Effets Spéciaux - Réalité Augmentée

Corentin SALAÜN* Thomas LEMETAYER† Julie FOURNIER‡ Brandon LEBON§
Etudiants, ESIR

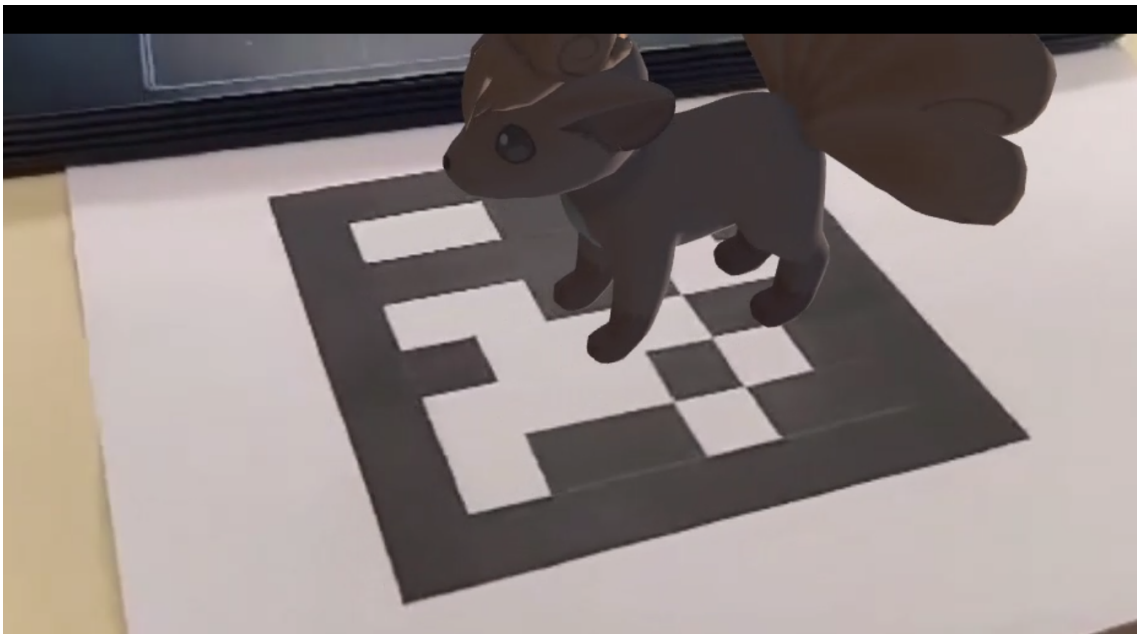

Figure 1: *Extrait d'une vidéo de réalité augmentée*

Mots clés: Tracking, Slam, Augmented Reality, Computer Vision

1 Introduction

Le but de ce TP est de mettre en oeuvre une expérience de réalité augmentée. À partir d'une vidéo acquise par une caméra, nous allons chercher à y incruster un modèle 3D en mouvement. Pour cela, nous devons suivre un certain nombre d'étapes afin que le rendu soit stable et cohérent avec la scène.

2 Calibration de la caméra

La première étape est de calibrer la caméra. En effet, sans ses paramètres intrinsèques, il est impossible de trouver la pose de la caméra de façon fiable, et l'incrustation à suivre ne peut pas fonctionner. Pour ce faire, nous utilisons une mire de calibration en jeu d'échec que nous avons filmée sous différents angles (voir figure 2), afin de pouvoir remonter aux paramètres de la caméra. Nous extrayons 15 images de cette vidéo qui vont servir pour la calibration.

Nous donnons ensuite ces images à l'algorithme développé par OpenCV pour calibrer une caméra [OpenCVDevTeam 2019], qui détecte les coins intérieurs de l'échiquier sur chacune. La position de ces coins lui permet ensuite de résoudre un système non linéaire afin d'obtenir les paramètres de la caméra. Nous obtenons alors une matrice de paramètres intrinsèques de la forme suivante :

$$\mathbf{K} = \begin{pmatrix} a_x & 0 & u_x \\ 0 & a_y & u_y \\ 0 & 0 & 1 \end{pmatrix}$$

