TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

# Projet de Réalité Augmentée
## Partie II – Remplacement de contenu
## *Transformation homographique & construction 3D*

## 1. Préambule

La transformation d'une image quelle qu'elle soit a pour vocation de modifier à loisir les propriétés des pixels, que cela touche à leur valeur et donc à l'intensité (ou la couleur) d'une région de l'image ou bien à leur position dans la « matrice image » correspondant à la forme qui sera perçue *in fine*. Ce travail sur les pixels peut être à fin corrective ou améliorative afin de compenser des imperfections apparues lors de la prise d'image (par exemple des distorsions géométriques) de sorte que l'on aura une image optimisée au bout du compte, ou bien à fin de pure modification ou d'assemblage, notamment pour créer des effets spéciaux, artistiques, de profondeur, mettre en correspondance des images d'un même objet mais différentes en leur instant de prise ou en leur point de vue ou encore remplacer un contenu (montage).

Dans le cadre de ce projet nous nous intéresserons à cette dernière application et nous efforcerons de remplacer le contenu d'une feuille A4 qu'une personne déplace au cours d'une vidéo. Cette dernière apparaît comme une succession d'images de point de vue fixe (plongeant/oblique) mais dont les différentes dates de prises de vue correspondent à différentes positions de la feuille. L'enjeu est donc tout d'abord de pouvoir **détecter les coins** de la feuille A4 sur une image de la vidéo : il s'agit de la **première partie** de ce projet.

Nous devons ensuite trouver la transformation spatiale (**homographie**) à appliquer au contenu afin de faire correspondre ses contours avec ceux de la feuille (dans le point de vue considéré), et enfin remplacer chacun des pixels visés de l'image par celui de notre contenu qui lui correspond. Afin d'éviter les problèmes liés au recouvrement/étirement des pixels liés à la perspective, nous aurons également recours à une transformation d'intensité (**interpolation bilinéaire**) de l'image obtenue. Pour finir nous aborderons l'**insertion de contenu 3D** en structure « fil de fer » qui est une projection des coordonnées 3D des points que l'on place sur l'espace 2D de l'image. Nous verrons que ce travail est basé sur le même principe que la transformation homographique puisqu'il s'agit de faire correspondre les coordonnées de pixels exprimées dans différents espaces à l'aide d'une matrice de projection ; il s'agit de la seconde partie du projet, détaillée dans le présent document.

Afin de se représenter le déroulement et l'architecture de ce traitement, nous schématisons ci-dessous l'articulation des différentes étapes qui le composent.

## 2. Démarche théorique & algorithmique

Avant de débuter l'étude des aspects techniques du traitement considéré, nous rappelons que tout le travail a été effectué sur une trame simple de la vidéo, pour une position aléatoire de la feuille mobile. Celle-ci importe peu étant donnée l'automatisation de l'algorithme de détection

des coins. Il suffit alors de répéter notre programme sur chaque trame afin que notre traitement suive le déroulement de la vidéo. Vous pourrez trouver toutes les manipulations relatives à l'ouverture/fermeture d'une vidéo, d'une image, isolation d'une trame, transformation en niveaux de gris ou code YCbCr… dans la première partie de ce rapport.

## ❖ Remplacement d'une image par un contenu 2D

### ➢ *Détermination de la transformation homographique*

Pour commencer, il ne semble pas superflu de rappeler qu'une image peut être vue comme une matrice 2D de pixels, définis par leurs coordonnées $x$ et $y$ correspondant à leur position dans l'image ainsi que leur valeur traduisant l'intensité lumineuse (valeur entre 0 et 255 pour une image en niveaux de gris, vecteur de trois valeurs RGB comprises entre 0 et 255 pour une image en couleurs). Ainsi les coordonnées d'un pixel $P_i$ peuvent, en première approche intuitive, être écrites dans un vecteur (2,1) comme suit :

$$P_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

Une homographie résulte de la composition de transformations classiques (translations, rotations, changements d'échelle) elle peut dans certains cas être associée à une transformation affine. Afin de simplifier au maximum le traitement et donc les opérations à réaliser, nous définissons donc un nouveau vecteur de coordonnées dites homogènes intégrant une troisième composante ainsi qu'un facteur d'échelle $s_i$. On représente les avantages comparés de ces deux expressions dans l'équation ci-dessous :

$$\begin{cases} P_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \Longrightarrow P_2 = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} P_1 + \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} = AP_1 + B \\ \underline{P_i} = s_i \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \Longrightarrow \underline{P_2} = \begin{pmatrix} A_{11} & A_{12} & B_1 \\ A_{21} & A_{22} & B_2 \\ 0 & 0 & 1 \end{pmatrix} \underline{P_1} = H\underline{P_1} \end{cases}$$

Cette écriture se généralise à n'importe quelle association de transformations, souvent plus complexes et plus difficiles à représenter ; on parle aussi de ***cascade de transformations***. En effet, si l'on applique des transformations successives $H_1, H_2, \dots, H_n$ à l'image (i.e. à chaque pixel) alors la transformation finale peut être écrite de manière simple (attention à l'ordre) :

$$\underline{P_2} = H_n H_{n-1} \dots H_3 H_2 H_1 * \underline{P_1} = H\underline{P_1}$$

La matrice homographique H est constituée de neuf éléments indéterminés tels que :

$$\begin{cases} \underline{P_2} = H\underline{P_1} \\ H = \begin{pmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{pmatrix} \end{cases} \Longrightarrow \begin{cases} x_2 = \dfrac{H_{11}x_1 + H_{12}y_1 + H_{13}}{H_{31}x_1 + H_{32}y_1 + H_{33}} \\ y_2 = \dfrac{H_{21}x_1 + H_{22}y_1 + H_{23}}{H_{31}x_1 + H_{32}y_1 + H_{33}} \end{cases}$$

L'application de H est la même lorsque l'on multiplie par un scalaire (on a en effet passé sous silence le facteur d'échelle $s_2$), on en déduit donc des degrés de liberté au nombre de huit et

non neuf, c'est pourquoi nous fixerons le coefficient $H_{33}$ à 1 dans la suite. Ainsi ayant deux équations par pixel (une par coordonnée) il nous faut quatre paires de points (caractéristiques en général, tels des coins) que l'on veut associer à travers cette transformation pour déterminer tout-à-fait la projection homographique (par résolution numérique de système linéaire, voir *Implémentation*).

