MSX Article

MARMSX

Maravilhas com a screen 2

Resumo

Nesse artigo apresentamos algumas maravilhas que podem ser feitas na limitada screen 2 do MSX, no qual somente temos 15 cores pré-definidas e agrupamentos de pontos 1x8 onde somente é possível termos 2 cores em cada grupo.

1- Um truque para aumentar o número de cores da screen 2

Diferente das screen 2-7 do MSX 2, a screen 2 do MSX 1 possui 15 cores pré-definidas e fixas. Graças à sua arquitetura, podemos aplicar um pequeno truque desenvolvido por Daniel Vik [1], autor do emulador BlueMSX e de demos como Utopia e MSX Unleashed, para aumentar o número de cores e chegar a 120 cores. Como isso é possível?

Se conseguirmos alternar dois pontos de cores muito rapidamente, teremos a impressão de ver a cor média desses pontos. Por exemplo, ao combinarmos a cor verde (código 3 do MSX) com a azul ciano (código 7 do MSX), teremos um novo verde com o valor médio das componentes RGB das cores 3 e 7. Veja a figura 2.1.

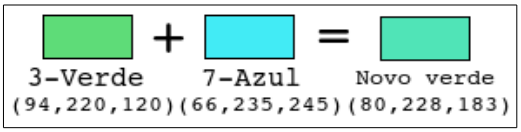


Figura 2.1. - Combinação de cores.

1.1. Quantas cores poderemos ter combinando as 15 cores nativas do MSX 1?

Utilizando o conceito de combinação da matemática, temos:

$$C(n,p)=\frac{n!}{(n-p)!\,p!}$$

$$C(15,2)=\frac{15!}{(15-2)!\,2!}=\frac{15\times14\times\cancel{13!}}{\cancel{13!}\,2!}=\frac{15\times14}{2}=15\times7=105$$

Entretanto, estamos falando de combinação de cores diferentes. Quando combinamos duas cores iguais, iremos obter uma cor original do MSX 1. Assim, temos o total de 105 + 15 = 120 cores.

A figura 2.2 apresenta todas as 120 cores possíveis através da combinação das cores nativas do MSX 1. Na diagonal mais externa temos as 15 cores originais do MSX 1.

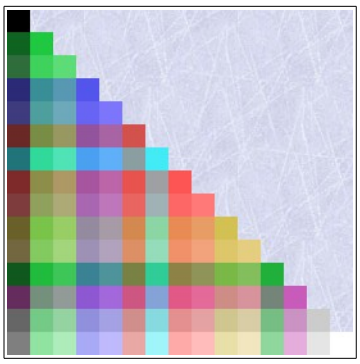


Figura 2.2. - As 120 cores da screen 2.

1.2. Como converter uma imagem para o formato da screen 2 com 120 cores

Como visto anteriormente, a geração de novas cores requer a combinação de duas cores do MSX. Dessa forma, deveremos ter duas imagens que irão combinar suas cores e gerar as cores novas. Com isso, ganharemos além das novas cores outra vantagem: 4 pontos por bloco. Como? Veja a seguir.

Para cada grupo de 1x8 pontos, podemos ter somente duas cores. Entretanto, como se trata de duas imagens, a combinação de cor de frente e fundo de cada imagem resulta no total de 4 cores diferentes por grupo. Observe as combinações possíveis [1]:

- A *cor de frente imagem 1* com a *cor de frente imagem 2*
- A *cor de frente imagem 1 com* a *cor de fundo imagem 2*
- A *cor de fundo imagem 1 com* a *cor de frente imagem 2*
- A *cor de fundo imagem 1 com* a *cor de fundo imagem 2*

