DIGINEXT
Advanced Computing Systems

Simulation & Virtual Reality

Soutenance de stage

4 septembre 2020

Test sur la possibilité d'intégration d'une bibliothèque permettant une abstraction aux diverses APIs graphiques sur un moteur existant

Présenté par Gauthier Bouyjou
M2 IGAI (Informatique Graphique et Analyse d'Images)
Université Paul Sabatier, Toulouse

Tuteur de stage : Charly Mourglia
Superviseur académique : David Vanderhaeghe

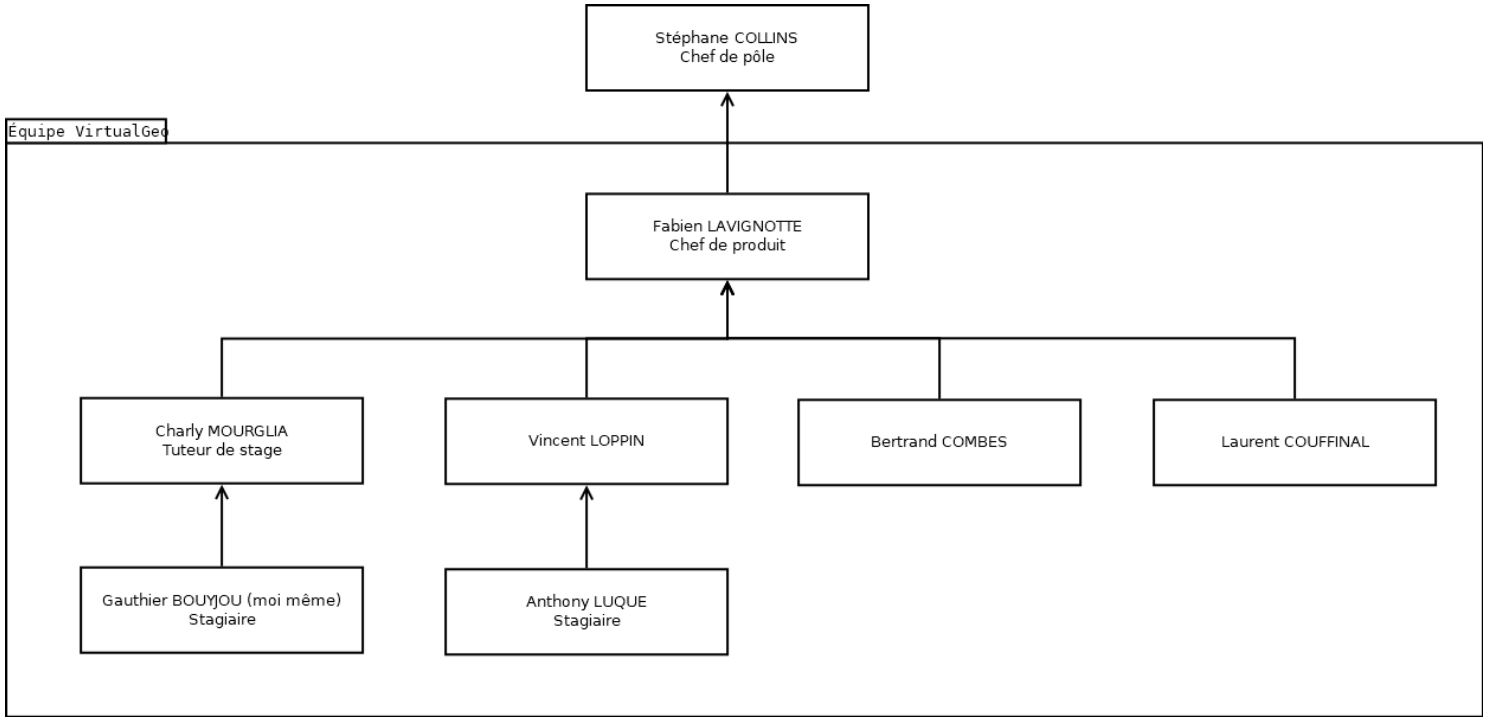
Présentation de l'entreprise

- Entreprise créee en 1996
- Filiale du groupe CS depuis 2004
- Rachetée par CS en mai 2020
- **3 principaux métiers :**
  - Défense
  - Transport
  - Navigation
- **Directeur général (mars 2020) :**

  Thomas FOURQUET
- **Nombre de collaborateurs :** 270
- **Chiffre d'affaire en 2017 :** 24.2 M€
- **3 sites en France :**
  - Siège social : Aix-en-Provence
  - Site du Plessis Robinson (Paris)
  - Site de Toulouse (lieu du stage)
- **Secteur d'activités :**
  - Entraînement et formation (Cerbère)
  - Simulation et réalité virtuelle (SIMENV)
  - Gestion de crise et maintien de la sécurité (Crimson)
  - Information voyageur
  - Liaisons de données tactiques
  - Systèmes de navigation
  - Surveillance maritime (Radar HF).

Contexte

- **Équipe VirtualGeo**



- **Tuteur/maître de stage :**
  - Charly Mourglia (Ingénieur en développement 3D C++, Responsable projet équipe SIMENV)
- Vincent Loppin (Responsable VirtualGeo)
- Bertrand Combes (Responsable Vertigo)

État des lieux

- Vertigo a plus de 10 ans (OpenGL 1)
- Brique de base pour plusieurs projets
- Sur conception de la hiérarchie de classe (orienté objet)
- Couche d'abstraction très coûteuse (défaut de cache)
- Vertigo est basé sur OpenGL
  - Impossible d'utiliser Direct3D (machines Windows)
  - Lunettes de réalité augmentée (HoloLens)
    - Solution : Angle (Google)
      - Coûteuse
      - Rend les applications complexes à déboguer
  - L'application de simulation militaire DirectCGF (écrite en C#)
    - Utilise Angle
      - Souffre donc des mêmes problèmes

Objectif de l’équipe

- **Objectif de l’équipe VirtualGeo :**
  - Meilleurs performances
  - Code plus facile à maintenir
  - Support des ‘’vieilles’’ plateformes clientes (Direct3D 9)
  - Support des ‘’nouvelles’’ APIs graphiques
  - Une meilleure utilisation du GPU / CPU

- **L’équipe VirtualGeo propose de :**
  - Remplacer la couche basse actuelle par une abstraction aux diverses APIs graphiques.
    - Permettre d’étendre l’application sur plusieurs autres plateformes
      - Metal → produits Apple (téléphone IOS, Mac OSX)
    - Choisir l’API graphique la plus performante sur le système client

