gamagora

LES FORMATIONS PROFESSIONNELLES AUX MÉTIERS DU JEU VIDÉO

UNIVERSITÉ LUMIÈRE LYON 2

Shaders

UE Rendu - Parcours Dev

Jean-Philippe.Farrugia@LIRIS.CNRS.fr

LIRIS

UCBL

Sources

- Cours J.C Iehl M2 Images (Lyon 1).
- GLSL Programming Guide
- MSDN
- Sources web diverses

A quoi ça sert ?

- A faire mieux que les fonctions standard !
  - Matériaux réalistes.
  - Ajouts de détails géométriques.
  - Eclairement évolué.
  - Phénomènes naturels (fumée, feu..).
  - Rendu expressif.
  - Post processing (motion blur...)
  - ...

LIRIS

Un shader, c'est quoi ?

- Un programme exécuté par le GPU.
  - Vertex shader : transformer la géométrie.
  - Geometry shader : ajouter de la géométrie.
  - Pixel shader : modifier l'image générée.
- Langage proche du C++ :
  - HLSL pour Direct X.
  - GLSL pour OpenGL.
  - CG pour l'un ou l'autre...
    - Très proche de HLSL..

LIRIS

Un shader, c'est quoi ?

- Les shaders remplacent les opérations standard dans le pipeline graphique.
  - **Ils ont des obligations !**
- Un shader n'est qu'une étape de calcul :
  - Il traite des entrées (résultats de l'étape précédente).
  - Il **doit** produire certains résultat, sous un certain format.
- Mais peut faire d'autre chose en plus...

LIRIS

Plan

- GPU - Pipeline graphique - rappels.
- Shaders
  - Vertex Shaders
  - Pixel Shaders
- API existantes
  - GLSL
  - Cg - HLSL
- HLSL : utilisation

Pipeline théorique

- Fonctionnalité de base d'un GPU :
  - Calcule une image à partir d'un flot de données 3D.
- Principales étapes :
  - Transformation objet - monde - vue
  - Calcul éclairement
  - Assemblage des primitives
  - Rasterization (= Discretisation)
  - Calcul couleur des pixels.

Unités programmables

- Transformations modelview et projection :
  - Vertex Processor.
- Couleur des fragments :
  - Fragment Processor.
- Assemblage des primitives :
  - Geometry processor.
- Programmes associés :
  - Vertex/Fragment/Geometry Shaders

Comparaisons

- Programme CPU :
  - Traite l'ensemble de ses données en une fois.
  - L'ensemble des données est accessible n'importe quand, dans n'importe quel ordre.
  - Le format de sortie est libre.
- Programme GPU :
  - Transforme un flux de données en un autre, tous deux de formats imposés.

