École thématique ARCHI 07
19-23 mars 2007

Processeurs Graphiques

David Defour

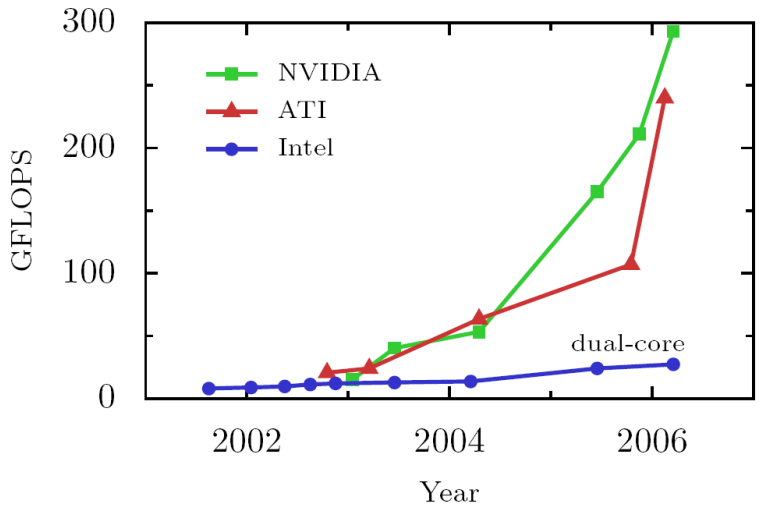
DALI, Université de Perpignan

UPVD

Plan

Partie 1 : Introduction

Puissance de calcul flottant 32 bits



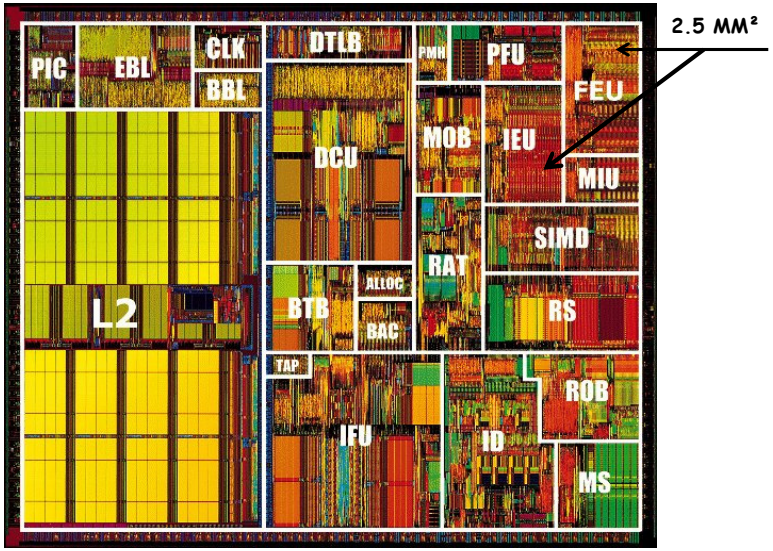
Source : "A Survey of General-Purpose Computation on Graphics Hardware",
J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell

Aujourd'hui le calcul ne coûte pas cher…

- Utilisation du silicium :
  - Communication (Bus)
  - Contrôle (Exécution d'instruction, ooo)
  - Mémoire (Cache, registres)
  - Calcul (Unités de calcul)

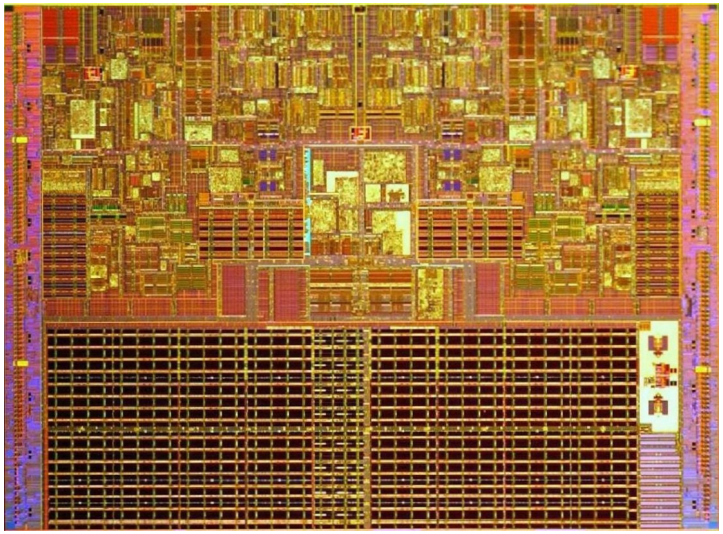
Si l'on ramène tout ça à la taille d'une puce …

Pentium III, 256Ko cache, 0.18 µm, 28.1 M transistors, 90 mm²



Source : www.sandpile.org

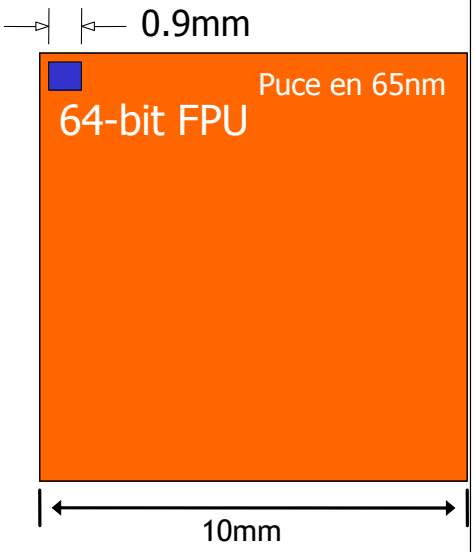
Pentium M, 2Mo cache, 65nm, 151.6 M transitors, 91 mm²



Source : www.sandpile.org

Concrètement …

- Le calcul ne coûte pas cher
  - FPU du Pentium III : ~2%
  - FPU de l'Intel Core 2 : <1%
- La mémoire augmente mais la latence aussi
  - Pentium III, cache L2 : 256Ko / 4 cycles
  - Intel Core 2, cache L2 : 2Mo / 14 cycles
- Le transfert de donnée devient très coûteux
  - 0.18 µm : 12.1 ps/mm
  - 65 nm : ~30-60 ps/mm



Introduction

- Utiliser les transistors disponibles pour calculer plus et plus vite
- Pourquoi plus de calcul ?
  - Cryptographie
    - Opérations entières (XOR, <<, …)
    - Opérations sur les grands entiers
  - Calcul scientifique
    - Multiplication matricielle
    - Simulation
  - Multimédia
    - Traitement de l'image
    - Traitement du son
    - Traitement de l'information

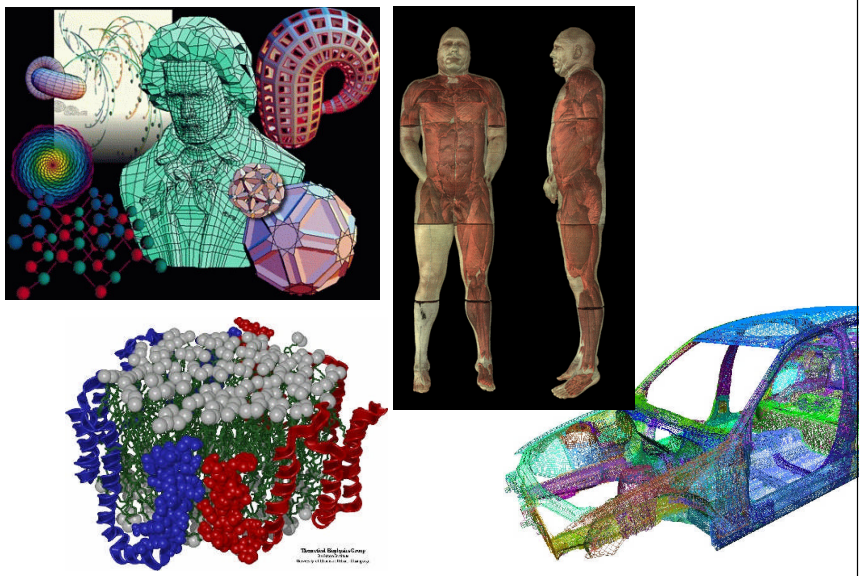
Comment calculer plus ?

- 2 approches :
  1. Mécanismes pour exploiter le parallélisme de données
     - Jeux d'instructions SIMD (MMX, SEE, SEE2)
  2. Architectures dédiées
     - Physics Processing Unit (Ageia)
     - Processeurs réseaux (AMD, Infineon, Motorola, …)
     - Cell (IBM, Sony, Toshiba)
     - Processeurs graphiques (ATI/AMD, Nvidia)

