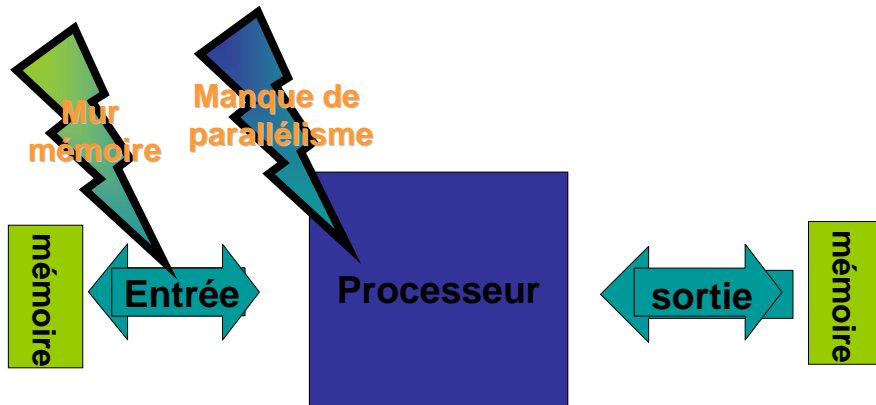

GPUs et calcul scientifique

Daniel Etiemble
de@lri.fr

GPU pour calcul scientifique

- Modèles de calcul : GPU et CPU
- Programmation GPU
- Algorithmes pour GPU : tris, recherches...

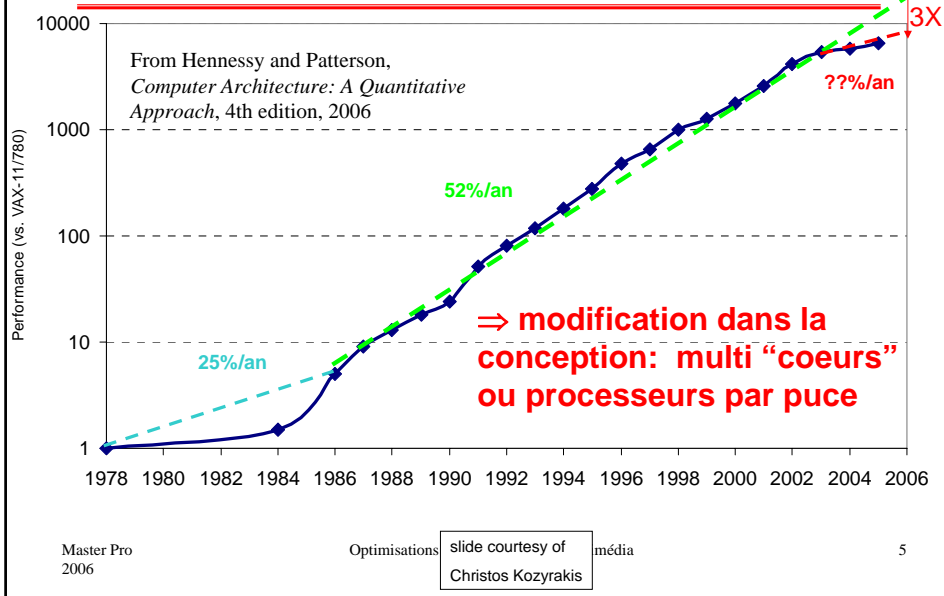
Traitement de données : cas général



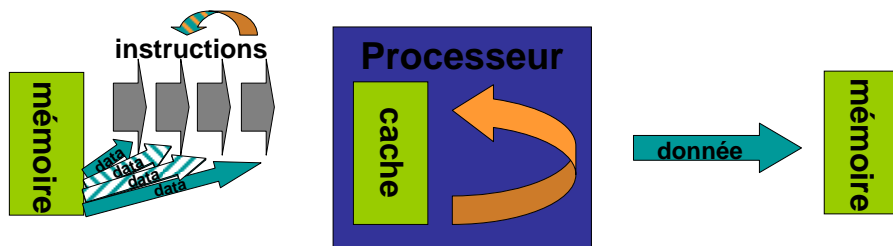
Architecture des ordinateurs : hier et aujourd'hui

- **Hier**: Pas de problème de puissance, les transistors sont coûteux
- **Aujourd'hui**: "Mur de puissance", La puissance coûte cher, les transistors non (On peut mettre plus de transistors sur une puce que le nombre que l'on peut contrôler)
- **Hier**: Les multiplications sont lentes, les accès mémoire rapides
- **Aujourd'hui**: "Mur mémoire", Les multiplications sont rapides, la mémoire est lente (200 cycles pour accéder à la DRAM, 4 cycles pour une multiplication flottante)
- **Hier**: Augmenter le Parallélisme au niveau instructions (ILP) via les compilateurs et l'innovations (exécution non ordonnées, spéculation, VLIW, ...)
- **Aujourd'hui**: "mur ILP", moins de gain avec plus de matériel pour l'ILP (Le parallélisme explicite de threads et de données doit être exploité)

Performance uniprocasseur (SPECint)



Calcul Von Neuman (flot d'instructions)



Flots d'instructions et de données

Additions de deux matrices : $C = A + B$

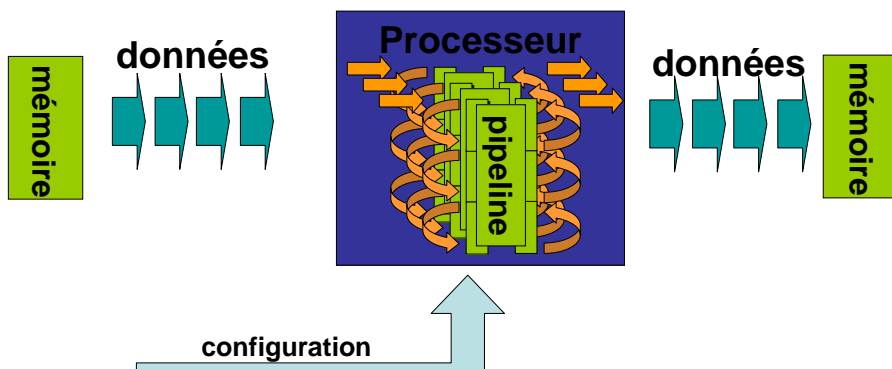
**Flot
d'instructions
traitant
des données**

```
for(y=0; y<HEIGHT; y++)  
for(x=0; x<WIDTH; x++) {  
    C[y][x] = A[y][x] + B[y][x];  
}
```

