

AMIGA NEWS-TECH

EN VENTE UNIQUEMENT PAR ABONNEMENT



**JOYEUX
NOEL !**

45 F.

Ce numéro 28 de l'Amiga NewsTech a été tiré à 3500 exemplaires.
Amiga NewsTech (ANT) est une publication Commodore Revue SARL 5, rue de la Fidélité, 75010 PARIS, France

Déjà Noël ! Dans quelques semaines, les cadeaux, les bisous, les vacances aux Deux Alpes, etc. Et après ça, le nouvelle an, re-cadeaux, re-bisous...
Ah ! Quelle période merveilleuse !

Ce numéro de décembre de l'ANT marquera la fin d'une année riche en événements. D'abord, l'émancipation du magazine, bien sûr, mais aussi le changement de formule puis de Rédacteur en Chef d'Amiga Revue, la naissance du fils d'icelui, le licenciement d'Alain Prost par Ferrari...

Quant au prochain numéro de janvier, il marquera le début d'une nouvelle année pleine de surprises. Avec notamment pas mal de nouvelles rubriques (Pascal, algorythmie...) et l'arrivée dans nos rangs d'une célèbre signature : mister Thomas Landspurg himself.

Ce n'est évidemment pas tout, mais je n'en dirai pas plus maintenant pour ne pas gâcher l'effet dramatique...

Stéphane Schreiber

PS : Le (superbe) dessin de couverture est signé Olivia de Berardinis et a été extrait du CD-Rom Exocita Rom édité (pour Mac) par Gazelle Technologies, Inc.

OURS

Amiga NewsTech est une publication à la fois mensuelle et vendue uniquement par abonnement d'Amiga Revue, 5 rue de la Fidélité, 75010 Paris.

L'enfoiré qui a un jour décidé qu'on n'aurait que 32 pages s'appelle Jean-Yves Primas. C'est le Directeur de Publication et rien qu'à le voir, on comprend qu'il ne programme pas, lui. Fais gaffe à toi, on s'en vengera.

Celui qui fait tout ici, c'est Bibi, alias Moi, alias Myself, alias enfin Stéphane Schreiber. Et quand je dis que je fais tout, je déconne pas, je fais vraiment tout : rédaction en chef, secrétariat de rédaction, chef de rubriques (oui, au pluriel !), chef de fab. PAO, piges... Tout ça pour un salaire de misère. Le communisme a mal choisi son moment pour se casser la gueule, c'est moi qui vous le dis.

Heureusement, quelques bonnes âmes m'aident à remplir ces pages. Par ordre d'apparition à l'écran, ce sont Denis Jarril, François Lionet, François Gueugnon, Herr Doktor Von GluttenStimmellmDorf, Jérôme Etienne, Denis Genot, Max, Loïc Far et les inséparables Ok et Cancel. Qu'ils soient ici publiquement tous remerciés, autant qu'ils le méritent (c'est-à-dire un peu plus qu'ils sont payés, eux aussi...).

Pour ce qui est de la fabrication, on fait appel à des gens sympas, qui acceptent nos délais sans broncher, pourvu qu'on les invite à bouffer de temps en temps. Ainsi par exemple, notre flasheur ColorWay (5, rue de la Fidélité, 75010 Paris, Tel: 47.70.87.87) est plutôt cool. Remarquez, y z'ont intérêt, vu qu'y sont juste en dessous de chez nous et que si y nous emmerdent, on va y foutre le feu, non mais.

L'imprimeur par contre, on le connaît un peu moins bien parce que ça fait pas longtemps qu'on travaille avec lui. Par exemple, moi, je sais seulement qu'il s'appelle INO-Montrouge, qu'il habite au 9, rue Auber, 92120 Montrouge et que son numéro de bigophone est le 46.56.27.69. Il sait pas encore sur qui il est tombé, mais il va vite comprendre.

Pour terminer, les abonnements sont gérés sur Thomson MO5 avec 2 Ko de mémoire centrale par MCM Europe, 16, quai Jean-Baptiste Clément, 94140 Alfortville. Tel : 43.78.92.10. Si vous avez Claude au bout du fil, faites lui une bise de ma part.

Rapidement, une petite illustration du fonctionnement des overlays, avec un programme parfaitement inutile, qui ne fera certainement pas date dans l'histoire de la presse informatique...

Il faut quand même reconnaître qu'il n'est pas évident de démontrer par un court programme l'utilisation - mais aussi et surtout l'utilité - des overlays, mais j'ai tout de même décidé de le faire. Les listings qui suivent font donc partie d'un seul programme nommé "Ovls" (raccourcis pratique pour 'Overlays'...). Dans l'ordre, nous avons la Racine (Racine.c), 3 overlays de niveau 1 (Ovl1.c, Ovl2.c et Ovl3.c) et deux overlays de niveau 2 (Ovl3_1.c et Ovl3_2.c). Un fichier WITH pour BLink est également inclus, histoire de montrer comment il est conçu.

Sans transition, je vous laisse méditer là-dessus et vous donne rendez-vous au mois prochain pour de nouvelles aventures.

Listing 1 : Racine.c

```

/*****
 * Arborescence des overlays :
 *
 *          +-----+
 *          | Racine.c |
 *          +-----+
 *
 *          |
 *
 *      +-----+ +-----+ +-----+
 *      | Ovl1.c +--+ Ovl2.c +--+ Ovl3.c |
 *      +-----+ +-----+ +-----+
 *
 *          |
 *          +-----+ | +-----+
 *          | Ovl3_1.c +--+ Ovl3_2.c |
 *          +-----+ +-----+
 *****/
#include <stdio.h>

/* Fonctions de main.c */
int GetChar(void);

/* Fonctions du premier niveau de l'arbre */
extern void Ovl1(void); /* dans Ovl1.C */
extern void Ovl2(void); /* dans Ovl2.C */
extern void Ovl3(void); /* dans Ovl3.C */

/* fonction main() */
void main(int argc, char **argv)
{
    int c = 0;

    if (!argc)
        return; /* N'accepte pas le lancement depuis le WB */

    /* Petit menu (très) facile... */
    while (c != 'q' && c != 'Q')
    {
        printf("\014Démonstration du mécanisme des Overlays.\n");
        printf("\n*** Ici la Racine\n\n");

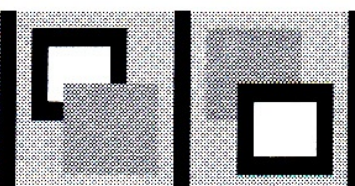
        printf("1 - Appeler l'overlay 1\n");
        printf("2 - Appeler l'overlay 2\n");
        printf("3 - Appeler l'overlay 3\n");
        printf("Q - Quitter\n");
        printf("Appeler quel overlay (1/2/3) ? ");

        switch(c = GetChar())
        {
            case '1':
                Ovl1();
                break;

            case '2':
                Ovl2();
                break;

            case '3':
                Ovl3();
        }
    }
}

```

```

        break;

    default:
        break;
}
}

int GetChar(void)
{
    int c1, c2;

    c1 = getchar();
    while ((c2 = getchar()) != '\n');
    return(c1);
}

```

Listing 2 : Ovl1.c

```

#include <stdio.h>

extern int GetChar(void); /* dans Racine.c */

void Ovl1(void)
{
    printf("\n*** Ici l'overlay 1.\n");

    printf("Une touche pour retourner à la Racine...");
    GetChar();
}

```

Listing 3 : Ovl2.c

```

#include <stdio.h>

extern int GetChar(void); /* dans Racine.c */

void Ovl2(void)
{
    printf("\n*** Ici l'overlay 2.\n");

    printf("Une touche pour retourner à la Racine...");
    GetChar();
}

```

Listing 4 ; Ovl3.c

```

#include <stdio.h>

extern int GetChar(void); /* dans Racine.c */
extern void Ovl3_1(void); /* dans Ovl3_1.c */
extern void Ovl3_2(void); /* dans Ovl3_2.c */

void Ovl3(void)
{
    int c = 0;

    while (c != 'r' && c != 'R')
    {
        printf("\014*** Ici l'overlay 3.\n");
        printf("A - Appeler l'overlay 3_1\n");
        printf("B - Appeler l'overlay 3_2\n");
        printf("R - Retourner à la Racine\n");
        printf("Qu'est-ce qu'on fait (A/B/R) ? ");

        switch(c = GetChar())
        {
            case 'a':
            case 'A':
                Ovl3_1();
                break;

            case 'b':
            case 'B':
                Ovl3_2();
                break;
        }
    }
}

```

```

        break;
    }
}

Listing 5 : Ovl3_1.c

#include <stdio.h>

extern int GetChar(void); /* dans Racine.c */

void Ovl3_1(void)
{
    printf("\n*** Ici l'overlay 3_1.\n");

    printf("Une touche pour retourner à l'overlay 3...");
    GetChar();
}

```

Listing 6 : Ovl3_2.c

```

#include <stdio.h>

extern int GetChar(void); /* dans Racine.c */

void Ovl3_2(void)
{
    printf("\n*** Ici l'overlay 3_2.\n");

    printf("Une touche pour retourner à l'overlay 3...");
    GetChar();
}

```

Listing 7 : makefile

```

Ovls:      Racine.o Ovl1.o Ovl2.o Ovl3.o Ovl3_1.o Ovl3_2.o
          Blink WITH Ovls.lnk

Racine.o:  Racine.c
          Lc Racine.c

Ovl1.o:    Ovl1.c
          Lc Ovl1.c

Ovl2.o:    Ovl2.c
          Lc Ovl2.c

Ovl3.o:    Ovl3.c
          Lc Ovl3.c

Ovl3_1.o:  Ovl3_1.c
          Lc Ovl3_1.c

Ovl3_2.o:  Ovl3_2.c
          Lc Ovl3_2.c

```

Listing 8 : Ovls.lnk (fichier WITH pour BLink)

```

ROOT lib:c.o Racine.o
TO Ovls
LIBRARY lib:lc.lib,lib:small.lib
SMALLCODE SMALLDATA NODEBUG NOICONS

OVERLAY
Ovl1.o
Ovl2.o
Ovl3.o
*Ovl3_1.o
*Ovl3_2.o
#

```



Salut, c'est encore moi ! Le mois dernier, je vous avais promis un traceur de courbes en AMOS. Je tiens mes promesses, voici un traceur de courbes en AMOS. Quel homme de parole !

C'est une application directe des procédures du mois dernier ; il suffit donc de les récupérer sous l'éditeur AMOS. Il faut tout de même en modifier quelques unes :

- **EVALINIT** : une nouvelle procédure qui se charge de définir les fonctions disponibles, juste histoire d'être un peu plus structuré. Dans votre programme, vous définissez les tableaux, et vous appelez EVALINIT.

- **EVALUE** : j'avais laissé un petit bug. Hum. Il est maintenant corrigé.

- **EVALIT** : il faut ajouter une nouvelle routine pour traiter la variable X. Celle-ci s'appelle VARX. Notez que j'ai également ajouté une gestion des erreurs : la variable **SYNT** est positionnée à -1 en cas d'erreur d'évaluation.

Les autres procédures ne sont pas modifiées, copiez les du programme du mois dernier (si vous avez loupé le dernier numéro, vous trouverez les routines sur la disquette du magazine).

Le reste du programme se passe de commentaires, si j'ose dire car justement, il en est truffé, le programme, de commentaires.

Remarquez cependant que tous les paramètres d'affichage de la courbe sont entrés au moyen des routines d'évaluation. Lorsque vous analysez un sinus par exemple, vous pouvez taper -2π au lieu de -6.2832. C'est quand même plus pratique.

Remarquez également la procédure **_GET_VALUE** : je remets l'ancienne valeur du paramètre dans le buffer clavier. Pas besoin de retaper tout à chaque fois.

C'est tout pour aujourd'hui ! Ouf, finies les maths... Je laisse le clavier à mon chien, qui devient de plus en plus mégalomane. Mais enfin sa petite démo n'est pas mal du tout.

```

'-----
' Traceur de courbes AMOS
' Par Francois Lionet
'-----
'
' Definition des variables
MXE=25
Dim PILE(MXE),PILE$(MXE),PP(MXE),PP$(MXE)
Global OPE$,FUN$,EV$,LEV$,SYNT,PILE(),PILE$(),PILE#(),PPILE
Global PP(),PP#(),PP,PPAR,CHAR$
'
' Nouvelle variable
Global XX#
'
' Appelle la nouvelle procedure
EVALINIT
'
' Ouverture de l'ecran
Screen Open 0,640,256,4,Hires
Flash Off : Palette 0,$FFF,$62,$FF0
Paper 0 : Pen 1
'
' Definition de la fenetre d'affichage de la courbe
DX=8 : DY=8 : TX=624 : TY=192
'
FONC$="Sin(X)"
DEBX#=-Pi#
FINX#=Pi#
PAS#=320
DEBY#=-1
FINY#=1

```

```

' Entree des expressions
Do
'
Cls
Centre At(,2)+">>>> Le bo traceur de courbes de Daisy <<<" :
Print
Locate 0,10
Print "Entrer une ligne vide pour quitter le programme."
Print
'
' Demande et verifie la fonction.
Do
Put Key FONC$
Line Input "Fonction à evaluer: ";A$
Exit If A$="" : 2 : _TOKENISE[A$] : Exit If Not SYNT
Print "Erreur de syntaxe. Veuillez ré-entrer
l'expression..."
Loop
FONC$=A$
'
' Intervalle d'etude
Do
_GET_VALUE[DEBX#,"Début de l'intervalle en X"] :
DEBX#=Param#
_GET_VALUE[FINX#,"Fin de l'intervalle en X"] : FINX#=Param#
Exit If FINX#>DEBX#
Print "En général, le début est inférieur à la fin. Juste
pour dire..."
Loop
'
' Pas du dessin
_GET_VALUE[PAS#,"Nombre de pas de dessin"] : PAS#=Param#
'
' Intervalle d'affichage
Do
_GET_VALUE[DEBY#,"Début de l'intervalle en Y"] :
DEBY#=Param#
_GET_VALUE[FINY#,"Fin de l'intervalle en Y"] : FINY#=Param#
Exit If FINY#>DEBY#
Print "Le début, y'en a moins grand que la fin. Ok?"
Loop
'
' Va tokeniser pour de bon!
_TOKENISE[FONC$]
'
' Dessine une beeelle boite toute verte
Cls : Curs Off
Clip : Gr Writing 1
Set Pattern 0 : Set Paint 1 : Ink 2,2,1
Bar DX-4,DY-4 To DX+TX+4,DY+TY+4
Gr Writing 0 : Ink 3
'
' Dessine l'axe en X
Draw DX+TX/2,DY To DX+TX/2,DY+TY
Draw DX+TX/2-4,DY To DX+TX/2+4,DY
Draw DX+TX/2-4,DY+TY To DX+TX/2+4,DY+TY
Text DX+TX/2+2,DY+6,Str$(FINY#)
Text DX+TX/2+2,DY+TY,Str$(DEBY#)
'
' Dessine l'axe des Y
Draw DX,DY+TY/2 To DX+TX,DY+TY/2
Draw DX,DY+TY/2-4 To DX,DY+TY/2+4
Draw DX+TX,DY+TY/2-4 To DX+TX,DY+TY/2+4
Text DX,DY+TY/2+8,Str$(DEBX#)
A$=Str$(FINX#) : Text DX+TX-8*Len(A$),DY+TY/2+8,A$
'
' Calcule les coefficients de multiplication
STP#=(FINX#-DEBX#)/PAS#
MULX#=TX/(FINX#-DEBX#)
MULY#=TY/(FINY#-DEBY#)
'
' Boucle de trace de la courbe
O=0 : Ink 1
For XX#=DEBX# To FINX# Step STP#
' Appel de l'evaluation
EVALUE
' Si pas d'erreur
P=0
If Not SYNT
' Calcule la position sur l'ecran
X#=XX#-(DEBX#+FINX#)/2.0
Y#=Param#-(DEBY#+FINY#)/2.0
X=TX/2+X#*MULX# : Y=TY/2-Y#*MULY#
'
If Y>=0 and Y<=TY
If O
Draw To DX+X,DY+Y : P=1
Else
Plot DX+X,DY+Y : P=1
End If

```

(suite page 31)

"Petite démo" ? Comment ça, "petite démo"? Le programme que j'ai tapé de mes pattes poilues va révolutionner le monde de l'Amiga... C'est tout simplement du concentré de génie. Non mais !

Tiens, comme je suis bonne patte, je vais essayer de vous en expliquer le fonctionnement... Vous connaissez les déformations sinusoïdales ? En voici une toute simple à réaliser, sur plus de 200 lignes d'écran et au 1/50 ième de secondes, s'il vous plaît ! Elle est de plus totalement paramétrable. J'explique.

Il n'est même pas envisageable de calculer une déformation sinusoïdale en temps réel, au 1/50 ième sur 200 lignes : c'est déjà très difficile en langage machine, donc impossible en AMOS. Il faut donc ruser et calculer tout à l'avance : les images étant déjà dessinées, il suffira de les faire défiler à l'écran l'une après l'autre. Bon.

La première idée qui vient à l'esprit est d'ouvrir plusieurs écrans et de les faire défiler avec l'instruction *SCREEN TO FRONT*. C'est pas mal, mais nous sommes limités à 8 images au maximum, trop peu pour une bonne animation. Même en passant les écrans en double buffer, on reste encore limités à 16 images, encore trop peu, surtout si l'on veut faire défiler un texte par dessus.

Mon idée de génie, je dois le dire, a été de réserver une banque de mémoire CHIP, contenant la place nécessaire à toutes les images pré-calculées. Par exemple, pour 20 pas d'animation d'une image de 320x200 en 2 couleurs, la taille de cette banque est de 160 Kilos.

Comment afficher le contenu de cette banque dans un écran ? Mon deuxième coup de génie, je dois l'avouer modestement, est d'utiliser l'instruction *SCREEN OFFSET*. Car avez-vous remarqué que cette instruction permet d'explorer toute la mémoire graphique de l'Amiga, simplement en utilisant de grandes valeurs en Y ? En calculant la distance entre la grosse banque de mémoire et l'écran numéro 0, on peut connaître les valeurs à utiliser dans l'instruction et afficher tout simplement le contenu de la banque au lieu de l'image normale.

Voilà pour la théorie, voyons le programme.

On commence par récupérer le maximum de mémoire possible, en fermant les écrans et l'éditeur. On définit ensuite les valeurs de la déformation :

* *NPAS* est le nombre d'images calculées. Plus *NPAS* est grand, plus lente est la déformation. Essayez donc 20 ou 30 pour commencer.

* *AMPX1#* et *AMPY1#* définissent la déformation en X.

* *AMPX2#* et *AMPY2#* définissent la déformation en Y. Modifiez ces valeurs et voyez le résultat.

Le programme réserve alors la grosse banque de mémoire CHIP puis ouvre l'écran 0. Il faut s'assurer que l'adresse de l'écran 0 est inférieure à celle de la banque, car *SCREEN OFFSET* n'accepte pas de valeurs négatives. On ouvre maintenant deux écrans de travail, que l'on cache le temps de calculer toutes les déformations. Celle en X est d'abord appliquée entre l'écran 0 et l'écran 1, puis celle en Y, de l'écran 1 dans l'écran 2. On peut alors utiliser l'instruction *COPY* pour copier le contenu de la bitmap dans la banque de

mémoire, en bonne position. Reste à calculer la "distance" entre l'écran 0 et la première image de la banque, et c'est tout.

C'est AMAL qui se charge de l'animation. Les registres à initialiser sont RA (valeur de base en X), RB (valeur de base en Y), RC (nombre de lignes d'une image) et (RD nombre de pas d'animation). La chaîne AMAL se contente de modifier les valeurs de *SCREEN OFFSET* en X et en Y. C'est tout.

Une fois la chaîne initialisée, on peut retourner en mode direct : l'animation marche toute seule !

Vous pourrez animer d'autres images en utilisant la même méthode. L'écran 0 peut faire partie d'un dual-playfield et dans ce cas, vous pourrez faire bouger un gros logo devant.

Avouez que c'est tout de même mieux qu'un minable traceur de courbes. Faut vraiment pas avoir d'idées pour publier quelque chose d'aussi éculé (pourquoi tant de haine ?). Holà, le voilà, il faut que je joue le bon chien-chien à son pépère et lui fasse la fête... On se voit le mois prochain, si je peux accéder au clavier.

Waf, waff, wouf ! Slurp, slurp, hop, hop, hop ! Ouah, ouah, slurp, slurp, slurp !

```

-----
' Daisy DemOS numéro 5
'
' "Le retour du come-back II"
'
' Par Daisy Lionet.
' (c) OS-Productions 1991
-----
Screen Close 0
Close Editor
Close Workbench
'
' Definition de la deformation
'
NPAS=16
NSCREEN=NPAS : SX=320 : SY=200
AMPX1#=6 : AMPY1#=0.05
AMPY2#=6 : AMPX2#=0.04
'
' Reservation de la memoire.
'
' Assure la presence d'une place EN DESSOUS de la banque
Reserve As Chip Work 9,10000
'
' Reserve la banque
'
SIZE=SX/8*SY
Reserve As Chip Work 10,SIZE*NSCREEN
'
' Libere la place en dessous de la banque
'
Erase 9
'
' Reserve l'ecran de base
'
Screen Open 0,SX,SY,2,Lowres
Curs Off : Cls 0
Colour 1,$EEE
'
' Imprime de zoulis dessins
'
Scroll Off : For X=0 To 80 : Print "Daisy DemOS !"; : Next
BONE[160,100,100,20,25]
'
' Verifie que tout peut marcher.
'
If Logbase(0)>Start(10) : Stop : End If
'
' Ouvre les deux ecrans de travail
'
Screen Open 1,SX,SY,2,Lowres
Curs Off : Screen Hide 1
Screen Open 2,SX,SY,2,Lowres
Curs Off : Screen Hide 2
'

```

(suite page 31)

AREXX : TRAVAIL PRATIQUE (III)

Voici déjà le troisième et dernier volet de notre premier programme ARExx : il concerne plus particulièrement la présentation du répertoire à l'écran.

Nous avons employé la pile pour stocker momentanément trois valeurs : le nom du disque, l'adresse en mémoire du bloc des fichiers créés dans le programme 2 et sa longueur. Nous avons pour cela utilisé trois fois l'instruction *PUSH*. Bien que cette méthode soit tout-à-fait légale, elle peut présenter un certain danger dans un système multitâches car la pile est publique. Chaque programme peut la charger ou la lire à sa guise. Il serait préférable de l'utiliser ainsi : d'abord créer une chaîne contenant un nom identificateur et les paramètres à transférer et n'effectuer qu'un seul *PUSH*. A la réception, on effectue un *PULL* qui retourne la ligne courante de la pile. On vérifie alors que le nom identificateur est correct ; si oui on continue, sinon c'est qu'un autre utilisateur s'en est servi. On stocke alors momentanément la ligne et on recommence jusqu'à la découverte de la bonne ligne. On replace ensuite dans la pile - et dans l'ordre ! - tout ce qu'on en a été extrait. Notons que ce n'est pas spécifique à ARExx mais à tout système partagé.

PARTICULARITES DU PROGRAMME

Nous avons laissé les défauts du programme précédent afin d'utiliser dans celui-ci les possibilités du traitement des chaînes de caractères. Ceci explique les différents traitements, dont l'intérêt est purement didactique.

Le bloc issu du programme précédent est une suite monolithique de caractères : il n'y a qu'un seul retour chariot en fin de bloc et bien que ce soient des caractères ASCII, ils représentent valablement tout ou partie d'un fichier standard dans un langage quelconque. Seule l'acquisition peut être différente (usage de *READCH()* par exemple).

