

Filtre "nettoyant" Hp 49

Dans la série "les fonctions graphiques", voici après les zooms un filtre qui permet de faire un peu de ménage dans vos images...

Désormais habitués à MASD, le très célèbre assembleur implanté dans la Hp 49G, continuons à développer nos applications en utilisant ce fabuleux langage. Profitons-en d'ailleurs pour inclure dans cet article la partie initiation.

• généralités •

A l'instar des très célèbres logiciels de retouche d'images comme Paint Shop Pro ou Photoshop (Gimp pour les linuxiens), dont moult filtres aux applications différentes sont intégrés, réalisons un petit filtre sur Hp 49 qui aura pour but d'éliminer tout parasite dans une image. Le parasite en question sera le pâtre pixel (noir sur fond blanc ou blanc sur fond noir) isolé sur une image. Cette application s'adresse aux utilisateurs qui convertissent une image PC en image Hp et qui la transfèrent brute sur la calculatrice. Et quelques résidus, parfois gênants, demandent bien souvent à être éliminés.

• comment s'y prendre •

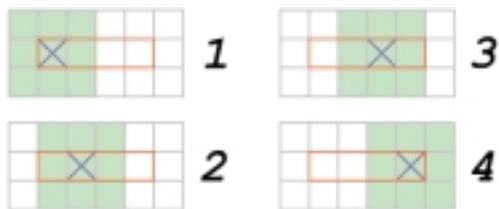
Pour chaque pixel, on sélectionne la bande de pixels qui l'entoure pour obtenir un carré de 9 pixels (3 sur 3), comme ce petit dessin :



Le carré de gauche représente un pixel noir entouré de huit pixels blancs, tandis que le carré de droite correspond à l'état de figure inverse. Donc, dans chaque cas, si le nombre de pixels de couleur différente au pixel courant est égal à huit, alors ce pixel devient de la même couleur que les autres. Autrement dit : un pixel noir perdu au beau milieu des blancs devient blanc et vis versa.

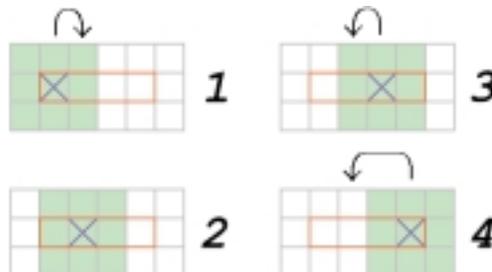
• la technique utilisée •

A priori, avec un énoncé posé comme ça, tout semble très facile. Mais en assembleur on ne gère pas des points mais des suites de quartets, et chaque quartet contient quatre points (comme chacun sait !). Il faut donc pour chaque bit de chaque quartet tester les bits qui les entourent... comme dans ce schéma :



Ici, le cadre rouge correspond au quartet, la croix bleue correspond au pixel courant et les pixels teintés en vert sont ceux qui vont être testés. On se rend assez vite compte que pour tester tous ces pixels, on risque d'obtenir une source assez lourde ! Il va donc falloir ruser un peu, ne serait-ce que pour créer une seule routine de test qui fonctionne dans tous les cas. On va donc choisir comme modèle un cas qui semble facile à tester, et opérer des décalages de bits sur tous les autres cas afin

d'obtenir l'identique du modèle. Le cas le plus simple à tester est le numéro 2. Donc pour le cas n°1 on réalisera un décalage d'un bit vers la droite sur le dessin (et à gauche en assembleur), un décalage d'un bit vers la gauche pour le cas n°3 et un décalage de deux bits vers la gauche pour le cas n°4 :



Donc pour le test de chaque bit d'un quartet, on fera un décalage avant d'appeler l'unique routine de calcul. Une fois le test réalisé, on peut éliminer le pixel parasite si la condition de test est validée.

• initiation à MASD •

Avant d'aller plus loin, présentons un peu plus le logiciel MASD et sa syntaxe. Déjà dans le précédent numéro, nous avons présenté une source avec de l'assembleur mais aussi un peu d'externals, comme dans cet article. Une source qu'on écrirait 100% en externals ressemblerait à ceci :

```
"! RPL !NO CODE
: :
;
@"
```

Et c'est entre les : : et le ; qu'on écrit le programme en externals. Le programme ci-dessus ne fait strictement aucune action, c'est le programme le plus simple que l'on puisse créer ! Comme exemple, réalisons un petit programme qui multiplie par 5 le réel qui se trouve en théorie sur le niveau 1 de la pile. Pour cela, on teste s'il y a bien un objet sur la pile et si effectivement c'est un réel. Si l'examen de passage est correct, on réalise la multiplication :

```
"! RPL !NO CODE
: :
CK1&Dispatch real
: :
%5
%*
;
;
@"
```

Simple non ? Maintenant, si l'on veut inclure un morceau d'assembleur, il suffit de le déclarer comme présenté dans la source ci-dessous et de placer les mnémoniques assembleur entre CODE et ENDCODE :

```
"! RPL !NO CODE
: :
CODE
ENDCODE
;
@"
```