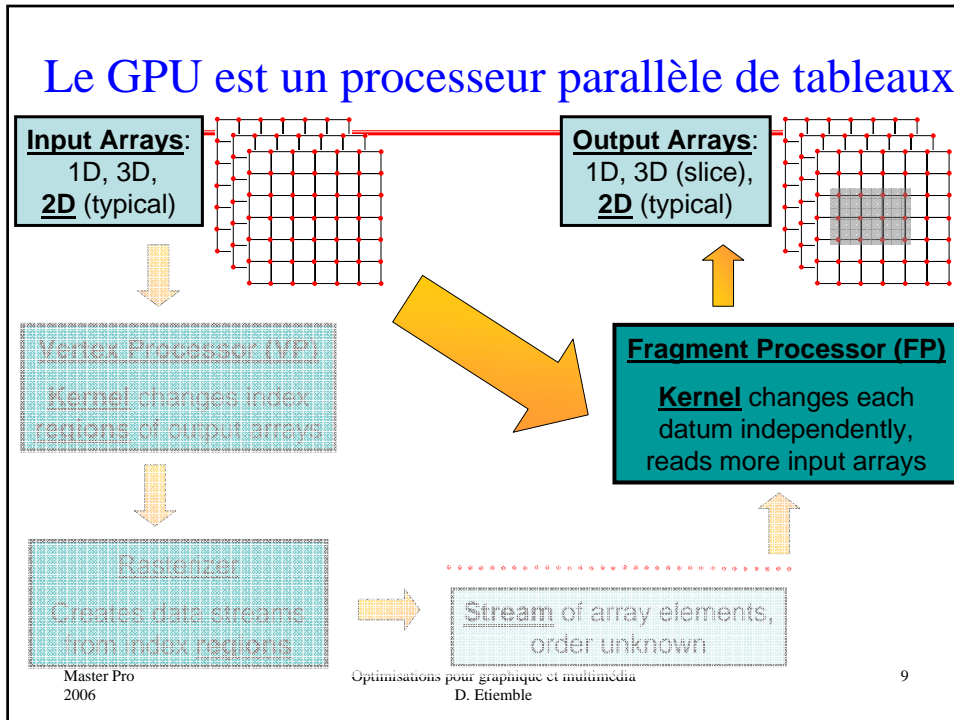
**Flot de données
Opération noyau**

```
inputStreams(A, B);  
outputStream(C);  
kernelProgram(OP_ADD);  
processStreams();
```

Traitement de flots de données



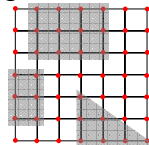
Le GPU est un processeur parallèle de tableaux



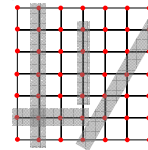
Régions d'indices dans les tableaux de sortie

- ~~Rectangles et triangles~~
 - Option idéale
- Lignes
 - Plus lent. Essayer d'apparier des lignes en rectangles 2xh, wx2
- Nuages de points
 - Le plus lent, essayer de constituer des formes plus grandes

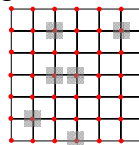
Région de sortie



Région de sortie



Région de sortie



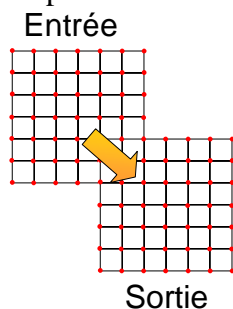
Langages graphiques pour noyaux

- Types de données flottantes
 - half 16-bit (s10e5), float 32-bit (s23e8)
- Vecteurs, structures et tableaux :
 - float4, float vec[6], float3x4, float arr[5][3], struct { }
- Opérations arithmétiques et logiques:
 - +, -, *, /; &&, ||, !
- Fonctions trigonométriques, exponentielles:
 - sin, asin, exp, log, pow, ...
- Fonctions définies par l'utilisateur
 - max3(float a, float b, float c) { return max(a,max(b,c)); }
- Conditionnelles et boucles:
 - if, for, while, branchement dynamique dans PS3
- Accès aux données (flots, aléatoires)

Tableaux entrée et sortie

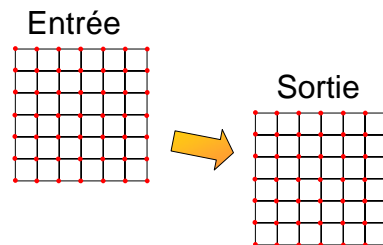
CPU

- Les tableaux d'entrée et de sortie peuvent se recouvrir



GPU

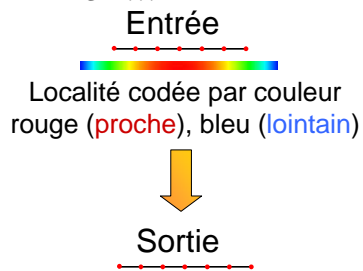
- Les tableaux d'entrée et de sortie NE peuvent se recouvrir



Implantation mémoire – Localité des données

CPU

- Entrée 1D
- Sortie 1D
- Déplacement pour 2D, 3D...

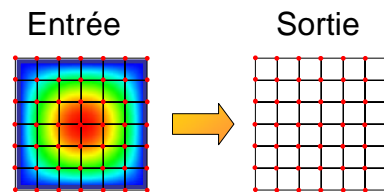


Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

GPU

- Entrée 1D, 2D, 3D
- Sortie 2D
- Autres dimensions avec déplacements

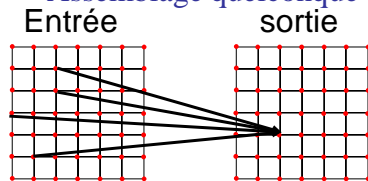


13

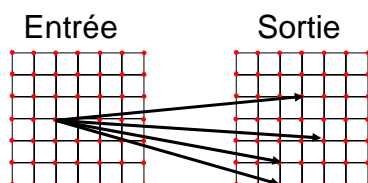
Flot de données: Gather et Scatter

CPU

- Assemblage quelconque



- Dispersion quelconque

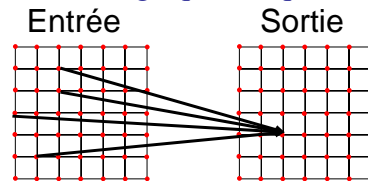


Master Pro
2006

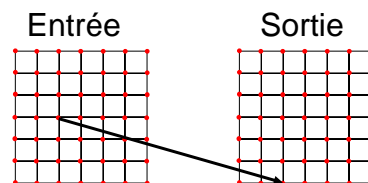
Optimisations pour graphique et multimédia
D. Etiemble

GPU

- Assemblage quelconque

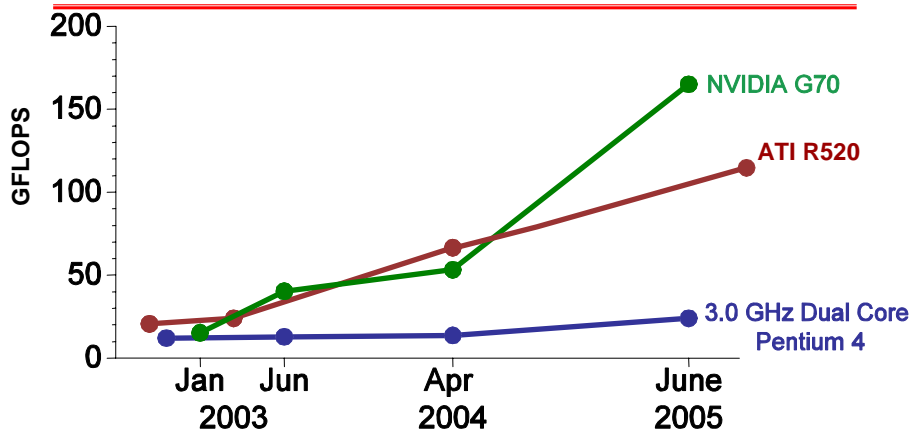


- Dispersion restreinte



14

1) Performance de calcul



Note: la performance soutenue est généralement beaucoup plus basse et dépend beaucoup du système mémoire.

