DOSSIER

AMSTRAD

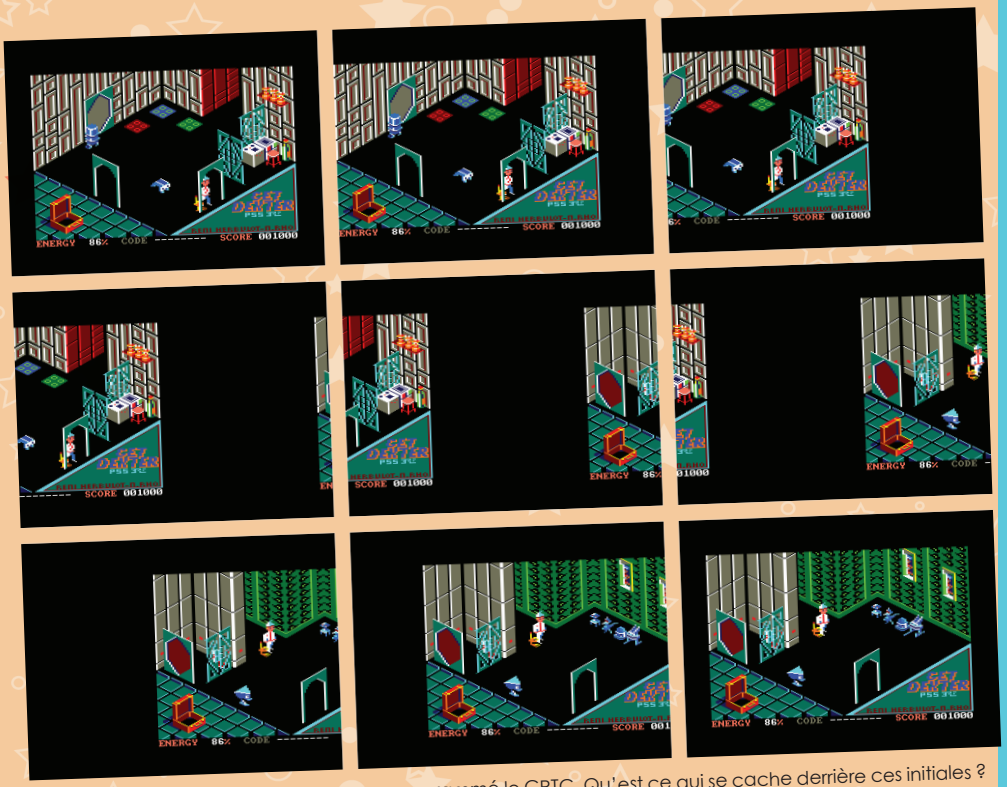
Le croco sort les crocs
CPC : Effets sur l'écran

Pourquoi est-ce que j'aborde ce sujet ?

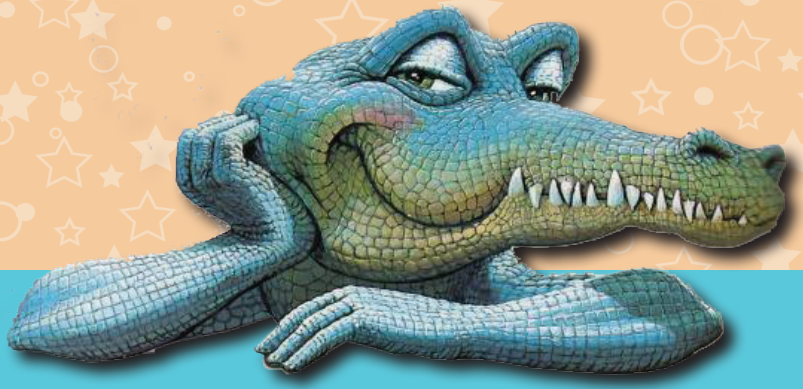
Nous avons vu précédement avec l'article sur le Multi-Mode dans COTE GAMERS n°2 que Rémi HERBULOT, le programmeur du cultisime «Crafton & Xunk / Get Dexter» (1986) avait utilisé des techniques avancées pour l'époque et aujourd'hui nous allons profiter d'un des effets présents dans ce jeu pour parler du CRTC.

Petit rappel, lorsqu'on change de pièce dans «Crafton & Xunk / Get Dexter», la zone de jeu se déplace vers la gauche et réapparaît par la droite de notre moniteur pour se replacer en plein milieu de celui-ci. Le programmeur a profité qu'une partie de la zone de jeu soit masquée pour afficher le nouveau décor au fur et a mesure. Très malin et très agréablement visuellement.

Les 9 captures écrans ci-dessous montrent parfaitement la technique utilisée et surtout que nous avons bien un bout des 2 pièces (celle d'avant à gauche et celle a venir à droite) sur les captures 5 et 6.



Pour réaliser cette prouesse, l'auteur a programmé le CRTC. Qu'est ce qui se cache derrière ces initiales ?

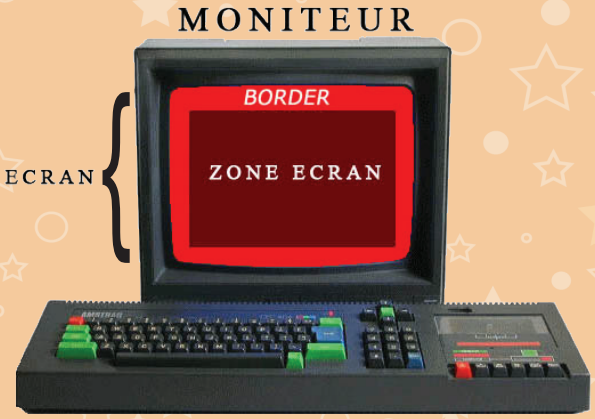


Pour changer l'affichage du border, il faut changer sa couleur uniquement (déjà abordé dans Côté Gamers n°1 et 2).

Pour changer l'affichage de la zone écran, il suffit de modifier la mémoire qui lui correspond et la couleur des pixels qui le composent (par défaut, la zone écran, correspond à la zone mémoire de &C000 à &FFFF soit 16 Ko de RAM).

En conclusion, l'ECRAN («ZONE ECRAN» + «BORDER») est géré par le CRTC et le «MONITEUR», c'est l'objet qui affichera notre image.

Cathode Ray Tube Controler, c'est le circuit 6845 qui contrôle la génération des signaux vidéo et par conséquent la forme de l'affichage sur notre MONITEUR en définissant deux ensembles :
- le BORDER qui est une zone où seule la couleur du border est affiché (Display Enable désactivé)
- la ZONE ECRAN qui correspond à une zone mémoire qui est affiché (Display Enable actif)



LES DIFFERENTS TYPE DE CRTC

Il existe plusieurs «type» de circuit CRTC :
- CRTC de type 0, référence «HD6845S» fabriqué par Hitachi et qui a équipé les Amstrad CPC 464, CPC 664 et KC Compact (c'est la version allemande du CPC 6128) entre 1984 et 1986.

- CRTC de type 0, référence «UM6845» fabriqué par Unicorn Microelectronics (UMC), a équipé essentiellement les Amstrad CPC 6128 entre 1985 et 1987.

- CRTC de type 1, référence «UM6845R» fabriqué par Unicorn Microelectronics (UMC), pour les Amstrad CPC 6128 entre 1988 et 1990.

- CRTC de type 2 (considéré comme un mauvais CRTC à cause des soucis de compatibilité avec les types 0 et 1), référence «MC6845» fabriqué par Motorola Semiconductor Products Inc et référence «UM6845S» fabriqué par Unicorn Microelectronics (UMC)

Il existe 2 autres types de CRTC, mais ils sont émulés via la puce ASIC et c'est pour ça que nous ne retrouvons pas le numéro «6845» dans la référence :
- CRTC de type 3, référence «40226» fabriqué par Amstrad et qui équipe la gamme des Amstrad CPC 464 Plus, CPC 6128 Plus et console GX 4000 de 1990 à 1993.

- CRTC de type 4 aussi appelé Pré-Asic, «AMS40489» fabriqué par Amstrad et qui a été utilisé sur les derniers CPC 6128. Le classement par type a été réalisé par les démomakers au fil de l'eau, et il faut savoir que le Pré-Asic a été découvert après l'ASIC, d'où l'inversion dans la numérotation.

Petite parenthèse : L'ASIC (Amstrad CPC Plus / GX4000) est un composant spécialisé. Quand une

production se mesure en plusieurs centaines de milliers d'exemplaires, le moindre centime compte (c'était le crédo de Sir Alan Michael SUGAR). Il est possible de rassembler plusieurs composants en 1 seul, le résultat permet d'obtenir une carte moins grande avec beaucoup moins de composants discrets pour les alims (condensateurs de découplages...). D'où des problèmes de compatibilité également car seules les fonctionnalités de bases ont pu être reproduites.

Soucis de compatibilité entre les types de CRTC ?
- Suivant les CRTC les valeurs par défauts ne sont pas toutes les mêmes.
- Certains registres peuvent être en lecture uniquement, en écriture uniquement ou les deux à la fois, mais varient suivant le type de CRTC :
Exemple, R12 et R13 sont en lecture/écriture pour les CRTC de type 0, 3 et 4, mais en écriture uniquement pour les CRTC de type 1 et 2.
- R0 : On peut avoir des effets différents suivant les types de CRTC !, en fait, c'est le nombre de lignes qui offre des effets variés suivant les moniteurs (Cf les démos de Chany qui passent sur certain moniteur mais pas sur d'autres).

- Le CRTC de type 2, suivant les paramètres dans le registre 2 peux planter l'affichage de votre CPC surtout si l'on n'a plus de VBL il sera très dur de reprendre la main.

Comment sous Basic détecter un type de CRTC (Astuces fournie par Tom & Jerry) ?
Type 0 :

```
BORDER 0:OUT &BC00,6:OUT &BD00,0
```

Si on voit une ligne bleue discontinue sur le moniteur, c'est un type 0.

Type 2 :

```
OUT &BC00,2:OUT &BD00,52
```

S'il n'y a plus rien sur le moniteur (synchro verticale perdue), c'est un type 2.

Il est facile de répérer un type 4 en regardant l'arrière de la machine au niveau du connecteur «EXPANSION».
Si on ne voit pas un Z80 à cet endroit, le CPC a une carte mère avec un pré-Asic.

Du coup, si la machine n'a pas les comportements attendus avec les tests précédents, c'est un type 1 ou un CPC Plus.

On pourrait éventuellement dresser une liste des types de machines (avec les pastilles au dos de la machine sur lesquelles sont imprimées une lettre), mais personne n'a encore fait une liste exhaustive des différents modèles existants. Accessoirement, cela ne serait pas fiable à 100%, certaines machines ayant pu voir leurs CRTC remplacés (suite à une panne ou par un démomaker exigeant).

Autant vous l'annoncer tout de suite, les deux premiers tests sous émulateurs sont inefficaces. L'émulation du CRTC même encore aujourd'hui, reste la bête noire des programmeurs d'émulateurs.

VCC, HCC, VBL, HBL, VSYNC, HSYNC

Nous avons besoin de regarder ce qui se cache derrière les 6 sigles pour mieux comprendre le fonctionnement du CRTC.

