Shaders Programmables Temps Réel

Etat des lieux sur leur programmation

Mathias Paulin

Equipe Synthèse d'Images et Réalité Virtuelle

IRIT-UPS-CNRS UMR 5505

Université Paul Sabatier

Toulouse

Shaders Programmables Temps Réel

- Historique
  - Définition.
  - Evolution.
  - Approches temps réel.

Shaders Programmables Temps Réel

- Historique
- Cg : un environnement de développement ?
  - Le langage Cg
  - Les profils de compilation
  - Intégration dans les applications

Shaders Programmables Temps Réel

- Historique
- Cg : un environnement de développement ?
- Shaders OpenGL : exemples d'utilisation
  - Environnement d'expérimentation VRML
  - OpenGL vertex program
  - OpenGL fragment program

Shaders Programmables Temps Réel

- Historique
- Cg : un environnement de développement ?
- Shaders OpenGL : exemples d'utilisation
- Limitations et problèmes
  - Shaders non portables
  - Compilateur naïf
  - Quelles solutions ?

Shaders Programmables Temps Réel

- Historique
- Cg : un environnement de développement ?
- Shaders OpenGL : exemples d'utilisation
- Limitations et problèmes
- Démonstrations

Définition

Shaders

- Procédure de calcul d’ombrage
- Par abus de langage, calcul de l’apparence

Définition

Shaders

- Procédure de calcul d'ombrage
- Par abus de langage, calcul de l'apparence

- Différents types de shaders :
  - Objet (basse fréquence)
    - Forme
    - Transformations
    - Eclairage

Définition

Shaders

- Procédure de calcul d'ombrage
- Par abus de langage, calcul de l'apparence

- Différents types de shaders :
  - Objet (basse fréquence)
  - Sommet (moyenne fréquence)
    - Couleur
    - Coordonnées de texture
    - Par extension, position, normale . . .

Définition

Shaders

- Procédure de calcul d'ombrage
- Par abus de langage, calcul de l'apparence

- Différents types de shaders :
  - Objet (basse fréquence)
  - Sommet (moyenne fréquence)
  - Fragment (haute fréquence)
    - Couleur et texture
    - Profondeur

Définition

Shaders

- Procédure de calcul d’ombrage
- Par abus de langage, calcul de l’apparence

- Différents types de shaders :
  - Objet (basse fréquence)
  - Sommet (moyenne fréquence)
  - Fragment (haute fréquence)

- Modèle de calcul SIMD

Evolution

- Shaders non temps réel
  - *Shade trees* de Cook en 1984
  - Fondés sur le lancer de rayons
  - Utilisés dans REYES et RenderMan

Evolution

- Shaders non temps réel
  - *Shade trees* de Cook en 1984
  - Fondés sur le lancer de rayons
  - Utilisés dans REYES et RenderMan
- Shaders temps réel
  - Shaders paramétrables
    - Pipeline graphique classique
    - Fonctionnalités fixées mais paramétrables
    - OpenGL $\leq$ 1.3

Evolution

- Shaders non temps réel
  - *Shade trees* de Cook en 1984
  - Fondés sur le lancer de rayons
  - Utilisés dans REYES et RenderMan
- Shaders temps réel
  - Shaders paramétrables
  - Shaders à base de textures
    - Limités aux shaders d'apparence
    - Approche multi-passes
    - SGI Interactive Shading Language

