CÉGEP DE MATANE

FORUM INNOVATION
INGÉNIERIE | INFORMATIQUE |
ENTREPRENEURIAT | UQAR

VectorGL un moteur 3D en OpenGL

par Kylian Eury

Présentés au FI3E par le Cégep de Matane

FORUM INNOVATION
INGÉNIERIE | INFORMATIQUE |
ENTREPRENEURIAT | UQAR

VectorGL un moteur 3D en OpenGL

par Kylian Eury

Ce projet consiste à créer un moteur de rendu 3D itératif utilisant C++ et OpenGL pour abstraire les appels OpenGL.

Il comprend une interface utilisateur pour manipuler scène, lumières, et shaders, un système de caméra avancé, et un éclairage de type Phong. Des fonctionnalités telles que le chargement de textures et de modèles 3D via Assimp sont également intégrées, rendant ce moteur adaptable pour divers effets visuels en 3D.

Pourquoi

La création de rendus itératifs 3D (3D forward rendering) nécessite une compréhension approfondie des appels OpenGL de bas niveau et de la communication avec la carte graphique.

Il y a donc un important besoin dans l'industrie pour un moteur de rendu léger.

- **Ce n'est généralement pas à la portée des petits projets d'inclure ce genre de travail approfondi.**

- **De plus, les engins existants sont très lourds, nécessitent des machines très puissantes et sont difficiles à prendre en main.**

Illustration par l'intelligence artificielle

Objectifs

Moteur

- Construire un moteur de rendu 3D itératif (3D forward renderer) en utilisant C++ et OpenGL.
- Architecturer une librairie avec un bon niveau d'abstraction des appels OpenGL.

Interface utilisateur

- Implémenter une interface pour l'interaction dynamique avec les éléments du rendu : scène, lumières, skybox, et shader.
- Implémenter un système de caméra et d'éclairage dans un environnement 3D

Méthodologie

Méthodologie

Des fonctionnalitées testées dans une démo.

📦 **Abstraction OpenGL :**
Création de classes pour abstraire les appels OpenGL de bas niveau.

🧱 **Chargement de textures :**
Implémentation du chargement de textures, y compris cubemaps et skyboxes.

🧊 **Chargement de modèles :**
Utilisation d'Assimp pour le chargement de modèles 3D.

✨ **Gestion des shaders :**
Compilation et utilisation de multiples shaders pour divers effets.

💡 **Système d'éclairage :**
Support de multiples sources lumineuses avec un éclairage de Phong

Choix technologique

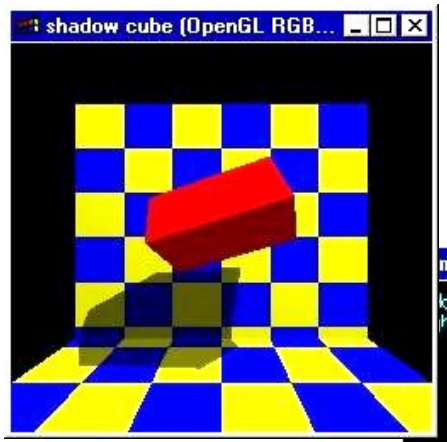
J'ai choisi OpenGL pour :

- Sa compatibilité multiplateforme, OpenGL est un standard reconnu depuis plus de 30 ans

- Sa flexibilité et l'accélération graphique qu'OpenGL offre

- La richesse de ses fonctionnalités tels que les shaders pour créer toutes sortes d'effets visuels

- Sa communauté développée



Choix technologique

J'ai choisi CMake pour :

- Création de builds multi-plateformes (génération de makefile, solutions Visual Studio, projets XCode etc.)
- Automatisation du processus de compilation
- Simplification du développement sur d'autres machines


```
cmake_minimum_required(VERSION 3.28)
project(VectorGL)

set(CMAKE_CXX_STANDARD 17)

find_package(OpenGL REQUIRED)

# Add library subfolders
add_subdirectory(libs/assimp EXCLUDE_FROM_ALL)
add_subdirectory(libs/glfw EXCLUDE_FROM_ALL)
add_subdirectory(libs/glm EXCLUDE_FROM_ALL)
```

Pourquoi VectorGL est un moteur et non une librairie ?

