

ATARI ST+STE

LOGICIEL

ROUTINES GRAPHIQUES ET SONORES EN GFA

SCROLLING,
SPRITES,

CARACTERES 3D,
DIGITALISATION MIDI...



EDITIONS MICRO APPLICATION



Routines Graphiques et Sonores en GFA

Copyright

- © 1990 GFA Systemtechnik GmbH
Heerdter Sandberg 30
D-4000 Düsseldorf 11
- © 1990 Micro Application
58, rue du Faubourg Poissonnière
75010 Paris

Auteurs

Hans-Peter Burk et Helmut Micko

Traducteurs

Mr et Mme Baudin

'Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de MICRO APPLICATION est illicite (Loi du 11 Mars 1957, article 40, 1er alinéa).

Cette représentation ou reproduction illicite, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

La Loi du 11 Mars 1957 n'autorise, aux termes des alinéas 2 et 3 de l'article 41, que les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à l'utilisation collective d'une part, et d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration'.

ISBN : 2-86899-376-1

*Collection dirigée par Philippe Olivier
Edition réalisée par Frédérique Beaudonnet*

Préface

Routines graphiques et sonores en GfA Basic a été conçue pour procurer, à l'amateur comme au programmeur averti, la possibilité d'accéder à plus de 40 effets spéciaux.

Ces effets spéciaux représentent un important travail de programmation et ne sont guère réalisables sans une connaissance approfondie tant du hardware que du software. C'est pourquoi notre bibliothèque GfA Basic comprend un ensemble de modules utilisables immédiatement sans qu'il soit nécessaire d'être un programmeur chevronné.

Les caractéristiques du langage BASIC, et surtout sa grande simplicité permettant un passage rapide à la programmation, facilitent également l'utilisation de cette bibliothèque.

En règle générale, la difficulté ne réside plus tant aujourd'hui dans le choix d'un langage de programmation mais dans la définition des objectifs poursuivis lors de l'écriture d'un programme.

Routines graphiques et sonores en GfA Basic vous sera d'une aide précieuse : qui ne rêve pas de programmer des 'sprites' se déplaçant sur l'écran ou de doter son programme d'une page d'accueil dont le texte défile sur l'écran ?

Inutile maintenant de recopier des pages et des pages de colonnes de chiffres, vous allez pouvoir commencêr immédiatement à programmer, et nous espérons que cette bibliothèque GfA Basic vous donnera entière satisfaction.

Sommaire

I. INTRODUCTION. 9

1. Contenu de la disquette.	9
2. Programmes de Démonstration	15
3. Utilisation de la Bibliothèque	15
4. Appel des procédures	17
5. Les variables	18
6. Les interruptions	18
7. Les taux de résolution.	18
8. Erreurs dans le passage des paramètres.	19
9. Trucs et astuces	19
10. Vue d'ensemble des procédures.	20
11. Les valeurs retournées	21

II. Description des procédures. 23

Graphisme : COLANI.	25
Graphisme : PLOAD	29
Graphisme : POPPAL	31
Graphisme : PSHOW.	33
Graphisme : INTANI.	39
Graphisme : L_SHAP16 M_SHAP16 H_SHAP16	45
Graphisme : L_SHAP32 M_SHAP32 H_SHAP32	55
Graphisme : L_SHAP64 M_SHAP64 H_SHAP64	63
Graphisme : L_TCONV M_TCONV.	71
Graphisme : L_TEXT88 M_TEXT88 H_TEXT88.	77
Graphisme : SETRES.	81
Graphisme : MRES.	85
Graphisme : MPAL	87
Graphisme : POLICE.	93
Graphisme : L_OPComp M_OPComp H_OPComp.	101
Graphisme : L_DECOMP M_DECOMP H_DECOMP.	105
Graphisme : H_LOUPE.	107
Graphisme : L_FADEOF M_FADEOF H_FADEOF	111
Graphisme : OFF14.	117

Graphisme : OFF15.	121
Graphisme : OFF16.	123
Graphisme : L_FADEOV M_FADEOV H_FADEOV	125
Graphisme : L_GREYS M_GREYS.	131
Graphisme : L_CPOFF_CPOFF M_CPOFF	135
Graphisme : L_CPON M_CPON	139
Graphisme : L_SCONV M_SCONV H_SCONV	141
Graphisme : L_HSCROL M_HSCROL H_HSCROL.	147
Graphisme : L_VCONV M_VCONV H_VCONV	151
Graphisme : L_VSCROL M_VSCROL H_VSCROL	157
Graphisme : GRAF2D	163
Graphisme : GRAF3D	169
Graphisme : VSCROL	175
Son : SMPLAY.	181
Son : SPEECH.	185
Son : REPSND.	189
Son : MIDIKY.	193
Divers : HORLOGE.	197
Divers : BLACK	201
Divers : PATMOV	203
Divers : NINPUT.	207
Divers : PINPUT.	211
Divers : NB5060.	213
Divers : RESET	215

1. INTRODUCTION

1. Contenu de la disquette

Les procédures sont réparties en trois lots :

A : Le graphisme

B : Le son

C : Divers

Pour faciliter l'accès à la disquette, nous avons rangé les procédures, les programmes d'exemple ainsi que les divers fichiers dans différents dossiers et sous-dossiers :

EXAMPLES\STUFF :	Fichiers complémentaires par exemple : images de démonstration
EXAMPLES\PROGRAMS.ALL :	Programmes de démonstration tournant dans les trois résolutions d'écran possibles
EXAMPLES\PROGRAMS.HIG :	Programmes de démonstration tournant en haute résolution
EXAMPLES\PROGRAMS.LOW :	Programmes de démonstration tournant en basse résolution
GFA_ALIB.3_0 :	Procédures de fonctions générales
GFA_GLIB.3_0\GFA_GLIB.ALL :	Procédures graphiques, tournant dans les trois résolutions possibles ou dans les deux résolutions couleur
GFA_GLIB.3_0\GFA_GLIB.HIG :	Procédures graphiques, tournant en haute résolution
GFA_GLIB.3_0\GFA_GLIB.LOW :	Procédures graphiques, tournant en basse résolution
GFA_GLIB.3_0\GFA_GLIB.MED :	Procédures graphiques, tournant en moyenne résolution
GFA_SLIB.3_0 :	Procédures sonores

Les procédures et les programmes de démonstration sont enregistrés sous forme de fichiers ASCII (extension LST) sur la disquette, et il vous suffit de les recharger par MERGE dans l'interpréteur GfA Basic. Ceci permet non seulement de gagner de la place en mémoire mais surtout de les intégrer

dans vos propres programmes, ce qui constitue d'ailleurs le principal intérêt de telles procédures.

Voici une vue d'ensemble de tous les fichiers se trouvant sur la disquette :

EXAMPLES

\EXAMPLES\STUFF

DEMO.PI1	Image Degas-Elite en basse résolution
DEMO.PIC	Image haute résolution en format 32000 octets
DEMO.SMP	Exemple sonore (son de cloche)
PHONEMS.SMP	Phonèmes pour la procédure SPEECH

\EXAMPLES\PROGRAMS.ALL

BLACK.LST	Démo de BLACK
HORLOGE.LST	Démo de HORLOGE
INTANI.LST	Démo de INTANI
MIDIKY.LST	Démo de MIDIKY
PINPUT.LST	Démo de PINPUT
REPSND.LST	Démo de REPSND
RESET.LST	Démo de RESET
SMPLAY.LST	Démo de SMPLAY
SPEECH.LST	Démo de SPEECH

\EXAMPLES\PROGRAMS.HIG

COLANI.LST	Démo de COLANI, PLOAD et POPPAL
GRAF2D.LST	Démo de GRAF2D
GRAF3D.LST	Démo de GRAF3D
H_FADEOF.LST	Démo de H_FADEOF
H_FADEOV.LST	Démo de H_FADEOV
H_LOUPE.LST	Démo de H_LOUPE
H_OPCDEC.LST	Démo de H_OPComp et H_DECOMP
H_SCOHSC.LST	Démo de H_SCONV et H_SCROLL
H_SHAP16.LST	Démo de H_SHAP16
H_SHAP32.LST	Démo de H_SHAP32
H_SHAP64.LST	Démo de H_SHAP64
H_TCOTEX.LST	Démo de H_TCONV et H_TEXT88
H_VSCROL.LST	Démo de H_VSCROL
NINPUT.LST	Démo de NINPUT
OFFXX.LST	Démo de OFFXX
PATMOV.LST	Démo de PATMOV
POLICE.LST	Démo de POLICE

PSHOW.LST	Démo de PSHOW
VSCROL.LST	Démo de VSCROL

\\EXAMPLES\\PROGRAMS.LOW

COLANI.LST	Démo de COLANI, PLOAD et POPPAL
GRAF2D.LST	Démo de GRAF2D
GRAF3D.LST	Démo de GRAF3D
L_CPOXX.LST	Démo de L_CPOFF et L_CPON
L_FADEOF.LST	Démo de L_FADEOF
L_FADEOV.LST	Démo de L_FADEOV
L_GREYS.LST	Démo de L_GREYS
L_OPDEC.LST	Démo de L_OPCOMP et L_DECOMP
L_SCOHSC.LST	Démo de L_SCONV et L_HSCROL
L_SHAP16.LST	Démo de L_SHAP16
L_SHAP32.LST	Démo de L_SHAP32
L_SHAP64.LST	Démo de L_SHAP64
L_TCOTEX.LST	Démo de L_TCONV et L_TEXT88
L_VSCROL.LST	Démo de L_VSCROL
MPAL.LST	Démo de MPAL
MRES.LST	Démo de MRES et SETRES
NB5060.LST	Démo de NB5060
NINPUT.LST	Démo de NINPUT
OFFXX.LST	Démo de OFFXX
PATMOV.LST	Démo de PATMOV
POLICE.LST	Démo de POLICE
PSHOW.LST	Démo de PSHOW
VSCROL.LST	Démo de VSCROL

GFA_ALIB.3_0

BLACK.LST	Procédure BLACK
HORLOGE.INL	Fichier INLINE de la procédure HORLOGE
HORLOGE.LST	Procédure HORLOGE
NB5060.LST	Procédure NB5060
NINPUT.LST	Procédure NINPUT
PATMOV.INL	Fichier INLINE de la procédure PATMOV
PATMOV.LST	Procédure PATMOV
PINPUT.LST	Procédure PINPUT
RESET.INL	Fichier INLINE de la procédure RESET
RESET.LST	Procédure RESET

GFA_GLIB.3_0

\GFA_GLIB.3_0\GFA_GLIB.ALL

COLANI.INL	Fichier INLINE de la procédure COLANI
COLANI.LST	Procédure COLANI
GRAF2D.LST	Procédure GRAF2D
GRAF3D.LST	Procédure GRAF3D
INTANI.INL	Fichier INLINE de la procédure INTANI
INTANI.LST	Procédure INTANI
MPAL.INL	Fichier INLINE de la procédure MPAL
MPAL.LST	Procédure MPAL
MRES.INL	Fichier INLINE de la procédure MRES
MRES.LST	Procédure MRES
OFFXX.LST	Procédure OFFXX
PLOAD.LST	Procédure PLOAD
POLICE.LST	Procédure POLICE
POPPAL.LST	Procédure POPPAL
PSHOW.LST	Procédure PSHOW
SETRES.LST	Procédure SETRES
VSCROL.INL	Fichier INLINE de la procédure VSCROL
VSCROL.LST	Procédure VSCROL

\GFA_GLIB.3_0\GFA_GLIB.HIG

H_DECOMP.INL	Fichier INLINE de la procédure H_DECOMP
H_DECOMP.LST	Procédure H_DECOMP
H_FADEOF.INL	Fichier INLINE de la procédure H_FADEOF
H_FADEOF.LST	Procédure H_FADEOF
H_FADEOV.INL	Fichier INLINE de la procédure H_FADEOV
H_FADEOV.LST	Procédure H_FADEOV
H_HSCROL.INL	Fichier INLINE de la procédure H_SCROL
H_HSCROL.LST	Procédure H_SCROL
H_LOUPE.INL	Fichier INLINE de la procédure H_LOUPE
H_LOUPE.LST	Procédure H_LOUPE
H_OPCOMP.INL	Fichier INLINE de la procédure H_OPCOMP
H_OPCOMP.LST	Procédure H_OPCOMP
H_SCONV.INL	Fichier INLINE de la procédure H_SCONV
H_SCONV.LST	Procédure H_SCONV
H_SHAP16.INL	Fichier INLINE de la procédure H_SHAP16
H_SHAP16.LST	Procédure H_SHAP16
H_SHAP32.INL	Fichier INLINE de la procédure H_SHAP32
H_SHAP32.LST	Procédure H_SHAP32

H_SHAP64.INL	Fichier INLINE de la procédure H_SHAP64
H_SHAP64.LST	Procédure H_SHAP64
H_TCONV.LST	Procédure H_TCONV
H_TEXT88.INL	Fichier INLINE de la procédure H_TEXT88
H_TEXT88.LST	Procédure H_TEXT88
H_VCONV.LST	Procédure H_VCONV
H_VSCROL.INL	Fichier INLINE de la procédure H_VSCROL
H_VSCROL.LST	Procédure H_VSCROL

\GFA_GLIB.3_0\GFA_GLIB.LOW

L_CPOFF.LST	Procédure L_CPOFF
L_CPON.LST	Procédure L_CPON
L_DECOMP.INL	Fichier INLINE de la procédure L_DECOMP
L_DECOMP.LST	Procédure L_DECOMP
L_FADEOF.INL	Fichier INLINE de la procédure L_FADEOF
L_FADEOF.LST	Procédure L_FADEOF
L_FADEOV.INL	Fichier INLINE de la procédure L_FADEOV
L_FADEOV.LST	Procédure L_FADEOV
L_GREYS.LST	Procédure L_GREYS
L_HSCROL.INL	Fichier INLINE de la procédure L_HSCROL
L_HSCROL.LST	Procédure L_HSCROL
L_OPCOMP.INL	Fichier INLINE de la procédure L_OPCOMP
L_OPCOMP.LST	Procédure L_OPCOMP
L_SCONV.INL	Fichier INLINE de la procédure L_SCONV
L_SCONV.LST	Procédure L_SCONV
L_SHAP16.INL	Fichier INLINE de la procédure L_SHAP16
L_SHAP16.LST	Procédure L_SHAP16
L_SHAP32.INL	Fichier INLINE de la procédure L_SHAP32
L_SHAP32.LST	Procédure L_SHAP32
L_SHAP64.INL	Fichier INLINE de la procédure L_SHAP64
L_SHAP64.LST	Procédure L_SHAP64
L_TCONV.LST	Procédure L_TCONV
L_TEXT88.INL	Fichier INLINE de la procédure L_TEXT88
L_TEXT88.LST	Procédure L_TEXT88
L_VCONV.LST	Procédure L_VCONV
L_VSCROL.INL	Fichier INLINE de la procédure L_VSCROL
L_VSCROL.LST	Procédure L_VSCROL

\GFA_GLIB.3_0\GFA_GLIB.MED

M_CPOFF.LST	Procédure M_CPOFF
M_CPON.LST	Procédure M_CPON

M_DECOMP.INL	Fichier INLINE de la procédure M_DECOMP
M_DECOMP.LST	Procédure M_DECOMP
M_FADEOF.INL	Fichier INLINE de la procédure M_FADEOF
M_FADEOF.LST	Procédure M_FADEOF
M_FADEOV.INL	Fichier INLINE de la procédure M_FADEOV
M_FADEOV.LST	Procédure M_FADEOV
M_GREYS.LST	Procédure M_GREYS
M_HSCROL.INL	Fichier INLINE de la procédure M_HSCROL
M_HSCROL.LST	Procédure M_HSCROL
M_OPCOMP.INL	Fichier INLINE de la procédure M_OPCOMP
M_OPCOMP.LST	Procédure M_OPCOMP
M_SCONV.INL	Fichier INLINE de la procédure M_SCONV
M_SCONV.LST	Procédure M_SCONV
M_SHAP16.INL	Fichier INLINE de la procédure M_SHAP16
M_SHAP16.LST	Procédure M_SHAP16
M_SHAP32.INL	Fichier INLINE de la procédure M_SHAP32
M_SHAP32.LST	Procédure M_SHAP32
M_SHAP64.INL	Fichier INLINE de la procédure M_SHAP64
M_SHAP64.LST	Procédure M_SHAP64
M_TCONV.LST	Procédure M_TCONV
M_TEXT88.INL	Fichier INLINE de la procédure M_TEXT88
M_TEXT88.LST	Procédure M_TEXT88
M_VCONV.LST	Procédure M_VCONV
M_VSCROL.INL	Fichier INLINE de la procédure M_VSCROL
M_VSCROL.LST	Procédure M_VSCROL

GFA_SLIB.3_0

MIDIKY.LST	Procédure MIDIKY
REPSND.INL	Fichier INLINE de la procédure REPSND
REPSND.LST	Procédure REPSND
SMPLAY.INL	Fichier INLINE de la procédure SMPLAY
SMPLAY.LST	Procédure SMPLAY
SPEECH.LST	Procédure SPEECH

2. Programmes de démonstration

Toutes les procédures présentées ici en GfA Basic sont accompagnées de programmes de démonstration sur la disquette ; ces programmes sont écrits en GfA Basic 3.0 et sont directement opérationnels.

Les programmes de démonstration se présentent sous la forme de fichiers ASCII et se trouvent dans les dossiers PROGRAMS.ALL, PROGRAMS.HIG et PROGRAMS.LOW ; il vous suffit de les charger dans l'interpréteur en utilisant la commande MERGE (*touche de fonction F2*).

Les fichiers de démonstration ne sont pas en état de tourner seuls : pour des raisons de place, les procédures reprises dans la démonstration ne sont pas enregistrées avec celle-ci. Vous devez les ajouter (**merge**) avant de lancer la démonstration.

Nous avons regroupé les démonstrations dans les trois dossiers désignés ci-dessus, selon le degré de résolution d'écran (HIG = haute résolution, LOW = basse résolution).

Les procédures ne tournant qu'en moyenne résolution (*toutes celles qui commencent par M_*) ne sont pas accompagnées de programmes de démonstration, car elles ne se distinguent guère des procédures similaires tournant en basse résolution.

Il arrive assez souvent qu'un même programme de démonstration comprenne l'usage de plusieurs procédures différentes, soit parce qu'il s'agit de procédures logiquement apparentées soit parce que cela est nécessaire par rapport aux différents objectifs poursuivis. Les deux procédures PLOAD et POPPAL ne sont ainsi accompagnées d'aucun programme de démonstration ; mais comme nous les utilisons pratiquement dans tous les autres programmes, cela suffit pour expliciter leur mode de fonctionnement.

3. Utilisation de la bibliothèque GfA Basic

Il est très facile d'intégrer une procédure dans un de vos programmes, et c'est ce que nous allons démontrer à l'aide d'un exemple concret.

Supposons que nous souhaitions utiliser la procédure HORLOGE pour faire apparaître sur l'écran une horloge indiquant l'écoulement des heures, minutes et secondes. Il vous faut commencer par charger dans l'interpréteur GfA Basic le programme HORLOGE.LST qui se trouve dans le dossier EXAMPLE\PROGRAMS.ALL.

Après quoi vous chargez la procédure correspondante, HORLOGE : appuyez sur <control> + <Z> pour amener le curseur à la fin du programme de démonstration. La procédure HORLOGE se trouve dans le dossier GFA_ALIB.3_0, car elle peut tourner avec n'importe lequel des trois degrés de résolution possibles.

Comme la procédure HORLOGE fait appel à une routine en assembleur enregistrée dans un fichier INLINE, il convient encore de recharger le fichier INLINE adéquat se trouvant sur la disquette. Notez que les procédures faisant ainsi appel à des fichiers INLINE sont accompagnées sur la disquette des fichiers adéquats, lesquels se trouvent dans le même dossier que la procédure et portent le même nom, suivi de l'extension '.INL'.

Pour charger un fichier INLINE :

- Dans la procédure, positionnez le curseur sur l'appel de INLINE
- Appuyez sur la touche <help>
- Appuyez sur <L> (pour 'load') et vous voyez apparaître une boîte de sélection vous permettant de sélectionner le fichier INLINE adéquat.

Une fois ainsi sélectionné, le fichier est chargé dans le buffer.

Attention : ∇

Notez bien que vous trouverez ci-après, dans le chapitre 2, une liste des procédures, dans laquelle nous précisons derrière chaque procédure si elle fait appel ou non à un fichier INLINE et, le cas échéant, où ce dernier se trouve. Veillez à ne charger que les fichiers adéquats, et non les fichiers INLINE affectés à d'autres procédures, car la moindre erreur entraînerait un plantage du système, ou tout au moins l'apparition d'un message d'erreur.

Plusieurs procédures font appel à l'instruction INLINE pour d'autres raisons, si bien qu'il n'est pas alors nécessaire de charger un fichier ; d'autres procédures contiennent deux fois l'instruction INLINE. Vous

devez donc lire attentivement les instructions fournies dans la description de chaque procédure.

Une instruction `INLINE` vous demandant de charger un fichier `INLINE` se reconnaît à ce qu'elle contient, en tant que paramètre, la variable `codeadr__%`. Seule exception, les procédures `SHAPE`, dont voici les variables `INLINE` :

```
- l_shap16mc__% - l_shap32mc__% - l_shap64mc__%  
- m_shap16mc__% - m_shap32mc__% - m_shap64mc__%  
- n_shap16mc__% - n_shap32mc__% - n_shap64mc__%
```

Après avoir chargé le fichier `INLINE` adéquat, vous pouvez lancer le programme.

4. Appel des procédures

Voici comment vous y prendre pour intégrer l'une des procédures dans un de vos propres programmes.

Nous vous précisons, dans les explications concernant chaque procédure, si cette dernière fait appel ou non à un fichier `INLINE` que vous devrez charger avant de lancer l'exécution de la procédure concernée. Si vous oubliez de charger ce complément `INLINE`, l'exécution de la procédure ne peut avoir lieu : la variable `lib_rv__%` prend la valeur -1, ce qui sert de code d'erreur et la valeur 0 lorsque tout se passe correctement.

Ces fichiers `INLINE` sont indispensables par exemple lorsqu'une procédure fait appel à une routine en assembleur, car il faut d'abord charger cette routine en mémoire avant de lancer l'exécution de la procédure.

Cette séparation des fichiers `INLINE` et des procédures correspondantes vient de ce que le fichier `INLINE` n'est pas sauvegardé lorsqu'on enregistre un programme en GfA Basic à l'aide de la commande `SAVE,A`.

Par contre, lorsque vous allez vous-même écrire votre programme en y intégrant une ou plusieurs procédures nécessitant éventuellement des codes `INLINE`, vous sauvegarderez ce texte en utilisant la commande `SAVE` (format 'tokenisé' et non pas ASCII comme ce serait le cas avec `SAVE,A`), si bien que le code `INLINE` sera sauvegardé en même temps que la procédure. Par la suite, lorsque vous chargerez votre programme à l'aide de

LOAD, vous n'aurez plus besoin de recharger les fichiers INLINE puisqu'ils seront déjà contenus dans le programme.

5. Les variables

Toutes les variables utilisées dans les procédures portent un nom se terminant par deux tirets bas (**anglais: underscores**) comme par exemple codeadr__%. Ceci vous permet d'éviter facilement toute confusion avec des variables de votre programme principal, dans lequel vous éviterez soigneusement de donner aux variables des noms se terminant ainsi par deux tirets bas.

C'est là pour vous la meilleure garantie d'éviter des conflits entre variables. Certes, la plupart des variables sont déclarées comme variables locales dans nos procédures, mais il était impossible d'éviter totalement d'avoir recours à quelques variables globales, dont il ne faut pas modifier la valeur.

6. Les interruptions

Certaines des procédures de cette bibliothèque utilisent des routines d'interruption. Lorsque vous les utilisez, vous devez absolument veiller à ce qu'elles soient suivies quelque part d'un 'stop', de façon à y mettre fin AVANT de ressortir de votre programme. En cas d'oubli, vous provoquerez un plantage général du système ou d'autres effets tout aussi peu agréables !

7. Les taux de résolution

Comme nous l'avons déjà dit, la disquette contient des procédures pour chaque degré de résolution. Il est facile de reconnaître les procédures qui ne peuvent tourner que sous telle ou telle résolution : leur nom commence par une lettre (H, L ou M) précisant le degré de résolution adéquat, comme par exemple dans le nom L_SHAP16 (L est l'abréviation de 'Low resolution' = basse résolution).

La plupart des procédures sont données dans les trois versions ; certaines cependant n'existent que dans telle ou telle résolution, puisqu'elles utilisent des propriétés liées à un degré de résolution déterminé (*palette des couleurs*,

fréquence de balayage de l'écran etc). C'est d'ailleurs la raison pour laquelle il y a moins de procédures tournant en haute résolution qu'en moyenne ou basse résolution.

8. Erreur dans le passage des paramètres

Nous n'avons inclue aucun contrôle des paramètres passés dans la plupart des procédures, car ceci aurait augmenté considérablement les dépenses de programmation, l'espace occupé en mémoire ainsi que la longueur des procédures. C'est à vous, utilisateur, de prendre en main ces contrôles pour qu'aucun paramètre erroné ne puisse être passé à une procédure.

Attention : ▽

Dans certains cas, le fait de passer un paramètre inexact à une procédure peut suffire pour provoquer un plantage général du système.

Certaines procédures disposent tout de même d'une vérification des paramètres passés afin de parer à un plantage. En cas d'erreur et lorsque cette procédure de contrôle existe, la valeur retournée par la procédure (*voir chapitre 2*) représente un code d'erreur (`lib_rv__%`), qui est explicité au début de la procédure.

9. Trucs et astuces

- Soyez patient lors du chargement d'un fichier LST, car cela peut prendre un certain temps ;
- Ne procédez à aucune modification dans les procédures (fichiers LST), ou alors ce sera à vos risques et périls : nous ne pouvons plus garantir le bon fonctionnement des procédures à partir du moment où vous intervenez dedans ;
- Il est possible d'appeler la procédure HORLOGE en lui faisant prendre la même valeur que l'heure système : retapez les quelques lignes suivantes :

```
hours__% = VAL(LEFT$(TIME$, 2))  
minutes__% = VAL(MID$(TIME$, 4, 6))  
seconds__% = VAL(RIGHT$(TIME$, 2))
```

> il est tout à fait possible d'utiliser plusieurs fois les procédures suivantes :

HORLOGE.LST heure actuelle
INTANI.LST animation par interruption

On peut faire tourner sept horloges simultanément : cela dépend du nombre de procédures utilisées faisant appel à l'interruption VBL (exemple : $5 * \text{HORLOGE} + 2 * \text{INTANI}$).

Veuillez vous reporter aux pages qui vont suivre pour savoir quels sont les procédures qui utilisent cette interruption.

Pour faire tourner par exemple deux horloges simultanément, vous chargez deux fois le fichier LST correspondant, après quoi vous modifiez tous les noms de variables dans l'une des deux procédures en leur ajoutant par exemple une lettre ou un chiffre ; il faut aussi changer le nom de la procédure.

10. Vue d'ensemble des procédures

BLACK	HORLOGE	M_CPOFF	MRES
COLANI	INTANI	M_CPON	NB5060
GRAF2D	L_CPOFF	M_DECOMP	NINPUT
GRAF3D	L_CPOFF	M_FADEOF	OFFXX
H_DECOMP	L_DECOMP	M_FADEOV	PATMOV
H_FADEOF	L_FADEOF	M_GREYS	PINPUT
H_FADEOV	L_FADEOV	M_HSCROL	PLOAD
H_LOUPE	L_GREYS	M_OPCOMP	POLICE
H_OPCOMP	L_HSCROL	M_SCONV	POPPAL
H_SCONV	L_OPCOMP	M_SHAP16	PSHOW
H_SCROL	L_SCONV	M_SHAP32	REPSND
H_SHAP16	L_SHAP16	M_SHAP64	RESET
H_SHAP32	L_SHAP32	M_TCONV	SETRES
H_SHAP64	L_SHAP64	M_TEXT88	SMPLAY
H_TCONV	L_TCONV	M_VCONV	SPEECH
H_TEXT88	L_TEXT88	M_VSCROL	VSCROL
H_VCONV	L_VCONV	MIDIKY	
H_VSCROL	L_VSCROL	MPAL	

11. Les valeurs retournées

Plusieurs procédures retournent une valeur, soit pour signaler une erreur soit parce que cela entre dans leurs attributions. Lorsque la valeur retournée est de type numérique, elle sera passée à la variable `lib_rv__%` ; lorsqu'il s'agit d'une chaîne de caractères, elle sera passée à la variable `lib_rv__$`.

Exemple :

Les procédures faisant appel à un fichier `INLINE` vont d'abord vérifier dans le buffer `INLINE` si le fichier a bien été chargé. Si tel n'est pas le cas, l'exécution de la procédure est suspendue, ce qui vous évite de recevoir quelques petites bombes sur votre écran : la valeur -1 est retournée à la variable `lib_rv__%`.

Veuillez consulter ce manuel pour savoir si la procédure utilisée renvoie ou non une valeur et quelle est sa fonction

II. Description des procédures

Dans les pages qui vont suivre, nous allons décrire une par une chacune des procédures. La description s'appuie sur un petit programme de démonstration.

Toutes les procédures font l'objet de la même grille d'explication, divisée en plusieurs rubriques, que nous allons préciser.

Utilisation : Vous trouverez dans cette rubrique une brève présentation de la fonction assurée par la procédure c'est-à-dire, pour parler plus couramment, ce à quoi la procédure peut servir.

Dossier : Cette rubrique vous indique l'endroit où se trouve la procédure sur la disquette (*chemin d'accès*) et sous quel nom de fichier elle est enregistrée.

INLINE : Cette rubrique vous précise si la procédure en question fait ou non appel à un fichier INLINE qu'il faut charger avant l'exécution de la procédure. Lorsque tel est le cas, nous vous indiquons le chemin d'accès et le nom du fichier INLINE à charger. En général, le fichier INLINE porte le même nom que la procédure correspondante, dont il se distingue par l'extension 'INL'.

Exemple : Cette rubrique vous indique s'il existe un programme de démonstration se rapportant à la procédure concernée, et, si tel est le cas, dans quel dossier et sous quel nom il se trouve.

Syntaxe : Cette rubrique vous indique comment lancer la procédure, et quels paramètres il convient de lui passer.

Paramètres : Nous vous expliquons, si nécessaire, quels paramètres il convient de passer à la procédure, leur fonction, leur valeur etc.

- Valeur retournée :** Cette rubrique vous indique si la procédure retourne une valeur, et, si tel est le cas, la nature de cette valeur (*numérique ou chaîne de caractères*) ainsi que sa signification.
- Description :** Cette rubrique contient une description plus détaillée de la procédure, éventuellement certains détails et particularités de fonctionnement.
- MERGE :** Vous trouverez ici les procédures (avec leur chemin d'accès) qu'il convient de charger pour que le programme de démonstration puisse tourner correctement.
- > :** Cette rubrique contient des explications détaillées relatives au programme de démonstration.

Graphisme : COLANI

Utilisation : Provoque une 'rotation' du registre des couleurs (cycle de couleurs) par interruption. Le nombre de registres utilisés ainsi que leur succession sont laissés à l'appréciation du programmeur.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\COLANI.LST
(quelle que soit la résolution).

INLINE : GFA_GLIB.3_0\GFA_GLIB.ALL\COLANI.INL

Exemples : EXAMPLES\PROGRAMS.LOW\COLANI.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\COLANI.LST
(haute résolution)

Syntaxe : GOSUB gfa_colani__(sourceadr__%,option__%)

Paramètres :

- *sourceadr__%*

Adresse d'un bloc de paramètres ayant la structure suivante :

1. **Word** = nombre de registres couleur utilisés dans le cycle de couleurs(n)
2. **Word** = vitesse, mesurée en 1/50ème ou 1/60ème de seconde (selon la fréquence de balayage de l'image) ; si les couleurs doivent changer à chaque seconde et si la fréquence de répétition de l'image est de 50 Hz, vous entrez la valeur 50, car $50/50 = 1$ seconde.
3. **Word** = Numéro du premier registre des couleurs (0 - 15).
4. **Word** = Numéro du deuxième registre des couleurs
- etc....
- x. **Word** = Numéro du n-ième registre des couleurs

y. Word = numéro du premier registre des couleurs.

Notez bien que le dernier mot 'word' sert à répéter le numéro du premier registre se trouvant déjà dans le 3ème word. Il faut aussi veiller à bien entrer le même nombre de registres de couleurs que celui déclaré dans le 1er 'word', soit 'n' registres : le premier registre (*répété en y*) ne compte que pour un seul registre. Le nombre de registres mis en oeuvre ainsi que leur succession restent à la discrétion du programmeur, qui peut d'ailleurs mobiliser plusieurs fois le même registre si bon lui semble, sans avoir à respecter quelque limite que ce soit.

- options__% =

Ce flag sert à préciser s'il convient de lancer ou d'arrêter l'interruption du cycle de couleurs ; il peut prendre les états suivants :

- > options__% <> 0 = lancer la rotation de couleurs
- > options__% == 0 = arrêter la rotation de couleurs.

Valeur retournée :

lib_rv__% = code d'erreur ; lib_rv__% retourne la valeur -1 lorsque vous oubliez de charger le fichier INLINE accompagnant COLANI, et la valeur 0 lorsque l'exécution de la procédure s'est bien déroulée.

Description :

vous pouvez sélectionner un nombre quelconque de registres de couleurs, selon la résolution de votre écran. Ces numéros de registres se présentent en quelque sorte sous la forme d'une chaîne ; les couleurs se mettent à se succéder, à la vitesse indiquée dans 'word 2', les couleurs des registres non impliqués restant inchangées.

Cela n'interrompt pas le déroulement normal du programme en cours, puisque cette routine tourne par interruption.

Veillez à bien interrompre la rotation de couleurs avant la fin du programme, faute de quoi il continuerait à tourner (*y compris au niveau du bureau GEM*) ou provoquerait un plantage de l'ordinateur.

Exemple (Basse résolution) :

EXAMPLES\PROGRAMS.LOW\COLANI.LST

MERGE: GFA_LIB.3_0\GFA_GLIB.ALL\PLOAD.LST
GFA_LIB.3_0\GFA_GLIB.ALL\POPPAL.LST

Le fonctionnement du programme de démonstration nécessite donc le chargement de trois procédures (COLANI.LST, PLOAD.LST, POPPAL.LST).

```
'                                     ! buffer pour le bloc de paramètres
INLINE pblockadr__%,20
'
GOSUB gfa_poppal__
palette$=lib_rv__$
'                                     ! écrire le bloc de paramètres
DPOKE pblockadr__%,7                ! 7 registres de couleurs mobilisés
DPOKE pblockadr__%+2,5              ! vitesse 5/50 sec = 10 Hz
DPOKE pblockadr__%+4,5              ! registre de couleurs numéro 5
DPOKE pblockadr__%+6,6              ! registre de couleurs numéro 6
DPOKE pblockadr__%+8,7              ! registre de couleurs numéro 7
DPOKE pblockadr__%+10,8             ! registre de couleurs numéro 8
DPOKE pblockadr__%+12,9             ! registre de couleurs numéro 9
DPOKE pblockadr__%+14,10            ! registre de couleurs numéro 10
DPOKE pblockadr__%+16,11            ! registre de couleurs numéro 11
DPOKE pblockadr__%+18,5             ! registre de couleurs numéro 5 = 1er
registre
GOSUB gfa_pload__("\EXAMPLES\STUFF\DEMO.PI1",XBIOS(2),0)
ALERT 1," |lancer le | cycle de couleurs",1," OUI ",button|
'
GOSUB gfa_colani__(pblockadr__%,1)
'
ALERT 2," |stopper le |cycle de couleurs",1," OUI ",button|
'
GOSUB gfa_colani__(0,0)
~XBIOS(6,L :V :palette$)
```

Le programme commence par l'installation, grâce à INLINE, d'un buffer de 20 octets, destiné à recevoir le bloc des paramètres indispensables au bon fonctionnement de COLANI. L'adresse de début du buffer se trouve sous 'pblockadr__%'.

