

---

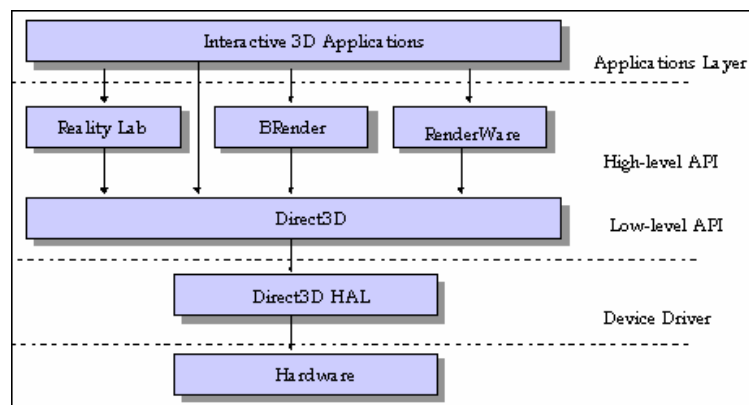
# Introduction aux processeurs graphiques (GPU) et Cg

Daniel Etiemble  
de@lri.fr

---

## Les applications graphiques

De l'application graphique au matériel : exemple Direct3D



# Le pipeline graphique

Géométrie  
Calcul flottant  
CPU ou GPU

Rendu  
Accès mémoire  
GPU

Initialisation

Eclairage

Projection et correction  
de perspective

Tessellation

Transformation

Culling

Eclairage du  
sommet

Clipping

Texture

Filtrage

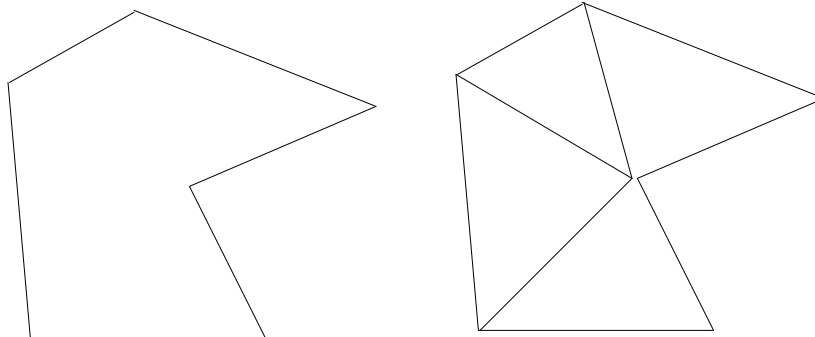
Brume

Anti-Aliasing

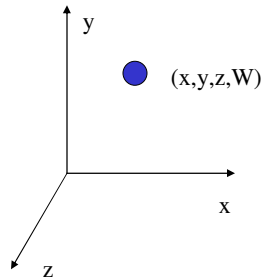
Z-buffer

# Tessellation

Découpage des objets en triangles



## Coordonnées géométriques



$(x/W, y/W, z/W, 1)$   
représentation homogène

$W=0$  : point à l'infini

## Transformations géométriques

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translation

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation z

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Changement  
d'échelle

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation x

$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation y

## Transformations géométriques

---

- Produits de translation, rotation et changement d'échelle

- Transformations comme un changement de coordonnées.

$$P^{(i)} = M_{i \leftarrow j} P^{(j)}$$

$$P^{(j)} = M_{j \leftarrow k} P^{(k)}$$

$$M_{i \leftarrow k} = M_{i \leftarrow j} \cdot M_{j \leftarrow k}$$

- Chaque objet est représenté dans son propre système de coordonnées

## Les matrices dans Direct3D

---

$$M_{\text{proj}} = \begin{pmatrix} \frac{2 * Z_n}{S_w} & 0 & 0 & 0 \\ 0 & \frac{2 * Z_n}{S_h} & 0 & 0 \\ 0 & 0 & \frac{Z_f}{Z_f - Z_n} & 1 \\ 0 & 0 & \frac{-Z_f * Z_n}{Z_f - Z_n} & 0 \end{pmatrix}$$

