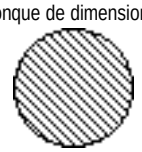
Zoom x2 *Hp 48 & 49*

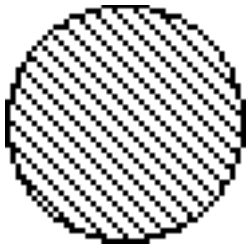
Petit grob deviendra plus grand... un programme ultra rapide qui agrandit par deux les dimensions des grobs.

Pour information, nous traiterons cet article pour une HP 48 série G. La conversion HP 49G sera proposée dans le dernier paragraphe. Les seules différences sont dans les parties Rpl-utilisateur et externals.

Comme d'accoutumée, nous allons commencer ce type d'article par un peu de théorie pour comprendre la suite des évènements. Tout d'abord, qu'est-ce qu'un zoom x2 ? C'est l'opération qui permet de modifier une image en l'agrandissant de manière à multiplier par deux sa hauteur et sa largeur (on peut aussi dire que la surface de l'image se retrouve multipliée par quatre). C'est à dire que si un grob (ou objet graphique, qui est une image) est de la forme suivante : `Graphic 45 x 126`, alors son zoom x2 donnera ce résultat : `Graphic 90 x 252`.
Concrètement, une image quelconque de dimensions `45x45` que voici :



formera une autre image aux dimensions `90x90` comme ceci :



Pour effectuer la transformation, nous allons tout d'abord créer un nouveau grob vierge qui aura les dimensions finales souhaitées (soit le double du grob d'origine), puis pour chaque pixel allumé du grob de départ, on allume quatre pixel dans le grob d'arrivée, un peu comme ceci :

Le programme que nous allons construire va fonctionner sur ce principe et pourra être appliqué pour des grobs de taille quelconque.

• La version Rpl •

Voyons comment réaliser ce programme en Rpl-utilisateur de manière simple. On crée d'abord une image vierge qui possède des dimensions deux fois plus grandes que l'image d'origine.Le grob de départ sera ensuite stocké dans l'environnement PICTURE, car c'est à partir de cet environnement qu'on prélèvera la valeur de chaque pixel. Si un pixel est allumé alors on effectue les modifications sur le grob d'arrivée sinon on passe au pixel suivant pour lui infliger la même opération. Le programme ne demande qu'une simple boucle et un simple test :

'Z.RPL' *(153,5 octets - CRC # 50DCh)*

```
« DUP PICT STO SIZE DUP2 SL SWAP SL SWAP BLANK
  0 ROT 1 - B¨R FOR L
    0 3 PICK 1 - B¨R FOR C
      IF C L 2 ¨LIST R¨B DUP PIX?
      THEN SL # 2h DUP BLANK NEG REPL
      ELSE DROP
      END
    NEXT
  NEXT SWAP DROP ERASE »
```

Un petit test de chronométrage sur le cercle `45x45` nous donne cind minutes et dix-neuf secondes. On imagine fort bien que pour des grobs de grande taille, ce programme est à proscrire ! L'assembleur s'impose...
Pour la Hp 49G, les temps d'exécution d'un programme très similaire est presque trois fois plus rapide ! (voir le programme en fin d'article).

• Bien plus rapide ! •

Avant d'attaquer l'assembleur, préparons un peu le terrain en faisant réaliser les tâches basiques par un soupçon d'externals. Cette partie est juste consacrée à créer le grob vierge qui possèdera des dimensions doubles par rapport au grob d'origine. Et puis on fera un test pour les grobs de dimensions égales à zéro, dans ce cas on n'exécutera pas le programme.

```
{ # 18AB2h          Teste la présence d'un objet sur le niveau 1 de la pile.
  # 18FB2           Teste le type de l'objet présent. Si c'est un grob,
  12                on continue.
  { # 3188h         Instruction DUP.
    # 50578h        Instruction SIZE pour les grobs.
    # 3EC2h         Multiplication de 2 system binary.
    # 3CC7h         Teste si le résultat est différent de zéro.
    # 619BCh        Instruction IFT.
    { # 3188h       Instruction DUP.
      # 50578h      Instruction SIZE pour les grobs.
      # 3E6Fh       Multiplication par 2.
      # 3223h       Instruction SWAP.
      # 3E6Fh       Multiplication par 2.
      # 3223h       Instruction SWAP.
      # 1158Fh      Instruction BLANK.
      # 3223h       Instruction SWAP.
      Code          Programme principal.
      # 3244h       Instruction DROP.
} } }
```

