

# MSX Article

**MARMSX**

*Rotação e  
Interpolação*

## Resumo

O objetivo desse artigo é mostrar como é feita a rotação em qualquer ângulo de imagens 2D, além de apresentar dois métodos de interpolação – vizinho mais próximo e bilinear.

## 1- Introdução

A rotação de uma imagem nada mais é do que aplicar uma função de transformação e obter novas coordenadas para cada pixel dessa imagem. Em outras palavras, após uma função de transformação de rotação aplicada a um pixel  $P(X,Y)$  obtém-se  $P(X',Y')$ . Uma vez que a identidade de um pixel é sua cor, copia-se a cor de  $P(X,Y)$  para  $P(X',Y')$ .


A rotação de objetos em ângulos múltiplos de 90 graus é fácil de ser realizada, uma vez que é necessário apenas aplicar uma transformação simples, como uma operação de matriz transposta. O exemplo a seguir, rotaciona uma imagem em 90 graus em sentido anti-horário.

```
10 SCREEN 5
20 COPY"BRASIL.IC5" TO (0,0),0
30 DX=0 : DY=80
40 FOR Y=0 TO 68 }
50 FOR X=0 TO 99 }
60 C=POINT(X,Y)
70 PSET(Y+DX,99-X+DY),C
80 NEXT X,Y
90 GOTO 90
```

Varre toda a imagem.

Obtém a cor do pixel a ser rotacionado.

Aplica rotação e copia a cor do pixel.



O programa acima realiza a operação de matriz transposta à imagem, também aplicando um espelhamento no eixo Y para que a imagem resultante não fique invertida. Além disso, foi aplicado na imagem resultante um deslocamento  $DX$  e  $DY$  para que ela não ficasse sobreposta à imagem original.

Entretanto, a rotação da imagem em qualquer ângulo necessita de um sistema de transformação mais sofisticado.

## 2- Rotação da imagem em qualquer ângulo

A partir da equação (1), é possível rotacionar uma imagem em qualquer ângulo, em torno da origem do sistema (coordenada 0,0).

$$\begin{aligned}x' &= x \cdot \cos(\theta) - y \cdot \sin(\theta) \\y' &= x \cdot \sin(\theta) + y \cdot \cos(\theta)\end{aligned}\tag{1}$$

A figura 1 apresenta os dois sistemas utilizados na rotação da imagem: o sistema de coordenadas da tela e o sistema de coordenadas do mundo real. O primeiro é o sistema utilizado pela imagem, enquanto que o segundo é o sistema utilizado para rotacionar a a imagem.

É desejável que a origem do sistema esteja posicionada no meio da figura e não em suas extremidades, conforme mostra a figura 1. Deve-se observar também que o sistema de coordenadas na tela possui o eixo  $Y$  no sentido inverso do sistema de coordenadas do mundo real.

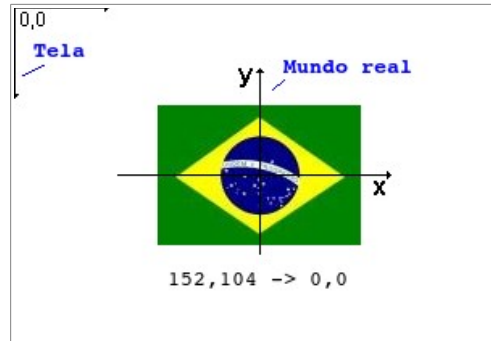


Figura 1 – A imagem e os sistemas de coordenadas do mundo real e da tela.

O primeiro passo é pegar uma coordenada de um pixel da imagem no sistema de coordenadas da tela e convertê-lo para o sistema de coordenadas do mundo real, para que possa ser aplicada a rotação. A equação (2) faz essa transformação, além de posicionar o sistema de coordenadas no centro da imagem.

$$\begin{aligned}x_r &= x_i - cx_i \\ y_r &= cy_i - y_i\end{aligned}\tag{2}$$

Onde:

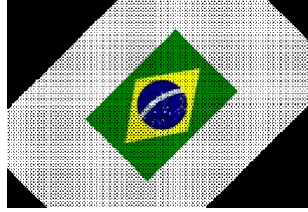
- $x_r$  e  $y_r$  – são as coordenadas no sistema do mundo real (rotação).
- $x_i$  e  $y_i$  – são as coordenadas no sistema da tela.
- $cx_i$  e  $cy_i$  – são as coordenadas do centro da imagem no sistema da tela.

