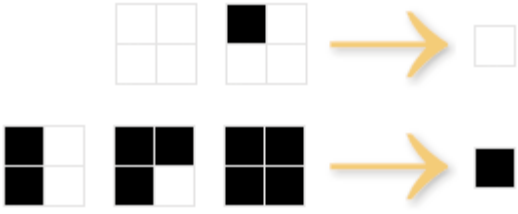
Zoom /2 *Hp 48 & 49*

Comme promis, voici l'opération inverse du programme du mois précédent : un algorithme de zoom à 50%.

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 externals. Et pour notre application, nous ne nous attarderons pas sur une version Rpl-utilisateur, mais nous nous intéresserons plutôt directement à une version externals et assembleur.

• Un peu de théorie •

Qui dit réduire de 50% les dimensions d'une image, dit aussi diviser sa surface par quatre. C'est à dire qu'un grob (grob = graphic object ou plutôt objet graphique in french) de dimensions **A x B** sera réduit en un grob de taille **A/2 x B/2**. Cela est réalisable si bien évidemment les dimensions du grob d'origine sont paires. Si l'une des dimensions (hauteur ou largeur) est impaire, on choisira l'arrondi inférieur de la division par deux pour une raison logique : si on choisit l'arrondi supérieur, cela provoquerait l'ajout d'une ligne ou d'une colonne vierge. Le calcul des dimensions du grob transformé sera basé sur cette formule : **FLOOR(A/2) x FLOOR(B/2)**. Pour exemple, un grob de **131 x 64** verra ses dimensions se réduire à **65 x 32**. Cette partie sera traitée en Rpl-système.
Passons maintenant à la réduction proprement dite en analysant l'algorithme utilisé. Comme on réduit la surface d'un grob par quatre, cela veut dire que quatre pixels du grob d'origine seront réduits à un seul pixel dans le grob transformé. Le problème est de définir quel va être l'état du pixel final en fonction des blocs de quatre pixels qu'on prélèvera dans le grob d'origine. Quand on passe les solutions possibles en revue, on arrive à cinq choix possibles en fonction du nombre de pixels différents dans chaque bloc : de 0 à 4 pixels. Si on a 0 ou 1 pixel au départ, le pixel final sera blanc, et si on a 3 ou 4 pixels à l'origine, alors le pixel final sera noir. Le cas litigieux survient quand on dispose de 2 pixels dans un bloc puisqu'ici nous avons deux possibilités : soit on choisit blanc, soit on choisit noir ! Que faire ? On pourrait regarder le nombre de pixels placés autour du bloc de quatre dans le grob d'origine pour avoir une idée plus précise, mais le nombre de pixels reste toujours pair donc il se pourrait qu'on obtienne aussi un cas problématique. Mais nous ne nous lancerons pas dans cette aventure... La solution est qu'il faut trancher ! Après de multiples essais, il s'avère qu'il faut noircir ce pixel, car on obtient en général une réduction bien meilleure sur la majorité des cas testés. Et cette phase est réalisée en assembleur pour des raisons de rapidité évidentes. Pour illustrer un peu cette théorie, schématisons les transformations possibles avec à gauche les blocs du grob de départ, et à droite l'état du pixel final du grob réduit :



• la préparation •

Le principe de la structure "externals" calque beaucoup le programme du zoom présenté dans le précédent numéro de Team Palmtops. Pour résumer, c'est la partie qui s'occupe de créer le grob réduit sur le niveau 1 de la pile, ceci avant exécution du programme en langage machine. Voyons tout cela en détails :

```
{ # 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.
    # 3E0Eh         On soustrait 1 à la hauteur.
    # 3223h         Instruction SWAP.
    # 3E0Eh         On soustrait 1 à la largeur.
    # 3EC2h         Et on multiplie le tout.
    # 3CC7h         Teste si le résultat est différent de zéro.
    # 32C2h         Instruction OVER.
    # 50578h        Instruction SIZE pour les grobs
    # 3EC2h         Et on multiplie les coordonnées.
    # 3CC7h         Teste si le résultat est différent de zéro.
    # 3B46h         Opérateur logique AND.
    # 61AD8h        Instruction IFTE.
    { # 3188h       Instruction DUP.
      # 50578h      Instruction SIZE pour les grobs.
      # 3E8Eh       Divise par 2.
      # 3223h       Instruction SWAP.
      # 3E8Eh       Divise par 2.
      # 3223h       Instruction SWAP.
      # 1158Fh      Instruction BLANK.
      # 3223h       Instruction SWAP.
      Code          Programme principal.
      # 3244h } Instruction DROP.
    "Pas réductible" } }
```