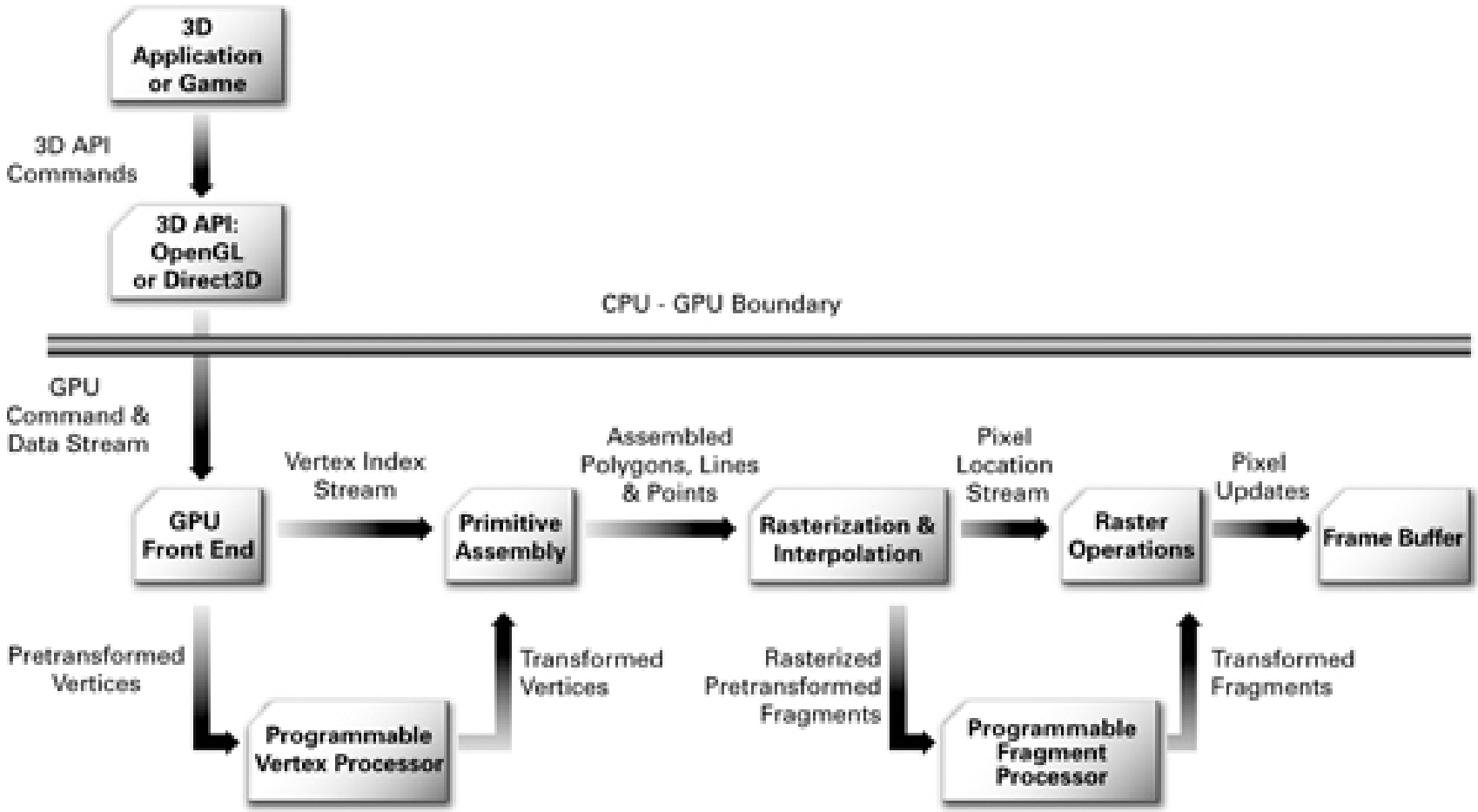
LIRIS

Comment ça marche ?

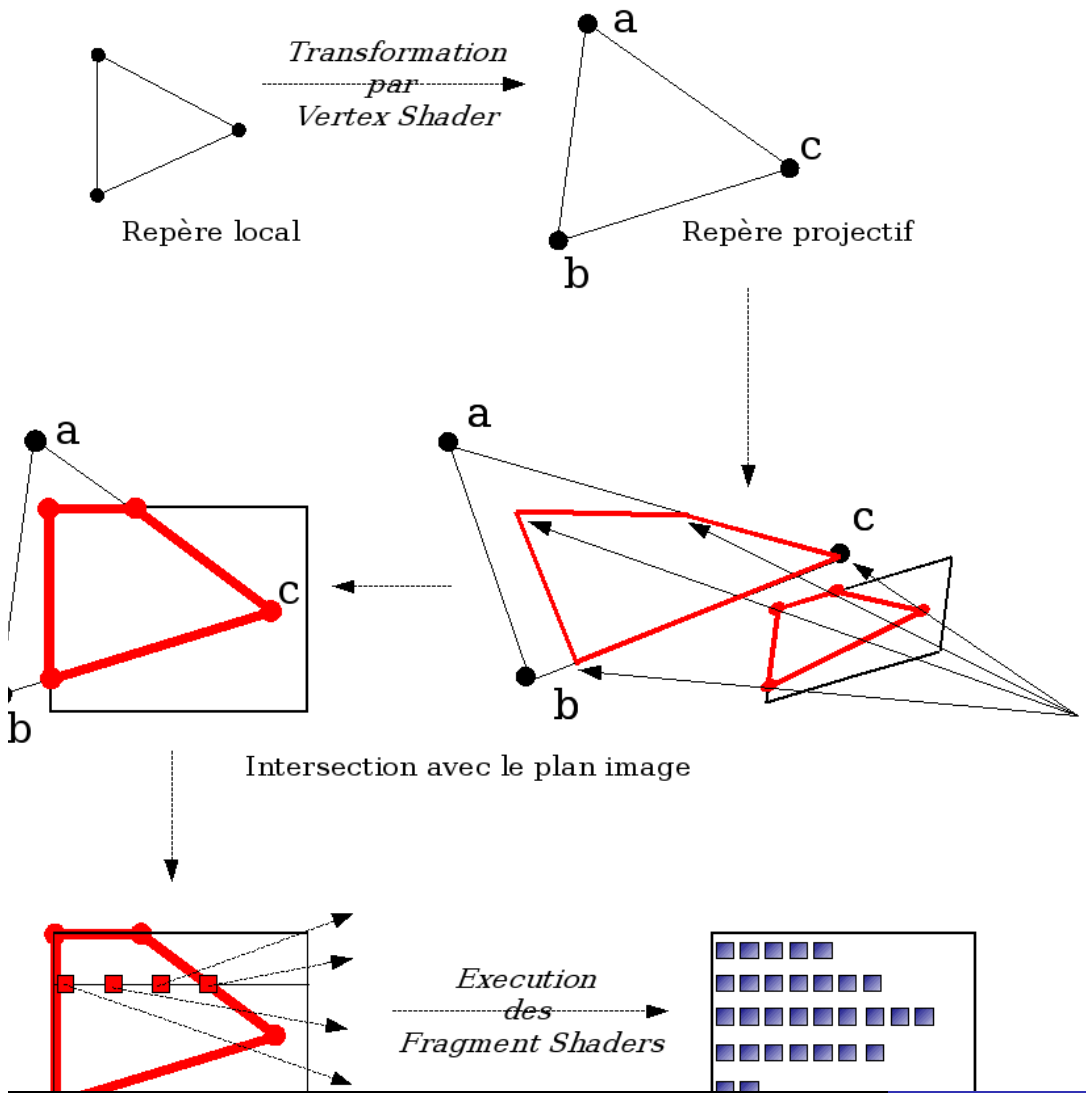
- Programmes chargés et compilés **à l'exécution** de l'application.
  - Instructions spécifiques.
- S'exécutent sur GPU :
  - Considéré comme une machine indépendante.
  - Pas de partage de données par défaut.
  - -> Passage de paramètres ?