- *Appariement des pixels du polygone et du contenu*

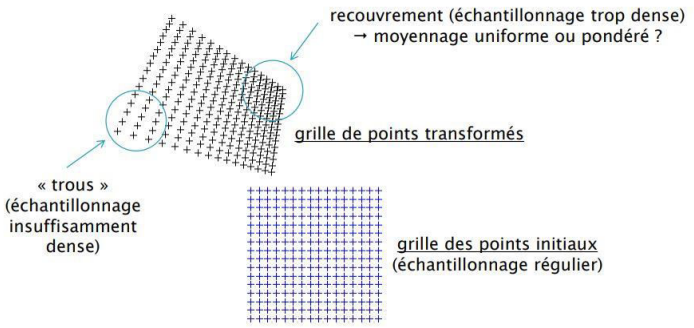
Une fois les coefficients connus, nous pouvons multiplier les coordonnées homogènes de chaque pixel de notre contenu par la matrice H afin d'en déduire ses coordonnées dans l'espace à remplacer sur la trame considérée de la vidéo. De la sorte, il nous suffit d'imposer à l'ensemble de la matrice image le respect de l'hypothèse de conservation d'intensité puis d'appliquer toutes les étapes précédentes à toutes les trames pour avoir une première version de notre vidéo retouchée. Cette hypothèse se traduit mathématiquement par l'équation suivante, ramenée aux coordonnées standard des pixels et pour un contenu à ajouter *ImRGB* :

$$\forall P_i \in ImRGB, I_2(P_2) = I_1(P_1)$$

Comme nous l'avons évoqué précédemment, cet algorithme intuitif ou naïf induit un problème d'échantillonnage de l'espace à remplir : en effet, les pixels de notre contenu sont régulièrement espacés suivant deux axes *x* et *y*, et ses bords sont parallèles et égaux deux à deux. En revanche, la forme à remplir est beaucoup moins régulière du fait de sa position différente à chaque fois, la perspective et le point de vue qui la déforment lors de sa projection sur le capteur (caméra). Il va donc falloir gérer d'une part un recouvrement des pixels en certaines zones qui va détériorer l'image et créer de la perte d'information, et d'autre part des éloignements de pixels laissant apparaître des trous (pixels noirs) entre eux.

Un moyennage (uniforme ou pondéré) devrait suffire à nous affranchir du recouvrement et résulter d'une transposition fidèle des pixels.

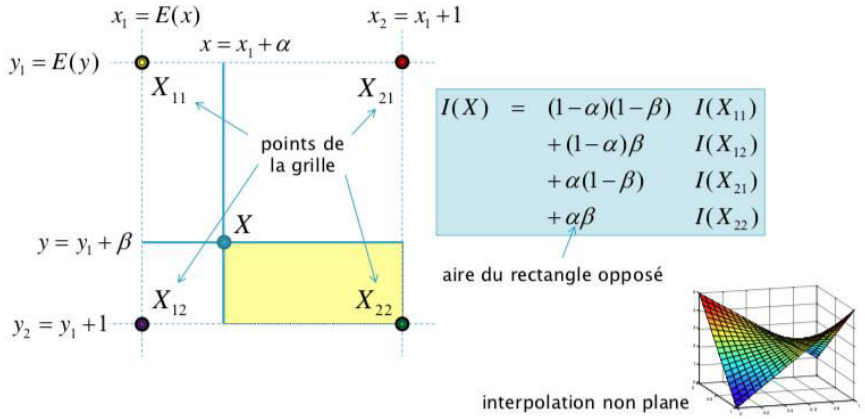
Le second problème est un peu plus délicat à traiter et va nécessiter de « remplir » l'image transformée en utilisant les points voisins (dans la grille d'échantillonnage) afin d'affecter une valeur (moyennée) à des pixels fictifs (coordonnées non entières). On appelle cette méthode l'**interpolation bilinéaire**.




- *Remplissage des trous – interpolation bilinéaire*

L'interpolation bilinéaire consiste à calculer la valeur d'un pixel à partir de ses deux plus proches voisins dans chaque direction. Elle permet d'obtenir de bien meilleurs résultats que l'interpolation à partir du plus proche voisin avec une bonne complexité. Les coefficients sont déterminés par la position relative du pixel avec ses quatre plus proches voisins comme suit :

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin




## ❖ Construction d'une structure *fil de fer* en trois dimensions

### ➢ *Création de l'espace 3D transposé sur la scène 2D considérée*

Une fois la méthode de transformation homographique complètement assimilée, construire un objet en trois dimensions (appelé *structure en fil de fer* car composé exclusivement de segments) au sein de notre image est loin d'être effrayant. En effet, si l'on se reporte au schéma conceptuel de cette partie, le raisonnement est analogue au moins de la matrice de projection à déterminer jusqu'à la conversion des coordonnées. Seulement ici, nous n'associons pas les pixels d'une image (en deux dimensions) à ceux d'une autre, mais nous convertissons plutôt les coordonnées d'un point, connues dans un espace à trois dimensions, à celles qu'il aurait si cet espace était projeté sur un plan (en conservant la perspective de la caméra). Si cela peut paraître flou en premier lieu, la transposition mathématique de cette démarche devrait rapidement mettre en évidence sa similitude avec le cas précédent. On peut commencer par modéliser les enjeux de ce traitement comme suit :

$$\begin{pmatrix} x_{2D} \\ y_{2D} \end{pmatrix} \leftrightarrow s_{2D} \begin{pmatrix} x_{2D} \\ y_{2D} \\ 1 \end{pmatrix} = P * s_{3D} \begin{pmatrix} x_{3D} \\ y_{3D} \\ z_{3D} \\ 1 \end{pmatrix} \leftrightarrow \begin{pmatrix} x_{3D} \\ y_{3D} \\ z_{3D} \end{pmatrix}$$

On note ici encore le passage par coordonnées homogènes, représenté par la double flèche. Ainsi on sait d'ores et déjà que la matrice P se doit d'être de taille 3x4 et donc comporter 12 coefficients. Comme précédemment, on fixera $P_{34} = 1$ pour éviter de résoudre un problème sur-contraint.

