DELON Romain
JÉGOUREL Adrien

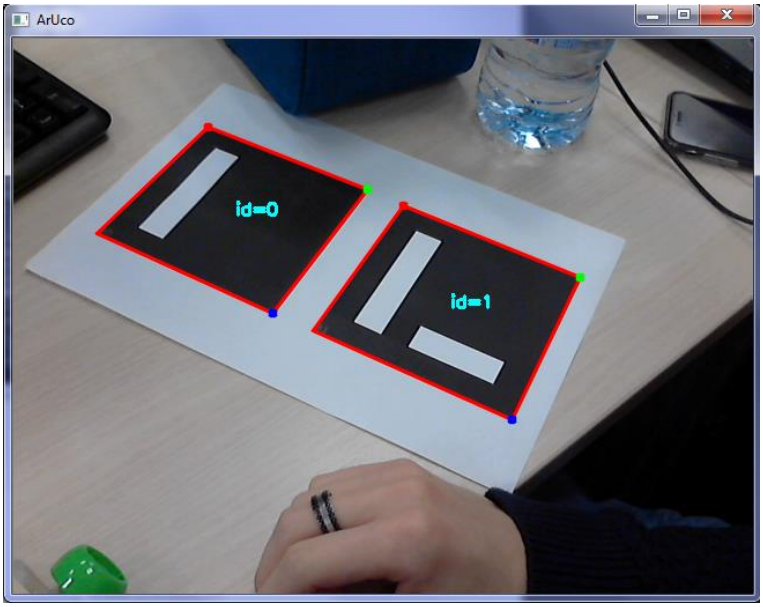
TP VSION – Réalité Augmentée

L'objectif de ce TP est de concevoir une application de RA en C++ à l'aide de deux bibliothèques : OpenCV et ArUco.

I. ArUco : Premier programme

Dans cette partie, l'objectif est de construire une première version de notre application qui détectera simplement des marqueurs ArUco. Les points caractéristiques de ces marqueurs seront par la suite utilisés pour calculer la pose (position et orientation) de la caméra.

Pour ce faire, nous commençons par créer un nouveau projet en y ajoutant ArUco et OpenCV. Dans le nouveau fichier, nous écrivons un morceau de code afin de lire une image et appelons les méthodes de détection des marqueurs proposées par le sujet. L'image d'intérêt utilisée est directement issue du flux de la webcam du PC. Le résultat renvoyé par le programme est visible ci-après.



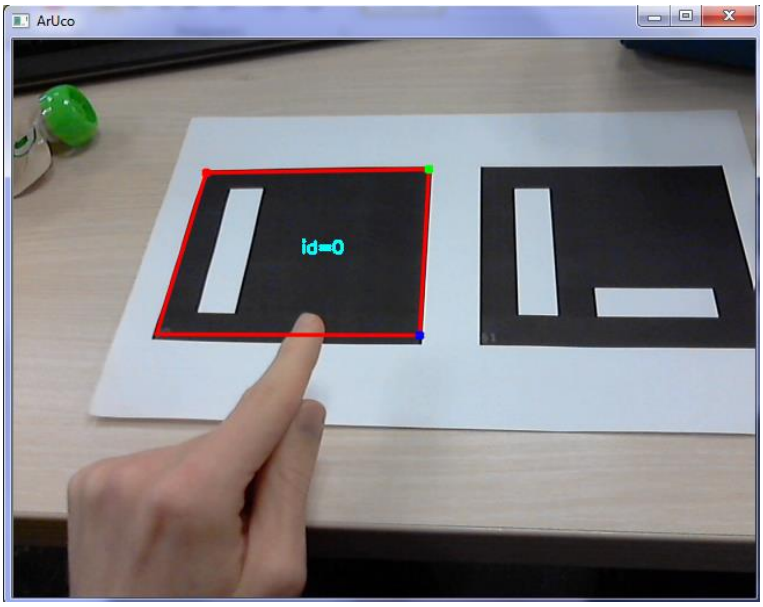
On étudie enfin le comportement d'ArUco vis-à-vis des marqueurs et de leur détection ou non sur l'image. Les différents cas étudiés sont les suivants :

- *Un ou plusieurs marqueurs visibles*

Si un ou des marqueurs sont détectés sur l'image alors on affiche un contour rouge autour de chaque marqueur ainsi que l'id des marqueurs identifiés.