Evolution

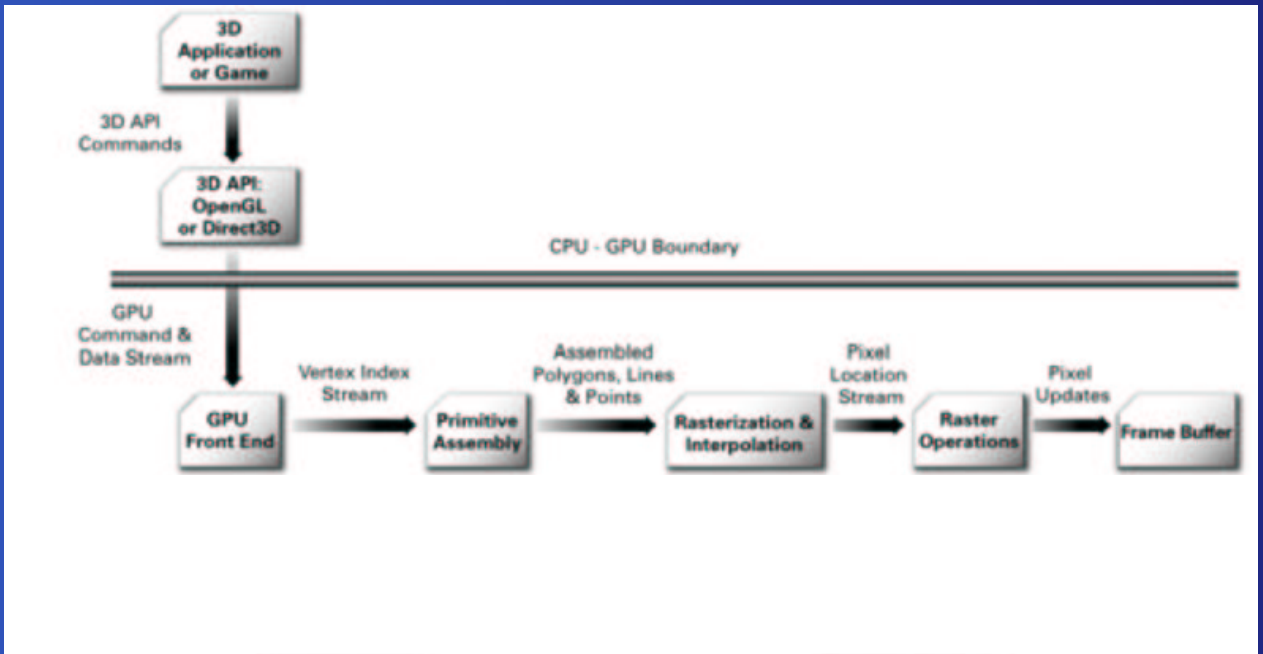
- Shaders non temps réel
  - *Shade trees* de Cook en 1984
  - Fondés sur le lancer de rayons
  - Utilisés dans REYES et RenderMan
- Shaders temps réel
  - Shaders paramétrables
  - Shaders à base de textures
  - Shaders programmables
    - Pipeline graphique logiciel et matériel
    - RTSL, Cg, OpenGL 2.0

Evolution

- Shaders non temps réel
  - *Shade trees* de Cook en 1984
  - Fondés sur le lancer de rayons
  - Utilisés dans REYES et RenderMan
- Shaders temps réel
  - Shaders paramétrables
  - Shaders à base de textures
  - Shaders programmables
- Nécessité d'un environnement de programmation.