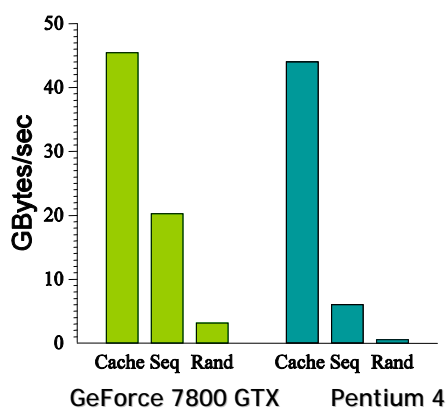
Master Pro
2006

Optimisations chart courtesy
of John Owens multimédia

15

2) Performance mémoire

Types d'accès mémoire : **Cache**, **Séquentiel**, **Aléatoire**



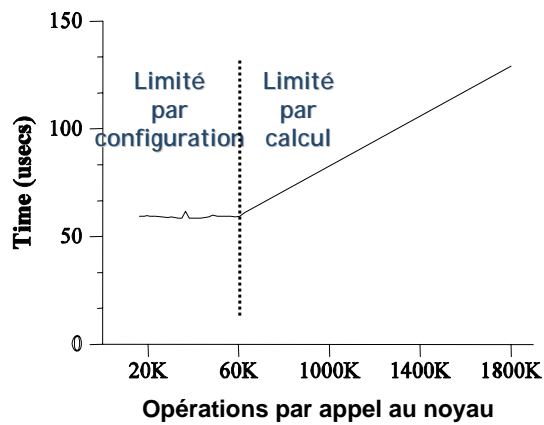
- **CPU**
 - **Gros** cache
 - **Peu** d'éléments de calcul
 - Optimisé pour la réutilisation de données **spatiale** et **temporelle**
- **GPU**
 - **Petit** cache
 - **Beaucoup** d'éléments de calcul
 - Optimisé pour l'accès données **séquentiel (flot)**

Master Pro
2006

Optimisations chart courtesy
of Ian Buck multimédia

16

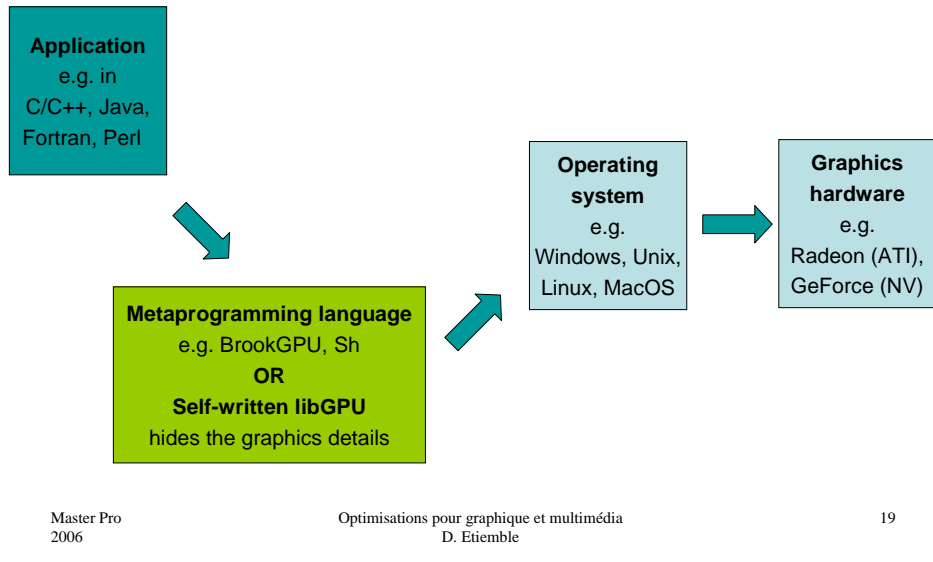
3) Coût de la configuration



GPU pour calcul scientifique

- Modèles de calcul : GPU et CPU
- **Programmation GPU**
- Algorithmes pour GPU : tris, recherches...

La programmation GPU



Les langages shader

- Les noyaux sont programmés dans un langage shader
 - Cg (NVIDIA)
 - HLSL (Microsoft, only Direct3D)
 - GLSL (OpenGL)
- Caractéristiques
 - Accès à des tableaux – Conditionnelles, boucles
 - Math – pas d'opérations sur des bits
- Faciles à apprendre
 - Les trois langages se ressemblent beaucoup

Structures de données

- CPU: Tableau 1 dimension

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|



- GPU: Tableau 2 dimensions

| | | | |
|----|----|----|----|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

$index(i,j)$

Indices are floats,
addressing array element
centers (GL) or top-left
corners (D3D).

Structure de données

Représentation des vecteurs

- Textures 2D
 - Opérations par fragment ou par pixels
 - Bande passante mémoire élevée pour les textures
 - Accès lecture-écriture, accès dépendants

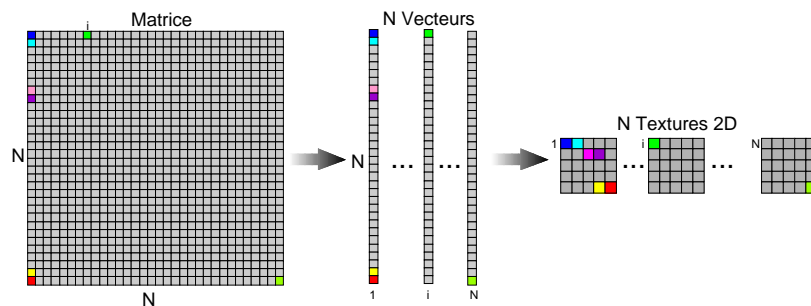
1 N



Structure de données

Représentation des matrices denses

- Matrice dense = ensemble de vecteurs colonnes
- Stocker ces vecteurs comme des textures 2D



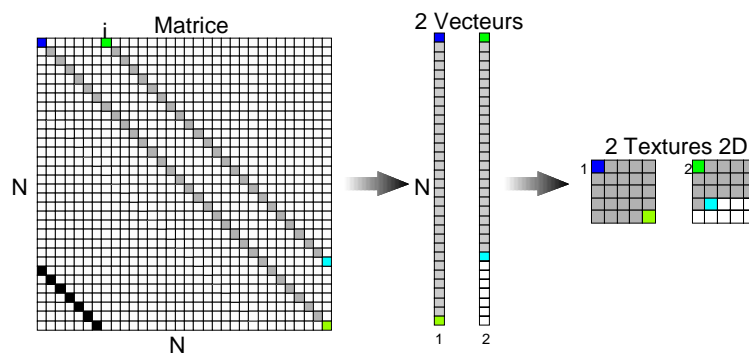
Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

Structure de données

Représentation des matrices creuses en bande

- Matrice en bande = ensemble de vecteurs diagonales



Master Pro
2006

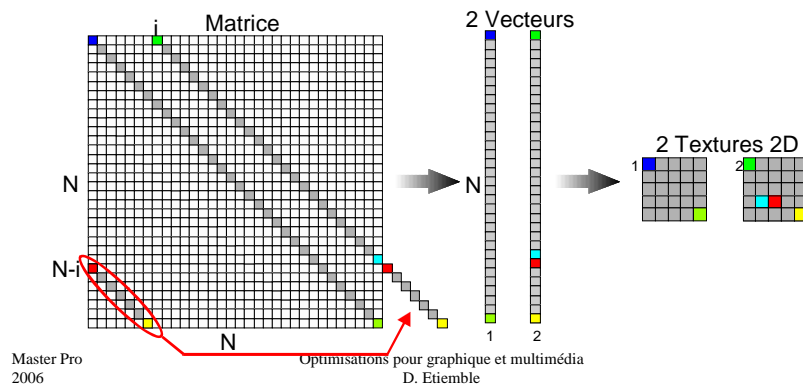
Optimisations pour graphique et multimédia
D. Etiemble