Após aplicar a rotação, os valores de  $x'$  e  $y'$  calculados devem ser convertidos de volta para o sistema da tela. A equação (3) realiza essa operação.

$$\begin{aligned}x_i &= x'_r + cx_i \\ y_i &= cy_i - y'_r\end{aligned}\tag{3}$$

Quando há transformação entre sistemas de coordenadas, deve-se partir da imagem de destino e procurar o pixel correspondente na imagem de origem, justamente o contrário do que muitos imaginam. Isto se deve ao fato de que, ao percorrermos os pixels na imagem original, nem sempre garantimos que todos os pontos da imagem destino serão preenchidos. O resultado dessa operação pode ser observado na figura 2a.

Quando percorrermos a imagem destino em busca dos pixels correspondentes na imagem original, garantimos que todos os pixels dessa imagem tenham alguma correspondência. O resultado dessa operação pode ser observado na figura 2b.



a) a partir da imagem de origem



b) a partir da imagem de destino

Figura 2. Rotação aplicada em uma imagem.

As mudanças que devemos realizar para partir da imagem destino são:

1. Assumir o ângulo theta ( $\theta$ ) da equação (1) como negativo.
2. Pegar a cor do pixel calculado (imagem original).
3. Preencher o pixel lido (imagem destino) com essa cor.

De forma a otimizar os cálculos, pode-se calcular o seno e cosseno de um ângulo fixo antes de varrer a imagem. O MSX leva quase um segundo para realizar cada uma dessas operações.

Quando a coordenada de um pixel na imagem original é calculado através do sistema de rotação (equação 1), o resultado é composto de números reais. Isto significa dizer que a coordenada encontrada não cai exatamente sobre o pixel dessa imagem, conforme mostra a figura 3.

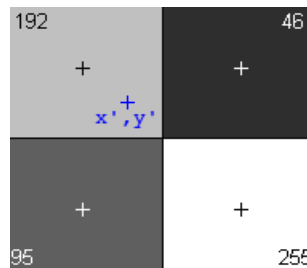


Figura 3. O problema da escolha da cor, a partir de uma coordenada com valores reais.

Uma vez que é um pixel (cor) que buscamos na imagem de origem, quando a coordenada calculada se localiza não mais exatamente sobre um pixel, mas sim entre pixels vizinhos, o que fazer?

A seção seguinte irá abordar o tema de interpolação, que busca interpretar esse resultado de diversas maneiras.

### 3- Métodos de interpolação

#### 3.1- Vizinho mais próximo (Nearest Neighbor)

A interpolação por vizinho mais próximo assume a cor do *pixel* mais próximo daquela coordenada. Dessa forma, é feito o arredondamento das coordenadas de valores reais. Por exemplo, a coordenada (98,5; 34,2) ficaria:

round(98.5) = 99  
round(34.2) = 34

Assim, a cor assumida na imagem destino seria a do pixel, cuja a coordenada fosse (99,34).

Esta interpolação é a mais simples de se implementar, entretanto, apresenta efeitos colaterais como o “dente de serra”.

### 3.2- Interpolação Bilinear

A interpolação bilinear faz uma média ponderada dos quatro *pixels* vizinhos da região em torno da coordenada encontrada. O valor da intensidade do *pixel* é calculado, segundo a equação (4):

$$f(x,y) = f(0,0) \cdot (1-x) \cdot (1-y) + f(1,0) \cdot x \cdot (1-y) + f(0,1) \cdot (1-x) \cdot y + f(1,1) \cdot x \cdot y \quad (4)$$

Onde  $f(x,y)$  é a cor do pixel, segundo a seguinte configuração:

f(0,0)	f(1,0)
f(0,1)	f(1,1)

Para cada pixel vizinho em relação a coordenada  $(x',y')$ , é considerado o inverso da distância, uma vez que quanto menor a distância do pixel ao ponto calculado, maior a contribuição daquele pixel na cor final.