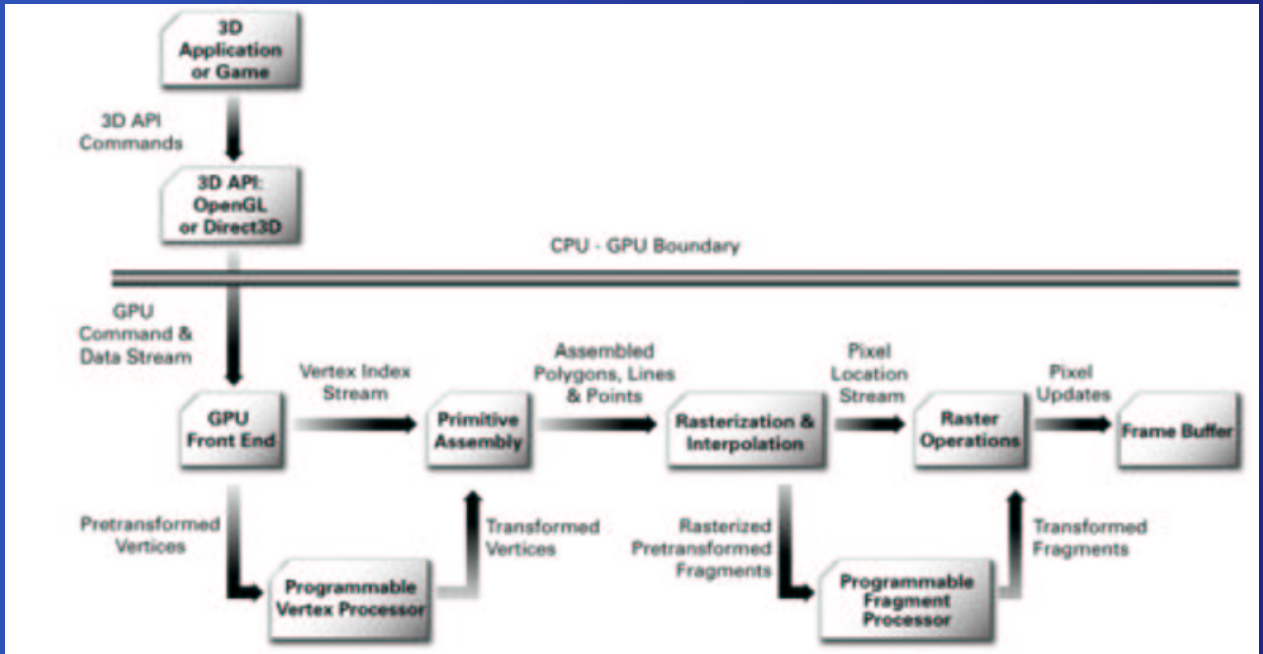
Shaders Programmables Temps Réel

- Pipe graphique en évolution



Shaders Programmables Temps Réel

- Pipe graphique en évolution
- Ajout d'étages programmables
  - Traitement matériel des shaders



Shaders Programmables Temps Réel

- Pipe graphique en évolution
- Ajout d'étages programmables
  - Traitement matériel des shaders
- Programmation en *assembleur* des shaders
  - Extensions ARB
    - Vertex Program
    - Fragment Program
  - Extensions propriétaires
    - NVidia
    - ATI

Shaders Programmables Temps Réel

- Shader Phong
  - Calcul de l'éclairage par pixel
  - Evaluation du modèle de Phong

Shaders Programmables Temps Réel

- Shader Phong
  - Calcul de l'éclairage par pixel
  - Evaluation du modèle de Phong

- Assembleur
  - Ecriture complexe
  - Réutilisation difficile

```
...
RSQR  R0.x, R0.x;
MULR  R0.xyz, R0.xxxx, R4.xyzz;
MOVR  R5.xyz, -R0.xyzz;
MOVR  R3.xyz, -R3.xyzz;
DP3R  R3.x, R0.xyzz, R3.xyzz;
SLTR  R4.x, R3.x, {0.000000}.x;
ADDR  R3.x, {1.000000}.x, -R4.x;
MULR  R3.xyz, R3.xxxx, R5.xyzz;
MULR  R0.xyz, R0.xyzz, R4.xxxx;
ADDR  R0.xyz, R0.xyzz, R3.xyzz;
DP3R  R1.x, R0.xyzz, R1.xyzz;
MAXR  R1.x, {0.000000}.x, R1.x;
LG2R  R1.x, R1.x;
MULR  R1.x, {10.000000}.x, R1.x;
EX2R  R1.x, R1.x;
MOVR  R1.xyz, R1.xxxx;
MULR  R1.xyz, {0.900000, 0.800000,
      1.000000}.xyzz, R1.xyzz;
DP3R  R0.x, R0.xyzz, R2.xyzz;
MAXR  R0.x, {0.000000}.x, R0.x;
MOVR  R0.xyz, R0.xxxx;
ADDR  R0.xyz, {0.100000, 0.100000,
      0.100000}.xyzz, R0.xyzz;
MULR  R0.xyz, {1.000000, 0.800000,
      0.800000}.xyzz, R0.xyzz;
ADDR  R1.xyz, R0.xyzz, R1.xyzz;


...
```

Shaders Programmables Temps Réel

- Shader Phong
  - Calcul de l'éclairage par pixel
  - Evaluation du modèle de Phong

- Assembleur
  - Ecriture complexe
  - Réutilisation difficile

```
...
RSQR  R0.x, R0.x;
MULR  R0.xyz, R0.xxxx, R4.xyzz;
MOVR  R5.xyz, -R0.xyzz;
MOVR  R3.xyz, -R3.xyzz;
DP3R  R3.x, R0.xyzz, R3.xyzz;
SLTR  R4.x, R3.x, {0.000000}.x;
ADDR  R3.x, {1.000000}.x, -R4.x;
MULR  R3.xyz, R3.xxxx, R5.xyzz;
MULR  R0.xyz, R0.xyzz, R4.xxxx;
ADDR  R0.xyz, R0.xyzz, R3.xyzz;
DP3R  R1.x, R0.xyzz, R1.xyzz;
MAXR  R1.x, {0.000000}.x, R1.x;
LG2R  R1.x, R1.x;
MULR  R1.x, {10.000000}.x, R1.x;
EX2R  R1.x, R1.x;
...
```

