Rapport Projet : Système Solaire Interactif

Équipe:
- DEHMANI Manar
- CHEN Michel
- ASMAT Valeria

Introduction

Ce projet s'inscrit dans le cadre du cours de Computer Graphics, dont l'objectif est de comprendre et d'implémenter des concepts fondamentaux du rendu 3D en temps réel. L'utilisation d'OpenGL nous permet de manipuler les différentes étapes de la pipeline graphique de manière explicite, offrant une grande liberté dans l'implémentation des techniques de rendu moderne.

Le projet met en œuvre ces connaissances à travers une scène 3D interactive représentant un système solaire simplifié, intégrant plusieurs types de shaders, objets 3D, techniques d'éclairage et effets de post-traitement. Cette réalisation démontre la maîtrise des concepts étudiés en cours, de la gestion des transformations géométriques aux techniques avancées de rendu.

Objectifs

L'objectif principal de ce projet est de mettre en pratique les notions étudiées en cours de Computer Graphics, en développant une scène 3D interactive avec OpenGL 3.x Plus précisément, il s'agit de :

- **Affichage multi-objets** : Rendre plusieurs objets 3D avec des shaders et des matériaux variés (textures, couleurs, environment mapping)
- **Éclairage réaliste** : Implémenter les modèles d'illumination de Phong et Blinn-Phong avec gestion des composantes ambiante, diffuse et spéculaire
- **Chargement de modèles** : Utiliser TinyOBJLoader pour charger des fichiers OBJ avec gestion complète des matériaux via les fichiers MTL
- **Navigation interactive** : Mettre en place une caméra 3D libre avec contrôles clavier/souris intuitifs
- **Post-traitement** : Implémenter des effets visuels avancés via les Framebuffer Objects (FBO)
- **Interface utilisateur** : Intégrer ImGui pour permettre des ajustements en temps réel
- **Optimisation** : Utiliser les Uniform Buffer Objects (UBO) pour optimiser les transferts de données vers le GPU
- **Gestion colorimétrique** : Assurer une gestion correcte de l'espace colorimétrique sRGB

Réalisations

Architecture générale

Le projet utilise OpenGL 3.x Core Profile avec une architecture modulaire comprenant :

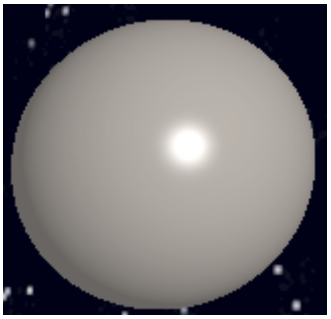
- Classe `Shader` pour la gestion des programmes GLSL
- Classe `Camera` pour la navigation 3D
- Classe `Model` intégrant TinyOBJLoader pour le chargement des assets
- Système UBO pour l'optimisation des uniforms
- Pipeline de post-traitement via FBO

Objets et shaders implémentés

La scène contient trois planètes distinctes, chacune démontrant un type de matériau différent :

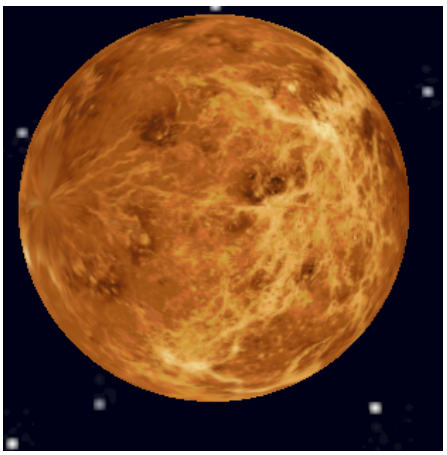
1. Pluto - Shader couleur avec éclairage Phong

- **Matériau** : Couleur unie avec propriétés physiques réalistes
- **Éclairage** : Modèle de Phong complet (ambiant + diffus + spéculaire)
- **Couleur** : Teinte gris-beige caractéristique (0.7, 0.65, 0.6)
- **Animation** : Rotation sur axe incliné - rotation basculée