Utilisation de la puissance de calcul : Films

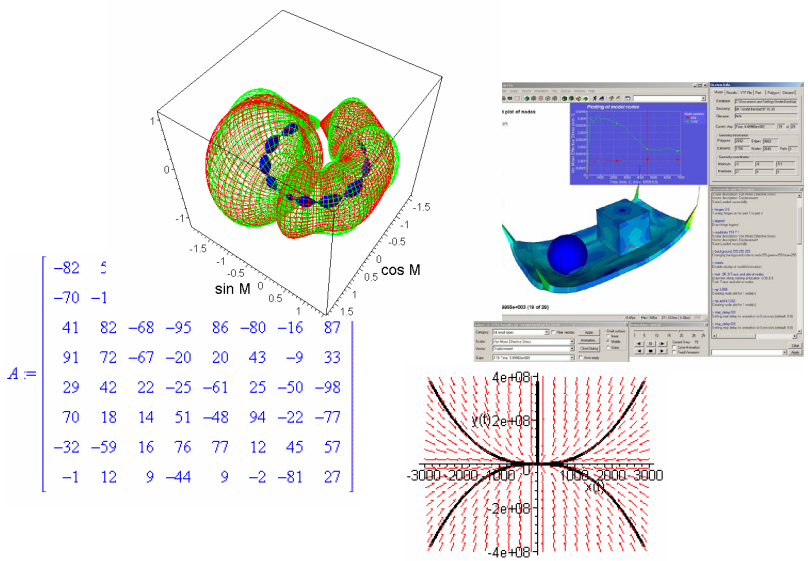
Applications des GPU: Jeux

Applications des GPU : Visualisation, CAD



Applications des GPU : Réalité virtuelle

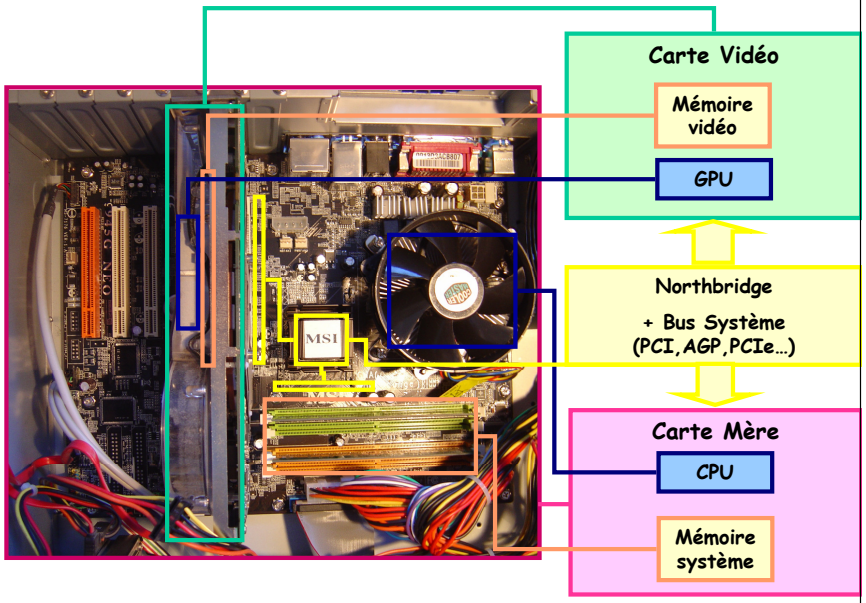
Applications des GPU : Calcul scientifique



GPU comme coprocesseur

- **Exemples de speed-up par rapport au CPU :**
  - **FFT : x 4**
    « High Performance GPU-based FFT Library », Govindaraiu et al.
  - **Décomposition LU : x3 - 5**
    « LU-GPU: Algorithms for dense Linear Systems on Graphics Hardware », Galoppo et al.
  - **Finite Difference Time Domain(FDTD, maxwell) : x30 - 50**
    http://www.emphotonics.com/fastfdtd.html
  - **Recherche, tri dans base de données : x4**
    « GPUTeraSort: High Performance Graphics Coprocessor Sorting for Large Database Management », Govindaraiu et al.
  - **Pliage de protéine : x 40**
    http://folding.stanford.edu/

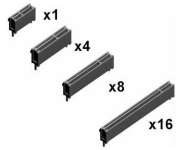
GPU dans le système



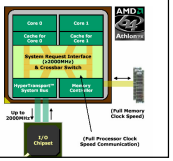
Bus : les autoroutes de la carte mère

- PCI
  - PCI cadencé à 33 MHz en 32 bit : 125 Mo/s maximum
  - PCI cadencé à 33 MHz en 64 bit : 250 Mo/s maximum
  - PCI cadencé à 66 MHz en 32 bit : 250 Mo/s maximum
  - PCI cadencé à 66 MHz en 64 bit : 500 Mo/s maximum

- AGP
  - AGP 1x (250 Mo par seconde),
  - AGP 2x (500 Mo par seconde, deux transferts par cycle au lieu d'un)
  - AGP 4x (1 Go par seconde),
  - AGP 8x (2 Go/s maximum)

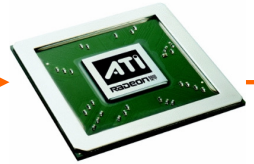


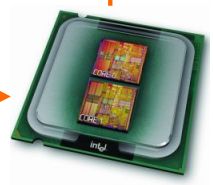
- PCI Express
  - PCI Express 1X (250 Mo/s)
  - PCI Express 32X (8 Go/s)
- HyperTransport
  - HyperTransport 2.0 (22,4 Go/s, 1,6 GHz)
  - HyperTransport 3.0 (41,6 Go/s, 2,6 Ghz)



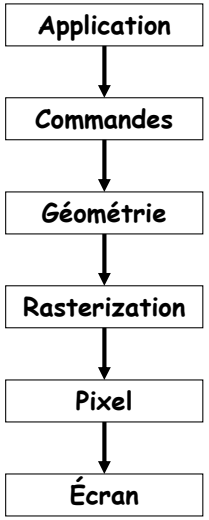
Communications entre CPU et GPU

- Comment passer d'ici à là ?

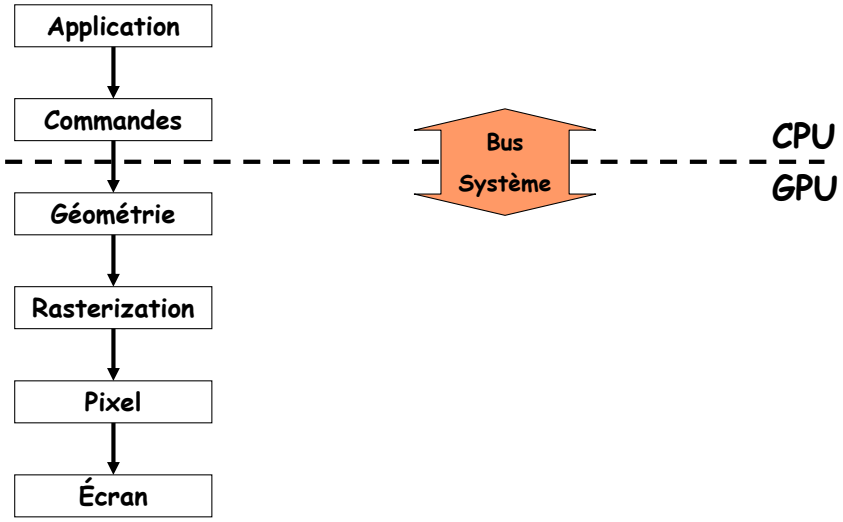




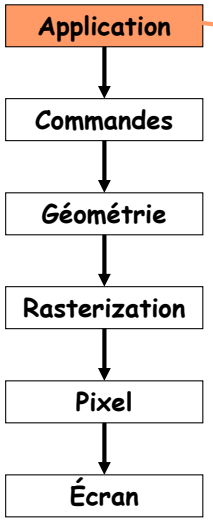
Le pipeline graphique



Le pipeline graphique

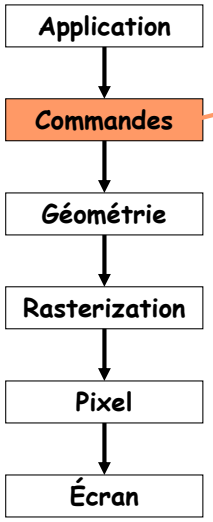


Le pipeline graphique



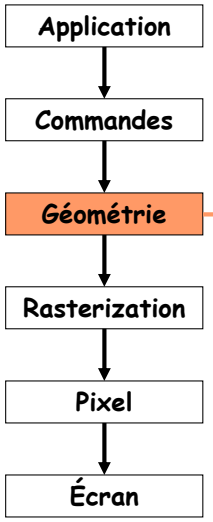
- Gears of war :
  - Définition du jeu
  - Réseaux
  - Interface avec l'utilisateur
  - Physiques
  - IA

Le pipeline graphique



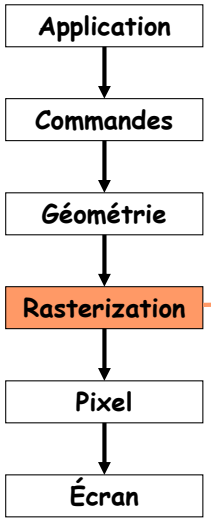
- Gears Of War :
  - Envoi des commandes DirectX
- Drivers DirectX :
  - Traitement des commandes DirectX
  - Envoi des instructions au GPU

Le pipeline graphique



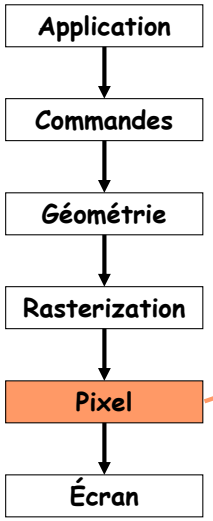
- GPU :
  - Transformation des Sommets
  - Calcul de la lumière
  - Elimination des objets en dehors de la scène
  - Assemblage des primitives

Le pipeline graphique



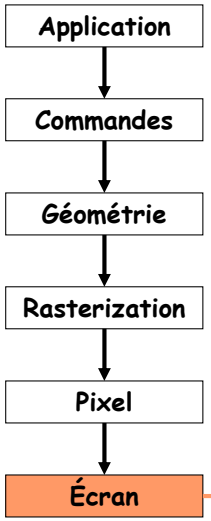
- GPU :
- Conversion des triangles en Pixel
- Interpolation des coordonnées de texture
- Interpolation de couleur

