Projet API : algorithme du peintre

1 Objectif

1.1 Présentation du problème

L'objectif de ce projet est d'implémenter *l'algorithme du peintre*. A partir d'une description d'un objet 3D sous la forme d'une liste de polygones, cet algorithme permet de produire une image 2D tenant compte de la visibilité de ces polygones compte tenu de leur distance respective par rapport à l'observateur.

Le principe de cet algorithme est illustré en figure 1. On cherche à produire une image d'un objet 3D, tel que le verrait un observateur situé au-dessus, et regardant parallèlement à l'axe Oz vers le bas.

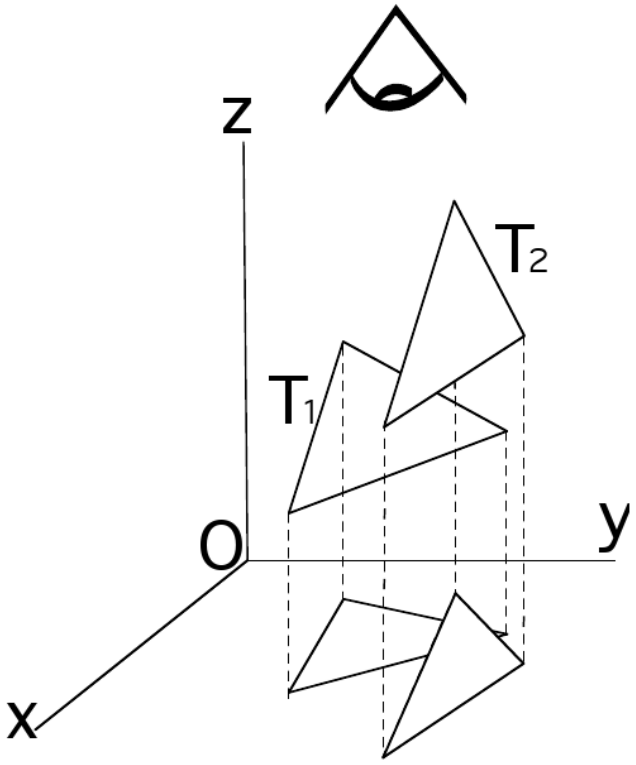

FIGURE 1 – Principe de l'algorithme du peintre

Chaque point est projeté dans le plan de l'image produite : seules les deux premières coordonnées (x, y) restent. La dernière coordonnée z permet de repérer si les triangles [1] sont proches ou éloignés de l'observateur. Dans cette configuration, les triangles situés le plus proche de l'observateur doivent "cacher" les triangles les plus éloignés. Ainsi dans la figure 1, le triangle T_1 doit être dessiné avant le triangle T_2, car il est plus éloigné de l'observateur, et ainsi T_2 viendra recouvrir partiellement T_1 lors de son dessin.

L'algorithme du peintre consiste à trier les triangles composant l'objet du plus éloigné vers le plus proche, et à les dessiner dans cet ordre. Ainsi le résultat final est cohérent : les triangles dessinés en dernier venant cacher partiellement les triangles dessinés en premier.

La figure 2 montre un objet dont les triangles sont dessinés dans un ordre quelconque. La figure 3 montre le même objet, dessiné cette fois dans l'ordre croissant des coordonnées z.

1. On suppose dans la suite que les objets sont constitués d'une collection de triangles : l'algorithme est facilement généralisable à une collection de polygones quelconques.

