Desenho de primitivas vetoriais em Canvas no MATLAB

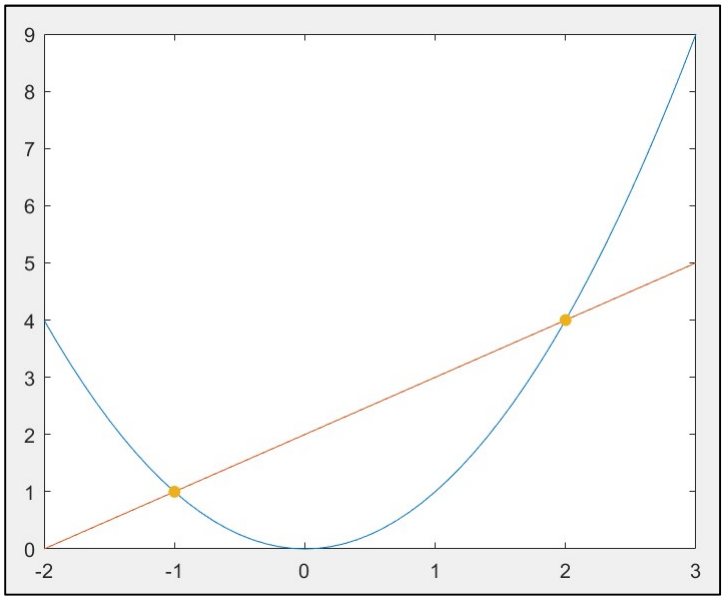
Pedro Cortez Lopes
Rafael Lopez Rangel
Luiz Fernando Martha

CIV2801 – Fundamentos da Computação Gráfica Aplicada
2025.2

Funções de plotagem no MATLAB

O MATLAB oferece funcionalidades de desenho e plotagem de relativo fácil uso. Comandos como *plot* permitem a visualização de curvas e gráficos, bem como a modelagem gráfica de objetos.

```
>> pol_1 = [1, 0, 0]; % x²
>> pol_2 = [1, 2];    % x + 2
>> intersections = getPolyIntsect(pol_1,pol_2); % x² = x + 2
>> x = -2:0.1:3;
>> plot(x,polyval(pol_1,x))
>> hold on
>> plot(x,polyval(pol_2,x))
>> hold on
>> scatter(intersections(:,1),intersections(:,2),'filled')
```



A função *plot* tem como entrada vetores *x* e *y*, onde cada par de coordenadas (x_i,y_i) representa um ponto a ser conectado por retas.

O comando *hold on* garante que o último *plot* seja mantido no *canvas* (eixos de plotagem).

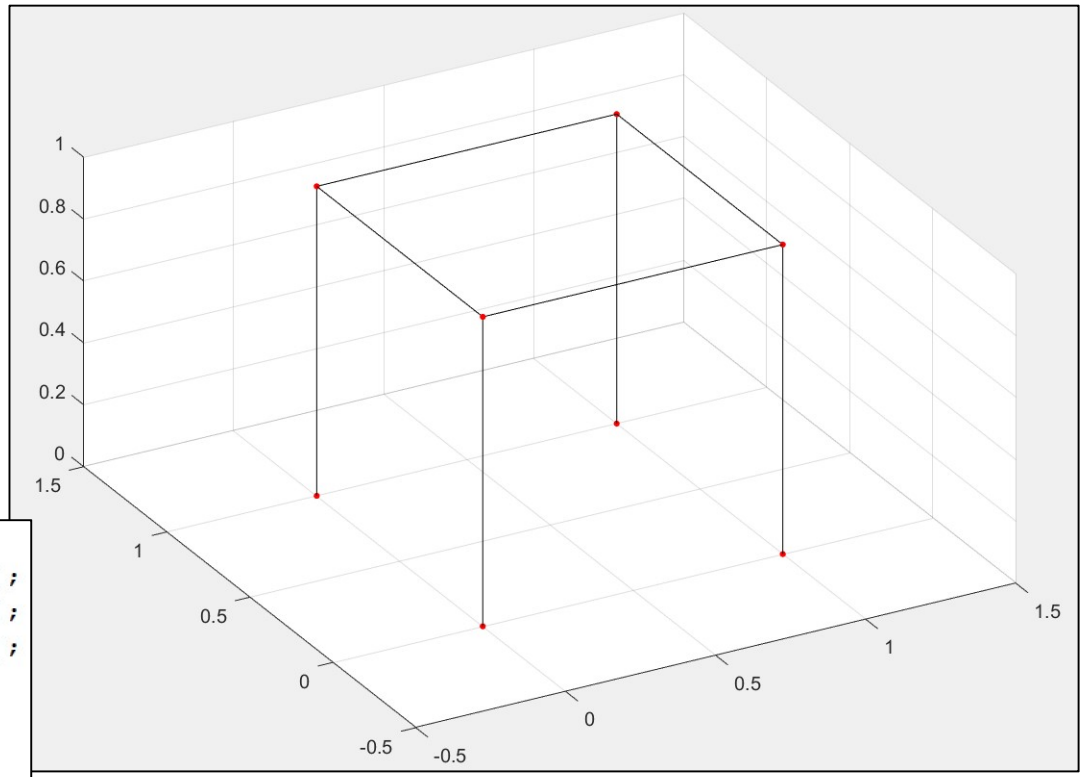
A função *scatter* tem como entrada vetores *x* e *y*, onde cada par de coordenadas (x_i,y_i) representa um ponto a ser plotado. A especificação *'filled'* faz com que esses pontos sejam círculos cheios.