Le pipeline graphique

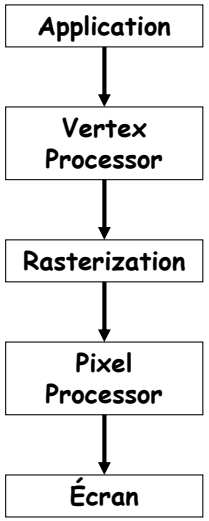


- GPU :
- Application de la texture
- Test de profondeur
- Mélange des couleurs

Le pipeline graphique

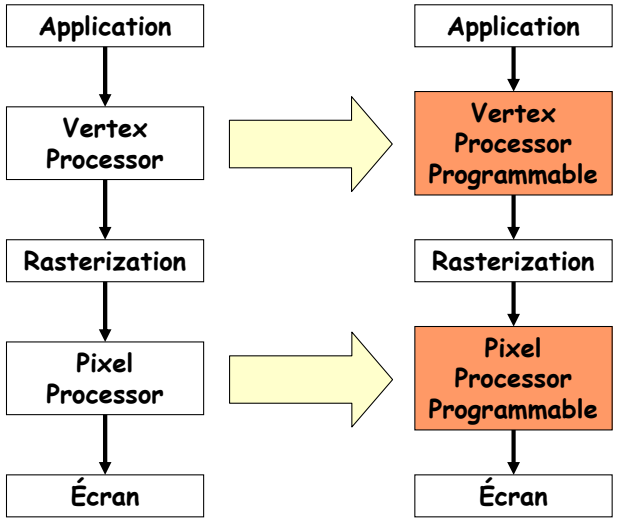


Pipeline graphique traditionnel

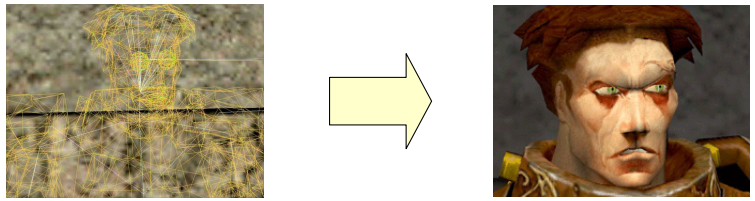


- Fonctions pré-définies
- Configurable en utilisant les états OpenGL / DirectX
- Fonctionnalités limitées

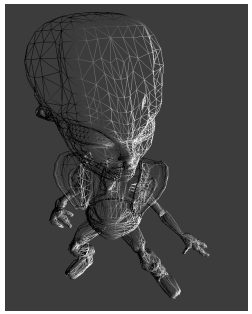
Nouveau Pipeline graphique



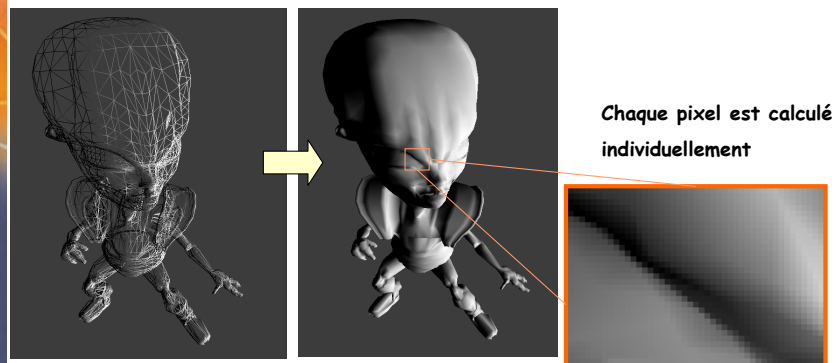
Vertex Shaders



- Les vertex shaders permettent d'animer/modifier les caractéristiques des sommets
- Calcul de la lumière
- Ils sont flexibles et rapides



Pixel Shaders



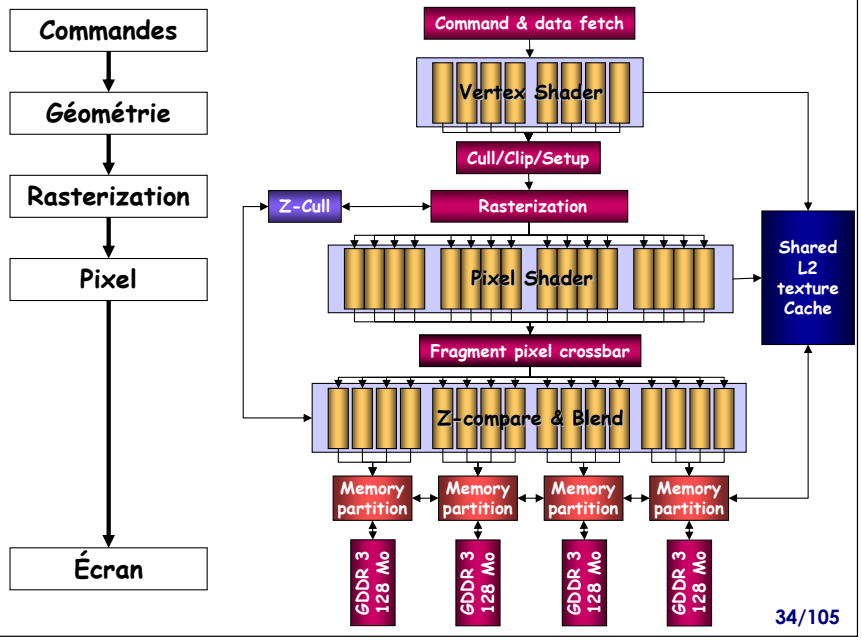
- Les pixels shaders ont une vue partielle des pixels voisins.

PARTIE 2 :

Fonctionnement interne du GPU

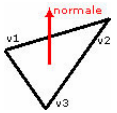
Partie 2 : Fonctionnement interne du GPU

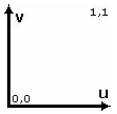
Bloc diagramme d'un GPU



Vertex / Vertices

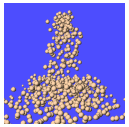
- Permettent de créer des formes géométriques
- Vertex {x, y, z, w}
  - Coordonnées finales : {x/w, y/w, z/w}
  - W est le caractère homogène généralement = 1
- + d'autres informations
  - Couleur diffuse : {r,g,b,a}
    Couleur du vertex lorsque celui-ci est éclairé par une lumière blanche
  - Couleur spéculaire :
    Couleur réfléchie par le vertex
  - Normale à la surface :
    Permet le calcul d'éclairage
  - Coordonnées de textures (u, v)
- Taille mémoire croissante ~ jusqu'à 256 octets
  - {x,y,z,w} : 16, {r,g,b,a} : 4, normale : 12 …



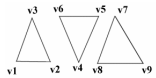


Données primitives

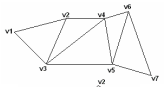
- Formes géométriques simples
  - Point (effet de particule)



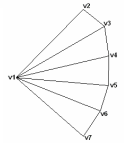
  - Ligne
  - Triangle



  - Triangle fan



  - Triangle strip



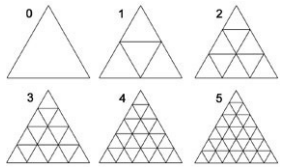
  - Tampons d'indexation
    Réutilisation des vertex en utilisant des indices

Format de représentation des données

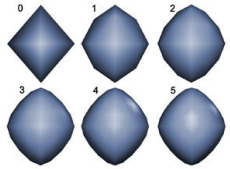
- **Virgule fixe** 123.45678
  - 8, 16, 32
  - Signé, non signé
  - Compression de plusieurs données dans un seul nombre (combinaison de 5,6,5 ; 1,5,5,5 ; …)
- **Virgule flottante** $1.12345678^{e\,2}$
  - fp16, fp24, fp32
  - Support partiel de la norme IEEE-754

2. Tesselation

- **Principe :**
  Augmenter le nombre de vertices pour augmenter la finesse de la surface.
- **Interpolation pour arriver à une résolution donnée**
  - Dépend de la position de l'objet (près, distant)
  - Calcul à partir de la Position du vertex + normale



Tesselation point-normal d'ATI

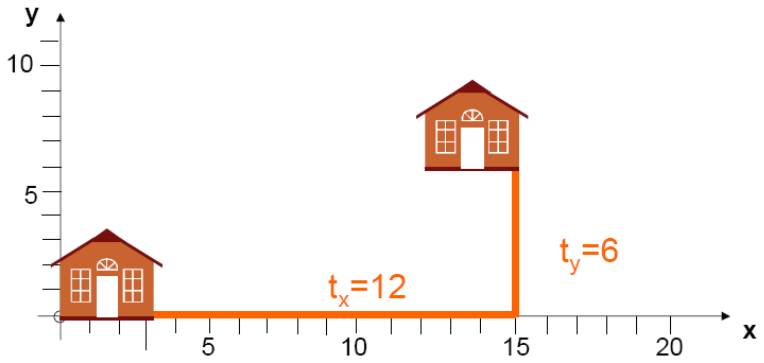


Tesselation d'un modèle complexe

3. Géométrie, lumière et vertex shader

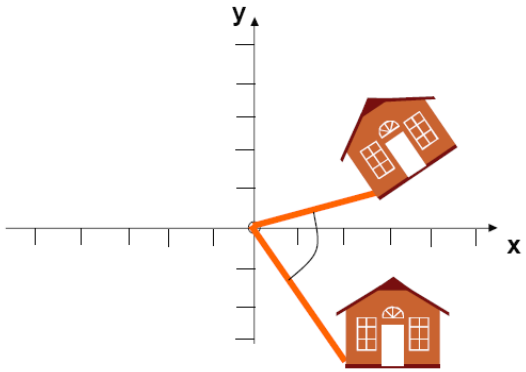
- **Objectif : Transformations géométriques + Lumière**
  - Rotation, translation
  - Transformation de système de coordonnées
  - Changer la forme d'un objet
  - Changer la position d'un objet dans une scène
  - Dupliquer des objets
  - Calcul de projection pour les caméras virtuelles
  - Animations
- **Solutions :**
  - Vertex processor fixe
  - Vertex processor programmable