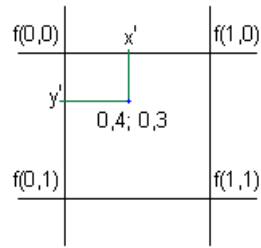
Suponha uma transformação de rotação, cujas coordenadas calculadas são (150,4; 200,3). Os quatro pixels vizinhos a esta coordenada são os seguintes:

- $P_1(150,200)$
- $P_2(151,200)$
- $P_3(150,201)$
- $P_4(151,201)$

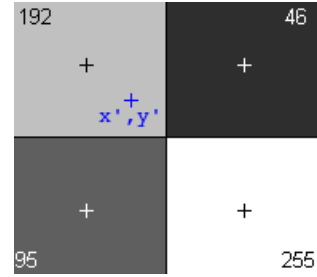
Tomando-se  $P_1$  como origem em (0,0), temos:

- $P_1(0, 0)$
- $P_2(1, 0)$
- $P_3(0, 1)$
- $P_4(1, 1)$
- $P(0,4; 0,3)$

A figura 4a ilustra o sistema de coordenadas normalizado para o cálculo da interpolação, enquanto que a figura 4b apresenta os níveis de cinza dos pixels.



a) sistema de coordenadas normalizado



b) intensidade dos pixels

Figura 4. Interpolação Bilinear.

Utilizando-se a equação (4), tem-se:

$$f(x, y) = 192 \times (1 - 0,4) \times (1 - 0,3) + 46 \times 0,4 \times (1 - 0,3) + 95 \times (1 - 0,4) \times 0,3 + 255 \times 0,4 \times 0,3$$

$$f(x, y) = 141,2200$$

$$f(x, y) = \text{round}(141,2200)$$

$$f(x, y) = 141$$

Como o pixel tem intensidade com o valor inteiro, arredonda-se o valor encontrado.

### 3.3- Comparação dos resultados

A figura 5 apresenta uma comparação entre os resultados da interpolação por vizinho mais próximo e bilinear.



a) interpolação por vizinho mais próximo



b) interpolação bilinear

Figura 5. Comparação entre as interpolações.

Observa-se que o resultado da interpolação bilinear é melhor do que o vizinho mais próximo. Entretanto, o custo computacional na interpolação bilinear é bem maior, problema este agravado em computadores como o MSX.

Outro problema para o uso da interpolação bilinear no MSX, diz respeito à média ponderada de cores. Somente a screen 8 poderia fazê-la com sucesso, visto que não possui sistema de cores indexados. Entretanto, o custo computacional seria aumentado ainda mais.

## 4- Um programa em Basic para rotacionar imagens

O programa a seguir rotaciona a bandeira do Brasil em 45 graus em sentido anti-horário, utilizando a interpolação por vizinho mais próximo. Alguns procedimentos foram adotados:

- A imagem foi carregada na página 1 da screen 5, centralizada em 128,105.
- O programa rotaciona a imagem da página 1 para a página 0, também centrado em 128,105.
- A área de desenho leva em conta a diagonal da bandeira, o maior comprimento possível do desenho. Assim, essa área é um quadrado com a medida de lado da diagonal da bandeira.
- Foi utilizado o truncamento em vez do arredondamento de  $(x',y')$ . Caso fosse utilizado o arredondamento, haveria mais uma operação, uma vez que o Basic do MSX não possui essa função.

```
10 SCREEN 5
20 SETPAGE 0,1:CLS
30 COPY"BRASIL.IC5" TO (77,71),1
40 SET PAGE 0,0
50 CX=128 : CY=105
60 PI=3.14159 : AN=-PI/4
70 ST=SIN(AN) : CT=COS(AN)
80 FOR Y=34 TO 175
90 FOR X=57 TO 198
100 XR = X-CX
110 YR = CY-Y
120 RX = FIX(XR*CT - YR*ST)
130 RY = FIX(XR*ST + YR*CT)
140 XI = RX+CX
150 YI = CY-RY
160 SETPAGE 0,1
170 C=POINT(XI,YI)
180 IF C<0 THEN C=0
190 SETPAGE 0,0
200 PSET(X,Y),C
210 NEXT X,Y
220 GOTO 220
```

Seno e cosseno calculados uma vez.

Varre toda a imagem.

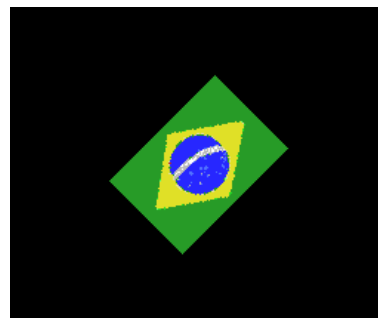
Converte coordenadas da imagem destino para o mundo real.