VCC et HCC :

Vertical **C**haracter **C**ount et **H**orizontal **C**haracter **C**ount. En gros, le CRTC n'est rien d'autre qu'un ensemble de compteurs qui sont incrémentés sur certains évènements. Deux de ces compteurs comptent les caractères : HCC le nombre de caractère par ligne, et VCC le nombre de lignes de caractère (à noter que VCC est en fait incrémenté lorsque VLC a atteint son total maximum et VLC est le compteur de lignes tout court)

VBL et HBL :

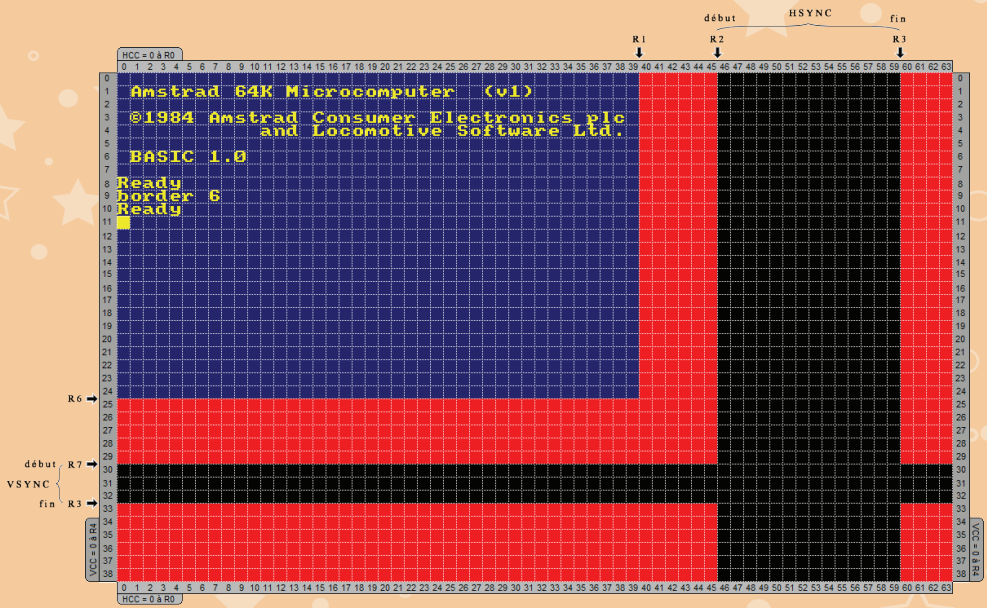
Vertical **B**lank et **H**orizontal **B**lank. Le faisceau électronique, lorsqu'il arrive en bout de ligne, va s'éteindre le temps de revenir au début de la ligne suivante (histoire d'éviter d'afficher n'importe quoi). C'est ce petit temps qui est appelé HBL. Même principe en bas du moniteur avec la VBL, on éteint le canon le temps de remonter en haut du moniteur.

VSync et HSync :

Permettent au CRTC de se synchroniser avec l'affichage sur le moniteur. Pour schématiser, lorsque le moniteur arrive en bout de ligne, il attend le signal Horizontal de Synchronisation (HSync) pour retourner au début de la ligne. Ce temps d'attente n'est pas infini : Ca explique que, si on n'envoies pas de signal de synchronisation, on vois tout de même quelque chose sur le moniteur. En bas de l'affichage, le moniteur attend également ce signal avant de remonter tout en haut. Ca explique beaucoup de bug d'affichage qui roulent : Si le signal arrive trop tôt ou trop tard, la synchronisation de fait est dans les «choux», et on génère des affichages qui défilent verticalement ou qui sont tout brouillés dans le cas de HSync.

Avant d'attaquer la lecture du tableau contenant les différents registres du CRTC, je vous conseille de jeter un oeil sur le schéma suivant, qui contient un affichage de démarrage du CPC 464 avec une bordure rouge (pour bien séparer visuellement les zones BORDER et ECRAN) et de vous y reporter au fur et a mesure de votre lecture.

A noter que l'intersection de R2 et R7 correspond au coin haut gauche de votre moniteur.



Les registres du CRTC

Ils sont au nombre de 21. Ce circuit possède un port bidirectionnel de 8 bits.

Deux adresses de port suffisent pour le programmer.
Le PORT &BCxx pour sélectionner le registre sur lequel on veux travailler.
Le PORT &BDxx pour écrire la valeur à envoyer dans le registre sélectionné.
Petit moyen mémotechnique pour se rappeler qui fait quoi : Connecter, Données

Deux adresses supplémentaires de ports sont disponibles pour lire les valeurs, mais dépendent du type de CRTC présent, les PORT &BExx et PORT &BFxx (Seuls les registres 12 à 17 sont en lecture. Et encore, ça dépend des CRTC).
Je vous l'accorde, tout ceci n'a pas l'air très digeste, et peux paraître très abstrait, nous allons éclaircir tout ça après avoir listé les différents registres utilisables et y avoir ajouté quelques commentaires :

REGISTRES		INFORMATIONS
FORMAT HORIZONTAL	R0	Nombre de caractères total maximum en horizontal : 0 à 255 La valeur par défaut est 63 (&3F) Conditionne le temps attribué au rayon pour balayer l'écran dans la largeur. Donc, si R0=0, on aura tout de même UNE ligne (vu que la comparaison est faite sur la ligne courante : VCC = R0 = 0 se fait en première ligne) Donc, 63 dans la valeur de registre se traduit par 64 lignes affichées (0 à 63).
	R1	Nombre de caractères maximum affichés en horizontal : 0 à 255 La valeur par défaut est 40 (&28) Si la valeur dépasse 63, alors c'est la premiere ligne qui sera affiché sur tout l'écran, mais pourquoi ? On change de ligne (au niveau du calcul de l'adresse à afficher en mémoire) lorsque HCC = R1, mais HCC est remis à 0 si HCC = R0 Donc, si R1>R0, on ne va jamais changer la ligne au niveau de l'adresse mémoire, ce qui explique que l'on n'affiche QUE la première ligne sur tout l'écran. Pour simplifier : NOMBRE DE COLONNES TEXTE VISIBLES (en se basant sur du MODE 1), engendre une déformation de l'affichage du contenu à l'écran.
	R2	Synchronisation horizontale (position) : 0 à 255 La valeur par défaut est 46 (&2E) Pour simplifier : POSITION HORIZONTALE DE L'ECRAN Si on change les valeurs en cours de balayage de l'écran, cela donnera l'effet de vagues (utilisé dans des démos).
	R3	Longueur de synchronisation 0 à 255 La valeur par défaut est 142 (&8E)

REGISTRES		INFORMATIONS
FORMAT VERTICAL	R4	Nombre de lignes total maximum en vertical : 0 à 127 La valeur par défaut est 38 (&26) Conditionne le temps attribué au rayon pour balayer l'écran sur toute la hauteur.
	R5	Synchronisation verticale : 0 à 31 La valeur par défaut est 0 (&00). Permet de modifier la fréquence du balayage écran
	R6	Nombre de caractères maximum affichés en vertical : 0 à 127 La valeur par défaut est 25 (&19). Attention : Cette valeur doit être inférieure au R4. Pour simplifier : NOMBRE DE LIGNES TEXTE VISIBLES
	R7	Synchronisation verticale (position) : 0 à 127 La valeur par défaut est 30 (&1E). Pour simplifier : POSITION VERTICALE DE L'ECRAN

REGISTRES		INFORMATIONS
R8		Mode entrelacé : 0 a 3. La valeur par défaut est 0 (&00 - No interlace) Si on utilise comme valeur 1 (&01 - Interlace Sync Raster Scan Mode), alors l'écran vibrera. Si on utilise comme valeur 3 (&03 - Interlace Sync and Video Raster Scan Mode), l'écran sera divisé en deux, les lignes paires en haut, les lignes impaires en bas. Mais comme l'amstrad CPC double l'affichage des lignes, nous avons 2 fois l'image a l'écran. Chose étrange, si on utilise la valeur 240 (&F0), il n'y aura plus que le BORDER de visible sur tout l'écran.
R9		Scanning 0-31. La valeur par défaut est 7 (&07) 0 à 7 nous donne 8 valeurs, ça correspond au nombre de lignes composants la hauteur d'un caractère.
CURSEUR	R10	Ligne de départ du scanning du curseur 0-31 La valeur par défaut est 0 (&00)
	R11	Ligne de fin du scanning du curseur 0-31 La valeur par défaut est 0 (&00)

REGISTRES		INFORMATIONS
ADRESSE DE BASE ET TAILLE DE LA MEMOIRE VIDEO	R12 R13	R12 est l'octet le plus significatif (poids fort), sa valeur par défaut est 48 (&30). R13 est l'octet le moins significatif (poids faible), sa valeur par défaut est 0 (&00). Grâce à ces deux registres ont peux déplacer la mémoire écran et aussi gérer sa taille qui peux aller de 16 ko (standard) a 32 ko (mais vu la taille de l'écran nous ne verrons qu'un peu plus de 24 ko sur les 32) : En fait si nous tenons compte des 2 registres, nous jouons sur 2x8bits, donc 16 bits (pour être précis, c'est 14 bits, car les bits 14 et 15 ne sont pas utilisé et seront donc toujours à zéro). Nous raisonnons au niveau BINAIRE, bit à bit.

R12								R13							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	Page		Taille		Offset									

Taille : Page mémoire :
&X00 : 16ko &X00 : &0000...
&X01 : 16ko &X01 : &4000...
&X10 : 16ko &X10 : &8000...
&X11 : 32ko &X11 : &C000...

Quelques exemples :
Ecran de 16Ko :
R12=00 (&00 - &X00000000) de &0000 a &3FFF
R12=16 (&10 - &X00010000) de &4000 a &7FFF
R12=32 (&20 - &X00100000) de &8000 a &BFFF
R12=48 (&30 - &X00110000) de &C000 a &FFFF
Le fait de basculer la zone écran avec une autre adresse permet de voir le contenu de celle-ci à l'écran et cette technique a été utilisée pour créer des animations fluides en basculant rapidement d'un écran à un autre (Surtout utilisé dans des démos).

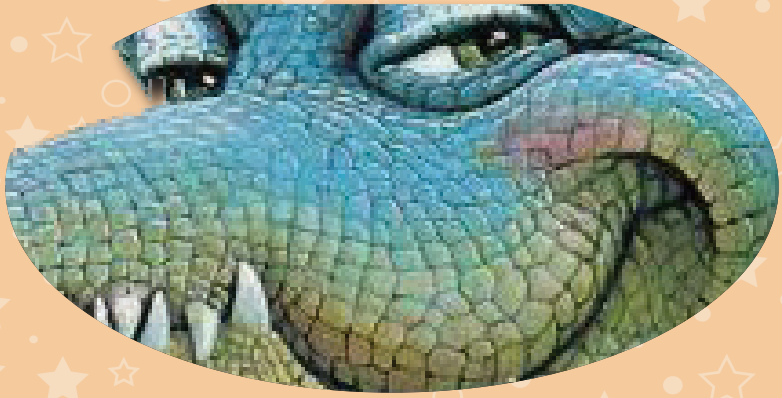
Ecran de 32Ko (2x 16Ko) :
R12=12 (&0C - &X00001100) de &0000 a &7FFF
R12=28 (&1C - &X00011100) de &4000 a &BFFF
R12=44 (&2C - &X00101100) de &8000 a &CFFF
R12=60 (&3C - &X00111100) de &C000 a &3FFF

Grâce à l'OFFSET, il est possible de décaler l'adresse de début de l'écran de plusieurs octets dans un sens ou dans un autre, ce qui peut aussi permette de créer un scrolling hard très fluide.