• la source commentée •

L'avantage de MASD, c'est que la source inclut les externals et l'assembleur, on obtient donc une seule et même source à assembler avec ASM :

```
"! RPL ! NO CODE
::
CK1&Dispatch grob
::
TOTEMPOB

CODE
SAVE I NTOFF %sauve registres et pointeurs et interdit les interruptions
A=DAT1. A D1=A %adresse du grob
D1+10 A=DAT1. A R3=A. A %nombre de lignes
D1+5 C=DAT1. A R4=C. A %nombre de colonnes
GOSBVL O3991 %on les multiplie
?B=0. A -> FI N %si le résultat est égal à zéro, on quitte
GOSUB FI LTRE %sinon on applique le filtre
*FI N
LOADRPL %restaure les registres et pointeurs et retourne au Rpl

*FI LTRE
D1+5 AD1EX D1=A R2=A. A %début du grob
SB=0 A=R4. A A+A. A ASR. A %division par 8
?SB=0 -> SUI TE %si la division est entière, on saute
A+1.A %sinon on ajoute un octet
*SUI TE
A+A. A R1=A. A %largeur du grob en quartets
C=R3. A GOSBVL O3991 B-1. A %boucle principale
P=15 LC 1 D=C. S LC 8 B=C. S P=0 %initialisation des limites des compteurs
*BOUCLE
%Test le premier pixel d'un quartet
C=0. S %initialisation du compteur
D1-1 GOSUB TEST1 %test de la 2ème ligne du carré
GOSUB DOWN GOSUB TEST1 %test de la 3ème ligne du carré
GOSUB UP2 GOSUB TEST1 %test de la 1ère ligne du carré
GOSUB DOWN D1+1 %retour sur le quartet courant
A=DAT1. B %on prend le quartet
?C=B. S -> { ABI T=1. 0 } %1er pixel noirci pour 8 pixels noirs dans le carré
?C=D. S -> { ABI T=0. 0 } %1er pixel blanchi pour 8 pixels blancs dans le carré
DAT1=A. B %et on réécrit le quartet
%Test le second pixel d'un quartet
C=0. S %initialisation du compteur
GOSUB TEST2 %test de la 2ème ligne du carré
GOSUB DOWN GOSUB TEST2 %test de la 3ème ligne du carré
GOSUB UP2 GOSUB TEST2 %test de la 1ère ligne du carré
GOSUB DOWN %retour sur le quartet courant
A=DAT1. B %on prend le quartet
```

```
?C=B. S -> { ABI T=1. 1 } %2ème pixel noirci pour 8 pixels noirs dans le carré
?C=D. S -> { ABI T=0. 1 } %2ème pixel blanchi pour 8 pixels blancs dans le carré
DAT1=A. B %et on réécrit le quartet
%Test le troisième pixel d'un quartet
C=0. S %initialisation du compteur
GOSUB TEST3 %test de la 2ème ligne du carré
GOSUB DOWN GOSUB TEST3 %test de la 3ème ligne du carré
GOSUB UP2 GOSUB TEST3 %test de la 1ère ligne du carré
GOSUB DOWN %retour sur le quartet courant
A=DAT1. B %on prend le quartet
?C=B. S -> { ABI T=1. 2 } %3ème pixel noirci pour 8 pixels noirs dans le carré
?C=D. S -> { ABI T=0. 2 } %3ème pixel blanchi pour 8 pixels blancs dans le carré
DAT1=A. 1 %et on réécrit le quartet
%Test le dernier pixel d'un quartet
C=0. S %initialisation du compteur
GOSUB TEST4 %test de la 2ème ligne du carré
GOSUB DOWN GOSUB TEST4 %test de la 3ème ligne du carré
GOSUB UP2 GOSUB TEST4 %test de la 1ère ligne du carré
GOSUB DOWN %retour sur le quartet courant
A=DAT1. B %on prend le quartet
?C=B. S -> { ABI T=1. 3 } %4ème pixel noirci pour 8 pixels noirs dans le carré
?C=D. S -> { ABI T=0. 3 } %4ème pixel blanchi pour 8 pixels blancs dans le carré
DAT1=A. B %et on réécrit le quartet
D1+1 B-1. A RTNC %on passe au quartet suivant, on décrémente la boucle
GOTO BOUCLE %et on recommence

*TEST1
A=DAT1. B A+A. B ASR. B %on avance chaque ligne du carré d'un pixel
GOSUB COMPTE %appelle la routine de comptage des pixels allumés
RTN

*TEST2
A=DAT1. B %on ne bouge pas les lignes du carré
GOSUB COMPTE %appelle la routine de comptage des pixels allumés
RTN

*TEST3
A=DAT1. B ASRB. B %on recule chaque ligne du carré d'un pixel
GOSUB COMPTE %appelle la routine de comptage des pixels allumés
RTN

*TEST4
A=DAT1. B ASRB. B ASRB. B %on recule chaque ligne du carré de deux pixels
```

```
GOSUB COMPTE %appelle la routine de comptage des pixels allumés
RTN
```

```
*COMPTE
```

```
?ABI T=1. 0 -> { C+1. S } %test du 1er pixel d'une ligne du carré
?ABI T=1. 1 -> { C+1. S } %test du 2ème pixel d'une ligne du carré
?ABI T=1. 2 -> { C+1. S } %test du 3ème pixel d'une ligne du carré
RTN
```

```
*DOWN
```

```
AD1EX C=R1. A A+C. A D1=A %on descend d'une ligne
RTN
```

```
*UP2
```

```
AD1EX C=R1. A A-C. A A-C. A D1=A %on remonte de deux lignes
RTN
```

```
ENDCODE
```