Translation



$$\varphi\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow \varphi\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotation



$$\varphi \begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \varphi \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Mise à l'échelle

- **Uniforme**

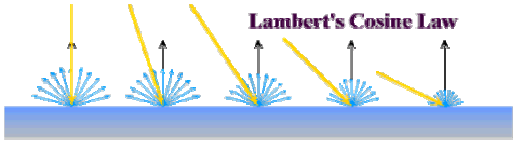
$$M = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- **Déformante**

$$M = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

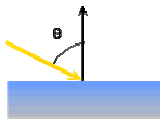
Lumière : Diffusion

- **Loi de Lambert :**
  L'intensité lumineuse réfléchie est proportionnelle au cosinus de l'angle d'incidence de la lumière



- **Calcul :**
  - I_{light} : intensité du rayon lumineux
  - $I_{diffuse}$: intensité du rayon diffusé
  - K_d : facteur de « diffusion »
  - n : vecteur normal à la surface **normalisé**
  - l : vecteur de la lumière **normalisé**

$$I_{diffuse} = k_d I_{light} \cos\theta$$

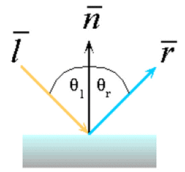


$$I_{diffuse} = k_d I_{light} (\bar{n} \cdot \bar{l})$$

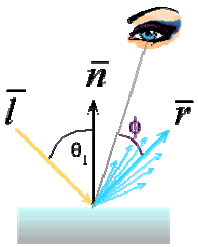
Lumière : Réflexion (miroir)

$$n_l \sin\theta_l = n_r \sin\theta_r$$

- **Miroir parfait :**
  - $\theta_l = \theta_r$



- **Model d'illumination de Phong**
  - n_{shiny} : scalaire entre 0 et 1 qui définit la « réflexivité » du matériau



$$I_{specular} = k_s I_{light} \cos^{n_{shiny}} f$$

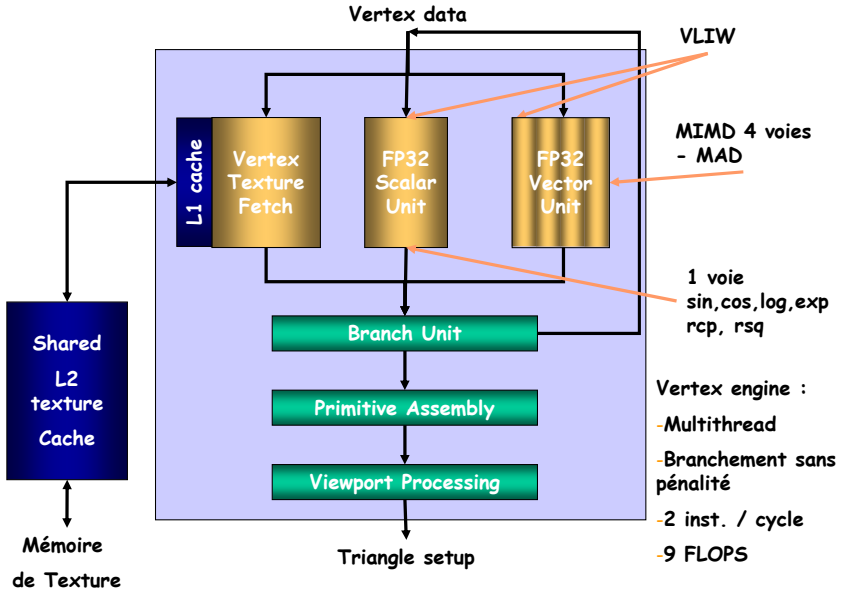
Récapitulatifs des opérations

- Rotation :
  - Calcul de sinus, cosinus
- Mise à l'échelle :
  - Multiplication, addition de matrice/vecteur
- Normalisation de vecteur :
  - Calcul de la longueur : $L = \sqrt{x^2+y^2+z^2}$
  - Division de chaque composante par L

Les vertex fixes

- Existent depuis longtemps
- Très efficaces pour réaliser les opérations de base (translation, rotation, …)
- Contraintes :
  - Algorithmes paramétrables mais non modifiables
  - Couleur au format RGBA
  - Coordonnées des vertex XYZW
  - …

Vertex Shader programmable



Unités d'exécution des shaders : Unité MAD : Multiply and accumulate

- L'unité MAD prend en entrée des flottants 32 bits (fp32) et fournit un résultat fp32
- Elle est capable d'effectuer les opérations suivantes :
  - FADD, FMUL, FMAD
- Pipelinée, mais pas optimisée pour la latence
- FADD et FMUL proche du standard IEEE *
  - Les dénormalisés sont arrondis à 0
  - Les cas spéciaux sont gérés correctement
  - Les arrondis héritent d'un passé original
- FMAD différent du FMA
  - Le produit intermédiaire est gardé avec moins de précision
  - Pour le calcul graphique c'est suffisant et cela permet de doubler le débit à faible coût.

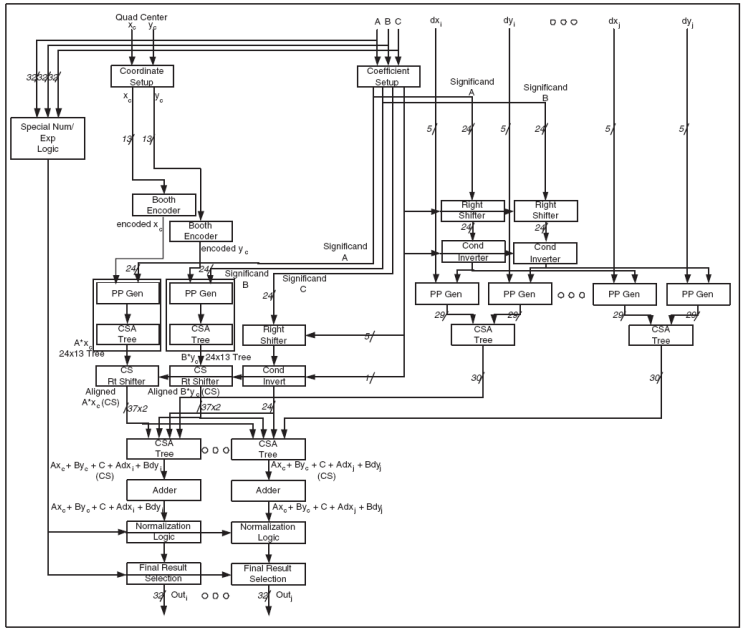
* : Caractéristiques arithmétiques des processeurs graphiques, M. Daumas, G. Da Graca, D. Defour, *SympA* 2006.

Évaluation des fonctions spéciales sur GPU Nvidia

Function	Input Interval	M	Configuration	Accuracy (good bits)	Ulp error	% exactly rounded	Monotonic	Lookup table size
1/X	[1,2)	7	26,16,10	24.02	0.98	87%	Yes	6.50Kb
1/sqrt(X)	[1,4)	6	26,16,10	23.40	1.52	78%	Yes	6.50Kb
2^X	[0,1)	6	26,16,10	22.51	1.41	74%	Yes	3.25Kb
$\log_2 X$	[1,2)	6	26,16,10	22.57	n/a	n/a	Yes	3.25Kb
Sin/cos	[0,pi/2)	6	26,15,11	22.47	n/a	n/a	No	3.25Kb
Total								22.75Kb

- Évaluation basée sur interpolation plus tabulation ou évaluation minimax
- Pleinement pipeliné

Interpolateur multifonction
(Pineiro and Oberman, ARITH 17, 2005)



Vertex programmable : les registres

- REMARQUE : On ne n'intéresse qu'aux shaders V3.0
- Registres de 128 bits semblables aux registres SSE
  - 4 nombres flottants sur 32 bits
- Registres temporaires
  - Contiennent les résultats d'opérations effectuées dans le vertex shader
  - 32 registres en lecture/écriture avec 3 ports de lecture
- Registres d'entrée
  - 16 registres en lecture seule
  - Contiennent la position des vertices, la couleur, normale …

Vertex programmable : les registres

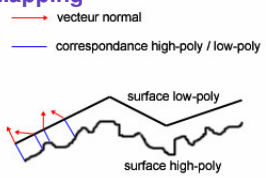
- Registres constant flottant
  - Définissent des valeurs qui ne changeront pas d'un vertex à l'autre
  - Contiennent les matrices de transformation
  - Plus de 256 registres en lecture seule
- Registres constant booléen
  - 16 registres
- Registres constant entier
  - 16 registres
- Registres d'échantillonnage
  - 4 registres d'accès aux textures
  - Accès à 4 textures différentes

Vertex programmable : les instructions

- Nombres très limités
- Permettent de réaliser les transformations géométriques
- Instructions de contrôle de flux : (shaders v3.0)
  - Boucles
  - Branchements conditionnels
  - Prédication
- Accès aux textures
  - Création d'un relief plus complexe par des modifications liées à la texture.

