ALGORITMO DE BRESENHAM: O USO MICROCONTROLADORES PARA TRAÇAR RETAS EM LCDs

Jefferson Zortea Moro

Seminário
Departamento de Engenharia Elétrica - Universidade Federal do Espírito Santo
Cx. Postal 01-9011 - Vitória - ES - 29060-970 – BRASIL
jefferson.moro@gmail.com

Resumo – Deseja-se com esse artigo mostrar os benefícios proporcionados por um algoritmo que usa apenas variáveis inteiras para traçar retas em displays LCD. As vantagens são aumentadas se for lembrado que o tempo computacional é um bem escasso em alguns circuitos como os micro-controlados.

I – INTRODUÇÃO

Quando um engenheiro precisa fazer um projeto de algum circuito eletrônico, ele tem várias decisões a serem tomadas. Talvez a primeira delas seja a de decidir qual plataforma de desenvolvimento será usada. Têm-se hoje várias alternativas no mercado: FPGAs, PSOCs, PLCs, microcontroladores etc. Quando uma das premissas desse projeto é ter uma interface homem máquina, aí aparece uma nova gama de caminhos a se seguir: LEDs, displays de sete seguimentos, alto-falantes, displays LCD, etc. Escolhido o hardware, uma segunda tarefa incumbida ao projetista é a de desenvolver o código a ser "gravado" no componente controlador, a fim de atribuir valor funcional ao circuito. Às vezes o tempo despendido pelo circuito na interface homem máquina é tão grande, que chega a atrapalhar o seu desempenho. Gerar códigos mais eficientes pode ser a solução para esse problema.

Na seção II desse artigo será dada uma motivação ao uso de algoritmos otimizadores para aplicações em microcontroladores. Na seção III, será apresentado o *Algoritmo de Bresenham*, e na seção IV será feita a conclusão. Vale a pena olhar o apêndice A que trás alguns algoritmos interessantes.

II – MOTIVAÇÃO

Traçar curvas elementares, como segmentos de reta ou arcos de circunferência, requer a construção de algoritmos capazes de determinar na matriz de pixels da superfície de exibição quais pixels devem ser alterados de forma a simular a aparência do elemento gráfico desejado.

Hoje em dia, com computadores que possuem freqüência de trabalho da ordem de 2GHz, munidos com vários processadores auxiliares (co-processadores) que fazem, por exemplo, conta com ponto flutuante, ficou fácil traçar curvas elementares. Prova disso é a perfeição encontrada nos jogos, com cenários de aparência bem reais. Mas por outro lado, os microcontroladores estão se tornando cada vez mais usados, inclusive para aplicações gráficas.

Microcontroladores podem ser vistos como microprocessadores envolvidos por vários periféricos, tudo dentro de um mesmo chip. Esses periféricos são contadores, comparadores, interfaces seriais, e vários outros dispositivos que fazem dos microcontroladores CIs muito funcionais e conseqüentemente, de grande aplicabilidade em projetos de eletrônica.

Sabe-se que a potencia dissipada em um microcontrolador é diretamente proporcional ao produto do quadrado da tensão de alimentação com a freqüência de trabalho do mesmo. Sabe-se também que o preço dos circuitos integrados cresce se for aumentada a freqüência na qual ele trabalhará. Por esses e outros motivos, os fabricantes de microcontroladores geralmente optam por fabricar dispositivos que operam em freqüências não muito elevadas (por volta de 8Mhz).

Gráficos complexos requerem o traçado de uma grande quantidade de segmentos de reta, e a velocidade é importante. Se as contas forem feitas por um

microprocessador moderno, não haverá problemas, contudo, se essa tarefa for incumbida a um microntrolador, o gráfico pode aparecer no visor depois de um longo tempo, e isso é desconfortável para nossa visão, sendo dessa forma impraticável animações em tempo real.

Uma alternativa muito interessante para traçar curvas, gastando para isso um tempo de processamento bem menor, é conseguida com o uso do *Algoritmo de Bresenham.* Na próxima seção, mostrar-se-á o processo de construção desse algoritmo para traçar uma reta. Outras curvas podem ser conseguidas usando o mesmo princípio (ver Apêndice A).

