TP – Moteur graphique 3D minimaliste

L'objectif de ce TP est de produire un petit moteur d'affichage d'objets en 3 dimensions. Les fichiers fournis en annexes se chargent de créer une fenêtre graphique dans laquelle il est possible d'allumer ou d'éteindre des pixels. Votre charge consistera en la gestion des éléments à afficher.

Attention, le but de cette réalisation reste pédagogique et vise des performances modestes (par exemple 60 images par secondes pour 1000 à 2000 triangles 3d).

1. Affichage graphique en deux dimensions

Pour cette série de TP, nous allons utiliser la bibliothèque SDL 1.2 pour la gestion de l'affichage. Les fichiers lib_surface.c et lib_surface.h donnent des méthodes d'abstraction permettant de limiter la dépendance à la bibliothèque SDL (en changeant de bibliothèque il faudra en pratique simplement redéfinir les fonction de lib_surface.c).

Les fichiers lib_2d.c et lib_2d.h définissent plusieurs fonctions ayant pour but l'affichage de triangles dans une fenêtre graphique.

Le fichier prog.c vous donne un exemple d'utilisation des fonctions définies dans lib_2d.h.

Pour tester une exécution complète, vous pouvez utiliser les commandes suivantes dans un environnement linux :

```
gcc -c lib_2d.c -o lib_2d.o
gcc -c lib_surface.c -o lib_surface.o
gcc prog.c lib_2d.o lib_surface.o -lm -lSDL -o prog
./prog
```

Si vous souhaitez utiliser un environnement windows, il convient d'installer la bibliothèque SDL[1].

Les algorithmes permettant de remplir correctement les triangles en deux dimensions seront vus en séance de TD, vous n'avez rien à développer à ce moment du TP, vous devez vous concentrer sur une exécution correcte du programme donné en exemple.

Par convention, on considérera la définition des axes (X,Y) dans le plan 2d telle que le pixel aux coordonnées (0,0) doit être dans le coin supérieur gauche de l'écran.

2. Gestion de triangles en 3 dimensions

Pour gérer les triangles en trois dimensions, nous allons utiliser des transformations matricielles s'exprimant sous la forme :

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix} \qquad \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

La matrice A pouvant être une matrice de translation, ou de rotation par exemple. Le cas de la matrice de projection B est différent puisque permettant de définir des coordonnées dans un espace à deux dimensions.

Par convention, on considérera la définition des axes (X,Y,Z) dans le plan 3d telle que le pixel aux coordonnées (0,0,0) doit être au centre de l'écran.

2.1 allocation de mémoire

Vous devez compléter les fichiers lib_3d.c et lib_3d.h afin d'ajouter une structure de données de triangle en 3 dimensions ainsi que les fonctions d'allocation de points ou triangles en 3 dimensions, et de remplissage de triangle en 3 dimensions en vous basant sur l'utilisation des versions 2d de ces fonctions.

1 http://openclassrooms.com/courses/apprenez-a-programmer-en-c/installation-de-la-sdl

2.2 rotation

En ce qui concerne la rotation d'un point en 3 dimensions, on cherchera à la décomposer par des rotations consécutives selon les 3 axes (qui restent donc des rotations en 2 dimensions). Vous devez donc retrouver écrire 3 fonctions de rotation selon chaque axe.

L'écriture de la matrice de rotation d'un point en 3 dimensions (et donc de rotation d'un triangle en 3 dimensions) ne pose pas de difficulté. La matrice de rotation d'un angle de r radians selon l'axe Z du point (x,y,z) s'exprime simplement comme :

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} \cos(r) & -\sin(r) & 0 \\ \sin(r) & \cos(r) & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

2.3 translation

En ce qui concerne la translation d'un point dans un espace à trois dimensions, la difficulté réside dans la recherche de la matrice permettant la translation...