$$M_{\text{clip}} = \begin{pmatrix} \frac{2}{C_w} & 0 & 0 & 0 \\ 0 & \frac{2}{C_h} & 0 & 0 \\ 0 & 0 & \frac{1}{Z_{\text{max}} - Z_{\text{min}}} & 0 \\ -1 - 2 * \frac{C_x}{C_w} & 1 - 2 * \frac{C_y}{C_h} & \frac{-Z_{\text{min}}}{Z_{\text{max}} - Z_{\text{min}}} & 1 \end{pmatrix}$$

$$M_{\text{projortho}} = \begin{pmatrix} \frac{2}{S_w} & 0 & 0 & 0 \\ 0 & \frac{2}{S_h} & 0 & 0 \\ 0 & 0 & \frac{1}{Z_f - Z_n} & 0 \\ 0 & 0 & \frac{-Z_n}{Z_f - Z_n} & 1 \end{pmatrix}$$

$$M_{\text{vs}} = \begin{pmatrix} \text{dwWidth} & 0 & 0 & 0 \\ 0 & -\text{dwHeight} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \text{dwX} & \text{dwHeight} + \text{dwY} & 0 & 1 \end{pmatrix}$$

## Suite du pipeline géométrique

---

- Culling
  - Elimination des faces cachées
- Calcul de l'éclairage du sommet
  - Modèle de Phong : luminosité = produit scalaire entre la normale en ce point et la direction de la lumière
- Projection des coordonnées sur l'écran
- Clipping
  - Elimination des sommets qui sortent de l'écran

## Eclairage

---



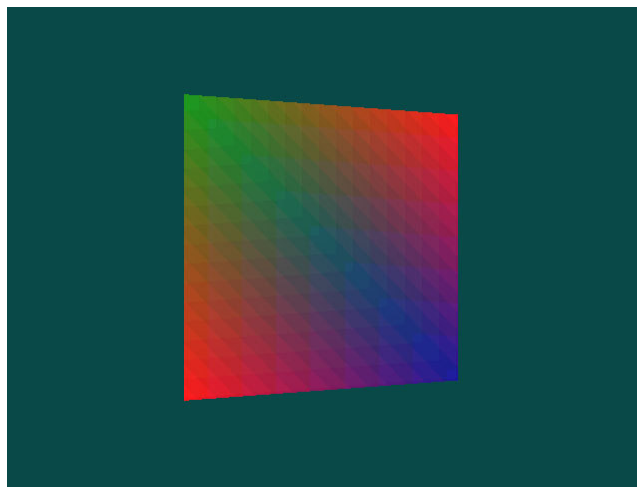
## Le rendu

---

- Réalisé par le processeur graphique (carte 3D)
- Etapes
  - Ombrage
  - Application des textures
  - Tampon Z
  - Brouillard
  - Alpha
  - Anti-aliasing

## Ombrage

---



## Ombrage

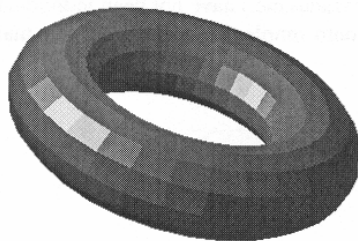
---

- Technique permettant de donner aux surfaces une teinte dépendant des sources d'éclairage. Les algorithmes les plus connus sont ceux de Phong et de Gouraud
  - Flat
    - technique la plus primitive d'ombrage, d'habillage de chaque triangle d'une seule couleur. Produit un effet de "bloc" car sans mélange de couleurs.
  - Gouraud
    - Technique d'ombrage faisant l'habillage des polygones en utilisant des effets de lumières et d'ombre. C'est une technique élaborée qui augmente la qualité du rendu polygone en y mélangeant les couleurs.