REGISTRES		INFORMATIONS
CURSEUR	R14 R15	Position du curseur : R14 est l'octet le plus significatif (poids fort), sa valeur par défaut est 0 (&00). R15 est l'octet le moins significatif (poids faible), sa valeur par défaut est 0 (&00). Bien entendu les R12 et R13 influencent la position du curseur.
LIGHTPEN	R16 R17	Position du Light Pen (Crayon Optique, Pistolet Phaser...) : R16 est l'octet le plus significatif (poids fort), sa valeur par défaut est 0 (&00). R17 est l'octet le moins significatif (poids faible), sa valeur par défaut est 0 (&00).
	R18 R19	Non supporté par tous les types de CRTC et parfois, même pas câblé.
	R11	Non supporté par tous les types de CRTC et parfois, même pas câblé. La lecture de ce registre via le port &BFxx donnera : Type 0 et 2 = 0 Type 1 : Hi-Z ([1]) Type 3 et 4 : R15

([1]) Z = impédance ; Hi-Z = haute impédance ; C'est une valeur quelconque de haute impédance qui varie selon l'électronique voire depuis le moment dont est allumé le CPC. Elle n'est donc pas toujours la même, donc impossible de se baser dessus.

R16 et R17 :
Vous ne rêvez pas, l'Amstrad CPC était prévu pour supporter un crayon optique ou un pistolet.
Nous sommes très très loin de l'utilisation du stylet sur les consoles Nintendo DS (vive les écrans tactiles) ou encore de la wiimote pour les jeux de tirs.
Le concept était assez révolutionnaire pour l'époque sur l'Amstrad CPC, bien entendu, cette époque peut être considéré par beaucoup comme la préhistoire, mais il faut bien un commencement pour que les techniques et le matériel s'améliorent.
Le crayon optique se branche sur le port joystick, et la pointe de ce dernier doit être posé sur écran cathodique. L'appuie sur la barre d'espace, va engendrer un flash de couleur blanche à l'écran (plutôt aveuglant et surement très gênant pour les personnes épileptyque) pour permettre au CPC de repérer la position où vous êtes située.
Pour les jeux de tir avec les pistolets, vous deviez être très très près de votre écran, et là aussi nous avions le système de flash de couleur blanche et autre soucis, les jeux étaient prévus pour un seul type de pistolet, aucune compatibilité entre eux, plutôt rageant lorsqu'on y pense. Les jeux ayant supportés cette technique ne sont pas légions, actuellement nous en avons 23 dans notre base de données concernant la gamme des CPC 464/664/6128 (3 types de pistolet : Phaser de Loriciels, Magnum Light Phaser de Virgin Games, Gunstick de MHT Ingenieros) et seulement 2 pour les CPC 464 Plus/6128 Plus (Uniquement le pistolet de la société Trojan). Mais ils ont eux au moins le mérite d'exister et de lancer la mode.



Format vidéo sur Amstrad CPC

A l'époque, le format vidéo est le PAL, soit 625 lignes à 50Hz (25 images entrelacées par seconde de 312 lignes). C'est pour ça que nous avions du scanline a l'écran (1 ligne sur 2 d'affichées), même si aujourd'hui sous émulateur nous avons pris l'habitude d'afficher les lignes cachés et au final nous avons des images avec des couleurs beaucoup plus vive qui ne refléte pas du tout la réalité.
50Hz pour être compatible avec le courant du secteur.
L'Amstrad CPC affiche une image de 312 lignes à 50Hz, ce qui fait un raster de 64µs (R0+1).
La fréquence d'horloge du CRTC à 1MHz permet juste de garantir que R0+1 fait 64µs (en clair, chaque ligne est affichée en 64 microsecondes).

Pour connaître la largeur en pixel affichées sur le moniteur :
En MODE 1 = (R0+1) * 8
R0=63+1=64 * 8 = 512 pixels

Pour connaître le nombre de lignes affichées sur le moniteur : (R4+1) * (R9+1) + R5
R4=38+1=39 ; R9=7+1=8 ; R5=0 donc 39 * 8 + 0 = 312 lignes (ce qui confirme les informations au dessus)

La fréquence du CRTC est de 1MHz (1000000 Hz)
A partir de là nous pouvons recalculer le taux de rafraichissement :
64 microsecondes * 312 lignes = 19968 microsecondes par écran affiché.
1000000 Hz / 19968 microsecondes par écran = 50,08 Hz.

La vraie image avec scanline (un mauvais ratio a l'affichage et nous avons de drôle de couleurs)

L'image sous émulateur avec remplissage de toutes les lignes (au niveau de la luminosité et du rendu, c'est le jour et la nuit).

Essais en BASIC sur le CRTC - les mains dans le cambouis

Si vous vous rappelez le premier article dans Côté Gamers sur les modes graphiques, vous savez donc que le MODE 1 sur un écran standard a une résolution de 320 * 200 (on ne compte pas la bordure).
Mais le CRTC lui est capable de gérer beaucoup plus comme nous venons de le calculer au dessus.
Reprenons nos calculs.
Une zone écran standard en MODE 1 fait 300 * 200 pixels
Un moniteur CPC n'est capable d'afficher que : 384 * 280 pixels (ECRAN + BORDER)
Et d'après la configuration de base du CRTC, nous pouvons avoir du 512 * 312 pixels
Nous pouvons en déduire que nous avons 128 pixels de cachés * 32 lignes de cachées.
Tout ça pour dire que si ça n'avait pas été le cas, l'astuce utilisée par Rémi HERBULOT n'aurait pas fonctionnée de la même façon, et lors du décalage de la zone écran à gauche nous aurions tout de suite eu l'apparition de celui-ci à droite (merci aux pixels cachés), alors qu'une très grosse partie de la zone écran disparaît devant nos yeux ébahis.

Beaucoup d'informations, je dirai même beaucoup trop d'informations.
Pour éclaircir tout ça, nous allons réaliser quelques petits essais très simple en BASIC.

Notre mission, va consister a modifier un registre très facilement avec affichage de la valeur.

Le code BASIC ci-dessous est assez simple, mais je vais tout de même apporter quelques explications.

GOSUB ligne permet de faire un saut dans un sous-programme en allant à la ligne indiquée. Dès que le CPC trouve la commande **RETURN**, ça lui signale la fin du sous-programme et il reprends sa lecture juste après le GOSUB ligne.

OUT &BC00,registre vous remarquez qu'on l'utilise qu'une fois, à partir du moment où on est branché sur un registre, le CPC sait qu'on travaille sur celui-ci, donc nous n'avons pas besoin d'écrire «OUT &BC00,registre:OUT &BD00,valeur» à chaque fois (en même temps dans mon exemple, nous ne travaillons que sur un seul registre, ce qui évite cette répétition inutile).

PEN LIGNE AND 3 J'ai utilisé une technique de fainéant, vu qu'on utilise du MODE 1, nous n'avons sous basic que 4 couleurs (encre 0 a 3 incluse), j'aurai pu utiliser une variable ENCRE, que j'incrémente a chaque chagement de la variable LIGNE et que je remet a zéro lorsqu'on dépasse la valeur 3 (version longue), mais j'ai choisit d'utiliser **ligne AND 3** qui permet de convertir la valeur de ligne (1 a 25) en 0 a 3 (très pratique et très court) :

Version longue	Version courte
encre=0 FOR LIGNE=1 TO 25 PEN encre encre=encre+1 IF encre >3 THEN encre=0 ... NEXT	FOR LIGNE=1 TO 25 PEN ligne AND 3 ... NEXT

PRINT STRING$(40,CHR$(127)); La fonction STRING$ permet d'afficher x fois une chaîne de caractères (ici, j'utilise la fonction **CHR$(code_ascii)** pour afficher le caractère 127).
A noter que le point virgule «;» à la fin de la ligne a toute son importance, le programme traduit ça par le fait que nous allons afficher le prochain texte à la suite du précédent, sans ça, nous afficherions nos informations en ligne 1, 3, 5, 7, 9... mais pourquoi ?
On commence en ligne 1, on affiche nos 40 caractères, du coups nous nous retrouvons en début de ligne 2, lorsque le programme va rencontrer le prochain PRINT, il va faire un saut de ligne, et on va se retrouver en ligne 3 pour l'affichage de nos 40 caractères... Une seule lettre placée au bon endroit fait parfois toute la différence.

```
IF INKEY(0)=0 THEN valeur=valeur+1:GOSUB 120
```

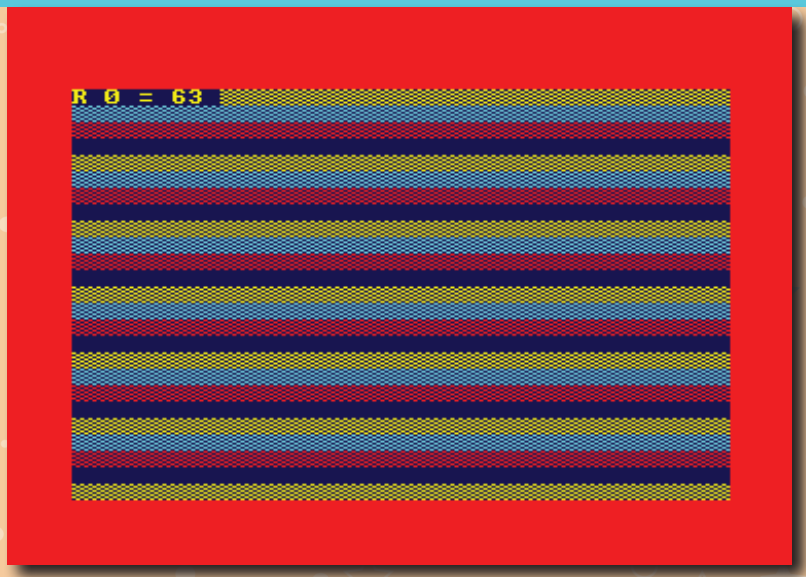
Si le test de la touche numéro "0" (c'est la flèche curseur haut - chaque touche de votre clavier a un numéro qui lui est propre) est égal a zéro (ça veux dire que vous avez enfoncé cette touche) alors on augmente notre valeur de 1 et on fait un saut dans le sous-programme situé en ligne 120.

La commande INKEY permet d'interroger le clavier tous les cinquantièmes de seconde. Voici les valeurs qui peuvent être retournées :

```
-1      la touche n'est pas enfoncée
 0      la touche est enfoncée
32      SHIFT + la touche est enfoncée
128     CTRL + la touche est enfoncée
160     SHIFT + CTRL + la touche est enfoncée
```

Vous n'êtes pas obligé de saisir ce qui est en vert et qui symbolise des remarques. (*Listing R0.BAS*)