Pour commencer on se propose d'établir un repère en fixant des coordonnées dans l'espace, arbitraires et intuitives, à des points connus de la scène. Ce repère n'est pas nécessairement orthogonal ni même normé, il doit cependant être facile à représenter et traduire le point de vue de la caméra. Par exemple, on peut délimiter notre repère à la feuille A4 mobile (dont les coins sont aisément détectables, en plus d'être la zone sur laquelle on veut travailler) en affectant les coordonnées (0,0,0) au coin supérieur gauche, (0,1,0) au coin supérieur droit, (1,0,0) au coin inférieur gauche et (1,1,0) au coin inférieur droit. Ces coordonnées ne suffisent toutefois pas à connaître entièrement P et nous avons besoin de placer deux points supplémentaires (11 coefficients à déterminer impliquent d'avoir 12 équations ou 12 coordonnées, soit 6 points). On remarque qu'un coin inférieur du cube blanc (voir vidéo) est fixé au centre de la feuille, on lui affecte donc les coordonnées en trois dimensions (0.5,0.5,0). Enfin, suivant cette arrête, on n'a qu'à prendre (0.5,0.5,1) pour le coin supérieur et voilà nos six points caractéristiques.

Nous pouvons enfin établir le système linéaire à résoudre, encore très similaire à celui trouvé pour l'homographie (attention, de 12 équations et non 2 comme écrit ci-dessous, pour i allant de 1 à 6, différenciant les points choisis) :

$$\begin{cases} \underline{P_{2D}^i} = P * \underline{P_{3D}^i} \\ P = \begin{pmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{pmatrix} \end{cases} \Longrightarrow \begin{cases} x_{2D}^i = \dfrac{P_{11}x_{3D}^i + P_{12}y_{3D}^i + P_{13}z_{3D}^i + P_{14}}{P_{31}x_{3D}^i + P_{32}y_{3D}^i + P_{33}z_{3D}^i + P_{34}} \\ y_{2D}^i = \dfrac{P_{21}x_{3D}^i + P_{22}y_{3D}^i + P_{23}z_{3D}^i + P_{24}}{P_{31}x_{3D}^i + P_{32}y_{3D}^i + P_{33}z_{3D}^i + P_{34}} \end{cases}$$

Avec la matrice P, nous pourrons donc convertir les coordonnées dans l'espace des points recherchés aux coordonnées planaires correspondant sur la trame (de la vidéo prise par la caméra bien entendu) entre lesquels nous aurons enfin simplement à tracer des segments.

- *Tracé (2D) des différents segments de la structure (3D)*

Pour tracer un segment entre deux points d'une image, il suffit de changer la valeur de tous les pixels situés sur la trajectoire. Il y a plusieurs facteurs à prendre en considération, notamment la façon dont on veut tracer ce segment sur l'image (conditionnant la complexité du code) mais surtout ce qui définit l'appartenance d'un pixel à cette trajectoire car cette dernière pouvant être oblique, certains pixels ne seront que « partiellement traversés ».

Nous pourrions envisager de leur affecter une valeur inhomogène (telle que représentée par un triangle ou une Gaussienne par exemple, d'intensité décroissante sur les bords et s'uniformisant progressivement avec les pixels environnants) mais nous nous restreindrons ici au cas simple de pixels unis. Cela se traduira par des contours de segments plus ou moins « crantés » dépendant de la résolution de l'image.

Quant à la méthode de tracé nous opterons pour la plus simple à implémenter : après avoir échantillonné la longueur du segment (exprimée en nombre de pixels), nous allons la parcourir et modifier la valeur des pixels un à un (voir *Implémentation*). Il ne reste plus qu'à répéter ce processus entre des points soigneusement choisis afin de dessiner toutes vos envies.

# 3. Implémentation

Nous allons ici détailler le passage des formules mathématiques à l'interprétation algorithmique nous permettant d'exprimer tout ce qui a été vu en un langage compréhensible par l'ordinateur. Cependant nous ne nous attarderons pas sur les codes MATLAB en eux-mêmes dans cette section puisqu'ils figureront intégralement en annexe.

## ❖ Remplacement d'une image par un contenu 2D

- *Détermination de la transformation homographique*

En fin de présentation de la démarche théorique nous permettant de déterminer une homographie, nous en étions restés aux équations suivantes (s'appliquant aux coordonnées d'un pixel, en gardant à l'esprit que ces équations sont répétées sur les quatre coins) :

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

$$\begin{cases} \underline{P_2} = H\underline{P_1} \\ H = \begin{pmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & 1 \end{pmatrix} \end{cases} \Rightarrow \begin{cases} x_2 = \dfrac{H_{11}x_1 + H_{12}y_1 + H_{13}}{H_{31}x_1 + H_{32}y_1 + 1} \\ y_2 = \dfrac{H_{21}x_1 + H_{22}y_1 + H_{23}}{H_{31}x_1 + H_{32}y_1 + 1} \end{cases}$$