Abstraction aux diverses APIs graphiques

- **Bgfx**
  - Projet open-source
  - Créé et développé par Mr Бранимир Караџић (surnommé Bkaradzic, actuellement développeur de jeu basé à Los Angeles)
  - 175 contributeurs
  - Sous licence BSD
  - **Compatible (multiplateforme) :**
    - Windows (XP, Vista, 7, 8, 10)
    - UWP (Universal Windows, Xbox One)
    - asm.js/Emscripten (1.25.0)
    - Linux
    - Android (14+, ARM, x86, MIPS)
    - FreeBSD
    - iOS (iPhone, iPad, AppleTV)
    - MIPS Creator CI20
    - OSX (10.12+)
    - PlayStation 4
    - RaspberryPi
- **APIs graphiques gérées :**
  - OpenGL (2.1, 3.1+)
  - OpenGL ES (2, 3.1)
  - WebGL (1.0, 2.0)
  - Direct3D (9, 11, 12)
  - Vulkan
  - Metal
  - GNM
- Choix de l'API de rendu à l'exécution
- Utilisé dans Minecraft

https://github.com/bkaradzic/bgfx

Bgfx

- Surcouche aux APIs graphiques
- Documentation en ligne non suffisante
- Simple d'utilisation
- **Fourni avec des exemples de code :**
  - Instancing
  - Bump
  - Hdr
  - Mesh LOD
  - Stencil reflections and shadows
  - Shadow volume
  - Shadow maps
  - IBL
  - N-body
  - Rsm
  - Assao
  - Tessellation
- Basé sur un système de commande

```
// render loop
while (1) {
  // setup viewport
  bgfx::setViewRect(view.id, 0, 0, uint16_t(width), uint16_t(height));
  bgfx::setViewClear(view.id, BGFX_CLEAR_COLOR | BGFX_CLEAR_DEPTH, 0X5555
55FF);

  // setup camera
  bgfx::setViewTransform(view.id, viewMtx, projMtx);

  // configure pipeline
  const uint64_t state = 0
    | BGFX_STATE_WRITE_RGB
    | BGFX_STATE_WRITE_A
    | BGFX_STATE_WRITE_Z
    | BGFX_STATE_DEPTH_TEST_LESS
    | BGFX_STATE_CULL_CCW
    | BGFX_STATE_MSAA;

  for (const auto& object : objects) {
    // setup object
    bgfx::setTransform(objet.mtx);
    bgfx::setState(state);

    bgfx::setIndexBuffer(object.ibh);
    bgfx::setVertexBuffer(0, object.vbh);

    const Material& material = object.material;
    bgfx::setUniform(uMaterialUH, material.data, num_vec4_material);

    // draw object
    bgfx::submit(view.id, program, 0.0, BGFX_DISCARD_STATE);
  }

  // Advance to next frame. Process submitted rendering primitives.
  bgfx::frame();
```

Problématique

- Performances avec bgfx ?
- **Évaluer les point durs d'une intégration dans Vertigo :**
  - Interface couche haute Vertigo
  - Intégration avec Qt
  - Coût d'une conversion d'un shader GLSL de Vertigo vers un shader bgfx ?
  - Fonctionnalité non disponible avec bgfx :
    - L'utilisation d'un geometry shader dans Vertigo ?

Apprentissage de bgfx

- Découverte de la bibliothèque (exemples)
- Prototypage d'un moteur de rendu en C++
- Chargement de scène obj
- Chargement des textures
- Rendu de type Blinn-Phong (une seule lumière directionnelle)

Test de performance

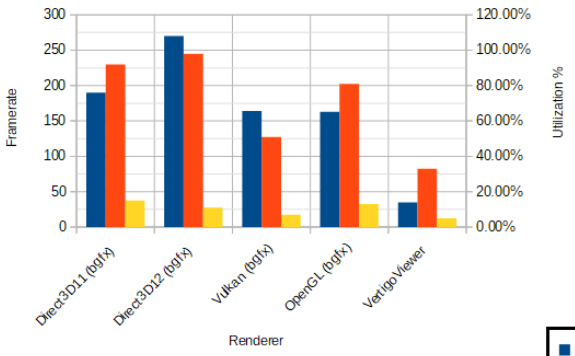
- **Comparaisons des performances relevées sur les 2 scènes**