III – O ALGORITMO

Para começar, propõe-se uma pergunta: como você escreveria um algoritmo para traçar uma reta, usando, por exemplo, a linguagem do Matlab? Talvez o algoritmo que você pensou seja parecido com o descrito na figura 1.

```
%Traça uma linha entre os
%pontos (x1,y1) e (x2,y2)
function []=line_tradicional(x1,y1,x2,y2)
a = (y2 - y1)/(x2 - x1); %coeficiente angular
for (x = x1:1:x2)
    y = a*(x - x1) + y1; %função de reta
    y = round(y);        %para fazer y inteiro
    plot(x,y,'ro');      %plota ponto
    hold on;
end;
axis equal;              %ajuste da tela
```

Figura 1- Algoritmo simples para traçar uma reta.

Esse algoritmo é pouco eficiente, pois, para todos os valores de *x*, uma conta que usa ponto flutuante deve ser efetuada. Contas com ponto flutuante exigem do processador vários ciclos de clock para serem executadas. Algoritmos de alto nível podem ser usados para minimizar o esforço computacional e assim tornar o processo mais rápido. O mais famoso desses algoritmos foi proposto em 1965 por Jack E. Bresenham, um então funcionário da IBM, formado em ciência da computação [1]. O *Algoritmo de Bresenham* é um algoritmo clássico para traçar curvas, ele usa apenas variáveis inteiras e permite que o cálculo de um próximo ponto (x_{i+1},y_{i+1}) seja feito de forma incremental, usando os cálculos já feitos para o ponto anterior (x_i,y_i).

Suponha que a superfície de exibição (LCD) possua igual densidade de pixels na horizontal e na vertical (razão de aspecto gráfica igual a 1). O algoritmo assume que a inclinação da linha está entre zero (0) e um (1), (outras inclinações podem ser tratadas por simetria). O ponto (x_1,y_1) seria o inferior esquerdo, e (x_2,y_2) o superior direito.

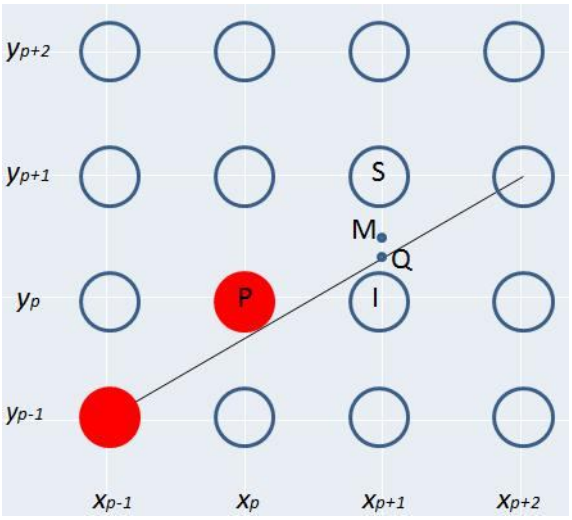

Figura 2 – Tomada de decisão.

Na figura 2, assumindo que o pixel que acabou de ser selecionado é P, em (x_p,y_p), e o próximo deve ser escolhido entre o pixel à direita *superior* a M (pixel S) e o pixel à direita *inferior* a M (pixel I). Seja Q o ponto de intersecção entre a reta e a coluna $x = x_p + 1$ da malha, e M o ponto intermediário entre os pixels S e I, o que se faz é observar de que lado da reta está o ponto M [2]. É fácil verificar que se M está acima de Q, o pixel I está mais próximo da reta; se M está abaixo de Q, S está mais próximo. Dessa forma, o teste do ponto-médio permite a escolha do pixel mais próximo da reta. Veja que assumindo uma distancia normalizada e igual a 1 (adimensional) entre os pixels adjacentes, tem-se com esse algoritmo um erro de no máximo ½ (adimensional), que é a metade da

distância entre dois pixels vizinhos da mesma coluna.

