Augmentation 3D de la scène basée sur le suivi de marqueur visuel

Partie 1 : Identification et suivi du marqueur

L'augmentation visuelle d'une scène réelle se définit par 3 étapes : l'identification de l'objet d'intérêt, sa localisation et enfin l'incrustation virtuelle. Insérer un objet 2D dans la scène requière le calcul de l'homographie qui lie le repère de l'image de référence au repère de l'image requête. Toutefois, pour ajouter un modèle 3D dans l'image, il est nécessaire d'estimer la pose de la caméra réelle et calculer les transformations de la caméra virtuelle. Ensuite, la transformation liant le repère de la caméra réelle au repère de la caméra virtuelle doit être déterminée pour permettre le mixage réel-virtuel.

Ce travail a pour objectif de réaliser une application de réalité augmentée qui superpose un objet 3D sur un marqueur visuel de type code QR. Le marqueur est identifié et suivi dans le flux de la caméra grâce à son code matriciel. Cette approche est robuste et rapide et utilisée dans plusieurs applications pour coder l'information par des carrés noirs disposés dans un carré à fond blanc.

Cette première partie consiste à identifier le code QR dans la scène. A cet effet, il est demandé d'utiliser la librairie **ZBAR** pour la lecture de code QR. Les traitements suivants doivent être réalisés pour identifier un code QR dans l'image :

1.1. Installer la librairie **zbar** : la librairie précompilée sous Windows est fournie, pour Linux il faut l'installer à l'aide de la commande suivante : `sudo apt-get install libzbar-dev`

1.2. Appeler le fichier en-tête **zbar.h** dans le fichier source et ajouter les dépendances externes dans la configuration du projet et dans les variables d'environnement.

1.3. Ouvrir la vidéo de test, convertir l'espace couleur en niveaux de gris.

1.4. Ajouter le code suivant pour obtenir le nom du code QR ainsi que ses 4 sommets :

```
// declare variables
ImageScanner scanner;
vector<Point2f> qrPoints;
string qrName;
int num_qrPoints = 0;
// convert image data to raw data
uchar* raw = (uchar*)gray.data;
// wrap image data
Image image(gray.cols, gray.rows, "Y800", raw, gray.cols * gray.rows);
// scan the image for barcodes
int n = scanner.scan(image);
// extract results
for (Image::SymbolIterator symbol = image.symbol_begin();
     symbol != image.symbol_end(); ++symbol)
{
   // get useful results
   qrName = symbol->get_data();
   num_qrPoints = symbol->get_location_size();
   for (int i = 0; i < num_qrPoints; i++)
     qrPoints.push_back(Point(symbol->get_location_x(i), symbol->get_location_y(i)));
}
```

1.5. Pour stabiliser le suivi, activer la reconnaissance du code QR uniquement pour des mouvements dont la norme L2 des 4 sommets est supérieure à 5px.

Partie 2 : Estimation de la pose de la caméra réelle

L'estimation de la pose de la caméra a pour objectif de déterminer la translation et la rotation du repère du modèle 3D par rapport au repère de la caméra. Pour cela, il est nécessaire de disposer de la matrice interne de la caméra ainsi que des coordonnées 3D des points caractéristiques du modèle 3D appariés à leurs homologues 2D dans la scène. Si la calibration de la caméra fournit la matrice interne par le biais d'un processus d'étalonnage hors-ligne, il est par ailleurs, difficile de déterminer des points 3D du modèle correspondant à des points 2D identifiés dans la scène. La solution à ce problème consiste à poser le modèle 3D au centre du code QR physique, représenté dans le repère monde. Le code QR physique définit des points caractéristiques identifiables exprimés dans son propre repère 3D et auquel est attaché le modèle virtuel. Par conséquent, les sommets du code QR physique sont appariés aux sommets du même code QR identifié dans la scène. Les appariements 2D-3D sont déterminés par association des points de référence 3D aux points caractéristiques 2D, ce qui permet donc, d'estimer la pose de la caméra réelle.
La matrice caméra ainsi que les sommets du code QR physique ont les valeurs suivantes :

$$CameraMatrix = \begin{pmatrix} 605.57 & 0 & 334.90 \\ 0 & 591.46 & 238.77 \\ 0 & 0 & 1 \end{pmatrix}$$

$$P_{3D} = \begin{pmatrix} -100/2 & +100/2 & 0 \\ -100/2 & -100/2 & 0 \\ +100/2 & -100/2 & 0 \\ +100/2 & +100/2 & 0 \end{pmatrix}$$