LIRIS

Comment ça marche ?



Comment ça marche ?



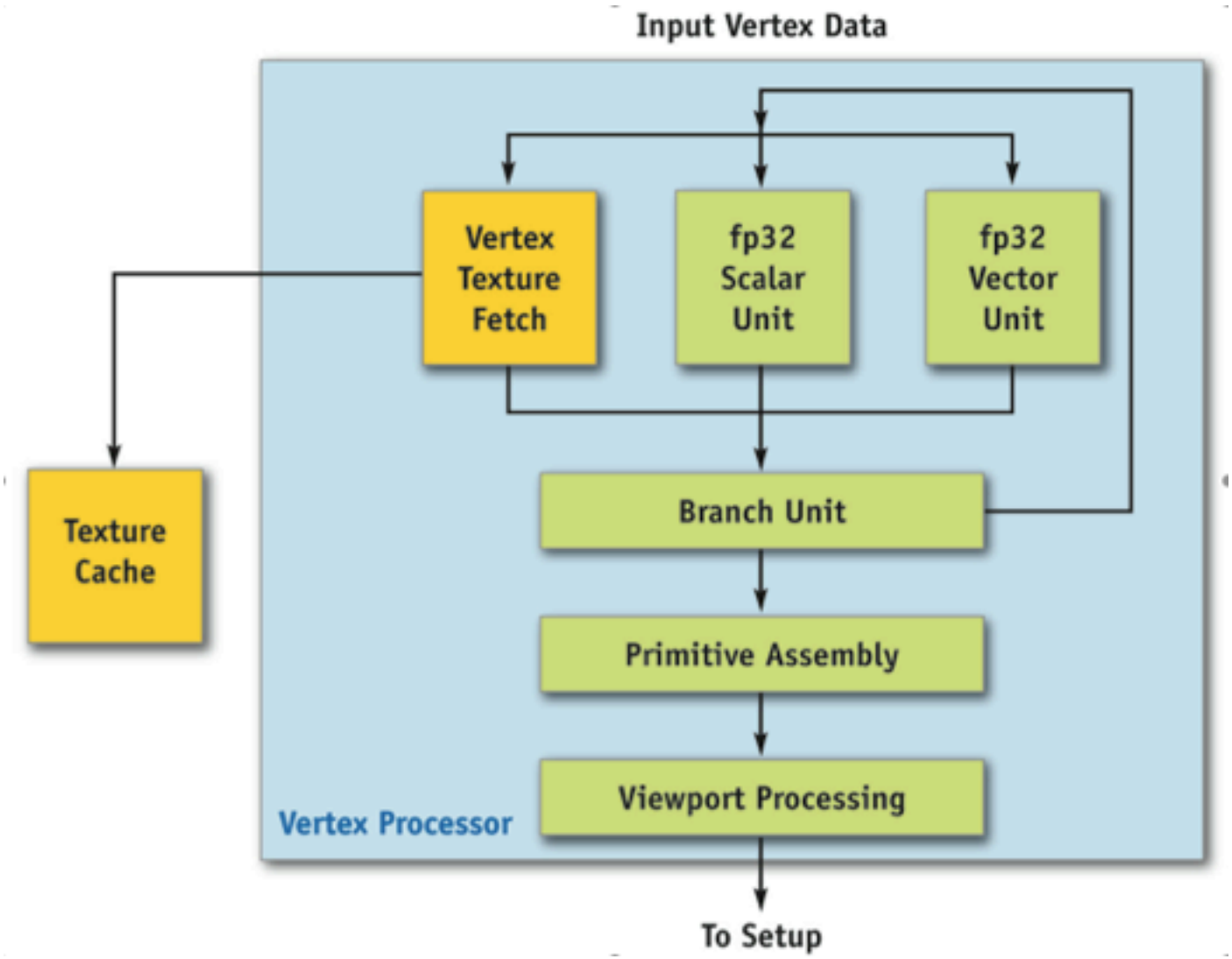
Programmation GPU

- Penser Stream processing !
  - Une seule donnée (vertex - fragment) à la fois.
  - Le frame buffer n'est pas accessible directement.
  - Des paramètres peuvent être transmis depuis l'application mais...
  - ... rien n'est modifiable en dehors des valeurs de sorties imposées.

