX - YR} = \frac{YPMAX - YPMIN}{YRMAX - YRMIN}$$

$$\frac{YP - YPMIN}{YRMAX - YR} = SY$$

$$YP - YPMIN = SY(YRMAX - YR)$$

$$\mathbf{YP = SY(YRMAX - YR) + YPMIN}$$

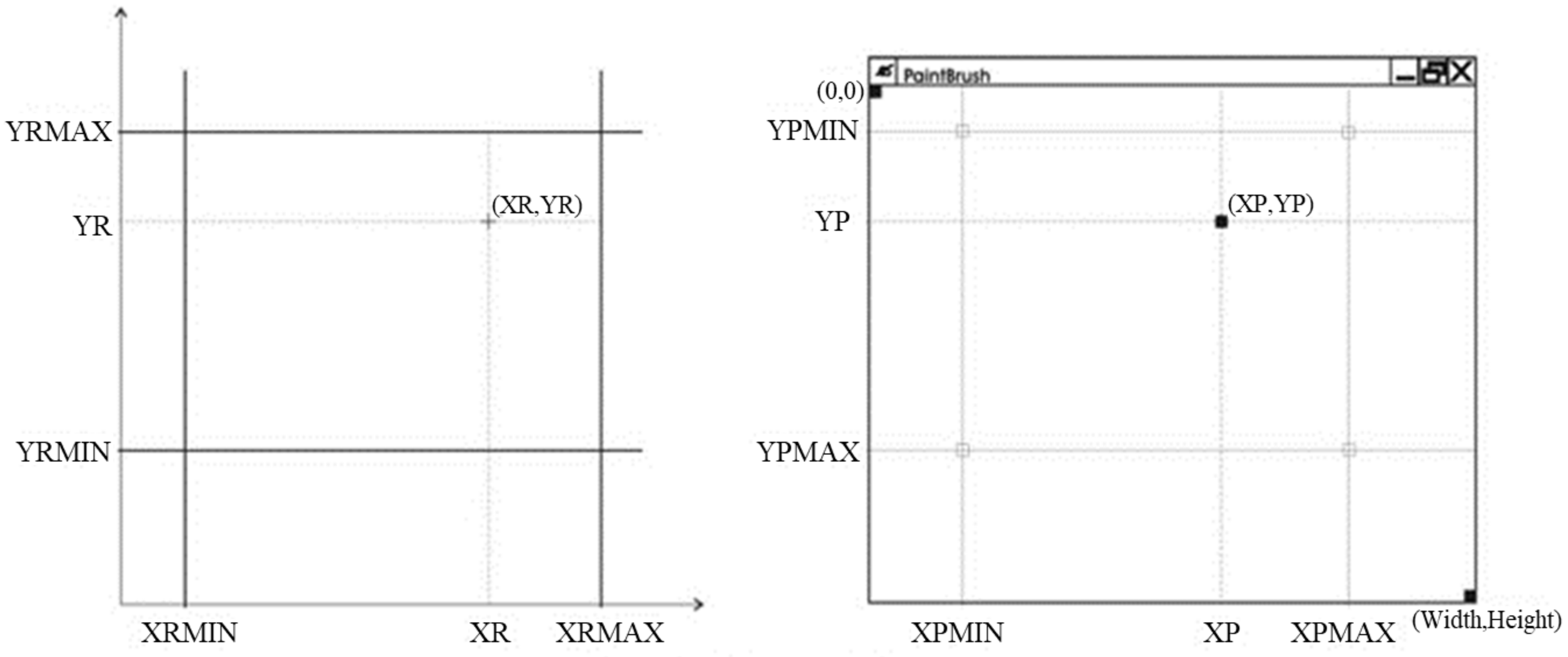


Figura 3. Rasterização de ponto

Fonte: autoria própria.