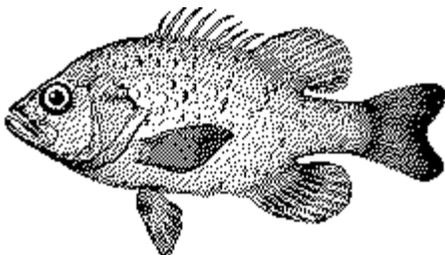
```
;
;
;
@"
```

Pour ceux qui préfèrent la chaîne hexadécimale, saisissez-la sans espace ni saut (542 octets - CRC # 1534h) et assemblez-la avec H' :

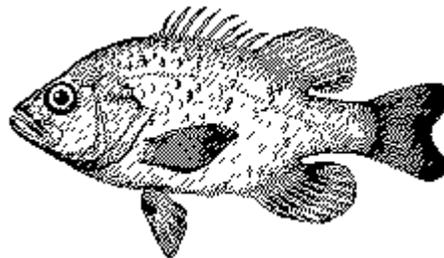
```
D9D20 00362 F7133 D9D20 75660 CCD20 1F100 8FB97
60808 F1431 31179 14381 AF031 74147 81AFO C8F19
9308A 96077 008D3 41501 74133 13181 AF028 2281A
F14C4 F4832 40E4C 481AF 0181A F1B8F 19930 CD2F3
01AC7 308AC 520AC 21C07 DD072 3175D 07A31 7DC07
22117 014B9 45708 08509 47708 08401 49AC2 74B07
AF07C AO720 174A0 7AE01 4B945 70808 51947 70808
41149 AC278 8075C 07080 7DC07 87075 B014B 94570
80852 94770 80842 1590A C2706 07F80 78507 79070
507F7 014B9 45708 08539 47708 08431 49170 CD400
6C1F1 4BA64 BE47C 20011 4B732 00114 B8196 07510
0114B 81960 81960 72000 18086 050B4 68086 150B4
68086 250B4 60113 381AF 19CA1 31011 3381A F19EA
EA131 01B21 30B21 30
```

• un petit exemple •

Reprenons l'image du poisson parue il y a deux mois dans Team Palmtops et appliquons-lui ce petit filtre. Tout d'abord, observez bien l'image avant traitement :



Les points isolés blancs ou noirs disparaissent après application du filtre :



Cette image de dimensions 223x127 est traitée en 8 secondes et 3 dixièmes. Et pour donner un ordre d'idée, une image qui correspond à la taille de l'écran (soit 131x64) est traitée en 2 secondes et 6 dixièmes. Inutile d'imaginer quel aurait pu être le chrono de la même application réalisée en Rpl...

• les améliorations possibles •

Dans ce petit programme, quelques petits points demandent à être améliorés. Tout d'abord, lorsqu'on traite une image, les points qui correspondent au cadre (ou aux bords de l'image) sont aussi traités. C'est à dire que pour le test d'un de ces points, on teste les pixels qui sont en dehors de l'image, comme par exemple pour le premier pixel en haut à gauche du grob où il est inutile de tester les pixels qui sont placés immédiatement à gauche et au dessus de lui.

Ensuite, on peut aisément adapter ce programme pour des images destinées à être utilisées en quatre ou huit niveaux de gris. Il faudrait tester par couleur isolée.

• la version Hp 48 série G •

On n'oublie pas les quarante-huitards avec le programme adapté pour leur machine favorite. Pour ceux qui disposent du Meta Kernel, la source présentée devrait pouvoir être compilée sans soucis. Sinon, la source est exactement la même en ce qui concerne l'assembleur. Et comme d'habitude, seules les adresses des externals ont changé. Voici juste la première ligne de la chaîne hexadécimale (542 octets - CRC # 6F54h), sachant que le reste est rigoureusement le même que pour la chaîne destinée à la Hp 49 :

```
D9D20 ECE81 76040 D9D20 75660 CCD20 . . .
```

• encore un petit mot •

Les filtres, comme tous les outils graphiques de manière générale, se programment assez facilement en assembleur. Bien évidemment, ce filtre reste une application basique qu'on réalise en très peu de temps. D'autres filtres graphiques (bien plus compliqués) seraient intéressants à programmer tels les déformations, les filtres artistiques, les flous, les contours, etc... Mais la qualité de l'écran de la Hp 49 en vaut-elle réellement la chandelle ?

Philippe Pamart
[phpamart@nordnet.fr]