Funções de plotagem no MATLAB

As funções gráficas do MATLAB podem ser usadas para visualizar modelos estruturais, como, por exemplo, um pórtico tridimensional.

```
>> nodes = [0 0 0;     % [x y z]
            0 0 1;
            1 0 1;
            1 0 0;
            0 1 0;
            0 1 1;
            1 1 1;
            1 1 0];
>> elems = [1 2;       % [initial node, final node]
            2 3;
            3 4;
            5 6;
            6 7;
            7 8;
            2 6;
            3 7];
```

```
>> for i = 1:size(elems,1)
      x = [nodes(elems(i,1),1), nodes(elems(i,2),1)];
      y = [nodes(elems(i,1),2), nodes(elems(i,2),2)];
      z = [nodes(elems(i,1),3), nodes(elems(i,2),3)];
      plot3(x,y,z,'color',[0 0 0])
      hold on
      scatter3(x,y,z,10,[1 0 0],'filled')
      hold on
      if i == size(elems,1)
         axis equal
         axis([-0.5 1.5 -0.5 1.5 0 1])
         grid on
      end
   end
```



Funções de auxiliares de desenho

```
%-------------------------------------------------------------------
% Plots a square with defined center coordinates, side length and
% color.
% This method is used to draw nodal points and rotation constraint
% on 2D models.
% Input arguments:
%  x: center coordinate on the X axis
%  y: center coordinate on the Y axis
%  S: side length
%  c: color (RGB vector)
function square(x,y,S,c)
    s = S/2;

    X = [x - s , x + s , x + s , x - s];
    Y = [y - s , y - s , y + s , y + s];
    fill(X, Y, c);
end
```

```
%-------------------------------------------------------------------
% Plots a circle with defined center coordinates, radius and color.
% This method is used to draw hinges on 2D models.
% Input arguments:
%  x: center coordinate on the X axis
%  y: center coordinate on the Y axis
%  r: circle radius
%  c: color (RGB vector)
function circle(x,y,r,c)
    circ = 0 : pi/50 : 2*pi;
    xcirc = x + r * cos(circ);
    ycirc = y + r * sin(circ);
    plot(xcirc, ycirc, 'color', c);
end
```

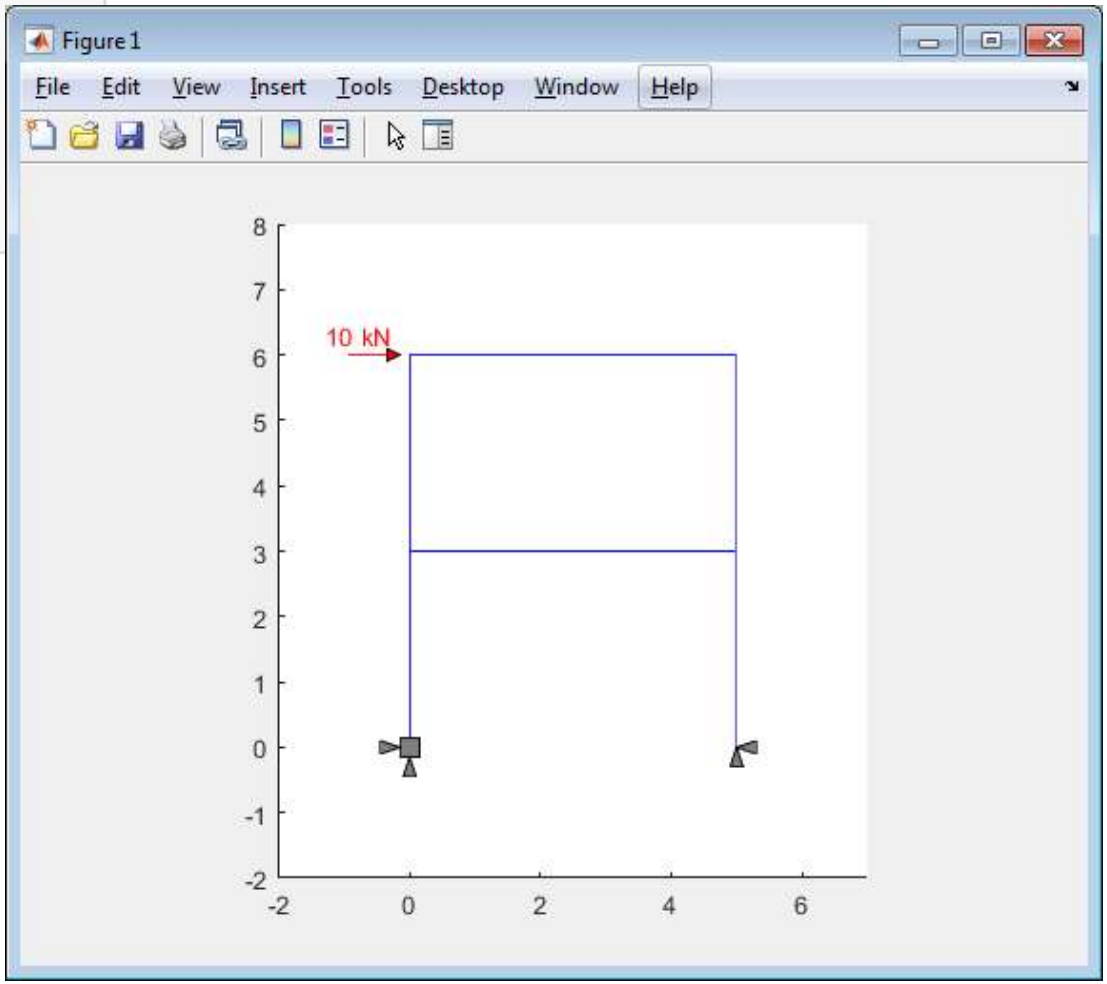
Funções de auxiliares de desenho

```
%-------------------------------------------------------------------
% Plots a triangle with defined top coordinates, height, base,
% orientation, and color.
% This method is used to draw translation constraints on 2D models.
% Input arguments:
%  x: top coordinate on the X axis
%  y: top coordinate on the Y axis
%  h: triangle height
%  B: triangle base
%  ang: angle (in radian) between the axis of symmetry and the
%       horizontal direction (counterclockwise) - 0 rad when
%       triangle is pointing left
%  c: color (RGB vector)
function triangle(x,y,h,B,ang,c)
    b = B/2;

    cx = cos(ang);
    cy = sin(ang);

    X = [x, x + h * cx + b * cy, x + h * cx - b * cy];
    Y = [y, y + h * cy - b * cx, y + h * cy + b * cx];
    fill(X, Y, c);
end
```

```
%-------------------------------------------------------------------
% Plots a 2D arrow with defined beggining coordinates, length,
% arrowhead height, arrowhead base, orientation, and color.
% This method is used to draw load symbols on 2D models.
% Input arguments:
%  x: beggining coordinate on the X axis
%  y: beggining coordinate on the Y axis
%  l: arrow length
%  h: arrowhead height
%  B: arrowhead base
%  ang: pointing direction (angle in radian with the horizontal
%       direction - counterclockwise) - 0 rad when pointing left
%  c: color (RGB vector)
function arrow2D(x,y,l,h,B,ang,c)
    b = B/2;

    cx = cos(ang);
    cy = sin(ang);

    % Draw body line
    X = [x, x + l * cx];
    Y = [y, y + l * cy];
    line(X, Y, 'Color', c);

    % Draw arrowhead
    triangle(x, y, h, B, ang, c);
end
```

