

MULTI-FENETRAGE DYNAMIQUE

Patrick GREUSSAY

Département d'Informatique
Université Paris VIII - Vincennes
75571 PARIS Cedex 12

&
C.N.R.S. LA 248 L.I.T.P.
2 place Jussieu
75005 Paris

RT-5-80

Mai 1980

Resumé :

Nous présentons les algorithmes de base permettant la réalisation de systèmes visuels d'entrées-sorties en multi-fenêtres. Ces systèmes sont interactifs et doivent permettre à leur utilisateur de reconfigurer à tout instant son dispositif multi-fenêtres de façon continue. Sous un tel système, l'écran de visualisation doit apparaître comme un plan de travail virtuel composé de documents multiples créables et déplaçables à volonté. Les algorithmes présentés visent à rendre, lors des déplacements de fenêtres, les transitions écran-écran les plus continues possible visuellement. Ces algorithmes ont permis, sur PDP11, la mise en place d'un tel système: WIN.

MOTS-CLES:

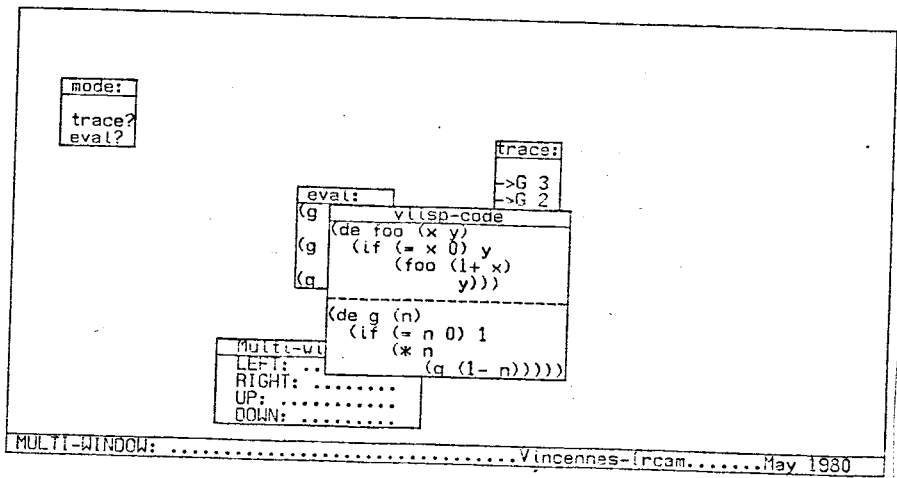
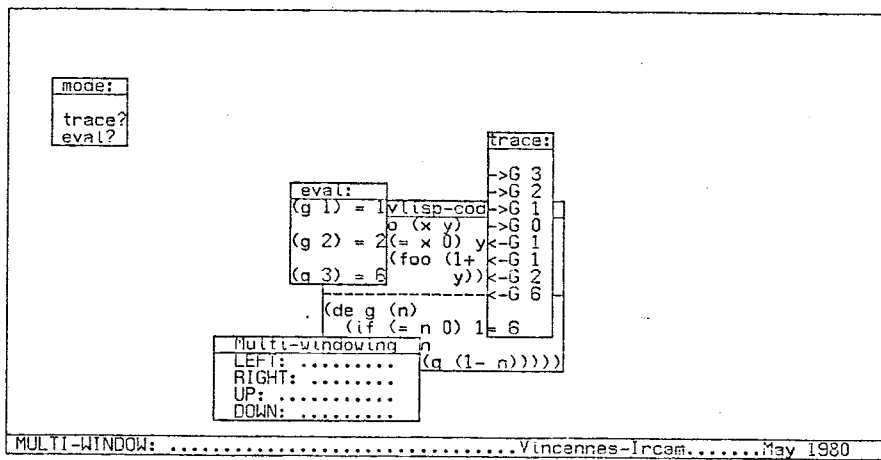
graphique, fenêtres, queue de priorité, bureautique, menus, éditeurs temps-réel, display-vecteurs, VLISP, représentations multiples.