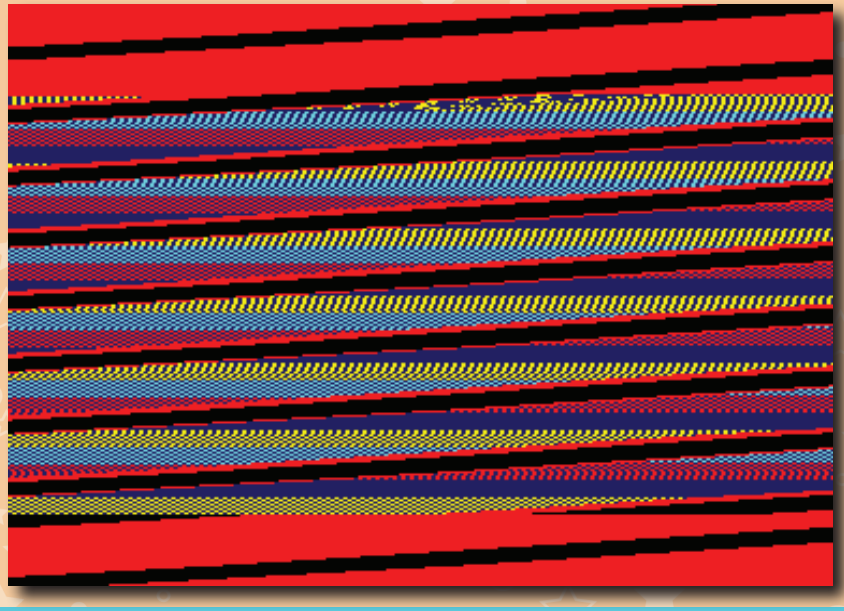
```
10 MODE 1:'efface l'ecran et positionne celui-ci en 4 couleurs
20 CALL &BC02:'charge les couleurs par defaut
30 BORDER 6:'le border va etre rouge
40 GOSUB 180:'sous programme - remplir l'ecran
50 registre=0:'le registre sur lequel on va travailler
60 valeur=63:'valeur par defaut du registre
70 OUT &BC00,registre:'selectionner le registre
80 GOSUB 150:'afficher le registre et sa valeur en haut a gauche de l'ecran
90 IF INKEY(0)=0 THEN valeur=valeur+1:GOSUB 120:'test curseur haut
100 IF INKEY(2)=0 THEN valeur=valeur-1:GOSUB 120:'test curseur bas
110 GOTO 90:'on boucle sur le test des touches
120 IF valeur <0 THEN valeur=0:'ne pas aller en dessous de zero
130 IF valeur>255 THEN valeur=255:'ne pas depasser 255 comme valeur
140 OUT &BD00,valeur:'envoyer la valeur au CRTC
150 LOCATE 1,1:PRINT "R";registre;"=";valeur
160 RETURN
170 'remplir l'ecran du caractere 127
180 FOR LIGNE=1 TO 25:'25 fois
190 PEN LIGNE AND 3:'stylo 0 a 3 par rapport a la valeur de la variable ligne
200 PRINT STRING$(40,CHR$(127));:'afficher 40 fois le caractere ASCII 127
210 NEXT:'ligne suivante
220 RETURN
```



Comme vous l'avez déjà compris, il suffira de changer les valeurs en ligne 50 et 60 pour tester les différents registres de 0 a 9

```
50 registre=0:'le registre sur lequel on va travailler
60 valeur=63:'valeur par defaut du registre
```

Suivant les registres utilisés et les valeurs envoyées (en plus ça varie suivant le type de CRTC, histoire que ça soit un peu plus compliqué), ça devient très vite une véritable catastrophe sur notre écran : Ecran noir, écran qui saute, affichage démultiplié... une vraie horreur.
Tout ceci est à manipuler avec une grande précaution, surtout si vous avez oublié de sauvegarder votre programme.



Nous allons écrire un deuxième petit programme en BASIC (il ne faut jamais rester sur un échec), mais cette fois-ci nous allons travailler sur seulement 4 registres (R1, R2, R6 et R7) en précisant la limite maximum pour éviter les soucis.

les touches curseur permettent de déplacer l'écran et SHIFT+les touches curseur modifient le nombre de lignes et de colonnes visibles sans jamais dépasser la limite que nous aurons bien entendu fixée.

Nous allons utiliser la technique des tableaux de données pour gérer nos valeurs.
Le tableau R(x) pour la valeur par défaut des 4 registres que nous allons modifier.
Dans Rmax(x) nous aurons la limite supérieure a ne pas dépasser.

Pour ceux qui ne connaissent pas la notion de tableau(Table) en BASIC, voici un petit résumé :

Les tables contiennent des éléments de même nature auxquels nous accédons par un indice ou rang. L'avantage, c'est que nous pouvons parcourir les données très facilement en utilisant des boucles (FOR...NEXT ; WHILE...WEND).
Pour déclarer un tableau a une dimension qui devra contenir 7 éléments au maximum nous devons écrire :

```
DIM nomdutableau(7)
```

A noter que si nous ne déclarons pas de tableau, par défaut le basic initialisera un tableau avec 10 éléments. Par contre, si nous déclarons à nouveau un tableau, nous aurons le droit au message d'erreur suivant «ARRAY ALREADY DIMENSIONNED».
Pour des entiers, je vous conseille de penser à rajouter %, chaque entrée ne prendra plus que 2 octets en mémoire au lieu de 5 : `DIM nomdutableau%(7)`

Pour écrire une valeur dans le tableau : `nomdutableau(indice)=valeur`
Nous remplacons «indice» par la valeur du rang dans le tableau, et valeur par la valeur à donner.
Par exemple, si l'on veux écrire au rang 1, la valeur 40, nous le ferons comme suit : `nomdutableau(1)=7`
Si nous écrivons dans un rang non défini dans notre tableau, par exemple `nomdutableu(20)=7` nous aurons le droit au message d'erreur suivant «SUBSCRIPT OUT OF RANGE». Notre tableau a été déclaré avec 7 indices, et ici nous voulons écrire dans l'indice 20, d'où le fait qu'on nous signale qu'on est en train d'essayer d'écrire en dehors des limites.
Il faut en conséquence définir dès le début la bonne taille pour notre table, sinon, c'est la catastrophe assurée.
Il est possible d'effacer un tableau avec la commande `ERASE nomdutableau`.

Imaginons un tableau R qui posséderait les 4 valeurs suivantes :

```
R(1)=40:R(2)=46:R(6)=25:R(7)=30
```

Indice ou Rang	Table R0 Valeur
1	40
2	46
3	
4	
5	
6	25
7	30

J'aborde très rapidement la notion de tableaux à deux dimensions.
Déclaration `DIM nomdutableau(ligne,colonne)`
Ecriture `nomdutableau(ligne,colonne)=valeur`
Supression : ERASE `nomdutableau`

Imaginons `DIM nomdutableau(3,4)`
On va le remplir avec cette méthode :

```
FOR i=1 TO 3:FOR j=1 TO 4:x(i,j)=i*j:NEXT j: NEXT i
```

	Colonne 1	Colonne 2	Colonne 3	Colonne 4
Ligne 1	1	2	3	4
Ligne 2	2	4	5	8
Ligne 3	3	6	9	12

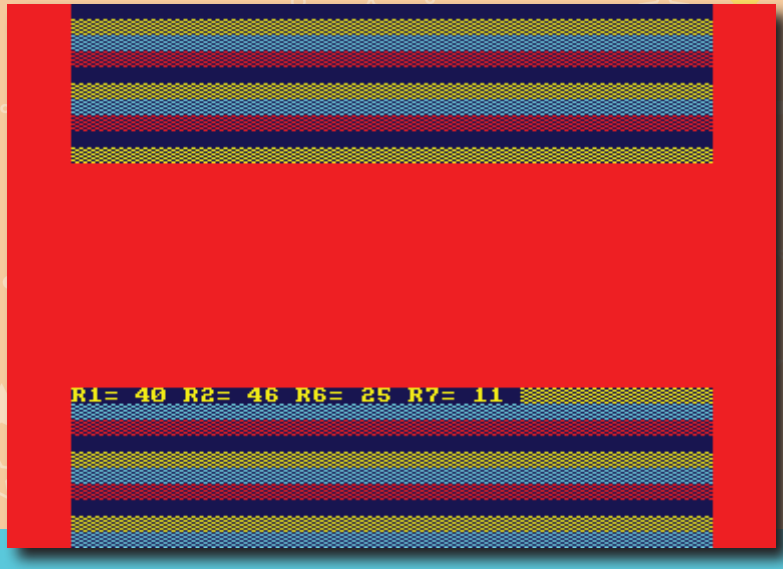
Pour afficher le résultat de nos opérations à l'écran :

```
FOR i=1 TO 3:FOR j=1 TO 4:PRINT i;",";j;"=";x(i,j):NEXT j: NEXT i
```

Revenons à nos moutons et par conséquent à notre deuxième essai de petit programme sur le CRTC. (*Listing R1267.BAS*)

```
10 MODE 1:CALL &BC02:BORDER 6
20 DIM R%(7):'definir la taille du tableau R
25 R(1)=40:R(2)=46:R(6)=25:R(7)=30:'Valeurs par défauts
30 DIM Rmax%(7):'definir la taille du tableau Rmax
35 Rmax(1)=63:Rmax(2)=63:Rmax(6)=39:Rmax(7)=38:'Valeurs maximum
40 GOSUB 170:'sous programme - remplir l'ecran
90 IF INKEY(0)=0 THEN x=7:y=+1:GOSUB 120:'curseur haut
91 IF INKEY(2)=0 THEN x=7:y=-1:GOSUB 120:'curseur bas
92 IF INKEY(8)=0 THEN x=2:y=+1:GOSUB 120:'curseur gauche
93 IF INKEY(1)=0 THEN x=2:y=-1:GOSUB 120:'curseur droite
94 IF INKEY(0)=32 THEN x=6:y=-1:GOSUB 120:'shift+curseur haut
95 IF INKEY(2)=32 THEN x=6:y=+1:GOSUB 120:'shift+curseur bas
96 IF INKEY(8)=32 THEN x=1:y=-1:GOSUB 120:'shift+curseur gauche
97 IF INKEY(1)=32 THEN x=1:y=+1:GOSUB 120:'shift+curseur droite
110 GOTO 90:'on boucle sur le test des touches
120 REM traitement des donnees
122 R(x)=R(x)+y
124 IF R(x)<0 THEN R(x)=0
126 IF R(x)>Rmax(x) THEN R(x)=Rmax(x)
130 OUT &BC00,x:'selectionne le registre
140 OUT &BD00,R(x):'envoyer la valeur au CRTC
150 LOCATE 1,1:PRINT "R1=";R(1);"R2=";R(2);"R6=";R(6);"R7=";R(7);
160 RETURN
170 'remplir l'ecran avec le caractere 127
180 FOR LIGNE=1 TO 25:'25 fois
190 PEN LIGNE AND 3:'stylo 0 a 3 par rapport a la valeur de la variable ligne
200 PRINT STRING$(40,CHR$(127));:'afficher 40 fois le caractere ASCII 127
210 NEXT:'ligne suivante
220 RETURN
```

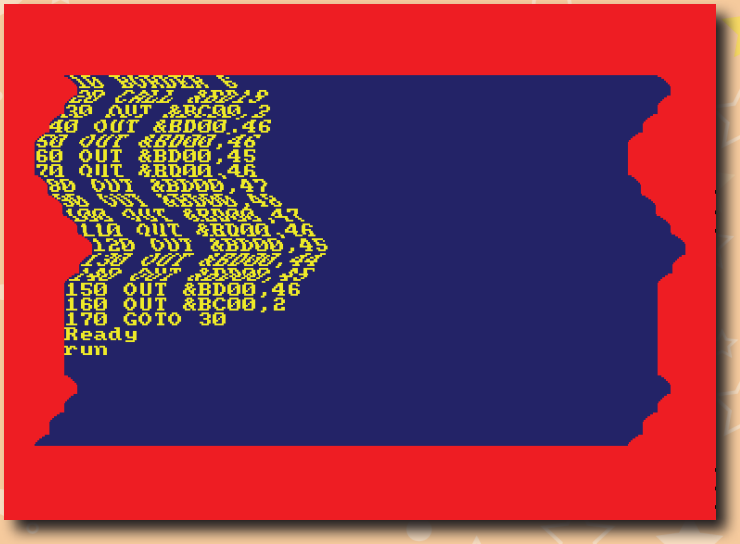
Plutôt incroyable, ça fonctionne et nous n'avons même pas planté l'affichage de notre Amstrad CPC et on découvre que la zone écran peux se balader n'importe où sur notre moniteur.
Ce qui était le haut de notre écran se retrouve en bas, et le bas vers le haut (petit délire pour vous montrer la puissance du CRTC).



