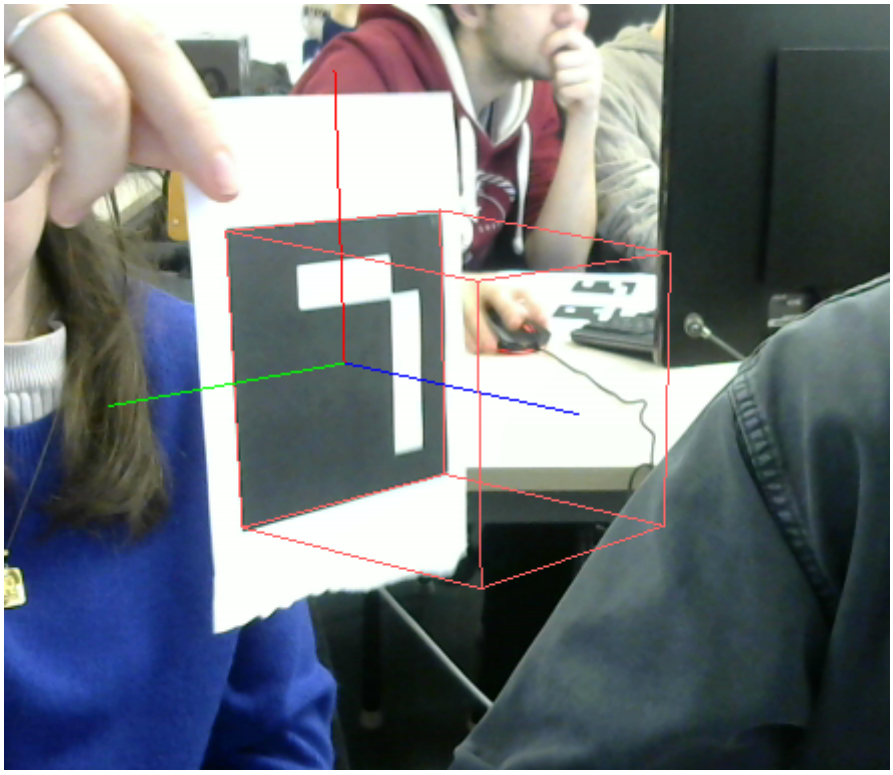
# Compte Rendu TP VSION

Que se passe t-il lorsque ?

- **on a un ou plusieurs marqueurs** : arUco détecte correctement chaque marqueur lorsqu'ils sont ajouté à l'image.
- **quand un marqueur est plus ou moins visible** : dès qu'on recouvre ne serait-ce qu'un tout petit peu le marqueur, la détection s'arrête.
- **l'angle de vue de la caméra est changé** : si on tourne la caméra ou les marqueurs, la détection reste relativement stable et correcte.
- **la taille du marqueur change** : la détection suit bien le marqueur

## 2. Première augmentation

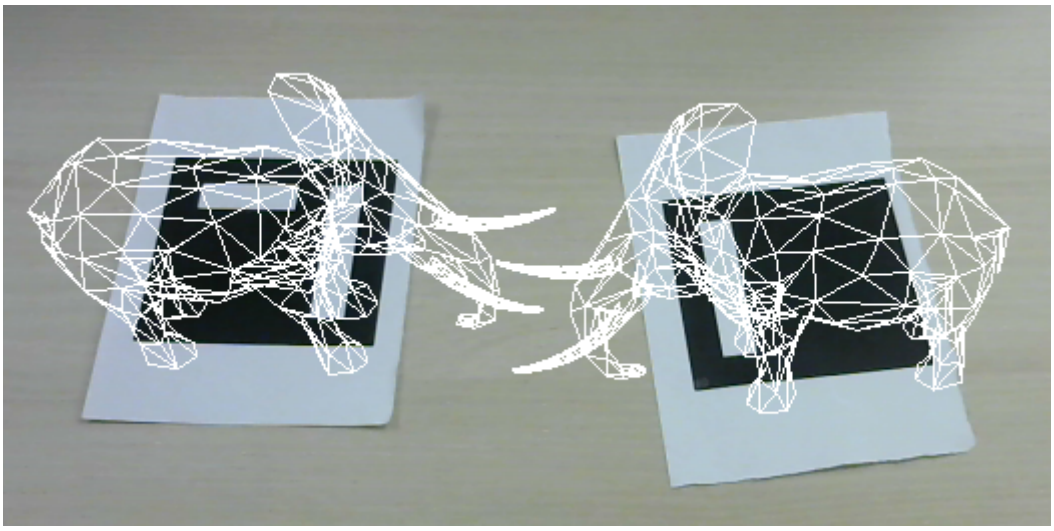


### arUco::drawScene()

1. On vérifie que l'image est bien chargée, si non, on ne fait rien.
2. On défini les différentes matrices de `OpenGL` : `GL_MODELVIEW`, `GL_PROJECTION` et on défini une fenêtre de visualisation.
3. On assigne les données de la caméra (`m_ResizedImage.ptr(0)`) au viewport qu'on vient de créer avec `glDrawPixels`.
4. On défini la matrice de projection telle qu'elle n'ait pas de distortion et on l'assigne au viewport.
5. Pour chaque marqueur détecté,
    1. on récupère la matrice `GL_MODELVIEW`, ce qui permettera de dessiner le cube et les axes d'une façon cohérente avec l'orientation du marqueur.
    2. On dessine les axes selon une méthode prédéfinie dans arUco.
    3. On dessine (en rouge) un cube en fil de fer devant le marqueur.

