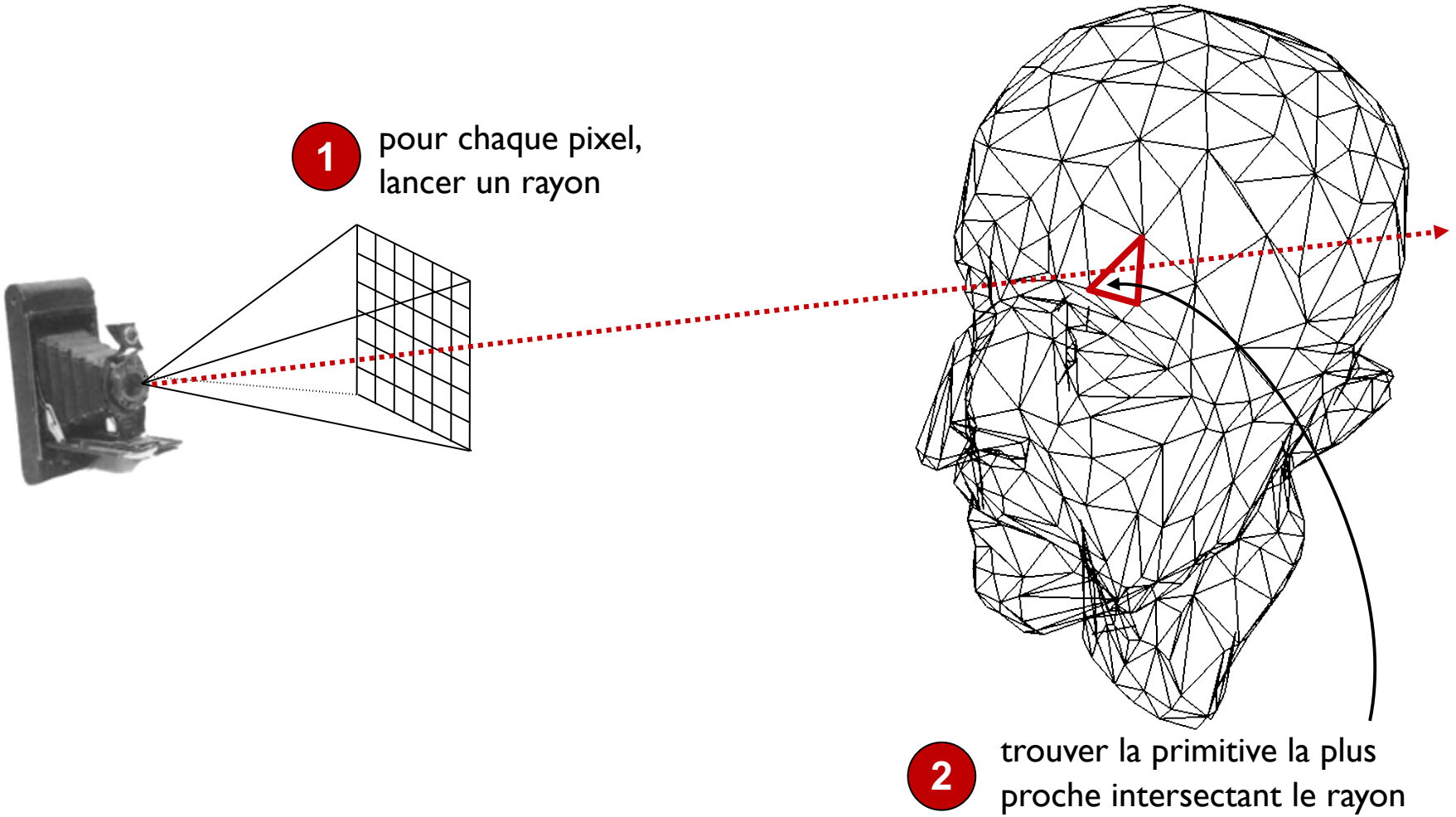
Rappels OpenGL

Visualiser une scène 3D

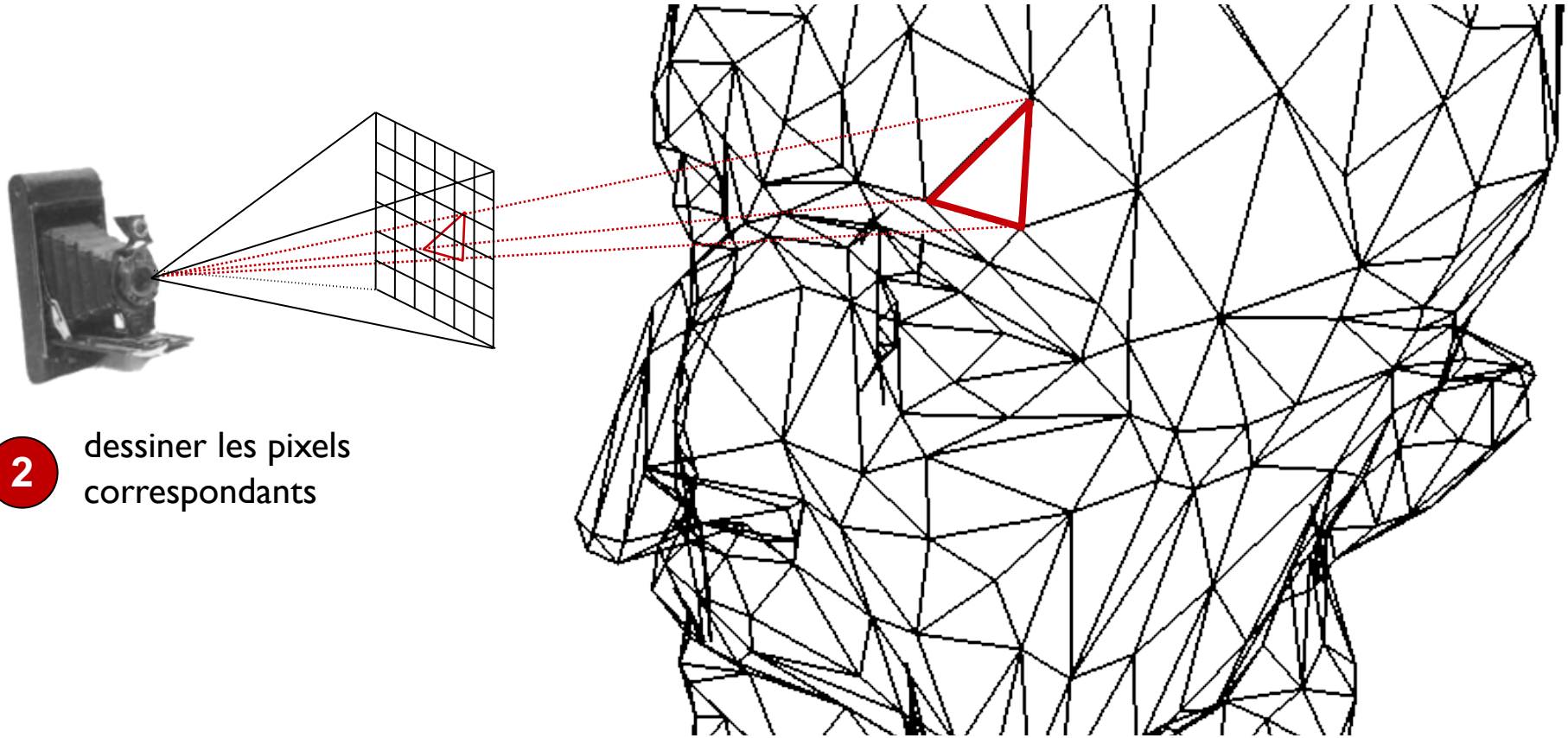
I ère approche : **lancer de rayons**



Visualiser une scène 3D

2ème approche : **rastérisation**

1. pour chaque triangle, projeter les 3 sommets sur le plan image ⟹ triangle 2D



2. dessiner les pixels correspondants

OpenGL (consortium Khronos)

Bibliothèque graphique 2D/3D

- API entre le GPU et le programme utilisateur
- rendu d'images composées de primitives
  - géométriques : points, lignes, polygones...
  - images : bitmap, textures…

Bas niveau (relativement)

- machine à états, contrôlée par des commandes
- sait uniquement convertir un triangle 2D en pixels !
  $\Longrightarrow$ accéléré par le matériel graphique

Portable (Linux, Windows, mac/iOS, Android, Web Browser)

langage C + interfaces pour tous les autres langages
(Java, Python, C#, OCaml, JavaScript, etc.)