La procédure d'estimation de la pose est définie par l'ensemble des traitements suivants :

2.1. Définir les variables nécessaires à la procédure d'estimation de la pose de la caméra.
2.2. Appeler la fonction solvePnP() pour le calcul de la pose.
2.3. Appeler la fonction Rodrigues() pour transformer le vecteur de rotation en une matrice de rotation.

Partie 3 : Gestion du modèle 3D et augmentation de la scène réelle

La gestion de la scène virtuelle consiste à charger un modèle 3D et réaliser différentes transformations pour l'afficher correctement. D'autre part, il s'agit d'une application de réalité augmentée, il est alors nécessaire de mixer le virtuel avec le réel. Cela nécessite l'utilisation du flux de la caméra réelle comme source d'information. De plus, il faut lancer le processus de reconnaissance et de localisation de la caméra réelle pour insérer le modèle virtuel 3D sur la cible réelle en calculant toutes les transformations requises pour l'affichage du modèle virtuel 3D sur la scène réelle.

OpenGL est utilisé pour manipuler la scène 3D et les fonctions de GLUT sont employées pour définir toutes les fonctions nécessaires à la gestion de l'interface utilisateur et le rendu 3D. Les traitements suivants doivent être réalisés afin de recaler dynamiquement l'objet virtuel sur l'objet d'intérêt dans le processus de tracking temps-réel :

3.1. Dans la fonction main, appeler les fonctions GLUT suivantes : glutInit(), glutInitDisplayMode(), glutInitWindowSize(), glutCreateWindow(), init(), glutDisplayFunc(), glutMainLoop().

3.2. Dans la fonction de rappel du rendu, réaliser les traitements suivants :

3.2.1. Appeler la fonction **matToTexture()** pour transformer l'image de la caméra en une texture OpenGL.

3.2.2. Appeler les matrices modèle-vue-projection en définissant un volume d'affichage avec la fonction **glFrustum()**.

3.2.3. Transformer la translation de la caméra réelle de l'unité métrique au pixel pour la représenter dans la fenêtre d'affichage OpenGL.

3.2.4. Exprimer la translation de la caméra réelle dans le repère clip et appliquer cette translation avec **glTranslatef()**.

3.2.5. Appliquer la rotation de la pose la caméra réelle ainsi que la rotation de passage du repère de la caméra réelle au repère clip de la caméra virtuelle en utilisant **glRotatef()**.

3.2.6. Charger le modèle **glutSolidTeapot()**.

Partie 4 : Interaction avec le modèle 3D

L'objet virtuel est maintenant associé à la cible visuelle et y reste incrusté tout au long du processus de tracking. Par ailleurs, dans une application de réalité augmentée, il est important d'interagir avec les modèles virtuels pour éprouver un sentiment de présence dans le monde mixte et enrichir l'expérience utilisateur. Dans cette partie, différentes interactions sont définies pour manipuler et animer l'objet virtuel. Les opérations à réaliser sont les suivantes :

4.1. Tourner l'objet virtuel dans les deux sens de rotation, autour des axes X et Y, en utilisant le mouvement de la souris.

4.2. Réaliser un zoom avant et arrière pour agrandir et rétrécir l'objet virtuel en utilisant la mollette de la souris.

4.3. Associer à la touche **a** du clavier une animation en rotation autour de l'axe +Y et associer à la touche **A** une animation autour de l'axe -Y.

4.4. Définir les fonctionnalités suivantes pour ces touches claviers :

4.3.1. **ESPACE** : réinitialiser toutes les transformations rajoutées par l'utilisateur (définies dans les questions 4.1, 4.2 et 4.3) .

4.3.2. Touche **s** : démarrer / arrêter le flux vidéo.

4.3.3. **ECHAP** : quitter l'application.