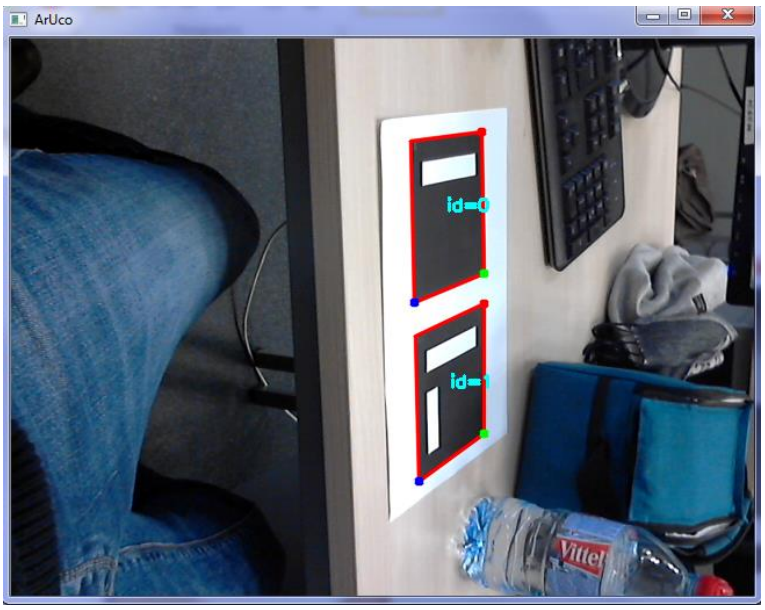
- *Quand un marqueur est plus ou moins visible*

Si un marqueur est partiellement visible alors la précision sur la détection du marqueur baisse (voir marqueur de gauche sur l'image ci-dessous) voire dans un cas plus extrême, il n'est même plus détecté (comme sur le marqueur de droite du fait que le coin en bas à droite soit en dehors du champ de la caméra).



- *Changement d'orientation et de perspective*

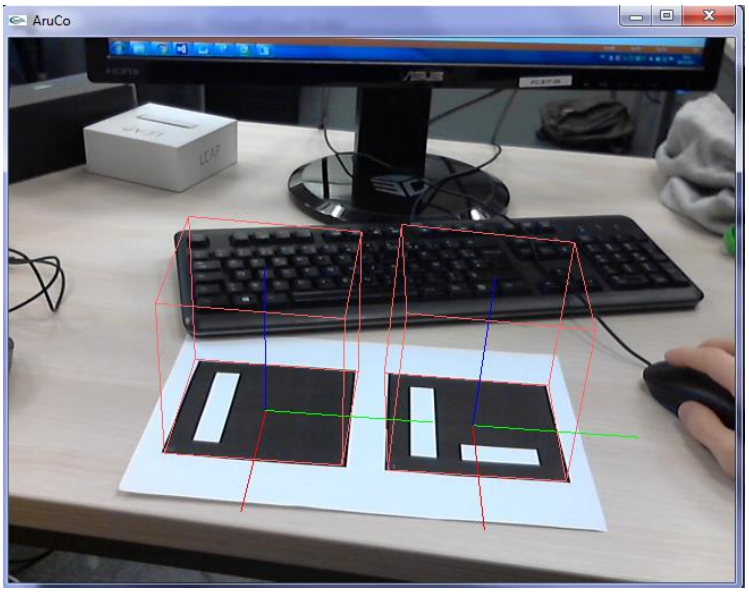
On constate que la détection des marqueurs ArUco est très robuste aux changements d'orientation et de perspective comme en témoigne la photo ci-dessous.