Le résultat de ce morceau d'externals a pour effet de placer le grob vierge au niveau 2 de la pile, et le grob de départ au niveau 1 de la pile. Après zoom (c'est le programme `Code` qui s'en charge), il reste plus qu'à effacer le grob d'origine.

Passons maintenant au gros du programme avec la source assembleur qui est rigoureusement identique pour la Hp 48 que pour la Hp 49.

```
"GOSBVL 0679B     On sauvegarde les registres et les pointeurs.
 C=0 A            On initialise C à 20 (en décimal) pour ensuite sauter le
 LC 14            prologue, la longueur et la taille du grob.
 A=DAT1 A         On charge l'adresse du grob
 D0=A             qu'on pointe.
 A=A+C A          On se place sur les premiers pixels du grob,
 B=A A            et on sauve l'adresse dans B.
 D1=D1+ 5         On pointe l'adresse de début du grob vierge,
 A=DAT1 A         qu'on charge.
 A=A+C A          On se déplace sur les premiers pixels du grob,
 D1=A             et on pointe.
 D0=D0+ 15        On avance sur la taille du grob d'origine.
 ST=0 5           Initialisation du flag 5 à 0.
 A=DAT0 A         On prend la largeur d'une ligne.
 SB=0
 ASRB A           On divise la largeur par quatre pour avoir le nombre
 ASRB A           de quartets.
 ?SB=0            S'il n'y pas de reste,
 GOYES ZERO       on passe,
 SB=0
 A=A+1 A          sinon on ajoute 1 quartet.
*ZERO
 R4=A A           On sauve la largeur en quartets.
 ASRB A           On la divise encore par 2 pour avoir la largeur en
                  octets d'une ligne.
 ?SB=0            S'il n'y a pas de reste,
 GOYES OK         on saute,
 ST=1 5           sinon on sauve le résultat.
*OK
 D0=D0- 5         On recule notre pointeur pour charger le nombre
 C=DAT0 A         de lignes,
 C=C-1 A
 D=C A            qu'on sauve dans D.
 C=B A            On récupère l'adresse du grob d'origine,
 D0=C             et on pointe.
*BOUCLE           Et on commence la grande boucle.
 C=R4 A           On récupère la largeur en quartets,
 C=C-1 A          qu'on sauve dans B pour faire
 B=C A            une boucle sur les lignes.
*LIGNE
 A=DAT0 B         On récupère un quartet qu'on va transférer dans le
                  grob vierge en faisant une boucle sur les 4 bits du
                  quartet.
 C=0 B            On initialise C qui deviendra le quartet zoomé en octet
                  dans le grob vierge.
 P= 15            On initialise la mini boucle.
 LC 3
 P= 0
*QUATRE
 C=C+C B          On décale de 2 quartets vers la gauche.
 C=C+C B
 ?ABIT=0 3        Si le dernier bit du quartet est égale à zéro,
 GOYES SUITE      on saute,
 CBIT=1 0         sinon on met les 2 premiers bits de C à 1.
 CBIT=1 1
*SUITE
 A=A+A A          On passe au bit suivant,
 C=C-1 S          et on décrémente la boucle,
 GONC QUATRE      jusqu'à ce que les 4 bits soient testés.
 DAT1=C B         On écrit l'octet dans le grob vierge.
 R3=C B           On sauve se résultat pour inscrire ensuite la deuxième
                  épaisseur.
 A=R4 A           Dans le grob vierge, on se déplace juste sur l'octet
 A=A+A A          directement en dessous...
 CD1EX
 A=A+C A
 C=R3 B
 D1=A
 DAT1=C B         ... pour écrire la deuxième épaisseur.
 A=R4 A           Puis on revient là où on en était resté.
 A=A+A A
 CD1EX
 C=C-A A
 D1=C
 D0=D0+ 1         On passe au quartet suivant pour le grob de départ,
 D1=D1+ 2         et à l'octet suivant pour le grob vierge.
 B=B-1 A          On décrémente,
 GONC LIGNE       et on boucle.
 CD1EX            On récupère l'adresse de l'endroit où l'on pointe dans
 A=R4 A           le grob vierge pour pouvoir sauter une ligne et se
 A=A+A A          préparer à la suivante.
 A=A+C A
 D1=A
 ?ST=0 5          Si le nombre de quartet est pair,
 GOYES FIXE       on saute,
 D0=D0+ 1         sinon on avance d'un quartet, pour se placer sur la
                  ligne suivante du grob d'origine.
*FIXE
 D=D-1 A          On décrémente la boucle
 GOC FIN          jusqu'à ce qu'on passe en dessous de zéro.
 GOTO BOUCLE
*FIN
 GOVLNG 05143     On récupère les pointeurs et registres avant de
@"                retourner au RPL.
```