Alternatives

DirectX (Microsoft) : seulement Windows

Metal (Apple) : seulement macOS / iOS

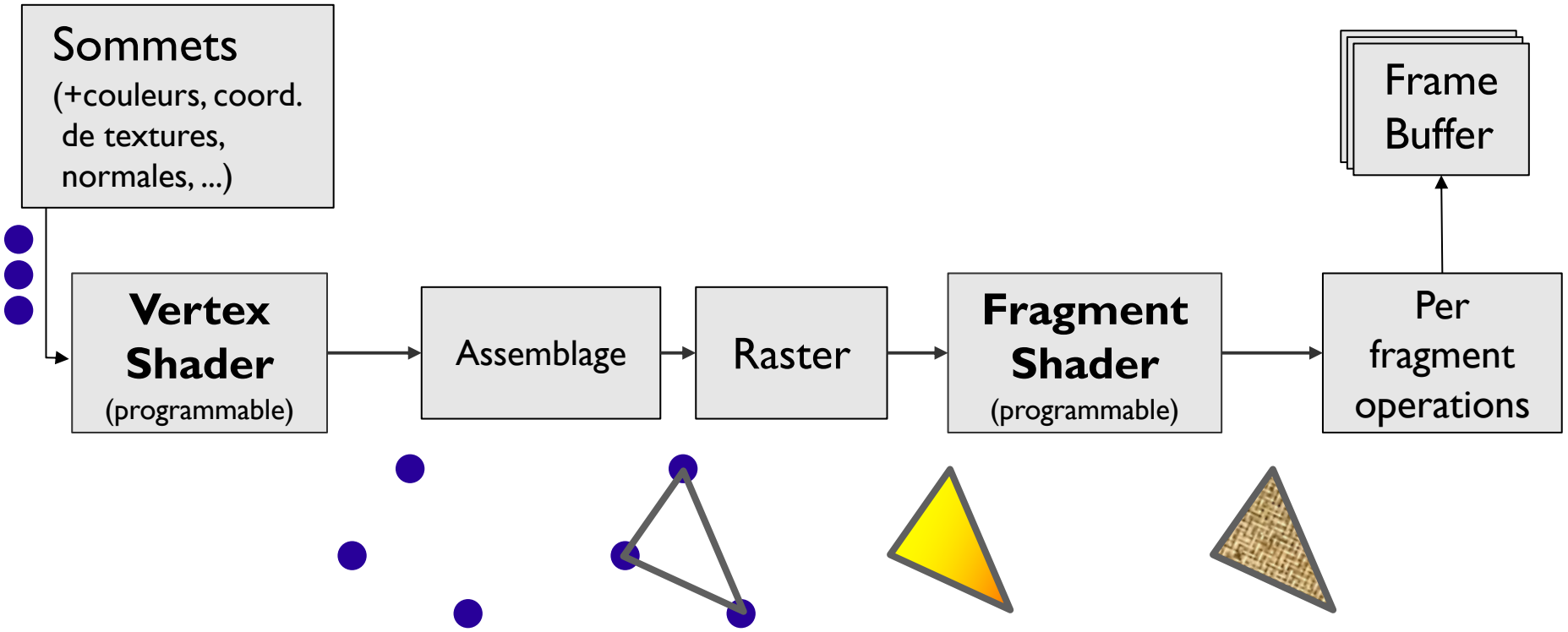
Vulkan (Khronos) : **multiplateformes** (langage C)

- couche fine au dessus du driver GPU
- encore plus **bas niveau** (gestion manuelle de la mémoire, de la synchronisation, etc.)
- meilleurs performances (si bien utilisé)

Programmation GPU générique (GPGPU)

OpenCL (Khronos), Cuda (Nvidia)