Primitivas gráficas em duas dimensões

RESOLUÇÃO - YR EM FUNÇÃO DE YP

Dado: $SY = \frac{YPMAX - YPMIN}{YRMAX - YRMIN}$

$$\frac{YR - YRMIN}{YPMAX - YP} = \frac{YRMAX - YRMIN}{YPMAX - YPMIN}$$

$$\frac{YR - YRMIN}{YPMAX - YP} = \frac{1}{SY}$$

$$\mathbf{YR = \frac{YPMAX - YP}{SY} + YRMIN}$$

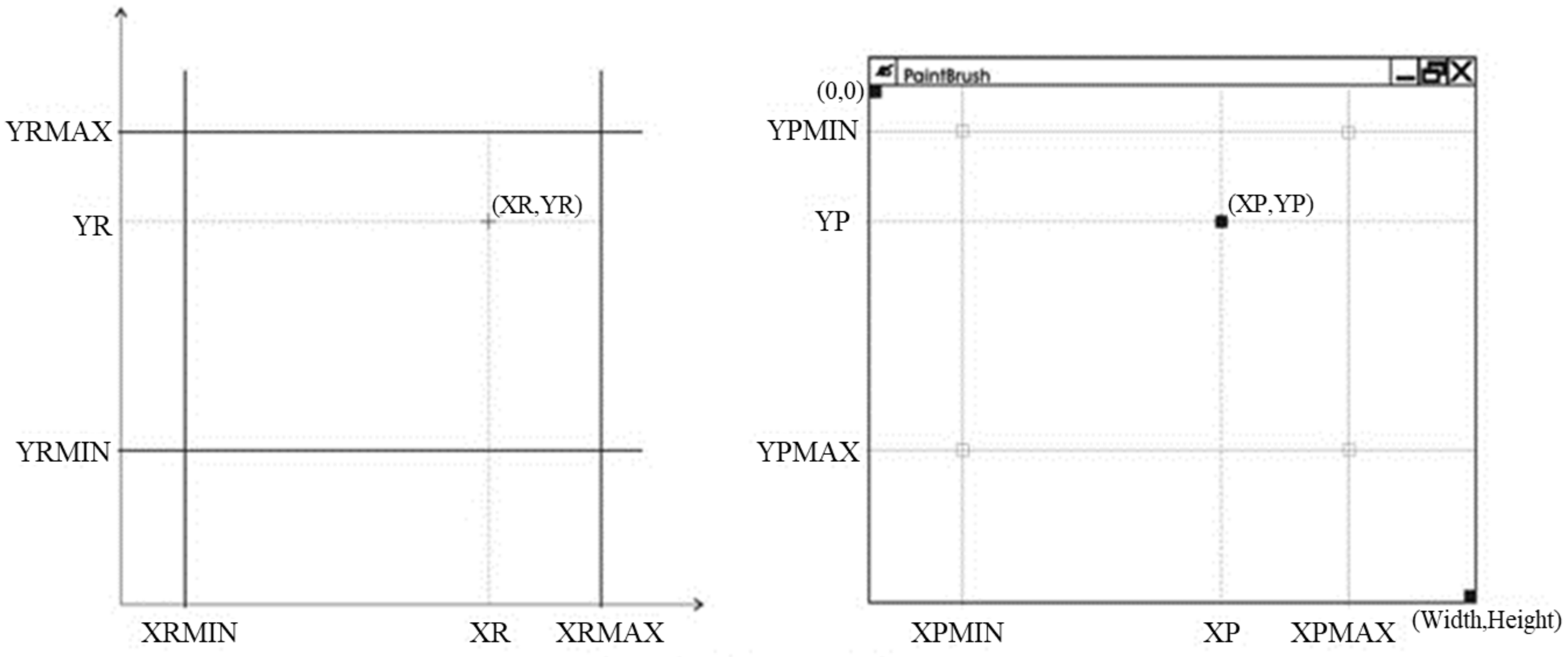


Figura 3. Rasterização de ponto

Fonte: autoria própria.

Interatividade

Assinale a alternativa correta. As propriedades fundamentais do *pixel* são:

a) Cor e forma.
b) Área e forma.
c) Área e posição.
d) Cor e posição.
e) *Pixel* não tem propriedades.

Resposta

Assinale a alternativa correta. As propriedades fundamentais do *pixel* são:

a) Cor e forma.
b) Área e forma.
c) Área e posição.
d) Cor e posição.
e) *Pixel* não tem propriedades.

Rasterização de linhas

- Vimos como transformar as coordenadas reais de um ponto nas coordenadas de um *pixel* no interior de uma janela de visualização (mapeamento *window-to-viewport*). Não devemos nos esquecer que esse mapeamento converte tanto pontos do SRU em *pixels* do SRD como o contrário, ou seja, dado um *pixel* no SRD calcula-se o ponto no SRU. Desta vez, vamos ligar dois *pixels* para obtermos uma linha reta.

- A motivação é simples. Em vez de mapearmos do SRU para o SRD cada ponto de uma aresta que liga dois vértices, mapeamos apenas os vértices no interior da *viewport* e completamos as arestas com um algoritmo que considere apenas os *pixels* do SRD.

Rasterização de linhas

- Sejam XRi e YRi as coordenadas de um ponto inicial Pi, e XRf e YRf as coordenadas de um ponto Pf, final, ambos pertencentes à R2 (logo, ao SRU). A equação da reta que passa por Pi e Pf é dada por: y = mx + b.

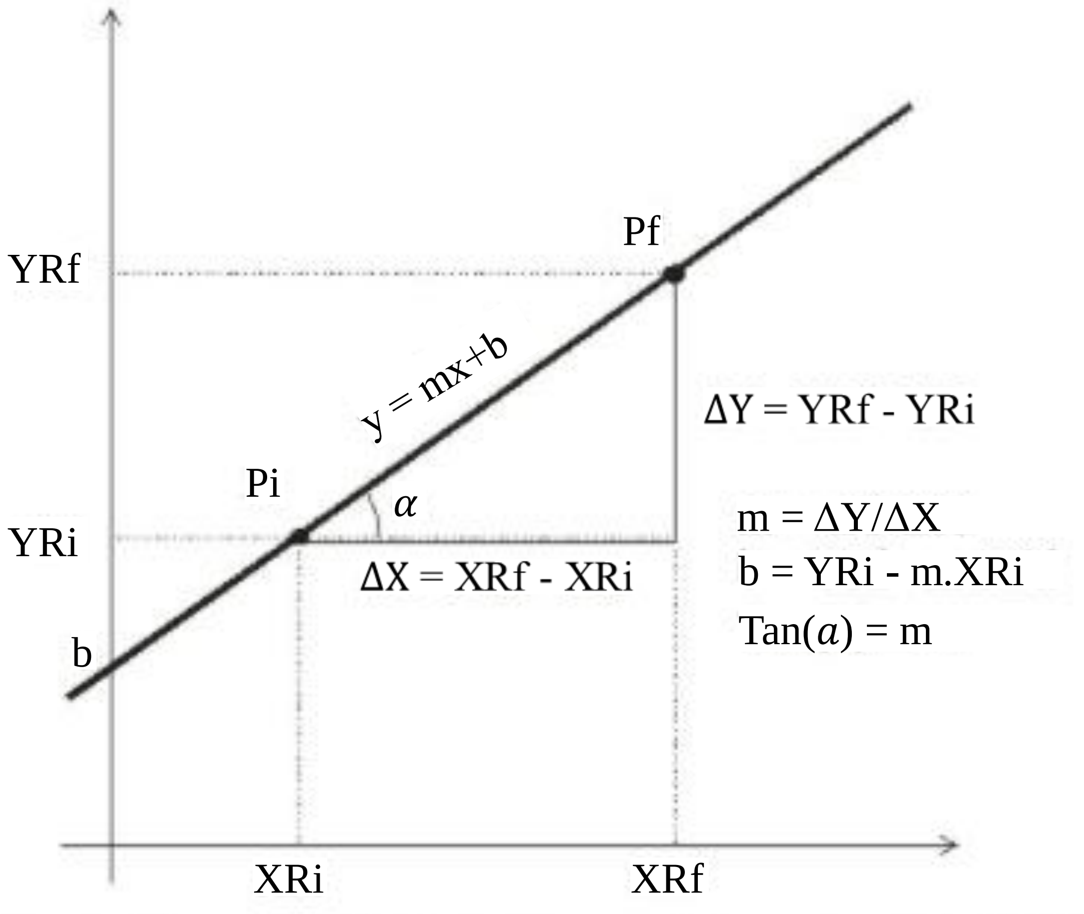


Figura 1. Elementos da reta que passa por dois pontos dados (Pi, Pf)

Fonte: autoria própria.