Que nous pouvons réécrire afin de faire apparaître un produit matriciel aboutissant aux coefficients $H_{ij}$ :

$$\begin{cases} x_2 = \dfrac{H_{11}x_1 + H_{12}y_1 + H_{13}}{H_{31}x_1 + H_{32}y_1 + 1} \\ y_2 = \dfrac{H_{21}x_1 + H_{22}y_1 + H_{23}}{H_{31}x_1 + H_{32}y_1 + 1} \end{cases} \Leftrightarrow \begin{cases} x_2(H_{31}x_1 + H_{32}y_1 + 1) = H_{11}x_1 + H_{12}y_1 + H_{13} \\ y_2(H_{31}x_1 + H_{32}y_1 + 1) = H_{21}x_1 + H_{22}y_1 + H_{23} \end{cases}$$

$$\Leftrightarrow \begin{cases} x_2 = H_{11}x_1 + H_{12}y_1 + H_{13} - H_{31}x_1x_2 - H_{32}y_1x_2 \\ y_2 = H_{21}x_1 + H_{22}y_1 + H_{23} - H_{31}x_1y_2 - H_{32}y_1y_2 \end{cases}$$

On place les quatre coordonnées $x_2^i$ sur les quatre premières lignes d'un vecteur B(8,1) et les quatre $y_2^i$ sur les lignes suivantes, de sorte que l'on résolve un système linéaire $B = AX$ avec les coefficients de H dans le vecteur X :

$$\begin{pmatrix} x_2^1 \\ x_2^2 \\ x_2^3 \\ x_2^4 \\ y_2^1 \\ y_2^2 \\ y_2^3 \\ y_2^4 \end{pmatrix} = \begin{pmatrix} x_1^1 & y_1^1 & 1 & 0 & 0 & 0 & -x_1^1x_2^1 & -y_1^1x_2^1 \\ x_1^2 & y_1^2 & 1 & 0 & 0 & 0 & -x_1^2x_2^2 & -y_1^2x_2^2 \\ x_1^3 & y_1^3 & 1 & 0 & 0 & 0 & -x_1^3x_2^3 & -y_1^3x_2^3 \\ x_1^4 & y_1^4 & 1 & 0 & 0 & 0 & -x_1^4x_2^4 & -y_1^4x_2^4 \\ 0 & 0 & 0 & x_1^1 & y_1^1 & 1 & -x_1^1y_2^1 & -y_1^1y_2^1 \\ 0 & 0 & 0 & x_1^2 & y_1^2 & 1 & -x_1^2y_2^2 & -y_1^2y_2^2 \\ 0 & 0 & 0 & x_1^3 & y_1^3 & 1 & -x_1^3y_2^3 & -y_1^3y_2^3 \\ 0 & 0 & 0 & x_1^4 & y_1^4 & 1 & -x_1^4y_2^4 & -y_1^4y_2^4 \end{pmatrix} * \begin{pmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \end{pmatrix}$$

Une fois ce système matriciel écrit sous MATLAB nous obtenons immédiatement les coefficients de H grâce à la touche « \ » contenant toute une fonction *résolution de système linéaire*. Notons bien que l'écriture de ce produit matriciel vient tout naturellement des systèmes d'équations écrits plus hauts. Ainsi, lorsque nous aboutirons à une écriture similaire des équations de projection pour la création de l'espace en trois dimensions, nous ne prendrons pas la peine de réécrire le produit matriciel, l'important est de savoir comment il a été obtenu.

Pour appairer les pixels associés du contenu et de la trame selon l'algorithme naïf, il ne reste qu'à traduire l'hypothèse de conservation d'intensité en une fonction MATLAB parcourant tous les pixels concernés et les modifiant un à un.

Il est possible d'optimiser ces codes de façon importante grâce à une solution classique en programmation : la vectorisation. En effet, des langages de programmation comme Matlab ou Octave sont optimisés pour le calcul algébrique, il convient donc d'utiliser des multiplications matricielles le plus possible et d'éviter les boucles *for* ou les tests *if*. Pour cela, on va mettre l'image entière dans un vecteur et les coordonnées à transformer vont toutes se retrouver dans la même matrice.

## ❖ Construction d'une structure *fil de fer* en trois dimensions

➢ *Création de l'espace 3D transposé sur la scène 2D considérée*

Comme cela a déjà été remarqué précédemment, nous trouvons les coefficients de la matrice de projection de manière rigoureusement identique à ceux de la matrice homographique, notamment en manipulant les équations déterminées dans la partie précédente :

$$\begin{cases} \underline{P_{2D}^i} = P * \underline{P_{3D}^i} \\ P = \begin{pmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & 1 \end{pmatrix} \end{cases} \Rightarrow \begin{cases} x_{2D}^i = \dfrac{P_{11}x_{3D}^i + P_{12}y_{3D}^i + P_{13}z_{3D}^i + P_{14}}{P_{31}x_{3D}^i + P_{32}y_{3D}^i + P_{33}z_{3D}^i + 1} \\ y_{2D}^i = \dfrac{P_{21}x_{3D}^i + P_{22}y_{3D}^i + P_{23}z_{3D}^i + P_{24}}{P_{31}x_{3D}^i + P_{32}y_{3D}^i + P_{33}z_{3D}^i + 1} \end{cases}$$

$$\Leftrightarrow \begin{cases} x_{2D}^i(P_{31}x_{3D}^i + P_{32}y_{3D}^i + P_{33}z_{3D}^i + 1) = P_{11}x_{3D}^i + P_{12}y_{3D}^i + P_{13}z_{3D}^i + P_{14} \\ y_{2D}^i(P_{31}x_{3D}^i + P_{32}y_{3D}^i + P_{33}z_{3D}^i + 1) = P_{21}x_{3D}^i + P_{22}y_{3D}^i + P_{23}z_{3D}^i + P_{24} \end{cases}$$

$$\Leftrightarrow \begin{cases} x_{2D}^i = P_{11}x_{3D}^i + P_{12}y_{3D}^i + P_{13}z_{3D}^i + P_{14} - P_{31}x_{3D}^i x_{2D}^i - P_{32}y_{3D}^i x_{2D}^i - P_{33}z_{3D}^i x_{2D}^i \\ y_{2D}^i = P_{21}x_{3D}^i + P_{22}y_{3D}^i + P_{23}z_{3D}^i + P_{24} - P_{31}x_{3D}^i y_{2D}^i - P_{32}y_{3D}^i y_{2D}^i - P_{33}z_{3D}^i y_{2D}^i \end{cases}$$

Puis en répétant verticalement ces 2 équations pour les 6 points choisis on voit apparaître un produit matriciel similaire à celui que l'on a trouvé pour l'homographie. On le résout grâce à « \ » et on obtient la matrice de projection P.