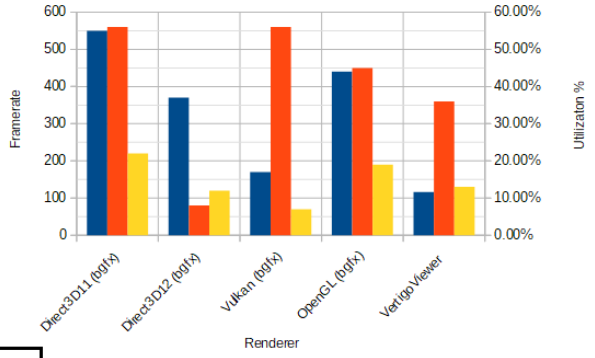
Measurements comparison

Intel® UHD Graphics 630

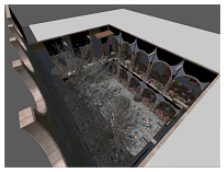
NVIDIA GeForce RTX 2060

Cougar (property of Diginext©)
400K triangles
228 textures

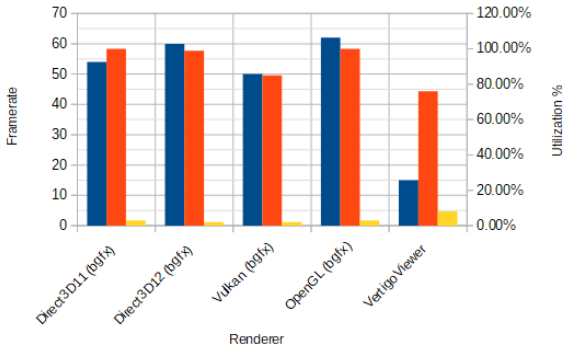


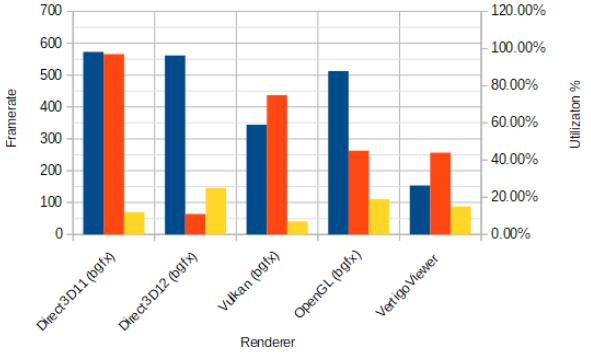


fps
gpu utilization %
cpu utilization %



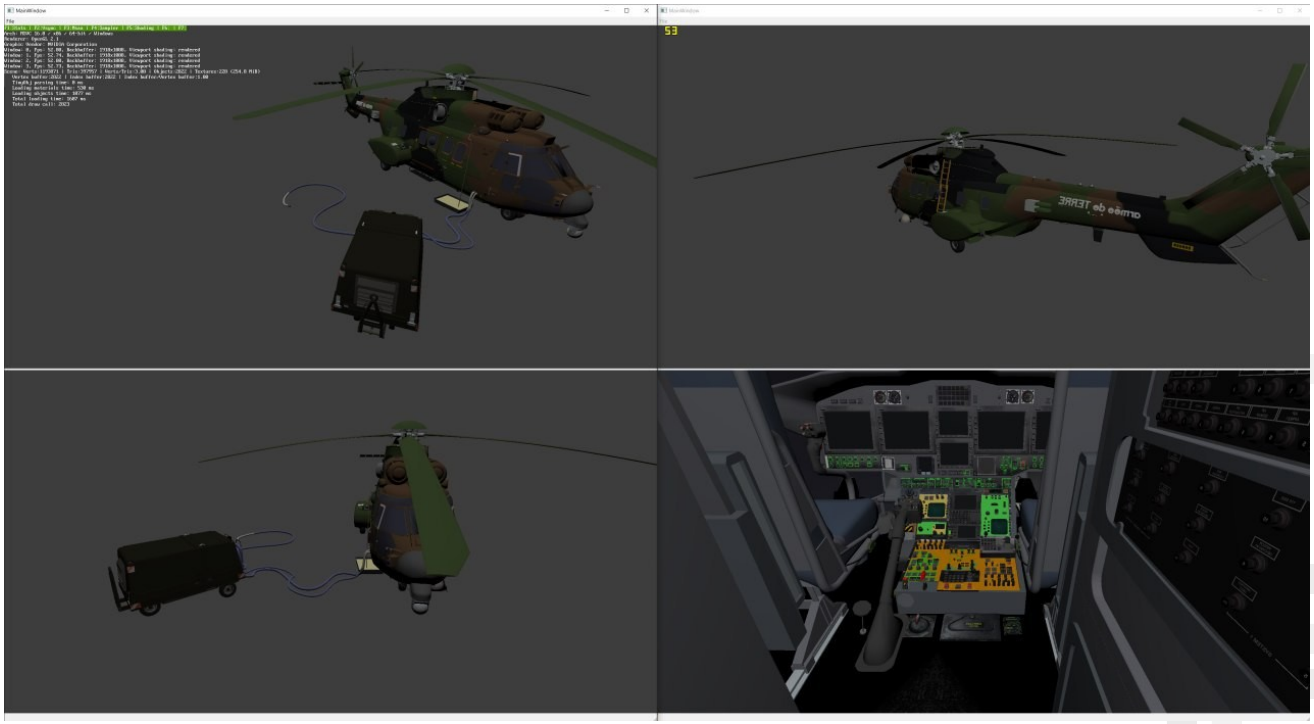
Morgan McGuire© San Miguel
10M triangles
219 textures





Multi-fenêtrage

- **Objectif :**
  - Tester si un multi-fenêtrage est possible avec bgfx
  - Utilisation d'une QWidget
- **Résultats :**
  - 4 vues de la scène Cougar dispersées sur 2 fenêtres sur un écran 4K
  - Sur carte graphique Nvidia GTX 1050

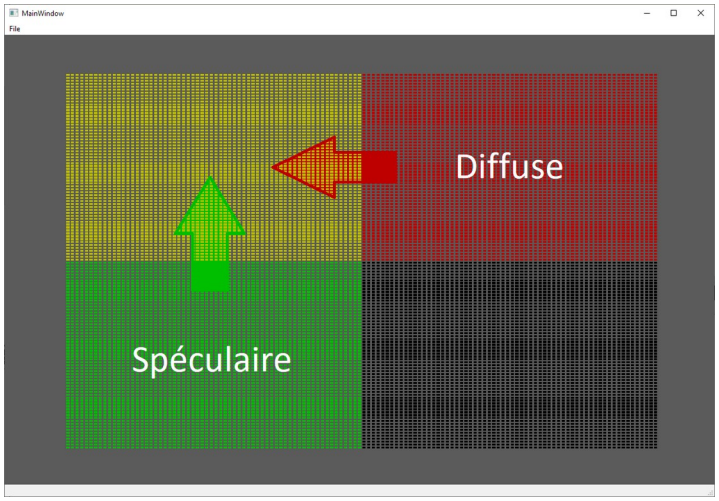


Multi pass

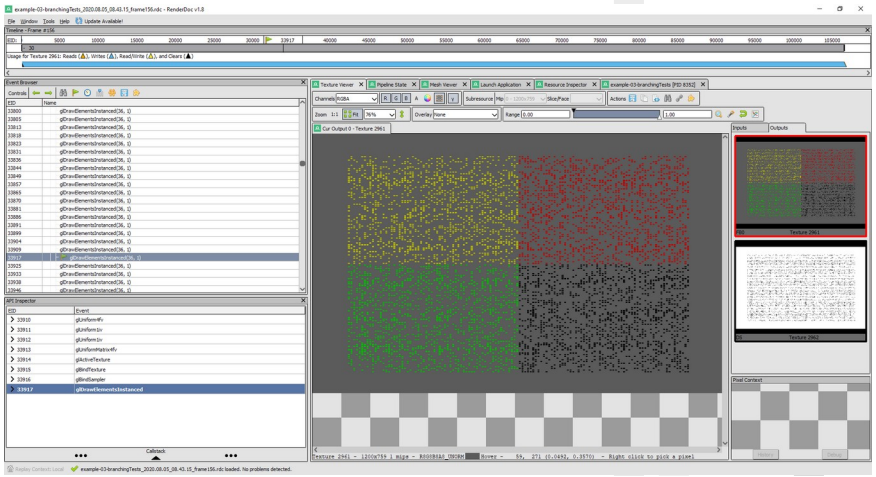
- **Objectif :**
  - Se familiariser avec les framebuffers de bgfx
- Difficultés : tableaux de textures (`SamplerArray`) ainsi que les tableaux de cartes de profondeur (`ShadowSamplerArray`) spécifiques à bgfx
- Rendu de San Miguel avec lumière directionnelle

Test de branching

- **Description des tests :**
  - Uniform booléen
  - Texture one / zero
  - Uber shader (Vertigo)