On sauvegarde ensuite la palette actuelle des couleurs à l'aide de la procédure POPPAL, à laquelle il n'est pas besoin de passer des paramètres, Mais qui retourne la palette en question dans la variable string 'lib_rv__\$'. Cette variable contient la palette des couleurs dans un format identique par exemple à celui de la routine 6 XBIOS ou de la procédure x_CPOFF de notre bibliothèque.

Juste après l'appel de la procédure POPPAL, le contenu de lib_rv__\$ est recopié dans la variable 'palette\$' puisque 'lib_rv__\$' sera appelé à reprendre des valeurs retournées éventuellement par d'autres procédures de notre bibliothèque.

Nous créons ensuite le bloc de paramètres en écrivant (instruction DPOKE) chacun des paramètres dans l'INLINE ; c'est là qu'il convient d'indiquer que nous mobilisons (*dans notre exemple*) les sept registres de couleurs numérotés 5 à 11, avec une vitesse de 'rotation' de 10 Hz (*pour une fréquence de répétition de l'image de 50 Hz*) ou de 12 Hz (*fréquence de répétition de 60 Hz*) ce qui donnera environ 10 ou 12 changements de registres à la seconde.

La procédure PLOAD nous sert alors à charger une image Degas-Elite dans la mémoire de l'écran (XBIOS(2)). Il convient d'entrer le chemin d'accès au fichier contenant l'image et d'attribuer la valeur 0 (précisant qu'il s'agit d'une image Degas-Elite) au paramètre 'options__%'. PLOAD charge l'image et sa palette de couleurs.

Le cycle de couleurs est lancé lorsque l'utilisateur répond positivement à la question s'affichant dans une boîte de dialogue (alertbox). La deuxième boîte de dialogue (stopper le cycle de couleurs ?) s'affiche à l'écran alors que le cycle de couleurs est en train de tourner (=> interruption), ce qui serait rigoureusement impossible à programmer uniquement en Basic.

Cette deuxième boîte de dialogue permet à l'utilisateur de sortir du programme de démonstration en interrompant la rotation des couleurs lorsqu'il le souhaite.

Graphisme : PLOAD

Utilisation : Chargement, incluant la définition de la palette de couleurs, d'une image Degas/Degas-Elite, Neochrome ou 32000 octets.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST
(quelle que soit la résolution)

INLINE : Ne nécessite aucun fichier INLINE

Exemples : Pas de programme de démonstration spécifique ; voir utilisation par exemple dans COLANI.LST

Syntaxe : GOSUB gfa_pload__(txt__\$,destadr__%,options__%)

Paramètres :

- *txt__\$* :

String contenant le chemin d'accès de l'image à charger ; lib_rv__% retourne la valeur 1 lorsque ce fichier s'avère introuvable, et la valeur 0 lorsque tout se passe bien.

- *destadr__%* :

Adresse-cible de l'image à charger ; on peut par exemple entrer ici XBIOS(2) lorsque l'on souhaite charger l'image dans la mémoire d'écran, mais il est tout à fait possible de charger l'image autre part dans la mémoire. Rappelez-vous cependant qu'avec Degas/Degas-Elite et Neochrome, vous chargez en même temps l'image et sa palette de couleurs.

- *options__%* :

Flag précisant le format de l'image à charger parmi les trois possibilités suivantes :

- options__% == 0 : Charger une image Degas/Degas-Elite avec sa palette
- options__% == 1 : Charger une image Neochrome avec sa palette

> options__% == 2 : Charger une image 32000 octets (*exp* : *DOODLE*) sans
Charger de palette de couleurs.

Valeur retournée :

lib_rv__% =

Code d'erreur ; *lib_rv__%* retourne la valeur 0 lorsque l'image a bien été chargée et la valeur 1 si le fichier indiqué n'existe pas ou reste introuvable.

Description :

La procédure PLOAD permet de charger des images de format différent ; qui plus est, la palette de couleur est chargée en même temps que l'image elle-même.

Vous pouvez charger des images Degas/Degas-Elite, Neochrome ou 32000 octets. La procédure ne vérifie pas si le fichier concerné est bien au format indiqué : cela signifie que, si la taille du fichier est correcte, vous pouvez charger n'importe quelle image avec n'importe quelles couleurs. Des erreurs peuvent survenir si le fichier s'avère trop petit. Voici la taille des images chargeables :

Degas :	32034 octets
Degas-Elite :	32066 octets
Neochrome :	32128 octets
32000 octets :	32 Ko

Attention : ▽

Cette procédure ne permet pas de charger des images comprimées sous Degas ou Degas-Elite.

Exemple :

Voir par exemple dans COLANI :
EXAMPLES\PROGRAMS.LOW\COLANI.LST

Graphisme : POPPAL

Utilisation : Sauvegarder la palette de couleurs actuelle puis la recharger.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST
(quelle que soit la résolution)

INLINE : Ne nécessite aucun fichier INLINE

Exemples : Pas de programme de démonstration spécifique ; voir par exemple dans COLANI.LST.

Syntaxe : GOSUB gfa_poppal__

Paramètres :

Aucun

Valeur retournée :

lib_rv__\$ = string contenant la palette des couleurs ; la procédure POPPAL retourne, dans la variable string lib_rv__\$, la palette actuelle des couleurs ; le string contient 32 caractères soit 16 mots 'words' contenant les 16 couleurs. En haute résolution (ou moyenne), 12 (ou 14) couleurs sont inutilisées, ce qui n'est pas bien grave. Si vous recourez à XBIOS(6), il vous faut entrer la palette complète dans les trois résolutions, même si vous n'utilisez en fait que certains registres.

La palette est retournée dans le même format que celui utilisé avec la routine 6 du XBIOS ou avec la procédure x_CPOFF.

Description :

POPPAL vous permet d'identifier la palette des couleurs actuelle, par exemple pour la sauvegarder lors du lancement d'un programme et la restaurer à la fin à l'aide de XBIOS(6). Vous vous êtes certainement déjà énervé de voir votre bureau GEM devenu jaune-orangé (*et surtout illisible*) après avoir fait tourner un programme modifiant les couleurs à l'écran sans les restaurer ensuite. POPPAL permet d'écrire facilement des programmes

restaurant les couleurs d'origine sans avoir à les consigner dans le détail avant de lancer le programme.

Attention : ▽

Avant de recourir à d'autres procédures de notre bibliothèque, pensez à passer à une autre variable-string la palette des couleurs contenue dans `lib_rv__$`, car certaines procédures utilisent cette variable et détruiront donc sont contenu.

Exemple : Voir par exemple dans COLANI : EXAMPLES\PROGRAMS.
 LOW\COLANI.LST

Graphisme : PSHOW

Utilisation : Scrolling ou défilement d'images ou de 'morceaux' d'images ; il est possible d'afficher des images complètes ou des extraits de type Degas/Degas-Elite, Néochrome et 32000 octets.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\PSHOW.LST
(quelle que soit la résolution)

INLINE : Ne nécessite aucun fichier INLINE.

Exemples : EXAMPLES\PROGRAMS.LOW\PSHOW.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\PSHOW.LST
(haute résolution)

Syntaxe : GOSUB gfa_pshow__(screenadr__%,bufadr__%,txt__\$,
char__%,flag__!,options__%)

Paramètres :

- *screenadr__%* :

Contient l'adresse de la mémoire de l'écran sur lequel doivent s'afficher les images ou extraits d'images. En général, on entre ici XBIOS(2), qui est l'adresse de l'écran actuellement visible.

- *bufadr__%* :

Adresse d'un buffer de 32300 octets, qui sert à charger l'image et sa palette de couleurs sans qu'elles soient encore visibles, pour l'afficher ensuite proprement, sans scintillement à l'écran.

- *txt__\$* :

Bloc des paramètres de l'image ; ce string contient toutes les informations importantes concernant l'image à afficher. Pour que la procédure PSHOW

fonctionne correctement, le string `text__$` doit respecter un certain format, que nous allons détailler ci-dessous.

Structure de `text__$` :

Octets 1 - 67 : chemin d'accès de la première image à afficher en entier ou dont on souhaite afficher un extrait. Lorsque le chemin d'accès ne nécessite pas 67 octets, il faut y ajouter des espaces vides (ASCII 32) pour l'amener à 67 caractères.

Octets 68 - 69 : durée d'affichage de l'image ; cette valeur correspond au paramètre de l'instruction PAUSE du GfA Basic. La durée de l'affichage est cependant toujours plus longue qu'avec l'instruction PAUSE, car un certain délai est nécessaire pour le chargement et la 'préparation' de l'affichage de l'image suivante.

Octet 70 : type d'image ; cet octet sert à préciser le type de l'image qui va être chargée.

Valeurs autorisées :

- > 0 == image Degas/Degas-Elite
- > 1 == image Neochrome
- > 2 == image en format 32000 octets (*sans palette couleurs*)

Octet 71 : mode PUT ; cet octet sert à indiquer le mode d'affichage de l'image à l'aide de l'instruction PUT (*voir procédure POLICE*).

Octets 72-73 : coordonnée X du coin supérieur gauche de l'image à afficher.

Octets 74-75 : coordonnée Y du coin supérieur gauche

Octets 76-77 : coordonnée X du coin inférieur droit.

Octets 78-79 : coordonnée Y du coin inférieur droit.

Octets 80-81 : coordonnée X de la position d'affichage du coin supérieur gauche de l'image

Octets 82-83 : coordonnée Y de la position d'affichage du coin supérieur gauche de l'image

Octets 84-150 : chemin d'accès vers la deuxième image à afficher ou dont on souhaite extraire une partie pour affichage.

Comme vous le constatez, il convient d'entrer une suite de 83 octets pour toute image à afficher, en respectant scrupuleusement la répartition de ces 83 octets, faute de quoi la procédure ne pourra pas fonctionner correctement.

Vous pouvez écrire ce string en utilisant les instructions MKI\$ et CHR\$ du GfA Basic, qui génère des strings à partir de nombres sur un ou deux octets.

La procédure PSHOW peut faire défiler un nombre d'images quasiment illimité, qu'elle peut dénombrer en se fiant à la longueur du string txt_\$, puisque ce dernier s'accroît de 83 octets à chaque image. La longueur du string doit donc toujours être un multiple de 83.

La variable lib_rv_% (*code d'erreur*) retourne la valeur 1 lorsque la procédure repère une longueur de string incorrecte ou un string vide (se composant de caractères 0), et la valeur 0 lorsque tout se déroule correctement.

- *char_*%

code-touche de la touche d'interruption du processus : vous indiquez ici le code ASCII d'une touche sur laquelle l'utilisateur devra appuyer pour interrompre le défilement des images. Une fois la procédure lancée, le programme reste dedans jusqu'à ce que la touche en question soit appuyée.

Attention : ▽

Evitez d'entrer ici le code ASCII d'un caractère ou d'un signe inaccessible par le clavier ! Ne vous étonnez pas non plus de ce que l'exécution de la procédure ne s'interrompe pas immédiatement après un appui sur la touche désignée, car la procédure termine d'abord d'afficher l'image chargée avant de prendre fin : cependant, même s'il ne réagit pas immédiatement, il suffit d'un seul appui sur la touche en question pour interrompre l'exécution de la procédure.

-flag__!

Ce flag sert à préciser si l'écran doit se vider entièrement ou non avant de passer à l'affichage suivant : il est donc possible soit de provoquer un enchaînement bien délimité des images ou de faire 'monter' la nouvelle image sans effacer l'ancienne (sorte de fondu-enchaîné).

➤ **flag!__ == TRUE** = vider l'écran entre chaque affichage

➤ **flag!__ == FALSE** = ne pas vider l'écran

-options__% :

Ce paramètre peut servir à influencer la durée du défilement des images, puisqu'il peut prendre les valeurs suivantes :

➤ **options__% == 0 :**

Le défilement des images tourne dans une boucle sans fin, et ne peut être interrompu que par un appui sur la touche indiquée sous 'char__%'

➤ **options__% < 0 :**

lorsque 'options__%' reçoit une valeur négative, toutes les images viennent s'afficher une seule fois à l'écran, sans que l'on puisse interrompre ce défilement en appuyant sur la touche désignée sous char__% : tout appui sur cette touche sera ignoré par le programme. Le défilement prend fin une fois que toutes les images ont été affichées.

➤ **options__% > 0 :**

lorsque 'options__%' reçoit une valeur positive, toutes les images s'affichent une fois à l'écran, mais il est possible d'interrompre leur défilement en appuyant sur la touche désignée sous char__%.

Valeur retournée :

lib_rv__% = code d'erreur ; lib_rv__% retourne la valeur 1 lorsque le bloc des paramètres text__% n'est pas correct (*longueur égale à 0 ou non-multiple de 83*) et la valeur 0 lorsque tout se déroule correctement.

Description :

La procédure PSHOW vous permet de faire défiler plusieurs images l'une à la suite de l'autre. Non seulement vous pouvez mélanger des images de type différent, mais vous pouvez aussi afficher des morceaux d'une image à différents endroits de votre écran.

Qui plus est, vous pouvez influencer le mode d'affichage de chaque image, et faire varier la durée d'affichage image par image.

Exemple : EXAMPLES\PROGRAMS.LOW\PSHOW.LST

MERGE: GFA_GLIB.3_0\GFA_GLIB.ALL\PSHOW.LST
GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```
'
                                Buffer pour le 2ème écran
INLINE bufadr __%,32300
'
GOSUB gfa_poppal__
palette$=lib_rv__$
pblock$=""
                                ! écrire le string des paramètres
d'image
RESTORE parameterblock
READ path$
WHILE path$<>"FIN"
    pblock$=pblock$+path$+SPACE$(67-LEN(path$))
    READ speed$,typ|,pmode|,x1&,y1&,x2&,y2&,x3&,y3&
    pblock$=pblock$+MKI$(speed&)          ! durée de l'affichage
    pblock$=pblock$+CHR$(typ|)            ! type d'image
    pblock$=pblock$+CHR$(pmode|)          ! mode PUT pour les extraits d'image
    pblock$=pblock$+MKI$(x1&)+MKI$(y1&) ! coin sup. gauche de l'extrait
    pblock$=pblock$+MKI$(x2&)+MKI$(y2&) ! coin inf. droit
    pblock$=pblock$+MKI$(x3&)+MKI$(y3&) ! position de l'affichage
    READ path$
WEND
'
ALERT 1," Pour sortir de | la démonstration | appuyez sur 'a'
",1,"START",button|
'
GOSUB gfa_pshow__(XBIOS(2),bufadr __%,pblock$,ASC("a"),TRUE,0)
'
~XBIOS(6,L :V :palette$)
```

parameterblock :

```
DATA \EXAMPLES\STUFF\DEMO.PI1,150,0,3,00,00,319,199,000,000
DATA \EXAMPLES\STUFF\DEMO.PI1,100,0,3,80,50,239,149,080,050
DATA \EXAMPLES\STUFF\DEMO.PI1,200,0,3,80,50,239,149,000,000
DATA \EXAMPLES\STUFF\DEMO.PI1,150,0,3,80,50,239,149,160,000
DATA \EXAMPLES\STUFF\DEMO.PI1,100,0,3,80,50,239,149,160,100
DATA \EXAMPLES\STUFF\DEMO.PI1,050,0,3,80,50,239,149,000,100
DATA \EXAMPLES\STUFF\DEMO.PI1,000,0,3,00,00,319,035,000,082
DATA FIN
```

On commence par installer, à l'aide de INLINE, le buffer de 32300 octets nécessaire avec PSHOW, qui servira de deuxième écran.

POPPAL nous permet ensuite de sauvegarder la palette actuelle des couleurs, après quoi nous écrivons le string contenant les paramètres d'image. Vous pouvez prendre exemple sur le petit programme ci-dessus, qui vous montre une des méthodes possibles pour écrire facilement ce string et le faire varier par de légères modifications dans les lignes DATA. Notez que notre chemin d'accès vers l'image compte moins de 67 octets, et que nous le complétons donc par des espaces vides :

```
SPACE$(67-LEN(path$))
```

Dans notre démonstration, nous ne nous servons que d'une seule image (DEMO.PI1), dont nous affichons tour à tour différents extraits, mais nous pourrions naturellement faire défiler successivement plusieurs images différentes.

Une fois le bloc des paramètres écrit nous utilisons une boîte de dialogue (alertbox) pour signaler à l'utilisateur qu'il peut mettre fin au défilement des images en appuyant sur la touche 'a'.

Ceci étant, nous lançons la procédure PSHOW, dont l'exécution ne s'arrêtera que par un appui sur la touche 'a' (options__% = 0).

Avant de ressortir de notre programme, nous prenons soin de restaurer l'ancienne palette de couleurs.

Graphisme : INTANI

Utilisation : Animation d'une partie de l'écran (de taille variable) sous interruption.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\INTANI.LST
(quelle que soit la résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.ALL\INTANI.INL

Exemples : EXAMPLES\PROGRAMS.ALL\INTANI.LST

Syntaxe : GOSUB gfa_intani_(x_%,y_%,width_%,height_%,
speed_%,cnt_%,sourceadr_%,options_%)

Paramètres :

- x_% :

Coordonnée X correspondant au coin supérieur gauche de la partie de l'écran à animer ; cette valeur doit être un multiple de 16, et il convient de respecter les valeurs minimales et maximales que voici :

- > En basse résolution : 0 - 304
- > En moyenne résolution : 0 - 624
- > En haute résolution : 0 - 624

- y_% :

Coordonnée Y correspondant au coin supérieur gauche de la partie de l'écran à animer ; il convient de ne pas dépasser les valeurs minimales et maximales que voici :

- > En basse résolution : 0 - 199
- > En moyenne résolution : 0 - 199
- > En haute résolution : 0 - 399

Faites particulièrement attention à ne pas entrer des coordonnées situées au-delà des limites possibles de l'écran, faute de quoi vous auriez quelques surprises désagréables.

- *width*__% :

Largeur du secteur d'écran dans lequel se produit l'animation ; cette largeur doit aussi être un multiple de 16, et être mesurée en pixels.

- *height*__% :

Hauteur du secteur d'écran dans lequel se produit l'animation, mesurée en lignes de pixels ; pour que l'animation se déroule correctement, il faut évidemment que ses données entrent dans la largeur et la hauteur indiquées.

- *speed*__% :

Vitesse de défilement des images de l'animation ; vous indiquez ici le délai après lequel une nouvelle image vient remplacer l'ancienne ; l'animation fonctionne exactement comme un dessin animé, et revient à faire défiler une suite d'images fixes. *speed*__% sert à préciser le délai qui s'écoule entre l'affichage de deux images consécutives.

- *cnt*__% :

Nombre d'images composant l'animation ; ce nombre doit avoir une valeur inférieure d'une unité au nombre réel d'images et vous devez par exemple entrer le nombre 11 pour une animation faisant défiler 12 images. Attention, une animation doit comprendre au moins une image (*cnt*__% = 0), ce qui d'ailleurs vous donnera une image fixe. Il est interdit d'entrer ici une valeur négative!

*cnt*__% Doit être compris entre 0 et 65535.

- *sourceadr*__% :

Adresse des données mises en oeuvre par l'animation ; sous cette adresse doivent se trouver, dans la mémoire vive, les images composant l'animation. Chacune de ces images doit respecter les indications de taille (largeur et hauteur) entrées sous *width*__% et *height*__% ainsi que le format correspondant au degré de résolution de l'écran.

Les lignes de données d'une image sont entrées l'une derrière l'autre. La méthode la plus simple pour préparer une image consiste à recourir à

l'instruction GET du GfA Basic, qui sert précisément à inscrire sous forme de string dans la mémoire vive, les données correspondant à une image.

Comme l'animation consiste à faire défiler plusieurs images, il convient alors de relier plusieurs strings pour que les images soient enregistrées consécutivement dans la mémoire centrale. Attention : lorsque l'on utilise l'instruction GET, celle-ci intercale six octets avant les données proprement dites, octets qu'il convient d'éliminer (en recourant par exemple à l'instruction RIGHT\$) avant de relier les strings par l'opérateur '+'.

Il faut que vous sachiez que cette méthode ne donne pas des résultats irréprochables et qu'il serait préférable de procéder autrement à chaque fois que c'est possible. En effet, cette méthode présente un inconvénient majeur : après avoir relié les strings entre eux, vous ne pouvez plus les afficher à l'aide de l'instruction PUT ni les reprendre dans des opérations de manipulation de texte ; qui plus, vous êtes alors limité par la capacité maximale admise pour un string d'un seul tenant, qui est de 32767 octets.

options__% :

Code de commande qui permet de lancer ou d'arrêter l'animation :

- > options__% == 0 : arrêter l'animation
- > options__% <> 0 : lancer l'animation

Attention : ▽

vous ne pouvez arrêter l'animation que si vous l'avez lancée auparavant. Veillez en tout cas à bien arrêter l'animation à la fin du texte de votre programme, faute de quoi elle se poursuivrait au niveau du bureau GEM ce qui provoquerait un plantage du système.

Tous ces paramètres (en dehors de options__%) sont ignorés lorsque vous appelez INTANI pour mettre fin à une animation en cours, si bien que vous pouvez leur donner une valeur quelconque.

Valeur retournée :

lib_rv__% = code d'erreur ; lib_rv__% retourne la valeur -1 lorsque vous oubliez de charger le fichier INLINE accompagnant INTANI et la valeur 0 lorsque l'exécution du module se déroule normalement.

Description :

Le module INTANI vous permet de créer des animations d'images tournant sous interruption, sur une portion de votre écran. Ces animations ressemblent à des dessins animés, puisqu'elles consistent à faire défiler des images ou des morceaux d'images sur une portion donnée de l'écran, ce qui peut donner une impression de mouvement.

Une fois la dernière image affichée, le module enchaîne en revenant sur la première image. Toutes les images viennent s'afficher sur la même partie de l'écran, INTANI ne vous permettant pas de 'promener' la même image à des endroits différents de l'écran (exemple : la soucoupe volante parcourant l'écran). Ce module est particulièrement recommandé pour animer des images d'en-tête assez complexes.

Exemple : EXAMPLES\PROGRAMS.ALL\INTANI.LST

MERGE : GFA_GLIB.3_0\GFA_GLIB.ALL\INTANI/LST

```
PRINT AT(1,1) ;"BM BM BM"                   ! Affichage des images de
PRINT AT(1,2) ;"           BM BM BM"       ! l'animation à l'écran
PRINT AT(1,3) ;"BM BM BM"
PRINT AT(1,4) ;"           BM BM BM"
PRINT AT(1,5) ;"BM BM BM"
PRINT AT(1,6) ;"           BM BM BM"
PRINT AT(1,7) ;"BM BM BM"
PRINT AT(1,8) ;"           BM BM BM"
BOX 0,0,63,63
BOX 64,0,127,63
GET 0,0,63,63,donnees1$                   ! pour créer les données
GET 64,0,127,63,donnees2$               ! correspondant à l'animation
donnees1$=donnees1$+RIGHT$(donnees2$,LEN(donnees2$)-6)
,
GOSUB gfa_intani__(16,130,64,64,40,1,V :donnees1$+6,1)
,
ALERT 2," | stopper | l'animation? | ",1," OUI ",button|
,
GOSUB gfa_intani__(0,0,0,0,0,0,0,0)
```

Nous commençons par créer un modèle très simple à l'écran, qui va nous servir à engendrer les deux petites images de cette animation ; nous utilisons l'instruction GET pour transformer ces deux images en deux

Graphisme : L_SHAP16 M_SHAP16 H_SHAP16

Utilisation : Créer des sprites ('shape') d'une largeur de 16 pixels et d'une hauteur maximale de 200 pixels (L_SHAP16 et M_SHAP16) ou de 400 pixels (H_SHAP16).

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_SHAP16.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_SHAP16.LST
(moyenne résolution)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_SHAP16.LST
(haute résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_SHAP16.INL
(L_SHAP16.LST)

GFA_GLIB.3_0\GFA_GLIB.MED\M_SHAP16.INL
(M_SHAP16.LST)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_SHAP16.INL
(H_SHAP16.LST)

Exemples : EXAMPLES\PROGRAMS.LOW\L_SHAP16.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\H_SHAP16.LST
(haute résolution)

Syntaxe : GOSUB gfa_l_shap16__(x__%,y__%,options__%,
rows__%,screenadr__%,bufadr__%,memadr__%)

GOSUB gfa_m_shap16__(x__%,y__%,options__%,
rows__%,screenadr__%,bufadr__%,memadr__%)

GOSUB gfa_h_shap16__(x__%,y__%,options__%,
rows__%,screenadr__%,bufadr__%,memadr__%)

Paramètres :

- $x_%$:

Coordonnée X de l'endroit où doit venir s'afficher le shape (sprite graphique). Celui-ci peut prendre n'importe quelle position de coordonnée X, ce qui revient à dire qu'il peut aussi 'sortir' partiellement ou complètement de l'écran.

- $y_%$:

Coordonnée Y de l'endroit où doit venir s'afficher le sprite ; là aussi, il peut prendre n'importe quelle position, mais il convient de veiller à ce qu'il ne sorte pas du tout (ni partiellement ni complètement) du bord supérieur et inférieur de l'écran, faute de quoi l'utilisateur serait amené à écrire dans la mémoire en-dessous ou au-dessus de l'écran.

Si cela s'avère vraiment indispensable, il est cependant possible de faire franchir le bord supérieur de l'écran au sprite, en adaptant en conséquence l'adresse de début de ses données ($memadr_%$) ainsi que sa hauteur ($rows_%$). Par exemple, lorsque seules les dix dernières lignes composant un sprite doivent rester visibles, il faut indiquer sous $memadr_%$ l'adresse des données correspondant à cette dixième ligne et entrer 10 sous $rows_%$.

Pour calculer ' $memadr_%$ ', on peut recourir aux formules suivantes :

➤ *basse résolution* :

$memadr_% =$ Adresse de début des données + 8 * numéro de la première ligne du sprite

➤ *moyenne résolution* :

$memadr_% =$ Adresse de début des données + 4 * numéro de la première ligne visible du sprite

➤ *haute résolution* :

$memadr_% =$ Adresse de début des données + 2 * numéro de la première ligne visible du sprite

Notez que la première ligne du sprite porte le numéro zéro.

strings que nous allons relier en veillant à éliminer les six octets figurant au début du deuxième string puisqu'ils ne contiennent aucune information relative à l'image elle-même. Nous obtenons ensuite un string unique, réunissant les données relatives à nos deux images, enregistrées dans la mémoire directement à la suite l'une de l'autre.

Nous appelons alors le module INTANI ; l'animation doit venir s'afficher sur la portion d'écran délimitée par x__%,y__% (16,130), les images ayant une hauteur de 64 pixels et une largeur de 64 pixels. Remarquez que nous avons pris soin d'enregistrer une image de 64 x 64 pixels lorsque nous avons fait par deux fois appel à l'instruction GET, pour que les données de notre animation aient le bon format.

Nous avons choisi une vitesse (speed__%) de 40, tandis que le nombre d'images est représenté par cnt__%=1 puisque nous ne jouons que sur deux images (cnt__% = nombre d'images - 1).

Sourcadr__% contient l'adresse des données de la première image ; il convient d'ignorer les 6 premiers octets puisque nous ne les avons retranchés que du string de la deuxième image.

Le paramètre 'options__%' est mis sur 1 pour que l'animation puisse être lancée. Elle tourne alors jusqu'à ce que le module INTANI soit relancé mais avec le paramètre 'options' mis sur 0. Il est possible d'exécuter d'autres instructions du GfA Basic pendant que l'animation tourne. Dans notre petit programme de démonstration, on se limite à attendre un clic sur le bouton 'OUI' de la boîte de dialogue : le module INTANI est alors appelé pour mettre fin à l'animation et au programme de démonstration.

Lorsque le sprite doit pouvoir franchir le bord inférieur de l'écran, il suffit d'adapter la valeur de `rows__%` et d'indiquer le nombre de lignes du sprite qui restent effectivement visibles.

- `options__%` :

Ce code de commande `options__%` précise si le sprite doit s'afficher ou disparaître de l'écran, selon le code suivant :

- > `options__% == 0` : sauver l'arrière-fonds et afficher le sprite
- > `options__% <> 0` : restaurer l'arrière-fonds et faire disparaître le sprite

Attention : ▽

Il faut veiller à ce que le sprite se soit bien affiché avant de restaurer l'arrière-fonds (et non l'inverse) ; qui plus est, vous pouvez utiliser autant de sprites que vous voulez (seule limite : la capacité de la mémoire) mais vous devez veiller à ce que le dernier sprite apparu soit aussi le premier effacé ; faute de quoi l'arrière-fond ne sera pas restauré correctement (principe dit 'LIFO' = Last In First Out = le dernier entré sera le premier à sortir).

- `rows__%` :

Hauteur du sprite, mesurée en lignes-pixels ; la hauteur maximale tolérée dépend du degré de résolution de l'écran :

- > **basse résolution :** 1 - 200 lignes
- > **moyenne résolution :** 1 - 200 lignes
- > **haute résolution :** 1- 400 lignes

- `screenadr__%` :

Adresse d'écran ; il convient d'indiquer ici l'adresse de début de la mémoire d'écran, qui correspond généralement à XBIOS(2).

Pour que le sprite n'ait pas l'air de trembloter, il est recommandé de travailler avec deux écrans : pendant qu'une des images s'affiche à l'écran, l'ordinateur élabore l'image suivante (invisible car se trouvant dans la mémoire). Une fois ce processus terminé, la deuxième image vient prendre la place de la première à l'écran etc... Cela permet d'éviter le phénomène de

scintillement ou de tremblement qui résulterait de l'affichage et de la disparition d'un ou plusieurs sprites directement à l'écran.

C'est pourquoi il est possible ici d'entrer une deuxième adresse correspondant à un deuxième écran : le sprite s'élabore d'abord sur ce deuxième écran invisible et non directement à l'écran (ne pas oublier d'en profiter pour recopier ce contenu d'écran).

- **bufadr__%** :

Vous devez entrer ici l'adresse d'un buffer suffisamment important pour pouvoir contenir l'arrière-plan recouvert par le sprite. Ce buffer est indispensable, car le principe même du module 'shape' consiste à sauvegarder dans un buffer le secteur d'écran qui va être recouvert par le sprite avant que ce dernier ne s'affiche. Cet arrière-plan est restauré dès que le sprite se déplace, le processus se répétant ensuite etc etc...

Voici des formules vous permettant de calculer la taille que doit avoir au minimum le buffer en question, vous ne devez en aucun cas lui attribuer une taille inférieure :

- **basse résolution :** taille du buffer = rows__% * 16 + 4
- **moyenne résolution :** taille du buffer = rows__% * 8 + 4
- **haute résolution :** taille du buffer = rows__% * 4 + 4

Lorsque vous vous servez de plusieurs sprites, il convient d'installer un buffer pour chacun d'entre eux.

- **memadr__%** :

Adresse des données correspondant au sprite : c'est la façon la plus simple de créer un sprite, de le 'teindre' de différentes couleurs puis de l'afficher à l'écran. En GFA²Basic, il suffit de recourir à l'instruction GET x,y1,x+15,y2,shape\$ pour inscrire ces données dans un string (shape\$) : les données du sprite sont ainsi directement chargées en mémoire dans le bon format (là où se trouve le string).

Voici la formule permettant de calculer memadr__% :

memadr__%=V:shape\$+6

Il est indispensable d'additionner 6, car le GfA Basic enregistre la hauteur et la largeur du secteur ainsi que le nombre de plans binaires devant les données proprement dites.

Si vous utilisez cette méthode, faites bien attention à ce que le secteur lu à l'aide de l'instruction GET ait effectivement une largeur de 16 pixels, ni plus, ni moins.

- *sourceadr*__% :

Adresse du masque du sprite ; vous n'entrez ce paramètre que dans la procédure H_SHAP16 (haute résolution). La variable *sourceadr*__% contient l'adresse de début d'un masque de sprite ; en haute résolution, ce masque sert à préciser quels secteurs de l'arrière-plan sont purement et simplement recouverts par le sprite et quels secteurs sont par contre visibles par 'transparence'.

La structure de ce masque ressemble fortement à celle du masque de la flèche de la souris, que vous créez en GfA Basic à l'aide de l'instruction DEFMOUSE.

Le masque du sprite prend exactement la même taille que le sprite lui-même et peut aussi venir s'inscrire dans un string (donc en mémoire) en utilisant l'instruction GET. Tout point du sprite (qu'il soit blanc ou noir) appelé à recouvrir et dissimuler l'arrière-plan doit être identifié dans le masque par un bit vrai (1) ; par contre, les points devant laisser transparaître l'arrière-plan seront identifiés par des bit faux (0).

Vous trouverez dans le dossier EXAMPLES\STUFF une image DEMO.PIC vous montrant l'aspect que peut prendre un sprite et son masque. Ce dossier contient 40 objets graphiques pouvant servir de sprite, flanqués de leur 40 masques.

La disquette contient aussi un programme de démonstration pour les moniteurs haute-résolution

(voir EXAMPLES\PROGRAMS.HIG\H_SHAP16.LST).

Tout cela suffit pour vous faire comprendre concrètement comment fonctionne cette procédure 'shape'.

Valeur retournée :

lib_rv__% = code d'erreur

lib_rv__% retourne la valeur -1 lorsque vous oubliez de charger le fichier **INLINE** accompagnant la procédure et 0 lorsque tout se déroule normalement.

Description :

Les modules **L_SHAP16**, **M_SHAP16** et **H_SHAP16** vous permettent de créer des sprites d'une largeur de 16 pixels ; contrairement aux limites imposées par l'instruction **SPRITE**, ceux-ci peuvent ici prendre quasiment n'importe quelle hauteur et contenir toutes les couleurs offertes par le degré de résolution caractérisant le moniteur.

Attention : ▽

Souvenez-vous qu'il convient d'entrer un paramètre de plus lorsque vous utilisez **H_SHAP16** (sourceadr__%) ; ce paramètre indique l'adresse du masque du sprite, qui n'est pas engendré automatiquement par le module contrairement à ce qui se passe sous **L_SHAP16** et **M_SHAP16**.

Comme l'utilisation d'un tel sprite joue beaucoup sur les délais d'affichage, il est sans doute avantageux d'appeler immédiatement la routine en assembleur. Ceci accélère considérablement le processus et rend l'affichage du sprite plus 'souple' puisqu'on peut omettre de sauvegarder l'arrière-plan. C'est de plus intéressant lorsque l'on souhaite restaurer l'arrière-plan en le faisant varier (par exemple au moyen d'une animation).

Voici dans ce cas la syntaxe pour lancer **L_SHAP16** :

```
~C : l_shap16mc__% (x__%,y__%,options__%,rows__%, L :screenadr__%,  
                  L : bufadr__%,L : memadr__%)
```

pour lancer **M_SHAP16** :

```
~C : m_shap16mc__% (x__%,y__%,options__%,rows__%, L :screenadr__%,  
                  L : bufadr__%,L : memadr__%)
```

et pour lancer **H_SHAP16** :

~C : h_shap16mc__% (x__%,y__%,options__%,rows__%,L : screenadr__%,
L : bufadr__%,L : memadr__%)

Les paramètres sont exactement les mêmes que lorsque vous lancez la procédure, à l'exception de options__%, qui peut prendre les valeurs suivantes :

- > options__% == 0 : GETBACK, sauvegarder l'arrière-plan
- > options__% == 1 : PUTBACK, restaurer l'arrière-plan
- > options__% == 2 : SETSHAPE, activer le sprite.

Toute autre valeur sera ignorée ; voici donc la succession logique pour activer un sprite et le faire disparaître :

- > D'abord options__% = 0 ==> GETBACK
- > Ensuite options__% = 2 ==> SETSHAPE
- > Enfin options__% = 1 ==> PUTBACK

Ce qui nous donne trois appels en tout, mais il est possible de laisser tomber GETBACK et PUTBACK lorsqu'on veut restaurer l'arrière-fond en le modifiant.

Attention : ∇

PUTBACK ne doit en principe intervenir qu'après un appel de GETBACK.

Notez cependant que, pour qu'il soit possible de lancer directement le module SHAPE, il faut dès le début du programme prévoir le lancement de ce module. En clair : avant de lancer la routine en assembleur à l'aide de la commande ~C : ... il faut avoir appelé au moins une fois le module en recourant à l'instruction GOSUB, faute de quoi l'adresse start de la routine en assembleur (l_shap16mc__%, m_shap16mc__%,h_shap16mc__%) est encore sur 0 et l'appel direct mène tout droit à une erreur de bus 'BUSERROR' (deux petites bombes à l'écran!). Une fois le module chargé, l'adresse start de la routine en assembleur est précisée, et vous pouvez donc la lancer à l'aide de ~C :

Exemple : EXAMPLES\PROGRAMS.LOW\L_SHAP16.LST

MERGE: GFA_GLIB.3_0\GFA_GLIB.LOW\L_SHAP16.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\LOAD.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```
'          Dimensionner le buffer pour sauvegarder
'          L'arrière-plan derrière le sprite
INLINE bufadr __%,260
'
GOSUB gfa_poppal__
Palette$=lib_rv__$
GOSUB gfa_pload__("\EXAMPLES\STUFF\DEMO.PI1",XBIOS(2),0)
'
GET 0,136,15,151,shape$ !inscrire le sprite dans un string
ALERT 2,"Appuyez sur une |touche quelconque pour|interrompre la
d mo",1,"Start",button|
x&=0
y&=136
xi&=1
yi&=-1
HIDEM
REPEAT
'
GOSUB gfa_l_shap16__(x&,y&,0,16,XBIOS(2),bufadr __%,V:shape$+6)
ADD x&,xi&
ADD y&,yi&
IF x&>293 OR x&<1
    MUL xi&,-1
ENDIF
IF y&>183 OR y&<124
    MUL yi&,-1
ENDIF
VSYNC
'
GOSUB gfa_l_shap16__(0,0,1,16,XBIOS(2),bufadr __%,V:shape$+6)
'
UNTIL INKEY$<>""
SHOWM
~XBIOS(6,L :V:palette$)
```


A l'aide de `INLINE`, nous commençons par créer un buffer destiné à recevoir le contenu de l'arrière-plan et nous calculons sa taille à l'aide de la formule indiquée ci-dessus. Comme notre sprite a une hauteur de 16 lignes, nous multiplions la constante 16 par 16, et nous ajoutons 4.