L'homogénéisation des séparateurs (début en ligne 30) peut sembler faire double emploi avec la fonction remplaçant les séparateurs (III) par des espaces, mise dans les lignes 101 et 102. Il faut se rappeler que dans ARExx, les espaces sont des séparateurs implicites et que certains titres de répertoires ou de fichiers peuvent en contenir, ce qui augmenterait la difficulté à les trier sans les tronquer.

La console a ses définitions et ses limitations. Certaines peuvent être outrepassées, d'autres non. Cela limite la réalisation et attire la critique des esthètes. Nous en sommes conscients et d'accord !

PROCHAINS ARTICLES

Notre prochain article traitera du débogage des programmes, car l'environnement ARExx est bien muni en aides diverses au déverminage et possède des particularités intéressantes. Ensuite, nous verrons comment utiliser les "libraries" ou bibliothèques venant avec la rexxarp.library, qui permet d'accéder aux fonctions d'Intuition (menus, gadgets, etc.). Après encore, nous regarderons comment les programmes spécialement prévus pour, fonctionnent de concert avec ARExx, et en particulier, nous vous suggérons de chercher dès à présent à vous procurer cet excellent éditeur de textes du Domaine Public qu'est DME (version 1.35), réalisé par le non moins excellent Matt Dillon, qui est "Rexxable" à satiété. Il sera bien entendu fourni sur la disquette ANT relative à cet article lors de sa parution.

LISTING DU PROGRAMME

Ce programme s'appelle Présentation (n'oubliez pas l'accent aigu sur le "é")... Comme d'habitude, les numéros de ligne sont des repères à ne pas copier.

```
21 /*récupération des paramètres*/
22 PULL disque
23 PULL adresse
24 PULL longueur
25 adresse=X2C(adresse)
26 texte=IMPORT(adresse,longueur)
27 SAY CENTER('Un peu de patience...',80)
28 /*homogénéisation des séparateurs*/
29 debut=1
30 position=1
31 DO WHILE position ~=0
32   position=POS('|||||',texte,debut)
33   texte=DELSTR(texte,position+3,3)
34   debut=position+3
35 END
36 position=1
37 debut=1
38 DO WHILE position ~=0
39   position=POS(' ',texte,debut)
40   IF SUBSTR(texte,position+1,1) ~= '|' THEN
41     texte=INSERT('|||',texte,position)
42   debut=position+4
43 END
44 /*extraction des titres des répertoires*/
45 liste.0=0
46 i=0
47 a=1
48 debut=POS('<',texte,1)
49 DO WHILE a ~=0
50   a=POS(':',texte,debut)+1
51   IF b=0 THEN BREAK
52   b=POS('>',texte,debut)
53   long=b-a
54   IF long <=0 THEN BREAK
55   debut=b+1
56   chaine=STRIP(SUBSTR(texte,a,long))
57   x=REVERSE(chaine)
58   PARSE VAR x rang chaine
59   rang=REVERSE(rang)
60   chaine=REVERSE(chaine)
61   i=i+1
62   liste.i=rang ':' chaine ':' b
63   liste.0=liste.0 + 1
64 END
65 /*recherche des répertoires extérieurs*/
66 DO i=1 TO liste.0
67   inter=0
68   PARSE VAR liste.i . ':' a1 ':' .
69   DO j=i+1 TO liste.0
70     PARSE VAR liste.j . ':' b1 ':' .
71     IF STRIP(a1)==STRIP(b1) THEN inter=j
72   END
73   liste.i=inter ':' liste.i
74 END
75 /*extraction des fichiers relatifs aux répertoires*/
76 DO i=1 TO liste.0 -1
77   PARSE VAR liste.i k1 ':' k2 ':' k3 ':' k4
78   k4=STRIP(k4)
79   pos1=POS('<',texte,k4)+1
80   long=pos1-k4
81   souschaine=SUBSTR(texte,k4,long)
82   pos2=POS('|||',souschaine,1)
83   IF pos2 =0 THEN DO
84     liste.i=liste.i ';'
85     ITERATE i
86   END
87   pos3=POS('|||<',souschaine,1)
88   souschaine=SUBSTR(souschaine,2,pos3)
89   DO FOREVER
90     posx=POS('|||',souschaine,1)
91     IF posx=0 THEN BREAK
92     souschaine=DELSTR(souschaine,posx,3)
93     souschaine=INSERT(' ',souschaine,posx-1)
94   END
95   long=LENGTH(souschaine)-2
96   souschaine=SUBSTR(souschaine,1,long)
97   posx=POS('///',souschaine,5)+1
98   souschaine=SUBSTR(souschaine,posx + 2)
99   liste.i=liste.i ';' souschaine
100 END
101 nb=liste.0
102 PARSE VAR liste.nb k1 ':' k2 ':' k3 ':' k4
103 souschaine=SUBSTR(texte,k4+2)
104 pos1=POS('///|||',souschaine,1)+5
105 pos2=POS('|||///',souschaine,1)
106 souschaine=SUBSTR(souschaine,pos1,pos2-pos1)
107 DO FOREVER
108   posx=POS('|||',souschaine,1)
109   IF posx=0 THEN BREAK
110   souschaine=DELSTR(souschaine,posx,3)
111   souschaine=INSERT(' ',souschaine,posx-1)
112 END
113 liste.nb=liste.nb ';' souschaine
114 /*calcul de la taille de chaque répertoire*/
115 DO i=1 TO liste.0
```



```

129 PARSE VAR liste.i k1 ':' k2 ':' k3 ':' k4 ';' k5
130 debut=1
131 somme=0
132 DO FOREVER
133   pos1=POS('(',k5,debut)
134   IF pos1=0 THEN BREAK
135   pos2=POS(')',k5,pos1)
136   long=pos2-1-pos1
137   somme=somme+STRIP(SUBSTR(k5,pos1+1,long))
138   debut=pos2+1
139 END
140 liste.i=k1 ':' k2 ':' k3 ':' somme ';' k5
141 END
142 /*élimination des lignes doubles*/
143 ligne.0=0
144 j=1
145 DO i=1 TO liste.0
146   PARSE VAR liste.i k1 ':'
147   k1=STRIP(k1)
148   IF k1='*' THEN ITERATE i
149   IF k1~='*' THEN DO
150     liste.i=liste.k1
151     liste.k1='*'
152   END
153   PARSE VAR liste.i . ':' ligne
154   ligne.j= STRIP(ligne)
155   j=j+1
156   ligne.0=ligne.0 +1
157   DROP liste.i
158 END
159 DROP liste.0
160 /*réduction des titres*/
161 DO i=1 TO ligne.0 -1
162   PARSE VAR ligne.i rang ':' titre ':' reste
163   rang=STRIP(rang)
164   SELECT
165     WHEN rang = 0 THEN NOP
166     WHEN rang = 1 THEN NOP
167     OTHERWISE DO
168       DO j=1 TO rang-1
169         posx=POS('/',titre)+1
170         titre=SUBSTR(titre,posx)
171       END
172       ligne.i= rang ':' titre ':' reste
173     END
174   END
175 END
176 /*calcul de la taille utilisée*/
177 DO i=1 TO ligne.0
178   PARSE VAR ligne.i . ':' . ':' taille ';' .
179   somme= somme+STRIP(taille)
180 END
181 /*ouverture de l'écriture sur la console*/
182 CALL OPEN('ecran','*',WRITE)
183 CALL WRITECH('ecran','9B 37 3B 33 33 3B 34 30 6D 0B 0D 0C
9B 4D 9b 37 39 75'X)
184 CALL gras
185 CALL ecrit(CENTER('Nom du disque:disque,79))
186 CALL lligne; CALL retour; CALL normal
187 x=ligne.0
188 PARSE VAR ligne.x . ':' . ':' root ';' fichiers
189 texte=CENTER(somme 'Octets utilisés. Taille
répertoire ROOT: root, 'Octets',80)
190 CALL ecrit(texte)
191 CALL lligne; CALL retour
192 CALL ecrit(fichiers)
193 CALL lligne; CALL retour; CALL normal; CALL lligne
194 /*traitement des lignes*/
195 DO i=1 TO ligne.0 - 1
196   PARSE VAR ligne.i rang ':' nom ':' taille ';' fichiers
197   nom=STRIP(nom)
198   fichiers=STRIP(fichiers)
199   SELECT
200     WHEN rang = 1 THEN CALL misenpage(0 3 0)
201     WHEN rang = 2 THEN CALL misenpage(1 0 3)
202     WHEN rang = 3 THEN CALL misenpage(2 1 3)
203     WHEN rang = 4 THEN CALL misenpage(3 2 3)
204     WHEN rang = 5 THEN CALL misenpage(4 2 1)
205     OTHERWISE NOP
206   END
207 END
208 x=FREESPACE(adresse,longueur)
209 EXIT
210 /*A partir d'ici ce sont les sous programmes*/
211 ecrit: /*écrit le texte à l'écran */
212 arg tex
213 CALL WRITECH('ecran',tex)
214 RETURN
215 retour:
216 CALL WRITECH('ecran','0D'X) /*retour chariot*/
217 RETURN
218 dec5:
219 CALL WRITECH('ecran','9B 30 35 43'X)
220 RETURN
221 normal:

```

```

247 CALL WRITECH('ecran','9B 30 6D'X) /* typographie normale*/
248 RETURN
249 souligner:
250 CALL WRITECH('ecran','9B 34 6D'X) /*typographie souligné*/
251 RETURN
252 italique:
253 CALL WRITECH('ecran','9B 33 6D'X) /*typographie italique*/
254 RETURN
255 gras:
256 CALL WRITECH('ecran','9B 31 6D'X) /*typographie grasse*/
257 RETURN
258 lligne:
259 CALL WRITECH('ecran','9B 31 45'X) /* curseur 1 lig en bas*/
260 RETURN
261 couleur: /* couleur texte et fond*/
262 ARG fond devant
263 coul=X2C('9B 34 3'fond '3B 33 3'fond)
264 CALL WRITECH('ecran',coul)
265 RETURN
266 filtre:
267 PARSE ARG d1 d2
268 d1=STRIP(d1)
269 d2=STRIP(d2)
270 posx=POS('<',texte,d1)
271 loin=posx-d1
272 essai=SUBSTR(texte,d1,loin)
273 y=POS('|||',essai,1)
274 IF y=0 THEN DO
275   chaine=''
276   return chaine
277 END
278 pos1=POS('|||',texte,d1)
279 IF d2=0 THEN pos2=STRIP(POS('<',texte,d1))
280 ELSE pos2=d2
281 long=pos2-pos1+1
282 IF long <=0 THEN DO
283   chaine=''
284   RETURN chaine
285 END
286 chaine=SUBSTR(texte,pos1,long)
287 DO FOREVER
288   y=POS('|||',chaine,1)
289   IF y=0 THEN BREAK
290   chaine=DELSTR(chaine,y,3)
291   chaine=INSERT(' ',chaine,y-1)
292 END
293 RETURN chaine
294 misenpage:
295 ARG rg back fore
296 CALL normal; CALL retour
297 rg=STRIP(rg)
298 DO zz=1 TO rg
299   CALL dec5
300 END
301 back=STRIP(back)
302 fore=STRIP(fore)
303 CALL couleur(back fore); CALL italique
304 dimension=nom taille 'Octets'
305 CALL ecrit(dimension)
306 CALL couleur(0 1); CALL retour; CALL lligne; CALL normal;
307 CALL lligne
308 CALL misenpaquet
309 CALL normal; CALL lligne; CALL lligne; CALL couleur(0 1)
310 RETURN
311 misenpaquet:
312 espreel=''
313 espace='
314 DO es=1 to rg
315   espreel=espreel||espace
316 END
317 nblignes=LENGTH(fichiers)%(80-LENGTH(espreel))
318 reste=LENGTH(fichiers)/(80-LENGTH(espreel))
319 SELECT
320   WHEN nblignes=0 THEN DO
321     fichiers=espreel||fichiers
322     CALL ecrit(fichiers)
323   END
324   WHEN nblignes~0 THEN DO WHILE LENGTH(fichiers)>0
325     ligne=SUBSTR(fichiers,1,80
326       -LENGTH(espreel)-1)
327     fichiers=DELSTR(fichiers,1,80
328       -LENGTH(espreel)-1)
329     ligne=espreel||ligne
330     CALL ecrit(ligne)
331   END
332   OTHERWISE NOP
333 END
334 RETURN

```

Par François Gueugnon.

Ce mois-ci, nous allons étudier Flex, version AmigaDOS de lex, le célèbre utilitaire que tout utilisateur d'Unix connaît, ne serait-ce que de nom.

Flex est un outil permettant de créer rapidement des analyseurs syntaxiques (ou "scanners"). Il s'agit de programmes capables de reconnaître des motifs lexicaux (ou "patterns") dans un texte et de déclencher des actions en fonction des motifs reconnus.

Flex, dans son utilisation, est fort similaire à GAWK, étudié le mois dernier : il lit un fichier d'entrée (ou l'entrée standard si aucun nom de fichier ne lui est précisé) correspondant à la description du scanner à générer. Il se compose d'une suite d'expressions propres au langage lex et de code source en C définissant, sous forme de fonctions, les actions à effectuer lors de la reconnaissance d'un motif lexical.

A l'aide de ces données d'entrée, Flex génère un autre programme source en langage C, se nommant *lex.yy.c* et définit la routine *yylex()* qui est l'analyseur syntaxique créé à partir des règles de l'utilisateur. Cet analyseur syntaxique peut ensuite être utilisé tel quel ou intégré dans une application, comme n'importe quel autre source en C.

Une fois compilé et linké, *lex.yy.c* produit un exécutable capable d'analyser un fichier et d'exécuter un certain nombre d'actions en fonction des patterns rencontrés.

SYNTAXE

Après ce rapide descriptif des fonctionnalités de Flex, il est temps désormais d'étudier la syntaxe du fichier d'entrée. Attention, le but de cet article n'est pas de vous expliquer toutes les finesses du langage, l'intégralité de la revue n'y suffirait pas. Non, il s'agit simplement de vous familiariser avec Flex afin de vous faire découvrir toute la puissance de ce produit.

Les différentes sections d'un programme lex.

Le fichier d'entrée de Flex peut être décomposé en trois sections distinctes, séparées entre elles par la ligne `%%`. Ces trois sections sont : Définitions, Règles et Code utilisateur (en C).

La section Définitions contient la déclaration de noms comme équivalence à une définition, ceci afin de simplifier l'écriture de certaines expressions. Cette déclaration s'effectue sous la forme :

nom définition

Le nom doit impérativement commencer à la première colonne de la ligne, sous peine d'être interprété d'une autre manière comme nous le verrons plus tard. De même, la définition commence dès le premier caractère non-blanc de la ligne.

Par exemple, *CHIFFRE [0-9]* permet de remplacer dans toutes les règles, l'expression *[0-9]* définissant un chiffre par le mot *CHIFFRE*. Ainsi

```
{CHIFFRE}+"."{CHIFFRE}* replace ([0-9])+"."([0-9])
```

La section Règles contient une série de règles de la forme :

pattern action

Attention, comme dans la section Définitions, la pattern doit obligatoirement commencer à la première colonne de la ligne sous peine d'une interprétation différente.

Enfin, la section Code utilisateur contient les fonctions appelées par l'analyseur syntaxique. Il s'agit de code en langage C qui sera copié tel quel dans le fichier *lex.yy.c*

Dans les sections Définitions et Règles, tout texte indenté ou compris entre `'{'` et `'}'` est copié tel quel vers le fichier de sortie (les caractères `'{'` et `'}'` étant enlevés lors de cette opération).

Dans la section Règles, toute ligne indentée ou comprise entre `'{'` et `'}'` apparaissant avant la première règle effective est considérée comme une déclaration de variable locale à la fonction d'analyse syntaxique.

Dans la section Définitions, pour inscrire un commentaire, il suffit que la ligne ne soit pas indentée et comprise entre `'/*'` et `'*/'`. Un commentaire est recopié tel quel vers le fichier de sortie. A noter la possibilité de définir une ligne de commentaire à l'aide du `'#'` situé en première colonne de la ligne. Cette pratique est à éviter car elle risque de disparaître dans de futures versions.

La notion de "Pattern"

Le motif lexical (ou pattern) est défini par une série de règles d'écriture, pour la plupart très proches de celle de GAWK. Notre propos n'est pas de passer en revue toutes ces règles mais d'en étudier quelques unes par l'exemple.

x	recherche le caractère 'x'
.	n'importe quel caractère sauf retour chariot
[xyz]	une classe de caractères, en l'occurrence 'x', 'y', 'z'
r*	zéro ou plusieurs r, où r est une expression régulière
\123	le caractère ayant pour valeur 123 (en octal)
\x2a	le caractère ayant pour valeur 2a (en hexadécimal)

et bien d'autres encore (reportez-vous à la documentation pour plus de précisions).

Quand l'analyseur de syntaxe est lancé, il analyse le fichier et pour chaque ligne, recherche les occurrences qui correspondent à une ou plusieurs patterns. S'il en trouve plus d'une, il recherche d'abord celle dont un maximum de caractères correspond à la pattern. Si l'ambiguïté n'est pas levée, il prend la première par ordre de déclaration des règles dans le fichier lex.

Une fois le texte trouvé, celui-ci est recopié dans le pointeur *yytext* et sa longueur dans la variable *ylleng*. L'action correspondante à la pattern est alors exécutée, et le reste de la ligne analysé.

Si aucune règle n'est déclenchée (c'est-à-dire aucune occurrence de pattern n'est trouvée dans la phrase), la règle par défaut est exécutée : elle consiste à copier les caractères un à un vers la sortie.

Donc, un programme Flex réduit à `%%` copiera simplement le fichier d'entrée vers le fichier de sortie, sans aucune modification.

La notion d'Action

Chaque pattern dans une règle possède une action qui lui correspond. La pattern se terminant dès le premier caractère blanc rencontré, le reste de la ligne est donc l'action associée.

A titre d'exemple, ce petit programme remplace les suites de caractères blancs (en anglais, "white space") par un espace. Il supprime également tous les caractères blancs en fin de ligne.

```
%%
[ \t]+    putchar(' ');
[ \t]+$   /* on ignore simplement */
```

Lorsqu'une action ne tient pas sur une seule ligne il faut la mettre entre accolades. Il est à noter que Flex sait reconnaître les accolades liées à l'action des éventuelles accolades se trouvant dans le source C définissant l'action.

Il existe de nombreuses directives lex qui peuvent être intégrées dans les actions et qui représentent en quelque sorte des macro-commandes. Un exemple, *ECHO* recopie systématiquement le contenu du pointeur de caractères yytext vers la sortie du scanner.

Il existe bien d'autres choses à dire sur Flex, mais la place qui m'est impartie n'étant pas élastique, passons sans plus attendre aux exemples qui vous permettront de mieux cerner son utilisation.

EXEMPLES

```
int nbr_ligne = 0, nbr_car = 0;
%%
\n      ++nbr_ligne; ++nbr_car;
.       ++nbr_car;
%%
main()
{
    yylex();
    printf( "Nombre de lignes = %d, nombre de caractères = %d\n", nbr_ligne, nbr_car );
}
```

Ce premier exemple compte le nombre de caractères et le nombre de lignes d'un fichier d'entrée. Il ne produit aucun autre résultat que ces deux nombres en sortie. La première ligne déclare deux variables globales *nbr_ligne* et *nbr_car*, qui sont accessibles à la fois par la routine *yylex()* (générée par Flex) et par la fonction *main()* (définie par l'utilisateur).

Deux règles ont été définies : la première recherche les caractères retour chariot (\n) et incrémente les valeurs *nbr_ligne* et *nbr_car* à chaque fois qu'elle en découvre un ; la seconde recherche un caractère quelconque autre que \n et incrémente *nbr_car* à chaque fois qu'elle en découvre un.

Passons maintenant à un exemple un peu plus compliqué.

```
/* Analyseur syntaxique pour un langage style C */
%{
/* Cet include est nécessaire pour récupérer */
/* la définition de la fonction atof() */
#include <math.h>
%}
CHIFFRE [0-9]
VAR      [a-z][a-z0-9]*
%%
{CHIFFRE}+ {
    printf( "Un entier: %s (%d)\n", yytext,
            atoi( yytext ) );
}
```

```
{CHIFFRE}+."{CHIFFRE}* {
    printf( "Un flottant: %s (%g)\n", yytext,
            atof( yytext ) );
}

if|while|do|switch|case|for|break|continue{
    printf( "Un mot clef: %s\n", yytext );
}

{VAR}+"(" printf( "Une fonction: %s\n", yytext );
{VAR} printf( "Une variable: %s\n", yytext );

"+|-|'|\"|*|/|\" printf( "Un opérateur: %s\n", yytext );

[ \t\n]+ /* saute les blancs */

. printf( "Caractère inconnu: %s\n", yytext );

%%
main( argc, argv )
int argc;
char **argv;
{
    ++argv, --argc;
    if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
    else
        yyin = stdin;

    yylex();
}
```

Ce petit exemple pourrait représenter le début d'un programme d'analyse de langage : il est en effet à même de reconnaître dans un fichier source, les variables, fonctions, mot-clefs du langage, expressions ainsi que les constantes entières et réelles présents dans le source. Il ne traite pas toutes les spécificités du langage, mais montre de manière simple comment on peut envisager la réalisation d'un véritable analyseur syntaxique. Nous y reviendrons d'ailleurs dans un prochain article.

BIBLIOGRAPHIE

Advanced Tools for Programmers : tiré de la documentation Unix, ce livre traite de Lex et Yacc.
FlexDoc.doc : la documentation accompagnant le logiciel, par Vern Paxson.

OU TROUVER FLEX ?

Comme bon nombre d'utilitaires étudiés dans ces colonnes, Flex est disponible dans le domaine public et plus particulièrement sur la disquette *Fred Fish n°412*. Cette disquette contient outre l'exécutable, une documentation fort complète de 35 pages (en anglais !) sur le produit et les sources du logiciel, archivés au format LHarc.

J'espère que cet article vous donnera envie d'en savoir plus sur cet excellent produit qu'est Flex. Le mois prochain, nous continuerons notre exploration du monde Unix avec Yacc, qui permet de réaliser facilement des analyseurs sémantiques, et nous verrons bientôt que couplé à Flex, il nous peut permettre de réaliser notre propre interpréteur ou compilateur ! Mais n'anticipons pas, et à dans un mois pour un premier descriptif de Yacc.

Dura Flex sed Flex !

Par Herr Doktor Von GlutenStimmellmDorf.

DEMOS : DE TOUTES LES COULEURS

Vous vous souvenez du crétin qui pestait contre le manque d'originalité dans le monde des démos ? Hé bien ce même crétin va vous présenter l'exemple type de ce qu'il ne faut absolument pas faire car cela relève des bases de la programmation : un superbe cycle de couleurs.

Je fais cela pour un raison bien simple : moi aussi je manque d'imagination, donc moi aussi je clone bêtement ce que font les autres... La réalisation de cet effet, plus connu sous le nom de Plasma, maintes et maintes fois répété dans les démos, et qui consiste à faire un dessin très coloré, ligne par ligne, tout en faisant cycliser les couleurs, est des plus simples : tout se fait grâce à une fonction à deux variables du type $z=F(x,y)$, où x et y sont les coordonnées du point considéré et z , sa couleur.

Nous allons donc balader notre point sur tout l'écran et obtenir ainsi un beau dessin. La courbe du programme qui suit est quelconque et le jeu des démomakers est, d'après ce que j'en ai compris, de faire la plus belle courbe possible. Je vous laisse donc le soin de la modifier à loisir, jusqu'à ce que le résultat vous satisfasse pleinement.