Quelques commentaires supplémentaires sont néanmoins nécessaires pour mieux comprendre les ficèles de cette source. Le programme va balayer chaque ligne du grob de départ pour en inscrire deux dans le grob final, d'où la première boucle. Ensuite, comme on travaille sur chaque quartet du grob, il y a une boucle imbriquée pour permet ne pas en oublier. Et enfin, comme dans un quartet il y a quatre bits

(ah bon ???), on réalise une mini boucle sur chacun des quartets du grob. Pourquoi calcule-t-on le nombre d'octets ? Tout simplement parce que la largeur d'un grob est défini en octets, contrairement au reste de la machine qui fonctionne uniquement avec des quartets. Si l'on prend un grob d'une largeur de 131 pixels, ça équivaut à 17 octets soient 34 quartets, au lieu de 33 (car CEIL(131/4)=33). Le problème est que le programme du zoom travaille sur les quartets, donc une petite astuce est utilisée vers la fin de la source pour corriger ce petit détail.
L'histoire de la copie des bits du grob de départ vers le grob vierge est résolue par une simple boucle. Le principe est de copier le dernier de chaque quartet du grob de départ à chaque tour de boucle vers le grob d'arrivée (en deux bits puisque c'est un zoom), en n'oubliant pas d'opérer un décalage : ce qui permet de balayer tous les bits du quartet à copier.

Pour tous ceux qui préfèrent saisir la source sous sa forme hexadécimale, voici la chaîne de caractères qu'il faut entrer sans espace ni saut (385 octets - CRC # 7916h) et compiler avec **GASS48** juste après :

```
D9D20 2BA81 2BF81 76040 D9D20 88130 87505 2CE30
7CC30 CB916 D9D20 88130 87505 F6E30 32230 F6E30
32230 F8511 32230 CCD20 60100 8FB97 60D23 14114
3130C AD817 4143C A1311 6E845 14282 2819F 0819F
08327 0822E 481AF 04819 F0832 50855 18414 6CED7
D9134 81AF1 CCED5 14AAE 22F30 320A6 6A668 0863C
08089 08089 1C4A4 E53E1 4D81A 60B81 AF14C 4137C
A81A6 1B131 14D81 AF14C 4137E 21351 60171 CD599
13781 AF14C 4CA13 18655 0160C F4606 F6F8D 34150
44230 B2130 B2130 B2130
```

'GASS48' *(86,5 octets - CRC # 1DB3h)*

```
« "GROB 8 " OVER SIZE 2 / + " " + SWAP + OBJ¨
  # 4017h SYSEVAL # 56B6h SYSEVAL DROP NEWOB »
```