Une fois le buffer installé et son adresse de début déterminée (`bufadr__%`), nous sauvegardons la palette actuelle des couleurs à l'aide du module `POPPAL`, dans la variable-string `palette$`. Nous chargeons alors l'image `Degas DEMO.PI1` ainsi que sa palette de couleurs à l'aide du module `PLOAD`. L'image sert ici de graphique d'arrière-plan.

L'instruction `GET` du `GfA Basic` nous permet d'inscrire dans le string `shape$` un secteur d'écran de 16 x 16 pixels, secteur qui va nous servir pour notre sprite. Après avoir répondu par l'affirmative à la demande de confirmation s'affichant à l'écran, nous fixons les coordonnées se rapportant au premier appel de `L_SHAP16` et nous entrons dans une boucle `REPEAT...UNTIL`.

A l'intérieur de cette boucle, nous commençons par afficher le sprite, ce qui provoque automatiquement la sauvegarde de l'arrière-plan (Attention! si vous appelez directement la routine en assembleur, la sauvegarde de l'arrière-plan ne se fait pas automatiquement, il faut ajouter un appel supplémentaire).

Lors de ce premier appel de `L_SHAP16`, `options__%` est sur 0, et nous entrons `XBIOS(2)` comme adresse d'image, si bien que le sprite va immédiatement s'afficher sur l'écran actuellement visible. Nous avons donné `bufadr__%` comme adresse du buffer. L'adresse des données du sprite (`memadr__%`) a été calculée par `V: shape$+6`.

Nous calculons alors les nouvelles coordonnées et effaçons le sprite (restauration de l'arrière-plan qui réapparaît) : comme nous n'avons plus besoin des coordonnées à ce stade là, nous avons mis sur 0 les paramètres `x__%` et `y__%`.

Ce processus (affichage/disparition du sprite) se répète sans arrêt jusqu'à ce que l'utilisateur l'interrompe en appuyant sur une touche quelconque. Lorsque ceci se produit, les couleurs d'origine réapparaissent (`XBIOS 6`) et le programme de démonstration prend fin.

Comme l'affichage et la disparition du sprite se produisent directement sur l'écran en cours (écran visible), nous utilisons l'instruction `VSynch` du `GfA`

Basic juste avant le troisième appel de L_SHAP16 pour réduire l'effet de tremblement ou de scintillement.

Graphisme : L_SHAP32 M_SHAP32 H_SHAP32

Utilisation : Créer des sprites ('shape') d'une largeur de 32 pixels et d'une hauteur maximale de 200 pixels (L_SHAP32 et M_SHAP32) ou de 400 pixels (H_SHAP32).

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_SHAP32.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_SHAP32.LST
(moyenne résolution)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_SHAP32.LST
(haute résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_SHAP32.INL
(L_SHAP32.LST)

GFA_GLIB.3_0\GFA_GLIB.MED\M_SHAP32.INL
(M_SHAP32.LST)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_SHAP32.INL
(H_SHAP32.LST)

Exemples : EXAMPLES\PROGRAMS.LOW\L_SHAP32.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\H_SHAP32.LST
(haute résolution)

Syntaxe : GOSUB gfa_l_shap32__(x__%,y__%,options__%,
rows__%,screenadr__%,bufadr__%,memadr__%)

GOSUB gfa_m_shap32__(x__%,y__%,options__%,
rows__%,screenadr__%,bufadr__%,memadr__%)

GOSUB gfa_h_shap32__(x__%,y__%,options__%,
rows__%,screenadr__%,bufadr__%,memadr__%)

Paramètres :

- x__% :

Voir L_SHAP16 et M_SHAP16.

- y__% :

Coordonnée Y de l'endroit où doit venir s'afficher le sprite ; là aussi, il peut prendre n'importe quelle position, mais il convient de veiller à ce qu'il ne sorte pas du tout (ni partiellement ni complètement) du bord supérieur et inférieur de l'écran, faute de quoi l'utilisateur serait amené à écrire dans la mémoire en-dessous ou au-dessus de l'écran.

Si cela s'avère vraiment indispensable, il est cependant possible de faire franchir le bord supérieur de l'écran au sprite, en modifiant en conséquence l'adresse de début de ses données (memadr__%) ainsi que sa hauteur (rows__%). Par exemple, lorsque seules les dix dernières lignes composant un sprite doivent rester visibles, il faut indiquer sous memadr__% l'adresse des données correspondant à cette dixième ligne et entrer 10 sous rows__%.

Pour calculer 'memadr__%', on peut recourir aux formules suivantes :

- basse résolution : $\text{memadr_}\% = \text{adresse de début des données} + 16 * \text{numéro de la première ligne du sprite}$
- moyenne résolution : $\text{memadr_}\% = \text{adresse de début des données} + 8 * \text{numéro de la première ligne visible du sprite}$
- haute résolution : $\text{memadr_}\% = \text{adresse de début des données} + 4 * \text{numéro de la première ligne visible du sprite}$

Notez que la première ligne du sprite porte le numéro zéro.

Lorsque le sprite doit pouvoir franchir le bord inférieur de l'écran, il suffit d'adapter la valeur de rows__% et d'indiquer le nombre de lignes du sprite qui restent effectivement visibles.

- **options** __% : voir L_SHAP16, M_SHAPE16
- **rows** __% : voir L_SHAP16, M_SHAPE16

➤ **screenadr** __% : voir L_SHAP16, M_SHAPE16

- **bufadr** __% :

Vous devez entrer ici l'adresse d'un buffer suffisamment important pour pouvoir contenir l'arrière-plan recouvert par le sprite. Ce buffer est indispensable, car le principe même du module 'shape' consiste à sauvegarder dans un buffer le secteur d'écran qui va être recouvert par le sprite avant que ce dernier ne s'affiche. Cette arrière-plan est restauré dès que le sprite se déplace, le processus se répétant ensuite etc etc.

Voici des formules vous permettant de calculer la taille que doit avoir au minimum le buffer en question, vous ne devez en aucun cas lui attribuer une taille inférieure :

➤ **basse résolution** : taille du buffer = rows__% * 24 + 4

➤ **moyenne résolution** : taille du buffer = rows__% * 12 + 4

➤ **haute résolution** : taille du buffer = rows__% * 6 + 4

Lorsque vous vous servez de plusieurs sprites, il convient d'installer un buffer pour chacun d'entre eux.

- **memadr** __% :

Adresse des données correspondant au sprite : c'est la façon la plus simple de créer un sprite, de le 'teindre' de différentes couleurs puis de l'afficher à l'écran. En GfA Basic, il suffit de recourir à l'instruction GET x,y1,x+31,y2,shape\$ pour inscrire ces données dans un string (shape\$) : les données du sprite sont ainsi directement chargées en mémoire dans le bon format (là où se trouve le string).

Voici la formule permettant de calculer memadr__% :

memadr __% = V: shape\$+6

Il est indispensable d'additionner 6, car le GfA Basic enregistre la hauteur et la largeur du secteur ainsi que le nombre de plans binaires devant les données proprement dites.

Si vous utilisez cette méthode, faites bien attention à ce que le secteur lu à l'aide de l'instruction GET ait effectivement une largeur de 32 pixels, ni plus, ni moins.

- *sourceadr__%* :

Adresse du masque du sprite ; vous n'entrez ce paramètre que dans la procédure H_SHAP32 (haute résolution). La variable *sourceadr__%* contient l'adresse de début d'un masque de sprite ; en haute résolution, ce masque sert à préciser quels secteurs de l'arrière-plan sont purement et simplement recouverts par le sprite et quels secteurs sont par contre visibles par 'transparence'.

La structure de ce masque ressemble fortement à celle du masque de la flèche de la souris, que vous créez en GfA Basic à l'aide de l'instruction DEFMOUSE. Le masque du sprite prend exactement la même taille que le sprite lui-même et peut aussi venir s'inscrire dans un string (donc en mémoire) en utilisant l'instruction GET. Tout point du sprite (qu'il soit blanc ou noir) appelé à recouvrir et dissimuler l'arrière-plan doit être identifié dans le masque par un bit vrai (1) ; par contre, les points devant laisser transparaître l'arrière-plan seront identifiés par des bit faux (0).

Vous trouverez dans le dossier EXAMPLES\STUFF une image DEMO.PIC vous montrant l'aspect que peut prendre un sprite et son masque. Ce dossier contient 40 objets graphiques pouvant servir de sprite, flanqués de leur 40 masques.

La disquette contient aussi un programme de démonstration pour les moniteurs haute-résolution (voir EXAMPLES\PROGRAMS.HIG\H_SHAP32.LST).

Tout cela suffit pour vous faire comprendre concrètement comment fonctionne ce module 'shape'.

Valeur retournée :

lib_rv__% =

Code d'erreur ; *lib_rv__%* retourne la valeur -1 lorsque vous oubliez de charger le fichier INLINE accompagnant le module et 0 lorsque tout se déroule normalement.

Description :

Les modules L_SHAP32, M_SHAP32 et H_SHAP32 vous permettent de créer des sprites d'une largeur de 32 pixels ; contrairement aux limites imposées par l'instruction SPRITE, le sprite peut ici prendre quasiment n'importe quelle hauteur et contenir toutes les couleurs offertes par le degré de résolution caractérisant le moniteur.

Attention : ▽

Souvenez-vous qu'il convient d'entrer un paramètre de plus lorsque vous utilisez H_SHAP32 (*sourceadr__%*) ; ce paramètre indique l'adresse du masque du sprite, qui n'est pas engendré automatiquement par le module contrairement à ce qui se passe sous L_SHAP32 et M_SHAP32.

Comme l'utilisation d'un tel sprite joue beaucoup sur les délais d'affichage, il est sans doute avantageux d'appeler immédiatement la routine en assembleur. Ceci accélère considérablement le processus et rend l'affichage du sprite plus 'souple' puisqu'on peut omettre de sauvegarder l'arrière-plan. C'est de plus intéressant lorsque l'on souhaite restaurer l'arrière-plan en le faisant varier (par exemple au moyen d'une animation).

Voici dans ce cas la syntaxe pour lancer L_SHAP32 :

```
~C :l_shap32mc__(x__%,y__%,options__%,rows__%,L :screenadr__%,  
L :bufadr__%,L :memadr__%)
```

pour lancer M_SHAP32 :

```
~C :m_shap32mc__(x__%,y__%,options__%,rows__%,L :screenadr__%,  
L :bufadr__%,L :memadr__%)
```

et pour lancer H_SHAP32 :

```
~C :h_shap32mc__(x__%,y__%,options__%,rows__%,  
L :screenadr__%,L :bufadr__%,L :memadr__%)
```

Les paramètres sont exactement les mêmes que lorsque vous lancez la procédure, à l'exception de *options__%*, qui peut prendre les valeurs suivantes :

➤ *options__%* == 0 : GETBACK, sauvegarder l'arrière-pla

➤ *options__%* == 1 : PUTBACK, restaurer l'arrière-plan

➤ **options__%** == 2 : SETSHAPE, activer le sprite.

Toute autre valeur sera ignorée ; voici donc la succession logique pour activer un sprite et le faire disparaître :

➤ D'abord **options__%** = 0 ==> GETBACK

➤ Ensuite **options__%** = 2 ==> SETSHAPE

➤ Enfin **options__%** = 1 ==> PUTBACK

Ce qui nous donne trois appels en tout, mais il est possible de laisser tomber GETBACK et PUTBACK lorsqu'on veut restaurer l'arrière-plan en le modifiant. Attention : PUTBACK ne doit en principe intervenir qu'après un appel de GETBACK.

Notez cependant que, pour qu'il soit possible de lancer directement le module SHAPE, il faut dès le début du programme prévoir le lancement de ce module. En clair : avant de lancer la routine en assembleur à l'aide de la commande ~C :... il faut avoir appelé au moins une fois le module en recourant à l'instruction GOSUB, faute de quoi l'adresse start de la routine en assembleur (**l_shap32mc__%**, **m_shap32mc__%**, **h_shap32mc__%**) est encore sur 0 et l'appel direct mène tout droit à une erreur de bus 'BUSERROR' (*deux petites bombes à l'écran!*).

Une fois le module chargé, l'adresse start de la routine en assembleur est précisée, et vous pouvez donc la lancer à l'aide de ~C :

Exemple : **EXAMPLES\PROGRAMS.LOW\L_SHAP32.LST**

MERGE : **GFA_GLIB.3_0\GFA_GLIB.LOW\L_SHAP32.LST**

GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```
'                                     Dimensionner le buffer pour sauvegarder
'                                     L'arrière-plan derrière le sprite
```

```
(16*24+4)
```

```
INLINE bufadr__%,388
```

```
,
```

```
GOSUB gfa_poppal__
```

```
palette$=lib_rv__$
```

```
GOSUB gfa_pload__ ("EXAMPLES\STUFF\DEMO.PI1",XBIOS(2),0)
```

```
,
```



```

GET 0,136,31,151,shape$           !inscrire le sprite dans un string
ALERT 2,"Appuyez sur une |touche quelconque pour|interrompre la
d mo",1,"Start",button|
x&=0
y&=136
xi&=1
yi&=-1
HIDEM
REPEAT
    ,
    GOSUB gfa_1_shap32__(x&,y&,0,16,XBIOS(2),bufadr__%,V :shape$+6)
    ,
    ADD x&,xi&
    ADD y&,yi&
    IF x&>287 OR x&<1
        MUL xi&,-1
    ENDIF
    IF y&>183 OR y&<124
        MUL yi&,-1
    ENDIF
    VSYNC
    ,
    GOSUB gfa_1_shap32__(0,0,1,16,XBIOS(2),bufadr__%,V :shape$+6)
    ,
UNTIL INKEY$<>""
SHOWM
~XBIOS(6,L :V :palette$)

```

Ce programme de d monstration est similaire   celui de L_SHAP16, la seule diff rence r sidant dans le calcul de la taille du buffer destin    contenir l'arri re-plan et dans le fait que l'instruction GET concerne un sprite de 32 pixels de large pour 16 pixels de haut.

La taille du buffer est   nouveau calcul e   l'aide de la formule indiqu e ci-dessus. Comme notre sprite a une hauteur de 16 lignes, nous multiplions la constante 24 par 16, et nous ajoutons 4. Le reste de la d monstration est rigoureusement semblable   la d monstration de L_SHAP16.

Graphisme : L_SHAP64 M_SHAP64 H_SHAP64

Utilisation : Créer des sprites ('shape') d'une largeur de 64 pixels et d'une hauteur maximale de 200 pixels (L_SHAP64 et M_SHAP64) ou de 400 pixels (H_SHAP64).

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_SHAP64.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_SHAP64.LST
(moyenne résolution)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_SHAP64.LST
(haute résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_SHAP64.INL
(L_SHAP64.LST)

GFA_GLIB.3_0\GFA_GLIB.MED\M_SHAP64.INL
(M_SHAP64.LST)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_SHAP64.INL
(H_SHAP64.LST)

Exemples : EXAMPLES\PROGRAMS.LOW\L_SHAP64.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\H_SHAP64.LST
(haute résolution)

Syntaxe : GOSUB gfa_l_shap64__(x__%,y__%,options__%,
rows__%, screenadr__%,bufadr__%,memadr__%)

GOSUB gfa_m_shap64__(x__%,y__%,options__%,
rows__%, screenadr__%,bufadr__%,memadr__%)

GOSUB gfa_h_shap64__(x__%,y__%,options__%,
rows__%, screenadr__%,bufadr__%,memadr__%)

Paramètres :

- **x__%** :

Voir **L_SHAP16** et **M_SHAP16**.

- **y__%** :

Coordonnée Y de l'endroit où doit venir s'afficher le sprite ; là aussi, il peut prendre n'importe quelle position, mais il convient de veiller à ce qu'il ne sorte pas du tout (ni partiellement ni complètement) du bord supérieur et inférieur de l'écran, faute de quoi l'utilisateur serait amené à écrire dans la mémoire en-dessous ou au-dessus de l'écran.

Si cela s'avère vraiment indispensable, il est cependant possible de faire franchir le bord supérieur de l'écran au sprite, en modifiant en conséquence l'adresse de début de ses données (**memadr__%**) ainsi que sa hauteur (**rows__%**). Par exemple, lorsque seules les dix dernières lignes composant un sprite doivent rester visibles, il faut indiquer sous **memadr__%** l'adresse des données correspondant à cette dixième ligne et entrer 10 sous **rows__%**.

Pour calculer 'memadr__%', on peut recourir aux formules suivantes :

- basse résolution : $\text{memadr_}\% = \text{adresse de début des données} + 32 * \text{numéro de la première ligne du sprite}$
- moyenne résolution : $\text{memadr_}\% = \text{adresse de début des données} + 16 * \text{numéro de la première ligne visible du sprite}$
- haute résolution : $\text{memadr_}\% = \text{adresse de début des données} + 8 * \text{numéro de la première ligne visible du sprite}$

Notez que la première ligne du sprite porte le numéro zéro.

Lorsque le sprite doit pouvoir franchir le bord inférieur de l'écran, il suffit d'adapter la valeur de **rows__%** et d'indiquer le nombre de lignes du sprite qui restent effectivement visibles.

- **options** __% : voir **L_SHAP16**, **M_SHAPE16**
- **rows** __% : voir **L_SHAP16**, **M_SHAPE16**
- **screenadr** __% : voir **L_SHAP16**, **M_SHAPE16**

- *bufadr*__% :

Vous devez entrer ici l'adresse d'un buffer suffisamment important pour pouvoir contenir l'arrière-plan recouvert par le sprite. Ce buffer est indispensable, car le principe même du module 'shape' consiste à sauvegarder dans un buffer le secteur d'écran qui va être recouvert par le sprite avant que ce dernier ne s'affiche. Cet arrière-plan est restauré dès que le sprite se déplace, le processus se répétant ensuite etc etc.

Voici des formules vous permettant de calculer la taille que doit avoir au minimum le buffer en question, vous ne devez en aucun cas lui attribuer une taille inférieure :

- > **basse résolution** : taille du buffer = rows__% * 40 + 4
- > **moyenne résolution** : taille du buffer = rows__% * 20 + 4
- > **haute résolution** : taille du buffer = rows__% * 10 + 4

Lorsque vous vous servez de plusieurs sprites, il convient d'installer un buffer pour chacun d'entre eux.

- *memadr*__% :

Adresse des données correspondant au sprite : c'est la façon la plus simple de créer un sprite, de le 'teindre' de différentes couleurs puis de l'afficher à l'écran. En GfA Basic, il suffit de recourir à l'instruction GET x,y1,x+31,y2,shape\$ pour inscrire ces données dans un string (shape\$) : les données du sprite sont ainsi directement chargées en mémoire dans le bon format (là où se trouve le string).

Voici la formule permettant de calculer memadr__% :

*memadr*__% = V : shape\$+6

Il est indispensable d'additionner 6, car le GfA Basic enregistre la hauteur et la largeur du secteur ainsi que le nombre de plans binaires devant les données proprement dites.

Si vous utilisez cette méthode, faites bien attention à ce que le secteur lu à l'aide de l'instruction GET ait effectivement une largeur de 64 pixels, ni plus, ni moins.

- *sourceadr*__% :

Adresse du masque du sprite ; vous n'entrez ce paramètre que dans la module H_SHAP64 (haute résolution). La variable *sourceadr*__% contient l'adresse de début d'un masque de sprite ; en haute résolution, ce masque sert à préciser quels secteurs de l'arrière-plan sont purement et simplement recouverts par le sprite et quels secteurs sont par contre visibles par 'transparence'.

La structure de ce masque ressemble fortement à celle du masque de la flèche de la souris, que vous créez en GfA Basic à l'aide de l'instruction DEFMOUSE.

Le masque du sprite prend exactement la même taille que le sprite lui-même et peut aussi venir s'inscrire dans un string (donc en mémoire) en utilisant l'instruction GET. Tout point du sprite (qu'il soit blanc ou noir) appelé à recouvrir et dissimuler l'arrière-plan doit être identifié dans le masque par un bit vrai (1) ; par contre, les points devant laisser transparaître l'arrière-plan seront identifiés par des bit faux (0).

Vous trouverez dans le dossier EXAMPLES\STUFF une image DEMO.PIC vous montrant l'aspect que peut prendre un sprite et son masque. Ce dossier contient 40 objets graphiques pouvant servir de sprite, flanqués de leur 40 masques.

La disquette contient aussi un programme de démonstration pour les moniteurs haute résolution

(voir EXAMPLES\PROGRAMS.HIG\HSHAP64.LST).

Tout cela suffit pour vous faire comprendre concrètement comment fonctionne ce module 'shape'.

Valeur retournée :

*lib_rv*__% = code d'erreur ; *lib_rv*__% retourne la valeur -1 lorsque vous oubliez de charger le fichier INLINE accompagnant le module et 0 lorsque tout se déroule normalement.

Description :

Les modules L_SHAP64, M_SHAP64 et H_SHAP64 vous permettent de créer des sprites d'une largeur de 64 pixels ; contrairement aux limites imposées par l'instruction SPRITE, le sprite peut ici prendre quasiment n'importe quelle hauteur et contenir toutes les couleurs offertes par le degré de résolution caractérisant le moniteur.

Attention : ▽

Souvenez-vous qu'il convient d'entrer un paramètre de plus lorsque vous utilisez H_SHAP64 (sourceadr__%) ; ce paramètre indique l'adresse du masque du sprite, qui n'est pas engendré automatiquement par le module contrairement à ce qui se passe sous L_SHAP64 et M_SHAP64.

Comme l'utilisation d'un tel sprite joue beaucoup sur les délais d'affichage, il est sans doute avantageux d'appeler immédiatement la routine en assembleur. Ceci accélère considérablement le processus et rend l'affichage du sprite plus 'souple' puisqu'on peut omettre de sauvegarder l'arrière-plan. C'est de plus intéressant lorsque l'on souhaite restaurer l'arrière-plan en le faisant varier (par exemple au moyen d'une animation).

Voici dans ce cas la syntaxe pour lancer L_SHAP64 :

```
~C:l_shap64mc__% (x__%,y__%,options__%,rows__%,  
                  L:screenadr__%,L:bufadr__%,L:memadr__%)
```

pour lancer M_SHAP64 :

```
~C:m_shap64mc__% (x__%,y__%,options__%,rows__%,  
                  L:screenadr__%,L:bufadr__%,L:memadr__%)
```

et pour lancer H_SHAP64 :

```
~C:h_shap64mc__% (x__%,y__%,options__%,rows__%,  
                  L:screenadr__%,L:bufadr__%,L:memadr__%)
```

Les paramètres sont exactement les mêmes que lorsque vous lancez la procédure, à l'exception de options__%, qui peut prendre les valeurs suivantes :

- > **options__%** == 0 : GETBACK, sauvegarder l'arrière-plan
- > **options__%** == 1 : PUTBACK, restaurer l'arrière-plan
- > **options__%** == 2 : SETSHAPE, activer le sprite.

Toute autre valeur sera ignorée ; voici donc la succession logique pour activer un sprite et le faire disparaître :

- > D'abord options__% = 0 ==> GETBACK
- > Ensuite options__% = 2 ==> SETSHAPE
- > Enfin options__% = 1 ==> PUTBACK

Ce qui nous donne trois appels en tout, mais il est possible de laisser tomber GETBACK et PUTBACK lorsque l'on veut restaurer l'arrière-plan en le modifiant. Attention : PUTBACK ne doit en principe intervenir qu'après un appel de GETBACK.

Notez cependant que, pour qu'il soit possible de lancer directement le module SHAPE, il faut dès le début du programme prévoir le lancement de cette procédure.

En clair : avant de lancer la routine en assembleur à l'aide de la commande ~C :...Il faut avoir appelé au moins une fois la procédure en recourant à l'instruction GOSUB, faute de quoi l'adresse start de la routine en assembleur (l_shap64mc__%, m_shap64mc__%, h_shap64mc__%)

est encore sur 0 et l'appel direct mène tout droit à une erreur de bus 'BUSERROR' (deux petites bombes à l'écran!).

Une fois le procédure chargée, l'adresse de début de la routine en assembleur est identifiée et vous pouvez donc la reprendre à l'aide de ~C :

Exemple : EXAMPLES\PROGRAMS.LOW\L_SHAP64.LST

MERGE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_SHAP64.LST

 GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST

 GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```
'                                     dimensionner le buffer pour sauvegarder
'                                     l'arrière-plan derrière le sprite
(16*40+4)
INLINE bufadr __%,644
'
GOSUB gfa_poppal__
palette$=lib_rv__$
```

```

GOSUB gfa_pload__("\EXAMPLES\STUFF\DEMO.PI1",XBIOS(2),0)
,
GET 0,136,63,151,shape$      !inscrire le sprite dans un string
ALERT 2,"Appuyez sur une |touche quelconque pour|interrompre
la démo",1,"Start",button|
x&=0
y&=136
xi&=1
yi&=-1
HIDEM
REPEAT
,
GOSUB gfa_l_shap64__(x&,y&,0,16,XBIOS(2),bufadr__%,V :shape$+6)
,
ADD x&,xi&
ADD y&,yi&
IF x&>255 OR x&<1
    MUL xi&,-1
ENDIF
IF y&>183 OR y&<124
    MUL yi&,-1
ENDIF
VSYNC
,
GOSUB gfa_l_shap64__(0,0,1,16,XBIOS(2),bufadr__%,V :shape$+6)
,
UNTIL INKEY$<>""
SHOWM
~XBIOS(6,L :V :palette$)

```

Ce programme de démonstration est similaire à celui de L_SHAP16, la seule différence résidant dans le calcul de la taille du buffer destiné à contenir l'arrière-plan et dans le fait que l'instruction GET concerne un sprite de 64 pixels de large pour 16 pixels de haut.

La taille du buffer est à nouveau calculée à l'aide de la formule indiquée ci-dessus. Comme notre sprite a une hauteur de 16 lignes, nous multiplions la constante 40 par 16, et nous ajoutons 4. Le reste de la démonstration est rigoureusement semblable à la démonstration de L_SHAP16.

Graphisme : L_TCONV M_TCONV

Utilisation : Conversion d'une police de caractères dans le but de la rendre utilisable avec les modules L_TEXT88, M_TEXT88 et H_TEXT88.

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_TCONV.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_TCONV.LST
(moyenne résolution)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_TCONV.LST
(haute résolution)

INLINE : Ne nécessite aucun fichier INLINE

Exemples : EXAMPLES\PROGRAMS.LOW\L_TCOTEX.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\H_TCOTEX.LST
(haute résolution)

Syntaxe : GOSUB
gfa_l_tconv__(screenadr__%,destadr__%,y__%)

GOSUB
gfa_m_tconv__(screenadr__%,destadr__%,y__%)

GOSUB
gfa_h_tconv__(screenadr__%,destadr__%,y__%)

Paramètres :

- *screenadr__%* :

Adresse de début de la mémoire d'écran ; lorsque la police de caractères (en couleur) à convertir doit s'afficher à l'écran, il suffit d'entrer XBIOS(2) à la place de cette adresse.

Mais on peut aussi enregistrer la police de caractères dans une autre partie de la mémoire vive (par exemple l'espace-mémoire contenant le deuxième écran momentanément invisible) dont il convient alors d'indiquer l'adresse exacte.

- **destadr__%** :

Adresse d'un buffer qui recevra la police de caractères convertie ; la quantité d'informations nécessaires pour représenter une police de caractères varie en fonction du degré de résolution de l'écran, ce qui fait aussi varier la taille qu'il convient de donner au buffer :

- **basse résolution** : 8192 octets
- **moyenne résolution** : 4096 octets
- **haute résolution** : 4096 octets

Il est possible de sauvegarder toutes ces données lorsque l'on souhaite pouvoir réutiliser la police de caractères sous sa forme convertie : on peut ainsi se constituer une collection de polices de caractères qu'il suffit de recharger, sans avoir à refaire la conversion préalable à chaque fois.

- **y__%** :

Numéro de la ligne sur laquelle commence la police de caractères ; il s'agit d'une ligne pixel, puisque la police peut commencer sur n'importe quelle ligne-pixel.

Valeur retournée :

Aucune.

Description :

les procédures L_TCONV, M_TCONV et H_TCONV vous permettent de convertir des polices de caractères dessinées à l'aide d'un logiciel quelconque, afin de pouvoir les réutiliser avec les modules L_TEXT88, M_TEXT88 et H_TEXT88.

Vous ne pouvez pas mélanger des polices de résolutions différentes : concrètement, vous ne pouvez pas convertir une police de caractères avec L_TCONV pour l'utiliser sous M_TEXT88 ou H_TEXT88 (et inversement).

Il convient de respecter un certain nombre de règles pour convertir correctement les polices de caractères :

➤ **En basse résolution :**

La police comprend 256 caractères, rangés sur 6 rangs de 40 caractères plus un rang de 16. Un caractère peut contenir les 16 couleurs accessibles en basse résolution, et doit occuper 8 x 8 pixels.

➤ **En moyenne résolution :**

La police comprend 256 caractères, rangés sur 3 rangs de 80 caractères plus un rang de 16. Un caractère peut contenir les 4 couleurs accessibles en moyenne résolution, et doit occuper 8 x 8 pixels.

➤ **En haute résolution :**

La police comprend 256 caractères, rangés sur 3 rangs de 80 caractères plus un rang de 16. Un caractère doit avoir une largeur de 8 pixels pour une hauteur de 16 pixels.

➤ **Règles applicables quelle que soit la résolution :**

Le premier caractère doit commencer sur le bord gauche de l'écran et reçoit le code ASCII 0 ; les codes ASCII augmentent de la gauche vers la droite et du haut vers le bas du tableau.

Il n'est pas indispensable de convertir l'ensemble des 256 caractères, mais il convient dans ce cas de laisser un espace vide à la place du caractère non converti.

L'image contenue dans DEMO.PI1 vous montre l'aspect que peut avoir une police en basse résolution et comment il convient de la ranger à l'écran ; vous trouvez un autre exemple dans DEMO.PIC, qui montre une police de caractères en haute résolution.

Ces deux images vous montrent d'ailleurs qu'il est possible d'attribuer des symboles graphiques à des caractères, symboles dont la combinaison vous permettra ensuite d'élaborer des dessins complexes.