Plan

- GPU - Pipeline graphique - rappels.
- Shaders
  - Vertex Shaders
  - Pixel Shaders
- API existantes
  - GLSL
  - Cg - HLSL
- HLSL : utilisation

Vertex Processor

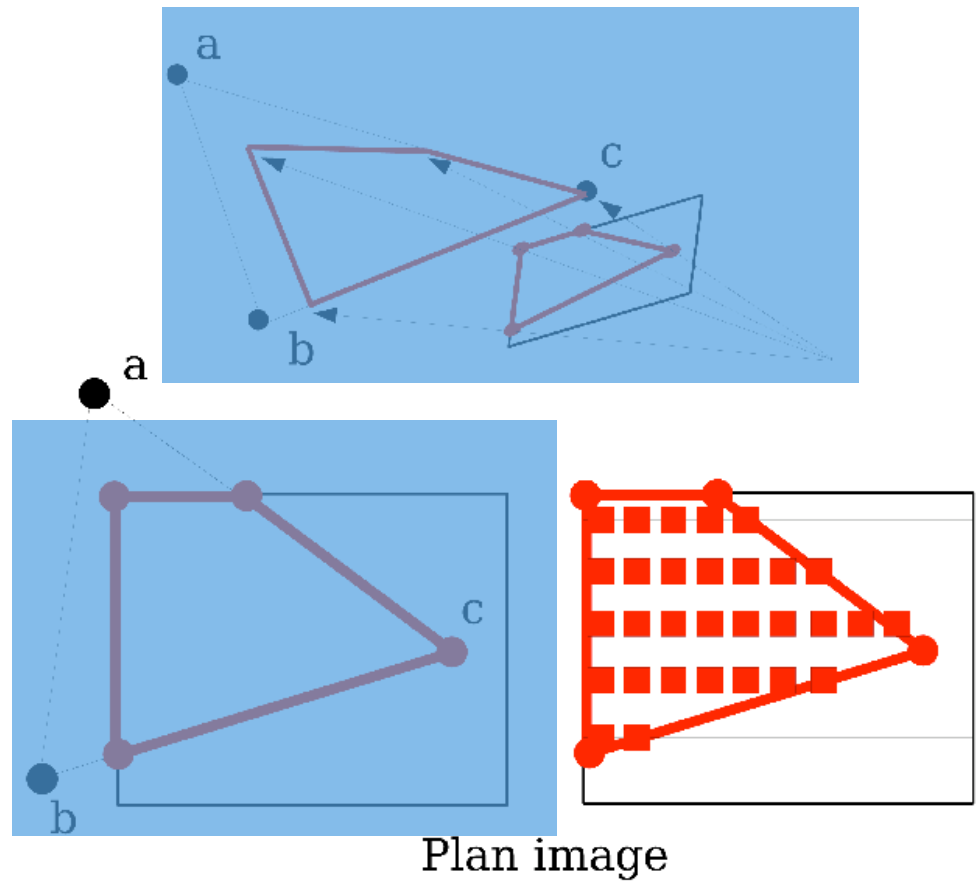


Vertex Shader

- En entrée :
  - Un vertex avec ses attributs.
- En sortie :
  - Une liste d'attributs pour ce vertex :
    - Nouvelle position.
    - Normale.
    - Couleur.
    - + tout traitement nécessaire...

Vertex Processor

- Reçoit les vertex de l'application.
- Transforme les vertex pour l'affichage.
- Calcule éventuellement leurs caractéristiques.

