TRACÉS DE DROITES SUR µCONTRÔLEUR

Les afficheurs LCD «classiques» de type caractères disparaissent progressivement au profit d'afficheurs graphiques. Ces afficheurs ont généralement un mode alphanumérique qui permet de les utiliser comme leurs prédécesseurs et disposent en outre d'un mode complètement graphique qui permet d'accéder indépendamment aux différents pixels composant l'écran. Ces nouvelles potentialités facilitent la réalisation de tracés de courbes ou le dessin de fenêtres pour améliorer la présentation des applications à microprocesseur.

Parmi les primitives graphiques nécessaires, la première est la primitive de tracé de droite. Si le tracé de droites horizontales ou verticales ne pose pas de problème (ainsi que les droites à 45°), il en va différemment pour le tracé de droites d'inclinaison quelconque.

Nous allons voir ce mois-ci comment fonctionne l'algorithme le plus répandu de tracé de droite, celle-ci étant définie par son point de départ (X1, Y1) et son point d'arrivée (X2, Y2). L'algorithme choisi doit satisfaire plusieurs critères :
- les lignes doivent apparaître comme continues (c'est à dire sans trous),
- les lignes doivent commencer et finir à une position précise,
- les lignes doivent être tracées rapidement.

Les points de départ et d'arrivée étant connus, la première idée qui vient à l'esprit est de calculer la pente de la droite a = (Y2 - Y1) / (X2 - X1) et de réécrire l'équation de la droite sous la forme Y = aX + b. On pourra donc pour chaque X calculer la valeur Y correspondante. Cette méthode oblige à calculer pour chaque point un produit et une addition en utilisant des opérations en calcul flottant puisque le rapport (Y2 - Y1)/(X2 - X1) n'est pas forcément une valeur entière. La lenteur des opérations flottantes sur les petits microcontrôleurs nous fera donc écarter cette solution.

La deuxième idée est de tracer la droite de manière incrémentale, c'est à dire en utilisant la valeur du dernier pixel tracé pour tracer le suivant. On remarquera que si on ne s'intéresse qu'à des segments dont l'angle par rapport à l'horizontale est compris entre 0 et 45 degrés, et si on trace le segment de la droite vers la gauche, la position verticale de chaque nouveau pixel est soit identique au pixel précédent, soit égale à la valeur verticale du pixel précédent plus 1. A la limite, pour un segment incliné à 45°, la position verticale de chaque nouveau pixel est incrémentée de 1 chaque fois que l'on se déplace d'un pixel en horizontal. On a donc :

$Yi+i = Yi + \Delta Y$

$Yi+i = Yi + \Delta X*((Y2 - Y1)/(X2 - X1))$

$Xi+i = Xi + \Delta X.$

En prenant $\Delta X = 1$, la nouvelle valeur du pixel sera :

Position_X = Xi+i
Position_Y = Valeur Entière(Yi+i).

Cet algorithme a l'inconvénient d'accumuler une valeur flottante ce qui, à la longue, aboutit à une imprécision sur la position du pixel, particulièrement pour celui qui correspond au point terminal. De plus, les pixels tracés peuvent être différents si on inverse le point de départ et le point d'arrivée.

L'algorithme de Bresenham permet de s'affranchir de ces problèmes. Voici sa version simplifié pour une droite dont la pente est inférieure à un angle de 45° :

```
;
; Tracé de droite. Algorithme de Bresenham
; trace la ligne depuis le point R4:R5 jusqu'au point R6:R7. R4 et R6 sont
; les «x» et R5, R7 sont les «y» L'espace dans lequel on trace est un carré
; pour lequel x et y varient de 0 a 255. La fonction qui trace le pixel est
; la fonction PIXEL qui a pour parametre R6 = X, R7 = Y
;
SEGMENT_DATA    SEGMENT DATA
SEGMENT_BIT     SEGMENT BIT
SEGMENT_CODE    SEGMENT CODE

                RSEG    SEGMENT_DATA
X_PIX :         DS      1               ; position du pixel courant
Y_PIX :         DS      1
DX :            DS      1               ; deltax
DY :            DS      1               ; deltay
INCX :          DS      1               ; incrément de X_PIX
INCY :          DS      1               ; incrément de Y_PIX
CPT :           DS      1               ; compteur de 1 a deltax ou deltay
ERREUR :        DS      2               ; l'erreur peut être supérieure à 255

                RSEG    SEGMENT_BIT
CHANGE :        DBIT    1               ; indique |DY| > |DX|

MSB             EQU     0
LSB             EQU     1
```

