

EnLight'INT

Outil pédagogique permettant de visualiser différents effets de lumière dans une scène 3D en temps réel.

Antoine MORRIER
Cédric PENAFIEL
David THERY-BULHA
Victor ROUQUETTE

Responsable : M.PREDA

4 mars 2015

Sommaire

I. Analyse du problème	4
a- Idée de base	4
b- Outils	4
II. Cahier des charges	5
III. Spécifications fonctionnelles	8
a- Support	8
b- Tests	8
IV. Exemple de code source	9

Introduction

Ce document constitue un pré-rapport, intitulé livrable 1, dans le cadre du module de première année CSC 3502, nommé projet informatique. Il présente le programme que nous voulons créer, ses spécifications et notre démarche pour le réaliser.

Le but est de créer un moteur de rendu 3D en temps réel capable d'utiliser différents effets de lumières. Celui-ci sera couplé avec une interface graphique permettant de contrôler ces effets et de voir les changements apportés à la scène de manière interactive.

Ce livrable est constitué de l'analyse du problème, d'un cahier des charges et de la description des spécifications fonctionnelles du programme.

Une dernière partie présente un exemple de code source à l'état actuel.

I. Analyse du problème

a- Idée de base

Que ce soit pour les jeux vidéo, le cinéma ou la CAO, l'illumination de modèles en 3 dimensions est de plus en plus utilisée et de plus en plus complexe au fur et à mesure que l'on essaye de se rapprocher de la réalité.

L'idée de notre programme est de se rapprocher le plus possible d'un moteur graphique malléable, avec, ou sans connaissance de la programmation. Il permettra donc de se familiariser et de visualiser l'effet de différents outils et méthodes d'illumination afin d'en améliorer sa compréhension.

En effet, une personne sans aucune notion de programmation pourra utiliser le moteur graphique à l'aide d'une interface homme machine et ainsi modifier facilement la plupart des paramètres. Une personne qui a des connaissances en programmation pourra tout de même, elle aussi, utiliser ce moteur directement en C++. La partie interface graphique sera pensée de la même manière qu'un plugin à « patcher » sur notre moteur.

b- Outils

Nous allons utiliser le langage C++, l'API OpenGL 4.4, la bibliothèque SDL2, le Framework Qt5, et les bibliothèques GLEW, GLM et Assimp.

Afin de mieux s'organiser sur ce projet, nous allons également utiliser Git (hébergé sur GitHub afin de pouvoir publier notre documentation en ligne) couplé à Doxygen. Le premier nous permettra de travailler chacun sur sa partie de manière efficace en parallèle, et le second nous permettra de fournir une documentation complète de notre projet. Étant donné que nous développons une bibliothèque en premier lieu, cela est plus que nécessaire.

À l'instar du C, le C++ présente les pointeurs comme fonctionnalité ce qui en fait un langage de bas niveau, cependant il possède également de nombreuses fonctionnalités haut niveau qui nous seront très utiles dans le développement de notre projet.

La première est la RAI (Acquisition de la ressource à l'initialisation), grâce au paradigme objet cela permet d'éviter des fuites de mémoires liées aux allocations dynamiques.

- OpenGL 4.4 nous permettra d'avoir un « plein » contrôle sur le GPU. En effet, cette API nous permet un accès relativement bas niveau, ce qui permettra de détacher

au maximum l'utilisation du CPU dans le rendu. Le CPU qui est le principal goulot d'étranglement dans un moteur de rendu en temps réel.

- La SDL2 nous permettra de créer une fenêtre, d'y initialiser un contexte OpenGL, et de pouvoir gérer les événements utilisateur (principalement, bouger la caméra).
- Qt nous permettra d'avoir une interface graphique (GUI : Graphic User Interface) avec des boutons, des curseurs, etc... afin de paramétrer les éléments qui vont constituer notre scène(Lumière,...).
- GLEW permet de charger toutes les fonctions OpenGL sous forme de pointeurs de fonctions.
- GLM est une bibliothèque mathématique permettant de gérer les matrices, quaternions et autres vecteurs que nous aurons à manipuler dans nos équations.
- Assimp quant à elle permet de charger en mémoire vive les modèles aux formats 3D (OBJ, MD2, ...).