Exemple : Normal Mapping

- R => X
- G => Y
- B => Z



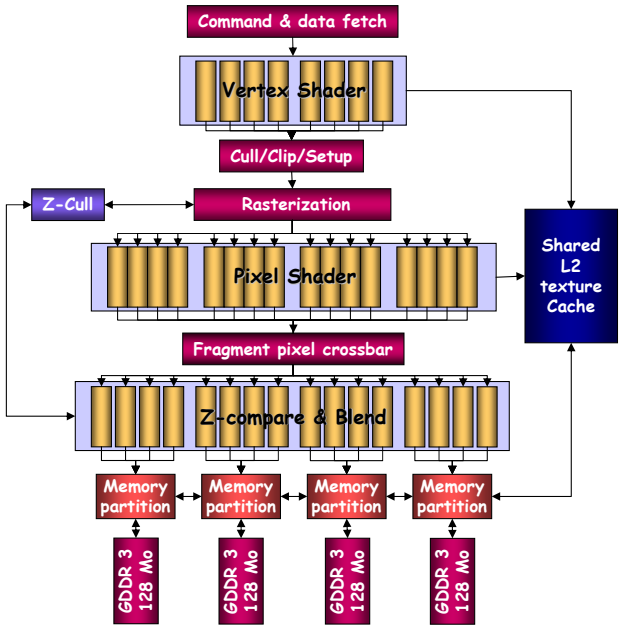
Caractéristiques dictées par DirectX 9c

- Registres des vertex shaders V3.0

Nom	Nombre	R / W	# de port de lecture
Input register	16	R	1
Temporary register	32	R / W	3
Constant float register	>256	R	1
Constant boolean register	16	R	1
Constant integer register	16	R	1
Sampler	4	R	1
Adress register	1	R / W	1
Loop counter register	1	R	1
Predicate register	1	R / W	1
Output register	12	W	-

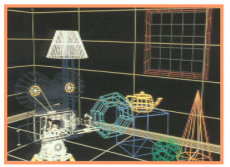
- Nombre d'instructions exécutées :
  - Statique : 512
  - Dynamique : 65 536

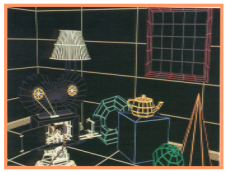
Bloc diagramme d'un GPU



4. Culling/Clipping

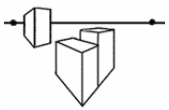
- Élimination des données 3D non visibles à l'écran
- Évite de faire du calcul pour des données qui n'apparaîtront pas dans le résultat final
- Culling :
  - Rejet basé sur l'ordre des vertices qui détermine les faces avant et arrière
- Clipping :
  1. Rejet si vertex en dehors de l'écran
  2. Utilisation de plan de clipping
- Rasterisation
  - Construction des pixels à partir des vertices





5. Rasterization : calcul de la perspective

- **Objectif :**
  - Calcul d'une projection Orthographique (plan 2D) d'un objet 3D
- **Technique :**
  - En pratique, il suffit d'enlever la coordonnée de profondeur z
    Car si une droite est parallèle en 3D, elle le reste en 2D
- **Problème :**
  - Et la perspective ???

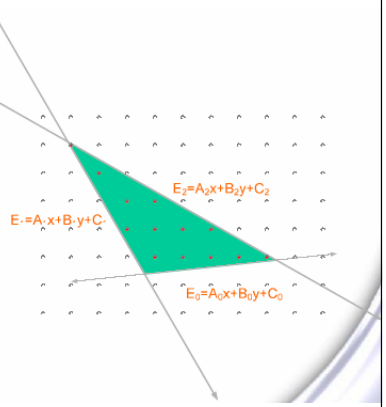


- **Solution :**
  - Division de chaque composante par w en utilisant la réciproque

[x/w y/w z/w 1]

5. Rasterization

- **Pour un triangle donné, il faut identifier les pixels appartenant à ce triangle.**
- **Échantillonnage de point :**
  - Un pixel appartient à un triangle ssi le centre du pixel est à l'intérieur du triangle
  - On évalue 3 équations de la forme E=Ax+By+C, où E=0 est exactement sur la ligne, E>0 est à l'intérieur du triangle
  - Le but est d'implémenter ces équations avec suffisamment de précision, en minimisant la latence, le délai et la surface

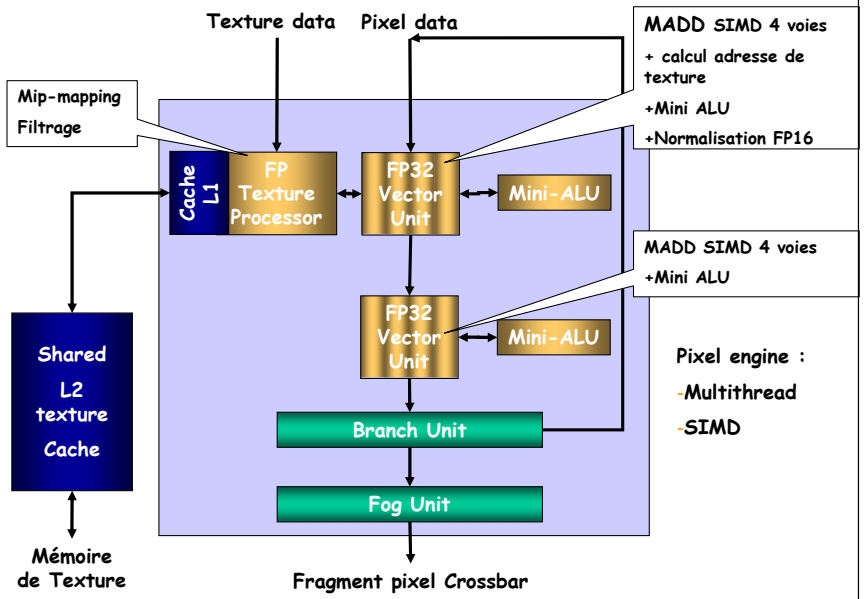


Source : « GPUs: Applications of computer arithmetic in 3D graphics », Stuart Oberman, RNC7

6. Pixel shader

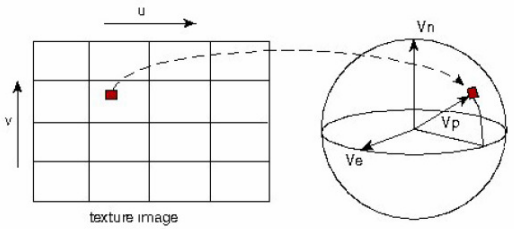
1. **Application d'une texture**
   - Accès à la texture via les coordonnées U, V
   - Une carte peut stocker beaucoup de texture
   - Seul un faible nombre peut être utilisé simultanément (4-8)
2. **Mélange des textures**
   - Opérations différentes appliquées à chaque texture
     ex : couleur du pixel X couleur de la texture N°1
     alpha du pixel X couleur de la texture N°2

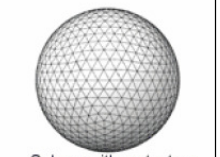
Pixel Shader

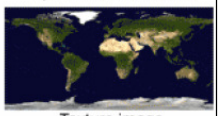


Mappage de texture

- On associe un point d'une image à un point géométrique
- Mélange entre les données de la couleur de texture avec les couleurs interpolées



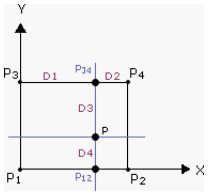

Sphere with no texture


Texture image


Sphere with texture

Accès aux textures

- Sur les GeForce 6 et 7
  - Calcul de l'adresse de la texture dans le premier MAD
  - + Unité d'accès et de filtrage (bilinéaire, trilinéaire, anisotropic, + mip - map)
- Filtrage bilinéaire
  - $P_{1,2} = d_1 * P_1 + d_2 * P_2$
  - $P_{3,4} = d_1 * P_3 + d_2 * P_4$
  - $P = d_4 * P_{1,2} + d_3 * P_{3,4}$
  - avec $d_2 = 1 - d_1$ et $d_4 = 1 - d_3$



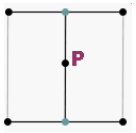
Les textures (suite)

- Le mip – map
  - Principe des poupées russes
  - Diminue l'effet d'aliasing généré lors de transformations
  - Diminue la quantité de mémoire nécessaire pour effectuer le mapping de texture

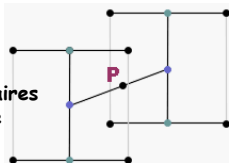
Les textures (suite et fin)

- Filtrage trilinéaire
  - Interpolations linéaires sur les 3 dimensions

Bilinéaire :
4 points de départ
3 interpolations linéaires

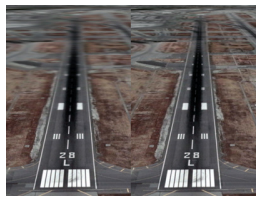


Trilinéaire :
8 points de départ
2 interpolations bilinéaires
1 interpolation linéaire



- Filtrage anisotropique
  - Adapte la texture aux déformations de l'objet (perspective)

Filtrage bilinéaire (floue)



Filtrage anisotropique

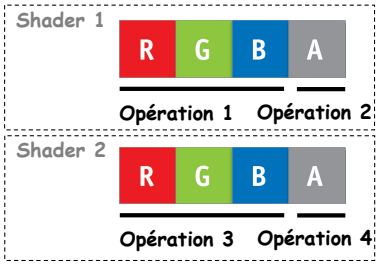
Pixel shader programmable

- Instructions spécifiques :
  - TexId :
    chargement d'une texture en U, V (bump mapping)
  - Texkill :
    annule le rendu pour un pixel
  - Support des branchements
  - Support pour le « Multiple Render Target »
    3 registres de sorties + 1 registre de profondeur

Exécution des instructions

- Co-issue :
  Exécution de 2 instructions différentes en parallèle sur les composantes RGBA

- Dual issue :
  Exécution de 2 instructions différentes sur 2 shaders différents