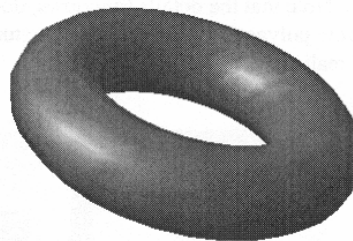
## Flat et Gouraud

---

Flat Shading

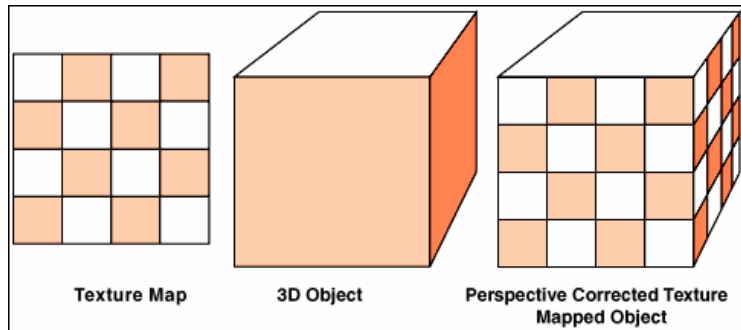


Gouraud Shading



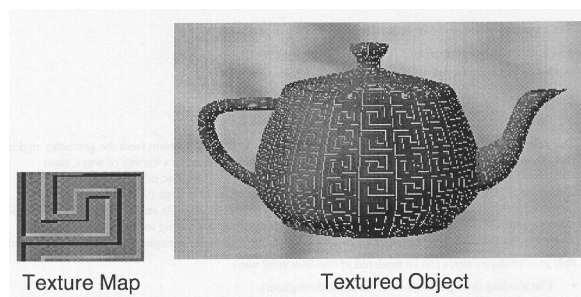
## Applications des textures

---



## Exemple de texture

---



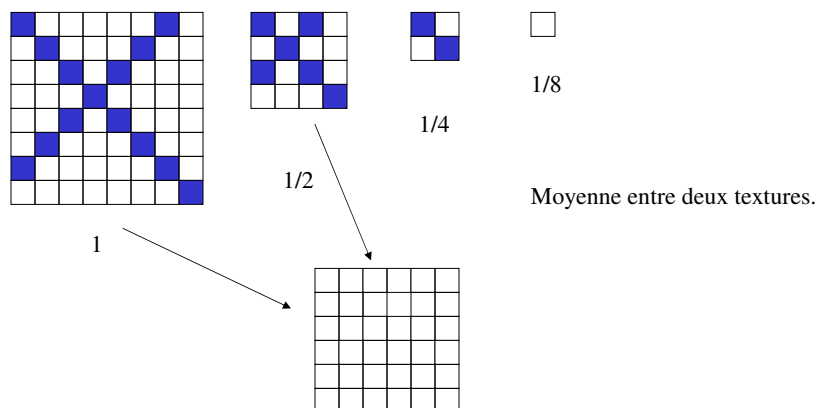
## Problèmes avec les textures

---

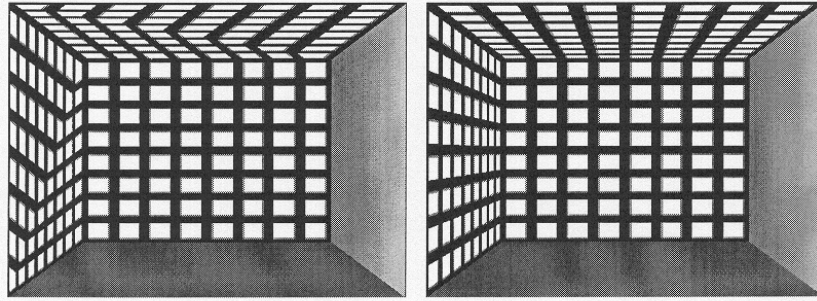
- Effet d'escalier dû au changement de résolution
- Techniques
  - filtrage bilinéaire
  - MIP-mapping
  - MIP-mapping trinéaire