- Cg
  - Ecriture simplifiée
  - Réutilisation par fonctions

```
...
float3 cSpec = pow(max(0, dot(Nf, H)), phongExp).xxx;
float3 cPlastic = Cd * (cAmbi + cDiff) + Cs * cSpec;
...
```

```
ADDR  R0.xyz, {0.100000, 0.100000,
      0.100000}.xyzz, R0.xyzz;
MULR  R0.xyz, {1.000000, 0.800000,
      0.800000}.xyzz, R0.xyzz;
ADDR  R1.xyz, R0.xyzz, R1.xyzz;

...
```

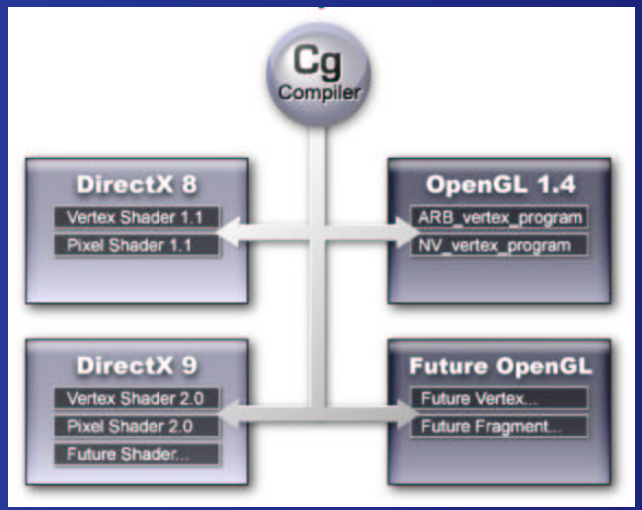
Cg : généralités

- Objectifs
  - Langage de programmation évolué
  - Multi plate-forme : DirectX et OpenGL

Cg : généralités

- Objectifs
  - Langage de programmation évolué
  - Multi plate-forme : DirectX et OpenGL

- Fonctionnement
  - Langage et compilation
    - Profils de compilation
  - API pour l'intégration
    - API générale
    - API spécifique (OpenGL ou DirectX)



Cg : le langage

Langage dérivé de C et C++

- Types de données
  - Vecteurs de réels
  - Tableaux de vecteurs

Cg : le langage

Langage dérivé de C et C++

- Types de données
  - Vecteurs de réels
  - Tableaux de vecteurs
- Opérateurs vectoriels surchargés

Cg : le langage

Langage dérivé de C et C++

- Types de données
  - Vecteurs de réels
  - Tableaux de vecteurs
- Opérateurs vectoriels surchargés
- Fonctions, séquence, sélection, boucles

Cg : le langage

Langage dérivé de C et C++

- Types de données
  - Vecteurs de réels
  - Tableaux de vecteurs
- Opérateurs vectoriels surchargés
- Fonctions, séquence, sélection, boucles
- Bibliothèque de base
  - Mathématique
  - Géométrique
  - Texture

Cg : profils de compilation

- Prise en charge des différents types de GPU
- Profils existants
  - OpenGL NVxx vertex et fragment program
    - NV30 $\Longleftrightarrow$ GeForceFX
    - NV20 $\Longleftrightarrow$ GeForce 3 et 4

Cg : profils de compilation

- Prise en charge des différents types de GPU
- Profils existants
  - OpenGL NVxx vertex et fragment program
  - DirectX vertex et fragment shaders
    - 9.0 : Cg $\Longleftrightarrow$ HLSL
    - 8.0

