Rendu temps-réel et programmation GPU

Le pipeline graphique

Modèle géométrique : objets, surfaces, sources de lumière…

Modèle d'illumination : calcul des interactions lumineuses

Caméra : point de vue et ouverture (frustum)

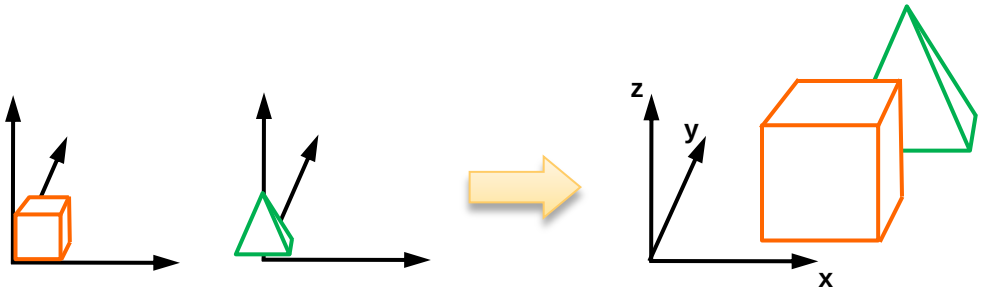
Fenêtre (viewport) : grille de pixel sur laquelle on plaque l'image

- **Modeling Transformations**
- **Illumination (Shading)**
- **Viewing Transformation (Perspective / Orthographic)**
- **Clipping**
- **Projection (to Screen Space)**
- **Scan Conversion (Rasterization)**
- **Visibility / Display**

Couleurs, intensités convenant à l'afficheur (ex : 24 bits, RVB)

Transformations objet

- Passage du système de **coordonnées local** de chaque objet 3D (object space) vers un **repère global** (world space)



Object space **World space**

Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

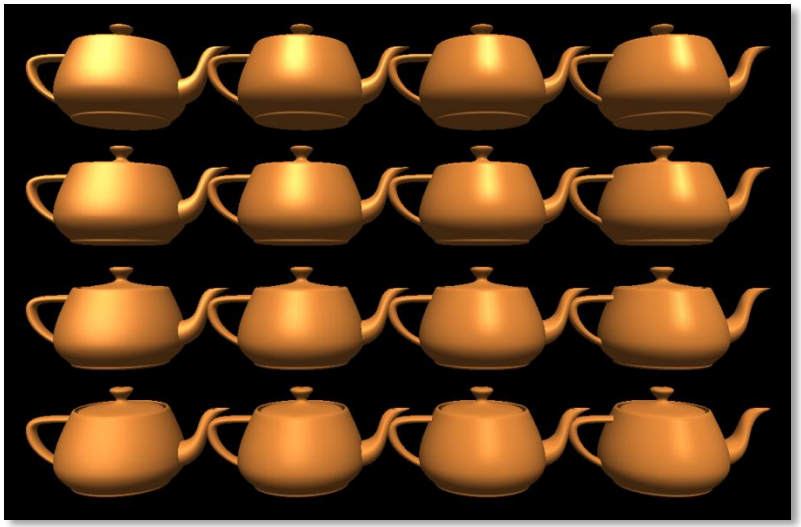
Projection (to Screen Space)

Scan Conversion (Rasterization)

Visibility / Display

Ilumination

- Les **primitives** sont éclairées selon leur matériau, le type de surface et les sources de lumière.
- Les **modèles d'illumination** sont **locaux** (pas d'ombres) car le calcul est effectué par primitive.



Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

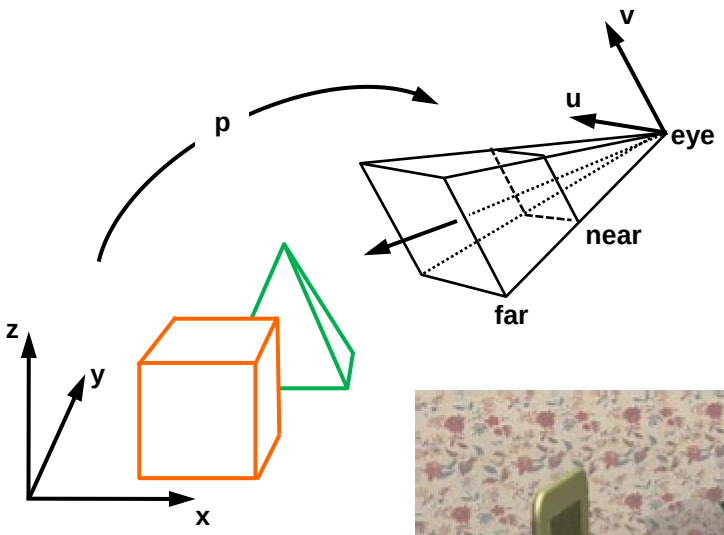
Projection (to Screen Space)

Scan Conversion (Rasterization)

Visibility / Display

Transformation caméra

- Passe des coordonnées du monde à celles du **point de vue** (repère caméra ou eye space).



Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

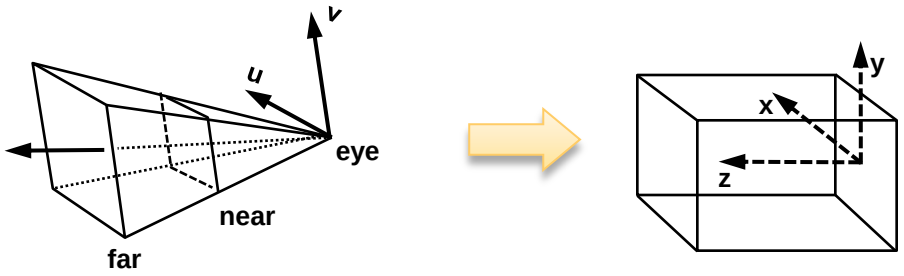
Projection (to Screen Space)

Scan Conversion (Rasterization)

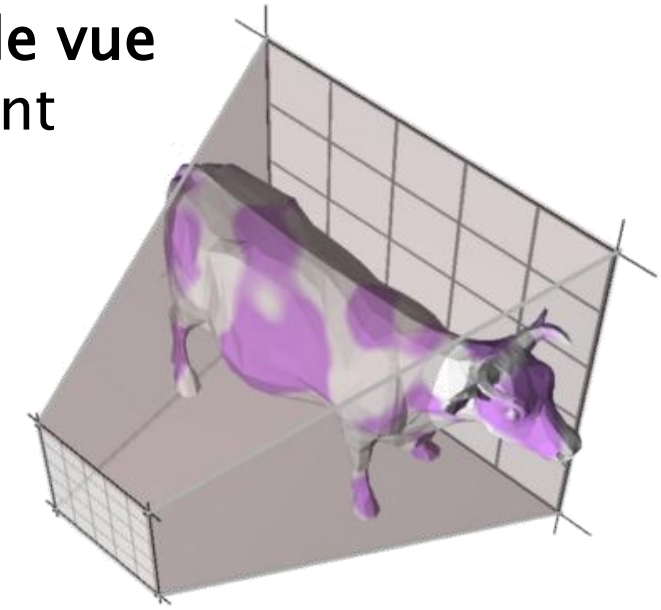
Visibility / Display

Clipping

- **Coordonnées normalisées :**



- Les portions en dehors du **volume de vue** (frustum) sont coupées.



Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

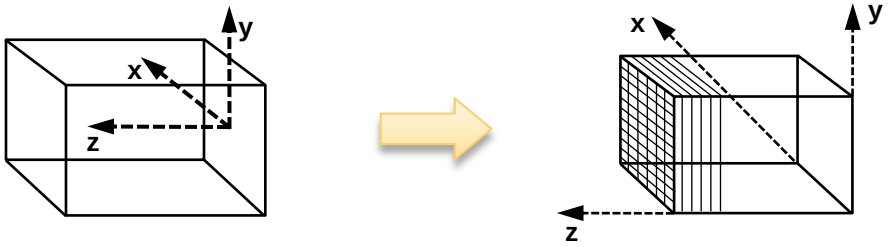
Projection (to Screen Space)

Scan Conversion (Rasterization)

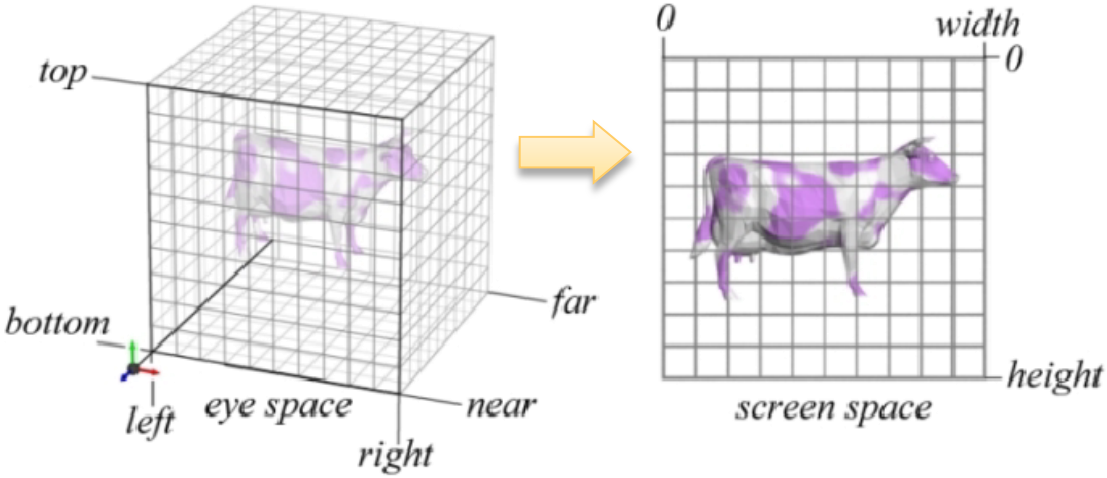
Visibility / Display

Projection

- Les primitives 3D sont **projetées** sur l'image 2D (screen space)



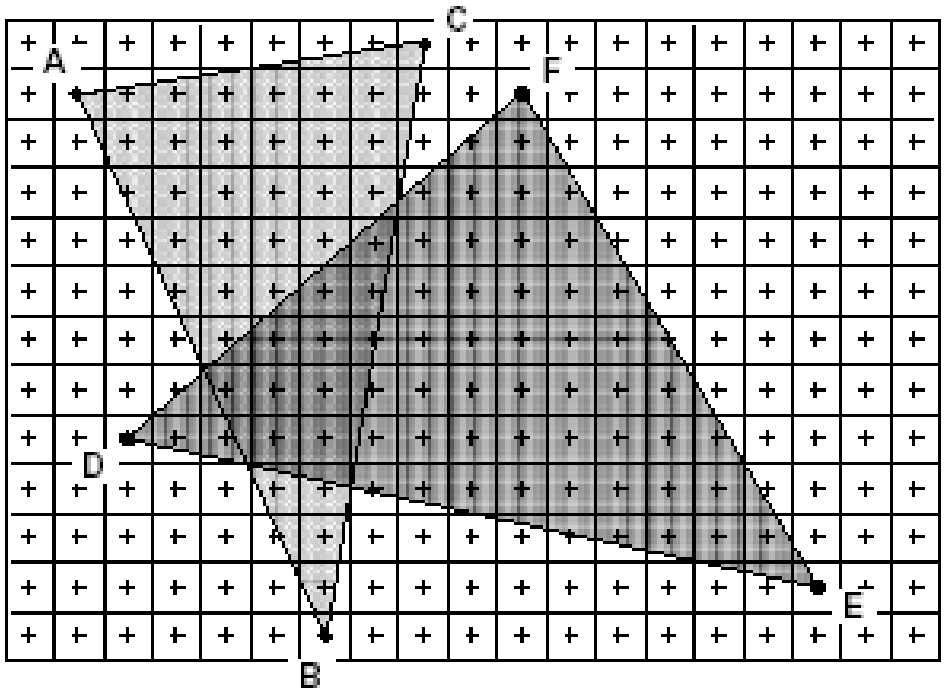
NDC **Screen Space**



- Modeling Transformations
- Illumination (Shading)
- Viewing Transformation (Perspective / Orthographic)
- Clipping
- **Projection (to Screen Space)**
- Scan Conversion (Rasterization)
- Visibility / Display

Rastérisation

- Découpe la primitive 2D en **pixels**
- **Interpole** les valeurs connues aux sommets : couleur, profondeur,…



Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

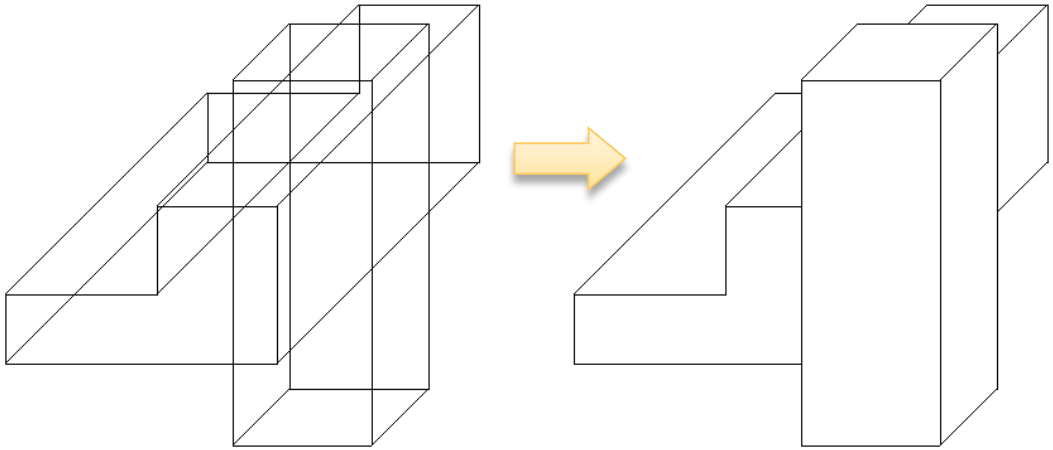
Projection (to Screen Space)

Scan Conversion (Rasterization)

Visibility / Display

Visibilité, affichage

- Calcul des **primitives visibles**
- Remplissage du **frame buffer** avec le bon format de couleur



Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

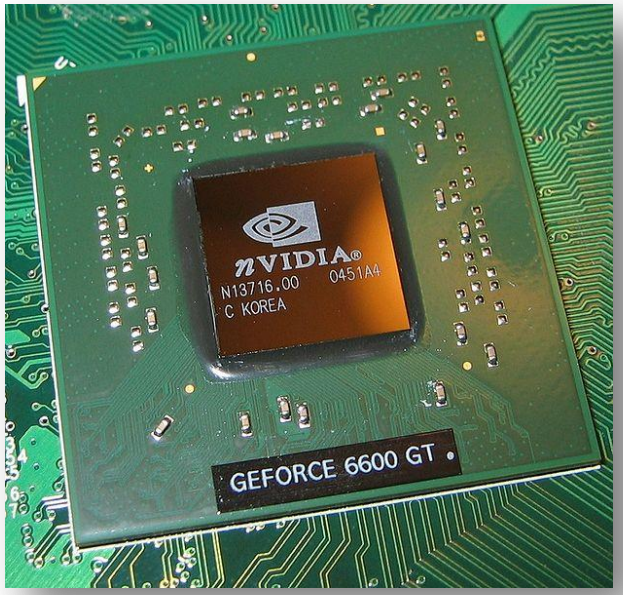
Projection (to Screen Space)

Scan Conversion (Rasterization)

Visibility / Display

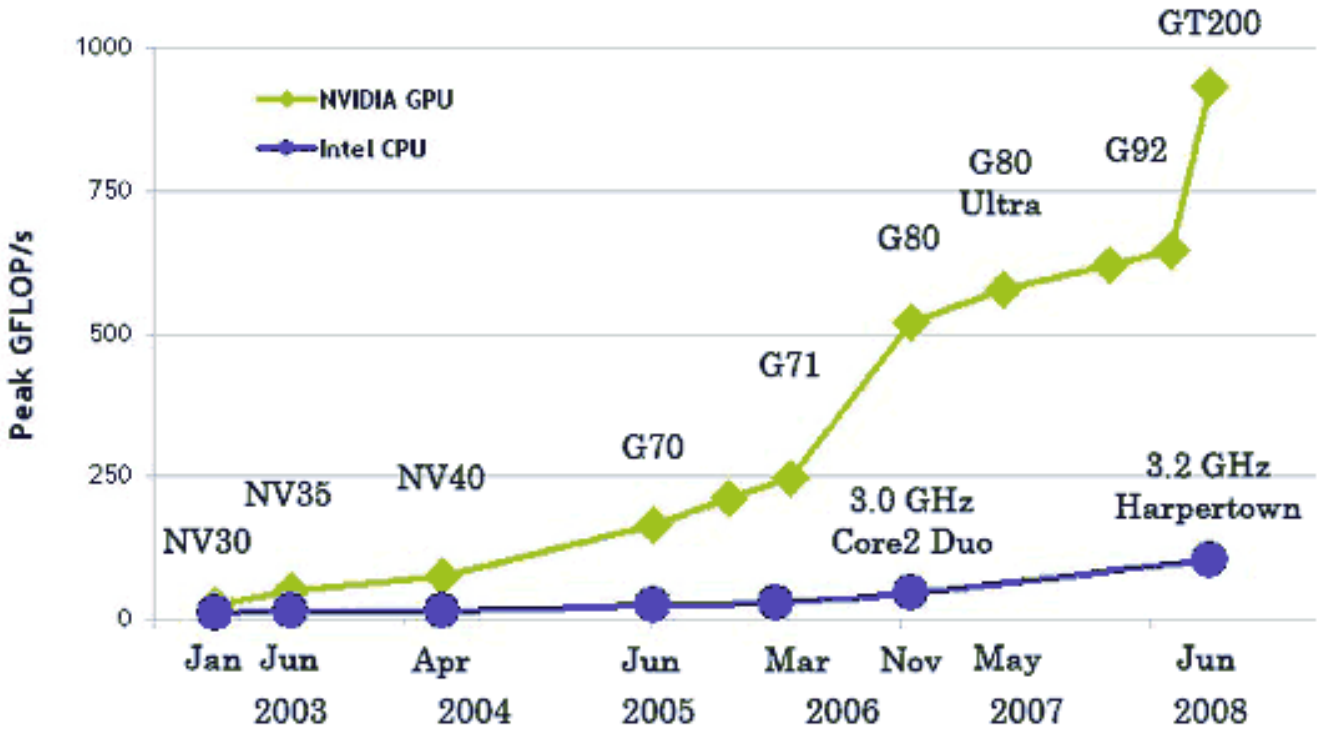
C’est quoi un GPU ?