## MIP-mapping

---



## Correction de perspective

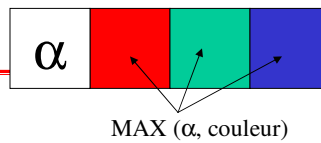


Master d'Informatique  
2003

Programmes graphiques et multimédia sur PC  
D. Etiemble

19

## Alpha Blending

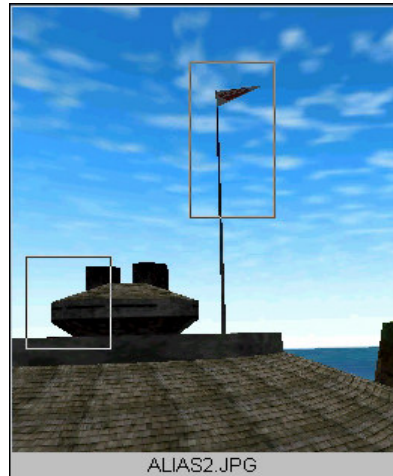
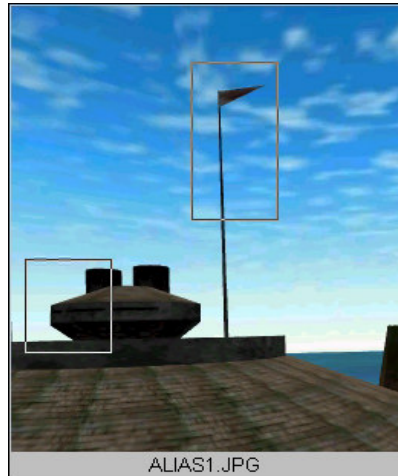


Master d'Informatique  
2003

Programmes graphiques et multimédia sur PC  
D. Etiemble

20

## Anti-aliasing

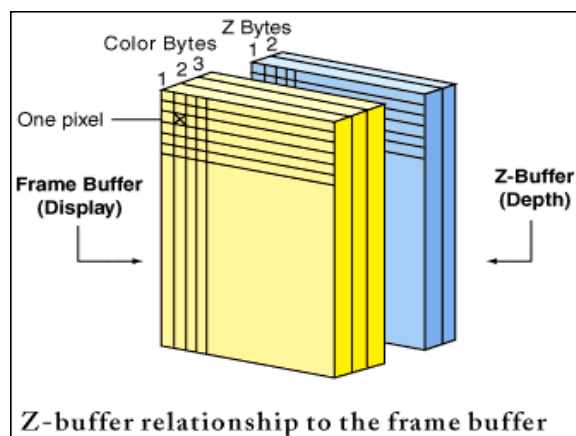


Master d'Informatique  
2003

Programmes graphiques et multimédia sur PC  
D. Etiemble

21

## Tampons Z



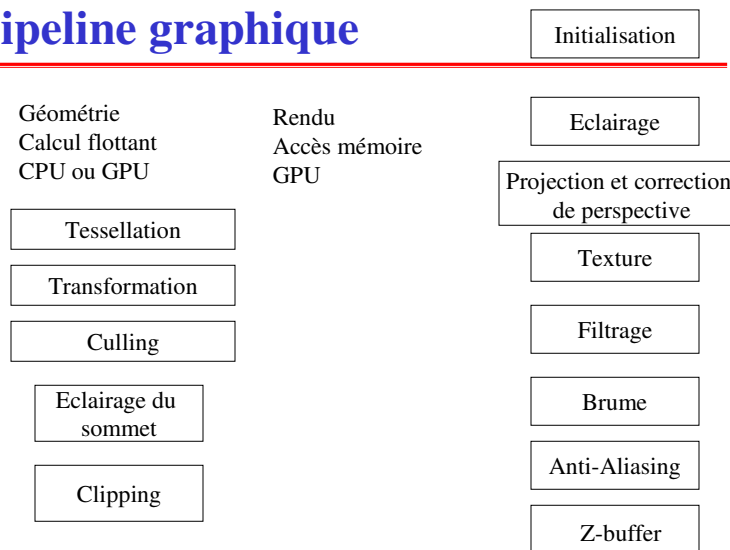
- Affichage de l'objet le plus proche de l'utilisateur (pas d'affichage des objets cachés)