Funções de auxiliares de desenho

```
clf
hold on
axis equal
line([0,0,0,5,5,5],[0,3,6,6,3,0],'color',[0,0,1])
line([0,5],[3,3],'color',[0,0,1])
square(0,0,0.3,[0.5,0.5,0.5])
triangle(0,-0.15,0.3,0.2,3*pi/2,[0.5,0.5,0.5])
triangle(-0.15,0,0.3,0.2,pi,[0.5,0.5,0.5])
triangle(5,0,0.3,0.2,3*pi/2,[0.5,0.5,0.5])
triangle(5,0,0.3,0.2,0,[0.5,0.5,0.5])
arrow2D(-0.15,6,0.8,0.2,0.2,pi,[1,0,0])
text(-1.3,6.3,'10 kN','color',[1,0,0])
axis([-2,7,-2,8])
```



Transformações Geométricas

$$\begin{aligned} T:\ \mathbf{R}^n &\rightarrow \mathbf{R}^m \\ \{P\} &\rightarrow \{P'\} = T\{P\} \end{aligned}$$

Transformação Linear:

$$T(\alpha\{P\} + \beta\{Q\}) = \alpha T\{P\} + \beta T\{Q\}$$

$$\forall\ \alpha, \beta \in \mathbf{R}$$
$$\{P\}, \{Q\} \in \mathbf{R}^n$$

$$\Rightarrow \begin{cases} T\{0\} = \{0\} \\ \{P'\} = T\{P\} = [M]\{P\} \end{cases}$$

Transformações Geométricas
Exemplo: $R^3 \rightarrow R^2$

$$T: \mathbf{R}^3 \rightarrow \mathbf{R}^2$$

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} \rightarrow \begin{Bmatrix} x' \\ y' \end{Bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \end{bmatrix} \begin{Bmatrix} x \\ y \\ z \end{Bmatrix}$$

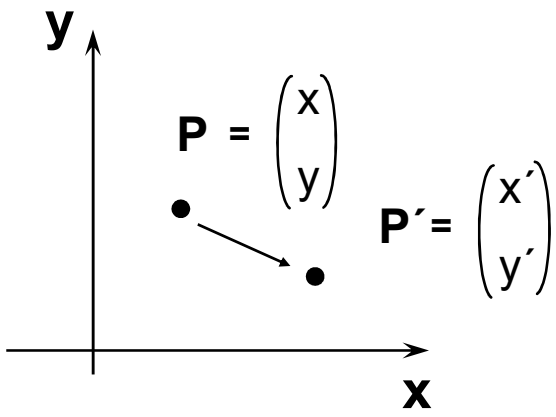
$$T \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} = [M] \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix} = \begin{Bmatrix} m_{11} \\ m_{21} \end{Bmatrix}$$

$$T \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix} = [M] \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix} = \begin{Bmatrix} m_{12} \\ m_{22} \end{Bmatrix}$$

$$T \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix} = [M] \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} m_{13} \\ m_{23} \end{Bmatrix}$$

Transformações Lineares

$R^2 \rightarrow R^2$

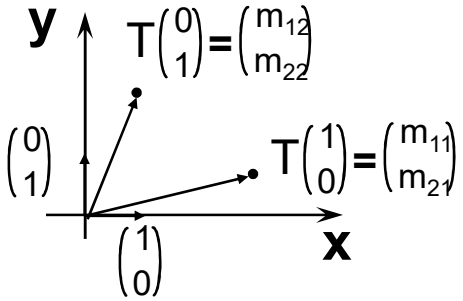


$$T(a_1 \mathbf{P}_1 + a_2 \mathbf{P}_2) = a_1 T(\mathbf{P}_1) + a_2 T(\mathbf{P}_2)$$

Mostre que:

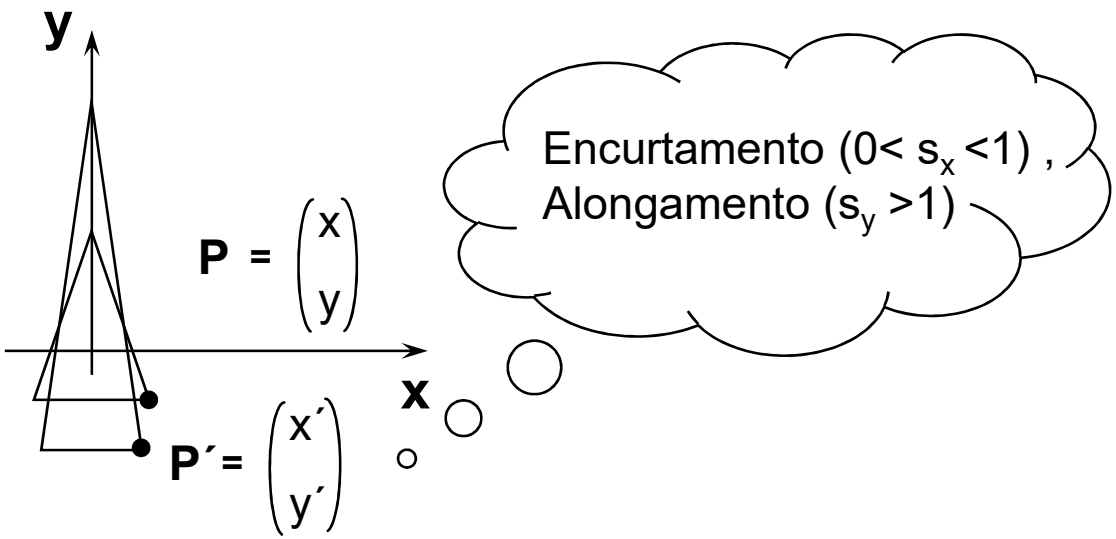
A) $T(\mathbf{0}) = \mathbf{0}$

B)



