Compte Rendu du Projet 1, Algo 1
"*Visualiseur 3D*"

Equipe n° 9, Mehdi BAHRI et Jérôme LAPOSTOLET, Groupe 4

25 Octobre 2013

Nous avons réussi à réaliser, comme demandé, un visualiseur opérationnel qui fonctionne sur un fichier STL au format ASCII respectant les spécifications.

1 Implémentation

1.1 Implémentation du paquetage `Algebre`

Nous avons implémenté les fonctions décrites dans le sujet en passant, lorsque cela nous semblait nécessaire, par des fonctions intermédiaires.
Ainsi nous avons réalisé une fonction `Produit_Matrice` pour des matrices de tailles quelconques afin qu'il puisse être utilisé à toute échelle. Cependant sa complexité en temps n'est pas optimisée. Cela ne nous a pas semblé une priorité puisqu'il n'est utilisé ici que sur des matrices de taille "3,3".
Nous avons également réalisé une surcharge de l'opérateur "-" pour les vecteurs ainsi qu'une fonction générant la matrice d'une rotation en dimension 3 à partir d'un caractère représentant l'axe autour duquel se fait ladite rotation et de l'angle de cette dernière.

1.2 Implémentation du paquetage `Scene`

Nous avons suivi les descriptions des fonctionnalités telles que décrites dans le sujet. Le seul point pouvant être considéré comme une initiative de notre part est l'ajout d'une matrice `Matrice_Rotations_Inverses` dans la procédure `Modification_Matrice_Rotation` afin de pouvoir ensuite l'utiliser lors des projections.

1.3 Implémentation du paquetage `STL`

Pour la fonction `Nombre_Facettes` nous avons choisi de compter le nombre de `endfacet` du fichier STL. Pour cela nous récupérons les lignes une par une et pour chacune nous ne conservons que les caractères de 'a' à 'z'. Notre choix s'est porté sur l'utilisation de chaînes de longueur non définie grâce au paquetage `Ada.Strings.Unbounded`. Ce choix permet de ne réserver que la mémoire nécessaire et ce, sans ajouter de compteur ou de test. Nous comparons ensuite la chaîne obtenue avec `endfacet`.
La fonction `Chargement_ASCII` est celle qui semble laisser le plus de choix quant à la méthode utilisée pour la traiter. Nous avons d'ailleurs perdu un certains temps en hésitant sur la méthode à choisir.
Finalement nous avons choisi de traiter directement le fichier dans son ensemble et non de prendre les lignes une par une.
Tout d'abord nous recherchons un "v". Ensuite nous comparons la chaîne suivant ce caractère à `ertex`. Si le test est positif, nous sommes assurés d'être sur un `vertex` et nous récupérons alors 3 flottants que nous enregistrons dans la facette. La récupération des flottants est faite avec un `Get(Fichier, variable)` qui permet de récupérer l'élément voulu directement dans le fichier et est présent dans les bibliothèques d'Ada, ce qui présente l'avantage d'être standard et optimisé. La variable `Point` gérée avec un modulo 3 nous permet de répéter trois fois cette opération en changeant le point de la facette dans lequel on enregistre les flottants.

1.4 Implémentation du paquetage `Frame`

Dans ce paquetage, nous avons rajouté une boucle qui permet de nettoyer l'écran de l'affichage précédent et ainsi ne pas obtenir une superposition des différentes vues de l'objet. Ensuite nous avons implémenté les fonctions décrites de projection de la facette dans le plan de visualisation et de tracé des triangles correspondants à ces facettes.

1.5 Modification des paquetages `Dessin` et `Ligne`

Dans le paquetage `Dessin`, nous avons modifié le type des coordonnées de `Natural` à `Integer` afin de pouvoir gérer assez simplement les dépassements d'écran. En effet nous testons l'appartenance de la coordonnée (éventuellement négative donc) à la zone d'affichage pour décider si l'on doit tracer le segment entier ou seulement une partie de celui-ci. Cependant les tests induits par cette modification ralentissent significativement l'affichage dans certaines situations. Le rendu s'en trouve toutefois bien amélioré, l'affichage étant plus complet.
De plus nous avons dû effectuer le même changement de `Natural` en `Integer` dans le paquetage `Ligne` qui appelle les procédures de `Dessin`.

2 Validation

2.1 Test du paquetage `Algebre`

Nous avons testé les différentes fonctions au fur et à mesure de leur implémentation. Par exemple nous avons testé les différents opérateurs en les appliquant à des matrices ou vecteurs quelconques et en vérifiant les résultats à l'aide de calculs manuels ou d'outils informatiques. Quant aux matrices de rotations nous avons également vérifié leur forme en les testant pour différents angles.
Seule la fonction `Projection` semblait difficile à réellement tester indépendamment et nous avons donc attendu de pouvoir afficher des modèles pour voir si elle fournissait des résultats cohérents.

2.2 Test du paquetage `STL`

Nous avons simplement testé les deux fonctions du paquetage sur les deux fichiers STL fournis dans le projet. Pour `Nombre_Facettes` nous avons vérifié les résultats fournis en effectuant un `grep -c` de `endfacet` sur les fichiers. Pour `Chargement_ASCII`, nous avons fait afficher les facettes récupérées et nous en avons comparé quelques unes choisies aléatoirement avec le fichier STL traité.
Pour plus de sécurité nous avons d'abord implémenté ces fonctions dans un parseur indépendant avant de les incorporer au paquetage.

Les paquetages `Scene` et `Frame` ne nous semblaient pas testables indépendamment, nous avons donc attendu d'avoir un affichage pour pouvoir les déboguer.

2.3 Test du programme `Visualiseur`

Nous avons testé le programme sur les deux fichiers STL fournis dans le projet. Ceci nous a permis dans un premier temps de déboguer notre code afin d'avoir effectivement les fonctionnalités recquises par le sujet. Ensuite nous avons essayé d'améliorer la gestion de certains cas comme lorsque l'objet dépasse de l'écran (voir Modification des paquetages `Dessin` et `Ligne`) afin d'obtenir un affichage le plus détaillé et cohérent possible, et ce quelle que soit la position de la caméra.

Cependant, par manque de temps, nous n'avons pu améliorer la gestion de quelques situations qui font apparaître des segments non voulus. Nous n'avons pas pu non plus limiter la perte de performance due aux nombreux tests rajoutés dans la gestion de ces dépassements d'écran en implémentant par exemple une gestion plus générale des collisions entre segments et bords de l'écran.