Vous remarquerez aussi que le dessin se fait à une lenteur extrême. Ceci est dû à deux choses : d'une part, il y a beaucoup de points (plus précisément, $640 \times 256 = 163.840$) et d'autre part, pour chaque point, on doit calculer sa couleur puis le tracer. Bien que l'opération soit relativement rapide si elle est utilisée ponctuellement, elle l'est bien moins si on la répète 163.840 fois. De plus, l'écran sur lequel on trace est en haute résolution 16 couleurs, ce qui signifie que l'Amiga prend environ 90% du temps-machine rien que pour envoyer les données des bitplans au moniteur. Donc, je me retrouve avec l'équivalent d'un 68000 à 0,714 MHz pour calculer mes 163.840 (je le replace une fois de plus car les gros chiffres, c'est toujours impressionnant) points.

Et voici la fin de cet article, d'un haut niveau technique. Si le manque d'inspiration continue, je crains le pire pour le mois prochain... Tous cela ne m'empêche absolument pas de vous donner la phrase du mois : "Lorsque notre fierté est en cause, notre mémoire préfère souvent céder".

```

      opt      c-
*****
Execbase      =      4
custom        =      $dff000
Forbid        =      -132
Permit        =      -138
Disable       =      -120
Enable        =      -126
openlibrary   =      -552
StartList     =      38
copllc       =      $80
copjmp1       =      $88
dmacon        =      $96
vposr         =      $04
KeyBoard      =      $bfec01
*****
BPL_X         =      640
BPL_Y         =      256
```

```

BPL_DEPTH     =      4
BPL_WIDTH     =      BPL_X/8
BPL_SIZE      =      BPL_WIDTH*BPL_Y

NB_COLOR      =      16
```

```

*****
VSYNC: macro   * la macro habituelle pour le vsync
.LOOP_VSYNC@:
    move.l     vposr(a6),d0
    and.l      #$1ff00,d0
    cmp.l      #\1*$100,d0
    bne.s      .LOOP_VSYNC@
endm
```

```

*****
Debut: bsr     CLR_SCREEN
      bsr     MAIN_INIT

      lea     $dff000,a6
      clr.w   COOR_Y          *\ init les parametres de
      clr.w   COOR_X          * > la fonction
      clr.l   CURRENT_ADR_Y   */

      bsr     Cycle_Courbe_Couleur

Vbl_Sync:
      bsr     Cycle_Courbe_Couleur

      cmp.w   #BPL_Y,COOR_Y
      bge.s   SCREEN_FINISH
      bsr     CALCUL_FONCTION      * 1 Ligne a chaque fois.
      clr.w   TIME_BETWEEN_2_CYCLE
      bra     SCREEN_NOT_FINISH   *\ Ne fais pas le vsync
SCREEN_FINISH                                */ si l'écran n'est pas
      vsync   $10                                */ fini
SCREEN_NOT_FINISH

      btst    #6,$bfe001      *\teste le bouton de la souris
      bne.s   Vbl_Sync        */ pour quitter le programme

      bsr     MAIN_END
      moveq   #0,d0
      rts
```

```

*****
CALCUL_FONCTION:
      move.l   #SCREEN,d3          *\d3= adr de la ligne a
      add.l    CURRENT_ADR_Y,d3    */ tracer

      lea     Sin_Tab(pc),a1      * a1 = table sinus
      lea     $100(a1),a2        * a2 = table cos
      clr.w   COOR_X

      move.l   #BPL_WIDTH-1,d7    * d7 = nb d'octet

LOOP_CALCUL_FONCTION_EACH_BYTE
      moveq    #8-1,d6            * d6 = nb de bit

LOOP_CALCUL_FONCTION_EACH_BIT
      moveq    #0,d0
      moveq    #0,d1
      move.w   COOR_X(pc),d0
      move.w   COOR_Y(pc),d1

      * calcul des differents parametres de la fonction
      move.w   d0,d2              * d2= angle du futur cosinus
      lsl.w    #2,d2              *\x2 pour obtenir le mot du cos
                                  */et par 2 pour un beau dessin.
      and.w    #$3fe,d2* 1'angle doit etre entre 0 et 512
      move.w   (a2,d2.w),d2      * d2= le cosinus
      muls     #40,d2            * on le multiplie par 40
      lsl.l    #1,d2            *\ savante manipulation avoir
      clr.w    d2                * > pour la veritable valeur
      swap     d2                */ de d2

      * les calculs ci-dessous sont sur le meme principe que ci-dessus
      move.w   d2,d4
```



```

move.w d1,d2
sub.w d4,d2
lsl.w #3,d2
and.w #$3fe,d2
move.w (a2,d2.w),d2
muls #32,d2
lsl.l #1,d2
clr.w d2
swap d2
move.w d2,d4

* les calculs ci-dessous sont sur le meme principe que ci-dessus
move.w d1,d2
lsl.w #2,d2
and.w #$3fe,d2
move.w (a1,d2.w),d2
muls #40,d2
lsl.l #1,d2
clr.w d2
swap d2

* les calculs ci-dessous sont sur le meme principe que ci-dessus
sub.w d2,d0
move.w d0,d2
lsl.w #3,d2
and.w #$3fe,d2
move.w (a1,d2.w),d2
muls #64,d2
lsl.l #1,d2
clr.w d2
swap d2
add.w d4,d2

* arrive ici on a la couleur dans d2.
* On ne recalcule pas a chaque fois l'adresse du point pour *
gagner du temps d'execution.
move.w COOR_X,d0      * \
and.w #$0007,d0      * >d0= du bit ou se trouve le
point
neg.w d0              * /
addq.w #7,d0          */
and.w #NB_COLOR-1,d2 * \ -1 pour que la couleur reste
or.w #1,d2            */ entre 1 et NB_COLOR

move.l d3,a0          *\a0= adr de l'octet ou se trouve
                      */ le point actuel

* trace le point dont la couleur est dans d2
moveq #BPL_DEPTH-1,d4
EACH_BPL:
lsl.w #1,d2
bcc.s CLR_BIT_IN_THIS_BPL
bset d0,(a0)
add.l #BPL_SIZE,a0
dbf d4,EACH_BPL
bra.s POINT_FINISH
CLR_BIT_IN_THIS_BPL
bclr d0,(a0)
add.l #BPL_SIZE,a0
dbf d4,EACH_BPL
POINT_FINISH

* continue de tracer des points dans l'octet courant
* et si ils sont tous faits on passe apres.
addq.w #1,COOR_X
dbf d6,LOOP_CALCUL_FONCTION_EACH_BIT

* continue de tracer dans les octets
* et si ils sont tous fait on s'en va.
addq.l #1,d3
dbf d7,LOOP_CALCUL_FONCTION_EACH_BYTE

* on incremente les parametres pour faire
* la ligne suivante la prochaine fois
add.l #BPL_WIDTH,CURRENT_ADR_Y
addq.w #1,COOR_Y
rts

COOR_X:      ds.w 1

```

```

COOR_Y:      ds.w 1
CURRENT_ADR_Y: ds.l 1

*****
* Cette routine s'occupe du cycle couleur
Cycle_Courbe_Couleur:
subq.w #1,TIME_BETWEEN_2_CYCLE
tst.w TIME_BETWEEN_2_CYCLE
bgt Fin_Cycle_Courbe_Couleur

lea ADR_CMAP_IN_CLIST,a0
addq.l #2,a0      * a0 --> registre couleur.

lea TAB_DEGRADE_COULEUR(pc),a1
move.w CURRENT_POS_TAB_DEGRADE_COULEUR,d0
moveq #NB_COLOR-1,d2

LOOP_Cycle_Courbe_Couleur:
and.w #NB_COLOR-1,d0
move.w d0,d1
add.w d1,d1
move.w (a1,d1.w),(a0)
addq.l #4,a0
addq.w #1,d0

dbf d2,LOOP_Cycle_Courbe_Couleur

addq.w #1,CURRENT_POS_TAB_DEGRADE_COULEUR
move.w #5,TIME_BETWEEN_2_CYCLE
Fin_Cycle_Courbe_Couleur:
rts

TIME_BETWEEN_2_CYCLE:      ds.w 1
CURRENT_POS_TAB_DEGRADE_COULEUR: ds.w 1
TAB_DEGRADE_COULEUR:
dc.w $0f00,$0d02,$0b04,$0906
dc.w $0708,$050a,$030c,$010e,$f
dc.w $010e,$030c,$050a,$0708
dc.w $0906,$0b04,$0d02

*****
* Cette routine efface les bitplans
CLR_SCREEN:
move.w #BPL_WIDTH*BPL_Y*BPL_DEPTH/4-1,d0

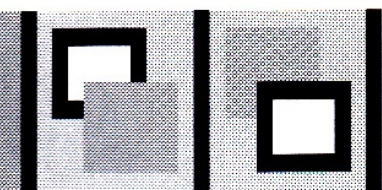
lea SCREEN,a0
LOOP_CLR_SCREEN:
clr.l (a0)+
dbf d0,LOOP_CLR_SCREEN
rts

*****
* Cette routine initialise les bitplans dans la copperlist
INIT_CLIST:
lea ADR_BPL_IN_CLIST,a0
move.l #SCREEN,d0

moveq #BPL_DEPTH-1,d1
.LOOP_INIT_BPL_IN_CLIST
swap d0
move.w d0,(a0)
addq.l #4,a0
swap d0
move.w d0,(a0)
addq.l #4,a0
add.l #BPL_SIZE,d0
dbf d1,.LOOP_INIT_BPL_IN_CLIST
rts

*****
MAIN_INIT:
move.l (excbase).w,a6
lea custom,a5
jsr Forbid(a6)
jsr Disable(a6)
move.w #$03e0,dmacon(a5)
bsr INIT_CLIST
move.l #CLIST,cop11c(a5)

```

```
clr.w    copjmp1(a5)
move.w   #$8380,dmacon(a5)
lea      custom,a6
rts

*****
MAIN_END:
move.l   (execbase).w,a6
lea      custom,a5
move.l   #GfxName,a1          *\
moveq    #0,d0                * >-remet l'ancienne
jsr      OpenLibrary(a6)      * > copper list
move.l   d0,a4                * >
move.l   StartList(a4),copllc(a5)
move.w   d0,copjmp1(a5)
move.w   #$83e0,dmacon(a5)
jsr      Enable(a6)
jsr      Permit(a6)
lea      custom,a6
rts
GfxName dc.b    "graphics.library",0
even

*****
* avis aux fainéants :
* a remplacer par un include avec le programme
* pour init les sinus en gfa paru precedemment.
sin_tab dc.w    $0000,$0192,$0324,$04B6,$0647,$07D9,$096A,$0AFB
dc.w     $0C8B,$0E1B,$0FAB,$1139,$12C8,$1455,$15E2,$176D
dc.w     $18F8,$1A82,$1C0B,$1D93,$1F19,$209F,$2223,$23A6
dc.w     $2528,$26A8,$2826,$29A3,$2B1F,$2C98,$2E11,$2F87
dc.w     $30FB,$326E,$33DE,$354D,$36BA,$3824,$398C,$3AF2
dc.w     $3C56,$3DB8,$3F17,$4073,$41CE,$4325,$447A,$45CD
dc.w     $471C,$4869,$49B4,$4AFB,$4C3F,$4D81,$4EBF,$4FFB
dc.w     $5133,$5269,$539B,$54CA,$55F5,$571D,$5842,$5964
dc.w     $5A82,$5B9D,$5CB4,$5DC7,$5ED7,$5FE3,$60EC,$61F1
dc.w     $62F2,$63EF,$64E8,$65DD,$66CF,$67BD,$68A6,$698C
dc.w     $6A6D,$6B4A,$6C24,$6CF9,$6DCA,$6E96,$6F5F,$7023
dc.w     $70E2,$719E,$7255,$7307,$73B5,$745F,$7504,$75A5
dc.w     $7641,$76D9,$776C,$77FA,$7884,$7909,$798A,$7A05
dc.w     $7A7D,$7AEF,$7B5D,$7BC5,$7C29,$7C89,$7CE3,$7D39
dc.w     $7D8A,$7DD6,$7E1D,$7E5F,$7E9D,$7ED5,$7F09,$7F38
dc.w     $7F62,$7F87,$7FA7,$7FC2,$7FD8,$7FE9,$7FF6,$7FFD
dc.w     $7fff,$7FFD,$7FF6,$7FE9,$7FD8,$7FC2,$7FA7,$7F87
dc.w     $7F62,$7F38,$7F09,$7ED5,$7E9D,$7E5F,$7E1D,$7DD6
dc.w     $7D8A,$7D39,$7CE3,$7C89,$7C29,$7BC5,$7B5D,$7AEF
dc.w     $7A7D,$7A05,$798A,$7909,$7884,$77FA,$776C,$76D9
dc.w     $7641,$75A5,$7504,$745F,$73B5,$7307,$7255,$719E
dc.w     $70E2,$7023,$6F5F,$6E96,$6DCA,$6CF9,$6C24,$6B4A
dc.w     $6A6D,$698C,$68A6,$67BD,$66CF,$65DD,$64E8,$63EF
dc.w     $62F2,$61F1,$60EC,$5FE3,$5ED7,$5DC7,$5CB4,$5B9D
dc.w     $5A82,$5964,$5842,$571D,$55F5,$54CA,$539B,$5269
dc.w     $5133,$4FFB,$4EBF,$4D81,$4C3F,$4AFB,$49B4,$4869
dc.w     $471C,$45CD,$447A,$4325,$41CE,$4073,$3F17,$3DB8
dc.w     $3C56,$3AF2,$398C,$3824,$36BA,$354D,$33DE,$326E
dc.w     $30FB,$2F87,$2E11,$2C98,$2B1F,$29A3,$2826,$26A8
dc.w     $2528,$23A6,$2223,$209F,$1F19,$1D93,$1C0B,$1A82
dc.w     $18F8,$176D,$15E2,$1455,$12C8,$1139,$0FAB,$0E1B
dc.w     $0C8B,$0AFB,$096A,$07D9,$0647,$04B6,$0324,$0192
dc.w     $0000,$FE6E,$FCDC,$FB4A,$F9B9,$F827,$F696,$F505
dc.w     $F375,$F1E5,$F055,$EEC7,$ED38,$EBAB,$EA1E,$E893
dc.w     $E708,$E57E,$E3F5,$E26D,$E0E7,$DFF6,$DDDD,$DC5A
dc.w     $DAD8,$D958,$D7DA,$D65D,$D4E1,$D368,$D1EF,$D079
dc.w     $CF05,$CD92,$CC22,$CAB3,$C946,$C7DC,$C674,$C50E
dc.w     $C3AA,$C248,$C0E9,$BF8D,$BE32,$BCDB,$BB86,$BA33
dc.w     $B8E4,$B797,$B64C,$B505,$B3C1,$B27F,$B141,$B005
dc.w     $AECF,$AD97,$AC65,$AB36,$AA0B,$A8E3,$A7BE,$A69C
dc.w     $A57E,$A463,$A34C,$A239,$A129,$A01D,$9F14,$9E0F
dc.w     $9D0E,$9C11,$9B18,$9A23,$9931,$9843,$975A,$9674
dc.w     $9593,$94B6,$93DC,$9307,$9236,$916A,$90A1,$8FDD
dc.w     $8F1E,$8E62,$8DAB,$8CF9,$8C4B,$8BA1,$8AFC,$8A5B
dc.w     $89BF,$8927,$8894,$8806,$877C,$86F7,$8676,$85FB
dc.w     $8583,$8511,$84A3,$843B,$83D7,$8377,$831D,$82C7
dc.w     $8276,$822A,$81E3,$81A1,$8163,$812B,$80F7,$80C8
dc.w     $809E,$8079,$8059,$803E,$8028,$8017,$800A,$8003
dc.w     $8000,$8003,$800A,$8017,$8028,$803E,$8059,$8079
dc.w     $809E,$80C8,$80F7,$812B,$8163,$81A1,$81E3,$822A
dc.w     $8276,$82C7,$831D,$8377,$83D7,$843B,$84A3,$8511
```

```
dc.w     $8583,$85FB,$8676,$86F7,$877C,$8806,$8894,$8927
dc.w     $89BF,$8A5B,$8AFC,$8BA1,$8C4B,$8CF9,$8DAB,$8E62
dc.w     $8F1E,$8FDD,$90A1,$916A,$9236,$9307,$93DC,$94B6
dc.w     $9593,$9674,$975A,$9843,$9931,$9A23,$9B18,$9C11
dc.w     $9D0E,$9E0F,$9F14,$A01D,$A129,$A239,$A34C,$A463
dc.w     $A57E,$A69C,$A7BE,$A8E3,$AA0B,$AB36,$AC65,$AD97
dc.w     $AECF,$B005,$B141,$B27F,$B3C1,$B505,$B64C,$B797
dc.w     $B8E4,$BA33,$BB86,$BCDB,$BE32,$BF8D,$C0E9,$C248
dc.w     $C3AA,$C50E,$C674,$C7DC,$C946,$CAB3,$CC22,$CD92
dc.w     $CF05,$D079,$D1EF,$D368,$D4E1,$D65D,$D7DA,$D958
dc.w     $DAD8,$DC5A,$DDDD,$DF61,$E0E7,$E26D,$E3F5,$E57E
dc.w     $E708,$E893,$EA1E,$EBAB,$ED38,$EEC7,$F055,$F1E5
dc.w     $F375,$F505,$F696,$F827,$F9B9,$FB4A,$FCDC,$FE6E
dc.w     $0000,$0192,$0324,$04B6,$0647,$07D9,$096A,$0AFB
dc.w     $0C8B,$0E1B,$0FAB,$1139,$12C8,$1455,$15E2,$176D
dc.w     $18F8,$1A82,$1C0B,$1D93,$1F19,$209F,$2223,$23A6
dc.w     $2528,$26A8,$2826,$29A3,$2B1F,$2C98,$2E11,$2F87
dc.w     $30FB,$326E,$33DE,$354D,$36BA,$3824,$398C,$3AF2
dc.w     $3C56,$3DB8,$3F17,$4073,$41CE,$4325,$447A,$45CD
dc.w     $471C,$4869,$49B4,$4AFB,$4C3F,$4D81,$4EBF,$4FFB
dc.w     $5133,$5269,$539B,$54CA,$55F5,$571D,$5842,$5964
dc.w     $5A82,$5B9D,$5CB4,$5DC7,$5ED7,$5FE3,$60EC,$61F1
dc.w     $62F2,$63EF,$64E8,$65DD,$66CF,$67BD,$68A6,$698C
dc.w     $6A6D,$6B4A,$6C24,$6CF9,$6DCA,$6E96,$6F5F,$7023
dc.w     $70E2,$719E,$7255,$7307,$73B5,$745F,$7504,$75A5
dc.w     $7641,$76D9,$776C,$77FA,$7884,$7909,$798A,$7A05
dc.w     $7A7D,$7AEF,$7B5D,$7BC5,$7C29,$7C89,$7CE3,$7D39
dc.w     $7D8A,$7DD6,$7E1D,$7E5F,$7E9D,$7ED5,$7F09,$7F38
dc.w     $7F62,$7F87,$7FA7,$7FC2,$7FD8,$7FE9,$7FF6,$7FFD
```

```
*****
* pour que la copper list et les bitplans soit en chip ram.
section copper_list,DATA_C
```

```
CLIST: dc.w     $008E,$2b81          *\
dc.w     $0090,$2bc1                * >init un ecran de
640*256
dc.w     $0092,$0038                * >
dc.w     $0094,$00d0                */
dc.w     $0108,$0000,$010A,$0000
dc.w     $0100,$0200,$0102,$0000,$0104,$0000
```

```
ADR_CMAP_IN_CLIST:
dc.w     $0180,$0000,$0182,$0000,$0184,$0000,$0186,$0000
dc.w     $0188,$0000,$018a,$0000,$018c,$0000,$018e,$0000
dc.w     $0190,$0000,$0192,$0000,$0194,$0000,$0196,$0000
dc.w     $0198,$0000,$019a,$0000,$019c,$0000,$019e,$0000
```

```
dc.w     $0180,$0005
dc.w     $1b0f,$fffe,$0100,(8+BPL_DEPTH)*$1000+$200
dc.w     $00E0
```

```
ADR_BPL_IN_CLIST:
```

```
dc.w     $0
dc.w     $00E2,$0
dc.w     $00E4
dc.w     $0
dc.w     $00E6,$0
dc.w     $00E8
dc.w     $0
dc.w     $00EA,$0
dc.w     $00EC
dc.w     $0
dc.w     $00EE,$0
dc.w     $00F0
dc.w     $0
dc.w     $00F2,$0
```

```
dc.l     $fffffffe                * montre la fin de la list
```

```
*****
section screen,BSS_C
```

```
SCREEN: ds.l    BPL_WIDTH*BPL_Y*BPL_DEPTH/4
```

```
END
```

Par Jérôme Etienne

Bienvenue dans cette nouvelle rubrique ! Je vais vous proposer ici quelques routines utilitaires en assembleur qui n'auront d'autre but que de vous faciliter la vie.

Pour commencer, voici un moyen vraiment légal (j'insiste !) pour empêcher les mouvements de la souris de parvenir au Workbench, lorsque l'on monopolise l'affichage. Allons bon, voilà que je me coupe mes effets, alors que j'avais préparé une belle introduction. Bon, c'est pas grave, la voici quand même.

LA BELLE INTRO

Que c'est bien de modifier les listes Copper du système... Comme l'on peut faire de jolies choses dans son propre affichage... On bouge la souris, on clique ici et là, on tape de belles commandes au clavier... Tout semble se passer pour le mieux. Puis on sort de son beau programme, et on constate, ô divine surprise, que toutes les actions de la souris on été enregistrées dans le Workbench ! En sélectionnant telle icône de son programme, on a fait démarrer un utilitaire dans la fenêtre Workbench. Un peu (NDLR : beaucoup) de malchance et on formate son disque dur !

Pourtant, il existe des moyens tout-à-fait légaux pour empêcher les mouvements de la souris et les touches du clavier d'arriver au Workbench. Des moyens vraiment légaux, j'entends, sans toucher aux CIAs, sans poker dans les vecteurs d'interruption, sans même interdire le multitâche.

Le nom magique est "input.device". Sous ce nom est regroupé toute la gestion des entrées de l'Amiga : souris, clavier, mais aussi joystick, disquette insérée ou enlevée, etc.

La routine d'aujourd'hui se place au beau milieu de l'input.device et joue le rôle de filtre : elles ne laisse pas passer la souris ou le clavier. A part ça, l'Amiga fonctionne tout-à-fait normalement : vous pouvez même faire des accès disques, lancer des tâches secondaires...

LE BEAU PROGRAMME

Comme toutes mes futures routines, celle d'aujourd'hui est totalement relogable, ce qui vous permettra par exemple de l'insérer dans un programme Basic (AMOS bien sûr, mais aussi Amiga et même GFA !). L'assemblage a été réalisé avec le Devpac.

La routine offre deux points d'entrée :

- **NEv_Start** : à appeler en début de programme, cette routine ouvre l'input.device et intercepte les événements clavier et souris.
- **NEv_Stop** : à appeler avant de sortir du programme, pour remettre les choses en place.

Attention, avant de faire un accès disque, il est sage de rebrancher le clavier et la souris : n'oubliez pas qu'un requester de l'AmigaDOS peut apparaître. Dans ce cas, il vaut mieux pouvoir bouger la souris...

Voici un tout petit exemple d'utilisation de cette première routine, dans lequel on se contente de modifier très rapidement la couleur zéro en testant le bouton gauche de la souris directement dans les circuits, en toute quiétude.

LA BELLE CONCLUSION

Bon, voilà pour aujourd'hui. Je ne peux pas vous dire ce que je vais vous trouver pour le mois prochain vu que je ne le sais pas et que je ne le saurai que la veille d'envoyer mon article à Stéphane... C'est ça l'aventure, la vraie.