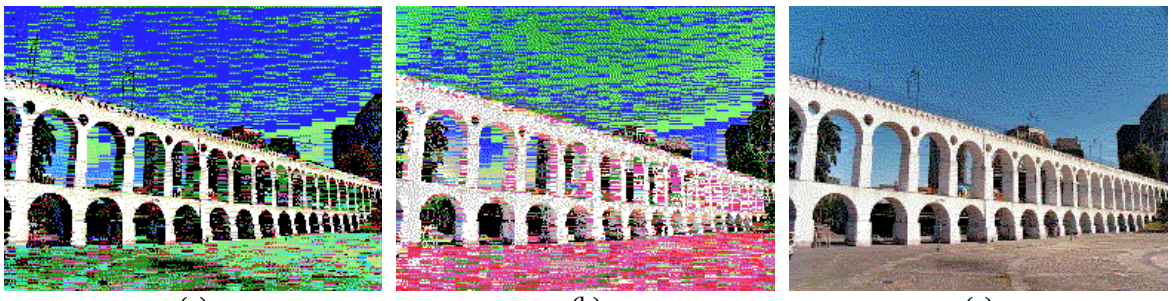
Figura 2.3. Geração das cores.

Ao utilizarmos duas cores nativas do MSX, poderemos esbarrar em um problema: a combinação delas somente permite gerar três cores diferentes em vez de quatro. Por exemplo, duas imagens com as cores 1 e 2 são capazes de gerar as combinações de cores 11, 22 e 12, pois a combinação 21 tem a mesma cor de 12.

A função de custo [1], vista no próximo capítulo, faz a quantização da imagem já descobrindo qual a melhor combinação das 4 cores para cada grupo de 1x8 pontos. Esse método obtém resultados bem melhores do que a quantização por pontos isolados seguido da escolha dos quatro pontos para cada grupo.

Outro ponto importante da função de custo é que ela já determina quem é o ponto de frente e fundo de cada imagem, simplificando essa tarefa. Observe na imagem da direita da figura 2.3 os pontos de resultado e imagine como determinar a combinação de frente/fundo de cada imagem a partir deles.

A figura 2.4 apresenta um exemplo da fusão de duas imagens em uma, devido à alternância rápida das imagens na tela. Observe que as imagens (a) e (b) possuem somente as cores nativas do MSX 1.



(a) (b) (c)

Figura 2.4. Fusão de duas imagens em uma: (a) + (b) = (c).

1.3. Como alternar as imagens

A screen 2 possui duas tabelas que controlam grupos de 1x8 pontos: a tabela de padrões, que informa o padrão de acendimento dos pontos (cor de frente ou fundo), e a tabela de cores, que contém o código das 2 cores de cada grupo. Essas duas tabelas são suficientes para criar uma tela na screen 2.

A cópia dessas duas tabelas da RAM para a VRAM é um pouco demorada para conseguirmos um efeito ilusório de uma nova cor, caso a imagem seja muito grande. Dessa forma, iremos utilizar outra tabela, muito utilizada em jogos para MSX, que é bem mais rápida para alternar os pontos na tela: a tabela de nomes.

A tabela de nomes relaciona a localização física na tela de um bloco de 8x8 pontos com um outro bloco de 8x8 pontos do mapa de padrões e cores. Essa tabela divide a tela verticalmente em 3 partes, numeradas de 0 a 255 cada. A transferência da tela toda consome 12288 bytes, enquanto que na tabela de nomes apenas 768 bytes, ou seja, 16 vezes menos dados.

Vejamos o exemplo a seguir. O programa irá trocar rapidamente dois blocos de 8x8 pontos (blocos 0 e 2 da tabela de padrões/cores) através da tabela de nomes, na posição física correspondente ao bloco 4 da VRAM, conforme mostra a figura 2.5.

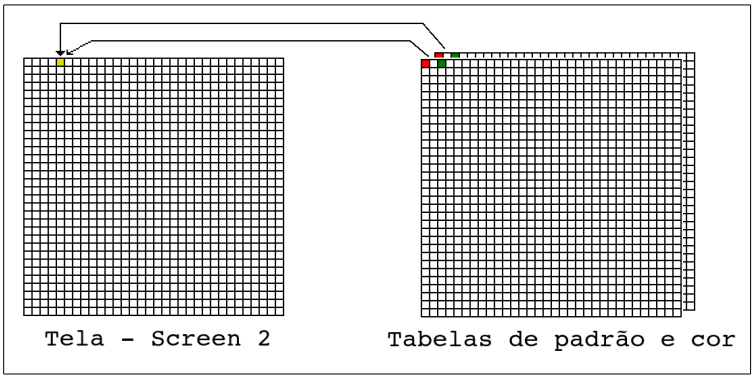


Figura 2.5. - Exemplo de fusão de cores na screen 2.