Pipeline graphique GPU



Langages de programmation

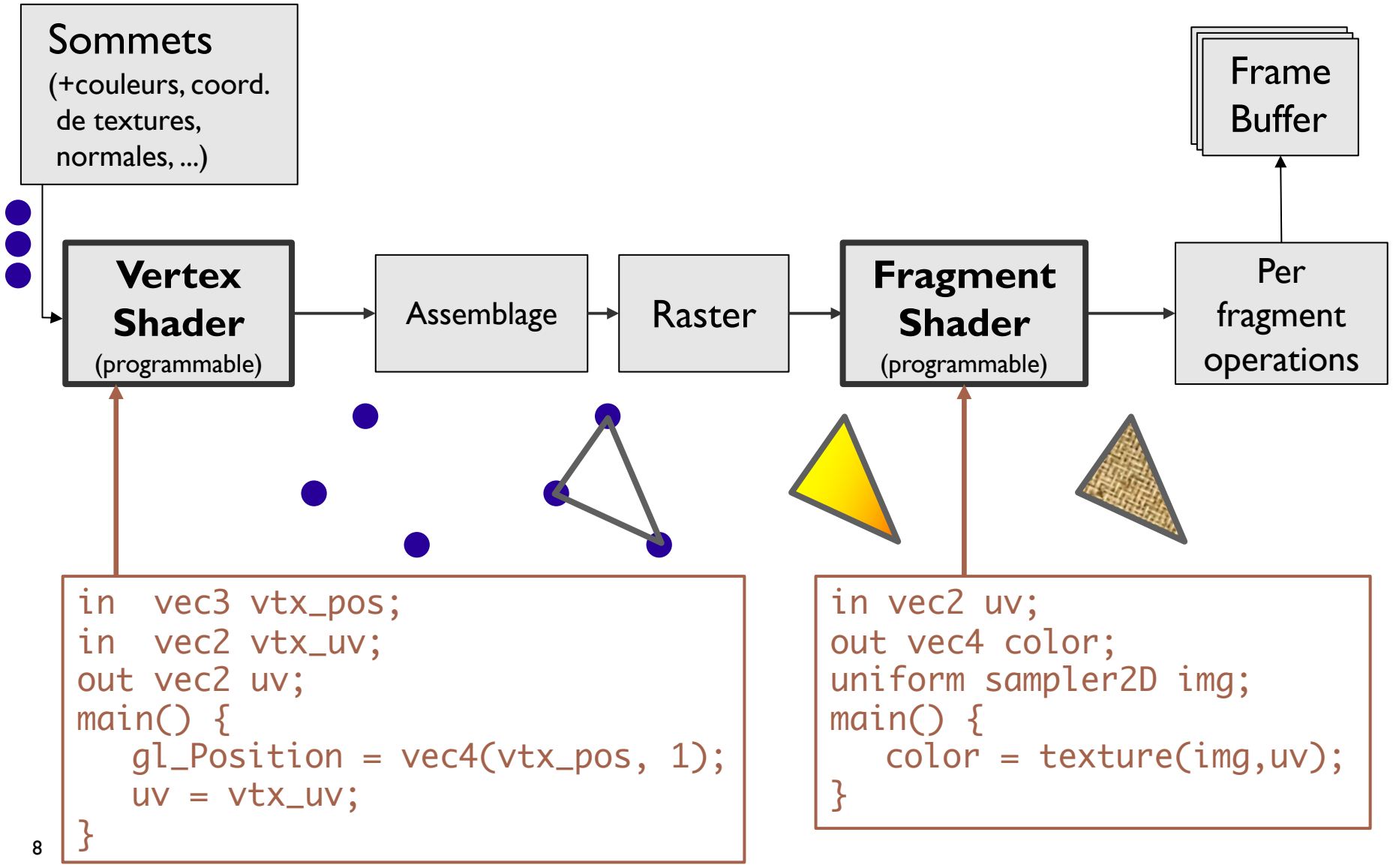
Shader = (petit) programme exécuté sur le GPU

Programmable via des langages de haut niveau
(proches du C/C++)