Rasterização de linhas

- É claro que, em um primeiro momento, precisamos transformar as coordenadas dos pontos Pi e Pf em *pixels*, coordenadas da janela de visualização (*viewport*). Para tanto, procedemos como no módulo anterior. Na maioria dos casos, contudo, já temos os pontos Pi e Pf em *pixels*.
- Por exemplo: Em um programa de desenho tipo PaintBrush, na ferramenta "linha", pressionamos o botão do *mouse* para indicar o *pixel* inicial (Pi). Em seguida, arrastamos o ponteiro pela tela com o botão do *mouse* pressionado e soltamos, indicando o ponto final (Pf). Nesse caso, observe, não houve necessidade de transformação de pontos reais em *pixels*. No entanto, em programas como CAD/CAM ou programas que desenham gráficos (Excel, Grapher etc.), o mapeamento *window-to-viewport* dos pontos Pi e Pf é necessário.
- No entanto, seja em um caso ou no outro, o PREENCHIMENTO do espaço entre os pontos para gerar uma linha reta pode ser feito mais eficientemente calculando-se apenas os *pixels* que devem ser ativados MAIS PRÓXIMOS da reta teórica. A RASTERIZAÇÃO DE LINHAS é, portanto, efetuada no espaço dos *pixels*, por meio de um algoritmo que ligue os dois pontos dados na *viewport* do SRD.

Rasterização de linhas

- Há vários algoritmos para esse fim.

- O mais simples pertence à categoria chamada DDA (*Digital Differencial Analizer*). Basicamente, os DDAs são ALGORITMOS INCREMENTAIS em uma das variáveis, que se utilizam diretamente da definição de reta dada por sua equação: $y = mx+b$.

- Algoritmos incrementais são aqueles em que uma das variáveis é obtida apenas incrementando o seu valor, por exemplo: $X = X + 1$, e a outra é calculada por alguma regra a partir da primeira.

Rasterização de linhas

Primeiro calculamos os parâmetros da reta e inicializamos variáveis:

- n = XPf-XPi
- m = (YPf-YPi)/n
- X0 = XPi
- Y0 = Ypi

- Observe que “n” é o número de colunas de *pixels* entre Pi e Pf e que Y0 também pode ser entendido como Y0 = m*X0 + b.

Rasterização de linhas

Em seguida incrementamos sucessivos Xs e calculamos os Ys correspondentes pela definição da reta:

- $x_1 = x_0 + 1$ e $y_1 = mx_1 + b = m(x_0 + 1) + b = mx_0 + m + b$
- logo $y_1 = y_0 + m$

Continuando (faça as contas!)...

- $x_2 = x_1 + 1,\ então\ x_2 = x_0 + 2,\ logo\ y_2 = y_0 + 2m$
- $x_3 = x_0 + 3,\ logo\ y_3 = y_0 + 3m$

...

Por indução, obtemos uma expressão geral para o j-ésimo ponto:

- $x_j = x_0 + j$
- $y_j = y_0 + jm$

O último ponto será o ponto n

- $x_n = x_0 + n$
- $y_n = y_0 + nm\sim YPf$

Rasterização de linhas

- Observe que Yn ~ YPf, porque o resultado da expressão n*m é truncado (ou, se preferir, arredondado), introduzindo um pequeno erro que pode ser de até um *pixel*.

- Os pontos fracos nos algoritmos DDAs são o uso da ARITMÉTICA REAL e uma pequena imprecisão por ERROS DE ARREDONDAMENTO.

- Além dessas desvantagens, que são bastante indesejáveis, o algoritmo apresenta uma restrição importante: ele funciona apenas para retas em que $|m| < 1$, ou seja, retas com inclinações entre -45° e 45°. Para obtermos corretamente as demais inclinações, devemos lançar mão de SIMETRIAS.

Rasterização de linhas

- Por exemplo, podemos dividir o plano cartesiano em oito fatias, ou OCTANTES. Para cada um deles, as retas assumem direções diferentes. O algoritmo descrito foi pensado para servir no PRIMEIRO OCTANTE, mas funciona também no OITAVO OCTANTE. Com uma pequena modificação pode-se facilmente estendê-lo para o QUARTO e o QUINTO OCTANTES (pense em como isso pode ser feito!). Os demais também podem ser obtidos a partir dessas extensões transpondo as variáveis X <=> Y.

Rasterização de linhas

Dados:

- $X0 = 6$, $Y0 = 9$, $m = 3/5 = 0,6$

- $X1 = X0 + 1 = 6 + 1 = 7$
- $Y1 = Y0 + m = 9 + 0,6 = 9,6 \Rightarrow (X1; Y1) = (7; 9,6)$

- $X2 = X0 + 2 = 6 + 2 = 8$
- $Y2 = Y0 + 2m = 9 + 1,2 = 10,2 \Rightarrow (X2; Y2) = (8; 10,2)$

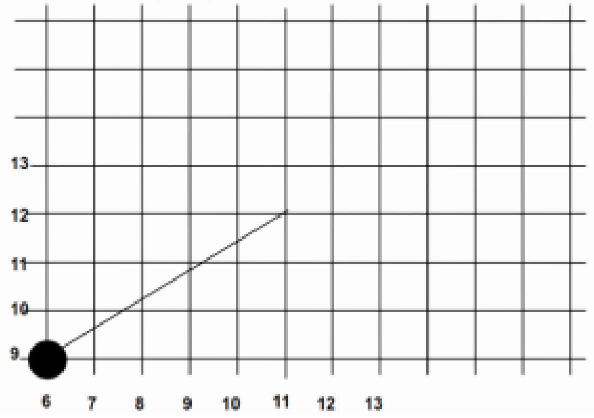
Exemplo: Usando o DDA, compute quais *pixels* devem ser escolhidos para representar a reta de (6,9) a (11,12)

```
step = max of fabs(Y2-Y1),
fabs(X2-X1) ;

Xinc = (X2 - X1)/step;
Yinc = (Y2 - Y1)/step;
```

```
step = Max of (fabs(11-6),
fabs(12-9)) = 5

Xinc = (11-6)/5 = 1
Yinc = (12-9)/5 = 0,6
```




```
X = X + Xinc
Y = Y + Yinc
```

Fonte: autoria própria.