Código em Assembly:

```
C000                10          ORG &HC000
C000 21 04 18       20 INICIO:  LD HL,&H1804 ; Bloco 004 da VRAM
C003 AF             30          XOR A        ; Pat+color a escrever: 0
C004 CD 4D 00       40          CALL &H4D    ; Envia dado VRAM
C007 3E 02          50          LD A,2       ; Pat+color a escrever: 2
C009 CD 4D 00       60          CALL &H4D    ; Envia dado VRAM
C00C CD 9C 00       70          CALL &H9C    ; Igual a A$=inkey$
C00F 28 EF          80          JR Z,INICIO  ; Se tecla não pressionada, retorna
C011 C9             90          RET          ; Retorna ao Basic
```

Código em Basic, que incorpora o código em Assembly acima:

```
10 COLOR 15,1:SCREEN 2
20 OPEN"grp:" AS #1
30 LINE(0,0)-(7,7),6,BF
40 LINE(16,0)-(23,7),2,BF
50 PRESET(9,0):PRINT#1,"+"
60 PRESET(25,0):PRINT#1,"="
70 DEFUSR=&HC000
80 E=&HC000
90 READ A$:IF A$="M" THEN 130
100 POKE E,VAL("&H"+A$)
110 E=E+1
120 GOTO 90
130 X=USR(0):END
200 DATA 21,04,18,AF,CD,4D,00,3E
210 DATA 02,CD,4D,00,CD,9C,00,28
220 DATA EF,C9,M
```

Para utilizar o mapa de nomes, as telas já deverão estar carregada na VRAM. As telas? Sim, como já dissemos anteriormente, descartamos a ideia de ficar movendo a tela da RAM para VRAM.

Uma vez que iremos carregar as duas telas no mesmo espaço de VRAM, então podemos somente utilizar metade da resolução horizontal da tela. Uma tela irá ficar com metade inferior dos blocos da tabela de nomes, enquanto a outra com a metade superior. Entretanto, devemos reservar um bloco de 8x8 pontos para preencher de preto a área não usada pela imagem.

Para reservar um bloco e manter o equilíbrio das imagens, removemos uma coluna de cada imagem, resultando em 120 x 192 pontos. Assim, cada imagem ocupa exatamente 120 blocos de 8x8 pontos da tabela de padrões e caracteres, conforme mostra a figura 2.6.

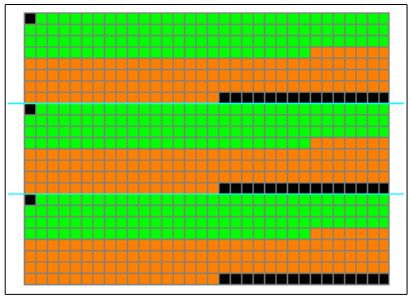
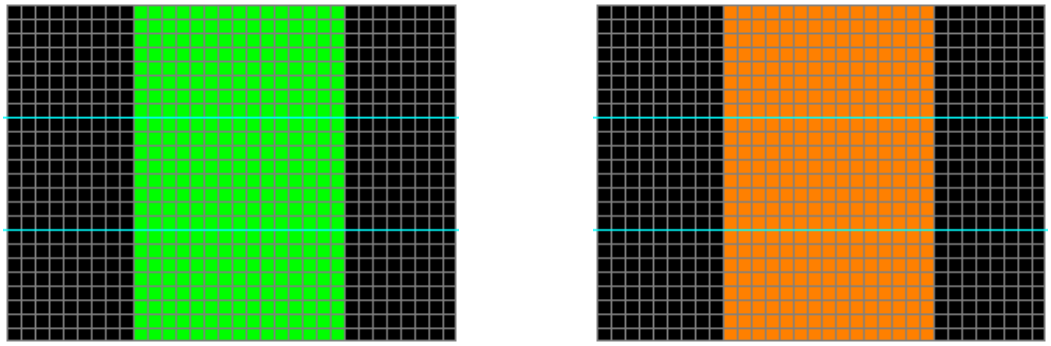


Figura 2.6. - Distribuição das imagens na tela.