Moteur graphique

- Une interface graphique permet à l'utilisateur d'interagir avec la scène sans programmation
- Une rétroaction visuelle permet à l'utilisateur de visualiser le résultat de sa préparation
- Dans un moteur, le cycle d'affichage et de rafraichissement de la scène dans une boucle infinie est souvent pris en charge nativement

Librairie graphique

- Une librairie graphique s'utilise directement dans la programmation, sans outil d'aide à l'édition
- La librairie graphique n'a pas d'image écran pour visualiser, il faut intégrer le code dans un logiciel
- Dans une librairie, il n'y a pas de prise en charge de la fréquence d'affichage, sinon cela devient un engin ou moteur

Interface

Tout moteur a son dashboard !

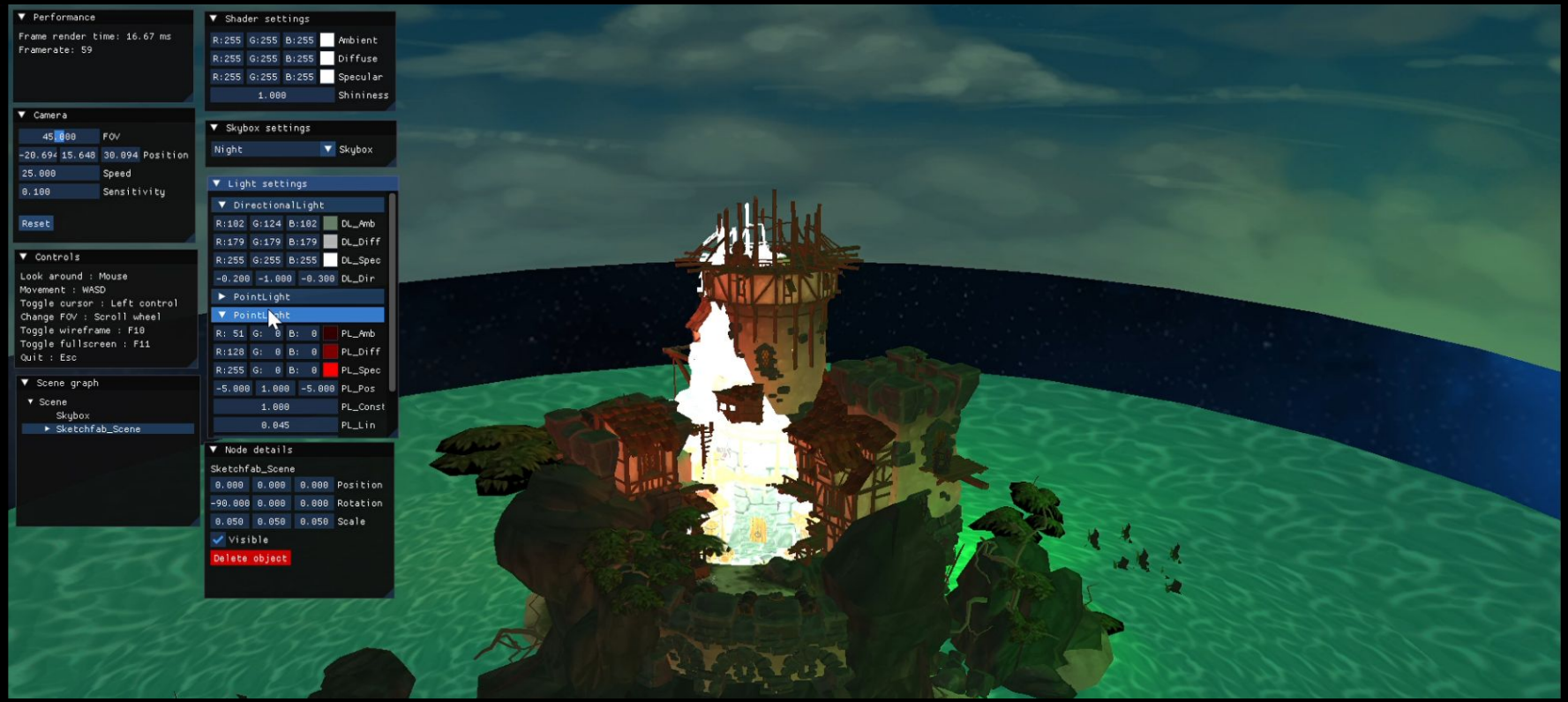
Voici les éléments de l'interface utilisateur