➢ *Tracé (2D) des différents segments de la structure (3D)*

Pour bien comprendre cette fonction il faut commencer par se représenter l'image comme un espace de vecteurs en deux dimensions ($x$,$y$) où chaque point est représenté par ses coordonnées (on n'est en réalité pas très éloignés de l'approche des images adoptée jusqu'ici).

On commence par calculer la norme *d* du vecteur reliant les deux points choisis (en nombre de pixels puisque les coordonnées des points correspondent à la position des pixels) puis on échantillonne notre segment en *2d-1* tronçons. Nous allons ensuite nous déplacer itérativement sur chacun de ces tronçons, déterminer les coordonnées arrondies ($x$,$y$) du pixel ciblé (leur calcul exact ne donnant pas une valeur entière) et modifier sa valeur. Nous pourrions alors très bien imaginer épaissir le trait en « coloriant » autant de pixels voisins qu'on le souhaite.

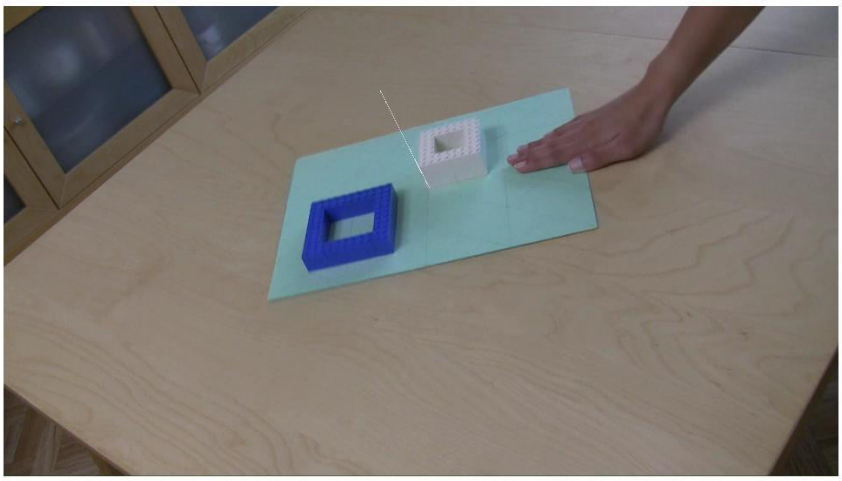
## 4. Résultats

On donne ci-dessous les résultats des différentes insertions effectuées :

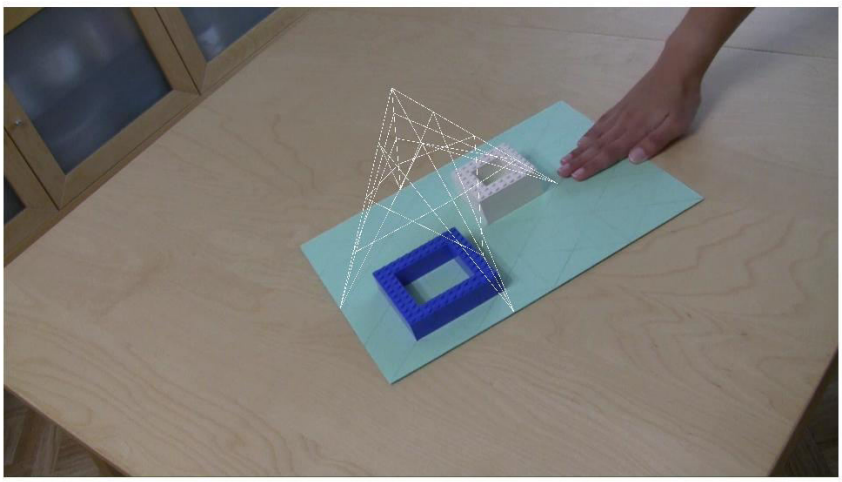
<u>Insertion d'une image</u>

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

Insertion d'un segment 3D



Insertion d'une pyramide



Après l'optimisation du code, on se rend compte de la force d'un programme vectorisé. En effet le temps d'exécution est divisé par 10 :

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

Temps d'exécution non optimisé : 2s

Profile Summary

*Generated 14-May-2019 22:52:46 using performance time.*

| Function Name | Calls | Total Time | Self Time* | Total Time Plot (dark band = self time) |
|---|---|---|---|---|
| bilinInterpol | 629451 | 0.833 s | 0.833 s | |
| remplaceImage | 371151 | 2.414 s | 1.581 s | |