- $X3 = X0 + 3 = 6 + 3 = 9$
- $Y3 = Y0 + 3m = 9 + 1,8 = 10,8 \Rightarrow (X3; Y3) = (9; 10,8)$

- $X4 = X0 + 4 = 6 + 4 = 10$
- $Y4 = Y0 + 4m = 9 + 2,4 = 11, 4 \Rightarrow (X4; Y5) = (10; 11,4)$

- $X5 = X0 + 5 = 6 + 5 = 11$
- $Y5 = Y0 + 5m = 9 + 3 = 12 \Rightarrow (X5; Y5) = (11; 12)$

Rasterização de linhas

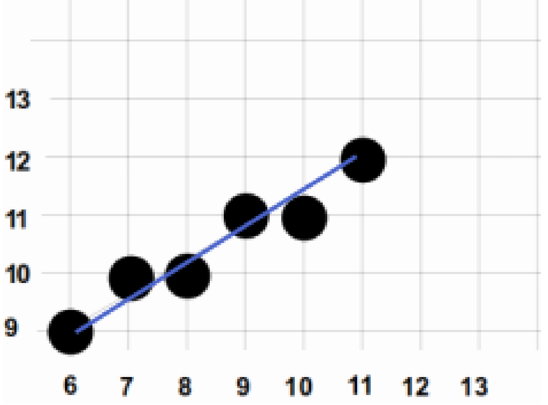
Dados:

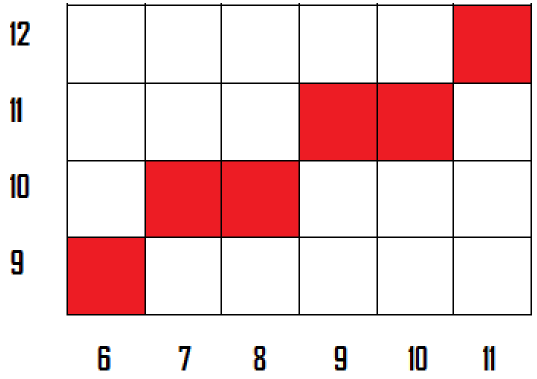
- $X0 = 6$, $Y0 = 9$, $m = 3/5 = 0,6$

- $X1 = X0 + 1 = 6 + 1 = 7$
- $Y1 = Y0 + m = 9 + 0,6 = 9,6 \Rightarrow (X1; Y1) = (7; 9,6)$

- $X2 = X0 + 2 = 6 + 2 = 8$
- $Y2 = Y0 + 2m = 9 + 1,2 = 10,2 \Rightarrow (X2; Y2) = (8; 10,2)$

Os pontos encontrados são:
(6,9), (7,9.6),
(8,10.2), (9,10.8),
(10,11.4), (11,12)





Fonte: autoria própria.

- $X3 = X0 + 3 = 6 + 3 = 9$
- $Y3 = Y0 + 3m = 9 + 1,8 = 10,8 \Rightarrow (X3; Y3) = (9; 10,8)$

- $X4 = X0 + 4 = 6 + 4 = 10$
- $Y4 = Y0 + 4m = 9 + 2,4 = 11, 4 \Rightarrow (X4; Y5) = (10; 11,4)$

- $X5 = X0 + 5 = 6 + 5 = 11$
- $Y5 = Y0 + 5m = 9 + 3 = 12 \Rightarrow (X5; Y5) = (11; 12)$

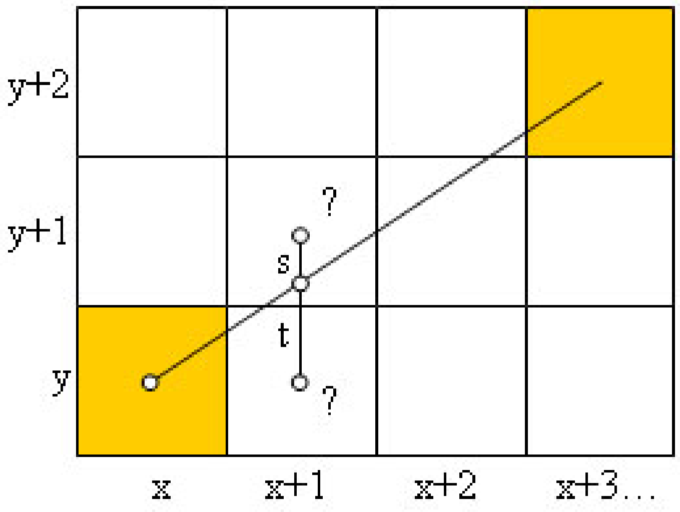
Algoritmo de Breseham para segmentos de retas

- Outra família de algoritmos é formada pelo ALGORITMO DE BRESENHAM ou pelo ALGORITMO DO PONTO MÉDIO.

- Esses algoritmos não usam diretamente a definição da reta, mas sim as DIFERENÇAS, ou as distâncias (erro) entre os *pixels* adjacentes à reta desejada. Como nos DDAs, são algoritmos incrementais. Entretanto, diferentemente, AMBAS AS VARIÁVEIS são incrementadas de uma quantidade inteira. Assim, para cada X = X + 1, o valor de Y é decidido pelo algoritmo incrementando-se ou não em função de alguns testes de diferenças. Ou seja, faz-se Y = Y ou Y = Y + 1 conforme o caso.

