Master 2 TSI - Option Programmation

TP de Réalité Augmentée

Au cours de ce TP, nous développerons une application simple de réalité augmentée pour afficher des éléments 3D à l'aide de tags imprimés.

Instructions préliminaires : téléchargez le dossier de code en utilisant le dépôt git :

— `git clone https://github.com/npiasco/ENSG_AR.git`

Compilez séparément la librairie apriltag (lire le `README` dans `ENSG_AR/code/external/apriltag`).

A l'intérieur du dossier `code`, générez le **Makefile** du projet :

— `cd ENSG_AR/code`
— `cmake .`

Compilez le projet :

— `make -j4`

Vous trouverez le support du cours ainsi que l'énoncé du TP dans le dossier `ENSG_AR/cours`.

1 Calibration de la caméra

Problématique : comme nous l'avons vu dans l'introduction précédente, nous allons avoir besoin d'une caméra calibrée pour réaliser notre application de réalité augmentée. Les webcams que nous utilisons présente une *faible* distorsion, on va donc considérer que la phase de calibration servira seulement à déterminer les paramètres intrinsèques de focal et de position du centre optique de notre caméra.

1. Branchez la webcam à votre ordinateur et lancez l'exécutable `calibration` dans le dossier `ENSG_AR/code/opencv_code/calib/` pour procéder à la calibration.

— fx =
— fy =
— cx =
— cy =

Note : si l'autofocus de la webcam est fonctionelle, vous pouvez le désactiver avec la commande suivante `v4l2-ctl -c focus_auto=0`(package `v4l-utils`).

Le champ de vue d'une caméra (FOV) est défini de la manière suivante :

$$FOV_{digonal} = 2 * atan\left(\frac{d}{2 * f}\right)$$

Avec :

— d = la taille de la diagonal du capteur
— f = la distance focale

2. En prenant en compte les paramètres de calibration obtenus, calculez le champ de vue de votre caméra.

— FOV horizontal =
— FOV vertical =

3. Proposez une méthode pour estimer expérimentalement le FoV de votre caméra. La focal nouvellement calculée correspond-t-elle à celle obtenue par calibration ?

2 Prise en main des AprilTags

Les **Apriltags** vont nous servir de balises dans notre repère monde pour positionner de façon précise les éléments 3D que nous allons rajouter à l'environnement.

1. Lancez le programme `mainCV` situé dans le dossier `ENSG_AR/code/opencv_code/` pour vérifier que les tags sont bien détectés.

On a mis à votre disposition une classe `AprilTagReader` dans le dossier `ENSG_AR/code/opencv_code/`.

2. Lisez la documentation présente dans le fichier `AprilTagReader.h`. Écrivez un programme dans le fichier `AprilTagReaderTest.cpp` permettant d'afficher la position des tags détectés par la classe.
3. En vous basant sur vos observations, dessinez le repère associé à la caméra selon la classe `AprilTagReader` :

3 Simulation de la caméra dans OpenGL

Nous allons écrire le code principale de notre programme dans le fichier `main.cpp` présent dans le dossier `ENSG_AR/code`. Il s'agit d'un code reprenant la structure d'un code **OpenGL** : en effet *la boucle principale* de notre application de réalité augmentée se situera dans la boucle habituel d'OpenGL.

1. Familiarisez vous avec la structure du code OpenGL. Les fichiers `Model.h` et `Shader.h`/`Shader.cpp` permettent respectivement de charger des modèles 3D et les shaders du programme.

Nous allons maintenant simuler une caméra dite *projective* (à la différence d'orthogonale) qui aura les même paramètres que notre webcam, ainsi le rendu de nos objets 3D dans la scène capturée par la caméra sera réaliste.

Pour afficher le flux vidéo dans OpenGL, nous allons coller l'image (que l'on récupère grâce à la classe `AprilTagReader`) en tant que texture à un rectangle composé de 4 vertices. Cette méthode est loin d'être optimal mais pour des images de dimension pas trop grande (on limitera la résolution de l'image à $640 * 480$) cela reste faisable. Lancez le programme `main` pour voir le résultat.

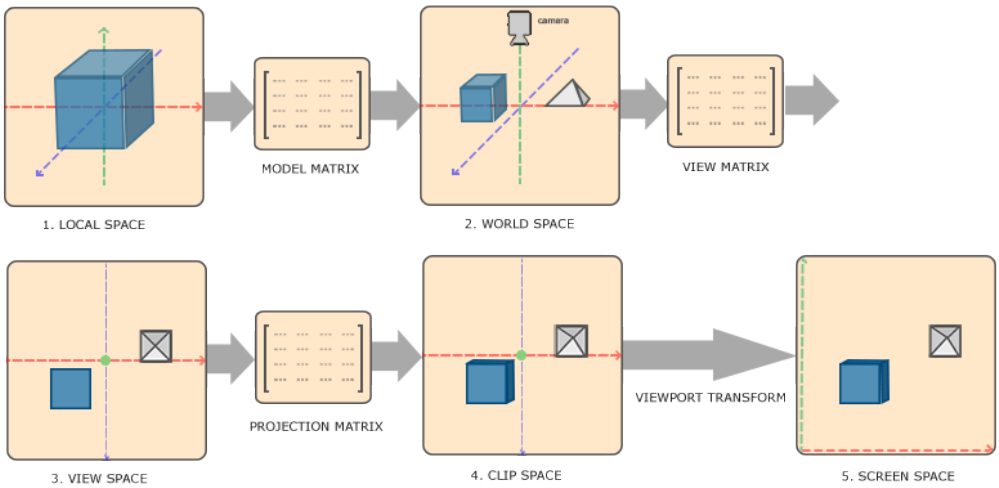

FIGURE 1 – Rappel des transformations successive avant l'affichage à l'écran de notre scène OpenGL

La figure ci-dessus rappelle les transformations successive opérée sur nos objets 3D avant le rendu dans notre fenêtre OpenGL. Dans le cas particulier de notre application de réalité augmentée nous devons fixer à l'initialisation : la position de la caméra (matrice `view`), de l'image représentant la texture (matrice `model`) et la matrice de projection (`projection`).