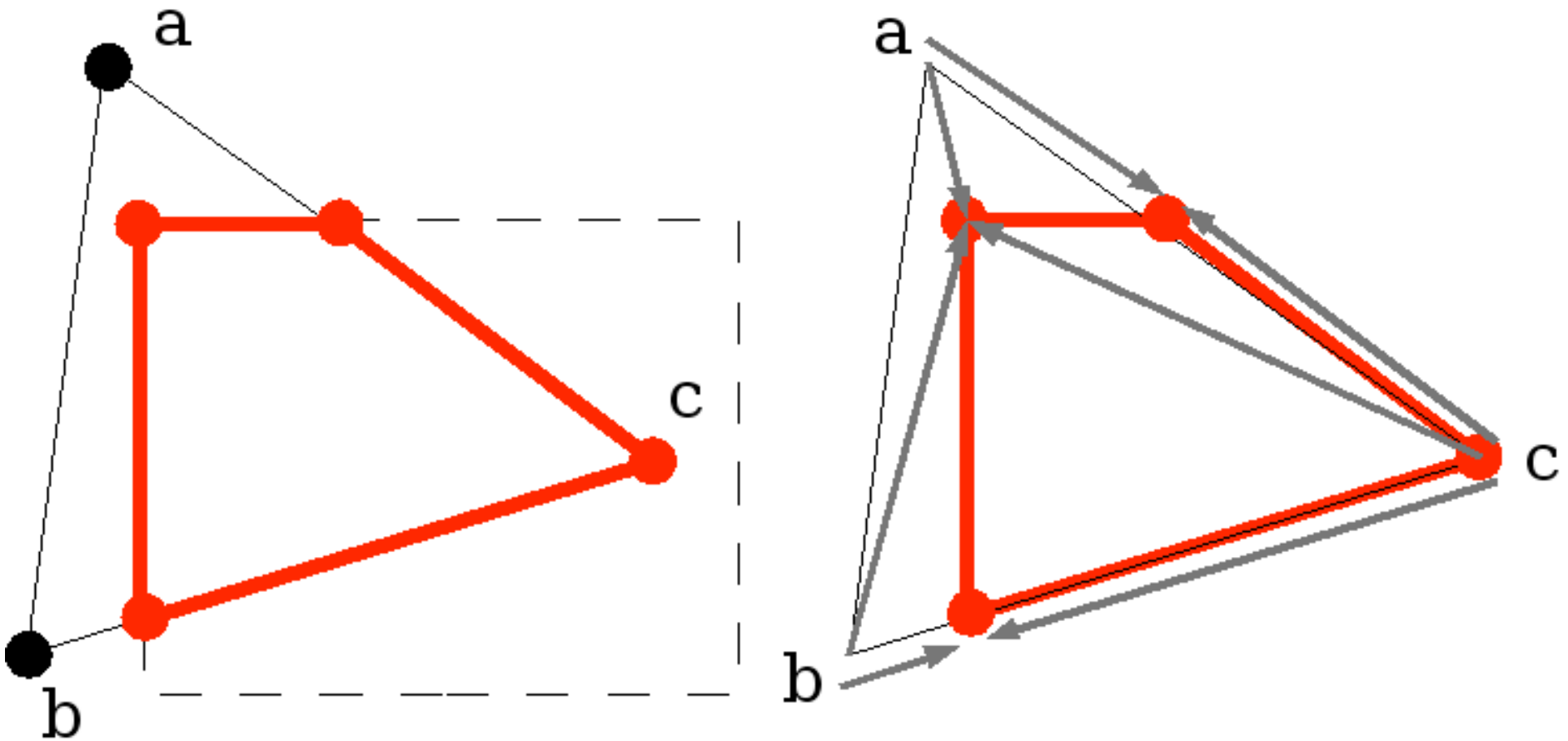


Vertex

- Sommet de la primitive + attributs
  - Position
  - Couleur
  - Matière
  - Normale
  - Autres définis par l'application
- Ces attributs seront interpolés dans le pipeline.

Interpolation des attributs

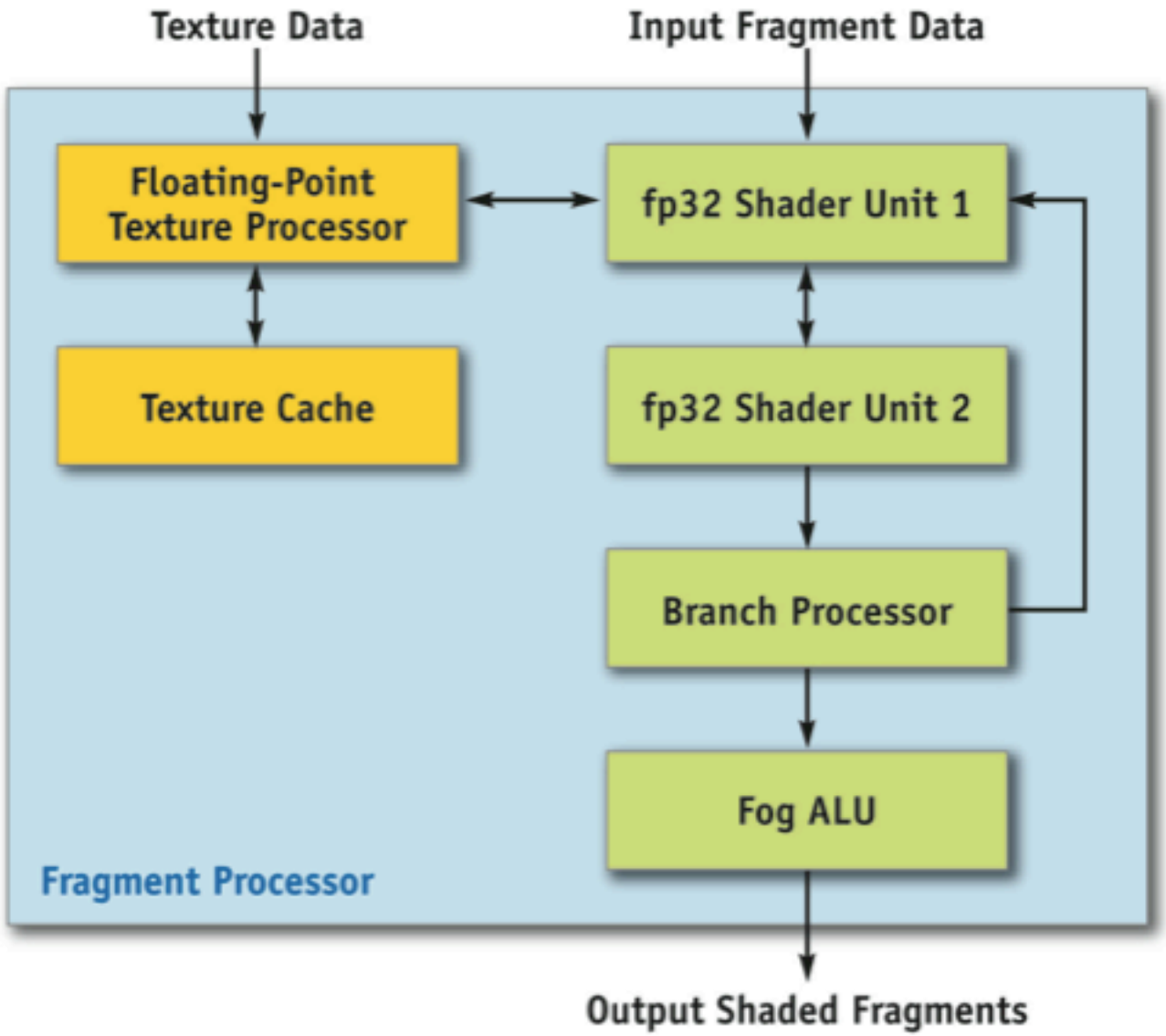
Le pipeline peut créer de nouveaux sommets...