Master d'Informatique  
2003

Programmes graphiques et multimédia sur PC  
D. Etiemble

22

# Le pipeline graphique



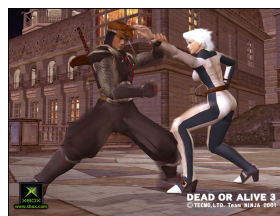
# Evolution des performances des cartes graphiques



**Virtua Fighter**  
(SEGA Corporation)

**NV1**  
**50K triangles/sec**  
**1M pixel ops/sec**

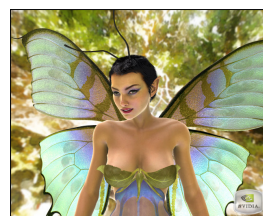
**1995**



**Dead or Alive 3**  
(Tecmo Corporation)

**Xbox (NV2A)**  
**100M triangles/sec**  
**1G pixel ops/sec**

**2001**



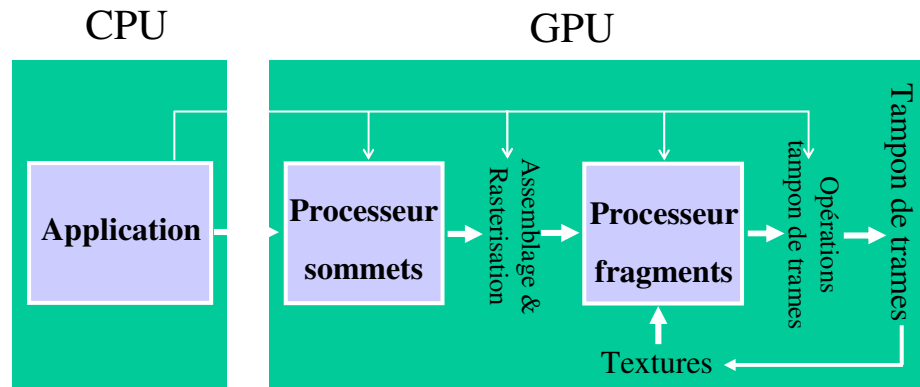
**Dawn**  
(NVIDIA Corporation)

**GeForce FX (NV30)**  
**200M triangles/sec**  
**2G pixel ops/sec**

**2003**

# Modèle de programmation des GPU

NVIDIA - 2002



Master d'Informatique  
2003

Programmes graphiques et multimédia sur PC  
D. Etiemble

25

## Types de données

- Flottants 32 bits à travers le pipeline
  - Tampon de trames
  - Textures
  - Processeur fragments
  - Processeur de sommet
  - Interpolations
- Le processeur fragment dispose aussi de
  - Flottants 16 bits "half"
  - Virgule fixe 12 bits
- Les textures et le tampon de trames disposent aussi de
  - Grande gamme de formats fixes
  - Par exemple, le format classique 8 bits par pixel
  - Ces formats utilisent moins de bande passante que les flottants 32 bits

Master d'Informatique  
2003

Programmes graphiques et multimédia sur PC  
D. Etiemble

26

## Fonctionnalités du processeur de sommets

---

- SIMD flottant vectoriel (4 éléments)
- Contrôle de flots dépendant des données
  - Instructions de branchement conditionnelles
  - Appel de fonctions, avec jusqu'à quatre appels imbriqués
  - Table de sauts (pour les instructions multi-voies)
- Codes conditions
- Nouvelles instructions arithmétiques (COS)
- 256 instructions par programme (beaucoup plus sans branchement)
- 16 registres vectoriels à 4 éléments temporaires
- 256 registres de paramètres
- 2 registres d'adresses (vecteurs de 4 éléments)
- 6 sorties de distance de clipping.