R2 - Vague

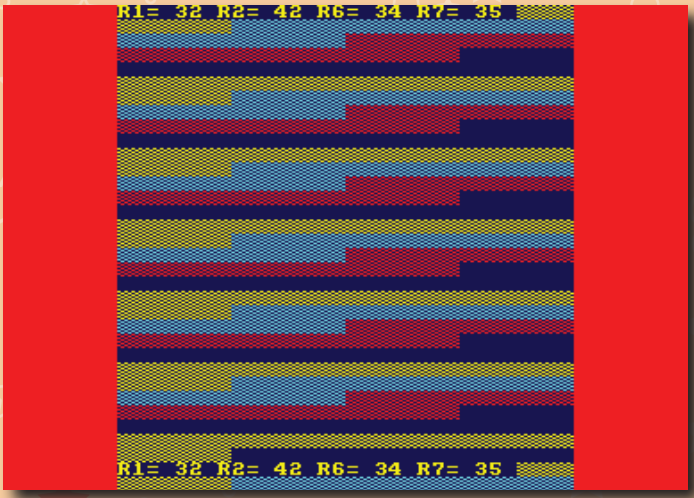
Avant de continuer nous allons réaliser une petite vague toute moche en BASIC en jouant sur le R2 du CRTC.
On voit bien la gestion de la «**P**hase **L**ock **L**oop» (l'acronyme **PLL** est assez courant) du moniteur sur la HSync fluctuante.
Pour ce qui est des jeux qui utilisent ce mode, il faut chercher dans ceux qui ont des scrollings horizontaux fluides parce qu'ils utilisent une gestion au demi octet et non a l'octet, mais ceci est une autre histoire. (*Listing VAGUE.BAS*)

```
10 BORDER 6
20 CALL &BD19
30 OUT &BC00,2
40 OUT &BD00,46
50 OUT &BD00,46
60 OUT &BD00,45
70 OUT &BD00,46
80 OUT &BD00,47
90 OUT &BD00,48
100 OUT &BD00,47
110 OUT &BD00,46
120 OUT &BD00,45
130 OUT &BD00,44
140 OUT &BD00,45
150 OUT &BD00,46
160 OUT &BC00,2
170 GOTO 30
```



Ce test est très simpliste, et vous devrez le programmer en assembleur si vous souhaitez pouvoir gérer ce genre d'effet uniquement sur une petite partie de l'écran.

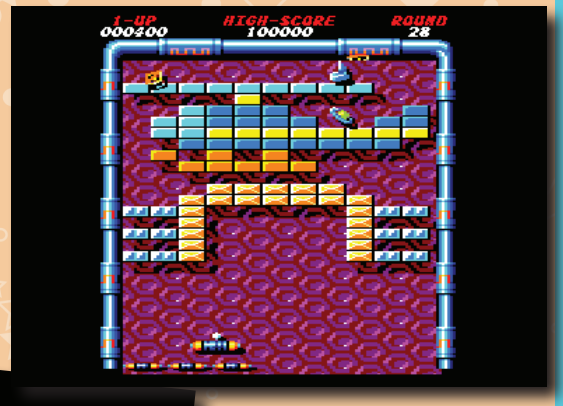
Le mot «OVERSCAN» (en dehors du scan) a été très utilisé dans le milieu du CPC que ce soit par les revues informatiques ou les démomakers même si on peux considérer celui-ci comme une faute de language, le terme «BORDERLESS» (sans bordure) serait plus approprié. Nous allons utiliser les 2 termes pour plus de facilité.



OVERSCAN VERTICAL ou VERTICAL BORDERLESS

On continue a changer les valeurs des 4 registres pour essayer d'obtenir ce qu'on appelle un «overscan vertical» ou «vertical borderless».
J'ai choisi 32 colonnes et ce n'est pas le fruit du hasard. Par contre pour le nombre de ligne, j'ai essayé de remplir tout le moniteur et on s'aperçoit que la ligne R1=32 R2=42 R6=34 R7=35 se répéte. La zone écran ne fait que 16 Ko, en temps normal le registre R1=40 colonnes et R6=25 lignes, ce qui pour simplifier les choses nous donnent 40*25=1000 cases texte de visible.
Or dans notre test, nous avons R1=32 colonnes et R6=34 lignes, 32*34 = 1088 cases texte de visible, on comprend tout de suite qu'on dépasse les limites autorisées d'où la répétition des informations dans la zone écran. La solution serait de modifier les registres R12 et R13 pour travailler avec un écran 32 Ko, nous verrons ça dans «OVERSCAN PLEIN ECRAN» ou «TOTAL BORDERLESS».

Les programmeurs ont très vite vu l'intérêt de jouer sur les registres du CRTC pour les jeux nécessitant une zone de jeu plus haute que large.
Voici deux exemples :

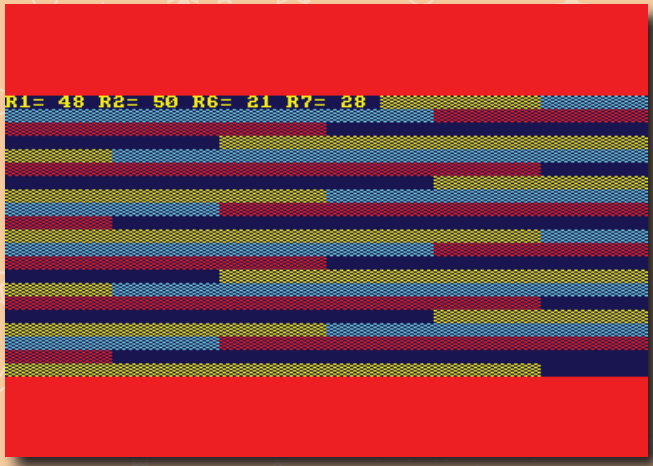


«Arkanoid Revenge Of Doh» 1988
© Ocean Software
R1=32 colonnes * R6=32 lignes = 1024

«Donkey Kong» 1986
© Ocean Software
R1=32 colonnes * R6=32 lignes = 1024

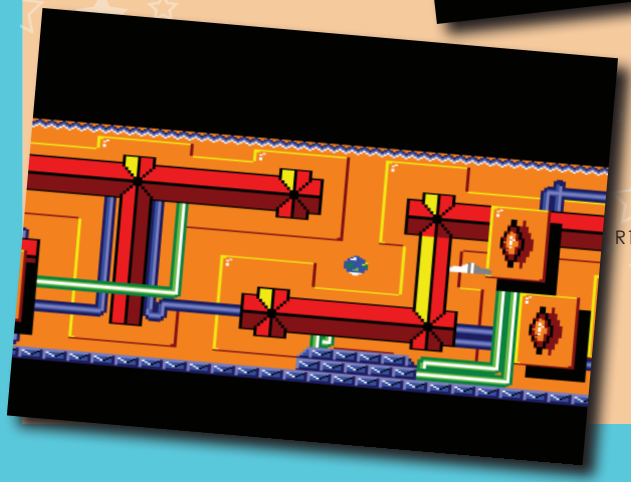
OVERSCAN HORIZONTAL ou HORIZONTAL BORDERLESS

On modifie les valeurs des 4 registres pour essayer d'obtenir cette fois-ci ce qu'on appelle un «overscan horizontal» ou «horizontal borderless», j'ai volontairement réduit le nombre de lignes a 21 pour ne pas dépasser la limite d'affichage et ainsi éviter la répétition des données à l'écran. R1=48 colonnes * R6=21 lignes soit 1008 cases textes de visible. Si vous avez bien suivit, nous avons dépasser les 1000, mais il faut savoir qu'en temps normal, nous avons 384 octets qui ne sont pas visible à l'écran dans un affichage dit «standard», du coups nous allons utiliser une partie de ses octets cachés.



Ce formatage d'écran a été très peu utilisé, dommage, car c'était une bonne idée, même si nous nous retrouvons avec 2 grosses bandes noires au dessus et en dessous de la zone écran de jeu. De plus les deux jeux ci-dessous posséde un scrolling du décor très fluide.

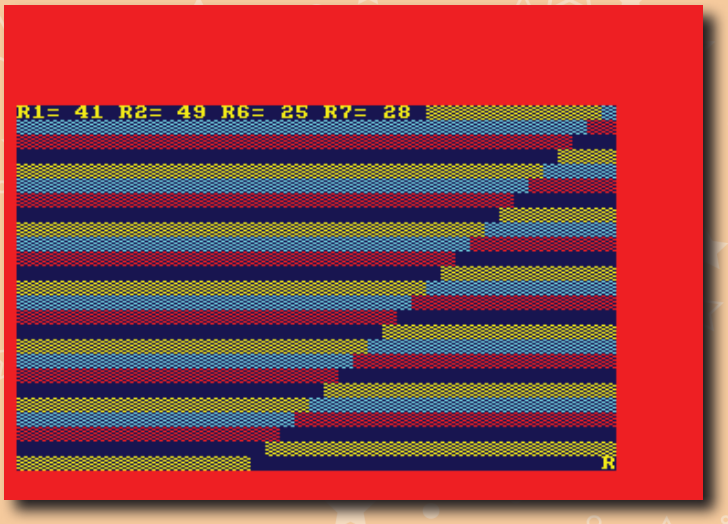
«Out Of This World» 1987 © Ariolasoft
R1=51 colonnes * R6=20 lignes = 1020

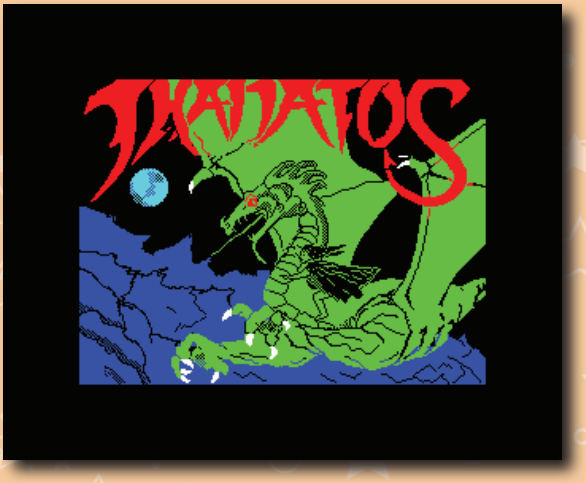


«Jinks» 1988 © Rainbow Arts
R1=51 colonnes * R6=20 lignes = 1020

ÉCRAN NORMAL, JUSQU'A QUEL POINT ?

Comment connaître la véritable limite à l'écran ?
Nous avons 16 Ko, c'est a dire 16384 octets (16 ko * 1024 octets) pour un écran dit «standard», si on divise ça par 40 colonnes et 25 lignes, ça nous donne 16 (16 * 25 * 40 = 16000, la différence correspond bien aux 384 octets qui ne sont pas visible)
Maintenant nous faisons 16384 / 16 = 1024 (2^{10}, en informatique nous aimons les puissance de 2, vu que tout est a base de binaire).
Donc la limite théorique serait de 1024 cases textes, nous allons vérifier ça tout de suite.
En passant R1 a 41 colonnes et sans changer R6 qui reste a 25 lignes.
On constate que la lettre R est reproduite en bas a droite de l'écran.
R1=41 * R6=25 nous arrivons a 1025, soit +1 par rapport à la limite que nous avons calculé précédement, donc il est tout a fait logique d'avoir une seule lettre de répété.