■ *Suite page suivante*

```
début
X = X1
Y = Y1
ΔX = X2 - X1
ΔY = Y2 - Y1
e = ΔY /ΔX - 1/2
pour i = 1 à ΔX
        Trace(X, Y)
        tant que (e >= 0)
                Y = Y+1
                e = e-1
        fin tant que
X = X+1
e = e+ΔY /ΔX
fin pour i ...
fin
```

Cet algorithme simplifié a encore une variable (e) flottante. On voit que dans le calcul, seul le signe de e intervient. On peut donc, en multipliant les équations où e apparaît par (2 x ΔX) transformer l'algorithme de manière à n'utiliser que des nombres entiers. Par ailleurs, en inversant le rôle de X et Y, on pourra tracer des droites dont la pente est supérieure à un. Voici l'algorithme de Bresenham définitif qui permet de tracer un segment dont on définit le point de départ (X1, Y1) et le point d'arrivée (X2, Y2) :

```
début
X = X1
Y = Y1
ΔX = abs(X2 - X1)
ΔY = abs(Y2 - Y1)
S1 = Signe(X2 - X1)
S2 = Signe(Y2 - Y2)
si ΔY >ΔX  alors
        tampon = ΔX
        ΔX = ΔY
        ΔY = tampon
        XYchange = 1
sinon
        XYchange = 0
fin si
e = 2 * ΔY - ΔX
pour i = 1 à ΔX
        Trace(X, Y)
        tant que (e >= 0)
                si XYchange = 1 alors
                        X = X+S1
                sinon
                        Y = Y+S2
                fin si
                e = e - (2 * ΔX)
        fin tant que
si XYchange = 1 alors
        Y = Y+S2
sinon
        X = X+S1
fin si
e = e + (2 * ΔY)
fin pour i ...
fin
```