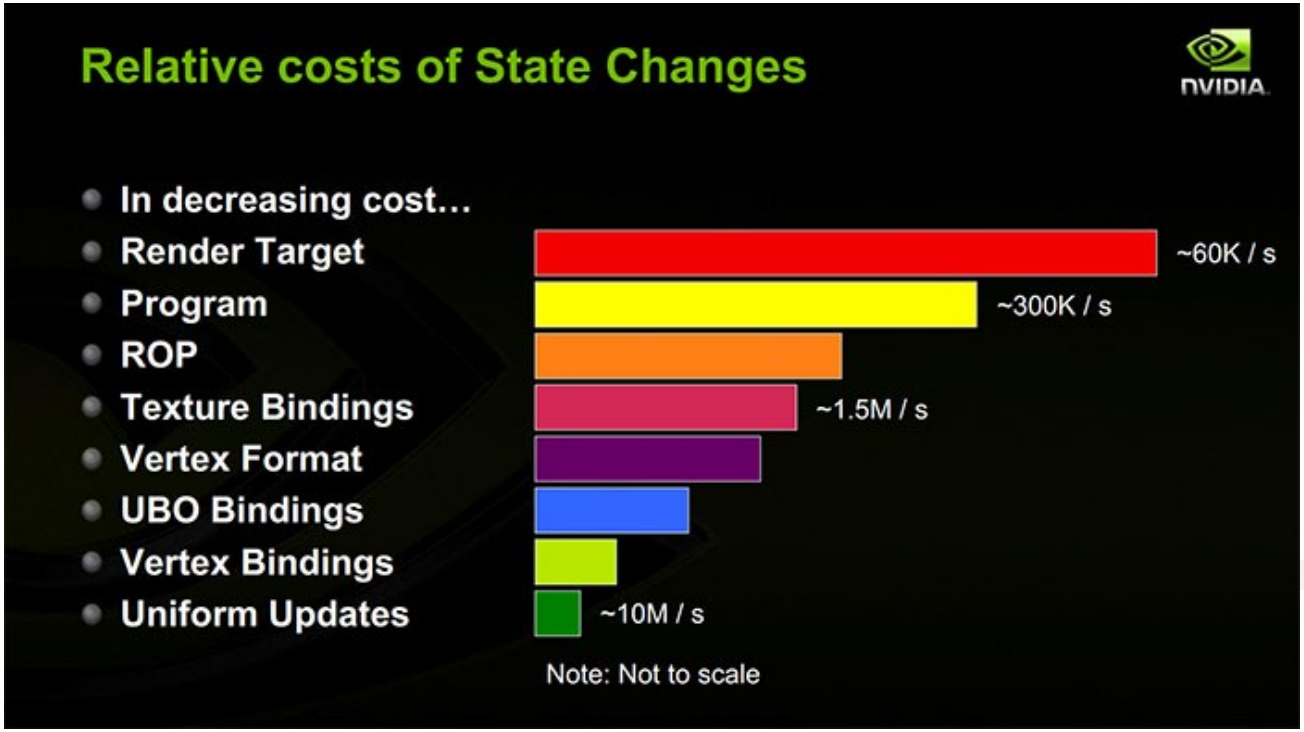


- Analyser si du tri est fait avec RenderDoc



Test de branching

- Slide présenté par McDonald's lors du Steam Dev Days GDC 2014 (Approaching Zero Driver Overhead in OpenGL)

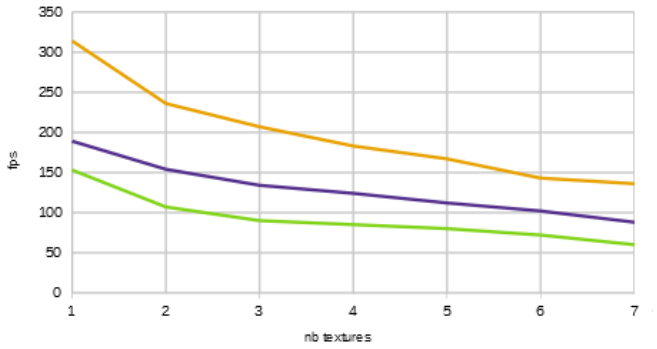


Test de branching

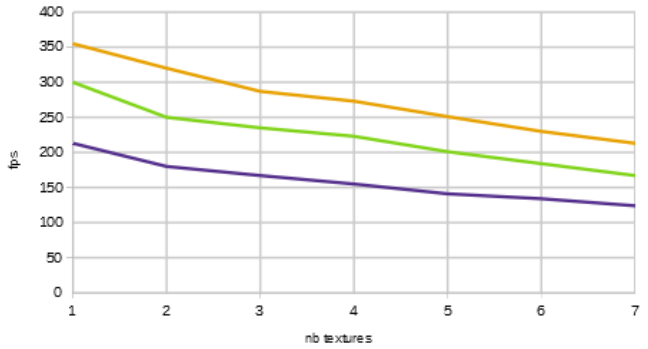
- Comparaison des mesures pratiques effectuées pour les cas 1, 2 et 3

Branching comparison tested with 16384 cubes under OpenGL

Unsorted cubes



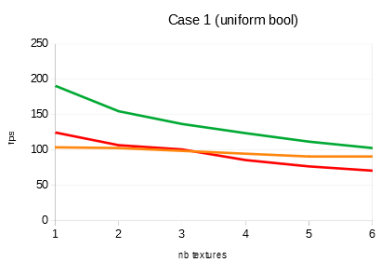
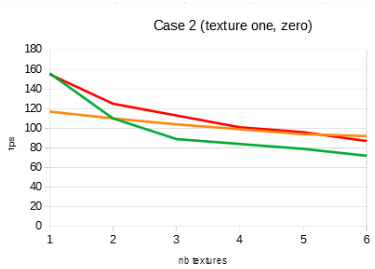
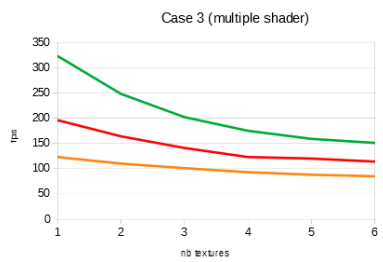
Sorted cubes



Measurements (practical)

NVIDIA GeForce RTX 2060

Case 1 (uniform bool)
Case 2 (texture one, zero)
Case 3 (multiple shader)