*e-mail:corentin.salaun@gmail.com
†e-mail:thomas.lemetayer.35@gmail.com
‡e-mail:julie.fournier.1996@gmail.com
§e-mail:lebonbrandon22@gmail.com

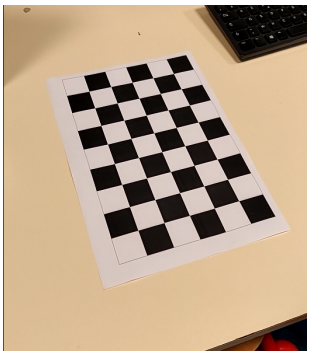

Figure 2: *Image de la mire de calibration utilisée extraite de la vidéo*

avec

- $a_x = f/l_x$ oùf est la distance focale et l_x la largeur d'un pixels ;
- $a_y = f/l_y$ où l_y est la hauteur d'un pixel ;
- (u_x, u_y) sont les coordonnées du centre de la caméra dans le plan image.

Nous transmettons alors les paramètre obtenus à l'algorithme de SLAM afin qu'il puisse avoir une précision suffisante.

3 Algorithme

Dans cette section, nous détaillons l'algorithme que nous avons implémenté. Celui-ci a pour but d'estimer la pose de la caméra

sur chaque image de la vidéo afin de pouvoir la transmettre à la partie de rendu qui va ensuite incruster le modèle 3D. Cette première version se base le suivi d'un marqueur connu à l'avance.

3.1 Extraction des points d'intérêt et matching

Pour définir le mouvement de la caméra d'une image à une autre de la vidéo, nous avons besoin de points de repère sur les deux images afin de pouvoir mettre en relation les coordonnées des points des deux images. Pour ce faire, nous utilisons une image de référence contenant des informations communes aux deux images de la vidéo. La mise en relation des informations des images de la vidéo avec les informations de l'image de référence permet la mise en relation des deux images de la vidéo entre-elles. Cependant dans notre cas, nous n'avons aucune information a priori sur la relation entre les images de la vidéo et l'image de référence. Par conséquent, nous devions effectuer une extraction de points d'intérêts dans les images.

Parmi les algorithmes les plus utilisés pour extraire des points d'intérêt dans une image, nous avons retenu SIFT [Lowe 2004]. Cette méthode se base sur la détection de coins de Harris qui permet d'obtenir un excellent descripteur de chaque pixel en fonction de son contexte local. Les descripteurs SIFT sont basés sur les gradients et non sur l'intensité des pixels, ce qui apporte une invariance aux transformations de rotation et de changement d'échelle. Cet apport est indispensable pour notre algorithme, car les transformations d'une image à une autre ne sont rarement que des translations.

Ainsi, pour réaliser un matching de point entre deux images, nous commençons par calculer les points d'intérêt de chaque image et les descripteurs associés. Puis, nous matchons les points d'intérêts dont la distance entre les descripteurs associés est inférieure à un seuil. Pour ce faire, la méthode la plus basique est de comparer chaque point d'intérêt avec l'ensemble des autres points d'intérêt, ce qui correspond à un algorithme dont le temps de calcul est de complexité en O(n). Nous nous sommes donc vite rendu compte de l'impact de cette méthode sur les performances. Comme nous souhaitons avoir un résultat en temps réel, nous avons opté pour une méthode de complexité en O(log(n)). Celle-ci utilise un KDTree nous permettant de diviser l'espace des descripteurs. Ainsi, lors de la recherche des plus proches descripteurs, on élimine rapidement une grande partie de l'espace des descripteurs.

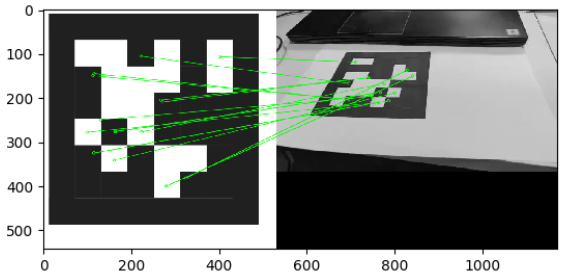

Figure 3: *Exemple de mise en correspondence de points*

3.2 Calcul d'Homographie

Afin d'estimer le déplacement de la caméra entre l'image capturée et l'image de référence du tag que nous utilisons, nous devons estimer une homographie grâce aux points mis en correspondance précédemment. Il existe différents algorithmes pour calculer l'homographie entre deux images, mais tous ont besoin d'un certain nombre de points mis en correspondance entre les deux images. Au minimum, il est nécessaire de disposer des coordonnées de 5 points présents dans les deux images. C'est le cas par exemple pour l'algorithme de la DLT que nous avons utilisé, comme [Marchand et al. 2015].