$$\Leftrightarrow \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

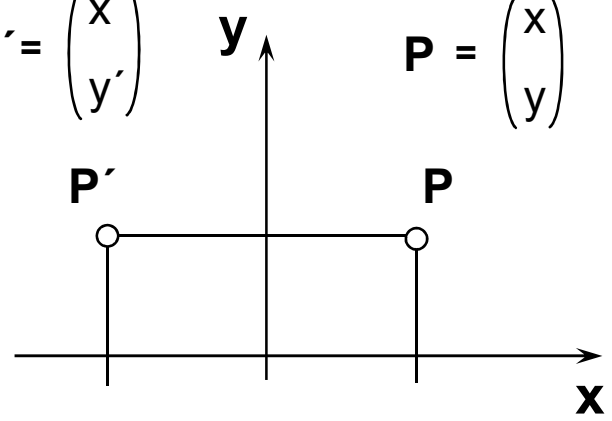
Transformação Linear
(escala em relação à origem)



$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Transformações Lineares
(espelhamento em relação ao eixo y)

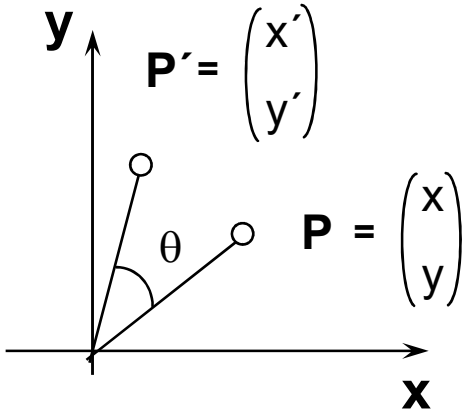
$$P' = \begin{pmatrix} x' \\ y' \end{pmatrix} \qquad P = \begin{pmatrix} x \\ y \end{pmatrix}$$



$$x' = -1.x$$
$$y' = \quad y$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

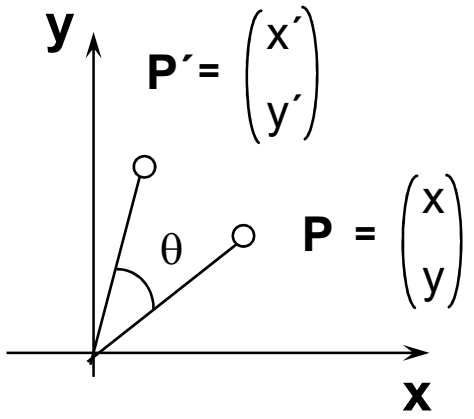
Transformações Lineares
(Rotação em relação à origem)

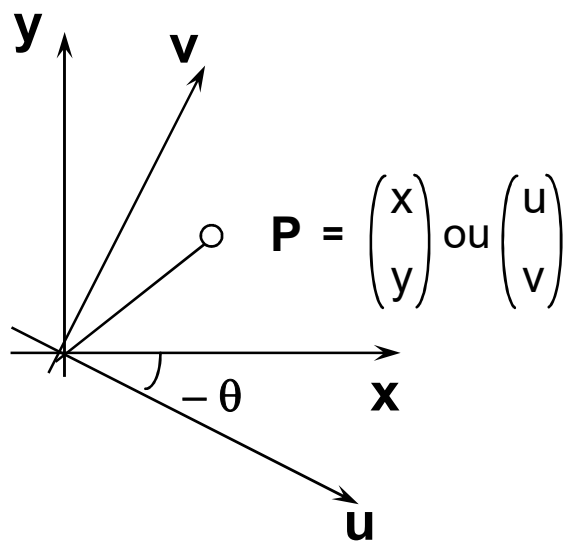


$$x' = x.\cos\theta - y.\operatorname{sen}\theta$$
$$y' = x.\operatorname{sen}\theta + y.\cos\theta$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta \\ \operatorname{sen}\theta & \cos\theta \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Transformação Linear
(rotação vs. mudança de base)





$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{bmatrix} \cos\theta & -\text{sen}\,\theta \\ \text{sen}\,\theta & \cos\theta \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

