TP VSION - Réalité Augmentée

Yvan Mezencev

Le code est disponible sur GitHub avec ce lien.

I. Compilation de ARUCO

Les étapes décrites permettent de compiler la bibliothèque ArUco. Les fichiers générés sont placés dans le dossier build

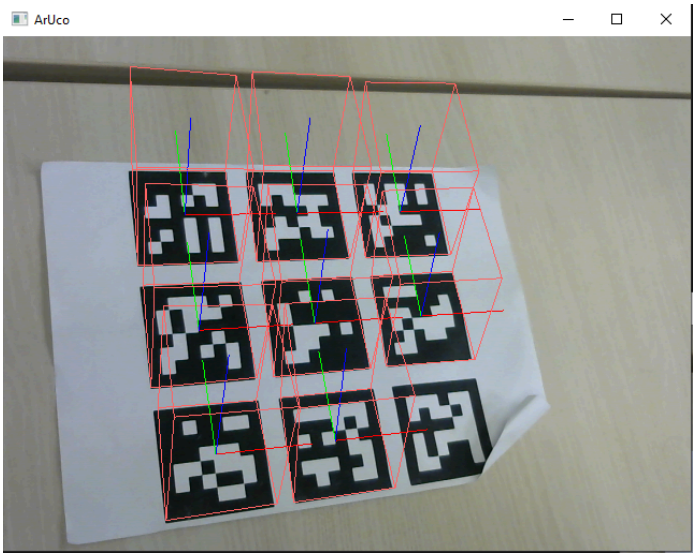
Cmake me paraît particulièrement utile lorsque l'on souhaite compiler un projet sur plusieurs plateformes différentes, ou lorsque le projet a un grand nombre de dépendances.

II. Première Augmentation

Pour compiler cette première application, on télécharge le projet fourni sur Hippocampus, ainsi que la librairie GLFW. On place cette librairie dans le dossier C:/Dev comme les autres dépendances.

Enfin, on ajuste les propriétés du projet sur visual studio en s'aidant du fichier config.txt. Il faut aussi penser à ajouter le dossier C:\Dev\Aruco\bin au PATH.

Une fois configuré, le programme fonctionne et les marqueurs sont détectés:



Explication de la méthode ArUco::drawScene():

La méthode commence par vérifier si une image a été chargée et réinitialise les matrices de modelview et de projection
Elle configure ensuite la vue (orthographique) et la fenêtre.
L'image capturée par la caméra (stockée dans m_ResizedImage) est affichée pixel par pixel comme fond de la scène.

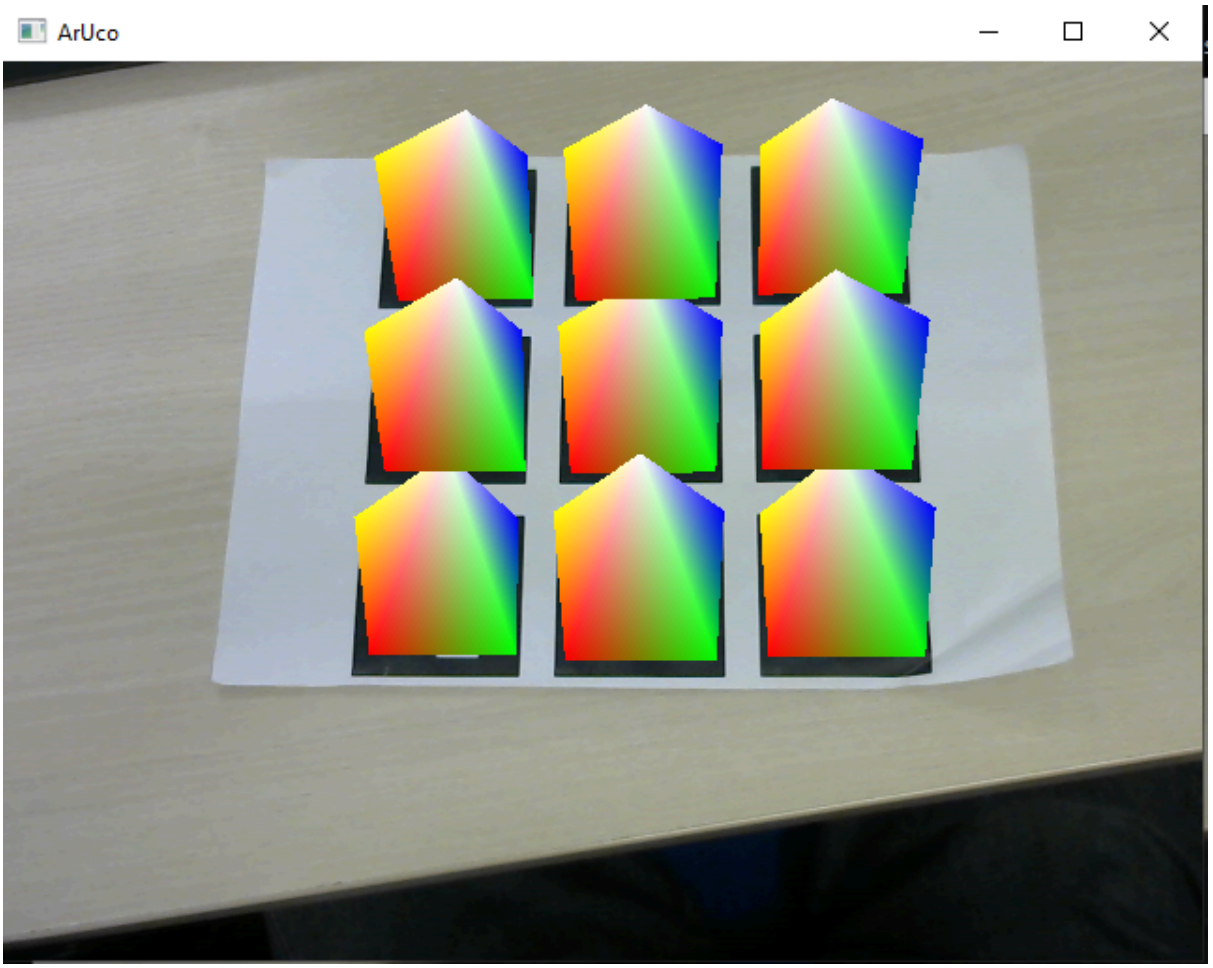
Ensuite, on charge la projection calculée par ArUco. Elle permet de convertir les coordonnées 3D des objets détectés dans le repère 2D de la caméra.