24

Structure de données

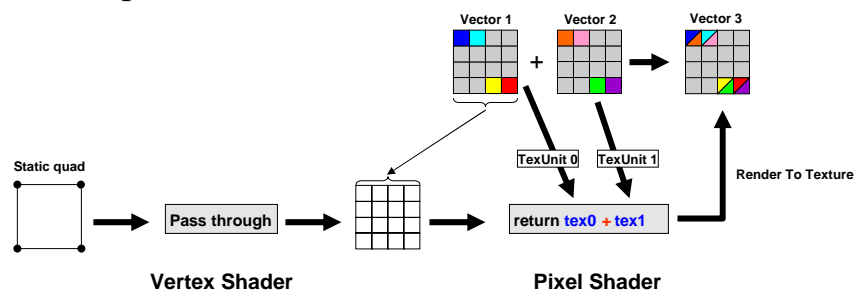
Représentation des matrices creuses en bande

- Combiner les vecteurs opposés pour gagner de la place



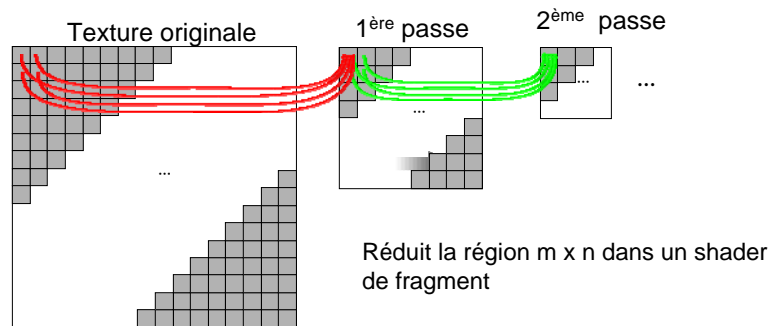
Opérations

- Opérations vecteur-vecteur
 - Ramenées à des opérations sur textures 2D
 - Codées dans des shaders de vertex/fragment
- Exemple: $\text{Vector1} + \text{Vector2} \rightarrow \text{Vector3}$



Opérations (suite)

- Opérations vecteur vecteur
 - Opération réduction pour produit scalaire



Exemple

- saxpy (de BLAS)
 - Soient deux vecteurs \mathbf{x} et \mathbf{y} de taille N et un scalaire a
 - Calculer $\mathbf{y} = \mathbf{y} + a*\mathbf{x}$
- Implémentation CPU
 - Ranger chaque vecteur dans un tableau, itérer sur tous les éléments

```
for (i=0; i<N; i++)  
y[i] = y[i] + a*x[i];
```

- Identifier le calcul dans la boucle comme un noyau
 - Uniquement le calcul dans le noyau de base
 - Calcul et contrôle sont complètement séparés

Comprendre les limitations du GPU

- Pas de lectures et écritures simultanées à la même mémoire
 - L'absence de tampons lecture – modification - écriture signifie qu'il n'y a pas besoin de contrôler des aléas lecture avant écriture
 - Pas une caractéristique manquante, mais une réalisation matérielle essentielle pour une bonne performance et un bon débit
 - saxpy: on introduit un tableau supplémentaire : $y_{\text{new}} = y_{\text{old}} + a * x$
- Accès mémoire cohérent
 - Pour un élément de sortie donné, on lit avec le même indice dans les deux tableaux d'entrée
 - Réalisé trivialement dans cet exemple

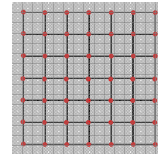
La réalisation des calculs

- Charger un programme noyaux
 - Exemples à suivre
- Spécifier les tableaux d'entrée et de sortie
 - Pseudo-code:

```
setInputArrays(yold, x);
setOutputArray(ynew);
```
- Lancer le calcul
 - GPU est avant tout un processeur graphique
 - On dessine juste ce qu'il faut

Calculer = Dessiner

- Spécifier les régions d'entrée et de sortie
 - Mappage du point de vue graphique sur les éléments de sortie du tableau, établissement des régions d'entrée
 - saxpy: les régions d'entrée et de sortie coïncident
- Générer les flots de données
 - Dessiner une certaine géométrie qui couvre tous les éléments dans le tableau de sortie
 - Dans cet exemple, un quad 4x4 à partir de 4 sommets
 - Le GPU calculera les indices du tableau de sortie à partir des sommets pour toute la région de sortie
 - Et générer le flots de données à travers les PE parallèle



Exemple

| | | | |
|----|----|----|----|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

| | | | |
|-----|-----|-----|-----|
| 101 | 102 | 103 | 104 |
| 105 | 106 | 107 | 108 |
| 109 | 110 | 111 | 112 |
| 113 | 114 | 115 | 116 |

Kernel
 $y + 0.5*x$

| | | | |
|------|----|------|----|
| | 53 | 54.5 | |
| | | | |
| | | | 68 |
| 69.5 | | | |

Réalisation des calculs

- Vue de haut niveau
 - Le noyau est exécuté simultanément sur tous les éléments dans la région de sortie
 - Le noyau connaît ses indices de sortie
 - Le dessin remplace les boucles CPU (exécution « foreach »)
 - Le tableau de sortie est en écriture seule
- Boucle de rétroaction (ping pong)
 - Le tableau de sortie peut être utilisé en lecture-seule comme entrée pour l'opération suivante

Noyaux GPU: saxpy

- Noyau CPU

```
y[i] = y[i] + a*x[i]
```

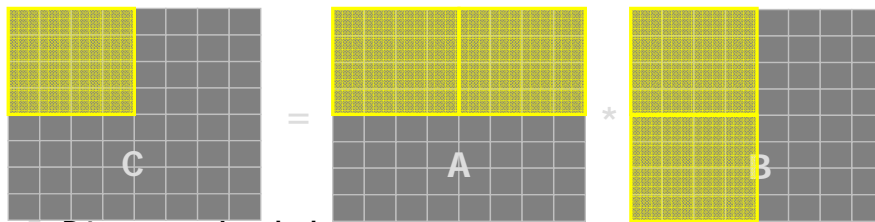
- Noyau Cg pour le GPU

```
float saxpy(float2 coords: WPOS,
uniform samplerRECT arrayX,
uniform samplerRECT arrayY,
uniform float a) : COLOR
{
float y = texRECT(arrayY,coords);
float x = texRECT(arrayX,coords);
return y+a*x;
}
```

Indices de tableau
Tableaux entrée
Assemblage
calcul

Multiplication de matrices (CPU)

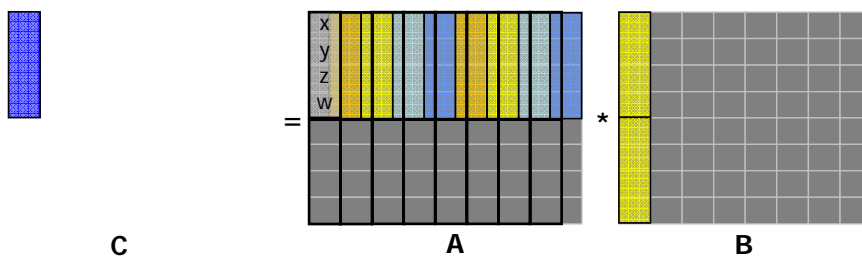
- Les algorithmes de multiplication de matrices haute performance tiennent compte des caches



Décomposer le calcul en multiplications de sous-matrices

- Charger les sous-matrices d'entrée dans le cache
- Multiplier les sous-matrices
- Ranger la sous-matrice de sortie en mémoire

Méthode 1: Rangement par colonnes



C

A

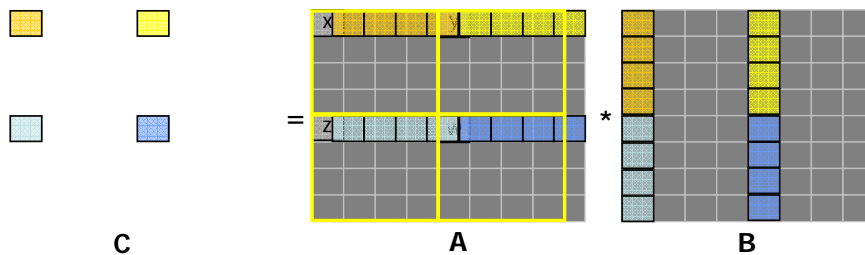
B

4 éléments par texel

Multiplications matrice 4x4 par vecteur

Larsen & McAllister [SC2001]
Moravansky [2003]