II. Première augmentation

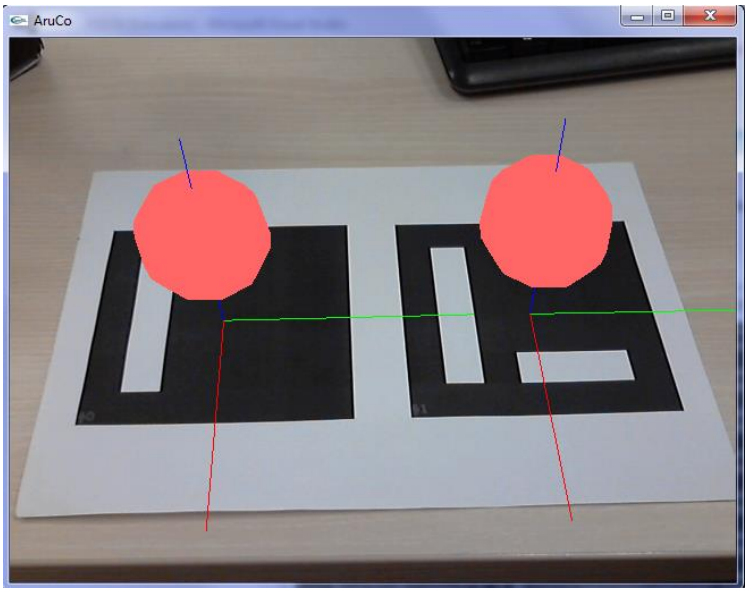
On utilise ensuite une petite application basée sur GLUT pour détecter comme précédemment des marqueurs et générer en plus des augmentations simples. Dans cette optique, on a recours à la méthode *drawScene* d'ArUco afin de dessiner à la fois le fond vidéo et les augmentations.

On commence donc par récupérer le code de l'application et y ajouter le fichier *camera.yml* nécessaire pour le fonctionnement et obtenu lors du précédent TP de calibration. Le programme renvoie alors le résultat suivant :



On s'intéresse à la méthode *drawScene.* Dans un premier temps, on met en place le pipeline graphique. On s'affranchit ensuite des distorsions de la caméra. Puis pour chaque marqueur détecté, on dessine les axes et un cube.

A la place des augmentations de base qui servent à vérifier le bon fonctionnement du programme, on décide de représenter une sphère. Le résultat est visible ci-dessous :



La seule différence entre la méthode *drawScene* initiale et la nôtre se situe ligne 130/131, où l'on affiche une sphère à la place d'un cube.