- « Graphics Processing Unit »
- Processeur spécialisé pour le rendu 3D
- Spécificités :
  - Architecture hautement parallèle
  - Accès mémoire rapide
  - Large bande passante



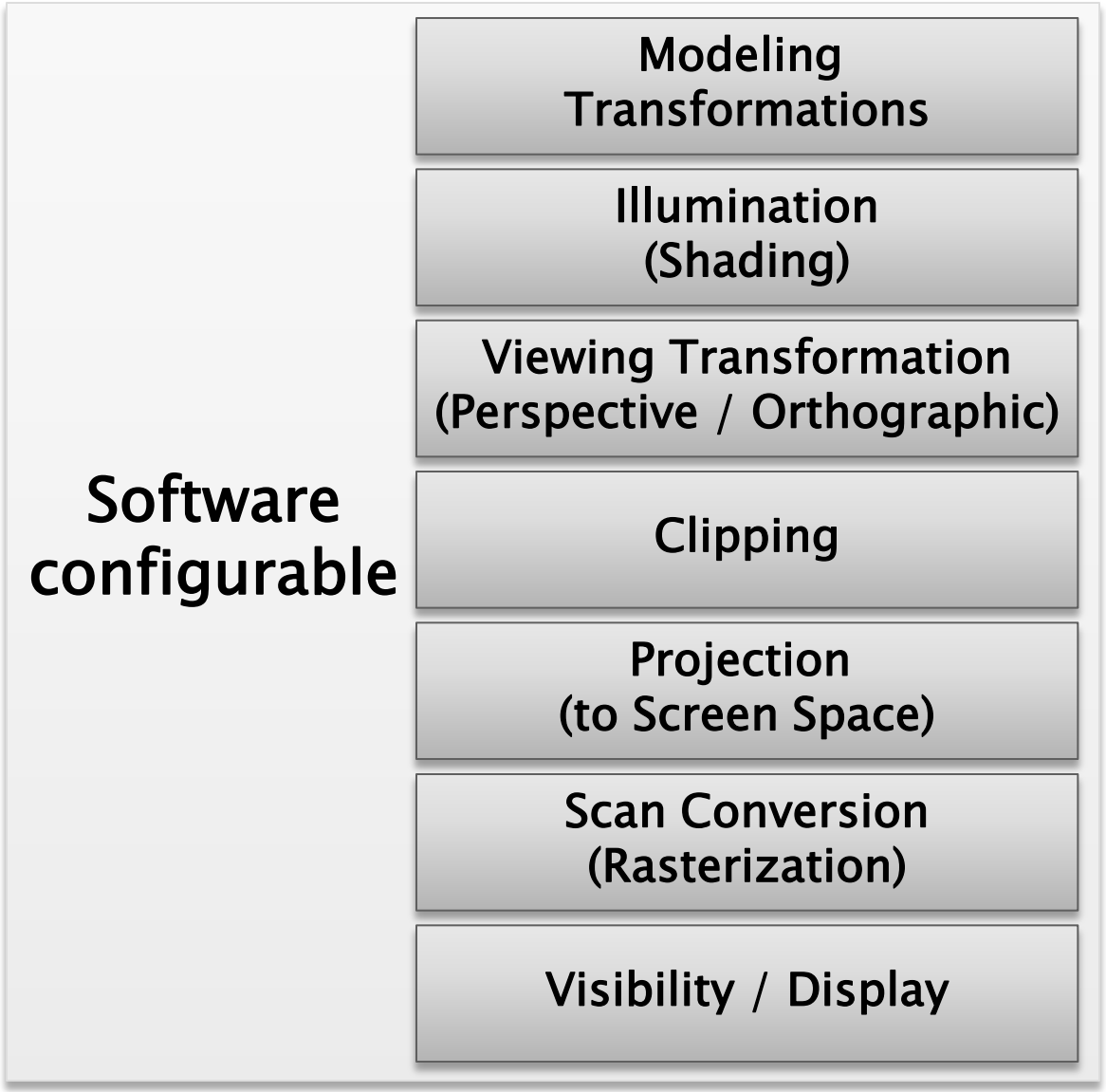
C'est quoi un GPU ?

- Un monstre de calcul parallèle :
  - GPGPU : « General-Purpose computation on GPU »



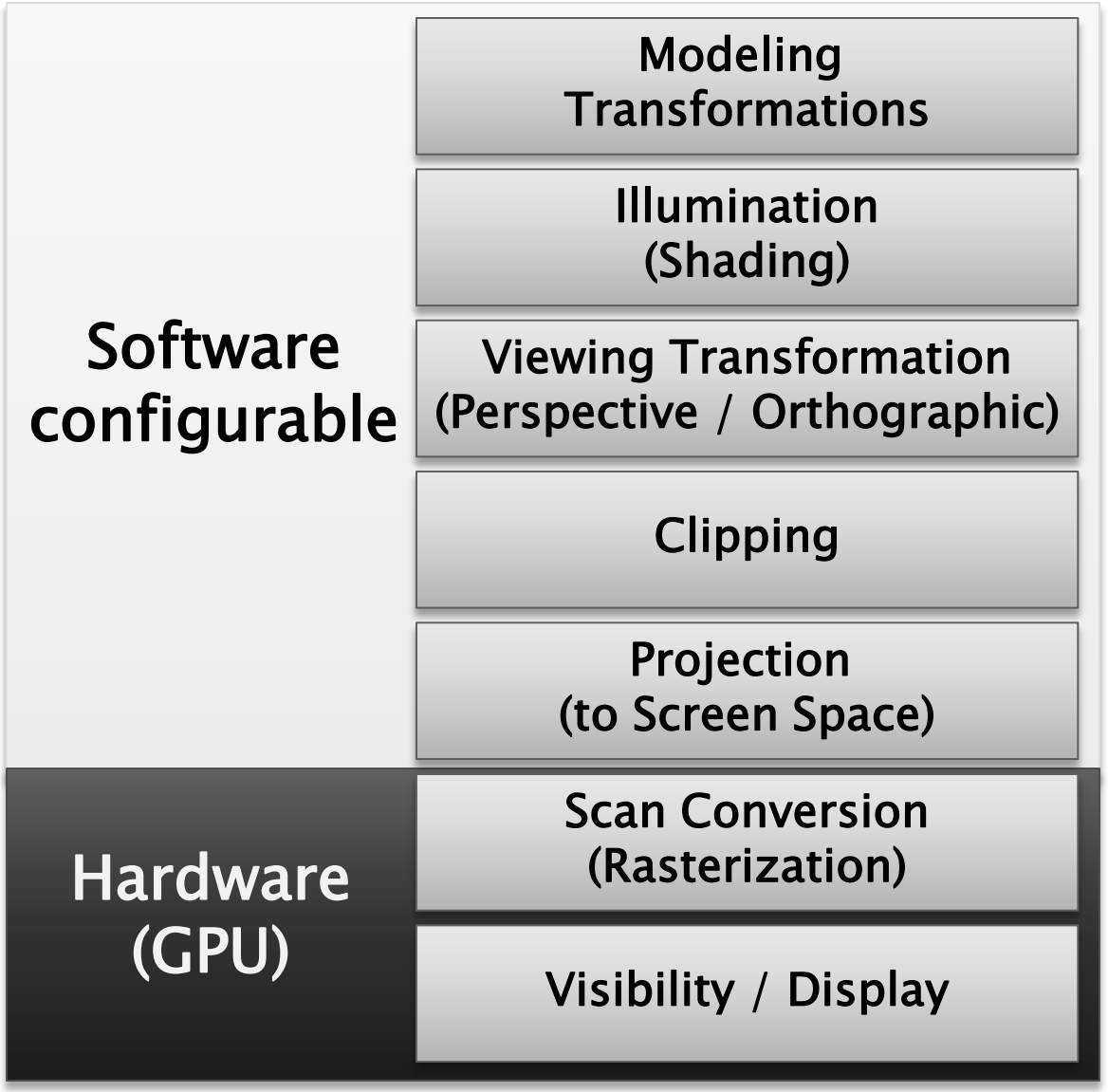
Le pipeline graphique

Sans carte graphique 3D
(1970s)