## Le processeur de fragments (1)

---

- Jeu d'instructions
  - Instructions générales et orthogonales
  - Même syntaxe que le processeur de sommets  
MUL R0, R1.xyz, R2.yxw;
  - Ensemble complet d'instructions arithmétiques
    - RCP, RSQ, COS, EXP, ...
- Instructions sur les textures
  - Les lectures de texture correspondent à une instruction (TEX, TXP, or TXD)
  - Permettent de calculer les coordonnées sur les textures, avec un niveau quelconque d'imbrication
  - Permettent plusieurs utilisations d'une unité de texture.
- Autres possibilités
  - Accès en lecture à une position dans la fenêtre
  - Accès en lecture écriture au Z fragment
  - Instructions de dérivation intégrées
    - Dérivées partielles par rapport au coordonnées x ou y de l'écran
    - Utile pour l'anti-aliasing
  - Instruction conditionnelle de suppression d'un fragment

## Le processeur fragments (2)

- Possibilités
  - 1024 instructions
  - 512 paramètres uniformes ou constantes
    - Chaque constante compte comme une instruction
  - 16 unités de textures
    - Réutilisables à volonté
  - Entrées : 4 x 8 flottants 32 bits
  - Sortie “couleur” tampon de trames de 128 bits
    - 4 flottants 32 bits, 8 half 16 bits, etc.
- Limitations
  - Pas de branchement, mais possibilité d’utiliser les codes condition
  - Pas de lectures indexées à partir des registres
    - Utilisation de lecture de textures
  - Pas d’écriture mémoire

## De l’assembleur à Cg

### Assembleur

```
...  
FRC R2.y, C11.w;  
ADD R3.x, C11.w, -R2.y;  
MOV H4.y, R2.y;  
ADD H4.x, -H4.y, C4.w;  
MUL R3.xy, R3.xyww, C11.xyww;  
ADD R3.xy, R3.xyww, C11.z;  
TEX H5, R3, TEX2, 2D;  
ADD R3.x, R3.x, C11.x;  
TEX H6, R3, TEX2, 2D;  
...
```

### Cg

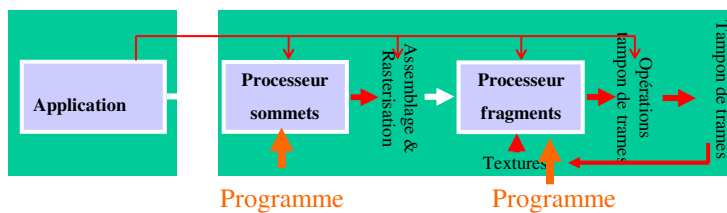
```
...  
L2weight = timeval - floor(timeval);  
L1weight = 1.0 - L2weight;  
ocoord1 = floor(timeval)/64.0 + 1.0/128.0;  
ocoord2 = ocoord1 + 1.0/64.0;  
L1offset = f2tex2D(tex2, float2(ocoord1, 1.0/128.0));  
L2offset = f2tex2D(tex2, float2(ocoord2, 1.0/128.0));  
...
```

## Stratégie de conception de Cg

- Partir de C (et un peu de C++)
  - Minimise le nombre de décisions
  - Permet de prendre en compte les erreurs “connues” au lieu des “inconnues”
- Permettre de réduire le langage
- Ajouter les caractéristiques voulues pour les processeurs graphiques
  - Pour prendre en compte le modèle de programmation des GPU
  - Pour la haute performance
- Faire une synthèse des différents aspects

## Différences CPU et GPU

1. Le processeur graphique est un processeur de flots
  - Plusieurs unités de traitement programmables
  - Connectées par les flots de données.



- Deux types de programmes distincts
  - Programme sommets et programme fragments
- Deux types d'entrées
  - Entrées variables (flot de données)
  - Entrées uniformes (état graphique)

## Différences CPU et GPU

---

2. Plus de variantes dans les possibilités de base
  - La plupart des processeurs n'ont toujours pas de branchements
  - Les processeurs de sommets n'ont pas de support pour le mappage des textures
  - Certains processeurs ont des types de données supplémentaires

- **Le compilateur ne peut cacher les différences**
- **Le plus petit commun dénominateur est trop restrictif**
- **Profils de langage différents**  
(liste des possibilités et types de données)
- **Convergence des profils espérée dans le futur**