A cor verde representa os dados da imagem A, enquanto que a cor laranja mostra os dados da imagem B.

Uma vez carregadas as imagens, utilizaremos a tabela de nomes para posicionar a imagem corretamente na tela e também alternar a exibição da imagem A e da imagem B, conforme mostra a figura 2.7.



(a) imagem A (a) imagem B

Figura 2.7. Posicionamento das imagens e alternância delas.

Uma forma de atingirmos esse objetivo é criar um mapa na RAM contendo as posições dos blocos da imagens A e B nas tabelas de padrão/cor. Então, copiamos o mapa da imagem A para a VRAM na área da tabela de nomes. Depois, repetimos a operação para o mapa da imagem B. Não copie os dois mapas de uma vez, pois a exibição alternada não poderá ser visualizada.

Esse artigo acompanha um código em Assembly para alternar as imagens, um mapa da tabela de nomes para a imagem A e B e exemplos de imagens com 120 cores. Para rodar um exemplo, digite no Basic:

```
BLOAD "IMAGEM.120",R
```

O programa muda automaticamente o modo de tela para screen 2 e exibe as duas imagens alternadamente. Ao pressionar uma tecla qualquer, o programa é interrompido e retorna-se à screen 0.

2- Função de Custo – as melhores 2/4 cores para um grupo de 1x8 pontos

A função de custo foi utilizada por Daniel Vik [1] na conversão de uma imagem de 24 bits para 15 cores do MSX 1. O objetivo é calcular o erro de cada pixel de um grupo de 1x8 pontos em relação a todas as combinações possíveis de duas cores do MSX 1. O par de cores que obtiver o menor erro total em relação aos pontos é eleito como representante do grupo.

A técnica consiste em criar uma tabela com todas as combinações de 2 cores sem repetição, e então comparar cada ponto do grupo de 1x8 pontos com todos os pares de cores dessa matriz.

Índice	Cor 1	Cor 2
1	1	2
2	1	3
3	1	4
4	1	5
...	...	...
104	13	14
105	14	15

Tabela 2.1 – Combinação de cores

O Erro Quadrático (SE em inglês) é calculado para cada ponto do grupo de 1x8 pontos em relação a cada par de cores da tabela.

$$SE(r,n)=(R_r-R_n)^2+(G_r-G_n)^2+(B_r-B_n)^2$$

Onde:

- r – cor a ser testada da imagem original.
- n – cor 1 ou 2 do par da tabela.
- R – componente de cor vermelha.
- G – componente de cor verde.
- B – componente de cor azul.

Então, calculamos o Erro Mínimo Quadrático (MSE em inglês):

$$MSE(r,n)=min[SE(r,1),SE(r,2)]$$

Dessa forma, assumimos o erro da cor do par atual da tabela que obtiver o menor erro em relação ao ponto testado. Ao final, somamos os erros dos 8 pontos do grupo e esse erro é o erro calculado para cada par de cores tabela. O par de cores da tabela com menor erro é então escolhido para representar o grupo.

Vejamos como funciona isso na prática.

Seja um grupo de 1x8 pontos, onde a cor do ponto 1 é equivalente a cor 7 do MSX 1. Ao testar esse ponto com a linha 4 da tabela de combinação de cores (tabela 2.1), devemos calcular o SE da cor 7 em relação a cor 1 e da cor 7 para a cor 5.

```
Erro 7-5: 12.100
Erro 7-1: 118.315
```

Ao calcularmos os erros, constatamos que o erro da cor 7 para a 5 é menor do que o erro da cor 7 para a 1, ou seja, a cor 7 (ciano) se parece mais com a cor 5 (azul) do que com a cor 1 (preta). Dessa forma, assumimos o valor do erro 7-5 como o erro para o ponto 1. A vantagem de fazer isso é que consideramos somente o erro de cores mais parecidas, eliminando a interferência de cores menos parecidas no cálculo geral.