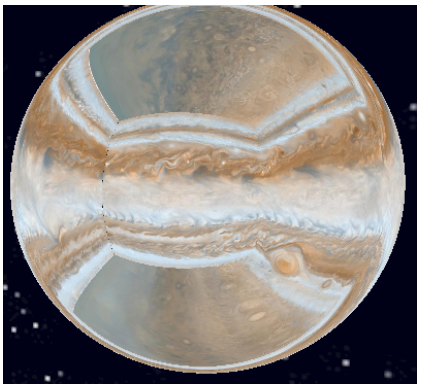
2. Neptune - Shader texture

- **Matériau** : Texture diffuse appliquée sur la géométrie
- **Technique** : Mapping UV avec gestion des coordonnées de texture
- **Animation** : Rotation sur axe diagonal - rotation complexe



3. Jupiter - Environment mapping

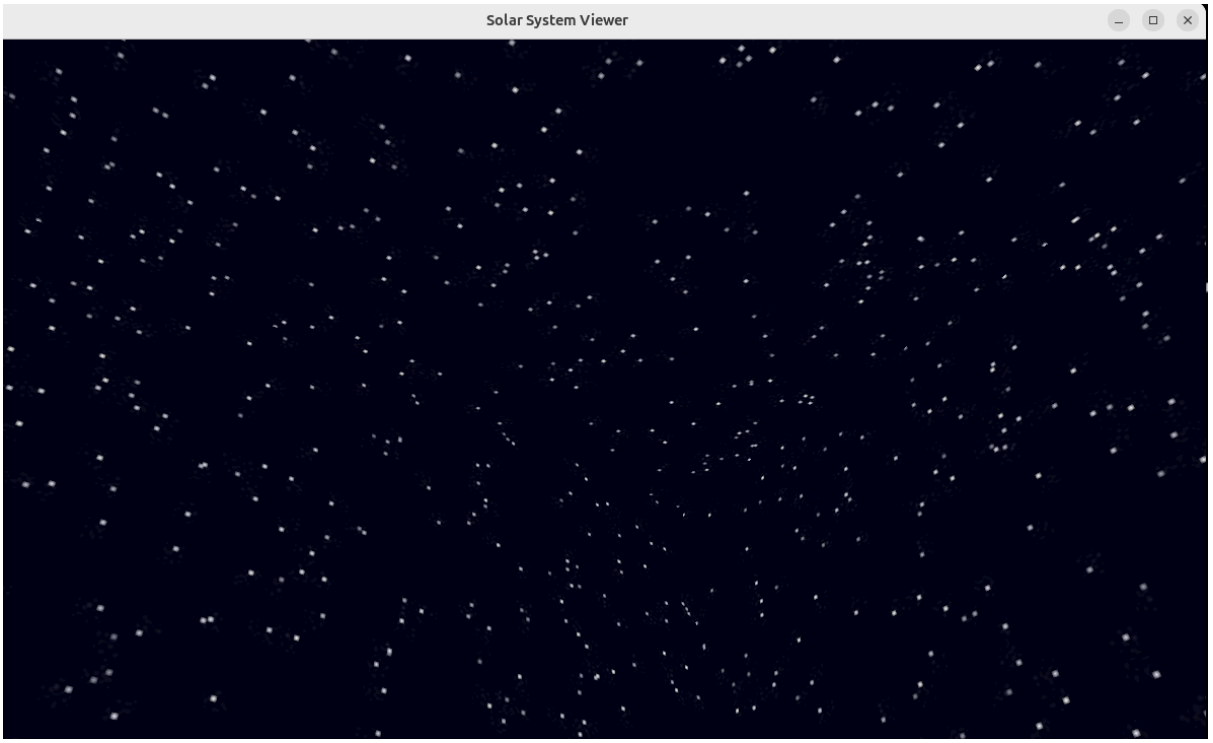
- **Matériau** : Cubemap spécifique pour simulation de surface gazeuse
- **Technique** : Réflexion environnementale basée sur la normale de surface
- **Cubemap** : Textures dédiées Jupiter (6 faces : droite, gauche, haut, bas, avant, arrière)
- **Animation** : Rotation sur axe diagonal - rotation à dominante verticale



