2020

INF4034 – Imagerie numérique

PROJET – IMPLEMENTATION DE L'ALGORITHME DE LANCER DE RAYON

TEXIER AURELIEN

Présentation du projet

Le but de ce microprojet est d'implémenter l'algorithme de lancer de rayon vu en cours. Les fonctionnalités minimum demandées sont les suivantes :

- Effectuer un lancer de rayon avec un modèle de matériau de Blinn-Phong ;
- Gérer les sphères, plans et faces triangulaires comme primitives ;
- Sauvegarder cette image dans un fichier de sortie.

D'autres fonctionnalités pourront aussi être implémentées comme :

- Afficher l'image calculée dans une fenêtre (SDL par exemple) ;
- Réflexion et réfraction avec plusieurs niveaux de récursivité ;
- Radiosité ;
- Anti-aliasing ;
- Gestion des textures ;
- Chargement d'objet depuis un fichier (.obj par exemple) ;
- …

Ce projet se déroulera utilisera l'environnement Visual Studio et sera développé en C++ avec la librairie FreeImage.

Déroulement du projet

Cette partie est là pour vous guider dans le développement mais libre à vous de faire autrement.

Mathématiques

Dans un premier temps, il est conseillé de faire une librairie mathématique permettant de gérer au minimum les vecteurs 3D. On y retrouvera plusieurs méthodes utiles comme :

- La normalisation d'un vecteur ;
- Le calcul du produit scalaire ;
- Le calcul du produit vectoriel ;
- Le calcul d'un rayon réfléchi par rapport à une direction ;
- …

Concernant la partie mathématique, les représentations cartésiennes des vecteurs et transformations suffisent et le recourt aux matrices homogènes n'est pas nécessaire. Il simplifie néanmoins grandement les choses et vous pouvez choisir de l'implémenter.

Gestion des objets 3D

Afin de gérer les différents modèles pour représenter un environnement 3D, il est recommandé d'implémenter tout ou partie des classes (ou structures) suivantes :

- Une classe **caméra** ;
- Une classe permettant de représenter les **sources lumineuses** ;
- Une (ou plusieurs) classe(s) gérant les **objets primitifs** (sphères, plans, faces triangulaires) ;
- Une classe gérant les **matériaux** ;
- Une classe représentant les rayons [*optionnel*];
- Une classe permettant la **gestion des intersections rayons/objets** ;
- Une classe représentant la **scène** ;
- …

Afin de rendre le code plus lisible et épuré, chaque classe sera définie dans un couple .cpp/.h et il n'y aura qu'une seule classe par couple .cpp/.h.

FreeImage

Dans le projet actuel, vous aurez à utiliser la librairie FreeImage pour gérer vos images (création, gestion, sauvegarde, …). Voici quelques types/fonctions utiles de la librairie FreeImage.

RGBQUAD : Structure représentant une couleur.
FIBITMAP * : Pointeur stockant les images dans la librairie FreeImage.

FreeImage_Allocate()

Fonction permettant d'allouer l'espace mémoire d'une image FIBITMAP.

FreeImage_Load()

Fonction permettant de charger un fichier image en image FIBITMAP.

FreeImage_Unload()

Fonction pour désallouer une image FIBITMAP.

FreeImage_SetPixelColor()

Fonction permettant d'affecter une couleur à un pixel de l'image FIBITMAP.

FreeImage_Save()

Fonction permettant de sauvegarder une image FIBITMAP en un fichier image. La fonction gère plusieurs types d'image (BMP, JPEG, PNG, …).

La librairie compilée pour Windows ainsi que la documentation de FreeImage vous est fourni avec le sujet.

Intersection rayon/sphère

Le principe de l'algorithme repose sur le calcul d'intersections entre les rayons qui se propagent dans l'espace et les primitives.

Par exemple pour calculer l'intersection entre un rayon et une sphère :

Soit un rayon d'origine $\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$ et de direction $\begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix}$ représenté par l'équation paramétrique :

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + t. \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix}. \qquad (1)$$

Soit une sphère de centre $\begin{bmatrix} x_c \\ y_C \\ z_c \end{bmatrix}$ et de rayon R.

Les points M de coordonnées $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$ appartiennent à la sphère si et seulement si :

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = R^2 \qquad (2)$$

En substituant *x*, *y* et *z* dans (2) à partir de leurs expressions tirées de (1) on obtient une équation du second degré en *t* :

$$t^2.\left({d_x}^2 + {d_y}^2 + {d_z}^2\right) + 2t.\left(X_0.d_x + Y_0.d_y + Z_0.d_Z\right) + \left({X_0}^2 + {Y_0}^2 + {Z_0}^2\right) - R^2 = 0$$

avec $X_0 = x_0 - x_c$, $Y_0 = y_0 - y_c$ et $Z_0 = z_0 - z_c$

En interprétant le discriminant de l'équation, on détermine si le rayon intersecte la sphère en 2 points (discriminant strictement positif), s'il est tangent ou s'il n'intersecte pas la sphère.

Dans le cas d'une intersection, le signe des solutions *t* permet d'identifier si l'intersection est située devant ou derrière le point de départ du rayon. Les valeurs de *t* permettent de trier les intersections dans l'ordre (pensez à les stocker pour tester avec les intersections d'autres primitives de la scène). Finalement, en injectant les solutions *t* trouvées dans (1), on détermine les coordonnées cartésiennes de l'intersection.

Rendu

Le projet est à faire par groupe de 4 au maximum (la taille du groupe sera prise en compte dans la notation).

Le projet sera à rendre sur un dépôt Moodle qui sera créé à cet effet.

Le projet sera rendu sous format **.zip** avec pour nom : INF4034_NOM(1_NOM2_...).zip.

L'archive contiendra :

- Le répertoire du ***projet Visual nettoyé***.
- Un répertoire ***bin*** contenant le ***programme compilé*** et les dépendances éventuelles. Il sera éventuellement accompagné d'un fichier ***README.txt*** expliquant l'usage du programme.
- Un ***répertoire des images*** que vous aurez générées.