Algoritmo de Breseham para segmentos de retas

- O algoritmo de Bresenham baseia-se no argumento de que um segmento de reta, ao ser plotado, deve ser contínuo, ou melhor, os *pixels* que compõem um segmento de reta devem ser vizinhos.
- Uma vez que o algoritmo também é pensado para o PRIMEIRO OCTANTE, discutiremos o caso de 0 < m < 1.
- Aqui, o ponto de partida é a seguinte pergunta: se 0 < m < 1, e dado um ponto de um segmento de reta (x,y), o próximo *pixel* a ser pintado será o (x+1, y) ou o (x+1, y+1)? O algoritmo de Breseham responde essa questão calculando uma variável de teste (p no algoritmo dado abaixo) para cada *pixel*, e passando para o *pixel* seguinte, até alcançar o último *pixel* do segmento de reta.



Fonte: autoria própria.

Algoritmo de Breseham para segmentos de retas

- Vejamos como podemos descrever o algoritmo de Bresenham.
- Parâmetros de entrada: (xi, yi) e (xf, yf) (pontos inicial e final do segmento de reta).

1. Calcula-se Dx = xf – xi e Dy = yf – yi.
2. Coloca-se nas variáveis de trabalho o ponto inicial: x = xi e y = yi.
3. Calcula-se o parâmetro de decisão: p = 2Dy – Dx.
4. Plota-se o ponto (x, y).
5. Se p for negativo (isto é, se p < 0) então: x = x + 1, p = p + 2Dy e passa-se para o passo 7.
6. Se p for positivo ou zero, então:

   x = x + 1

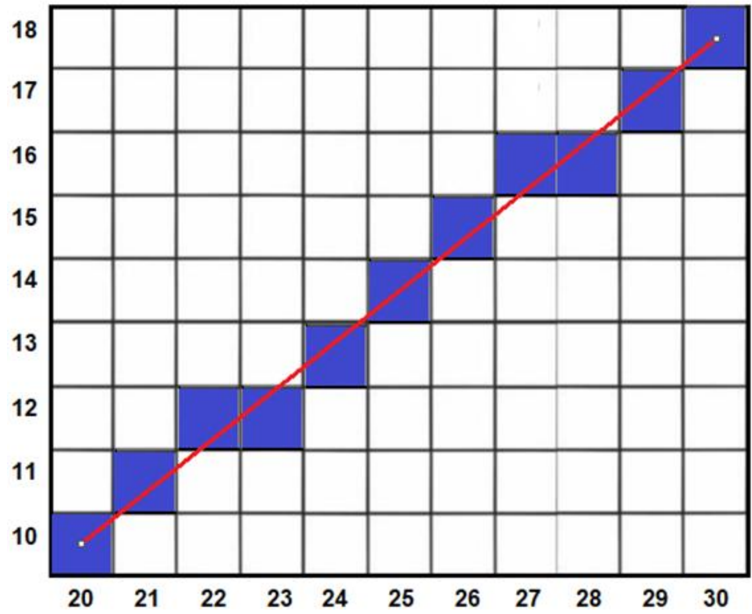
   y = y + 1

   p = p + 2Dy - 2Dx
7. Repete-se os passos 4 a 6 até que o ponto (xf, yf) seja alcançado.

Algoritmo de Breseham para segmentos de retas

Exemplo: Pontos (20,10) e (30,18)

- Dx = x2 – x1 = 30 – 20 dx = 10 dy = y2 – y1 = 18 – 10 dy = 8
- p0 = 2dy – dx = 2*8 – 10 = 6, então p0 = 6, p0 > 0, então (21,11)
- p1 = p0 + 2dy – 2dx = 6 + 16 – 20 = 2, p1 > 0, então (22,12)
- p2 = p1 + 2dy – 2dx = 2 + 16 – 20 = – 2, p2 < 0, então (23,12)
- p3 = p2 + 2dy = -2 + 16 = 14, p3 > 0, então (24,13)
- p4 = p3 + 2dy – 2dx = 14 + 16 – 20 = 10, p4 > 0, então (25,14)
- p5 = p4 + 2dy – 2dx = 10 + 16 – 20 = 6, p5 > 0, então (26,15)



Fonte: autoria própria.

- A partir daqui, os parâmetros se repetem. Logo, os próximos pontos serão (27,16); (28,16); (29,17) e (30,18).

Interatividade

“Efeito gráfico que busca corrigir imperfeições existentes em objetos tridimensionais, tais como os famosos serrilhados encontrados nas bordas de objetos e em volta de pequenas estruturas (como cercas, cabelos e folhas de árvore).” Analisando essa afirmação é correto dizer que ela se refere a que tipo de efeito? Quais técnicas são específicas desse efeito?

a) Rasterização, *Raster-Ray* e *Low-Filter*.

b) *Antialising*, *Super-Sampling* e *Multi-Sampling*.

c) *Super-Sampling*, *Ray-Faster* e *Low-Filter*.

d) Cisalhamento, *Raster-Ray* e *Low-Filter*.

e) Cisalhamento, *Low-Filter* e *Multi-Sampling*.

Resposta

“Efeito gráfico que busca corrigir imperfeições existentes em objetos tridimensionais, tais como os famosos serrilhados encontrados nas bordas de objetos e em volta de pequenas estruturas (como cercas, cabelos e folhas de árvore).” Analisando essa afirmação é correto dizer que ela se refere a que tipo de efeito? Quais técnicas são específicas desse efeito?

a) Rasterização, *Raster-Ray* e *Low-Filter*.

b) *Antialising*, *Super-Sampling* e *Multi-Sampling*.