*-----
* ROUTINETTE numéro 1
* A'pu souris, a'pu clavier. In English: No_Events
* Par François Lionet. Paru dans l'A.N.T. du 32 Février 1789
*-----

```
io_command      equ      $1c
io_data         equ      $28
Lof_io          equ      $30
AddHandler      equ      9
RemHandler      equ      10
Lof_msg         equ      $20
ln_type         equ      $08
ln_pri          equ      $09
ln_name         equ      $0a
is_data         equ      $0e
is_code         equ      $12
Lof_is          equ      $16
DoIo            equ      -456
```

```
FindTask        equ      -294
OpenDevice      equ      -444
CloseDevice     equ      -450
AddPort         equ      -354
RemPort         equ      -360
Ie_NextEvent    equ      0
Ie_Class        equ      4
Ie_SubClass     equ      5
Ie_Code         equ      6
Ie_Qualifier    equ      8
Ie_EventAddress equ      10
Ie_X            equ      10
Ie_Y            equ      12
IECLASS_RAWKEY  EQU      $01
IECLASS_RAWMOUSE EQU      $02
*----- Pitit exemple...
```

```
Exemple:
    bsr      NEv_Start      ; Enlève le clavier et la souris
; Boucle de test
.loop    move.w    d0,$dff180      ; Changement de couleur
        addq.w    #1,d0
        btst     #6,$BFE001      ; Test du bouton souris
        bne.s     .loop
        bsr      NEv_Stop      ; On remet tout!
        moveq    #0,d0          ; Basta, finito !
        rts
```

*----- Point d'entrée 1: démarrage du filtre

```
NEv_Start:
    movem.l    d0-d7/a0-a6,-(sp)
    move.l     $4.w,a6
    lea        NEv_msg(pc),a2      ; Adresse des structures
    lea        NEv_io(pc),a3
    lea        NEv_int(pc),a4
    sub.l      a1,a1                ; Fabrication du port
    jsr        FindTask(a6)
    move.l     a2,a1
    move.l     d0,$10(a1)
    jsr        AddPort(a6)
    lea        NEv_name(pc),a0      ; Ouverture de l'input.device
    move.l     a3,a1
    moveq      #0,d0
    moveq      #0,d1
    jsr        OpenDevice(a6)
    move.l     a2,$14(a3)          ; Associe le port et le device
; Branchement de la routine d'interruption
    lea        NEv_Handler(pc),a0
    move.l     a0,is_code(a4)
    clr.l      is_data(a4)
    move.b     #100,ln_pri(a4)
    move.l     a3,a1
    move.l     a4,io_data(a1)
    move.w     #AddHandler,io_command(a1)
    jsr        DoIo(a6)
    movem.l    (sp)+,d0-d7/a0-a6    ; Voilà, c'est tout.
    rts
```

*----- Point d'entrée 2: arrêt du filtre.

```
NEv_Stop:
    movem.l    d0-d7/a0-a6,-(sp)
    move.l     $4.w,a6
    lea        NEv_msg(pc),a2      ; Adresse des structures
    lea        NEv_io(pc),a3
    lea        NEv_int(pc),a4
    move.l     a3,a1                ; Enleve l'interruption
    move.l     a4,io_data(a1)
    move.w     #RemHandler,io_command(a1)
    move.l     $4.w,a6
    jsr        DoIo(a6)
    move.l     $4.w,a6              ; Ferme l'input.device
    jsr        CloseDevice(a6)
    move.l     a2,a1                ; Ferme le port
    jsr        RemPort(a6)
    movem.l    (sp)+,d0-d7/a0-a6
    rts
```

*----- Routine de filtre en interruption

```
NEv_Handler:
    move.l     a0,d0
    move.l     a0,d2
    moveq      #0,d3
    .loop      move.b     Ie_Class(a0),d1      ; Boucle d'exploration
                cmp.b     #IECLASS_RAWMOUSE,d1 ; Mouvement souris?
                beq.s     .trash              ; Oui! A la poubelle!
                cmp.b     #IECLASS_RAWKEY,d1  ; Touche clavier?
                beq.s     .trash              ; Oui! A la poubelle!
                move.l     d2,d3              ; On garde l'évènement,
                move.l     (a0),d2
    .next      move.l     d2,a0                ; Un autre à voir?
                bne.s     .loop
                rts                          ; Non! C'est fini.
; Balance l'évènement à la poubelle.
    .trash     tst.l      d3                  ; Un évènement avant?
                beq.s     .skip
                move.l     d3,a1
                move.l     (a0),d2
                move.l     d2,(a1)
                bra.s     .next
    .skip      move.l     (a0),d0              ; Va voir les suivants..
                move.l     d0,a0              ; Pas d'évènement avant.
                bne.s     .loop              ; On peut voir le suivant.
                rts                          ; Y'en a plus: on sort!
```

*----- Toute petite tchote zone de data.

```
NEv_io          ds.b     Lof_io      ; Structure io
NEv_msg         ds.b     Lof_msg     ; Structure message-port
NEv_int         ds.b     Lof_is      ; Structure interrupt
NEv_name        dc.b     "input.device",0
even
```

END

Par François Lionet

UTILITAIRE : LE TRANSFORMEUR

Qui n'a pas rêvé de se faire de beaux gadgets ou de superbes images agrémentant la présentation d'un soft que l'on sait en son fort intérieur être le précurseur d'une nouvelle génération de logiciels ?

Ne désespérez plus, voici l'outil ultime, j'ai nommé le Transformeur. La démarche est la suivante : vous prenez votre logiciel de DAO préféré, vous faites le dessin de vos rêves, puis, grâce ce programme, vous transformez l'image (ou la brosse) en une suite de nombres au format ASCII, ce qui constitue (comme chacun sait) un fichier source que vous pouvez insérer ou linker... Ah que ouais !

IMAGES IFF

Il est nécessaire de sauvegarder votre dessin au format IFF, ce que font les 9/10èmes des utilitaires de dessin (sinon, prenez-en un autre !). Les images IFF sont codées sous une forme standard qui permet le chargement par d'autres logiciels (mon mari).

Pour mieux saisir le fonctionnement, un bref rappel du format IFF n'est sans doute pas inutile ; suivez sur le listing, à la fonction 'analyse()'.

Chaque image IFF comporte une suite de 'chunks' dont l'ordre n'est pas imposé. Un chunk peut être représenté par une structure C :

```
struct Chunk
{
    ULONG ckID;
    LONG ckSize;
    /* UBYTE ckData[ckSize]; */
};
```

où ckID est un mot long servant d'identificateur au chunk et dont la valeur est exprimée sous forme ASCII (FORM, BMHD, CMAP, BODY, etc...), où ckSize représente la taille du chunk en octets et où ckData (qui suit directement le chunk) contient les octets de données (de longueur variable, donc).

Si le fichier IFF est uniquement constitué d'une image, son premier chunk doit avoir pour identificateur 'FORM'. Les autres identificateurs possibles (CAT, PROP, LIST) sont donc rejetés, car ils indiquent un mélange de divers types, par exemple texte et dessin.

Les données de ce premier chunk sont elles-mêmes formées d'autres chunks. Si le fichier étudié est une image, le second identificateur qui se présente doit être de type ILBM. Les autres identificateurs (FTXT, SMUS, 8SVX) sont rejetés car ils indiquent respectivement un fichier texte, musique ou instrument.

On arrive alors aux chunks utilisables : BMHD, BODY, CMAP (il y en a d'autres, mais seuls ces trois là nous intéressent ici) :

- BMHD contient les caractéristiques de l'image : largeur, hauteur, nombre de bitplanes, compression, etc. (voir 'case ID_BMHD' dans le listing).

- BODY contient les bitplanes eux-mêmes, la valeur de chaque bit déterminant la couleur du pixel

correspondant (voir 'case ID_BODY'). Quand ce chunk a été trouvé, il importe de décompacter les données si c'est nécessaire, c'est ce que fait la fonction 'transfert()'.

- CMAP contient les couleurs de l'image, à noter pour pouvoir restituer l'image dans sa pureté primitive... Là encore, voyez la partie 'case ID_CMAP'.

Ces 3 chunks peuvent être trouvés dans n'importe quel ordre, ce qui explique l'étude du fichier sur toute sa longueur et un test final pour savoir si les données nécessaires ont été trouvées. Il ne reste plus alors qu'à sortir les valeurs des bitplanes en fichier source.

COMMENT CA MARCHE ?

Le Transformeur vous demande le type de source que vous désirez créer : 'C' si vous programmez en C et 'A' si vous programmez en assembleur (logique !). Il vous demande alors le nom du fichier à charger : image ou brosse. Il travaille quelque temps, affiche les couleurs de l'image en hexadécimal, vous demande le nom du source qu'il va créer et ça baigne.

Ah, j'oubliais : ce programme utilise l'arp.library, qui doit figurer dans le tiroir LIBS: de votre disquette système (NDLR : l'arp.library est fournie sur la disquette de ce numéro).

Les sources produits en C sont de type "USHORT data[]". Les bitplanes sont séparés par une remarque qui rappelle le nombre de lignes et de colonnes. Un programme d'exemple est fourni sur la manière d'utiliser ces sources.

Les sources produits en assembleur sont aussi en hexadécimal, de type "dc.w" avec une séparation des bitplanes comme ci-dessus. Le programme d'exemple pourra être adapté de celui qui est fourni en C.

```
/* *****
 * Convertit une image IFF en SOURCE ASCII C ou ASM *
 *
 * par D.GENOT sous LATTICE SAS v5.10 :
 * options de compil : -L
 * *****
 */

#include<stdio.h>
#include<string.h>
#include<exec/types.h>
#include<exec/memory.h>
#include<libraries/dos.h>
#include<libraries/dosextns.h>
#include<intuition/intuition.h>
#include<graphics/gfx.h>
#include<graphics/view.h>

#pragma libcall ArpBase FileRequest 126 801

/* ***** déclarations ***** */
LONG FileRequest();
LONG input(UBYTE *);
void output_C();
void output_A();

struct FileLock *Lock(), *clef=NULL;
struct FileInfoBlock bloc;
short Examine(), ExNext();
void UnLock(), cleanexit(), load_fichier(), analyse();
void lib_mem(), transfert(UBYTE *);

/* ***** variables ***** */
short res;
ULONG filesize, planes_long, plane1;
UBYTE fdir, prof;
```



```

UWORD haut;

char categ=' ';
LONG resul;
int dummy=0, hdlc, ncoul;
ULONG nom, mode, taille;
int vu, long_forme;
SHORT bpr; /* Bytes Per Row = par rangée */

APTR *filemem=NULL;
UBYTE *pteurb=NULL, *planepttr=NULL;
ULONG *pteurl=NULL;

char fname[80], dirname[120], nom_complet[200];

APTR ArpBase=NULL;

/***** definitions diverses utilisées pour le maniement des
 * images IFF (simples...)
 *****/

#define MAKEID(a,b,c,d) ((a<<24)|(b<<16)|(c<<8)|d)

#define ID_CAT MAKEID('C', 'A', 'T', ' ')
#define ID_FTXT MAKEID('F', 'T', 'X', 'T')
#define ID_PROP MAKEID('P', 'R', 'O', 'P')
#define ID_8SVX MAKEID('8', 'S', 'V', 'X')
#define ID_SMUS MAKEID('S', 'M', 'U', 'S')
#define ID_LIST MAKEID('L', 'I', 'S', 'T')
#define ID_FORM MAKEID('F', 'O', 'R', 'M')
#define ID_ILBM MAKEID('I', 'L', 'B', 'M')
#define ID_BMHD MAKEID('B', 'M', 'H', 'D')
#define ID_CMAP MAKEID('C', 'M', 'A', 'P')
#define ID_GRAB MAKEID('G', 'R', 'A', 'B')
#define ID_DEST MAKEID('D', 'E', 'S', 'T')
#define ID_CAMG MAKEID('C', 'A', 'M', 'G')
#define ID_BODY MAKEID('B', 'O', 'D', 'Y')

/***** structures *****/

struct Chunk
{
    LONG ckID;
    LONG ckSize;
};

struct Chunk *chk=NULL;

#define isodd(a) ((a)&1) /* --> 1 si impair */
#define wa(size) ((size)+isodd(size)) /* complète au mot près */
#define chsize(datasize) (wa(datasize) + 8) /* taille du chunk */

struct FileRequester
{
    UBYTE *fr_titre; /* texte fenetre */
    UBYTE *fr_file; /* filename */
    UBYTE *fr_dir; /* dirname */
    struct Window *fr_window; /* Win voulue ou NULL */
    UBYTE fr_flags; /* flags */
    UBYTE fr_reserved1; /* mettre à 0 */
    APTR fr_fonction; /* UserFunction */
    LONG fr_reserved2; /* occupe ! */
} Freq =
{" Mon Requester ", fname, dirname, NULL, NULL, NULL, NULL, NULL};

struct BMHD
{
    UWORD bmhd_w;
    UWORD bmhd_h; /* largeur et hauteur */
    UWORD bmhd_x;
    UWORD bmhd_y; /* position de l'image */
    UBYTE bmhd_nplanes;
    UBYTE bmhd_masking;
    UBYTE bmhd_compress;
    UBYTE bmhd_pad1;
    UWORD bmhd_transpcolor;
    UBYTE bmhd_xaspect;
    UBYTE bmhd_yaspect;
    UWORD bmhd_pagew;
    UWORD bmhd_pageh;
} *bmhd=NULL;

/***** ROUTINES *****/
LONG input(UBYTE *txt)
{
    register LONG reponse;

    Freq.fr_titre=txt;

    ArpBase=(APTR)OpenLibrary("arp.library",0);

```

```

    if (ArpBase==NULL)
    {
        printf("Il me faut la 'ARP.LIBRARY' ...");
        exit(FALSE);
    }

    reponse=FileRequest(&Freq);

    CloseLibrary(ArpBase);

    if(reponse)
        strmfp(nom_complet,dirname,fname);

    return(reponse);
}

void cleanexit()
{
    if(clef) UnLock(clef);
    if(hdlc) Close(hdlc);
    if(planepttr) FreeMem(planepttr,planes_long);
    if(filemem) FreeMem(filemem,filesize);
    exit(RETURN_OK);
}

void load_fichier()
{
    filemem=(APTR *)AllocMem(filesize, MEMF_CLEAR|MEMF_PUBLIC);
    if(filemem==NULL)
    {
        printf("\n Impossible de réserver la mémoire ...\n");
        cleanexit();
    }

    hdlc=Open(nom_complet,MODE_OLDFILE); /* lecture */
    if(hdlc==NULL)
    {
        printf("\n Impossible d'ouvrir ce fichier ...\n");
        cleanexit();
    }
    else
    {
        Read(hdlc,filemem,filesize);
        Close(hdlc);
        hdlc=0;
        printf("\n Fichier lu ...\n");
    }
}

void analyse()
{
    BOOL filbm=FALSE, fbmhd=FALSE, fbbody=FALSE;
    SHORT var;
    UWORD couleur;
    int ncoul;
    UBYTE re,gr,bl; /* red ,green ,blue */

    chk=(struct Chunk *)filemem;
    switch (chk->ckID)
    {
        case ID_CAT:
        case ID_PROP:
        case ID_LIST:
            printf("\n C'est un fichier IFF composé\n");
            break;
        case ID_FORM:
            filbm=TRUE;
            break;
        default:
            printf("\n Ce n'est pas un fichier IFF ...\n");
            break;
    }
    if( !filbm )
        cleanexit();

    taille=chk->ckSize;
    long_forme=taille;

    pteurl=(ULONG *)chk;
    pteurl++;
    pteurl++;
    chk=(struct Chunk *)pteurl;

    vu=8;

    switch (chk->ckID)
    {
        case ID_FTXT:
            printf("\n C'est un TEXTE ...\n");
            break;
        case ID_SMUS:
            printf("\n C'est de la MUSIQUE ...\n");

```



```

        break;
    case ID_8SVX:
        printf("\n C'est un INSTRUMENT musical ...\n");
        break;
    case ID_ILBM:
        filbm=TRUE;
        break;
    default:
        printf("\n Ce n'est pas un format ILBM\n");
        break;
}
if( !filbm )
    cleanexit();

vu+=4;

pteurl=(ULONG *)chk;
pteurl++;
chk=(struct Chunk *)pteurl;

do
{
    taille=chsize(chk->ckSize);    /* taille du CHUNK */
    vu+=taille;

    switch(chk->ckID)
    {
        case ID_BMHD:
            fbmhd=TRUE;

            pteurl=(ULONG *)chk;
            pteurl++;
            pteurl++;
            bmhd=(struct BMHD *)pteurl;

            prof = bmhd->bmhd_nplanes;
            haut = bmhd->bmhd_h;

            bpr=bmhd->bmhd_w;
            var=bpr%16;
            var=16-var;
            bpr+=var;    /* multiple de 16 */
            bpr/=8;      /* octets / rangée */
            break;

        case ID_BODY:
            if ( fbmhd )
            {
                /* taille datas : chk->ckSize */
                pteurl=(ULONG *)chk;
                pteurl++;
                pteurl++;
                fbody=TRUE;
                transfert( (UBYTE *)pteurl );
            }
            else
                fbody=FALSE;
            break;

        case ID_CMAP:
            pteurl=(ULONG *)chk;
            pteurl++;
            pteurl++;
            pteurb=(UBYTE *)pteurl;
            ncoul=(int)( (chk->ckSize)/3);
            printf("\n      %d COULEUR(S) = '$RGB'\n ",ncoul);
            while(ncoul--)
            {
                re=*pteurb++;
                gr=*pteurb++;
                bl=*pteurb++;
                couleur=((UWORD)(re)*0x100+(UWORD)(gr)*0x10+(UWORD)bl)/0x10;
                printf("%3X ",couleur);
            }
            break;
        default:
            break;
    }

    pteurb=(UBYTE *)chk+taille;
    chk=(struct Chunk *)pteurb;

}while( vu < long_forme );

if( !fbody )
    cleanexit();

printf("\n Analyse terminée ...\n");
}

void transfert(dptr)
UBYTE *dptr;
{
    register UBYTE pixel, *pixelptr;

```

```

    register SHORT plane,row,nbytes,count;

    planes_long=bpr * bmhd->bmhd_h; /* 1 bitplane */
    plane1=planes_long;    /* long d'un bitplane */
    planes_long*=prof;     /* 4 bitplanes ou plus ...*/

    planeptr=(UBYTE *)AllocMem(planes_long,MEMF_CLEAR|MEMF_PUBLIC);
    if(planeptr==NULL)
    {
        printf("\nImpossible de réserver la mémoire des datas...\n");
        cleanexit();
    }

    if(bmhd->bmhd_compress==1)    /* compression habituelle */
    {
        for(row=0;row<bmhd->bmhd_h;row++)
        {
            for(plane=0;plane<bmhd->bmhd_nplanes;plane++)
            {
                pixelptr=planeptr+(plane*plane1)+(row*bpr);
                nbytes=bpr;
                while(nbytes > 0)
                {
                    count=(char)*dptr++;
                    if(count >= 0) /* copier n+1 littéral */
                    {
                        count++;
                        nbytes-=count;
                        while(count--)
                            *pixelptr++=*dptr++;
                    }
                    else if(count >= -127) /* répéter -n+1 fois */
                    {
                        count=1-count;
                        nbytes-=count;
                        pixel=*dptr++;
                        while(count--)
                            *pixelptr++=pixel;
                    }
                }
            }
        }
    }
    else /* pas de compression ( ou inconnue !! ) */
    {
        for(row=0;row<bmhd->bmhd_h;row++)
        {
            for(plane=0;plane<bmhd->bmhd_nplanes;plane++)
            {
                pixelptr=planeptr+(plane*plane1)+(row*bpr);
                nbytes=bpr;
                while(nbytes--)
                    *pixelptr++=*dptr++;
            }
        }
    }
}

void lib_mem()
{
    if(hdle)
        Close(hdle);
    hdl=0;

    if(filemem)
        FreeMem(filemem,filesize);
    filemem=0;
}

void output_C()
{
    register SHORT plane,row,nword,wpr;
    UWORD *wordptr;
    FILE *fp,*fopen();
    char tpn[80];
    SHORT cpt;
    int i;

    fp=fopen(nom_complet,"w"); /* écriture */

    if(fp==NULL)
    {
        printf("\n Impossible d'ouvrir ce #&!$!$? fichier ...\n");
        cleanexit();
    }

    wpr=bpr/2;

    for(plane=0;plane<prof;plane++)
    {
        fputc('\12',fp);    /* ligne suivante */
    }

```



```

sprintf(tpn, " /* bitplane No %d : %d lignes de %d mots */ ",
        plane+1, haut, wpr);
fwrite(tpn, 1, (UBYTE)strlen(tpn), fp);
fputc('\12', fp); /* ligne suivante */
cpt=0;
sprintf(tpn, " ");
fwrite(tpn, 1, 5, fp); /* # tabulation */

for(row=0; row<haut; row++)
{
    wordptr=(UWORD *) ( planeptr+(plane*plane1)+(row*bpr) );
    nword=wpr;

    while(nword--)
    {
        sprintf(tpn, "0x%4X", *wordptr);
        for(i=0; i<=3; i++)
        {
            if( tpn[2+i] == ' ' )
                tpn[2+i]='0';
        }
        fwrite(tpn, 1, (UBYTE)strlen(tpn), fp);
        wordptr++;
        cpt++;
        if( cpt==8 )
        {
            fputc('\12', fp); /* ligne suivante */
            sprintf(tpn, " ");
            fwrite(tpn, 1, 5, fp); /* # tabulation */
            cpt=0;
        }
    }
    /*fin de la rangée */
}
/* fin du bitplane */
}
fclose(fp);
}

void output_A()
{
    register SHORT plane, row, nword, wpr;
    UWORD *wordptr;
    FILE *fp, *fopen();
    char tpn[80];
    SHORT cpt;
    int i;

    fp=fopen(nom_complet, "w"); /* écriture */

    if(fp==NULL)
    {
        printf("\n Impossible d'ouvrir ce #&!!$/\n? fichier ... \n");
        cleanexit();
    }

    wpr=bpr/2;

    for(plane=0; plane<prof; plane++)
    {
        fputc('\12', fp); /* ligne suivante */
        sprintf(tpn, " ; bitplane No %d : %d lignes de %d mots ",
                plane+1, haut, wpr);
        fwrite(tpn, 1, (UBYTE)strlen(tpn), fp);
        fputc('\12', fp); /* ligne suivante */
        cpt=0;
        sprintf(tpn, " DC.W ");
        fwrite(tpn, 1, 9, fp); /* # tabulation */

        for(row=0; row<haut; row++)
        {
            wordptr=(UWORD *) ( planeptr+(plane*plane1)+(row*bpr) );
            nword=wpr;

            while(nword--)
            {
                sprintf(tpn, "%4X", *wordptr);
                for(i=0; i<=3; i++)
                {
                    if( tpn[1+i] == ' ' )
                        tpn[1+i]='0';
                }
                fwrite(tpn, 1, (UBYTE)strlen(tpn), fp);
                wordptr++;
                cpt++;
                if( cpt==8 )
                {
                    fputc('\12', fp); /* ligne suivante */
                    if( (row != haut-1) || (nword != 0) )
                    {
                        sprintf(tpn, " DC.W ");
                        fwrite(tpn, 1, 9, fp); /* # tabulation */
                    }
                }
            }
        }
    }
}

```

```

        cpt=0;
    }
    else
    {
        if( (row != haut-1) || (nword != 0) )
            fputc(',', fp); /* + la virgule */
    }
}
/*fin de la rangée */

}
/* fin du bitplane */
}
fclose(fp);
}

/***** ROUTINE PRINCIPALE *****/

void main()
{
    do
    {
        printf("\n Type de SOURCE voulu :");
        printf("\n 'C' pour LATTICE ");
        printf("\n 'A' pour DEVPAC ");
        categ=fgetchar();
        if(categ == 'a')
            categ='A';
        if(categ == 'c')
            categ='C';
        while( (categ != 'C') && (categ != 'A') );

        resul=input("NOM de l'IMAGE IFF à LIRE");
        if( resul == 0 )
            cleanexit();

        printf("\n CHEMIN : '%s' ; FICHER : '%s' \n",
                dirname, fname);

        clef=Lock(nom_complet, ACCESS_READ);
        if(clef==NULL)
        {
            printf("\n Accès lecture impossible : le chemin existe ?\n");
            cleanexit();
        }

        res=Examine(clef, &bloc);

        if(bloc.fib_DirEntryType>0)
        {
            printf("\n C'est un répertoire ... \n\n");
            cleanexit();
        }

        filesize=bloc.fib_Size;

        if(filesize != 0)
        {
            load_fichier();
            analyse();
        }
        lib_mem(); /* libère une partie de la mémoire et le handle */

        Unlock(clef);
        clef=0;

        if(categ == 'C')
            resul=input("NOM du SOURCE-C à CREER");
        else
            resul=input("NOM du SOURCE-ASSEMBLEUR à CREER");

        if( resul == 0 )
            cleanexit();

        printf("\n CHEMIN : '%s' ; FICHER : '%s' \n",
                dirname, fname);

        printf("\n Sortie en cours ... \n");

        if(categ == 'C')
            output_C();
        else
            output_A();

        cleanexit();
    }
}

```

Par Denis Genot

SUBWAY : LES SPRITES VIRTUELS (II)

Il y a de cela deux mois, nous avons étudié l'animation à base de VSprites à l'aide d'un programme faisant bouger des petits logos ANT sur un écran Intuition. Le programme étant assez long, peu d'explications avaient été fournies.