Si on note X_1 les coordonnées d'un point dans l'image 1, X_2 les coordonnées du même point dans l'image 2, et $\mathbf{H}$ l'homographie permettant de passer de l'image 1 à l'image 2, on a :

$$X_2 = {}^2H_1 X_1$$

Si on pose $X_2 = (x_2, y_2, w_2)$, et qu'on développe la précédente expression, on finit par trouver l'équation suivante : $\Gamma h = 0$, avec Γ tel que :

$$\Gamma = \begin{pmatrix} & \vdots & \\ 0^T & -w_{i2}X_1^{iT} & y_{i2}X_1^{iT} \\ w_{i2}X_1^{iT} & 0^T & -x_{i2}X_1^{iT} \\ & \vdots & \end{pmatrix}$$

On peut résoudre cette équation en faisant une décomposition SVD de cette matrice. h est le vecteur propre associé à la plus petite valeur propre.

Cependant, parmis les points mis en correspondance, il est inévitable qu'il y ait des erreurs. Ces erreurs ont un gros impact sur la qualité de l'homographie estimée, ce qui risque de donner une incrustation du modèle 3D dans a vidéo instable. Afin de pallier ce problème, nous avons ajouté un algorithme RANSAC [Fischler and Bolles 1981] à notre estimation d'homographie.

L'objectif de l'algorithme RANSAC est de calculer, itérativement, plusieurs homographies, chacune à partir d'un sous ensemble de l'ensemble des points mis en correspondance, de tester ces homographies sur l'ensemble des points et de sélectionner celle qui commet le moins d'erreur. La sélection de l'homographie la plus performante correspond à prendre celle qui maximise le nombre de points pour lesquels l'erreur de prédiction est inférieure à un ϵ, c'est à dire minimiser le nombre d'outliers. Pour chacune des homographies calculées, on retient donc les points pour lesquels elle est précise. Afin d'obtenir l'homographie finale, nous calculons alors une dernière homographie à partir des points retenus pour celle qui minimisait l'erreur de prédiction.

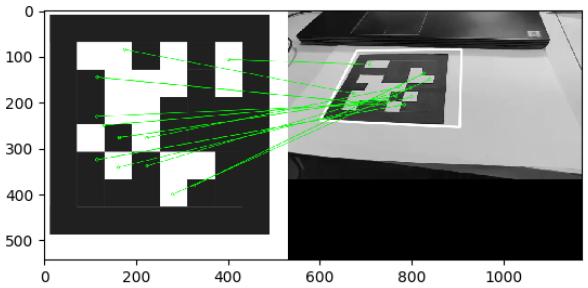

Figure 4: *Exemple d'homographie*

3.3 Estimation de la Pose

Tout l'objectif d'une application de réalité augmentée est de calculé correctement la pose de la caméra. D'après [Marchand et al. 2015], on peux facilement extraire cette information grâce à la matrice d'homographie. En effet, l'homographie permet de faire une correspondance entre des plans. On prend comme a priori que la profondeur ${}^wZ = 0$ dans l'image de référence ce qui nous donne tout les points 3D de l'image de référence ${}^w\mathbf{X} = ({}^wX, {}^wY, 0, 1)^T$.