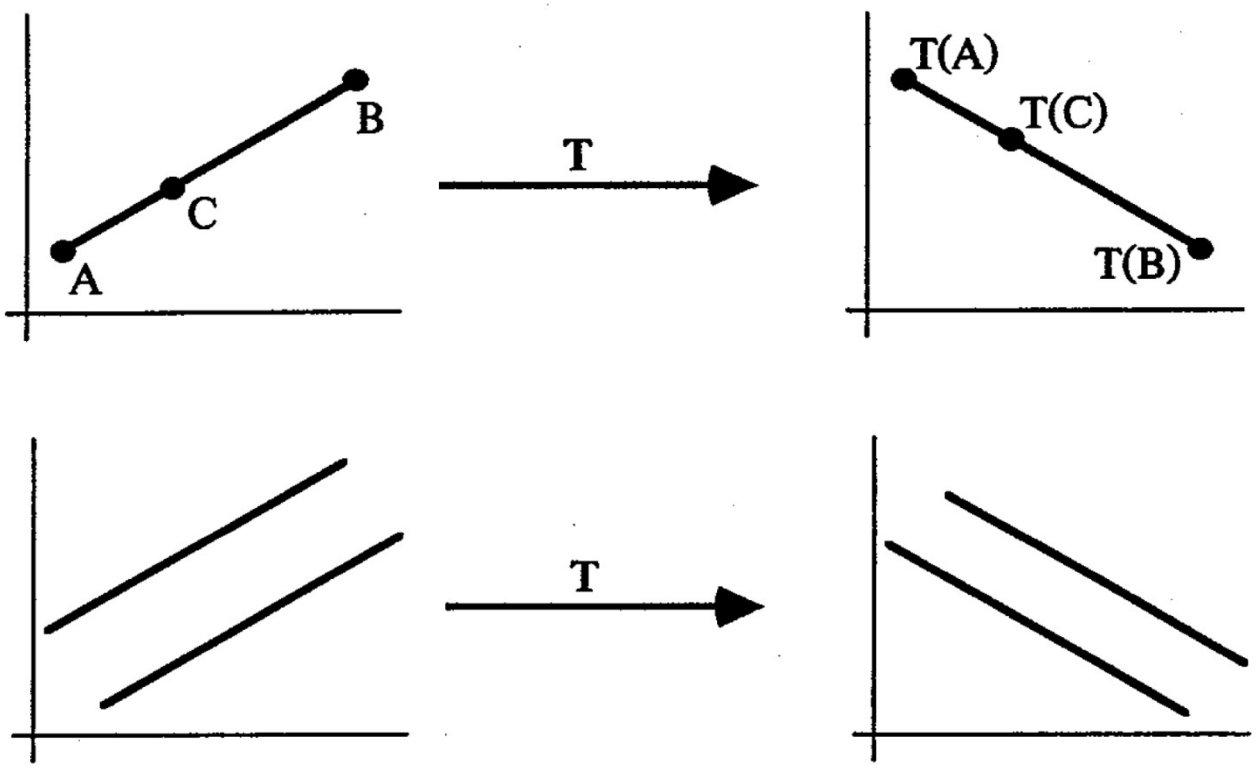
$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{bmatrix} u_x & u_y \\ v_x & v_y \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Rotação de um ponto por um ângulo θ tem o mesmo efeito da rotação dos eixos da base por um ângulo –θ. A matriz que implementa a transformação é construída por linhas, colocando os cossenos diretores dos vetores da nova base em relação à antiga em cada linha.

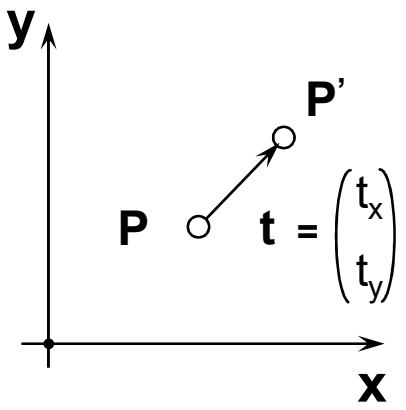
Transformações Lineares
Propriedades

Linha reta $\xrightarrow{T}$ **Linha reta**

(Propriedade muito importante em computação gráfica)



Transformação por Translação



$$\mathbf{P'} = \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \quad \Leftarrow$$ Não pode ser escrita dessa maneira ☒

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \quad \Leftarrow$$ Não é prático para implementação ☒

Transformações Afins
(transformations lineares com translação)

São transformações que preservam colinearidade, razão e paralelismo.

A matriz de uma transformação afim tem a seguinte forma:

$$[T] = \begin{bmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

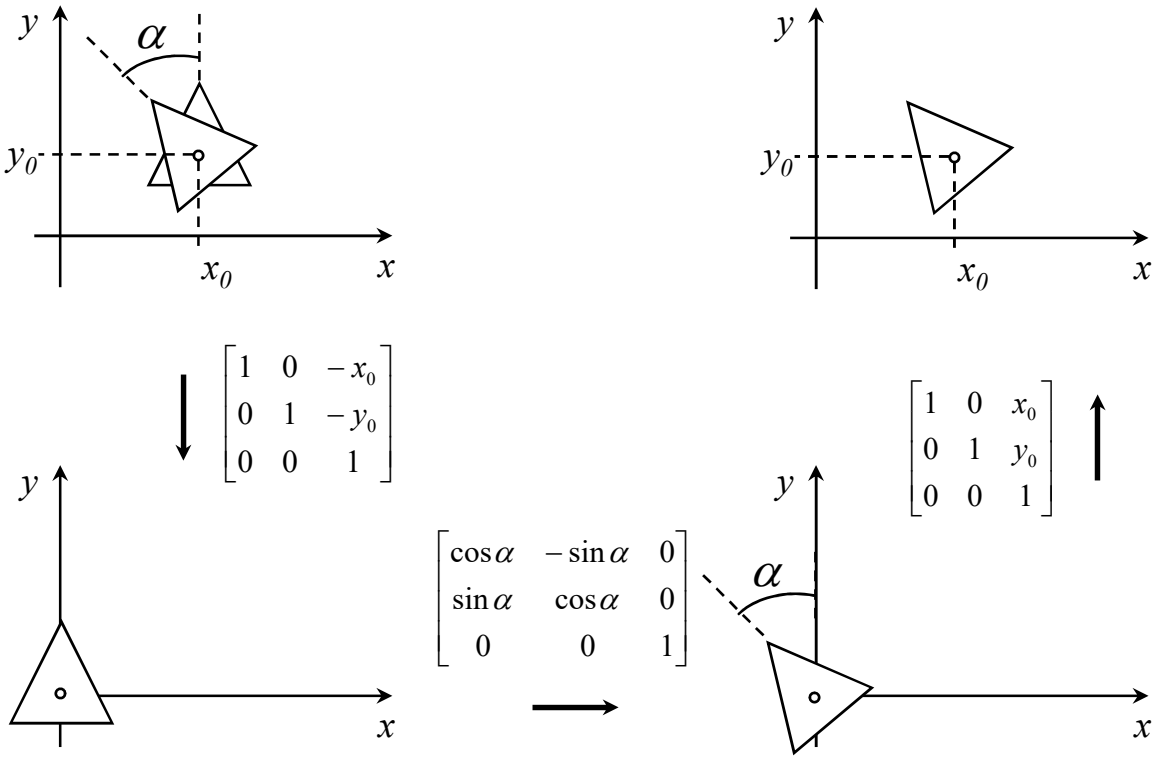
de tal forma que

$$\begin{bmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x \\ y \\ 1 \end{Bmatrix} = \begin{Bmatrix} x' \\ y' \\ 1 \end{Bmatrix}$$

