ROMON Pierre
CANDUSSO Alix

TP8
Traitement d'image
Partie 3

Table des matières

Deux pixels à la fois

Chargement des pixels

Afin de charger 2 pixels dans un registre SSE, on exploite l'adjacence des pixels dans la mémoire. En effet, les pixels sont des doubles-mots stockés en enfilade dans la mémoire. Ainsi en chargeant un mot quadratique avec *movd* associé à un *qword ptr*, on peut récupérer 2 pixels successifs. Comme dans les TP précédent, on obtient l'adresse des pixels en fonction de l'adresse du premier pixel de l'image *r8* ou *r9* et le nombre de pixel dans l'image *rcx* (produit des dimensions en pixels de l'image). Il faut cependant faire attention à ne pas essayer de charger un pixel inexistant en fin ou début d'image maintenant qu'on procède par 2 pixels à la fois. Cette méthode n'est donc applicable qu'aux images avec un nombre de pixels pair, ce qui est le cas ici. Toujours dans la même optique, il faut partir de l'avant dernier pixels afin de stocker dans le registre SSE l'avant-dernier et le dernier pixel sans déborder de la mémoire (on rappelle que l'on parcourt l'image de la fin vers le début)

Conversion des pixels

On a désormais 2 pixels que l'on peut considérer comme des vecteurs d'octets dans notre registre SSE. Cependant les opérations futures requièrent des vecteurs de mots de 16 bits et non d'octets. Pour travailler sur les pixels dans un tel format, on utilise l'opérande *punpcklbw* avec un autre registre SSE contenant uniquement des 0. Grâce à cette méthode d'unpack, on récupère des pixels dont les composantes sont des mots de 16 bits : une juxtaposition de deux 0 et de leur composante initiale.

Construction du vecteur des coefficients

Les opérations futures impliquant les coefficients de couleur, on cherche également à stocker ces coefficients dans 2 vecteurs de mots de 16 bits au sein d'un registre SSE. Pour cela, on utilise la même méthode d'unpack sur un registre SSE contenant dans sa partie basse les 2 séries de coefficients.

Calcul des intensités

Les vecteurs de mots de 16 bits maintenant correctement construits dans 2 registres SSE, il s'agit maintenant de calculer de l'intensité de chaque pixel. Ce calcul est effectué en 2 étapes. La première utilise la mnémonique *pmaddwd* qui agit sur des mots de 16 bits dans un vecteur. En effet, cet opérande multiplie 2 à 2 les mots de 16 bits des 2 registres puis fait la somme 2 à 2 des mots de 32 bits adjacents résultant du produit précédent. On obtient l'intensité selon le bleu et le vert et l'intensité selon le rouge.Pour obtenir l'intensité finale, on utilise *phaddd* qui, entre autre, somme les double-mots adjacents. Enfin, par un décalage d'un octet sur la droite avec *psrldq*, on élimine la partie décimale.

Stockage des pixels

Il ne reste plus qu'à stocker cette intensité dans la composante bleue de l'image destination. Pour cela, on procède de la même manière (mais dans le sens opposé que lors du chargement des 2 pixels dans le registre SSE puisque les intensités sont positionnées correctement dans le registre SSE) avec un *movd* et un *qword ptr*

Performance du programme

En comparant les vitesses d'exécution entre la méthode pixel par pixel et cette méthode-ci, on note une flagrante différence en faveur de cette nouvelle méthode. En effet, on rappelle que le programme pixel par pixel traite une image en environ 10,3 ms tandis qu'avec 2 pixels à la fois le programme traite l'image en 1,3 ms.

Quatre pixels à la fois

Chargement des pixels

De la même manière que pour charger 2 pixels, on exploite la juxtaposition des pixels dans la mémoire. Ainsi en chargeant 128 bits dans le registre SSE, on prend 4 pixels successifs. En pratique, il n'est pas possible de charger 128 bits d'un coup dans un registre SSE, on procède donc en 2 étapes de 64 bits. On stocke tout d'abord 2 pixels dans la partie basse du registre SSE de la même manière que précédemment. Ces 2 pixels sont translatés dans la partie haute (de 8 octets sur la gauche) du registre SSE avec un *pslldq*. On stocke dans un registre SSE temporaire les 2 pixels suivants qu'on copie dans la partie basse de notre registre SSE principal avec un *por*. Ici aussi, il faut faire attention à ne pas déborder de la mémoire en partant du quatrième pixel avant la fin. La méthode n'est d'ailleurs applicable qu'aux images possédant comme nombre de pixel un multiple de 4.

Transformation des coefficients

Dans le cas présent les 4 pixels occupent la totalité du registre SSE, il est donc impossible de les convertir en vecteur de mots de 16 bits pour appliquer la même méthode de calcul d'intensité. On doit donc utiliser une autre mnémonique, à savoir, *pmaddubsw* qui réalise le même travail que *pmaddwd* mais avec des octets au lieu de mots de 16 bits et donc un résultat en mot de 16 bits et non de 32 bits. Cependant cet opérande requiert un vecteur d'octet non-signés et un vecteur d'octets signés. Il nous faut donc un vecteur d'octet signés qui contient les séries de coefficients de couleurs. Cependant, le coefficient multiplicatif du vert est égal à 96 en hexadécimal, son bit de poids fort est donc un 1 et il sera interprété comme un nombre négatif. Pour pallier à cela, en supposant que les coefficients sont tous pairs, on divise par 2 les coefficients. L'intensité calculé en sera tout simplement divisé par 2. À partir des coefficients initiaux 4d (rouge), 96 (vert) et 1d (bleu), on détermine les nouveaux coefficients : 26 (rouge), 4b (vert) et 0e (bleu).

Construction du vecteur des coefficient

La construction du vecteur des coefficient est assez simple. On commence par charger 2 séries de nouveaux coefficients dans la partie basse du registre SSE et on les duplique dans la partie haute grâce à un *punpcklqdq*.

Calcul des intensités

Le calcul des intensités se déroule désormais exactement comme dans le cas 2 pixels avec un *pmaddubsw* et un *phaddw* somme des mots et non des double-mots adjacents. On n'oublie pas à la fin de sommer avec elles-mêmes les intensités grâce à un *paddw* afin d'obtenir le bon résultat.

Stockage des pixels

On construit le registre SSE contenant les 4 intensité de manière à ce que le stockage soit immédiat, c'est à dire un registre ne contenant que de 0 et les intensités aux emplacements théoriques de la composante bleu des pixels. Le stockage se fait donc en 2 étapes : partie basse puis partie haute (le décalage de 8 octets se fait avec *psrldq)*. On utilise ici encore un *movd* associé à un *qword ptr*.

Performance du programme

Ce programme est moins efficace que l'on pourrait le penser malgré le fait de traiter 4 pixels par itération. En effet, les opérations supplémentaires afin d'être en mesure de traiter 4 pixels à la fois diminue les performances du programme. Ainsi, ce programme traite une image en 1,35 ms, ce qui est tout de même mieux que la version traitant 2 pixels à la fois.