Le système d'animation de l'Amiga, dénommé GELS (pour *Graphics ELements System*), est constitué d'un certain nombre de couches logicielles superposées, chacune utilisable indépendamment des autres. Ainsi, lorsque nous avons étudié les sprites Hardware (*simple sprites*), nous avons utilisé la couche la plus basse du GELS qui consiste littéralement en une interface pour piloter les circuits spécialisés de l'Amiga.

En programmant des VSprites, nous avons utilisé la couche supérieure du GELS : en effet, un VSprite n'existe pas physiquement, il ne s'agit que d'une représentation virtuelle (d'où son nom !) qui est utilisée par le GELS pour créer, animer et détruire des "simple sprites" à l'écran. Cette phase de création, animation et destruction n'étant plus contrôlable par l'utilisateur, mais gérée automatiquement par le système.

Le "simple sprite" alloué pour un VSprite ne l'est que le temps nécessaire à sa visualisation, ce qui permet d'avoir plus de huit VSprites à l'écran, pour peu qu'ils ne soient pas positionnés à plus de 8 sur les mêmes lignes au même instant (cf. figure 1).

Un VSprite possède une structure de données associée contenant la position du sprite, sa taille, son image et ses couleurs propres (cf. descriptif paru dans le numéro 26), ce qui permet au système d'animation de travailler convenablement.

AVANTAGES ET INCONVENIENTS

Le gros avantage d'utiliser les VSprites est que l'on n'a plus à se préoccuper exactement de la manière d'allouer les sprites Hardware : on indique seulement où les sprites doivent apparaître et le GELS fait le reste. Autre avantage, la possibilité d'utiliser plus de huit VSprites en même temps, sachant que le système peut réutiliser le même sprite Hardware pour afficher deux objets différents à des endroits différents de l'écran.

Malheureusement, il n'y a pas que des avantages dans l'utilisation des VSprites et un certain nombre de contraintes sont à prendre en compte si l'on ne veut pas avoir la désagréable surprise de voir son programme fonctionner de manière stochastique...

Comme mentionné plus haut, le hardware de l'Amiga n'autorise que huit "vrais" sprites maximum. Cela signifie que si l'on demande au système d'afficher plus de huit VSprites à une même coordonnée, certains ne seront pas affichés. C'est comme cela et on y peut rien. Mais là où le problème se corse (comme disait Napoléon), c'est que, pratiquement, on ne peut pas allouer les huit sprites : le sprite 0 est réservé à Intuition pour afficher le pointeur de souris, il est donc inutilisable. Et comme le sprite 1 lui est associé (les sprites Hardware allant par paire), il est également déconseillé de s'en servir. En effet cette paire de sprites utilise les mêmes registres de couleur (17 à 19, le 16 étant considéré comme transparent). Toute modification de la couleur du sprite 1 entraînera donc une modification de celle du sprite 0 et vous aurez la désagréable surprise de voir flasher votre pointeur de souris.

Un deuxième problème peut survenir dans le cas de superposition des sprites : il faut savoir que les sprites Hardware, lorsqu'ils se superposent, se cachent mutuellement suivant un ordre de priorité bien établi. Celui ayant le plus faible numéro étant toujours "au dessus" de l'autre (c'est pour cela que le pointeur de souris ne peut jamais être

caché). Quand l'on utilise des VSprites, le système graphique associe arbitrairement un numéro de sprite à chaque VSprite, et ce dynamiquement. Cette allocation dont l'utilisateur n'est pas maître pose donc deux problèmes :

- on ne sait pas quel sprite passera "au dessus" de l'autre ;
- il peut arriver que, lors d'un changement d'assignation, il y ait basculement des deux sprites, ce qui a pour effet de les permuter à l'écran.

Troisième contrainte, si tous vos VSprites utilisent des couleurs différentes, vous ne pouvez pas avoir au maximum huit sprites mais seulement quatre. Pourquoi ? Hé bien c'est fort simple : comme chaque sprite d'une paire utilise les mêmes registres de couleurs que son petit copain, le système ne peut afficher les huit sur une même ligne... Pour se simplifier la tâche, il n'en affiche que quatre !

Enfin, quatrième contrainte, mais qui est propre à l'utilisation des sprites en général, plus l'on utilise de sprites, moins l'on a de couleurs disponibles pour le playfield. N'oublions pas que les circuits de gestion des sprites Hardware utilisent les registres de couleurs 17 à 19, 21 à 23, 25 à 27, 29 à 31, qu'il ne faudra donc pas utiliser pour le playfield, ou alors très judicieusement.

GELS

Que vous désiriez utiliser des VSprites, des Bobs ou tout autre objet géré par le GELS, il faudra commencer par l'initialiser. Le système d'animation graphique tient à jour une liste chaînée des éléments graphiques à visualiser, liste qui doit toujours au minimum contenir deux éléments "bidons", c'est-à-dire deux VSprites qui ne seront jamais représentés à l'écran, mais qui servent au GELS à repérer facilement le début et la fin de la liste. Ces VSprites sont appelés "sprite de tête" et "sprite de queue" puisque tous les objets créés ensuite seront insérés entre eux. La première chose qu'il faut faire est donc créer cette liste de deux éléments bidons.

Par défaut, le système de gestion des sprites s'alloue tous les sprites disponibles, y compris le sprite 1, ce qui, nous l'avons vu précédemment, peut entraîner des problèmes avec le pointeur de souris. Pour palier à cela, il suffit d'indiquer quels sprites doivent être réservés par le système : il existe un champ dans la structure GelsInfo qui permet cela, le champ `sprRsrvd`. Ainsi, pour réserver les sprites 2 à 7, on tapera la ligne suivante :

```
struct GelsInfo *gel;
gel->sprRsrvd = 0xFC; /* en binaire : %11111100 */
```

Si l'on veut réserver le maximum de sprites disponibles (hors les sprites 0 et 1), il faudra se renseigner au préalable quels sont les sprites utilisés par d'autres tâches (pour éviter les conflits). On est capable de savoir quels sont les sprites effectivement réservés par d'autres tâches en regardant le champ `SpriteReserved` de la structure `GfxBase`. Chaque sprite est visualisé par un bit de ce champ : si le bit est à 1, le sprite correspondant est réservé.

Pour s'allouer les sprites encore disponible on tapera donc :

```
gel->sprRsrvd = 0xFC & (! (GfxBase->SpriteReserved));
```

Le système d'animation a besoin de certaines données d'entrée pour décider quel sprite Hardware il va utiliser pour visualiser le prochain VSprite. Ces variables d'état sont placées dans la structure `GelsInfo` et comprennent deux tableaux, `nextLine` et `lastColor`.

Dans `nextLine`, le système stocke le numéro de la prochaine ligne écran ou sera disponible chaque sprite hardware. Ceci lui permet de savoir quels sont les sprites Hardware qui peuvent être candidats pour afficher un VSprite à une position donnée, et ceux qui ne peuvent le faire car déjà occupés.

Dans `lastColor`, le système stocke pour chaque sprite Hardware un pointeur vers le jeu de couleurs utilisés en dernier. Ainsi, lorsqu'un VSprite a besoin d'être utilisé, le système recherche dans le tableau `lastColor` et en fonction des sprites susceptibles d'être utilisés, quel

sprite Hardware possède le même jeu de couleur que le VSprite à afficher. S'il en existe un, il l'utilise en premier car cela évite des modifications des registres de couleurs associés au sprite, très coûteuses en temps-machine.

Ces deux champs de la structure GelsInfo doivent bien entendu être alloués dans votre programme, ce qui se fera sans problème de la manière suivante :

```
gel->nextLine = (WORD *)AllocMem(sizeof(WORD) * 8,
MEMF_PUBLIC|MEMF_CLEAR));
et
gel->lastColor = (WORD *)AllocMem(sizeof(WORD) * 8,
MEMF_PUBLIC|MEMF_CLEAR));
```

CREATION DU VSPRITE

La création d'un VSprite s'effectue de manière simple. La première étape consiste à remplir une structure VSprite, soit déclarée statiquement, soit allouée dynamiquement avec AllocMem(). Cette structure possède un certain nombre de champs dont certains ont les mêmes fonctions que ceux de la structure SimpleSprite décrite dans un précédent numéro.

Pour placer à l'écran un VSprite, il faut d'abord indiquer sa position dans les champs *v->X* et *v->Y*. Ensuite, sa largeur et sa hauteur seront placées dans *v->Width* (la largeur s'exprime en mots de 16 bit et un VSprite a toujours une largeur de 16 pixels, donc de 1 mot) et *v->Height*. La profondeur *v->Depth* doit également être indiquée : pour un VSprite, la profondeur est toujours égale à 2, mais on peut avoir plus de couleur avec un Bob (qui utilise la même structure). On associe ensuite l'image du sprite au champ *v->ImageData* et sa couleur au champ *v->SprColors*.

On continue en mettant dans le champ *v->Flag* la valeur *VSPRITE* pour indiquer qu'il s'agit d'un VSprite et non d'un Bob, et on termine allègrement par la gestion des collisions :

- en mettant la valeur 1 dans *v->MeMask* et *v->HitMask* afin d'avoir une détection de collision sur les bordures de l'écran ;
- en réservant de la mémoire pour le champ *v->BorderLine*, qui contiendra une image rectangulaire, sur un seul plan de bits, du VSprite afin de permettre une détection plus rapide, bien qu'approximative, des collisions ;
- en réservant également de la mémoire pour le champ *v->CollMask* qui contiendra une image de la projection du VSprite sur un bitplane afin de faire des détections exactes de collision.

Que reste-t'il à faire ? Presque rien, sinon mettre à 0 les champs *v->PlanePick* et *v->PlaneOnOff* qui ne servent que pour les Bobs et appeler la fonction *InitMasks()* de la *graphics.library*, qui finira automatiquement le travail en créant effectivement les deux masques *CollMask* et *BorderLine*.

Pour ajouter le VSprite ainsi créé, il suffit de faire appel à la fonction *AddVSprite()* qui fait le travail tout seul. On lui passe en paramètres l'adresse (de la structure) du VSprite à ajouter et celle du RastPort de l'écran visé.

AFFICHAGE ET DEPLACEMENT

Nos VSprites sont maintenant dans la liste, tout est donc pour le mieux. Pour déplacer un VSprite, on modifiera les champs *v->X* et *v->Y* et on lancera un processus qui consiste à réactualiser l'ensemble de la visualisation pour prendre en compte cette modification de position.

Les VSprites devant être affichés du haut de l'écran vers le bas, il faut commencer par trier la liste des éléments graphiques à afficher, en appelant la fonction *SortGList()*. On crée ensuite les instructions Copper de visualisation des sprites pour le ViewPort, en appelant la routine *DrawGList()*, avec comme argument le RastPort et le ViewPort.

Enfin, on demande à Intuition d'effectuer elle-même les modifications sur l'écran et de le réafficher grâce à la séquence *MakeScreen()* / *RethinkDisplay()*. Si on n'utilise pas Intuition, *MakeVPort()* et *LoadView()* seront mises à contribution.

FIN DU TRAVAIL

Une fois nos VSprites devenus inutiles, il faut les effacer et libérer proprement les ressources utilisées, à savoir la structure *v* elle-même, *v->CollMask* et *v->BorderLine*, sans oublier *v->VSBob->SaveBuffer*, *v->VSBob->DBuffer* et *v->VSBob* si notre VSprite était un Bob.

Un petit bout de code valant des fois mieux qu'un long discours, voici la routine permettant de libérer toutes ces ressources.

```
void FreeVSprite(v)
struct VSprite *v;
{
    ULONG vSize1, vSize2;

    vSize1 = sizeof(SHORT) * v->Width * v->Height;
    vSize2 = vSize1 * v->Depth;

    if (v != NULL) {
        if (v->VSBob != NULL) {
            if (v->VSBob->SaveBuffer != NULL) {
                FreeMem(v->VSBob->SaveBuffer, vSize1);
            }
            if (v->VSBob->DBuffer != NULL) {
                if (v->VSBob->DBuffer->BufBuffer != 0) {
                    FreeMem(v->VSBob->DBuffer->BufBuffer,
                        vSize1);
                }
                FreeMem(v->VSBob->DBuffer,
                    sizeof(struct DBufPacket));
            }
            FreeMem(v->VSBob, sizeof(struct Bob));
        }
        if (v->CollMask != NULL) {
            FreeMem(v->CollMask, vSize2);
        }
        if (v->BorderLine != NULL) {
            FreeMem(v->BorderLine, vSize2);
        }
        FreeMem(v, sizeof(struct VSprite));
    }
}
```

Ensuite on détruira le GELS, ce dernier n'ayant désormais plus de raison d'être. On libèrera donc les zones *collHandler*, *lastColor*, *nextLine*, *gelHead* et *gelTail*.

Soit par programme :

```
if (gelinfo.collHandler != NULL)
    FreeMem(gelinfo.collHandler, sizeof(struct collTable));
if (gelinfo.lastColor != NULL)
    FreeMem(gelinfo.lastColor, sizeof(LONG) * 8);
if (gelinfo.nextLine != NULL)
    FreeMem(gelinfo.nextLine, sizeof(WORD) * 8);
if (gelinfo.gelHead != NULL)
    FreeMem(gelinfo.gelHead, sizeof(struct VSprite));
if (gelinfo.gelTail != NULL)
    FreeMem(gelinfo.gelTail, sizeof(struct VSprite));
```

Bon, hé bien voilà pour la description du programme sur les VSprites proposé il y a deux mois. J'espère que c'est beaucoup plus clair pour vous désormais. Le reste du programme concerne des parties que nous avons vues en détail précédemment, je ne reviendrai donc pas dessus.

Le mois prochain, nous terminerons cette partie animation sur Amiga par les Bobs, ou Blitter Objects. Vous pourrez faire ainsi la comparaison avec les VSprites et utiliser l'un ou l'autre selon la nature de votre application. En attendant, je vous souhaite une bonne lecture et vous dis : "à dans un mois".

Par Herr Doktor Von GlutenStimmellmDorf.

UTILITAIRE II : ExecMaster (fin)

Comme promis, voici la suite du méga-listing de notre utilitaire du mois dernier, ExecMaster. Après ça, vous pourrez enfin disposer de cet excellent programme de l'ami Max.

Histoire de vous simplifier les choses, voici tout de même un petit mode d'emploi. Rassurez-vous, rien de bien compliqué, mais il paraît que ça ne se fait pas de commencer un article directement par du listing, alors il faut que j'écrive quelques lignes... O, pas grand chose, juste un ou deux paragraphes, histoire de préserver l'harmonie visuelle du magazine. En jargon de journaliste, on appelle ça du remplissage. Ah ma pauvre dame, on est bien peu de choses, c'est moi qui vous l'dis.

Or donc, voici un mode d'emploi des deux listings suivants : vous les tapez, vous les mettez dans le même répertoire que *ExecMaster.s* publié dans notre précédent numéro et vous assemblez le tout pour obtenir un programme exécutable du nom de ExecMaster, et c'est tout ! Avouez que c'était simple. Si, juste une petite précision : le premier listing s'appelle *em_windows.s* et contient la définition de l'écran et des fenêtres d'ExecMaster, et le second s'appelle *em_menus.s* et contient la définition de tous les menus du programme.

Bon, ben cette fois-ci, c'est tout, je retourne me coucher. Atchao bonsoir.

Listing 1 : em_windows.s

```
ns      dc.w    0,0,640,256,2
        dc.b    1,2
        dc.w    V_HIRES,CUSTOMSCREEN
        dc.l    0,sctitre,0,0

nw      dc.w    0,0,640,256
        dc.b    1,2
        dc.l    MENUPIK
        dc.l    BACKDROP|BORDERLESS|ACTIVATE|NOCAREREFRESH
        dc.l    0,0,0,0,0
        dc.w    0,0,0,0
        dc.w    CUSTOMSCREEN

nw3     dc.w    134,0,300,10
        dc.b    1,2
        dc.l    GADGETUP|ACTIVEWINDOW
        dc.l    ACTIVATE|RMBTRAP
        dc.l    w3gad,0,0,0,0
        dc.w    0,0,0,0
        dc.w    CUSTOMSCREEN

w3gad   dc.l    0
        dc.w    120,1,150,8
        dc.w    GADGHCOMP|SELECTED
        dc.w    TOPBORDER|RELVERIFY|LONGINT
        dc.w    STRGADGET
        dc.l    0,0,0,0,0,0,0,0
        dc.w    0
        dc.l    0

strinfo dc.l    strbuf,strbuf+40
        dc.w    1,40,0,0,0,0,0,0
        dc.l    0,0,0,0
        dc.b    '0'
        dc.b    79,0

Palette dc.w    $0356,$0FFF,$0000,$00AF

sctitre dc.b    "ExecMaster 0.7 - 1991, Max pour ANT",0
even
```

Listing 2 : em_menus.s

```
; ***** Menu 'Affichage'
MainMenuList:
m1      dc.l    0
        dc.w    2,0,79,0,MENUENABLED
        dc.l    mName,m1Item1
```

```
dc.w    0,0,0,0

m1Item1 dc.l    m1Item2 ; Affiche/Taches
        dc.w    0,0,195,10
        dc.w    CHECKIT|ITEMTEXT|COMMSEQ|ITEMENABLED|HIGHCOMP
        dc.l    %111111111111111111111111111111110,m1IT1,0
        dc.b    'T',0
        dc.l    0
        dc.w    MENUNULL
        dc.l    Affiche_taches ; xi_Func
m1IT1   dc.b    1,2,RP_JAM2,0
        dc.w    20,1
        dc.l    0,m1Ttxt1,0

m1Item2 dc.l    m1Item3 ; Affiche/Libraries
        dc.w    0,10,195,10
        dc.w    CHECKIT|ITEMTEXT|COMMSEQ|ITEMENABLED|HIGHCOMP
        dc.l    %1111111111111111111111111111111101,m1IT2,0
        dc.b    'L',0
        dc.l    0
        dc.w    MENUNULL
        dc.l    Affiche_libraries ; xi_Func
m1IT2   dc.b    1,2,RP_JAM2,0
        dc.w    20,1
        dc.l    0,m1Ttxt2,0

m1Item3 dc.l    m1Item4 ; Affiche/Devices
        dc.w    0,20,195,10
        dc.w    CHECKIT|ITEMTEXT|COMMSEQ|ITEMENABLED|HIGHCOMP
        dc.l    %11111111111111111111111111111111011,m1IT3,0
        dc.b    'D',0
        dc.l    0
        dc.w    MENUNULL
        dc.l    Affiche_devices ; xi_Func
m1IT3   dc.b    1,2,RP_JAM2,0
        dc.w    20,1
        dc.l    0,m1Ttxt3,0

m1Item4 dc.l    m1Item5 ; Affiche/Ports
        dc.w    0,30,195,10
        dc.w    CHECKIT|ITEMTEXT|COMMSEQ|ITEMENABLED|HIGHCOMP
        dc.l    %111111111111111111111111111111110111,m1IT4,0
        dc.b    'P',0
        dc.l    0
        dc.w    MENUNULL
        dc.l    Affiche_ports ; xi_Func
m1IT4   dc.b    1,2,RP_JAM2,0
        dc.w    20,1
        dc.l    0,m1Ttxt4,0

m1Item5 dc.l    m1Item6 ; Affiche/Mémoire
        dc.w    0,40,195,10
        dc.w    CHECKIT|ITEMTEXT|COMMSEQ|ITEMENABLED|HIGHCOMP
        dc.l    %1111111111111111111111111111111101111,m1IT5,0
        dc.b    'M',0
        dc.l    0
        dc.w    MENUNULL
        dc.l    Affiche_memoire ; xi_Func
m1IT5   dc.b    1,2,RP_JAM2,0
        dc.w    20,1
        dc.l    0,m1Ttxt5,0

m1Item6 dc.l    m1Item7 ; Affiche/Resources
        dc.w    0,50,195,10
        dc.w    CHECKIT|ITEMTEXT|COMMSEQ|ITEMENABLED|HIGHCOMP
        dc.l    %11111111111111111111111111111111011111,m1IT6,0
        dc.b    'R',0
        dc.l    0
        dc.w    MENUNULL
        dc.l    Affiche_resources ; xi_Func
m1IT6   dc.b    1,2,RP_JAM2,0
        dc.w    20,1
        dc.l    0,m1Ttxt6,0

m1Item7 dc.l    m1Item8 ; Affiche/Modules résidents
        dc.w    0,60,195,10
        dc.w    CHECKIT|ITEMTEXT|COMMSEQ|ITEMENABLED|HIGHCOMP
        dc.l    %111111111111111111111111111111110111111,m1IT7,0
        dc.b    'O',0
        dc.l    0
        dc.w    MENUNULL
        dc.l    Affiche_modules ; xi_Func
m1IT7   dc.b    1,2,RP_JAM2,0
        dc.w    20,1
        dc.l    0,m1Ttxt7,0

m1Item8 dc.l    m1Item9 ; Affiche/Interruptions
        dc.w    0,70,195,10
        dc.w    CHECKIT|ITEMTEXT|COMMSEQ|ITEMENABLED|HIGHCOMP
        dc.l    %1111111111111111111111111111111101111111,m1IT8,0
        dc.b    'I',0
        dc.l    0
        dc.w    MENUNULL
        dc.l    Affiche_interr ; xi_Func
m1IT8   dc.b    1,2,RP_JAM2,0
```



```

dc.w 20,1
dc.l 0,m1Txt8,0

m1Item9 dc.l 0 ; Affiche/Quitter
dc.w 0,85,195,10
dc.w ITEMTEXT|COMMSEQ|ITEMENABLED|HIGHCOMP
dc.l 0,m1IT9,0
dc.b 'Q',0
dc.l 0
dc.w MENUNULL
dc.l Affiche_Quitter ; xi_Func
m1IT9 dc.b 1,2,RP_JAM2,0
dc.w 20,1
dc.l 0,m1Txt9,0

; ***** Menu 'Tâches'
m2 dc.l 0
dc.w 87,0,79,0,MENUENABLED
dc.l m2Name,m2Item1
dc.w 0,0,0,0

m2Item1 dc.l m2Item2 ; Tâches/SetTaskPri
dc.w 0,0,120,10
dc.w ITEMTEXT|ITEMENABLED|HIGHCOMP
dc.l 0,m2IT1,0
dc.b 0,0
dc.l 0
dc.w MENUNULL
dc.l Pri_tache ; xi_Func
m2IT1 dc.b 1,2,RP_JAM2,0
dc.w 20,1
dc.l 0,m2Txt1,0

m2Item2 dc.l m2Item3 ; Tâches/Geler
dc.w 0,10,120,10
dc.w ITEMTEXT|ITEMENABLED|HIGHCOMP
dc.l 0,m2IT2,0
dc.b 0,0
dc.l 0
dc.w MENUNULL
dc.l Gele_tache ; xi_Func
m2IT2 dc.b 1,2,RP_JAM2,0
dc.w 20,1
dc.l 0,m2Txt2,0

m2Item3 dc.l 0 ; Tâches/Réchauffer
dc.w 0,20,120,10
dc.w ITEMTEXT|ITEMENABLED|HIGHCOMP
dc.l 0,m2IT3,0
dc.b 0,0
dc.l 0
dc.w MENUNULL
dc.l Chauffe_tache ; xi_Func
m2IT3 dc.b 1,2,RP_JAM2,0
dc.w 20,1
dc.l 0,m2Txt3,0

; ***** Menu 'Libraries'
m3 dc.l 0
dc.w 87,0,79,0,MENUENABLED
dc.l m3Name,m3Item1
dc.w 0,0,0,0

m3Item1 dc.l m3Item2 ; Libraries/Ouvrir
dc.w 0,0,90,10
dc.w ITEMTEXT|ITEMENABLED|HIGHCOMP
dc.l 0,m3IT1,0
dc.b 0,0
dc.l 0
dc.w MENUNULL
dc.l Ouvre_lib ; xi_Func
m3IT1 dc.b 1,2,RP_JAM2,0
dc.w 20,1
dc.l 0,m3Txt1,0

m3Item2 dc.l m3Item3 ; Libraries/Fermer
dc.w 0,10,90,10
dc.w ITEMTEXT|ITEMENABLED|HIGHCOMP
dc.l 0,m3IT2,0
dc.b 0,0
dc.l 0
dc.w MENUNULL
dc.l Ferme_lib ; xi_Func
m3IT2 dc.b 1,2,RP_JAM2,0
dc.w 20,1
dc.l 0,m3Txt2,0

m3Item3 dc.l 0 ; Libraries/Purger
dc.w 0,20,90,10
dc.w ITEMTEXT|ITEMENABLED|HIGHCOMP
dc.l 0,m3IT3,0
dc.b 0,0
dc.l 0
dc.w MENUNULL
dc.l Purge_lib ; xi_Func