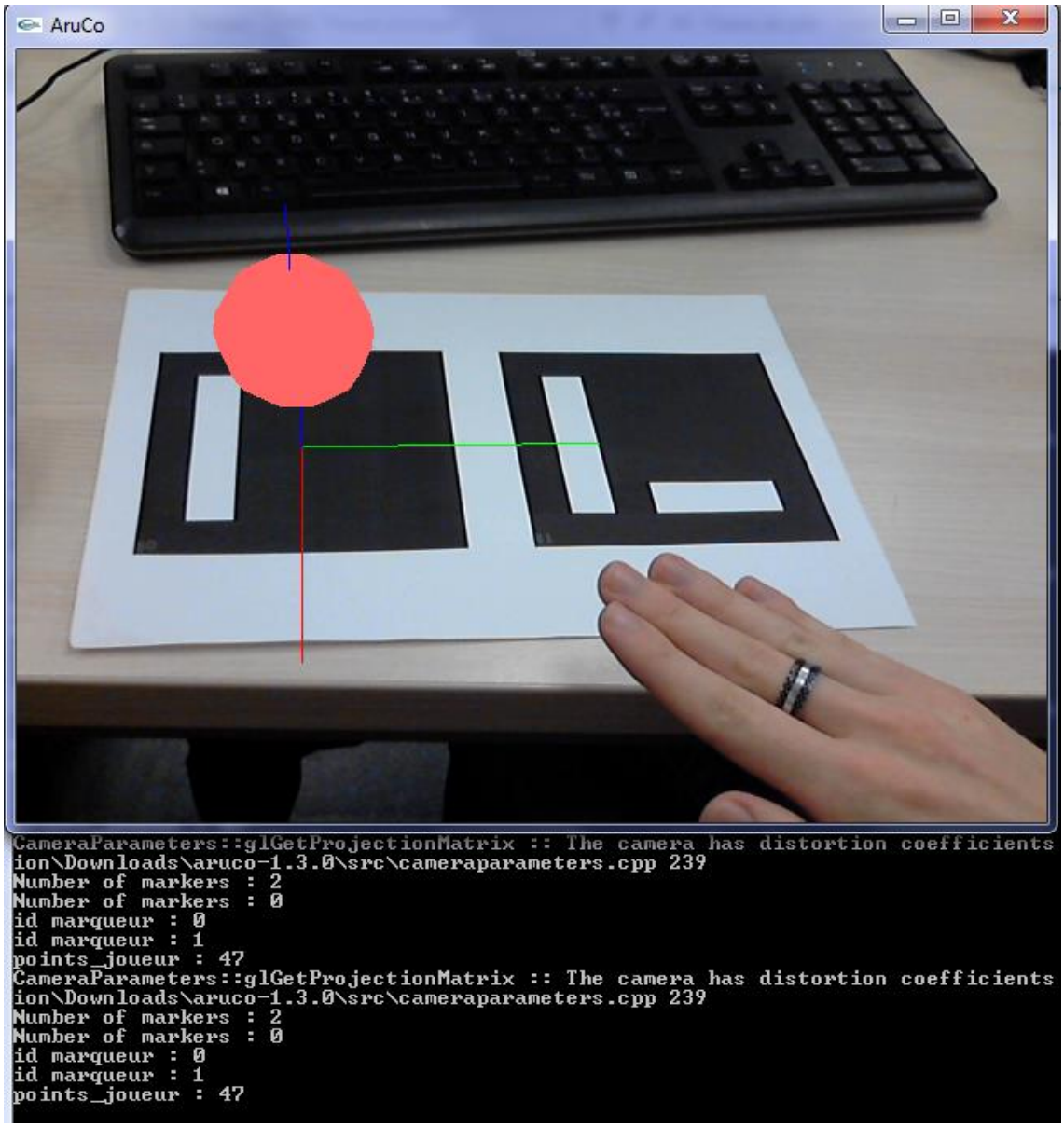
```
106     // For each detected marker
107     for (unsigned int m=0;m<m_Markers.size();m++)
108     {
109        m_Markers[m].glGetModelViewMatrix(modelview_matrix);
110        glMatrixMode(GL_MODELVIEW);
111        glLoadIdentity();
112        glLoadMatrixd(modelview_matrix);
113
114        // Disabling light if it is on
115        GLboolean lightOn = false;
116        glGetBooleanv(GL_LIGHTING, &lightOn);
117        if(lightOn) {
118           glDisable(GL_LIGHTING);
119        }
120
121        // Drawing axis
122        drawAxis(m_MarkerSize);
123
124        // Drawing a cube
125        glColor3f(1,0.4f,0.4f);
126        //glTranslatef(0, m_MarkerSize/2,0);
127        glTranslatef(0, 0, m_MarkerSize/2);
128
129        glPushMatrix();
130        //glutWireCube( m_MarkerSize );
131        glutSolidSphere(m_MarkerSize/4, 10, 10);
132
133        // Re-enabling light if it is on
134        if(lightOn) {
135           glEnable(GL_LIGHTING);
136        }
137
138        glPopMatrix();
139     }
```