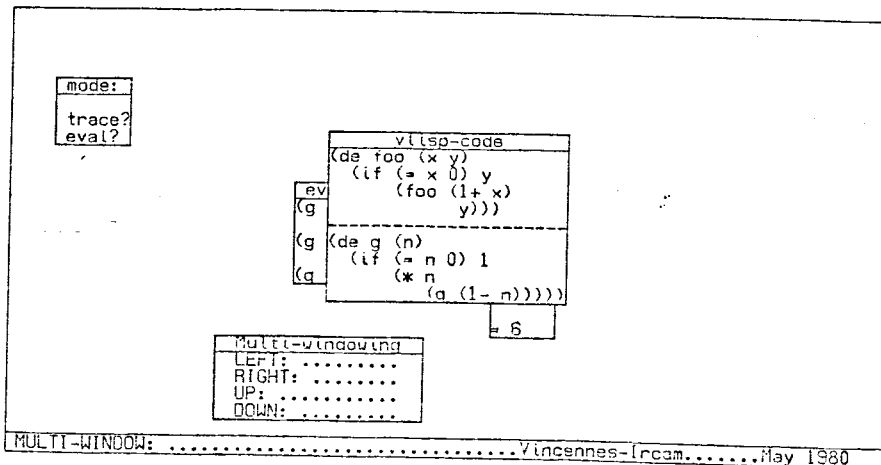
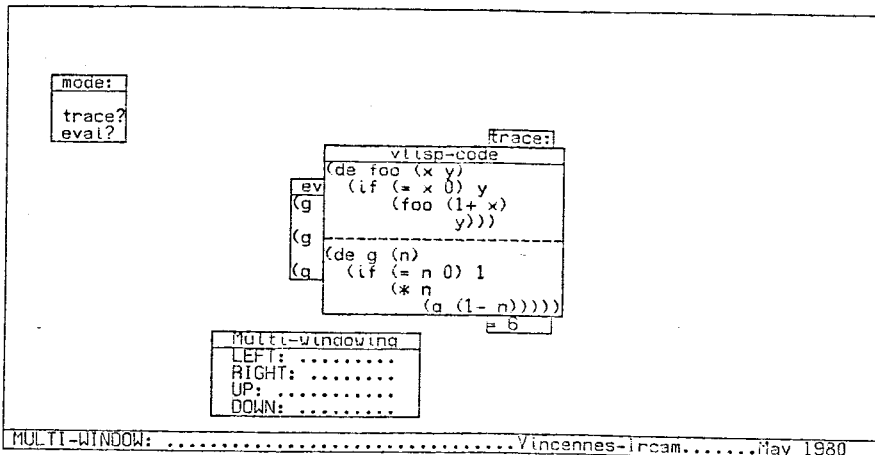
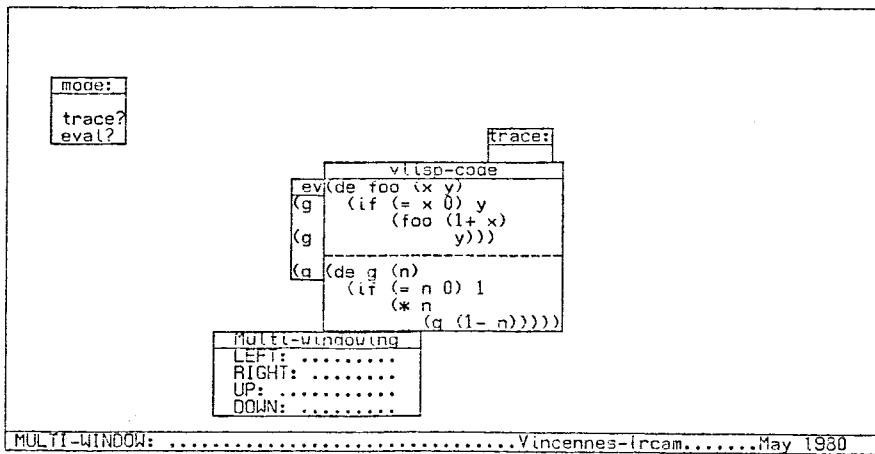
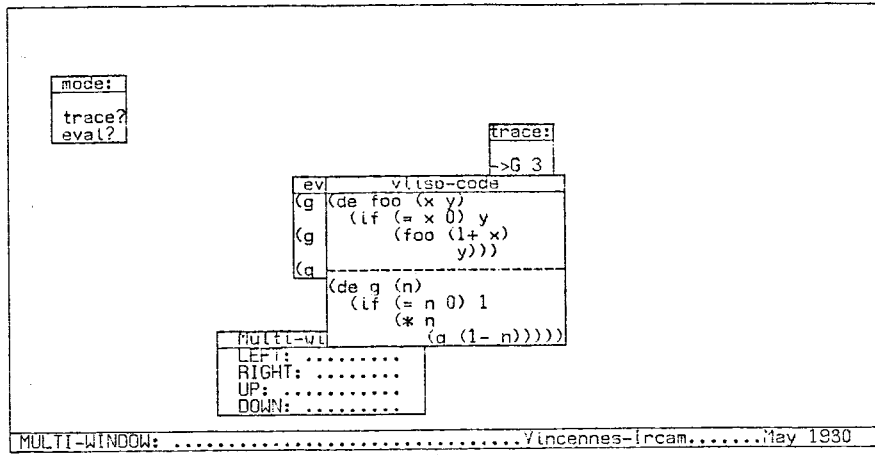
L'adjonction à la vision d'un bord arbitraire souligne le contraste entre la limitation de la perception humaine et l'infinité de Dieu.

1. INTRODUCTION

Nous avons construit, pour le système VLISP, un système de visualisation par fenêtres, déployables et déplaçables dynamiquement, tant par l'utilisateur en mode interactif que par programme, sous la forme d'appels de fonctions VLISP spécialisées.