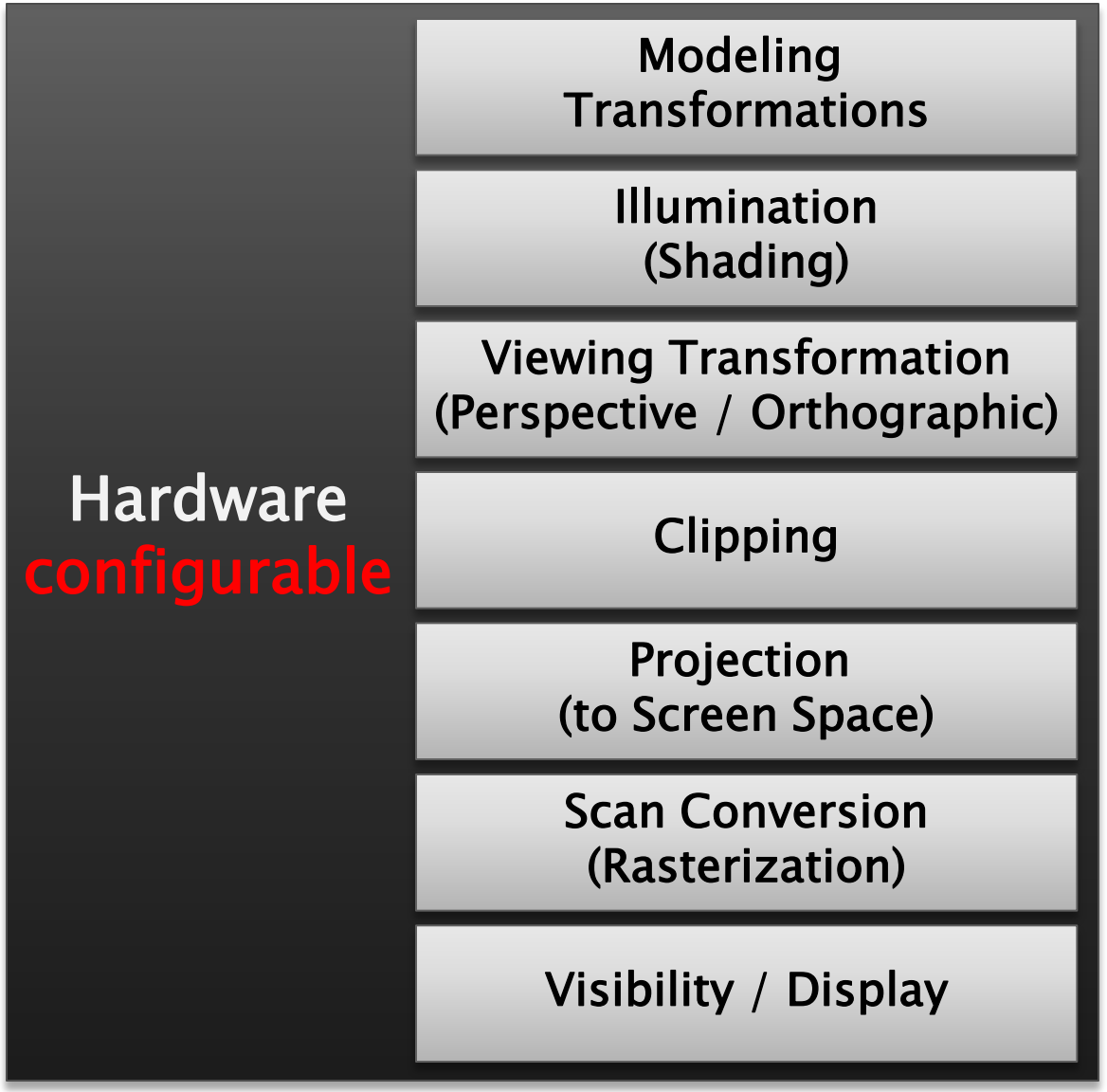
Le pipeline graphique

Cartes graphiques première génération (1980s)



Le pipeline graphique

Cartes graphiques deuxième génération
(1990s)