III. Application de RA

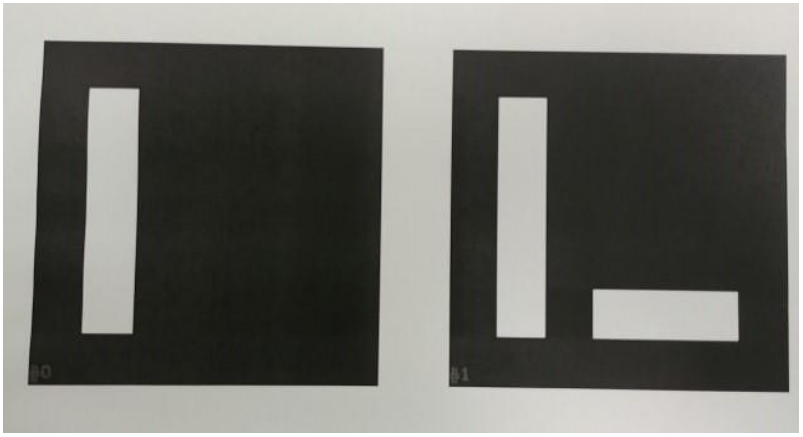
Désormais, on décide de réaliser une application de RA utilisant plusieurs marqueurs. Dans notre cas, il s'agit d'un petit jeu : le jeu de la taupe. L'objectif est ici de frapper sur les taupes (représentées par des sphères) qui apparaissent sur les tags pour gagner des points.

Le principe est donc le suivant : lorsque l'ensemble des marqueurs est visible, on en sélectionne un au hasard sur lequel on affiche une sphère. Le joueur a alors un certain temps pour frapper le tag correspondant (dans le cas contraire on relance automatiquement le choix d'un tag au hasard). Une fois que le marqueur n'est plus visible, c'est-à-dire qu'on a placé sa main sur le bon tag, et que l'ensemble des marqueurs est de nouveau détecté par la caméra, on choisit un autre tag au hasard pour dessiner la sphère.

Un exemple d'affichage renvoyé par l'application est donné ci-après. Dans la console, il est possible de voir son nombre de points évoluer au fur et à mesure que l'on frappe les sphères.



A noter que le programme et les tests ont été effectués avec seulement deux marqueurs d'id 0 et 1. Ainsi, le programme ne fonctionnera pas ou mal avec d'autres configurations (par exemple avec plus de marqueurs). Cependant, il est facile d'étendre le jeu à un nombre plus grand de marqueurs et de le rendre plus complet.



L'un des principaux problèmes de cette technique est qu'il suffit que le tag ne soit plus détecté par la caméra pour que le point soit accordé. Ainsi, si le bras ou tout autre objet passe devant le tag ArUco et obstrue la vision de la caméra, alors il n'est pas nécessaire de frapper le marqueur pour valider le point.

Par ailleurs, pour rejoindre l'idée citée ci-dessus et profiter ainsi pleinement du jeu, il est nécessaire de placer la caméra au-dessus du plateau pour ne pas obstruer d'autres cases que celle désirée par inadvertance.

IV. Conclusion

Ce TP nous a permis d'aborder la réalité augmentée sous l'angle des tags ArUco. Leurs principaux avantages résident dans la simplicité d'utilisation, ainsi que la robustesse et la rapidité de leur détection. Cependant, ils brisent l'immersion et nécessitent une certaine mise en œuvre qui peut-être problématique dans certaines applications : il n'est pas toujours possible de labelliser des objets ou des endroits avec un tag dans la pratique.

Pour notre petit jeu, d'autres pistes d'amélioration ont été envisagées mais le temps nous a manqué pour les réaliser : ajout d'autres sphères de couleurs différentes, avec des temps d'apparition différents et un nombre de points lui aussi différent, améliorer l'affichage du résultat et des sphères (pour les remplacer par des taupes comme dans le vrai jeu), ajouter plus de tags, limiter le temps d'une partie...