Ce système de fenêtrage dynamique, WIN, est illustré dans ce qui suit par un bref scénario.





mode:
trace?
eval?

```

v1isp-code
(de foo (x y)
  (if (= x 0) y
      (foo (1+ x)
            y)))
-----
(g 1)
(g 2) (de g (n)
      (if (= n 0) 1
          (* n
             (g (1- n))))))
(g 3)

```

= 6

Multi-windowing
LEFT:
RIGHT:
UP:
DOWN:

MULTI-WINDOW: Vincennes-Ircam..... May 1980

mode:
trace?
eval?

```

v1isp-code
(de foo (x y)
  (if (= x 0) y
      (foo (1+ x)
            y)))
-----
(g 1)
(g 2) (de g (n)
      (if (= n 0) 1
          (* n
             (g (1- n))))))
(g 3)

```

= 6

Multi-windowing
LEFT:
RIGHT:
UP:
DOWN:

MULTI-WINDOW: Vincennes-Ircam..... May 1980

mode:
trace?
eval?

```

v1isp-code
(de foo (x y)
  (if (= x 0) y
      (foo (1+ x)
            y)))
-----
(g 1) =
(g 2) = (de g (n)
      (if (= n 0) 1
          (* n
             (g (1- n))))))
(g 3) =

```

= 6

Multi-windowing
LEFT:
RIGHT:
UP:
DOWN:

MULTI-WINDOW: Vincennes-Ircam..... May 1980

mode:
trace?
eval?

```

v1isp-code
(de foo (x y)
  (if (= x 0) y
      (foo (1+ x)
            y)))
-----
(g 1) = 1
(g 2) = 2
(g 3) = 6

```

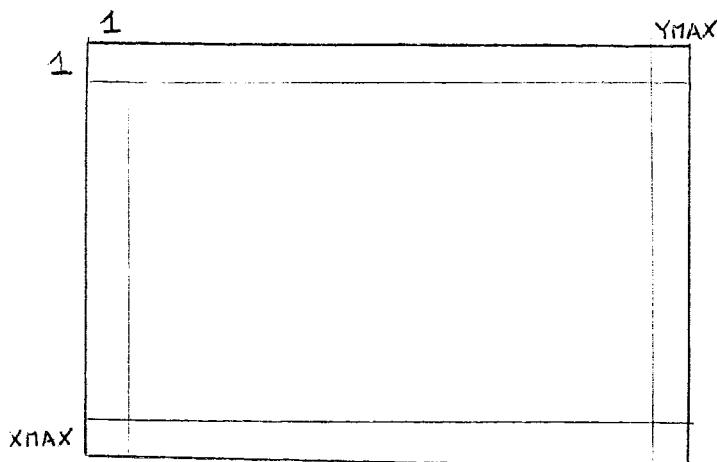
= 6

Multi-windowing
LEFT:
RIGHT:
UP:
DOWN:

MULTI-WINDOW: Vincennes-Ircam..... May 1980

2. DEFINITIONS

L'écran du terminal comporte XMAX lignes et YMAX colonnes.



La position courante du curseur est décrite par le couple de coordonnées: $\langle xc, yc \rangle$, xc désignant le rang de la ligne courante, yc désignant le rang de la colonne courante.

La procédure:

`tyo(caractère)`

imprime le *caractère* sur la position courante du curseur, et avance le curseur d'une colonne. On notera que dans nos algorithmes, l'écran ne sera *pas* représenté globalement en mémoire.

Certains caractères, dits d'échappement, n'apparaîtront pas visiblement, mais induiront une modification globale de l'écran, ainsi:

`tyo(home)`

rangera le curseur dans le coin supérieur gauche de l'écran, de même que:

`tyo(cleos)`

effacera l'écran à partir de la position courante du curseur.

La procédure:

`poscur(xc, yc)`

placera le curseur dans la position correspondant aux valeurs des variables xc , yc .

L'ensemble des fenêtres apparaissant sur l'écran sera organisé comme une liste à chaînage bi-directionnel représentant une queue de priorité, chaque élément de la queue étant constitué des champs suivants:

suiv : adresse de la fenêtre immédiatement plus prioritaire.
prec : adresse de la fenêtre immédiatement moins prioritaire.
numf : numéro de la fenêtre.
xf : x-coordonnée coin supérieur gauche de la fenêtre.
yf : y-coordonnée coin supérieur gauche de la fenêtre.
ndisjpp : adresse de fenêtre plus prioritaire non-disjointe.
xcf : x-coordonnée curseur privé de la fenêtre.
ycf : y-coordonnés curseur privé de la fenêtre.
nl : nombre de lignes de la fenêtre.
nc : nombre de colonnes de la fenêtre.
contf : adresse de la chaîne-contenu de la fenêtre.

3. QUEUE DE PRIORITE

La liste bi-directionnelle des fenêtres comporte une *fenêtre-butée* servant de tête de liste, d'adresse *premfen* qui ne comportera qu'un unique champ, le champ *suivant*. La fenêtre *premfen* sera toujours la *moins prioritaire*.

La dernière fenêtre de la liste sera toujours la fenêtre la plus récemment créée ou déplacée, son adresse sera repérée par la variable *fcour*. Le champ *suivant* de cette fenêtre contiendra le marqueur de fin de liste *eoq*.