Caractéristiques dictées par DirectX 9

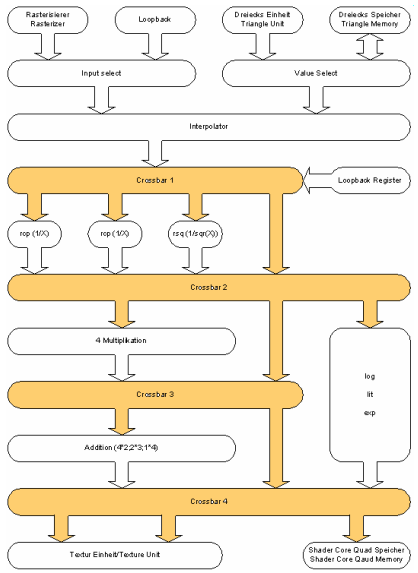
- Registres des pixel shaders V3.0

Nom	Nombre	R / W	# de port de lecture
Input register	10	R	1
Temporary register	32	R / W	3
Constant float register	224	R	1
Constant integer register	16	R	1
Constant boolean register	16	R	1
Predicate register	1	R	1
Sampler	16	R	1
Face register	1	R	1
Position register	1	R	1
Loop counter register	1	R	1
Output color register	1 ou 4	W	1
Output depth register	1	W	1

- Nombre d'instructions exécutées :
  - Statique : 65 536
  - Dynamique : 65 536

Swizzle

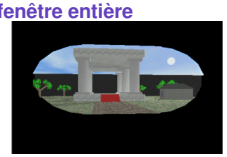
- Les données sont dans des registres de 4 flottants
- L'opérateur swizzle permet :
  - Le réarrangement des données
  - La duplication
  - La suppression
- R.xyzw = A.wxzy * B.wwww



http://www.3dcenter.org/artikel/cinefx/index2_e.php

7. Tests alpha

- **Tests alpha :**
  - Alpha : transparence d'un pixel
  - Supprime l'affichage des pixels dont la valeur alpha est inférieure à un seuil
- **Test de profondeur**
  - Utilise un tampon de profondeur
  - Supprime des pixels en fonction de leurs profondeurs
- **Stencil test**
  - Test la valeur qui se trouve dans le stencil buffer
  - Utile pour les ombres simples et le reflet dans un miroir
  - Dessine la scène dans une forme quelconque et non la fenêtre entière

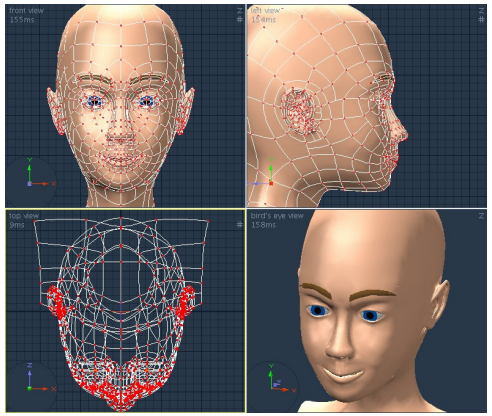


8. Le brouillard

- **Brouillard volumétrique**
  - Mélange la couleur du pixel avec la couleur du brouillard en fonction de sa distance par rapport à la caméra
- **Mélange alpha**
  - Mélange les couleurs de deux pixels en fonction de la valeur alpha
  - Celui arrivant et celui déjà présent

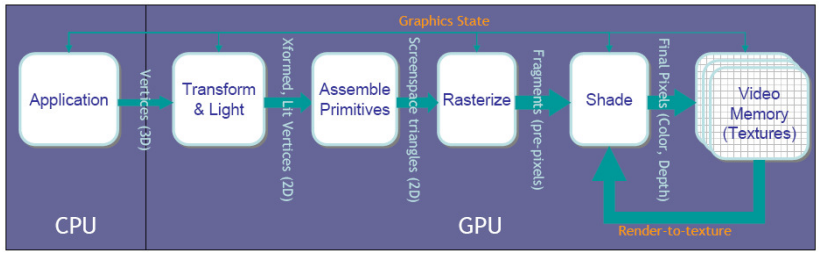
Viewport

- **Définit la zone géométrique d'affichage des pixels dans la fenêtre**
  Exemple d'utilisation :



Les goulots d'étranglement

- **Architecture sous la forme d'un pipeline**
- **Si une unité travaille plus qu'une autre**
  => problème
- **Goulots d'étranglement possibles:**



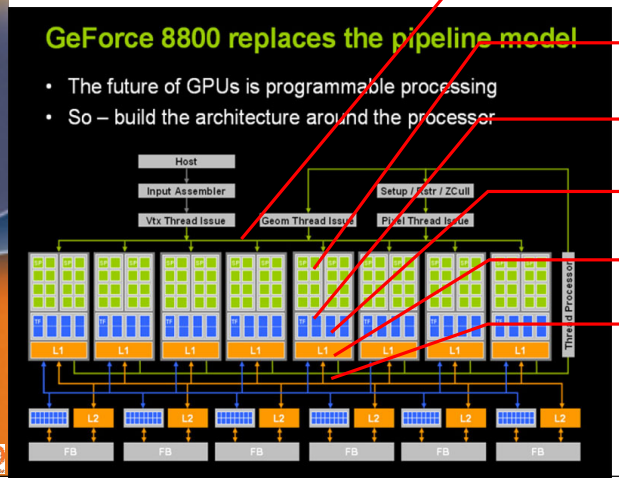
Impacts architecturaux de DirectX 10

- Limité à Windows Vista
- Supporté uniquement par les R600 et GeForce 8
- Unification des ressources de calcul
- Tout ce que je viens de vous dire sur le pipeline graphique est à revoir !!!

Resources	DirectX 9	DirectX 10
Temporary Registers	32	4,096
Constant Registers	256	16 x 4,096
Textures	16	128
Render Targets	4	8
Maximum Texture Size	4,048 x 4,048	8,096 x 8,096

Architecture unifiée imposée par DirectX 10

- Unification des ressources de calcul !
  - Pixel, vertex, géométrie, physique
- Moins de goulot d'étranglement



8 processeurs

16 processeurs de flot (vert)

8 unités de filtrage de texture (bleu)

4 unités d'adressage de texture

Cache L1

6 bus 64bits d'interface mémoire avec cache L2 dédié

GeForce 8 et CUDA

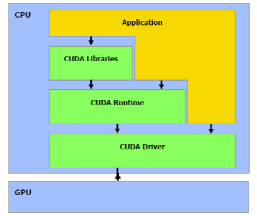
- Informations issues de CUDA sur la 8800 :
  - Cœur *double-pumped* (1350 Mhz)
  - 16 multiprocesseurs à 675 Mhz
    - traitent des groupes de 32 threads sur 8 processeurs généraux + 2 spécialisés
    - Disposent d'une mémoire partagée de 16ko (communication inter-thread)
  - Tous les 2 cycles, 1 inst. Courantes sur 512 threads
    - 256 opérations / cycles (ou 512 si MAD)
    - + 64 inst. spéciales

Gflops	FMUL	FADD	50% FMUL / 50% FADD	FMAD
GeForce 8800 GTX	172.8	172.8	172.8	345.6
GeForce 8800 GTS	115.2	115.2	115.2	230.4
Core 2 Extreme X6800	23.5	23.5	46.9	23.5
Core 2 Extreme Q6700	42.7	42.7	85.3	42.7

Puissance mesurée d'une 8800GTX

Environnement de programmation CUDA

- Environnement et architecture conçus ensemble => ne fonctionne qu'avec les GeForce 8
- Programmation par threads
- Extension du langage C
  __global__
  __device__
  __shared__
- Mémoire en lecture / écriture
  (plus de distinction entre vertex : a[i]=p et pixel p=a[i])
- Support des pointeurs



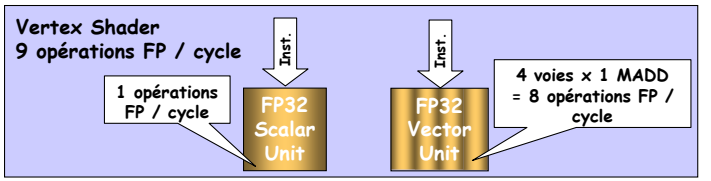
- Environnement CTM (*Close to The Metal*) avec ATI

PARTIE 3 :

Utilisation du GPU

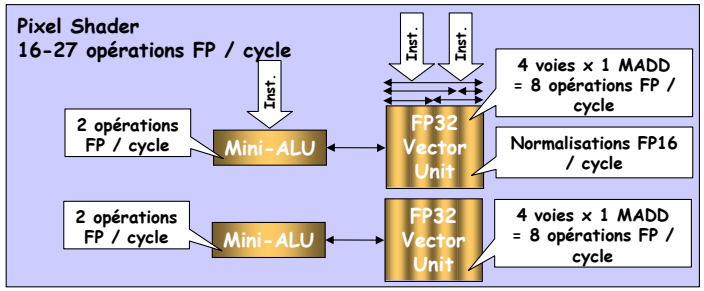
Partie 3 : Utilisation du GPU

Évaluation des Gflops de la 7800GTX



X 8 shaders = 72 opérations FP / cycle
X 550 Mhz = 39 GFLOPS

+



X 24 shaders = 648 opérations FP / cycle
X 550 Mhz = 356 GFLOPS
TOTAL = 400 GFLOPS

Les GPU en quelques chiffres : ATI R600

AMD

- **Modèle : X2000XTX**
  - 80nm, 500 10^6 transistors, 750 Mhz, 180 W, 420 mm^2 !!!
  - Mémoire : 1 Go de DDR4, 2400 MHz
  - Bande passante max : 153 Go/s
  - Vertex + Pixel Shader : 64
  - Unité de texture : 32
  - Pipeline de sortie : 16
  - Puissance de calcul attendue : 512 GFLOPS
- **Support pour DirectX 10.1 et OpenGL 2.0**

Source : http://en.wikipedia.org/wiki/Comparison_of_ATI_Graphics_Processing_Units