Repetimos o cálculo do erro para os demais 7 pontos do grupo ainda para a linha 4 e somamos o erro, obtendo o MSE total para o par.

$$MSE_{total} = \sum_{i=1}^{n} MSE_i$$

Repetimos toda a operação para as demais linhas da matriz e aquela linha que tiver o menor erro MSE será a melhor combinação de cores escolhida para o grupo.

Ao descobrirmos o melhor par de cores para representar um grupo de 1x8 pontos, devemos calcular as cores finais para cada ponto do grupo. Dessa forma, verificamos qual das duas cores é a cor mais parecida com o ponto em questão. Isso pode ser feito através da Distância Euclidiana:

$$d = \sqrt{(R_r - R_n)^2 + (G_r - G_n)^2 + (B_r - B_n)^2}$$

A cor que apresentar a menor distância é a cor que cada ponto irá assumir (cor de frente ou cor de fundo).

2.1. Adicionando Error Diffusion ao processo

Nos trabalhos de Jannone, Robsy e Ragozini [2,3], foi adicionado o Error Diffusion [4] à conversão da imagem, o que resulta em impressionantes imagens para as limitações da screen 2.

A técnica de Error Diffusion espalha o erro de quantização de um ponto para seus vizinhos à direita e abaixo, minimizando os efeitos das Bandas de Mach [4]. Esse erro é calculado como a diferença entre o valor RGB do ponto original para o novo valor RGB obtido.

Devido às características da técnica da função de custo, só iremos obter a nova cor de um ponto de um grupo de 1x8 pontos após processar todos os 8 pontos desse grupo e descobrir o par de cores. Dessa forma, o espalhamento deve ser feito em cada ponto somente depois de aplicada a função de custo.

Entretanto, há um pequeno problema quando espalhamos o erro em pontos do grupo, isto porque estamos modificando a cor original de um de seus membros, comprometendo o

cálculo do par de cores para representar o grupo. Dessa forma, a cada espalhamento de erro que afete um ponto do grupo, devemos calcular novamente a função de custo e determinar o novo par de cores.

Felizmente, não iremos encontrar problemas em espalhar o erro em pontos localizados abaixo do grupo, pois eles se encontram em outros grupos de 1x8 pontos que irão ser processados mais adiante.

O pseudo-algoritmo é apresentado a seguir.

```
// Para o grupo
para i ← 0 até 7 faça
      [c1 c2] ← funcao_custo(grupo)
      quant_erro ← ponto(x+i, y) – melhor_cor(ponto(x+i, y), c1, c2)
      ponto(x+i+1, y) ← ponto(x+i+1, y) + quant_erro * 7/16
fim_para

// Para os demais
para i ← 0 até 7 faça
      quant_erro ← ponto(x+i, y) – melhor_cor(ponto(x+i, y), c1, c2)
      ponto(x+i, y) ← melhor_cor(ponto(x+i, y), c1, c2)

      ponto(x+i-1, y) ← ponto(x+i-1, y) + quant_erro * 3/16
      ponto(x+i, y) ← ponto(x+i, y) + quant_erro * 5/16
      ponto(x+i+1, y) ← ponto(x+i+1, y) + quant_erro * 1/16
fim_para
```

Realizar o cálculo da função de custo oito vezes para cada grupo de 1x8 pontos compromete bastante o desempenho do algoritmo. Quando realizamos o cálculo da função de custo apenas uma vez e o espalhamento da mesma linha junto com os demais, como é feito normalmente para o Error Diffusion, o resultado não é muito comprometido. A figura 3.1 compara o resultado obtido com o algoritmo ideal e o que ignora a reclassificação dos pontos do mesmo grupo. Podemos observar na figura 3.1 (b) o aparecimento de algumas linhas com a mesma cor, devido ao não espalhamento do erro nelas.

(a) (b)

Figura 3.1. Comparação do algoritmo ideal (a) e do que ignora a reclassificação do grupo (b).