Pour commenter un peu, on exécute la réduction si l'objet posé sur le niveau 1 de la pile est un grob. Si c'est le cas, on teste si le grob à réduire possède une taille strictement supérieure à 1 en largeur et à 1 en hauteur pour éviter d'obtenir des grobs de taille 0 sur l'une ou l'autre dimension. Si le grob à transformer n'est pas réductible, un message apparaît sous la forme d'une chaîne de caractères sur le niveau 1 de la pile. Quand toutes ces vérifications sont effectuées, on crée un grob vierge réduit avant de lancer le programme principal réalisé en assembleur.

• le gros morceau ! •

La source du zoom 50% est bien plus complexe que pour le zoom 200%, et une fois de plus cette source est identique sur Hp 48 comme sur Hp 49. Cette fois, les commentaires seront placés en début de chaque partie afin de rendre plus compréhensible cette source. Une grosse partie d'initialisation consiste à calculer toutes les dimensions sur les deux grobs qui servira pour les différentes boucles. Et ensuite on réalise réellement l'opération de réduction telle qu'elle est présentée en théorie. Mais regardez plutôt :

On sauve registres et pointeurs, on récupère l'adresse du grob vierge, on le pointe, et on avance sur la largeur, qu'on charge.

```
"GOSBVL 0679B
A=DAT1 A
D0=A
D0=D0+ 15
A=DAT0 A
```

On réalise une petite opération qui calcule le nombre de quartets que contient le grob d'origine, et on stocke le tout dans R2 champ A.

```
SB=0
A=A+A A
ASR A
?SB=0
GOYES OK1
A=A+1 A
*OK1
A=A+A A
R2=A A
```

On récupère l'adresse du début des quartets correspondant aux premiers pixels du grob de départ, on la stocke dans R0 champ A.

```
D0=D0+ 5
CD0EX
R0=C A
```

On se place sur le grob vierge réduit, on charge l'adresse de l'objet, on charge la hauteur pour avoir le nombre de lignes qu'on stocke dans B champ A qui est en fait le compteur de lignes.

```
D1=D1+ 5
A=DAT1 A
D0=A
D1=C
D0=D0+ 10
C=DAT0 A
B=C A
B=B-1 A
```

On se déplace sur la largeur pour calculer le nombre de quartets selon le même principe que pour le grob de départ qu'on stocke dans R4 champ A.

```
D0=D0+ 5
C=DAT0 A
C=C+C A
C=C+C A
SB=0
CSR A
?SB=0
GOYES OK2
C=C+1 A
*OK2
C=C-1 A
R4=C A
```

On calcule maintenant le nombre de quartets effectifs que comprend une ligne pour éviter les problèmes de décalage et d'écrasement. On stocke cette valeur dans R3 champ A.

```
C=C+1 A
SB=0
CSRB A
?SB=0
GOYES OK3
C=C+1 A
*OK3
C=C+C A
R3=C A
```

On récupère enfin l'adresse du début des quartets correspondant aux premiers pixels du grob réduit, on la stocke dans R1 champ A.

```
D0=D0+ 5
AD0EX
R1=A A
```

Pour la suite, D0 pointe le grob réduit et D1 le grob de départ (chose déjà faite plus haut).

```
D0=A
```

On détermine la limite du nombre de pixels noirs de chaque bloc du grob de départ. Si le nombre de pixels est inférieur à 2 (soit 1 ou 0) alors le pixel est blanc, sinon il est noir. On réalise cette opération pour tous les pixels de chaque quartet du grob réduit.

```
P= 15
LC 2
P= 0
```

La boucle 1 correspond à la boucle sur les lignes, et la boucle 2 est utilisée sur les quartets de chaque ligne.

```
*BOUCLE1
C=R4 A
D=C A
*BOUCLE2
```