Sendo assim, pode-se trabalhar apenas com uma matriz 2x3:

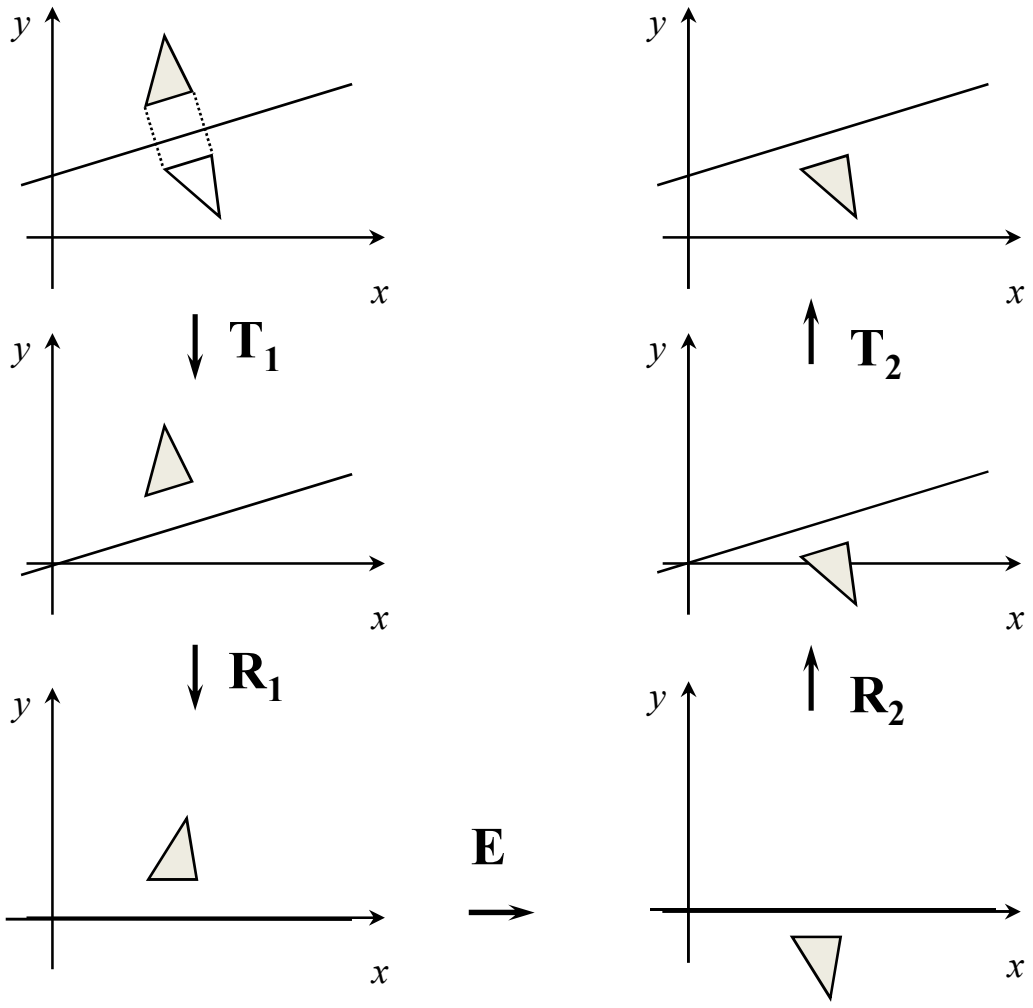
$$\begin{Bmatrix} x' \\ y' \end{Bmatrix} = \begin{bmatrix} a & b & e \\ c & d & f \end{bmatrix} \begin{Bmatrix} x \\ y \\ 1 \end{Bmatrix}$$

Vantagem de expressar uma tranformação por uma única matriz: concatenação



$$\begin{Bmatrix} x' \\ y' \\ 1 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x \\ y \\ 1 \end{Bmatrix}$$

Concatenação de Transformações



$$P' = T_2\, R_2\, E\, R_1\, T_1\, P$$

Transformação Window-Viewport

Definições

Sistema de Coordenadas do Mundo (*World Coordinate System*) - É o espaço de coordenadas do objeto (modelo). É o espaço onde o modelo de aplicacação é definido.

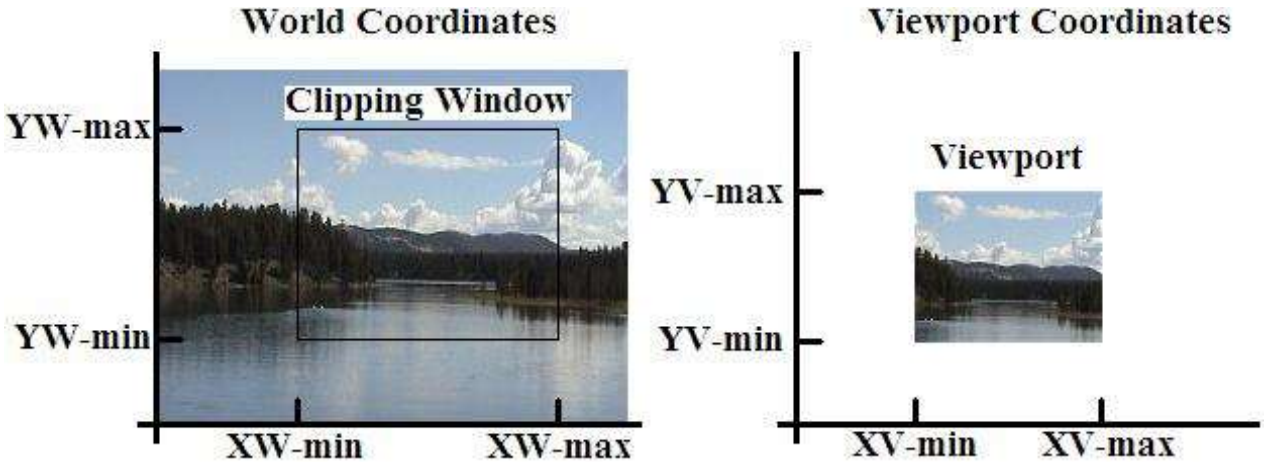
Sistema de Coordenadas da Tela (*Screen Coordinate System*) - É o espaço de cooordenadas onde a imagem é mostrada.