Cg : profils de compilation

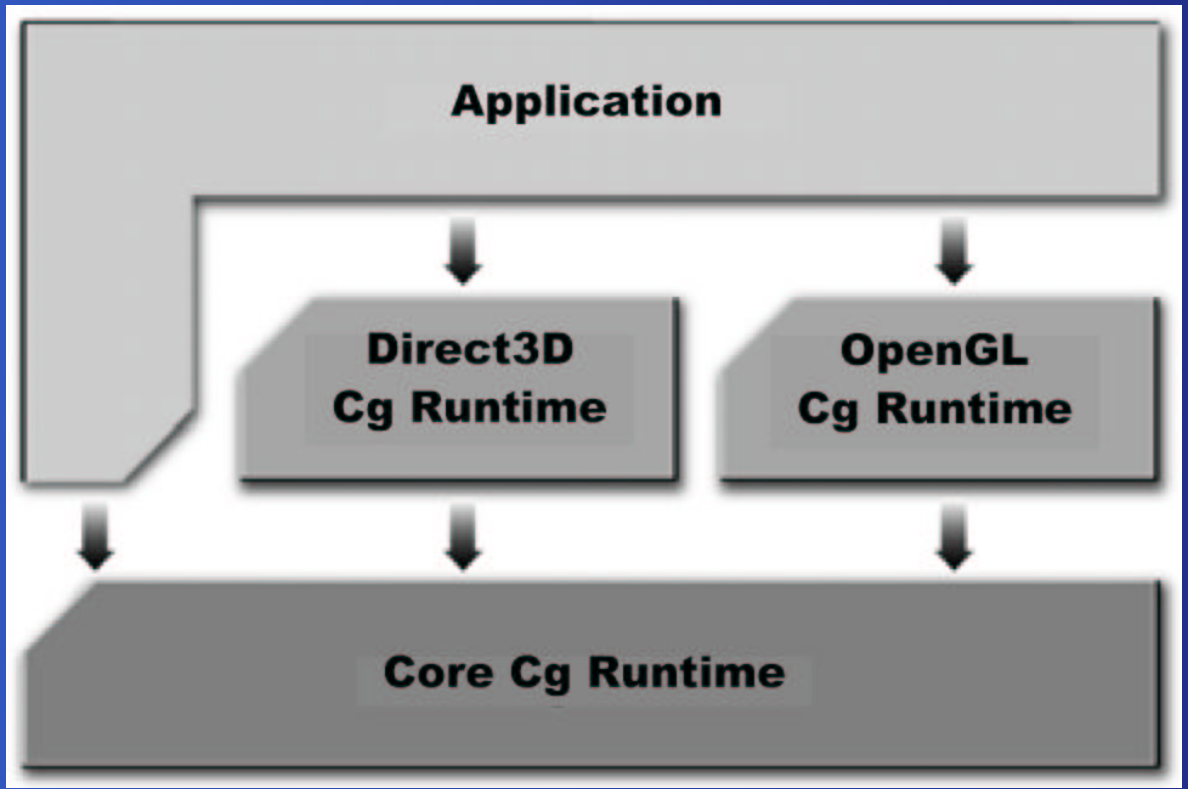
- Prise en charge des différents types de GPU
- Profils existants
  - OpenGL NVxx vertex et fragment program
  - DirectX vertex et fragment shaders
  - OpenGL ARB vertex et fragment program
    - Seul profil réellement portable

Cg : profils de compilation

- Prise en charge des différents types de GPU
- Profils existants
  - OpenGL NVxx vertex et fragment program
  - DirectX vertex et fragment shaders
  - OpenGL ARB vertex et fragment program
    - Seul profil réellement portable
- Un profil définit
  - Les tailles des types de données
  - Les instructions utilisables
  - Les variables prédéfinies