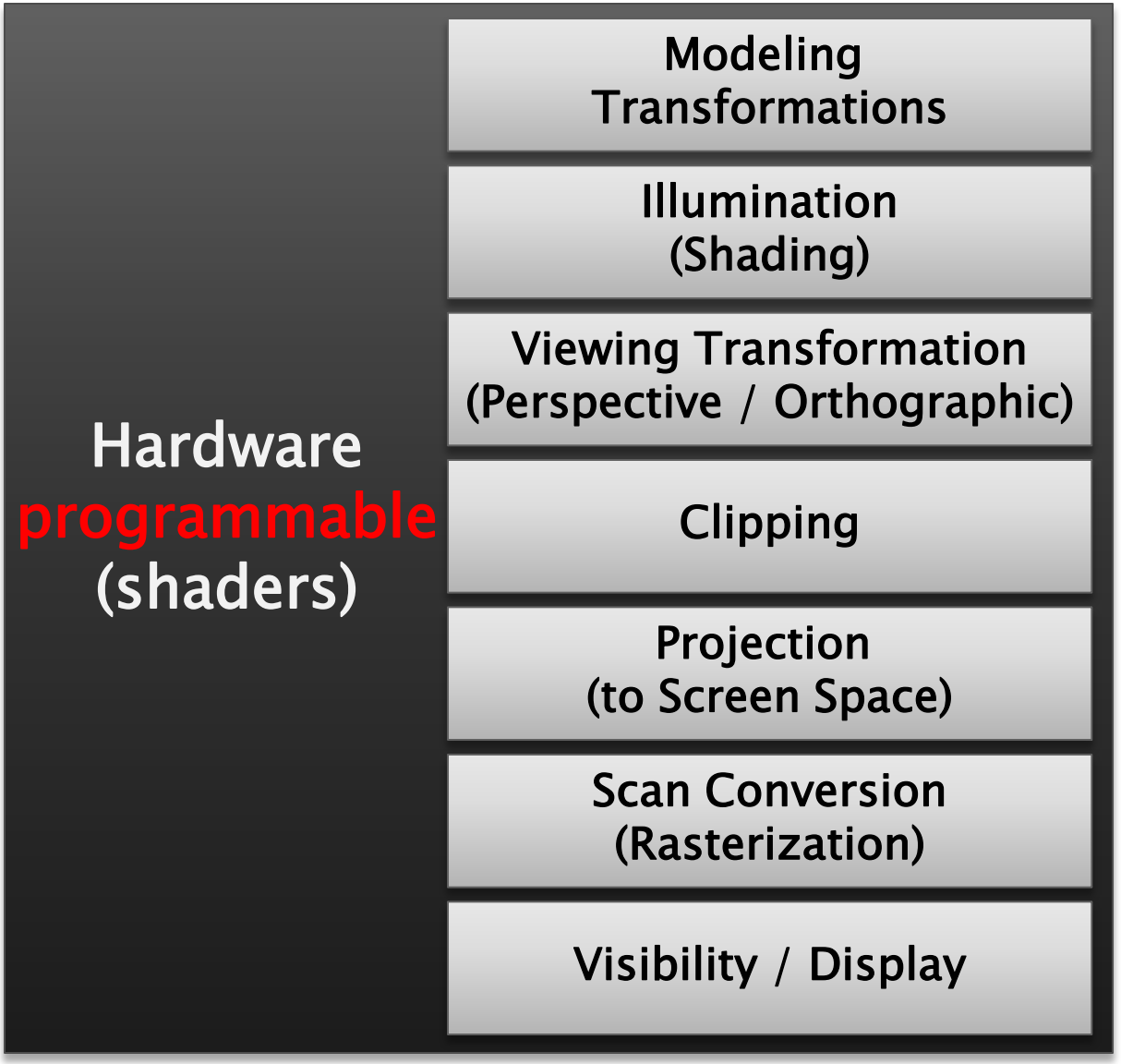


Configurable ?

- **API** (Application Programming Interface) pour l'architecture graphique
- 2 API prépondérante en graphique :
  - Direct3D (Microsoft)
  - **OpenGL** (Khronos Group)

Le pipeline graphique

Cartes graphiques troisième génération (2000s)

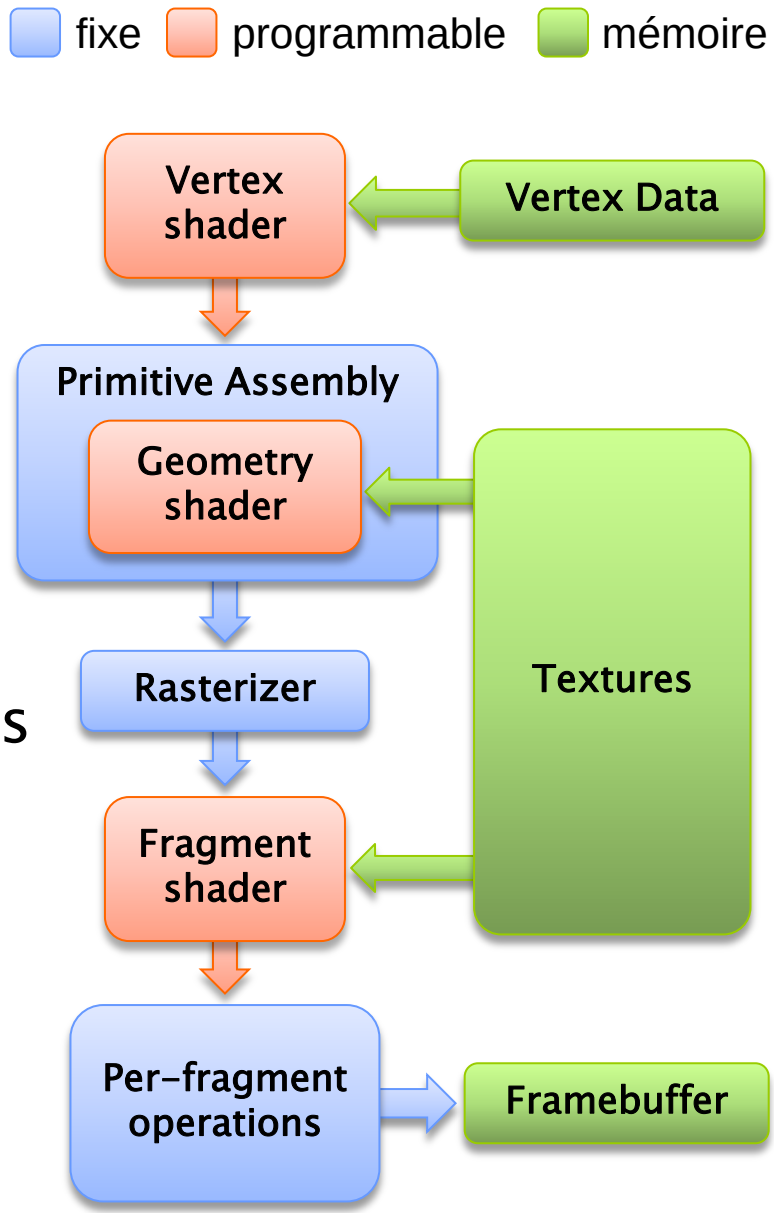


Programmable ?