Precisa-se agora de um método para calcular de que lado da reta está o ponto M. Se $\Delta y = y_2 - y_1$, e $\Delta x = x_2 - x_1$, pode-se escrever a equação da reta:

$$y = \frac{\Delta y}{\Delta x} x + B$$

Tem-se então uma equação implícita *F(x,y) :*

$$F(x, y) = y - \frac{\Delta y}{\Delta x} x - B = 0$$

$$F(x, y) = x\Delta y - y\Delta x + B\Delta x = 0$$

Pode-se reescrever a equação acima como:

$$F(x, y) = ax + by + c = 0$$

$$a = \Delta y \quad b = -\Delta x \quad c = B\Delta x$$

Verifica-se que *F(x,y)* é 0 (zero) para pontos sobre a reta, positiva para pontos abaixo dela, e negativa para pontos acima dela. Com base nisso, para o teste do ponto-médio, basta calcular $F(M) = F(x_p + 1, y_p + 1/2)$ e verificar o seu sinal. Como a decisão será tomada com base no valor da função no ponto *(x_p + 1,y_p + 1/2)*, defini-se uma "variável de decisão" d:

$$d = F(M) = a(x_p + 1) + b(y_p + 1/2) + c$$

Se *d > 0*, significa que M está abaixo de onde a curva ideal passa, então é escolhido o pixel S; se *d < 0*, significa que M está acima de onde a reta ideal passa, então é escolhido o pixel I. No possível caso em que *d = 0* é escolhido qualquer um dos dois pixels, S ou I.

Após a decisão de qual pixel será considerado, deve-se atualizar o valor de *d*. Se I foi o último pixel escolhido, M será incrementado somente na direção *x*. Têm-se então as seguintes igualdades:

$$d_{old} = F(x_p + 1, y_p + 1/2) = a(x_p + 1) + b(y_p + 1/2) + c$$

$$d_{new} = F(x_p + 2, y_p + 1/2) = a(x_p + 2) + b(y_p + 1/2) + c$$

Subtraindo d_{old} de d_{new} para obter a diferença incremental, tem-se $d_{new} = d_{old} + a$. Dessa forma, quando o último pixel escolhido é I, deve-se incrementar d de $a = \Delta y$. Em outras palavras, pode-se derivar o valor da variável de decisão do próximo passo a partir do seu valor atual, sem necessidade de calcular *F(M)* diretamente.

Por outro lado, se S foi escolhido, M é incrementado de 1 em ambas as direções, *x* e *y*. Obtém-se o seguinte:

$$d_{old} = F(x_p + 1, y_p + 1/2) = a(x_p + 1) + b(y_p + 1/2) + c$$

$$d_{new} = F(x_p + 2, y_p + 3/2) = a(x_p + 2) + b(y_p + 3/2) + c$$

Subtraindo d_{old} de d_{new} para obter a diferença incremental, tem-se $d_{new} = d_{old} + a + b$. Dessa forma, quando o último pixel escolhido é S, deve-se incrementar d de $a + b = \Delta y - \Delta x$. Em outras palavras, pode-se derivar o valor da variável de decisão do próximo passo a partir do seu valor atual, sem necessidade de calcular *F(M)* diretamente.

Já se tem quase tudo para escrever o *Algoritmo de Bresenham, só* falta decidir qual será o valor inicial de *d* [3]. Sendo (x_1, y_1) o primeiro ponto a ser traçado, a próxima decisão será feita para $(x_1 + 1, y_1 + 1/2)$:

$$d_{start} = F(x_1 + 1, y_1 + 1/2) = a(x_1 + 1) + b(y_1 + 1/2) + c$$

$$d_{start} = ax_1 + by_1 + c + b/2 + a = F(x_1, y_1) + b/2 + a$$

Sendo *(x_1,y_1)* um ponto da reta, *F(x_1, y_1)*= 0. Dessa forma:

$$d_{start} = b/2 + a = \Delta y - \Delta x/2$$

Pode-se eliminar a fração acima multiplicando $F(x, y)$ por 2. Isto multiplica cada constante e a variável de decisão por 2, mas não afeta o sinal da variável de decisão, que é o que interessa para o teste do ponto-

médio. Veja um resumo dos limiares de decisão:

$d_{start} = 2\Delta y - \Delta x;$ *condição inicial*

$d_{new} = d_{old} + 2\Delta y;$ *se I foi escolhido*

$d_{new} = d_{old} + 2(\Delta y - \Delta x);$ *se S foi escolhido*

Com esses valores em mãos, o próximo passo é desenvolver realmente o algoritmo. Na figura 3, uma versão simplificada do *Algoritmo de Bresenham* pode ser verificada. Note que as operações mais complexas a serem feitas são multiplicações por 2, que na lógica binária nada mais é do que deslocamento de um bit para a esquerda.

```
%Traça uma linha entre (x1,y1) e (x2,y2)
%usando o algoritmo de Bresenham:
%       line_bresenham(x1,y1,x2,y2)
function []=line_bresenham(x1,y1,x2,y2)

x = x1;
y = y1;                 %inicialização
dx = x2 - x1;
dy = y2 - y1;
incx = 1;
incy = 1;

plot(x,y,'ro');         %plota primeiro valor
hold on;                %mantém imagem

d = (2*dy - dx);        %valor de decisão inicial
incI = 2*dy;        %incremento se I foi escolhido
incS = 2*(dy-dx);   %incremento se S foi escolhido
while(x ~= x2)      %enquanto não é x2, faça:
    if(d <= 0)      %ponto acima ou abaixo da reta
        d = d + incI;  %escolhe I (pixel de baixo)
        x = x + incx;  %incrementa x
    else
        d = d + incS;  %escolhe S (pixel de cima)
        x = x + incx;  %incrementa x
        y = y + incy;  %incrementa y
    end;
    plot(x,y,'ro');     %plota pixel escolhido
end;

axis equal;
```

Figura 3 – Algoritmo de Bresenham para traçar retas, em sua forma simplificada.

CONCLUSÃO

Com o *Algoritmo de Bresenham*, um novo dinamismo é apresentado para o uso dos microcontroladores. Códigos como os sugeridos por Bresenham, além de acrescentarem bons efeitos aos LCDs e darem maior dinamismo ao circuito, também possuem a vantagem de não serem longos, o que possibilita economia de espaço de memória, uma vez que tal recurso é diretamente proporcional ao número de linhas de comando.

REFERÊNCIAS

[1] http://en.wikipedia.org/wiki/Jack_E._Bresenham. Acessado em 18/05/2009

[2] http://www.dpi.ufv.br/disciplinas/inf390/files/paginas/gbdi.icmc.sc.usp.br/documentacao/apostilas/cg/. Acessado em 18/05/2009

[3] J. Bresenham. Algorithm for computer control of a digital plotter. IBM, Systems Journal, 4(1):25–30, 1965.

APÊNDICE – A

Algoritmo para traçar circunferências:

```
%Desenha um círculo de raio r
%centralizado no ponto (xc,yc):

function[]=circle(r,xc,yc)
x = 0;
y = r;
d = (1 - r);
hold on;
plot(xc,yc+r,'rs'); %ponto extremo superior
plot(xc,yc-r,'rs'); %ponto extremo inferior
plot(xc+r,yc,'rs'); %ponto extremo direita
plot(xc-r,yc,'rs'); %ponto extremo esquerda
hold on;
while(y >= x)
   if(d < 0)   %seleciona E (pixel superior)
      d = (d + 2*x + 3);
      x = x + 1;
   else       %seleciona SE (pixel inferior)
      d = (d + 2*(x - y) + 5);
      x = x + 1;
      y = y - 1;
   end;
   plot(x+xc,y+yc,'rs');   %segundo octante
   plot(x+xc,yc-y,'rs');   %sétimo octante
   plot(y+xc,x+yc,'rs');   %primeiro octante
   plot(y+xc,yc-x,'rs');   %oitavo octante
   plot(xc-x,yc+y,'rs');   %terceiro octante
   plot(xc-x,yc-y,'rs');   %sexto octante
   plot(xc-y,yc-x,'rs');   %quinto octante
   plot(xc-y,x+yc,'rs');   %quarto octante
end;
axis equal;
```

