# *Shimera*

*Make it shine. ✨*

-

# Plan d’actions

# Sommaire

# Contexte

Il nous est déjà arrivé de bloquer sur l'implémentation et la création de shaders sur divers projets. Nous avons donc cherché une idée pour rendre l'implémentation des shaders plus simple et rapide, sans avoir à pratiquer de langages interprétables par la carte graphique.

De nos jours, pour intégrer des graphismes avancés, des effets visuels ou tout autre élément visuel dans une application, il faut utiliser des programmes spécifiques qu'on appelle des shaders. Le problème, c'est que leur écriture repose sur des compétences pointues en mathématiques, en physique et en programmation critique/bas-niveau, ce qui les rend assez complexes à maîtriser et à apprendre.

Notre solution, Shimera, est une librairie de shaders open source. Le but étant de faire en sorte que ceux-ci soient le plus modulables possibles et que les utilisateurs n'aient pas à faire de programmation GPU. Nous avons aussi pour objectif de rendre la librairie multi langages et multi API.

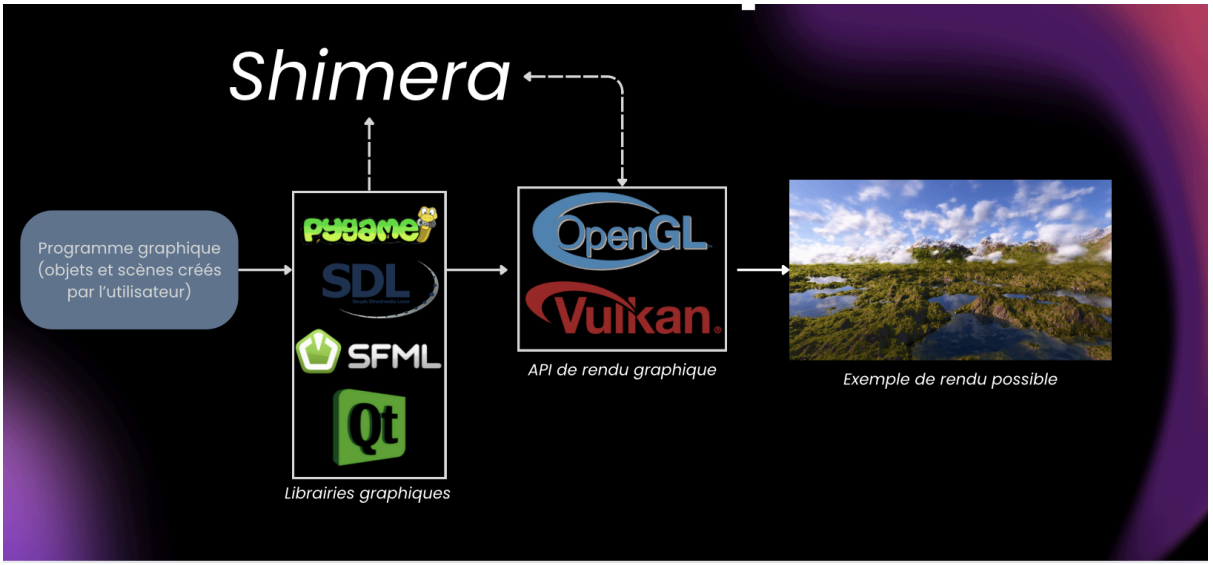
Le projet Shimera est considéré dans la branche technique du *Epitech Innovative Project*.

# Spécifications techniques

## Fonctionnement

L'objectif de notre bibliothèque est de simplifier l'intégration des shaders pour les développeurs. Shimera s'adapte à de nombreuses bibliothèques en interagissant directement avec le contexte graphique de l'utilisateur, offrant ainsi une intégration fluide et flexible.

Voici un schéma du fonctionnement :




Pour intégrer Shimera efficacement, l'utilisateur initialise la bibliothèque en parallèle de son code.

Il peut ensuite importer, créer et personnaliser les shaders qu'il souhaite appliquer à son espace de rendu.

Shimera intercepte alors le contenu affiché et applique les shaders en exploitant directement l'API graphique, garantissant une intégration fluide et performante.