Méthode 2: rangement par sous-matrice



C

A

B

Sous-matrice 2x2 par texel

Multiplication 2x2 par sous-matrice 2x2

Hall et al. [2003]

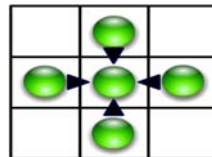
Noyaux GPU : Itération de Jacobi

• Solveur linéaire simple

- Des solveurs linéaires simples peuvent être construits avec ces techniques de base

• Exemple: Différences finies

- \mathbf{x} : vecteur d'inconnues, échantillonné avec un stencil à 5 points (déplacement)
- \mathbf{b} : partie droite
- Grille régulière équidistante
- Résolution avec l'itération de Jacobi



Noyaux GPU : Itération de Jacobi

Calcul déplacements

Assemblage valeurs

Matrice-vecteur

Étape Jacobi

```
float jacobi (float2 center : WPOS,
             uniform samplerRECT x,
             uniform samplerRECT b,
             uniform float one_over_h) : COLOR
{
    float2 left   = center - float2(1,0);
    float2 right  = center + float2(1,0);
    float2 bottom = center - float2(0,1);
    float2 top    = center + float2(0,1);

    float x_center = texRECT(x, center);
    float x_left   = texRECT(x, left);
    float x_right  = texRECT(x, right);
    float x_bottom = texRECT(x, bottom);
    float x_top    = texRECT(x, top);
    float rhs      = texRECT(b, center);

    float Ax = one_over_h *
        ( 4.0 * x_center - x_left -
          x_right - x_bottom - x_top );
    float inv_diag = one_over_h / 4.0;

    return x_center + inv_diag * (rhs - Ax);
}
```

Maximum d'un tableau

- **Opération totalement différente**
 - La sortie est un scalaire, l'entrée un tableau de longueur N
- **Approche naïve**
 - Utiliser un tableau 1x1 comme sortie, rassembler les N valeurs en une étape.
 - Problème : n'utilisera qu'un seul PE. Aucun parallélisme
 - Plein d'autres problèmes
- **Solution : la réduction parallèle**
 - Similaire aux communications globales en calcul parallèle
 - Entrelacement des régions de sortie et d'entrée
 - Même technique pour le produit scalaire, les calculs de norme, etc.

Maximum d'un tableau

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 47 | 2 | 3 | 57 | 5 | 12 | 7 | 8 |
| 10 | 20 | 6 | 13 | 14 | 15 | 16 | 17 |
| 19 | 11 | 21 | 22 | 23 | 68 | 25 | 26 |
| 38 | 29 | 64 | 31 | 32 | 33 | 35 | 34 |
| 37 | 28 | 39 | 49 | 53 | 42 | 41 | 52 |
| 46 | 1 | 48 | 40 | 61 | 51 | 44 | 43 |
| 55 | 71 | 4 | 58 | 69 | 62 | 50 | 60 |
| 30 | 65 | 66 | 67 | 24 | 59 | 70 | 56 |

| | | | |
|----|----|----|----|
| 47 | 57 | 15 | 17 |
| 38 | 64 | 68 | 35 |
| 46 | 49 | 61 | 52 |
| 71 | 67 | 69 | 70 |

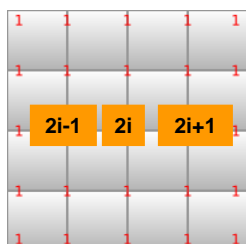
| | |
|----|----|
| 64 | 68 |
| 71 | 70 |

| |
|----|
| 71 |
|----|

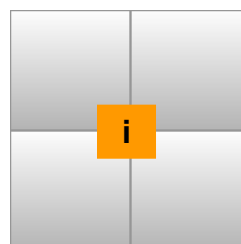
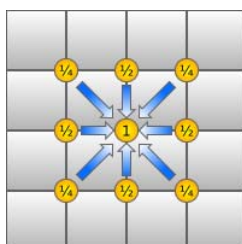
```
float maximum (float2 coords: WPOS,
               uniform samplerRECT array) : COLOR {
    float2 topleft = ((coords-0.5)*2.0)+0.5;
    float val1 = texRECT(array, topleft);
    float val2 = texRECT(array, topleft+float2(1,0));
    float val3 = texRECT(array, topleft+float2(1,1));
    float val4 = texRECT(array, topleft+float2(0,1));
    return max(val1,max(val2,max(val3,val4)));
}
```

Transferts multigrilles

- Restriction
 - Interpolation de valeurs d'un réseau fin à un réseau plus gros
 - Assemblage à partir des voisins (pondération) sur CPU et GPU.



Grain fin

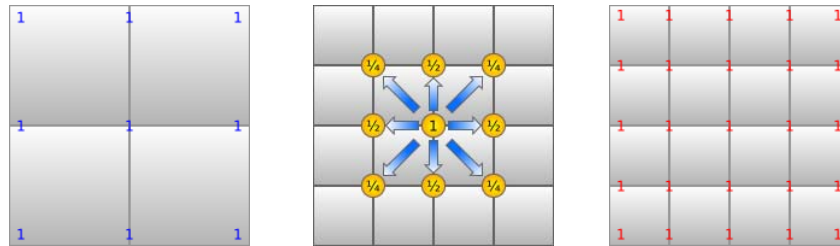


Gros grain

Transferts multigrilles

- Extension

- Génération de valeurs de gros vers fin avec un stencil
- Sur CPU : boucle sur le réseau « gros » avec DAXPY de pas 2



Transferts multigrilles

- Trois cas

- 1) Noeud fin au centre d'un élément (4 interpolants)
- 2) Noeud fin sur un arc (2 interpolants)
- 3) Noeud fin sur un noeud « gros » (copie)

- Reformuler « scatter » comme « gather » pour GPU

- Réseau fin = région de sortie
- Échantillonner avec un déplacement d'indice de 0.25

GPU pour calcul scientifique

- Modèles de calcul : GPU et CPU
- Programmation GPU
- Algorithmes pour GPU : tris, recherches...

Les tris

- Étant donné une liste non ordonnée d'éléments, produire une liste ordonnée
 - Noyau : comparaison et échange
- Les environnements de programmation GPU limitent les algorithmes utilisables
 - Tri fusion bitonique [Batcher 68]
 - Réseaux de tri équilibrés périodiques [Dowd 89]

Tri par fusion bitonique

- On construit des suites de listes bitoniques et on les trie
 - Une liste bitonique est constituée de deux listes monotones concaténées, l'une croissante et l'autre décroissante.
 - Liste A: (3, 4, 7, 8) monotone croissante
 - Liste B: (6, 5, 2, 1) monotone décroissante
 - List AB: (3, 4, 7, 8, 6, 5, 2, 1) bitonique