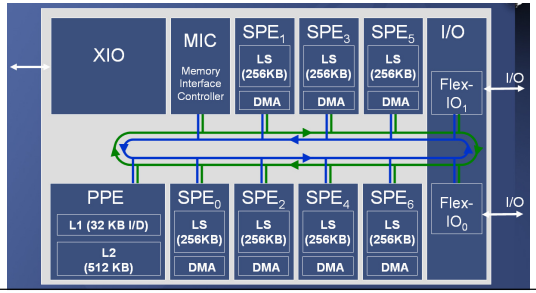
Les GPU en quelques chiffres : NVidia GeForce 8

- Modèle : 8800GTX
  - 90nm, 681 10^6 transistors, 575MHz, 484 mm² !!!
  - Mémoire : 768 Mo de GDDR3, 1800 MHz
  - Bande passante max : 86.4 Go/s
  - Vertex + Pixel Shader : 128
  - Unité de texture : 32
  - Pipeline de sortie : 24
  - Puissance de calcul attendue : 520 GFLOPS
- Support pour DirectX 10.0 et OpenGL 2.1

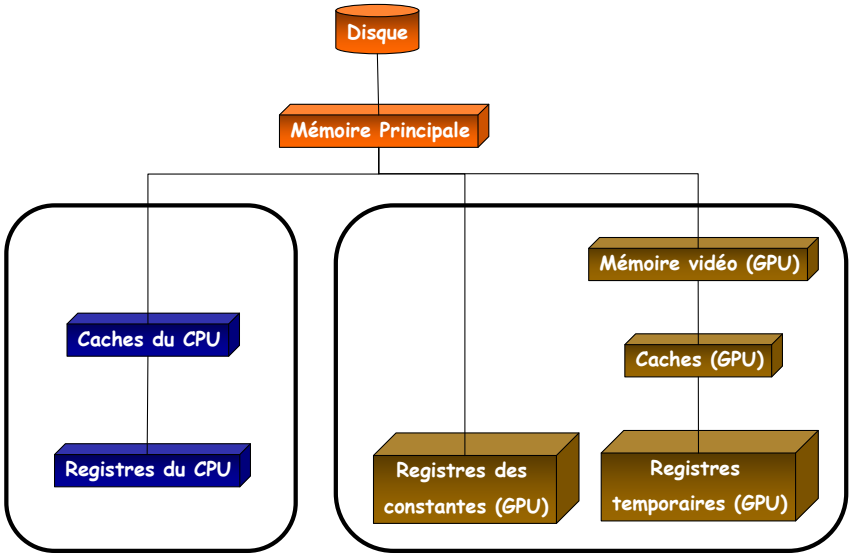
Source : http://en.wikipedia.org/wiki/Comparison_of_NVIDIA_Graphics_Processing_Units

Les GPU en quelques chiffres

- Et pour comparer… le Cell
  - 90nm, 234 10^6 transistors, 3.2GHz, 235 mm² !!!
  - Mémoire : 512Ko de cache L2
  - Bande passante max : (optimisée pour le débit et non la latence)
    - *204.8 Go/s (bus interne)*
    - *25.6 Go/s (mémoire externe)*
  - 8 *Synergistic Processing Elements* 128bits SIMD
  - 1 *Power Processor Element*
  - Puissance de calcul attendue : 205 GFLOPS



Hiérarchie mémoire au sein du PC + GPU



Modèle mémoire du CPU

- Mémoire de type accès aléatoire
- Accessible à partir de n'importe quel endroit du programme
  - Lecture/Écriture vers les registres
  - Lecture/Écriture vers la mémoire locale (pile)
  - Lecture/Écriture vers la mémoire globale (tas)
  - Lecture/Écriture vers le disque

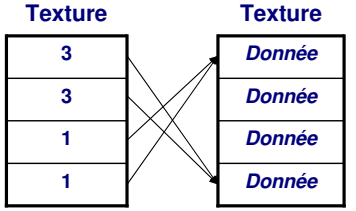
Modèle mémoire du GPU

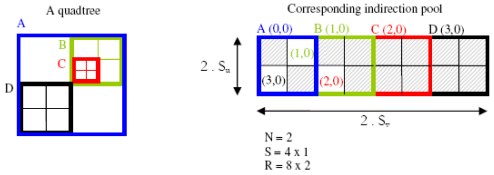
- Accès mémoire plus restreint
- Programme GPU
  - Allocation/Libération de la mémoire uniquement avant le calcul
  - Pas de pile locale
  - Pas d'accès disque direct
  - Lecture/Écriture vers les registres temporaires
  - Mémoire globale en lecture seulement
    - V = a[i];
  - Écriture vers la mémoire globale à la fin du traitement (Shader v3.0)
    - Position d'écriture fixée par la position de l'élément traité
      Pas d'écriture vers une adresse mémoire calculée
    - Les vertex shader peuvent écrire dans un flux de sommet
      Écriture de 12 composants de 4 flottants
    - Les pixel shader peuvent écrire dans le frame buffer
      Écriture de 4 composants de 4 flottants

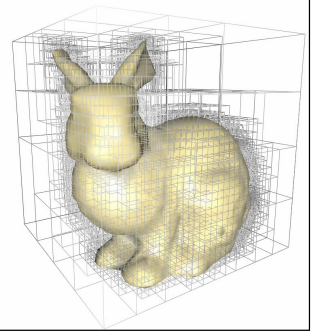
Question : *a[i] = v;*
Comment implémenter de façon logicielle une écriture « aléatoire » ?

Les pointeurs

- Utilisation de textures dépendantes (texture indirections)
- Créer des dépendances potentiellement non-recouvrables par du calcul non-dépendant
- Exemple: les octree
  - GPUGems 2, Chapitre 37

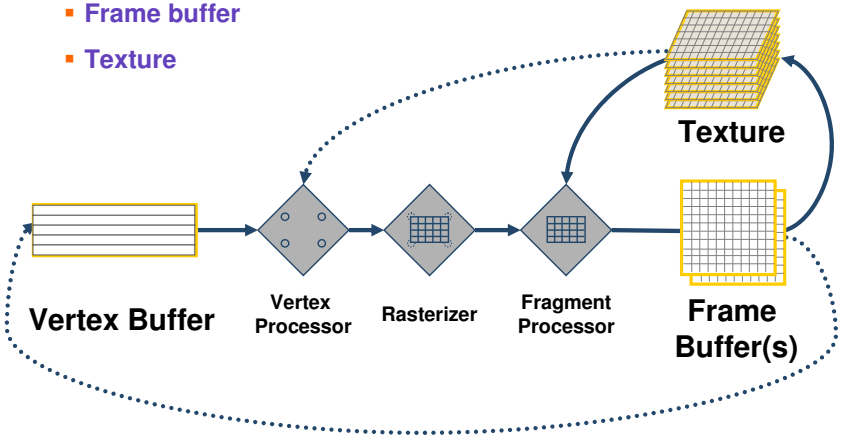






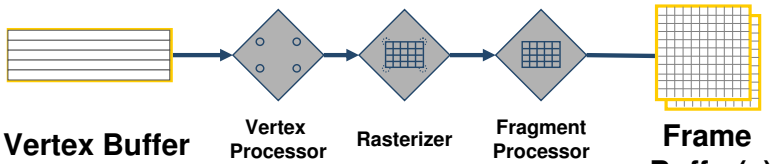
Stockage des données

- Les données se trouvent :
  - Vertex buffer
  - Frame buffer
  - Texture



Utilisation du Framebuffer

- Mémoire écrite par le fragment processor
- Zone mémoire en écriture seule
- Affichage du résultat
- Utile pour mémoriser le résultat de calcul GPGPU

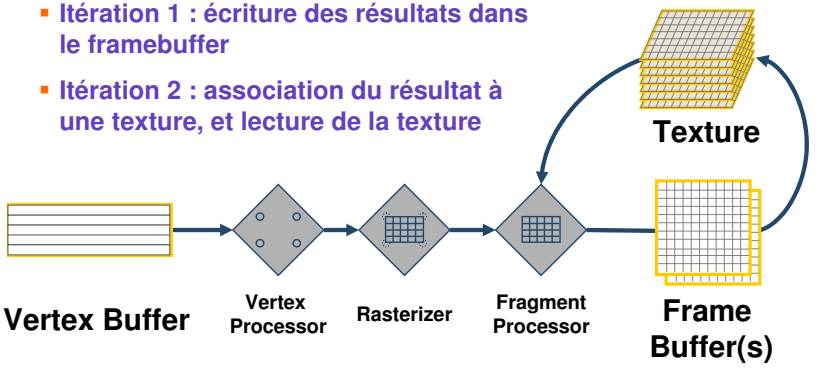


Utilisation de la mémoire de texture

- Idée :
  - Écrire le résultat dans la mémoire de texture
  - Permet les itérations (+ Ping-Pong)
  - Support OpenGL
    - Sauvegarde sur des flottants 16/32 bits / pixel
    - Utilisation des Multiple Render Targets (MRTs) ATI et Nvidia
    - 1. Copie vers la texture - glCopyTexSubImage
    - 2. Rendu direct vers la texture - GL_EXT_framebuffer_object

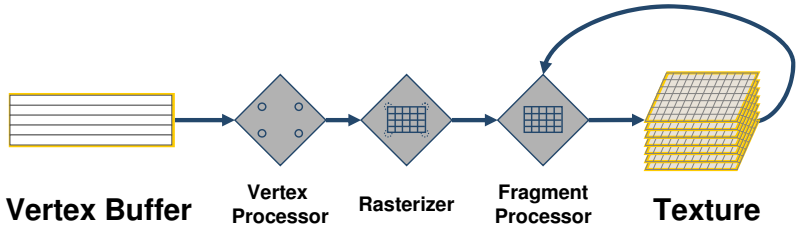
Utilisation de la mémoire de texture

- Le plus utilisé en GPGPU
- Création d'algorithme en plusieurs passes
  - Itération 1 : écriture des résultats dans le framebuffer
  - Itération 2 : association du résultat à une texture, et lecture de la texture