A figura 3.2 apresenta mais resultados do Error Diffusion com o algoritmo ideal e a função de custo aplicados a imagens de 24 bits, de modo a convertê-las para a screen 2 do MSX 1.

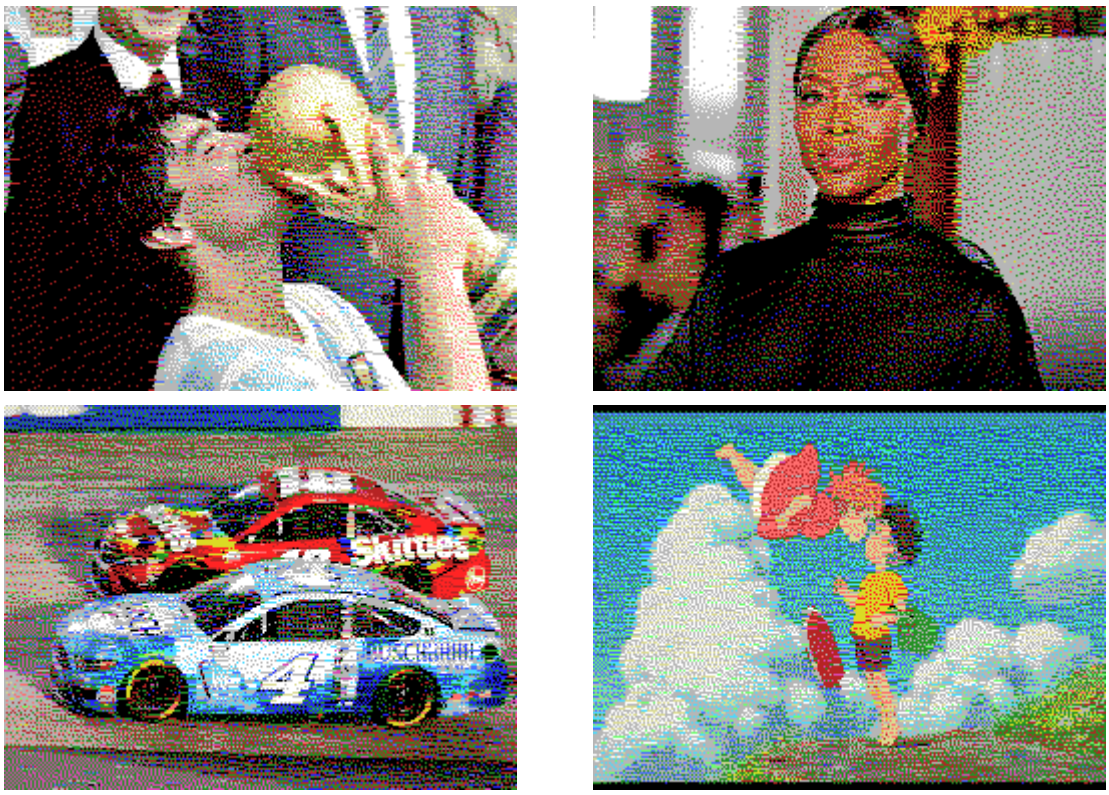


Figura 3.2. Resultado da conversão de cores de imagens de 24 bits para a screen 2 do MSX.

Créditos e Bibliografia

Este artigo foi escrito por Marcelo Silveira, em dezembro de 2020.

Data: 12 / 2020
E-mail: flamar98@hotmail.com
Homepage: http://marmsx.msxall.com

Referências Bibliográficas:

[1] – BMPto105, Daniel Vik, 2006. http://vik.cc/dvik-joyrex/download/105Colors.ppt
[2] – MSX Screen Convertor, Rafael Janone. http://msx.jannone.org/conv/
[3] – TMSOPT v.0.1, Eduardo Robsy, Arturo Ragozini e Rafael Janone, 2007.
[4] – Artigo Error Diffusion – Marcelo Silveira. http://marmsx.msxall.com/artigos
[5] – MSX Viewer 5 – Marcelo Silveira. http://marmsx.msxall.com/msxvw/msxvw5