```

```

m3IT3 dc.b 1,2,RP_JAM2,0
dc.w 20,1
dc.l 0,m3Txt3,0

; ***** Menu Ports
m4 dc.l 0
dc.w 87,0,79,0,MENUENABLED
dc.l m4Name,m4Item1
dc.w 0,0,0,0

m4Item1 dc.l m4Item2 ; Ports/Cacher
dc.w 0,0,100,10
dc.w ITEMTEXT|ITEMENABLED|HIGHCOMP
dc.l 0,m4IT1,0
dc.b 0,0
dc.l 0
dc.w MENUNULL
dc.l Cache_port ; xi_Func
m4IT1 dc.b 1,2,RP_JAM2,0
dc.w 20,1
dc.l 0,m4Txt1,0

m4Item2 dc.l 0
dc.w 0,10,100,10
dc.w ITEMTEXT|ITEMENABLED|HIGHCOMP
dc.l 0,m4IT2,0
dc.b 0,0
dc.l 0
dc.w MENUNULL
dc.l Revele_port ; xi_Func
m4IT2 dc.b 1,2,RP_JAM2,0
dc.w 20,1
dc.l 0,m4Txt2,0

; ***** Menu 'Mémoire'
m5 dc.l 0
dc.w 87,0,79,0,MENUENABLED
dc.l m5Name,m5Item1
dc.w 0,0,0,0

m5Item1 dc.l 0
dc.w 0,0,100,10
dc.w ITEMTEXT|ITEMENABLED|HIGHCOMP
dc.l 0,m5IT1,0
dc.b 0,0
dc.l 0
dc.w MENUNULL
dc.l Pri_memoire ; xi_Func
m5IT1 dc.b 1,2,RP_JAM2,0
dc.w 20,1
dc.l 0,m5Txt1,0

; ***** Chaînes ASCII
TOPAZ80 dc.l TOPAZname
dc.w TOPAZ_EIGHTY
dc.b 0,0

TOPAZname:
dc.b "topaz.font",0

m1Name dc.b "Affichage",0
m1Txt1 dc.b "Tâches",0
m1Txt2 dc.b "Libraries",0
m1Txt3 dc.b "Devices",0
m1Txt4 dc.b "Ports",0
m1Txt5 dc.b "Mémoire",0
m1Txt6 dc.b "Resources",0
m1Txt7 dc.b "Modules résidents",0
m1Txt8 dc.b "Interruptions",0
m1Txt9 dc.b "Quitter",0

m2Name dc.b "Contrôles",0
m2Txt1 dc.b "SetTaskPri",0
m2Txt2 dc.b "Geler",0
m2Txt3 dc.b "Réchauffer",0

m3Name dc.b "Contrôles",0
m3Txt1 dc.b "Ouvrir",0
m3Txt2 dc.b "Fermer",0
m3Txt3 dc.b "Purger",0

m4Name dc.b "Contrôles",0
m4Txt1 dc.b "Cacher",0
m4Txt2 dc.b "Révéler",0

m5Name dc.b "Contrôles",0
m5Txt1 dc.b "Priorité",0
even

```


TRANSACTION : SINUS-SCROLL

La parole est ici donnée à Pierre Chalamet dont le Sinus-Scroll est du plus bel effet. Pierre gagne les 1500 F promis ainsi que toute notre considération. Et la vôtre. Ce qui n'est pas peu dire.

Le principe du scroll est assez simple : il est d'abord créé dans une zone qui ne sera jamais affichée (hidden_screen) puis recopié tranche par tranche grâce au blitter. Mais pour faire ce genre de travail, il faut beaucoup de temps et utiliser deux écrans en alternance (double buffering).

Durant la VBL on effectue les opérations suivantes :

- permutation des écrans
- effacement de l'écran non affiché
- on rentre les lettres s'il le faut dans la zone hidden_screen et on donne un coup de blitter pour faire bouger de droite à gauche
- on recopie par tranche de deux colonnes la zone hidden_screen dans l'écran caché. Le blitter ne permettant des transferts que d'un mot minimum, on est obligés de ruser et de faire un OR avec la source et la destination, car sinon on effacerait les autres tranches. Il ne faut donc pas s'étonner de voir que bltBpt et bltDpt pointent sur la même adresse (A or B -> D).

Remarques :

- l'image à insérer au label lettres doit être de 320 pixels (40 octets) sur 256 lignes au format RAW (surtout pas IFF !) et un seul bitplan. Chaque lettre fait 32 pixels de large sur 32 de haut.
- l'impression d'une onde qui passe est donnée par le déplacement d'un pointeur sur la table des déplacements.

Il est possible d'optimiser le programme (code relatif au PC, initialisation du blitter avant d'entrer dans les boucles...) mais pas des masses, car le blitter prend un sacré temps pour recopier tous les blocs d'image, même s'il tourne à plein régime (le bit 10 de dmacon est mis).

Listing 1 : MakeSinus.bas

```
c=0
OPEN "o",#1,"sinus.s"
FOR a=0 TO 2*3.14159 STEP .01
b=INT(SIN(a)*100+105)
PSET(c,b)
c=c+1
b=b*40
IF c=1 THEN
a$=CHR$(9)+"dc.w"+STR$(b)
END IF
IF c>1 THEN
IF LEN(a$)<60 THEN
a$=a$+" "+STR$(b)
ELSE
PRINT #1,a$
a$=CHR$(9)+"dc.w"+STR$(b)
END IF
END IF
NEXT a
PRINT #1,a$
CLOSE #1
```

Listing 2 : Registers.i

```
dmacon=$96
dmaconr=$02
intreq=$9c
intreqr=$1e
intena=$9a
intenaar=$1c
ciaapra=$bfe001
color00=$180
cop11c=$80
copjmp1=$88
startlist=38
diwstrt=$8e
ddfstrt=$92
bplcon0=$100
```

```
bplcon1=$102
bplcon2=$104
bpl1mod=$108
bpl2mod=$10a
bpl1ptH=$e0
bpl1ptL=$e2
forbid=-132
permit=-138
openlibrary=-552
closelibrary=-414
execbase=4
bltapt=$50
bltbpt=$4c
bltdpt=$54
bltamod=$64
bltbmod=$62
bltdmod=$66
bltafwm=$44
bltalwm=$46
bltcon0=$40
bltcon1=$42
bltsize=$58
bltadat=$74
```

Listing 3 : Save_All.s

```
save_all lea $dff000,a5
move.b #10000111,$bfd100
move.l execbase,a6
jsr forbid(a6)
move.l $6c,save_vecteur_irq
move.w intena(a5),save_intena
or.w #$8100,save_intena
move.w dmaconr(a5),save_dmacon
or.w #$c000,save_dmacon
rts

restore_all lea $dff000,a5
move.l save_vecteur_irq,$6c
move.w #$7fff,intena(a5)
move.w save_intena,intena(a5)
move.w #$7fff,dmacon(a5)
move.w save_dmacon,dmacon(a5)
move.l execbase,a6
lea name_glib,a1
moveq #0,d0
jsr openlibrary(a6)
move.l d0,a0
move.l startlist(a0),cop11c(a5)

coplist clr.w copjmp1(a5)
move.l d0,a1
jsr closelibrary(a6)
jsr permit(a6)
rts

save_intena dc.w 0
save_dmacon dc.w 0
save_vecteur_irq dc.l 0
name_glib dc.b "graphics.library",0
even
```

Listing 4 : SinusScroll.s

```
include Registers.i

bsr save_all

lea $dff000,a6
move.w #$7fff,d0
move.w d0,dmacon(a6)
move.w d0,intena(a6)
move.l #$003800d0,ddfstrt(a6)
move.l #$2c8129c1,diwstrt(a6)
move.w #0,bpl1mod(a6)
move.w #0,bpl2mod(a6)
move.w #$1200,bplcon0(a6)
move.w #0,bplcon1(a6)
move.w #0,bplcon2(a6)
move.l #$00000fff,color00(a6)
move.l #vbl,$6c
move.l #coplist,cop11c(a6)
clr.w copjmp1(a6)
move.w #$87c0,dmacon(a6)
move.w #$c020,intena(a6)

mickey btst #6,ciaapra
bne mickey
bsr restore_all
moveq.l #0,d0
rts

include Save_All.s
vbl movem.l d0-d7/a0-a6,-(sp)
lea $dff000,a6
btst #5,intreqr+1(a6) ;test la provenance
beq pas_vbl ;de l'interrupt niveau 3
bsr flip_screen ;permuter les ecrans
bsr clear_hidden_screen ;efface l'ecran caché
bsr do_scroll ;deplace le scroll caché
bsr put_scroll ;copie scroll dans ecran caché
move.w #$20,intreq(a6) ;fin vbl
pas_vbl movem.l (sp)+,d0-d7/a0-a6
```



```

rte

next_screen
    dc.l écran1                ;adresse de l'écran caché

wait_blit
    btst #14,dmaconr(a6)
loop_blit
    btst #14,dmaconr(a6)
    bne loop_blit
    rts

flip_screen
    move.l next_screen,d0      ;echange les ecrans
    move.l #écran1,d1         ;pour faire du double-buffering
    cmp.l d0,d1
    beq put_écran1

    move.l #écran2,bpl1ptH(a6)
    move.l #écran1,next_screen
    rts

put_écran1
    move.l #écran1,bpl1ptH(a6)
    move.l #écran2,next_screen
    rts

clear_hidden_screen
    bsr wait_blit              ;efface l'écran caché
    clr.w bltadat(a6)
    clr.w bltamod(a6)
    move.w #ffff,bltafwm(a6)
    move.w #ffff,bltalwm(a6)
    move.w #01f0,bltcon0(a6)
    move.w #0,bltcon1(a6)
    move.w #0,bltdmod(a6)
    move.l next_screen,bltdpt(a6)
    move.w #256*64+20,bltsize(a6)
    rts

do_scroll
    move.l pointeur_text,a0 ;fait entrer les lettres
    move.w letter_count,d0 ;dans l'écran caché
    beq new_letter
    sub.w #1,letter_count
    bra scroll

new_letter
    move.w #7,letter_count
    clr.l d0
    move.b (a0)+,d0
    bne continue
    lea text,a0
    move.b (a0)+,d0

continue
    cmp.b #32,d0              ;test pour le character
    bne continue2             ;espace chr=32=$20
    move.b #"Z"+1,d0

continue2
    sub.b #"A",d0              ;A est la base de la table
    lsl.l #1,d0                ;table composée de mot => *2
    lea table_pos,a1
    bsr wait_blit
    move.l #0,a2                ;// calcul adresse lettre
    move.w 0(a1,d0.1),a2        ;// dans image
    add.l #lettres,a2           ;//
    move.l a2,bltapt(a6)
    move.w #36,bltamod(a6)
    move.l #hidden_screen+42,bltdpt(a6)
    move.w #44,bltdmod(a6)
    move.w #09f0,bltcon0(a6)
    move.w #0,bltcon1(a6)
    move.w #32*64+2,bltsize(a6)

scroll
    bsr wait_blit              ;scroll tout le texte
    move.l #hidden_screen+2,bltapt(a6) ;caché
    move.w #2,bltamod(a6)
    move.l #hidden_screen,bltdpt(a6)
    move.w #2,bltdmod(a6)
    move.w #c9f0,bltcon0(a6)
    move.w #0,bltcon1(a6)
    move.w #32*64+23,bltsize(a6)
    move.l a0,pointeur_text
    rts

put_scroll
    move.l #159,d0              ;met scroll dans
    move.l pointeur_onda,a1      ;écran caché
    move.l #hidden_screen+2,a2
    move.l next_screen,a0
    lea table_mask,a4

loop_cut
    move.l a0,a3                ;et on le met 2 colonnes par
    clr.l d1                    ;2 colonnes
    move.w (a1)+,d1

    bne continue_loop
    lea table_sinus,a1
    move.w (a1)+,d1

continue_loop
    add.l d1,a3                ;a3=arrivée, a2=source
    bsr wait_blit
    move.l a2,bltapt(a6)

```

```

    move.w #46,bltamod(a6)
    move.l a3,bltbpt(a6)
    move.w #38,bltbmod(a6)
    move.l a3,bltdpt(a6)
    move.w #38,bltdmod(a6)
    move.w (a4),bltafwm(a6)
    move.w (a4)+,bltalwm(a6)
    move.w #0dfc,bltcon0(a6) ;fait un or pour ne pas
    move.w #0,bltcon1(a6) ;effacer les autres tranches
    move.w #32*64+1,bltsize(a6) ;transfer !
    cmp.w #0,(a4) ;test les masques
    beq change_bloc ;si =0 alors on reprend au debut
    bra next_bloc ;sinon on continue

change_bloc
    lea table_mask,a4
    add.l #2,a0
    add.l #2,a2

next_bloc
    dbf d0,loop_cut

    move.l pointeur_onda,a0 ;fait avancer le pointeur
    add.l #10,a0 ;de l'onda => donne impression
    move.w (a0),d0 ;qu'une onda passe
    bne fin_sinus
    move.l #table_sinus,pointeur_onda
    rts

fin_sinus
    move.l a0,pointeur_onda
    rts

pointeur_text
    dc.l text
letter_count
    dc.w 0
text
    dc.b "SINUS SCROLL POUR ANT",0
even
table_pos
    dc.w 0,4,8,12,16,20,24,28,32,36
    dc.w 1280,1284,1288,1292,1296,1300,1304,1308,1312,1316
    dc.w 2560,2564,2568,2572,2576,2580,2584
pointeur_onda
    dc.l table_sinus
table_sinus
    include Sinus.s
    dcb.l 3,0
table_mask
    dc.w $c000,$3000,$c00,$300,$c0,$30,$c,$3
    dc.w 0

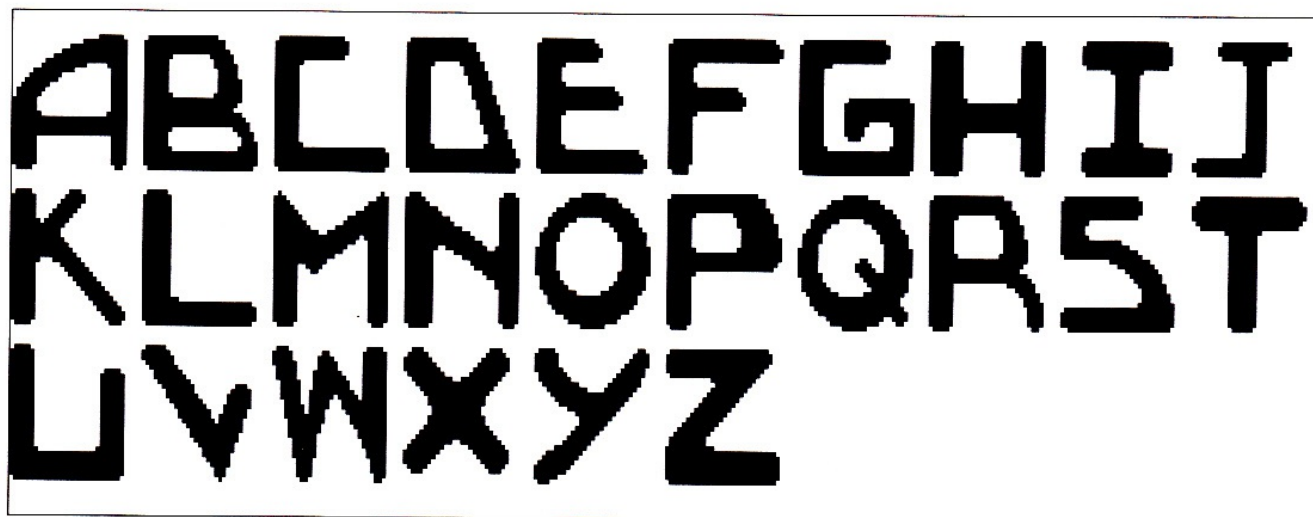
    section écran,data_c
écran1 dcb.b 10240,$0 ;écran de 40*256
écran2 dcb.b 10240,$0 ;écran de 40*256
hidden_screen
    dcb.b 1536,0 ;hidden_screen de 48*32

lettres incbin Font

coplist dc.w color00,0
    dc.w $640f,$fffe
    dc.w color00,$2
    dc.w $8c0f,$fffe
    dc.w color00,$5
    dc.w $b40f,$fffe
    dc.w color00,$9
    dc.w $dc0f,$fffe
    dc.w color00,$d
    dc.w $ff0f,$fffe
    dc.w color00,$f
    dc.l $ffffff

```

Figure 1 : la fonte



Chaque lettre mesure 32x32 pixels.
La taille totale de l'image est 320x256x2.

Par Pierre Chamalet

SUBWAY II : LES LAYERS (fin)

Ce qui suit est le programme que vous auriez normalement du trouver dans le précédent numéro de l'ANT et qui s'est perdu quelque part dans les méandres d'une PAO encore hésitante...

C'est donc avec toutes nos excuses que nous vous le présentons aujourd'hui, accompagné de quelques lignes d'explications que Max a écrites spécialement pour l'occasion. Mais il est grand temps de lui laisser la parole.

Ce programme met en oeuvre trois layers : deux de type SIMPLELAYER et un de type SUPERLAYER, c'est-à-dire utilisant une BitMap qui lui est personnelle, plus grande que le layer lui-même. Une seule partie en est donc effectivement affichée et l'on peut en faire défiler le contenu, histoire de visualiser la BitMap entière. Ce layer est au départ rempli - par le biais de *SetAfPt()* et de *RectFill()* - par un motif multicolore, pour être sûr que le scrolling sera bien visible...

Ce SuperLayer se superpose à un SimpleLayer dans lequel un Quix joue tout seul. Les lignes sont clippées par le système, de sorte qu'elles ne coupent pas le layer supérieur.

Enfin, le troisième layer sert uniquement à afficher un petit texte de bienvenue.