Janela do Mundo (*World Window* ou *Clipping Window*) - É o retângulo no sistema de coordenadas do mundo que define a região que é mostrada.

Janela de Interface - É a janela aberta na tela em que a imagem será mostrada.

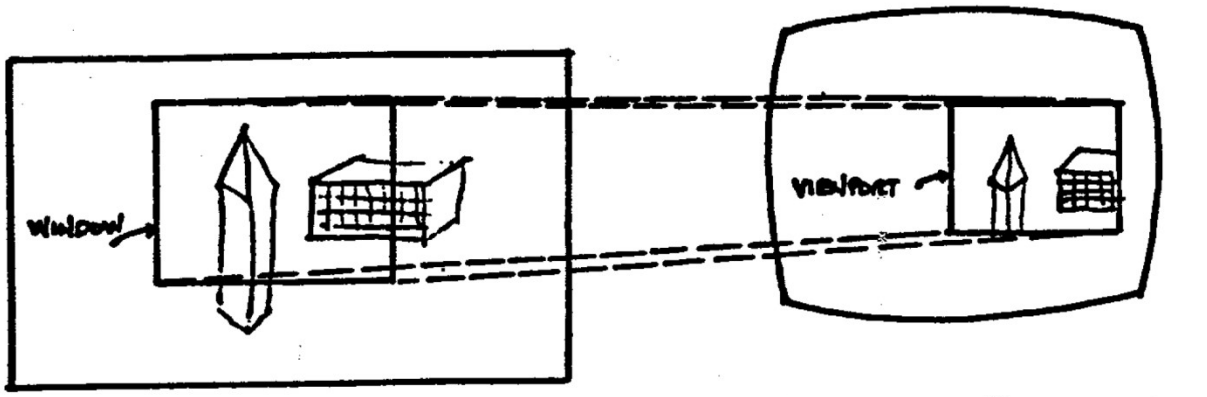
Viewport - É a porção retangular da janela de interface que define onde a imagem é mostrada (usualmente é toda a janela de interface, mas em alguns casos pode ser modificada para uma porção da janela de interface).

Transformação *Window-Viewport* - É o processo de mapeamento de uma *world window* em coordenadas do mundo para coordenadas da tela de um *Viewport*.



(Fonte: Hankins, 2014 – Lecture Notes on Computer Graphics at MTSU)

Transformação Window-Viewport



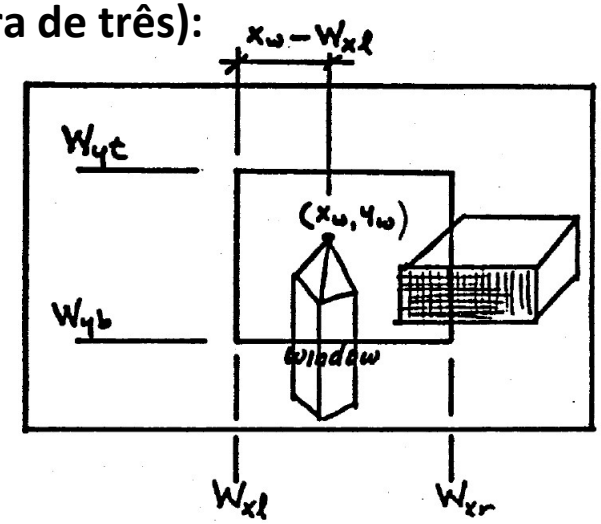
Um *viewport* é uma região na tela, usualmente retangular, dentro do qual a imagem é desenhada. O *viewport* pode ser toda a tela ou uma porção desta.

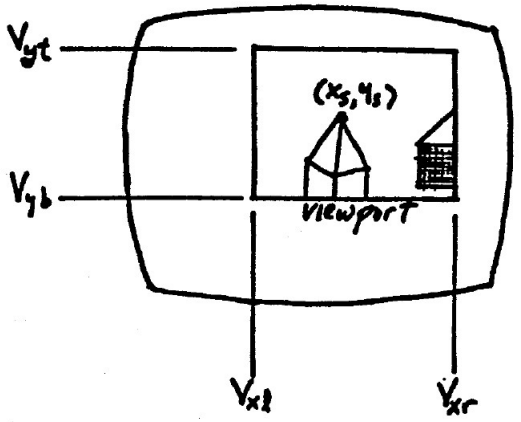
Uma *window* (janela) define a porção do espaço do objeto (mundo) a ser desenhado que vai ser vista no *viewport*. A *window* é definida nas coordenadas do espaço do objeto (*world coordinates*).

A *window* e o *viewport* são uma forma conveniente para definir a transformação da imagem do objeto na tela. Por exemplo, se nós quizermos examinar uma figura grande, nós mantemos os tamanhos da *window* e do *viewport* fixos e movemos a posição da *window*. Isto possibilita percorrer a figura com uma magnificação fixa. Para magnificar ou reduzir a imagem do objeto no *viewport*, basta reduzir ou aumentar o tamanho da *window*, respectivamente.