Rotaciona o pixel.

Converte as coordenadas encontradas para o sistema de coordenadas da imagem.

Obtém a cor do pixel da imagem original.

Copia a cor para a imagem destino.



Esse programa leva em torno de 30 minutos para completar a rotação em um MSX. Acrescente a seguinte linha para acompanhar o processamento:

```
75 LINE(57,34)-(192,175),13,BF
```

Obs: os programas utilizados nesse artigo, bem como as figuras se encontram disponíveis para download na página de artigos da página MarMSX Development – <http://marmsx.msxall.com>

## 5- Fontes para testes em Matlab / GNU-Octave

Vizinho mais próximo	Bilinear
<pre> % Rotation % Marcelo Teixeira Silveira  br=imread('flag.bmp'); [a b c]=size(br); br2 = zeros(a,b,c); br2=uint8(br2);  theta = -0.78539816339744830961566084581988; % 45 degrees centerX=153; centerY=105;  for x=1 : b     for y=1 : a         % From screen system to center system         % Center coordinate system         cx = x-centerX;         cy = centerY-y;         % Rotate         nx = cx*cos(theta)-cy*sin(theta);         ny = cx*sin(theta)+cy*cos(theta);         % Nearest neighbor         nx=ceil(nx);         ny=ceil(ny);         % Return old system         cx=nx+centerX;         cy=centerY-ny;         if (cx&gt;0)             if (cy&gt;0)                 if (cx&lt;=b)                     if (cy&lt;=a)                         br2(y,x,:) = br(cy,cx,:);                     end                 end             end         end     end end;  figure(1); subplot(1,2,1); imshow(br); subplot(1,2,2); imshow(br2); </pre>	<pre> % Rotation % Marcelo Teixeira Silveira  br=imread('flag.bmp'); [a b c]=size(br); br2 = zeros(a,b,c); br2=uint8(br2);  theta = -0.78539816339744830961566084581988; % 45 degrees centerX=153; centerY=105;  for x=1 : b     for y=1 : a         % From screen system to center system         % Center coordinate system         cx = x-centerX;         cy = centerY-y;         % Rotate         nx = cx*cos(theta)-cy*sin(theta);         ny = cx*sin(theta)+cy*cos(theta);         % Return old system         cx=nx+centerX;         cy=centerY-ny;         % Pick up neighbors coordinates origin and float value of         % coordinate         Pox = floor(cx);         Poy = floor(cy);         xn = cx-floor(cx);         yn = cy-floor(cy);         if (cx&gt;1)             if (cy&gt;1)                 if (cx&lt;b)                     if (cy&lt;a)                         % Bilinear                         br2(y,x,1) = ceil(br(Poy,Pox,1))*(1-xn)*(1-yn) +br(Poy,Pox+1,1)*xn*(1-yn)+br(Poy+1,Pox,1)*(1- xn)*yn+br(Poy+1,Pox+1,1)*xn*yn);                         br2(y,x,2) = ceil(br(Poy,Pox,2))*(1-xn)*(1-yn) +br(Poy,Pox+1,2)*xn*(1-yn)+br(Poy+1,Pox,2)*(1- xn)*yn+br(Poy+1,Pox+1,2)*xn*yn);                         br2(y,x,3) = ceil(br(Poy,Pox,3))*(1-xn)*(1-yn) +br(Poy,Pox+1,3)*xn*(1-yn)+br(Poy+1,Pox,3)*(1- xn)*yn+br(Poy+1,Pox+1,3)*xn*yn);                     end                 end             end         end     end end;  figure(1); subplot(1,2,1); imshow(br); subplot(1,2,2); imshow(br2); </pre>



## 6- Créditos e referências

Este artigo foi escrito por Marcelo Silveira, formado em Engenharia de Sistemas e Computação pela UERJ.

Escrito em: Abril de 2007.

Revisado em: Julho de 2017 e Maio de 2018.

Site: <http://marmsx.msxall.com>

E-mail: [flamar98@hotmail.com](mailto:flamar98@hotmail.com)

### Referências:

Wikipédia em inglês: <http://en.wikipedia.org>

- Artigo sobre interpolação bilinear

- Artigo sobre rotação