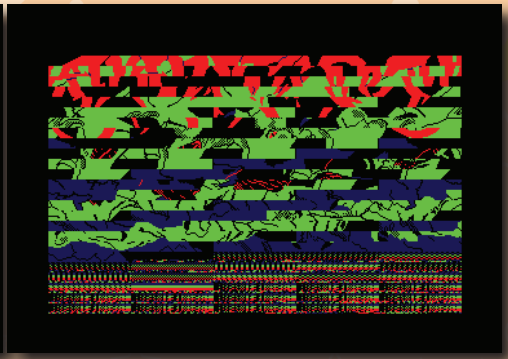
L'écran de présentation pour ZX Spectrum

Une chose que vous avez dû constater, c'est que les lignes textes sont complètement décalées.
Le CRTC affichant les lignes bande par bande les unes derrière les autres, si l'écran est plus petit ou plus grand que la normal (40 colonnes), nous avons un effet visuel assez intéressant.

L'écran de «Thanatos» 1986 © Durell Software est plus petit que la normal (R1=32 au lieu de 40), mais pourquoi ?
Le gros soucis ce sont les portages, beaucoup de jeux ont été créé pour l'ordinateur ZX Spectrum et ont été ensuite adaptée sur Amstrad CPC au lieu d'avoir une version vraiment spécifique.

Revenons sur notre Amstrad CPC.
Etant donnée que la zone visible est réduite, est-ce que les programmeurs en ont tiré profit ?
Le fait d'utiliser une zone écran plus petite, ça permet d'utiliser moins de mémoire, pour les calculs 32 est une puissance de 2, donc plus facile et rapide a utiliser que 40.
En remettant les valeurs par défaut pour R1 et R6 on constate que la zone non visible sous l'écran de présentation a été utilisé par les programmeurs. Toute place libre peut être utile sur des machines n'ayant que très peu de RAM (mémoire vive).

L'écran titre R1=32 et R6=28



Le mystère dévoilé avec R1=40 et R6=25

Malheureusement beaucoup de jeux utilisent R1=32 colonnes et R6 = 24 lignes comme standard. Tout ça à cause des portages provenant du Spectrum. Les deux jeux ci-dessous, sont considérés comme de grand "hit" sur CPC malgrè l'écran réduit.

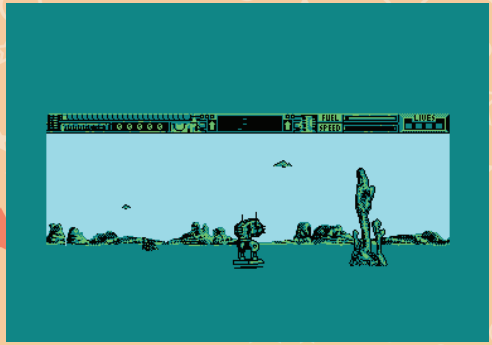
«Ikari Warriors» 1986 © Elite Systems
R1=32 colonnes * R6=24 lignes = 768

«Operation Wolf» 1988 © Ocean Software
R1=32 colonnes * R6=24 lignes = 768

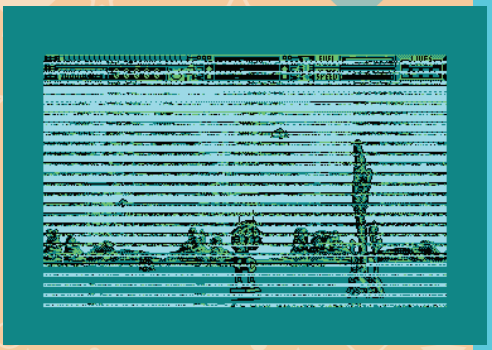
MON ECRAN DE JEU EST RIDICULEMENT PETIT

J'en profite pour vous parler d'une bizarrerie concernant le jeu «Thunder Burner» 1991 © Loriciel
La taille écran du jeu est normale (Registre R1 et R6 du CRTC inchangé) mais la zone réellement utilisée est plus petite. En fait le R9 a été modifié, une ligne de texte n'a plus 8 pixels (&07 car la notation va de 0 a 7) de haut d'affichés mais seulement 5 (&04), ce qui donne un effet de tassement de l'image. Les programmeurs ont été obligés de jouer sur les registres R4, R5 et R7 pour stabiliser l'écran et obtenir l'effet souhaité. Du reste si on ne modifie que R9 vous aurez le droit a un écran illisible qui saute (dans le même style que nos premiers essais).

La définition du programmeur
R4=60 ; R5=6 ; R7=38 ; R9=4



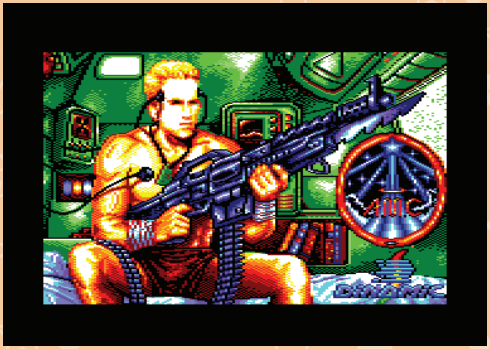
Ecran normal : R4=38 ; R5=0 ; R7=30 ; R9=7
Les données cachées apparaissent.



Y'a pas a dire, ça fait vraiment petit. A la limite, vu la couleur utilisée pour le sol et la bordure, il aurait peut être été bon de remonter la zone de jeu pour donner l'impression que le sol était encore plus grand, histoire de compenser ce grand vide... mais ceci n'est qu'une histoire de goût, et les goût et les couleurs ça ne se discute pas.

On continue dans le ridiculement petit avec un superbe jeu créé par Dinamic Software en 1989, j'ai nommé "Astro Marine Corps", qui prouve que malgré une zone de jeu au format timbre poste (Et oui, seulement la moitié de la taille d'un écran normal d'utilisé), le succès peux être au rendez-vous (scrolling fluide, bande son qui décoiffe, animation impeccable et boss très grand).

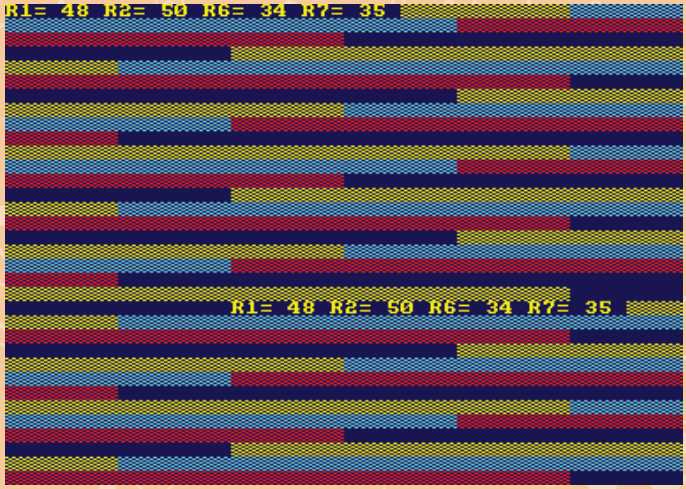
Juste pour le plaisir,
voici l'écran titre qui lui a une taille normale



Un écran de la phase 2
R1=32 colonnes * R6=16 lignes = 512

OVERSCAN PLEIN ECRAN - 32 Ko ou TOTAL BORDERLESS

Lorsqu'on essai de réaliser un affichage sur tout l'écran en éclatant le «BORDER», nous avons la répétition des données affichées à l'écran a cause de notre utilisation de seulement 16 Ko.



Mais il est possible d'associer 2 zones mémoires pour obtenir 32 Ko (Cf le tableau CRTC sur R12 et R13).

Ecran titre de taille normal (Version cassette)
R1=40 colonnes * R6=25 lignes soit 1000

Ecran titre en overscan (Version disquette)
R1=46 colonnes * R6=34 lignes soit 1564

Nous pourrions chipoter sur le terme de plein écran, sachant que l'image de "Crazy Cars II" 1989 © Titus Software, sur disquette aurait pu être plus large de 2 colonnes et peut être d'une ligne texte de plus. Mais l'essentiel n'est pas là. La différence entre un écran de 16 Ko et l'écran en quasi plein écran est impressionnante. Ensuite, en ce qui concerne le jeu, le choc est encore plus grand, sachant que nous retombons à seulement 32 colonnes.

Du reste, au fil des années, les pages écran de présentation où les scènes intermédiaires en overscan sur Amstrad CPC étaient la marque de fabrique de la société Titus Software.

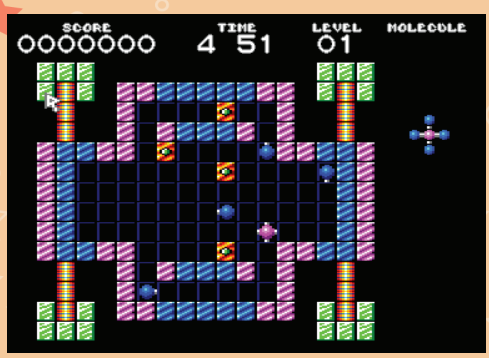
Cette technique étant très gourmande en mémoire, elle a été surtout utilisé par les démomakers.

Nous avons tout de même quelques très très rares jeux en plein écran, je ne parle pas d'écran titre, mais bien de la zone de jeu et même à ce niveau on peu chipoter dans le sens où tout l'écran n'est pas forcément utilisé pour le jeu.

«Xyphoes Fantasy» 1991 © Silmaris
La seule phase du jeu en plein écran,
un clone de Barbarian

Tiens on dirait que Schwarzy nous
passe le bonjour

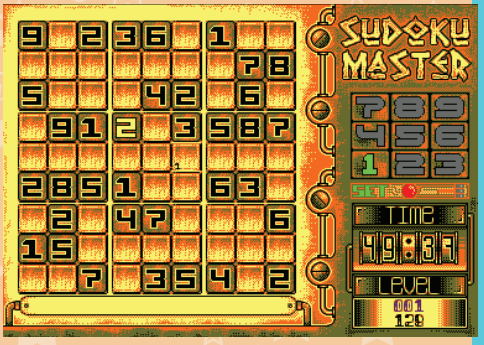
«Moleculaarr 2» 1991 © Amstrad Cent Pour Cent



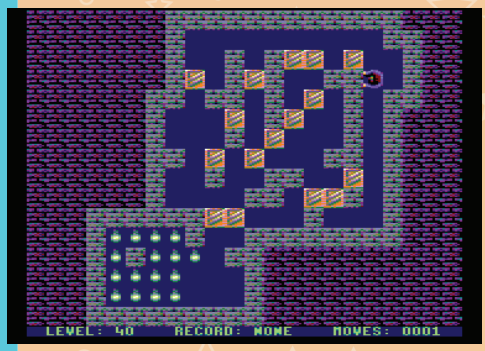
«Les Mondes Paralleles» 1993

«Groops» 2007 © Binary Sciences

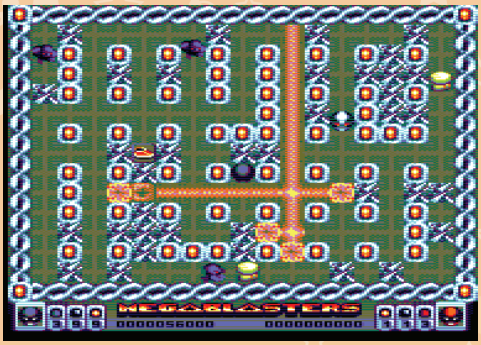
«Sudoku Master» 2009 © Binary Sciences



«Overkoban» 2013 © Futility Games

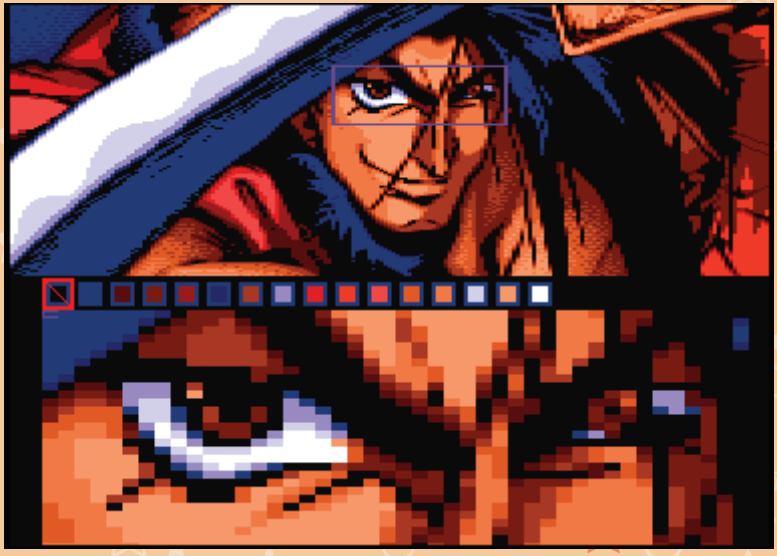


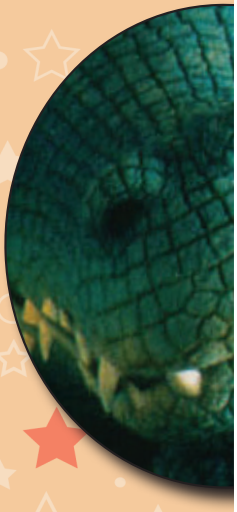
«Megablasters - Escape From Castle In The Clouds» 2015 © Project Argon