II. Cahier des charges

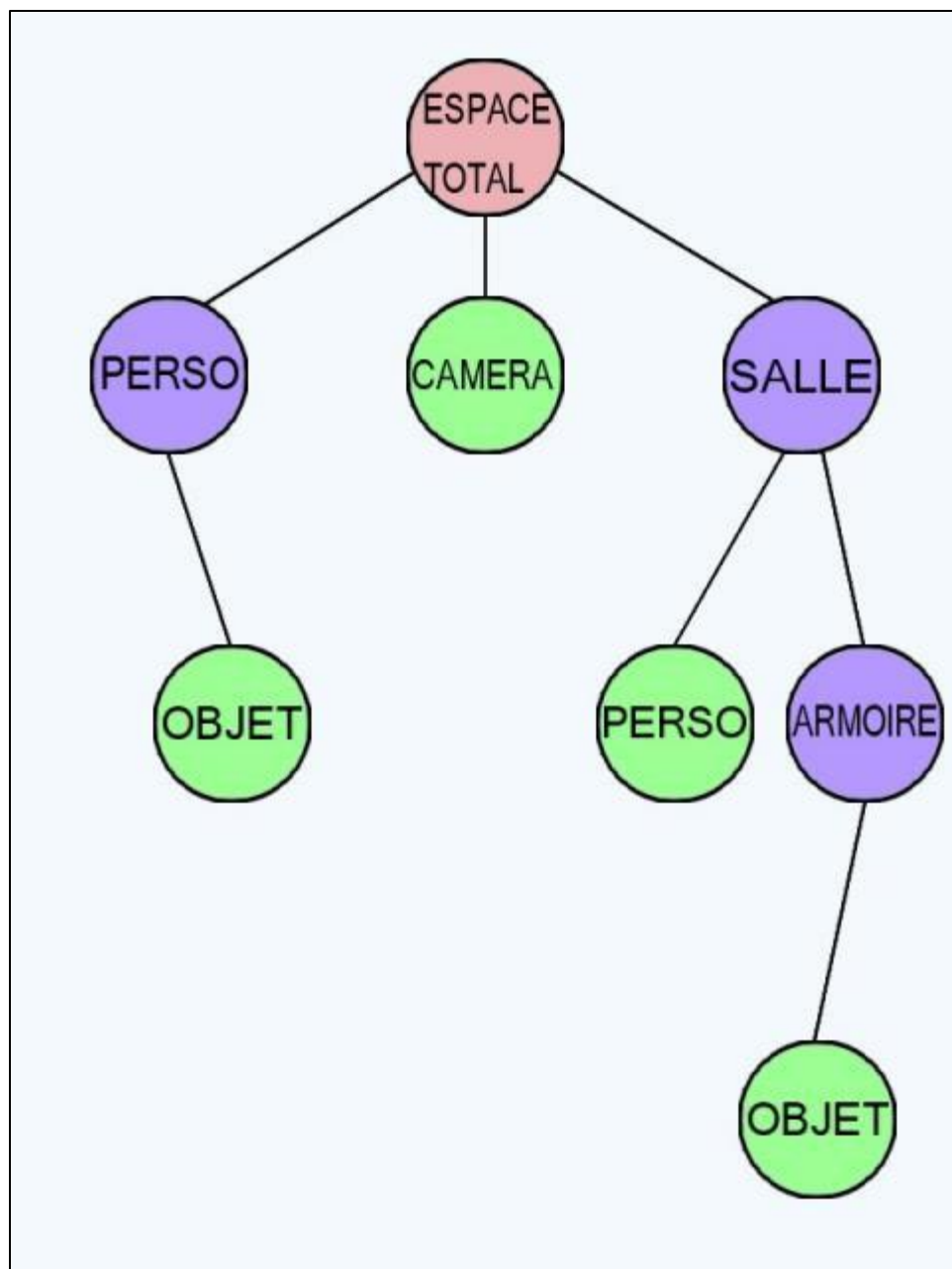
La partie moteur graphique comportera les modules suivants :

- Lecture et affichage de modèles 3D.
- Chargement automatique des textures des modèles 3D.
- Système de dépendances des objets.
- Éclairage direct.
- Gestion des ombres.
- Gestion des réflexions, et réfractions.
- Première approximation de l'illumination globale (jusqu'au second rebond de la lumière).
- L'occlusion ambiante, comme première approximation des ombres liées à l'illumination globale.

Explications succinctes des points ci-dessus :

- La partie chargement des fichiers se fera à l'aide des bibliothèques Assimp et SDL.
- Le système de dépendances des objets sera représenté sous un graphe de scène. C'est-à-dire comme un arbre, les éléments fils dépendant des données relatives de leurs parents (voir figure 1).

FIGURE 1 : LE
GRAPHE DE
SCENE



- Toute la partie concernant l'éclairage, les réflexions et réfractions découleront de l'équation suivante (figure 2) :

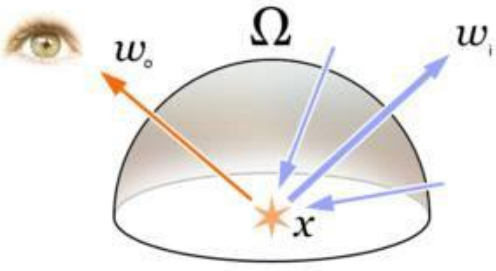
$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_{\Omega} f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t) L_i(\mathbf{x}, \omega_i, \lambda, t) (\omega_i \cdot \mathbf{n}) d\omega_i$$


FIGURE 2 : THE RENDERING EQUATION : JAMES T. KAJIYA (SIGGRAPH 1986)

L'objectif sera d'approximer cette équation le mieux possible.