D'après la projection perspective, on peut trouver l'équivalent des points dans le plan image :

$$\mathbf{x_0} = \mathbf{\Pi}\ {}^0\mathbf{T}_w^w\mathbf{X} = \mathbf{\Pi}\begin{pmatrix}\mathbf{c}_1 & \mathbf{c}_2 & \mathbf{c}_3 & {}^0\mathbf{t}_w\end{pmatrix}\begin{pmatrix}{}^wX \\ {}^wT \\ 0 \\ 1\end{pmatrix}$$

En supposant que la matrice de rotation est de norme 1, on peut réécrire l'équation en supprimant la troisième colonne de la matrice :

$$\mathbf{x_0} = \mathbf{\Pi}\ {}^0\mathbf{T}_w^w\mathbf{X} = \mathbf{\Pi}\begin{pmatrix}\mathbf{c}_1 & \mathbf{c}_2 & {}^0\mathbf{t}_w\end{pmatrix}\begin{pmatrix}{}^wX \\ {}^wT \\ 1\end{pmatrix}$$

On obtient finalement l'équation suivante :

$$\begin{pmatrix}\mathbf{c}_1 & \mathbf{c}_2 & {}^0\mathbf{t}_w\end{pmatrix} = \mathbf{\Pi}^{-1}\ {}^0\mathbf{H}_w$$

Si l'on considère la matrice de rotation comme une matrice orthogonale, on calcule $\mathbf{c}_3 = \mathbf{c}_1 \times \mathbf{c}_2$. On peut étendre ces équations pour calculer toute les poses ${}^n\mathbf{T}_w$.

Ainsi à partir des calculs d'homographies, on peut en déduire la pose entre chaque caméra et notre repère (cf figure 5).

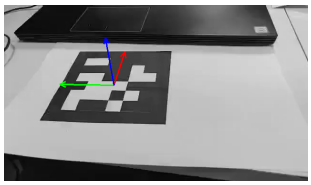

Figure 5: *Exemple d'estimation de pose*

4 Insertion du modèle dans la vidéo

A partir des calculs de poses pour chaque images de la vidéo, on peut désormais intégrer un modèle 3D pour réaliser notre réalité augmentée. Pour des raisons de simplicité, on a choisit de faire l'affichage du rendu dans une application externe, nous allons donc utiliser Unity.

4.1 Export des résultats

Tout d'abord, il est nécessaire de transmettre l'information à Unity. Pour cela, on transmet simplement un fichier qui contient les différentes poses pour chaque image. Lors d'une première implémentation, nous avions choisi de transmettre directement la matrice 4×4 mais nous nous sommes confrontées à un problème de stabilité dû à des erreurs d'arrondi lors de l'écriture fichier.

Pour nous convaincre que l'erreur vient bien de la transmission de l'information, on compare la stabilité à l'aide d'un repère côté moteur et côté Unity. Sachant que Unity gère les rotations sous forme de quaternions qui représentent de manière compacte les rotations, on a choisi de réduire directement la rotation en quaternions lors de l'estimation de pose, de transmettre ces quaternions plutôt que les matrices de rotation. Ainsi, on réduit la sensibilité du système.

4.2 Repère Direct vers Indirect

Unity est basé sur un moteur OpenGL et utilise donc un système basé sur les repères indirects. Or, nos matrices de transformations s'expriment selon un repère direct. Dans le cas de la translation, il suffit d'inverser la composante y mais cela reste plus complexes pour la rotation. Après quelques recherches, nous avons trouvé une implémentation pour transformer les repères directs en indirects [Eberly 2003] basé sur les angles d'Eulers.

Listing 1: *Left to Right Handed*

```
// left-handed rotation angles
float LH_AngleH, LH_AngleP, LH_AngleB;

// left-handed coordinate rotation matrices
Matrix3 LH_H((0,1,0), LH_AngleH); // heading
Matrix3 LH_P((1,0,0), LH_AngleP); // pitch
Matrix3 LH_B((0,0,1), LH_AngleB); // bank

// left-handed composite rotation
Matrix3 LH_Rotate = LH_H * LH_P * LH_B;

// right-handed composite rotation
Matrix3 RH_Rotate = LH_Rotate;

RH_Rotate[0][2] = -RH_Rotate[0][2];
RH_Rotate[1][2] = -RH_Rotate[1][2];
RH_Rotate[2][0] = -RH_Rotate[2][0];
RH_Rotate[2][1] = -RH_Rotate[2][1];
```

Par ailleurs, en utilisant directement les quaternions, on simplifie le problème car le concept d'angle heading, pitch, et back sont directement résumés en 1 vecteur directeur. Il suffit donc de la même manière que pour la translation, inverser la composante y du vecteur directeur. Il faut cependant faire attention au signe de l'angle de rotation en fonction des axes inversés.