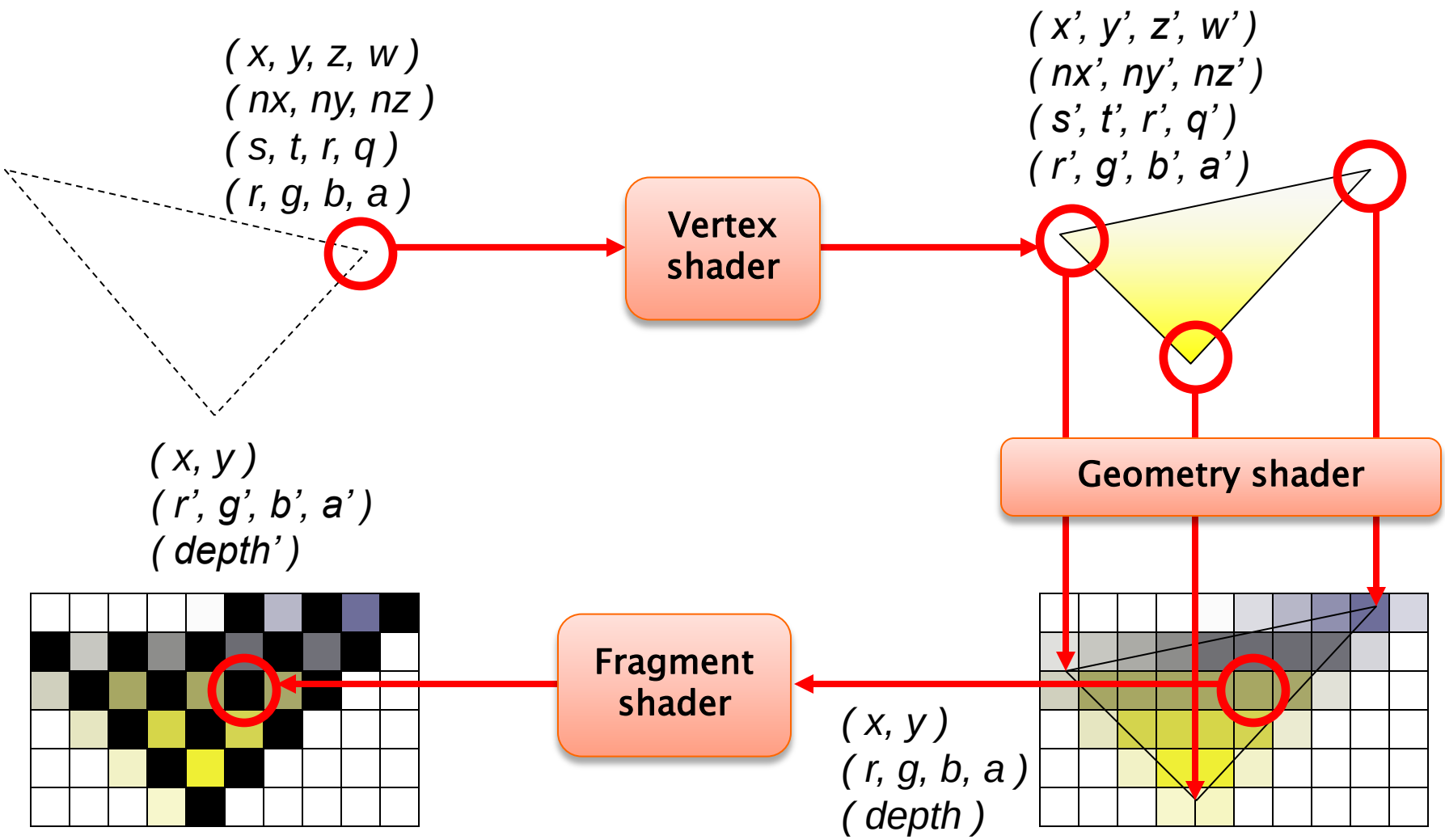
- **Shaders :**
  - suite d'instructions exécutable par le GPU à différentes étapes du pipeline
  - Langage différent (pseudo-C) en fonction des API :
    - NVIDIA ⇨ Cg (2002)
    - Direct3D ⇨ HLSL (2003)
    - OpenGL ⇨ **GLSL** (2004)
- Pour le GPGPU :
  - CUDA (NVIDIA)
  - ATI Stream
  - OpenCL (Khronos Group)

Shaders

- 3 types de shaders
  1. **Vertex** shader
  2. **Geometry** shader
  3. **Pixel** shader
- Action **locale**
  1. un sommet
  2. une primitive et ses voisines
  3. un pixel

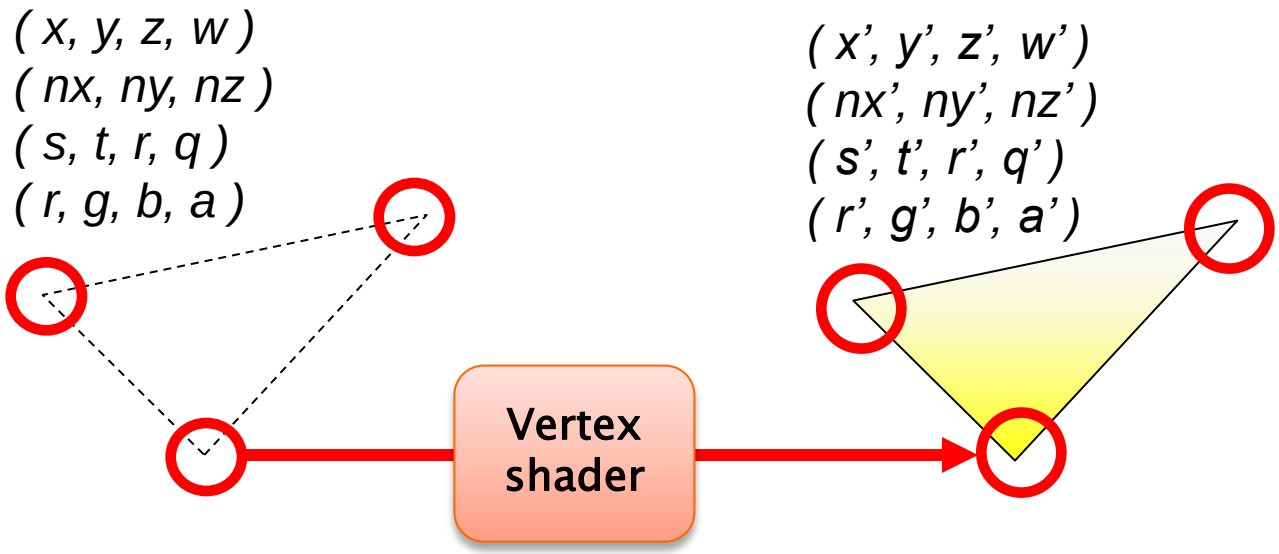


Shaders


(x, y, z, w)
(nx, ny, nz)
(s, t, r, q)
(r, g, b, a)
Vertex shader
(x', y', z', w')
(nx', ny', nz')
(s', t', r', q')
(r', g', b', a')
Geometry shader
(x, y)
(r', g', b', a')
(depth')
Fragment shader
(x, y)
(r, g, b, a)
(depth)