Dans ce qui suit on dira qu'une fenêtre récente est *plus prioritaire* qu'une ancienne.

Pour modifier cet ordre si on déplace une fenêtre quelconque d'adresse *adf*, i.e. pour la placer en position de priorité maximum, on utilisera la procédure:

```

Remettre-en-Tête:
  si adf ≠ fcour alors
    suiv(prec(adf)) ← suiv(adf) ;
    prec(suiv(adf)) ← prec(adf) ;
    suiv(fcour) ← adf ; suiv(adf) ← eoq ;
    prec(adf) ← fcour ; fcour ← adf
  fsi

```

3. VISUALISATION DES CONTENUS DE FENETRES

Pour afficher isolément sur l'écran une fenêtre d'adresse *adf*, on utilisera la procédure suivante:

```

Afficher-Fenêtre:
  xc ← xf(adf) ; yc ← yf(adf) ; adr ← contf(adf) ;
  répéter nl(adf) fois
    poscur(xc, yc) ;
    répéter nc(adf) fois
      tyo(m[adr]) ; adr ← adr + 1
    frépéter
  xc ← xc + 1
  frépéter

```

Une première méthode, très grossière, d'affichage de *toutes* les fenêtres de la queue serait (accompagnant chaque déplacement *relatif* du curseur et de la fenêtre qui lui est temporairement attachée: *monter, descendre, avancer, reculer*):

```

Afficher-Toutes-les-Fenêtres:
  xc ← 1 ; yc ← 1 ; tyo(home) ; tyo(cleos) ;
  adf ← suiv(premfen) ;
  tant-que adf ≠ eoq faire
    Afficher-Fenêtre ;
    adf ← suiv (adf)
  ftant-que

```

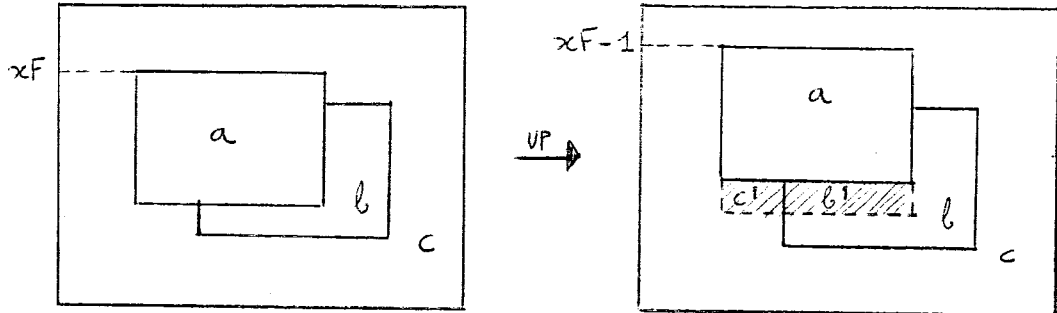
On notera qu'en cas de fenêtres non-disjointes, les caractères des fenêtres les plus récentes masqueront les caractères des fenêtres plus anciennes.

On obtient ainsi un premier procédé de multi-fenêtrage mobile.

4. AFFICHAGE MINIMAL

La méthode Afficher-Toutes-les-Fenêtres provoque des transitions écran-écran trop brutales, demeure très inefficace, en nécessitant la réécriture de *tout* l'écran à chaque transition, se prête enfin difficilement au tracé d'un *cadre* de fenêtre si la définition de l'écran en permet la visualisation.

Ré-examinons à nouveau le problème du déplacement progressif d'une fenêtre. Prenons, à titre d'exemple, une configuration d'écran à 2 fenêtres non-disjointes:



La remontée de **A** doit provoquer l'apparition des zones:

B' appartenant à la fenêtre **B**

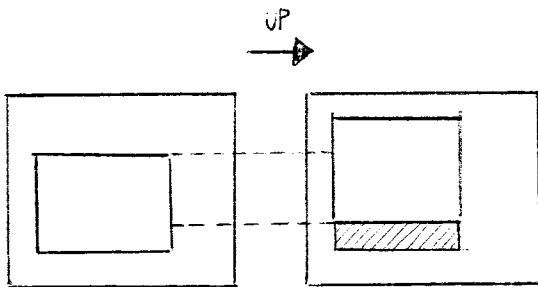
C' appartenant au fond **C**

Ceci suggère alors de traduire l'action demandée par la commande de curseur **haut** sous la forme de la séquence d'actions

1) réécrire **A** une ligne plus haut

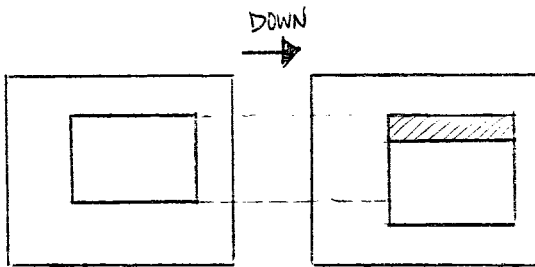
2) pour chacune des positions de la zone pointillée libérée par le déplacement de **A**, rechercher quelle est la fenêtre **F** la plus prioritaire des fenêtres moins prioritaires que **A** à qui appartient la position (elle sera nécessairement *visible*) et écrire le caractère correspondant de **F**. Si la position n'appartient à aucune fenêtre, écrire un espace.