4.3 Unity

Enfin, nous avons modélisé une scène simple avec Unity, filmée par une caméra virtuelle suivant les poses transmises par les parties précédentes. Pour ce faire, nous avons implémenté un script de caméra qui ouvre le fichier contenant les poses de caméra à l'initialisation et les stock dans la mémoire de la caméra. Nous appliquons notre vidéo sur un écran virtuel mis au second plan de la caméra quel que soit la pose de la caméra. Il suffit alors de changer la pose de la caméra à chaque changement d'image de la vidéo.

De cette façon, on superpose le rendu de notre caméra réel avec celui de la caméra virtuelle suivant les mêmes transformations. Un exemple de résultat est visible sur la figure 6.

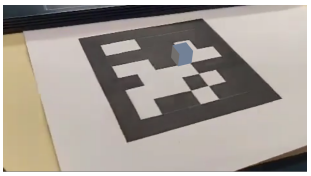

Figure 6: *Résultat sur Unity*

Pour optimiser les résultats, nous avons donné les mêmes paramètres à notre caméra virtuelle que ceux que nous avons estimé pour notre caméra réelle.

5 Présentation des résultats et Amélioration

A partir des ces différents travaux, on peut désormais analyser nos résultats.

5.1 Premier Résultat

Cette première méthode donne des résultats cohérents cependant le repère reste très instable. Le problème est principalement dû à un nombre trop faible de correspondances entre les descripteurs. Nous avons refait plusieurs tests en améliorant la qualité de l'image de référence, sa version imprimée et en augmentant la résolution de la vidéo. On observe des résultats légèrement plus stables.

5.2 Amélioration du suivi

On remarque que les problèmes de stabilité sont principalement dus au problème de mise en correspondances et donc de la détection des points d'intérêts. Nous avons testé une seconde méthode basé sur des tags (cf figure 7) afin d'utiliser au maximum nos connaissances sur la cible suivi.

Figure 7: *Marqueur utilisé pour le suivi*

Pour réalisé ce tracking, nous avons utilisé l'algorithme de détection de marqueur d'OpenCV [Garrido-Jurado et al. 2014]. Cette détection se base sur une suite de transformation simple et rapide.

Listing 2: *Détection de marqueurs ArUco*

```
1. Detections des marqueurs
    1.1. Binarisation de l'image par Otsu
    1.2. Detection de contours
2. Identification des marqueurs
    2.1. Suppression de la perspective
    2.2. Decoupage en cellule
    2.3. Extraction des bits
    2.4. Identification avec un dictionnaire
3. Raffinement de la detection
```

Cette deuxième technique est celle que nous avons retenu pour notre dernière vidéo car c'est la seule méthode qui assure une véritable stabilité.

6 Conclusion

Lors de ce TP, nous avons pu mettre en oeuvre différentes techniques utilisées pour réaliser un algorithme de SLAM. On remarque finalement qu'une grande partie du travail consiste à suivre correctement les différents éléments de la scène que ce soit des points d'intérêts issus de descripteurs, ou en se basant sur des notions plus abstraites comme les lignes ou les plans. Finalement, on peut coupler le suivi purement visuel avec d'autres types d'informations comme la profondeur ou le gyroscope qui pourra améliorer la stabilité de l'incrustation.

References

EBERLY, D., 2003. Conversion of left-handed coordinates to right-handed coordinates.

FISCHLER, M. A., AND BOLLES, R. C. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*, 6, 381–395.

GARRIDO-JURADO, S., MUÑOZ-SALINAS, R., MADRID-CUEVAS, F. J., AND MARÍN-JIMÉNEZ, M. J. 2014. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition 47*, 6, 2280–2292.

LOWE, D. G. 2004. Distinctive image features from scale-invariant keypoints. *International journal of computer vision 60*, 2, 91–110.

MARCHAND, E., UCHIYAMA, H., AND SPINDLER, F. 2015. Pose estimation for augmented reality: a hands-on survey. *IEEE transactions on visualization and computer graphics 22*, 12, 2633–2651.

OPENCVDEVTEAM, 2019. Camera Calibration with OpenCV.