Master 2 TSI - Option Programmation

TP 2 de Réalité Augmentée

(1) Nom, Prénom :
(2) Nom, Prénom :

Au cours de ce TP, nous développerons une application simple de réalité augmentée pour afficher des éléments 3D dans un environnement libre.

1 ORB_SLAM2

Instructions préliminaires : Téléchargez et installez Pangolin en utilisant le dépôt git :

- `git clone https://github.com/stevenlovegrove/Pangolin.git`
- `cd Pangolin`
- `mkdir build && cd build`
- `cmake ..`
- `make -j4`
- `sudo make install`

Téléchargez le code d'ORBSLAM2 en utilisant le dépôt git :

- `git clone https://github.com/npiasco/ORB_SLAM2.git`

Compilez le code en exécutant la commande `build.sh`.

1. Modifiez le fichier de configuration `Examples/default.yaml` en y inscrivant les paramètres intrinsèques de votre caméra (ceux calculés lors du dernier TP).
2. Vous pouvez lancer l'algorithme de mapping grâce à l'exécutable `Examples/orbslam`. Les arguments attendus sont un dictionnaire de *visual worlds* (`Vocabulary/ORBvoc.txt`) et le fichier de configuration précédemment rempli.
3. Testez les limites du SLAM en effectuant différents mouvements de caméra. Testez également plusieurs *environnements* : votre bureau, le sol, le plafond, etc. Citez deux cas d'échecs de l'agorithme :

—
—

2 SLAM & OpenGL

Instructions préliminaires : téléchargez le code du TP 2 en utilisant le dépôt git :

— `git clone https://github.com/npiasco/ENSG_AR_TP2.git`
— `cd ENSG_AR_TP2`

Modifiez dans le `CMakeLists.txt` le chemin vers votre dossier ORB_SLAM2. A l'intérieur du dossier, générez le **Makefile** du projet :

— `mkdir build && cd build`
— `cmake ..`

Compilez le projet :

— `make -j4`

1. Le fichier `main.cpp` a une architecture similaire à celui du TP 1 : modifiez les paramètres de la matrice de projection de la caméra OpenGL pour qu'ils correspondent à celui de votre caméra.

A chaque appelle de la méthode `cv::Mat SLAM::TrackMonocular(cv::Mat image, int timeStamps)`, ORB_SLAM met à jour son état interne et renvoie sous forme de matrice `cv::Mat` la pose de la caméra par rapport à un repère fixe (qui correspond à la première *keyframe* enregistrée).

Attention : la matrice de pose de la caméra sera vide dans le cas ou l'algorithme ne s'est pas encore initialisé ou a perdu le tracking.

2. Nous allons utiliser la pose renvoyer par le SLAM à la place de la matrice view d'openGL. Quel est la transformation entre le repère de la caméra et le repère openGL (= origine en bas à gauche de l'écran, x vers la droite et y vers le haut) ?
3. Créer cette matrice de transformation en utilisant la fonction `glm::mat4 glm::rotate(glm::mat4 m_init, GLfloat angle_rad, glm::vec3 vect_rot)`.
4. Remplacer la matrice `view` d'OpenGL par la position de la caméra calculée par le SLAM dans le repère d'openGL. On oubliera pas d'appliquer le changement de repère entre la caméra et le repère OpenGL : `view = matrice_tf * pose_slam;`.

Note Vous pouvez utiliser la fonction `void fromCV2GLM(cv::Mat cvmat, glm::mat4 glmmat)` qui permet de passer d'une matrice openCV à une matrice glm.

Vous constatez maintenant que la caméra se déplace dans la scène virtuel comme dans le monde réel.

3. Faites en sorte que le rectangle qui a pour texture l'image de la caméra se situe en permanence devant la caméra virtuelle.
4. Affichez un objet à la position $(0, 0, 0)$ de la scène virtuelle.

3 Augmentation virtuel de l'environnement

La première incrustation virtuelle que nous venons de réaliser ne prend pas en compte l'environnement extérieur mais simplement les mouvements intrinsèques de l'utilisateur. Dans cette partie, nous allons exploiter la carte créée par ORB_SLAM pour ancrer dans l'environnement des modèles 3D de façon plus réaliste.

Pour récupérer les points triangulés par l'algorithme de mapping, nous disposons de la méthode `vector <MapPoint*> SLAM::GetMapPoints()`. Nous pouvons ensuite itérer sur la structure de donnée renvoyée par

la méthode et accéder aux coordonnés du points par la méthode `cv::mat GetWorldPos()` de la classe `MapPoint`.

1. Afficher une sphère sur chacun des 50 premiers points 3D calculés par le SLAM. On vérifiera bien que la *map* possède suffisamment de points.

Pour donner plus de réalisme à nos incrustations virtuelles, nous allons maintenant introduire du mouvement dans notre monde artificiel.

2. Faites apparaître un modèle sur l'un des points 3D de l'environnement, puis faite le progressivement changer de taille ; grossir puis diminuer de volume de façon periodique.
3. Appliquez le mouvement de la question précédente à plusieurs points. On initialisera de façon aléatoire le fréquence de modification de chacun des objets afin de rendre l'ensemble plus `naturel`.
4. Faites apparaître un modèle 3D en face de vous puis faites le se diriger vers l'un des points 3D de l'environnement. Une fois que le modèle à atteint ce point, fait le se diriger aléatoirement vers un autre point proche.

4 Conclusion

1. Quel(s) est/sont le(s) avantage(s) de la méthode vue au premier TP par rapport à celle vue au second ?

2. Quel(s) est/sont le(s) avantage(s) de la méthode vue aujourd'hui par rapport à celle du premier TP ?