Puis, pour chaque marqueur:

- On se place dans le repère du marqueur
- On trace les axes de ce repère
- On dessine un cube en fil de fer.

Ajout d'augmentation

J'ai décidé de remplacer les cubes en wireframe par des pyramides colorées.
Le rendu d'une pyramide est géré par la fonction drawPÿramide(), qui est appelée dans drawScene(). Le code de cette fonction est donnée en annexe.
J'ai donné à chaque sommet une couleur différente pour pouvoir les distinguer.



L'augmentation est stable tant que le déplacement de la caméra ou du marqueur est faible.

III. Application de RA

IV. Annexe

```
void ArUco::drawPyramid(float size) {
    float halfSize = size / 2.0f;

    // Dessine la base (un carré)
    glBegin(GL_QUADS);

    // Couleurs des sommets de la base
    glColor3f(1.0f, 0.0f, 0.0f);  // Rouge
    glVertex3f(-halfSize, 0.0f, -halfSize);

    glColor3f(0.0f, 1.0f, 0.0f);  // Vert
    glVertex3f(halfSize, 0.0f, -halfSize);

    glColor3f(0.0f, 0.0f, 1.0f);  // Bleu
    glVertex3f(halfSize, 0.0f, halfSize);

    glColor3f(1.0f, 1.0f, 0.0f);  // Jaune
    glVertex3f(-halfSize, 0.0f, halfSize);

    glEnd();

    // Dessine les 4 faces triangulaires
    glBegin(GL_TRIANGLES);

    // Sommet de la pyramide (couleur unique pour le sommet)
    glColor3f(1.0f, 1.0f, 1.0f);  // Blanc

    // Face avant
    glVertex3f(0.0f, size, 0.0f);         // Sommet
    glColor3f(1.0f, 0.0f, 0.0f);         // Rouge
    glVertex3f(-halfSize, 0.0f, -halfSize);
    glColor3f(0.0f, 1.0f, 0.0f);         // Vert
    glVertex3f(halfSize, 0.0f, -halfSize);

    // Face droite
    glColor3f(1.0f, 1.0f, 1.0f);         // Blanc
    glVertex3f(0.0f, size, 0.0f);         // Sommet
    glColor3f(0.0f, 1.0f, 0.0f);         // Vert
    glVertex3f(halfSize, 0.0f, -halfSize);
    glColor3f(0.0f, 0.0f, 1.0f);         // Bleu
    glVertex3f(halfSize, 0.0f, halfSize);
```

```
    // Face arrière
    glColor3f(1.0f, 1.0f, 1.0f);         // Blanc
    glVertex3f(0.0f, size, 0.0f);         // Sommet
    glColor3f(0.0f, 0.0f, 1.0f);         // Bleu
    glVertex3f(halfSize, 0.0f, halfSize);
    glColor3f(1.0f, 1.0f, 0.0f);         // Jaune
    glVertex3f(-halfSize, 0.0f, halfSize);

    // Face gauche
    glColor3f(1.0f, 1.0f, 1.0f);         // Blanc
    glVertex3f(0.0f, size, 0.0f);         // Sommet
    glColor3f(1.0f, 1.0f, 0.0f);         // Jaune
    glVertex3f(-halfSize, 0.0f, halfSize);
    glColor3f(1.0f, 0.0f, 0.0f);         // Rouge
    glVertex3f(-halfSize, 0.0f, -halfSize);

    glEnd();
}
```