Exemple : `EXAMPLES\PROGRAMS.LOW\L_TCOTEXT.LST`

```

MERGE:  GFA_GLIB.3_0\GFA_GLIB.LOW\L_TCONV.LST
        GFA_GLIB.3_0\GFA_GLIB.LOW\L_TEXT88.LST
        GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST
        GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```

```

'                                     ! Buffer pour la police de caractères (8192 octets)
INLINE fontadr__%,8192
'
GOSUB gfa_poppal__
Palette$=lib_rv__$
GOSUB gfa_pload__("\EXAMPLES\STUFF\DEMO.PI1",XBIOS(2),0)
'
GOSUB gfa_l_tconv__(XBIOS(2),fontadr__%,36)
'
REPEAT
  CLS
  PRINT AT(1,1) ;"veuillez entrer un texte "
  INPUT txt$
  '
  GOSUB gfa_l_text88__(XBIOS(2),fontadr__%,V:txt$,0,140,LEN(txt$),&X11)
  '
  ALERT 2," | Continuer? | ",1," OUI | NON ",button|
UNTIL button|=2
~XBIOS(6,L:V:palette$)

```

INLINE nous sert à installer un buffer de 8192 octets, destiné à recevoir la police de caractères convertie en basse résolution.

Nous nous servons ensuite de POPPAL pour sauvegarder la palette des couleurs, et nous chargeons, à l'aide de PLOAD, l'image DEMO.PI1 ainsi que sa palette dans le buffer d'écran.

Notre image contient, à partir de sa ligne 36, une police de caractères pour la basse résolution. Grâce à L_TCONV, cette police est convertie en un format utilisable sous L_TEXT88. Les données sont mémorisées dans le secteur de la mémoire vive que nous avons réservé par INLINE.

Vous pouvez alors entrer un texte dans une boucle REPEAT... UNTIL, texte qui s'affichera dans la police de caractères redéfinie dès que vous aurez

appuyé sur la touche <return>. Dans notre exemple, nous avons sélectionné un mode L_TEXT88 tel, que l'affichage de textes comprenant plus de 40 caractères se poursuit sur la ligne suivante.

Vous pouvez répéter cette manoeuvre autant de fois que vous le souhaitez ; cliquez sur le bouton 'NON' de la boîte de dialogue pour interrompre la démonstration, ce qui réactive l'ancienne palette des couleurs d'origine.

Graphisme : L_TEXT88 M_TEXT88 H_TEXT88

Utilisation : Affichage d'un string à l'aide d'une police de caractères redéfinie et dans un certain mode d'affichage.

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_TEXT88.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_TEXT88.LST
(moyenne résolution)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_TEXT88.LST
(haute résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_TEXT88.INL
(L_TEXT88.LST)

GFA_GLIB.3_0\GFA_GLIB.MED\M_TEXT88.INL
(M_TEXT88.LST)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_TEXT88.INL
(H_TEXT88.LST)

Exemples : EXAMPLES\PROGRAMS.LOW\L_TCOTEX.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\H_TCOTEX.LST
(haute résolution)

Syntaxe : GOSUB gfa_l_text88__(screenadr__%,fontadr__%,
memadr__%,x__%,y__%,length__%,options__%)

GOSUB gfa_m_text88__(screenadr__%,fontadr__%,
memadr__%,x__%,y__%,length__%,options__%)

GOSUB gfa_h_text88__(screenadr__%,fontadr__%,
memadr__%,x__%,y__%,length__%,options__%)

Paramètres :

- *screenadr*__% :

Adresse (du début) de la mémoire d'écran ; on entre ici XBIOS(2) lorsqu'on souhaite que le string vienne s'afficher directement dans l'écran visible. La sortie des données peut se faire aussi vers une autre partie de la mémoire vive (par exemple vers un écran non visible), dont il convient alors de préciser l'adresse.

- *fontadr*__% :

Adresse d'un buffer contenant la police de caractères (font) reconvertie ; avant même de pouvoir sortir un texte à l'aide de L_TEXT88, M_TEXT88 ou H_TEXT88, vous devez disposer en mémoire d'une police convertie grâce à L_TCONV, M_TCONV ou H_TCONV.

> *fontadr*__%

Indique l'adresse de début de cet espace mémoire : il faut veiller à ce que la police en question corresponde bien au module utilisé (et plus précisément au degré de résolution), faute de quoi l'affichage sera illisible.

- *memadr*__% :

Vous entrez ici l'adresse de la chaîne de caractères (string) à afficher ; pour sortir par exemple un string txt\$, on peut déterminer cette adresse par V:txt\$. Il n'est pas indispensable cependant que le texte à afficher se trouve dans un string, même si l'on peut ici entrer l'adresse d'une chaîne de caractères, car le texte peut aussi se trouver dans un buffer de la mémoire vive.

- *x*__% :

Coordonnée X de la colonne où le texte doit venir s'afficher ; il convient de respecter les minima/maxima suivants :

> **basse résolution :** 0 - 39

> **moyenne résolution :** 0 - 79

> **haute résolution :** 0 - 79

Aucun affichage ne se produit lorsque *x*__% dépasse ces valeurs.

- **y__%** :

Coordonnée Y de l'endroit où le texte doit venir s'afficher, en pixels, de sorte que le texte peut se trouver sur n'importe quelle ligne de pixels ; il convient de respecter les minima/maxima suivants :

- > **basse résolution** : 0 - 192
- > **moyenne résolution** : 0 - 192
- > **haute résolution** : 0 - 384

Aucun affichage ne se produit lorsque y__% dépasse ces valeurs.

- **length__%** :

Longueur de la chaîne de caractères à afficher ; lorsque le texte est enregistré sous forme de string txt\$, on peut calculer sa longueur grâce à LEN(txt\$) ; cette indication de longueur peut aussi servir à ne sortir que des fragments d'un string. Lorsque le texte ne se présente pas sous forme de string mais qu'il est enregistré quelque part dans la mémoire vive, il faut absolument indiquer sa longueur pour préciser la fin du texte.

- **options__%** :

Ce paramètre est exploité bit par bit ; seuls les deux bits 0 et 1 prennent une signification :

> **options__%** = Integer de 32 bits de la forme :

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxmm

x = Bit ne servant pas à préciser le mode (peut prendre un état quelconque)

m = Bit servant à préciser le mode d'affichage

Bit 0 = 0 => l'affichage du texte est coupé au bord de l'écran dans le cas où le texte dépasse le côté droit de l'écran

Bit 0 = 1 => L'affichage du texte se poursuit sur la ligne suivante lorsque le texte dépasse le côté droit de l'écran.

Bit 1 = 0 => Le texte laisse voir l'arrière-plan 'par transparence', si bien que le texte peut venir surcharger des graphiques.

Bit 1 = 1 => Le texte recouvre et fait disparaître l'arrière-plan (blocked text).

Chaque bit de mode ayant sa propre fonction, il est possible de combiner à volonté les deux options. Pour simplifier la saisie du mode, il est conseillé de recourir à la saisie des paramètres sous forme binaire, ce qui nous donne les quatre combinaisons suivantes :

&X00 = texte coupé à la fin de la ligne + effet de transparence

&X01 = texte se poursuivant sur la ligne du dessous + effet de transparence

&X10 = texte coupé à la fin de la ligne + blocked text

&X11 = texte se poursuivant sur la ligne du dessous + blocked text.

Valeur retournée :

lib_rv__% :

Code d'erreur ; lib_rv__% prend la valeur -1 lorsque vous oubliez de charger le fichier **INLINE** accompagnant le module et la valeur 0 lorsque tout se déroule normalement.

Description :

Les modules **L_TEXT88**, **M_TEXT88** et **H_TEXT88** vous permettent de redéfinir des polices de caractères, sans qu'il soit besoin de charger le **GDOS** ni de procéder à des préparatifs sans fin, et ce, sans que la conversion se limite à une seule couleur (pour les écrans couleur évidemment).

Grâce à ces modules, vous pouvez utiliser autant de polices que vous voulez (la seule limite étant la capacité mémoire de votre configuration) pour sortir des textes dans des mode d'affichage très variés. Des textes peuvent ainsi venir surcharger des graphiques.

Comme nous avons renoncé à certains effets spéciaux (par exemple windowclipping) l'affichage du texte est encore plus rapide qu'avec les instructions **TEXT** ou **PRINT** du **GfA Basic**.

Exemple : voir **L_TCOTEXT.LST** avec **L_TCONV**, **M_TCONV** ou **H_TCONV**.

Graphisme : SETRES

Utilisation : Sélection de la résolution qui s'appliquera ensuite pour toutes les instructions de sortie vers l'écran lorsque la procédure MRES sera utilisée avec un affichage ne différenciant pas entre la basse et moyenne résolution.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\SETRES.LST
(basse et moyenne résolution)

INLINE : Ne nécessite aucun fichier INLINE.

Exemples : EXAMPLES\PROGRAMS.LOW\MRES.LST
(basse résolution)

Syntaxe : GOSUB gfa_setres__(options__%)

Paramètres :

- *options__%* :

Degré de résolution s'appliquant ensuite à toutes les instructions d'affichage à l'écran après l'appel de la procédure SETRES ; *options__%* peut prendre les valeurs suivantes :

- > *options__%* == 0 : basse résolution
- > *options__%* == 1 : moyenne résolution

Toute autre valeur sera ignorée et n'aura donc aucun effet.

Valeur retournée :

Aucune

Description :

La procédure SETRES permet un affichage correct, tenant compte du bon degré de résolution lorsque l'écran est divisé en plusieurs parties grâce au procédure MRES.

MRES permet en effet de recourir simultanément à la basse et moyenne résolution ; mais avant d'afficher des données sur une des parties découpées dans l'écran, il convient de procéder aux ajustements nécessaires en respectant la résolution adéquate : c'est là qu'intervient SETRES.

Il n'est pas indispensable de lancer SETRES avant chaque affichage : il suffit de l'appeler lorsqu'on change de degré de résolution, c'est-à-dire par exemple lorsque l'affichage A doit se faire dans telle partie de l'écran en basse résolution et que l'affichage suivant B doit se faire dans une autre partie de l'écran et dans une autre résolution.

Après avoir sélectionné un degré de résolution à l'aide de SETRES, il est possible de procéder à plusieurs affichages successifs dans le même degré de résolution. L'affichage serait par contre illisible si vous passez dans une partie de l'écran demandant un autre degré de résolution sans être passé auparavant dans ce dernier.

Exemple : EXAMPLES\PROGRAMS.LOW\MRES.LST

MERGE : GFA_GLIB.3_0\GFA_GLIB.ALL\SETRES.LST
 GFA_GLIB.3_0\GFA_GLIB.ALL\MRES.LST

```
'                                     installer un buffer de 6 octets
INLINE pblockadr%,6
'                                     écrire le bloc des paramètres
BYTE{pblockadr%}=0                  ! résolution du premier secteur en
haut de l'écran
BYTE{pblockadr%+1}=40                ! 40 lignes en basse résolution
BYTE{pblockadr%+2}=1                 ! résolution du deuxième secteur
BYTE{pblockadr%+3}=60                ! 60 lignes en moyenne résolution
BYTE{pblockadr%+4}=0                 ! résolution du troisième secteur
BYTE{pblockadr%+5}=100               ! tout le reste en basse résolution
'
GOSUB gfa_mres__(1,pblockadr%)
'
GOSUB gfa_setres__(0)                 ! basse résolution
PRINT AT(1,1) ; "texte en basse résolution"
'
GOSUB gfa_setres__(1)                 ! moyenne résolution
PRINT AT(1,7) ; "texte en moyenne résolution"
```

Graphisme : MRES

Utilisation : Affichage simultané de plusieurs secteurs de l'écran en résolution basse et moyenne ; il est possible de diviser l'écran en bandes horizontales d'affichage en degré de résolution différents l'un de l'autre.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\MRES.LST
(basse et moyenne résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.ALL\MRES.INL
(MRES.LST)

Exemples : EXAMPLES\PROGRAMS.LOW\MRES.LST
(basse résolution)

Syntaxe : GOSUB gfa_mres__(options__%,sourceadr%)

Paramètres :

- *options__%* :

Code de commande permettant d'activer ou désactiver la procédure MRES et donc la partition de l'écran en plusieurs secteurs, en entrant l'une des valeurs suivantes :

➤ *options__%* == 0 : annuler la partition de l'écran en secteurs

➤ *options__%* == 1 : activer la partition de l'écran en secteurs

Toute autre valeur restera sans effet et sera ignorée.

- *sourceadr__%* :

Adresse du bloc des paramètres précisant le découpage de l'écran en secteurs, et ayant la structure suivante :

➤ **octet** Indiquant le degré de résolution du premier secteur (0 = basse résolution et 1 = moyenne résolution)

- **octet** Indiquant le nombre de lignes occupées par le premier secteur
- **octet** Indiquant le degré de résolution du deuxième secteur (0 = basse résolution et 1 = moyenne résolution)
- **octet** Indiquant le nombre de lignes occupées par le deuxième secteur
- ...
- etc.

Il convient d'entrer autant de paires d'octets (résolution, nombre de lignes) que de secteurs découpés dans l'écran.

Attention : ▽

Le total des lignes doit être supérieur à 270, afin de ne pas avoir à repréciser un degré de résolution pour le bas de l'écran.

Valeur retournée :

lib_rv__% = code d'erreur ; lib_rv__% retourne la valeur -1 lorsque vous oubliez de charger le fichier INLINE accompagnant la procédure, et la valeur 0 lorsque tout se déroule normalement.

Description :

Cette procédure vous permet de procéder à des affichages dans les deux degrés de résolution à la fois, en découpant l'écran en secteurs horizontaux ayant chacun son propre degré de résolution.

Les instructions du GfA Basic vous permettent de procéder à des affichages dans les différents secteurs de l'écran, mais il convient de les adapter auparavant à la résolution voulue grâce à la procédure SETRES (voir ci-dessus).

Exemple : voir MRES.LST sous SETRES.

Graphisme : MPAL

Utilisation : Avec un écran couleur, affichage en 16 ou 4 couleurs simultanément.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\MPAL.LST
(basse et moyenne résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.ALL\MPAL.INL
(MPAL.LST)

Exemples : EXAMPLES\PROGRAMS.LOW\MPAL.LST
(basse résolution)

Syntaxe : GOSUB gfa_mpal__(options__%,memadr__%)

Paramètres :

- *options__%* :

Code de commande, permettant de préciser si l'on veut activer/désactiver l'affichage de plus de 16 (basse résolution) ou de 4 (moyenne résolution) couleurs ; voici les valeurs autorisées :

- **options__% <> 0** : Début de l'affichage en plus de 16/4 couleurs
- **options__% == 0** : Annulation de l'affichage en plus de 16/4 couleurs simultanément.

- *memadr__%* :

Adresse du début du bloc de paramètres qui précise les différentes palettes de couleurs ainsi que la taille du secteur d'écran concerné par cette palette.

Le bloc de paramètres a la structure suivante :

- 1er mot : hauteur du secteur le plus en haut de l'écran, indiquée en lignes de pixels ;
- 2ème - 17ème mots : palette des couleurs, valable pour le secteur occupant le haut de l'écran (un mot par couleur)

- 18ème mot : hauteur du deuxième secteur d'écran, en lignes de pixels
- 19ème - 34ème mots : palette des couleurs valable pour le deuxième secteur
- etc.

La structure de ce bloc de paramètres est donc toujours semblable : d'abord un mot pour indiquer la hauteur de la bande d'écran puis 16 mots pour préciser la palette de couleurs valable dans ce secteur de l'écran.

Le bloc de paramètres peut avoir une longueur quasiment quelconque, mais il convient de respecter les règles suivantes :

- il faut préciser les 16 couleurs de la palette, même en moyenne résolution (où seules quatre couleurs sont utiles)
- la hauteur de la bande d'écran ne peut être inférieure à 2 lignes de pixels puisque la palette de couleurs doit rester valable sur au moins une ligne
- les couleurs sont encodées dans les mots en respectant les codes usuels (voir L_GREYS et M_GREYS)
- le dernier secteur d'écran doit avoir une hauteur telle que la somme des hauteurs de tous les secteurs soit supérieure à 250.

Valeur retournée :

`lib_rv__%` = Code d'erreur ; `lib_rv__%` retourne la valeur -1 lorsque vous oubliez de charger le fichier `INLINE` accompagnant la procédure, et la valeur 0 lorsque tout se déroule normalement.

Description :

La procédure `MPAL` vous permet de changer de palette de couleurs au maximum toute les deux lignes, et donc d'utiliser dans un même affichage les 512 couleurs possibles avec l'Atari ST, tant en basse qu'en moyenne résolution.

Après avoir lancé `MPAL`, les couleurs indiquées restent valables jusqu'au prochain appel de `MPAL` suivi du code de commande `options__%=0`.

Comme MPAL tourne sous interruption, vous pouvez fort bien exécuter en même temps d'autres instructions du GfA Basic. MPAL utilise les mêmes interruptions que MRES, il faut donc veiller à ne pas les lancer en même temps, ce qui entraînerait un plantage du système.

Comme avec toute routine utilisant des interruptions, vous devez ici aussi faire attention à bien ressortir de la procédure avant de sortir de votre programme, faute de quoi la routine se poursuivrait sous le bureau GEM ce qui entraînerait un plantage du système.

Exemple : EXAMPLES\PROGRAMS.LOW\MPAL.LST

MERGE: GFA_GLIB.3_0\GFA_GLIB.ALL\MPAL.LST

 GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```
'
                                Bloc des paramètres, contenant la palette des
                                couleurs
'
                                Et la hauteur du secteur d'écran ((2+32)*5)

INLINE pblockadr%,170'
GOSUB gfa_poppal__
palette$=lib_rv__$
RESTORE parameterblock
FOR i&=0 TO 5*17-1              ! lecture du tableau des couleurs et des
                                hauteurs
    READ word&                  ! des secteurs => élaboration du bloc des
                                paramètres
    WORD{pblockadr%+(i&*2)}=word&
NEXT i&
FOR i&=0 TO 304 STEP 20        ! élaboration de l'image
    DEFFILL i&/20,2,8
    PBOX i&,0,i&+39,199
NEXT i&
PRINT AT(1,1) ; "<<< pour continuer, appuyer sur une touche >>>"
HIDEM
'
GOSUB gfa_mpал__(1,pblockadr%)! pour lancer MPAL
'
~INP(2)                        ! guette l'appui sur une touche
'
GOSUB gfa_mpал__(0,0)          ! stopper MPAL
'
```


SHOWM

~XBIOS(6,L :V :palette\$)

Parameterblock :

DATA 40

DATA &H777,&H765,&H567,&H707,&H770,&H077,&H713,&H700

DATA &H070,&H007,&H767,&H677,&H776,&H766,&H676,&H667

DATA 40

DATA &H666,&H654,&H456,&H606,&H660,&H066,&H613,&H600

DATA &H060,&H006,&H656,&H566,&H665,&H655,&H565,&H556

DATA 40

DATA &H555,&H543,&H345,&H505,&H550,&H055,&H513,&H500

DATA &H050,&H005,&H545,&H455,&H554,&H544,&H454,&H445

DATA 40

DATA &H444,&H432,&H234,&H404,&H440,&H044,&H413,&H400

DATA &H040,&H004,&H434,&H344,&H443,&H433,&H343,&H334

DATA 90

DATA &H333,&H321,&H123,&H303,&H330,&H033,&H313,&H300

DATA &H030,&H003,&H323,&H233,&H332,&H322,&H232,&H223

Dès le début du programme, nous sauvegardons la palette actuelle des couleurs grâce à POPPAL, après quoi nous chargeons le bloc des paramètres affectés à MPAL. Les hauteurs et les palettes de couleurs se trouvent dans les lignes DATA et viennent s'inscrire dans un buffer de 170 octets créé à l'aide d'un INLINE.

En lisant les lignes DATA, vous pouvez voir qu'il est possible d'entrer très clairement les palettes de couleurs : on peut en effet pour chaque couleur lire immédiatement la valeur du rouge, du vert et du bleu.

Nous voyons qu'il s'agit ici de créer cinq secteurs d'écran disposant chacun de sa propre palette de couleurs et d'une hauteur de 40 lignes (mais ils pourraient fort bien être de hauteurs différentes).

Nous dessinons ensuite sur l'écran 16 bandes verticales représentant les 16 couleurs. Nous lançons MPAL (options__%=1) en indiquant comme adresse du bloc des paramètres l'adresse du string pblock\$. Dès ce moment, on peut voir jusqu'à 80 couleurs s'afficher simultanément à l'écran.

Il est possible de mettre fin à cette démonstration en appuyant sur une touche quelconque ; il est possible de faire exécuter d'autres instructions du GfA Basic au lieu de guetter l'appui sur une touche.

Dès qu'une touche est appuyée, cela provoque un nouvel appel de MPAL mais cette fois options__%=0, ce qui interrompt l'affichage de nos 80 couleurs. Seules 16 couleurs restent visibles. Lors de l'arrêt de la démonstration, on donne une valeur quelconque à memadr__% car elle est de toute façon ignorée.

Graphisme : POLICE

Utilisation : Sortie de chaînes de caractères (strings) avec une police de caractères que l'on peut dessiner soi-même et dont les caractères peuvent avoir n'importe quelle hauteur.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\POLICE.LST
(basse résolution)

INLINE : Ne nécessite aucun fichier INLINE

Exemples : EXAMPLES\PROGRAMS.LOW\POLICE.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\POLICE.LST
(haute résolution)

Syntaxe : GOSUB gfa_police__(x__%,y__%,width__%,height__%,
value__%,txt__\$,options__%)

Paramètres :

- x__% :

Coordonnée X

- y__% :

Coordonnée Y ; les paramètres x__% et y__% ont une signification différente selon que l'on appelle la procédure POLICE pour initialiser ou pour afficher une chaîne de caractères.

En initialisation, la procédure POLICE sert à enregistrer une police de caractères qui servira plus tard pour l'affichage d'une chaîne de caractères. Dans ce cas, les coordonnées x__% et y__% indiquent le coin supérieur gauche du premier caractère à enregistrer.

Lorsqu'il s'agit d'afficher une chaîne de caractères, ces coordonnées servent à positionner la chaîne de caractères à afficher : elles désignent le coin supérieur gauche du premier caractère de la chaîne.

- *width*__% :

Largeur d'un caractère.

- *height*__% :

Hauteur d'un caractère ; les deux paramètres *width*__% et *height*__% prennent aussi une signification différente selon leur utilisation.

En initialisation, ils servent à enregistrer correctement les caractères de la police qui vient d'être traitée.

Lors de l'affichage, *height*__% n'a aucune signification, alors que *width*__% sert à déterminer l'écart entre deux caractères de la chaîne. Si l'on donne alors à *width*__% la valeur correspond à la largeur d'un caractère, tous les caractères seront collés l'un à l'autre ; en lui donnant une valeur supérieure à la largeur d'un caractère, le texte s'étire sur l'axe des X ; si on lui donne une valeur inférieure à la largeur d'un caractère, ceux-ci se chevauchent.

- *value*__% :

Cette valeur est semblable à l'indication de mode dans l'instruction PUT du GfA Basic . Comme les caractères de la police sont affichés à l'aide de l'instruction PUT, *value*__% permet de préciser comment le texte à afficher va se combiner avec le contenu actuel de l'écran. Dans la table suivante, S(Source) désigne le motif de bits contenu dans la chaîne de caractères et D (Destination) le contenu originel de l'écran :

Mode	Combinaison	Effet
0	0	Tous les points sont effacés
1	S AND D	Seuls restent fixés les points fixés dans les deux motifs
2	S AND (NOT D)	Seuls sont fixés les points fixés dans la grille source mais effacés dans la grille de destination

3	S	La grille source est transférée sans modification (GRAPHMODE 1,valeur pré définie par défaut)
4	(NOT S) AND D	Seuls sont fixés les points fixés dans la grille source et fixés dans la grille de destination
5	D	La destination n'est pas modifiée
6	S XOR D	Seuls sont fixés les points fixés dans une seule des grille (GRAPHMODE 3)
7	S OR D	Sont fixés tous les points fixés dans l'une des deux grilles (GRAPHMODE 2)
8	NOT (S OR D)	Sont fixés tous les points qui ne sont fixés dans aucune des deux grilles
9	NOT (S XOR D)	Sont fixés tous les points fixés dans les deux grilles ou dans aucune d'entre elles
10	NOT D	Inversion de la grille de destination
11	S OR (NOT B)	Sont fixés tous les points fixés dans la grille source ainsi que tous ceux qui n'étaient pas fixés dans la grille de destination
12	NOT B	Inversion de la grille source avant transfert

13	(NOT S) OR B	GRAPHMODE 4
14	NOT (S AND D)	Sont fixés tous les points qui ne figuraient pas dans les deux grilles à la fois
15	1	Tous les points sont fixés.

Si en outre le bit 4 du mode est fixé, le motif de remplissage subira une combinaison supplémentaire AND avec la grille source.

- txt__\$: texte à afficher (string), qui ne doit en principe contenir que des caractères prévus dans la police de caractères ; cette dernière peut contenir les caractères suivants :

'A'-'Z', '0'-'9', '!', '.', ': ' et enfin ' '

les minuscules sont affichées en majuscules ; les caractères non contenus dans la police sont ignorés et deviennent des espaces vides à l'affichage.

- options__% :

Flag ; ce paramètre n'est important qu'en initialisation (= premier appel de la procédure pour charger la police de caractères). La police de caractères ne contient pas de minuscules lorsque options__% = 0 : les minuscules sont alors affichées comme des majuscules. Les minuscules sont par contre affichées en tant que telles lorsque options__% a une valeur différente de 0.

En affichage de texte, POLICE ignore ce paramètre options__% mais il faut tout de même l'entrer avec une valeur quelconque.

Valeur retournée :

Aucune

Description :

La procédure POLICE vous permet de sortir des textes à l'aide de polices de caractères que vous avez vous-même créées. La taille des caractères n'est pas limitée, contrairement à ce qui était le cas avec L_TEXT88 et

M_TEXT88. Il faut cependant veiller à n'afficher qu'une ligne de texte à la fois, car tout texte dépassant cette longueur sera coupé à la fin de la ligne.

En dehors des modes d'affichage définis sous PUT, il n'existe aucune possibilité de 'surimprimer' le texte sur l'arrière-plan.

Sous POLICE, le texte peut venir s'afficher à partir de n'importe quelle position, indiquée en pixels.

Pour utiliser une police de caractères sous POLICE, il convient de la créer en respectant les contraintes suivantes :

- Lorsque **options__%=0**, la police peut contenir les caractères suivants:

ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!:

- Lorsque **options__%** est différent de 0, la police peut contenir les caractères suivants :

ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!.:abcdefghijklmnopqrstuvwxyz

- Les caractères doivent être entrés dans l'ordre indiqué ci-dessus
- Les caractères doivent tous avoir la même taille
- Lors de l'initialisation, les caractères sont lus et enregistrés de la gauche vers la droite et du haut vers le bas, c'est pourquoi il convient de les ranger dans cet ordre
- Il est inutile de prévoir un espace de séparation entre les caractères
- Il n'est pas indispensable de prévoir un jeu complet de caractères, mais il faut alors laisser un espace vide à la place des caractères absents de la police.

L'image contenue dans DEMO.PI1 (dans le dossier EXAMPLES\STUFF) vous montre l'aspect que doit avoir une police de caractères (sans les minuscules) ; autre exemple :

Exemple : EXAMPLES\PROGRAMS.LOW\POLICE.LST

MERGE: GFA_GLIB.3_0\GFA_GLIB.ALL\POLICE.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```
GOSUB gfa_poppal__
palette$=lib_rv__$
GOSUB gfa_pload__("\EXAMPLES\STUFF\DEMO.PI1",XBIOS(2),0)
,
GOSUB gfa_police__(0,136,16,16,3,"",0) ! enregistrer la police de caractères
,                                     et initialiser la procédure
REPEAT
  CLS
  PRINT AT(1,1) ;"veuillez entrer un texte s.v.p."
  INPUT txt$
  ,
  GOSUB gfa_police__(0,20,16,16,3,txt$,0)
  ,
  ALERT 2," | Continuer? | ",1," OUI | NON ",button|
UNTIL button|=2
~XBIOS(6,L :V :palette$)
```

On commence par sauvegarder la palette actuelle des couleurs à l'aide de POPPAL, puis on utilise PLOAD pour charger DEMO.PI1 et sa palette de couleurs. A partir de la ligne 136, cette image contient une police de caractères utilisable pour afficher des textes à l'aide de POLICE. Cette police de caractères est chargée lors du premier appel de POLICE (*initialisation*).

La procédure POLICE est donc lancé une première fois, en entrant 0,136 comme coordonnées x__% et y__%, car c'est là que se situe le coin supérieur gauche du premier caractère de la police (i.e. 'A').

Comme tous les caractères font 16 pixels de large pour 16 pixels de haut, nous attribuons la valeur 16 tant à width__% qu'à height__%.

Le mode PUT reçoit la valeur 3, qui est de toute façon ignorée lors de ce premier appel d'initialisation et nous aurions donc pu entrer n'importe quelle autre valeur. Comme txt__\$ est aussi ignoré lors de l'initialisation, nous entrons un string vide.

Le paramètre options__% reçoit la valeur 0 puisque notre police de caractères ne contient que des majuscules.

Après ce premier appel de POLICE, nous entrons dans une boucle REPEAT... UNTIL pour saisir le texte à afficher. Ce texte viendra s'afficher à la position (en pixels) 0,20 (valeurs de x__% et y__%) en mode PUT 3 (mode normal). Les paramètres height__% et options__% sont ignorés et peuvent prendre des valeurs quelconques.

La procédure POLICE peut servir autant de fois qu'on le souhaite ; pour mettre fin à la démonstration, cliquer sur le bouton 'NON' à la fin de la boucle REPEAT... UNTIL.

Graphisme : L_OPCOMP M_OPCOMP H_OPCOMP

Utilisation : Cette procédure sert à compresser des images, quelle que soit la résolution ; il sera ensuite possible de les remettre à l'état normal à l'aide de L_DECOMP M_DECOMP H_DECOMP

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_OPCOMP.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_OPCOMP.LST
(moyenne résolution)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_OPCOMP.LST
(haute résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_OPCOMP.INL
(L_OPCOMP.LST)

GFA_GLIB.3_0\GFA_GLIB.MED\M_OPCOMP.INL
(M_OPCOMP.LST)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_OPCOMP.INL
(H_OPCOMP.LST)

Exemples : EXAMPLES\PROGRAMS.LOW\L_OPCDEC.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\H_OPCDEC.LST
(haute résolution)

Syntaxe : GOSUB gfa_l_opcomp__(sourceadr__%,destadr__%,
length__%)

GOSUB gfa_m_opcomp__(sourceadr__%,destadr__%,
length__%)

GOSUB gfa_h_opcomp__(sourceadr__%,destadr__%,
length__%)

Paramètres :

- *sourceadr*__% :

Adresse à partir de laquelle l'image va être comprimée

- *destadr*__% :

Adresse d'un secteur de la mémoire vive qui va recevoir les données correspondant à l'image comprimée

- *length*__% :

Longueur de l'image à compresser, en mots simples (L_OPCOMP et H_OPCOMP) ou longs (M_OPCOMP)

Valeur retournée :

*lib_rv*__% =

Nombre d'octets de l'image après compression, ou code d'erreur ; *lib_rv*__% retourne la valeur -1 lorsque vous oubliez de charger le fichier INLINE accompagnant la procédure et la valeur 0 lorsque tout se déroule normalement.

Description :

Ces trois procédures L_OPCOMP, M_OPCOMP et H_OPCOMP vous permettent de compresser des images quelle que soit la résolution.

Les images en basse résolution sont comprimées à l'aide de L_OPCOMP et pourront être décompressées grâce à L_DECOMP.

Les images en moyenne résolution sont comprimées à l'aide de M_OPCOMP et pourront être décompressées grâce à M_DECOMP.

Les images en haute résolution sont comprimées à l'aide de H_OPCOMP et pourront être décompressées grâce à H_DECOMP.

La compression/décompression ne fonctionne que si vous utilisez les deux procédures allant de paire pour un degré déterminé de résolution.

```

GOSUB gfa_setres__(0) ! retour à la basse résolution
PRINT AT(1,20) ; "texte en basse résolution"
PRINT AT(1,23) ; "appuyez sur une touche quelconque pour sortir"
~INP(2) ! attendre l'appui sur une touche

```

```

GOSUB gfa_mres__(0,0) ! annuler la division de l'écran en secteurs

```

On commence par installer, à l'aide de `INLINE`, un buffer d'une taille suffisante (6 octets) pour qu'il puisse contenir le bloc des paramètres pour `MRES`. Six octets suffisent dans notre exemple, car nous divisons l'écran en trois secteurs, ce qui donne un bloc de paramètres de 6 octets. Evidemment, plus on divise l'écran, plus il faut augmenter la taille de ce buffer.

Nous écrivons ensuite le bloc des paramètres nécessaires pour lancer `MRES`, bloc qui décrit le découpage de l'écran en divers secteurs de divers degrés de résolution. Cela se fait à l'aide d'octets groupés par paire, le premier octet précisant le degré de résolution et le deuxième précisant le nombre de lignes (en pixels) concernées par ce degré de résolution.

Dans notre exemple, nous divisons l'écran en trois secteurs. Le secteur supérieur compte 40 lignes en basse résolution ; le deuxième secteur occupe 60 lignes en moyenne résolution ; le dernier secteur occupe le reste du bas de l'écran, en basse résolution : nous lui attribuons 200 lignes, alors qu'il ne peut occuper que les 100 lignes restantes. En effet, le total des lignes des trois secteurs délimités doit dépasser 270 pour que nous n'ayons pas à rechanger de degré de résolution en bas de l'écran.

Le bloc des paramètres étant écrit, nous appelons la procédure `MRES`, en attribuant la valeur 1 à son paramètre `options__%` pour activer la division de l'écran en secteurs.

Nous demandons alors l'affichage d'une ligne de texte en basse résolution dans le haut de l'écran : `SETRES` nous sert à préparer cet affichage en basse résolution (`options__% = 0`).

Cette première instruction `PRINT` étant exécutée, nous demandons l'affichage d'un autre texte en moyenne résolution dans le deuxième secteur de l'écran ; `SETRES` nous sert à passer d'une résolution à l'autre : `options__% = 1`.

Nous voulons enfin afficher un troisième texte en basse résolution et dans le troisième secteur, en bas de l'écran ; SETRES nous sert à nouveau à changer de degré de résolution : $\text{options_}\% = 0$. Nous demandons l'exécution de deux instructions PRINT, afin de bien montrer qu'un seul appel de SETRES suffit du moment qu'on reste dans le même degré de résolution dans le même secteur.

Le programme guette alors un appui sur une touche quelconque pour s'interrompre. Avant de ressortir du programme, il est important de penser à annuler la division de l'écran en secteurs, en appelant MRES avec $\text{options_}\% = 0$, faute de quoi la division persisterait au niveau du bureau GEM ou d'un autre programme, ce qui entraînerait un plantage général du système ou en tout cas des affichages illisibles.

Attention : ▽

Il peut arriver que l'image soit d'une telle complexité (grille très fine, points isolés etc) qu'elle ne puisse être effectivement comprimée ; il peut même arriver qu'elle prenne plus de place en mémoire après sa compression qu'avant, ce qui se produit le plus souvent en moyenne et haute résolution, avec M_OPCOMP et H_OPCOMP. Ce phénomène vient de ce que la procédure renonce à compresser l'image lorsque les algorithmes à mettre en oeuvre sont trop complexes ou trop coûteux en temps de traitement.

Avec des images (ou des morceaux d'image) simples, ces deux procédures mènent toutefois à des résultats satisfaisants. La grande simplicité des algorithmes de compression se révèle même constituer un avantage lors de la décompression, puisque celle-ci se déroule en une fraction de seconde.

Exemple : EXAMPLES\PROGRAMS.LOW\L_OPCDEC.LST

MERGE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_OPCOMP.LST

 GFA_GLIB.3_0\GFA_GLIB.LOW\L_DECOMP.LST

 GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST

 GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```
'               !installer un buffer pour recevoir les données comprimées
INLINE imageadr __%,32000
'
GOSUB gfa_poppal__
palette$=lib_rv__$
GOSUB gfa_pload__("\EXAMPLES\STUFF\DEMO.PI1",XB IOS(2),0)
'
GOSUB gfa_l_opcomp__(XB IOS(2),imageadr __%,32000/2)
'
CLS
PRINT "Nombre d'octets avant compression : 32000"
PRINT "Nombre d'octets après compression : " ;lib_rv__%
PRINT "<<< appuyez sur une touche pour continuer >>>"
~INP (2)
'
' pour sauvegarder l'image comprimée
' BSAVE "DEMO.PCO",imageadr%,lib_rv__%
```

```
GOSUB gfa_1_decomp__(imageadr_%,XBIOS(2))
```

```
~INP(2)
```

```
~XBIOS(6,L :V :palette$)
```

A l'aide d'un `INLINE`, nous commençons par installer un buffer de 32000 octets destiné à recevoir les données de l'image comprimée ; notez que ce buffer ne sera pas complètement occupé puisque l'image comprimée occupera en principe moins de place qu'à l'origine.

`POPPAL` nous sert ensuite à sauvegarder la palette d'origine des couleurs et nous chargeons à l'écran une image `DEGAS` en basse résolution (`DEMO.PI1`) accompagnée de sa propre palette de couleurs.

Après quoi nous comprimons l'image en lançant `L_OPComp` ; nous nous servons de l'adresse de l'écran (`XBIOS(2)`) pour `sourceadr_%` et de l'adresse `imageadr_%` de l'instruction `INLINE` comme `destadr_%`. Nous reprenons pour `length_%` la taille de la mémoire d'écran indiquée en mots ($32000 \text{ octets} / 2 = 16000 \text{ words}$).

Une fois l'image comprimée et l'écran vidé de son contenu, le programme indique le nombre d'octets occupés par les données de l'image ainsi comprimée. La procédure `L_OPComp` renvoie ce nombre d'octets, après la compression, dans la variable `lib_rv_%`. Cette valeur peut également servir à enregistrer les données comprimées.

Il suffit d'appuyer sur une touche quelconque pour que les données soient décomprimées et réaffichées à l'écran. Pour lancer `L_DECOMP`, nous nous servons de l'adresse de début du buffer (`imageadr_%`) comme source des données de l'image comprimée et de l'adresse de la mémoire d'écran (`XBIOS(2)`) comme adresse cible. Notez qu'il n'est pas nécessaire d'indiquer la longueur des données lors du processus de décompression, car la procédure de décompression sait reconnaître le signe de fin des données comprimées.

Il suffit de ré-appuyer sur une touche pour restaurer la palette des couleurs d'origine et mettre fin au programme de démonstration.