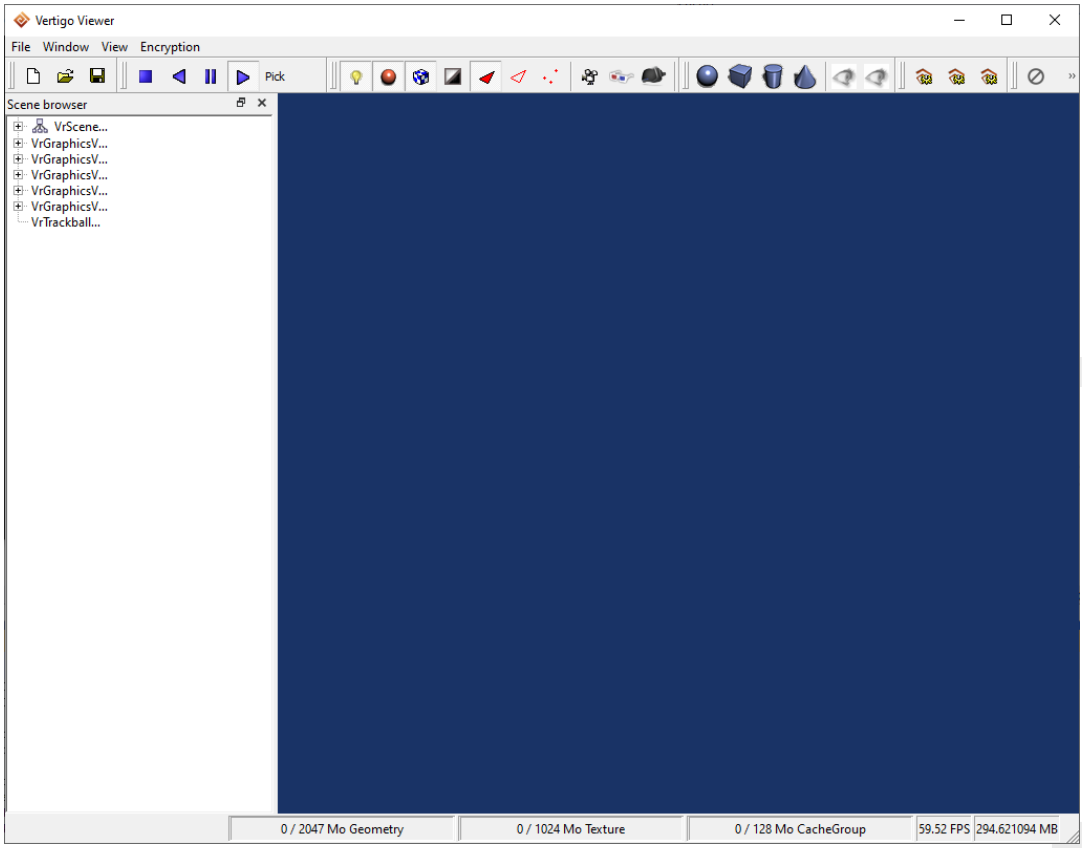
	Case 1 (uniform bool)	Case 2 (texture one, zero)	Case 3 (multiple shader)
APIs comparison unordered cubes Direct3D11 Direct3D12 OpenGL			

Intégration Vertigo

- **Approche technique :**
  - Garder couche ‘‘haute’’
  - Remplacement `VrGraphicsDevice`
- **Objectifs :**
  - Clear color
  - Chargement de modèle
  - Recyclage des shaders GLSL
  - Utilisation automatique des uber shaders

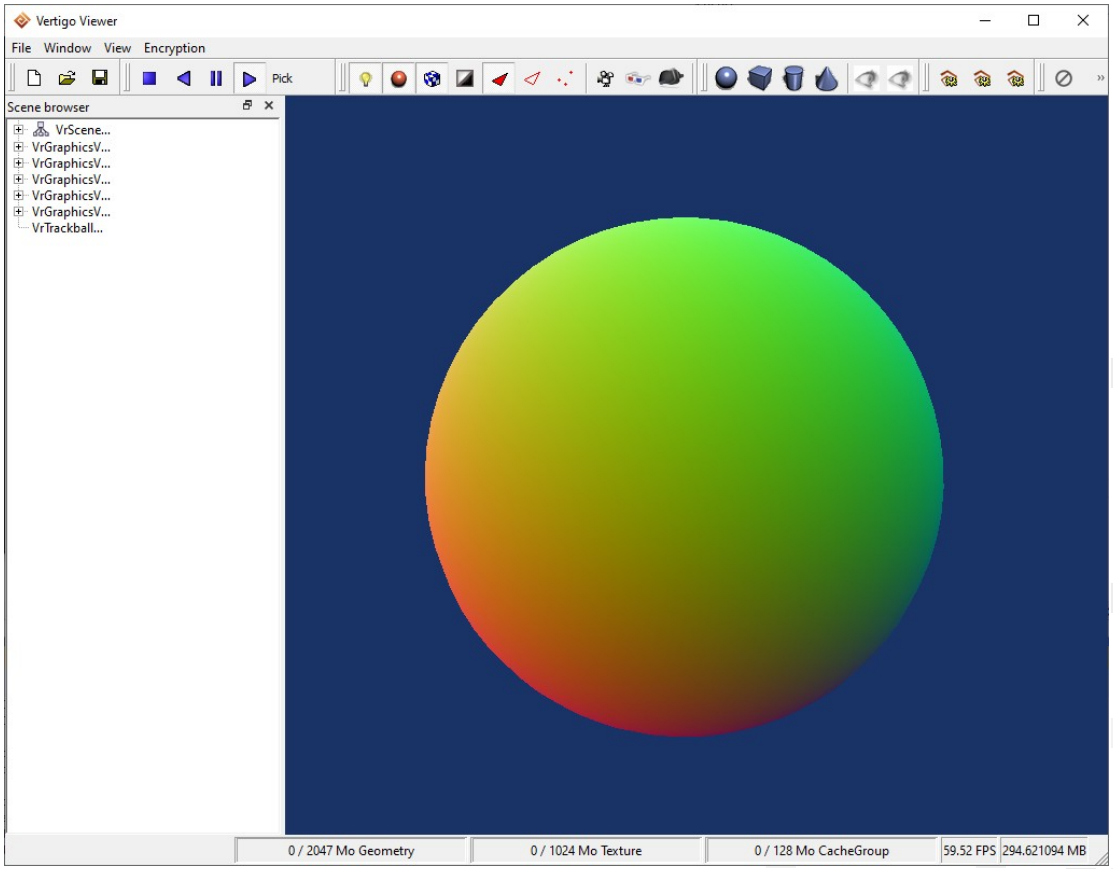
Intégration Vertigo

- **Clear color**
  - Suppression du code mort OpenGL
  - Utilisation de la `QOpenGLWidget` avec bgfx (`VrQOpenGLWindow`)
  - Difficulté : Initialisation de bgfx avec le backbuffer de Qt