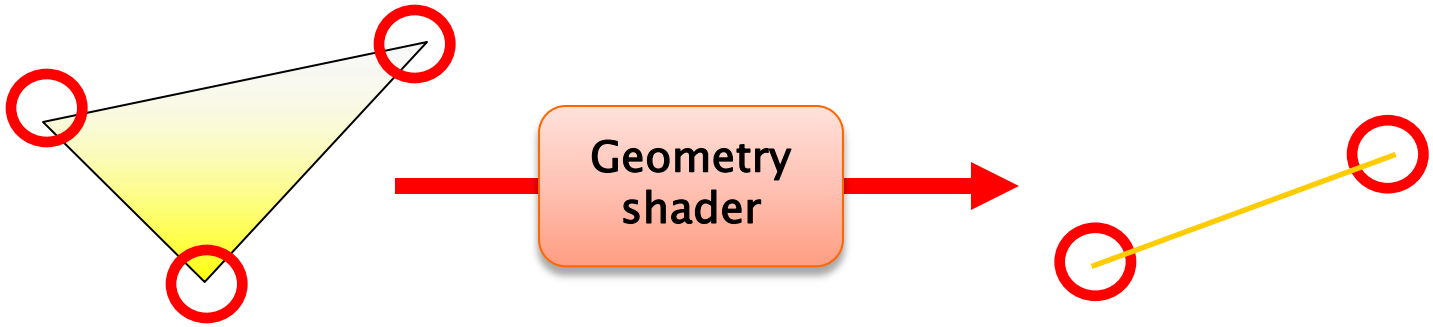
Vertex shader

- Ce qu'on peut faire
  - des transformations de position
  - des calculs d'illumination, de couleurs par sommet
  - des calculs de coordonnées de textures



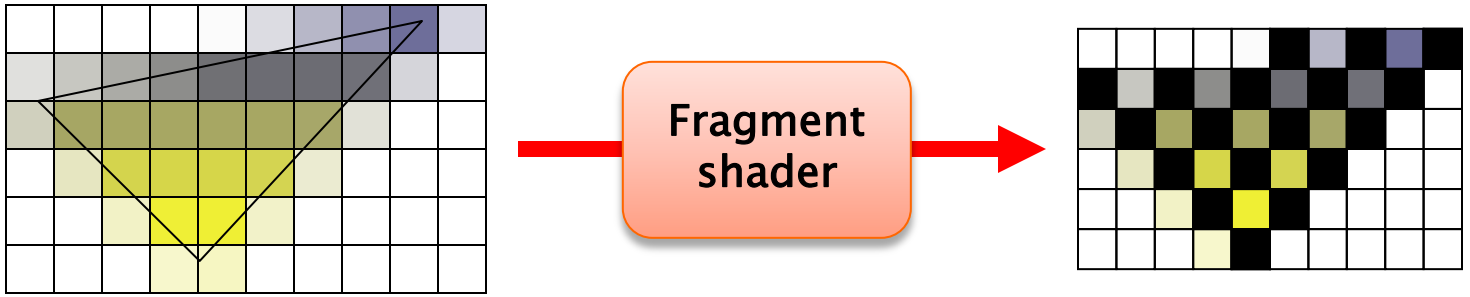
Geometry shader

- Ce qu'on peut faire
  - ajouter/supprimer des sommets
  - modifier les primitives
  - récupérer directement la géométrie sans « tramage » (avant la rasterisation)



Fragment shader

- Ce qu'on peut faire
  - la même chose qu'aux sommets, mais par pixel
  - utiliser le contenu de textures dans des calculs
  - changer la profondeur des pixels



Premier Vertex Shader

```
uniform mat4 modelViewProjectionMatrix;
in vec4 vertex;
out vec3 color;

vec4 UneFonction( vec4 Entree )
{
        return Entree.zxyw;
}

void main()
{
        vec4 pos = modelViewProjectionMatrix * vertex;
        gl_Position = pos + UneFonction( vertex );
        color = vec3(1.0,0.0,0.0);
}
```

Entrées

Sortie

Fonction

Swizzle

Point d'entrée

Variable locale

Multiplication matrice-vecteur

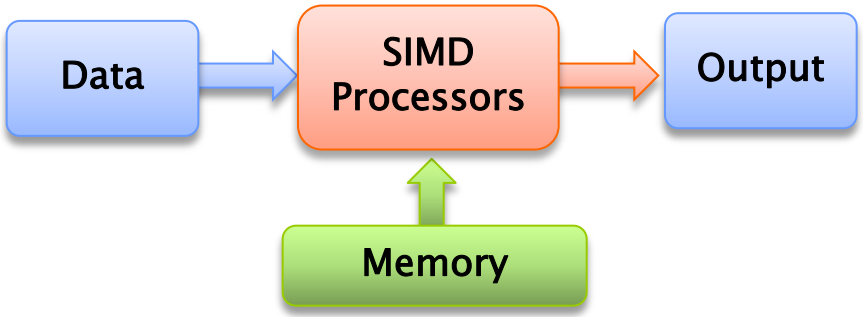
Sortie OpenGL

Conseils

- Développez petit à petit et **testez souvent** !
  - Débuggage très difficile
- **Optimisation**
  - Réfléchissez au meilleur endroit pour placer un calcul :
    - Vertex shader : 1x par sommet
    - Fragment shader : 1x par fragment : beaucoup plus souvent !
  - Textures pour encoder les fonctions trop complexes
  - **Utilisez les fonctions fournies** plutôt que de les re-développez.

GPGPU

- *General-Purpose Computation Using Graphics Hardware*
- Un GPU = un processeur SIMD (*Single Instruction Multiple Data*)
- Une texture = un tableau d'entrée
- Une image = un tableau de sortie



GPGPU – Applications

- Rendu avancé
  - Illumination globale
  - Image-based rendering
  - …
- Traitement du signal
- Géométrie algorithmique
- Algorithmes génétiques
- A priori, tout ce qui peut massivement se paralléliser

GPGPU

- Récupérer l'image rendue = lent
  - PCI Express
- Opérateurs, fonctions, types assez limités
- Un algorithme parallélisé n'est pas forcément plus rapide que l'algorithme séquentiel

Références/Liens utiles

- Le red book : http://www.opengl-redbook.com/
- La spec GLSL : http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.30.08.pdf
- Cg : http://developer.nvidia.com/page/cg_main.html
- Cuda : http://www.nvidia.com/cuda
- OpenCL : http://www.kronos.org/opencl/

- Librairie pour les extensions
  - GLEW : http://glew.sourceforge.net/

- Un éditeur spécial shader (malheureusement pas à jour, mais bien pour débuter)
  - http://www.typhoonlabs.com/

- Erreurs OpenGL/GLSL : un débogueur simple, efficace, super utile, vite pris en main.
  - glslDevil : http://www.vis.uni-stuttgart.de/glsldevil/

- Des tas d'exemples (à tester, éplucher, torturer) :
  - http://developer.nvidia.com/object/sdk_home.html

- La référence GPGPU avec code, forums, tutoriaux : http://www.gpgpu.org/