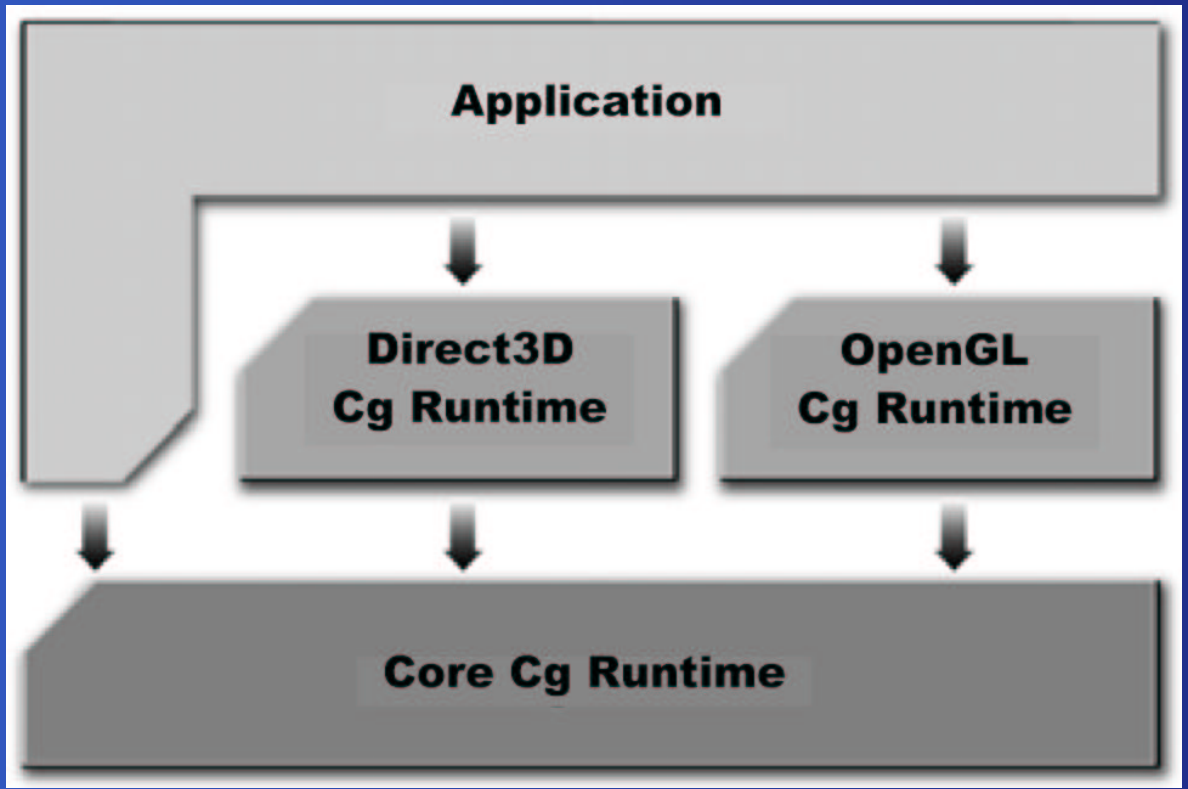
Cg : intégration dans les applications

- API générale d'exécution permettant de
  - Compiler à la volée
  - Gérer les paramètres des shaders



Cg : intégration dans les applications

- API spécifique d'exécution permettant de
  - Activer/désactiver les shaders
  - Faire le suivi des paramètres des shaders



Shaders et VRML

- VML : Vrml Modelling Library
  - Bibliothèque de gestion de mondes VRML
  - Navigateur et utilitaires associés
  - Développés à l'IRIT depuis 1999
  - Orientée vers l'expérimentation en rendu temps réel
  - http://www.irit.fr/∼Mathias.Paulin/Logiciels

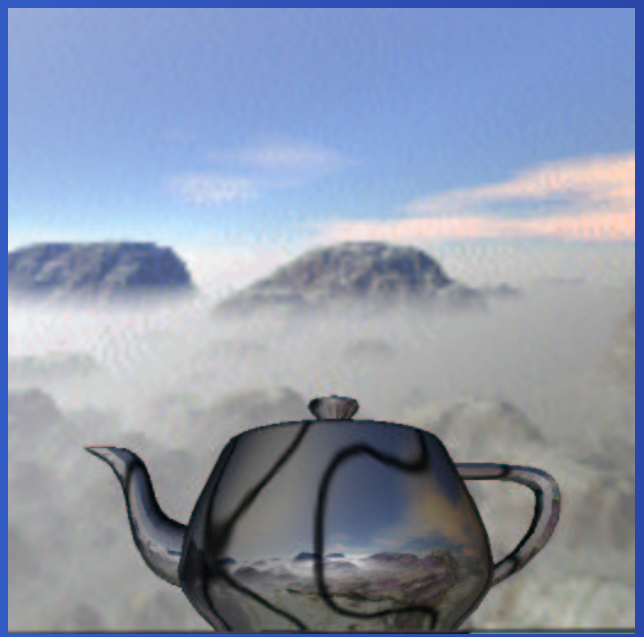
Shaders et VRML

- VML : Vrml Modelling Library
  - Bibliothèque de gestion de mondes VRML
  - Navigateur et utilitaires associés
  - Développés à l'IRIT depuis 1999
  - Orientée vers l'expérimentation en rendu temps réel

  - Mécanisme d'extension de la grammaire
    - Génération de code C++
    - Chargement dynamique des extensions

Shaders et VRML

- Intégration de shaders dans la VML
- Shaders à base de textures
  - Approche multi-textures
    - Noeud VRML TextureSet



Shaders et VRML

- Intégration de shaders dans la VML
- Shaders à base de textures
  - Approche multi-textures
  - Approche multi-passes
    - Noeud VRML CartoonShader

Shaders et VRML

- Intégration de shaders dans la VML
- Shaders à base de textures
  - Approche multi-textures
  - Approche multi-passes
- Uniquement par paramétrage OpenGL
  - Limitation aux shaders basse fréquence

Shaders et VRML

- Intégration de shaders dans la VML
- Shaders à base de textures
  - Approche multi-textures
  - Approche multi-passes
- Uniquement par paramétrage OpenGL
  - Limitation aux shaders basse fréquence
- Besoin de programmation
  - Complexité des Shaders
  - Shaders moyenne et haute fréquence

Shaders et VRML.