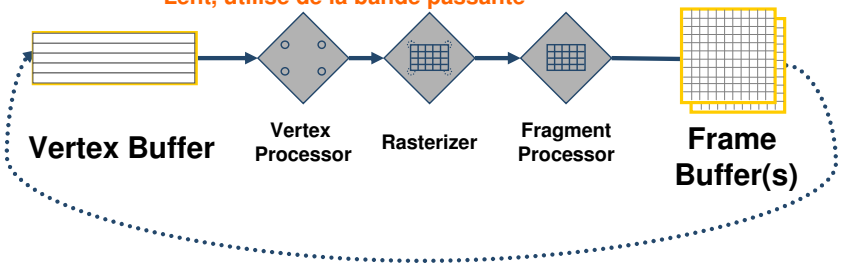
Utilisation de la mémoire de texture

- Render-To-Texture (RTT) basé sur les pbuffers
  - Rapide, mais ne fonctionne que sous Windows
- Support multi-plateforme
  - Frame Buffer Object (FBO)



Utilisation de la zone d'un vecteur de sommet

- Idée :
  - Écrire le résultat dans un vecteur de sommet
  - Permet aux données de reboucler sur le pipeline graphique
  - Support OpenGL
    - Sauvegarde sur des flottants 16/32 bits / pixel
    - Utilisation des Multiple Render Targets (MRTs)
    - Copie vers un vecteur de sommet - GL_ARB_pixel_buffer_object
    - Lent, utilise de la bande passante

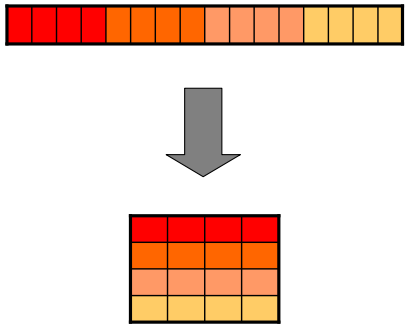


Résumé sur la mémoire

- **Contraintes sur la programmation en GPU**
  - Pas de lecture / écriture simultanée sur une même texture.
  - Écriture uniquement dans le pixel courant.
  - On peut combiner la valeur d'un pixel avec la valeur du pixel antérieur.
  - Il est possible d'accéder n'importe où dans une texture mais pour un coût élevé.
  - Le nombre de textures adressables simultanément est limité.
  - La taille des textures est limitée (généralement 4096 pixels).

Structure de base

- **Tableaux :**
  - Limité à 2048 ou 4096 éléments
  - Reformulé en tableau 2D
- **Matrices :**
  - Limité à 2048x2048 ou 4096x4096 éléments
  - Structure naturelle
  - Pas de traduction d'adresse



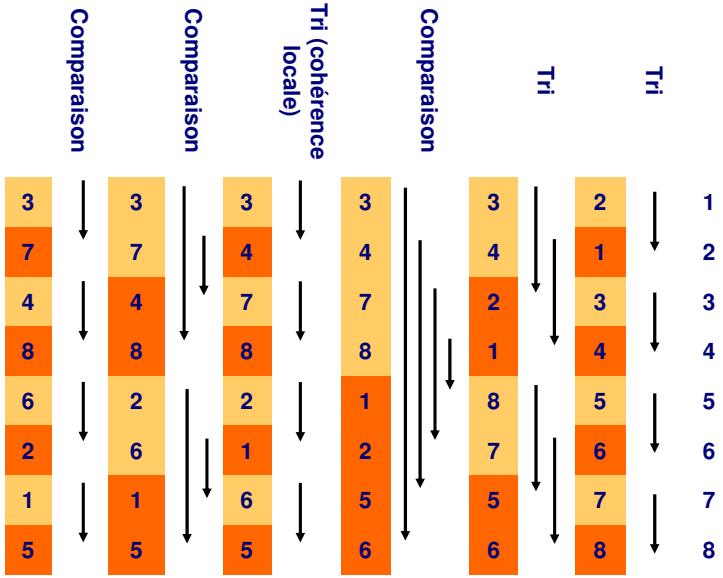
Structures creuses

- **Intérêts :**
  - Réduction du temps de calcul
  - Réduction de la bande passante
- **Solutions:**
  1. Stockage complet des données
     - Rejet des données en utilisant le culling
  2. Stockage des données intéressantes
     - Table de page
     - Table de hashage

Algorithmique

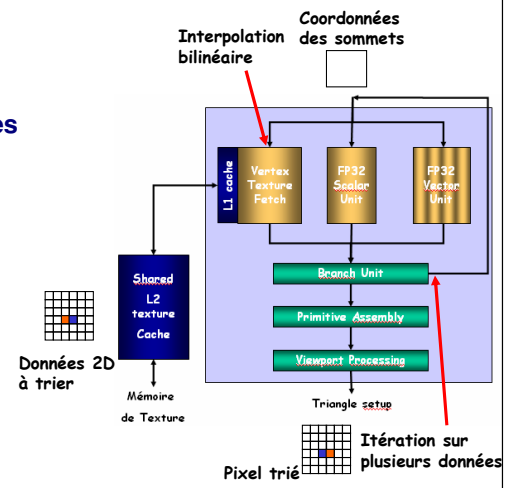
- **Le tri bitonique**
- **Tabulation de données**
- **Cluster de GPU appliqué à la bioinfo**

Algorithmique sur GPU : Le tri bitonique



Algorithmique sur GPU : Le tri bitonique

- Tri en $O(\log^2 n)$ étapes
- Chaque étape effectue n comparaisons / échanges
- Coût total : $O(n \log^2 n)$
- Implémentation sur GPU :



- GPUTeraSort: High Performance Graphics Coprocessor Sorting for Large Database Management, N. K. Govindaraju et al.

Tabulation de données

- Objectif :
  - Recalculer une données est parfois coûteux
  - Utilisation de table (ex: évaluation des fonctions élémentaires)
- Solution :
  - Mémoriser les données dans une texture

+

???

+ … Interpolation linéaire quasi gratuite

- Une texture flottante peut mémoriser 4 canaux (= 4 fonctions: sin, cos, exp, log)
- Utiliser le filtrage disponible dans l'unité de texture
  - Fournit des valeurs entre les valeurs tabulées
- Filtrage linaire : $y = a \cdot x_1 + (1-a) \cdot x_2$
- Filtrage bilinéaire, Filtrage trilinéaire, Filtrage anisotropique + mipmapping
- Inconvénients :
  - Gourmand en bande passante

Cluster de GPU en Bioinformatique

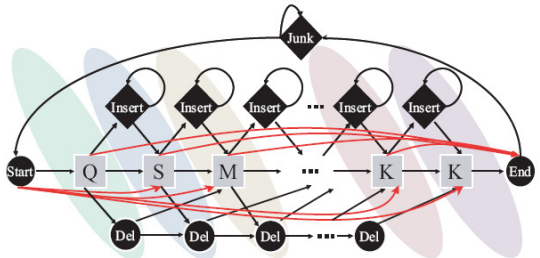
- **HMMer**
  - **Base de données de protéines**
    - **SWISS-PROT (200 000 protéines)**
    - **NCBI (> 2.5 millions de protéines)**
  - **Séquences d'ADN**
    - **DDBJ Japan (42 millions de gènes)**
    - **GenBank (10 millions de séquences)**
  - **Problème :**
    - **Plusieurs protéines codent la même fonction**
    - **Comparaison d'une nouvelle protéine avec celles de la base de données**
    - **Recherche de similarités en utilisant un modèle probabiliste**

HMMer sur GPU : ClawHMMer

- « ClawHMMER: A Streaming HMMer-Search Implementation », D. R. Horn, Supercomputing 2005
  - x3 par rapport à PowerPC G5
  - x25 par rapport à Pentium 4
  - Dérivé de la version HMMer pour CPU
  - Basé sur l'algorithme de Viterbi
    - Recherche du chemin le plus probable
    - Implémentation en Brook (langage de haut niveau)

Implémentation de ClawHMMer (1)

- **Pré-tri de la base de données par longueur des séquences**
  - **Pour utiliser la mémoire GPU disponible (256-512 Mo)**
  - **Pour que la quantité de calcul sur ces données soit identiques**
- **Hidden Markov Model**



- **Pour une séquence de longueur L :**
  - **L multiplications matrice-vecteur**

Implémentation de ClawHMMer (1)

- **Implémentation :**
  - **Utilisation d'une texture de 4 flottants**
    3 pour la sauvegarde des probabilités, 1 pour la max
  - **Modèle précédent déroulé et exécuté dans le pixel shader**
  - **Les probabilités de transitions sont stockées dans les registres constants**
  - **Exécution vectorielle en testant plusieurs protéines en parallèle**
- **Limites :**
  - **Nombre de registres intermédiaires limité**
  - **Nombre de vecteurs en sortie limité**
- **Version exécutée sur machine SPIRE**
  (16 nœuds graphiques)
  - **Découpage manuel des jeux de données**
  - **Utilisation >95% des 16 processeurs**

Conclusion

- De nouveaux co-processeurs sont disponibles
  - Puissance de calcul et bande passante
- Environnement de programmation complexe
  - Mais les choses s'améliorent (DirectX10, CUDA, CTM, ...)
- Paradigme de programmation intéressant
  - Recherche très active sur des représentations de données efficaces
  - Les algorithmes doivent s'adapter à l'architecture (idem pour le cell)

Pointeurs

GPGPU

http://www.gpgpu.org
http://www.nvidia.com
http://www.ati.com

GPUGems 2,
Programming Techniques for high performance graphics and General-Purpose Computation
Matt Pharr
2005
Addison-Wesley
ISBN : 0-32133-559-7