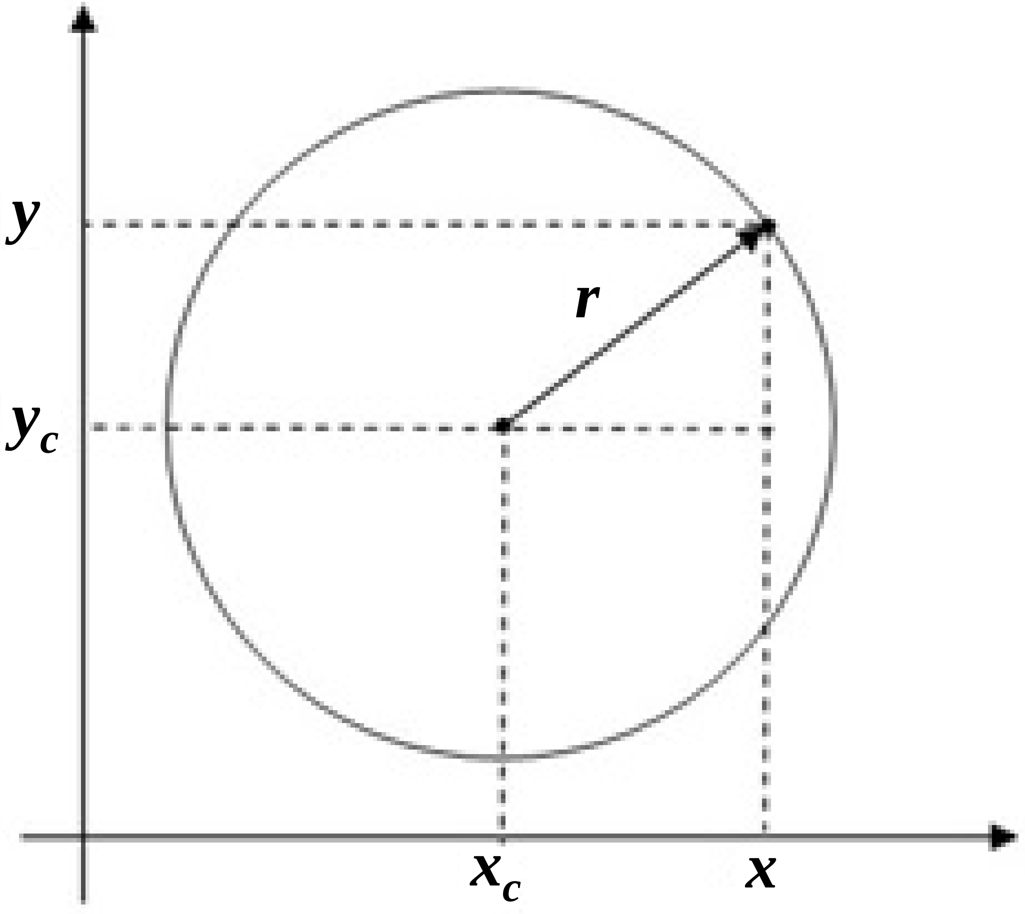
c) *Super-Sampling*, *Ray-Faster* e *Low-Filter*.

d) Cisalhamento, *Raster-Ray* e *Low-Filter*.

e) Cisalhamento, *Low-Filter* e *Multi-Sampling*.

Rasterização de curvas

- Veremos métodos para traçarmos circunferências de círculos e circunferências de elipses usando a aritmética inteira do algoritmo de Bresenham.



Fonte: autoria própria.

Figura 1: Geometria de circunferência

Rasterização de curvas

- Além desses algoritmos, bastante eficientes, mas limitados às formas integrais que expressam, veremos também como podemos traçar a maioria das curvas e arcos usando coordenadas polares. Infelizmente, esses algoritmos padecerão da aritmética de ponto flutuante, por conta das funções SENO e COSSENO que entram nas suas definições. Para adiantar, por favor, dê uma boa olhada nas funções trigonométricas seno, cosseno e tangente para relembrar seus conceitos e aplicações.

Fonte: autoria própria.

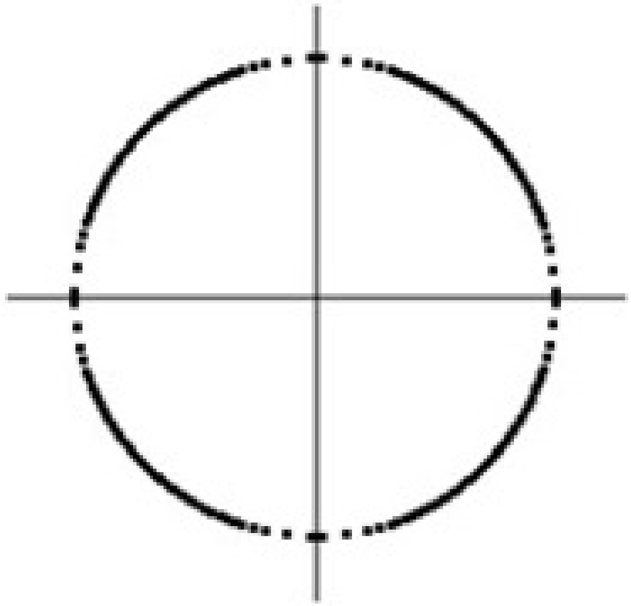


Figura 2: Imprecisões no traçado de circunferências

Rasterização de curvas

- Uma circunferência de círculo, ou simplesmente circunferência, é definida como sendo o conjunto dos pontos que estão a uma mesma distância de um ponto central. A distância é o raio (r) da circunferência e o ponto equidistante a todos os outros é o centro da circunferência (xc, yc). Matematicamente: $(x - x_c)^2 + (y - y_c)^2 = r^2$

- (xc, yc) é o centro da circunferência e r é o raio.

- Essa equação não é conveniente para ser usada em computação gráfica, pois sequer é uma função (lembra-se da definição de função?), porquanto deseja-se uma definição do tipo y = f(x) ou x = g(y).