Graphisme : L_DECOMP M_DECOMP H_DECOMP

Utilisation : Cette procédure sert, quelle que soit la résolution, à décompresser des images qui ont été comprimées à l'aide de L_OPCOMP M_OPCOMP H_OPCOMP

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_DECOMP.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_DECOMP.LST
(moyenne résolution)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_DECOMP.LST
(haute résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_DECOMP.INL
(L_DECOMP.LST)

GFA_GLIB.3_0\GFA_GLIB.MED\M_DECOMP.INL
(M_DECOMP.LST)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_DECOMP.INL
(H_DECOMP.LST)

Exemples : EXAMPLES\PROGRAMS.LOW\OPCDEC.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\OPCDEC.LST
(haute résolution)

Syntaxe : GOSUB gfa_l_decomp__(sourceadr __%,destadr __%)

GOSUB gfa_m_decomp__(sourceadr __%,destadr __%)

GOSUB gfa_h_decomp__(sourceadr __%,destadr __%)

Paramètres :

- *sourceadr*__% :

Adresse à partir de laquelle se trouvent les données correspondant à l'image comprimée

- *destadr*__% :

Adresse à laquelle l'image va être décomprimée

Valeur retournée :

*lib_rv*__% =

D'erreur ; *lib_rv*__% retourne la valeur -1 lorsque vous oubliez de charger le fichier **INLINE** accompagnant la procédure et la valeur 0 lorsque tout se déroule normalement.

Description :

Ces trois procédures **L_DECOMP**, **M_DECOMP** et **H_DECOMP** vous permettent de décompresser des images dans les trois résolutions.

Les images en basse résolution comprimées à l'aide de **L_OPCOMP** pourront être décompressées grâce à **L_DECOMP**.

Les images en moyenne résolution comprimées à l'aide de **M_OPCOMP** pourront être décompressées grâce à **M_DECOMP**.

Les images en haute résolution comprimées à l'aide de **H_OPCOMP** pourront être décompressées grâce à **H_DECOMP**.

La compression/décompression ne fonctionne que si vous utilisez les deux procédures allant de paire pour un degré déterminé de résolution.

Exemple : Voir **L_OPCDEC.LST** pour **L_OPCOMP**, **M_OPCOMP** et **H_OPCOMP**.

Graphisme : H_LOUPE

Utilisation : Agrandissement d'un secteur d'écran, de forme rectangulaire, selon un facteur à déterminer.

Dossier : GFA_GLIB.3_0\GFA_GLIB.HIG\H_LOUPE.LST
(haute résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.HIG\H_LOUPE.INL

Exemples : EXAMPLES\PROGRAMS.HIG\H_LOUPE.LST
(haute résolution)

Syntaxe : GOSUB gfa_h_loupe__(x1__%,y1__%,x2__%,y2__%,
x__%,y__%,screenadr__%,value__%)

Paramètres :

- x1__% :

Coordonnée X du coin supérieur gauche du secteur à agrandir.

- y1__% :

Coordonnée Y du coin supérieur gauche du secteur à agrandir.

- x2__% :

Coordonnée X du coin inférieur droit du secteur à agrandir

Remarquez qu'il est tout à fait possible d'intervertir x1__% et x2__%, sans toutefois dépasser les limites suivantes :

➤ **haute résolution** : minimum 0 et maximum 639

- y2__% :

Coordonnée Y du coin inférieur droit du secteur à agrandir.

Ici aussi, il est tout à fait possible d'intervertir y1__% et y2__% (ce qui revient à indiquer d'abord le coin inférieur droit puis le coin supérieur gauche), sans toutefois dépasser les limites suivantes :

➤ **haute résolution** : minimum 0 et maximum 399

- x__% :

Coordonnée X du coin supérieur gauche du secteur de destination ; le paramètre x__% doit être un multiple de 16 et correspondre à un point se trouvant à l'intérieur de l'écran ; l'agrandissement ne se fera pas si vous donnez à x__% une valeur erronée et lib_rv__% retournera la valeur 1 ; par contre, lib_rv__% retournera la valeur 0 si vous donnez une valeur correcte à x__% et que tout se déroule bien.

- y__% :

Coordonnée Y du coin supérieur gauche du secteur de destination.

- screenadr__% :

Adresse de l'écran, qui est généralement celle de l'écran visible (XBIOS(2)) ; vous pouvez aussi entrer l'adresse de l'écran non-visible si vous vous en servez. On admet toutefois que le secteur à agrandir se trouve sur l'écran visible, et l'agrandissement s'affichera sur le même écran.

- value__% :

Facteur d'agrandissement, compris entre : value__% = 2 - 640

L'agrandissement ne se fera pas, et lib_rv__% retournera la valeur 1

- Si vous entrez un facteur d'agrandissement en dehors des limites ci-dessus
- Si le secteur une fois agrandi dépasse la taille de l'écran ou
- S'il se trouve partiellement en dehors de l'écran.

Si par contre vous entrez un facteur autorisé et si le secteur agrandi tient entièrement dans l'écran, lib_rv__% retourne la valeur 0.

Valeur retournée :

lib_rv__% =

Code d'erreur ; *lib_rv__%* retourne la valeur 0 si tous les paramètres sont correctes et si vous avez chargé le fichier **INLINE** accompagnant la procédure ; *lib_rv__%* retournera par contre un code d'erreur (et l'agrandissement ne se produira pas) dans l'un des cas suivants :

- Les coordonnées X du secteur de destination ne sont pas divisibles par 16 (*lib_rv__%* retourne la valeur 1) ;
- Le facteur d'agrandissement *value%* est inférieur à 2 ou supérieur à 640 (*lib_rv__%* retourne la valeur 1) ;
- Le secteur agrandi dépasse partiellement de l'écran ou se trouve complètement en dehors en ce dernier (*lib_rv__%* retourne la valeur 1) ;
- Le secteur agrandi est plus grand que l'ensemble de l'écran (*lib_rv__%* retourne la valeur 1) ;
- Vous avez oublié de charger le fichier **INLINE** (*lib_rv__%* retourne la valeur -1).

Description :

Cette procédure vous permet d'obtenir l'agrandissement d'un secteur quelconque de l'écran selon un facteur à déterminer ; le secteur à agrandir et sa représentation agrandie se trouvent tous deux sur le même écran.

Attention : ▽

la procédure **H_LOUPE** n'existe que pour la haute résolution.

Exemple : **EXAMPLES\PROGRAMS.HIG\H_LOUPE.LST**

MERGE : **GFA_GLIB.3_0\GFA_GLIB.HIG\H_LOUPE.LST**

```

DEFFILL 1,2,24      ! élaboration de l'image
PBOX 0,0,639,399
TEXT 30,32,"LOUPE"
ALERT 1," | Agrandir? | ",1,"START",button|
  REPEAT
    y%=15
    FOR x%=1 TO 80
      ,
      GOSUB gfa_h_loupe__(x__%,y__%,x__%+25,y__%+25,160,50,XBIOS(2),8)
      ,
    NEXT x__%
  ALERT 1," | Continuer? | ",1," OUI|NON",button|
UNTIL button|=2

```

Nous commençons par créer une image sur l'écran visible, pour pouvoir en agrandir une partie.

Le programme attend ensuite que l'utilisateur clique sur le bouton START de la boîte de dialogue qui s'affiche ; ceci étant fait, nous assistons à l'agrandissement d'un secteur d'écran se déplaçant progressivement vers la droite grâce à une boucle (effet ressemblant au 'scrolling').

H_LOUPE exerce son action sur un secteur de 26 x 26 pixels dont le contenu est agrandi à la position 160,50 (x__%,y__%). L'agrandissement se fait selon un facteur 8 (value__%), ce qui revient à dire que chaque pixel du dessin original est agrandi à 8 pixels.

Nous avons utilisé XBIOS(2) comme adresse de l'écran puisque le secteur à agrandir se trouve dans l'écran visible.

Une fois la boucle entièrement parcourue, le programme envoie une boîte de dialogue demandant si la démonstration doit se répéter (cliquer sur 'OUI') ou s'interrompre (cliquer sur 'NON').

Graphisme : L_FADEOF M_FADEOF H_FADEOF

Utilisation : Masquer une partie de l'écran à l'aide d'une couleur quelconque et de différentes façons.

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_FADEOF.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_FADEOF.LST
(moyenne résolution)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_FADEOF.LST
(haute résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_FADEOF.INL
(L_FADEOF.LST)

GFA_GLIB.3_0\GFA_GLIB.MED\M_FADEOF.INL
(M_FADEOF.LST)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_FADEOF.INL
(H_FADEOF.LST)

Exemples : EXAMPLES\PROGRAMS.LOW\L_FADEOF.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\H_FADEOF.LST
(haute résolution)

Syntaxe : GOSUB gfa_l_fadeof__(destadr__%,rows__%,col__%,
speed__%,options__%,pattern__%)

GOSUB gfa_m_fadeof__(destadr__%,rows__%,col__%,
speed__%,options__%,pattern__%)

GOSUB gfa_h_fadeof__(destadr__%,rows__%,col__%,
speed__%,options__%,pattern__%)

Paramètres :

- *destadr*__% :

Adresse de la ligne d'écran à partir de laquelle on souhaite masquer un nombre *rows*__% de lignes ; voici comment calculer l'adresse de la ligne :

➤ **En basse et moyenne résolution :**

Adresse = XBIOS(2) + numéro de la ligne * 160

➤ **En haute résolution :**

Adresse = XBIOS(2) + numéro de ligne * 80

Par exemple, pour masquer un secteur à partir de la ligne 10 en basse résolution, nous écrivons :

$$\text{Destadr}\% = \text{XBIOS}(2) + 10 * 160 = \text{XBIOS}(2) + 1600$$

Rappelons que XBIOS(2) vous donne l'adresse de début de l'écran.

- *rows*__% :

Nombre de lignes à masquer ; évidemment, rien ne sera masqué si vous entrez 0 ligne!

➤ **En basse et moyenne résolution :** 0 - 200 lignes

➤ **En haute résolution :** 0 - 400 lignes.

- *col*__% :

Couleur servant à remplir le secteur d'écran qui est masqué

➤ **En basse résolution :** 0 - 15 soit 16 couleurs différentes

➤ **En moyenne résolution :** 0 - 3 soit 4 couleurs différentes

➤ **En haute résolution :** 0 - 1 soit 2 couleurs différentes (noir/blanc).

- *speed*__% :

Vitesse du processus de masquage, comprise entre 0 (le plus rapide) et 65535 (le plus lent)

- options__% :

Numéro de l'effet de masque, à choisir parmi 13 effets possibles désignés par les valeurs 1 à 13, dont l'une est affectée au paramètre 'options__%' ; voici une brève description de ces 13 effets :

- | | |
|--------------------|--|
| options__% == 1 : | Le secteur d'écran concerné s'efface point par point et il est possible de rentrer un schéma précisant l'ordre de disparition des points |
| options__% == 2 : | Le secteur d'écran disparaît ligne par ligne, en commençant par celle du haut |
| options__% == 3 : | Le secteur d'écran disparaît ligne par ligne, en commençant simultanément en haut et en bas |
| options__% == 4 : | Le secteur d'écran disparaît ligne par ligne, en commençant par celle du bas |
| options__% == 5 : | Le secteur d'écran disparaît colonne par colonne (verticalement), en commençant par le bord gauche |
| options__% == 6 : | Le secteur d'écran disparaît colonne par colonne (verticalement), en commençant par le bord droit |
| options__% == 7 : | Le secteur d'écran disparaît colonne par colonne (verticalement), en commençant par les deux bords simultanément |
| options__% == 8 : | Le secteur d'écran disparaît tout en étant remplacé immédiatement par des bandes apparaissant simultanément en haut et en bas |
| options__% == 9 : | Le secteur d'écran disparaît tout en étant remplacé immédiatement par des lignes augmentant alternativement à droite et à gauche |
| options__% == 10 : | Le secteur d'écran semble disparaître derrière un store qui se baisse |
| options__% == 11 : | Le secteur d'écran est masqué par des bouts de lignes qui prennent peu à peu toute la place |

options__% == 12 : Le secteur d'écran disparaît vers le bas, par 'scrolling'

options__% == 13 : Le secteur d'écran disparaît vers le haut, par 'scrolling'

- *pattern*__% :

Motif du masque ; il convient d'entrer ici un nombre sur 16 bits (soit de 0 à 65535) qui détermine le processus de masquage dans les effets 1 à 8 mentionnés ci-dessus. Ce paramètre est ignoré lorsque vous utilisez les effets 9 à 13, et vous lui donnez donc une valeur quelconque, sans l'omettre pour autant.

Valeur retournée :

*lib_rv*__% =

Code d'erreur ; *lib_rv*__% retourne la valeur -1 si vous avez oublié de charger le fichier *INLINE* accompagnant la procédure et la valeur 0 si tout se déroule correctement.

Description :

Les procédures *L_FADEOF*, *M_FADEOF* et *H_FADEOF* permettent de masquer des secteurs horizontaux sur l'écran ou d'autres secteurs indiqués par leur adresse. Ceci rend plus attrayantes les démonstrations ou les présentations de graphiques. Pour que le processus se déroule normalement, il faut veiller à bien charger le bon procédure en fonction du degré de résolution de l'écran utilisé.

Exemple : EXAMPLES\PROGRAMS.LOW\L_FADEOF.LST

MERGE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_FADEOF.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

effect|=1 ! commençons par l'effet 1

GOSUB gfa_poppal__

```

GOSUB gfa_pload__("\EXAMPLES\STUFF\DEMO.PI1",XBIOS(2),0)
SGET image$ ! enregistrer dans un string
,
REPEAT
  ALERT 2,"      | Masque      | ",1," OUI ",button|
  ,
  HIDE
  GOSUB gfa_1_fadeof__(XBIOS(2),200,0,10000,effect|,&X1111000011110000)
  SHOW
  ,
  INC effect|          ! passons à l'effet suivant
  IF effect|=14
    effect|=1
  ENDIF
  ,
  ALERT 2," | Continuer?      | ",1," OUI      | NON ",button|
  SPUT image$
UNTIL button|=2
~XBIOS(6,L :V :palette$)

```

Nous commençons par indiquer le numéro du premier effet de masque utilisé (options__% = 1) puis nous sauvegardons la palette des couleurs d'origine à l'aide de POPPAL.

La procédure PLOAD nous sert à charger l'image de démonstration DEMO.PI1 dans l'écran (ou la mémoire d'écran) ainsi que sa palette de couleurs. Une boucle REPEAT... UNTIL permet de faire subir les 13 effets de masque à cette image.

Nous entrons XBIOS(12) comme adresse de début de l'écran et nous souhaitons masquer 200 lignes (soit tout l'écran) à l'aide du motif 1111000011110000 (en binaire) qui n'interviendra que pour les effets 1 à 8. Le fait de saisir cette valeur en mode binaire facilite la saisie et la modification du motif du masque.

Nous attribuons la valeur 10000 à 'speed__%' pour préciser la rapidité de la superposition du masque. La démonstration se poursuit jusqu'à ce que l'utilisateur clique sur le bouton NON du deuxième panneau. La palette des couleurs d'origine est restaurée avant de ressortir du programme.

Graphisme : OFF14

Utilisation : Masquer une partie de l'écran à l'aide d'une couleur quelconque quelle que soit la résolution ; le secteur d'écran concerné est masqué par de petits rectangles allant en diminuant.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\OFFXX.LST
(quelle que soit la résolution)

INLINE : Ne nécessite aucun fichier INLINE

Exemples : EXAMPLES\PROGRAMS.LOW\OFFXX.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\OFFXX.LST
(haute résolution)

Syntaxe : GOSUB gfa_off14__(x1__%,y1__%,x2__%,y2__%,
speed__%,col__%)

Paramètres :

- x1__%,y1__% :

Coordonnées du coin supérieur gauche du secteur à masquer

- x2__%,y2__% :

Coordonnées du coin inférieur droit du secteur à masquer. Remarquez qu'il est tout à fait possible d'intervertir x1__% et x2__%, ainsi que y1__% et y2__% puisque la procédure est capable de rectifier cette interversion.

Il convient cependant de ne pas dépasser les limites suivantes :

x1__%/x2__% y1__%/y2__%

- | | | |
|------------------------|---------|---------|
| > Basse résolution : | 0 - 319 | 0 - 199 |
| > Moyenne résolution : | 0 - 639 | 0 - 199 |
| > Haute résolution : | 0 - 639 | 0 - 399 |

Aucune erreur ne se produit si vous entrez des valeurs en dehors de ces limites.

- *speed*__% :

Vitesse du processus de masquage : plus cette valeur est grande plus le processus est lent ; cette vitesse dépend également de la taille du secteur à masquer.

- *col*__% :

Couleur servant à remplir le secteur d'écran qui est masqué

- **En basse résolution :** 0 - 15 soit 16 couleurs différentes
- **En moyenne résolution :** 0 - 3 soit 4 couleurs différentes
- **En haute résolution :** 0 - 1 soit 2 couleurs différentes (noir/blanc).

Valeur retournée :

Aucune

Description :

La procédure OFF14 permet de masquer un secteur d'écran quelconque : il est possible de paramétrer la rapidité du processus et la couleur servant à masquer le secteur concerné.

En ce qui concerne la couleur, il convient d'observer les mêmes règles qu'avec l'instruction COLOR du GfA Basic.

Exemple : EXAMPLES\PROGRAMS.LOW\OFFXX.LST

MERGE : GFA_GLIB.3_0\GFA_GLIB.ALL\OFXX.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```
GOSUB gfa_poppal__
```

```
palette$=lib_rv__$
```

```
GOSUB gfa_pload("EXAMPLES\STUFF\DEMO.PI1",XBIO$(2),0)
```

```
SGET image$ ! enregistrer dans un string
```

```

ALERT 2,"|Routine de masque| ",0," 14 | 15 | 16 ",button|
FOR i|=1 TO 20
  x1&=RANDOM(320)
  x2&=RANDOM(320)
  y1&=RANDOM(200)
  y2&=RANDOM(200)
  col|=RANDOM(16)
  IF button|=1
    GOSUB gfa_off14__(x1&,y1&,x2&,y2&,0,col|)
  ELSE
    IF button|=2
      GOSUB gfa_off15__(x1&,y1&,x2&,y2&,0,col|)
    ELSE
      GOSUB gfa_off16__(x1&,y1&,x2&,y2&,0,col|)
    ENDIF
  ENDIF
NEXT i|
ALERT 2,"| Continuer? |",1," OUI | NON ",button|
SPUT image$
UNTIL button|=2
~XBIOS(6,L :V :palette$)

```

Nous commençons par sauvegarder la palette des couleurs d'origine à l'aide de POPPAL, après quoi la procédure PLOAD nous sert à charger l'image de démonstration DEMO.PI1 dans l'écran (ou la mémoire d'écran) ainsi que sa palette de couleurs.

Une boucle REPEAT... UNTIL permet à l'utilisateur de lancer l'une des trois procédures OFF14, OFF15 ou OFF16 en cliquant sur le bouton correspondant dans le panneau qui s'affiche. Ceci étant fait, on voit 20 secteurs différents disparaître derrière le masque correspondant au procédé choisi ; la position et la couleur du masque s'appliquant à ces vingt secteurs sont sélectionnées par un générateur de nombres aléatoires (RANDOM).

On peut répéter cette démonstration autant de fois qu'on le souhaite ; il convient de cliquer sur le bouton 'NON' du deuxième panneau pour mettre fin à la démonstration.

Graphisme : OFF15

Utilisation : Masquer une partie de l'écran à l'aide d'une couleur quelconque quelle que soit la résolution ; le secteur d'écran concerné est masqué par des diagonales.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\OFFXX.LST
(quelle que soit la résolution)

INLINE : Ne nécessite aucun fichier INLINE

Exemples : EXAMPLES\PROGRAMS.LOW\OFFXX.LST
(FPbasse résolution)

EXAMPLES\PROGRAMS.HIG\OFFXX.LST
(haute résolution)

Syntaxe : GOSUB gfa_off15__(x1__%,y1__%,x2__%,y2__%,
speed__%,col__%)

Paramètres :

- x1__%,y1__% : voir OFF14

- x2__%,y2__% : voir OFF14

- speed__% : voir OFF14

- col__% : voir OFF14

Valeur retournée :

aucune

Description : voir OFF14

Exemple : voir OFFXX.LST avec OFF14

Graphisme : OFF16

Utilisation : Masquer une partie de l'écran à l'aide d'une couleur quelconque quelle que soit la résolution ; le secteur d'écran concerné est masqué ligne par ligne, en partant du haut et du bas.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\OFFXX.LST
(quelle que soit la résolution)

INLINE : Ne nécessite aucun fichier INLINE

Exemples : EXAMPLES\PROGRAMS.LOW\OFFXX.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\OFFXX.LST
(haute résolution)

Syntaxe : GOSUB gfa_off16__(x1__%,y1__%,x2__%,y2__%,
speed__%,col__%)

Paramètres :

- *x1__%*,*y1__%* : voir OFF14

- *x2__%*,*y2__%* : voir OFF14

- *speed__%* : voir OFF14

- *col__%* : voir OFF14

Valeur retournée :

Aucune

Description : voir OFF14

Exemple : voir OFFXX.LST avec OFF14.

Graphisme : L_FADEOV M_FADEOV H_FADEOV

Utilisation : Prendre un secteur d'écran pour le 'surimprimer' ou l'afficher sur une autre partie de l'écran

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_FADEOV.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_FADEOV.LST
(moyenne résolution)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_FADEOV.LST
(haute résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_FADEOV.INL
(L_FADEOV.LST)

GFA_GLIB.3_0\GFA_GLIB.MED\M_FADEOV.INL
(M_FADEOV.LST)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_FADEOV.INL
(H_FADEOV.LST)

Exemples : EXAMPLES\PROGRAMS.LOW\L_FADEOV.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\H_FADEOV.LST
(haute résolution)

Syntaxe : GOSUB gfa_l_fadeov__(destadr__%,sourceadr__%,
rows__%,speed__%,options__%,pattern__%)

GOSUB gfa_m_fadeov__(destadr__%,sourceadr__%,
rows__%,speed__%,options__%,pattern__%)

GOSUB gfa_h_fadeov__(destadr__%,sourceadr__%,
rows__%,speed__%,options__%,pattern__%)

Paramètres :

- *destadr__%* :

Adresse de la ligne d'écran à partir de laquelle on souhaite afficher ou surimprimer un nombre *rows__%* de lignes ; voici comment calculer l'adresse de la ligne :

➤ En basse et moyenne résolution :

Adresse = XBIOS(2) + numéro de la ligne * 160

➤ En haute résolution :

Adresse = XBIOS(2) + numéro de ligne * 80

Par exemple, pour afficher/surimprimer un secteur à partir de la ligne 10 en basse résolution, nous écrivons :

$$\text{destadr_}\% = \text{XBIOS}(2) + 10 * 160 = \text{XBIOS}(2) + 1600$$

Rappelons que XBIOS(2) vous donne l'adresse de début de l'écran.

- *sourceadr__%* :

Adresse de la ligne à partir de laquelle on va transférer un nombre '*rows__%*' de ligne vers l'adresse '*destadr__%*' ; en clair : le secteur d'écran se trouvant sous *sourceadr__%* va être transférer vers *destadr__%* pour affichage ou surimpression.

On calcule cette adresse de la même façon que *destadr__%* (voir ci-dessus).

- *rows__%* :

Nombre de lignes à transférer ; évidemment, rien ne sera transféré si vous entrez 0 ligne!

➤ En basse et moyenne résolution : 0 - 200 lignes

➤ En haute résolution : 0 - 400 lignes.

- *speed__%* :

Vitesse du processus de transfert/surimpression, comprise entre 0 (le plus rapide) et 65535 (le plus lent)

- options__% :

Numéro de l'effet de transfert/surimpression, à choisir parmi 13 effets possibles désignés par les valeurs 1 à 13, dont l'une est affectée au paramètre options__% ; voici une brève description des ces 13 effets :

- options__% == 1 : Le nouveau secteur d'écran apparaît point par point et il est possible de rentrer un schéma précisant l'ordre d'apparition des points
- options__% == 2 : Le secteur d'écran apparaît ligne par ligne, en commençant par celle du haut
- options__% == 3 : Le secteur d'écran apparaît ligne par ligne, en commençant simultanément en haut et en bas
- options__% == 4 : Le secteur d'écran apparaît ligne par ligne, en commençant par celle du bas
- options__% == 5 : Le secteur d'écran apparaît colonne par colonne (verticalement), en commençant par le bord gauche
- options__% == 6 : Le secteur d'écran apparaît colonne par colonne (verticalement), en commençant par le bord droit
- options__% == 7 : Le secteur d'écran apparaît colonne par colonne (verticalement), en commençant par les deux bords simultanément
- options__% == 8 : Le secteur d'écran apparaît bande par bande, apparaissant simultanément en haut et en bas
- options__% == 9 : Le secteur d'écran apparaît par des lignes augmentant alternativement à droite et à gauche
- options__% == 10 : Le secteur d'écran semble apparaître comme à travers un store
- options__% == 11 : Le secteur d'écran apparaît par des bouts de lignes qui prennent peu à peu toute la place

options__% == 12 : Le secteur d'écran apparaît en 'montant' du bas, par 'scrolling'

options__% == 13 : Le secteur d'écran apparaît en 'descendant' d'en haut, par 'scrolling'

Ces différents effets ressemblent fortement à ceux qui servent à faire disparaître un secteur d'écran avec L_FADEOF, M_FADEOF ou H_FADEOF.

- pattern__% :

Motif de surimpression ; il convient d'entrer ici un nombre sur 16 bits (soit de 0 à 65535) qui détermine le processus d'affichage/surimpression dans les effets 1 à 8 mentionnés ci-dessus. Ce paramètre est ignoré lorsque vous utilisez les effets 9 à 13, et vous lui donnez donc une valeur quelconque, sans l'omettre pour autant.

Valeur retournée :

lib_rv__% =

Code d'erreur ; lib_rv__% retourne la valeur -1 si vous avez oublié de charger le fichier INLINE accompagnant la procédure et la valeur 0 si tout se déroule correctement.

Description :

les procédures L_FADEOV, M_FADEOV et H_FADEOV permettent d'afficher ou de surimprimer, en les déplaçant vers une nouvelle adresse, des secteurs horizontaux de l'écran ou d'autres secteurs indiqués par leur adresse.

Vous pouvez ainsi afficher des images ou des secteurs d'image se trouvant dans la mémoire vive ou qui y sont en cours d'élaboration. Lorsque l'écran contient déjà quelque chose à l'endroit de destination, ce contenu disparaît par 'surimpression' de la nouvelle image. Ceci rend plus attrayantes les démonstrations ou les présentations de graphiques. Pour que le processus se déroule normalement, il faut veiller à bien charger le bon procédure en fonction du degré de résolution de l'écran utilisé.

Exemple : EXAMPLES\PROGRAMS.LOW\L_FADEOV.LST

MERGE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_FADEOV.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```
effect|=1                                ! commençons par l'effet 1
,
GOSUB gfa_poppal__
palette$=lib_rv__$
GOSUB gfa_pload("\EXAMPLES\STUFF\DEMO.PI1",XBIOS(2),0)
SGET image$                               ! enregistrer dans un string
,
REPEAT
  CLS
  FOR i&=0 TO 319 STEP 4                   ! délimitation du secteur à effacer
    COLOR i&/4 MOD 16                     ! qui va céder la place à la nouvelle
    LINE 0,i&*0.625,i&,199                ! image se trouvant sous image$
    LINE 319,199-i&*0.625,319-i&,0        ! et ce, par surimpression
    LINE i&,199,319,199-i&*0.625
    LINE 319-i&,0,0,i&*0.625
  NEXT i&
  ALERT 2," | | Surimpression? ",1," OUI ",button|
  ,
  HIDE
  GOSUB gfa_1_fadeov__(XBIOS(2),V:image$,200,10000,effect|,&HF0F0)
  SHOW
  ,
  INC effect|
  IF effect|=14                           ! autre effet de surimpression
  effect|=1
  ENDIF
  ,
  ALERT 2," | Continuer? | ",1," OUI | NON ",button|
  SPUT image$
UNTIL button|=2
~XBIOS(6,L :V :palette$)
```

Nous commençons par indiquer le numéro du premier effet utilisé (options__% = 1) puis nous sauvegardons la palette des couleurs d'origine à l'aide de POPPAL.

La procédure PLOAD nous sert à charger l'image de démonstration DEMO.PI1 dans l'écran (ou la mémoire d'écran) ainsi que sa palette de couleurs. Qui plus est, nous enregistrons le contenu de l'écran dans le string 'image\$' grâce à SGET, si bien que l'image se trouve maintenant dans la mémoire à partir de l'adresse que nous recevons par V:image\$.

Attention : ▽

Nous utilisons SGET et non GET. En effet, si l'on enregistre un écran ou un secteur d'écran dans un string à l'aide de GET, il faudra utiliser V:image\$+6 pour retrouver son adresse et le secteur d'écran devra obligatoirement avoir la même largeur que l'écran lui-même.

Une boucle REPEAT... UNTIL permet de délimiter le secteur appelé à céder la place à la nouvelle image qui vient se surimprimer sur le contenu précédent (DEMO.PI1) dès que l'utilisateur clique sur le bouton confirmant la surimpression. On peut répéter cette opération autant de fois qu'on le souhaite, ce qui fera d'ailleurs défiler les 13 effets de surimpression possibles.

Nous entrons XBIOS(2) comme adresse de destination destadr__% car la nouvelle image doit venir s'afficher à partir de la première ligne de l'écran. Comme nous souhaitons surimprimer l'écran complet, nous entrons 200 lignes (totalité de l'écran) sous 'rows__%', en utilisant le motif F0F0 (écriture hexadécimale).

Pour préciser la rapidité de la surimpression, nous utilisons speed__% en le faisant varier à chaque nouveau parcourt de la boucle. La démonstration se poursuit jusqu'à ce que l'utilisateur clique sur le bouton 'NON' du deuxième panneau. La palette des couleurs d'origine est restaurée avant de ressortir du programme.

Graphisme : L_GREYS M_GREYS

Utilisation : Transformer une palette de couleurs en un dégradé de gris ; chaque couleur est remplacée par un gris correspondant, plus ou moins soutenu, accompagné d'une valeur de luminosité. Une fois la procédure L_GREYS (ou M_GREYS) lancé, la palette des couleurs d'origine ne contient plus que des valeurs grisées.

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_GREYS.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_GREYS.LST
(moyenne résolution)

INLINE : Ne nécessite aucun fichier INLINE.

Exemples : EXAMPLES\PROGRAMS.LOW\L_GREYS.LST
(basse résolution)

Syntaxe : GOSUB gfa_l_greys__(paladr__%,value__%)
GOSUB gfa_m_greys__(paladr__%,value__%)

Paramètres :

- *paladr__%* :

Adresse d'une palette de couleurs se composant de 16 mots (quatre mots suffisent avec M_GREYS), contenant les informations relatives aux 16 (avec L_GREYS) ou aux 4 (avec M_GREYS) registres de couleur. Le format de la palette de couleurs à indiquer correspond à celui à utiliser lorsqu'on appelle XBIOS 6 (setpalette).

Voici la structure de cette palette de couleurs :

à un endroit quelconque de la mémoire vive, on doit trouver 16 words (soit 32 bits) entrés consécutivement l'un derrière l'autre ; le premier mot contient les informations relatives au registre 0, le deuxième mot contient

les informations relatives au registre 2 etc... Chacun de ces mots écrits en binaire se décompose de la façon suivante :

xxxxrrrxgggxbbb

x = bit non utilisé (état indifférent)

rrr = 3 bits pour les valeurs rouges

ggg = 3 bits pour les valeurs vertes (anglais : green)

bbb = 3 bits pour les valeurs bleues.

Si l'on s'en tient à l'instruction SETCOLOR 0,7,2,3 le mot devrait avoir l'aspect suivant (les bits inutilisés étant mis sur 0) :

0000011100100011

> en écriture hexadécimale : 0723

> en écriture décimale : 1827

- *value__%* :

Facteur d'intensité des couleurs, compris entre les valeurs 0 et 7 ; les autres valeurs seront traitées à l'aide de MOD 8, si bien qu'il en résultera forcément une valeur comprise entre 0 et 7.

Le paramètre *value__%* représente un facteur de luminosité qui permet d'influencer la luminosité de toute la palette finale : les valeurs faibles donnent des teintes plus sombres que les valeurs élevées.

Valeur retournée :

aucune

Description :

Ces deux procédures L_GREYS et M_GREYS permettent de transformer la palette actuelle des couleurs en un dégradé de gris, mais il ne s'agit absolument pas de convertir une image pour la rendre affichable sur un moniteur monochrome. Il n'existe pas de procédure correspondant pour la haute résolution, puisque la conversion d'une palette de couleurs n'a de sens que dans l'affichage en couleur!

Il existe certes une procédure par degré de résolution, mais la procédure L_GREYS peut s'utiliser avec la moyenne résolution. La différence étant

que L_GREYS est conçu pour traiter une palette complète de 16 mots alors que M_GREYS ne traitera que les quatre premiers mots (car il n'y a que quatre couleurs). M_GREYS devrait logiquement s'avérer plus rapide que L_GREYS. Il n'est par contre pas intéressant d'utiliser M_GREYS en basse résolution, car seuls les quatze premiers registres des couleurs seront convertis en grisés.

Le facteur de luminosité valeur__% détermine la luminosité de l'ensemble de la palette résultante. Cette valeur s'additionne avec les valeurs de gris calculées auparavant, si bien que plus la valeur est élevée, plus le gris sera clair.

La palette d'origine des couleurs demeure inchangée durant la conversion en grisés : il n'est donc pas nécessaire de la restaurer après l'appel de la procédure.

Exemple : EXAMPLES\PROGRAMS.LOW\L_GREYS.LST

MERGE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_GREYS.LST

 GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST

 GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```
GOSUB gfa_poppal__
opalette$=lib_rv__$
GOSUB gfa_pload__("\EXAMPLES\STUFF\DEMO.PI1",XBIOS(2),0)
GOSUB gfa_poppal__
npalette$=lib_rv__$
luminosite|=0                ! facteur de luminosité d'abord sur 0
REPEAT
  ~XBIOS(6,L :V :npalette$)  ! déterminer la palette des couleurs
  ALERT 2,"|transformer en grisés? | ",1," OUI ",button|
  ,
  GOSUB gfa_l_greys__(V :npalette$,luminosite|)
  ,
  IF luminosite|=6            ! facteur de luminosité pour le
  luminosite|=0                ! prochain parcours de la boucle
  ELSE
  INC luminosite|
  ENDIF
  ALERT 2,"| Continuer? |",1," OUI | NON ",button|
```



```
UNTIL button|=2  
~XBIOS(6,L :V :opalette$)
```

Nous commençons par sauvegarder la palette des couleurs d'origine grâce à POPPAL dans la variable opalette\$, puis nous nous servons de PLOAD pour charger l'image de démonstration se trouvant dans DEMO.PI1 ainsi que sa palette de couleurs. Un nouvel appel de la procédure POPPAL permet de fixer la palette de l'image Degas/Elite de DEMO.PI1 et de l'enregistrer dans la variable-string npalette\$.

Dans la boucle REPEAT... UNTIL qui suit, nous fixons d'abord la palette originale de l'image ; la procédure L_GREYS est lancé lorsque l'utilisateur

clique sur le bouton 'OUI' du panneau qui apparaît, ce qui lance la conversion de la palette des couleurs en une gamme de gris.

On peut répéter autant de fois qu'on le souhaite cette démonstration, le facteur de luminosité variant à chaque fois.

Pour sortir de la boucle, il suffit de cliquer sur le bouton 'NON' du deuxième panneau ; avant de se terminer, le programme prend soin de restaurer la palette d'origine par le biais de 'opalette\$'.