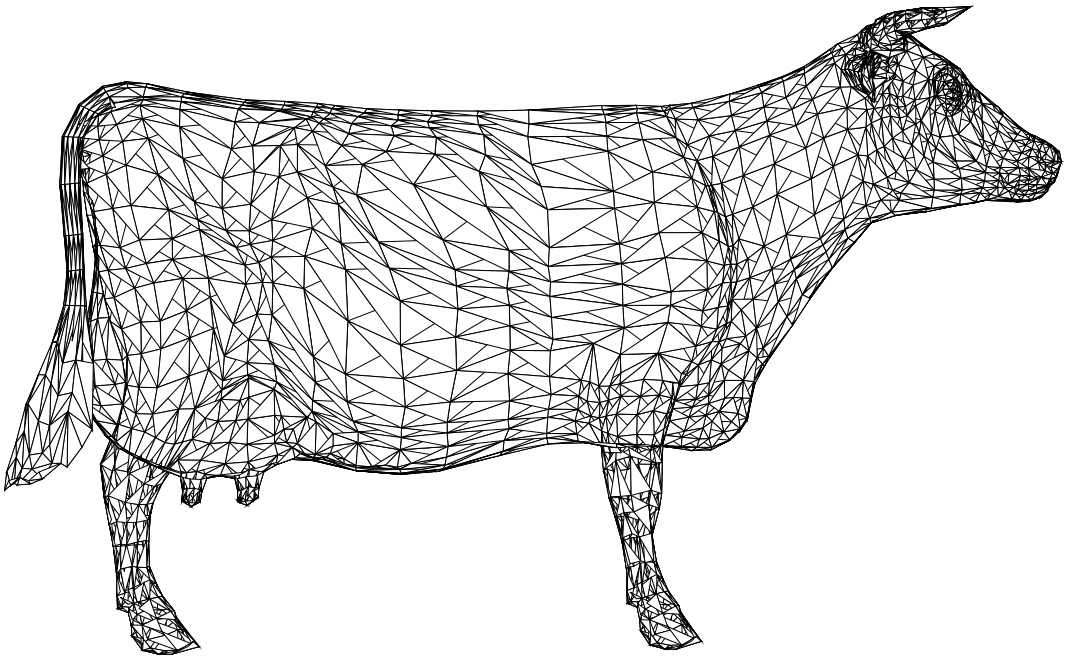

FIGURE 2 – Objet dessiné dans un ordre quelconque

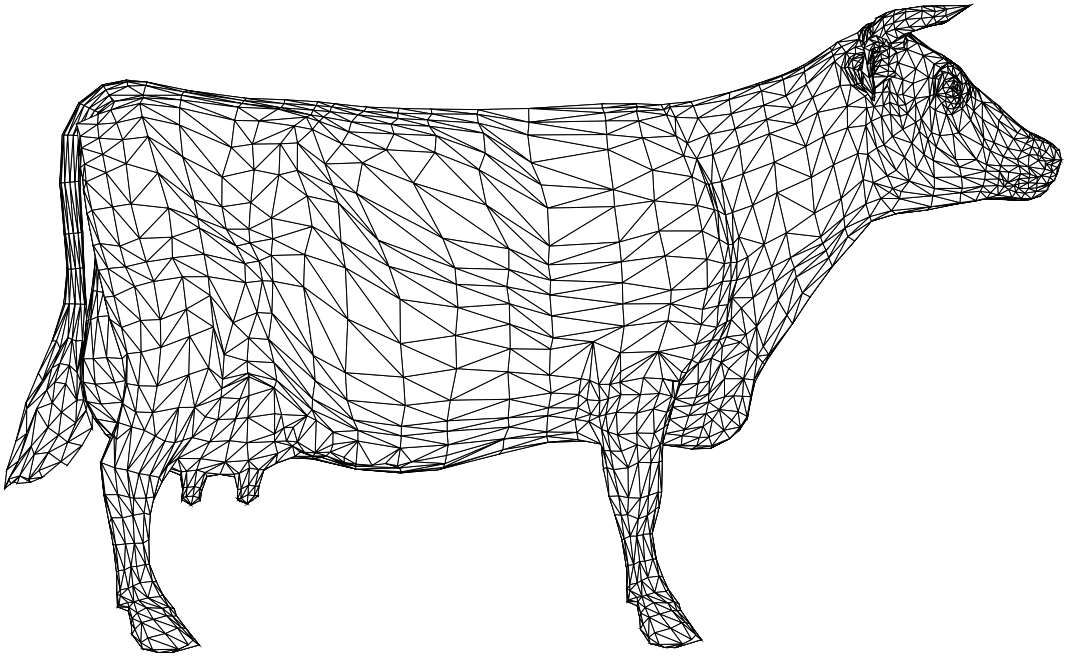

FIGURE 3 – Objet dessiné par ordre croissant des coordonnées z des triangles

1.2 Travail à réaliser

Le travail à réaliser est donc d'écrire un programme complet qui va :

1. lire un objet dans un fichier au format OFF (*Object File Format*) (section 2 ;
2. trier, *en utilisant l'algorithme du tri par paquets* les triangles composant cet objet (section 3) ;
3. produire un fichier PostScript contenant le dessin en 2D de l'objet, dans l'ordre fourni par le tri (section 4).

L'ordre des sections ci-dessous ne reflète pas nécessairement l'ordre dans lequel vous devez traiter les questions. Vous devez avoir lu et compris l'ensemble du sujet avant de commencer toute réalisation.

2 Lecture de l'objet à afficher

Un objet est constitué d'une collection de triangles. Un triangle est constitué de trois sommets. Un sommet possède trois coordonnées (x, y, z).

2.1 Travail a realiser : structures de donnees

Spécifier et implémenter un paquetage contenant les structures de données permettant de représenter les différents objets à manipuler (points, triangles, etc.).