- **Performance** : affiche le temps de rendu
- **Camera** : permet de changer le point de vue
- **Controls** : précise les raccourcis et actions
- **Scene graph** : affiche la hiérarchie de la scène
- **Shader settings** : mets à jour le shader d'éclairage
- **Skybox settings** : sélectionne la boîte du ciel
- **Light settings** : ajuste les paramètres des lumières
  - Directional/Point/Spot lights
- **Node details** : permet d'ajuster les transformations et la visibilité d'un objet sur la scène

Résultats

Résultats - Exemple avec scène vide

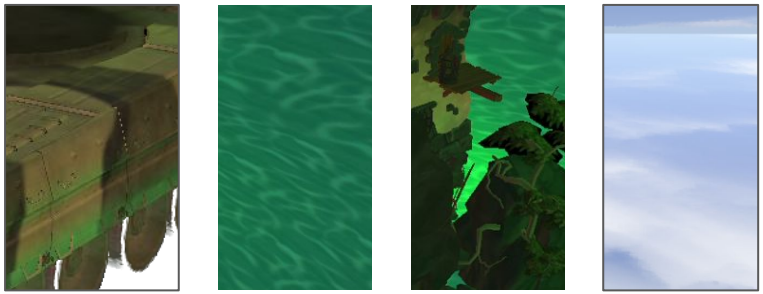
Résultats - Exemple avec château + lumières



Résultats - Exemple avec tank

Résultats - Conclusion

Le moteur de rendu développé a permis de réaliser un rendu 3D basique avec des fonctionnalités de chargement de textures et de modèles, de gestion des lumières.



Il a aussi permis des interactions tels que des ajustements à la scène via une interface utilisateur.

Bien qu'il ait été conçu comme un outil pédagogique plutôt que pour une application professionnelle, le projet a atteint plusieurs objectifs clés dans :

- l'abstraction des appels OpenGL,
- le chargement des modèles,
- et la manipulation des shaders.

On peut ainsi conclure que j'ai atteint mon objectif de comprendre les principes fondamentaux du rendu graphique en temps réel.

Futur

Les améliorations futures du projet sont de deux ordres :

=> Développer l’interaction

- Implémenter le chargement de plus de formats de modèles (par exemple .glb avec textures intégrées au fichier)
- Exportation et importation de scènes (sérialisation)
- Ajout de surbrillance sur les éléments sélectionnés et déplacement, rotation, mise à l’échelle directement en 3D (gizmos)

=> Améliorer le rendu visuel

- Implémenter un shader de PBR (rendu basé sur la réalité), prototype déjà fait en JavaScript/WebGL
- Rajouter d’autres effets visuels (occlusion ambiante par exemple)

Téléchargez le moteur ici

https://github.com/razor7877/VectorGL

RÉSUMÉ

Le résumé officiel du projet pour le site web

Ce projet vise à développer un moteur de rendu 3D itératif en utilisant C++ et OpenGL, avec une architecture bien structurée pour abstraire les appels OpenGL de bas niveau. Le moteur inclut une interface dynamique permettant aux utilisateurs d'interagir avec les éléments du rendu tels que la scène, les lumières, la skybox et les shaders.

Il intègre également un système de caméra et un éclairage sophistiqué dans l'environnement 3D, supportant de multiples sources lumineuses via un éclairage de type Phong. Des fonctionnalités supplémentaires comprennent le chargement de textures, y compris les cubemaps et les skyboxes, ainsi que l'utilisation d'Assimp pour le chargement de modèles 3D, et la gestion de divers shaders pour créer des effets visuels variés.

Références

OpenGL® 4.5 Reference Pages
https://registry.khronos.org/OpenGL-Refpages/gl4/

Learn OpenGL
https://learnopengl.com/

C++ reference
https://en.cppreference.com/w/

C++ Standards
https://github.com/cplusplus/draft
https://www.open-std.org/jtc1/sc22/wg21/