## importation `elephant.obj`

Nous avons importé à l'aide de la bibliothèque `tiny_obj_loader` un modèle lowpoly trouvé sur internet. Nous avons chargé le fichier dans le constructeur de la classe `ArUco`.



```
void ArUco::drawScene() {
    //[...]
    // For each detected marker
    for (unsigned int m=0;m<m_Markers.size();m++)
    {
        m_Markers[m].glGetModelViewMatrix(modelview_matrix);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glLoadMatrixd(modelview_matrix);

        // Disabling light if it is on
        GLboolean lightOn = false;
        glGetBooleanv(GL_LIGHTING, &lightOn);
        if(lightOn) {
            glDisable(GL_LIGHTING);
        }

        glScalef(0.02f,0.02f,0.02f); //pour ajuster la taille de notre éléphant.
        glRotatef(90,1,0,0); //pour qu'il ait les pieds sur terre.
        glTranslatef(0, 0, m_MarkerSize/2);

        glPushMatrix();
        drawOBJ();

        // Re-enabling light if it is on
        if(lightOn) {
            glEnable(GL_LIGHTING);
        }

        glPopMatrix();
    }

    // Disabling depth test
    glDisable(GL_DEPTH_TEST);

    m_pixels.create(m_GlWindowSize.height , m_GlWindowSize.width, CV_8UC3);
    //use fast 4-byte alignment (default anyway) if possible
```

```
    glPixelStorei(GL_PACK_ALIGNMENT, (m_pixels.step & 3) ? 1 : 4);
    //set length of one complete row in destination data (doesn't need to equal
img.cols)
    glPixelStorei(GL_PACK_ROW_LENGTH, m_pixels.step/m_pixels.elemSize());
    // Reading back the pixels
    glReadPixels(0, 0, m_GlWindowSize.width , m_GlWindowSize.height, GL_RGB,
GL_UNSIGNED_BYTE, m_pixels.data);
    // Flip the pixels since OpenCV stores top to bottom and OpenGL from bottom
to top
    cv::flip(m_pixels, m_pixels, 0);
}
```

la fonction appelée dans `drawScene` est :

```
void ArUco::drawOBJ(){
    // Loop over shapes
    for (size_t s = 0; s < shapes.size(); s++) {
        // Loop over faces(polygon)
        size_t index_offset = 0;
        for (size_t f = 0; f < shapes[s].mesh.num_face_vertices.size(); f++) {
            int fv = shapes[s].mesh.num_face_vertices[f];

            // Loop over vertices in the face.
            glPolygonMode(GL_FRONT, GL_LINE);
            glBegin(GL_TRIANGLES);
            for (size_t v = 0; v < fv; v++) {
                // access to vertex
                tinyobj::index_t idx = shapes[s].mesh.indices[index_offset + v];
                glVertex3f(attrib.vertices[3*idx.vertex_index+0],
attrib.vertices[3*idx.vertex_index+1], attrib.vertices[3*idx.vertex_index+2]);
                glNormal3f(attrib.normals[3*idx.normal_index+0],
attrib.normals[3*idx.normal_index+1], attrib.normals[3*idx.normal_index+2]);
            }
            glEnd();
            index_offset += fv;

        }
    }
}
```

L'augmentation est bien stable pour un fichier `obj` relativement petit. Nous avons donc importé un éléphant lowpoly.

# 3. Application de réalité augmentée

Notre idée d'application est la suivante :

- On génère aléatoirement des "fruits" sur le plan de la table (la caméra filme du dessus)
- Sur chaque marqueur (possibilité multijoueur), est généré un animal (idée du "snake"), qui grossirait à chaque fois qu'il entre en collision avec un fruit

Pour cela, on aurait :

- une fonction qui génère des fruits avec des positions aléatoires sur le plan de la table, détecté grace à un marqueur.

- une fonction `proximite` qui gère la détection de proximité entre la matrice modelview de chaque marqueur et celle des fruits.
- la mise à jour de la fonction `drawScene`, en particulier `glScalef` dès que la fonction `proximite` renvoie un booléen `true`.

Nous n'avons malheureusement pas eu le temps de faire ces fonctions car beaucoup de temps nous a été nécessaire pour faire compiler le projet (d'abord sur mint, puis sur visual studio : environ 1h30).