Temps d'exécution optimisé : 0.2s

Profile Summary

*Generated 14-May-2019 22:49:36 using performance time.*

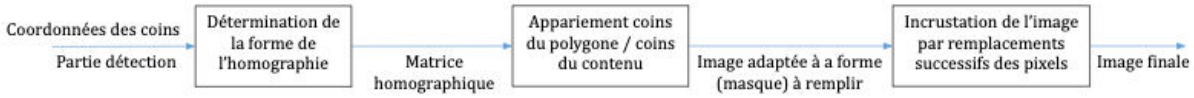
| Function Name | Calls | Total Time | Self Time* | Total Time Plot (dark band = self time) |
|---|---|---|---|---|
| testOptimisation | 1 | 9.806 s | 0.267 s | |
| getpts | 12 | 7.952 s | 4.465 s | |
| ...t;ToolbarController.handleMouseMotion | 495 | 2.750 s | 0.621 s | |
| ...r>@(e,d)obj.handleMouseMotion(e,d) | 491 | 2.746 s | 0.012 s | |
| ...rController.setToolbarBackgroundColor | 494 | 0.612 s | 0.090 s | |
| imshow | 2 | 0.582 s | 0.113 s | |
| VideoReader.read | 1 | 0.541 s | 0.009 s | |

## 5. Conclusion

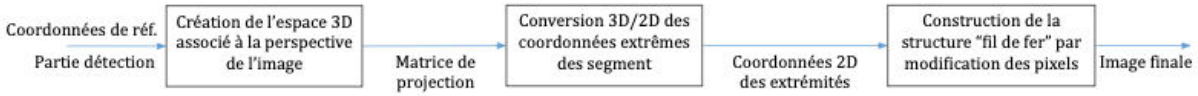
Ainsi à partir de quatre points (et/ou six si on veut insérer une structure 3D) dont on connait les coordonnées, on peut insérer une image (et/ou une structure 3D) à la place de la feuille. Ce code à la force de pouvoir s'appliquer à n'importe quelle vidéo contenant une feuille dont les coins ne sont pas masqués, et l'optimisation vectorielle rend le code très rapide et efficace. Nous pouvons résumer le principe de cet algorithme de détection de contours sous la forme du schéma-bloc suivant :

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

**Traitement de la transformation homographique**




**Construction d'un dessin 3D - *structure "fil de fer"***




## 6. Annexe

```
function [x_max,y_max,P] =
infoImage(A)

% Summary : On stocke les coordonnees
des coins de l'image a incruster
%
% Description : On prend en argument
le double de la photo, les coordonnees
% des coins sont fonction de la taille
de l'image

x_max = length(A(1,:));
y_max = length(A(:,1));
P =
[[1,1];[x_max,1];[x_max,y_max];[1,y_ma
x]];

end
```

---

```
function H = homographie(P, P1, P2,
P3, P4)

% Summary : On determine la
transformation pour incruster l'image
%
% Description : On prend en argument
les points detectes, les coordonnees
% des coins initiaux sont fonction de
la taille de l'image

D = [P1;P2;P3;P4];

A = zeros(8);

for i = 1:4

    A(i,:) = [D(i,1) D(i,2) 1 0 0 0 -
D(i,1)*P(i,1) -D(i,2)*P(i,1)];
    A(i+4,:) = [0 0 0 D(i,1) D(i,2) 1
-D(i,1)*P(i,2) -D(i,2)*P(i,2)];

end

B = [P(:,1);P(:,2)];
X = A\B;
H =
[[X(1),X(2),X(3)];[X(4),X(5),X(6)];[X(
7),X(8),1]];

end
```

---

```
function [P,M] = projection(P1, P2,
P3, P4, P5, P6)

% Summary : On determine la projection
pour passer de l'espace 3D a
% l'espace 2D
%
% Description : On prend en argument
les points detectes, les coordonnees
% des coins initiaux sont fonction de
la taille de l'image

M = [[0 0 0 1];[1 0 0 1];[1 1 0 1];[0
1 0 1];[0.5 0.5 0 1];[0.5 0.5 1 1]];
N = zeros(6,4);

B =
[P1(1);P2(1);P3(1);P4(1);P5(1);P6(1);P
1(2);P2(2);P3(2);P4(2);P5(2);P6(2)];

A = zeros(12,11);
A(1:6,1:4) = M;
A(7:12,5:8) = M;
A(1:6,5:8) = N;
A(7:12,1:4) = N;

for i = 1:6
    for j = 1:3
        A(i,j+8) = -M(i,j)*B(i);
        A(i+6,j+8) = -M(i,j)*B(i+6);
    end
end

X = A\B;
```

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

```
P = [[X(1) X(2) X(3)
X(4)];[X(5),X(6),X(7) X(8)];[X(9)
X(10) X(11) 1]];

end
```

---

```
function S = conv2Dnorm(V,P)
S = P*V;
S = S(1:2)/S(3);
end


function ImRGB = insertPyr(ImRGB,
S0,S1,S2,S3,S4)

% Summary : On insert une pyramide
stylise dans une image
%
% Description : On insert une pyramide
stylise de sommet S0 dont la base
% est definie par S1, S2, S3 et S4

B = cell(4);

for j = 1:4
        B{1,j} = [round(((4-
j)/4)*S0(1)+(j/4)*S1(1));round(((4-
j)/4)*S0(2)+(j/4)*S1(2))];
        B{2,j} = [round(((4-
j)/4)*S0(1)+(j/4)*S2(1));round(((4-
j)/4)*S0(2)+(j/4)*S2(2))];
        B{3,j} = [round(((4-
j)/4)*S0(1)+(j/4)*S3(1));round(((4-
j)/4)*S0(2)+(j/4)*S3(2))];
        B{4,j} = [round(((4-
j)/4)*S0(1)+(j/4)*S4(1));round(((4-
j)/4)*S0(2)+(j/4)*S4(2))];
end

ImRGB = insertSegment(S0,S1,ImRGB);
ImRGB = insertSegment(S0,S2,ImRGB);
ImRGB = insertSegment(S0,S3,ImRGB);
ImRGB = insertSegment(S0,S4,ImRGB);

for j = 1:4
    ImRGB =
insertSegment(B{1,j},B{4,5-j},ImRGB);
    ImRGB =
insertSegment(B{2,j},B{1,5-j},ImRGB);
    ImRGB =
insertSegment(B{3,j},B{2,5-j},ImRGB);
    ImRGB =
insertSegment(B{4,j},B{3,5-j},ImRGB);
end

end
```