Transformação Window-Viewport é uma Transformação Afim

Proporção (regra de três):





$$x_s = \underbrace{\frac{(x_w - W_{xl})}{(W_{xr} - W_{xl})}}_{\text{fraction of displacement of full viewport}} \cdot \underbrace{(V_{xr} - V_{xl})}_{\text{viewport width}} + \underbrace{V_{xl}}_{\text{viewport offset}}$$

$$y_s = \frac{(y_w - W_{yb})}{(W_{yt} - W_{yb})} \cdot (V_{yt} - V_{yb}) + V_{yb}$$

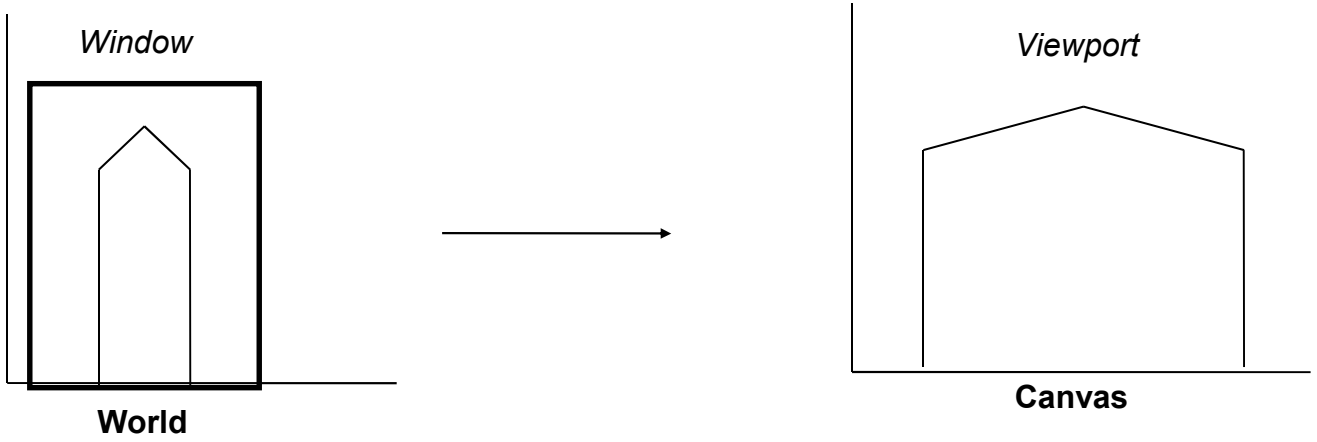
$$x_s = \frac{(V_{xr} - V_{xl})}{(W_{xr} - W_{xl})} (x_w - W_{xl}) + V_{xl}$$

$$y_s = \frac{(V_{yt} - V_{yb})}{(W_{yt} - W_{yb})} (y_w - W_{yb}) + V_{yb}$$

$$\begin{Bmatrix} x_s \\ y_s \end{Bmatrix} = \begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix} \begin{Bmatrix} x_w \\ y_w \end{Bmatrix} + \begin{Bmatrix} e \\ f \end{Bmatrix} = \begin{bmatrix} a & 0 & e \\ 0 & d & f \end{bmatrix} \begin{Bmatrix} x_w \\ y_w \\ 1 \end{Bmatrix}$$

Ajuste dos limites da Window para manter razão de aspecto do Viewport

Quando os retângulos que definem *window* e *viewport* não são proporcionais, ocorre o que se chama de distorção, fenômeno este que pode ser visualizado na figura abaixo, onde a “torre de uma igreja” passou a ser vista como um “galpão”.



Para que tal efeito não ocorra, é necessário que os lados da *window* e *viewport* sejam proporcionais.

Ajuste dos limites da Window para manter razão de aspecto do Viewport (cont.)

Considerando que o *viewport ocupa integralmente o canvas,* para que se consiga definir um par *window-viewport* de modo a não ocorrer a distorção e utilizar toda a superfície de visualização, deve-se proceder da seguinte forma:

1. Obter as dimensões do canvas (*viewport*) em pixels:

   `w (largura)` **`h (altura)`**

2. Calcular a razão de aspecto do retângulo do canvas:

   `h_w = h / w` **`(altura / largura)`**

3. Inicializar limites da *window* com os mínimos e máximos da caixa envolvente do modelo: **`xmin, xmax, ymin, ymax`**

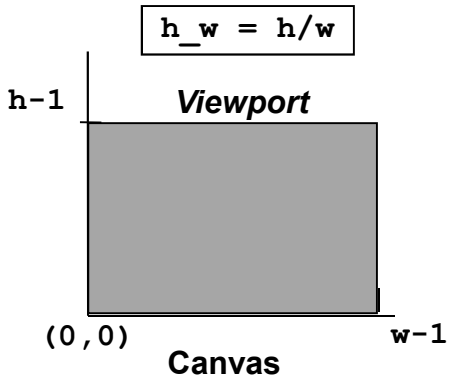
4. Calcular o centro da *window* e tamanhos mínimos com folga de 10%:

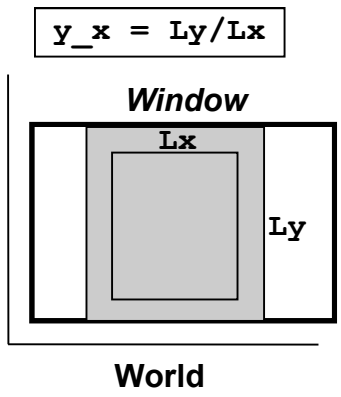
   `Lx = (xmax – xmin) * 1.10` **`Ly = (ymax – ymin) * 1.10`**

   `Mx = (xmin + xmax) / 2` **`My = (ymin + ymax) / 2`**

Ajuste dos limites da Window para manter razão de aspecto do Viewport (cont.)

5. Ajustar tamanho da *window* para que não haja distorção:





y_x > h_w	y_x < h_w
Lx = Ly / h_w	Ly = Lx * h_w

6. Calcular os limites ajustados da *window*:

```
xmin = Mx - Lx / 2          ymin = My - Ly / 2
xmax = Mx + Lx / 2          ymax = My + Ly / 2
```