2. A l'aide de la fonction `glm::mat4 glm::perspective(GLfloat HFOV, GLfloat aspect_ratio, GLfloat near_plane , GLfloat fare_plan)`, créez une matrice de projection simulant les paramètres de votre webcam. On oubliera pas de la transmettre aux shaders une fois créée !

On constate maintenant que les dimensions et la position de "l'image" n'est pas optimal pour notre application : on aperçoit les bords de la texture dans la fenêtre OpenGL.

3. Déterminez la taille et la position idéale pour notre image.
4. Modifier les parties correspondantes dans le code pour que l'image prenne l'ensemble de l'espace visible de la fenêtre OpenGL.

4 Affichage des objets dans l'environnement augmenté

Maintenant que notre flux vidéo a été incorporé dans OpenGL, nous pouvons facilement intégrer des modèles 3D pour augmenter notre environnement. L'utilisation d'Apriltag va nous permettre de positionner nos éléments dans notre scène OpenGL.

Attention : Le repère associé à la caméra (celui dans lequel sont exprimé les positions des tags) est différent du repère d'OpenGL. Le repère d'OpenGL à son origine en bas à gauche de l'écran, $\vec{x}$ pointant à droite et $\vec{y}$ en haut.

1. Quelle est la rotation relative permettant de passer du repère de la caméra au repère d'OpenGL ?

2. En récupérant la position d'un tag, créez la matrice `model` traduisant la position de ce tag dans le repère d'OpenGL.

Attention : les matrices `glm` d'OpenGL ont la convention inverse de celle utilisé par OpenCV pour stocker leur élément : `mat[i][j]` fait référence à l'élément de la i^{eme} colonne, j^{eme} ligne. Pour créer une matrice de transformation homogène pour effectuer une rotation utilisez la commande suivante : `glm::mat4 glm::rotate(glm::mat4 m_init,(GLfloat)angle_rad, glm::vec3 vect_rot)`.

La classe `Model` permet de charger et d'afficher facilement des modèles 3D (.obj, .3ds, .stl, etc.). Pour charger un modèle indiquez simplement le chemin vers le modèle à charger comme argument du constructeur, pour l'afficher utilisez la méthode `void Draw(Shader shader)`.

3. Essayez d'afficher le singe de blender suzanne (`ENSG_AR/code/opengl_code/model/suzanne.obj`) sur l'apriltag.

Attention : vous devez rafraichir le buffer de profondeur avec la commande `glClear(GL_DEPTH_BUFFER_BIT )` au bon moment dans le code pour être sur de voir votre modèle (vu qu'il sera surement positionné *derrière* votre image).

4. Essayez d'appliquer des transformations de mise à l'échelle et de rotation sur votre modèle 3D afin d'obtenir un affichage de réalité augmentée *réaliste* . Pour créer une matrice de transformation homogène pour effectuer une mise à l'échelle utilisez la commande suivante : `glm::mat4 glm::scale(glm::mat4 m_init, glm::vec3 vect_scale)`.
5. Essayez d'importer le modèle que vous avez créé avec `QGIS` lors des TPs précédents.

5 Aller plus loin

Félicitations, votre application de réalité augmentée est maintenant fonctionnelle ! Nous vous proposons dans cette partie de l'améliorez en y rajoutant différents modules. Vous pouvez choisir ceux que vous voulez implémenter en priorité.

5.1 Affichage de différents objets

Les apriltags nous permettent d'une part de connaître la position dans l'espace d'une feuille de papier à partir d'une caméra monoculaire, d'autre part de lire le numéro associé à la cible détectée. Modifiez votre code pour permettre l'affichage simultané de différents modèles en fonction du numéro du tag.

5.2 Positionnement de la source lumineuse

Pour un rendu plus réaliste, nous avons pour l'instant éclairé nos modèles (`lightShader`) en considérant la position de la source lumineuse confondue avec celle de la caméra. Modifiez les shaders et le code pour associer la position de la source lumineuse à un tag donné.

5.3 Persistance des modèles

Actuellement, la moindre occultation du tag provoque la disparition du modèle associé. Intégrez dans votre code un moyen de continuer l'affichage du modèle même si le tag n'est plus visible un court instant à la dernière position connue.

5.4 Interface augmentée

L'une des utilisations de la réalité augmentée est de créer des interfaces plus ergonomique pour l'utilisateur. Créez un "bouton" virtuel à l'aide d'un tag qui contrôlera l'affichage d'un objet 3D associé à un autre tag. Par exemple on peut imaginer contrôler la taille d'un objet afficher sur un tag en modifiant l'angle de rotation d'un autre tag.