• Et un Zoom x4 ... •

Sur un principe un peu plus simple, on peut réaliser un zoom x4 qui remplit chaque quartet du grob final si chaque bit du grob de départ est égal à 1. Mais à défaut d'avoir cette source, on peut écrire un programme de ce type si le zoom x2 s'appelle **ZOOM2** : `« ZOOM2 ZOOM2 »`

• Tout pour la Hp 49 •

Comme promis voici tous les programmes convertis pour la Hp 49G, à commencer par le premier en Rpl-utilisateur :

'Z.RPL' *(162 octets - CRC # 3A8Fh)*

```
« DUP PICT STO SIZE DUP2 SL SWAP SL SWAP BLANK
  # 0h ROT 1. - FOR L
    # 0h PICK3 1. - FOR C
      IF C L 2. ¨LIST DUP PIX?
      THEN SL # 2h DUP BLANK NEG REPL
      ELSE DROP END
    NEXT
  NEXT NIP ERASE »
```

L'avantage par rapport à la Hp 48 est qu'ici on peut utiliser des entiers binaires pour les bornes des boucles, ce qui simplifie beaucoup le programme. Par contre, la taille du programme est plus imposante ?!?

Dans la deuxième partie, les adresses externals ont bien évidemment bougé. Voici la liste sans commentaire, sachant qu'ils sont identiques :

```
{ # 26297h # 26328h 12 { # 3188h # 26085h
  # 3EC2h # 3CC7h # 34A22h { # 3188h # 26085h
  # 3E6Fh # 3223h # 3E6Fh # 3223h # 260B2h
  # 3223h Code # 3244h } } }
```

Voici la chaîne de caractères qu'il faut entrer sans espace ni saut (385 octets - CRC # E385h) et compiler avec `256 ATTACH H->` juste après :

```
D9D20 79262 82362 F7133 D9D20 88130 58062 2CE30
7CC30 22A43 D9D20 88130 58062 F6E30 32230 F6E30
32230 2B062 32230 CCD20 40100 8FB97 60D23 14114
3130C AD817 4143C A1311 6E845 14282 2819F 0819F
08327 0822E 481AF 04819 F0832 50855 18414 6CED7
D9134 81AF1 CCED5 14AAE 22F30 320A6 6A668 0863C
08089 08089 1C4A4 E53E1 4D81A 60B81 AF14C 4137C
A81A6 1B131 14D81 AF14C 4137E 21351 60171 CD599
13781 AF14C 4CA13 18655 0160C F4606 F6F8D 34150
44230 B2130 B2130 B2130
```

• Les performances •

Avant d'entâmer les tests de rapidité, il faut savoir qu'on ne peut pas faire de zoom x2 sur des grobs de trop grandes dimensions, sous peine de saturer la mémoire (en particulier sur une Hp 48G). Pour exemple, un grob de taille 250x250 (soit une taille de 8 Ko en mémoire) deviendra un grob de taille 500x500, qui occupera une place de 32 Ko, soit quatre fois plus volumineux en taille que le Grob d'origine !
Par ailleurs on ne s'attardera que sur les temps réalisés par les versions assembleur des deux machines.
Le tableau présenté ci-après donne la taille d'origine et la taille finale accompagnées de la place mémoire occupée ainsi que du temps de la transformation.

Grob de départ	Grob d'arrivée	Temps d'exécution Hp 48 / Hp 49
45x45 (280 octets)	90x90 (1090 octets)	0"13 / 0"13
131x64 (1098 octets)	262x128 (4234 octets)	0"'44 / 0"44
250x250 (8010 octets)	500x500 (31510 octets)	3"08 / 3"01

Vous disposez désormais d'une fonction puissante et rapide permettant des zooms x2, mais avez-vous pensé comment réaliser l'opération inverse ? Le principe de réduction d'une image en divisant ses dimensions par deux est bien plus complexe à programmer. Alors à vous de jouer...

Philippe Pamart
[phpamart@nordnet.fr]