```

GOSUB gfa_poppal__
opalette$=lib_rv__$
GOSUB gfa_pload__("\EXAMPLES\STUFF\DEMO.PI1",XBIOS(2),0)
GOSUB gfa_poppal__
npalette$=lib_rv__$
,
speed|=8                      ! vitesse du noircissement puis du réaffichage
REPEAT
  ALERT 2,"|noircir/réafficher| ",1," OUI ",button|
  ,
  GOSUB gfa_l_cpoff__(V :npalette$,speed|)
  ,
  GOSUB gfa_l_cpon__(V :npalette$,speed|)
  ,
  IF speed|=0                  ! déterminer la vitesse du processus pour le
  speed|=8                     ! prochain parcours de la boucle
  ELSE
  DEC speed|
  ENDIF
  ALERT 2,"| Continuer? |",1," OUI | NON ",button|
UNTIL button|=2
~XBIOS(6,L :V :opalette$)

```

Nous commençons par sauvegarder la palette des couleurs d'origine grâce à POPPAL dans la variable 'opalette\$', puis nous nous servons de PLOAD pour charger l'image de démonstration se trouvant dans DEMO.PI1 ainsi que sa palette de couleurs.

Après la confirmation donnée en cliquant dans le bouton 'OUI' de la boîte de dialogue, le programme lance L_CPOFF puis L_CPON l'un à la suite de l'autre, ce qui fait disparaître puis réapparaître l'image. A chacun de ces appels, nous rappelons la palette des couleurs telle qu'elle était après le chargement de l'image à l'aide de POPPAL. Cette palette est sauvegardée dans la variable-string 'npalette\$', et l'appel d'un de ces procédures ne la modifie aucunement, si bien que nous pouvons la réutiliser à volonté.

Le tout se déroule dans une boucle REPEAT... UNTIL, en faisant varier la vitesse d'exécution à chaque parcours de la boucle.

Pour sortir de la boucle, il suffit de cliquer sur le bouton 'NON' de la deuxième boîte de dialogue ; avant de se terminer, le programme prend soin de restaurer la palette d'origine qui se trouve sous 'opalette\$'.

Graphisme L_CPOFF_CPOFF M_CPOFF

Utilisation : Faire disparaître une image en faisant passer progressivement tous ses registres de couleur au noir.

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_CPOFF.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_CPOFF.LST
(moyenne résolution)

INLINE : Ne nécessite aucun fichier INLINE.

Exemples : EXAMPLES/PROGRAMS.LOW\L_CPOXX.LST
(basse résolution)

Syntaxe : GOSUB gfa_l_cpoff__(paladr__%,speed__%)

GOSUB gfa_m_cpoff__(paladr__%,speed__%)

Paramètres :

- paladr__% :

Adresse d'une palette de couleurs se composant de 16 mots (quatre mots suffisent avec M_CPOFF), contenant les informations relatives aux 16 (avec L_CPOFF) ou aux 4 (avec M_CPOFF) registres de couleur. Le format de la palette de couleurs est le même que pour L_GREYS et M_GREYS.

- speed__% :

Vitesse du processus de noircissage : plus la valeur est petite, plus le processus est rapide et inversement : plus la valeur est grande, plus le processus est lent (valeur minimale :0).

La valeur figurant dans la procédure engendre le même délai entre les étapes du processus de noircissage qu'avec l'instruction PAUSE du GfA Basic.

Valeur retournée :

aucune

Description :

Ces deux procédures L_CPOFF et M_CPOFF permettent d'amener progressivement jusqu'au noir les différentes couleurs composant la palette actuelle (ou une autre, mais cela n'a guère de sens). La palette d'origine des couleurs demeure inchangée durant ce processus : il n'est donc pas nécessaire de la restaurer après l'appel de la procédure si on en a encore besoin pour d'autres tâches.

Il existe certes une procédure par degré de résolution, mais la procédure L_CPOFF peut s'utiliser avec la moyenne résolution. La différence étant que L_CPOFF est conçu pour traiter une palette complète de 16 mots alors que M_CPOFF ne traitera que les quatre premiers mots (car il n'y a que quatre couleurs). M_CPOFF devrait logiquement s'avérer plus rapide que L_CPOFF. Il n'est par contre pas intéressant d'utiliser M_CPOFF en basse résolution, car seuls les quatre premiers registres des couleurs vireraient au noir.

Il est possible d'utiliser ces deux procédures conjointement avec L_CPON et M_CPON pour passer d'une image à l'autre : on fait disparaître la première image en la noircissant et on élabore la deuxième image sur l'écran noir (l'apparition de l'image reste invisible puisque toutes les couleurs sont fixées sur le noir) ; il suffit ensuite de 'révéler' cette deuxième image en lui attribuant ses couleurs. Cela ressemble alors à un fondu-enchaîné.

Ce genre de procédure n'existe pas pour la haute résolution, puisque le noircissement de la palette des couleurs serait impossible.

Exemple : EXAMPLES\PROGRAMS.LOW\L_CPOXX.LST

MERGE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_CPOFF.LST

GFA_GLIB.3_0\GFA_GLIB.LOW\L_CPON.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

Graphisme : L_CPON M_CPON

Utilisation : Faire réapparaître une image en faisant repasser progressivement tous ses registres du Noir jusqu'à leur couleur normale (correspondant à la palette des couleurs).

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_CPON.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_CPON.LST
(moyenne résolution)

INLINE : Ne nécessite aucun fichier INLINE.

Exemples : EXAMPLES\PROGRAMS.LOW\L_CPOXX.LST
(basse résolution)

Syntaxe : GOSUB gfa_l_cpon__(paladr__%,speed__%)
GOSUB gfa_m_cpon__(paladr__%,speed__%)

Paramètres :

- *paladr__%* :

Adresse d'une palette de couleurs se composant de 16 mots (quatre mots suffisent avec M_CPON), contenant les informations relatives aux 16 (avec L_CPON) ou aux 4 (avec M_CPON) registres de couleur. Le format de la palette de couleurs est le même que pour L_GREYS et M_GREYS.

- *speed__%* :

Vitesse du processus de remontée de l'image : plus la valeur est petite, plus le processus est rapide et inversement : plus la valeur est grande, plus le processus est lent (valeur minimale :0).

La valeur figurant dans la procédure engendre le même délai entre les étapes du processus de remontée de l'image qu'avec l'instruction PAUSE du GfA Basic.

Valeur retournée :

Aucune

Description :

Ces deux procédures L_CPON et M_CPON permettent de faire 'remonter' progressivement, depuis le noir, les différentes couleurs composant la palette actuelle de l'image. La palette d'origine des couleurs demeure inchangée durant ce processus : il n'est donc pas nécessaire de la restaurer après l'appel de la procédure si on en a encore besoin pour d'autres tâches. Tout cela n'a de sens que si l'ensemble des registres de couleur étaient sur le noir avant de lancer l'un de ces procédures.

Il existe une procédure pour chaque degré de résolution, sauf pour la haute résolution, puisqu'il serait impossible de faire 'monter' la palette des couleurs.

La procédure L_CPON peut s'utiliser aussi en moyenne résolution. La différence étant que L_CPON est conçu pour traiter une palette complète de 16 mots alors que M_CPON ne traitera que les quatre premiers mots (car il n'y a que quatre couleurs). M_CPON devrait logiquement s'avérer plus rapide que L_CPON. Il n'est par contre pas intéressant d'utiliser M_CPON en basse résolution, car seuls les quatre premiers registres des couleurs seraient traités.

Il est possible d'utiliser ces deux procédures conjointement avec L_CPOFF et M_CPOFF pour passer d'une image à l'autre : on fait disparaître la première image en la noircissant et on élabore la deuxième image sur l'écran noir (l'apparition de l'image reste invisible puisque toutes les couleurs sont fixées sur le noir) ; il suffit ensuite de 'révéler' cette deuxième image en lui attribuant ses couleurs. Cela ressemble alors à un fondu-enchaîné.

Exemple : voir L_CPOXX.LST sous L_CPOFF et M_CPOFF.

Graphisme : L_SCONV M_SCONV H_SCONV

Utilisation : Conversion d'une police de 40 caractères qui sera ensuite utilisée avec les procédures L_HSCROL, M_HSCROL ou H_SROL pour faire défiler un texte .

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_SCONV.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_SCONV
LST(moyenne résolution)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_SCONV.LST
(haute résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_SCONV.INL
(L_SCONV.LST)

GFA_GLIB.3_0\GFA_GLIB.MED\M_SCONV.INL
(M_SCONV.LST)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_SCONV.INL
(H_SCONV.LST)

Exemples : EXAMPLES\PROGRAMS.LOW\L_SCOHSC.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\H_SCOHSC.LST
(haute résolution)

Syntaxe : GOSUB gfa_l_sconv__(sourceadr__%,destadr__%)
GOSUB gfa_m_sconv__(sourceadr__%,destadr__%)
GOSUB gfa_h_sconv__(sourceadr__%,destadr__%)

Paramètres :

- *sourceadr__%* :

Adresse de la ligne sur laquelle commence la première série de 20 caractères de la police ; on calcule cette adresse de la même façon que sous L_OFFxx, M_OFFxx ou H_OFFxx.

- *destadr__%* :

Adresse d'un buffer qui recevra les données correspondant à la police de caractères convertie ; la masse d'informations nécessaires pour représenter une police de caractères varie en fonction du degré de résolution de l'écran, ce qui fait aussi varier la taille qu'il convient de donner au buffer :

- **basse résolution** : 81920 octets
- **moyenne résolution** : 30720 octets
- **haute résolution** : 30720 octets

Il est possible de sauvegarder les polices de caractères après leur conversion, et le nombre des octets vous indique la taille des fichiers à prévoir.

Valeur retournée :

lib_rv__% = code d'erreur ; lib_rv__% retourne la valeur -1 si vous avez oublié de charger le fichier INLINE accompagnant la procédure et la valeur 0 si tout se déroule correctement.

Description :

la procédure L_SCONV sert à convertir une police de caractères de 16 couleurs pour la réutiliser avec L_HSCROL.

La procédure M_SCONV sert à convertir une police de caractères de 4 couleurs pour la réutiliser avec M_HSCROL.

La procédure H_SCONV sert à convertir une police de caractères de 2 couleurs pour la réutiliser avec H_HSCROL.

Les polices ainsi converties doivent être utilisée sous la procédure corespondant : il est impossible par exemple d'utiliser une police convertie à l'aide de L_SCONV avec la procédure M_HSCROL.

Lorsque l'on souhaite créer une nouvelle police de caractères, il convient de le faire dans la résolution adéquate. Attention : ces procédures ne convertissent que 40 caractères, qui doivent obligatoirement être alignés l'un contre l'autre sur deux lignes de 20 signes.

La taille des caractères varie en fonction de la résolution :

- > **basse résolution** : 16 x 16 pixels
- > **moyenne résolution** : 32 x 16 pixels
- > **haute résolution** : 32 x 32 pixels

Les caractères doivent être alignés dans l'ordre suivant :

ABCDEFGHIJKLMNOPQRSTUVWXYZ

UVWXYZ0123456789!..:

Vous trouverez un exemple de police de caractères dans le fichier DEMO.PI1, qui contient une police de ce genre conçue pour la basse résolution.

Exemple : EXAMPLES\PROGRAMS.LOW\SCOHSC.LST

MERGE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_SCONV.LST

GFA_GLIB.3_0\GFA_GLIB.LOW\L_HSCROL.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```
DIM buffer_$(20479)           !installer un buffer de 81920 octets
```

```
GOSUB gfa_poppal_
```

```
palette$=lib_rv_
```

```
GOSUB gfa_pload_("&EXAMPLES\STUFF\DEMO.PI1",XBIOS(2),0)
```

```
GOSUB gfa_l_sconv_(XBIOS(2)+168*160,V:buffer_$(0))
```

```

' BSAVE "FONT2.FTL",V :buffer__%(0),81920      !sauvegarder la police de
                                                caractères
'
txt__$="appuyez sur une touche pour interrompre cette démonstration"
'
GOSUB gfa_1_hscrol__(XBIOS(2)+10*160,V :buffer__%(0),txt__$,0)
'
~XBIOS(6,L :V :palette$)

```

Ce petit programme de démonstration sert à la fois pour L_SCONV et pour L_HSCROL. Nous commençons par installer un buffer de 81920 octets qui recevra par la suite la police de caractères convertie, puis nous sauvegardons la palette des couleurs d'origine à l'aide de POPPAL.

La procédure PLOAD nous sert à charger l'image de démonstration DEMO.PI1 dans la mémoire d'écran (adresse XBIOS(2)) ainsi que sa palette de couleurs.

L'image de démonstration contient une police de caractères, à partir de la ligne 168, police que nous allons convertir (deuxième appel de L_SCONV) pour la reprendre ensuite sous L_HSCROL.

'sourceadr__%' contient l'adresse de la 168ème ligne de l'écran (adresse de début XBIOS(2) + 160 octets par ligne * 168 lignes)

'destadr__%' contient l'adresse de destination de la police convertie, qui est, comme nous l'avons dit, l'adresse du buffer commençant à 'V :buffer__%(0)'.

Les caractères étant convertis, nous pouvons les sauvegarder à l'aide de l'instruction BSAVE afin de pouvoir éventuellement nous en resservir par la suite. Il est ainsi possible de se constituer une collection de polices de caractères déjà converties, que l'on pourra appeler directement dans un programme pour la faire tourner sous une des procédures HSCROL.

La sauvegarde n'a pas lieu dans notre exemple ci-dessus, puisque les lignes BSAVE sont introduites par une apostrophe caractérisant les commentaires.

Après la conversion, nous entrons un texte dans la variable-string 'txt__\$' et nous lançons la procédure L_HSCROL. A partir de ce moment, le texte défile continuellement grâce à une boucle sans fin (options__%=0). Seule possibilité d'en ressortir : appuyer sur une touche quelconque, ce qui

permet de ressortir immédiatement de L_HSCROL. Nous prenons soin de restaurer la palette des couleurs d'origine avant de sortir du programme.

Graphisme : L_HSCROL M_HSCROL H_HSCROL

Utilisation : Ces procédures permettent de faire défiler horizontalement (scrolling) un texte mémorisé dans un string, en utilisant une police de caractères convertie à l'aide de L_SCONV, M_SCONV ou H_SCONV.

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_HSCROL.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_HSCROL.LST
(moyenne résolution)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_HSCROL.LST
(haute résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_HSCROL.INL
(L_HSCROL.LST)

GFA_GLIB.3_0\GFA_GLIB.MED\M_HSCROL.INL
(M_HSCROL.LST)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_HSCROL.INL
(H_HSCROL.LST)

Exemples : EXAMPLES\PROGRAMS.LOW\L_SCOHSC.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\H_SCOHSC.LST
(haute résolution)

Syntaxe : GOSUB gfa_l_hscrol__(destadr__%,sourceadr__%,
txt__\$,options__%)

GOSUB gfa_m_hscrol__(destadr__%,sourceadr__%,
txt__\$,options__%)

GOSUB gfa_h_hscrol__(destadr__%,sourceadr__%,
txt__\$,options__%)

Paramètres :

- *destadr*__% :

Adresse de la ligne de pixels à partir de laquelle on souhaite faire défiler le texte ; cette adresse se calcule exactement comme sous les procédures L_OFF, M_OFF ou H_OFF.

- *sourceadr*__% :

Adresse des données de la police de caractères convertie ; comme les procédures HSCROL tournent obligatoirement avec des polices de caractères converties à l'aide des procédures SCONV, il convient d'indiquer ici l'adresse à laquelle se trouve une telle police dans la mémoire vive (en tenant bien compte du degré de résolution!).

- *txt*__\$:

Chaîne de caractères qui va défiler et qui peut avoir une longueur (quasiment) quelconque.

- *options*__% :

Ce paramètre sert à préciser la durée pendant laquelle le texte va défiler ; il doit obligatoirement prendre une des valeurs suivantes :

- *options*__% == 0 : Le texte défile sans arrêt à l'écran, grâce à une boucle sans fin ; pour en ressortir, il faut appuyer sur une touche quelconque ;
- *options*__% < 0 : Lorsque le paramètre '*options*__%' a une valeur négative, le texte ne défile qu'une fois à l'écran, et il est impossible d'interrompre ce défilement ;
- *options*__% > 0 : Lorsque le paramètre '*options*__%' a une valeur positive, le texte ne défile également qu'une fois à l'écran, mais il est possible d'interrompre ce défilement avant qu'il ne le fasse tout seul.

Valeur retournée :

`lib_rv__%` =

Code d'erreur ; `lib_rv__%` retourne la valeur -1 si vous avez oublié de charger le fichier `INLINE` accompagnant la procédure et la valeur 0 si tout se déroule correctement.

Description :

La procédure `L_HSCROL` permet de faire défiler un texte à l'écran, en utilisant une police de caractères convertie auparavant à l'aide de la procédure `L_SCONV`.

La procédure `M_HSCROL` permet de faire défiler un texte, avec une police de caractères convertie à l'aide de la procédure `M_SCONV`.

La procédure `H_HSCROL` permet de faire défiler un texte, avec une police de caractères convertie à l'aide de la procédure `H_SCONV`.

Attention : ▽

Le string à faire défiler ne doit contenir que des caractères contenus dans la police de caractères ; les autres caractères seront remplacés par des espaces vides et n'apparaîtront pas à l'écran. La procédure ne distingue pas entre minuscules et majuscules : que vous entriez un 'a' ou un 'A', le résultat sera le même, c'est le premier caractère de la police qui apparaître à l'écran. Voici donc la suite des caractères utilisables :

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!..:

Exemple : voir `SCOHSC.LST` sous `L_SCONV`, `M_SCONV` ou `H_SCONV`.

Graphisme : L_VCONV M_VCONV H_VCONV

Utilisation : Conversion d'une police de 40 caractères qui sera ensuite utilisée avec les procédures L_VSCROL, M_VSCROL ou H_VSCROL pour faire défiler un texte.

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_VCONV.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_VCONV.LST
(moyenne résolution)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_VCONV.LST
(haute résolution)

INLINE : Ne nécessite aucun fichier INLINE

Exemples : EXAMPLES\PROGRAMS.LOW\L_VSCROLL.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\H_VSCROLL.LST
(haute résolution)

Syntaxe : GOSUB
gfa_l_vconv__(screenadr__%,fontadr__%,y__%)

GOSUB
gfa_m_vconv__(screenadr__%,fontadr__%,y__%)

GOSUB
gfa_h_vconv__(screenadr__%,fontadr__%,y__%)

Paramètres :

- *screenadr__%* :

Adresse de l'écran ; ce paramètre donne l'adresse de la mémoire de l'écran sur lequel s'affiche la police de caractères à convertir ; en principe, on entre ici XBIOS(2) qui est l'adresse de l'écran actuellement visible.

Il est cependant possible d'entrer ici une autre adresse si la police à convertir se trouve dans un deuxième écran.

- **fontadr__%** :

Adresse d'un buffer qui recevra les données correspondant à la police de caractères convertie ; la quantité d'informations nécessaires pour représenter une police de caractères varie en fonction du degré de résolution de l'écran, ce qui fait aussi varier la taille qu'il convient de donner au buffer :

- **basse résolution** : 5120 octets
- **moyenne résolution** : 5120 octets
- **haute résolution** : 10240 octets

Il est possible de sauvegarder les polices de caractères après leur conversion, et le nombre des octets vous indique la taille des fichiers à prévoir.

Le buffer appelé à contenir la police de caractères après sa conversion ne saurait en aucun cas être plus petit que les valeurs indiqués ci-dessus, faute de quoi les données viendraient écraser d'autres données importantes.

- **y__%** :

Numéro de la ligne contenant la première série de caractères ; selon le degré de résolution, y__% doit être compris entre les minima et maxima suivants :

- **basse résolution** : 0 - 168
- **moyenne résolution** : 0 - 168
- **haute résolution** : 0 - 272

Si vous dépassez ces valeurs, la conversion ne se fait pas et lib_rv__% renvoie la valeur 1 ; si vous entrez une valeur correcte, lib_rv__% renvoie la valeur 0 et la conversion se déroule normalement.

Valeur retournée :

`lib_rv__%` =

Code d'erreur ; `lib_rv__%` retourne la valeur -1 si vous avez oublié de charger le fichier `INLINE` accompagnant la procédure et la valeur 0 si tout se déroule correctement.

Description :

La procédure `L_VCONV` sert à convertir une police de caractères de 16 couleurs pour la réutiliser avec `L_VSCROL`.

La procédure `M_VCONV` sert à convertir une police de caractères de 4 couleurs pour la réutiliser avec `M_VSCROL`.

La procédure `H_VCONV` sert à convertir une police de caractères de 2 couleurs pour la réutiliser avec `H_VSCROL`.

Les polices ainsi converties doivent être utilisée sous la procédure correspondante : il est impossible par exemple d'utiliser une police convertie à l'aide de `L_VCONV` avec la procédure `M_VSCROL`.

Lorsque l'on souhaite créer une nouvelle police de caractères, il convient de le faire dans la résolution adéquate. Attention : ces procédures ne convertissent que 40 caractères, qui doivent obligatoirement être alignés l'un contre l'autre sur deux lignes de 20 signes.

La taille des caractères varie en fonction de la résolution :

- **basse résolution :** 16 x 16 pixels
- **moyenne résolution :** 32 x 16 pixels
- **haute résolution :** 32 x 32 pixels

Les caractères doivent être alignés dans l'ordre suivant :

ABCDEFGHIJKLMNOPQRSTUVWXYZ

UVWXYZ0123456789!.:;

Vous trouverez un exemple de police de caractères dans le fichier DEMO.PI1, qui contient une police de ce genre conçue pour la basse résolution.

En haute résolution, il convient de créer de plus 40 masques pour les caractères correspondant. Il faut ranger ces masques exactement dans le même ordre que les caractères ; ils doivent commencer juste au-dessous du caractère correspondant. On commence donc par entrer les 40 caractères sur deux lignes, et juste au-dessous, on entre les 40 'masques de caractères', sur deux lignes supplémentaires.

Ces masques sont indispensables en haute résolution, car ils permettent de spécifier quelle partie du caractère recouvre l'arrière-plan et quelle partie le laisse au contraire transparaître. Les parties blanches d'un caractère peuvent servir à la fois de couleur et de 'lucarne'.

Ces masques sont créés comme ceux des procédures 'shape' en haute résolution.

Chaque masque a la même taille que le caractère correspondant, c'est-à-dire 32 x 32 pixels ; les parties du caractère appelées à masquer l'arrière-plan doivent être marquées comme des bits activés.

L'image contenue dans DEMO.PIC contient une police de caractères de ce type, accompagnée de ses masques, qui sert d'exemple dans le programme de démonstration H_VSCROL.

Exemple : EXAMPLES\PROGRAMS.LOW\L_VSCROL.LST

MERGE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_VCONV.LST

 GFA_GLIB.3_0\GFA_GLIB.LOW\L_VSCROL.LST

 GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST

 GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```

'                                     ! installer un buffer de 5120 octets
INLINE fontadr __%,5120
'                                     ! buffer pour L_VSCROL
INLINE bufadr __%,5536
'
GOSUB gfa_poppal__
palette$=lib_rv__$
GOSUB gfa_pload__("\EXAMPLES\STUFF\DEMO.PI1",XBIOS(2),0)
'
GOSUB gfa_l_vconv__(XBIOS(2),fontadr __%,136)
'
' BSAVE "VFONT.FNT",fontadr __%,5120 !sauvegarder la police de caractères
'
txt__$="appuyez sur une touche pour interrompre cette démonstration"
'
GOSUB gfa_l_vscrol__(XBIOS(2),fontadr __%,bufadr __%,152,11,173,txt__$,0)
'
~XBIOS(6,L :V :palette$)

```

Ce programme de démonstration sert à la fois pour L_VCONV et pour L_VSCROL. Nous commençons par installer un buffer de 5120 octets (à l'aide d'une instruction INLINE) qui recevra par la suite la police de caractères convertie. Nous installons un deuxième buffer par INLINE, qui servira sous L_VSCROL à enregistré l'arrière-plan recouvert par le texte, puisqu'il faudra le restaurer à la fin de L_VSCROL, de façon à ne laisser aucune trace du texte à l'écran. Pour calculer la taille du buffer, reportez-vous aux formules données sous L_VSCROL, M_VSCROL et H_SCRÖL.

Nous sauvegardons ensuite la palette des couleurs d'origine à l'aide de POPPAL et la procédure PLOAD nous sert à charger l'image de démonstration DEMO.PI1 dans la mémoire d'écran (adresse XBIOS(2)) ainsi que sa palette de couleurs.

L'image de démonstration contient une police de caractères, à partir de la ligne 136, police que nous allons convertir (appel de L_VCONV) pour la reprendre ensuite sous L_VSCROL. Comme cette police de caractères se trouve sur l'écran visible, nous indiquons l'adresse de ce dernier (XBIOS(2)) sous 'screenadr__%'.

y__% prend la valeur 136 puisque la police de caractères commence sur la 136ème ligne. Pour que la police convertie parvienne dans le buffer qui lui est réservé, il convient d'en indiquer l'adresse (fontadr__%).

Les caractères étant convertis, nous pouvons les sauvegarder à l'aide de l'instruction BSAVE afin de pouvoir éventuellement nous en resservir par la suite. Il est ainsi possible de se constituer une collection de polices de caractères déjà converties, que l'on pourra appeler directement dans un programme pour la faire tourner sous un des procédures VSCROL.

La sauvegarde n'a pas lieu dans notre exemple ci-dessus, puisque les lignes BSAVE sont introduites par une apostrophe caractérisant les commentaires.

Après la conversion, nous entrons un texte dans la variable-string txt__\$: la longueur de ce texte n'est limitée que par la longueur maximale admise pour un string.

Attention : ▽

Le string peut certes contenir n'importe quel code appartenant au tableau ASCII, mais seuls les 40 signes mentionnés ci-dessus pourront défiler. Tout autre caractère sera remplacé par un espace vide, à l'exception des minuscules qui seront assimilées à des majuscules.

Nous lançons L_VSCROL (voir ci-après) en indiquant l'adresse de l'écran visible (XBIOS(2)) sous 'screenadr__%' de façon à ce que le texte se mette bien à défiler sur l'écran actuellement visible (belle lapalissade!).

Nous avons affecté les valeurs 152 et 11 respectivement aux paramètres x__% et y__% : il s'agit des coordonnées du coin supérieur gauche du secteur d'écran dans lequel le texte se met à défiler : nous avons défini pour cela un secteur de 173 lignes (pixel) à partir de la ligne 11 (rows__%=173). Le texte se met donc à défiler dans un rectangle défini par les sommets suivants : 152,11 et 167,183.

On peut ressortir de L_VSCROL en appuyant sur une touche quelconque (options__% = 0) ; ceci étant fait, l'arrière-plan est restauré ainsi que la palette des couleurs d'origine par XBIOS(6) et le programme prend fin.

Graphisme : L_VSCROL M_VSCROL H_VSCROL

Utilisation : Ces procédures permettent de faire défiler verticalement (scrolling) un texte mémorisé dans un string, en utilisant une police de caractères convertie à l'aide de L_VCONV, M_VCONV ou H_VCONV.

Dossier : GFA_GLIB.3_0\GFA_GLIB.LOW\L_VSCROL.LST
(basse résolution)

GFA_GLIB.3_0\GFA_GLIB.MED\M_VSCROL.LST
(moyenne résolution)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_VSCROL.LST
(haute résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.LOW\L_VSCROL.INL
(L_VSCROL.LST)

GFA_GLIB.3_0\GFA_GLIB.MED\M_VSCROL.INL
(M_VSCROL.LST)

GFA_GLIB.3_0\GFA_GLIB.HIG\H_VSCROL.INL
(H_VSCROL.LST)

Exemples : EXAMPLES\PROGRAMS.LOW\L_VSCROL.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\H_VSCROL.LST
(haute résolution)

Syntaxe : GOSUB gfa_l_vscrol__(screenadr__%,fontadr__%,
bufadr__%,x__%,y__%,rows__%,txt__\$,options__%)

GOSUB gfa_m_vscrol__(screenadr__%,fontadr__%,
bufadr__%,x__%,y__%,rows__%,txt__\$,options__%)

Paramètres :

- **screenadr__%** :

Adresse de l'écran sur lequel le texte (txt__\$) devra défiler ; en principe, on entre ici XBIOS(2) qui est l'adresse de l'écran actuellement visible.

- **fontadr__%** :

Adresse d'une police de caractères convertie ; comme les procédures VSCROL utilisent obligatoirement une police de caractères convertie à l'aide d'une procédure VCONV, il faut ici entrer l'adresse d'une police de caractères de ce type, en respectant bien le degré de résolution.

- **bufadr__%** :

Adresse d'un buffer nécessaire pour ensuite restaurer le contenu de l'arrière-plan recouvert par le texte ; il convient de créer un buffer de taille suffisante, qui varie en fonction du nombre de lignes réservées à l'aide du paramètre rows__%, faute de quoi les données écraseraient d'autres données importantes.

Voici comment calculer la taille de ce buffer :

- **basse résolution :** rows__% * 16 * 2
- **moyenne résolution :** rows__% * 12 * 2
- **haute résolution :** rows__% * 6 * 2

- **x__%** : coordonnée X du coin supérieur gauche du secteur dans lequel le texte défilera ; valeurs autorisées :

- **basse résolution :** 0 - 304
- **moyenne résolution :** 0 - 608
- **haute résolution :** 0 - 608

Si vous dépassez ces limites, la procédure L_VSCROL, M_VSCROL ou H_VSCROL se plantera, et le texte ne se mettra donc pas à défiler.

- **y__%** :

Coordonnée Y du coin supérieur gauche du secteur dans lequel le texte défilera ; valeurs autorisées :

- > **basse résolution** : 0 - 199
- > **moyenne résolution** : 0 - 199
- > **haute résolution** : 0 - 399

Si vous entrez une valeur supérieure à ces maxima, la procédure se plante ; si vous affectez à `y__%` une valeur négative, c'est la valeur de `rows__%` qui décidera si le texte va ou non se mettre à défiler. Si `rows__%` est assez grand pour qu'une partie du secteur de défilement figure tout de même dans l'écran, le défilement du texte se fera dans ce secteur ; si `rows__%` est par contre assez petit, tout le secteur se trouve en dehors de l'écran, et la procédure se plante.

Exemple :

`y__%=-100` et `rows__%=200`

=> les cent lignes supérieures de l'écran pourront servir à faire défiler le texte

`y__%=-100` et `rows__%=50`

=> plantage.

- `rows__%` :

Nombre de lignes d'écran à l'intérieur desquelles le texte va défiler ; vous pouvez entrer n'importe quelle valeur, mais attention : si `rows__% <= 0`, vous ressortez immédiatement de la procédure ; si vous entrez plus de lignes que l'instruction n'en contient réellement, la procédure se charge de ramener le nombre de lignes au maximum possible.

Le nombre de lignes se compte toujours en tenant compte de la valeur entrée sous `y__%` ; si la ligne désignée par `y__%` se trouve en dehors de l'écran (valeur négative), cela revient à soustraire ce nombre de ligne de `rows__%`.

- `txt__$` :

Chaîne de caractères qui va défiler et qui peut avoir une longueur (quasiment) quelconque. Le string peut certes contenir n'importe quel code appartenant au tableau ASCII, mais seuls les 40 signes convertis à l'aide de `L_VCONV`, `M_VCONV` ou `H_VCONV` pourront défiler ; les minuscules

seront assimilées à des majuscules. Vous pouvez donc saisir votre string `txt_$` en minuscules ou majuscules.

Les caractères ne faisant pas partie des caractères convertibles seront remplacés par des espaces vides.

- options__% :

Ce paramètre sert à préciser la durée pendant laquelle le texte va défiler ; il doit obligatoirement prendre une des valeurs suivantes :

- > `options__% == 0` : Le texte défile sans arrêt à l'écran, grâce à une boucle sans fin ; pour en ressortir, il faut appuyer sur une touche quelconque ;
- > `options__% < 0` : Lorsque le paramètre '`options__%`' a une valeur négative, le texte ne défile qu'une fois à l'écran, et il est impossible d'interrompre ce défilement ;
- > `options__% > 0` : Lorsque le paramètre '`options__%`' a une valeur positive, le texte ne défile également qu'une fois à l'écran, mais il est possible d'interrompre ce défilement avant qu'il ne le fasse tout seul.

Valeur retournée :

`lib_rv__%` =

Code d'erreur ; `lib_rv__%` retourne la valeur -1 si vous avez oublié de charger le fichier `INLINE` accompagnant la procédure et la valeur 0 si tout se déroule correctement.

Description :

Les procédures `L_VSCROL`, `M_VSCROL` et `H_VSCROL` permettent de faire défiler verticalement un texte sur l'écran, de bas en haut.

Vous pouvez utiliser une police de caractères de votre création, se composant de 40 signes.

Il vous faudra convertir cette police à l'aide d'un des trois procédures VCONV et charger la police convertie en mémoire vive avant de lancer L_VSCROL, M_VSCROL ou H_VSCROL.

Attention : ▽

Le string à faire défiler ne doit contenir que des caractères contenus dans la police de caractères ; les autres caractères seront remplacés par des espaces vides et n'apparaîtront pas à l'écran. La procédure ne distingue pas entre minuscules et majuscules.

Voici donc la suite des caractères utilisables :

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789 !.:

Le scrolling ne se fait que si tous les paramètres ont des valeurs correctes ; le paramètre options__% vous permet de choisir entre trois possibilités d'interruption du programme.

Particularité des procédures L_VSCROL, M_VSCROL et H_VSCROL : le texte peut défiler en laissant transparaître un arrière-plan par les 'lucarnes' formées par les boucles des lettres. C'est pourquoi, en haute résolution (procédure H_VSCROL), chaque caractère doit être accompagné de son masque, qui est considéré comme une partie de la police de caractères.

Les images contenues dans DEMO.PI1 et DEMO.PIC vous montrent l'aspect que peuvent avoir de telles polices de caractères en basse et haute résolution (dossier EXAMPLES\STUFF).

Notez bien que les procédures, avant de prendre fin, prennent soin de restaurer intégralement l'arrière-plan d'origine, de façon à ce qu'il ne reste aucune trace du texte ayant défilé à l'écran.

Ne confondez pas L_VSCROL, M_VSCROL et H_VSCROL avec VSCROL : ce dernier sert à faire défiler des graphiques et non des textes !

Exemple : voir L_VSCROL.LST sous L_VCONV, M_VCONV ou H_VCONV

Graphisme : GRAF2D

Utilisation : Affichage de lettres en deux dimensions, à l'aide d'un ensemble de vecteurs, ce qui permet ensuite de déformer, refléter (miroir), agrandir, rapetisser, tourner ou remplir le caractère.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\GRAF2D.LST
(quelle que soit la résolution)

INLINE : Ne nécessite aucun fichier INLINE.

Exemples : EXAMPLES\PROGRAMS.LOW\GRAF2D.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\GRAF2D.LST
(haute résolution)

Syntaxe : GOSUB gfa_graf2d__(x__%,y__%,char__\$,options__%,
miroir__%,angle__%,xfacteur__,yfacteur__,
xpenche__%,flag__!)

Paramètres :

- x__% :

coordonnée X du coin gauche inférieur de la lettre.

- y__% :

Coordonnée Y du coin gauche inférieur de la lettre. x__% et y__% peuvent prendre n'importe quelle valeur.

- char__\$:

Caractère à afficher ; le jeu de caractères ne comprenant que les lettres majuscules dépourvues de tout accent, seules ces lettres sont autorisées. La variable lib_rv__% renvoie la valeur 1 lorsque vous entrez un caractère non autorisé.

Si vous entrez ici une chaîne de plusieurs caractères, seul le premier de ceux-ci sera retenu et traité.

- *options*__% :

Flag ; seuls les bits 0 à 3 de ce paramètre sont utiles, les autres peuvent prendre n'importe quelle valeur.

*options*__% permet de préciser quelle manipulation on entend faire subir à la lettre concernée ; on peut par exemple la faire tourner, auquel cas il conviendra de préciser l'angle de rotation.

Voici la signification des bits 0 à 3 :

➤ ***Bit 0 = échelle de restitution :***

Le caractère concerné va être grossi ou rapetissé à l'affichage, selon le contenu des facteurs *xfacteur*__ (axe des X) et *yfacteur*__ (axe des Y).

➤ ***Bit 1 = rotation :***

Le caractère concerné va s'afficher vu sous l'angle précisé à l'aide du paramètre *angle*__%.

➤ ***Bit 2 = inclinaison :***

Le caractère va s'afficher penché, un peu comme une italique, selon la valeur donnée au paramètre *xpenche*__%

➤ ***Bit 3 = miroir :***

Le caractère semble se refléter dans un miroir, le paramètre *miroir*__% permettant de spécifier si ce miroir est horizontal ou vertical.

Règle s'appliquant à ces trois bits : un bit mis signale que la manipulation en question est activée, alors qu'un bit non mis signale que le caractère ne sera pas affecté par la manipulation concernée.

Ainsi par exemple, lorsqu'aucun bit du paramètre *options* n'est mis, les autres paramètres complétant *options*__% sont ignorés. Il convient cependant de les entrer malgré tout, quitte à leur donner une valeur purement arbitraire.

Il est plus facile d'activer ou désactiver individuellement les quatre bits de ce paramètre options__% en les écrivant dans le binaire usuel sous le GfA Basic. Par exemple, pour indiquer que les bits 0 et 2 sont vrais, on écrira :

&X101.

- **miroir**__% :

Sert à déterminer l'axe du miroir, lorsque le bit 3 du paramètre options__% est sur 1, faute de quoi ce paramètre est ignoré.

Valeurs autorisées :

> **miroir**__% == 1 : axe vertical, le caractère est 'mis sur la tête'

> **miroir**__% == 0 : axe horizontal.

- **angle**__% :

Sert à déterminer l'angle de rotation (indiqué en degrés) du caractère lorsque le bit 1 du paramètre options__% est mis sur 1, faute de quoi ce paramètre est ignoré. La rotation s'exerce par rapport au coin inférieur gauche du caractère, dans le sens des aiguilles d'une montre.

- **xfacteur**__ :

Facteur d'inclinaison du caractère, dans l'axe X, lorsque le bit 0 du paramètre options__% est mis sur 1, faute de quoi ce paramètre est ignoré.

- **yfacteur**__ :

Facteur d'inclinaison du caractère, dans l'axe Y (voir xfacteur__).

Notez que xfacteur__ et yfacteur__ sont des variables à virgule flottante (floatingpoint variable), si bien que vous pouvez entrer ici non seulement des nombres entiers mais aussi des nombres décimaux (à virgule). Ceci vous permet d'agrandir ou rapetisser de façon très fine le caractère concerné.

- *xpenche*__% :

Facteur d'inclinaison du caractère, dans l'axe X, qui vous permet de pencher le caractère lorsque le bit 2 du paramètre options__% est sur 1, faute de quoi ce paramètre est ignoré.

Les valeurs positives font pencher le caractère vers la gauche et les valeurs négatives vers la droite.

- *flag*__! :

Flag du motif de remplissage ; ce paramètre permet d'indiquer si le caractère doit être rempli par un motif ou non ; il est possible de déterminer le motif de remplissage avant l'appel de la procédure, à l'aide de l'instruction DEFFILL du GfA Basic.

Valeurs autorisées :

- > *flag*__! == TRUE : le caractère sera rempli par un motif
- > *flag*__! == FALSE : le caractère ne sera pas rempli par un motif.

La routine de remplissage est prise en charge par l'instruction FILL du GfA Basic. Comme il s'agit d'une routine 'floodfill', il peut arriver que le caractère ne soit pas complètement rempli s'il est trop penché ou s'il chevauche un autre caractère.

Valeur retournée :

*lib_rv*__% =

Code d'erreur ; aucun caractère ne s'affiche, et *lib_rv*__% retourne la valeur 1 lorsque vous entrez sous *char*__\$ un caractère qui ne fait pas partie du jeu de caractères vectorisés ; lorsque vous entrez un caractère autorisé (toutes les majuscules non accentuées) *lib_rv*__% retourne la valeur 0 et le caractère vient s'afficher correctement à l'écran.

Description :

Cette procédure vous permet d'afficher en deux dimensions à l'écran une lettre de l'alphabet qui tienne compte d'un jeu de vecteurs intégré dans la procédure ; le caractère peut être agrandi, rapetissé, reflété dans un miroir,

Le jeu des caractères autorisés comprend toutes les majuscules non accentuées, aucun autre caractère n'étant pris en compte.

Comme nous l'avons déjà dit, cette procédure permet l'affichage en deux dimensions : notez qu'il est suivi d'une autre procédure permettant l'affichage en trois dimensions nommé GRAF3D (voir ci-après). Ce dernier procédé permet d'obtenir un effet de profondeur du caractère.

Attention : ▽

Il convient d'appeler la procédure GRAF2D une première fois avant toute utilisation pour initialiser le jeu des vecteurs. On entre alors aucun caractère tout en attribuant des valeurs quelconques aux paramètres.

Exemple : EXAMPLES\PROGRAMS.LOW\GRAF2D.LST

MERGE : GFA_GLIB.3_0\GFA_GLIB.ALL\GRAF2D.LST