Certes, cette "Little LayersDemo" comme je l'ai appelée ne montre pas toutes les capacités de la layers.library... Il faut en effet savoir que mis à part le clipping en lui-même, la gestion des layers est très lente (déplacement, changement de taille, rafraîchissement...), ce dont on peut de temps en temps se rendre compte au clignotement du SuperLayer et au ralentissement du Quix, d'autant plus que le multitâche est toujours actif. Mais enfin, son but était de démontrer l'utilisation des layers en dehors de la bibliothèque Intuition et de ses fenêtres, et je pense qu'il a été atteint...

```
/* *****\
* Démonstration des Layers sans passer par Intuition.
* Septembre 1991, par Max pour Amiga NewsTech
* Domaine Public.
* *****
* Serait sans doute plus joli en utilisant le double-buffering.
* \*****/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <exec/types.h>
#include <exec/memory.h>
#include <graphics/gfxbase.h>
#include <graphics/gfxmacros.h>
#include <graphics/view.h>
#include <graphics/layers.h>
#include <graphics/clip.h>
```

```
/* Constantes */
#define SALUT1 "Little Layers Demo, par Max pour ANT"
#define SALUT2 "Bouton gauche de la souris pour quitter"
```

```
#define GFXNAME "graphics.library"
#define LAYERSNAME "layers.library"
```

```
#define SCR_W 320 /* Largeur de l'écran */
#define SCR_H 256 /* Hauteur de l'écran */
#define SCR_D 2 /* Profondeur */

#define LAY1_X1 0 /* Position et taille du premier layer */
#define LAY1_Y1 0
#define LAY1_W 320
#define LAY1_H 30
#define LAY1_X2 (LAY1_X1+LAY1_W-1)
#define LAY1_Y2 (LAY1_Y1+LAY1_H-1)

#define LAY2_X1 0 /* Position et taille du second layer */
#define LAY2_Y1 40
#define LAY2_W 320
#define LAY2_H 216
#define LAY2_X2 (LAY2_X1+LAY2_W-1)
#define LAY2_Y2 (LAY2_Y1+LAY2_H-1)

#define LAY3_X1 40 /* Position & taille du troisième layer*/
#define LAY3_Y1 100
#define LAY3_W 240
#define LAY3_H 100
#define LAY3_X2 (LAY3_X1+LAY3_W-1)
#define LAY3_Y2 (LAY3_Y1+LAY3_H-1)

#define SUPER_W 320 /* Largeur de la BitMap du SuperLayer */
#define SUPER_H 256 /* Hauteur de la BitMap du SuperLayer */

/* Prototypes */
#include <proto/exec.h>
#include <proto/graphics.h>
#include <proto/layers.h>

void main(void);
void LayersDemo(void);

BOOL initScreen(void);
void freeScreen(void);

BOOL initLayers(void);
void freeLayers(void);

void doLayer1(void);
void doLayer2(void);
void doLayer3(void);

/* variables globales pour l'affichage graphique */
/* pas besoin de RastPort ici, on utilisera celui des layers */
struct View view, *oldview = NULL;
struct ViewPort vport;
struct RasInfo rinfo;
struct BitMap bmap;
struct ColorMap *cmap = NULL;
USHORT Palette[] = { 0x0000, 0x000f, 0x0f00, 0x0fff };

/* variables globales pour la gestion des layers */
struct Layer_Info *linfo = NULL;
struct Layer *lay1 = NULL;
struct Layer *lay2 = NULL;
struct Layer *lay3 = NULL;

struct BitMap superbmap; /* Ca, c'est pour le SuperLayer */
USHORT motif[] = { /* Un zouli 'A' en 4 couleurs */
    0x0002, 0x0002, 0x0002, 0x0002, 0x0002, 0x0012, 0x0022, 0x0042,
    0x0082, 0x0102, 0x0002, 0x07f2, 0x0802, 0x1002, 0x2002, 0x0000,
    0x0002, 0x0006, 0x000e, 0x001e, 0x003e, 0x007e, 0x00ee, 0x01ce,
    0x038e, 0x070e, 0x0ffe, 0x1ffe, 0x380e, 0x700e, 0xe00e, 0x0000
};
```



```

/* Autres variables globales */
struct GfxBase *GfxBase = NULL;
struct Library *LayersBase = NULL;

/*****
* main()
*****/
void main(void)
{
    if (GfxBase = (struct GfxBase *)OpenLibrary(GFXNAME, 33L))
    {
        if (LayersBase = OpenLibrary(LAYERSNAME, 33L))
        {
            if (initScreen())
                LayersDemo();
            freeScreen();
            CloseLibrary(LayersBase);
        }
        CloseLibrary((struct Library *)GfxBase);
    }
    exit(0);
}

/*****
* initScreen()
*****/
BOOL initScreen(void)
{
    register i;

    /* Alloue et initialise une ColorMap */
    if (!(cmap = GetColorMap(1L << 2)))
        return(FALSE);
    memcpy(cmap->ColorTable, Palette, sizeof(Palette));

    /* Prépare le View */
    InitView(&view);
    view.ViewPort = &vport;

    /* Prépare le ViewPort */
    InitVPort(&vport);
    vport.DWidth = 320;
    vport.DHeight = 256;
    vport.RasInfo = &rinfo;
    vport.ColorMap = cmap;

    /* Prépare la BitMap principale */
    InitBitMap(&bmap, 2, 320, 256);

    /* Alloue les bitplanes */
    for (i = 0; i < 2; i++)
    {
        if (!(bmap.Planes[i] = AllocRaster(320, 256)))
            return(FALSE);
        BltClear(bmap.Planes[i], RASSIZE(320, 256), 1);
    }

    /* Prépare le RasInfo */
    rinfo.BitMap = &bmap;
    rinfo.RxOffset = 0;
    rinfo.RyOffset = 0;

    /* Sauve le View actif (celui d'Intuition) */
    oldview = GfxBase->ActiView;

    /* Construit l'affichage */
    MakeVPort(&view, &vport);
    MrgCop(&view);
    LoadView(&view);
    WaitTOF();

    return(TRUE);
}

```

```

/*****
* freeScreen()
*****/
void freeScreen(void)
{
    register i;

    if (oldview) /* si non null, notre view a été chargé */
    {
        LoadView(oldview);
        WaitTOF();
        FreeVPortCopLists(&vport);
        FreeCprList(view.LOFCprList);
    }

    if (cmap)
        FreeColorMap(cmap);

    for (i = 0; i < 2; i++)
    {
        if (bmap.Planes[i])
            FreeRaster(bmap.Planes[i], 320, 256);
    }
}

/*****
* LayersDemo() - routine principale de démonstration des layers
*****/
void LayersDemo(void)
{
    register UBYTE *ciaapra = (UBYTE *)0xbfe001;

    if (initLayers())
    {
        while (*ciaapra & 0x40)
        {
            doLayer1(); /* Gère le 1er layer */
            doLayer2(); /* Gère le layer quix */
            doLayer3(); /* Gère le SuperLayer */
            WaitTOF();
        }
        freeLayers();
    }

    /*****
    * initLayers() - Crée et initialise les 3 layers
    *****/
    BOOL initLayers(void)
    {
        register SHORT i;

        /* Alloue la structure Layer_Info */
        if (!(linfo = NewLayerInfo()))
            return(FALSE);

        /* Crée le premier layer (bleu) */
        if (!(lay1 = CreateBehindLayer(linfo, &bmap,
            LAY1_X1, LAY1_Y1, LAY1_X2, LAY1_Y2,
            LAYERSIMPLE, NULL)))
            return(FALSE);

        SetRast(lay1->rp, 1);
        SetDrMd(lay1->rp, JAM1);
        SetAPen(lay1->rp, 3);

        Move(lay1->rp, (LAY1_W-(sizeof(SALUT1)-1)*8)/2, 14);
        Text(lay1->rp, SALUT1, sizeof(SALUT1)-1);

        Move(lay1->rp, (LAY1_W-(sizeof(SALUT2)-1)*8)/2, 24);
        Text(lay1->rp, SALUT2, sizeof(SALUT2)-1);

        /* Crée le second layer (blanc) */

```



```

if (!(lay2 = CreateBehindLayer(linfo, &bmap,
    LAY2_X1, LAY2_Y1, LAY2_X2, LAY2_Y2,
    LAYERSIMPLE, NULL)))
    return(FALSE);

SetRast(lay2->rp, 3);
SetDrMd(lay2->rp, JAM1);

/* Crée le troisième layer (rouge) de type LAYERSUPER */
InitBitMap(&superbmap, 2, SUPER_W, SUPER_H);
for (i = 0; i < 2; i++)
{
    if (!(superbmap.Planes[i]=AllocRaster(SUPER_W, SUPER_H)))
        return(FALSE);
    BltClear(superbmap.Planes[i], RASSIZE(SUPER_W, SUPER_H), 1);
}

if (!(lay3 = CreateUpfrontLayer(linfo, &bmap,
    LAY3_X1, LAY3_Y1, LAY3_X2, LAY3_Y2,
    LAYERSUPER, &superbmap)))
    return(FALSE);

SetAfPt(lay3->rp, motif, -4); /* motif 16 lignes de haut */
SetAPen(lay3->rp, 255);
SetBPen(lay3->rp, 0);
SetDrMd(lay3->rp, JAM2);
RectFill(lay3->rp, 0, 0, SUPER_W-1, SUPER_H-1);

return(TRUE);
}

/*****
* freeLayers() - Libère la mémoire des 3 layers et du Layer_Info*
*****/
void freeLayers(void)
{
    register SHORT i;

    if (lay3) DeleteLayer(NULL, lay3);
    if (lay2) DeleteLayer(NULL, lay2);
    if (lay1) DeleteLayer(NULL, lay1);

    for (i = 0; i < 2; i++)
    {
        if (superbmap.Planes[i])
            FreeRaster(superbmap.Planes[i], SUPER_W, SUPER_H);
    }

    if (linfo) DisposeLayerInfo(linfo);
}

/*****
* doLayer1() - Gère le premier Layer
*****/
void doLayer1(void)
{
    /* Rien à faire !!! */
}

/*****
* doLayer2() - Gère le Quix dans le second Layer
*****/
void doLayer2(void)
{
    /* Données statiques pour le Quix */
    static new = 0, old = 0, full = FALSE;
    static SHORT x1[32], y1[32], x2[32], y2[32];
    static SHORT veloc[4] = { -4, 5, 3, -7 };
    static SHORT start[4] = { 110, 25, 160, 74 };
    static SHORT max[4] = { LAY2_W+20, LAY2_H+20,
        LAY2_W+20, LAY2_H+20 };

```

```

    register SHORT i, temp;

    for (i = 0; i < 4; i++)
    {
        temp = start[i] + veloc[i];
        if (temp >= max[i])
        {
            temp = (max[i] * 2) - start[i] - veloc[i];
            veloc[i] = -veloc[i];
        }
        else if (temp < -20)
        {
            temp = -20;
            veloc[i] = -veloc[i];
        }
        start[i] = temp;
    }
    if (full)
    {
        SetAPen(lay2->rp, 3);
        Move(lay2->rp, x1[old], y1[old]);
        Draw(lay2->rp, x2[old], y2[old]);
        old++;
        old %= 32;
    }
    x1[new] = start[0];
    y1[new] = start[1];
    x2[new] = start[2];
    y2[new] = start[3];

    if (new == 31)
        full = TRUE;

    SetAPen(lay2->rp, 0);
    Move(lay2->rp, x1[new], y1[new]);
    Draw(lay2->rp, x2[new], y2[new]);
    new++;
    new %= 32;
}

/*****
* doLayer3() - Gère le scrolling du SuperLayer
*****/
void doLayer3(void)
{
    static SHORT posx = 0, posy = 0, dirx = -2, diry = -1;

    if ((posx <= 0) || (posx >= SUPER_W-LAY3_W-2))
        dirx = -dirx;

    if ((posy <= 0) || (posy >= SUPER_H-LAY3_H-1))
        diry = -diry;

    ScrollLayer(NULL, lay3, dirx, diry);
    posx += dirx;
    posy += diry;
}

/*****
* C'est fini !!!
*****/

```

Par Max 

HARDWARE : LE CLAVIER (II)

*Comme disait mon grand-père,
après la théorie vient la pratique.
Et puisque l'on sait maintenant
tout sur le fonctionnement du
clavier, ne pensez-vous pas qu'il
est grand temps de mettre ces
nouvelles connaissances en
application ? No problemo.*

Le programme qui suit peut paraître démesurément grand ou en tout cas, hors de proportion avec une gestion simple du clavier. Il n'en est rien : la gestion elle-même s'effectue en quelques lignes de code à peine. Ce qui prend de la place, c'est toute la partie d'initialisation. Car pour obtenir un résultat tangible, il fallait que l'on puisse voir à l'écran ce qui se passe lorsque l'on appuie sur une touche. Aussi ai-je dû initialiser un RastPort et une BitMap, ceci afin de pouvoir utiliser la fonction Text() de la graphics.library et donner quelques indications visuelles. De même, la sauvegarde du contexte hardware actuel et sa restauration en fin de programme prennent quelques lignes supplémentaires... Les fainéants, qui sont toujours débrouillards lorsqu'il s'agit de ne pas travailler, pourront éviter de recopier ces lignes, pour ne plus se consacrer qu'à l'essentiel (NDLR : ou bien s'abonner avec disquette !).

L'ESSENTIEL

Pour la partie qui nous intéresse aujourd'hui, l'essentiel se limite à l'initialisation des deux CIAs (le premier pour l'interruption clavier, le second pour le Timer), aux deux routines d'interruptions elles-mêmes (NewIrq2 et NewIrq6), ainsi qu'aux routines GetKey et TestKey.

S'il n'est même pas besoin de rappeler pourquoi l'on utilise le premier CIA (cf. ANT numéro 26), on peut à juste titre se demander pourquoi l'on utilise également le second... Pour, une fois de plus, une raison qui est vraiment toute bête.

En effet, si l'on utilise le premier CIA à la fois pour l'interruption clavier et l'interruption timer, deux cas peuvent se présenter : soit la gestion du clavier est suffisamment rapide et elle se termine avant que l'interruption timer ne se déclenche, auquel cas tout ira bien, soit elle est trop lente et l'interruption timer surviendra pendant l'interruption clavier. Les deux étant de même niveau pour le 68000 (niveau 2), l'interruption timer devra attendre que l'interruption clavier soit terminée avant d'être traitée, d'où une possible perte d'information (le processeur clavier rentrant alors dans une phase de correction d'erreur, re-cf. ANT numéro 26). Le CIA-B quant à lui provoque une interruption de niveau 6 ; en l'utilisant, on est donc sûr qu'elle sera de toute façon immédiatement prise en compte par le 68000, même si elle devait survenir pendant celle de niveau 2. Notez au passage et tout-à-fait dans un autre domaine que pour ces mêmes raisons, il est préférable d'utiliser le CIA-B pour réguler le "stepping delay" du contrôleur de disquette, les interruptions DSKBLK (Disk Block) et DSKSYNC (Disk Synchro) étant respectivement de niveaux 1 et 5...

Le timer est programmé avec la valeur 120, ce qui correspond à (exactement) 85,125 microsecondes sur un Amiga PAL et 85,90 microsecondes sur un Amiga NTSC (cf. cette fois-ci l'ANT numéro 23). Plus précis, tu meurs. On atteint ainsi et quoiqu'il arrive le délai minimum de 85 microsecondes recommandé par les concepteurs de l'Amiga, quelle que soit la version du KickStart et le microprocesseur utilisé (ce programme tourne sans aucun problème sur un A3000/25, ce qui prouve bien qu'il est parfaitement compatible !).

Tiens, en parlant de clavier et de timer, je ne peux m'empêcher d'ouvrir ici une petite parenthèse pour vous narrer une anecdote intéressante... Il est écrit partout dans les RKM que le keyboard.device utilise le Timer A du CIA-A pour gérer le

handshaking du clavier, et tout le monde, y compris votre serviteur, l'a cru et rapporté dans plusieurs articles. Or, il n'en est rien ! Comme on peut s'en rendre compte en traçant la routine d'interruption de niveau 2, ce timer n'est absolument pas utilisé du tout par le système (tout au moins dans sa version 1.3 ; qu'en est-il de la 2.0 ? Mystère et boules de gomme), lequel système se contente de compter sur la vitesse d'exécution de sa routine pour être sûr de retomber sur ses pattes ! Peut-être d'ailleurs est-ce là la raison d'une fourchette aussi importante (il y a loin de 85 us à 145 ms...) : on s'assure ainsi de la compatibilité avec tout type de microprocesseur... Je sais bien que tout le monde s'en fout, mais il me semblait important de révéler la supercherie à la face du monde. Fin de la parenthèse et retour à des considérations nettement plus matérielles.

LA MATRICE CLAVIER

Or donc, ce programme permet d'obtenir indifféremment soit le code de la dernière touche appuyée, soit l'état d'une touche particulière (enfoncée ou non), ce qui permet, ô joie exquise, de gérer le multi-touche.

Le premier s'obtient en appelant la routine GetKey : l'interruption clavier engrange au fur et à mesure qu'elle les récolte, tous les codes des touches appuyées dans un buffer dont la taille a été arbitrairement fixée à 128 octets, et GetKey se contente de les en ressortir en ordre inverse, à raison d'un par appel. Si le buffer est vide, GetKey renvoie -1.

Le remplissage du buffer est donc séquentiel et s'arrête dès que celui-ci est plein. Il existe évidemment de bien meilleurs algorithmes (le "buffer tournant" pour ne citer que lui) mais celui-ci présente l'avantage indéniable d'être à la fois rapide et simple à programmer. Jeu : amusez-vous à programmer une routine ClearInput qui se charge de "vider" le buffer, sans altérer son contenu (une seule ligne de code suffit amplement !).

L'état d'une touche donnée s'obtient en appelant la routine TestKey, le code de la touche à tester étant placé dans d0.W. Au retour, le bit Z de CCR est positionné à 1 si la touche est actuellement appuyée et effacé sinon. La méthode est tout aussi simple, bien que moins courante : on utilise un second buffer, baptisé "matrice clavier", dont chaque bit représente l'état d'une touche (si le bit est mis, la touche est effectivement enfoncée). Ce buffer est évidemment remis à jour à chaque interruption en provenance du clavier.

Comment ça marche ? Les codes touches pouvant être compris entre \$00 et \$7F (les codes \$80 à \$FF étant réservés pour des cas particuliers, comme par exemple le signalement d'une erreur de synchronisation), 128 bits sont nécessaires, soit seulement $(128 / 8) = 16$ octets pour représenter le clavier entier. Le code de la touche concernée sert d'index lors de l'accès à la matrice. Le calcul est très simple : (code / 8) donne le numéro de l'octet dans la matrice et (code MOD 8) donne le numéro du bit dans cet octet. Un exemple concret sera peut-être plus efficace.

Soit à accéder au bit représentant la touche ESC dans la matrice. Le code-clavier de ESC est \$45. On calcule $(\$45 / 8) = 8$ et $(\$45 \text{ MOD } 8) = 5$. La touche ESC est donc représentée par le bit numéro 5 de l'octet numéro 8 de la matrice.

Un autre exemple ? Bon, d'accord, mais c'est bien parce que c'est vous. Juste le temps de modifier une variable, et je suis à vous... *CODE = &H5F : GOTO Exemple.* Voilà, ça y est, merci.

Soit à accéder au bit représentant la touche HELP dans la matrice. Le code-clavier de HELP est \$5F. On calcule $(\$5F / 8) = 11$ et $(\$5F \text{ MOD } 8) = 7$. La touche HELP est donc représentée par le bit numéro 7 de l'octet numéro 11 de la matrice.

Cette technique est très rapide en assembleur, puisqu'une division par 8 du code s'obtient par un décalage logique de 3 bits vers la droite (instruction LSR) et que le modulo par 8 s'obtient par une simple

instruction AND 7 (de même que le modulo par 4 s'obtient par AND 3, que le modulo par 16 s'obtient par AND 15, etc. Essayez, vous verrez bien ! Ca marche avec toutes les puissances de 2).

La routine d'interruption clavier n'a donc plus qu'à positionner (respectivement effacer) le bit de la matrice correspondant à la touche appuyée (respectivement relâchée). TestKey pour sa part se contente de tester le bit correspondant au code touche passé en paramètre. Notez que l'instruction BTST positionne Z à 1 si le bit testé était à 0 et inversement. Il faut donc inverser Z avant de revenir au programme appelant, ce qui se fait très simplement avec l'instruction EOR.LB #-1,CCR (qui n'est pas une instruction privilégiée, c'est EOR to SR qui l'est). On aurait tout aussi bien pu convenir depuis le début que TestKey positionnerait Z à 0 si la touche testée était appuyée, mais ça me semblait beaucoup moins naturel...

Voilà, juste un mot pour finir : ne faites pas attention à la manière dont j'affiche le contenu de la matrice entière dans la boucle principale, les premières inspirations ne sont pas forcément toujours les meilleures... Asta la vista, baby !

```
;
; Exemple de programmation du Clavier.
;
; 1991, Loïc Far pour Amiga NewsTech. Aule Raillete Riserveude.
;
opt      o+,ow-

incdir   "include:"
include  "graphics/rastport.i"
include  "hardware/custom.i"
include  "hardware/cia.i"

include  "exec/exec_lib.i"
include  "graphics/graphics_lib.i"

; *****
CALLSYS MACRO
    IFNC    '','\2'
    movea.l \2,a6
    ENDC
    jsr     _LVO\1(a6)
    ENDM

; *****
Custom EQU    $dff000
CIAA EQU     $bfe001
CIAB EQU     $bfd000
DELAIR EQU    120      ; Delai pour 85 s (en PAL; NTSC=86 s)
KBSIZE EQU    128      ; Taille du buffer clavier

; *****
rsreset
oldcop1 rs.l 1      ; Liste Copper Système 1
oldcop2 rs.l 1      ; Liste Copper Système 2
olddma rs.w 1      ; DMACON Système
oldint rs.w 1      ; INTENA Système
oldirq2 rs.l 1      ; IRQ Level 2 Système
oldirq6 rs.l 1      ; IRQ Level 6 Système
keymat rs.b 16      ; Matrice clavier
keybuf rs.b KBSIZE ; Buffer clavier
keypos rs.w 1      ; Position dans le buffer
VARSIZE rs.w 0

; *****
Start lea VARS(pc),a5

lea gfxname(pc),a1 ; Ouvre la graphics.library
moveq #0,d0
CALLSYS OpenLibrary,$4.w
move.l d0,_GfxBase
beq NoGfx
movea.l d0,a6
move.l $26(a6),oldcop1(a5)
move.l $32(a6),oldcop2(a5)

lea rport(pc),a1 ; Initialise le RastPort
CALLSYS InitRastPort
move.l #bitmap,rport+rp_BitMap

move.l #ecran,d0 ; Initialise la CopperList
move.w d0,CopBpl+6
swap d0
move.w d0,CopBpl+2

lea Custom,a6 ; Sauve la configuration système
move.w dmaconr(a6),olddma(a5)
ori.w #$8200,olddma(a5)
move.w intenar(a6),oldint(a5)
```

```
ori.w    #$c000,oldint(a5)

move.l    $68.w,oldirq2(a5)
move.l    $78.w,oldirq6(a5)

move.w    #$7fff,d0      ; Maintenant, on s'installe !
move.w    d0,dmacon(a6)
move.w    d0,intena(a6)
move.w    d0,intreq(a6)

lea CIAA,a0      ; Prépare le CIA-A :
move.b    #$1f,ciaicr(a0) ; Interruptions off
move.b    #$88,ciaicr(a0) ; Interruption SP on

lea CIAB,a0      ; Prépare le CIA-B :
move.b    ciacra(a0),d0
andi.b    #%11000000,d0
ori.b     #%00001000,d0 ; Timer A mode one-shot
move.b    d0,ciacra(a0)

move.b    #$1f,ciaicr(a0) ; Interruptions off
move.b    #DELAIR,ciatalo(a0) ; Pré-programme le délai :
move.b    #0,ciatahi(a0) ; Le timer démarre aussitôt,
btst      #0,ciaicr(a0) ; donc on attend la fin
beq.s     .wait ; de ce premier décompte.

move.b    #$81,ciaicr(a0) ; Interruption TA on

move.l    #NewCop,copllc(a6) ; Nouvelle CopperList on
move.w    d0,copjmp1(a6)
move.l    #NewIrq2,$68.w ; Irq 2 (CIA-A)
move.l    #NewIrq6,$78.w ; Irq 6 (CIA-B)

move.w    #$83c0,dmacon(a6) ; Copper, Bitplane, Blitter
move.w    #$e008,intena(a6) ; IRQs 2 et 6

; *****
movea.l    _GfxBase(pc),a6 ; Prépare a6

Wait btst    #6,CIAA+ciapra
beq        Fini

moveq      #$63,d0      ; Code clavier de Ctrl
bsr        TestKey      ; La touche est appuyée ?
bne.s      .noCtrl      ; non
moveq      #$45,d0      ; Code clavier de Esc
bsr        TestKey      ; La touche est appuyée ?
beq        Fini         ; Oui, fin avec Ctrl-Esc

.noCtrl bsr    GetKey      ; Touche appuyée ?
bmi.s      .1

lea        buf1+19(pc),a0 ; Oui, affiche son code
bsr        Hex8

.1 lea        rport(pc),a1
moveq      #10,d0
moveq      #20,d1
CALLSYS Move
lea        rport(pc),a1
lea        buf1(pc),a0
moveq      #21,d0
CALLSYS Text

lea        rport(pc),a1 ; Affiche la matrice
moveq      #10,d0
moveq      #40,d1
CALLSYS Move
lea        rport(pc),a1
lea        titre(pc),a0
moveq      #9,d0
CALLSYS Text

lea        keymat(a5),a2
moveq      #0,d2
MatLoop lea        rport(pc),a1
moveq      #10,d0
move.w     rp_cp_y(a1),d1
addq.w     #8,d1
CALLSYS Move

lea        buf2(pc),a0
move.w     d2,d1
bsr.s      Hex4

moveq      #0,d3
moveq      #0,d4
bset       d2,d4
MatLop2 move.b    1(a2,d3.w),d5
lsl.w      #8,d5
or.b       0(a2,d3.w),d5

moveq      #'-',d0
and.w      d4,d5
beq.s      .non
```



```

.non    moveq    #'*',d0
        move.b   d0,(a0)+
        addq.w   #2,d3
        cmpi.w   #16,d3
        blt.s    MatLop2

        lea      buf2(pc),a0
        lea      rport(pc),a1
        moveq    #9,d0
        CALLSYS  Text

        addq.w   #1,d2
        cmpi.w   #16,d2
        blt.s    MatLoop
        bra      Wait

Hex8     move.w   d0,d1
        lsr.b    #4,d1
        bsr.s    Hex4
        move.w   d0,d1
Hex4     andi.b   #$f,d1
        cmpi.b   #9,d1
        ble.s    .1
        addq.b   #7,d1
.1       addi.b   #'0',d1
        move.b   d1,(a0)+
        rts

buf1     dc.b     "Dernière touche : $--"
buf2     dc.b     "x-----"
titre    dc.b     " 01234567"

; *****
Fini     lea      Custom,a6          ; Restaure le système
        move.w   #$7fff,d0
        move.w   d0,dmacon(a6)
        move.w   d0,intena(a6)
        move.w   d0,intreq(a6)

        move.b   #$1F,CIAA+ciaicr
        move.b   #$9F,CIAA+ciaicr
        move.b   #$1F,CIAB+ciaicr
        move.b   #$9F,CIAB+ciaicr

        move.l   oldirq2(a5),$68.w
        move.l   oldirq6(a5),$78.w

        move.l   oldcop1(a5),cop1lc(a6)
        move.l   oldcop2(a5),cop2lc(a6)
        move.w   d0,copjmp1(a6)

        move.w   olddma(a5),dmacon(a6)
        move.w   oldint(a5),intena(a6)

        movea.l   _GfxBase(pc),a1
        CALLSYS  CloseLibrary,$4.w

NoGfx    moveq    #0,d0
        rts

; *****
; Renvoie dans d0.w le code de la dernière touche appuyée
; ou -1 si le buffer clavier est vide.
GetKey   move.w   keypos(a5),d0
        subq.w   #1,d0
        bmi.s    .nokey
        move.w   d0,keypos(a5)
        lea      keybuf(a5),a0
        move.b   (a0,d0.w),d0
        ext.w    d0
.nokey    rts

; *****
; Teste si la touche dont le code est dans d0.w est appuyée
; et positionne Z en conséquence (Z=1 -> touche appuyée)
TestKey  move.w   d0,d1
        lsr.w    #3,d0
        andi.w   #7,d1
        lea      keymat(a5),a0
        btst     d1,(a0,d0.w)
        eori.b   #-1,ccr          ; Inverse (entre autres) Z !
        rts

; *****
; IRQ 2 (CIA-A, clavier)
NewIrq2  movem.l  d0-d2/a0-a1,-(sp)

        lea      Custom,a0
        move.w   intena(a0),d0
        btst     #14,d0          ; Bit INTEN mis ?
        beq.s    Irq2Ret
        and.w    intreqr(a0),d0
        btst     #3,d0          ; Interruption n°3 ?
        beq.s    Irq2Ret