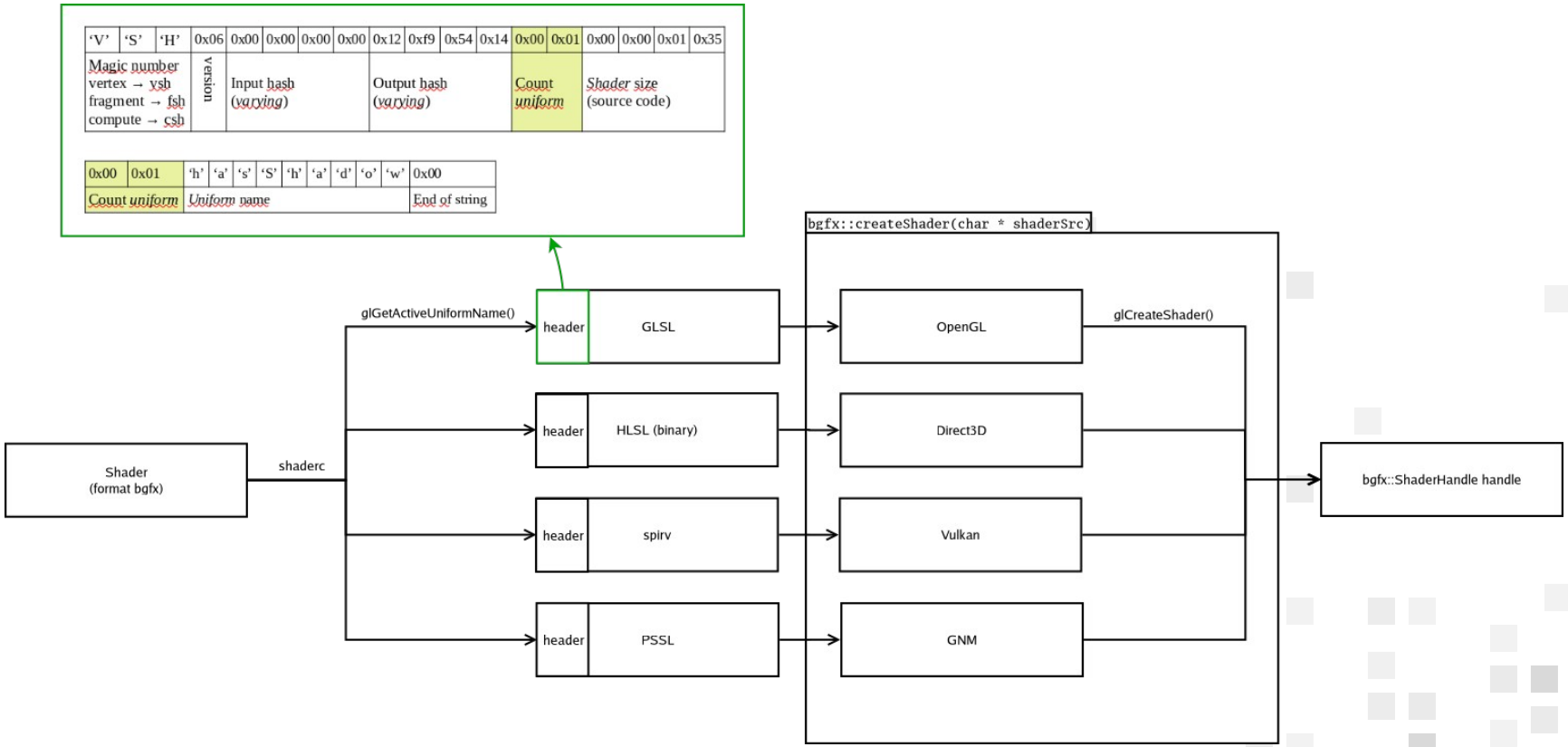
Intégration Vertigo

- **Chargement de modèle**
  - Buffers (`VrVertexBuffer` et `VrIndexBuffer`) stockés dans `VrGraphicsObject`
  - Utilisation d'un shader bgfx précompilé



Intégration Vertigo

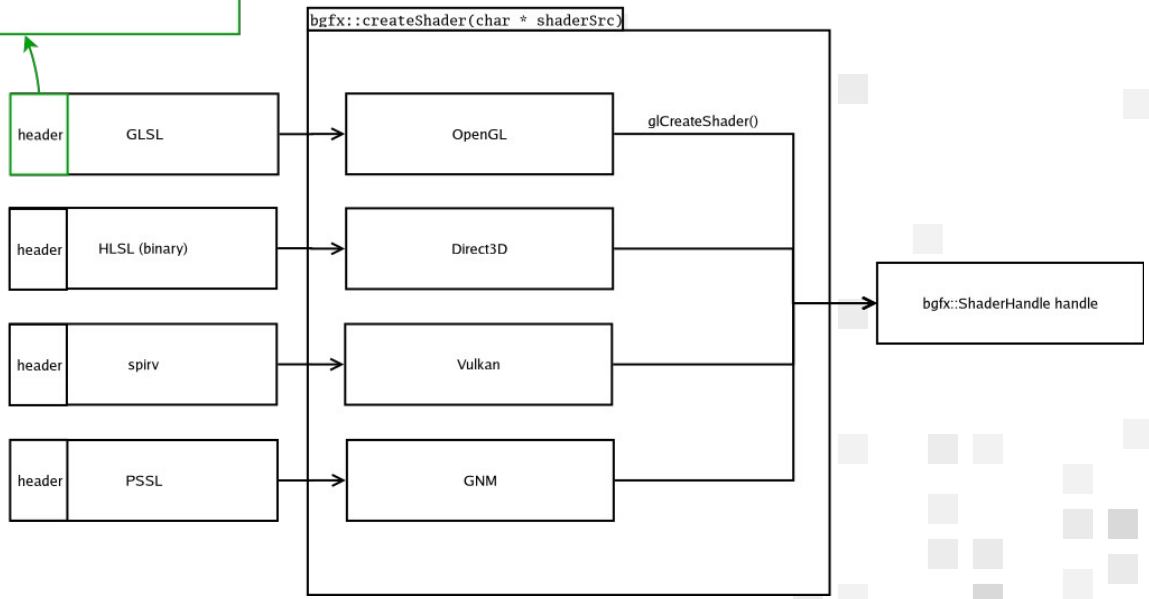
- **Recyclage des shaders GLSL**
  - Donner un shader Vertigo GLSL directement à bgfx sans passer par shaderc
  - **Shaderc :**
    - Compilateur de shader bgfx (langage proche du GLSL)



Intégration Vertigo

- **Recyclage des shaders GLSL**
  - **Sans shaderc**
    - On perd l'introspection (plus de liste des uniforms utilisés)
  - À terme on privilégiera l'utilisation des shader bgfx

'V'	'S'	'H'	0x06	0x00	0x00	0x00	0x00	0x12	0xf9	0x54	0x14	0x00	0x00	0x00	0x00	0x01	0x35
Magic number vertex → vsh fragment → fsh compute → csh			version	Input hash (*varying*)				Output hash (*varying*)				Count *uniform*		*Shader* size (source code)			