```
GOSUB gfa_graf2d__(0,0,"",0,0,0,0,0,0,FALSE) ! initialisation de la procédure
,
DEFFILL 1,1,0
,
GOSUB gfa_graf2d__(25,100,"G",&X11,0,30,2,2,0,TRUE)
GOSUB gfa_graf2d__(81,132,"F",&X11,0,30,2,2,0,TRUE)
GOSUB gfa_graf2d__(120,155,"A",&X11,0,30,2,2,0,TRUE)
,
PRINT AT (1,23) ;"Appuyez sur une touche pour ressortir"
~INP (2)
```

Nous appelons une première fois la procédure GRAF2D pour initialiser le jeu des vecteurs qu'il contient, car ce jeu de vecteurs se compose de lignes de données à charger.

L'instruction DEFFILL du GfA Basic nous permet ensuite de créer un motif de remplissage des caractères à afficher.

Nous affichons les trois lettres majuscules "G", "F" et "A" ; le "G" s'affiche à la position 25,100 (x__%,y__%). Les coordonnées désignent le coin inférieur gauche de la lettre.

Pour sortir le "G", nous affectons le string "G" au paramètre `char__$` ; le string ne se compose que d'une seule lettre, puisqu'il convient d'appeler GRAF2D pour chaque caractère à afficher.

Les bits 0 et 1 du paramètre `options__%` sont vrais, et les bits 2 et 3 faux. Cela signifie que le caractère va être allongé d'un facteur 2 tant dans l'axe des X (`xfacteur__`) que dans l'axe des Y (`yfacteur__`) (ce qui revient à agrandir le caractère). Par contre, le caractère ne sera ni reflété dans un miroir (bit 2 faux dans `options__%`) ni penché (bit 3 sur 0 dans `options__%`) : les paramètres `miroir__%` et `xpenche__%` sont ignorés.

Pour les trois caractères, nous avons affecté la valeur TRUE au flag de remplissage `flag__!`, si bien que nos trois caractères vont être complétés par le motif créé à l'aide de l'instruction DEFFILL du GfA Basic.

Pour interrompre la démonstration, il suffit d'appuyer sur une touche quelconque après l'affichage des trois caractères.

Graphisme : GRAF3D

Utilisation : Affichage de lettres en trois dimensions, à l'aide d'un ensemble de vecteurs, ce qui permet ensuite de déformer, refléter (miroir), agrandir, rapetisser, tourner ou remplir le caractère.

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\GRAF3D.LST
(quelle que soit la résolution)

INLINE : Ne nécessite aucun fichier INLINE.

Exemples : EXAMPLES\PROGRAMS.LOW\GRAF3D.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\GRAF3D.LST
(haute résolution)

Syntaxe : GOSUB gfa_graf3d__(x__%,y__%,char__\$,options__%,
miroir__%,angle__%,xfacteur__,yfacteur__,zfacteur__,
xpenche__%)

Paramètre :

- x__% :

Coordonnée X du coin gauche inférieur de l'avant-plan de la lettre.

- y__% :

Coordonnée Y du coin gauche inférieur de l'avant-plan de la lettre. x__% et y__% peuvent prendre n'importe quelle valeur.

- char__\$:

Caractère à afficher ; seules sont autorisées les lettres majuscules dépourvues de tout accent, car le jeu de caractères ne comprend que ces lettres. La variable lib_rv__% renvoie la valeur 1 lorsque vous entrez un caractère non autorisé.

Si vous entrez ici une chaîne de plusieurs caractères, seul le premier de ceux-ci sera retenu et traité.

- options__% :

Flag ; seuls les bits 0 à 3 de ce paramètre sont utiles, les autres peuvent prendre n'importe quelle valeur.

options__% permet de préciser quelle manipulation on entend faire subir à la lettre concernée ; on peut par exemple la faire tourner, auquel cas il conviendra de préciser l'angle de rotation.

Voici la signification des bits 0 à 3 :

➤ **Bit 0 = échelle de grandeur :**

Le caractère concerné va être grossi ou rapetissé à l'affichage selon le contenu des facteurs xfacteur__ (axe des X) et yfacteur__ (axe des Y).

➤ **Bit 1 = rotation :**

Le caractère concerné va s'afficher vu sous l'angle précisé à l'aide du paramètre angle__%.

➤ **Bit 2 = inclinaison :**

Le caractère va s'afficher penché, un peu comme une italique, selon la valeur donnée au paramètre xpenche__%

➤ **Bit 3 = miroir :**

Le caractère semble se refléter dans un miroir, le paramètre miroir__% permettant de spécifier si ce miroir est horizontal ou vertical.

Règle s'appliquant à ces trois bits : un bit mis signale que la manipulation en question est activée alors qu'un bit non mis signale que le caractère ne sera pas affecté par la manipulation concernée.

Ainsi par exemple, lorsqu'aucun bit du paramètre options n'est mis, les autres paramètres complétant options__% sont ignorés. Il convient cependant de les entrer malgré tout, quitte à leur donner une valeur purement arbitraire.

Il est plus facile d'activer ou désactiver individuellement les quatre bits de ce paramètre options__% en les écrivant dans le binaire usuel sous le GfA Basic. Par exemple, pour indiquer que les bits 0 et 2 sont mis, on écrira :

&X101.

- *miroir*__% :

Sert à déterminer l'axe du miroir, lorsque le bit 3 du paramètre options__% est sur 1, faute de quoi ce paramètre est ignoré.

Valeurs autorisées :

- *miroir*__% == 1 : axe vertical, le caractère est 'mis sur la tête'
- *miroir*__% == 0 : axe horizontal.

- *angle*__% :

Sert à déterminer l'angle de rotation (indiqué en degrés) du caractère lorsque le bit 1 du paramètre options__% est mis sur 1, faute de quoi ce paramètre est ignoré. La rotation s'exerce par rapport au coin inférieur gauche du caractère, dans le sens des aiguilles d'une montre.

- *xfacteur*__ :

Facteur d'inclinaison du caractère, dans l'axe X, lorsque le bit 0 du paramètre options__% est mis sur 1, faute de quoi ce paramètre est ignoré.

- *yfacteur*__ :

Facteur d'inclinaison du caractère, dans l'axe Y (voir *xfacteur*__).

- *zfacteur*__ :

Facteur d'inclinaison dans l'axe Z (voir *xfacteur*__).

Notez que *xfacteur*__, *yfacteur*__ et *zfacteur*__ sont des variables à virgule flottante (floatingpoint variable), ce qui vous permet d'entrer ici non seulement des nombres entiers mais aussi des nombres décimaux (à virgule). Ceci vous permet d'agrandir ou rapetisser de façon très fine le caractère concerné.

- *xpenche*__% :

Facteur d'inclinaison du caractère, dans l'axe X, qui vous permet de pencher le caractère lorsque le bit 2 du paramètre options__% est sur 1, faute de quoi ce paramètre est ignoré.

Les valeurs positives font pencher le caractère vers la gauche et les valeurs négatives vers la droite.

Valeur retournée :

*lib_rv*__% =

Code d'erreur ; aucun caractère ne s'affiche, et *lib_rv*__% retourne la valeur 1 lorsque vous entrez sous *char*__\$ un caractère qui ne fait pas partie du jeu des caractères vectorisés ; lorsque vous entrez un caractère autorisé (toutes les majuscules non accentuées) *lib_rv*__% retourne la valeur 0 et le caractère vient s'afficher correctement à l'écran.

Description :

Cette procédure vous permet d'afficher en trois dimensions à l'écran une lettre de l'alphabet qui tient compte d'un jeu de vecteurs intégré dans la procédure ; le caractère peut être agrandi, rapetissé, reflété dans un miroir, penché ou soumis à une rotation.

Le jeu des caractères autorisés comprend toutes les majuscules non accentuées, aucun autre caractère n'étant pris en compte.

Comme nous l'avons déjà dit, cette procédure permet l'affichage en trois dimensions : notez qu'il est précédé d'un autre procédure permettant l'affichage en deux dimensions nommé GRAF2D (voir ci-devant).

Attention : ▽

Il convient d'appeler la procédure GRAF3D une première fois avant toute utilisation pour initialiser le jeu des vecteurs. On n'entre alors aucun caractère tout en attribuant des valeurs quelconques aux paramètres.

Exemple : EXAMPLES\PROGRAMS.LOW\GRAF3D.LST

MERGE: GFA_GLIB.3_0\GFA_GLIB.ALL\GRAF3D.LST

```
GOSUB gfa_graf3d__(0,0,"",0,0,0,0,0,0,FALSE) ! initialisation de la procédure
,
GOSUB gfa_graf3d__(50,85,"G",&X11,1,-20,2,2,2,-31)
GOSUB gfa_graf3d__(123,158,"F",&X11,1,-20,2,2,2,-31)
GOSUB gfa_graf3d__(178,138,"A",&X11,1,-23,2,2,2,-31)
,
PRINT AT (1,24) ;"Appuyez sur une touche pour ressortir"
~INP(2)
```

Nous appelons une première fois la procédure GRAF3D pour initialiser le jeu des vecteurs qu'il contient, car ce jeu de vecteurs se compose de lignes de données à charger.

Nous affichons les trois lettres majuscules "G", "F" et "A" ; le "G" s'affiche à la position 50,185 (x__%,y__%). Les coordonnées désignent le coin inférieur gauche de la lettre.

Pour sortir le "G", nous affectons le string "G" au paramètre char__\$; le string ne se compose que d'une seule lettre, puisqu'il convient d'appeler GRAF3D pour chaque caractère à afficher.

Les bits 0 et 1 du paramètre options__% sont mis, et les bits 2 et 3 non-mis. Cela signifie que le caractère va être allongé d'un facteur 2 tant dans l'axe des X (xfacteur__) que dans l'axe des Y (yfacteur__) et celui des Z (zfacteur__) (ce qui revient à agrandir le caractère). Par contre, le caractère ne sera ni reflété dans un miroir (bit 2 non-mis dans options__%) ni penché (bit 3 sur 0 dans options__%) : les paramètres miroir__% et xpenche__% sont ignorés.

Pour stopper la démonstration, il suffit d'appuyer sur une touche quelconque après l'affichage des trois caractères.

Graphisme : VSCROL

Utilisation : Défilement vertical d'un secteur de l'écran

Dossier : GFA_GLIB.3_0\GFA_GLIB.ALL\VSCROL.LST
(quelle que soit la résolution)

INLINE : GFA_GLIB.3_0\GFA_GLIB.ALL\VSCROL.INL

Exemples : EXAMPLES\PROGRAMS.LOW\VSCROL.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\VSCROL.LST
(haute résolution)

Syntaxe : GOSUB gfa_vscrol__(x__%,y__%,width__%,height__%,
value__%,sourceadr__%,rows__%ncnt__%)

Paramètres :

- x__% :

Coordonnée X du coin supérieur gauche du secteur d'écran dans lequel le graphisme va défiler ; cette valeur X doit être divisible par 16, et être comprise entre les limites suivantes :

- > **basse résolution :** 0 - 304
- > **moyenne résolution :** 0 - 624
- > **haute résolution :** 0 - 624

- y__% :

Coordonnée Y du coin supérieur gauche du secteur d'écran dans lequel le graphisme va défiler. Remarquez que les coordonnées X et Y sont indiquées en pixels ; ces dernières doivent être comprises entre les valeurs suivantes :

- > **basse résolution :** 0 - 199
- > **moyenne résolution :** 0 - 199
- > **haute résolution :** 0 - 399

- *width__%* :

Largeur du secteur d'écran dans lequel le graphisme va défiler, indiqués en mots longs (1 'long' = 4 octets) ; on calcule le nombre de mots longs nécessaires selon les formules suivantes :

- **basse résolution** : 16 points = 2 mots longs
- **moyenne résolution** : 16 points = 1 mot long
- **haute résolution** : 32 points = 1 mot long

width__% doit être compris entre les valeurs suivantes :

- **basse résolution** : 2 - 40 longs (16 - 320 points)
- **moyenne résolution** : 1 - 40 longs (16 - 640 points)
- **haute résolution** : 1 - 20 longs (16 - 640 points)

- *height__%* :

Hauteur, en lignes de pixels, du secteur d'écran ; il convient de ne pas dépasser les valeurs suivantes :

- **basse résolution** : 1 - 200
- **moyenne résolution** : 1 - 200
- **haute résolution** : 1 - 400

- *value__%* :

Direction du défilement ; valeurs autorisées :

- **value__% == 1** : défilement du bas vers le haut
- **value__% <> 1** : défilement du haut vers le bas.

- *sourceadr__%* :

Adresse des données à faire défiler ; c'est à partir de cette adresse que doivent se trouver les données correspondant au secteur à faire défiler, dans le format adapté au degré de résolution de l'écran. Les données doivent être enregistrées de façon séquentielle, c'est-à-dire l'une à la suite de l'autre.

On tiendra compte des éléments suivants pour se faire une idée de la quantité de données mises en oeuvre dans le défilement :

➤ **En basse résolution** : 16 points consécutifs nécessitent 8 octets de données image ; lorsque l'on souhaite par exemple faire défiler un secteur aussi large que l'écran, il faut compter 160 octets par ligne de 320 points.

➤ **En moyenne résolution** : 16 points consécutifs nécessitent 4 octets de données image ; lorsque l'on souhaite par exemple faire défiler un secteur aussi large que l'écran, il faut compter 160 octets par ligne de 640 points.

➤ **En haute résolution** : 16 points consécutifs nécessitent 2 octets de données image ; lorsque l'on souhaite par exemple faire défiler un secteur aussi large que l'écran, il faut compter 80 octets par ligne de 640 points.

- *rows*__% :

Nombre de lignes à faire défiler entièrement dans le secteur d'écran concerné. Attention : il ne s'agit pas ici de la quantité de lignes des données appelées à défiler mais du nombre de lignes des données appelées à défiler moins le nombre de lignes d'une hauteur d'écran. En effet, chaque ligne comptée est amenée à défiler entièrement d'un bord à l'autre du secteur d'écran ; si on entrerait ici tout simplement le nombre de lignes des données appelées à défiler, la dernière ligne viendrait bien elle aussi défiler d'un bord à l'autre de l'écran, mais elle serait suivie des diverses données se trouvant juste après elle dans la mémoire vive puisque le secteur d'écran est toujours complètement rempli.

- *cnt*__% :

Facteur de défilement ; on peut ici préciser de combien de lignes le défilement doit avancer à chaque pas ; en indiquant 1, le scrolling se fait ligne par ligne, mais on peut aussi entrer par exemple 2, ce qui fera avancer le défilement de deux lignes à chaque pas etc.

Ce facteur de défilement ne doit toujours être compris entre 1 et la valeur de *rows*__%.

Valeur retournée :

lib_rv__% =

Code d'erreur ; *lib_rv__%* retourne la valeur -1 si vous avez oublié de charger le fichier **INLINE** accompagnant cette procédure et la valeur 0 lorsque tout se déroule normalement.

Description :

la procédure **VSCROL** vous permet, indépendamment du degré de résolution, de réaliser un scrolling vertical ; vous pouvez choisir quasiment n'importe quel secteur d'écran pour ce défilement.

La procédure ne fonctionnera correctement que si vous lui fournissez des données préparées en tenant compte de la résolution et de la taille du secteur d'écran concerné.

Exemple : **EXAMPLES\PROGRAMS.LOW\VSCROL.LST**

MERGE : **GFA_GLIB.3_0\GFA_GLIB.ALL\L_VSCROL.LST**

GFA_GLIB.3_0\GFA_GLIB.ALL\PLOAD.LST

GFA_GLIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```
,                                ! installons un buffer de 32000 octets pour
,                                une image Degas (sans sa palette de couleurs)
INLINE imageadr__%,32000
,
GOSUB gfa_poppal__
palette$=lib_rv__$
GOSUB gfa_pload__("\EXAMPLES\STUFF\DEMO.PI1",imageadr__%,0)
REPEAT
    ALERT 2," | scrolling | commencer?",1," OUI ",button|
    CLS
,
    GOSUB gfa_vscrol__(0,0,40,101,1,imageadr__%,100,1)
,
    ALERT 1," | dans l'autre | direction?",1," OUI ",button|
,
    GOSUB gfa_vscrol__(0,0,40,101,1,imageadr__%+16000,100,1)
```

```
ALERT 2," | Recommencer? | ",1," OUI | NON ",button|  
UNTIL button|=2  
~XBIOS(6,L :V :palette$)
```

Nous commençons par installer, à l'aide de `INLINE`, un buffer de 32000 octets, qui servira à recevoir l'image Degas-Elite contenue dans `DEMO.PI1`, sans sa palette de couleurs. Nous utilisons `POPPAL` pour sauvegarder la palette des couleurs d'origine, et nous chargeons l'image Degas dans le buffer grâce à `PLOAD`, en même temps que nous activons sa palette de couleurs.

Après avoir reçu la confirmation demandée dans la boîte de dialogue, nous entrons dans une boucle `REPEAT... UNTIL` qui fait défiler les 100 (`rows__%`) premières lignes de notre image dans un secteur d'écran de 101 (`height__%`) lignes sur toute la largeur de l'écran (`width__% = 40` longs = 320 pixels). Le secteur d'écran en question se trouve tout en haut de l'écran (`x__%=0`, `y__%=0`). Le défilement se fait ligne par ligne (`cnt__%=1`), de bas en haut (`value__%=1`). Nous entrons comme adresse des données à faire défiler celle qui correspond au début du buffer créé par `INLINE` (`imageadr__%`) et contenant les informations relatives à l'image chargée.

Nous attendons ensuite une confirmation, demandée par la deuxième boîte de dialogue ; ceci étant fait, le défilement reprend du haut vers le bas (`value__%=0`).

On peut répéter cette démonstration à volonté en cliquant sur le bouton "OUI" de la troisième boîte de dialogue ; on sort de la démonstration en cliquant sur "NON". Nous prenons soin de restaurer la palette des couleurs d'origine avant de sortir du programme.

Son : SMPLAY

Utilisation : Digitalisation et restitution de morceaux de musique, de bruits ou de paroles.

Dossier : GFA_SLIB.3_0\SMPLAY.LST
(quelle que soit la résolution)

INLINE : GFA_SLIB.3_0\SMPLAY.INL

Exemples : EXAMPLES\PROGRAMS.ALL\SMPLAY.LST
(quelle que soit la résolution)

Syntaxe : GOSUB gfa_smplay__(sourceadr__%,memadr__%,
cnt__%,speed__%,options__%)

Paramètres :

- *sourceadr__%* :

Dans la phase de digitalisation (*options__%* = 0), adresse à laquelle commence les données ; dans la phase de restitution sonore (*options__%* = 1), adresse de début des données à restituer.

- *memadr__%* :

Dans la phase de digitalisation, cette adresse désigne la fin du secteur de la mémoire vive contenant les données ; dans la phase de restitution, cette adresse indique la fin des données à restituer. Dans les deux phases, digitalisation et restitution, cette adresse est celle du dernier octet des données.

Lorsqu'il s'agit par exemple de restituer les 10000 octets se trouvant à l'adresse X, on affecte la valeur X à *sourceadr__%* et la valeur X + 10000 - 1 (attention, on compte à partir de zéro!) au paramètre *memadr__%*.

- *cnt__%* :

Cette variable n'a de signification que dans la phase de restitution sonore, mais il convient tout de même de lui attribuer une valeur quelconque dans

la phase de digitalisation ; `cnt__%` indique le nombre de répétitions du fragment sonore. Lorsque `cnt__% = 0`, le fragment est joué une fois sans répétition ; lorsque `cnt__% = 1`, le fragment sonore est joué deux fois, car il se répète une fois.

- *speed__%* :

Ce paramètre ne prend de signification qu'en phase de restitution, mais il convient (comme pour `cnt__%`) de lui affecter une valeur quelconque en phase de digitalisation. On peut lui donner une valeur comprise entre 0 et 65565, mais en étant conscient du fait que les valeurs supérieures à 100 entraînent des délais de restitution extrêmement longs :

avec `speed__% = 65565`, la restitution peut durer plusieurs heures!

- *options__%* :

Flag servant à indiquer si l'on est en phase de digitalisation ou de restitution ; valeurs autorisées :

- > `options__% == 0` : digitalisation
- > `options__% == 1` : restitution sonore.

Valeur retournée :

lib_rv__% =

Code d'erreur ; `lib_rv__%` retourne la valeur -1 lorsque vous avez oublié de charger le fichier `INLINE` accompagnant la procédure et la valeur 0 lorsque tout se déroule correctement.

Description :

Cette procédure `SMPLAY` vous permet de digitaliser quasiment n'importe quel morceau de musique (gare aux droits d'auteur!), n'importe quel bruitage ou fragment parlé que l'on peut acheter dans le commerce et tournant en 8 bits. Vous pouvez ainsi les réinsérer dans vos propres programmes écrits en GfA Basic, et varier la vitesse de restitution. Vous pouvez également reprendre des fragments provenant d'autres appareils de digitalisation, car `SMPLAY` traite les données en 8 bits.

Pour restituer le fragment sonore exactement à sa vitesse originale, il faut affecter la valeur 22 au paramètre speed__%.

Exemple : EXAMPLES\PROGRAMS.ALL\SMPLAY.LST

MERGE : GFA_SLIB.3_0\SMPLAY.LST

```
DIM buffer__%(3749)           ! buffer contenant les données à restituer
,
BLOAD "\EXAMPLES\STUFF\DEMO.SMP",V :buffer__%(0)
,
REPEAT
,
GOSUB gfa_smplay__(V :buffer__%(0),V :buffer__%(0)+14999,0,30,1)
,
ALERT 2,"| Répéter? |",1," OUI | NON ",button|
UNTIL button|=2
```

Nous commençons par installer un buffer de taille suffisante en dimensionnant un tableau numérique (integerarray) de 3750 éléments, chacun ayant une longueur de 4 octets.

On charge ensuite le fichier contenant l'exemple sonore (bruits d'épées!) dans le buffer (integerarray) commençant à l'adresse V:buffer__%(0).

Une boucle REPEAT... UNTIL permet de demander la répétition à volonté de ce programme de démonstration ; comme nous avons affecté la valeur 0 à cnt__%, le fragment n'est joué qu'une fois (pas de répétition). La vitesse de restitution est fixée à 30. L'adresse de fin est celle du dernier octet du fragment sonore : comme celui-ci occupe 15000 octets, et que le premier octet se trouve à l'adresse V:buffer__%(0), nous calculons de la façon suivante l'adresse de fin :

```
memadr__%=V :buffer__%(0)+15000-1
```

Il suffit de cliquer sur le bouton 'NON' de la boîte de dialogue pour interrompre la démonstration.

Son : SPEECH

Utilisation : Restitution de fragments parlés sur l'Atari ST ; traitée à l'aide des procédures SPEECH et SMPLAY, une chaîne de caractères est restituée sous forme acoustique.

Dossier : GFA_SLIB.3_0\SPEECH.LST
(quelle que soit la résolution)

INLINE : Ne nécessite aucun fichier INLINE.

Exemples : EXAMPLES\PROGRAMS.ALL\SPEECH.LST
(quelle que soit la résolution)

Syntaxe : GOSUB gfa_speech__(memadr__%,txt__%,speed__%)

Paramètres :

- *memadr__%* :

Adresse des données phonétiques, qui se trouvent sur la disquette dans le dossier EXAMPLES\STUFF dans le fichier PHONEMS.SMP.

Pour pouvoir restituer un fragment parlé à l'aide de la procédure SMPLAY, il convient de charger d'abord en mémoire le fichier PHONEMS.SMP, qui contient sous forme digitalisée, tous les phonèmes importants. Le paramètre *memadr__%* contient l'adresse des données phonétiques en mémoire vive.

- *txt__%* :

Chaîne de caractères à restituer acoustiquement ; cette chaîne de caractères peut contenir toutes les lettres de l'alphabet, y compris les lettres accentuées. La procédure ne distingue pas entre majuscules et minuscules.

- *speed__%* :

Vitesse de la restitution parlée ; en la faisant varier, on peut restituer le même fragment parlé depuis le registre des aigus rapides jusqu'à celui des basses profondes et lentes.

Valeur retournée :

Aucune.

Description :

Cette procédure SPEECH vous permet de restituer acoustiquement des textes enregistrés sous forme de strings.

Attention : ▽

Ceci ne peut fonctionner que si vous avez auparavant chargé en mémoire vive la procédure SMPLAY et le fichier PHONEMS.SMP contenant les phonèmes de la langue utilisée.

Ce fichier PHONEMS.SMP se trouve dans le dossier EXAMPLES\STUFF et occupe 35000 octets.

Exemple : EXAMPLES\PROGRAMS.ALL\SPEECH.LST

MERGE: GFA_SLIB.3_0\SPEECH.LST
GFA_SLIB.3_0\SMPLAY.LST

```
DIM buffer__%(8749)           ! buffer contenant les phonèmes (35000 octets)
,
BLOAD "\EXAMPLES\STUFF\DEMO.SMP",V :buffer__%(0)
,
REPEAT
  CLS
  PRINT AT(1,1) ;"veuillez entrer un texte"
  INPUT txt__$
  ,
  GOSUB gfa_speech__(V :buffer__%(0),txt__$,40)
  ,
  ALERT 2,"| Répéter? |",1," OUI | NON ",button|
UNTIL button|=2
```

Nous commençons par installer un buffer de taille suffisante (35000 octets) pour contenir les phonèmes, en dimensionnant un tableau numérique (integerarray) de 8750 éléments, chacun ayant une longueur de 4 octets.

On charge ensuite le fichier contenant les phonèmes dans le buffer (integerarray).

Une boucle REPEAT... UNTIL permet de demander la répétition à volonté de ce programme de démonstration ; il suffit pour cela de commencer par entrer un texte quelconque et de l'attribuer à la variable-string txt_\$. Cette dernière sera transmise directement à SPEECH et restituée acoustiquement.

La vitesse de restitution est fixée à 40, ce qui est un peu plus rapide que la vitesse normale d'élocution.

Il suffit de cliquer sur le bouton 'NON' de la boîte de dialogue pour interrompre la démonstration.

Son : REPSND

Utilisation : Pilotage de morceaux de musique ou d'effets sonores restitués sous interruption à l'aide de la routine XBIOS 32 (dosound) ; la procédure REPSND vous permet de répéter automatiquement ces fragments sonores ou de les interrompre à un moment donné.

Dossier : GFA_SLIB.3_0\REPSND.LST
(quelle que soit la résolution)

INLINE : GFA_SLIB.3_0\REPSND.INL

Exemples : EXAMPLES\PROGRAMS.ALL\REPSND.LST
(quelle que soit la résolution)

Syntaxe : GOSUB
gfa_repsnd__(options__%,sourceadr__%,length__%)

Paramètres :

- *options__%* :

Code de commande précisant le traitement appliqué au fragment sonore engendré par DOSOUND ; valeurs admises :

- options__% == 0 : Interruption immédiate du son
- options__% == 1 : Interruption du son une fois le fragment terminé
- options__% == 2 : Lancer la restitution du fragment sonore avec répétition automatique.

- *sourceadr__%* :

Adresse de début des données sonores ; il s'agit de la même adresse que celle qui est entrée lors de l'appel de DOSOUND.

- *length__%* :

Longueur des données sonores, mesurée en octets.

Valeur retournée :

`lib_rv__%` =

Code d'erreur ; `lib_rv__%` retourne la valeur -1 lorsque vous avez oublié de charger le fichier `INLINE` accompagnant la procédure et la valeur 0 lorsque tout se déroule correctement.

Description :

La fonction `DOSOUND` (`XBIOS32`) du système d'exploitation permet de restituer une fois un morceau de musique ou un fragment sonore sous interruption. Après une restitution, ce fragment prend fin, et il faut éventuellement le relancer si l'on souhaite qu'il se répète.

La procédure `REPSND` vous permet de répéter sans interruption le fragment sonore, sans empêcher le déroulement du programme normal.

Les données sonores correspondent aux données destinées à la routine `DOSOUND`.

Cette procédure `REPSND` vous permet aussi d'interrompre à tout moment la restitution du fragment sonore, de deux façons différentes : soit en l'interrompant immédiatement (`options__% = 0`) soit en ne l'interrompant qu'après son entière exécution (`options__% = 1`).

Notez que la procédure `REPSND` remplace complètement `DOSOUND`, qu'il n'est donc plus nécessaire d'appeler, puisque la procédure s'en charge.

Exemple : `EXAMPLES\PROGRAMS.ALL\REPSND.LST`

MERGE : `GFA_SLIB.3_0\REPSND.LST`

```
RESTORE sons
```

```
donnees__$=""
```

```
DO
```

! charger les données sonores pour

```
DOSOUND
```

```
  READ byte&
```

```
  EXIT IF byte&=-1
```

```
  donnees__$=donnees__$+CHR__(byte__$)
```

```
LOOP
```

```

' qui doivent être en format DOSOUND
SPOKE &H484,PEEK(&H484) AND NOT 1      ! désactiver le clic du clavier
'
GOSUB gfa_repsnd__(2,adr__,60)          ! lancer DOSOUND
'
PRINT "le fragment sonore sera automatiquement répété"
PRINT "après son entière exécution."
PRINT "Mais vous pouvez l'interrompre à tout moment"
PRINT "en appuyant sur une touche quelconque."
'
~INP(2)                                  ! guetter l'appui sur une touche
'
GOSUB gfa_repsnd__(0,adr__,60)          ! interruption immédiate de DOSOUND
'
PRINT "Appui sur une touche => poursuite de la démonstration"
~INP(2)                                  ! guetter l'appui sur une touche
CLS
'
GOSUB gfa_repsnd__(2,adr__,60)          ! relancer DOSOUND
'
PRINT " Un appui sur une touche provoquera l'arrêt
PRINT " de la restitution sonore après l'entière
PRINT " exécution du fragment"
'
~INP(2)                                  ! guetter l'appui sur une touche
'
GOSUB gfa_repsnd__(1,adr__,60)          ! interruption de DOSOUND à la fin
' du fragment
'
SPOKE &H484,PEEK(&H484) OR 1            ! réactiver le clic du clavier
'
sons :                                   !données sonores pour DOSOUND
DATA &H07,&HFF
DATA &H08,&H0F
DATA &H09,&H0F
DATA &H0A,&H0F
DATA &H00,&HEE,&H01,&H00
DATA &H02,&HBD,&H03,&H00
DATA &H04,&H9F,&H05,&H00
DATA &H07,&HF8
DATA &HFF,&H32

```

```
DATA &H00,&H1C,&H01,&H01
DATA &H02,&HE1,&H03,&H00
DATA &H04,&HBD,&H05,&H00
DATA &HFF,&H32
DATA &H00,&H65,&H01,&H01
DATA &H02,&H1C,&H03,&H01
DATA &H04,&HEE,&H05,&H00
DATA &HFF,&H50
DATA &H08,&H00
DATA &H09,&H00
DATA &H0A,&H00
DATA &HFF,&H00
DATA -1
```

Nous commençons par enregistrer dans un string (donnees__\$) les données DOSOUND se trouvant dans les lignes de DATA ; c'est là une des nombreuses possibilités de charger un certain nombre d'octets dans la mémoire vive.

