



Architecture des ordinateurs

Compte-rendu de TP/TD

Germain Le | Erwan Keraval

1. Présentation

Durant les quatre séances de TP, nous avons développé un mini-jeu pour GameBoy Advance, programmé en langage C et en langage assembleur ARM. Le but était d'obtenir un vaisseau dirigeable à l'aide des flèches directionnelles, avec un paysage défilant en arrière plan et des soucoupes volantes au mouvement aléatoire. Il a fallu également gérer les collisions entre le vaisseau, les soucoupes et le fond. À travers ce compte-rendu, nous verrons la démarche que nous avons entreprise pour réaliser notre programme, ainsi que quelques améliorations que nous avons pu y apporter.

2. Réponses aux questions de TP

2.1 TP n°1 – Environnement de projet, exemples de projets, création de projet

Le but du TP1 était une initiation à l'environnement de développement VHAM.

- La procédure assembleur *asmDrawPixel* s'utilise effectivement de la même manière que la procédure C *drawPixel* : à la compilation, les deux procédures affichent la même chose.

- Comparons les tailles des codes sources .asm et .c :

"main.c" (1,50Ko) | "asm.s" (1,52Ko)

"armgba.c" (2,69Ko) | "asmarmgba.s" (8,09 Ko)

La version C n'est pas forcément plus performante que la version assembleur, car le code à compiler est plus long, car plus proche du langage machine.

- Au départ, on a : **u32 color32=0x00007fff;**

0x00007fff = 0b0[...]0111111111111111;

On a bleu = 31, rouge = 31, vert = 31. Cela correspond à du blanc.

- La ligne **color32 = (color32>>1)|(color32<<31);** sert à calculer la nouvelle couleur.

On fait d'abord un décalage à droite de 1 bit, puis un décalage à gauche de 31 bits, et on combine les 2 dans un "OU". La rotation se fait en 32 étapes.

- Lorsqu'on enlève la procédure *vSynch()*; il n'y a plus de rafraîchissement de l'écran : ainsi, toutes les couches se superposent, et l'écran est rempli de couleurs.

- Pour changer la couleur du carré lorsqu'on appuie sur W ou X (A et B sur la GBA), il suffit d'ajouter le code :

```

if ((~REG_KEYINPUT)&KEY_A) {
    col=asmMakeColor(20,15,90);
}

if ((~REG_KEYINPUT)&KEY_B) {
    col=asmMakeColor(255,255,255);
}

```

2.2 TP n°2 – Développement d'applications graphiques en assembleur ARM

Dans le TP n°2, nous avons affiché d'abord des lignes en langage assembleur, puis un petit vaisseau se déplaçant sur l'écran à l'aide des touches du clavier.

- Pour tracer un segment à 45°, on pourrait faire :

add r2,r2,#482 @ Adresse du pixel suivant

Cependant, cette solution ne marche pas, car 482 n'est pas compatible avec le format que prend "add".

On peut alors faire :

add r2,r2,#2 @ Adresse du pixel suivant

add r2,r2,#480 @ Adresse du pixel suivant

On peut aussi créer un nouveau registre r4 :

lsl r4,#482 @ Nouveau registre

add r2,r2,r4 @ Pixel suivant

2.3 TP n°3 – Implémentation d'un scrolling et importation d'image bitmap, détection de collision

Dans le TP n°3, nous avons eu à mettre en place un système de scrolling circulaire pour faire défiler un paysage derrière le vaisseau, dans le but de donner une impression de déplacement.

- Quand on met la procédure *asmRectcpy* dans la boucle *while(1)*, on voit que la partie inférieure défile de droite à gauche.

2.4 TP n°4 – Intégration des éléments du programme de jeu, optimisations et estimations de performances

Le but du TP n°4 était de finaliser le programme en y intégrant des éléments en plus (soucoupes volantes, barre de progression etc), ainsi que d'utiliser des timers pour évaluer les performances du programme.

- Temps d'exécution du sous programme *drawSaucer* :

On trouve $0x0075$, soit $4,46 \cdot 10^{-4} s = 0,446 ms$.

Avec *asmDrawImage*, on trouve $0x0018$, soit $9,16 \cdot 10^{-5} s = 0,0916 ms$.

On note une amélioration significative du temps d'exécution avec la version assembleur.

- Temps d'exécution du scrolling circulaire :

On trouve $0x0986$, soit $9,30 ms$.

- Lorsque l'on enlève la procédure *VSync*, on observe des clignotements sur l'écran lors du changement de position du vaisseau et des soucoupes.

3. Aspects fonctionnels du programme final

Dans le programme final, nous avons tout d'abord repris tous les éléments donnés en TP et TD, avec l'affichage du vaisseau et la gestion de son déplacement à l'aide des touches du clavier, l'effacement du vaisseau (en le redessinant, mais en remplaçant toutes les couleurs par du noir), la gestion du scrolling circulaire, l'affichage et le déplacement aléatoire des soucoupes volantes. Les collisions sont également gérées grâce à des conditions posées sur le déplacement des objets. La transparence des soucoupes a aussi été codée, directement dans le programme d'affichage en assembleur.

Nous avons également rajouté une barre de progression. Le principe est simple : la partie commence avec une seule soucoupe volante. Le joueur doit éviter la soucoupe, et la barre de progression augmente avec le temps. Lorsque celle-ci atteint le bord droit de l'écran, on rajoute une soucoupe, etc...jusqu'à ce qu'il y ait dix soucoupes sur l'écran. La barre revient à zéro si le joueur touche une soucoupe volante. Si le joueur gagne, on affiche une image de victoire à l'écran. Pour l'image, nous avons utilisé le programme codé avec ALLEGRO fourni dans le TP3, que nous avons compilé avec une de nos images bitmap.

- Pour plus de précisions, voir directement sur le code source commenté -

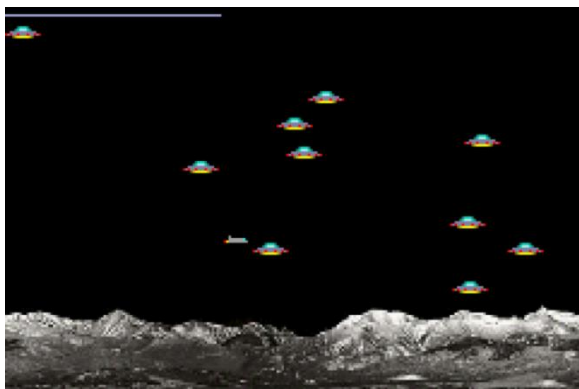
Captures d'écran du programme :



Début du jeu : une seule soucoupe



Exemple de gestion de transparence : on ne voit pas les bords noirs des images qui se superposent.



Le dernier niveau, avec les dix soucoupes



Affichage d'un message de victoire lorsque le joueur gagne le dernier niveau