En systématisant cette idée, on obtient alors les 4 cas suivants à considérer:



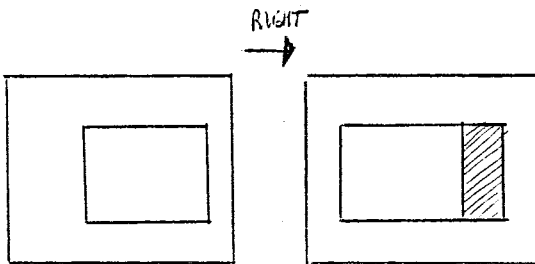
POSITIONS DEMASQUEES

x-coor : $xf + nl - 1$
y-coor : de yf à $yf + nc - 1$



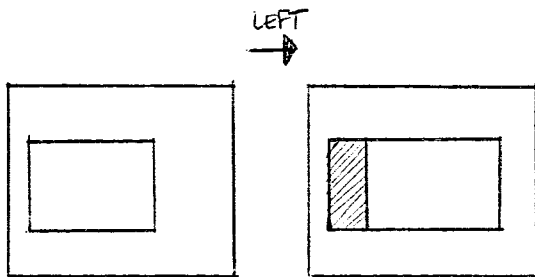
POSITIONS DEMASQUEES

x-coor : xf
y-coor : de yf à $yf + nc - 1$



POSITIONS DEMASQUEES

x-coor : de xf à $xf + nl - 1$
y-coor : yf



POSITIONS DEMASQUEES

xcoor : de xf à $xf + nl - 1$
ycoor : $yf + nc - 1$

xf , yf , nl , nc étant les valeurs des champs correspondants de la fenêtre déplacée.

Nous obtenons, à la suite de cette analyse par cas, les algorithmes suivants:

Rechercher à quelle fenêtre *moins prioritaire* que **FCOUR** appartient la position de coordonnées $\langle xc, yc \rangle$ et ramener dans **adf** l'adresse de la fenêtre qui possède cette position. Si la position visible n'appartient à aucune fenêtre, ramener la valeur de **premfen** (la butée) dans **adf**.

Fenêtre-de-Position:

```
adf ← prec(fcour) ;
tant-que adf ≠ premfen faire
  si xc ∈ [xf(adf), xf(adf) + nl(adf) - 1] alors
  sinssi yc ∈ [yf(adf), yf(adf) + nc(adf) - 1] alors
  sinon exit
  fsi
adf ← prec(adf)
ftant-que
```

On balaye donc la liste des fenêtres, à partir de la précédente de **fcour**, en ordre de priorité décroissante.

Etant donné une fenêtre d'adresse **adf** qui possède la position d'écran de coordonnées $\langle xc, yc \rangle$, ramener dans **caractère** le caractère correspondant de la fenêtre (on transforme les coordonnées-écran absolues $\langle xc, yc \rangle$ en coordonnées relatives à la fenêtre). Si cette position n'appartient à aucune fenêtre, placer le caractère "espace" dans **caractère**.

Caractère-de-Position:

```
si adf = premfen alors caractère ← " "
sinon
  rxc ← xc - xf(adf); ryc ← yc - yf(adf) ;
  caractère ← m[contf(adf) + nc(adf) * rxc + ryc]
fsi
```

Imprimer le caractère correspondant à la position $\langle xc, yc \rangle$ de la plus prioritaire des fenêtres moins prioritaires que **fcour**.

Imprimer-Caractère:

```
Fenêtre-de-Position ;
Caractère-de-Position ;
tyo(caractère) ;
```

On obtient enfin les algorithmes de mouvement ascendant, descendant, gauche et droite de la fenêtre courante, d'adresse *fcour*:

Monter:

```
adf ← fcur ; xf(adf) ← xf(adf) - 1 ;  
Afficher-Fenêtre ;  
xc ← xf(adf) + nl(adf) ;  
yc ← yf(adf) ;  
poscur(xc, yc) ;  
répéter nc(adf) fois  
  Imprimer-Caractère ;  
  yc ← yc + 1  
frépéter
```

Descendre:

```
adf ← fcur ; xf(adf) ← xf(adf) + 1 ;  
Afficher-Fenêtre ;  
xc ← xf(adf) - 1 ;  
yc ← yf(adf) ;  
poscur(xc, yc) ;  
répéter nc(adf) fois  
  Imprimer-Caractère ;  
  yc ← yc + 1  
frépéter
```

Avancer:

```
adf ← fcur ; yf(adf) ← yf(adf) + 1 ;  
Afficher-Fenêtre ;  
yc ← yf(adf) - 1 ;  
xc ← xf(adf) ;  
répéter nl(adf) fois  
  poscur(xc, yc) ;  
  Imprimer-Caractère ;  
  xc ← xc + 1  
frépéter
```