Essas definições podem ser facilmente obtidas isolando-se as variáveis:

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

Rasterização de curvas

- A operação ± é necessária, pois a raiz quadrada pode ser positiva ou negativa. Isso significa que para cada valor de x (na primeira expressão) são obtidos dois valores de y: um para a metade superior da circunferência e o outro para a metade inferior.

Essas expressões, apesar de rigorosamente corretas, apresentam dois problemas para serem usadas em computação gráfica:

- Exigem muitos cálculos (quadrados e raiz quadrada).

- Geram imprecisão no traçado. Quando o arco de circunferência fica quase horizontal ou vertical, um pequeno incremento em $x = x + 1$ (ou $y = y + 1$) leva a um salto, e o arco apresentar-se-á descontínuo.

Rasterização de curvas

Pode-se chegar a uma outra formulação da circunferência convertendo o sistema de coordenadas cartesiano (x, y) para um sistema de coordenadas polares (r, q). As novas expressões apresentam pontos de uma circunferência como função do raio e de um ângulo:

$$\begin{cases} x = x_c + rcos\theta \\ y = y_c + rsen\theta \end{cases}$$

θ (téta) é um ângulo que varia entre 0 e 2π (os ângulos devem ser tratados com unidades em radianos, 2 π radianos é igual a 360°). O número de passos deverá aumentar com o raio. Um algoritmo trivial de traçado de circunferências pode ser idealizado a partir da escolha de um passo para θ, que chamaremos de Dθ, e um *loop* que calcula os valores de (x, y) para vários valores de θ. Cada ponto calculado é inserido numa tabela que mais tarde é fornecida como parâmetro para um traçado de uma polilinha fechada. A circunferência assim desenhada será na realidade um polígono, cuja quantidade de lados está relacionada à precisão desejada:

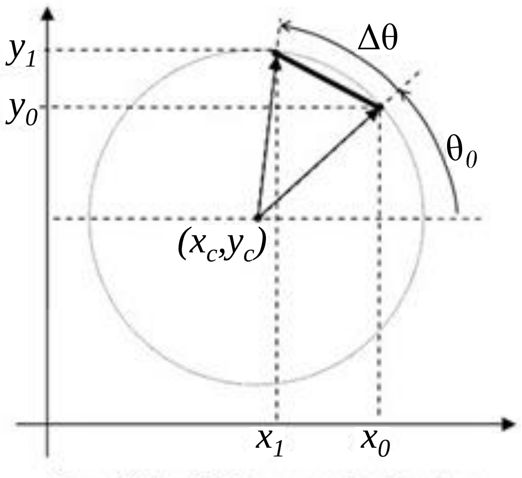


Figura 3: Circunferência em coordenadas polares

Fonte: autoria própria.

Rasterização de curvas

1. Lê-se as coordenadas do centro (xc, yc) e o raio r.
2. Lê-se o número (n) de vezes em que será dividida a circunferência.
3. Calcula-se Dθ = 2p/n.
4. Para i de 0 até n, faz-se:
   q = i *Dθ
   x = xc + r cosθ
   y = yc + r senθ
5. Converte-se (x, y) para *pixels* da *viewport*.
6. Une-se os *pixels* na sequência com linhas retas usando-se, por exemplo, o algoritmo de Bresenham.

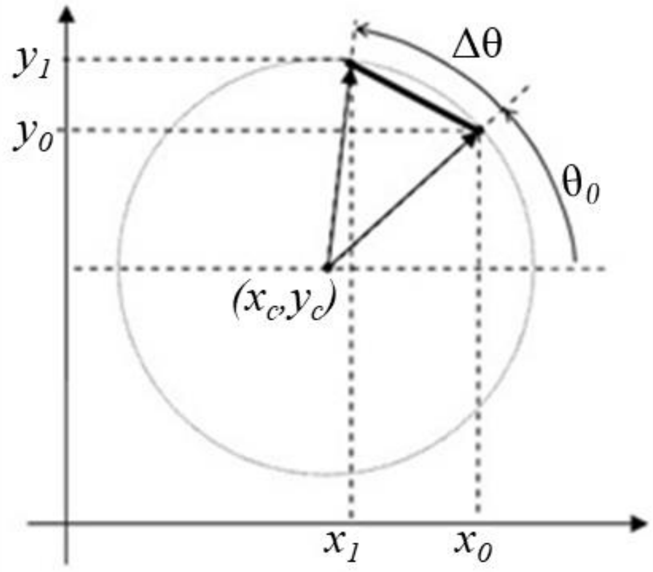


Figura 3: Circunferência em coordenadas polares

Fonte: autoria própria.

- Note que os passos acima servem não apenas para o traçado de circunferências, mas para qualquer curva representável em coordenadas polares. Para tanto, basta substituir, convenientemente, a fórmula do passo 4.
- Esse algoritmo apresenta dois defeitos: a precisão é dependente do raio da circunferência e o cálculo de funções trigonométricas (senos e cossenos), apesar dos poucos pontos, implica em perda de tempo e agilidade.

Rasterização de curvas

- Algoritmo do ponto médio para CIRCUNFERÊNCIAS DE CÍRCULO.

- O algoritmo do ponto médio para circunferências foi desenvolvido com o objetivo de agilizar o traçado da circunferência. É otimizado levando-se em conta as simetrias presentes nas circunferências, a simetria por OCTANTES. O ponto calculado em um octante (um oitavo de circunferência, com θ indo de 0 até $\pi/4$ apenas, ou 45°) pode ser copiado para os outros octantes, conforme a tabela a seguir.

- Observe que os pontos calculados como (x, y), diferentemente dos algoritmos de rasterização de retas, são obtidos para o II OCTANTE, como veremos em seguida. Outra coisa muito importante referente às simetrias é a seguinte:

Rasterização de curvas

- Cada algoritmo incremental que se crie pode usar um critério próprio que não obedeça, necessariamente, o da figura 4. Portanto, a tabela 4 não é para ser decorada ou memorizada, mas deve ser ENTENDIDA. Dessa forma, o conceito de simetria como empregado em CG será muito mais útil e geral.