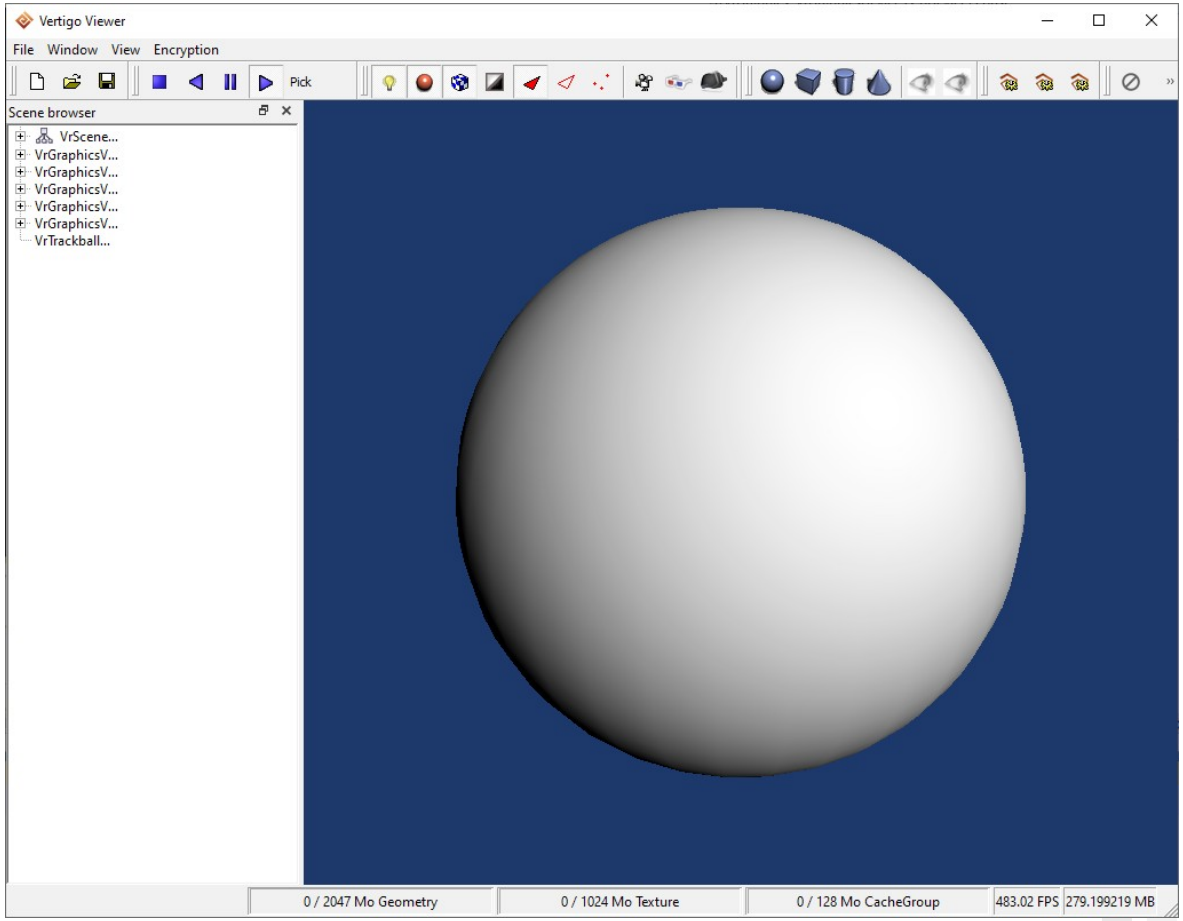


Intégration Vertigo

- **Recyclage des shaders GLSL**
  - **Attributs**
    - `attribute vec4 aPosition` -> `a_position` (bgfx).
    - `#define aPosition a_position`
  - **Varying**
    - Même nommage
  - **Syntaxe**
    - Aucune erreur de syntaxe
    - Pour les multiplications de matrice avec un vecteur
    - Déclaration de texture
      - uniform sampler2D s_diffuse → Sampler2D(s_diffuse, 0) (bgfx)

Intégration Vertigo

- **Recyclage des shaders GLSL**
  - Portage d'un *shader* Vertigo (VrLowPhong.glsl)



Intégration Vertigo

- **Utilisation automatique des uber shaders**

  La première fois qu'on rencontre un meta shader

  Extraction des variables préprocesseurs moteurs (DEF_NUM_LIGHTS, DEF_HAS_EMISSIVE, DEF_HAS_DIFFUSE, DEF_HAS_SHADOW, etc)

  Attacher la liste de ces variables au meta shader (éviter un reparsing)

  On va ensuite récupérer les informations moteur (num_lights, has_emissive, has_diffuse, has_shadow, etc)

  Calcul de la clé de meta program à partir des valeurs précédemment récupérées (bit mask)

	num_lights	has_emissive	has_diffuse	has_shadow	
valeur	2	false	true	true	
Domaine de définition	0-15	0-1	0-1	0-1	
clé	0010	0	1	1	19

  Si le shader n'a pas été compilé avec la clé courante

  Alors

  On compile le shader

  On récupère la liste des uniforms utilisés (introspection)

  On stocke le tout dans une map<clé, {shader compilé, uniforms utilisés}>

  Sinon

  On récupère le shader précédemment compilé de même clé (unicité)