```

```

        lea      CIAA,a0
        move.b   ciaicr(a0),d0
        btst     #3,d0          ; Interruption clavier ?
        beq.s    .ret

        moveq    #0,d0
        move.b   ciasdr(a0),d0  ; Lit la touche reçue
        ori.b    #$40,ciacra(a0) ; Met SP en sortie (hanshaking)
        bset     #0,CIAB+ciacra ; Démarre le Timer A du CIA-B

        lea      VARS(pc),a1
        not.b    d0
        ror.b    #1,d0

        move.w   d0,d1
        andi.w   #$7f,d1
        lsr.w    #3,d1          ; d1 = Code / 8
        move.w   d0,d2
        andi.w   #7,d2          ; d2 = Code MOD 8

        lea      keymat(a1),a0
        bclr     #7,d0
        bne.s    .KeyUp

.KeyDn   bset     d2,0(a0,d1.w)  ; Touche enfoncée : met le bit...
        move.w   keypos(a1),d1  ; et insère le code-touche
        cmpi.w   #KBSIZE,d1    ; dans le buffer clavier.
        bge.s    .ret
        addq.w   #1,keypos(a1)
        lea      keybuf(a1),a0
        move.b   d0,(a0,d1.w)
        bra.s    .ret

.KeyUp   bclr     d2,0(a0,d1.w)  ; Touche relâchée : efface le
        bit.

        .ret     move.w   #$8,Custom+intreq
Irq2Ret  movem.l  (sp)+,d0-d2/a0-a1
        rte

; *****
NewIrq6  movem.l  d0/a0-a1,-(sp)

        lea      Custom,a0
        move.w   intena(a0),d0
        btst     #14,d0          ; Bit INTEN mis ?
        beq.s    Irq6Ret
        and.w    intreqr(a0),d0
        btst     #13,d0          ; Interruption CIAB ?
        beq.s    Irq6Ret

        lea      CIAB,a1
        move.b   ciaicr(a1),d0
        btst     #0,d0          ; Interruption Timer A ?
        beq.s    .ret

        andi.b   #$bf,CIAA+ciacra ; Met SP du CIA-A en sortie

        .ret     move.w   #$2000,intreq(a0)
Irq6Ret  movem.l  (sp)+,d0/a0-a1
        rte

; *****
VARS     ds.b     VARSIZE
_GfxBase dc.l     0
gfxname  dc.b     "graphics.library",0
        even

rport    ds.b     rp_SIZEOF      ; RastPort pour Text()
bitmap   dc.w     40,256,1,0
        dc.l     ecran,0,0,0,0,0,0

; *****
section  CopperList,DATA_C

NewCop   dc.w     diwstrt,$2981,diwstop,$29c1
        dc.w     ddfstrt,$0038,ddfstop,$00d0
        dc.w     bplcon0,$1200
        dc.w     bplcon1,$0000,bplcon2,$0000
        dc.w     bpl1mod,$0000,bpl2mod,$0000
CopBpl   dc.w     bplpt+0,$0000,bplpt+2,$0000
        dc.w     color+0,$0000,color+2,$0fff
CopEnd   dc.l     -2

; *****
section  Screen,BSS_C

ecran    ds.b     $2800          ; écran 320x200, 1 bitplan

; *****
END

```

Par Loïc Far 

Salut à tous ! Avant que de commencer à éplucher votre courrier toujours plus abondant, voici quelques recommandations qui nous passent par la tête, comme ça, sans prévenir.

Dabord et le plus important, il est complètement inutile de joindre une enveloppe timbrée libellée à votre adresse dans l'espoir d'une réponse personnelle : nous n'avons absolument pas le temps de répondre à chacun d'entre vous, même avec toute la meilleure volonté du monde. Faîtes donc des économies, et cessez de faire grossir notre collection de timbres...

Deuxièmement, évitez les questions du style "mon imprimante XYZ ne fonctionne pas sur mon Amiga 500 avec 2 mégas de Ram et un disque dur alors que chez mon copain qui a un 2000, si" (rigolez pas, j'en reçois plein). D'une part, je m'en fous royalement et d'autre part, je n'y puis rien.

Enfin, si vous joignez un programme qui ne marche pas et que vous espérez que l'on déboguera pour vous, n'attendez pas une réponse dans les trois jours... Donnez-nous d'abord le temps de... trouver le temps de nous pencher sur la question et de repérer le bug (en supposant qu'on y arrive !).

Voilà. On terminera en vous rappelant l'adresse à laquelle votre courrier doit nous parvenir (certains trouvent encore le moyen de confondre rédaction et service abonnements) :

Amiga NewsTech - Requester
5, rue de la Fidélité
75010 Paris

Merci de votre attention.

Bonjour. Connaissant le Basic et ayant quelques notions d'assembleur sur une autre machine, je souhaiterais maintenant apprendre l'assembleur sur Amiga. Je dois donc acquérir un certain nombre de logiciels et de livres. Quel est à votre avis la meilleure configuration (assembleur, débogueur, environnement permettant à fois de travailler avec l'assembleur et le débogueur) ? Et quels sont les livres nécessaires (initiation à l'assembleur, dictionnaire des mnémoniques, ouvrages détaillant le jeu d'instructions du 68000 ou le système...) ? Et où puis-je trouver ces livres ?

Benoît Kerdraon, Brest

D'accord, mais c'est la première et la dernière fois que nous répondons à ce genre de lettre : beaucoup de "débutants" sur Amiga sont dans votre cas, et 80% du courrier que nous recevons pose ces mêmes questions. Veuillez donc tous bien noter ce qui va suivre, on ne le répètera pas.

Le meilleur environnement de développement en assembleur est à notre humble avis, on n'a pas cessé de le dire, le Devpac version II de la société anglaise HiSoft : éditeur, assembleur et débogueur travaillent de concert et main dans la main et garantissent un développement rapide et sûr. Toutefois, de petits nouveaux font leur apparition depuis quelques temps, notamment AsmOne de l'allemand DMV-Verlag, pas mal non plus dans son genre. Pour les petites bourses, nous vous conseillons de vous tourner vers le domaine public : A68K, pour ne citer que lui, est un excellent assembleur (qui ne présente qu'un seul défaut, celui d'être incapable de produire un fichier exécutable sans passer par une phase supplémentaire de linkage).

Côté livres, en plus de l'ANT, il faudra (qui a dit hélas ?) vous tourner du côté de Micro-Application : "Bien débiter en Assembleur", "Le livre du langage-machine", "La Bible de l'Amiga"... vous permettront, une fois toutes les erreurs détectées et corrigées, de vous en sortir. Par la suite, si vous désirez vous attaquer plus avant au système multitâche, les "Rom Kernal Manuals" (en 3 volumes) seront indispensables.

Divers ouvrages sur le 68000 lui-même existent ça et là, le meilleur étant - je pense - "Mise en oeuvre du 68000" par Christine Viellefond

aux éditions Sybex. Sachez également que le mode d'emploi du Devpac contient un guide du MC 68000, véritable petit bijou que tout programmeur en assembleur se devrait de posséder.

Chère ANT, je me permets de vous écrire afin que vous puissiez apporter un peu de lumière au pauvre programmeur que je suis. Je voudrais connaître le moyen d'appliquer un programme systématiquement après que toutes les tentatives de recherche ont échouées et ce à partir de n'importe quel CLI. Par exemple, on tape "bonjour" et si le programme bonjour n'existe pas (ou bien s'il n'est pas trouvé dans le path courant), ce programme par défaut s'exécute pour afficher, par exemple, "tu t'as gouru." Je cherche aussi à encapsuler un programme, comme pour IF, ELSE, ECHO... du Shell d'ARP (ASH), mais pas à la manière du CSH, c'est-à-dire un programme lancé, qui prend la main et utilise ses propres fonctions pour les exécuter et donc avec aucune modification possible. Ceci dans le but de créer une sorte de Shell qui serait adaptable au plaisir de chacun, mais qui prendrait moins de place sur disquette et éviterait de jouer au grille-pain pour ceux qui n'ont qu'un seul lecteur. Quelles sont les limites de Forbid() et Disable(), dans quelle mesure peut-on les utiliser sans risquer un effondrement du système ?

Quelques remarques avant de vous quitter. D'abord, quelques erreurs se sont glissées dans l'article de Loïc Far sur le MC 68000 : l'instruction MOVE générale se code comme suit :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	SIZE	ADRESSE_EFFECTIVE				EFFECTIVE_ADRESSE								

où SIZE égale 01 pour un octet, 11 pour un mot et 10 pour un mot long, ADRESSE_EFFECTIVE est codée comme expliqué dans le premier article, ainsi que EFFECTIVE_ADRESSE, avec toutefois inversion des tribits, c'est-à-dire Registre (3 bits) + Code-Op (3 bits).

De plus, certaines instructions n'ont pas tous les modes d'adressage. Mais sinon, l'article était bien fait, et je le remercie de l'avoir écrit.

Olivier Garrigues, Lunel

Si l'on excepte le fait qu'ANT est masculin (on dit un Amiga NewsTech, hé oui, qui l'eût cru ?), votre lettre est exemplaire. Laissez-nous donc vous remercier comme vous le méritez.

Merci donc, et tout d'abord pour cette longue lettre elle-même, dont nous n'avons évidemment pu publier que quelques passages. Merci également pour les rectifications apportées à l'article de Loïc Far. Comme quoi personne n'est jamais parfait...

Désolé, mais nous ne sommes pour l'instant pas en mesure de répondre à vos questions sur le PATH et l'encapsulation de programmes... Avec quelques recherches, la première pourra sans doute être rapidement résolue (nous vous tiendrons au courant)... Quant à la seconde, je crains qu'il ne vous faille renoncer à cette idée. Une solution acceptable, aux résultats identiques, serait de profiter de la possibilité de résidence de quasiment toutes les commandes CLI : une fois en mémoire, elles y restent, et ce sont les mêmes segments de code qui sont réutilisés à chaque appel...

Forbid() interdit le multitâche, c'est-à-dire qu'Exec ne gère plus le partage du temps CPU entre les différentes tâches actives, gestion qui intervient à chaque interruption, lesquelles sont d'ailleurs toujours autorisées. Car c'est justement Disable() qui se charge de les interdire - et donc le multitâche avec. Le problème, c'est que certaines tâches particulières du système, comme le keyboard.device et le trackdisk.device, comptent énormément sur la disponibilité d'autres sous-tâches qui leur sont rattachées... Ainsi par exemple, si vous travaillez en mode multitâche et appelez Fobrid() ou Disable() pendant un accès disque, des erreurs de lecture peuvent survenir. Par contre, il n'y a aucune problème à interdire les interruptions, même pendant une très longue période, si votre programme s'approprie la machine entière (démon, jeu...). Le retour au système (si retour au système il y a...) ne posera aucun problème.

Par Ok et Cancel


```

End If
End If
O=P
Next
Centre At(,27)+"Y="+FONC$
Centre At(,29)+">>> Pressez une touche <<<"
Clear Key : Wait Key
Loop
Procedure _GET_VALUE[N#,A$]
Do
' On met l'ancienne valeur dans le buffer clavier
Put Key Str$(N#)
' On imprime le texte
Print A$; : Line Input " : ";B$
' Si pas de reponse -> la fin!
If B$="" : _THE_END : End If
' On essaie de tokeniser
_TOKENISE[B$] : Exit If Not SYNT
' Un erreur, on boucle!
Print "On enlève les mouffles, et on retape l'expression.
Merci."
Loop
' On calcule la belle valeur de retour
EVALUE
End Proc[Param#]
Procedure _THE_END
Edit
End Proc
'-----
' PROCEDURES DE TOKENISATION
Procedure EVALINIT
' Definition des operateurs, en ordre inverse
OPE$="+PLUS-MINS*TIME/DIVD^POWR"
' Definition des fonctions
FUN$="{X}VARX{PI}PIPI{VAR{ }VARS2{SIN{ }SINU1{COS{ }COSI1"
FUN$=FUN$+"{TAN{ }TANG1{SQR{ }SQRR1{LOG{ }LOGG1{EXP{ }EXPP1"
FUN$=FUN$+"{LN{ }LNN1{ABS{ }ABSS1{INT{ }INTT1{SGN{ }INTT1"
FUN$=FUN$+"{RND{ }RNDD1"
End Proc
Procedure _TOKENISE[A$]
Procedure _TEVALUE[N]
Procedure _TOPERAND
Procedure CHAR
'-----
' PROCEDURES DE CALCUL
Procedure EVALUE
PP=-1 : PPILE=-1 : SYNT=0 : EVALIT
End Proc[PP#(PP)]
Procedure EVALIT
On Error Goto _ERROR
Repeat
Inc PPILE : Gosub PILE$(PPILE)
Until E
Pop Proc
END: E=1 : Return
_CONST: Inc PP : PP#(PP)=PILE$(PPILE) : Return
_PLUS: Dec PP : PP#(PP)=PP#(PP)+PP#(PP+1) : Return
_MINUS: Dec PP : PP#(PP)=PP#(PP)-PP#(PP+1) : Return
_DIVD: Dec PP : PP#(PP)=PP#(PP)/PP#(PP+1) : Return
_TIME: Dec PP : PP#(PP)=PP#(PP)*PP#(PP+1) : Return
_POWR: Dec PP : PP#(PP)=PP#(PP)^PP#(PP+1) : Return
_SINU: EVALIT : PP#(PP)=Sin(PP#(PP)) : Return
_COSI: EVALIT : PP#(PP)=Cos(PP#(PP)) : Return
_TANG: EVALIT : PP#(PP)=Tan(PP#(PP)) : Return
_PIPi: Inc PP : PP#(PP)=Pi# : Return
_SQRR: EVALIT : PP#(PP)=Sqr(PP#(PP)) : Return
_LOGG: EVALIT : PP#(PP)=Log(PP#(PP)) : Return
_LNND: EVALIT : PP#(PP)=Ln(PP#(PP)) : Return
_ABSS: EVALIT : PP#(PP)=Abs(PP#(PP)) : Return
_EXPP: EVALIT : PP#(PP)=Exp(PP#(PP)) : Return
_INTT: EVALIT : PP#(PP)=Int(PP#(PP)) : Return
_RNDD: EVALIT : PP#(PP)=Rnd(PP#(PP)) : Return
_VARX: Inc PP : PP#(PP)=XX# : Return
_ERROR: Resume _ERROR2
_ERROR2: PP#(PP)=0.0 : SYNT=-1 : Return
End Proc

```

Par François Lionet

```

' Calcul du pas de progression des sinus
STP#=2*Pi#/NPAS
'
' Boucle de fabrication de la deformation
For N=0 To NPAS-1
P#=P#+STP#
Screen 1 : Cls 0 : Screen 2 : Cls 0
' Deformation en X, ecran 0 -> ecran 1
For Y=0 To SY-1
X=AMPX1#*Sin(Y*AMPY1#-P#)
Screen Copy 0,0,Y,SX,Y+1 To 1,X,Y
Next
' Deformation en Y, ecran 1 -> ecran 2
For X=0 To SX-1
Y=AMPY2#*Sin(X*AMPX2#-P#)
Screen Copy 1,X,0,X+1,SY To 2,X,Y
Next
' On peut copier dans la banque
Copy Logbase(0),Logbase(0)+SIZE To Start(10)+N*SIZE
Next
' Efface les ecrans de travail
Screen Close 1 : Screen Close 2
Screen 0 : Cls 0
' Calcule la distance entre l'ecran 0 et la banque
TL=SX/8
D=Start(10)-Logbase(0)
OY=D/TL : OX=(D mod TL)*8
' Met les valeurs dans les registres AMAL
Amreg(0)=OX : Amreg(1)=OY : Amreg(2)=SY : Amreg(3)=NPAS
' Initialise l'animation
Channel 0 To Screen Offset 0
A$=A$+"L: For R0=0 To RD;"
A$=A$+" Let X=RA;"
A$=A$+" Let Y=R0*RC+RB;"
A$=A$+" Next R0;"
A$=A$+" Jump L;"
' On peut démarrer et revenir en mode direct!
Amal 0,A$ : Amal On
Direct
Procedure BONE[X,Y,SX,SY,C]
Bar X-SX,Y-SY To X+SX,Y+SY
For R=1 To C
Circle X-SX,Y-SY,R
Circle X-SX,Y+SY,R
Circle X+SX,Y-SY,R
Circle X+SX,Y+SY,R
Next
End Proc

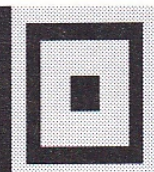
```

Par François Lionet

Tiens, un blanc...
(ou plutôt un noir...)

Et si on en profitait de
ces petits carrés pour
caser de la pub,
ça vous dérangerait
beaucoup ?

Pensez aux
conséquences et
donnez-nous votre avis!



Transactor

Règlement et Proposition d'Article

- 1.) La rubrique *Transactor* de l'*Amiga NewsTech* est la rubrique écrite par les lecteurs, pour les lecteurs. Son but est de leur permettre de s'exprimer librement sur tout sujet qui les passionne, dans quelques langage de programmation que ce soit. Seule restriction : la Rédaction doit avoir la possibilité de tester le bon fonctionnement de tous les programmes fournis, complets ou simples routines d'illustration. Non mais.
- 2.) Tout lecteur désirant participer à la rubrique *Transactor* devra envoyer son article sur disquette au format Amiga, en respectant les quotas définis aux paragraphes 4 et 5. Les disquettes ne seront pas retournées, sauf demande expresse et écrite de l'auteur, et encore.
- 3.) Tout article publié sera rémunéré au tarif forfaitaire de 1, 500 F HT les deux pages ; un article de plus de deux pages sera publié en plusieurs fois, à concurrence de trois épisodes maximum. Le paiement a lieu le mois suivant celui de la parution. Les auteurs des articles retenus pour parution seront directement prévenus par téléphone.
- 4.) Les articles doivent parvenir au format ASCII standard (utilisez un éditeur de textes ou l'option de sauvegarde "text only" ou "ASCII" de votre traitement de texte). Les programmes illustrant les articles, qu'ils soient complets ou de simples routines d'exemple, doivent également être fournis sous forme ASCII, et éventuellement en format "brut" le cas échéant (*Basic*, *AMOS*...). Le(s) fichier(s) exécutable(s) doivent se trouver sur la disquette. L'auteur doit fournir à l'ANT la possibilité de recompiler ou d'interpréter ses programmes sans difficulté (joindre au besoin une version "run-only" du langage utilisé).
- 5.) La taille totale du fichier ASCII donne le nombre de signes de votre article ; une page de l'ANT contient en moyenne 7, 000 signes de texte. Une colonne de listing contient 80 lignes de 65 caractères de large. Un programme de 160 lignes occupe donc une page entière. Les programmes en assembleur particulièrement longs peuvent être publiés en trois colonnes de 40 caractères chaque (240 lignes par page).
- 6.) Toute proposition d'article pour *Transactor* devra être accompagnée du bon ci-dessous, découpé ou photocopié, mais en aucun cas reproduit à la main.
- 7.) Toute proposition d'article pour *Transactor* entraîne l'acceptation par l'auteur que les programmes joints soient placés sur la disquette d'accompagnement de l'ANT. L'auteur doit préciser dans des commentaires en tête de listing s'il place son programme dans le Domaine Public ou s'il désire conserver son copyright. Aucun commentaire implique la mise en Domaine Public.
- 8.) Ni l'ANT, ni *Commodore Revue SARL*, ni les auteurs collaborant à la rubrique *Transactor* ne sauraient être tenus pour responsables en cas de dommages quelconques survenus à l'ordinateur, suite à une mauvaise utilisation des documents (textes et programmes) publiés dans cette rubrique.
- 9.) Toute proposition d'article pour *Transactor* entraîne l'entière acceptation par l'auteur de ce règlement, et sans discuter.

Abonnement à L'ANT et Anciens Numéros

C'est nouveau, ça vient de sortir : si vous êtes déjà abonné, vous pourrez parrainer des copains à vous. Le deal est simple : vous nous trouvez cinq nouveaux z'amis et vous empochez un digitaliseur sonore PerfectSound offert par CIS (merci, CIS). De plus, tout le monde recevra un logiciel de téléchargement ChanelSoft gratuit. Alors qu'est-ce que vous attendez ?

Vous pouvez également commander les anciens numéros de l'ANT, histoire de compléter votre collection. Il vous en coûtera 45 F pièce ou 60 F avec sa disquette. Y'a un bon de commande là-dessous, servez-vous en.

Bon de commande à découper ou photocopier (pas de recopie manuelle SVP) et à retourner correctement rempli et accompagné de votre règlement à **MCM Europe, 16, Quai Jean-Baptiste Clément, 94140 Alfortville, France.**

OUI, je m'abonne à *Amiga NewsTech*. Je choisis la formule :

- ☐ 1 an (11 numéros) sans disquette 350 FF
- ☐ 1 an (11 numéros) avec disquette 600 FF
- ☐ (cochez cette case pour un ré-abonnement)

Je commande les anciens numéros suivants :

- ☐ n°20 (☐ avec disquette) ☐ n°25 (☐ avec disquette)
- ☐ n°21 (☐ avec disquette) ☐ n°26 (☐ avec disquette)
- ☐ n°22 (☐ avec disquette) ☐ n°27 (☐ avec disquette)
- ☐ n°23 (☐ avec disquette)
- ☐ n°24 (☐ avec disquette)

au tarif unitaire de 45 F (60 F avec disquette), pour un total de F.

Nom :
Prénom :
Adresse :
Code Postal : Ville : Pays :

Signature (des parents pour les mineurs) :