Skybox et environnement

Un skybox immersif représente un ciel nocturne étoilé :

- **Implémentation** : Cubemap 6 faces avec depth test optimisé (GL_LEQUAL)
- **Technique** : Suppression de la translation dans la matrice de vue pour effet d'infini
- **Assets** : Textures haute résolution du ciel nocturne



Chargement et gestion des modèles

TinyOBJLoader intégration

- **Formats supportés** : .obj avec fichiers .mtl associés
- **Gestion des matériaux** :
    - Couleurs diffuse (Kd), spéculaire (Ks) et ambiante (Ka)
    - Coefficient de brillance (Ns)
    - Textures diffuses et spéculaires
- **Optimisation** : Regroupement des vertices et gestion efficace des indices

Gestion sRGB

- Activation de `GL_FRAMEBUFFER_SRGB` pour correction gamma automatique
- Chargement des textures en format `GL_SRGB8` pour les textures diffuses
- Pipeline linéaire pour les calculs d'éclairage

Système d'éclairage

Modèle de Phong implémenté

Le modèle d'éclairage combine les composantes ambiante, diffuse et spéculaire pour un rendu réaliste des surfaces.

Configuration lumière

- **Position** : positionnement optimal pour l'éclairage de la scène
- **Composantes** :
    - Ambiante : éclairage de base subtil
    - Diffuse : éclairage principal modéré
    - Spéculaire : reflets blanc pur

Transformations et animations

Placement spatial différencié

- **Pluto** : Translation, échelle 2.0x
- **Neptune** : Translation, échelle 5.0x
- **Jupiter** : Translation, échelle 7.0x

Animations dynamiques

- Rotations continues avec axes différents pour chaque planète
- Vitesses de rotation basées sur `glfwGetTime()` pour fluidité
- Transformations appliquées via matrices de transformation composées

Système de caméra avancé

Navigation libre 3D

- **Navigation caméra :** Déplacement dans la scène avec les flèches directionnelles
- **Déplacement objets** :
    - Z/S : Avancer/Reculer
    - Q/D : Gauche/Droite
    - A/E : Descendre/Monter
- **Rotation** : Contrôle souris avec sensibilité ajustable
- **Zoom** : Molette souris modifiant le champ de vision FOV (zoomer - dézoomer)
- **Implémentation** : Classe Camera avec gestion des vecteurs directionnels (front, right, up,down)

Paramètres caméra

- Position initiale : vue d'ensemble optimale
- FOV dynamique : 45° ± variation zoom
- Clipping planes : near=0.1, far=100.0

Optimisations via UBO (Uniform Buffer Objects)

Architecture UBO mise en place

Structure optimisée pour le transfert des données uniformes vers le GPU avec alignement mémoire approprié.

Structure et alignement mémoire

- Structure optimisée pour le transfert des données uniformes vers le GPU
- Alignement mémoire std140 : garantit la compatibilité entre CPU et GPU
- Utilisation de alignas(16) pour les matrices et vecteurs selon les spécifications OpenGL

Binding points définis

- CameraUBO : Binding point 0
    - Matrices de vue et projection partagées entre tous les objets
    - Position de la caméra pour les calculs de lighting
    - Mis à jour une fois par frame
- TransformUBO : Binding point 1
    - Matrice de transformation (model) individuelle par objet
    - Matrice normale
    - Mis à jour pour chaque objet rendu

Mode de compatibilité automatique

Le système détecte automatiquement la présence d'UBO dans les shaders et bascule en mode compatibilité si nécessaire, garantissant le fonctionnement avec tous types de shaders.