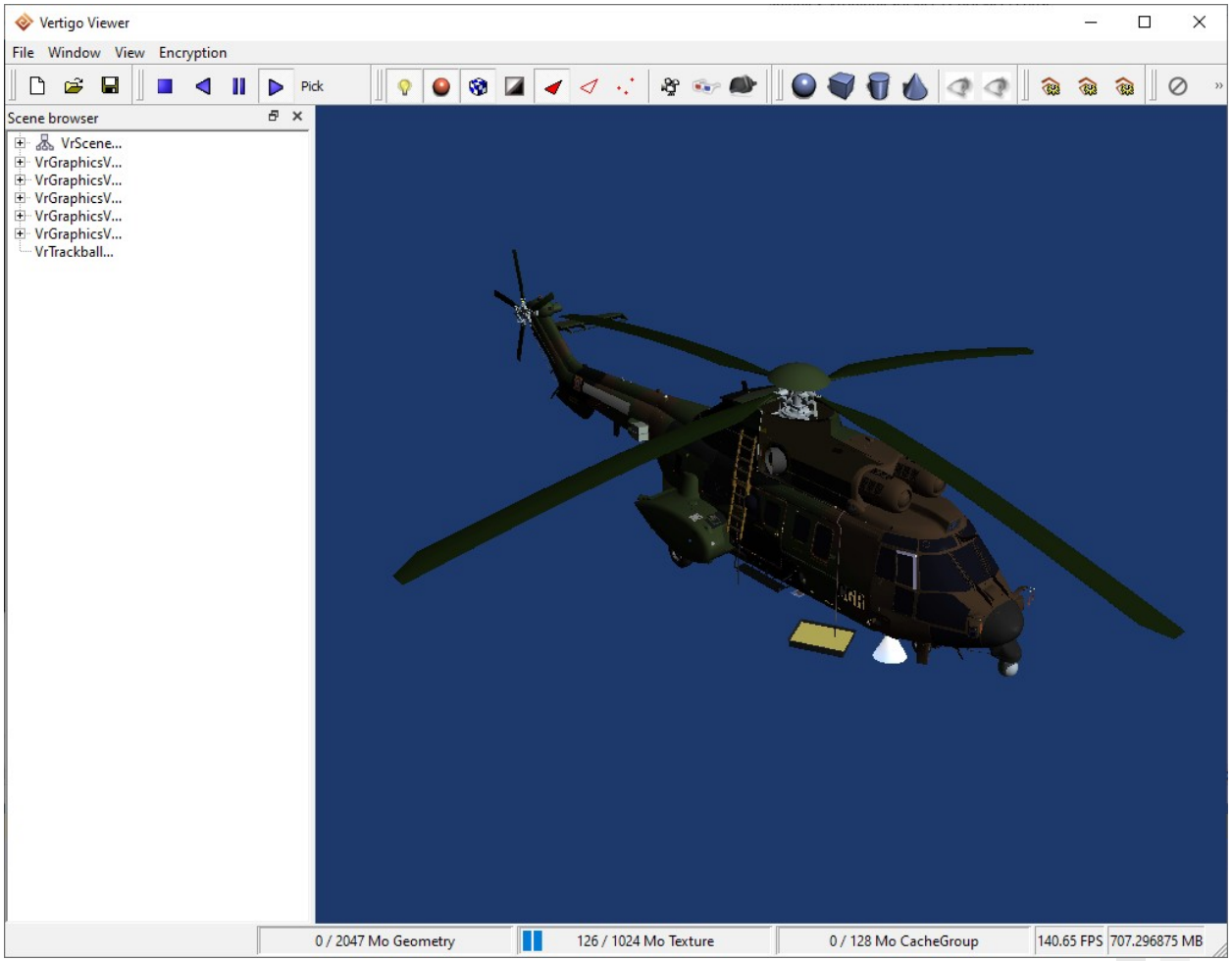
  On bind le shader compilé

  Pour tout les uniforms utilisés du shader compilé

  On cherche sa valeur dans l'état du moteur (u_normalMtx, u_diffuseTexture, u_pcfCoeff, etc)

Intégration Vertigo

- **Utilisation automatique des uber shaders**



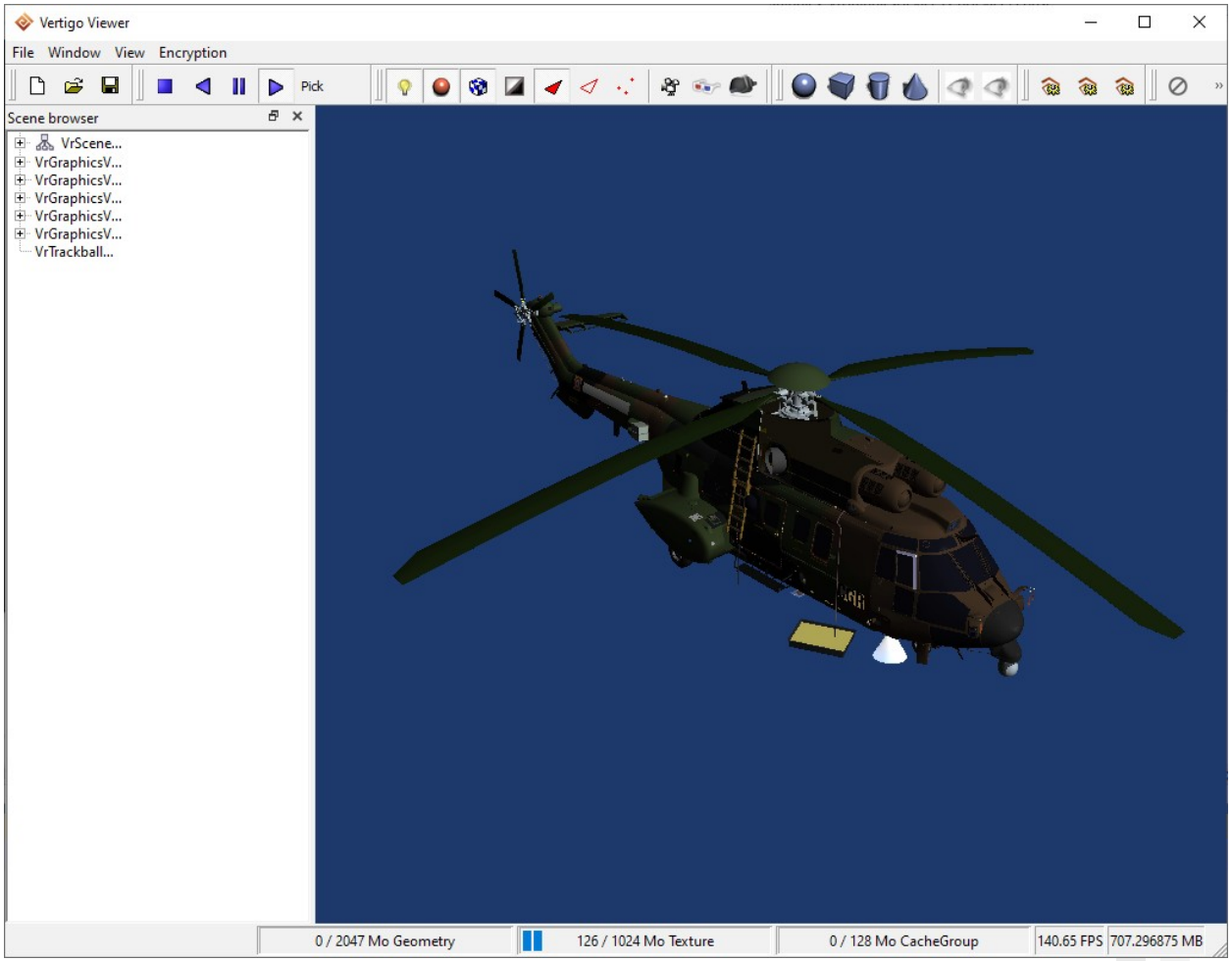
Conclusion

- **Une intégration facile avec bgfx :** pas de gros problème particulier
- Bgfx casse l'architecture déjà existante
- Peu de performances acquises (+20%), stabilités des performances
- Travaux à faire plus haut dans la hiérarchie du moteur
  - Utilisation de multiples classes objets et polymorphisme ralentissent les accès mémoires
- Gains potentiels atteignables, voir dépassables
  - Intelligence de rendu (frustum culling)
  - Parallélisation

Bilan personnel

- **Point de vue personnel :** stage enrichissant, Covid-19 (travail à distance)
- J'ai été accompagné par des personnes expérimentées dans le domaine de l'informatique graphique
- Perfectionnement aux langages C++ et CMake
- Apprentissage de bgfx
- **Utilisation d'outil de profilage cpu :** Visual Studio Profiler
- **Utilisation d'outil de débogage et profilage gpu :** Nsight (NVIDIA)

Questions ?


Vertigo Viewer
File Window View Encryption
Pick
Scene browser
VrScene...
VrGraphicsV...
VrGraphicsV...
VrGraphicsV...
VrGraphicsV...
VrGraphicsV...
VrTrackball...
0 / 2047 Mo Geometry
126 / 1024 Mo Texture
0 / 128 Mo CacheGroup
140.65 FPS
707.296875 MB