Le premier pixel d'un quartet du grob réduit est testé.

```
GOSUB TEST1
?C>D S
GOYES FINPIX1
A=DAT0 B
ABIT=1 0
DAT0=A B
*FINPIX1
```

Le second pixel du même quartet est aussi testé.

```
GOSUB TEST2
?C>D S
GOYES FINPIX2
A=DAT0 B
ABIT=1 1
DAT0=A B
*FINPIX2
D1=D1+ 1
```

Le troisième pixel de ce quartet est encore testé.

```
GOSUB TEST1
?C>D S
GOYES FINPIX3
A=DAT0 B
ABIT=1 2
DAT0=A B
*FINPIX3
```

Et le dernier pixel du quartet est testé.

```
GOSUB TEST2
?C>D S
GOYES FINPIX4
A=DAT0 B
ABIT=1 3
DAT0=A B
*FINPIX4
```

On opère tous les décalages avant de passer au quartet suivant.

```
D1=D1+ 1
D0=D0+ 1
D=D-1 A
GONC BOUCLE2
```

On fait quelques calculs qui vont permettre de passer à la ligne suivante et reprendre le même processus.

```
A=R0 A
C=R2 A
A=A+C A
A=A+C A
R0=A A
D1=A
A=R1 A
C=R3 A
A=A+C A
R1=A A
D0=A
B=B-1 A
GOC FIN
GOTO BOUCLE1
```

Pour finir, on retourne au Rpl en ayant préalablement récupéré registres et pointeurs.

```
*FIN
GOVLNG 05143
```

Voici les deux routines qui s'occupent de tester les blocs de quatre pixels dans le grob de départ. Le premier test s'occupe des quatre pixels de gauche (sur deux lignes) et le second des quatre pixels de droite.

```
*TEST1
D=0 S
GOSUB PART1
C=R2 A
AD1EX
A=A+C A
D1=A
GOSUB PART1
AD1EX
A=A-C A
D1=A
RTN
*TEST2
D=0 S
GOSUB PART2
C=R2 A
AD1EX
A=A+C A
D1=A
GOSUB PART2
AD1EX
A=A-C A
D1=A
RTN
```

Les deux routines qui suivent sont utilisées par les deux précédentes et s'occupent du calcul des bits allumés proprement dits.

```
*PART1
A=DAT1 B
```

```
?ABIT=0 0
GOYES PIX1
D=D+1 S
*PIX1
?ABIT=0 1
RTNYES
D=D+1 S
RTN
*PART2
A=DAT1 B
```

```
?ABIT=0 2
GOYES PIX2
D=D+1 S
*PIX2
?ABIT=0 3
RTNYES
D=D+1 S
RTN

@"
```

Pour tous ceux qui préfèrent saisir la source sous sa forme hexadécimale (externals compris), voici la chaîne de caractères qu'il faut entrer sans espace ni saut (646 octets - CRC # 4525h) et compiler avec **GASS48** juste après :

```
D9D20 2BA81 2BF81 76040 D9D20 88130 87505 E0E30
32230 E0E30 2CE30 7CC30 2C230 87505 2CE30 7CC30
64B30 8DA16 D9D20 88130 87505 E8E30 32230 E8E30
32230 F8511 32230 CCD20 BB100 8FB97 60143 13016
E1428 22C4F 48324 0E4C4 81AF0 21641 3681A F0817
41431 30135 16914 6D5CD 16414 6C6C6 822F6 83240
E6CE8 1AF0C E6822 819F2 83240 E6C68 1AF0B 16413
281AF 01130 2F302 2081A F1CD7 7A909 C7D01 4A808
50148 79A09 C7D01 4A808 51148 1707F 609C7 D014A
80852 1487E 709C7 D014A 80853 14817 0160C F54A8
1AF10 81AF1 ACACA 81AF0 01318 1AF11 81AF1 BCA81
AF011 30CD4 60646 F8D34 150AC 37F30 81AF1 A133C
A1317 D2013 3EA13 101AC 37530 81AF1 A133C A1317
32013 3EA13 10114 B8086 050B4 78086 100B4 70114
B8086 250B4 78086 300B4 70144 230B2 130C2 A2012
00005 16370 2279E 46573 64796 26C65 6B213 0B213
0
```