Post-traitement et effets visuels

Framebuffer Objects (FBO)

- **Architecture** : Rendu en deux passes
    - Pass 1 : Rendu de la scène dans texture off-screen
    - Pass 2 : Post-traitement et affichage final
- **Textures** : Couleur (RGBA) + Profondeur (24-bit)
- **Résolution** : Dynamique selon taille fenêtre

Effets disponibles (sélection clavier 1-7)

1. **Aucun effet** - Rendu direct
2. **Flou (Blur)** - Convolution 9x9 pour effet de flou gaussien
3. **Détection de contours** - Filtre Sobel pour mise en évidence des arêtes
4. **Inversion couleurs** - 1.0 - color pour effet négatif
5. **Noir et blanc** - Conversion luminance avec coefficients perceptuels
6. **Contraste élevé** - Amplification des écarts de luminosité
7. **Vignette** - Assombrissement progressif vers les bords

Interface utilisateur ImGui

Fonctionnalités disponibles

- **Contrôle rotation** : Sliders pour ajuster la vitesse de rotation de chaque planète
- **Arrêt/reprise** : Boutons pour stopper individuellement les animations
- **Sélection effets** : Menu déroulant pour choisir l'effet de post-traitement
- **Gestion logs** : Affichage/masquage des informations de debug
- **Informations système** : Affichage FPS, résolution, état des shaders

Interface intuitive

- Fenêtres flottantes repositionnables
- Contrôles en temps réel sans interruption du rendu
- Thème sombre adapté à l'environnement spatial

Gestion des matériaux et textures

Pipeline de chargement

- **Textures** : Format sRGB pour les textures diffuses, Linear pour les normales
- **Filtering** : GL_LINEAR pour interpolation douce
- **Wrapping** : GL_CLAMP_TO_EDGE pour les cubemaps
- **Mipmapping** : Génération automatique pour optimisation distance

Matériaux par shader

- **Color shader** : Propriétés diffuse/spéculaire via uniforms
- **Texture shader** : Sampling de texture + éclairage combiné
- **Environment shader** : Réflexion cubemap basée sur normale de surface

Technologies et bibliothèques utilisées

Cœur graphique

- **OpenGL 3.x Core Profile** - Pipeline moderne programmable
- **GLFW 3.x** - Gestion fenêtre et événements
- **GLAD** - Chargement extensions OpenGL

Mathématiques et géométrie

- **GLM** - Bibliothèque mathématique compatible GLSL
- **TinyOBJLoader** - Chargement modèles OBJ/MTL

Interface et assets

- **ImGui** - Interface graphique immédiate
- **stb_image** - Chargement textures (JPEG, PNG)

Compilation et build

- **CMake** - Système de build multiplateforme
- **C++17** - Standard moderne pour optimisations

Résultats et performance

Rendu en temps réel

- **Framerate** : 60+ FPS constants sur matériel moderne
- **Résolution** : Support multi-résolution
- **Fluidité** : Animations lisses sans saccades

Qualité visuelle

- **Éclairage** : Rendu physiquement plausible avec Phong
- **Textures** : Qualité haute résolution avec filtering optimal
- **Post-traitement** : Effets temps réel sans impact performance significatif

Interactivité

- **Contrôles** : Réponse immédiate et précise
- **Interface** : Modifications en temps réel sans interruption
- **Navigation** : Exploration libre et intuitive de la scène

Problèmes Rencontrés

Le principal défi de ce projet a été la synchronisation du travail en équipe avec des environnements de développement hétérogènes. L'équipe travaillant sur trois systèmes d'exploitation différents (macOS, Windows, Linux), nous avons rencontré de nombreux conflits liés aux différences de gestionnaires de paquets, versions de bibliothèques, configurations de build et chemins de fichiers. Ces incompatibilités ont créé des problèmes récurrents de compilation, des difficultés de partage des fichiers et des configurations de développement, ralentissant considérablement la progression du projet.

Conclusion

En guise de conclusion, il paraît clair que ce projet démontre une maîtrise complète des concepts fondamentaux de Computer Graphics, depuis les transformations géométriques de base jusqu'aux techniques avancées de post-traitement. L'implémentation d'un système solaire interactif offre un cadre visuel attrayant tout en mettant en œuvre l'ensemble des techniques demandées.