Reculer:

```
adf ← fcur ; yf(adf) ← yf(adf) - 1 ;  
Afficher-Fenêtre ;  
yc ← yf(adf) + nc(adf) ;  
xc ← xf(adf) ;  
répéter nl(adf) fois  
  poscur(xc, yc) ;  
  Imprimer-Caractère ;  
  xc ← xc + 1  
frépéter
```

Sur ces bases, de nombreuses améliorations sont possibles, essentiellement dépendantes de la place mémoire disponible, parmi lesquelles:

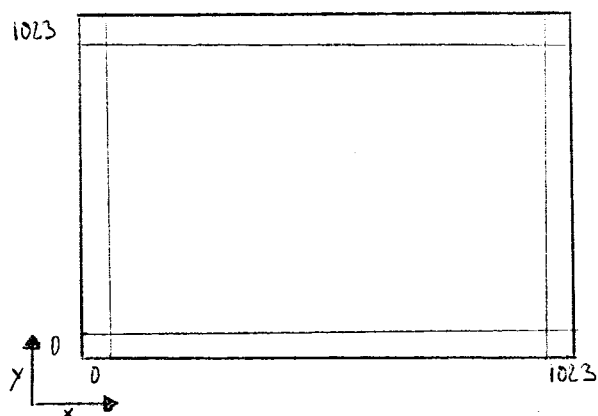
- introduire un ou plusieurs champs supplémentaires dans les fenêtres permettant de trier les fenêtres par ordre de *xf*, *yf*, *xf+nl*, *yf+nc* croissants, pour accélérer la recherche.
- si les caractères sont stockés sur 7 bits, utiliser le 8ème bit comme bit de marquage, placé par la fenêtre à qui *appartient* la position absolue correspondant à ce caractère.
- réorganiser le système sous forme d'un système de *passage de messages*: les couples <fenêtre, position> seront considérés comme des acteurs recevants et s'envoyant mutuellement des messages d'occupation et de libération de positions.

5. BORDS

Le tracé efficace de *bords* de fenêtres, sous forme de vecteurs, qui devront être également déplaçables et masquables, sera relativement plus délicat. Nous examinerons une série de solutions d'efficacité croissante et de simplicité décroissante.

Naturellement ces bords, pour être mis en jeu, nécessitent un écran convenable: de type display-vecteur ou bit-map. Les algorithmes qui suivent concerneront un écran de type display-vecteur, tel que le VT11 de la compagnie DEC.

L'écran est encore une matrice de positions adressables en coordonnées x-y, que nous supposons disposer de 1024 lignes et 1024 colonnes.



Le contrôleur de cet écran doit être conçu comme un processeur séparé à part entière, mettant en jeu les instructions (d-v-instructions) suivantes:

{POINT x y}

Se positionner sur l'écran à la position <x, y>.

{LONGV Δx Δy }

Tracer sur l'écran, à partir de la position courante, un vecteur de longueur définie par l'incrément horizontal Δx et par l'incrément vertical Δy .

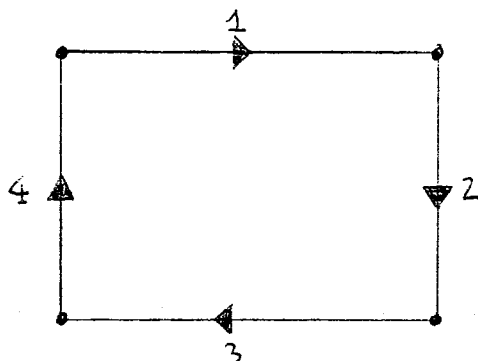
Un bit positionné dans l'incrément Δ décidera si le tracé de vecteur a lieu dans la direction horizontale décroissante: *minusx* ou dans la direction horizontale décroissante: *minusy*.

{DRET}

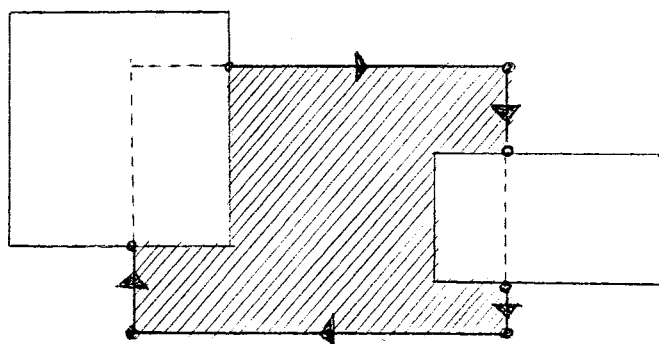
Re-exécuter tout le d-v-programme à partir de sa première instruction. On nommera *rafraîchissement* de l'écran une re-exécution du d-v-programme.

Comment afficher les bords d'une fenêtre de la liste des fenêtres ?

Si la fenêtre était *totale*ment visible, nous aurions à tracer les 4 vecteurs orientés orthogonaux



Si, en revanche, la fenêtre n'était que partiellement visible, ce serait l'effet de fenêtres plus prioritaires partageant des positions de sa surface. Nous pourrions, à titre d'exemple, avoir les 5 vecteurs suivants:



Une première procédure de génération du d-v-programme correspondant sera alors:

Pour chacun des bords 1, 2, 3 et 4, construire à partir de la première position visible $\langle xp, yp \rangle$, un vecteur de longueur égale au nombre de positions visibles à partir de celle-ci dans cette direction, et recommencer à partir de la prochaine position visible du bord.