NB : Ces structures de données sont dépendantes de l'usage qui en sera fait par la suite (cf sections 3 et 4).

2.2 Travail à réaliser : lecture d'un objet dans un fichier

Les objets sont décrits dans des fichiers au format OFF (*Object File Format*). La description détaillée de ce format se trouve en annexe B.

Ces fichiers contiennent :
— une liste de sommets, décrits par leurs coordonnées (x, y, z)
— une liste de triangles, chaque triangle étant composé de trois sommets, représentés par leur indice dans la liste précédente.

3 Tri par paquets

Pour le tri de triangles, on dira qu'un triangle T_1 est plus petit qu'un triangle T_2 ssi le *minimum des coordonnées en Z des sommets de* T_1 est plus petit que le *minimum des coordonnées en Z des sommets de* T_2.

Pour trier les triangles d'un objet, vous allez utiliser l'algorithme du *tri par paquet* (*bucket sort*). Cet algorithme permet de trier n objets auxquels sont associé une *clé*, nombre réel compris dans un intervalle $[a, b]$. Pour trier une collection d'objets par clé croissante, on utilise un *tableau* $T[0..n]$ *de* $n + 1$ *listes*, dans lequel $T[i]$ va contenir la *liste ordonnée* (le paquet) des objets dont la clé appartient à l'intervalle

$$\left[a + i.\frac{(b-a)}{n}, a + (i+1).\frac{(b-a)}{n}\right[.$$

L'algorithme est donc :

Pour chaque objet de clé c

— { *calculer l'indice i du paquet correspondant à c* }

— i $\longleftarrow \left\lfloor n \cdot \frac{c-a}{b-a} \right\rfloor$

— insérer l'objet dans la liste ordonnée T[i]

NB : en Ada, la partie entière d'un nombre flottant `X` s'obtient par `Float'Floor(X)`.

Le calcul de i étant de coût constant (il n'y a pas besoin de parcourir le tableau T), si les clés sont uniformément distribuées sur l'intervalle $[a, b]$, alors les listes $T[i]$ contiendront en moyenne un seul élément, et le coût moyen de l'insertion dans une liste sera en $O(1)$. Le tri complet est alors en $O(n)$.

3.1 Travail à réaliser : paquetage de tri

Dans le cadre de ce projet, la *clé* d'un triangle est le minimum des coordonnées en Z des sommets de ce triangle. Avant d'allouer et d'initialiser T, il est nécessaire de connaitre :

1. la taille de T, soit le nombre de triangles composant l'objet,
2. l'intervalle $[a, b]$, soit le minimum et le maximum des coordonnées en Z des sommets.

L'algorithme complet à implémenter devient donc :

— Ouvrir le fichier OFF
— Lire le nombre de sommets m
— Lire le nombre de faces n
— Créer un tableau de sommets de taille m
— Lire les m sommets du tableau, en calculant le minimum a et le maximum b des coordonnées en Z
— Créer un tableau de listes de triangles de taille n
— Lire les n triangles en les insérant dans le tableau de listes
— Parcourir le tableau de listes pour produire le dessin 2D

Réaliser un paquetage permettant de déclarer et d'initialiser un tableau de listes, spécifier et implémenter l'insertion d'un triangle dans cette structure.

4 Ecriture de l'image dans un fichier

Une fois la collection de triangle lue et triée, on peut produire l'image 2D dans un fichier PostScript.

4.1 Travail à réaliser : écriture d'un fichier PostScript

Dans le paquetage de tri par paquets, spécifier et implémenter des primitives, **sur le modèle d'une machine séquentielle**, permettant de parcourir la structure de tableau de listes contenant les triangles triés.

Réaliser un paquetage utilisant ces primitives pour parcourir la structure et produire un fichier PostScript contenant le dessin en 2D de l'objet.

5 Tests

Les fichiers OFF fournis comportent de très nombreux points et triangles, et permettent difficilement de s'assurer de la correction du programme. Il est donc nécessaire de tester celui-ci sur un jeu d'essai composé de tests simples, permettant de tester le maximum de situations possibles.

5.1 Travail à réaliser : jeu d'essai

Ecrire un jeu d'essai permettant de tester tout ou une partie de votre programme. Des tests unitaires (testant un module donné) peuvent être réalisés à l'aide de programmes Ada annexes (qui ne seront pas utilisés dans le programme principal).

6 Evaluation

On souhaite évaluer l'intérêt du choix du tri par paquets. En effet, l'intérêt de cet algorithme suppose que les clés considérées soient uniformément distribuées dans l'intervalle $[a, b]$.