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

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

• le même programme pour Hp 49 •

Commençons par la partie en externals où seules les points d'entrée changent. Chaque point d'entrée correspond exactement à un point d'entrée de la Hp 48. Donc aucun commentaire n'est proposé :

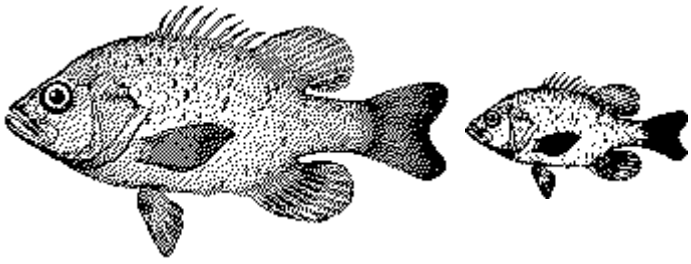
```
{ # 26297h # 26328h # 3317Fh
  { # 3188h # 26085h # 3E0Eh # 3223h # 3E0Eh
    # 3EC2h # 3CC7h # 32C2h # 26085h # 3EC2h
    # 3CC7h # 3B46h # 34B3Eh
    { # 3188h # 26085h # 3E8Eh # 3223h
      # 3E8Eh # 3223h # 260B2h # 3223h
      Code
      # 3244h }
    "Pas réductible" }
}
```

Voici maintenant la chaîne de caractères qu'il faut entrer sans espace ni saut (646 octets - CRC # 3D0Dh) et compiler en utilisant `256 ATTACH H¨` juste après :

```
D9D20 79262 82362 F7133 D9D20 88130 58062 E0E30
32230 E0E30 2CE30 7CC30 2C230 58062 2CE30 7CC30
64B30 E3B43 D9D20 88130 58062 E8E30 32230 E8E30
32230 2B062 32230 CCD20 BB100 8FB97 60143 13016
E1428 22C4F 48324 0E4C4 81AF0 21641 3681A F0817
41431 30135 16914 6D5CD 16414 6C6C6 822F6 83240
E6CE8 1AF0C E6822 819F2 83240 E6C68 1AF0B 16413
281AF 01130 2F302 2081A F1CD7 7A909 C7D01 4A808
50148 79A09 C7D01 4A808 51148 1707F 609C7 D014A
80852 1487E 709C7 D014A 80853 14817 0160C F54A8
1AF10 81AF1 ACACA 81AF0 01318 1AF11 81AF1 BCA81
AF011 30CD4 60646 F8D34 150AC 37F30 81AF1 A133C
A1317 D2013 3EA13 101AC 37530 81AF1 A133C A1317
32013 3EA13 10114 B8086 050B4 78086 100B4 70114
B8086 250B4 78086 300B4 70144 230B2 130C2 A2012
00005 16370 2279E 46573 64796 26C65 6B213 0B213
0
```

• Le mot de la fin •

Nous nous sommes penchés sur la solution où l'on noircit un pixel sur le grob réduit si les blocs de quatre pixels du grob de départ contiennent au moins deux pixels noirs. Sur le cd-rom, vous retrouverez aussi le programme (nommé **ZP2B**) qui noircit un pixel du grob réduit quand un bloc contient au moins trois pixels noirs. Quelque part en plein milieu de la source, seule la ligne `LC 2` devient `LC 3`. Dans le principe, comme une image est réduite de 50%, la qualité du grob réduit est assez médiocre pour des petits grobs. Pour exemple, si notre programme s'appelle **ZP2N**, alors faites : `LCD¨ ZP2N` et vous verrez un image d'assez mauvaise qualité, même si on imagine assez bien ce que ça représente. Il faut donc faire ce genre d'opération sur des images de grande taille. Question rapidité, le programme est quasiment instantané pour réduire un image de taille quelconque.
Pour illustrer cet article, voici une image assez grande (à gauche) de **223 x 127** qu'on à réduite à 50% (à droite) avec **ZP2N** pour atteindre une taille de **111 x 63** :



Après rétrécissement, le petit frère a quand même perdu en qualité, mais on distingue quand même une réelle ressemblance à l'original.

Philippe Pamart
[phpamart@nordnet.fr]