Pour contourner le problème, nous utiliserons des coordonnées homogènes. Ces coordonnées homogènes introduisent une quatrième dimension. Par convention la valeur de cette quatrième coordonnée sera nulle pour un vecteur et non nulle pour un point.

$$\begin{pmatrix} u \\ v \\ w \\ 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Vous l'avez compris, à ce stade il est probable que vous deviez réécrire une partie des fonctions précédentes...

3. Gestion d'un objet en 3 dimensions

3.1 allocation de mémoire et objets simples

Le fichier lib_objet3d.h vous donne la structure de données qui sera utilisée pour la gestion d'un objet à 3 dimensions : une liste chaînée de triangles à 3 dimensions.

Vous devez compléter l'ensemble des fonctions de définition des objets basiques qui pourront être utilisés. Vous pouvez noter que la définition de la sphère utilise des dimensions en latitude et longitude (nombre de divisions sur les deux axes de la surface), vous devez donc passer en coordonnées polaires pour définir les sommets des triangles[2].

3.2 transformations d'objets

Pour les fonctions de transformations des objets, vous devez réutiliser les fonctions de transformations des triangles en 3 dimensions, et appliquer ces transformations sur l'ensemble des faces de l'objet.

3.3 composition d'objets

Les fonctions de compositions d'objets servent à définir des objets complexes comme basés sur des objets simples. Par exemple un arbre pourra être défini comme un parallélépipède pour le tronc et une pyramide pour les feuilles, ou une voiture comme deux parallélépipèdes et 4 sphères.

4. Algorithme du peintre

Afin de permettre un affichage correct d'un objet complet, vous pouvez vous inspirer de l'algorithme du peintre pour afficher votre objet (commencer par les faces les plus éloignées et terminer par les faces les plus proches afin

2 On aurait pu tenter une définition de sphère géodésique, mais l'algorithme le permettant est plus complexe qu'avec des coordonnées polaires.

que ces dernières ne soient pas masquées).

L'ordre de ces faces est important et vous aurez (à cause d'une composition d'objets, ou d'une transformation quelconque) à retrier régulièrement les faces de l'objet. Pour faire ce tri, vous pouvez vous baser sur la coordonnée Z moyenne de chaque face comme clé de tri. Afin d'accélérer votre traitement, vous pouvez ajouter un champ indiquant si votre objet possède encore ses faces dans un ordre trié ou non (ce champ de la structure pourra être modifié en fonction des transformations).

5. Pour aller plus loin

Cette partie sort du cadre du module pour le groupe R&T et fait intervenir la notion de fonction récursive. Vous pouvez néanmoins poursuivre votre travail afin d'introduire la notion de scène d'objets 3d (de manière plus sereine que par la composition d'objets...).

Vous avez à écrire l'ensemble des fichiers lib_scene3d.h et lib_scene3d.c pour définir une scène comme un arbre dont chaque nœud est un objet et le rendu se fait en affichant tous les objets à partir de la racine. Cet arbre vous permet de définir des relations (le cube est posé sur le sol) et appliquer une transformation du sol (rotation par exemple) au cube. En pseudo code cet algorithme s'écrit :

```
procédure construction_recursive(entrée : nœud_arbre, \
                      entrée : matrice_transf, \
                      entrée : matrice_transf_inverse)
        appliquer matice_transf aux coordonnées de  nœud_arbre
        ajouter les faces de nœud_arbre à la liste des faces à trier
        appliquer matice_transf_inverse aux coordonnées de  nœud_arbre
        pour tous fils de nœud_arbre
                construction_recursive(fils,\
                       matrice_transf*matrice_noeud_arbre,\
                       matrice_transf_inverse*matrice_noeud_arbre_inverse)
        fin pour
fin
```

Cette procédure produit une liste de faces représentant la composition de l'ensemble des objets. Cette liste doit être triée puis affichée comme dans l'algorithme du peintre.

Vous pouvez ainsi, en positionnant un ou plusieurs objets caméra au niveau des nœuds de l'arbre, changer la position de la caméra en élisant un de ces nœuds comme étant une nouvelle racine (c'est compliqué mais intéressant...).