Comment réaliser une belle image en plein écran sur Amstrad CPC ?
Ce genre d'utilitaire ne courre pas les rues et je vais vous parler très brièvement d'une version pour CPC Plus et une version pour CPC (Tous les deux sont disponibles sur le site de www.cpc-power.com)

Je commence par la version CPC Plus, nous avons l'utilitaire Graph'OS de **BDC Iron** qui a vu le jour en 2001.
A noter que la particularité de ce programme, c'est qu'il tient dans une ROM et qu'il faut du coup une RAMCARD pour le faire fonctionner sur un vrai CPC Plus, vous pouvez bien entendu l'utiliser avec par exemple l'émulateur WinApe sur votre ordinateur. Cet utilitaire posséde un maximum de touches de raccourcis afin de simplifier l'utilisation.
Magnifique non ? Vous avez bien entendu reconnu Haohmaru le personnage de Samurai Shodown sur Neogeo (le 1er opus étant sortie en 1993)

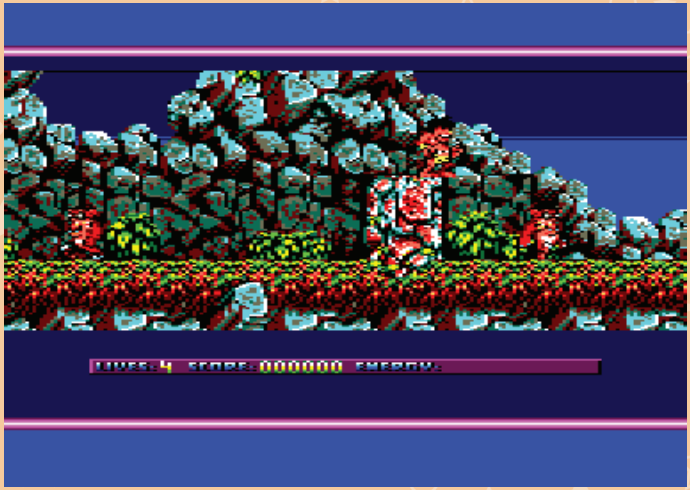




Nous sommes en 2015, et nous avons vu arriver sur nos Amstrad CPC, «iMPdraw Lite v1.132» d'AST au format DSK (disquette). Très ressemblant au niveau de la présentation avec son aieul pour CPC Plus et utilisant lui aussi une grande panoplie de raccourcis claviers pour obtenir la meilleure ergonomie possible.
On notera que l'image du Poulpe a été créé par CED et qu'il a fini 2ème à la compétition Reset #20

RUPTURE

Qu'est ce qui se cache derrière ce mot ?
C'est la possibilité d'afficher plusieurs zones mémoires sur le moniteur.
En BASIC pur, c'est mission impossible, et en assembleur, il faut déjà être un programmeur averti pour pouvoir gérer ce genre de technique avancée. Nous allons prendre l'exemple du jeu «Prehistorik II - Back To Hungerland» 1993 © Titus Software qui allie Overscan Horizontal (que nous avons précédemment décrit) + Rupture (affichage de 2 zone) + Raster (les barres de couleurs horizontales en haut et en bas), Le tout permettant de faire croire qu'on utilise une très grande partie de l'écran.



Registre	R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13
Zone jeu	3F	32	32	0C	2D	00	14	26	00	07	00	00	33	C9
Zone score	3F	2A	32	0C	2D	18	09	26	00	00	00	00	10	00

R12 = &33 (converti en binaire ça nous donne 00110011 ; R13 = &C9 soit 11001001)

R12								R13							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	0	1	1	1	1	0	0	1	0	0	1
X	X	Page &C000		Taille =16ko		Offset =&03C9									

R12 = &10 (converti en binaire ça nous donne 00010000 ; R13 = &00)

R12								R13							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
X	X	Page &4000		Taille =16ko		Offset =&0000									

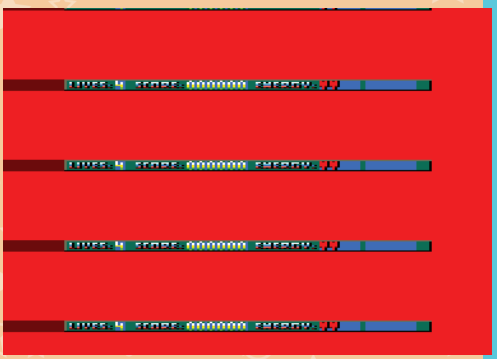
Nous connaissons maintenant, les 2 adresses mémoires pour les 2 zones affichées.

Le petit programme ci-dessous, va permettre d'afficher les 2 zones, mais en aucun cas de gérer une rupture.
ATTENTION, les 2 fichiers chargés "PRE-4000.BIN" et "PRE-C000.BIN" proviennent d'une sauvegarde de la mémoire du jeu a un instant T. (*Listing RUPTURE.BAS*)

```
10 MEMORY &3FFF
20 MODE 0:BORDER 6:RESTORE 20:FOR i=0 TO 15:READ a:INK i,a:NEXT
30 DATA 11,16,6,3,11,15,0,25,9,18,24,3,10,13,20,26
40 LOAD "PRE-4000.BIN",&4000:'longueur &300
50 LOAD "PRE-C000.BIN",&C000:'longueur &4000
60 DATA 1,&32,2,&32,5,&00,6,&14,7,&26,9,&07,12,&33,13,&C9
70 DATA 1,&2A,2,&32,5,&18,6,&09,7,&26,9,&00,12,&10,13,&00
80 RESTORE 60:GOSUB 200:CALL &BB06
90 RESTORE 70:GOSUB 200:CALL &BB06
100 GOTO 80
200 FOR i=1 to 8:READ registre,valeur
210 OUT &BC00,registre:OUT &BD00, valeur:NEXT i:RETURN
```

L'overscan horizontal a partir de &C000

La zone de score a partir de &4000



QUELQUES EXEMPLES D'EFFETS

Le but de cette article n'est pas de vous assomer avec l'aspect technique, mais bien de vous montrer qu'il est possible et ça très facilement de réaliser des petits effets sympathiques et très peu gourmand en mémoire en jouant sur les registres du CRTC.

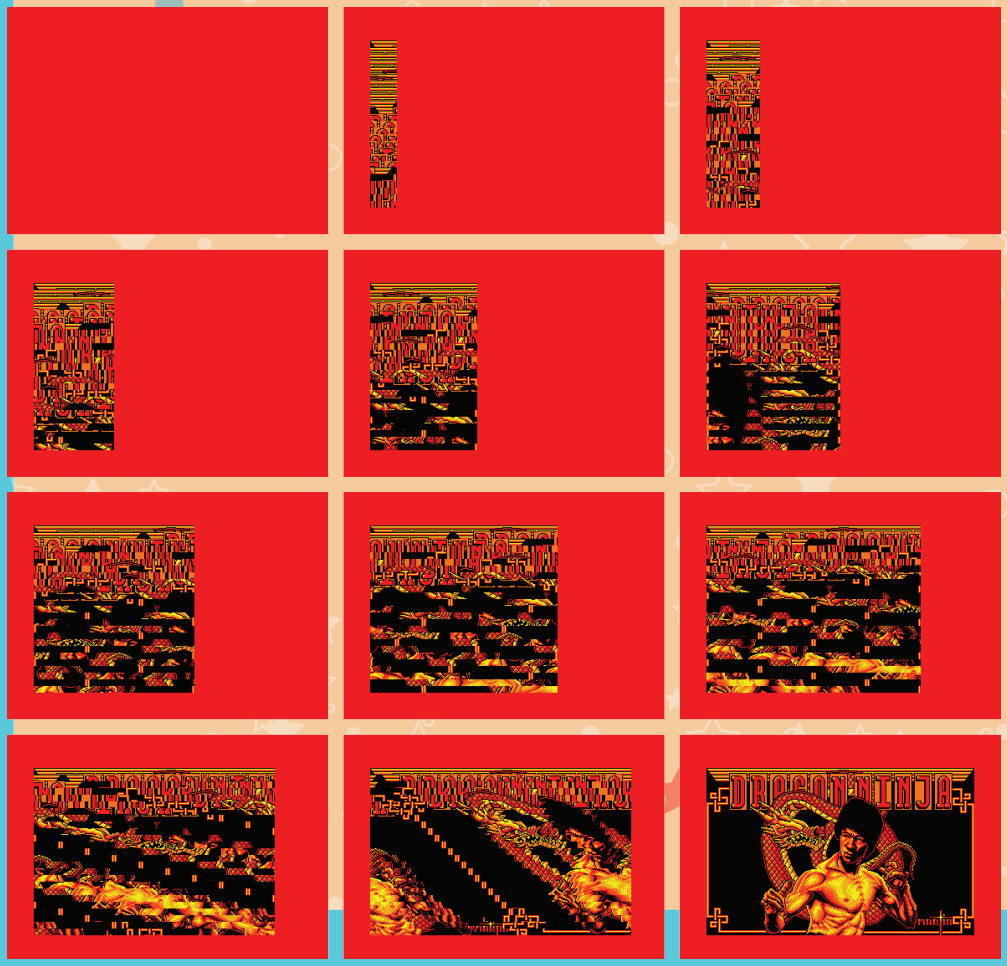
Les différents émulateurs ont parfois du mal a reproduire fidélement les effets présenté ci-dessous (lorsqu'il les supportent), mais dans tous les cas, ça fonctionne sans soucis sur une vraie machine.

A noter que pour les exemples en BASIC ci-dessous, j'ai utilisé l'écran titre de "Dragon Ninja" 1988 © Ocean Software.

EXEMPLE 1 : (*Listing EXEMPLE1.BAS*)
Effet de masquage et d'affichage de l'écran par bande en jouant uniquement sur R1.
Le principe a été utilisé dans les jeux "Beach-Head" 1985 © Amsoft ou "Fighting Warrior" 1985 © Melbourne House pour ne citer qu'eux.

```
10 MODE 1:BORDER 6:INK 0,24:INK 1,0:INK 2,15:INK 3,6
20 GOSUB 100
30 LOAD "ECRAN.SCR",&C000
40 GOSUB 150
50 CALL &BB06
60 END
100 REM Masquer
110 OUT &BC00,1:FOR i=40 TO 0 STEP -1:OUT &BD00,i:GOSUB 200:NEXT i:RETURN
150 REM Afficher
160 OUT &BC00,1:FOR i=0 TO 40:OUT &BD00,i:GOSUB 200:NEXT i:RETURN
200 REM temporisation un peu trop forte pour bien voir l'effet
210 FOR x=1 TO 100:NEXT x:RETURN
```

Uniquement 12 captures de présente sur les 41 changements (valeur 0 à 40) de valeur dans le registre R1. On devine l'effet de décalage de l'image par bande avec ces quelques écrans.
Le visualiser sur un vrai CPC ou sur un émulateur est beaucoup plus agréable bien entendu.