Intégration des Shaders en VRML

- définition d'un noeud Appearance Shaders
- Interface avec CG
  - Profils ARB
  - Compilation du fichier source
  - Gestion des paramètres
    - Multi textures
    - Paramètres numériques

Shaders et VRML.

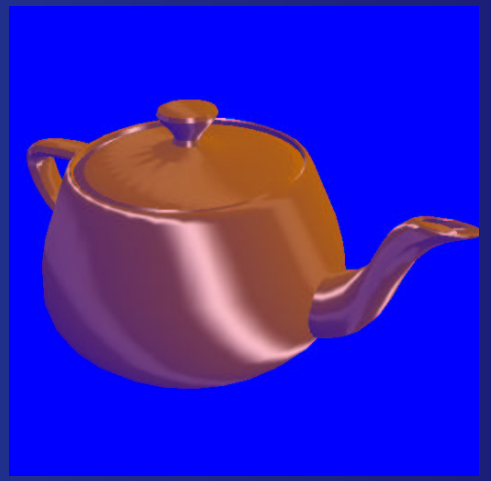
Intégration des Shaders en VRML

- définition d'un noeud Appearance Shaders
- Interface avec CG
  - Profils ARB
  - Compilation du fichier source
  - Gestion des paramètres
    - Multi textures
    - Paramètres numériques
- Shaders ARB
  - Même fonctionnement

ARB Vertex Shaders et Cg.

Exemple de shader : algorithme NPR de Gooch.

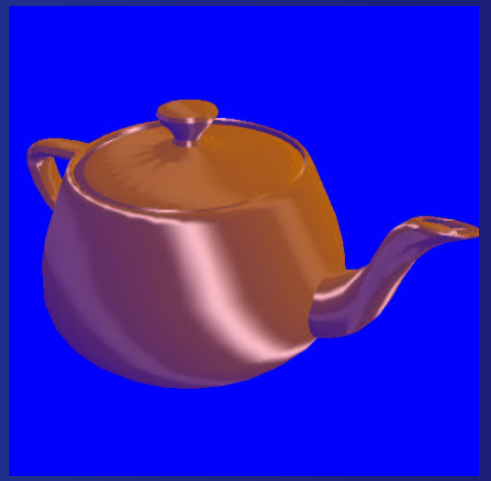
- Codage (simple)
  - 9 lignes de Cg
  - 20 lignes d’assembleur



ARB Vertex Shaders et Cg.

Exemple de shader : algorithme NPR de Gooch.

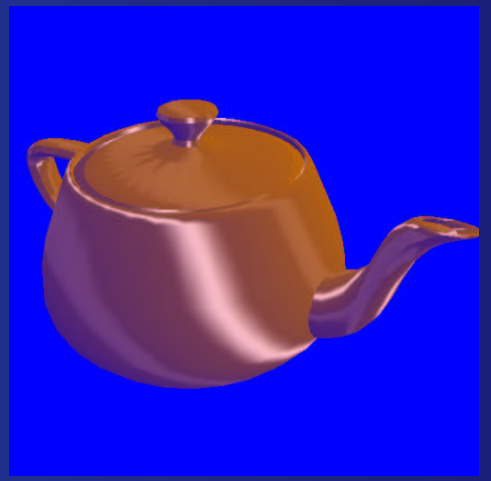
- Codage (simple)
  - 9 lignes de Cg
  - 20 lignes d'assembleur
- Compilation
  - 22 lignes produites par Cg
  - Nombre de registres identiques



ARB Vertex Shaders et Cg.

Exemple de shader : algorithme NPR de Gooch.

- Codage (simple)
  - 9 lignes de Cg
  - 20 lignes d'assembleur
- Compilation
  - 22 lignes produites par Cg
  - Nombre de registres identiques



Comportement vérifié sur tout type de shaders mais . . .

ARB Fragments Shaders et Cg.

Exemple de shader : Texture de Perlin.

- Codage (moyen)
  - 82 lignes de Cg (boucles et fonctions)
  - 290 lignes d'assembleur

ARB Fragments Shaders et Cg.

Exemple de shader : Texture de Perlin.

- Codage (moyen)
  - 82 lignes de Cg (boucles et fonctions)
  - 290 lignes d'assembleur
- Compilation
  - 340 lignes produites par Cg
  - Nombre de registres en hausse

ARB Fragments Shaders et Cg.