Le premier terme de cette équation est la radiance sortante au point x, c'est à dire la « couleur » vu depuis la caméra. Ce terme-là sera donc la « couleur » finale du pixel.

Le second terme de cette équation est la radiance émise au point x, il est nul lorsque ce n'est pas une source lumineuse, sinon, il est égale à la « couleur » de la lumière.

Le troisième terme, est une intégrale prise sur un hémisphère (cf. figure 2) qui représente tout ce qui est l'illumination (éclairage direct, réflexions, illumination globale), la réfraction sera alors une donnée « ajoutée » à cette équation.

Pour simplifier le calcul de ce dernier, nous allons utiliser deux types de réflexions : La réflexion diffuse (Si on prend un mur rouge sur une route blanche, et que l'on éclaire le mur, de la lumière rouge, très faible, sera réfléchi sur la route), et spéculaire (comme sur un miroir).

La réflexion diffuse correspond à notre modèle d'illumination globale et sera calculé en « rajoutant » des lumières virtuelles dans la scène.

La réflexion spéculaire correspond à notre modèle de réflexion pure et sera calculé en utilisant une technique d'environnement mapping.

Toutes les options présentées ci-dessus seront réglables grâce à notre interface graphique.

III. Spécifications fonctionnelles

a- Support

Le but est de créer une interface simple à utiliser pour un utilisateur n'ayant aucune expérience avec le rendu 3D mais lui permettant de tester des fonctionnalités.

Le programme devra fonctionner sous Linux, Windows et MacOS (sous réserve de mise à jour des drivers pour gérer OpenGL 4.4).

Il requiert toutefois une carte graphique supportant OpenGL 4.4 (cela concerne la plupart des GPU sortie après 2012). Si ce n'est pas le cas le programme en informera l'utilisateur.

Afin d'avoir une expérience agréable nous souhaitons obtenir un rendu en moins de 50ms, mais nous nous accordons un rendu d'environ 200ms (toutes options comprises : réflexion spéculaire, ombre douce, occlusion ambiante et illumination globale).

Enfin le programme sera aussi capable d'afficher certains indicateurs :

- FPS
- nombre de millisecondes par étapes
- erreurs de compilation des shaders.

En cas de problème (crash, erreur de compilation ou de lancement de certaines fonctionnalités...) il sera possible de récupérer un journal d'erreur simple.

b- Tests

Les tests unitaires concernant les effets du moteur de rendu seront fait « à la main ». Il est en effet difficile de tester leur bon fonctionnement autrement que par l'œil humain.

La solution pourrait être d'utiliser une référence en faisant un rendu photo-réaliste (grâce à OptiX par exemple) puis comparer les images et l'erreur quadratique entre la référence et notre image. Cette solution est toutefois longue à réaliser et elle requiert des outils supplémentaires qu'il faudrait apprendre à maîtriser.

De même l'interface graphique ne pourra pas être testée avec un autre programme. Nous essayerons donc de simuler les comportements d'un utilisateur dans les moindres détails.

IV. Exemple de code source

A titre d'exemple voici la fonction main à l'état actuel qui ne prend en charge, pour l'instant, que la partie moteur de rendu.

```
#include "include/include.h"
#include "SceneManager/scenemanager.h"
#include "Debug/debug.h"
#include "SceneManager/modelnode.h"
#include "SceneManager/pointlightnode.h"

using namespace std;
using namespace glm;
using namespace GXY;

int main(int argc, char *argv[])
{
    try
    {
        Device device(800, 600); // Create a Device

        // Capture and hide cursor
        device.mouse()->captureCursor(true);
        device.mouse()->showCursor(false);

        SceneManager sceneManager; // Create a Scene Manager

        // Create a FPS Camera
        sceneManager.createCameraFPS(vec3(0.1, 200.0, .0), 300, 0.5);

        // Get the Root Node
        shared_ptr<Node> rootNode = sceneManager.getRootNode();

        // Add a PointLight on RootNode
        shared_ptr<PointLightNode> light = addPointLight(rootNode);

        // Add a Model "Sponza Atrium"
        shared_ptr<ModelNode> model = addModel(rootNode,
                                                "models/OBJ/crytek-sponza/sponza.obj");

        // Configure the light
        light->setColor(vec3(1.0, 1.0, 1.0));
        light->setPosition(vec3(0.0, 200.0, .0));
        light->setRadius(10000);
        light->setIntensity(1.0);
        light->enableShadowMaps(0);

        while(device.run())
        {
            device.begin(); // Clear Window

            sceneManager.render(); // Render Scene
        }
    }
}
```

```
        device.end(); // Swap Buffer
    }
}

catch(Except &exc)
{
    std::cerr << exc.what() << std::endl;
    return EXIT_FAILURE;
}

return EXIT_SUCCESS;
}
```