Plan

- GPU - Pipeline graphique - rappels.
- Shaders
  - Vertex Shaders
  - Pixel Shaders
- API existantes
  - GLSL
  - Cg - HLSL
- HLSL : utilisation

Fragment Processor

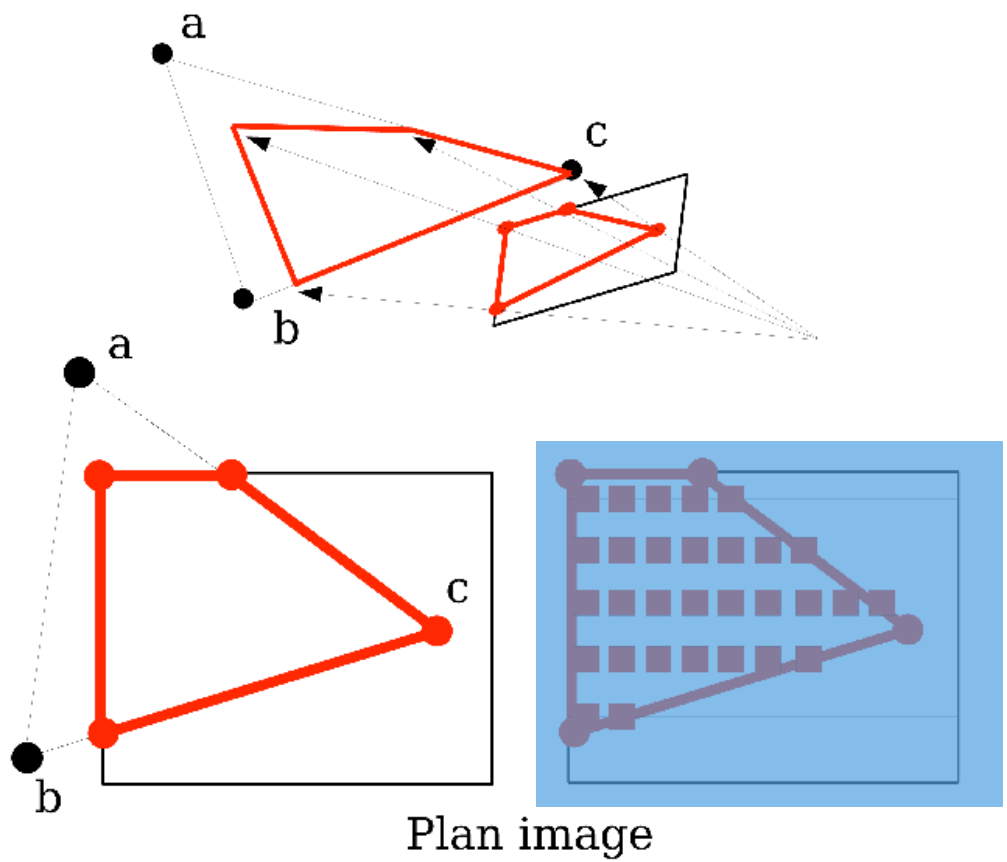


Fragment Shader

- En entrée :
  - Un fragment provenant du Rasterizer.
- En sortie :
  - Une liste de caractéristiques pour ce fragment.
    - Couleur.
    - Profondeur.
    - Transparence.
    - + Autres...

Fragment Processor

- Reçoit les données du Rasterizer.
- Transforme le fragment en pixel
  - =>Calcule l'apparence finale.



Fragment

- Elément de l'image + tout ses attributs
  - Position 3D, distance à la caméra.
  - Matière, couleur, transparence.
  - Autres définis par l'application...
- Ces attributs ont été définis par le Vertex Shader puis interpolés par le Rasterizer.
- Attention : il est impossible affecter un attribut à un fragment directement !