**Shimera** - *Make it shine.* ✨

# Fonctionnalités

## Nombreuses bibliothèques graphiques

La bibliothèque Shimera permettra aux développeurs d'utiliser des shaders avec différentes bibliothèques graphiques telles que GLFW, SDL, Qt ou PyGame.

## Shaders modulables

Les shaders seront composés de plusieurs modules indépendants, permettant aux développeurs de former leurs propres shaders adaptés aux besoins de leurs projets.

Différents shaders seront proposés par Shimera :

- Particules GPU
- God Rays
- Occlusion Ambiante
- Matériaux
- Brouillard
- Flou
- Et plein d'autres...

## Liberté de développement

Shimera sera disponible dans plusieurs langages, permettant ainsi à l'utilisateur de posséder le plus de choix sans être trop contraint dans son langage (ex : PyGame / Python, Qt / C++, SDL / C, ....).

Shimera sera aussi proposé sur la plupart des systèmes d'exploitations : Windows, Linux, MacOS.

Une des fonctionnalités importantes de notre projet est de permettre à l'utilisateur de sourcer ses propres fichiers de shaders pour qu'il puisse utiliser notre moteur pour le rajouter dans son application graphique.

# Langages et Outils de développement

## Construction

Pour construire notre projet à travers les plateformes et les architectures, nous avons choisi l'outil CMake. Cet outil existe depuis longtemps et est donc consolidé autour d'une large communauté ainsi que de plusieurs outils internes (VCPKG, FetchContent, FindPackage, ...) très utiles au développement.

## Librairie

Pour développer notre librairie, nous souhaitons utiliser le langage C++. De par son ancienneté, il a pu prouver depuis bientôt 50 ans qu'il permet facilement de combiner le développement haut-niveau et la gestion mémoire bas-niveau. De nombreuses interfaces de programmation, graphiques comme logicielles, ont été écrites dans le langage C ce qui permet facilement de les utiliser en C++. C'est le cas de plusieurs interfaces que nous utiliserons telles que OpenGL, Vulkan, DirectX et Apple Metal pour la contrepartie graphique, et Python API pour les interfaces de langages.

## Shaders

Pour développer notre base de shaders pour fonctionner sur les interfaces graphiques choisies, nous devrons travailler sur les langages HLSL (Vulkan et DirectX), GLSL (OpenGL) et MSL (Apple Metal).

# Récits utilisateur

- En tant que développeur, je souhaite pouvoir ajouter des shaders à mon projet simplement.
- En tant que développeur, je souhaite pouvoir ajouter des shaders à mon projet sans connaissances dans un langage GPU.
- En tant que développeur, je souhaite pouvoir personnaliser au maximum les shaders.
- En tant que développeur, je souhaite ne pas être bloqué parce que Shimera n'est pas compatible avec mon projet graphique.

# Cycle de vie du projet

Notre projet va se découper en plusieurs étapes, ce qui nous permettra d'être plus efficace dans son développement.

Etape 1:

Dans un premier temps, nous devons faire un POC (Proof Of Concept) permettant de valider la faisabilité technique de notre projet. Pour ce faire, nous essaierons de prouver une cohabitation possible entre notre librairie de shaders et la librairie graphique dans la fenêtre générée par cette dernière.

Etape 2:

Par la suite, notre objectif sera de trouver un moyen d'implémenter des shaders dans des librairies graphiques (SFML, SDL, Cocos2d....) par l'intermédiaire des API graphiques (OpenGL, Vulkan). Nous commencerons par l'implémentation de shaders post-processing, puis nous ferons dans un second temps l'implémentation des shaders par entités notamment celles créées par la librairie graphique.

Etape 3:

Ensuite, étant donné que nous avons peu d'expérience dans les langages graphiques, nous ferons un maximum de shaders simples et basiques pour monter en compétences. Pour pouvoir au fur et à mesure passer sur la création de shaders plus complexes. Tout en gardant le maximum de paramètres modifiables par l'utilisateur.

Etape 4:

Pour finir, le but sera de rendre l'expérience utilisateur agréable et de rendre la librairie la plus accessible possible.

# Spécifications non techniques

## Sujets à explorer

### Obligatoires

*À terminer.*

### Optionnels

*À terminer.*