## Différence GPU et CPU

---

3. Optimisés pour l'arithmétique sur des vecteurs de 4 éléments
  - Utile pour le graphique –couleurs, vecteurs, coordonnées
  - Bonne manière d'obtenir la performance/coût

- **La philosophie C expose les types de données au matériel**
- **Cg a des types de données et des opérations vectorielles : float2, float3, float4**
- **Rend évident la manière d'obtenir une performance élevée**
- **Cg a aussi des types de données matriciels : float3x3, float3x4, float4x4**

## Quelques opérations vectorielles

```
//  
// Clamp components of 3-vector to [minval,maxval] range  
//  
float3 clamp(float3 a, float minval, float maxval) {  
    a = (a < minval.xxx) ? Minval.xxx : a;  
    a = (a > maxval.xxx) ? Maxval.xxx : a;  
    return a;  
}
```

Comparaisons  
vectorielles élément par  
élément et résultat  
vectoriel.

? : par élément  
des vecteurs

Duplication et/ou réarrangement  
des éléments

## Opérations vectorielles

- Swizzle – dupliquer/réarranger les éléments

```
a = b.xxyy;
```

- Masque d'écriture – écriture partielle

```
a.w = 1.0;
```

- Construction de vecteurs

```
- a = float4(1.0, 0.0, 0.0, 1.0);
```

## Cg a des tableaux

---

- Déclarés comme en C
- Les tableaux sont distincts des types vectoriels  
`float4 != float[4]`
- Les profils peuvent restreindre l'utilisation des tableaux

```
vout MyVertexProgram(float3 lightcolor[10],  
                    ...) {  
    ...  
}
```

## Différences CPU et GPU

---

4. Pas de pointeurs
  - Les tableaux sont les types de données privilégiés de Cg
5. Pas de type de données entiers
  - Cg ajoute le type "bool" pour les opérations entières

### LES TYPES DE DONNÉES CG

- Tous profils:
  - float
  - bool
- Tous profils avec recherche de texture :
  - sampler1D, sampler2D, sampler3D, samplerCUBE
- Profil du programme fragment NV:
  - half -- flottant demi-précision
  - fixed -- virgule fixe [-2,2)

## Fonctions intégrées Cg

---

- Mappage de textures (dans les profils fragment)
- Math
  - Produit scalaire
  - Multiplication de matrices
  - Sin/cos/etc.
  - Normalisation
- Diverses
  - Dérivées partielles (quand elles sont supportées)

## Exemple Cg (1)

---

```
// In:
//   eye_space position = TEX7
//   eye space T = (TEX4.x, TEX5.x, TEX6.x)   denormalized
//   eye space B = (TEX4.y, TEX5.y, TEX6.y)   denormalized
//   eye space N = (TEX4.z, TEX5.z, TEX6.z)   denormalized

fragout frag program main(vf30 In) {

    float m = 30;                               // power
    float3 hiCol  = float3( 1.0, 0.1, 0.1 ); // lit color
    float3 lowCol = float3( 0.3, 0.0, 0.0 ); // dark color
    float3 specCol = float3( 1.0, 1.0, 1.0 ); // specular color

    // Get eye-space eye vector.
    float3 e = normalize( -In.TEX7.xyz );

    // Get eye-space normal vector.
    float3 n = normalize( float3(In.TEX4.z, In.TEX5.z, In.TEX6.z ) );
```

## Exemple Cg (2)

---

```
float edgeMask = (dot(e, n) > 0.4) ? 1 : 0;
float3 lpos = float3(3,3,3);
float3 l = normalize(lpos - In.TEX7.xyz);
float3 h = normalize(l + e);

float specMask = (pow(dot(h, n), m) > 0.5) ? 1 : 0;

float hiMask = (dot(l, n) > 0.4) ? 1 : 0;
float3 ocoll = edgeMask *
               (lerp(lowCol, hiCol, hiMask) + (specMask * specCol));

fragout O;
O.COL = float4(ocoll.x, ocoll.y, ocoll.z, 1);
return O;
}
```