LIRIS

Particularités

- Un Fragment shader peut avoir une sortie multiple :
  - Multiple Render Target.
  - Ecriture de 4 fragments dans 4 buffers distincts.
- Un Fragment shader n'écrit pas nécessairement dans le framebuffer visible.
  - RenderBuffers, FBOs...

Plan

- GPU - Pipeline graphique - rappels.
- Shaders
  - Vertex Shaders
  - Pixel Shaders
- API existantes
  - GLSL
  - Cg - HLSL
- HLSL : utilisation

Notions globales

- Syntaxe proche du C++.
- Types de base : scalaires, matrices.
- Samplers : accès au texture.
- Accès au contexte OpenGL.
- Accès aux paramètres définis par l'application.

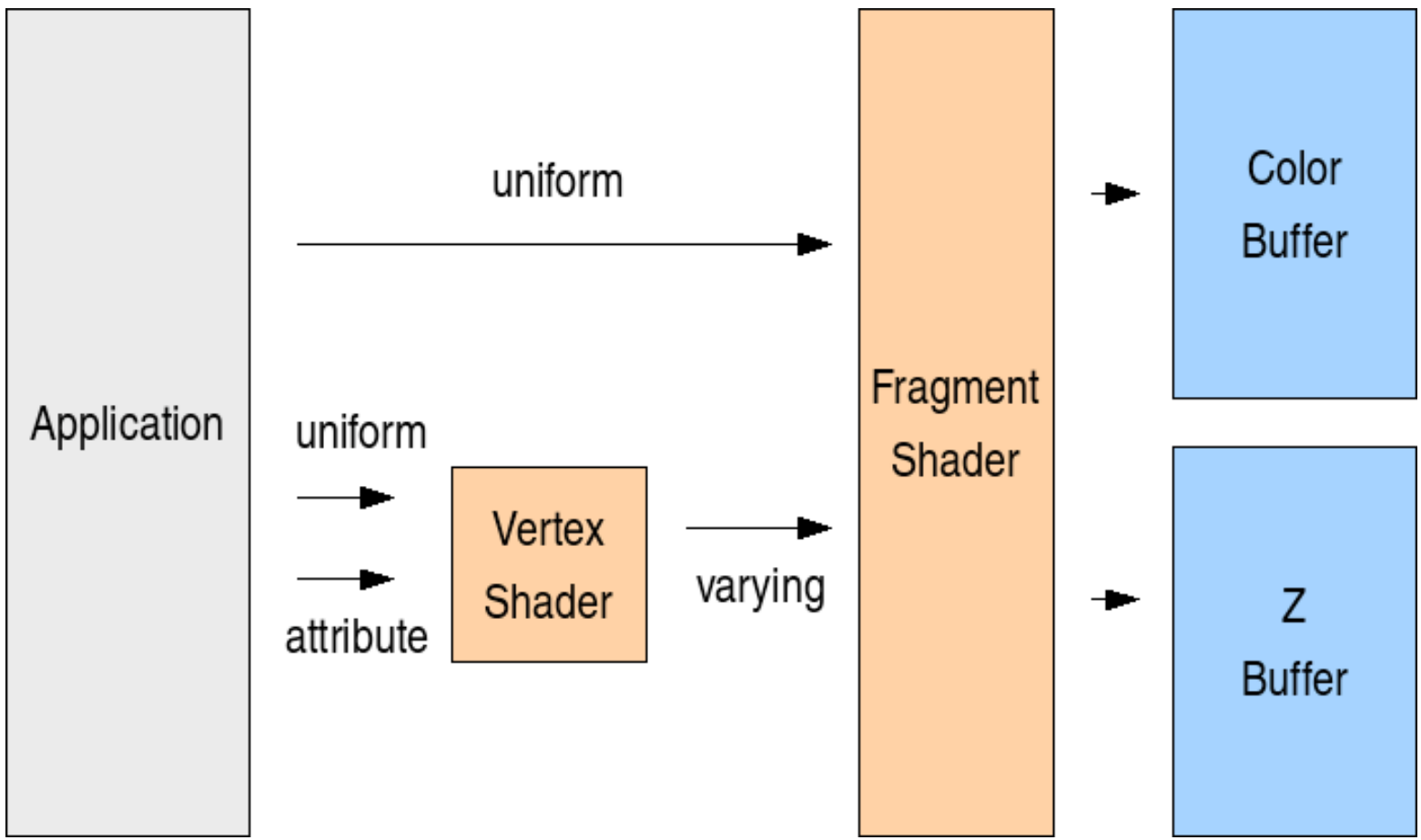
Création d'un shader

- Création d'un shader.
  - Un shader est une fonction.
  - Chargement du source depuis l'application.
  - Compilation.
- Création d'un programme
  - Il faut deux shaders pour un programme.
  - Edition de liens (link) des deux objets pour obtenir un progamme.

Paramètres

- 4 types de paramètres :
  - Uniform : définis par l'application pour chaque primitive.
  - Attribute : définis par l'application pour chaque vertex.
  - Varying : définis par le vertex shader, puis interpolés lors de la fragmentation.