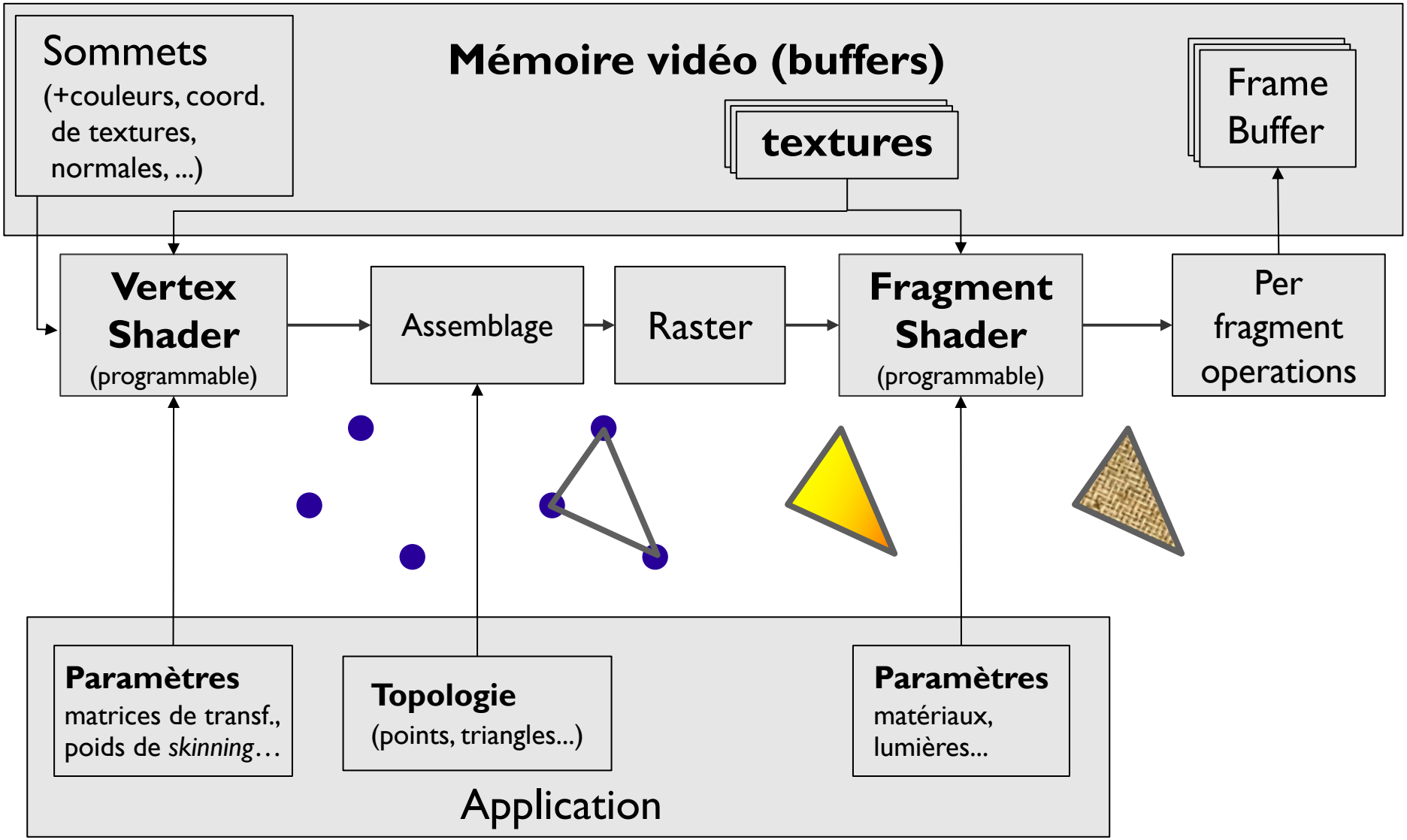
- **GLSL** (OpenGL Shading Language)
  - compilateur intégré dans le driver OpenGL (>=2.0)
  - génération et compilation de code à la volée (shader = char*)
  - standard ouvert
- **HLSL** (Microsoft) – seulement pour DirectX

Compilable en un binaire intermédiaire **SPIR-V**
(compatible avec Vulkan, OpenGL 4.6 et OpenCL)

Pipeline graphique GPU



Pipeline graphique GPU



Opérations par fragment