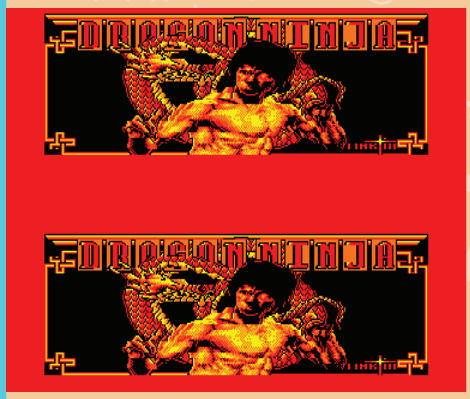
EXEMPLE 2 : (*Listing EXEMPLE2.BAS*)
Effet de masquage et d'affichage de l'écran en jouant uniquement sur R6 (nombre de ligne texte visible). Simple et efficace.

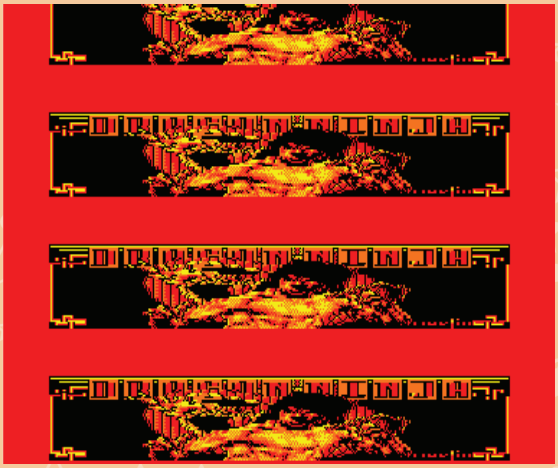
```
10 MODE 1:BORDER 6:INK 0,24:INK 1,0:INK 2,15:INK 3,6
20 GOSUB 100
30 LOAD "ECRAN.SCR",&C000
40 GOSUB 150
50 CALL &BB06
60 END
100 REM Masquer
110 OUT &BC00,6:FOR i=25 TO 0 STEP -1:OUT &BD00,i:GOSUB 200:NEXT i:RETURN
150 REM Afficher
160 OUT &BC00,6:FOR i=0 TO 25:OUT &BD00,i:GOSUB 200:NEXT i:RETURN
200 REM temporisation un peu trop forte pour bien voir l'effet
210 FOR x=1 TO 100:NEXT x:RETURN
```

Uniquement 12 captures de présente sur les 26 changements (valeur 0 à 25) de valeur dans le registre R6. Ce qui permet de se rendre compte de l'effet.

EXEMPLE 3 : *(Listing EXEMPLE3.BAS)*
Effet permettant de réduire l'image et de l'afficher plusieurs fois à l'écran, en jouant uniquement sur le registre R9. Il faudrat appuyer sur une touche pour passer a l'effet suivant. Suivant les CRTC, la zone affichée peut trembler.

```
10 MODE 1:BORDER 6:INK 0,24:INK 1,0:INK 2,15:INK 3,6
20 LOAD "ECRAN.SCR",&C000
30 OUT &BC00,9
40 REM ecran x2
50 OUT &BD00,3:CALL &BB06
60 REM ecran x3
70 OUT &BD00,2:CALL &BB06
80 REM ecran x4
90 OUT &BD00,1:CALL &BB06
100 GOTO 40
```

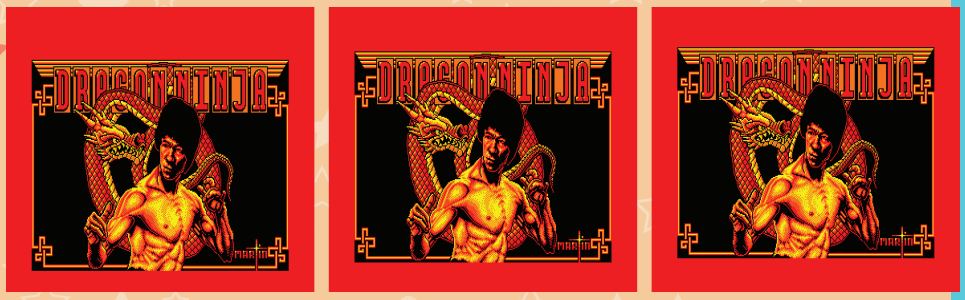




EXEMPLE 4 : (*Listing EXEMPLE4.BAS*)
Effet permettant de donner l'impression qu'on a été touché ou qu'une collision s'est produite en jouant sur la synchronisation verticale (R5).
Utilisé dans le jeu commercial "Pepe Bequilles" 1987 © Softhawk ou "Cyber-Chicken" 2013 © AMC Soft.

```
10 MODE 1:BORDER 6:INK 0,24:INK 1,0:INK 2,15:INK 3,6
20 LOAD "ECRAN.SCR",&C000
30 FOR i=1 TO 19
40 FOR x=20 TO 1 STEP -2
50 OUT &BC00,5:OUT &BD00,x
60 NEXT x,i
70 OUT &BC00,5:OUT &BD00,0
80 CALL &BB06
90 GOTO 30
```

Avec les 3 captures, on constate que l'écran est tout simplement décaler verticalement, alors imaginez tout simplement que l'on secoue l'écran et l'image va monter et descendre très rapidement, l'effet de choc ou de rebond va être ascentué.



EXEMPLE 5 : (*Listing EXEMPLE5.BAS*)
Afficher alternativement et très rapidement une image à l'écran, par exemple, soit celle en &C000 ou celle en &4000.
Cette technique a été utilisé entre autre, sur des portages de jeu ZX Spectrum utilisant un buffer (Tampon).
La technique est simple, le programmeur travaille avec l'équivalent de 2 zones écran, pendant qu'il affiche la première, il prépare l'écran suivant sur la zone que nous ne voyons pas, et ainsi de suite.
Pour information, le jeu "Profanation" 1987 © Chip utilise cette technique de double zones écran.

```
10 MEMORY &3FFF
20 MODE 1:BORDER 6:INK 0,24:INK 1,0:INK 2,15:INK 3,6
30 LOAD "ECRAN.SCR",&4000
40 LOAD "ECRAN2.SCR",&C000
50 OUT &BC00,12
60 OUT &BD00,&30:'&C000
70 GOSUB 110
80 OUT &BD00,&10:'&4000
90 GOSUB 110
100 GOTO 60
110 REM temporisation (Tellement rapide qu'il faut ralentir tout ça)
120 FOR i=0 TO 20:CALL &BD19:NEXT i:RETURN
```

Cette technique a été utilisé dans des discmags ou des mini-démos pour créer de petites animations, couplées avec l'utilisation des banks mémoires, permettant ainsi d'avoir sur un CPC ayant 128 Ko de Ram, 6 images non compressées très facilement affichables en BASIC.

EXEMPLE 6 : (*Listing EXEMPLE6.BAS*)
Faire tourner la zone écran de la droite vers la gauche (sans finesse, je dois le reconnaître).
On se rapproche de l'effet utilisé dans Crafton & Xunk.

```
10 MODE 1:BORDER 6:INK 0,24:INK 1,0:INK 2,15:INK 3,6
20 LOAD "ECRAN.SCR",&C000
30 OUT &BC00,2
40 FOR i=1 TO 45
50 CALL &BD19:'attendre la synchronisation du balayage ecran
60 OUT &BD00,i
70 NEXT i
80 CALL &BB06
90 GOTO 40
```

EXEMPLE 7 : (*Listing EXEMPLE7.BAS*)
Faire tourner la zone écran de la droite vers la gauche, en 2 parties.
Comme vous pouvez le constater, il n'y a pas une, mais des solutions pour arriver a se rapprocher de l'effet utilisé dans Crafton & Xunk.

```
10 MODE 1:BORDER 6:INK 0,24:INK 1,0:INK 2,15:INK 3,6
20 LOAD "ECRAN.SCR",&C000
30 OUT &BC00,2
40 FOR a=46 TO 62 STEP 2:OUT &BD00,a:CALL &BD19:NEXT
50 FOR a=0 TO 46 STEP 2:OUT &BD00,a:CALL &BD19:NEXT
80 CALL &BB06
90 GOTO 40
```

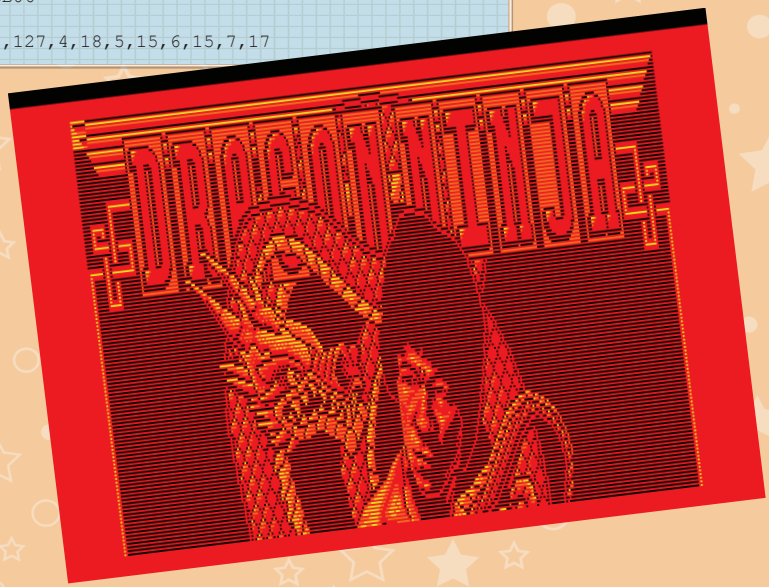
EXEMPLE 8 : (*Listing EXEMPLE8.BAS*)
Voici une technique que j'ai utilisé il y a fort longtemps pour obtenir un affichage aggrandi pour le jeu de football "Buitre - Emilio Butragueno Futbol" 1988 © Topo Soft. Il faut savoir que cette technique est très mal supportée, voir, pas supporté du tout par les émulateurs.

L'écran de jeu de Buitre.
La zone de jeu est un peu petite
mais reste correct tout de même.

L'application de l'astuce sur le jeu, nous donnant
l'impression d'avoir une zone de jeu plus grande,
alors qu'il n'en est rien, ce n'est qu'une tricherie.

Maintenant nous testons sur l'écran de Dragon Ninja :

```
10 MODE 1:BORDER 6:INK 0,24:INK 1,0:INK 2,15:INK 3,6
20 LOAD "ECRAN.SCR",&C000
30 RESTORE 100
40 FOR i=1 TO 5
50 READ registre,valeur
60 OUT &BC00,registre:OUT &BD00,valeur
70 NEXT i
80 CALL &BB06
90 GOTO 30
100 DATA 0,127,4,18,5,15,6,15,7,17
```



Pour vous faciliter la tâche et tester plus facilement les listings et exemples présent dans les articles, ils seront dorénavant présents sous forme de compilation et mis à disposition gratuitement sur le site de cpc-power.

L'équipe de CPC-POWER