Paramètres


Application
uniform
uniform
attribute
Vertex
Shader
varying
Fragment
Shader
Color
Buffer
Z
Buffer

API existantes

- Cg (nVidia)
- GLSL (3DLabs)
- HLSL (Microsoft - nVidia)

C for graphics

- Avantages :
  - Indépendant de l'API 3d (direct3D ou OpenGL).
  - Support actif.
  - Nombreux outils.
- Inconvénients :
  - Plus performant sur carte nVidia...
  - Outils annexes disponibles sous windows principalement.

GL Shading Language

- Avantages :
  - Utilisation plus simple que Cg.
  - Multi-plateformes, multi-matériel.
- Inconvénients :
  - OpenGL pas toujours bien supporté...
  - Pas vraiment d'outils de conception.

High Level SL

- Conçu par Microsoft et nVidia.
- De loin le plus utilisé en développement de jeux vidéos.
- Outils de création et de débug performants.
- Exclusif windows - directX...
- Celui que nous étudierons...

Plan

- GPU - Pipeline graphique - rappels.
- Shaders
  - Vertex Shaders
  - Pixel Shaders
- API existantes
  - GLSL
  - Cg - HLSL
- HLSL : utilisation

Déclaration

- Un shader = une chaine de caractère.
  - Stocké dans un fichier par exemple...
- Création : plus facile avec D3DX.
  - D3DXGetVertexShaderProfile
  - D3DXGetPixelShaderProfile
  - D3DXCompileShaderFromFile
  - CreateVertexShader
  - CreatePixelShader
- Rmq : pas de geometry shader dans Direct X 9...

Utilisation

- SetVertexShader()
- SetPixelShader()
- Fixer les valeurs des paramètres
- Dessiner la géométrie.

Paramètres

- Paramètres uniform :
  - Paramètres généraux, constants pour tous les sommets d'un objet.
- Paramètres varying
  - Paramètres associés aux entrées du shader.
    - Pour un vertex shader : les attributs du sommets.
    - Pour un pixel shader : les résultats du vertex shader.

Paramètres

- Déclaration :
  - Dans le code du shader.
- Affectation :
  - Par l'application (uniform).
  - Par l'application ou le shader précédent (varying).

Exemples

```
float4x4 modelviewproj;  // World * View * Projection

void main(in float4 position : POSITION,
  out float4 hpos : POSITION,
  out float4 diffus : COLOR0)
{
  hpos= mul(position, modelviewproj); // LH
  diffus= float4(1.f, 0.f, 0.f, 1.f);
}
```

```
void main(uniform float4x4 modelviewproj,
  in float4 position : POSITION,
  out float4 hpos : POSITION,
  out float4 diffus : COLOR0)
{
  hpos= mul(position, modelviewproj); // LH
  diffus= float4(1.f, 0.f, 0.f, 1.f);
}
```

Type des paramètres

- Passage par copie :
  - Déclaration explicite des paramètres d'entrée et de sortie.
  - in, out, inout.
  - Préciser éventuellement uniform.

Sémantique

- Important : déclarer l'utilisation des paramètres dans le shader.
  - Position.
  - Couleur.
  - Normale...
- Permet de connecter les attributs des sommets aux entrées du shader.
- Permet de connecter les paramètres de sortie d'un shader aux entrées du shader suivant.

Exemple

```
void vertex_main(uniform float4x4 modelviewproj,
  in float4 position : POSITION,
  out float4 hpos : POSITION,
  out float4 diffus : COLOR0)
{
  hpos= mul(position, modelviewproj); // LH
  diffus= float4(1.f, 0.f, 0.f, 1.f);
}

float4 fragment_main(in float4 position : POSITION,
  in float4 couleur : COLOR0) : COLOR
{
  return couleur;
}
```

Passage de paramètres

- Depuis l'application
- Uniform :
  - ID3DXConstantTable::SetMatrix(«param», valeur)
  - ID3DXConstantTable::SetFloat («param», valeur)
  - ID3DXConstantTable::SetVector («param», valeur)
  - ID3DXConstantTable::SetInt («param», valeur)
- Varying :
  - SetVertexDeclaration
  - SetStreamSource

LIRIS

Vertex shader

- Exécuté sur chaque sommet.
- Doit calculer :
  - Transfo projection.
    - Valeur par défaut : modelview * position.
    - Position est dans le repère local de l'objet.
    - Résultat dans l'espace projectif de la caméra.
  - Tous les varying utilisés par le reste du pipeline.
- Peut éventuellement calculer :
  - couleur, coordonnées de textures, etc...
  - Cf référence HLSL, semantics.

LIRIS

Fragment shader

- Exécuté sur chaque fragment dessiné.
- Doit calculer la couleur (COLOR) et la profondeur (DEPTH).
- Travaille dans le repère de la fenêtre d'affichage (viewport).
- Attention : les varyings sont interpolés !

ToDo

- Chargement, installation et compilation d'un shader avec HLSL.

Travail pratique

- TD de Jean-Claude Iehl :
  - http://www710.univ-lyon1.fr/~jciehl/Public/educ/GAMA/2008/dxtd1.html