6.1 Travail à réaliser : évaluation de performances

En théorie, l'hypothèse énoncée ci-dessus est-elle vérifiée dans le cas de l'application du tri par paquets à l'algorithme du peintre ?

Proposer et implémenter une méthode permettant de mesurer la performance de l'algorithme de tri par paquets sur les exemples fournis.

7 Travail supplémentaire

A faire s'il reste du temps, une fois que le travail est entièrement terminé. Ce travail supplémentaire ne compte qu'en bonus dans l'évaluation du projet. Les deux questions sont indépendantes, et peuvent être traitées dans n'importe quel ordre.

7.1 Paquetage générique tri par paquets

Rendre générique le paquetage de tri par paquets : quels sont les paramètres du paquetage générique ?

7.2 Polygones quelconques

Traiter le cas ou les faces d'un objet ne sont plus forcément des triangles, mais des polygones quelconques. Dans un fichier OFF, une ligne décrivant un polygone se présente comme suit :

n `sommet`$_1$ `sommet`$_2$ `...` `sommet`$_n$

où n est le nombre de sommets du polygone, et *sommet* est la liste des indices des sommets composant le polygone.

Quelles sont les modifications à apporter aux différentes structures de données ?

8 Aspects pratiques

8.1 Rendu

Le projet s'effectue en binôme. Le rendu de projet se fera par mail à l'enseignant de TD. Vous devrez envoyer une archive au format zip contenant un répertoire dont le nom est constitué des 2 noms des membres du binôme, séparé par un tiret. Ce répertoire devra contenir :

- un répertoire nommé *source* contenant l'intégralité des sources de votre projet,
- un document *au format pdf* décrivant votre projet. Il doit contenir au moins les sections suivantes :
  - Listes des paquetages : contient la liste des paquetages et une brève description de la fonction de chacun d'eux.
  - Utilisation : décrit comment utiliser votre programme.
  - Tests : une description du jeu d'essai, des tests unitaires réalisés.
  - Evaluation de performances : une réponse argumentée à la question posée en section 6, une description détaillée de la méthode d'évaluation proposée, et les résultats de son application.
  - Difficultés rencontrées : donne une liste des difficultés que vous avez rencontrées, des problèmes techniques qui empêchent votre programme de fonctionner, des eventuelles fonctionnalités que vous n'avez pas eu le temps de coder.

8.2 Calendrier et échéance

L'archive doit être envoyée par mail avant le Vendredi 11 décembre à 23h.

Ce projet s'étale sur trois séances de TP encadrées. Il est fortement conseillé d'avancer entre deux séances.

8.3 Travailler en équipe

Travailler même à deux est un travail d'équipe. Nous vous conseillons de vous répartir les tâches, de manière à pouvoir travailler indépendamment l'un de l'autre.

Un travail préliminaire au codage est donc *la conception* qui vous permettra d'identifier les différents paquetages dont vous avez besoin, et les dépendances entre eux.

Pensez ensuite à *planifier* les différentes tâches, pour trouver un ordre cohérent dans les différentes implémentations que vous ferez (même si vos encadrants seront là pour vous guider).

A Pointeurs et allocation dynamique en Ada

A.1 Déclaration

Les pointeurs en Ada s'utilisent grâce au mot clef **access**. Soit `T_Quelconque` un type quelconque, un type "pointeur sur un objet de type `T_Quelconque`" se déclare comme ceci :

```
type T_Pointeur_T_Quelconque is access T_Quelconque ;
```

Désormais, toute variable (ou instance) de ce nouveau type `T_Pointeur_T_Quelconque` sera considérée comme pouvant pointer sur un objet de type `T_Quelconque`. On notera que le 'type pointeur n'existe pas en soi, seul un type "pointeur sur un type donné" peut être défini. On déclare deux pointeurs de ce type de la façon suivante :

```
Pointeur_1, Pointeur_2 : T_Pointeur_T_Quelconque ;
```

Ils pourront ainsi désigner chacun un objet de type `T_Quelconque`. A ce moment là , ils ne pointent sur rien : leur valeur est automatiquement initialisée à **NULL** lors de la déclaration.