Le **listing 1** est la traduction en assembleur 80X51 de l'algorithme de Bresenham. Il permet de tracer des segments dans un espace dont les coordonnées X et Y des points peuvent prendre toutes les valeurs de 0 à 255.

```
                RSEG    SEGMENT_CODE
; BRESENHAM... Tracé de droite ((R4, R5) ; (R6, R7))
LIGNE :         MOV     A, R4
                MOV     X_PIX, A
                MOV     A, R5
                MOV     Y_PIX,A
                CLR     A
                MOV     INCX, A                 ; incrément X_PIX = 0
                MOV     INCY, A                 ; incrément Y_PIX = 0
                MOV     ERREUR+MSB, A
                CLR     CHANGE
                CLR     C                       ; calcule DY = |Y2 - Y1|
                MOV     A, R7
                SUBB    A, R5
                JZ      LINE0                   ; si = 0 continue
                INC     INCY                    ; INCY = 1 si DY >= 0
                JNC     LINE0                   ; DY >= 0
                CPL     A                       ; DY < 0, complémente a 2 (inverse)
                INC     A
                DEC     INCY
                DEC     INCY                    ; INCY = -1
LINE0 :         MOV     DY, A                   ; DY = |Y2 - Y1|
; après avoir calcule dy, calcule dx.
        CLR     C
        MOV A, R6                               ; calcule DX = |X2 - X1|
        SUBB A, R4
        JZ      LINE1                           ; si <> 0, continue
        INC     INCX                            ; INCX = 1 si DX >= 0
        JNC     LINE1                           ; DX >= 0
                CPL     A                       ; DX < 0, complémente a 2 (inverse)
                INC     A
                DEC     INCX
                DEC     INCX                    ; INCX = -1 si DX < 0
LINE1 :         MOV     DX , A                  ; DX = |X2 - X1|
; compare IDXI et IDYI pour savoir si la pente est > à 1
                CJNE    A , DY , LINE4
LINE4 :         JNC     LINE5                   ; |DY| <= |DX|, pente <= 1
                XCH     A, DY                   ; |DY| > |DX|, inverse DX et DY
                MOV     DX, A
                SETB    CHANGE                  ; indique que DX et DY interchangés
                MOV     A, DX                   ; ancien |DY|
; valide le compteur de boucle
LINE5 :         MOV     CPT,A                   ; compte jusqu'à |DX| ou |DY|
; calcule l'erreur initiale
                MOV     A, DY                   ; |DY|
                ADD     A, DY                   ; X 2
                JNC     LINE6
                INC     ERREUR+MSB
LINE6 :         CLR     C
                SUBB    A, DX                   ; ( 2 X |DY|) - |DX|
                MOV     ERREUR+LSB, A
                JNC     LOOP
                DEC     ERREUR+MSB
; teste le compteur de boucle (for CPT = 1 to DX ...)
; ici on ne trace pas le dernier pixel pour permettre un raccordement en
; cas de tracés successifs. Si on désire tracer le dernier pixel, il faut
; modifier le contrôle de boucle pour effectuer la boucle une fois de plus,
; ou ajouter un appel a PIXEL en fin de tracé de ligne
LOOP :          MOV     A, CPT                  ; teste CPT = 0
                JZ      LOOPFN                  ; fin
; début de la boucle d'affichage de pixel
LOOPDB :
; appelle la fonction externe qui affiche le pixel R6, R7. Cette fonction
; est propre au hardware employé... A vous de l'écrire...
                MOV     A,X_PIX
                MOV     R6, A                   ; R6 = X_PIX
                MOV     A,Y_PIX
                MOV     R7, A                   ; R7 = Y_PIX
                CALL    PIXEL                   ; affiche le pixel (R6, R7)
; tant que ERREUR >= 0
ERRSUP :        MOV     A, ERREUR+MSB
                JB      ACC 7,ERRINF; MSB = 1, négatif
; suivant le flag CHANGE, incrémente X_PIX ou Y_PIX
                JNB     CHANGE, ERRSU0
                MOV     A,X_PIX                 ; incrémente X_PIX
                ADD     A, INCX                 ; de la valeur 1, 0, ou 1
                MOV     X_PIX, A
                AJMP    ERRSU1                  ; fin tant que
ERRSU0 :        MOV     A,Y_PIX                 ; Incrémente Y_PIX
                ADD     A, I NCY                ; de la valeur -1, 0, ou 1
                MOV     Y_PIX,A
ERRSU1 :
; calcule ERREUR = ERREUR-(2 X |DX|)
                CLR     C
                MOV     A, ERREUR+ LSB
                SUBB    A, DX
                JNC     ERRSU2
                DEC     ERREUR+MSB
                CLR     C
ERRSU2 :        SUBB    A,DX
                MOV     ERREUR+LSB, A
                JNC     ERRINF
                DEC     ERREUR+MSB
                AJMP    ERRSUP
; suivant change incrémente Y_PIX ou X_PIX
ERRINF          JNB     CHANGE, ERRIN0
                MOV     A, Y_PIX                ; incrémente Y_PIX
                ADD     A, INCY                 ; de la valeur -1, 0, ou 1
                MOV     Y_PIX, A
                AJMP    ERRIN1
ERRIN0 : MOV    A, X_PIX                        ; incrémente X_PIX
                ADD     A, INCX                 ; de la valeur -1, 0, ou 1
                MOV     X_PIX, A
; ERREUR = ERREUR + (2 X |DY|)
ERRIN1 :        MOV     A, ERREUR+LSB
                ADD     A,DY                    ; ERREUR + DY
                JNC     ERRIN2                  ; pas de retenue
                INC     ERREUR+MSB
ERRIN2 : ADD    A, DY                           ; ERREUR + DY + DY
                MOV     ERREUR+LSB, A
                JNC     ERRIN3
                INC     ERREUR+MSB
ERRIN3 :
; Fin de boucle. Décrémente le compteur de boucle et saute si différent de 0
                DJNZ CPT,LOOPDB
LOOPFN : RET                                    ; fin de boucle de 1 à CPT
```

Listing 1

J.-L. VERN