---

```
% Test de l'optimisation pour une
insertion

clear
clc
close all

v = VideoReader('vid_in2.mp4');
ImRGB=read(v,100);

% On vectorise la frame
nI1=reshape(double(ImRGB(:,:,1)),1,[])
;
nI2=reshape(double(ImRGB(:,:,2)),1,[])
;
nI3=reshape(double(ImRGB(:,:,3)),1,[])
;


imshow(ImRGB);
[x,y] = getpts;
P1=[floor(x),floor(y)];
[x,y] = getpts;
P2=[floor(x),floor(y)];
[x,y] = getpts;
P3=[floor(x),floor(y)];
[x,y] = getpts;
P4=[floor(x),floor(y)];


tic

A = double(imread('cameraman.tif'));

[x_max,y_max,P] = infoImage(A);
H = homographie(P,P1,P2,P3,P4);

% Zone autour de la feuille
x = min([P1(1) P2(1) P3(1)
P4(1)]):max([P1(1) P2(1) P3(1)
P4(1)]);
y = min([P1(2) P2(2) P3(2)
P4(2)]):max([P1(2) P2(2) P3(2)
P4(2)]);

I1=reshape(double(ImRGB(y,x,1)),1,[]);
I2=reshape(double(ImRGB(y,x,2)),1,[]);
I3=reshape(double(ImRGB(y,x,3)),1,[]);

% Ici on place tout les pixels de la
zone dans une matrice U auquelle on
% applique l'homographie. Tout les
pixels homographies se trouvent donc
% dans VN.
n=length(x);
m=length(y);

[X,Y]=meshgrid(x,y);

vY=reshape(Y,1,m*n);
vX=reshape(X,1,m*n);
vI=ones(1,m*n);

U=[vX;vY;vI];
V=H*U;
VN=[V(1,:)./V(3,:);V(2,:)./V(3,:)];

% Test pour savoir si on doit inserer
ou non l'image
t=(VN(1,:)>=1).*(VN(1,:)<x_max).*(VN(2
,:)>=1).*(VN(2,:)<y_max);
% Masque faisant le tour de la main
tM=(VN(1,:)<=x_max).*(VN(1,:)>=3*x_max
/4).*(VN(2,:)>=y_max/4).*(VN(2,:)<=3*y
_max/4);
% Filtrage de la main
tI=(I1>=87).*(I1<=135).*(I2>=117).*(I3
>117);

s1=t+tM;
s2=t+tM+tI;
```

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

```
ind1=find(s1==1); % La condition t est
validee mais pas tM
ind2=find(s2==3); % Les conditions t,
tM et tI sont validees

% On ne garde que les positions qui
nous interesse, ie les positions ou
% l'on va inserer l'image
U1=U(:,ind1);
U2=U(:,ind2);
VN1=VN(:,ind1);
VN2=VN(:,ind2);

% Insertion de l'image
nI1(floor((U1(1,:)-
1)*1080+U1(2,:)))=bilinInterpolOpt(VN1
(1,:),VN1(2,:),x_max,y_max,A);
nI2(floor((U1(1,:)-
1)*1080+U1(2,:)))=bilinInterpolOpt(VN1
(1,:),VN1(2,:),x_max,y_max,A);
nI3(floor((U1(1,:)-
1)*1080+U1(2,:)))=bilinInterpolOpt(VN1
(1,:),VN1(2,:),x_max,y_max,A);

nI1(floor((U2(1,:)-
1)*1080+U2(2,:)))=bilinInterpolOpt(VN2
(1,:),VN2(2,:),x_max,y_max,A);
nI2(floor((U2(1,:)-
1)*1080+U2(2,:)))=bilinInterpolOpt(VN2
(1,:),VN2(2,:),x_max,y_max,A);
nI3(floor((U2(1,:)-
1)*1080+U2(2,:)))=bilinInterpolOpt(VN2
(1,:),VN2(2,:),x_max,y_max,A);

% On recree une image non vectorise
nI1=reshape(nI1,1080,1920);
nI2=reshape(nI2,1080,1920);
nI3=reshape(nI3,1080,1920);
nImRGB=cat(3,nI1,nI2,nI3);


toc


figure

imshow(uint8(nImRGB))
```

---

```
% Test de l'insertion d'un segment


clear
clc
close all


v = VideoReader('vid_in2.mp4');
% ImRGB=readFrame(v);
ImRGB=read(v,100);


imshow(ImRGB);
[x,y] = getpts;
P1=[floor(x),floor(y)];
[x,y] = getpts;
P2=[floor(x),floor(y)];
[x,y] = getpts;
P3=[floor(x),floor(y)];
[x,y] = getpts;
P4=[floor(x),floor(y)];
[x,y] = getpts;
P5=[floor(x),floor(y)];
[x,y] = getpts;
P6=[floor(x),floor(y)];


tic


[P,M] = projection(P1,P2,P3,P4,P5,P6);

% Deux extremites du segment en
coordonnees 3D
v1=[0.5;0.5;5;1];
v2=[0.5;0.5;0;1];

% Application de la projection
s1=P*v1;
s1=s1(1:2)/s1(3);
s2=P*v2;
s2=s2(1:2)/s2(3);

% Insertion du segment
ImRGB=insertSegment(s1,s2,ImRGB);


toc


figure
imshow(ImRGB)
```

