

## TD-8 : SIMD et traitement d'images

Dans ce TD, on aborde l'utilisation des instructions SIMD pour des algorithmes de traitement d'images avec le jeu d'instructions IA-32. Pour tous les filtres 3x3 considérés, on ne se préoccupera pas des bords de l'image résultat, c'est-à-dire que le traitement d'une image N\*N produira une image (N-2)\*(N-2) en scalaire et (N-32)\*(N-32) en SIMD..

### **Filtre MIN**

On travaille sur des images avec pixels en niveau de gris (0-255) représentés par des « unsigned char ».

Soit un filtre MIN (3x3) qui remplace le point milieu du filtre par la valeur min des neuf valeurs.

- Définir un algorithme (version naïve) pour le calcul scalaire du filtre MIN
- Définir un algorithme optimisé, minimisant le nombre d'accès mémoire et d'opérateurs min, pour le calcul scalaire du filtre MIN
- Définir un algorithme (version naïve) pour le calcul SIMD du filtre MIN
- Optionnel : écrire un algorithme optimisé pour le calcul SIMD du filtre MIN

Ecrire les programmes C correspondant avec IA-32

### **Filtre conservatif**

On travaille sur des images avec pixels en niveau de gris.

Soit le filtre conservatif (3x3) qui conserve la valeur du point milieu du filtre si celle-ci est comprise entre le MIN et le MAX des huit pixels voisins et la remplace par le MIN (respectivement) le MAX des huit points voisins dans le cas contraire.

- Définir un algorithme pour le calcul scalaire du filtre conservatif
- Définir un algorithme pour le calcul SIMD du filtre conservatif

Ecrire les programmes C correspondant avec IA-32

### **Benchmark Grayscale**

On travaille sur des images avec pixels en niveau de gris.

Soit le benchmark Grayscale qui applique le filtre 3x3 suivant :

$$\frac{1}{256} \begin{bmatrix} -28 & -28 & -28 \\ -28 & 255 & -28 \\ -28 & -28 & -28 \end{bmatrix}$$

Pour ce filtre

- Définir un algorithme pour le calcul scalaire
- Définir un algorithme pour le calcul SIMD

Ecrire les programmes C correspondant avec IA-32

### **Annexe : Instructions SIMD (intrinsics)**

```
#define ld16(a)      _mm_load_si128(&a)           //chargement aligné
#define st16(a, b)  _mm_store_si128(&a, b)       //rangement aligné
#define or(a,b)     _mm_or_si128(a,b)           //ou logique
#define xor(a,b)    _mm_xor_si128(a,b)         //ou exclusif
#define maxbu(a,b)  _mm_max_epu8(a,b)          // max 8 bits non signés
#define minbu(a,b)  _mm_min_epu8(a,b)          // min 8 bits non signés
#define addh(a,b)    _mm_add_epi16(a,b)         // addition 16 bits signée
#define addhu(a,b)  _mm_add_epu16(a,b)         // addition 16 bits non signée
#define subh(a,b)    _mm_sub_epi16(a,b)         //soustraction 16 bits signée
#define b2hl(a,b)   _mm_unpacklo_epi8 (a,b)     //4 octets bas entrelacés vers 4 shorts
#define b2hh(a,b)   _mm_unpackhi_epi8 (a,b)     //4 octets haut entrelacés vers 4 shorts
#define h2b(a,b)    _mm_packus_epi16(a,b)       //8 shorts (a) dans 8 octets bas
                                                    //8 shorts (b) dans 8 octets haut
#define srli128(a,v) _mm_srli_si128(a,v)        // décalage droite des 128 bits de a de v octets
#define slli128(a,v) _mm_slli_si128(a,v)        // décalage gauche des 128 bits de a de v octets
#define srli16(a,v)  _mm_srli_epi16(a,v)        // déc. droite des 8x 16 bits de a de v bits
#define slli16(a,v)  _mm_slli_epi16(a,v)        //déc. gauche des 8x16 bits de a de v bits
```

### ***Accès SIMD aux pixels voisins***

```
#define decd(va,vb)  _mm_or_si128 (_mm_srli_si128(va,1),_mm_slli_si128(vb,15))
#define decg(va,vb)  _mm_or_si128 (_mm_slli_si128(va,1),_mm_srli_si128(vb,15))
```

#### **Exemple**

```
aij= _mm_load_si128(&XS[i][j]);
aijp = decg(_mm_load_si128(&XS[i][j]),_mm_load_si128(&XS[i][j-1]));
aijm= decd(_mm_load_si128(&XS[i][j]),_mm_load_si128(&XS[i][j+1]));
```

### ***Alignements mémoire***

Variables allouées statiquement

```
__declspec( align(16) ) V1, V2, V3;
```

Variables allouées dynamiquement

- #include malloc.h
- Fonctions \_mm\_malloc et \_mm\_free

```
byte** bmatrix(long nrl, long nrh, long ncl, long nch)
{long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
  byte **m;
  /* allocate pointers to rows */

  m=(byte **) _mm_malloc((size_t)((nrow+NR_END)*sizeof(byte*)), 16);
  if (!m) nrorror("allocation failure 1 in bmatrix()");
  m += NR_END;
  m -= nrl;
  /* allocate rows and set pointers to them */
  .....
  return m;}
```