Algoritmo para traçar elipses:

```
%Desenha uma elípse com eixo maior igual a "a" e eixo
%menor igual a "b" centralizada no ponto (xc,yc):

function[]=elipse(a,b,xc,yc)
x = 0; y = b;
d1 = b*b - a*a*b + a*a/4;
hold on;
plot(xc,yc+b,'rs'); %ponto extremo superior
plot(xc,yc-b,'rs'); %ponto extremo inferior
plot(xc+a,yc,'rs'); %ponto extremo direita
plot(xc-a,yc,'rs'); %ponto extremo esquerda
hold on;
while(a*a*(y-1/2) > b*b*(x+1))
   if(d1 < 0)   %seleciona E (pixel superior)
      d1 = (d1 + b*b*(2*x + 3));
      x = x + 1;
   else        %seleciona SE (pixel inferior)
      d1 = (d1 + b*b*(2*x + 3) + a*a*(-2*y + 2));
      x = x + 1;
      y = y - 1;
   end;
   plot(x+xc,y+yc,'rs');   %primeiro quadrante
   plot(x+xc,yc-y,'rs');   %quarto quadrante
   plot(xc-x,yc+y,'rs');   %segundo quadrante
   plot(xc-x,yc-y,'rs');   %terceiro
end;
d2 = (b*b*(x+1/2)*(x+1/2)+a*a*(y-1)*(y-1)-a*a*b*b);
while(y > 0)
   if(d2 < 0)   %seleciona E (pixel superior)
      d2 = (d2 + b*b*(2*x + 2) + a*a*(-2*y + 3));
      x = x + 1;
      y = y - 1;
   else        %seleciona SE (pixel inferior)
      d2 = (d2 + a*a*(-2*y + 3));
      y = y - 1;
   end;
   plot(x+xc,y+yc,'rs');   %primeiro quadrante
   plot(x+xc,yc-y,'rs');   %quarto quadrante
   plot(xc-x,yc+y,'rs');   %segundo quadrante
   plot(xc-x,yc-y,'rs');   %terceiro quadrante
end;
axis equal;
```

Algoritmo para traçar retas em qualquer quadrante:

```
%Traça uma linha entre os pontos 1 e 2
%usando o algoritmo de Bresenham:

function []=line(x1,y1,x2,y2)
if(x2 >= x1)
   dx = (x2 - x1);  %estabelecendo o módulo de dx
   incx = 1;            %da esquerda para a direita
else
   dx=(x1 - x2);
   incx = -1;       %da direita para a esquerda
end;
if(y2 >= y1)
   dy =(y2 - y1);   %estabelecndo o módulo de dy
   incy = 1;            %de baixo para cima
else
   dy = (y1 - y2);
   incy = -1;         %de cima para baixo
end;
x = x1;
y = y1;
plot(x,y,'ro');        %pinta primeiro valor
hold on;
if (dx==0)           % é uma coluna, plota mais rápido!!
          if (incy > 0)
      for y=y1:1:y2
         plot(x,y,'ro');
      end;
   else
      for( y=y2:1:y1 )
         plot(x,y,'ro');
      end;
   end;
elseif (dy==0)      %é uma linha,plota mais rápido!!!
   if (incx > 0)
      for( x=x1:1:x2 )
         plot(x,y,'ro');
      end;
   else
      for( x=x2:1:x1 )
         plot(x,y,'ro');
      end;
   end;
else        %inclinação diferente de 0 ou 90 graus
   if(dx >= dy)
      d = (2*dy - dx);      %valor de decisão inicial
      incE = 2*dy;          %incremento em direção a E
      incNE = 2*(dy-dx); %incremento em direção a NE
      while(x ~= x2)
         if(d <= 0)         %verifica onde está o ponto
            d = d + incE;   %escolhe E (pixel de baixo)
            x = x + incx;   %incrementa ou decrementa x
         else
            d = d + incNE;  %escolhe NE (pixel de cima)
           x = x + incx;
            y = y + incy;   %incrementa ou decrementa y
         end;
         plot(x,y,'ro');
      end;
   else
      d = (2*dx - dy);
      incE = 2*dx;
      incNE= 2*(dx - dy);
      while(y ~= y2)
         if(d <= 0)
            d = d + incE;   %escolhe E
            y = y + incy;
         else
            d = d + incNE;  %escolhe NE
            y = y + incy;
            x = x + incx;
```

```
            end;
            plot(x,y,'ro');
        end;
    end;
end;
axis equal;
```