---

```
%% Creation de la video finale

clear
clc
close all

tic


newV = VideoWriter('video.avi');
newV.FrameRate = 25;
open(newV);

Vid = VideoReader('vid_in2.mp4');
ImRGB=readFrame(Vid);
nI1=reshape(double(ImRGB(:,:,1)),1,[])
;
nI2=reshape(double(ImRGB(:,:,2)),1,[])
;
nI3=reshape(double(ImRGB(:,:,3)),1,[])
;
ImYCbCr=rgb2ycbcr(ImRGB);
I=double(ImYCbCr(:,:,1));


imshow(ImRGB);
[x,y] = getpts;
P1=[x,y];
[x,y] = getpts;
P2=[x,y];
[x,y] = getpts;
P3=[x,y];
[x,y] = getpts;
P4=[x,y];
[x,y] = getpts;
P5=[x,y];
[x,y] = getpts;
P6=[x,y];
```

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

```
k=0;

while hasFrame(Vid)

    [hG1,hD1,bG1,bD1] =
zoneCoin(P1,25);
    [hG2,hD2,bG2,bD2] =
zoneCoin(P2,25);
    [hG3,hD3,bG3,bD3] =
zoneCoin(P3,25);
    [hG4,hD4,bG4,bD4] =
zoneCoin(P4,25);
    [hG5,hD5,bG5,bD5] =
zoneCoin(P5,25);
    [hG6,hD6,bG6,bD6] =
zoneCoin(P6,25);

    [Ix,Iy]=intensityGradient(I,2);

    D1=harrisDetect(Ix,Iy,0.05,3);
    D2=harrisDetect(Ix,Iy,0.05,5);

    A1=D1>0;
    A2=D2>0;

    D=A1.*A2.*D1.*D2;

    P1 = maxCoin(D,hG1,hD1,bG1,bD1);
    P2 = maxCoin(D,hG2,hD2,bG2,bD2);
    P3 = maxCoin(D,hG3,hD3,bG3,bD3);
    P4 = maxCoin(D,hG4,hD4,bG4,bD4);
    P5 = maxCoin(D,hG5,hD5,bG5,bD5);
    P6 = maxCoin(D,hG6,hD6,bG6,bD6);


    k=k+1

    A =
double(imread('cameraman.tif'));

    [x_max,y_max,P] = infoImage(A);
    H = homographie(P,P1,P2,P3,P4);


    x = min([P1(1) P2(1) P3(1)
P4(1)]):max([P1(1) P2(1) P3(1)
P4(1)]);
    y = min([P1(2) P2(2) P3(2)
P4(2)]):max([P1(2) P2(2) P3(2)
P4(2)]);


I1=reshape(double(ImRGB(y,x,1)),1,[]);

I2=reshape(double(ImRGB(y,x,2)),1,[]);

I3=reshape(double(ImRGB(y,x,3)),1,[]);


    n=length(x);
    m=length(y);

    [X,Y]=meshgrid(x,y);

    vY=reshape(Y,1,m*n);
    vX=reshape(X,1,m*n);
    vI=ones(1,m*n);

    U=[vX;vY;vI];
    V=H*U;

VN=[V(1,:)./V(3,:);V(2,:)./V(3,:)];

t=(VN(1,:)>=1).*(VN(1,:)<x_max).*(VN(2
,:)>=1).*(VN(2,:)<y_max);

tM=(VN(1,:)<=x_max).*(VN(1,:)>=3*x_max
/4).*(VN(2,:)>=y_max/4).*(VN(2,:)<=3*y
_max/4);

tI=(I1>=87).*(I1<=135).*(I2>=117).*(I3
>117);

    s1=t+tM;
    s2=t+tM+tI;

    ind1=find(s1==1);
    ind2=find(s2==3);


    U1=U(:,ind1);
    U2=U(:,ind2);
    VN1=VN(:,ind1);
    VN2=VN(:,ind2);


    nI1(floor((U1(1,:)-
1)*1080+U1(2,:)))=bilinInterpolOpt(VN1
(1,:),VN1(2,:),x_max,y_max,A);
    nI2(floor((U1(1,:)-
1)*1080+U1(2,:)))=bilinInterpolOpt(VN1
(1,:),VN1(2,:),x_max,y_max,A);
    nI3(floor((U1(1,:)-
1)*1080+U1(2,:)))=bilinInterpolOpt(VN1
(1,:),VN1(2,:),x_max,y_max,A);

    nI1(floor((U2(1,:)-
1)*1080+U2(2,:)))=bilinInterpolOpt(VN2
(1,:),VN2(2,:),x_max,y_max,A);
    nI2(floor((U2(1,:)-
1)*1080+U2(2,:)))=bilinInterpolOpt(VN2
(1,:),VN2(2,:),x_max,y_max,A);
    nI3(floor((U2(1,:)-
1)*1080+U2(2,:)))=bilinInterpolOpt(VN2
(1,:),VN2(2,:),x_max,y_max,A);


    nI1=reshape(nI1,1080,1920);
    nI2=reshape(nI2,1080,1920);
    nI3=reshape(nI3,1080,1920);

    nImRGB=cat(3,nI1,nI2,nI3);


    [P,M] =
projection(P1,P2,P3,P4,P5,P6);

    S0 =
conv2Dnorm([0.375;0.375;3;1],P);
    S1 = conv2Dnorm([0;0.5;0;1],P);
    S2 = conv2Dnorm([0.375;0;0;1],P);
    S3 = conv2Dnorm([0.75;0.5;0;1],P);
    S4 = conv2Dnorm([0.375;1;0;1],P);

    nImRGB = insertPyr(nImRGB,
S0,S1,S2,S3,S4);


    writeVideo(newV, uint8(nImRGB));

    ImRGB=readFrame(Vid);

nI1=reshape(double(ImRGB(:,:,1)),1,[])
;

nI2=reshape(double(ImRGB(:,:,2)),1,[])
;
```

TALLON Quentin
PHELIX Pierre-Louis
d'ARMAGNAC de CASTANET Quentin

```
nI3=reshape(double(ImRGB(:,:,3)),1,[])
;
    ImYCbCr=rgb2ycbcr(ImRGB);
    I=double(ImYCbCr(:,:,1));
end
```