Comment déterminer, pour une fenêtre *adf*, si une position de coordonnées $\langle xR, yR \rangle$ est ou non visible ? Cette position sera effectivement visible si elle n'appartient pas à une fenêtre plus prioritaire que *adf*.

On a alors l'algorithme:

```
VISIBLE:
  auxADF ← suiv(adf) ;
  tant-que auxADF ≠ 0 faire
    si xR ∈ [xf(auxADF), xf(auxADF) + nl(auxADF) - 1] alors
      sinsi yR ∈ [yf(auxADF), yf(auxADF) + nc(auxADF) - 1] alors
        sinon visible ← faux ; exit
      fsi
    auxADF ← suiv(auxADF)
  ftant-que
  visible ← vrai
```

Comment obtenir, dans x_P et y_P , les coordonnées physiques de l'écran-d-v correspondant aux coordonnées $\langle x_f, y_f \rangle$ de la position supérieure gauche d'une fenêtre d'adresse adf ?

L'algorithme suivant mettra en jeu les constantes suivantes:

homex:	x-coordonnée de la position supérieure gauche du d-v-écran.
homey:	y-coordonnée de la position supérieure gauche du d-v-écran.
car.largueur:	largeur d'un caractère.
car.hauteur:	hauteur d'un caractère.

POSITION-BORD-FENETRE:

```
xp ← homex + car.largueur * (yf(adf) - 1) ;
yp ← homey + car.hauteur * xf(adf)
```

On obtiendrait, à titre d'exemple, pour le bord 1 de la fenêtre adf , l'algorithme suivant, utilisant le booléen **programme-généré**: ce dernier sera *vrai* lorsque la longueur Δv d'un vecteur est en cours de construction pour un segment visible du bord. La procédure **Construire** place ses arguments à la suite des d-v-instructions du d-v-programme en construction.

```
Position-Bord-Fenêtre ;
xR ← xf(adf) ; yR ← yf(adf) ;
programme-généré ← faux ;
répéter nc(adf) fois
  si visible alors
    si programme-généré alors
       $\Delta v \leftarrow \Delta v + \text{car.largueur}$ 
    sinon
      Construire{POINT xp yp} ;
       $\Delta v \leftarrow \text{car.largueur}$  ;
      programme-généré ← vrai
    fsi
  sinsi programme-généré alors
    Construire{LONGV  $\Delta v$  0} ;
    programme-généré ← faux
  fsi
  xP ← xP + car.largueur ;
  yR ← yR + 1
frépéter
si programme-généré alors
  Construire{LONGV  $\Delta v$  0} ;
programme-généré ← faux
fsi
yR ← yR - 1
```

La généralisation de cet algorithme nous donne alors l'algorithme d'affichage de bord d'une fenêtre d'adresse *adf*:

```

BORD-FENETRE:
  Position-Bord-Fenêtre ;
  xR ← xf(adf) ; yR ← yf(adf) ;
  programme-généré ← faux ;
  Bord(nc(adf), car.largeur, car.largeur, xP, Δv, 0, yR, 1) ;
  Bord(nl(adf), car.hauteur, -car.hauteur, yP, 0, Δvouminusy, xR, 1) ;
  Bord(nc(adf), car.largeur, -car.largeur, xP, Δvouminusx, 0, yR, -1) ;
  Bord(nl(adf), car.hauteur, car.hauteur, yP, 0, Δv, xR, -1)
  
```

qui utilisera la macro:

```

BORD(nbr, iΔv, iBordPos, BordPos, xΔv, yΔv, Pos, iPos)
  répéter nbr fois
    si visible alors
      si programme-généré alors
        Δv ← Δv + iΔv
      sinon
        Construire{POINT xP yP} ;
        Δv ← iΔv ; programme-généré ← vrai
      fsi
    sinon programme-généré alors
      Construire{LONGV xΔv yΔv} ;
      programme-généré ← faux
    fsi
    BordPos ← BordPos + iBordPos ;
    Pos ← Pos + iPos
  si programme-généré alors
    Construire{LONGV xΔv yΔv} ;
    programme-généré ← faux
  fsi
  Pos ← Pos - iPos
  
```

L'algorithme d'affichage de bords de *toutes* les fenêtres de la liste des fenêtres devient alors:

```

"stopper le d-v-programme d'affichage de bords" ;
adf ← fcour ;
tant-que adf ≠ premfen faire
  Bord-Fenêtre ;
  adf ← prec(adf)
ftant-que
  Construire{DRET} ;
"redémarrer le d-v-programme d'affichage de bords"
  
```

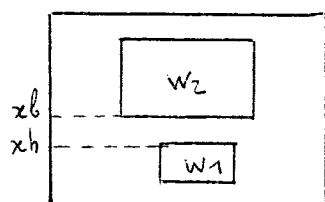
Cet algorithme, quoique correct, demeure relativement inefficace: une première amélioration peut consister à afficher, au fur et à mesure de leur construction, les {POINT x y} et les {LONGV Δx Δy}, afin d'obtenir un réaffichage plus continu.