Le tri par fusion bitonique

3
7
4
8
6
2
1
5

Sort the Bitonic Lists

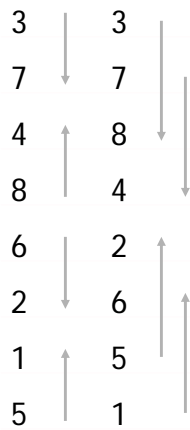
Le tri par fusion bitonique

3 ↓
7 ↓
4 ↑
8 ↑
6 ↓
2 ↓
1 ↑
5 ↑

Le tri par fusion bitonique

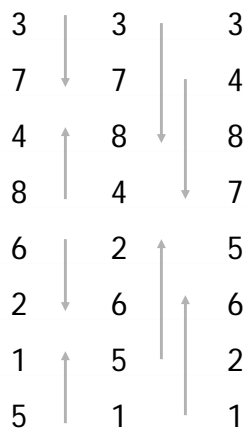
3 ↓ 3
7 ↓ 7
4 ↑ 8
8 ↑ 4
6 ↓ 2
2 ↓ 6
1 ↑ 5
5 ↑ 1

Le tri par fusion bitonique



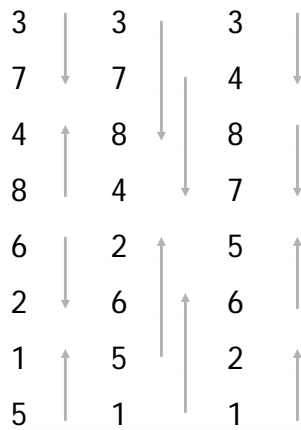
Sort the bitonic lists

Le tri par fusion bitonique



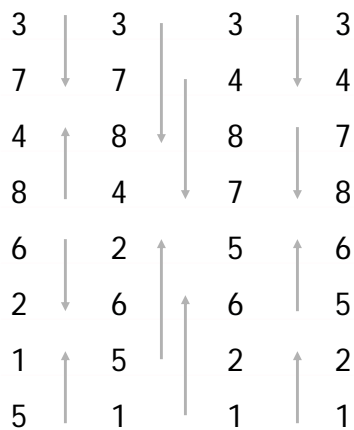
Sort the bitonic lists

Le tri par fusion bitonique



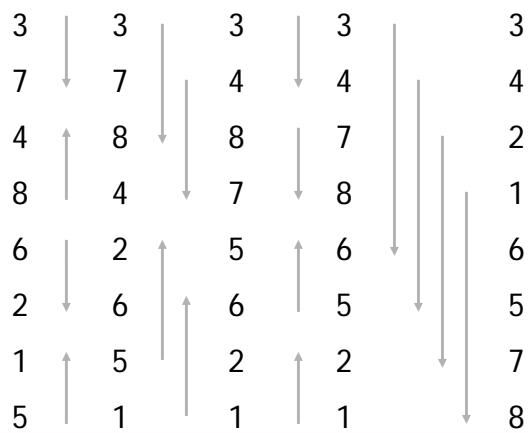
2x non stable lists (3, 4, 7, 8), (6, 5, 2, 1)
1x bitonic ()

Le tri par fusion bitonique



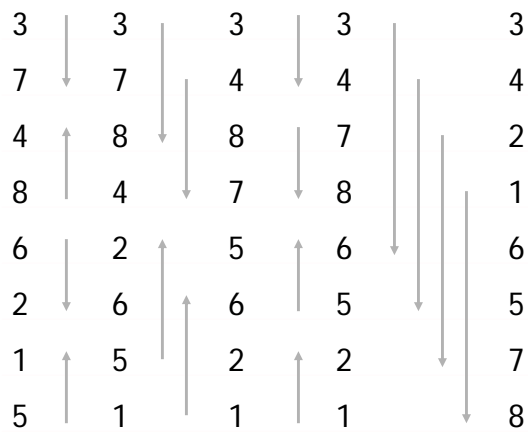
Sort the bitonic list

Le tri par fusion bitonique



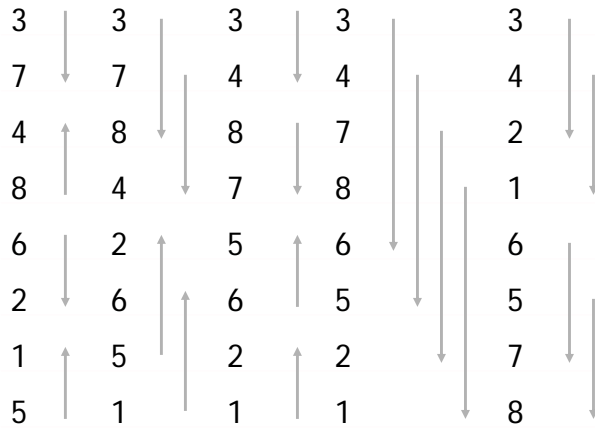
Sort the Bitonic List

Le tri par fusion bitonique

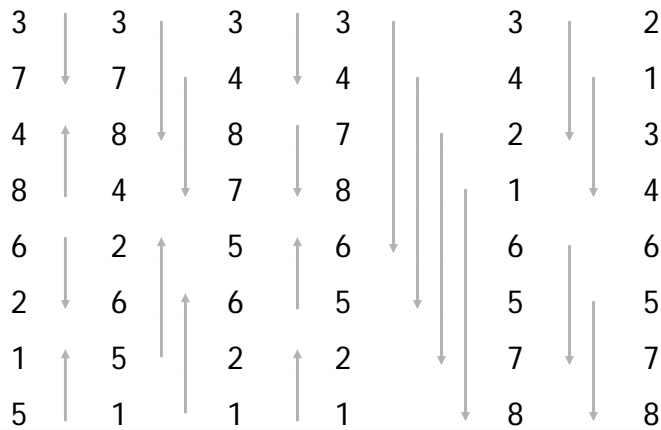


Sort the Bitonic List

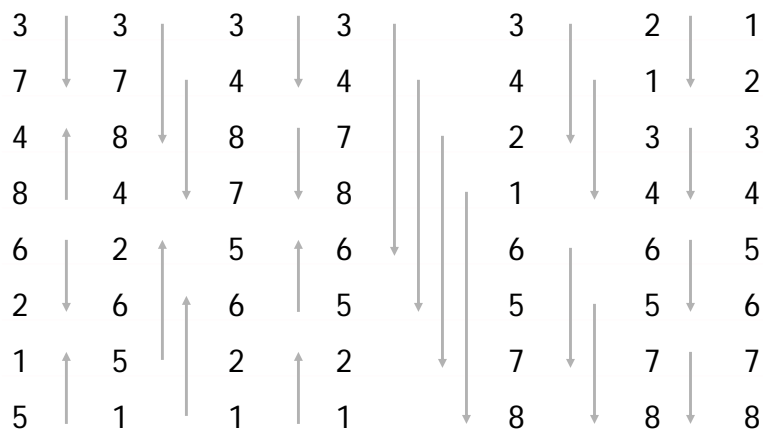
Le tri par fusion bitonique



Le tri par fusion bitonique



Le tri par fusion bitonique



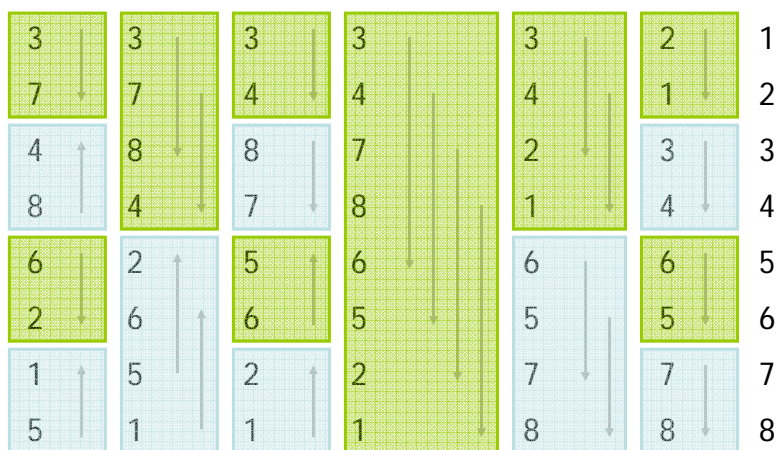
Le tri par fusion bitonique

- Passe de rendu séparé pour chaque série d'échanges
 - $O(\log^2 n)$ passes
 - Chaque passe effectue n comparaisons/échanges
 - Nombre total de comparaisons/échanges: $O(n \log^2 n)$
 - Les spécificités du GPU coûtent un facteur $\log n$ par rapport aux meilleurs algorithmes de tri sur CPU

