Auteurs : Strebler Olivier & Scheidel Arthur L3S5

Projet POO Automne 2014
Compte-rendu

1) Choix de modélisation et précisions supplémentaires

Nous avons écrit le projet en langage C++ en utilisant la bibliothèque graphique Qt et la bibliothèque 3D OpenGL. Nous organisons notre programme en trois fenêtres : **une fenêtre principale** qui comporte les fonctions de chargement des deux images stéréoscopiques et un bouton pour lancer la reconstitution 3D avec calcul de distance ; **une console** démarrée à coté, en même temps que la fenêtre principale, qui servira à afficher des informations complémentaires comme la taille des images chargées, des précisions sur les éventuelles erreurs rencontrées (pas de point de repère trouvé, taille des deux images différentes, etc.), les divers informations calculés par la reconstitution 3D, etc. ; **une fenêtre OpenGL** pour l'affichage de la scène 3D reconstituée (qui va donc afficher un nuage de points segmenté et leur distance respective à l'objectif en cm).

La fenêtre OpenGL ne s'ouvre qu'une fois la reconstitution lancée et terminée. Elle peut se refermer et se recréer à tout moment sans quitter le programme principal.

Pour que les images stéréoscopiques soient exploitables par notre programme, elles doivent comporter au moins deux points de repère muraux qui seront des petits carrés (d'au moins 4*4 pixels) rouges (255, 0, 0) pour indiquer une distance de référence connue sur le mur (la distance entre ces deux points rouges) et un nombre arbitraire de petits carrés verts (0, 255, 0) pour symboliser les points de repérage des objets à analyser, nombre qui doit être identique sur les deux images.

Pour calculer la distance des points de repérage à l'objectif, nous avions besoin de connaître la distance de l'objectif au mur, et nous l'avons fixé (dans un *#define*) à 200cm (modifiable directement dans le code).

2) Classes et organisation générale du code

Nous avons programmé sur QtCreator qui utilise un créateur graphique de fenêtre (QtDesigner) qui en génère automatiquement le code (technique utilisé pour dessiner la fenêtre de la console et celle de la fenêtre principale). Ce sont les fichiers du type *ui_peuimporte.h* , inclus dans les fichiers écrits par nos soins. Il ne s'agit que d'un descripteur physique de l'apparence des fenêtres, et ne gère en rien leur comportement.

Nous avons donc un code découpé en 4 modules (chaque module correspondant à un fichier d'entête (*.h*) associé à un fichier de définition de même nom (*.cpp*)) :

- **Le module principal** : Il coordonne les 3 autres, s'occupe de charger les images, d'en afficher un aperçu et de lancer la reconstitution 3D. Ce module est représenté par l'entête *imagewindow.h* et comporte une seule classe nommée *ImageWindow*.

- **Le module console** : Il gère la console, son affichage formaté et propose une interface agréable pour y afficher différents types d'informations (infos principales ou complémentaires, messages d'erreur, etc.). Fichier : *infoconsole.h*, une seule classe : *InfoConsole*.

- **Le module d'algorithmie** : Il s'occupe de retrouver les points de référence dans les deux images et de calculer deux éléments : tout d'abord un nuage de points affichable dans un espace OpenGL (cube où x, y et z sont inclus dans [-1 ; 1]) puis la distance de chaque point de repérage à l'objectif. Le fichier de ce module est *algorithm.h* et comporte deux classes : *Point3D* et *Algorithm.*

- **Le module 3D** : Il va recevoir les informations calculées dans le module d'algorithmie et va afficher la scène 3D de la reconstitution (en construisant les bons segments entre les points). Fichier : *glwindow.h*. Classe : *GLWindow*.

Il n'y a pas d'héritage supplémentaire que celui imposé par Qt pour organiser du code utilisant cette bibliothèque (*ImageWindow* héritant de *QMainWindow*, etc.).

3) Algorithmes principaux

Dans cette section nous allons brièvement résumer les algorithmes exécutés par les deux modules respectivement d'algorithmie et de 3D car les algorithmes présents dans le module principal n'ont rien d'intéressant : ouverture d'image, connexion des signaux aux slots, gestion des boites de dialogue, etc. De même pour le module console. Simplement, pour le stockage des images et leur manipulation, on utilise la classe *QImage*.

- Module d'algorithmie :

Entrée : Les deux images stéréoscopiques

Sortie : Un nuage de points 3D (x, y, z) avec x, y et z inclus dans [-1 ; 1] (donc affichable par OpenGL) accompagné pour chaque point de sa distance réelle en cm par rapport à l'objectif.

Principe :

1 : On commence par trouver les deux points rouges du mur sur chaque image et les points verts de repérage. Pour cela, on balaye chaque image pixel par pixel verticalement de gauche à droite et de haut en bas. Vu qu'entre les deux images, on suppose qu'il n'y a eu que translation horizontale de l'objectif vers la gauche ou vers la droite, on peut intelligemment en déduire que l'ordre dans lequel les différents points seront trouvés sur l'une et l'autre image est conservé. En clair, si on stocke dans respectivement deux tableaux les points de repérage trouvés sur les deux images, on peut les associer deux à deux en miroir, sans avoir besoin d'une fonction précise qui va tenter de retrouver le point correspondant dans l'autre image à partir des couleurs alentours et d'une estimation par corrélation maximale de la zone environnante la plus ressemblante.

2 : A partir du ration pixel / cm que l'on peut calculer avec la référence murale, on va déterminer une distance fictive entre les deux instances dans chaque image de chaque point de repérage. Cette distance est proportionnel à l'éloignement relatif de chacun de ses points entre eux dans un espace 3D. On convertie ensuite les données x, y, z calculées en coordonnés utilisable par OpenGL. On complète ces (x, y, z) par une information de distance réelle du point de repérage en question à l'objectif (en nous servant de la distance supposée connue de l'objectif au mur). On renvoie ces points (x, y, z, distance) au module 3D.

- Module 3D :

Entrée : Un nuage de points 3D (x, y, z) accompagné de leur distance réelle à l'objectif en cm.

Sortie : Une fenêtre OpenGL affichant ce nuage de point intelligemment relié par des segments, accompagné, pour chaque point, de l'information de la distance à l'objectif.

Principe : On parcours la liste des quadruples (x, y, z, distance) en supposant que l'on a affaire à une structure cubique (cela permettra de déterminer les segments à placer). On imprime les segments et l'information, pour chaque point, de sa distance à l'objectif, en léger décalé par rapport au point en question.

3) Répartition du travail

Théorie schématique et mathématique : Arthur Scheidel et Olivier Strebler
Implémentation C++ avec Qt et OpenGL : Olivier Strebler
Écriture du rapport : Arthur Scheidel et Olivier Strebler