Par ailleurs, il n'est pas nécessaire de recalculer, pour *toutes* les positions de bord d'une fenêtre, si elles sont visibles dans cette fenêtre ou masquées par une fenêtre plus prioritaire.

Une première amélioration consiste à déterminer en temps constant si la fenêtre w1 dont on veut afficher le bord et une autre fenêtre w2 plus prioritaire sont ou non disjointes. Si elles sont effectivement disjointes, cette fenêtre disjointe prioritaire pourra être *ignorée*, pour toutes les positions de bord de la fenêtre candidate à l'affichage, dans le test de visibilité. Ce pré-test améliorera alors considérablement la vitesse de fabrication du d-v-programme d'affichage.

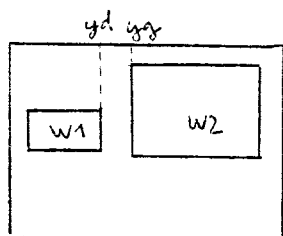
On peut distinguer, en première approche, 4 cas de conditions suffisantes de disjonction.

CAS 1



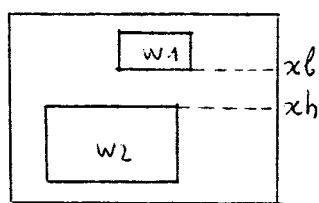
$$xh(w1) > xb(w2)$$

CAS 2



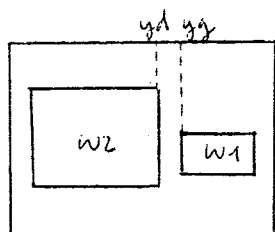
$$yd(w1) < yg(w2)$$

CAS 3



$$xb(w1) < xh(w2)$$

CAS 4



$$yg(w2) > yd(w1)$$

avec $xh = xf$, $xb = xf + nl - 1$
 $yg = yf$, $yd = yf + nc - 1$

Le pré-test de visibilité consistera à constituer, à partir de la liste des fenêtres plus prioritaires, la sous-liste des fenêtres *non-disjointes*. L'algorithme de visibilité d'une position ne testera le masquage possible de cette position *que* pour ces fenêtres non-disjointes.

Dans le cas où une fenêtre d'adresse ADF , de priorité quelconque, est disjointe de toutes les autres (liste des fenêtres non-disjointes vide), ce qui s'applique également à la fenêtre courante $FCOUR$, l'algorithme direct d'affichage total des bords sera mis en jeu dans la fabrication du d-v-programme:

```
Construire{POINT xp yp} ;
xD ← car.largeur * nc(adf) ; yD ← car.hauteur * nl(adf) ;
Construire{LONGV xD 0} ;
Construire{LONGV 0 yDcrminusy} ;
Construire{LONGV xDcrminux 0} ;
Construire{LONGV 0 yD}
```

On notera qu'après l'exécution d'un LONGV, la position courante sur le d-v-écran est celle de l'extrémité *finale* du vecteur affiché par LONGV.

6. REFERENCES

- [1] CHAILLOUX J. *Le modèle VLISP: description, implémentation et évaluation*, (Thèse de 3e cycle), Rapport LITP no 80-20, Avril 1980
- [2] DEC *VT11 graphic display processor*, EK-VT11-TM-001, Digital Equipment Corporation, Maynard Mass., September 1974
- [3] GOLDBERG A., KAY A. (eds) *SMALLTALK-72 Instruction Manual*, SSL76-6, Xerox PARC, Palo Alto, Ca., March 1976
- [4] GREUSSAY P. *VLISP-11 Reference Manual*, Université Paris-8-Vincennes, (à paraître), 1980
- [5] NEWMAN W.M., SPROULL R.F. *Principles of Interactive Computer Graphics*, 2d edition, Mc Graw Hill, 1979
- [6] RIEGER C. *The LISP Window Slave*, Unpublished Manuscript, M.I.T. Artificial Intelligence Laboratory, November 1975
- [7] SPROULL R.F. *The design of gray-scale graphics software*, Proc. of the IEEE Conf. on Computer Graphics, Pattern Recognition and Data Structure, May 1975, 18-20
- [8] SPROULL R.F. *Raster Graphics for Interactive Programming Environments*, CSL-79-6, Xerox PARC, Palo Alto, Ca., June 1979
- [9] TEITELMAN W. *A Display-Oriented Programmer's Assistant*, Proc. 5th Int. Joint Conf. Artificial Intelligence, August 1977, 905-915
- [10] WARREN S.K., ABBE D. *Rosetta Smalltalk: A conversational, Extensible Microcomputer Language*, SIGSMALL Newsletter, Vol 5, no 2, April 1979, 13-22
- [11] WEINREB D., MOON D. *Lisp Machine Manual*, 2nd preliminary version, M.I.T. AI Lab, January 1979