Trier plus vite sur GPU

- Dessiner plusieurs rectangles avec le même calcul plutôt qu'un seul rectangle
 - Réduire les prises de décision dans le programme fragment
- Utiliser le processeur de sommets et l'interpolateur
 - Réduire les calculs dans le programme fragment
- Plusieurs comparaisons/échanges par appel au noyau de tri
 - Réduire la complexité du calcul

Calculs groupés

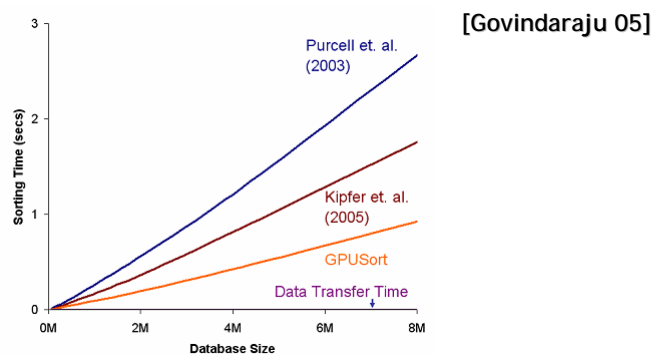


Détails d'implémentation

- Spécifier des interpolants pour des rectangles plus petits
 - Comparaison/échange “haut” et “bas”
 - Distance par rapport à l'élément à comparer
- Voir l'article de Kipfer & Westermann dans « GPU Gems 2 » et Kipfer et al. dans Graphics Hardware 04 pour plus de détails

Tri GPU

- Utilise des opérateurs de blending pour les comparaisons
- Utilise le matériel de mappage de texture pour les opérations de tri



Types de recherche

- Recherche d'un élément spécifique
 - Tri binaire
- Recherche des éléments les plus proches
 - Recherche des k voisins les plus proches
- Les deux recherches impliquent des données ordonnées

Recherche binaire

- Trouver un élément spécifique dans une liste ordonnée
- Implémenter exactement comme l'algorithme CPU
 - On suppose que le matériel supporte suffisamment de shaders longs
 - Trouver le premier élément de valeur donnée v
 - Si v n'existe pas, trouver l'élément le plus petit supérieur à v
- L'algorithme de recherche est séquentiel, mais beaucoup de recherches peuvent être exécutées en parallèle
 - Le nombre de pixels dessinés définit le nombre de recherches effectuées en parallèle
 - 1 pixel == 1 recherche

Recherche binaire

- Recherche de v0

Initialisation

4

La recherche démarre au milieu de la liste triée

$v_2 \geq v_0$ on cherche à gauche de la sous liste

Liste triée

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| v0 | v0 | v0 | v2 | v2 | v2 | v5 | v5 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Master Pro
2006

Optimisations pour graphique et multimédia
D. Etienne

67

Recherche binaire

- Recherche de v0

Initialisation

4

$v_0 \geq v_0$ on cherche à gauche de la sous liste

Etape 1

2

Liste triée

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| v0 | v0 | v0 | v2 | v2 | v2 | v5 | v5 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Master Pro
2006

Optimisations pour graphique et multimédia
D. Etienne

68

Recherche binaire

- Recherche de v_0

Initialisation

| |
|---|
| 4 |
|---|

Etape 1

| |
|---|
| 2 |
|---|

Etape 2

| |
|---|
| 1 |
|---|

$v_0 \geq v_0$ on cherche à gauche de la sous liste

Liste triée

| | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|-------|
| v_0 | v_0 | v_0 | v_2 | v_2 | v_2 | v_5 | v_5 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

69

Recherche binaire

- Recherche de v_0

Initialisation

| |
|---|
| 4 |
|---|

Etape 1

| |
|---|
| 2 |
|---|

Etape 2

| |
|---|
| 1 |
|---|

Etape 3

| |
|---|
| 0 |
|---|

A ce point, soit on a trouvé v_0 , soit on est une position trop loin à gauche

Une dernière étape à effectuer

Liste triée

| | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|-------|
| v_0 | v_0 | v_0 | v_2 | v_2 | v_2 | v_5 | v_5 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

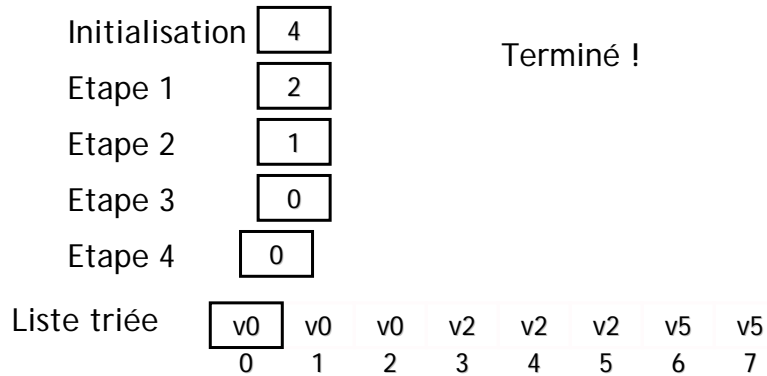
Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

70

Recherche binaire

- Recherche de v0



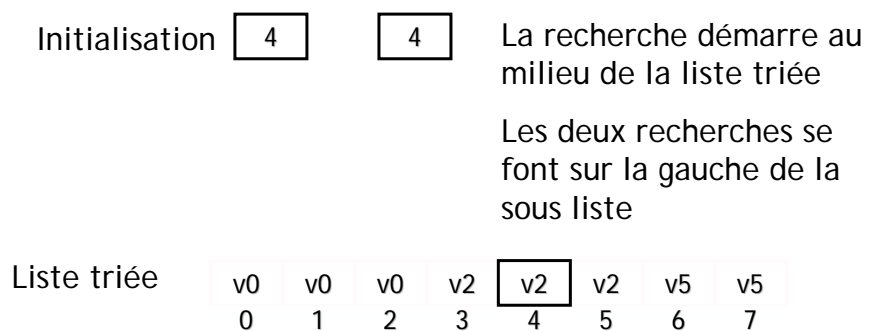
Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

71

Recherche binaire

- Recherche de v0 et v2



Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

72

Recherche binaire

- Recherche de v0 et v2

| | | | | |
|----------------|---|--|---|--|
| Initialisation | 4 | | 4 | La recherche de v0 continue comme avant |
| Etape 1 | 2 | | 2 | |

La recherche de v2 est allée trop loin et on revient à droite

Liste triée

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| v0 | v0 | v0 | v2 | v2 | v2 | v5 | v5 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Recherche binaire

- Recherche de v0 et v2

| | | | | |
|----------------|---|--|---|---|
| Initialisation | 4 | | 4 | On a trouvé la bonne valeur de v2 mais on cherche encore v0 |
| Etape 1 | 2 | | 2 | |
| Etape 2 | 1 | | 3 | |

Les deux recherches continuent

Liste triée

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| v0 | v0 | v0 | v2 | v2 | v2 | v5 | v5 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Recherche binaire