Ceci étant fait, nous calculons l'adresse de début de ce string à l'aide de V : donnees__\$, ce qui nous donne en même temps l'adresse de début des données sonores à transmettre au procédure REPSND.

Pour plus de sécurité, nous désactivons le clic du clavier car il viendrait éventuellement perturber notre morceau de musique. Nous lançons immédiatement REPSND avec le code 2 qui permet de répéter sans fin le fragment sonore. Cette répétition peut être interrompue en appuyant sur une touche quelconque, puisque cela relance REPSND, avec cette fois le code 0 correspondant à 'interruption immédiate de la restitution sonore'.

Il suffit de ré-appuyer sur une touche pour relancer l'exécution du fragment sonore ce qui nous permet de faire une démonstration du deuxième mode d'interruption : un appui sur une touche quelconque appelle REPSND avec le code 1, signifiant que le fragment sonore est d'abord exécuté entièrement avant de s'interrompre, alors que le programme lui-même a déjà pris fin (message "fin de programme").

Son : MIDIKY

Utilisation : Restitution d'un fragment sonore à l'aide d'un clavier ayant un port MIDI ; selon sa hauteur, le fragment est joué plus ou moins vite.

Dossier : GFA_SLIB.3_0\MIDIKY.LST
(quelle que soit la résolution)

INLINE : Ne nécessite aucun fichier INLINE.

Exemples : EXAMPLES\PROGRAMS.ALL\MIDIKY.LST
(quelle que soit la résolution)

Syntaxe : GOSUB gfa_midiky__(sourceadr__%,length__%,
 value__%,char__%)

Paramètres :

- sourceadr__% :

Adresse de début des données sonores à exécuter ; exactement comme avec la procédure SMPLAY, il doit ici s'agir d'un fragment au format usuel sur 8 bits.

- length__% :

Longueur des données sonores, mesurée en octets.

Avec la procédure MIDIKY, il est conseillé de n'utiliser que des fragment assez courts, car chaque fragment sera joué d'un bout à l'autre avant que le programme n'accepte un signal provenant de l'appui sur une touche ; c'est pourquoi nous vous recommandons de faire exécuter plutôt des mélodies assez courtes, car la procédure peut créer plus de sons en un délai plus court.

- value__% :

Canal de sortie MIDI ; ce paramètre vous permet de choisir le canal MIDI par lequel le clavier MIDI que vous utilisez va envoyer ses données vers

l'ordinateur ; en règle générale, c'est le canal 1 qui est sélectionné par défaut.

Si vous entrez un numéro de canal ne permettant pas de recevoir des données, le fragment sonore ne pourra pas être restitué ; il convient alors de sortir de la procédure par la touche d'interruption (voir ci-après) et de le relancer éventuellement en entrant cette fois un numéro de canal qui soit correct.

- *char*__% :

Code-touche de la touche d'interruption ; vous entrez ici le code ASCII de la touche qui doit servir à sortir de la procédure MIDIKY

Attention : ▽

Evitez dans toute la mesure du possible d'entrer ici le code d'une touche non accessible depuis le clavier.

Valeur retournée :

Aucune.

Description :

Cette procédure permet d'exécuter des fragments musicaux, créés par exemple à l'aide de la procédure SMPLAY, sur un instrument à clavier, muni d'un port MIDI. Chaque touche de ce clavier se voit attribuer une vitesse de restitution : un son aigu entraîne une exécution rapide, un son grave une restitution lente. Cela donne l'impression d'une restitution à différentes hauteurs du fragment sonore.

Attention : ▽

La procédure MIDIKY appelle la procédure SMPLAY, qui doit donc être intégré dans votre programme.

Exemple : EXAMPLES\PROGRAMS.ALL\MIDIKY.LST

MERGE : GFA_SLIB.3_0\MIDIKY.LST

GFA SLIB.3 0\SMPLAY.LST

```

DIM buffer_$(9999)                                ! buffer pour les données sonores
                                                    (40000 octets)
,
midicanal_#=1
,
BLOAD "\EXAMPLES\STUFF\DEMO.SMP",V :buffer_$(0)
,
PRINT AT(1,1) ; "veuillez appuyer sur la touche qui
PRINT " servira à interrompre le programme"
,
key|=INP(2)
,
PRINT "Merci ! vous pouvez commencer la démonstration"
PRINT
PRINT "pour interrompre la démonstration, appuyez sur '" ;CHR_$(key|) ;'"
,
GOSUB gfa_midiky__(V :buffer_$(0)+5000,5000,midicanal_#,key|)

```

Nous commençons par installer un buffer (buffer_\$(0)) dimensionné de telle sorte qu'il puisse contenir 40000 octets (soit 10000 éléments de 4 octets chacun). Nous chargeons dans ce buffer un fragment sonore de 40000 octets.

Le programme vous demande ensuite d'appuyer sur la touche qui servira par la suite à ressortir de la démonstration de MIDIKY.

Pour finir, nous lançons la procédure MIDIKY.

Vous constatez que nous avons affecté la valeur V:buffer_\$(0)+5000 à sourceadr_# et la valeur 5000 à length_#. Cela signifie que la procédure MIDIKY ne va pas exécuter d'un trait tout le fragment sonore (qui fait 40000 octets) mais tout d'abord un extrait de 5000 octets, qui commence à 5000 octets du début.

Nous avons sélectionné le canal 1 comme canal MIDI (value_# = midicanal_#), mais vous pouvez sélectionner un autre canal en modifiant la valeur attribuée à midicanal_#.

Le dernier paramètre transmis est le code-touche de la touche d'interruption.

Après avoir chargé la procédure, vous pouvez exécuter l'exemple sonore, à l'aide de MIDIKY, sur une claviers MIDI ou un autre instrument doté d'un port MIDI.

Divers : **HORLOGE**

Utilisation : Affichage de l'heure par interruption, à n'importe quelle position-curseur.

Dossier : GFA_ALIB.3_0\HORLOGE.LST
(quelle que soit la résolution)

INLINE : GFA_ALIB.3_0\HORLOGE.INL

Exemples : EXAMPLES\PROGRAMS.ALL\HORLOGE.LST
(quelle que soit la résolution)

Syntaxe : GOSUB gfa_horloge__(options__%,x__%,y__%,
 hours__%,minutes__%,seconds__%,col__%)

Paramètres :

- options__% :

Code de commande, indiquant la fonction à lancer dans la procédure HORLOGE ; valeurs autorisées :

- > **options__% == 0** : lancer l'affichage de l'horloge
- > **options__% == 1** : régler l'heure affichée
- > **options__% == 2** : interrompre l'affichage de l'horloge.

Toute autre valeur sera ignorée ; veuillez également à régler d'abord l'heure affichée à l'aide de options__%=1, avant de passer à l'affichage de l'horloge.

- x__% :

Coordonnée X à laquelle l'horloge vient s'afficher ; les valeurs admises varient en fonction du degré de résolution de l'écran :

- > **basse résolution :** 0 - 32
- > **moyenne résolution :** 0 - 72
- > **haute résolution :** 0 - 72

Si vous entrez une valeur trop grande ou trop petite, lib_rv__% retournera la valeur 1 et vous sortirez de la procédure HORLOGE.

- *y*__% :

Coordonnée Y à laquelle l'horloge vient s'afficher ; valeurs admises, dans les trois degrés de résolution : 0 - 24.

- *hours*__% :

Ce paramètre sert à entrer l'heure lorsque vous passez par options__%=1 ; vous ne devez entrer que des valeurs comprises entre 0 et 23 ; lib_rv__% retournera la valeur 1 et vous sortirez de la procédure si vous entrez une valeur non autorisée.

- *minutes*__% :

Voir ci-dessus ; vous entrez ici le chiffre des minutes (de 0 à 59) lorsque vous passez par options__%=1 ; lib_rv__% retournera la valeur 1 si vous entrez une valeur non autorisée.

- *seconds*__% :

Voir ci-dessus ; vous entrez ici les secondes lorsque vous réglez votre horloge ; lib_rv__% renverra la valeur 1 si vous entrez une valeur inférieure à 0 ou supérieure à 59.

- *col*__% :

Vous pouvez entrer ici une valeur * désignant une couleur, ce qui déterminera la couleur des chiffres servant à communiquer l'heure ; valeurs admises :

- > basse résolution : 0 - 15
- > moyenne résolution : 0 - 3
- > haute résolution : 0 - 1

Valeur retournée :

lib_rv__% = code d'erreur ; lib_rv__% retourne la valeur 0 lorsque vous avez entré des paramètres corrects (c'est-à-dire respectant les limites indiquées ci-dessus) et l'exécution de la procédure se déroule normalement

`lib_rv__%` retourne par contre la valeur 1 lorsque vous avez entré une valeur non autorisée, ce qui vous fait immédiatement sortir de la procédure HORLOGE.

lib_rv__% :

Retourne la valeur -1 lorsque vous avez oublié de charger le fichier `INLINE` accompagnant la procédure.

Description :

La procédure HORLOGE est la solution à tous vos problèmes d'affichage de l'heure sous interruption ; l'affichage peut se faire dans les trois degrés de résolution et à n'importe quelle position du curseur.

Cela ne vous empêche pas de poursuivre vos autres travaux en GfA Basic, ce qui revient quasiment à faire tourner deux programmes simultanément.

L'affichage de l'heure s'avère important dans de nombreuses applications.

Exemple : `EXAMPLES\PROGRAMS.ALL\HORLOGE.LST`

MERGE : `GFA_ALIB.3_0\HORLOGE.LST`

```
PRINT AT(1,1) ; " PROGRAMME POUR TESTER L'HORLOGE"
,
GOSUB gfa_horloge__(1,0,0,23,59,50,0)      ! régler l'heure
,
PRINT AT(1,3) ; "heure : 23 :59 :50"
,
GOSUB gfa_horloge__(0,2,14,0,0,0,1)        ! lancer l'affichage de l'horloge
,
PRINT AT(1,4) ; "horloge mise en marche"
ALERT 2, " | stopper l'horloge? | ",1, " OUI ",button|
,
GOSUB gfa_horloge__(2,0,0,0,0,0,0)        ! arrêter l'horloge
,
PRINT "horloge arrêtée"
```

Le programme commence par une ligne de texte, suivie de la possibilité de régler l'heure valable au moment du lancement de la procédure HORLOGE. Nous attribuons la valeur 1 au paramètre options %, et les

valeurs 23, 59 et 50 respectivement à heures__%, minutes__% et seconds__%.

Une fois l'heure ainsi réglée, nous appelons la procédure une deuxième fois, pour lancer l'affichage de l'heure ; il convient ici de préciser les coordonnées et la couleur d'affichage ; options__% reçoit la valeur 0.

A partir de ce moment, l'horloge tourne sous interruption, en changeant toutes les secondes, et le programme GfA Basic continue lui aussi à tourner. Lorsque l'utilisateur clique sur le bouton OUI de la deuxième boîte de dialogue, cela relance la procédure HORLOGE en attribuant la valeur 2 au paramètre options__%, ce qui désactive l'horloge.

Le programme de démonstration prend fin lorsque l'on confirme l'arrêt de l'horloge. *

Attention : ▽

Lorsque vous intégrez la procédure HORLOGE dans un programme en GfA Basic, pensez à stopper l'affichage de l'horloge avant de ressortir du programme général, faute de quoi cet affichage continuera au niveau du bureau GEM ; cela vous mènera tout droit à un plantage général dès que vous lancerez une autre application. Rappelons que cette règle vaut pour tous les procédures tournant sous interruption.

Une astuce :

lorsque vous êtes en train d'écrire et de concevoir un programme, nous vous conseillons de désactiver l'appel de la procédure tournant sous interruption en l'introduisant par un signe de commentaire et de ne le rendre actif qu'après avoir mis au point définitivement votre programme. En cas d'erreur, il vous serait en effet impossible de stopper la procédure HORLOGE, à moins que vous n'ayez pensé à introduire une procédure ON ERROR GOSUB permettant un branchement vers une autre procédure (servant à désactiver l'horloge) lorsque survient une erreur.

Divers : BLACK

Utilisation : Eteindre l'écran du moniteur

Dossier : GFA_ALIB.3_0\BLACK.LST
(quelle que soit la résolution)

INLINE : Ne nécessite aucun fichier INLINE.

Exemples : EXAMPLES\PROGRAMS.ALL\BLACK.LST
(quelle que soit la résolution)

Syntaxe : GOSUB gfa_black__(options__%)

Paramètre :

- options__% :

Flag servant à préciser si l'écran doit être allumé ou éteint ; valeurs autorisées :

> **options__% <> 0 :** écran allumé

> **options__% == 0 :** écran éteint.

Valeur retournée :

Aucune.

Description :

On souhaite le plus souvent dissimuler le processus de chargement d'une image qui va s'afficher à l'écran ; cette procédure vous permet de désactiver l'écran, qui devient tout noir, sans que cela ne modifie la situation de la mémoire d'écran ou le contenu de cet écran.

Exemple : EXAMPLES\PROGRAMS.ALL\BLACK.LST

MERGE : GFA_ALIB.3_0\BLACK.LST

```
ALERT 1 " |éteindre l'écran?",1," OUI ",button|
,
GOSUB gfa_black__()    ! pour éteindre l'écran
,
~INP(2)              ! guetter l'appui sur une touche
,
GOSUB gfa_black__(1)  ! rallumer l'écran
```

L'exemple est suffisamment parlant pour ne nécessiter aucune explication.

Divers : PATMOV

Utilisation : Décaler un secteur de la mémoire vive ayant des marges équidistantes.

Dossier : GFA_ALIB.3_0\PATMOV.LST
(quelle que soit la résolution)

INLINE : GFA_ALIB.3_0\PATMOV.INL

Exemples : EXAMPLES\PROGRAMS.LOW\PATMOV.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\PATMOV.LST
(haute résolution)

Syntaxe : GOSUB gfa_black__(sourceadr__%,destadr__%,
cnt__%,b1__%,b2__%)

Paramètres :

- *sourceadr__%* :

Adresse de début du secteur de mémoire à déplacer

- *destadr__%* :

Adresse cible pour le secteur déplacé

- *cnt__%* :

Nombre de groupes d'octets (voir ci-après) à déplacer ; ce paramètre n'accepte qu'une valeur comprise entre 0 et 65535 (incluses) faute de quoi lib_rv__% retournera la valeur 1.

- *b1__%* :

Nombre d'octets formant un groupe ; ce paramètre n'admet lui-aussi qu'une valeur comprise entre 0 et 65535 ; la somme de b1__% avec cnt__% ne doit pas dépasser 65535

- b2__% :

Nombre d'octets servant d'espace (de 'marge') entre deux groupes d'octets ;
là aussi, valeur comprise entre 0 et 65535.

Valeur retournée :

lib_rv__% = code d'erreur ; lib_rv__% retourne la valeur 1 et le décalage ne se fait pas en mémoire lorsque vous entrez une valeur non autorisée pour un ou plusieurs paramètres (cnt__%, b1__% ou b2__%) ; lib_rv__% retourne par contre la valeur 0 lorsque tout se déroule normalement.

lib_rv__% retourne la valeur -1 lorsque vous avez oublié de charger le fichier INLINE accompagnant la procédure.

Description :

Cette procédure PATMOV ressemble fortement à l'instruction BMOVE du GfA Basic ; la différence réside cependant dans le fait que BMOVE décale dans la mémoire vive un bloc se composant d'octets s'enchaînant les uns à la suite des autres, alors que PATMOV permet de décaler un secteur de mémoire se composant de blocs séparés par des vides (marges) à intervalle régulier, lesquels vides ne sont pas décalés. A l'aide de BMOVE, il serait par exemple impossible de faire défiler vers le haut un secteur rectangulaire positionné en plein milieu de l'écran.

PATMOV permet par contre de déterminer combien d'octets il faut décaler par ligne (groupe d'octets), l'endroit où se trouve le premier groupe d'octets et combien d'octets vides (formant la marge) s'intercalent entre deux groupes consécutifs.

Admettons par exemple que nous souhaitions décaler 40 octets sur une ligne qui en compte 160 en basse résolution, et ce, sur une hauteur de 10 lignes, soit un rectangle de 10 lignes de haut pour 40 octets de large (= 80 pixels) : il nous suffira d'entrer l'adresse du premier groupe de 40 octets ainsi que l'adresse-cible de ce groupe. Le nombre de groupe d'octets s'élève à 10 (cnt__%), le nombre d'octets composant un groupe s'élève à 40, et le nombre d'octets servant à faire la marge s'élève à 120 (160-40).

Exemple : EXAMPLES\PROGRAMS.LOW\PATMOV.LST

MERGE: GFA_ALIB.3_0\PATMOV.LST

GFA_ALIB.3_0\GFA_GLIB.ALL\PLOAD.LST

GFA_ALIB.3_0\GFA_GLIB.ALL\POPPAL.LST

```
GOSUB gfa_poppal__
palette__$=lib_rv__$
GOSUB gfa_pload__("\EXAMPLES\STUFF\DEMO.PI1",XBIOS(2),0)
SGET image__$
,
REPEAT
    SPUT image__$
    ALERT 1," | Commencer? | ",1," OUI ",button|
    ,
    HIDEM
    GOSUB gfa_patmov__(XBIOS(2)+160*100+80,XBIOS(2),100,80,80)
    SHOWM
    ,
    ALERT 2," | Recommencer? | ",1," OUI | NON ",button|
UNTIL button|=2
~XBIOS(6,L :V :palette__$)
```

Nous nous servons tout d'abord de POPPAL pour sauvegarder la palette actuelle des couleurs, puis de PLOAD pour charger l'image se trouvant dans le fichier DEMO.PI1 ainsi que sa palette de couleurs et enregistrer le contenu de l'écran dans une variable-string (SGET).

Vient ensuite une boucle REPEAT... UNTIL qui nous sert à recopier le quart inférieur droit dans le quart supérieur gauche de l'écran.

Le quart de l'écran commence à la ligne 100 (\Rightarrow 160 octets par ligne * 100 lignes) c'est-à-dire au milieu de l'écran (\Rightarrow + 80 octets) ; il s'étend sur une hauteur de 100 lignes (demi-hauteur de l'écran) et sur une demi-largeur d'écran (80 octets = 160 pixels = un groupe d'octets). Chaque groupe d'octets est donc séparé du suivant par une marge de 80 octets, puisque l'addition d'un groupe d'octets avec la marge doit être égale à une ligne entière, qui compte 160 octets.

Nous lançons PATMOV, qui sert à déplacer ce quart inférieur droit de l'écran vers le quart supérieur gauche (début d'écran = XBIOS(2)), processus que l'on peut répéter à volonté.

Pour sortir de ce programme de démonstration, cliquer sur le bouton 'NON' de la boîte de dialogue.

Divers : NINPUT

Utilisation : Saisie de nombre positifs, compris entre certaines limites à définir, à partir de n'importe quelle position, pixel par pixel .

Dossier : GFA_ALIB.3_0\NINPUT.LST.
(quelle que soit la résolution)

INLINE : Ne nécessite aucun fichier INLINE

Exemples : EXAMPLES\PROGRAMS.LOW\NINPUT.LST
(basse résolution)

EXAMPLES\PROGRAMS.HIG\NINPUT.LST
(haute résolution)

Syntaxe : GOSUB
 gfa_ninput__(cnt__%,x__%,y__%,mih__%,max__%)

Paramètres :

- *cnt__%* :

Nombre de rangs numériques par nombre ; ce paramètre doit recevoir une valeur permettant la saisie des nombres compris entre *mih__%* et *max__%*, valeur qui ne peut être inférieure ou égale à 0 ou supérieure à 9, faute de quoi la saisie sera impossible et *lib_rv__%* retournera la valeur 2.

Pour pouvoir saisir par exemple un nombre compris entre 10 et 100, il ne faut pas affecter à *cnt__%* la valeur 1 (ou moins), car elle ne permettrait que la saisie de nombres compris entre 0 et 9 ; il est par contre tout à fait possible de prévoir un nombre de rangs numériques supérieur aux besoins réels.

- *x__%* :

Coordonnée X de la position à laquelle le nombre doit être entré ; il s'agit d'une position mesurée en pixels

- *y*__% :

Coordonnée Y de la position à laquelle le nombre doit être entré, mesurée aussi en pixels ; la saisie des ces coordonnées (*x*__%,*y*__%) se fait exactement comme sous l'instruction TEXT du GfA Basic.

- *min*__% :

Valeur minimale ; les valeurs numériques à entrer seront supérieures ou égales à cette valeur minimale ; on confirme la saisie en appuyant sur la touche <return> et en cas d'erreur, il faut recommencer la saisie, car on ne peut ressortir de la procédure NINPUT qu'après avoir entré des valeurs correctes.

- *max*__% :

Valeur maximale ; les valeurs numériques à entrer seront inférieures ou égales à cette valeur maximale ; on confirme la saisie en appuyant sur la touche <return> et en cas d'erreur, il faut recommencer la saisie.

Attention : ▽

La valeur maximale *max*__% ne saurait en aucun cas être inférieure à la valeur minimale *min*__%, faute de quoi aucune saisie ne serait possible, et *lib_rv*__% retournerait la valeur -2.

Valeur retournée :

*lib_rv*__% =

Nombre saisi ou code d'erreur ; cette variable contient la valeur entrée, retournée par la procédure NINPUT, et qui est comprise entre les limites précisées à l'aide de *min*__% et *max*__% (incluses).

La procédure retournera la valeur -2 et interdira toute saisie de nombre si vous entrez des paramètres inexacts (par exemple *max*__% < *min*__%).

Description :

La procédure NINPUT représente la solution idéale à tous vos problèmes de saisie formatée de nombres positifs ; comme il est possible de déterminer des limites, on obtient toujours un résultat précis et exploitable.

Cela rend superflue la saisie fastidieuse de nombres, avec tous les contrôles qui s'ensuivent généralement.

NINPUT s'accompagne de possibilités diverses, comme le positionnement de la saisie et le choix des attributs de texte à conférer aux nombres saisis. Cette dernière possibilité vient de ce que la procédure NINPUT utilise l'instruction TEXT pour tous les affichages, si bien qu'il est possible de recourir à des instructions DEFTEXT pour fixer sous NINPUT les même attributs que sous TEXT. La couleur des caractères, les attributs de sortie (gras, souligné etc) l'angle du texte et la hauteur des caractères sont ainsi variables sous NINPUT.

Qui plus est, NINPUT veille à ce que l'appui (pour confirmation de la saisie) sur la touche <return> n'entraîne pas la destruction d'un éventuel masque d'affichage à l'écran : en effet, à la différence de ce qui se passe avec l'instruction INPUT lors d'une saisie erronée, il faut dans cette procédure répéter la saisie dans la ligne suivante.

NINPUT ne permet que la saisie de nombres, si bien que tout appui sur une touche alphabétique reste sans effet.

Exemple : EXAMPLES\PROGRAMS.LOW\NINPUT.LST

MERGE : GFA_ALIB.3_0\NINPUT.LST

```
DEFTEXT 1,0,0,6                   ! précision des attributs
```

```
,
```

```
REPEAT
```

```
  CLS
```

```
  PRINT AT(1,1) ;"veuillez entrer un nombre compris"
```

```
  PRINT AT(1,2) ;"entre 12 et 6500"
```

```
,
```

```
  GOSUB gfa_ninput__(4,120,40,12,6500)
```

```
,
```

```
  PRINT AT(1,4) ;"valeur saisie : " ;lib_rv__%
```

```
,
```

```
ALERT 2," | Recommencer? | ",1," OUI | NON ",button|  
UNTIL button|=2
```

Comme la procédure NINPUT recourt à l'instruction TEXT pour tous les affichages, il est possible de fixer des attributs d'affichage à l'aide de la commande DEFTEXT : il est par exemple possible d'afficher le texte en le faisant tourner de 90 degrés. Dans l'exemple ci-dessus, nous n'avons pas fait usage de la commande DEFTEXT pour fixer des attributs ; nous nous limitons à déterminer la hauteur des caractères à l'aide du paramètre 6, ce qui correspond à la hauteur normale en basse et moyenne résolution.

L'instruction DEFTEXT est suivie de l'envoi d'un message demandant la saisie d'un nombre compris entre 12 (min__%) et 6500 (max__%) ; nous attribuons la valeur 4 à cnt__% pour pouvoir entrer les nombres à quatre rangs supérieurs à 1000, puisque notre valeur maximale s'élève à 6500 ; l'affichage doit se faire à la position 120,40 (x__%,y__%).

La variable numérique (variable integer) lib_rv__% nous renvoie la valeur correcte après sa saisie (confirmer votre saisie en appuyant sur la touche <return>, et cette valeur ensuite exploitée normalement, par exemple en étant affichée à l'aide de l'instruction PRINT du GfA Basic.

Dans notre programme de démonstration, on peut répéter à volonté la saisie à l'aide de NINPUT, puisque cette procédure est lancée (GOSUB...) à l'intérieur d'une boucle REPEAT... UNTIL.

Pour ressortir du programme, cliquer sur le bouton 'NON' de la boîte de dialogue envoyée à la fin du parcours de la boucle.

Divers : PINPUT

Utilisation : Saisie de chaînes de caractères (strings) d'une longueur à définir, à partir de n'importe quelle position, pixel par pixel.

Dossier : GFA_ALIB.3_0\PINPUT.LST
 (quelle que soit la résolution)

INLINE : Ne nécessite aucun fichier INLINE

Exemples : EXAMPLES\PROGRAMS.ALL\PINPUT.LST
 (quelle que soit la résolution)

Syntaxe : GOSUB gfa_pinput__(x__%,y__%,length__%)

Paramètres :

- x__% :

Coordonnée X

- y__% :

Coordonnée Y ; les coordonnées x__% et y__% désignent des positions mesurées en pixels, exactement comme sous l'instruction TEXT du GfA Basic et non comme sous l'instruction PRINT.

- length__% :

Longueur de la chaîne de caractères à saisir ; ce paramètre permet de limiter la chaîne de caractères à un certain nombre de caractères.

Valeur retournée :

lib_rv__% =

Chaîne de caractères saisie

Description :

Un curseur apparaît à la position indiquée par les coordonnées x__%,y__%, et il est alors possible de saisir des caractères au clavier ; le mode de fonctionnement de la procédure est exactement semblable à celui de l'instruction INPUT du GfA Basic (touches fléchées pour le curseur, backspace, delete etc) à la différence qu'il faut ici préciser la longueur du string à saisir.

Les touches 'cursor up' et 'cursor down' (touches fléchées vers le haut et le bas) vous permettent de vous déplacer respectivement jusqu'au début / à la fin du texte en cours de saisie.

Exemple : EXAMPLES\PROGRAMS.ALL\PINPUT.LST

MERGE : GFA_ALIB.3_0\PINPUT.LST

```
GOSUB gfa_pinput__(10,100,10)
PRINT AT(1,20) ;lib_rv__$
```

Vous pouvez maintenant saisir une chaîne de 10 caractères à la position (10,100), chaîne qui va s'afficher à l'écran.

Divers : NB5060

Utilisation : Faire passer de 50 à 60 Hz la fréquence de balayage de l'écran.

Dossier : GFA_GLIB.3_0\NB5060.LST
(basse et moyenne résolution)

INLINE : Ne nécessite aucun fichier INLINE.

Exemples : EXAMPLES\PROGRAMS.LOW\NB5060.LST
(basse et moyenne résolution)

Syntaxe : GOSUB gfa_nb5060__(value__%)

Paramètre :

- *value__%* :

Fréquence de balayage de l'image à l'écran ; *value__%* peut prendre la valeur 50 ou 60, correspondant à la fréquence de 50 ou 60 Hertz. Si vous entrez une autre valeur, *lib_rv__%* retournera la valeur 1 et la fréquence ne sera pas modifiée ; lorsque *value__%* se voit affecter une valeur correcte (50 ou 60), *lib_rv__* retourne la valeur 0.

Valeur retournée :

lib_rv__% = code d'erreur ; *lib_rv__%* retourne la valeur 0 lorsque *value__%* reçoit une valeur adéquate (50 ou 60) et la valeur 1 pour toute autre valeur non autorisée.

Description :

La procédure NB5060 vous permet de modifier la fréquence de balayage sur les moniteurs couleurs ; vous avez le choix entre la fréquence 50 (l'image se 'reconstruit' 50 fois par seconde à l'écran) et la fréquence 60 (60 images par seconde).

Avec les moniteurs couleurs, il est vivement recommandé de travailler le plus souvent possible en 60 Hertz, car cela diminue considérablement la fatigue visuelle. Attention : si vous sortez vos données de l'Atari ST vers un écran de télévision, il se peut fort que ce dernier soit incapable d'assumer une fréquence de 60 Hertz : il convient de lire le mode d'emploi du téléviseur pour vérifier s'il peut supporter cette fréquence.

Cette procédure reste sans effet lorsque vous travaillez avec un moniteur monochrome.

Exemple : EXAMPLES\PROGRAMS.LOW\NB5060.LST

MERGE : GFA_ALIB.3_0\NB5060.LST

REPEAT

```
ALERT 2," | passer sur | 50 Hertz ? ",1, " OUI ",button|
```

```
GOSUB gfa_nb5060__(50)
```

```
ALERT 2," | passer sur | 60 Hertz ? ",1, " OUI ",button|
```

```
GOSUB gfa_nb5060__(60)
```

```
ALERT 2," | Répéter ? | ",1, " OUI | NON ",button|
```

UNTIL button|=2

Lors de son premier appel, la procédure fait passer la fréquence de l'écran sur 50 Hz, ce qui revient à 'reconstruire' l'image 50 fois par seconde.

En confirmant le deuxième message envoyé par le programme, l'utilisateur relance la procédure NB5060, qui fait alors passer l'écran sur une fréquence de 60 Hz : l'image est reconstruite 60 fois par seconde sur l'écran du moniteur.

On peut répéter à volonté ce changement de fréquence, mais cela n'est guère conseillé car de nombreux changements rapides de la fréquence s'avèrent nuisibles pour le moniteur.

Cliquez sur le bouton 'NON' de la troisième boîte de dialogue pour mettre fin au programme de démonstration.

Divers : RESET

Utilisation : Réinitialisation de la configuration sans l'éteindre (reset à chaud).

Dossier : GFA_ALIB.3_0\RESET.LST
 (quelle que soit la résolution)

INLINE : GFA_ALIB.3_0\RESET.INL

Exemples : EXAMPLES\PROGRAMS.ALL\RESET.LST
 (quelle que soit la résolution)

Syntaxe : GOSUB gfa_reset__

Paramètre :

Aucun

Valeur retournée :

lib_rv__% = code d'erreur ; lib_rv__% retourne la valeur -1 lorsque vous avez oublié de charger le fichier INLINE accompagnant la procédure et la valeur 0 lorsque tout se déroule normalement.

Description :

La procédure RESET vous permet de lancer une réinitialisation de la configuration à partir d'un programme (softwarereset).

Attention : ▽

Cela interrompt le programme en cours et vous ramène au niveau du bureau GEM (desktop).

L'effet de cette réinitialisation est exactement le même que celui obtenu en appuyant sur le bouton 'reset' se trouvant au dos de votre Atari ST ; la procédure RESET est la méthode la plus radicale qui soit pour ressortir d'un programme en GfA Basic (!) ce qui peut souvent s'avérer utile.

Exemple : EXAMPLES\PROGRAMS.LST\RESET.LST

MERGE : GFA_ALIB.3_0\RESET.LST

```
ALERT 2," | software | reset ? ",2, " OUI | NON ",button|
,
IF button|=1
  GOSUB gfa_reset__
ENDIF/
```

Dans notre programme de démonstration, nous commençons par envoyer un message demandant à l'utilisateur s'il souhaite réellement lancer une réinitialisation du système. S'il clique sur le bouton 'NON', rien ne se passe et il ressort du programme, alors que s'il clique sur le bouton 'OUI', il lance effectivement une réinitialisation de sa configuration, et se retrouve ensuite au niveau du bureau GEM.

!

512 couleurs.	88
-----------------------	----

A

Agrandissement	107
Animation.	39
Apparition d'une image.	139

B

Bibliothèque.	15
-----------------------	----

C

Compression d'image	101
Couleurs.	87
Cycle de couleurs.	25

D

Décompression d'images	105
Défilement vertical.	175
Déformation de caractère 2D.	163
Déformation de caractère 3D.	169
Dégradés de gris	131
Démonstration	15
Déplacement de bloc mémoire.	203
Désactivation de l'écran.	201
Digitalisation sonore.	181
Disparition d'image	135

F

Fondu-enchaîné	36, 136
Fréquence de balayage	213

H

HORLOGE197

I

INLINE 16, 17, 23
Interruption 18

L

Loupe107

M

Masque 111, 117, 121, 123

P

Palette de couleurs 29, 31
Paramètres 19
Police de caractères 71, 93, 141, 147, 151

R

Réinitialisation 215
Résolution 18, 81, 85

S

Saisie de caractères 211
Saisie de nombre 207
SAVE 17
SAVE,A 17
Scrolling horizontal 147
Scrolling vertical 157
Son MIDI 193
Sprite 45

Surimpression125
Synthèse vocale.185

V

Variables.	18
--------------------	----

X

XBIOS(2).	33
XBIOS(6).	31

Achevé d'imprimer
sur les presses de l'imprimerie IBP
à Rungis (Val-de-Marne 94) (1) 46.86.73.54
Dépôt légal - Septembre 1990
N° d'impression: 5359

ROUTINES GRAPHIQUES ET SONORES

Hans-Peter BURK/Helmut MICKO

Développeurs, voici une nouvelle série de routines et programmes en GFA Basic 3.0 destinée à faciliter considérablement vos travaux tels que : la gestion d'écran, d'images, de sprites mais aussi du son. Ces routines se présentent sous forme de fichiers ASCII prêts à l'emploi, pouvant être intégrés directement au sein des programmes en GFA (à partir de la version 3.0). Parmi les quarante proposées, certaines existent sous plusieurs versions selon la résolution d'écran utilisée. Disposez de toutes les explications nécessaires à leur exploitation : rôle, syntaxe, paramètres, adresses spécifiques... ainsi que de plusieurs programmes de démonstration.

PARMI LES SOUS-PROGRAMMES PROPOSÉS :

GRAPHISME

COLANI : provoque une rotation des couleurs.

PSHOW : scrolling d'images ou de morceaux d'images.

INTANI : animation d'une partie de l'écran.

L-SHAP 16 : crée des sprites d'une largeur de 16 par 200 ou 400 pixels.

MRES : divise l'écran en secteurs de résolutions différentes.

POLICE : affichage de chaînes de caractères personnalisées.

L-HSCROL : scrolling horizontal de textes.

H-LOUPE : agrandissement d'un secteur de l'écran selon un facteur paramétrable.

GRAF3D : affichage de lettres en 3 dimensions avec déformation, rotation, agrandissement...

SON

SPEECH : restitution vocale d'une chaîne de caractères.

SMPLAY : digitalisation et restitution de morceaux de musique, de bruit, de paroles...

REPSND : exécution de sons sous interruption.

MIDIKY : restitution d'un fragment sonore suivant un clavier midi.



9 782868 993762

Réf. ML 855. Prix: 345 F.
ISBN: 2-86899-376-1 / ISSN: 0980-1928

EDITIONS MICRO APPLICATION

58 RUE DU FAUBOURG POISSONNIERE 75010 PARIS
TEL (1) 47 70 32 44