Pour la déclaration de types mutuellement récursifs (contenant des pointeurs vers ces types, pour les listes chainées par exemple), on doit effectuer des déclarations incomplètes :

```
type Doublet;  -- d\'eclaration incompl\'ete (pour la r\'ecursivit\'e)

type AdDoublet is access Doublet;  -- type pointeur

type Doublet is
  record
    Info : Integer;  -- type des \'el\'ements
    Suc : AdDoublet;  -- pointeur sur un autre Doublet
  end record;
```

A.2 Création et accès à l'objet pointé

A tout moment l'utilisateur peut créer des objets dans le corps d'un sous-programme au moyen de l'allocateur **new** portant sur le type de l'objet qui doit être pointé. La valeur de retour est l'adresse mémoire dynamique que le système lui a allouée :

```
Pointeur_1 := new T_Quelconque ;
Pointeur_2 := new T_Quelconque ;
```

Les zones mémoire pour chaque objet pointé sont réservées ; cependant elles n'ont toujours pas de valeurs. Celles-ci sont accessibles avec **.all** :

```
Pointeur_1.all := XXX ;
Pointeur_2.all := YYY ;
```

A.3 Libérer la mémoire allouée dynamiquement

Il faut instancier la procédure générique `Ada.Unchecked\_Deallocation`. L'instance fournit une procédure permettant de libérer un pointeur du type voulu, comme illustré dans l'Exemple suivant :

```
with Ada.Unchecked_Deallocation;

package Foo is

    type Tab is array (Natural range <>) of Integer;

    type Tab_Access is access Tab;
```

```
    procedure Free is
      new Ada.Unchecked_Deallocation (Tab, Tab_Access);
      --  Si on a une variable T de type Tab_Access, on
      --  peut maintenant invoquer Free (T).
    end Free;

end Foo;
```

On doit instancier cette procédure pour chaque type de pointeur.

B Description du format *Object File Format*

Les fichiers au format *Object File Format* (OFF) sont des fichiers ASCII, d'extension `.off`. Ces fichiers contiennent, dans l'ordre :

— une première ligne avec la chaine de caractères `OFF`
— une deuxième ligne avec trois entiers : l'entier NSommet (nombre de sommets), l'entier NFace (nombre de faces), et l'entier 0 (nombre d'arêtes, non utilisé ici)
— NSommet lignes avec sur chaque ligne les trois coordonnées (x, y, z) d'un sommet
— NFace lignes de quatre entiers : l'entier 3 (nombre de sommets du polygones : 3 pour un triangle), puis trois entiers entre 0 et NSommet $-$ 1 donnant les indices des sommets de la face.

Exemple de fichier représentant deux triangles qui relient quatre sommets :

```
OFF
4       2       0
0.0     0.0     0.0
10.0    0.0     10.0
10.0    10.0    0.0
0.0     10.0    0.0
3       0       1       2
3       0       2       3
```

C Dessin de triangles en PostScript

Le fichier de sortie PostScript doit débuter par la ligne suivante :

```
%!PS
```

et se terminer par la ligne suivante :

```
showpage
```

Entre ces deux lignes, on dessine des triangles. Pour dessiner un triangle plein avec les sommets de coordonnées (x_1, y_1), (x_2, y_2), (x_3, y_3), les commandes PostScript sont les suivantes :

```
x1 y1 moveto
x2 y2 lineto
x3 y3 lineto
x1 y1 lineto
gsave
1 setgray
fill
grestore
0 setgray
stroke
```

L'intérieur du triangle est alors blanc et le contour noir. Si vous préférez de la couleur, vous pouvez changer les lignes `1 setgray` et `0 setgray` par des lignes du type :

```
r g b setrgbcolor
```

oú `r g b` sont les coefficients (rouge, vert, bleu) de la couleur souhaitée. Par exemple pour obtenir un triangle rempli en rouge avec un contour vert, on remplace `1 setgray` par `1 0 0 setrgbcolor`, et on remplace `0 setgray` par `0 1 0 setrgbcolor`.

On peut aussi définir la largeur des lignes tracées à l'aide de l'instruction **setlinewidth** :

```
0 setlinewidth
```

Attention : les coordonnées visibles dans une page A4 PostScript sont comprises entre 0 et 590 pour la coordonnée horizontale et entre 0 et 840 pour la coordonnée verticale. Lors de la sortie PostScript, vous serez donc amené à modifier les coordonnées des sommets de l'objet, afin que celui-ci rentre correctement dans ce cadre. Utilisez la même modification le long des deux axes, afin de ne pas déformer les proportions de la triangulation originale.

Voici un exemple de fichier PostScript complet dessinant 2 triangles (le 2ième triangle venant obscurcir en partie le premier) :

```
%!PS
0 0 moveto
100 0 lineto
20 100 lineto
0 0 lineto
gsave
1 setgray
fill
0 setgray
grestore
stroke
10 10 moveto
200 10 lineto
100 100 lineto
10 10 lineto
gsave
1 setgray
fill
grestore
0 setgray
stroke
showpage
```