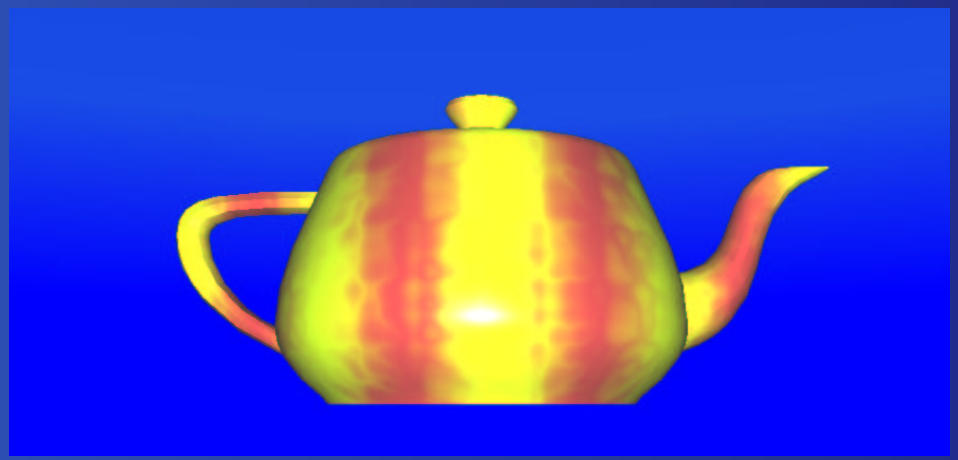
Exemple de shader : Texture de Perlin.

- Codage (moyen)
  - 82 lignes de Cg (boucles et fonctions)
  - 290 lignes d'assembleur
- Compilation
  - 340 lignes produites par Cg
  - Nombre de registres en hausse
- Utilisation
  - Limitée pour le moment
  - Emulation logicielle

ARB Fragments Shaders et Cg.

Texture de Perlin : Description

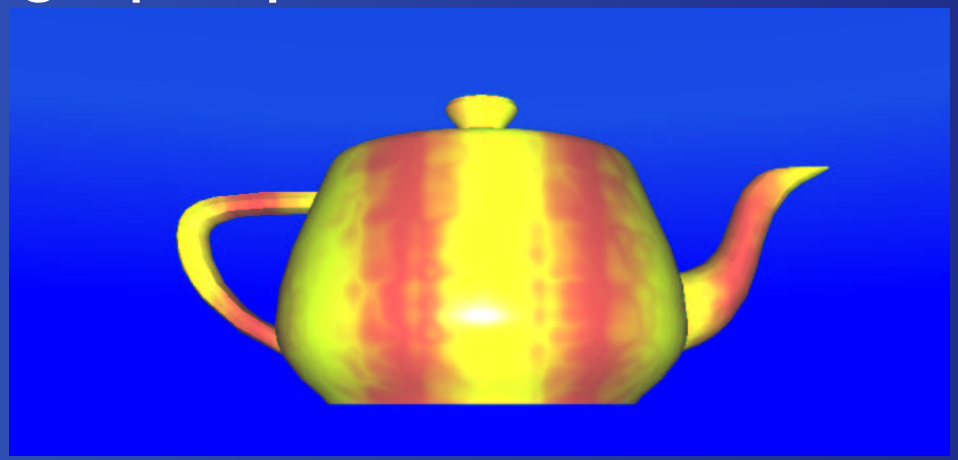
- Vertex shader
  - Transformations géométriques
  - calcul des coordonnées de textures



ARB Fragments Shaders et Cg.

Texture de Perlin : Description

- Vertex shader
  - Transformations géométriques
  - calcul des coordonnées de textures
- Fragment Shader
  - Somme 5 octaves de bruit (2 avec Cg)
  - Eclairage par pixel



Cg pour les vertex Shaders

- Assez bonne utilisabilité
- Simplicité de développement

- Code généré non optimal
  - Opérations inutiles ($*1$ ou $+0$)
  - Implantation bizarres
- Impossible d'accéder aux états OpenGL

Cg pour les fragment Shaders

- Utilisabilité limitée
- Simplicité de développement

- Code généré non optimal
  - Opérations inutiles ($*1$ ou $+0$)
  - Implantation bizarres
- Gestion fantaisiste des registres
- Gestion fantaisiste des boucles

Quelles solutions ?