OCTANTE	X_n	Y_n
I	Y	X
II	$\boldsymbol{X}$	$\boldsymbol{Y}$
III	$-X$	Y
IV	$-Y$	X
V	$-Y$	$-X$
VI	$-X$	$-Y$
VII	X	$-Y$
VIII	Y	$-X$

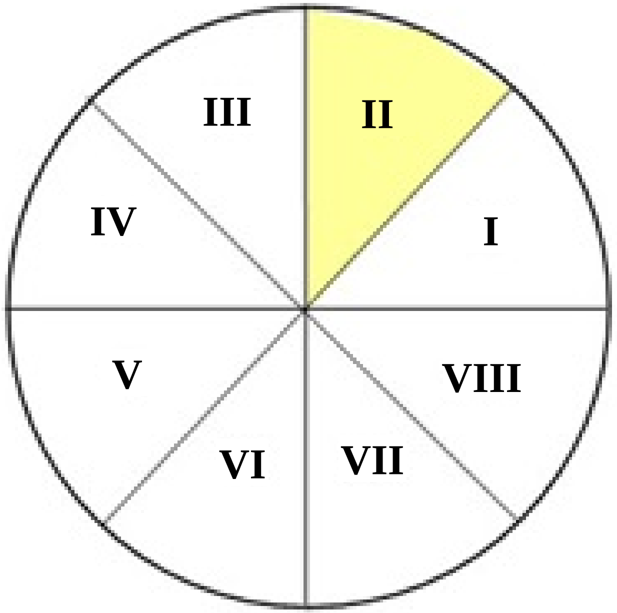


Figura 4: Simetria de octantes para o algoritmo de circunferência

Fonte: autoria própria.

Rasterização de curvas

- Algoritmo do ponto médio para círculos.
- O algoritmo do ponto médio para círculos foi desenvolvido com o objetivo de otimizar o traçado desse tipo de objeto.

1. Elege-se como ponto inicial (0,r) que é plotado nos oito octantes, e ainda calcula-se uma variável auxiliar p = 5/4 - r (ou simplesmente p = 1 - r).
2. Incrementa-se x.
3. Se p for negativo, então calcula-se um novo p = p + 2x + 1; caso contrário, decrementa-se y e calcula-se p = p + 2x + 1 - 2y.
4. Plota-se o novo ponto (x,y) nos oito octantes.
5. Os passos 2 e 4 são repetidos enquanto x < y.

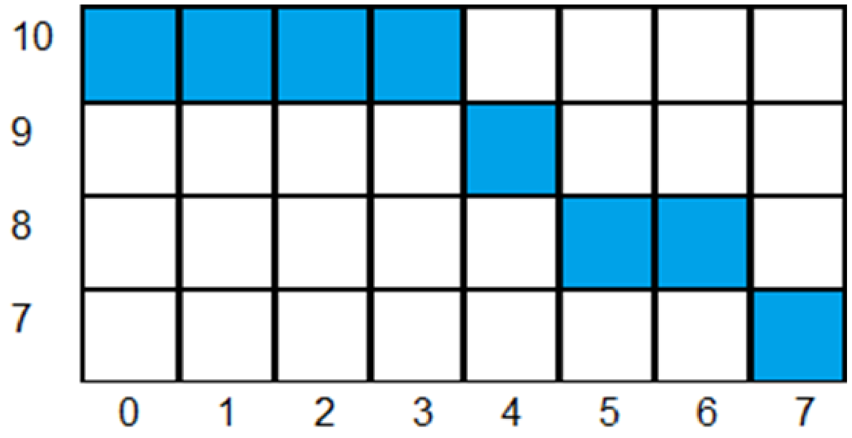
- A seguir é representada um simulação da execução desse algoritmo para um círculo de raio 10 *pixels*, com centro em (0,0).

Rasterização de curvas

- Ponto inicial (0,r) = (0,10)
- p = 1 – r
- p < 0 – p = p + 2x + 1
- p > 0 – p = p + 2x + 1 – 2y

Loop	Fórmula p	Valor de p	(x,y)	2x	2y
0	1 – 10	-9	(0,10)	0	20
1		-9	(1,10)	2	20
2	-9 + 2 + 1	-6	(2,10)	4	20
3	-6 + 4 + 1	-1	(3,10)	6	20
4	-1 + 6 + 1	6	(4,9)	8	18
5	6 + 8 + 1 – 18	-3	(5,9)	10	18
6	-3 + 10 + 1	8	(6,8)	12	16
7	8 + 12 + 1 – 16	5	(7,7)	14	14

Fonte: autoria própria.



Interatividade

Suponha que um programador tenha criado um algoritmo semelhante ao algoritmo de Bresenham só que calculando valores para o primeiro octante. Então, x e y são valores no primeiro octante. Assinale a alternativa que indica corretamente a relação de simetria para o segundo octante. Considere que (xc, yc) são as coordenadas do centro no SRD:

a) PLOTA (x, y)

b) PLOTA (xc + x, yc + y)

c) PLOTA (yc + y, xc + x)

d) PLOTA (yc + x, xc + y)

e) PLOTA (xc + y, yc + x)

Resposta

Suponha que um programador tenha criado um algoritmo semelhante ao algoritmo de Bresenham só que calculando valores para o primeiro octante. Então, x e y são valores no primeiro octante. Assinale a alternativa que indica corretamente a relação de simetria para o segundo octante. Considere que (xc, yc) são as coordenadas do centro no SRD:

a) PLOTA (x, y)
b) PLOTA (xc + x, yc + y)
c) PLOTA (yc + y, xc + x)
d) PLOTA (yc + x, xc + y)
e) PLOTA (xc + y, yc + x)

Referências

- MACHADO, F. C. *Primeiro projeto de análise numérica II*. Curvas de Bézier e Desenho de Fontes Tipográficas. UNICAMP. Disponível em: https://www.ime.unicamp.br/~sandra/MS612/handouts/Proj1Tema4.pdf. Acesso em: 04 abr. 2023.

ATÉ A PRÓXIMA!