- Recherche de v0 et v2

| | | | | |
|----------------|---|----|---|---------------------------|
| Initialisation | 4 | v0 | 4 | Maintenant, on a le bon |
| Etape 1 | 2 | | 2 | v0, mais on est trop loin |
| Etape 2 | 1 | | 3 | pour v2 |
| Etape 3 | 0 | | 2 | Une étape finale va en |
| | | | | tenir compte |

| | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|
| Liste triée | v0 | v0 | v0 | v2 | v2 | v2 | v5 | v5 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Recherche binaire

- Recherche de v0 et v2

| | | | | |
|----------------|---|----|---|---------------------------|
| Initialisation | 4 | v0 | 4 | Terminé ! v0 et v2 sont à |
| Etape 1 | 2 | | 2 | la bonne place |
| Etape 2 | 1 | | 3 | |
| Etape 3 | 0 | | 2 | |
| Etape 4 | 0 | | 3 | |

| | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|
| Liste triée | v0 | v0 | v0 | v2 | v2 | v2 | v5 | v5 |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

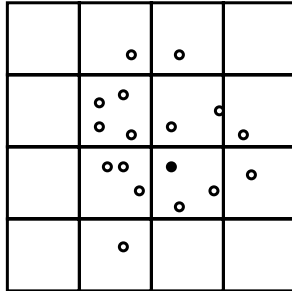
Recherche binaire

- Une seule passe de rendu
 - Chaque pixel dessiné effectue une recherche indépendante
- $O(\log n)$ étapes

Recherche des plus proches voisins

- Soit un point p . Trouver les k points les plus proches de p dans un ensemble de données
- Avec le CPU, c'est fait facilement avec un "tas" ou une liste de priorité
 - On peut ajouter ou retirer des voisins quand la recherche progresse
 - Sur GPU ???
- Grille kNN
 - On peut seulement ajouter des voisins...

Algorithme de grille kNN



- Point considéré
- Voisin candidat
- Voisins trouvés

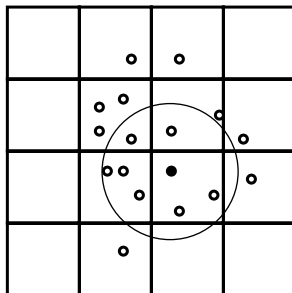
On veut 4 voisins

Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

79

Algorithme de grille kNN



- Point considéré
- Voisin candidat
- Voisins trouvés

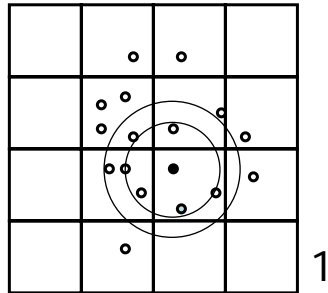
On veut 4 voisins

Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

80

Algorithme de grille kNN



- Point considéré
- Voisin candidat
- Voisins trouvés

On veut 4 voisins

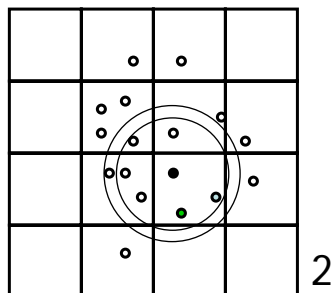
Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

81

- Si le nombre courant de voisins trouvés est inférieur au nombre voulu, on augmente le rayon de recherche

Algorithme de grille kNN



- Point considéré
- Voisin candidat
- Voisins trouvés

On veut 4 voisins

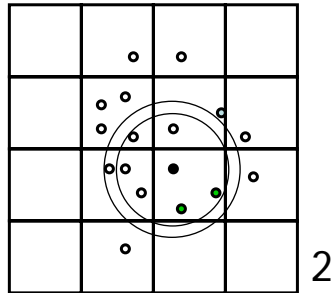
Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

82

- Si le nombre courant de voisins trouvés est inférieur au nombre voulu, on augmente le rayon de recherche

Algorithme de grille kNN



- Point considéré
- Voisin candidat
- Voisins trouvés

On veut 4 voisins

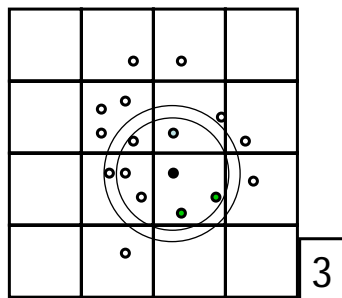
Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

83

- Ne pas ajouter de voisins à l'extérieur du cercle de rayon maximal
- Ne pas augmenter le rayon de recherche quand le voisin est à l'extérieur de cercle de rayon maximal

Algorithme de grille kNN



- Point considéré
- Voisin candidat
- Voisins trouvés

On veut 4 voisins

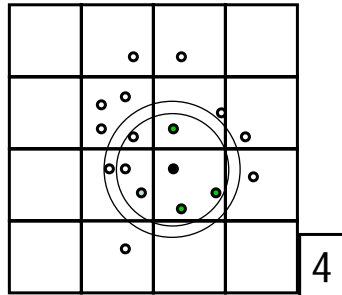
Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

84

- Ajout de voisins dans le rayon de recherche

Algorithme de grille kNN

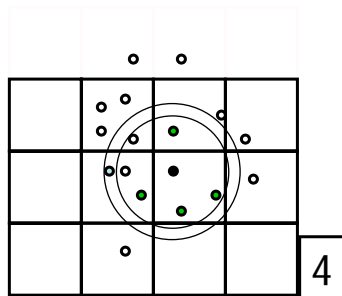


- Ajout de voisins dans le rayon de recherche

- Point considéré
- Voisin candidat
- Voisins trouvés

On veut 4 voisins

Algorithme de grille kNN

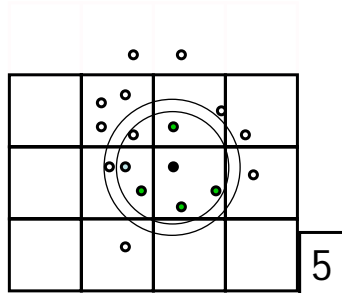


- Ne pas étendre le rayon de recherche si suffisamment de voisins ont été trouvé

- Point considéré
- Voisin candidat
- Voisins trouvés

On veut 4 voisins

Algorithme de grille kNN



- Ajouter des voisins dans le rayon de recherche

- Point considéré
- Voisin candidat
- Voisins trouvés

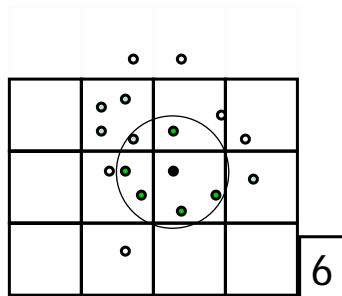
On veut 4 voisins

Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

87

Algorithme de grille kNN



- Visiter tous les autres points accessible dans le rayon de recherche déterminé
- Ajouter les voisins dans le rayon de recherche

- Point considéré
- Voisin candidat
- Voisins trouvés

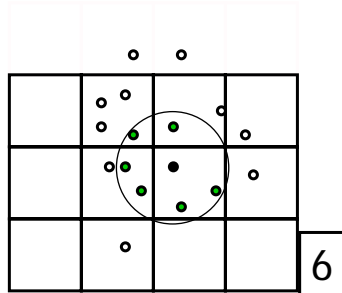
On veut 4 voisins

Master Pro
2006

Optimisations pour graphique et multimédia
D. Etiemble

88

Résumé : Algorithme de grille kNN



- Point considéré
- Voisin candidat
- Voisins trouvés

On veut 4 voisins

- Trouver tous les voisins à l'intérieur d'une sphère centrée autour du point considéré
- Peut contenir plus que les k voisins les plus proches